AGARD-R-807

AGARD

ADVISORY GROUP FOR AEROSPACE RESEARCH & DEVELOPMENT
7 RUE ANCELLE, 92200 NEUILLY-SUR-SEINE, FRANCE

DTIC
ELECTE
DEC 06 1995
D

AGARD REPORT R-807

Special Course on Parallel Computing in CFD

(l'Aérodynamique numérique et le calcul en parallèle)

The material assembled in this report was prepared under the combined sponsorship of the AGARD Fluid Dynamics Panel, the Consultant and Exchange Program of AGARD, and the von Karman Institute (VKI) for Fluid Dynamics. It was presented in an AGARD-FDP-VKI Special Course at the VKI, Rhode-Saint-Genèse, Belgium, 15-19 May 1995 and 16-20 October 1995 at NASA Ames, United States.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

19951205 022

NORTH ATLANTIC TREATY ORGANIZATION

Published October 1995

Distribution and Availability on Back Cover

AGARD

ADVISORY GROUP FOR AEROSPACE RESEARCH & DEVELOPMENT

7 RUE ANCELLE, 92200 NEUILLY-SUR-SEINE, FRANCE

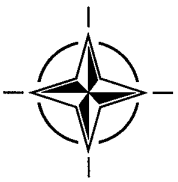
AGARD REPORT R-807

Special Course on Parallel Computing in CFD

(l'Aérodynamique numérique et le calcul en parallèle)

The material assembled in this report was prepared under the combined sponsorship of the AGARD Fluid Dynamics Panel, the Consultant and Exchange Program of AGARD, and the von Karman Institute (VKI) for Fluid Dynamics. It was presented in an AGARD-FDP-VKI Special Course at the VKI, Rhode-Saint-Genèse, Belgium, 15-19 May 1995 and 16-20 October 1995 at NASA Ames, United States.

DTIC QUALITY INSPECTED 5



North Atlantic Treaty Organization
Organisation du Traité de l'Atlantique Nord

The Mission of AGARD

According to its Charter, the mission of AGARD is to bring together the leading personalities of the NATO nations in the fields of science and technology relating to aerospace for the following purposes:

- Recommending effective ways for the member nations to use their research and development capabilities for the common benefit of the NATO community;
- Providing scientific and technical advice and assistance to the Military Committee in the field of aerospace research and development (with particular regard to its military application);
- Continuously stimulating advances in the aerospace sciences relevant to strengthening the common defence posture;
- Improving the co-operation among member nations in aerospace research and development;
- Exchange of scientific and technical information;
- Providing assistance to member nations for the purpose of increasing their scientific and technical potential;
- Rendering scientific and technical assistance, as requested, to other NATO bodies and to member nations in connection with research and development problems in the aerospace field.

The highest authority within AGARD is the National Delegates Board consisting of officially appointed senior representatives from each member nation. The mission of AGARD is carried out through the Panels which are composed of experts appointed by the National Delegates, the Consultant and Exchange Programme and the Aerospace Applications Studies Programme. The results of AGARD work are reported to the member nations and the NATO Authorities through the AGARD series of publications of which this is one.

Participation in AGARD activities is by invitation only and is normally limited to citizens of the NATO nations.

The content of this publication has been reproduced
directly from material supplied by AGARD or the authors.

Published October 1995

Copyright © AGARD 1995
All Rights Reserved

ISBN 92-836-1025-3

Printed by Canada Communication Group
45 Sacré-Cœur Blvd., Hull (Québec), Canada K1A 0S7

Contents

	Page
Preface/Préface	iv
Special Course Staff	vi
Recent Publications of the Fluid Dynamics Panel	vii
	Reference
Parallel Computers and Parallel Algorithms for CFD: An Introduction by D. Roose and R. Van Driessche	1
Load Balancing Computational Fluid Dynamics Calculations on Unstructured Grids by R. Van Driessche and D. Roose	2
Parallel Interative Solution Methods for Linear Systems arising from Discretized PDE's by H.A. Van der Vorst	3
Structured Grid Solvers I — Accurate and Efficient Flow Solvers for 3D Applications on Structured Meshes by N. Kroll, R. Radespiel and C.-C. Rossow	4
Structured Grid Solvers II — Parallelization of Block Structured Flow Solvers by B. Eisfeld, H.-M. Bleecke, N. Kroll and H. Ritzdorf	5
Parallel Automated Adaptive Procedures for Unstructured Meshes by M.S. Shephard, J.E. Flaherty, H.L. de Cougny, C. Ozturan, C.L. Bottasso and M.W. Beall	6
Parallel CFD Algorithms on Unstructured Meshes by T.J. Barth	7
High Performance Simulation of Coupled Nonlinear Transient Aeroelastic Problems by C. Farhat	8

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Preface

This volume is a compilation of the edited proceedings of the "Parallel Computing in CFD" course held at the von Kármán Institute (VKI) in Rhode-Saint-Genèse, Belgium, 15-19 May 1995 and at the NASA Ames Research Center, Moffett Field, California, USA 16-20 October 1995.

In order to circumvent the limits posed by processor performance, today's advanced computer architectures permit simultaneous computations on multiple functional units. This approach, termed parallel processing, has the potential for a dramatic improvement in overall computational speed. This revolution in parallel processing is expected to strongly influence the choice of algorithms used in computational fluid dynamics (CFD).

This series of lectures, supported by the AGARD Fluid Dynamics Panel and the von Kármán Institute, presents and discusses the latest in advances and future trends in the application of parallel computing to solve computationally intensive problems in CFD. Topics in this lecture series focus on the increasingly sophisticated types of architectures now available, and how to exploit these architectures by appropriate algorithms for the simulation of fluid flow.

Some of the subjects discussed are: parallel algorithms for computing compressible and incompressible flow; domain decomposition algorithms and partitioning techniques; and parallel algorithms for solving linear systems arising from the discretized partial differential equations.

We want to thank all the speakers for their outstanding work, as well as the organizers at AGARD, VKI, and NASA Ames.

Préface

Ce volume est un recueil des exposés du cours sur «Le calcul en parallèle en CFD» organisé à l'Institut Von Karman (VKI) à Rhodes-Saint-Genèse, en Belgique, du 15 au 19 mai 1995, ainsi qu'au NASA Ames Research Center, Moffett Field, en Californie aux Etats-Unis, du 16 au 20 octobre 1995.

Afin de circonvenir les limitations imposées par les performances des microprocesseurs, les architectures informatiques avancées d'aujourd'hui permettent le calcul simultané réalisé sur de multiples unités fonctionnelles. Cette approche, appelée «le calcul en parallèle», pourrait amener une amélioration spectaculaire des vitesses de calcul générales. Cette révolution dans le calcul en parallèle devrait exercer une forte influence sur le choix des algorithmes à utiliser en aérodynamique numérique (CFD).

Ce cours, organisé sous l'égide conjointe du Panel AGARD de la dynamique des fluides et de l'Institut Von Karman, présente et examine les derniers progrès réalisés ainsi que les perspectives d'avenir en ce qui concerne l'application du calcul en parallèle à la résolution de certains problèmes en CFD impliquant une grande puissance de calcul. Les sujets examinés lors du cours ont porté, principalement, sur les architectures de plus en plus sophistiquées qui sont actuellement disponibles et sur leur exploitation par l'intermédiaire des algorithmes appropriés, pour la simulation des écoulements des fluides.

Parmi les sujets examinés l'on distingue: les algorithmes parallèles pour le calcul des écoulements compressibles et non-compressibles; les algorithmes de décomposition de domaine et les techniques de découpage en partitions et les algorithmes parallèles pour la résolution de systèmes linéaires résultant des équations aux dérivées partielles discrétisées.

Nous tenons à remercier l'ensemble des conférenciers pour la qualité de leurs contributions, ainsi que les organisateurs à l'AGARD, au VKI et au NASA Ames.

Special Course Staff

SPECIAL COURSE DIRECTOR

Dr. H. Simon
Silican Graphics
Mail Stop 7L 580
2011 N. Shoreline Blvd.
Mountain View, CA 94043
United States

LECTURERS

Professor D. Roose
Katholieke Universiteit Leuven
Dept. of Computer Science
Celestijnenlaan 200A
B-3001 Heverlee-Leuven
Belgium

Mr. B. Eisfeld
Institute for Design Aerodynamics
DLR, Braunschweig
Germany

Dr. N. Kroll
Institute for Design Aerodynamics
DLR, Lilienthalplatz 7
38108 Braunschweig
Germany

Professor H.A. Van der Vorst
Mathematical Institute
University of Utrecht
P.O. Box 80010
NL-3508 TA Utrecht
The Netherlands

Mr. T.J. Barth
Advanced Algorithms and
Applications Branch
NASA Ames Research Center
Moffett Field, CA 94035
United States

Professor C. Farhat
Dept. of Aerospace Engineering
Sciences and Center for Aerospace
Structures
University of Colorado at Boulder
Boulder, CO 80309-0429
United States

Professor M.S. Shephard
Scientific Computation Research
Center
Rensselaer Polytechnic Institute
Troy, NY 12180-3590
United States

NATIONAL COORDINATORS

Professor H. Deconinck
von Kàrmàn Institute for Fluid Dynamics
Chaussée de Waterloo, 72
1640 Rhode-Saint-Genèse
Belgium

Mr. T.J. Barth
Advanced Algorithms and
Applications Branch
NASA Ames Research Center
Moffett Field, CA 94035
United States

PANEL EXECUTIVE

Mr. J.K. Molloy

Mail from Europe:
AGARD/OTAN
Attn: FDP Executive
7 rue Ancelle
92200 Neuilly-sur-Seine
France

Mail from US and Canada:
Attn: FDP Executive
PSC 116
AGARD/NATO
APO AE 09777

Recent Publications of the Fluid Dynamics Panel

AGARDOGRAPHS (AG)

Computational Aerodynamics Based on the Euler Equations
AGARD AG-325, September 1994

Scale Effects on Aircraft and Weapon Aerodynamics
AGARD AG-323, July 1994

Design and Testing of High-Performance Parachutes
AGARD AG-319, November 1991

Experimental Techniques in the Field of Low Density Aerodynamics
AGARD AG-318 (E), April 1991

Techniques expérimentales liées à l'aérodynamique à basse densité
AGARD AG-318 (FR), April 1990

A Survey of Measurements and Measuring Techniques in Rapidly Distorted Compressible Turbulent Boundary Layers
AGARD AG-315, May 1989

Reynolds Number Effects in Transonic Flows
AGARD AG-303, December 1988

REPORTS (R)

Optimum Design Methods for Aerodynamics
AGARD R-803, Special Course Notes, November 1994

Missile Aerodynamics
AGARD R-804, Special Course Notes, May 1994

Progress in Transition Modelling
AGARD R-793, Special Course Notes, April 1994

Shock-Wave/Boundary-Layer Interactions in Supersonic and Hypersonic Flows
AGARD R-792, Special Course Notes, August 1993

Unstructured Grid Methods for Advection Dominated Flows
AGARD R-787, Special Course Notes, May 1992

Skin Friction Drag Reduction
AGARD R-786, Special Course Notes, March 1992

Engineering Methods in Aerodynamic Analysis and Design of Aircraft
AGARD R-783, Special Course Notes, January 1992

Aircraft Dynamics at High Angles of Attack: Experiments and Modelling
AGARD R-776, Special Course Notes, March 1991

ADVISORY REPORTS (AR)

Aerodynamics of 3-D Aircraft Afterbodies
AGARD AR-318, Report of WG17, September 1995

A Selection of Experimental Test Cases for the Validation of CFD Codes
AGARD AR-303, Vols. I and II, Report of WG-14, August 1994

Quality Assessment for Wind Tunnel Testing
AGARD AR-304, Report of WG-15, July 1994

Air Intakes for High Speed Vehicles
AGARD AR-270, Report of WG13, September 1991

Appraisal of the Suitability of Turbulence Models in Flow Calculations
AGARD AR-291, Technical Status Review, July 1991

Rotary-Balance Testing for Aircraft Dynamics
AGARD AR-265, Report of WG11, December 1990

Calculation of 3D Separated Turbulent Flows in Boundary Layer Limit
AGARD AR-255, Report of WG10, May 1990

Adaptive Wind Tunnel Walls: Technology and Applications
AGARD AR-269, Report of WG12, April 1990

CONFERENCE PROCEEDINGS (CP)

Aerodynamics and Aeroacoustics of Rotorcraft
AGARD CP-552, August 1995

Application of Direct and Large Eddy Simulation to Transition and Turbulence
AGARD CP-551, December 1994

Wall Interference, Support Interference and Flow Field Measurements
AGARD CP-535, July 1994

Computational and Experimental Assessment of Jets in Cross Flow
AGARD CP-534, November 1993

High-Lift System Aerodynamics
AGARD CP-515, September 1993

Theoretical and Experimental Methods in Hypersonic Flows
AGARD CP-514, April 1993

Aerodynamic Engine/Airframe Integration for High Performance Aircraft and Missiles
AGARD CP-498, September 1992

Effects of Adverse Weather on Aerodynamics
AGARD CP-496, December 1991

Manoeuvring Aerodynamics
AGARD CP-497, November 1991

Vortex Flow Aerodynamics
AGARD CP-494, July 1991

Missile Aerodynamics
AGARD CP-493, October 1990

Aerodynamics of Combat Aircraft Controls and of Ground Effects
AGARD CP-465, April 1990

Computational Methods for Aerodynamic Design (Inverse) and Optimization
AGARD CP-463, March 1990

Applications of Mesh Generation to Complex 3-D Configurations
AGARD CP-464, March 1990

Fluid Dynamics of Three-Dimensional Turbulent Shear Flows and Transition
AGARD CP-438, April 1989

Validation of Computational Fluid Dynamics
AGARD CP-437, December 1988

Aerodynamic Data Accuracy and Quality: Requirements and Capabilities in Wind Tunnel Testing
AGARD CP-429, July 1988

Aerodynamics of Hypersonic Lifting Vehicles
AGARD CP-428, November 1987

Aerodynamic and Related Hydrodynamic Studies Using Water Facilities
AGARD CP-413, June 1987

Applications of Computational Fluid Dynamics in Aeronautics
AGARD CP-412, November 1986

Parallel Computers and Parallel Algorithms for CFD : An Introduction

Dirk Roose and Rafael Van Driessche

Katholieke Universiteit Leuven
Dept. of Computer Science
Celestijnenlaan 200A
B-3001 Heverlee-Leuven, Belgium
E-mail: Dirk.Roose@cs.kuleuven.ac.be

1 SUMMARY

This text presents a tutorial on those aspects of parallel computing that are important for the development of efficient parallel algorithms and software for Computational Fluid Dynamics.

We first review the main architectural features of parallel computers and we briefly describe some parallel systems on the market today. We introduce some important concepts concerning the development and the performance evaluation of parallel algorithms. We discuss how work load imbalance and communication costs on distributed memory parallel computers can be minimised. We present performance results for some CFD testcases. We focus on applications using structured and block structured grids, but the concepts and techniques are valid also for unstructured grids.

2 PARALLEL COMPUTING

2.1 Parallel computer architectures

2.1.1 Classification of Flynn

For many years the taxonomy of Flynn has been used for the classification of high-performance computers. This classification is based on the way instruction and data streams are handled (Single or Multiple Instruction / Data Streams). This leads to a classification in three main architectural classes, see e.g. [1].

SISD systems. These are the conventional systems (workstations, compute-servers) that contain one CPU and hence can execute one instruction stream in serial mode. Nowadays many large compute-servers or mainframes have more than one CPU but these are most often used to execute unrelated jobs (instruction streams). Therefore, such systems should be regarded as (a couple of) SISD machines.

SIMD systems. Such systems have a large number of (simple) processing units, ranging from 1,024 up to 64K, that all may execute the same instruction on different data in lock-step. Thus a single instruction manipulates many data items in parallel. In the past, SIMD machines such as the Connection Machine CM-2 of Thinking Machines and the MasPar have been quite successful. Today, the SIMD architecture has nearly disappeared, except in systems for specific application areas, such as image processing, that are dominated by highly structured data sets and data access patterns.

MIMD systems. In 'Multiple Instruction, Multiple Data' systems, the processors independently execute different instruction streams, each on its own data. Hardware and software are designed so that processors can cooperate efficiently. Parallel processing occurs when tasks executed on different processors together form one single job.

Vector processors are often considered as a subclass of SIMD systems. Vector processors contain special hardware ('vector units') to perform operations on arrays of similar data in a pipelined fashion. These vector units can deliver results with a rate of one, two and—in special cases—three per clock cycle. From the programmer's point of view, vector processors operate on their data in an almost parallel way (SIMD-style) when executing in vector mode. Vector processors are used in the Cray C90, J916 and T90-series, the Convex C-series, Fujitsu VP-series, NEC SX-series, etc.
The pipelined execution of floating point operations is also a key concept in *RISC processors*, used in high performance workstations. Advanced RISC processors can also execute several instruction in parallel (e.g. 'dual instruction mode').

2.1.2 Memory organisation of MIMD systems

Parallel computers can also be classified based to other criteria. MIMD systems are further distinguished according to the organisation of the memory.

Paper presented in an AGARD-FDP-VKI Special Course on "Parallel Computing in CFD", held at the VKI, Rhode-Saint-Genèse, Belgium, from 15-19 May 1995 and 16-20 October 1995 at NASA Ames, United States and published in R-807.

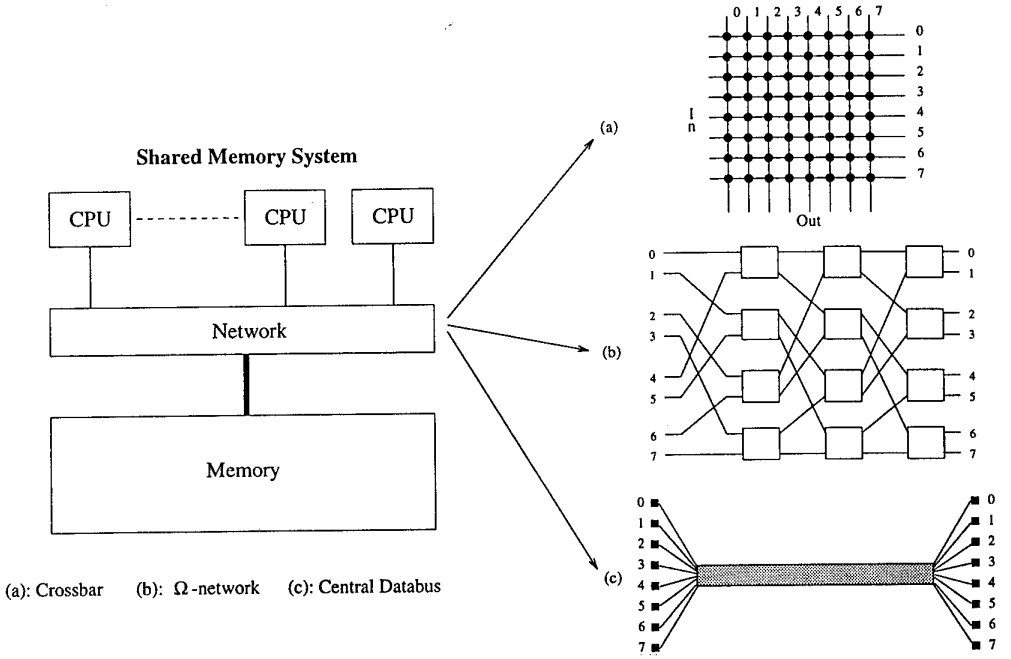


Fig. 1: *Shared Memory MIMD parallel computers: possible interconnections.*

Shared memory MIMD systems. In shared memory MIMD systems all processors have access to a common memory. The main architectural problem in shared memory systems is that of the connection of the processors to the memory (or memory modules). As more processors are added, the collective bandwidth to the memory ideally should increase linearly with the number of processors, P. Unfortunately, full interconnection is very costly, requiring $O(P^2)$ connections. So, various alternative interconnection networks are used, some of which are shown in Fig. 1. A crossbar uses P^2 interconnections and a omega-network uses $Plog_2P$ connections, while a central bus represents only one connection. In all present-day multi-processor vector processors, a crossbar is used. Due to the limited capacity or the cost of the interconnection network, shared memory parallel computers are not scalable to a very high number of processors.

The shared memory concept is used already for $\pm$ 10 years in multiprocessor vector machines (Cray X-MP, IBM 3090, their successors and similar systems from other vendors). However these systems have not been used very often as truly parallel systems: most jobs use only one processor. One reason for this is the limited number of processors (often 4 or 8), which limits the possible 'speedup' of the execution. Moreover, because of time-sharing, the user normally has no full control on the number of processors allocated to his job at a particular moment. This may also limit the speedup that can be achieved.

Nowadays, a number of vendors (Convex, Silicon Graphics, ...) offer shared memory MIMD systems based on RISC processors, with up to $\pm$ 20 processors.

Distributed memory MIMD systems. A distributed memory MIMD parallel computer consists of a number of processors, each with its own local memory, interconnected by a *communication network*. The combination of a processor and its local memory is often called a *processing node*. Each processing node is in fact a complete computer, operating rather independently from the other nodes. Processing nodes can only communicate by passing messages over the communication network.

Also for distributed memory machines, the structure of the communication network is of crucial importance. Ideally, one would like to have a completely connected system where each processing node is directly connected to every other node. However, this is not feasible for a large number of nodes. Therefore the processing nodes are arranged in some interconnection topology. The richness of the connection structure has to be balanced against the costs.

The *hypercube topology* has been used in several systems in the past. A nice feature is that for a hypercube with $P = 2^d$ processing nodes the 'diameter' of the network (i.e. the maximum number of links between any two nodes) is d. So, the diameter grows only logarithmically with the number of nodes. In addition, it is possible to simulate on a hypercube many other topologies, such as trees, rings, 2-D and 3-D meshes, since these topologies are subsets of the hypercube topology.

In the current parallel systems, the network topology and the communication diameter are of less importance, because these systems employ some form of 'wormhole routing' of messages. This means that as soon as a communication path between two nodes

is established, the data is sent through this path without disturbing the operation of the intermediate nodes. Except for a small amount of time in setting up the communication path between nodes, the communication time has become virtually independent of the distance between the nodes.

Some systems use a *2-D or 3-D mesh* structure for the network. The rationale for this is that this interconnection topology is sufficient for most algorithms used in large-scale scientific computing and that a richer interconnection structure hardly pays off. In other systems a *multi-stage network* is used, e.g. an omega-network as shown in Figs. 1 and 4. Multistage networks have the advantage that the 'bisection bandwidth' can scale linearly with the number of processors, while maintaining a fixed number of communication links per processor. The bisection bandwidth of a distributed memory system is defined as the bandwidth available on all communication links that connect one half of the system ($P/2$ processors) with the second half.

An important advantage of distributed memory systems is that this architecture suffers less from the scalability problem. The network (with its limited bandwidth) has to be used only when processing nodes communicate, not for every memory access. A disadvantage is that the communication overhead is (much) higher than the overhead caused by using shared data in a shared memory system. When the structure of a problem dictates a frequent exchange of data between processors, it may well be that only a very small fraction of the theoretical peak performance can be achieved.

The first generation DM-MIMD systems were based on simple, inexpensive microprocessors. Thus even when 100 to 1000 processors were interconnected, the peak performance of these machines was lower than that of typical vector processors and shared memory parallel supercomputers. Today, distributed memory parallel computers are often outperforming more traditional supercomputers. This is due to the fast growing performance of the RISC processors used in distributed memory systems and due to the greatly improved network technology. Moreover, many systems now have sophisticated hardware and software that allow fast *parallel I/O* to disk storage (i.e. a 'Concurrent File System'). As a result, distributed memory parallel systems are rapidly gaining importance in fields where computational performance is important such as Computational Fluid Dynamics.

Examples of distributed memory machines are the Intel Paragon, the CM-5 of Thinking Machines, Cray T3D, IBM SP2. Distributed memory systems with more than a thousand processors exist, but most systems have 16 to 128 processors.

Since a few years, networks or *clusters of workstations* are used as 'low cost' distributed memory parallel computers. A workstation cluster allows to exploit otherwise unused computing capacity. Of course, if workstations are simply connected together via Ethernet (or even via a fast FDDI interconnection), the number of workstation that can be used effectively together as a parallel system is limited, because of the limited communication performance. Some workstation vendors offer interconnection switches to provide fast communication (e.g. Digital). Such clusters are bridging the gap with 'truly parallel computers'.

Thus a whole range of systems are used nowadays as parallel computers, ranging from small workstation clusters to large systems with many processors and sophisticated communication network technology.

Hybrid memory organisations. Although the difference between shared and distributed memory systems seems clear cut, many parallel systems have a *hybrid* memory organisation. In a shared memory system, every processor may have a large cache, which can be considered as a local memory. Some systems have a two-level organisation: processors are grouped together in shared memory modules, which are interconnected via a communication network. Finally, a distributed memory system may contain hardware and software support to access data in other processor's memories in a way that is transparent to the user. Depending on the precise form of this support, this is called 'virtual shared memory', 'global shared memory', 'global virtual memory', etc.

Memory hierarchy and performance. Both vector processors and RISC processors can perform floating point operations much faster than data can be read and written into main memory. Vector registers (in vector processors) or cache memories (in RISC processors) are placed between the processor and the main memory. These very fast memory modules should keep the processors busy with computation without having to frequently reference main memory.

Vector registers, caches, local memories and/or the global (shared) memory form together a *memory hierarchy*. The performance that can be achieved for a given application program critically depends on the (re-)use of data stored in the 'higher levels' of the memory hierarchy. *Thus in order to achieve a high performance, the algorithms should exhibit locality of data access, both in (address) space and time.*

2.2 Programming parallel computers

We have indicated that there is no clear cut distinction between shared memory and distributed memory parallel architectures, and that some recent parallel systems have a hybrid organisation. However we can clearly distinguish between two different programming models, the shared memory and the distributed memory programming model.

In both models, the execution of a program is split in several processes that are executed in parallel. In most cases, on each processor only one process is executed and therefore we will use the term 'processor' in the discussion below, although 'process' would often be more precise.

2.2.1 Shared memory programming model

Regardless of the physical organisation of the memory, the shared memory programming model is based on the existence of a common or *global address space*, i.e. every processor can address every memory location. Thus processors communicate by accessing (writing, reading) shared data. The time to access the shared data may differ very much, depending on the physical location of the shared data (cache, local memory, another processor's memory).

Also, situations are possible where different processors wish to use a part of the common memory simultaneously. In that case *synchronisation* of the processes is necessary. Synchronisation is also needed before a sequential part of a program, in order to assure that *all* processors have finished their parallel actions prior to this sequential part.

Thus the performance may deteriorate substantially due to 'memory conflicts', synchronisation and 'sequential bottlenecks'. Further, the overhead associated with the creation of (parallel) tasks can be very high.

At present, Fortran and C compilers exist that perform automatic parallelisation in the shared memory programming model. The programmer can influence the parallelisation via directives (as for vector processing). Parallelisation can be carried out at loop level or at task (or routine) level, also called 'fine grained', resp. 'coarse grained' parallelism, or 'micro-tasking' resp. 'macro-tasking'.

2.2.2 Distributed memory programming model

In the distributed memory programming or message passing model, processors can only access their own private memory. Whenever a processor needs data that reside in the memory of another processing node, the data must be sent between the processing nodes. Such a *message passing* or *communication* step involves preparation of the message in the sending node, transmission over the communication network and reception of the message in the destination node. When the message passing model is used on a shared memory system, the actual transmission is replaced by storage of the information in shared memory.

Also in a distributed memory model, synchronisation problems can occur. It is possible that a processor does not have the data available yet at the moment they are needed by another processor ; at this *synchronisation point* the processor has to wait for the other processor to catch up. Synchronisation may also be needed to assure that the communication between processors proceeds in a correct way.

Although each processor can execute a different program, most often the 'Single Program, Multiple Data' (SPMD) programming style is used: all processors execute the same program acting on different parts of the data set. This requires an appropriate partitioning (distribution) of the data of the data and of the operations that have to be performed on them. The partitioning of the data must be so that the work load is well balanced between processors and so that communication and synchronisation is minimised.

Programming in the distributed memory model is often more difficult than programming in the shared memory model. The programmer must be aware of the location of the data in the local memories and has to move or distribute these data explicitly when needed. The partitioning of the data and all necessary communication has to be included explicitly into the program. A sequential program often needs significant changes in order to parallelise it.

Distributed memory programs are written in conventional languages (Fortran, C, C++, ...) and a *communication library* is used to implement the communication and synchronisation operations. Basic communication routines allow messages to be sent and received between arbitrary processing nodes. Incoming messages are normally buffered by the operating system at the destination node until the application program requests the message. Also various 'higher level' routines are provided, e.g. for 'global operations' on a set of data distributed across the nodes (broadcast, global sum, global maximum, ...) and for synchronisation.

In addition to the machine dependent communication libraries, several machine independent libraries have been developed. Widely used libraries are PVM [2], MPI [3], PARMACS [4]. Some of these libraries or environments (e.g. PARMACS) contain utility routines that perform automatic partitioning and mapping of vectors and matrices, and facilities for performance monitoring and analysis. The PVM environment provides facilities to use a (heterogeneous) network of workstations as a distributed memory parallel computer.

Compilers and software tools that perform (semi) automatic parallelisation for DM-MIMD machines are becoming available now. High Performance Fortran is a set of extensions to Fortran 90 for writing parallel applications [5]. HPF includes features for mapping multi-dimensional arrays (i.e. structured data sets) to parallel processors and for specifying data parallel operations. Extensions to HPF are being developed that offer a similar functionality for more complex data structures, e.g. multi-block grids [6]. FORGE 90 is a software tool for the analysis and the (semi) automatic parallelisation of existing sequential codes: based on a user defined partitioning of the data arrays, FORGE allows interactive or automatic selection of do-loops to be parallelised [7].

2.3 Description of some parallel systems

Intel Paragon. The Intel Paragon is a distributed memory system in which the processing nodes are interconnected in a 2D mesh network, see Fig. 2. A Paragon system with 1874 processors is operational at Sandia Nat. Labs. Two types of processing nodes

are available, both based on the Intel i860 processor. General Purpose nodes contain 2 processors (one for calculation and one for communication) and an I/O expansion port. Multiprocessor nodes contain two processors for calculation (with shared memory) and one for communication. Wormhole message passing through the network is carried out by Mesh Router Chips, one for each node. The processing nodes are logically divided into a compute partition (for parallel program execution), an I/O partition (nodes designated to disk I/O and networking) and a service partition (interactive use, compilation).
The distributed operating system provides a single system image (single process ID space, single file system, etc.) and automatic scheduling of jobs. The distributed memory programming model is supported via Intel's NX communication system or via the SUNMOS environment (Sandia). The Parallel Development Environment contains various tools for software development and performance monitoring. For more information, see e.g. [8].

Cray T3D. The Cray T3D is a distributed memory parallel system with 32 to 2048 processing nodes. The processing nodes (DEC Alpha processors) are connected by a bidirectional 3D torus (periodic mesh) network (each switch of the network is shared by two nodes), see Fig. 3. Various mechanisms are implemented to reduce the communication cost over the interconnection network and to synchronise processing nodes. The memory is physically distributed, but is globally addressable. Hence, three programming models are supported: SIMD (date parallel), shared memory MIMD and distributed memory MIMD programming styles. The software environment includes a Fortran compiler with Fortran 90 features (array syntax, etc.) which allows the user to mix all three programming models in one program. Also included are PVM, a performance analyser, etc. The T3D system needs a Cray vector processor as host system.

IBM SP2. The IBM SP2 is a distributed memory system with up to 128 processing nodes. Two types of processing nodes are available, 'thin nodes' and 'wide nodes', both based on the POWER2 processor. Wide nodes allow larger memories, provide a faster processor-to-memory connection and allow to attach various storage devices. The nodes are interconnected by a 'High-Performance Switch', see Fig. 4. The switch is a multi-stage omega network that performs wormhole routing. The available communication bandwidth over the switch scales linearly with the number of processors. Support for short messages with low latency and minimal message overhead is provided. For more information, see e.g. [9].
The AIX Parallel Environment contains a Message Passing Library (MPL), performance monitoring and visualisation tools. An optimised version of PVM is also available. Only the distributed memory programming model is supported. Job scheduling support is provided by the 'Loadleveler' software.

Silicon Graphics Power Challenge. The Silicon Graphics Power Challenge systems are shared memory multiprocessors, with up to 18 processors (MIPS R8000), having multiple functional units, that can operate simultaneously. Each processor has a cache hierarchy with a small, fast on-chip cache and a large, slower but pipelined off-chip cache. The main memory can be up to 8-way interleaved.
The Fortran and C compilers are able to restructure programs to reduce cache misses by interchanging loops, by 'tiling' or 'blocking' in case of nested loops, etc. (Loop blocking is a technique for optimising the performance of the memory hierarchy, in case of e.g. operations on matrices.) Further, the compilers support automatic and user-directed (via directives) parallelisation of Fortran and C programs. For more information, see e.g. [10].
Up to eight Power Challenge systems can be interconnected by a (switch-based) communication network, forming a 'CHALLENGEarray' system. Communication over this network must be programmed in distributed memory style using message-passing libraries (PVM, MPI) or using High-Performance Fortran.

Convex Examplar. The Convex Exemplar consists of a number of hypernodes, connected to each other via a low latency ring network with four interleaved links. Each hypernode is a shared-memory multiprocessor, consisting of 8 processors (HP PA-RISC 7200) that are connected to 4 memory modules by a crossbar, see Fig. 5.
The Exemplar programming environment provides both shared memory and distributed memory programming support. For message passing, the PVM communication library is used. The shared memory programming environment is implemented through what is called 'Global Shared Distributed Virtual Memory'. An application, programmed in shared memory style, can use processors located on various hypernodes. In that case, three levels in the memory hierarchy are used: the large cache of a particular processor, the global memory of the hypernode to which the processor belongs and memories located on different hypernodes.
The time needed to access data located on a different hypernode is higher than to access data within a hypernode. In order to reduce the delay caused by using the ring interconnect, each hypernode contains a cache of memory references made over the interconnect. The information hold in this cache can be used to locate any global data that is currently encached in the hypernode. The system automatically maintains cache coherence between multiple hypernodes.

2.4 Parallel performance parameters

The quality of a parallel implementation is often measured by the achieved *speedup* or *efficiency*.

The *parallel speedup* achieved by a parallel algorithm running on P processors is defined as the ratio of the execution time of the parallel algorithm on a single processor and the execution time of the parallel algorithm on P processors. The *parallel efficiency* is equal to the speedup divided by P. We have thus the following definitions for the *parallel speedup* $S(n, P)$

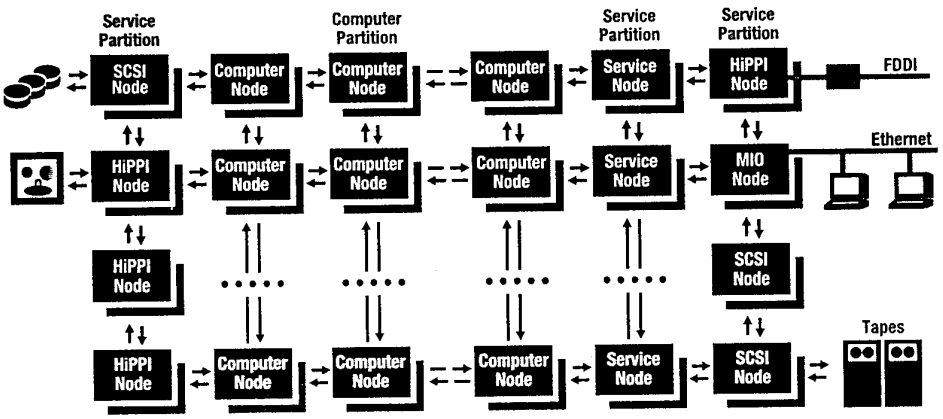


Fig. 2: *The architecture of the Intel Paragon, based on a two-dimensional interconnection network.*

and the *parallel efficiency* $E(n,P)$

$$S(n,P) = \frac{T(n,1)}{T(n,P)}$$
$$E(n,P) = \frac{S(n,P)}{P} = \frac{T(n,1)}{P\,T(n,P)} \qquad (1)$$

where n denotes the problem size, $T(n,1)$ and $T(n,P)$ denote the execution times of the algorithm on one and P processors respectively.

Note that (1) does not give any information about the quality of the parallel algorithm. It solely measures how well an algorithm has been parallelised. As such, it should always be complemented with data which indicate the *numerical efficiency* of the parallel algorithm, which can be defined as the ratio of the following single processor execution times: $T_{best}(n)/T(n,1)$, where $T_{best}(n)$ denotes the time taken by one processor of the parallel computer executing the fastest known sequential algorithm. Combination of the definitions of parallel speedup (or efficiency) and numerical efficiency leads to the notion of *total speedup* and *total efficiency*, defined by

$$\bar{S}(n,P) = \frac{T_{best}(n)}{T(n,P)}$$
$$\bar{E}(n,P) = \frac{\bar{S}(n,P)}{P} = \frac{T_{best}(n)}{P\,T(n,P)}\,. \qquad (2)$$

Practical considerations limit the usefulness of the latter definitions. First of all, it is often very difficult to determine what algorithm is the best sequential one; this may depend on the problem size n, on the particular hardware used, on implementation issues, etc. Moreover, the notion of 'best' algorithm may change in time, as better algorithms become available. Also, a good implementation of that algorithm is not always available. In practice one can define the total speedup by using the execution time of a good sequential algorithm instead of $T_{best}(n)$.

If we assume that a P-processor machine cannot execute more than P times faster than a single processor machine, we obviously have that $S(n,P) \leq P$ and $E(n,P) \leq 100\%$. We now enumerate some overheads that may cause a deviation from linear speedup.

• **the sequential fraction.** The speedup achievable on a parallel computer can significantly be limited by the existence of a small fraction of inherently sequential code which cannot be parallelised. This is expressed by *Amdahl's law*, see e.g. [11]:

> Let α be the fraction of operations in a computation that must be performed sequentially, where $0 \leq \alpha \leq 1$. The maximum speedup achievable by a parallel computer with P processors is then limited as follows,
>
> $$S(n,P) \leq \frac{1}{\alpha + (1-\alpha)/P} \leq \frac{1}{\alpha}\,. \qquad (3)$$

For example, when 10% of the code must be executed sequentially, the maximum speedup is limited by 10, independent of the number of processors available.

Amdahl's law has been a central argument of people doubting the usefulness of massively parallel systems. Their criticism is justified as long as one considers solving a particular problem of a fixed size (i.e., with a constant value of α). In actual practice, however, this is rarely the case, as problem sizes tend to scale with the number of processors and with the computing power available. (Large scale parallel systems are used to solved bigger problems than the ones solved on small-scale parallel systems.)

For many computational problems the sequential fraction α rapidly goes to zero as the problem size increases. Consequently, when problem scaling is in effect, α depends on the number of processors, and (3) looses much of its significance. An alternative to Amdahl's law was formulated by Gustafson et al. [12][13].

> Let $\bar{\alpha}$ denote the sequential fraction of the time spent during a computation on a par-

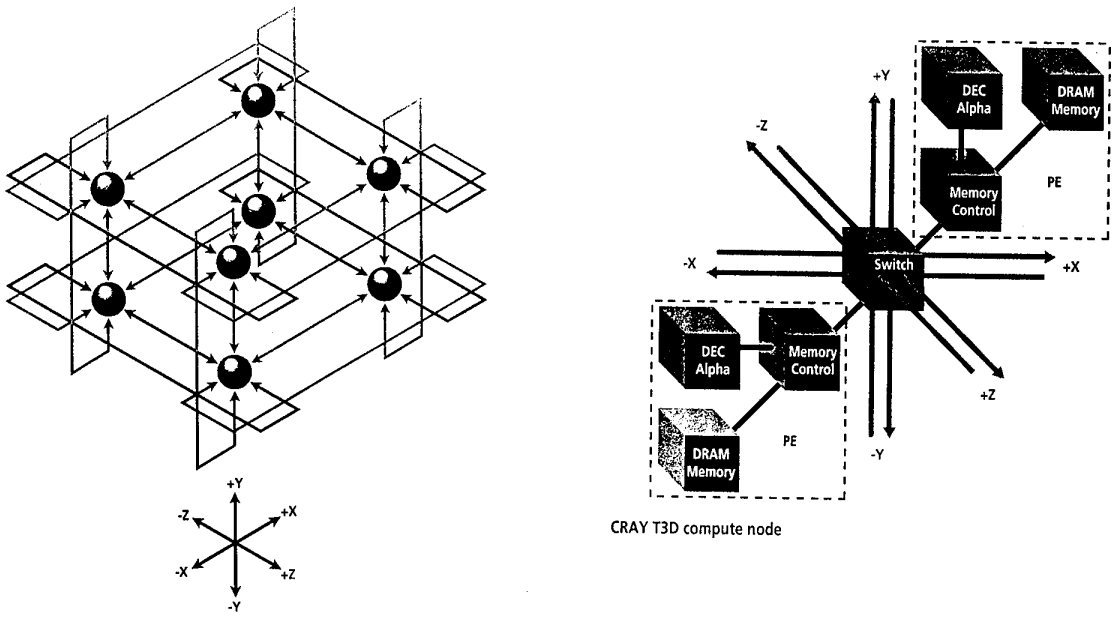


Fig. 3: *The architecture of the Cray T3D (three-dimensional interconnection network).*

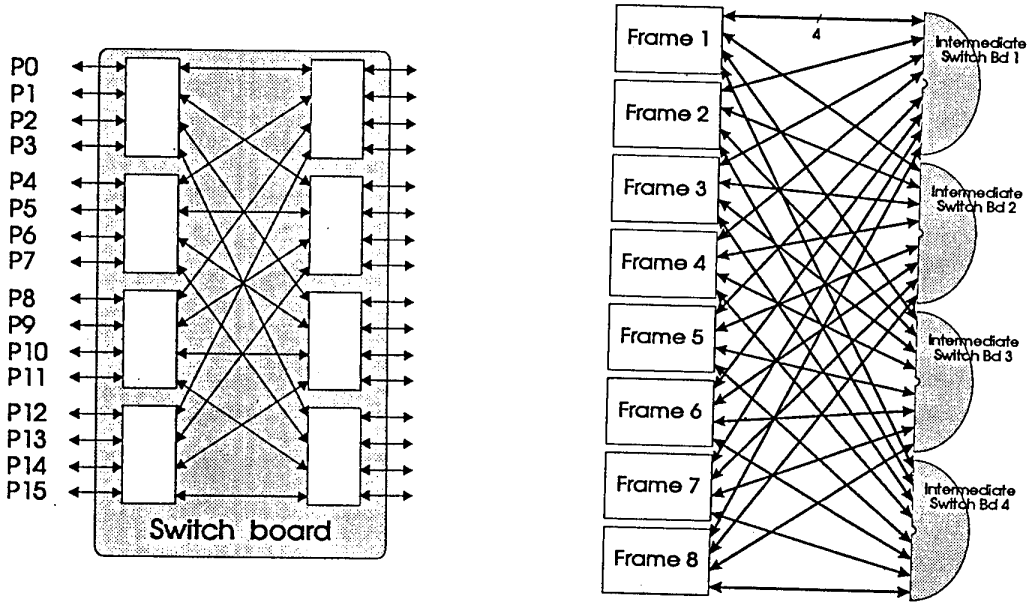


Fig. 4: Left: *A 16-node bidirectional multi-stage network, forming the basic building block ('frame') for the High-Performance Switch in the IBM SP2.* Right: *Twelve frames are used to interconnect 128 processors.*

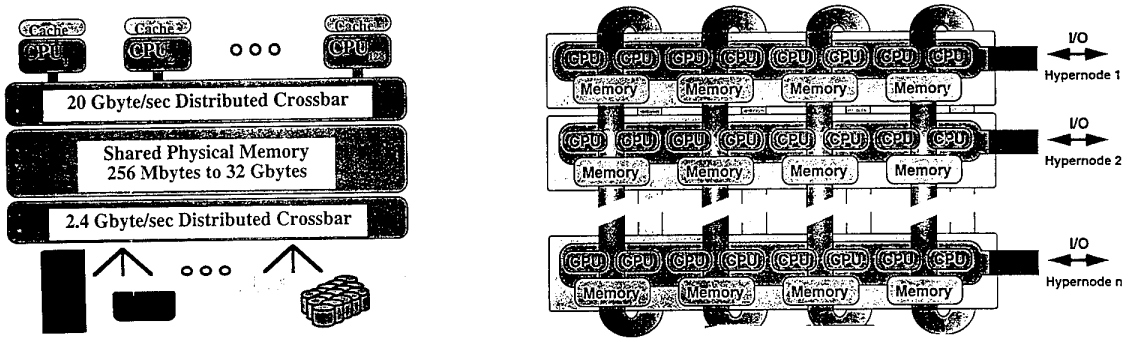


Fig. 5: *The architecture of the Convex Exemplar : logical system view (left) and physical system view (right).*

allel system with P processors. The maximum speedup achievable is then limited as follows,

$$S'(n,P) \leq P\,(1-\bar{\alpha}) + \bar{\alpha}\,. \qquad (4)$$

$S'(n,P)$ is usually called the *scaled speedup*. It is equal to the ratio $T'(n,1)$ over $T'(n,P)$, where $T'(n,1)$ is the time the parallel program would take to run on a single processor if sufficient resources (memory) were available.

In large scale applications, $\bar{\alpha}$ is often a small number, and very high scaled speedups are attainable on large-scale parallel processors. Fig. 6 shows the dependence of $S(n,P)$ and $S'(n,P)$ on the serial fraction α, resp. $\bar{\alpha}$.

• **non-optimal algorithms and algorithmic overhead.** The best sequential algorithm may often be difficult or impossible to parallelise (e.g., Thomas algorithm for solving tridiagonal linear systems). In that case the parallel algorithm may have a larger operation count than the sequential one. Additionally, in order to avoid communication overhead one may wish to duplicate some calculations on different processors, rather than having one processor doing the calculation and then distributing the result (e.g. 'double flux calculations', see further).

• **software overhead.** Parallelisation often results in an increase of the (relative) software overheads such as the overheads associated with indexing, procedure calls, etc. Also, this approach usually results in shorter loops, thus restricting vector lengths. This reduces the potential gain of using vectorisation.

• **load imbalance.** The execution time of a parallel algorithm is determined by the execution time of the processor having the largest amount of work. As soon as the computational workload is not evenly distributed, load imbalance will result, and processor idling will occur : processors must wait for other processors to finish a particular computation.

• **communication and synchronisation overhead.** Finally, any time spent in communication and synchronisation is pure overhead.

In the next section, we will discuss in detail these various sources of overhead.

3 PARALLELISATION OF GRID-ORIENTED PROBLEMS

3.1 Introduction

In the remainder of this text, we will focus on *distributed memory parallelism* for two reasons. Firstly, parallel systems with only distributed memory systems have an 'extreme' parallel architecture. Secondly, in the distributed memory programming model, the parallelism must be introduced explicitly in the application program. Algorithms designed for distributed memory systems will also perform well on shared memory (or hybrid) systems. Data partitioning, which is necessary for distributed memory systems, is also beneficial for shared memory systems. For example, entire matrices typically do not fit into the cache. The performance of the memory hierarchy can be optimised, by decomposing the matrix operations into submatrix operations, with a submatrix size chosen so that the operands fit in the cache.

The basic issues of parallel algorithm design are nowadays well understood and are described in various books and papers. The book of G. Fox et al. [14] is a key reference (although somewhat outdated). The textbooks of E. Van de Velde [15] and C. de Moura [16] also provide a good introduction. The proceedings of the yearly conferences on Parallel Computational Fluid Dynamics give an overview of research and achievements in this field [17, 18, 19]. Also the proceedings of the Scalable High Performance Computing Conferences [20, 21], the SIAM Conferences on Parallel Processing for Scientific Computing [22, 23] and the HPCN conferences [24, 25] are valuable sources of information.

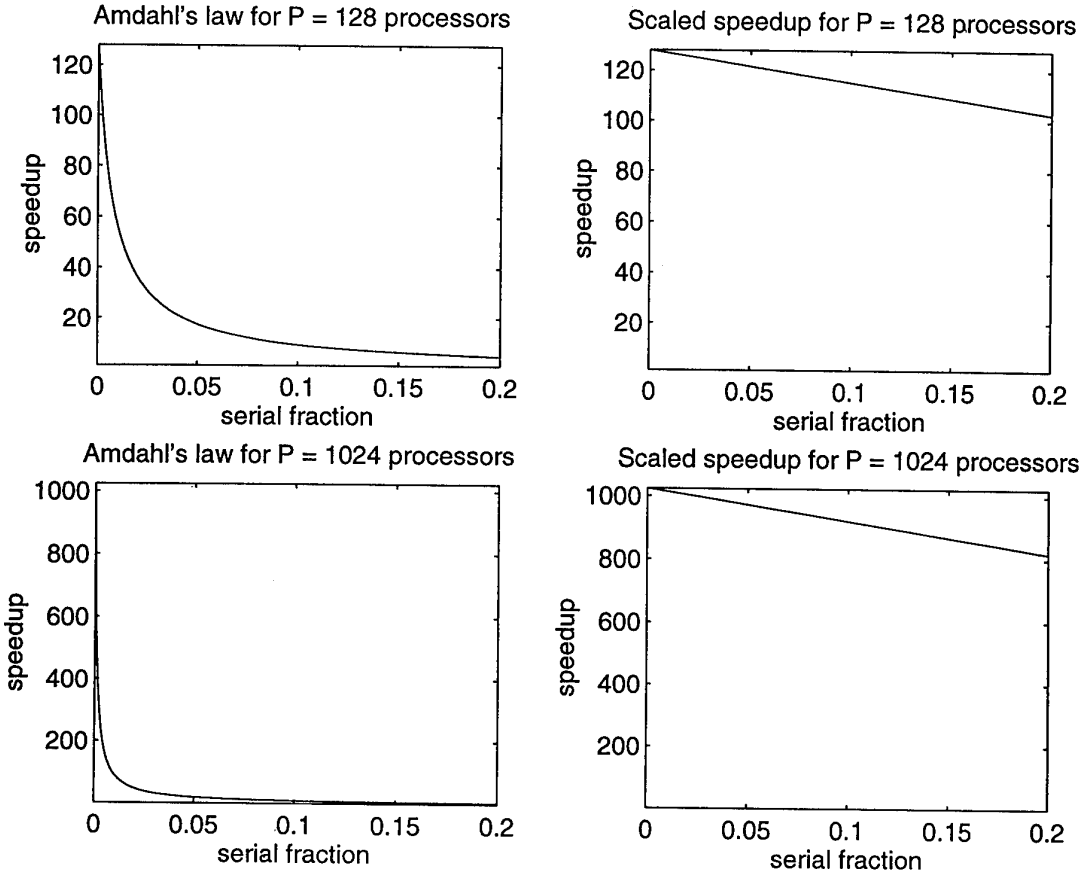


Fig. 6: *Dependence of the speedup on the sequential fraction for $P = 128$ and $P = 1024$. Left : parallel speedup $S(n,P)$; right : scaled speedup $S'(n,P)$.*

For grid-oriented problems, such as the numerical solution of partial differential equations, the data are defined on a discrete grid of grid points or finite volumes or finite elements. In this paper, we will use the term *(grid) point* as a generic name for a grid point, finite volume or element and its data.

Parallelisation of grid oriented applications is seriously facilitated because the calculations on a grid point typically involve only grid points that are geometrically adjacent. Parallelisation is achieved by *partitioning* the grid into *subdomains* (subgrids) and assigning these subdomains to the processors of the parallel system. Each processor performs the calculations associated with the subdomain(s) assigned to that processor. Dependency (and communication) between subdomains is restricted to the perimeters of the subdomains.

Many important issues concerning parallelisation of grid-oriented problems and performance analysis of parallel algorithms can be understood by studying the parallel execution of a 'model problem', representing the explicit time-integration of a finite difference or finite volume discretisation of a partial differential equation on a structured grid.

Assume that a 2D structured grid is partitioned in subdomains of equal size, such that each processor deals with $n_x \times n_y$ grid points or cells. Assume further that the explicit time-integration is based on a five-point stencil, i.e.

$$u_{ij}^{(k+1)} = f(u_{i-1j}^{(k)}, u_{i+1j}^{(k)}, u_{ij-1}^{(k)}, u_{ij+1}^{(k)})$$

Due to the local nature of the calculations, each processor can perform the updates for all interior grid points (the white area in Fig. 7). The other grid points of the subdomain are called (subdomain) 'boundary grid points'. In order to perform the updates of the subdomain boundary grid points, the processor must know also function values $u_{ij}^{(k)}$ corresponding to grid points lying at the other side of the subdomain boundaries. This information must be received from the neighbouring processors and can be stored in the overlap regions indicated in Fig. 7. On the other hand, the boundary grid points must be send to neighbouring processing nodes, where they are part of the overlap region. Hence, before each integration step, neighbouring processors *exchange* information with each other, see Fig. 8.

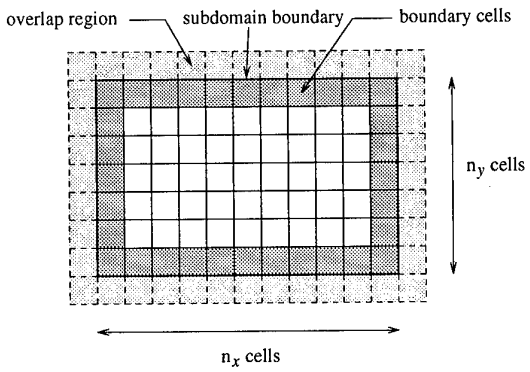


Fig. 7: *Grid partitioning and communication requirements.*

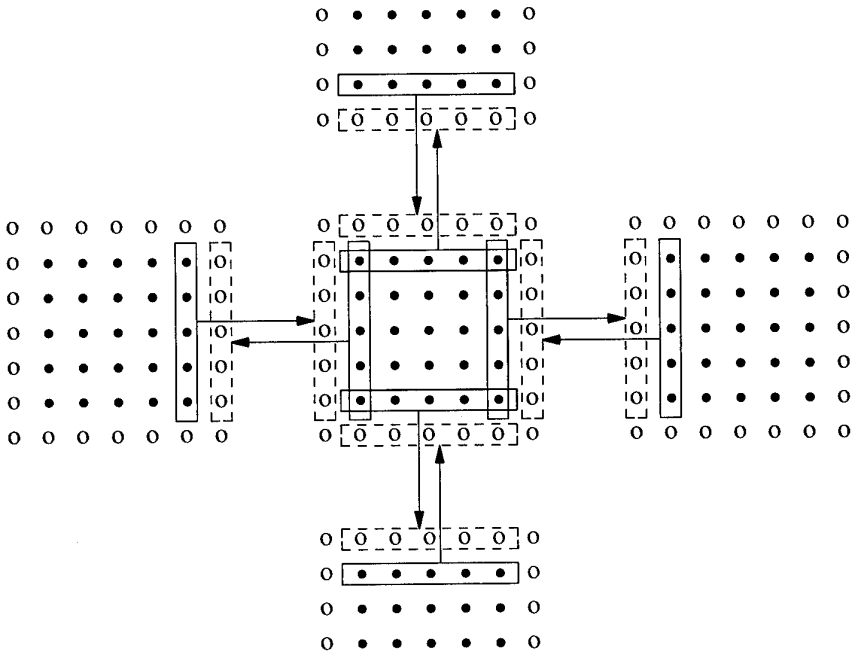


Fig. 8: *Concurrent exchange of local boundaries.*

3.2 Analysis of the communication overhead

For the simple model problem described above, the execution time of the algorithm for a problem size denoted by n, on a parallel system with P processors can be written as

$$T(n,P) = T_{calc} + T_{comm}$$

where T_{calc} denotes the calculation time and T_{comm} denotes the time spent in communication. Assuming that no other overhead occurs except communication of the overlap regions, the sequential execution time is

$$T(n,1) = P \cdot T_{calc}$$

Hence the speedup and parallel efficiency are given by

$$S(n,P) = \frac{PT_{calc}}{T_{calc} + T_{comm}} = \frac{P}{1 + \frac{T_{comm}}{T_{calc}}} = \frac{P}{1+f_c}$$

$$E(n,P) = \frac{1}{1 + \frac{T_{comm}}{T_{calc}}} = \frac{1}{1+f_c}$$

where $f_c = T_{comm}/T_{calc}$ denotes the communication overhead. If f_c is small, a nearly optimal speedup ($S(n,P) \simeq P$) and efficiency ($E(n,P) \simeq 1$) is obtained.

The amount of data sent and received per processor is proportional to the number of boundary cells, while the amount of computations performed by each processor is proportional to the number of interior cells. For the model problem we have

$$T_{calc} = c_1 \; n_x n_y \; t_{calc}$$

$$T_{comm} = c_2 \; \cdot 2(n_x + n_y) \; t_{comm}$$

where t_{calc} represents the time required to perform a floating point operation, t_{comm} denotes the time needed to communicate one floating point number, and c_1, c_2 are constants. This leads to the important

formula

$$f_c = \frac{c_2}{c_1} \frac{2(n_x + n_y)}{n_x \times n_y} \frac{t_{comm}}{t_{calc}} \qquad (5)$$

which indicates that the overhead depends on 3 factors:

1. the size of the subdomain: large subdomains have a small 'perimeter to surface' ratio $2(n_x + n_y)/n_x n_y$, leading to a small value for f_c;
2. the machine characteristic t_{comm}/t_{calc}, indicating how fast communication can be performed compared with floating point operations;
3. the algorithm via the ratio c_2/c_1. The overhead f_c will be small for problems for which many floating point operations per grid point must be performed (c_1 large), compared with the amount of data to be communicated per grid point (represented by c_2).

Remark. An important characteristic of most communication systems is the rather high *message startup time*. The cost of sending a message between neighbouring processors can be written as

$$T(n) = t_{startup} + n\ t_{send}$$

where n indicates the length of the message (number of words transferred), $t_{startup}$ is the message startup time (caused by hardware and software delays) and t_{send} is the marginal communication time per word. For many systems $t_{startup}$ is much larger than t_{send} (even by a factor 1000). An immediate conclusion is that sending many short messages should be avoided if possible.

In (5) t_{comm} denotes the average time to communicate one word. This clearly depends very much on the average length of the messages that are sent : for small messages $t_{comm} \simeq t_{startup}$, while for very large messages $t_{comm} \simeq t_{send}$. This must be taken into account when analysing parallel algorithms by using (5).

A further analysis of this model problem reveals some important guidelines that should be taken into account when parallelising CFD algorithms.

Different grid partitioning strategies. For this model problem, the communication volume is proportional to the number of grid points on the (interior) subdomain boundaries, i.e. proportional to the 'perimeter' of the subdomain. When the size of the subdomain is fixed, the perimeter (and thus the communication volume) is minimal if the number of grid points in each direction is equal, i.e. $n_x = n_y$. We will use the term 'square subdomain' to denote the latter case. Hence, partitioning into square subdomains leads to a minimal communication volume.

This observation can be generalised as follows. A stripwise (or one-dimensional) partitioning (Fig. 9, left) yields subdomains with long boundaries but with at most two adjacent subdomains. A blockwise (or two-dimensional) partitioning (Fig. 9, right) gives subdomains with shorter boundaries but with up to four neighbours. Thus a blockwise partitioning minimises the total communication volume, while a stripwise partitioning minimises the number of messages. What will be the best choice depends on the characteristics of the problem and of the parallel computer. When the message startup time dominates the communication time per message the stripwise partitioning will be beneficial.

Note that the communication requirements are not always 'isotropic' in all directions, but they may depend on characteristics of the problem or the numerical algorithm. This may influence the partitioning strategy. Consider for example the solution of the compressible Navier-Stokes equations around an airfoil. The inclusion of an algebraic turbulence model may lead to a global dependence (and communication) in the direction perpendicular of the airfoil. Then a stripwise partitioning is to be preferred.

Dependence on the size of the subdomains. When each subdomain contains $N = n_x \times n_y$ gridpoints and a blockwise partitioning is used with square subregions ($n_x = n_y$), Eq. (5) yields

$$f_c \propto \frac{1}{\sqrt{N}} \qquad (6)$$

This indicates that the communication overhead f_c remains constant, independent of the number of processors, as long as the size of the subdomains remains constant ! Of course this implies that to maintain a given parallel efficiency, the total problem size M must grow when the number of processors P grows, since $M = N \times P$. The relation $f_c \propto \frac{\sqrt{P}}{\sqrt{M}}$ also indicates that, for fixed (total) problem size M, the efficiency and speedup decrease when the number of processors increases (cf. Amdahl's law). This analysis is only valid when the only communication is the exchange of information between neighbouring processors. Any 'global communication' (e.g., the collection of the local residuals to compute the global residual) implies an overhead which grows with increasing number of processors. However, the relative importance of such global communications is often very low and does not really affect the overall speedup and efficiency.

Extension to larger stencils and to 3D grids. The analysis presented above remains valid when other computational stencils are used instead of a 5-point stencil [14]. It may be necessary to use a larger overlap region, (e.g., with a width of 2 points). In that case the communication volume increases (and the constant c_2), but the number of operations per cell (and thus the constant c_1) also increases. Hence, the communication overhead does not necessarily grow.

In case of three-dimensional grids, 1D-, 2D- and 3D-partitionings can be used. The communication volume is then determined by the 'surface to volume' ratio of the subdomains, leading to a factor $N^{1/3}$ in Eq. (6), see e.g. [14].

Dependence on the machine characteristics. The speedup and parallel efficiency of a given al-

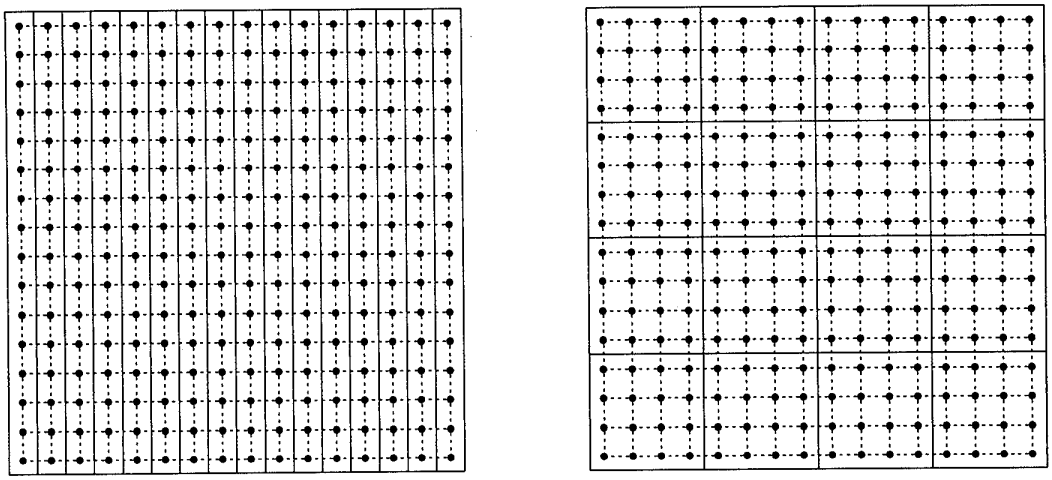


Fig. 9: *Strip- and blockwise partitioning of a grid.*

gorithm is proportional to the machine characteristic t_{comm}/t_{calc}. The various parallel systems available have quite different values for this characteristic. Thus the communication overhead may vary substantially on different machines.

Note that computer manufacturers may upgrade alternatively the processors and the communication network of their systems. Upgrading the processors without also increasing the communication speed, may result in an 'unbalanced' system with a large ratio t_{comm}/t_{calc}.

Dependence on the problem characteristics. Computational Fluid Dynamics applications are characterised by a high number of floating point operations per grid point or cell per iteration, while only a few variables are associated with each point. Thus the factor c_2/c_1 in the communication overhead will be small.

As a result, minimisation of the communication overhead does not always influence the speedup and parallel efficiency very much. However, it is always important to minimise the work load imbalance. Below we show that in general a blockwise partitioning also minimises the load imbalance. Thus in many cases minimisation of communication overhead and minimisation of the load imbalance go hand in hand.

3.3 Analysis of the load imbalance

Let $T_i^{calc}, i = 1 \ldots P$, denote the time spent by the i-th processor in calculation, and let $T_{average}^{calc}$ and T_{max}^{calc} denote respectively the average and the maximum calculation time for the P processors. The *load balance factor* is defined as

$$\lambda(n,p) = \frac{T_{average}^{calc}}{T_{max}^{calc}} \tag{7}$$

The load balance factor is a good estimate for the parallel efficiency, if the number of operations to be performed (counted sequentially) does not depend on the number of processors, and if the communication time can be neglected. Indeed, in this case the parallel efficiency is given by

$$E(n,p) = \frac{T(n,1)}{P\,T(n,P)} \approx \frac{\sum_{i=1}^{P} T_i^{calc}}{P\;max T_i^{calc}} = \frac{T_{average}^{calc}}{T_{max}^{calc}}$$

and thus $E(n,p) \approx \lambda(n,p)$.

Note that a commonly made mistake is to measure load (im)balance by comparing the maximum and the *minimum* calculation times.

In many applications, the processors are (implicitly) synchronised by the communication needed to update the 'overlap regions' at the end of each step of an iterative procedure. In that case, we can determine the efficiency and speedup by analysing one iteration step. Assume now that the amount of work per grid point is constant, and that the communication time can be neglected. We then obtain

$$E(n,P) = \frac{T(n,1)}{P\,T(n,P)} \approx \frac{M}{P\,N_{max}} = \frac{N_{average}}{N_{max}}$$

where M is the total number of grid points, N_{max} is the maximum number of grid points in a subdomain and $N_{average} = M/P$.

Assume that a rectangular grid is distributed among P processors. If the grid cannot be equally distributed among the processors, then a blockwise partitioning leads to a higher load balance factor than a stripwise partitioning. A partitioning into square subdomains will lead to a maximal load balance factor, i.e. a minimal load imbalance.

The treatment of boundary conditions is also a potential source of load imbalance. Indeed, in general the computational work to be done for boundary cells differs from the work for interior cells. In order to minimise the load imbalance caused by the treatment of the boundary conditions, the boundary cells should be distributed as equal as possible among the processors. This is achieved when the subdomains are (nearly) square.

The assumption that the amount of work per grid point or cell is constant is not always valid in CFD.

For example, the computational effort may differ for cells lying in a subsonic region and in a supersonic region. This can cause load imbalance, which cannot be accurately predicted beforehand. Similar problems arise when the mathematical model differs in various parts of the domain, e.g. when chemical reactions are taken into account in high-temperature zones.

3.4 Numerical efficiency of parallel algorithms

Until now, we have discussed how the parallel overhead affects the performance, by comparing the parallel execution time with the time needed by the *same algorithm* on one processor.

In many cases the algorithm used on the parallel machine is different from the one typically used on a sequential machine. In order to obtain acceptable parallel efficiencies, sequential algorithms are often modified to decrease the communication volume or the number of synchronisation points. This may deteriorate the *numerical* efficiency of the algorithm. It may even be necessary to use a rather different algorithm – with different numerical properties: number of operations, convergence properties, etc. – on a parallel computer, if the sequential algorithm cannot be parallelised easily and efficiently.

Explicit methods. Explicit methods are inherently parallel and the numerical properties are not affected by parallelisation (grid partitioning), when all necessary communication is performed. For example, communication is needed after each substep of a Runge-Kutta method. One can reduce the communication overhead by updating the overlap regions only after a complete time-integration step. Omitting some communication can result in slightly worse convergence properties, but can lead to a higher 'total speedup'. The effect is very problem dependent. Note that this technique results in a 'block-structured' approach, but here the number of blocks is determined by the number of processors, not by the geometry of the domain.

Implicit methods. The situation is more complex when implicit methods are used.

- Assume that the resulting linear systems are solved by a point relaxation scheme. *Jacobi relaxation* is inherently parallel. In this case the communication requirements are exactly the same as in the model problem described above (exchange of the overlap regions). *Gauss-Seidel* relaxation usually has better convergence properties. On a sequential computer, a Gauss-Seidel iteration typically sweeps through the grid cells in lexicographic order. On a vector or parallel computer, a *Red-Black ordering* of the grid points is necessary. All 'red' points can be updated in parallel, and afterwards the 'black' points can be updated. The convergence rate of lexicographic and Red-Black Gauss-Seidel can differ substantially. This will be illustrated in the section 4.

- When *line relaxation schemes* are used, (block) tridiagonal systems must be solved. This leads to data dependencies between the grid points lying on the same gridline. If one only sweeps in one direction, the tridiagonal systems — and the associated data dependencies — only occur along that direction. By using a stripwise partitioning, one can ensure that each tridiagonal system belongs to only one processor. Then each system can be solved by the Thomas algorithm (i.e. Gaussian elimination), which is the optimal sequential solver.

  The parallelisation of line relaxation is not so easy, if a blockwise partitioning is used, or if one performs line relaxation in different directions. Then (part of) the tridiagonal systems are distributed among processing nodes. Parallel solvers for (block) tridiagonal systems have been developed, based on substructured Gaussian elimination and/or on cyclic reduction [26][27]. However, the operation count of these solvers is ± 2 times higher than for the Thomas algorithm — hence their numerical efficiency is low — and they contain a sequential part. Since many tridiagonal systems must be solved, the latter drawback can be avoided by distributing the sequential parts equally over the processors (at the expense of some communication). Attempts are made to reduce the computational cost of the parallel algorithms by using approximate solvers [28][29].

  An alternative is to solve the set of tridiagonal systems by using the Thomas algorithm in a *pipelined fashion*. This strategy however requires the communication of many short messages and leads to some load imbalance (during the start-up and the end phase of the pipeline). Another alternative strategy to solve tridiagonal systems oriented in two directions goes as follows. We know that when a stripwise partitioning of the data is used, the tridiagonal systems oriented in one of the two directions can be solved by the Thomas algorithm. The Thomas algorithm can be used to solve the tridiagonal systems in both directions, if in both phases of the algorithm a different stripwise partitioning is used, such that in each phase a tridiagonal system is stored in only one processor. This requires that a complete 'data transposition' is carried out between both phases. The communication volume of the data transposition is proportional with the number of grid points per processor. Since the same holds for the calculation cost, the parallel efficiency may still be acceptable. The latter strategy is the most efficient one (in terms of *total efficiency*) to implement the semi-implicit ADI time integration scheme on finite difference grids with irregular boundaries [30].

- Another example of the interaction between numerical and parallel aspects can be found in multigrid. W-cycles are usually more efficient than V-cycles in terms of work-units needed to achieve convergence. On a parallel computer

however, W-cycles result in poor parallel efficiency and one therefore frequently resorts to V-cycles despite their inferior numerical properties [31].

- Parallel algorithms for solving partial differential equations can also be based on *domain decomposition* in the mathematical sense. Two approaches are possible.
  In the Schwartz domain decomposition approach, overlapping subdomains are used. The differential equations are solved on each subdomain separately, using an approximation for the solution at the subdomain boundaries. The resulting approximate solutions provide a new approximation for the solution on the boundaries of the (overlapping) neighbouring subdomains. This process must be repeated in an iterative way.
  In the Schur Complement approach, non-overlapping subdomains are used. The subdomain problems are solved in terms of the variables on the borders of the subdomains. After computation of the variables on these borders (interface or 'Schur complement' problem), the variables on the subdomains can be determined. Note that both domain decomposition approaches often require *extra calculations* compared to when no decomposition is used. These extra calculations must be considered as *algorithmic overhead* caused by the parallelisation. Domain decomposition techniques for CFD problems are described in e.g. [32, 33, 34, 35].

4 EXAMPLES

In this section we illustrate some of the concepts introduced in the previous sections. We first discuss the parallel performance of an explicit Euler Solver on Intel iPSC/2 and iPSC/860 distributed memory computers. We show that 'double flux calculations', caused by the grid partitioning, may form a substantial algorithmic overhead in the parallel code. We comment on various approaches to measure the parallel performance and we introduce the *effectivity* as an alternative performance measure.

We then present results of experiments on the parallelisation of implicit Euler solvers. We discuss the achieved parallel speedup and parallel efficiency, but we also show how the numerical efficiency of parallel algorithms may influence the *total speedup* and *total efficiency*, which is a better measure for the actual performance.

Finally, we describe results obtained with a block structured Euler solver, in which an adaptive block refinement procedure leads to the creation of new blocks. We show that the use of *mapping heuristics* allows to map the block structure onto the processors of the parallel machine, such that the load imbalance and the communication cost is low.

4.1 Parallel performance of an explicit Euler Solver

We first describe some experiments with a parallel multi-block explicit Euler solver [36, 37, 38]. We have used the following schemes:

Scheme 1) a first order upwind discretisation with Van Leer flux vector splitting, combined with a forward Euler time integration with local timestepping;

Scheme 2) a second order Roe scheme, with a min-mod limiter on characteristic variables, combined with a five stage Runge-Kutta time integrator.

In the parallel version of the Runge-Kutta scheme, communication occurs only once per time step, i.e. before the first stage. Scheme 2 has a much higher ratio of calculation time to communication than scheme 1.

We have used the following testcase: transonic flow around the NACA0012 airfoil, with boundary conditions : $M = 0.80$ (Mach number), angle of attack of $1.25°$, $T_0 = 278K$ (total temperature) and $P_0 = 150000Pa$ (total pressure). The structured C-grid (240×19 cells) shown in Fig. 11 can be split in $2, 4, \ldots, 16$ blocks of equal size (1D partitioning, orthogonal to the airfoil), see Fig. 11a). Thus all these partitionings allow a nearly perfect calculation load balance.

Two sets of tests were done : the first set with $N = P$ blocks and the second set with $N = 16$ blocks regardless of the number of processors, P. The first case corresponds to a situation where the grid is partitioned for parallel processing purposes only. Extra calculations caused by the partitioning must be considered as algorithmic overhead, as discussed below. The second case corresponds to a situation where the partitioning into blocks results from physical considerations. A sequential code would use the same partitioning into blocks.

4.1.1 Algorithmic overhead: double flux computations

We first discuss a typical 'algorithmic overhead' that occurs in parallel CFD codes. At subdomain boundaries, the parallel code cannot exploit the symmetry properties of the numerical fluxes. The fluxes are calculated twice on every edge of a subdomain boundary, once in every block. If the grid is partitioned for parallel processing purposes only, these 'double flux computations' form an overhead, not present in the sequential code. Experiments indicate that the time required for those extra flux computations is of the same order of magnitude as or even larger than the communication time, see [37] and Tables 1 and ??.

Suppose that the number of blocks N is equal to the number of processors P. The time lost in the extra flux computations and some additional overhead introduced by splitting the grid in subdomains is given

Number of processors	scheme 1	scheme 2
2	0.7	4.8
4	1.6	9.0
8	2.4	12.6
16	2.7	14.6

Table 1: *Double flux computation overhead, defined as* $(t^{calc}(N=P,P) - t^{calc}(1,1))/t^{comm}$, *for the two explicit schemes on iPSC/2.*

Number of processors	scheme 1	scheme 2
2	4.7	36.9
4	3.8	33.3

Table 2: *Double flux computation overhead, defined as* $(t^{calc}(N=P,P) - t^{calc}(1,1))/t^{comm}$, *for the two explicit schemes on iPSC/860.*

P	$N=P$			$N=16$		
	$\alpha(P,P)$ (%)	$E(P,P)$ (%)	$S(P,P)$	$\alpha(16,P)$ (%)	$E(16,P)$ (%)	$S(16,P)$
1	99.9	100	1	98.7	100	1
2	99.7	99.8	1.995	98.4	99.8	1.996
4	99.1	98.5	3.94	98.1	99.5	3.98
8	97.9	96.3	7.70	97.1	98.4	7.87
16	95.9	90.2	14.43	95.9	97.1	15.54

Table 3: *Effectivity* α, *efficiency* E *and speedup* S *on the iPSC/2 for first order Van Leer, Euler time integrator.*

by $t^{calc}(N=P,P) - t^{calc}(1,1)$, where $t^{calc}(N,P)$ denotes the total calculation time on P processors for a grid partitioned into N blocks. (Note that $t^{calc}(N,P)$ is equal to the sum of the sequential calculation times for all blocks; communication time and processor idle time is not taken into account.)

The double flux computation overhead on the iPSC/2 for the two schemes mentioned above is given in Table 1. The results show that the time spent in the extra flux calculations in scheme 1 (first order Van Leer, forward Euler) is of the same order of magnitude as the communication time, while for scheme 2 (second order Roe, Runge Kutta) the double flux computation overhead is much larger than the communication overhead. For a Navier-Stokes computation, the double flux computation overhead would become even more dominant. Table 2 shows the results obtained on an iPSC/860 system, for which both calculation and communication are much faster than on the iPSC/2. For this example, the double flux computation overhead is even larger than on iPSC/2 systems. Note however that the code has not been optimised for the cache memory on the processors of the iPSC/860. In an optimised code, the double flux calculation overhead would be smaller.

This experiment shows that it does not always make sense to try to minimise the communication time. In some cases, it would be better to eliminate the double flux computations via additional communication of the fluxes.

4.1.2 Parallel performance measurements

We have measured the parallel performance on the Intel iPSC/2 of the explicit Euler solver using scheme 1 (first order, forward Euler), because this scheme has a low calculation to communication ratio as compared to other methods, so the parallel performance of this scheme reflects a worst-case situation.

The parallel efficiency $E(N,P)$ and the speedup $S(N,P)$ compare the execution times on one and on P processors. Another measure for the parallel performance of an algorithm can be defined as follows.

The *effectivity* α of a parallel algorithm is defined as the amount of time spent in the actual calculation relative to the total execution time; for a multi-block code this can be computed as

$$\alpha(N,P) = \frac{\sum_i t_i^{calc}}{P \cdot T(N,P)} \tag{8}$$

where where t_i^{calc} denotes the calculation time for block i and where $T(N,P)$ denotes the execution time of a parallel iteration step for an N-block grid on P processors (incl. communication). The effectivity takes three factors into account : the load imbalance, the communication overhead and the scheduling overhead. For compute-intensive problems, one can expect that $\alpha(N,P)$ is approximately equal to the load balance factor $\lambda(N,P)$. Note that one must not be able to run the program on a single processor to determine α.

In Table 3 we present the parallel efficiency, speedup and effectivity obtained for the two sets of tests men-

tioned above : *a)* $N = P$ blocks; *b)* $N = 16$ blocks regardless of the number of processors. Only in case *a)* the extra flux calculations are considered to be a loss. Therefore, the parallel efficiency and speedup is higher in case *b)*. However, lower effectivities are reported as more blocks have to be managed and more interblock communication occurs.

This comparison stresses the importance of clearly stating how efficiencies and speedups are measured. It also demonstrates that for this type of applications, a high parallel efficiency and speedup can be obtained when load imbalance is insignificant.

4.2 Parallel implicit Euler solvers

4.2.1 Influence of the partitioning strategy

In this section we report on some experiments with parallel implicit Euler solvers. A first series of tests has been done with a solver, based on a first order discretisation with Van Leer flux vector splitting and backward Euler time integration, see e.g. [39]. As test case, we have used the GAMM channel with circular bump and inlet Mach number $M = 0.85$, discretised on a structured grid with 96×32 interior finite volume cells. Treatment of the boundary conditions leads, for each cell on the boundary, to a vector of 'boundary unknowns', which can be associated with a grid point on the boundary. Thus the computational domain consists of a grid of $m_x \times m_y = 98 \times 34$ 'grid points', to be distributed among up to 16 processors.

We have considered several partitionings of the computational domain, leading to different subdomain configurations $N_x \times N_y$, where N_x and N_y denote the number of subdomains in respectively $x-$ and $y-$direction. In all cases, the interior 'grid points' are equally distributed among the processors, but some load imbalance occurs, due to the unequal distribution of the boundary points.

In the previous section we have indicated that the load balance factor (7) gives a good prediction of the parallel efficiency and the parallel speedup, when the communication overhead is small and when all grid points require approximately the same amount of work. Table 4 presents these predicted efficiencies and speedups, and also shows the parallel efficiencies and speedups that are obtained when the linear systems are solved with a Red-Black Gauss-Seidel relaxation scheme on an Intel iPSC/2 parallel computer. (Similar performances will be obtained on other parallel computers with a similar machine characteristic t_{comm}/t_{calc}.)

The results differ from the predicted values for two reasons : (1) the actual load imbalance is smaller than predicted because the boundary points (causing load imbalance) require less operations than the other points; (2) the parallel overhead is higher due to the communication overhead. Note that (1) and (2) have opposite effects on the parallel efficiency and speedup. Since the ratio $m_x/m_y = 98/34$ is approximately equal to 3, subdomain configurations with $N_x/N_y \simeq 3$ yield nearly square subdomains. The results in Table 4 clearly show that, for a fixed number of subdomains, the load balance factor and the achieved parallel efficiency is maximal for nearly square subdomains.

4.2.2 Total efficiency and speedup

However, to measure the actual performance of a parallel solver, one should rather consider the *total speedup* — see Eq. 2 — instead of the parallel speedup, by taking into account the numerical quality of the parallel algorithms. We have therefore compared the convergence properties of Red-Black and lexicographic Gauss-Seidel relaxation schemes. For the lexicographic Gauss-Seidel scheme, two sweep directions were used alternatingly. For this test problem, the number of relaxation steps required to achieve convergence for lexicographic and Red-Black Gauss-Seidel were 492 and 1090 respectively. Thus the (sequential) execution time with the Red-Black Gauss-Seidel scheme is more than 2 times higher than with lexicographic Gauss-Seidel. As a result, the total efficiency (taking into account the numerical efficiency) of the parallel Red-Black relaxation scheme is less than 50 %, even when the parallel efficiency is nearly 100 % !

An alternative is to use a *multi-block approach*: each subdomain is treated independently (i.e. in parallel) and in each subdomain a lexicographic Gauss-Seidel relaxation is performed. Information on subdomain boundaries is exchanged after each complete relaxation step (i.e. after an upward and a downward sweep through the cells).

Because the blocks (subdomains) themselves are treated in a Jacobi fashion, we expect convergence degradation when the number of blocks grows. This is indeed the case, as reported in Table 5. The required number of relaxation steps depends on (a) the number of subdomains, i.e. the number of processors and (b) the *aspect ratio* of the subdomains. Often, the configuration with nearly square subdomains yields the fastest convergence.

The *total speedup* of this multi-block solver can now be defined as the ratio of the execution times on P processors and on one processor to reach the prescribed convergence criterion. The achieved total speedup and total efficiency for some subdomain configurations are given in Table 6. Clearly, for this test problem and when the number of subdomains is not too high, the 'Block Jacobi, lexicographic Gauss-Seidel' scheme is to be preferred above the (single block) Red-Black Gauss-Seidel scheme, because the total efficiency of the latter scheme is less that 50 %.

The convergence degradation of multi-block implicit methods due to the increase of the number of blocks is problem dependent, but in most cases the number of iterations increases only slightly. In [40] a study of the performance degradation for several implicit schemes is presented. The transonic flow computation over the NACA0012 airfoil (see section 4.1, but with 0° angle of attack) has been used as a testcase. In all implicit solvers considered in this study, the Ja-

subdomain configuration $N_x \times N_y$	grid points		predicted parallel efficiency (%)	achieved parallel efficiency (%)	predicted parallel speedup	achieved parallel speedup
	$N_{average}$	N_{max}				
16×1	208.25	238	87.5	90.5	14.0	14.5
8×2	208.25	221	94.2	92.0	15.1	14.7
4×4	208.25	225	92.6	89.1	14.8	14.3
2×8	208.25	245	85.0	83.2	13.6	13.3
1×16	208.25	294	70.8	73.4	11.3	11.7
8×1	416.5	442	94.2	95.7	7.54	7.66
4×2	416.5	425	98.0	97.4	7.84	7.79
2×4	416.5	441	94.4	93.4	7.55	7.47
1×8	416.5	490	85.0	86.4	6.80	6.92
4×1	833	850	98.0	98.9	3.92	3.96
2×2	833	833	100	99.3	4.00	3.97
1×4	833	882	94.4	94.7	3.78	3.79
2×1	1666	1666	100	99.6	2.00	1.99
1×2	1666	1666	100	98.9	2.00	1.98
1×1	3332	3332	100	100	1.00	1.00

Table 4: *Effect of the partitioning strategy on the load balance (= predicted parallel efficiency) and the achieved parallel efficiency of an implicit Euler solver.*

subdomain configuration $N_x \times N_y$	Block Jacobi lexicographic Gauss-Seidel	Block Jacobi line Gauss-Seidel
16×1	623	349
8×2	570	302
4×4	574	326
2×8	627	394
1×16	921	620
8×1	527	265
4×2	503	278
2×4	546	312
1×8	646	386
4×1	489	234
2×2	467	262
1×4	540	304
2×1	438	234
1×2	457	250
1×1	430	230

Table 5: *Implicit Euler solver based on a multi-block approach: number of iterations as function of the subdomain configuration.*

subdomain configuration	number of steps	total speedup	total efficiency (%)
8×2	570	11.3	70.4
4×4	574	10.9	67.9
4×2	503	6.66	83.2
4×1	489	3.48	86.9
2×2	467	3.65	91.2
2×1	438	1.96	98.0
1×1	430	1.00	100

Table 6: *Total speedup and efficiency of the multi-block implicit Euler Solver (Block Jacobi, lexicographic Gauss-Seidel).*

cobian matrices of the residual are evaluated with a first order upwind discretisation (Van Leer, Approximate Steger-Warming or Yoon-Jameson), while the residual driven to zero is either first order or second order. Both line Gauss-Seidel and ADI methods are used to solve the linearised system in each time step. The second order residuals are based on MUSCL extrapolation (third order upwind biased) and a generalised minmod limiter with a compression factor of 2.

The following implicit methods have been investigated, where the first item refers to the implicit solver and the second item to the residual driven to zero:

- Van Leer/Van Leer - Line Gauss-Seidel (VL/VL-LGS)
- Van Leer/Van Leer - ADI (VL/VL-ADI)
- Approximate Steger-Warming/Roe - Line Gauss-Seidel (ASW/R-LGS)
- Approximate Steger-Warming/Roe - ADI (ASW/R-ADI)
- Yoon-Jameson/Roe - LU-SSOR (YJ/R-LU-SSOR)

For the line Gauss-Seidel scheme, four different sweep directions are possible, namely in the positive and negative i- and j-directions. These sweep patterns are indicated in Figure 10.

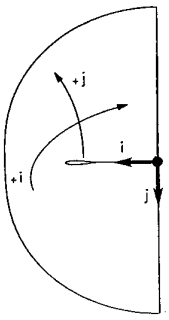


Fig. 10: *Sweep patterns for LGS*

Experiments indicate that, for the single block case, fewer iterations are needed when sweeping in the j-direction ('j-sweeps') than when sweeping in the i-direction ('i-sweeps'). This is to be expected, since within a j-sweep, 240 cells along the i-direction are taken implicitly, while within an i-sweep only 19 cells along the j-direction are taken implicitly. The higher implicitness of the solver for j-sweeps leads to faster convergence. Sweeping in the positive or negative direction has only a slight influence on the number of iterations.

Assume now that a multi-block approach is used, with up to 16 blocks obtained by a 1D partitioning of the grid, orthogonal to the airfoil (as in Fig. 11a).

The performance degradation of the multi-block implementation of the schemes presented above is shown in Table 7. As an initial guess, a first order solution computed with the same explicit operator as the one used in the second order computation, was employed. The convergence criterion was a reduction of the residual by a factor of 10^4. The first order and the second order calculations have been done with respectively CFL = 30 and CFL = 4. For the Line Gauss-Seidel schemes, j-sweeps were used. The combination ASW/R-LGS did not converge for CFL = 4 in the single block case (a decrease of the CFL-number was necessary for convergence). The LU-SSOR scheme needs more iterations than the other schemes; but one LU-SSOR iteration is considerably cheaper than an LGS or ADI iteration.

The results in Table 7 show that no severe degradation in performance occurs for any of the schemes tested, with up to 16 blocks. Note that for the Line Gauss-Seidel schemes, a stronger degradation is to be expected when j-sweeps are used — as in the tests reported here — than when i-sweeps are used. Indeed, because of the 1D partitioning orthogonal to the airfoil, the block boundaries are along the j-direction. When j-sweeps are used, the 'implicitness' in the i-direction is cut by the block boundaries. If i-sweeps would have been used, the implicitness (in the j-direction) would not have been affected by the partitioning. Thus for a very large number of blocks, i-sweeps will be more efficient, since line Gauss-Seidel with j-sweeps degenerates to a point Jacobi scheme, while with i-sweeps the scheme degenerates to a line Jacobi scheme, which is still a powerful scheme. However for a moderate number of blocks, j-sweeps are more efficient, since only 384 iterations are needed when j-sweeps (in the positive j-direction) are used, compared to 561 iterations when +i-sweeps are used.

Convergence degradation can also be observed when a preconditioned Krylov subspace iteration is used as linear system solver : often an efficient preconditioner (e.g. ILU) is replaced in the parallel code by a less effective preconditioner (e.g. diagonal preconditioner), that can be parallelised more easily. Also in this case, the pure parallel efficiency and speedup are not the appropriate measures for the performance, and the different convergence properties of the sequential and parallel solver must be taken into account. Also in this case, a multi-block approach can be useful [41].

4.3 Load balancing of block structured CFD codes

We now describe some results on the 'mapping' of block structured grids on distributed memory systems and the obtained parallel performance. We refer to [37] for more information.

Starting from the grid for the NACA0012 testcase with 16 blocks, each having the same number of cells, we have created block structured grids with up to 103 blocks via grid refinement. Blocks have been refined by doubling the number of grid lines in both directions, using refinement criteria based on the stream-

scheme	1 block	2 blocks	4 blocks	8 blocks	16 blocks
first order					
VL/VL-LGS	384	384	399	411	433
ASW/R-LGS	382	383	442	456	481
second order					
VL/VL-LGS	1561	1580	1567	1577	1588
VL/VL-ADI	1332	1340	1300	1299	1322
ASW/R-LGS	$\star$	1885	1897	1892	1919
ASW/R-ADI	1786	1782	1788	1788	1797
YJ/R-LU-SSOR	3338	3332	3533	3590	3670

Table 7: *Number of iterations required to achieve convergence: influence of the number of blocks (LGS-schemes: sweeps in the j-direction).*

wise entropy gradient. Refined blocks are split into four blocks. Thus all blocks contain approximately the same number of cells. The resulting block structures are shown in figure 11. The first grid counts 16 blocks, the second one 52 blocks and the third one 103 blocks.

A similar procedure has been used for a second testcase : the computation of the supersonic flow in a scramjet geometry. The inlet conditions are $M = 3.6$, angle of attack of $0°$, $T_0 = 300K$ and $P_0 = 100000Pa$. The first grid contains 8160 cells, partitioned into 24 blocks, with sizes varying from 10×15 to 34×15. The block structure of the refined grids with 24, 66, 132 and 161 blocks (corresponding to 82152 cells) is shown in figure 12.

Since the grid is already partitioned into blocks, load balance and communication minimisation must be achieved by an appropriate *mapping* of the blocks onto the processors. Various mapping strategies are incorporated into a software library, that we have developed to hide most of the parallel implementation details from the application programmer [42]. The software library is especially designed to support run-time load balancing for applications that use adaptively refined grids, see [37][43][38]. This software library has been used for the parallelisation of the multiblock code used for the tests described in this section and in section 3.1. The mapping strategy used for the test described here was based on a recursive bisection technique using a costfunction, that takes into account the calculation cost for each block, the communication between blocks mapped onto different processors, and the machine architecture (network topology).

Table 8 shows the *effectivity* for a parallel forward Euler timestep of the multi-block code on an Intel iPSC/860. The loss of effectivity is due to load imbalance and communication overhead. The achieved *load balance* is reported in table 9. The correlation with the effectivity in table 8 reveals that load imbalance is the dominant loss factor.

For the NACA0012 testcase, all 16 initial blocks have the same size, which allows a perfect load balance on up to 16 processors. Refinement of this grid leads to 52 and 103 blocks of almost equal size. As they cannot be equally distributed among the processors, some imbalance remains. For the scramjet testcase, we start with 24 blocks of varying size. Table 9 shows that load balancing works very well if the number of blocks is much larger than the number of processors (or if the block sizes are well-chosen). A certain variation in block sizes is beneficial for load balancing. It is easily verified that the best possible load balance that can be obtained with blocks of equal size is worse than the values reported in Table 9.

Table 10 shows the *communication cost*, including the overhead of the message preparation ('packing' and 'unpacking' of the information in buffers). The communication cost does not grow fast with the number of processors.

Table 11 lists the *estimated parallel efficiency*. It was impossible to determine the true parallel efficiency and speedup, as the refined grid did not fit in a single node's memory. Therefore, the single processor time was estimated as the total calculation time plus the time for copying the data to or from the communication buffer, using the same block structure. The *estimated speedup* is reported in table 12. The speedups obtained are high, due to the good load balance and the fact that the flow solver is so compute-intensive.

These results indicate that the mapping strategy computes a good mapping of blocks onto processors. Note that even for block structured grids of moderate complexity, it is very difficult or even impossible to find a good block distribution by hand and an automatic procedure is needed. Mapping techniques are closely related to *grid partitioning techniques* used to partition unstructured grids for parallel processing. A tutorial on grid partitioning techniques is given in [44].

ACKNOWLEDGMENT

This text presents research results of the Belgian Incentive Programme 'Information Technology'— Computer Science of the Future (IT/IF/5), and the Belgian programme on Interuniversity Poles of Attraction (IUAP 17), initiated by the Belgian State— Prime Minister's Service— Federal Office for Scientific, Technical and Cultural Affairs. The scientific responsibility is assumed by its authors.

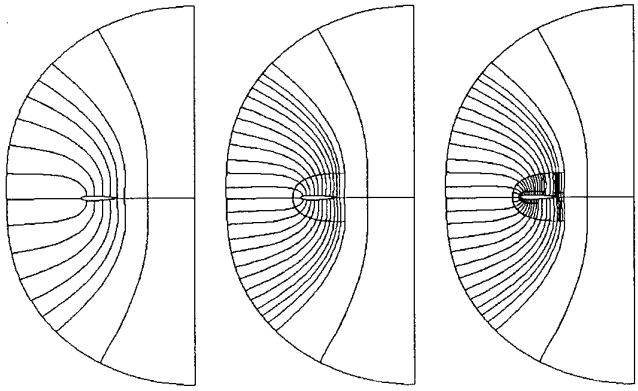


Fig. 11: *Block structure of the NACA0012 grids (16, 52 and 103 blocks)*

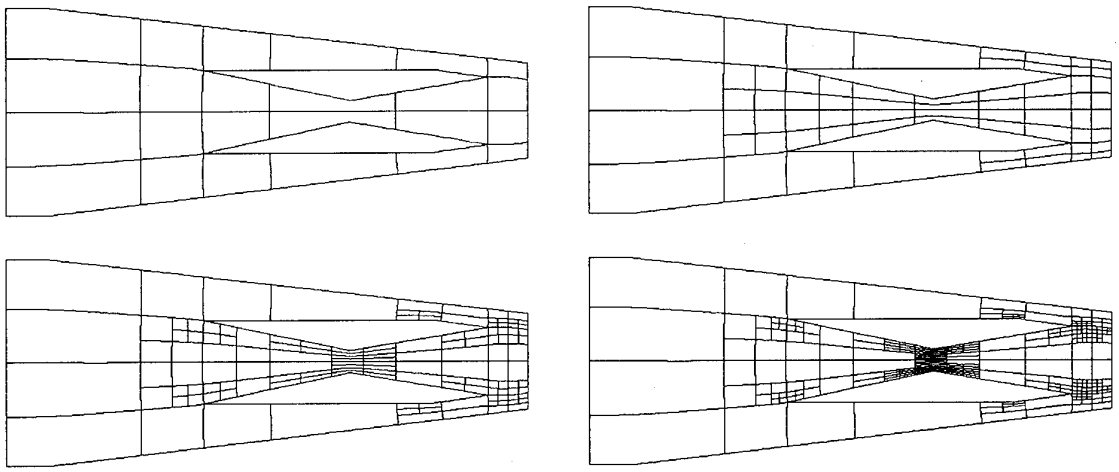


Fig. 12: *Block structure of the scramjet grids (24, 66, 132 and 261 blocks)*

number of processors	1	2	4	8	16	32	64
NACA 16 blocks	99.5	98.8	97.8	96.0	94.4	47.1	
NACA 52 blocks	/	98.5	97.0	90.7	81.1	78.2	
NACA 103 blocks	/	/	95.9	94.6	86.9	78.9	
Scramjet 24 blocks	/	98.7	97.6	97.3	94.3	50.1	24.8
Scramjet 66 blocks	/	/	95.9	92.9	87.5	83.4	58.1
Scramjet 132 blocks	/	/	/	94.5	93.2	88.3	82.0
Scramjet 261 blocks	/	/	/	/	91.2	90.2	80.6

Table 8: *Effectivity α (%) on the iPSC/860*

number of processors	1	2	4	8	16	32	64
NACA 16 blocks	100.0	100.0	98.8	97.7	95.9	48.0	
NACA 52 blocks	/	99.7	98.4	92.7	82.8	80.4	
NACA 103 blocks	/	/	98.3	97.7	89.6	81.4	
Scramjet 24 blocks	/	100.0	99.8	99.7	94.3	51.0	25.5
Scramjet 66 blocks	/	/	98.6	96.2	90.8	87.0	60.6
Scramjet 132 blocks	/	/	/	98.3	97.2	92.5	87.8
Scramjet 261 blocks	/	/	/	/	95.9	94.8	87.8

Table 9: *Calculation load balance λ (%) on the iPSC/860*

number of processors	1	2	4	8	16	32	64
NACA 16 blocks	0.5	1.1	0.8	1.7	1.3	0.6	
NACA 52 blocks	/	1.2	1.2	2.0	1.9	2.0	
NACA 103 blocks	/	/	2.4	3.0	2.9	2.6	
Scramjet 24 blocks	/	1.2	1.8	2.1	1.7	1.0	0.7
Scramjet 66 blocks	/	/	2.5	3.1	3.2	3.0	2.6
Scramjet 132 blocks	/	/	/	3.6	4.1	4.1	5.2
Scramjet 261 blocks	/	/	/	/	4.2	4.3	5.9

Table 10: *Communication cost (%) on the iPSC/860*

number of processors	1	2	4	8	16	32	64
NACA 16 blocks	100.0	99.5	98.4	96.4	95.4	47.4	
NACA 52 blocks	/	99.3	97.8	91.4	81.7	79.0	
NACA 103 blocks	/	/	97.0	95.5	87.8	79.6	
Scramjet 24 blocks	/	99.5	98.8	98.1	95.3	50.3	25.1
Scramjet 66 blocks	/	/	97.0	94.0	88.4	84.4	59.0
Scramjet 132 blocks	/	/	/	95.7	94.4	89.3	83.8
Scramjet 261 blocks	/	/	/	/	92.4	91.3	82.2

Table 11: *Estimated efficiency E (%) on the iPSC/860*

number of processors	1	2	4	8	16	32	64
NACA 16 blocks	1	1.99	3.94	7.71	15.26	15.17	
NACA 52 blocks	/	1.99	3.91	7.31	13.07	25.28	
NACA 103 blocks	/	/	3.88	7.64	14.05	25.47	
Scramjet 24 blocks	/	1.99	3.95	7.85	15.25	16.10	16.06
Scramjet 66 blocks	/	/	3.88	7.52	14.14	27.01	37.76
Scramjet 132 blocks	/	/	/	7.66	15.10	28.58	53.63
Scramjet 261 blocks	/	/	/	/	14.78	29.22	52.61

Table 12: *Estimated speedup S on the iPSC/860*

REFERENCES

[1] A.J. van der Steen. An overview of (almost) available parallel systems. Ncf report, Stichting Nationale Computerfaciliteiten, 's Gravenhage, the Netherlands, December 1994. (Available via ftp from ftp.cc.ruu.nl, dir. /pub/BENCHMARKS/reports).

[2] V.S. Sunderam, G.A. Geist, J. Dongarra, and R. Manchek. The PVM concurrent computing system: Evolution, experiences and trends. *Parallel Computing*, 20(4):531–746, 1994. (PVM is available via ftp from netlib2.cs.utk.edu).

[3] D. Walker. The design of a standard message passing interface for distributed memory concurrent computers. *Parallel Computing*, 20(4):657–673, 1994.

[4] R. Calkin, R. Hempel, H.-C. Hoppe, and P. Wypior. Portable programming with the PARMACS message-passing library. *Parallel Computing*, 20(4):615–632, 1994.

[5] Ch. H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L. Steel, and M.E. Zosel. *The High Performance Fortran Handbook.* MIT Press, 1994.

[6] B. Chapman, H. Zima, and P. Mehrotra. Extending HPF for advanced data-parallel applications. *IEEE Parallel and Distributed Technology*, 2(3):59–70, 1994.

[7] Applied Parallel Research. *FORGE 90 Distributed Memory Parallelizer User's Guide.* APR, Placerville, CA, 1993. (Information on FORGE available by email request from forge@netcom.com).

[8] R. Berrendorf et.al. Intel Paragon XP/S – Architecture, software environment, and performance. Technical Report KFA ZAM IB 9409, Forchungszentrum Julich GmbH, Germany, 1994.

[9] C.B. Stunkel et. al. The SP2 Communication Subsystem. Technical report, IBM Thomas Watson Research Center, Yorktown Heights, NY, 1994.

[10] Silicon Graphics. The advent of PowerComputing. Technical Report Power Challenge, Silicon Graphics, CA, 1994.

[11] J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM, 1991.

[12] J. Gustafson, G. Montry, and R. Benner. Development of parallel methods for a 1024-processor hypercube. *SIAM J. Sci. Statist. Comput.*, 9(4):609–638, July 1988.

[13] J.L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[14] G. Fox, M. Johnson, G. Lyzenga, S. Otto, and J. Salmon. *Solving Problems on Concurrent Processors, Volume I.* Prentice-Hall, Englewood Cliffs, 1988.

[15] E. Van de Velde. *Concurrent Scientific Computing.* Springer Verlag, 1994.

[16] C.A. de Moura. *Parallel algorithms for differential equations.* Lectures Notes, LNCC, Rio de Janeiro, Brazilia, 1994. (Available by email request from demoura@zeus.funceme.br).

[17] R.B. Pelz, A. Ecer, and J. Häuser, editors. *Parallel Computational Fluid Dynamics '92.* North-Holland, Elsevier Science, 1993.

[18] A. Ecer, J. Häuser, P. Leca, and J. Periaux, editors. *Parallel Computational Fluid Dynamics '93.* North-Holland, Elsevier Science, 1995.

[19] N. Satofuka et. al., editor. *Parallel Computational Fluid Dynamics '94.* North-Holland, Elsevier Science, 1995 (to be published).

[20] R. Voigt and J. Saltz, editors. *Proceedings of the Scalable High Performance Computing Conference '92.* IEEE, 1992.

[21] *Proceedings of the Scalable High Performance Computing Conference '94.* IEEE, 1994.

[22] J. Dongarra et.al., editor. *Parallel Processing for Scientific Computing.* SIAM, 1993.

[23] D.H. Bailey et. al., editor. *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing.* SIAM, 1995.

[24] W. Gentzsch and U. Harms, editors. *High Performance Computing and Networking*, LNCS 796-797. Springer, 1994.

[25] B. Hertzberger and G. Serazzi, editors. *High Performance Computing and Networking*, LNCS 919. Springer, 1995.

[26] A. Krechel, H. Plum, and K. Stüben. Parallel solution of tridiagonal linear systems. In F. André and J. Verjus, editors, *Hypercube and distributed computers*, pages 49–64, Amsterdam, 1989. North-Holland.

[27] H. Wang. A parallel method for tridiagonal equations. *ACM Trans. on Math. Software*, 7:170–183, 1981.

[28] M. Honman. The use of an approximate tridiagonal solver in a parallel ADI code. In K.G. Reinsch et.al., editor, *Parallel Computational Fluid Dynamics '91*, pages 227–242. North-Holland, Elsevier Science, 1992.

[29] E.N. Curchitser and R.B. Pelz. Implementation of the Euler equations on MIMD, Distributed Memory multiprocessor computers using cyclic reduction algorithms. In K.G. Reinsch et.al., editor, *Parallel Computational Fluid Dynamics '91*, pages 97–112. North-Holland, Elsevier Science, 1992.

[30] Z.W. Song, D. Roose, C.S. Yu, and J. Berlamont. A comparison of parallel solvers for the 2D shallow water equations on distributed memory parallel computers. In L. Dekker, W. Smit, and J.C. Zuidervaart, editors, *Massively Parallel Processing Applications and Development*, pages 87–94. Elsevier, 1994.

[31] J. Linden, B. Steckel, and K. Stüben. Parallel multigrid solution of the Navier-Stokes Equations on general 2D-domains. *Parallel Computing*, 7:461–475, 1988.

[32] M.E. Braaten. Development of a parallel CFD algorithm on a hypercube computer. *International Journal for Numerical Methods in Fluids*, 12:947–963, 1991.

[33] L.C. Cowsar, E.J. Dean, R. Glowinski, P. Le Tallec, C.H. Li, J. Periaux, and M.F. Wheeler. Decomposition principles and their applications in Scientific Computing. In J. Dongarra et.al., editor, *Parallel Processing for Scientific Computing*, pages 213–237. SIAM, 1992.

[34] D. Keyes. Domain decomposition: a bridge between nature and parallel computers. Technical Report ICASE Rep. 92-44, NASA Langley Res. Center, 1992.

[35] C. Farhat and F.X. Roux. Implicit parallel processing in structural mechanics. 2:1–124, 1994.

[36] C. Mensink and H. Deconinck. A 2D parallel multiblock Navier-Stokes solver with applications on shared and distributed memory machines. In C. Hirsch et.al., editor, *Proceedings of the First European Computational Fluid Dynamics Conference*. Elsevier, 1992.

[37] J. De Keyser, K. Lust, and D. Roose. Run-time load balancing support for a parallel multiblock Euler/Navier-Stokes code with adaptive refinement on distributed memory computers. *Parallel Computing*, 20:1069–1088, 1994.

[38] K. Lust, J. De Keyser, and D. Roose. A parallel block-structured Euler/Navier-Stokes code with adaptive refinement and run-time load balancing. In A. Ecer, J. Häuser, P. Leca, and J. Periaux, editors, *Parallel Computational Fluid Dynamics '93*, pages 243–350. North-Holland, Elsevier Science, 1995.

[39] L. Beernaert, D. Roose, R. Struys, and H. Deconinck. A multigrid solver for the Euler equations on the iPSC/2 parallel computer. *IMACS Appl. Num. Math.*, 7:379–398, 1991.

[40] L. Beernaert and M. Franke. Parallel implementation of iterative solvers. Study note esa-wp6, Von Karman Institute & K.U.Leuven, 1993.

[41] G. Degrez and E. Issman. Acceleration of compressible flow solvers by krylov subspace methods. Von Karman Institute Lecture Series 1994-05, 1994.

[42] J. De Keyser and D. Roose. Load balancing data-parallel programs on distributed memory computers. *Parallel Computing*, 19(11):1199–1219, 1993.

[43] J. De Keyser and D. Roose. Run-time load balancing techniques for a parallel unstructured multi-grid Euler solver with adaptive grid refinement. *Parallel Computing*, 21:179–198, 1995.

[44] R. Van Driessche and D. Roose. Load balancing computational fluid dynamics calculations on unstructured grids. In H. Deconinck et.al., editor, *Parallel Computing in Computational Fluid Dynamics*. AGARD FDP / Von Karman Institute Lecture Notes (this volume), 1995.

Load Balancing Computational Fluid Dynamics Calculations on Unstructured Grids

Rafael Van Driessche and Dirk Roose

Katholieke Universiteit Leuven
Dept. of Computer Science
Celestijnenlaan 200A
B-3001 Leuven, Belgium

1 SUMMARY

Efficient use of a parallel computer requires the data and the operations that must be performed on them to be distributed over the processors in such a way that the work load is balanced and the communication cost minimised. This distribution problem is called the load balancing problem. For CFD applications, the load balancing problem amounts to finding a partition of the grid and subsequently a mapping of the subgrids to the processors, that balance the work load and minimise the communication costs. This tutorial contains a description of well-established methods for partitioning and mapping unstructured grids. They range from simple heuristics, over global optimisation methods to very powerful and cost-effective algorithms that combine the strengths of simpler heuristics. Most of the methods that will be discussed, have been implemented in some well-documented and -supported partitioning tools. The tutorial discusses two of the most important ones: Chaco and TOP/DOMDEC

2 INTRODUCTION

Today's parallel computers potentially allow very high performances. Obtaining these in reality however, requires a careful analysis of the problem and of the solution methods, and often requires that the characteristics of the parallel computer are taken into account during the development of the parallel code.
More specifically, to obtain high performance on a parallel computer, it is of paramount importance to distribute the data and the operations that have to be performed on them in such a way that the work load is balanced over the processors in the parallel computer, while at the same time the communication cost is kept as small as possible. We call this distribution problem the *load balancing problem*.

In this tutorial, we will discuss the load balancing problem for Computational Fluid Dynamics (CFD) applications. Most CFD calculations are *grid-oriented*, i.e. the data are defined on a discrete grid of points, finite volumes or finite elements, and the calculations consist of applying certain operations on (the data associated with) all the points, volumes or elements of the grid.

Grid-oriented applications are usually parallelised by partitioning the grid and by distributing the subgrids among the processors of the parallel computer. Each processor then performs the calculations on its own grid points, volumes or elements. For grid-oriented problems, the load balancing problem amounts to finding a partition of the grid and subsequently a mapping of the subgrids to the processors, that balance the work load and minimise the communication costs.

2.1 Static and dynamic load balancing

The grids that are used in Computational Fluid Dynamics can either be structured or unstructured, static or adaptive and single level or multilevel (the latter in case multigrid is used). If both the grid and the amount of work that is involved with each grid point do not change during the calculations, the distribution of a grid-oriented application can be done statically, usually as a pre-processing step on a sequential computer. If the grid or the calculations do change however, the grid points must be redistributed dynamically over the processors of the parallel machine to maintain load balance. This problem is much more difficult than the static one for several reasons:

Paper presented in an AGARD-FDP-VKI Special Course on "Parallel Computing in CFD", held at the VKI, Rhode-Saint-Genèse, Belgium, from 15-19 May 1995 and 16-20 October 1995 at NASA Ames, United States and published in R-807.

1. the performance of the parallel computer must be monitored to detect the load imbalance,

2. a decision must be made as to whether the gain of the redistribution will outweigh the cost of calculating the new distribution and transferring the grid points; if this cost is very high, it can in fact be advantageous to proceed with an unbalanced distribution,

3. the new distribution must be calculated on the parallel computer, which requires that the distribution algorithm is parallelised,

4. the execution time of the balancer is much more critical than in the static case, because the new distribution is used for a shorter time period,

5. the rebalancing algorithm must preferably find a distribution that is similar to the current distribution, so that only a mimimal number of grid points must be transferred.

When adaptive refinement is used in a CFD-code, the grid remains fixed during rather long periods. In this case one can invoke a load balancer after each grid refinement. This type of load balancing is called *iterative static* load balancing [1, 2] or *quasi-dynamic* load balancing [3]. The techniques that will be discussed in this tutorial are meant to be used for static load balancing. Nevertheless, many of them are also useful for quasi-dynamic load balancing. More specific algorithms, that explicitly try to take the cost of transferring grid points into account, can be found in [4, 5, 6].

2.2 Partitioning and mapping

While distributing the grid points of a structured grid among the processors of a parallel computer is a straightforward task, doing the same for an unstructured grid is very complex. The problem can be alleviated by performing the distribution of the grid points among the processors in two steps. First, the *grid* is *partitioned* in a number of subgrids and subsequently these *subgrids* are *mapped* onto the processors. Typically, the number of subgrids is chosen equal to the number of processors. In principle, the partitioning only depends on the characteristics of the problem while the mapping takes the characteristics of the machine into account. Therefore, these two separate problems are easier to solve than the original distribution problem. On the other hand, solving the partitioning problem separately from the mapping problem usually restricts the quality of the distribution that can be obtained because decisions made during the partitioning step may inhibit finding a good mapping afterwards.

2.3 Requirements for partitioning

In general, an algorithm based on *grid partitioning* or *domain decomposition* involves interface operations and local computations. The *interface* operations consist of communication between subdomains and, in some cases, the solution of a true interface problem (i.e. a Schur-complement operator) or the assembly of subgrid quantities at their common interfaces. The *local* computations correspond either to the solution of a local subproblem or simply to the explicit evaluation of a subgrid quantity.
It is clear that in order to keep the global calculation time as small as possible, the interface operations should take as little time as possible. The local computations should also take as little time as possible and should be balanced evenly among the processors. If this is not the case, the processors will have to wait for the overloaded processor(s) to catch up before they can start with the interface calculations.
From these general requirements, we can deduce the requirements for a mesh partitioner.

1. The time taken by the interface operations is a function of the number of points on the boundary of the subgrid. Very often it is also a function of the number of adjacent subgrids. Therefore the length of the boundary and the number of adjacent subgrids should be minimised for each subgrid. The latter requirements are very often conflicting, and their relative importance depends on the problem and on the characteristics of the parallel computer (especially the start-up to transfer time ratio).

2. The time for the local calculations is a function of the number of points in the subgrid. If the amount of work is the same for all the grid points, each subgrid should have the same number of points to balance the work load.

If the local and/or the interface calculations are *implicit*, i.e. involve the solution of a system of equations, a number of additional considerations come into play:

3. If the local calculations are implicit, the condition of this problem is (strongly) influenced by the aspect ratio of the subgrid. It can be shown that subgrids with bad aspect ratios (i.e. subgrids that are very elongated) generate local problems that are poorly

conditioned and are difficult to solve iteratively [7, 8]. Moreover, ill-conditioned local problems have a negative impact on the iterative solution of the interface problem [9] as well. Elongated subgrids tend to have long perimeters, therefore trying to obtain interfaces with minimal length will typically yield grids with good aspect ratios.

4. If the local calculations are implicit, and if a direct method is used to solve the local system, the calculation time for each subgrid is influenced by the bandwidth of the local matrix. This bandwidth depends on the shape of the subgrid.

5. If a frontal method is used to solve the linear systems arising from a finite element approach, the frontwidth associated with each subgrid should not be greater than the frontwidth of the global grid. Ideally, the partitioning should generate subgrids in which the number of unknowns at the interface of any subgrid is smaller than the frontwidth associated with the undecomposed grid and the frontwidth of each subgrid is at most comparable to the frontwidth of the global grid [8].

2.4 Requirements for mapping

The mapping algorithm must assign the subgrids to the processors of a parallel machine. Preferably, subgrids that are mutually dependent are mapped onto processors that can communicate rapidly with each other. For a fully connected machine, the mapping task is trivial: any mapping is as good as the other. For the existing machines with a limited communication topology (hypercube, 2D-, or 3D-mesh, ...) this is not the case. Although communication between arbitrary processors can be done efficiently, nearest-neighbour communication is preferable because it decreases the risk for communication link contention.

As mentioned earlier, the mapping task is not completely independent from the partitioning task. The mapping task can be seriously facilitated by already taking the machine topology into account during the partitioning step to ensure that the dependency topology of the subgrids matches the communication topology of the machine.

3 A CLASSIFICATION OF PARTITIONING ALGORITHMS FOR UNSTRUCTURED GRIDS

3.1 General optimisation techniques based on a cost function

The most general approach to finding an optimal distribution of the grid points among the processors of a parallel machine is to model the total calculation time as a function of the mapping. In this way one obtains a function that associates a cost with each feasible distribution. Thus it is possible to solve the partitioning and the mapping problem together. In fact, the cost function can be quite sophisticated, taking into account hardware characteristics and communication topology of the parallel computer, contention of the communication links etc. Normally, the cost function contains a term that takes the communication cost into account and another that is related to the load imbalance. The relative importance of those two terms depends on the characteristics of the problem and of the parallel computer. Indeed it is sometimes beneficial to tolerate a (slight) load imbalance if this decreases the communication.

The cost function can be minimised by a general optimisation technique that is appropriate for global combinatorial optimisation. For a grid with N points that must be mapped onto P processors, the search space has cardinality N^P. Because the search space grows exponentially in the grid size, total enumeration is infeasible for realistic problems, even when one takes advantage of possible symmetry properties or uses branch-and-bound techniques to exclude whole parts of the search space.

However, some techniques that yield good suboptimal solutions for combinatorial optimisation problems do exist. Two of them, viz. simulated annealing and genetic algorithms are frequently used.

Simulated annealing. Simulated annealing [10, 11] is a very general optimisation method which stochastically simulates the slow cooling of a physical system. A parameter T, analogous to the temperature, is slowly lowered in the course of the calculations. For each temperature a number of transitions of the current solution are consecutively proposed and either accepted or rejected according to the Metropolis criterion: If the cost function decreases (cost increase $\Delta C < 0$), the change is accepted unconditionally, otherwise it is accepted with probability $\exp(-\Delta C/T)$. It can be proved that under certain conditions the probability to find the

global optimum tends to 1. In practice, for sufficiently slow cooling rates this method produces good solutions, but then the method is very expensive. Results of simulated annealing for grid partitioning can be found in [12, 3].

Genetic algorithms. Genetic algorithms [13, 14] resemble simulated annealing in that they are also general and robust optimisation methods that simulate an optimisation process found in nature. More specifically, genetic algorithms simulate the processes of reproduction, crossover and selection that make living beings optimally adapted to their environment. Genetic algorithms are potentially able to yield optimal or near-optimal solutions but take a large amount of time. Results of genetic algorithms for partitioning problems are reported in [15, 16, 17].

Modelling the execution time as a function of the distribution of the grid points has the advantage that the partitioning and mapping problem can be solved together, and that sophisticated cost models can be used. However, stochastic optimisation algorithms are extremely slow, can be trapped in local minima, and their behaviour depends on a lot of parameters, that must be carefully tuned to optimise performance.

3.2 Specific grid partitioning heuristics

3.2.1 Introduction

To make the distribution problem more tractable, one normally makes the following simplifications:

1. The partitioning and the mapping problem are handled separately. Subsequently, we will restrict ourselves to the partitioning problem.

2. Rather than trying to minimise both computational workload imbalance and communication simultaneously, only one of both terms is explicitly modelled while the other is used implicitly in guiding the search. In this way the search space can be substantially reduced. Most often, one explicitly tries to minimise communication while the heuristic implicitly provides equally-sized subgrids.

3.2.2 Clustering techniques

Some authors have proposed partitioning and mapping strategies based on clustering techniques. In these approaches clusters of grid points are formed with high intra-cluster dependencies and low inter-cluster communication. The clustering is based on a sorting of the grid points and subsequent partitioning.

Mapping algorithm of Sadayappan. Sadayappan et al. [18, 19] proposed a *nearest-neighbour mapping algorithm*, that proceeds in two steps:

1. An initial mapping is generated by grouping grid points in clusters and assigning clusters to processors so that the nearest-neighbour property is satisfied, i.e. neighbouring points are assigned either to the same processor or to neighbouring processors.

2. The initial mapping is successively modified using a boundary refinement procedure in which points are reassigned among the processors in a manner that improves calculation load balance but always maintains the nearest-neighbour property.

Thus the nearest-neighbour mapping scheme explicitly attempts to minimise calculation load imbalance, while low communication costs are achieved implicitly by the search strategy.

Bandwidth reduction algorithms. Algorithms that *reduce the bandwidth and the profile* of a (sparse) matrix by re-ordering the equations and the unknowns of the linear system can also be used for partitioning meshes [8, 20].
For a given numbering of the n elements of a mesh, we can associate an *adjacency matrix* A, which is a symmetric $n \times n$ matrix with elements a_{ij} that are equal to either 1 or 0 according to whether the elements i and j are or are not adjacent in the mesh. Let m_i $(i = 1, \ldots, n)$ be the smallest number for which $a_{ij} = 0$ if $| i-j | > m_i$. The bandwidth of A is then defined as $\max_i m_i$, and the profile as $\sum_{i=1}^{n} m_i$.
If the elements of the mesh have been numbered in such a way that the adjacency matrix has a small profile and bandwidth, a *lexicographic* partitioning of the mesh will often place adjacent elements in the same subgrid, and each subgrid will only have a limited number of neighbouring subgrids. Figure 1 illustrates this. Notice that two adjacent elements are assigned to different subgrids if the corresponding element in the adjacency matrix is not in one of the blocks on the main block diagonal and that two subgrids are adjacent if the corresponding off-diagonal block is non-zero.
The Reverse Cuthill-McKee (RCM) ordering scheme [21] is one of the most popular techniques for reducing the bandwidth and the profile of

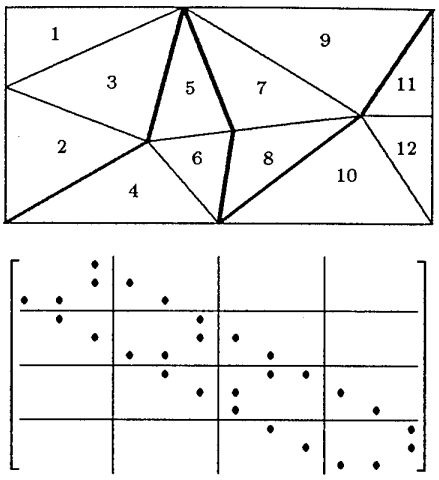


Fig. 1: Mesh partitioned into 4 submeshes and the corresponding adjacency matrix

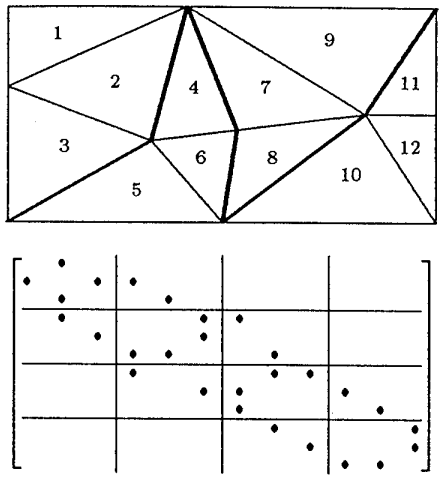


Fig. 2: Mesh in Fig. 1, numbered with the Cuthill-McKee heuristic and the corresponding adjacency matrix

sparse matrices. The Cuthill-McKee scheme, applied to the adjacency matrix of a mesh, essentially clusters the elements in level sets:

choose an initial element ;
$S_1 := \{\text{initial element}\}$;
while not all elements have been added to a level set **do**
  $S_{i+1} := \emptyset$;
  forall elements $e_k \in S_i$ in the order that they have been added to S_i **do**
    add to S_{i+1} the adjacent elements of e_k that have not yet been added to a level set ;
  endfor;
endwhile;

Next, the elements are numbered in the order that they have been added to the level sets.
For the mesh in Fig. 1, the Cuthill-McKee algorithm, initiated with the upper left element creates the level sets $S_1 = \{1\}$, $S_2 = \{3\}$, $S_3 = \{2, 5\}$, $S_4 = \{4, 6, 7\}$, $S_5 = \{8, 9\}$, $S_6 = \{10, 11\}$, and $S_7 = \{12\}$. The resulting numbering, partitioning, and adjacency matrix are shown in Fig. 2.
Usually, the order obtained with the Cuthill-McKee algorithm is reversed. This does not affect the bandwidth of the matrix but it often decreases its profile.
Bandwidth minimiser algorithms have the advantage that for each subdomain, the number of adjacent subdomains is small. Therefore, each processor must only send messages to a small number of neighbours. This can be important if the start-up cost of sending a message is high. However, bandwidth minimiser algorithms have a tendency to generate very elongated subgrids with rather large interface sizes. Usually these subdomains enjoy a very small local bandwidth, but suffer from a very bad aspect ratio. These problems are alleviated if the RCM algorithm is used recursively [8].

Greedy heuristic of Farhat. For the partitioning of finite element meshes, Farhat [22] proposed a *greedy algorithm* that uses only connectivity information. A variation of the algorithm that also uses geometrical information was proposed by Al-Nasra and Nguyen [23]. We will discuss Farhat's heuristic into more detail in Section 4.

3.2.3 Geometry-based techniques

In the *geometry-based techniques* the partitioning is based on geometrical information about the grid points (i.e. their coordinates). This is sensible because, in most problems, interdependent grid points are geometrically adjacent. Geometry-based techniques are typically cheap methods that are nevertheless able to produce acceptable partitionings. They are dealt with in Section 5.

3.2.4 Heuristics for graph partitioning

The *graph partitioning problem* can be formulated as follows. An undirected graph $G = (V, E)$ with vertex set V and edge set E is given. Often, weights $w_e(e_{ij})$ are attributed to the edges $e_{ij} \in E$. Also given are P positive integers $n_1, \ldots, n_P$, satisfying $\sum_{i=1}^{P} n_i = n \equiv |V|$. The problem is then to partition the vertex set V into P disjoint subsets $V_1, \ldots, V_P$ of sizes $n_1, \ldots, n_P$, respectively, in such a way that the *sum of the weights* of edges connecting different subsets is *minimal*. In most cases we require that $n_1 = n_2 = \cdots = n_P$. An edge which connects two distinct subsets is said to be *cut* by the partition. The graph partitioning problem can also be generalised to the case that the vertices v_i too have a weight $w_v(v_i)$. In this case, the weights of the subsets are imposed instead of their sizes.
Now, for grid-oriented problems, a *dependency graph* can be defined as follows:

1. Each grid point has a corresponding vertex in the graph. The weight of the vertex is proportional to the amount of computational work that is involved with the grid point. Very often the weights of all vertices are equal, e.g. when iterative solution schemes are used.

2. For each pair of mutually dependent grid points, the corresponding vertices are connected by an edge in the graph. The weight of the edge is proportional to the strength of the interdependency ('communication volume').

For a finite element mesh, two elements are usually dependent on each other if they share an edge in two dimensions or a face in three dimensions. Therefore, the interdependency graph is simply the dual graph of the mesh. An example is given in Fig. 3.
Obviously, the grid partitioning problem is equivalent to the graph partitioning problem for the dependency graph. The graph partitioning problem is an NP-complete problem but a number of specific heuristics that yield good near-optimal solutions do exist. We will discuss the Kernighan-Lin heuristic, the Recursive Graph Bisection algorithm, and the Recursive Spectral Bisection algorithm.

Kernighan-Lin heuristic. Already in 1970, Kernighan and Lin introduced a heuristic to partition a graph into two or more subgraphs [24]. Their heuristic only partitions graphs without vertex weights, but generalisation to graphs with

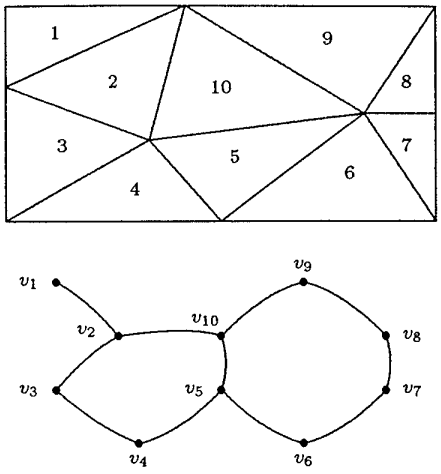


Fig. 3: Finite element mesh and corresponding dual graph

unequal vertex weights is straightforward. Fiduccia and Mattheyses [25] demonstrated that, if the vertex weights are small integer numbers, the algorithm can be organised in such a way that the complexity of the algorithm is only linear in the number of edges.

Recursive Graph Bisection algorithm. The Recursive Graph Bisection algorithm (RGB) recursively determines two vertices of maximal or near maximal distance in the (sub)graph, and subsequently assigns the vertices to one or to the other subset, according to whether they are closer to one or to the other extremal vertex. To determine the distance between two vertices, the *graph distance* is used, i.e. the length of the shortest path between the vertices. A more thorough discussion of this algorithm and a comparison with other partitioning algorithms can be found in [26, 27].

Recursive Spectral Bisection algorithm. The Recursive Spectral Bisection algorithm (RSB) is based upon results from spectral graph theory, in which eigenvectors of a matrix are used to bisect a graph. This algorithm is discussed in detail in Section 6.

4 THE GREEDY HEURISTIC OF FARHAT

4.1 Description

The *greedy algorithm* of Farhat [22] is a heuristic that despite its simplicity often yields subgrids

with short boundaries and good aspect ratios. The algorithm first assigns to each node n_i of the mesh a weight w_i that is equal to the number of elements that are connected to it. Let Ω^s, Γ^s and $\mathcal{C}^s$ respectively denote the body, the interface boundary, and the computational cost of a subdomain s and let $\mathcal{C}$ denote the computational cost of the whole domain. The algorithm consecutively finds the domains $\Omega^1, \ldots, \Omega^P$. Once the first $s-1$ domains $\Omega_1, \ldots, \Omega_{s-1}$ have been found, it constructs the next domain Ω^s as follows:

locate a node $n_i \in \Gamma^{s-1}$ that has a minimal current weight w_i ;
initialise Ω^s with all un-masked elements that are connected to node n_i ;
for each element $e_k \in \Omega^s$ **do recursively**
  mask e_k ;
  for each node n_i attached to e_k **do**
    reduce the weight w_i by one ;
  endfor;
  add to Ω^s all un-masked elements that are adjacent to e_k ;
  update $\mathcal{C}^s$;
  break when $\mathcal{C}^s = \mathcal{C}/P$.
endfor;

Figure 4 illustrates how the algorithm expands a

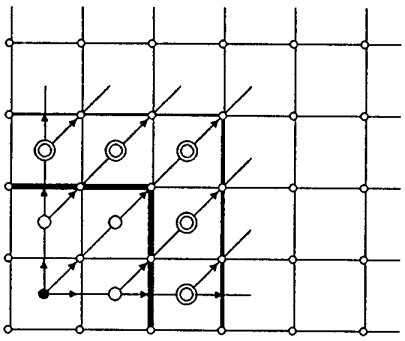


Fig. 4: Expansion of a subdomain using the greedy algorithm (after Farhat [22]).

subdomain starting from the lower left element. The greedy heuristic of Farhat is probably the fastest partitioning algorithm. Since it can immediately partition a grid into the desired number of subgrids, it is not necessary to use it recursively. This has the advantage that the calculation time is essentially independent of the desired number of subgrids. In general this algorithm generates subgrids with good aspect ratios, but it often yields disconnected subgrids.

4.2 Some examples

Figure 5 shows a two-dimensional finite element mesh with 9000 triangular elements and with 13278 internal boundary edges round an airfoil. Only the part of the mesh that lies in the vicinity of the airfoil, and which is strongly refined, is shown. Partitioning this mesh into eight subgrids with the greedy algorithm of Farhat yields the partitioning in Fig. 6. This partition cuts 355 edges. Notice that the subgrids have a good aspect ratio. However, the subgrid that was created last (darkly shaded in Fig. 6) is disconnected into three parts.
Figure 7 shows the partition into eight subgrids of the two-dimensional RYMAMO model [28]. This model is a finite difference grid with 18675 points that covers the mouth of the Rhine and the Meuse and the coastal zone near the harbour of Rotterdam. The greedy algorithm was actually applied on a finite element mesh with quadrilateral elements, so that each element of this mesh corresponds to a point in the finite difference grid and in such a way that the connectivities were preserved. It is very difficult to partition this mesh into connected subgrids and in the partition that one obtains with the greedy algorithm, effectively four subgrids out of eight are disconnected. The partition yields 22 connected parts and cuts 365 edges.

5 GEOMETRY BASED BISECTION ALGORITHMS

5.1 Introduction

In the *geometry-based bisection algorithms*, one tries to exploit the geometric properties of the mesh, since data dependent grid points are geometrically adjacent. Clearly, this limits the applicability of this type of methods to problems where such geometric information is both meaningful and available.
Based on the geometrical information, a scalar quantity σ_i is associated with each grid point. Following Williams [3], we call σ_i a *separator field*. By evaluating the median S of the set $\{\sigma_i\}$, we can bisect the grid, according to whether σ_i is greater or less than S. In this way two subgrids with an equal number of grid points are created. By recursively applying this strategy to the subgrids, the grid can be partitioned into $2^d, d = 1, 2, \ldots$ subgrids.
Notice that, based on the ordering of the separator field, a grid could easily be partitioned into more than two parts at once. However, the aspect ratio of the subgrids is usually better if a grid is only partitioned into *two* parts.

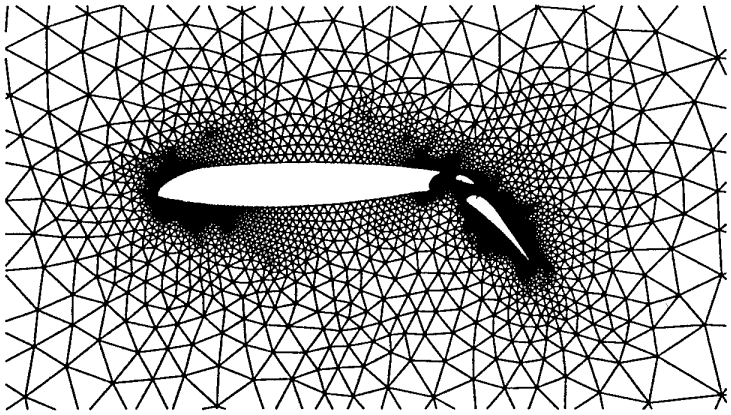


Fig. 5: Part of a finite element mesh with 9000 elements round an airfoil.

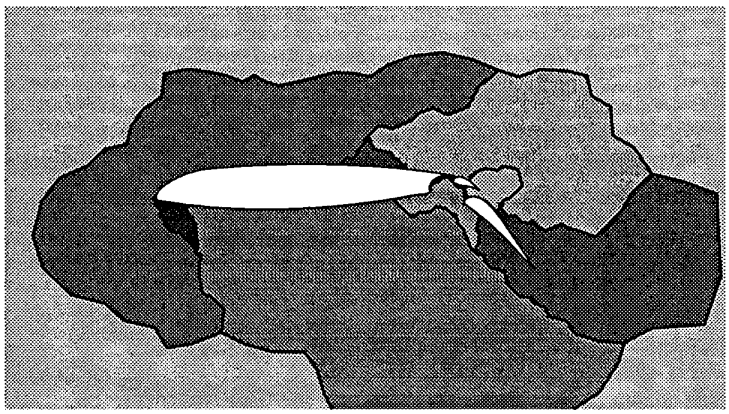


Fig. 6: Partitioning into 8 subgrids of the finite element grid in Fig. 5 with the greedy heuristic of Farhat.

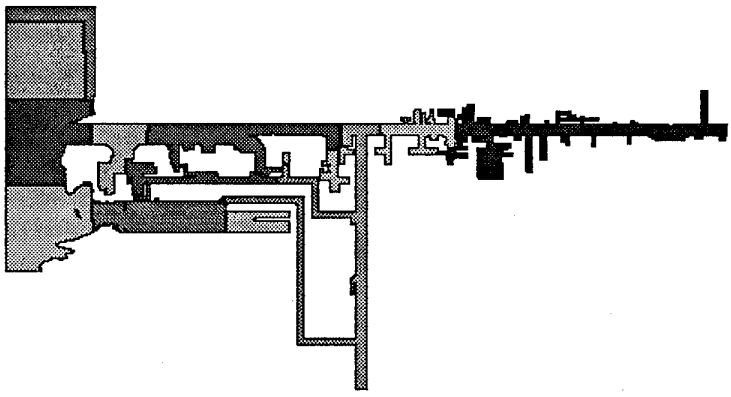


Fig. 7: Partitioning into 8 subgrids of the RYMAMO grid with the greedy heuristic of Farhat.

5.2 Repeated x-bisection

The simplest choice for σ_i is $\sigma_i = x_i$ with x_i the x-coordinate of the grid point. Recursive application of this technique on the subgrids gives rise to a stripwise partitioning, with strips parallel to the y-axis. Such a partitioning causes long inter-subgrid interfaces and thus a large communication volume.

5.3 Recursive Coordinate Bisection

Recursive Coordinate Bisection (RCB) [27], also called *Orthogonal Recursive Bisection* (ORB) [3], consists of alternately bisecting the grid according to the x-, and y-coordinate and, for three-dimensional grids, the z-coordinate. This technique leads to grids with a better aspect ratio than the ones that are obtained with repeated x-bisection. This has a positive effect on the communication volume.

5.4 Recursive Inertial Bisection

Using the x-, y- or z-coordinate of the grid points has the disadvantage that the partitioning depends on the coordinate system used, which is not an intrinsic problem characteristic.

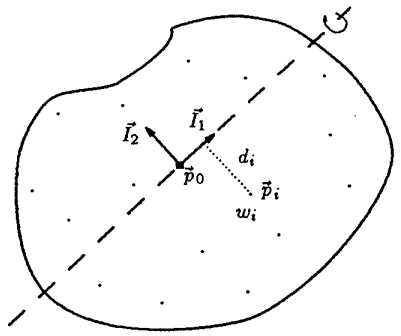


Fig. 8: Angular momentum of a discrete point set.

The basic idea behind the inertial bisection strategy is the following. The principal inertia direction of an object (or a discrete point set) is the direction for which the rotational inertial momentum $I = \sum_i w_i d_i^2$ is minimal when this direction is taken as the rotation axis (see Fig. 8). If the domain is more or less convex-shaped, the minimal momentum axis will be aligned with the overall shape of the grid. We can therefore expect that the grid will have its smallest spatial extent in the direction orthogonal to this axis of rotation. This direction is then heuristically chosen to be the bisection direction.

The inertia directions of the mesh are the eigenvectors I_1, I_2 and I_3 corresponding to the eigenvalues $\lambda_1 \leq \lambda_2 \leq \lambda_3$ of the 3×3 inertia matrix:

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix},$$

with

$$\begin{aligned}
I_{xx} &= \sum_i (y_i - y_c)^2 + (z_i - z_c)^2, \\
I_{yy} &= \sum_i (x_i - x_c)^2 + (z_i - z_c)^2, \\
I_{zz} &= \sum_i (x_i - x_c)^2 + (y_i - y_c)^2, \\
I_{xy} &= I_{yx} = -\sum_i (x_i - x_c)(y_i - y_c), \\
I_{yz} &= I_{zy} = -\sum_i (y_i - y_c)(z_i - z_c), \\
I_{zx} &= I_{xz} = -\sum_i (z_i - z_c)(x_i - x_c),
\end{aligned}$$

where the summations must be taken over all the grid points and where (x_i, y_i, z_i) and (x_c, y_c, z_c) respectively denote the coordinates of the grid points and the coordinates of the center of gravity of the mesh. The eigenvector I_1 which is associated with the smallest eigenvalue corresponds to the axis of minimal angular momentum. Once I_1 is determined, the grid points are projected (orthogonal projection) onto it and this projection is used as the separator field σ_i for the bisection of the grid.

The *Recursive Inertial Bisection* (RIB) algorithm, also called the Inertial Recursive Bisection (IRB) [29] or the Recursive Principal Inertia (RPI) [26] algorithm, is more expensive than repeated x-bisection or Recursive Coordinate Bisection but generally gives much better results. Because the minimal rotational momentum axis is an inherent property of the grid, this partitioning does not depend on the orientation of the coordinate system. It still depends however, on the relative scaling of the x-, y- and z-axes.

Recursive Inertial Bisection is now a widely used partitioning technique [8, 30], especially in combination with the Kernighan-Lin heuristic (see Section 3.2.4).

5.5 Some examples

We will first make a comparison between the results obtained with repeated x-bisection, Recursive Orthogonal Bisection, and Recursive Inertial Bisection. These methods have been thoroughly

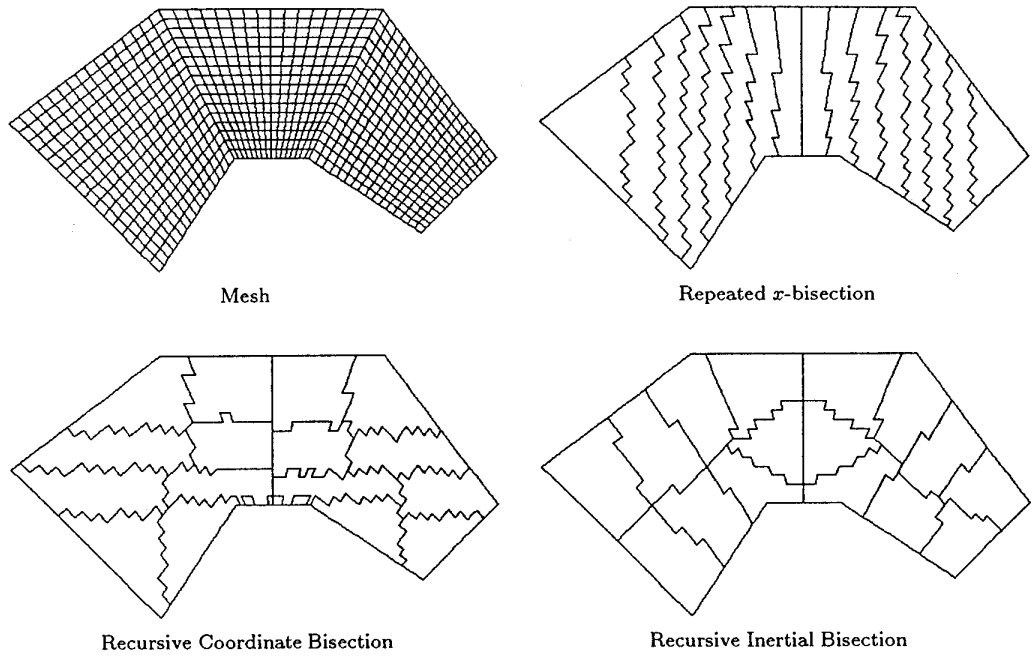


Fig. 9: Narrowing curved channel (structured grid) and partitions with geometry-based methods.

studied and compared with each other in [29]. The following examples have been taken from it. Figure 9 shows a structured finite volume mesh for a narrowing curved channel. It consists of 768 cells and 1472 edges. It is partitioned into 16 parts, using the repeated x-bisection, Recursive Coordinate Bisection and Recursive Inertial Bisection heuristics. The load balance is in all cases (nearly) perfect, but the number of edges cut by the partition interfaces is respectively 324, 236 and 191. Notice that for this structured grid of 48×16 cells, an optimal partitioning can easily be found by splitting it into 8×2 nearly square subgrids of each 6×8 cells. In this case only 160 edges are cut by the partition interfaces.

However for unstructured meshes like the one in Fig. 10, such an optimal partitioning cannot be found so easily. This mesh is used to calculate the supersonic flow through a channel with a forward step. It consists of 1186 cells and 1652 edges. It is again partitioned into 16 parts. The number of edges cut by the partition interfaces for the repeated x-bisection, Recursive Coordinate Bisection, and Recursive Inertial Bisection heuristics is respectively 430, 297 and 281. Examination of the figures reveals how in the Recursive Inertial Bisection method the axis direction adapts itself to the non-convexity of the narrowing curved channel and to the increased mesh density in the channel. The above experiments illustrate that the Recursive Inertial Bisection heuristic should be prefered over the other two geometry-based techniques.

Let us now compare the Recursive Inertial Bisection algorithm with Farhat's greedy heuristic. Figure 11 shows the partitioning into eight subgrids that one obtains with the Recursive Inertial Bisection algorithm of the grid in Fig. 5. This partition cuts 515 edges, which is more than with Farhat's greedy heuristic. Also notice that some subgrids are quite elongated, which might be a problem with some iterative methods [7].

Inertial bisection implicitly assumes that the mesh is convex. For the RYMAMO model, this is obviously not the case. We can therefore expect the Recursive Inertial Bisection algorithm to perform poorly. Figure 12 shows the partitioning into eight subgrids. This partition yields 20 connected parts, less than Farhat's greedy heuristic, but it cuts 485 edges which is more than the 365 edges that are cut by the greedy heuristic.

6 THE RECURSIVE SPECTRAL BISECTION ALGORITHM

6.1 Introduction

The use of spectral methods to bisect graphs was first considered by Donath and Hoffman [31], and

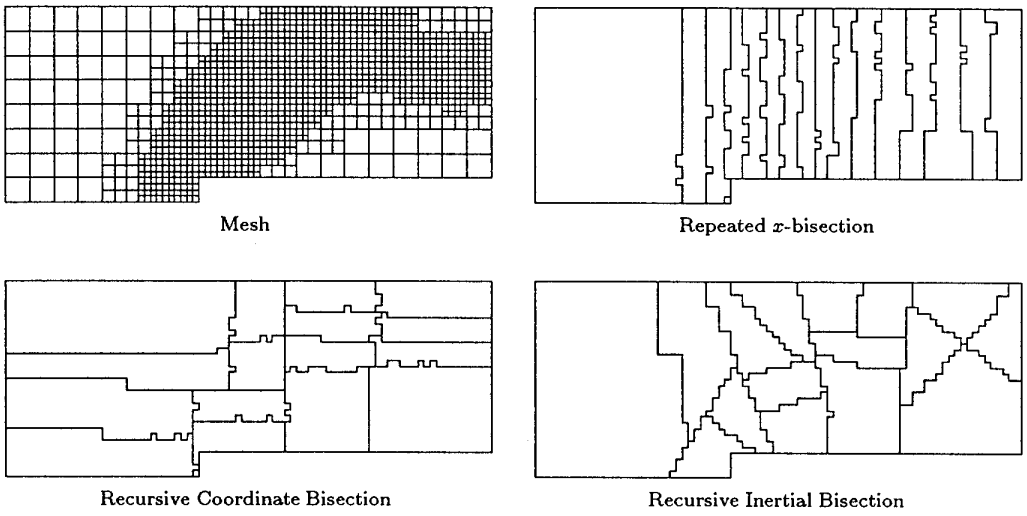


Fig. 10: Channel with forward step (unstructured grid) and partitions with geometry-based methods.

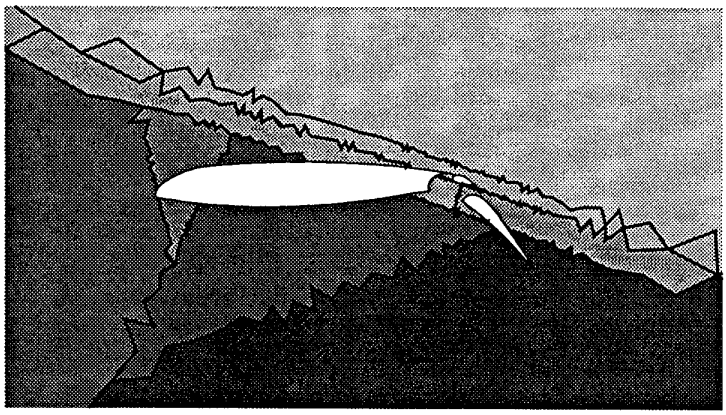


Fig. 11: Partitioning into 8 subgrids of the finite element grid in Fig. 5 with the Recursive Inertial Bisection algorithm.

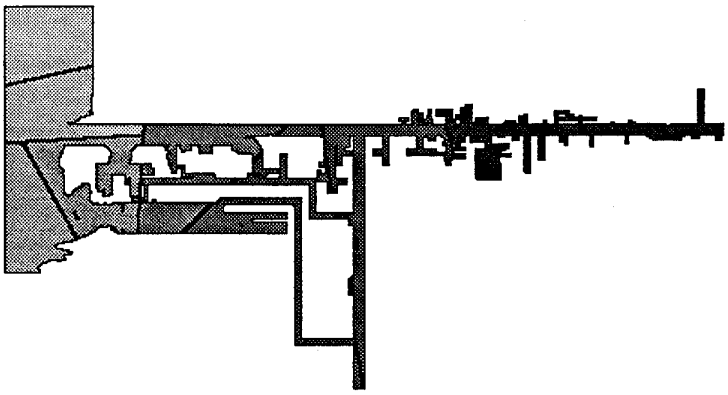


Fig. 12: Partitioning into 8 subgrids of the RYMAMO grid with the Recursive Inertial Bisection algorithm.

since then, spectral methods for computing various graph parameters have been used by several others.
Barnes [32] introduced a bisection technique that uses the eigenvectors corresponding to the largest two eigenvalues of the adjacency matrix of the graph.
The most frequently used spectral bisection technique was introduced by Pothen et al. [33]. In this method, the graph is bisected according to the eigenvector that corresponds to the second smallest eigenvalue of the Laplacian matrix of the graph.
By recursively applying the spectral bisection algorithm to the subgraphs, it is possible to partition a graph into $2, 4, \ldots, 2^d$ subgraphs. This heuristic was first used to partition finite element meshes by Simon [27], who used the name *Recursive Spectral Bisection*, and by Williams [3] who named the method *Eigenvalue Recursive Bisection*.

6.2 The algorithm

Intuitively, it is not immediately obvious that the second eigenvector of the Laplacian matrix of a graph is a good separator for the graph. The following deduction of the algorithm helps to understand why this is nevertheless the case.
We denote the graph by $G = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the vertex set, and E is the edge set. A weight $w_e(e_{ij})$ is associated with each edge $e_{ij} \in E$. This graph is completely determined by its *adjacency matrix* A. This is a symmetric $n \times n$ matrix with elements,

$$\begin{aligned} a_{ii} &= 0, && i = 1, \ldots, n, \\ a_{ij} &= \begin{cases} w_{ij} \text{ if } e_{ij} \in E, \\ 0 \quad \text{otherwise,} \end{cases} && i, j = 1, \ldots, n; i \neq j. \end{aligned}$$

We search a mapping $m : \{1, 2, \ldots, N\} \to \{1, 2\}$ that minimises:

$$\sum_{e_{ij} \in E} w_e(e_{ij}) \ (1 - \delta_{m(i),m(j)}).$$

This mapping partitions the vertex set V into two subsets V_1 and V_2:

$$\begin{aligned} V_1 &= \{v_i \in V \mid m(i) = 1\}, \\ V_2 &= \{v_i \in V \mid m(i) = 2\}. \end{aligned}$$

The vertex sets V_1 and V_2 should have the same number of elements.
Let x be a vector of length n whose components are defined as follows:

$$x_i = \begin{cases} -1 & \text{if } m(i) = 1, \\ 1 & \text{if } m(i) = 2. \end{cases}$$

This is convenient because $1 - \delta_{m(i),m(j)} = \frac{1}{2}(1 - x_i x_j)$. Moreover, the requirement that V_1 and V_2 have the same number of elements is equivalent to $\sum_{i=1}^{n} x_i = 0$.
We must therefore minimise

$$\frac{1}{2} \sum_{e_{ij} \in E} w_e(e_{ij}) \ (1 - x_i x_j),$$

subject to $x_i \in \{-1, 1\}$ and $\sum_{i=1}^{n} x_i = 0$. It will prove advantageous to add to this expression the term

$$\frac{1}{4} \sum_{i=1}^{n} t_i \left(x_i^2 - 1\right).$$

Since $x_i = \pm 1$, this term is zero and does not change the value of the object function, but later on, we will relax the constraint $x_i = \pm 1$ and then this term will become important.

Using the notation $\tau = \sum_{i=1}^{N} t_i$, $W_e = \sum_{e_{ij} \in E} w_e(e_{ij})$, $D = \mathrm{Diag}(t_i)$ and $B = D - A$, we can write our object function as

$$\frac{1}{2}(W_e - \frac{\tau}{2}) + \frac{1}{4} x^T B x.$$

This function must be minimised subject to the constraints $x_i \in \{-1, 1\}$ and $\sum_{i=1}^{n} x_i = 0$.
We choose the diagonal values t_i so that each row sum of B is zero. This choice is convenient for several reasons:

1. Since $t_i = \sum_{e_{ij} \in E} w_e(e_{ij})$ implies that $\tau = 2W_e$, the first term in the object function is identically zero.

2. The matrix B is positive semidefinite. If the graph is connected, then B only has a single null vector consisting entirely of 1's.

3. If the edge weights are all 1, then the diagonal elements b_{ii} of B are equal to the degree of the corresponding vertex v_i, and the off-diagonal elements b_{ij} are equal to -1 if the corresponding vertices v_i and v_j are connected by an edge and are equal to 0 otherwise. This matrix is the so-called Laplacian matrix of the graph, and we can say that in general the matrix B is a weighted Laplacian. This is advantageous because a number of interesting properties about the Laplacian matrix that can give us some guarantees about the quality of the solution, are already known [34, 35]. We will say more about the Laplacian matrix later.

Using the notation $e = [1\,1 \ldots 1]^T$, our discrete minimisation problem becomes:

> Minimise $\frac{1}{4} x^T B x$, subject to
> $x_i \in \{-1, 1\}$, and $e^T x = 0$.

This minimisation problem is still a discrete NP-complete problem. We now relax the constraint that each of the components of the vector x must be ± 1. Instead, we impose the norm constraint $x^T x = n$. In this way we replace our discrete problem by the following continuous one:

> Minimise $\frac{1}{4} x^T B x$, subject to $x^T x = n$,
> $e^T x = 0$, and $x_i \in \mathbb{R}$ $(i = 1, 2, \ldots, n)$.

It must be noticed that all the feasible solutions of the discrete problem are also feasible solutions of the continuous problem. Therefore, the solution of the continuous problem provides a lower bound for the solution of the discrete one. Contrary to the discrete problem however, the continuous optimisation problem can be solved easily thanks to the special properties of the matrix B:

1. B is symmetric positive semidefinite ;

2. The eigenvectors of B can always be chosen to be pairwise orthogonal ;

3. The vector e is an eigenvector of B with eigenvalue zero ;

4. If the graph is connected, e is the only eigenvector of B with eigenvalue zero.

Let $0 = \lambda_1 < \lambda_2 \leq \lambda_3 \leq \cdots \leq \lambda_n$ be the eigenvalues of B with corresponding orthonormal eigenvectors $e = u^1, u^2, u^3, \ldots, u^n$. We can write x as $x = c_1 e + c_2 u^2 + \cdots + c_n u^n$. Hence, $x^T x = \sum_{i=1}^{n} c_i^2$, and the requirement that $e^T x = 0$ is satisfied if and only if $c_1 = 0$. Therefore, our minimisation problem can be formulated as:

> Minimise $\frac{1}{4} \sum_{i=2}^{N} \lambda_i c_i^2$, subject to
> $\sum_{i=2}^{n} c_i^2 = n$, and $c_i \in \mathbb{R}$ $(i = 2, \ldots, n)$.

If $\lambda_2 < \lambda_3$, the object function is minimised for $c_2 = \sqrt{n}$, and $c_3 = \cdots = c_n = 0$.
As the solution of the original discrete optimisation problem, we take the vector x with components $x_i \in \{-1, 1\}$ that lies closest to the solution of the continuous problem. We obtain this vector by finding the median value among all the x_i's and mapping x_i values above the median to $+1$, and values below to -1. This gives a balanced decomposition with hopefully, a low cut-weight.

6.3 Example

We illustrate the spectral bisection algorithm by applying it to the mesh on the left in Fig. 13 [36]. The corresponding interdependency graph is shown on the right side of the figure. The Laplacian matrix L of this graph is

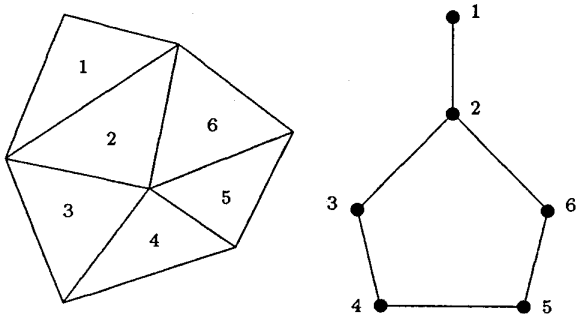


Fig. 13: Example mesh and corresponding interdependency graph [36].

$$L = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & -1 & 0 & 0 & -1 & 2 \end{bmatrix}.$$

The second smallest eigenvalue of L is $\lambda_2 = 0.6972$, and the corresponding eigenvector is

$$u^2 = [\, -0.78 \;\; -0.24 \;\; 0.12 \;\; 0.39 \;\; 0.39 \;\; 0.12 \,]^T.$$

Therefore the vector x is

$$x = [\, -1.91 \;\; -0.58 \;\; 0.29 \;\; 0.96 \;\; 0.96 \;\; 0.29 \,]^T.$$

On the basis of this we can partition the mesh as $P_1 = \{1,2,3\}$ and $P_2 = \{4,5,6\}$, or $P_1 = \{1,2,6\}$ and $P_2 = \{3,4,5\}$.

6.4 Laplacian spectrum of regular grids

In general, dropping the discreteness constraint in an optimisation problem and taking the discrete solution that lies closest to the solution of the relaxed continuous optimisation problem as the solution of the discrete optimisation problem is a dangerous technique that does not guarantee good solutions.
Confidence can be gained about the fact that using the spectrum of the Laplacian matrix of a graph does yield good partitionings, by studying this spectrum for regular grid graphs. The following results for the path graph and for the five-point grid have been taken from [33].

6.4.1 The path graph

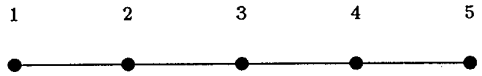


Fig. 14: Path graph with 5 vertices.

Let P_n denote the path graph on n vertices (see Fig. 14). In the following discussion, we assume that $n \geq 2$ is even. We number the vertices of the path from 1 to n in the natural order from left to right.
The Laplacian matrix of P_n is tridiagonal. Its eigenvalues are $\lambda_k = 4\sin^2[(k-1)\pi/(2n)]$ $(k = 1,\ldots,n)$, thus $0 = \lambda_1 < \lambda_2 < \cdots < \lambda_n$. An eigenvector x^k that corresponds to the eigenvalue λ_k has components

$$x_i^k = \cos\frac{(2i-1)(k-1)\,\pi}{2n}, \quad i = 1,\ldots,n.$$

Therefore, $\lambda_2 = 4\sin^2[\pi/(2n)]$, and $x_i^2 = \cos[(2i-1)\,\pi/(2n)]$. The components of x^2, plotted against the vertices of P_n decrease monotonically from left to right. The first $n/2$ components are positive and the last $n/2$ are negative. Thus bisection based on the separator $\sigma_i = x_i^2$ splits the path graph in the middle. Intuitively, it is clear that this is the optimal partitioning.

6.4.2 The five-point grid

We consider the $m \times n$ five-point grid, and without loss of generality take $m \leq n$. We assume that $n \geq 2$ is even.
The spectrum of the five-point grid can be derived from the spectrum of the path graph. The eigenvalues are

$$\mu_{k,l} = 4\left[\sin^2\left(\frac{(k-1)\,\pi}{2n}\right) + \sin^2\left(\frac{(l-1)\,\pi}{2m}\right)\right],$$
$$k = 1,\ldots,n; l = 1,\ldots,m.$$

An eigenvector $y^{k,l}$ that corresponds to the eigenvalue $\mu_{k,l}$ has components

$$y_{i,j}^{k,l} = \cos\frac{(2i-1)(k-1)\,\pi}{2n}\cos\frac{(2j-1)(l-1)\,\pi}{2m},$$
$$i = 1,\ldots,n; j = 1,\ldots,m.$$

The smallest eigenvalue $\mu_{1,1}$ is zero. If $m < n$, the second smallest eigenvalue is $\mu_{2,1} = 4\sin^2[\pi/(2n)]$ and the corresponding eigenvector $y^{(2,1)}$ has components $y_{i,j}^{2,1} = \cos[(2i-1)\,\pi/(2n)]$. The components of $y^{2,1}$ are constant along each column of m vertices, and the components decrease from left to right across a row. Columns numbered from 1 to $n/2$ have positive components, and the rest of the columns have negative components. The components of this eigenvector of the $m \times n$ five-point grid are shown in Fig. 15.

If $m = n$, $\mu_{1,2} = \mu_{2,1}$ and the second smallest eigenvalue of the Laplacian matrix has multiplicity two. The linearly independent eigenvectors $y^{1,2}$ and $y^{2,1}$ span the two-dimensional eigenspace that correspond to this eigenvalue. These vectors correspond respectively to bisecting the graph horizontally and vertically in the middle, which are indeed optimal solutions. Notice that in practice, an eigensolver will in general yield vectors that are linear combinations of these two optimal solutions.

6.5 Connectivity of the subgraphs

If the original graph is connected, it can be guaranteed that at least one of the resulting subgraphs will be connected too. This follows from the following theorem by Fiedler [35].

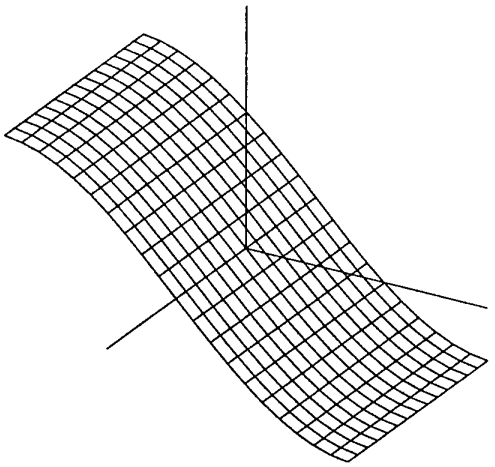


Fig. 15: The second Laplacian eigenvector of the five-point grid.

Theorem. *Let G be a connected graph and let x be an eigenvector, corresponding to the second smallest eigenvalue of the Laplacian matrix of the graph. For a real number $r \geq 0$, define $V_1(r) = \{v \in V \mid x_v \geq -r\}$. Then the subgraph induced by $V_1(r)$ is connected. Similarly, the subgraph, induced by the set $V_2(r) = \{v \in V \mid x_v \leq r\}$, is also connected. If $r = 0$, it is necessary to include the vertices with zero components in both sets V_1 and V_2 for the theorem to hold.*

In practice, most often both subgraphs will be connected. It must be noticed that in general it is not possible to bisect a connected graph in two connected and equally sized subgraphs anyway. A simple example of such a graph is shown in Fig. 16.

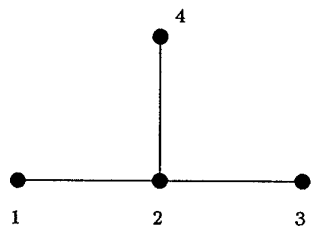


Fig. 16: A connected graph that cannot be bisected in connected and equally sized subgraphs.

6.6 Examples and experiments

In Table 1, some results obtained by Venkatakrishnan et al. [37] are presented. A mesh-vertex upwind finite volume scheme was used on a 64-processor iPSC/860 machine to solve the Euler equations around a multi-flap airfoil on Barth5, a two-dimensional triangular unstructured fluid dynamics mesh from NASA Ames with 15606 vertices, 45878 edges, 30269 faces and 949 boundary edges. The finite volume mesh was partitioned using the Spectral Bisection and the Coordinate Bisection technique on the original mesh-graph and using the Spectral Bisection method on the dependency graph of the mesh. For this example, the use of the Spectral Bisection technique leads to a performance that is 30% higher than if the Coordinate Bisection method is used.

Figure 17 shows the partitioning into eight subgrids of the mesh in Fig. 5 that one obtains with the Recursive Spectral Bisection algorithm. This method cuts only 258 edges (97 less than the greedy heuristic of Farhat), and yields eight connected subgrids. Notice also that the subgrids have good aspect ratios.

For the RYMAMO grid as well, the Recursive Spectral Bisection algorithm gives very good results. Figure 18 shows the partitioning. It cuts 280 edges (greedy heuristic: 365) and yields 10 connected parts (greedy heuristic: 22).

6.7 Generalisations of the spectral bisection algorithm

Hendrickson and Leland extended the spectral bisection method to quadri- and octasection of graphs [38]. Moreover, they also generalised it to the case that not only the edges but also the vertices are weighted [39]. Hendrickson and Leland show that the partitions that are obtained in this way are better than the ones obtained by recursively applying the bisection algorithm if the *hypercube hop* (or *Manhattan*) metric is used as the cost measure. Empirical study [40] has shown that this is an appropriate measure for modelling the performance of hypercube architecture machines since minimising this metric corresponds to minimising congestion within the communication network. The hop metric is also appropriate for and three dimensional mesh architectures.

Van Driessche and Roose [5, 41] developed a spectral bisection algorithm for the *constrained* graph bisectioning problem, a generalisation of the graph bisectioning problem in which the assignment of part of the vertices is imposed a priori. Although this spectral algorithm was originally developed for dynamic load balancing, it is

Table 1: Comparison between Spectral Bisection and Coordinate Bisection.

Method	Spectral Bisection	Coordinate Bisection	Spectral (Depend. Graph)
Total time (sec)	0.31	0.41	0.31
Performance (Mflops)	187.5	143	188
Communication time (sec)	0.084	0.173	0.082
Average number of neighbours	4.7	6.7	4.5
Number of intern. bound. vertices	1819	2631	1791
Maximum number of neighbours	12	14	14
Maximum number of vertices	101	120	109

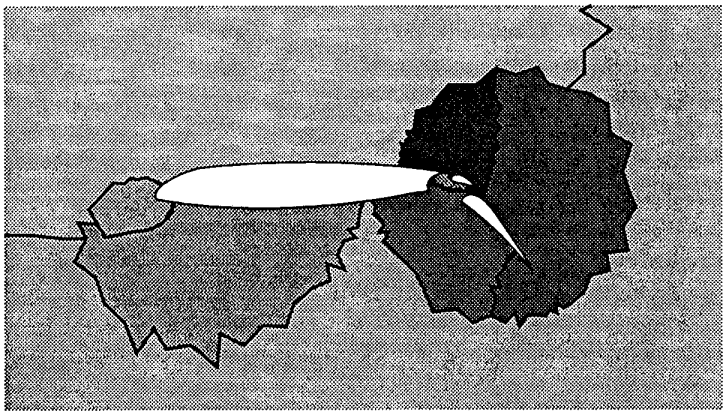


Fig. 17: Partitioning into 8 subgrids of the finite element grid in Fig. 5 with the Recursive Spectral Bisection algorithm.

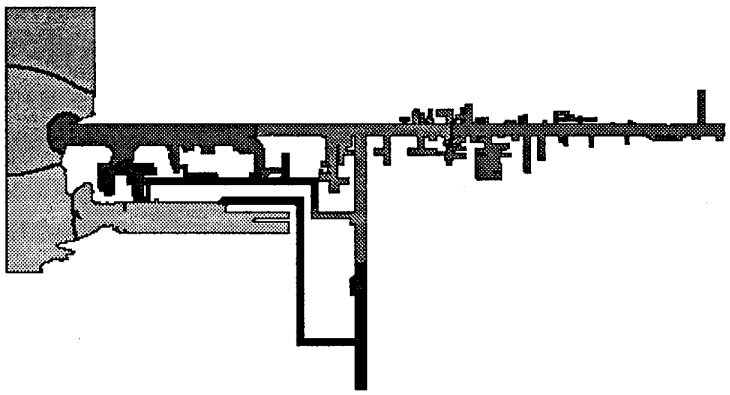


Fig. 18: Partitioning into 8 subgrids of the RYMAMO grid with the Recursive Spectral Bisection algorithm.

also useful for static load balancing. By solving a sequence of constrained graph bisectioning problems, it is possible to take the mapping problem already into account during the mesh partitioning [42]. In this way, it is possible to ensure that the subgrids are assigned to processors that are close to each other in the communication topology. Moreover, the number of neighbouring subgrids per subgrid is smaller than if the standard Recursive Spectral Bisection is used.
Figure 19 shows the partitioning of the mesh in Fig. 5, that is yielded by this Recursive Constrained Spectral Bisection algorithm for a hypercube topology. This partition cuts slightly more edges than the partition one obtains with the standard Recursive Spectral Bisection algorithm (viz. 282 versus 258) but the maximal number of adjacent subgrids per subgrid is smaller (viz. 4 versus 5). Moreover, a small readjustment of the boundaries is sufficient to ensure that each subgrid has no more than 3 neighbouring subgrids. Fig. 20 illustrates that the interdependency topology between the subgrids more closely matches the hypercube communication topology of the parallel computer, than if the standard Recursive Spectral Bisection algorithm is used.

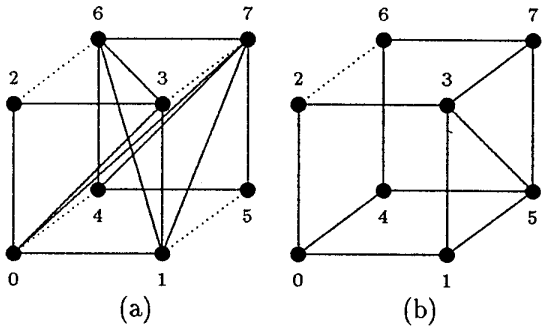


Fig. 20: (a) Interdependency topology of the subgrids in Fig. 17 *(standard Bisection Algorithm)*. (b) Interdependency topology of the subgrids in Fig. 19 *(Constrained Bisection Algorithm)*.

6.8 Calculation of the eigenvectors

For the spectral bisection technique, one has to calculate the eigenvector that corresponds to the second smallest eigenvalue of a large, sparse (and symmetric) matrix. The *Lanczos* algorithm [43] is particularly well-suited for this problem because it only uses the matrix through matrix-vector products, so that the sparsity of the Laplacian matrix can be exploited, and because it typically converges to the extreme eigenvalues and eigenvectors of a matrix in $O(\sqrt{n})$ steps, each of which has a complexity of order n.
For large graphs, the calculation time and especially the memory requirements of the Lanczos algorithm are often unacceptable. Barnard and Simon [44, 45] introduced a *multilevel algorithm* that calculates the Fiedler vector considerably faster and with less memory than the Lanczos algorithm. This algorithm first constructs a sequence of graphs, in such a way that the initial graph is the first graph in the sequence, and that the other graphs are the *contractions* of the previous graph in the sequence.
A graph is contracted as follows. First, a maximal number of non-adjacent vertices are selected that will form the vertex set of the contracted graph. Next, the edge set is constructed by growing domains in the original graph round the vertices of the contracted graph, and by adding an edge to the contracted graph whenever two domains intersect.
Once the sequence of graph contractions has been constructed, the Fiedler vector of the smallest graph is calculated and is *prolongated* to the previous graph in the sequence. This prolongation is already a good approximation for the Fiedler vector of this graph and can therefore be rapidly improved with *Rayleigh quotient iteration*. This procedure is recursively applied until the Fiedler vector of the first graph in the sequence, i.e. the Fiedler vector of the original graph, has been calculated.
Using this technique, Barnard and Simon claim to obtain partitions with comparable quality in up to 20 times less time than with the Lanczos algorithm.
Van Driessche and Roose [46] have presented an alternative graph contraction algorithm that uses the same procedure to select the vertex set but that assigns weights to the edges of the contracted graph. This algorithm yields very good eigenvector approximations. They are also able to give a formal analysis that helps to explain why and when the algorithm gives such good results.
Hendrickson and Leland [47] use a completely different graph contraction algorithm, in which they contract some edges of the graph. They first search for a *maximal matching* in the graph. This is a maximal set of edges, no two of which are incident on the same vertex. The edges in this set are then contracted as follows. The vertices joined by an edge that must be contracted, are merged into one 'super vertex', and the new super vertex is given edges to the union of the neighbours of the merged vertices. The weight of the super vertex is set equal to the sum of the weights

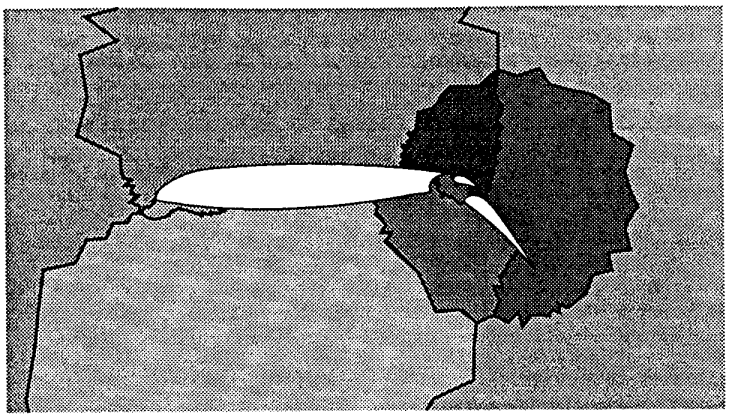


Fig. 19: Partitioning into 8 subgrids of the finite element grid in Fig. 5 with the Recursive Constrained Spectral Bisection algorithm.

of its constituent vertices. Edge weights are left unchanged unless both merged vertices are adjacent to the same neighbour. In this case the new edge that represents the two original edges is given a weight equal to the sum of the weights of the two edges it replaces.

7 COMBINATIONS OF PARTITIONING ALGORITHMS

The most powerful partitioning heuristics use a combination of the techniques discussed in the previous sections. We will discuss three examples, viz. the combination of recursive spectral and inertial bisection with the Kernighan-Lin heuristic, the two-step approach of Vanderstraeten and Keunings to optimise complicated cost functions, and the multilevel partitioning algorithm of Bui and Jones and Hendrickson and Leland.

7.1 Improving a partition with the Kernighan-Lin heuristic

The Kernighan-Lin heuristic is an iterative algorithm that improves an initial partition by repeatedly swapping elements among the partitions. Starting with a random assignment of grid points to processors usually gives disappointing results because of the inherently greedy and local nature of the algorithm. On the other hand, the Recursive Spectral Bisection algorithm often yields partitions that are globally good but that perform poorly in the fine details. It is therefore advantageous to calculate an initial partitioning with the Recursive Spectral Bisection algorithm, and improve this with the Kernighan-Lin heuristic. As an example, in Fig. 17, the boundaries between the subdomains are not very smooth but this is considerably improved, and the number of cut edges reduced from 258 to 226, with the Kernighan-Lin heuristic. Figure 21 shows the partitioning into eight subgrids of the mesh in Fig. 5, that results from applying the Kernighan-Lin heuristic to the partitioning in Fig. 17. Notice that the boundaries between the subgrids have become much smoother.

The Recursive Inertial Bisection algorithm also benefits greatly from a Kernighan-Lin post-processing step. The quality of the resulting partitions is often comparable to what one obtains with Recursive Spectral Bisection (but worse than what the combination of spectral bisection with Kernighan-Lin gives), while the calculation time is considerably lower.

Figure 22 shows the partitioning of the RYMAMO mesh after applying the Kernighan-Lin heuristic to the result of the inertial bisection algorithm. This partition has 13 connected parts (20 without the Kernighan-Lin heuristic) and cuts 281 edges (485 without the Kernighan-Lin heuristic), thus only 1 edge more than the partition, obtained with the Recursive Spectral Bisection algorithm.

7.2 The multilevel-Kernighan-Lin algorithm of Hendrickson and Leland

7.2.1 Description

The good performance of the Kernighan-Lin heuristic at locally improving a partition that is

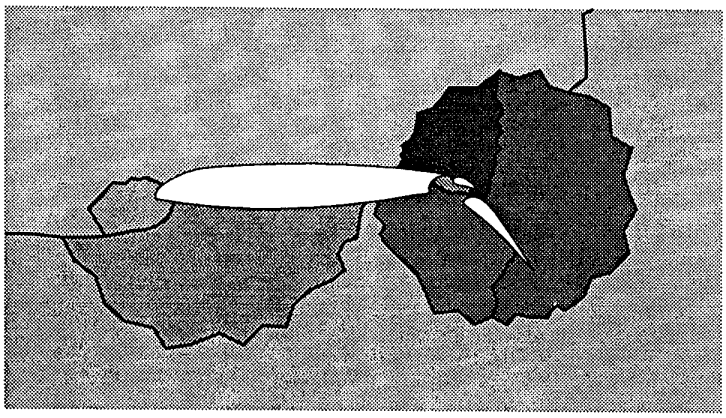


Fig. 21: Partitioning into 8 subgrids of the finite element grid in Fig. 5 with the Recursive Spectral Bisection algorithm and the Kernighan-Lin heuristic.

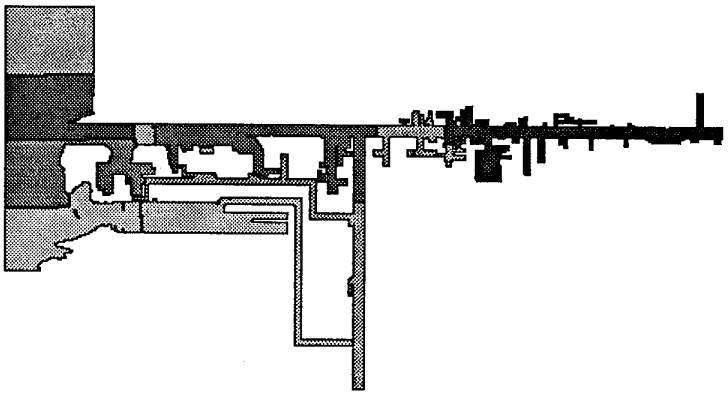


Fig. 22: Partitioning into 8 subgrids of the RYMAMO grid with the Recursive Inertial Bisection algorithm and the Kernighan-Lin heuristic.

already globally good, is also put to use in the multilevel algorithms of Bui and Jones [48] and Hendrickson and Leland [47].
The idea is to create a sequence of increasingly smaller graphs that in some sense approximate the original graph. The smallest graph is partitioned (with, say, Recursive Spectral Bisection) and this *partition* is projected back through the intermediate levels. Every few levels of projection, the Kernighan-Lin heuristic is used to refine the partition.
For the construction of the smaller graphs, Hendrickson and Leland use the same contraction algorithm, described in Section 6.8, that they use in the multilevel algorithm for the calculation of the Fiedler vector of a graph. Bui and Jones have proposed a similar algorithm, but in contrast to the algorithm of Hendrickson and Leland, it does not use vertex and edge weights. In practice, the difference between the two methods is small with neither method being consistently superior [49].

Table 2: Partitioning of Barth5 with the multilevel algorithm of Hendrickson and Leland, and with the Recursive Inertial and Spectral Bisection algorithms, both with and without Kernighan-Lin refinement [30].

	Inertial		Spectral		Multilevel
	Alone	+ KL	Alone	+ KL	
Number of cut edges					
2	245	200	200	139	175
4	897	520	521	367	379
8	1441	917	888	693	662
16	2266	1383	1382	1148	1106
32	3141	2057	2075	1824	1824
64	4253	3128	3170	2927	2943
Calculation time in seconds					
	2.0	13.8	136.7	146.0	28.4

7.2.2 Examples

Figure 23 shows the partitioning into eight subgrids of the mesh in Fig. 5, which one obtains with the multilevel algorithm of Hendrickson and Leland. This partition cuts 202 edges, fewer than the partition that we obtained with the Recursive Spectral Bisection algorithm even if it is improved with the Kernighan-Lin heuristic. Notice that the subgrids have very good aspect ratios.
For the RYMAMO grid as well, the multilevel algorithm finds a partition that only cuts a small number of edges. Figure 24 shows the partition into eight subgrids It cuts 246 edges (greedy heuristic: 365, Recursive Spectral Bisection algorithm: 280).
Table 2, which is taken from [30], gives results about the partitioning of Barth5, a two-dimensional fluid dynamics mesh, for which we presented results in Section 6.6 that demonstrate the influence of the partitioning on the calculation time of an Euler solver. The dual graph, which has 15606 vertices and 45878 edges, was partitioned into 2, 4, 8, 16, 32 and 64 parts, both with the inertial and the spectral algorithm, alone and in combination with the Kernighan-Lin heuristic. The graph was also partitioned with the multilevel algorithm.
A comparison of the number of cut edges on one hand and the calculation times on the other, demonstrates that the combination of Recursive Inertial Bisection with the Kernighan-Lin heuristic yields partitions of comparable quality with the Recursive Spectral Bisection algorithm at a fraction of the cost. However, the most cost-effective method turns out to be the multilevel algorithm: it finds partitions that are comparable to or even better than what one obtains with a combination of the Recursive Spectral Bisection algorithm and the Kernighan-Lin heuristic while the calculation time (and the memory usage) is considerably smaller.

7.3 Improving a partition with a stochastic optimisation algorithm

Vanderstraeten and Keunings have tried to improve an initial partition with stochastic optimisation algorithms [50]. They have tested three algorithms, viz. simulated annealing (see Section 3.1 and the references therein), tabu search [51, 52], and stochastic evolution [53]. These algorithms are expensive, but they can start from a good initial solution. Moreover, only a relatively small search space must be explored because only subdomain interfaces are readjusted.

This two-step approach, first generating an initial mesh decomposition with a suboptimal but fast partitioning algorithm, and next optimising this partition with a stochastic optimization algorithm, is able to generate partitions with smooth boundaries and a small number of cut edges. Moreover, thanks to the general applicability of the stochastic algorithms, it is also possible to optimise much more complicated cost functions that do not just take the number of cut edges into account [54].

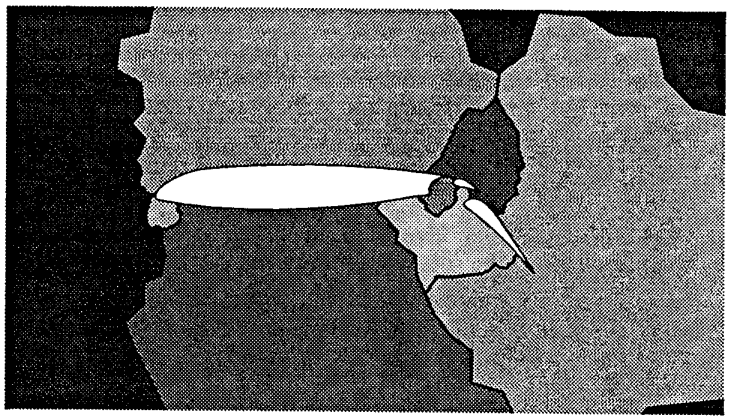


Fig. 23: Partitioning into 8 subgrids of the finite element grid in Fig. 5 with the multilevel algorithm of Hendrickson and Leland.

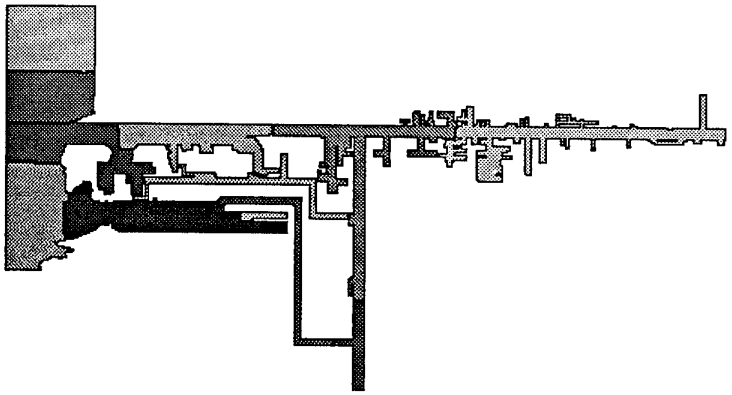


Fig. 24: Partitioning into 8 subgrids of the RYMAMO grid with the multilevel algorithm of Hendrickson and Leland.

8 SOFTWARE TOOLS FOR PARTITIONING

8.1 Introduction

Graph partitioning, and hence mesh partitioning for parallel computing, is by now a fairly well-understood problem, and several efficient software tools exist for this purpose. We will discuss two tools that contain state-of-the-art partitioning algorithms and that are well-supported and -documented, viz. *Chaco* and *TOP/DOMDEC*. For most CFD applications these tools will be sufficient to obtain good mesh partitions, so that it is not necessary for users to develop their own code.
Some open issues remain however. Firstly, these tools are meant to be used as a pre-processing tool on a sequential computer. They are therefore not suitable for parallel applications that also want to calculate the partitions in parallel, like e.g. applications that need quasi-static load balancing. Secondly, it has been argued that the standard optimisation criterion (minimal number of cut edges) is not suitable for many applications [8, 30, 54]. Although the tools that will be discussed here, provide a limited number of optimisation criteria besides this standard criterion, it is still not clear whether any of these can accurately model the execution of an application on a parallel computer, where factors like communication link contention and cache usage, which are difficult to model, often greatly influence performance.

8.2 Chaco

8.2.1 Introduction

Chaco is a software package designed to partition graphs. It was written by Bruce Hendrickson and Robert Leland of Sandia National Laboratories (Albuquerque, New Mexico, USA). Version 1.0 was released in 1993. The much improved Version 2.0 will be released in May 1995. It is this version that will be discussed here.

8.2.2 Description

Chaco implements four classes of global partitioning algorithms:

Simple: three very simple partitioning schemes, in which vertices are assigned to processes randomly or according to their numbering in the original graph.

Inertial: recursive inertial bi-, quadri- or octasection (see Section 5).

Spectral: recursive spectral bi-, quadri- or octasection (see Section 6). The user can specify whether the eigenvectors of the Laplacian matrix must be calculated with a Lanczos algorithm or with a multilevel algorithm.

Multilevel: the multilevel algorithm described in Section 7.2.

The output of any of these global methods can be fed into a Kernighan-Lin algorithm which locally refines the partition.
Chaco only offers graph partitioning and uses a non-graphics interface, so there are no visualisation tools, or tools to create meshes. However, several people have written MATLAB interfaces for Chaco. In particular, John Gilbert at Xerox Park has written and agreed to maintain visualisation software that is freely available.
Chaco is normally used interactively with the program prompting for the name of input and output files, for the number of sets the graph should be partitioned into, and also for data about the requested partitioning heuristic. The behaviour of Chaco is determined by a large number of parameters and tolerances, for which the program chooses suitable default values. However, the user can create a file with alternative values, and is thus able to experiment with Chaco. Although normally used interactively, Chaco also provides an interface routine that allows it to be called from user code.

8.2.3 Availability

Chaco is available under license from Sandia National Laboratories. It is distributed along with technical documentation and some sample input files via e-mail. To obtain a copy, contact the authors

Bruce Hendrickson
Dept. 1422, Mail Stop 1110
Sandia National Laboratories
Albuquerque, NM 87185, U.S.A.
E-mail: `bah@cs.sandia.gov`

and

Robert Leland
Dept. 1424, Mail Stop 1110
Sandia National Laboratories
Albuquerque, NM 87185, U.S.A.
E-mail: `leland@cs.sandia.gov`

At the time of writing this text, licensing conditions for academics were not completely fixed. For corporations, Chaco will be licensed on a case-by-case basis.

Chaco is written in Kernighan and Ritchie style, but ANSI-compliant C, and, except for the mathematics library, uses no external libraries. It should therefore compile and run correctly under any UNIX system with any ANSI-C standard compiler, and can usually be compiled without too many problems with non-standard compilers as well.

8.3 TOP/DOMDEC

8.3.1 Introduction

TOP/DOMDEC is, in the words of the manual [55], a Totally Object oriented Package for visualisation, DOMain DEComposition, and parallel processing on finite element meshes. It was developed by PGSoft and by the research group of C. Farhat at the University of Colorado at Boulder.
As a partitioning tool, TOP/DOMDEC offers several state-of-the-art mesh partitioning algorithms, whose partitions can subsequently be smoothed and optimised using one of several non-deterministic optimisation schemes. TOP/DOMDEC also provides real-time means for assessing a priori the quality of a mesh partition and discriminating between different partitioning algorithms. The user interface includes high speed three-dimensional graphics, an interprocessor communication simulator with a built-in cost model for some real-world parallel computers and for a generic message-passing parallel computer, and an output function that automatically generates parallel I/O data structures.

8.3.2 Description

Here, we will only concisely describe TOP/DOMDEC as a mesh partitioning tool. A more thorough discussion, which also discusses the other capabilities of TOP/DOMDEC can be found in the manual [55], or in Chapter 9 of [26]. Just like in Chaco, the idea in TOP/DOMDEC is that you first partition the mesh with a global partitioning algorithm, and that this initial partition is subsequently refined with a local optimisation algorithm. TOP/DOMDEC provides the following global partitioning algorithms:

Greedy: the greedy heuristic of Farhat (see Section 4).

RCM and Recursive RCM: the Reverse Cuthill-McKee ordering scheme (RCM), and the Recursive RCM algorithm (see Section 3.2.2).

Principal Inertia (PI) and Recursive PI: the Principal Inertia algorithm projects all the mesh points onto the principal inertia direction of the mesh and sorts the mesh points according to this projection into the requested number of subdomains. The Recursive PI algorithm uses the above procedure recursively to bisect the mesh and submeshes and is therefore identical to the Recursive Inertial Bisection heuristic, described in Section 5.4.

Recursive Spectral Bisection: the standard Recursive Spectral Bisection algorithm (see Section 6). The Fiedler vector is calculated with the multilevel algorithm of Barnard and Simon [44] (see Section 6.8).

Recursive Graph Bisection: the Recursive Graph Bisection heuristic described in Section 3.2.4.

1D Topology Frontal Algorithm: this algorithm tries to ensure that every subdomain has two neighbours at most. It was developed to partition meshes on which subdomain-based multifrontal solution schemes are used [56].

Three non-deterministic optimisation algorithms are provided to further optimise the partitions, viz. tabu search, simulated annealing, and stochastic evolution. Since these algorithms are very general, it is possible in principle to optimise the partitions for very complicated cost functions. The following functions are provided in TOP/DOMDEC: interface size, subdomain frontwidth, the product of interface size and subdomain frontwidth, node-wise load balance, element-wise load balance, edge-wise load balance, subdomains aspect ratio, or a weighted sum of the above items.

8.3.3 Availability

To obtain TOP/DOMDEC, contact

Charbel Farhat
College of Engineering
University of Colorado
Campus Box 429
Boulder, CO 80309, U.S.A.
E-mail: `charbel@boulder.colorado.edu`

Users must pay a one-time fee, the amount of which depends on whether the requestor is a research partner, a research institution, a US government sponsored institution, or an industrial corporation.
TOP/DOMDEC is written in C++. It currently runs on the SGI Iris and the IBM RISC System/6000 with GL graphics workstations. However, if the graphics capabilities are not required, TOP/DOMDEC can run on other systems as well.

9 ACKNOWLEDGMENTS

This paper presents research results of the Belgian Incentive Program "Information Technology"—Computer Science of the Future (IT/IF/5), and of the Belgian Programme on Interuniversity Poles of Attraction (IUAP 17), initiated by the Belgian State, Prime Minister's Office for Science, Technology and Culture. The scientific responsibility rests with its authors. The RYMAMO grid was kindly made available by the RIKZ (Rijksinstituut voor Kust en Zee), The Netherlands. The partitions in Fig. 21, 22, 23, and 24 have been calculated with Chaco. We thank Bruce Hendrickson for carefully reading the first version of this text and for making numerous suggestions that allowed us to improve the text in many places.

REFERENCES

[1] J. De Keyser and D. Roose. A software tool for load balanced adaptive multiple grids on distributed memory computers. In *Proceedings of the 6th Distributed Memory Computing Conference*, pages 122–128. IEEE Computer Society Press, 1991.

[2] D. Roose, J. De Keyser, and R. Van Driessche. Load balancing grid-oriented applications on distributed memory parallel computers. In P. Dewilde and J. Vandewalle, editors, *Computer Systems and Software Engineering*, pages 191–216. Kluwer Academic Publishers, 1992.

[3] R. D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice and Experience*, 3(5):457–481, Oct. 1991.

[4] E. Pramono, H. D. Simon, and A. Sohn. Dynamic load balancing for finite element calculations on parallel computers. In David H. Bailey et al., editors, *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 599–604. SIAM, 1995.

[5] R. Van Driessche and D. Roose. A spectral algorithm for constrained graph partitioning I: The bisection case. Report TW 216, K.U.Leuven, Dept. of Computer Science, Belgium, Oct. 1994.

[6] R. Van Driessche and D. Roose. An improved spectral bisection algorithm and its application to dynamic load balancing. *Parallel Computing*, 21:29–48, 1995.

[7] L. Beernaert, D. Roose, R. Struijs, and H. Deconinck. A multigrid solver for the Euler equations on distributed memory parallel computers. *IMACS J. Appl. Num. Math.*, 7:379–393, 1991.

[8] C. Farhat and M. Lesoinne. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *International Journal for Numerical Methods in Engineering*, 36:745–764, 1993.

[9] F. X. Roux. Acceleration of the outer conjugate gradient by reorthogonalization for a domain decomposition method for structural analysis problems. In *Proceedings of the Third International Conference on Supercomputing*, pages 471–477, 1989.

[10] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[11] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing : Theory and Applications.* D. Reidel Publishing Company, Dordrecht, 1987.

[12] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part I: Graph partitioning. *Opns. Res.*, 37:865–892, 1989.

[13] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[14] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[15] R. Van Driessche and R. Piessens. Load balancing with genetic algorithms. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2*, pages 341–350. North-Holland, Amsterdam, 1992.

[16] G. von Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In R. K. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 45–52. Morgan-Kaufman, 1991.

[17] G. von Laszewski and H. Mühlenbein. Partitioning a graph with a parallel genetic algorithm. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, number 496 in Lecture

Notes in Computer Science, pages 165–169. Springer-Verlag, 1990.

[18] P. Sadayappan and F. Ercal. Nearest-neighbor mapping of finite-element graphs onto processor meshes. *IEEE Trans. Computers*, C-36(12):1408–1424, 1987.

[19] P. Sadayappan, F. Ercal, and J. Ramanujam. Cluster partitioning approaches to mapping parallel programs onto a hypercube. *Parallel Computing*, 13(1):1–16, 1990.

[20] J. G. Malone. Automated mesh decomposition and concurrent finite element analysis for hypercube multiprocessors computers. *Comp. Meth. Appl. Mech. Eng.*, 70(1):277–289, 1988.

[21] W. Chan and A. George. A linear time implementation of the Reverse Cuthill McKee algorithm. *BIT*, 20:8–14, 1980.

[22] C. Farhat. A simple and efficient automatic FEM domain decomposer. *Computers and Structures*, 28(5):579–602, 1988.

[23] M. Al-Nasra and D. T. Nguyen. An algorithm for domain decomposition in finite element analysis. *Computers and Structures*, 39(3/4):277–289, 1991.

[24] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49:291–308, 1970.

[25] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, pages 175–181. IEEE, 1982.

[26] C. Farhat and F. X. Roux. *Implicit Parallel Processing in Structural Mechanics.* Computational Mechanics Advances. 1993.

[27] H. D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2/3):135–148, 1991.

[28] E. D. de Goede and G. S. Stelling. A multi block method for the three-dimensional shallow water equations. In L. Dekker, W. Smit, and J. C. Zuidervaart, editors, *Massively Parallel Processing Applications and Development*, pages 71–78. Elsevier, 1994.

[29] J. De Keyser and D. Roose. Grid partitioning by inertial recursive bisection. Report TW 174, K.U.Leuven, Dept. of Computer Science, Belgium, July 1992.

[30] R. Leland and B. Hendrickson. An empirical study of static load balancing algorithms. In *Proceedings of the Scalable High-Performance Computing Conference*, pages 682–685, 1994.

[31] W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM J. Res. Develop.*, 17:420–425, 1973.

[32] E. R. Barnes. An algorithm for partitioning the nodes of a graph. *SIAM J. Algebraic Discrete Methods*, 3(4):541–550, 1982.

[33] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[34] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[35] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(100):619–633, 1975.

[36] M. Berzins and C. Walshaw. Tutorial on domain decomposition for unstructured meshes. ERCOFTAC Workshop on Domain Decomposition for CFD, University of Leeds, Sept. 1992.

[37] V. Venkatakrishnan, H. D. Simon, and T. J. Barth. A MIMD implementation of a parallel Euler solver for unstructured grids. In J. Dongarra et al., editors, *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*, pages 253–256. SIAM, 1992.

[38] B. Hendrickson and R. Leland. An improved spectral load balancing method. In Richard F. Sincovec et al., editors, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 953–961. SIAM, 1993.

[39] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. Technical Report SAND92-1460, Sandia National Laboratories, Albuquerque, NM, Sept. 1992.

[40] S. Hammond. *Mapping unstructured grid computations to massively parallel computers.* PhD thesis, Rensselaer Polytechnic Institute, Dept. of Computer Science, Rensselaer, NY, 1992.

[41] R. Van Driessche and D. Roose. Dynamic load balancing with a spectral bisection algorithm for the constrained graph partitioning problem. In *High-Performance Computing and Networking*, number 919 in Lecture Notes in Computer Science, pages 392–397. Springer, 1995.

[42] B. Hendrickson, R. Leland, and R. Van Driessche. Enhancing data locality by terminal propagation. Submitted to the minitrack on Partitioning and Scheduling at the HICSS-29 Conference, Maui, Hawaii, January 3–6, 1996.

[43] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, MD, second edition, 1989.

[44] S. T. Barnard and H. D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, Apr. 1994.

[45] S. T. Barnard and H. D. Simon. A parallel implementation of multilevel recursive spectral bisection for application to adaptive unstructured meshes. In David H. Bailey et al., editors, *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 627–632. SIAM, 1995.

[46] R. Van Driessche and D. Roose. A graph contraction algorithm for the fast calculation of the Fiedler vector of a graph. In David H. Bailey et al., editors, *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 621–626. SIAM, 1995.

[47] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, Albuquerque, NM, Oct. 1993.

[48] T. N. Bui and C. Jones. A heuristic for reducing fill-in in sparse matrix factorization. In Richard F. Sincovec et al., editors, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 445–452. SIAM, 1993.

[49] B. Hendrickson and R. Leland. The Chaco user's guide. Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, NM, Oct. 1994.

[50] D. Vanderstraeten and R. Keunings. Optimized partitioning of unstructured finite element meshes. *International Journal for Numerical Methods in Engineering*, 38:433–450, 1995.

[51] F. Glover, C. McMillan, and B. Novick. Interactive decision software and computer graphics for architectural and space planning. *Ann. Opns. Res.*, 5:557–573, 1985.

[52] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.

[53] Y. G. Saab and V. B. Rao. Combinatorial optimization by stochastic evolution. *IEEE Trans. Computer-Aided Design*, 10(4):525–535, Apr. 1991.

[54] D. Vanderstraeten, R. Keunings, and C. Farhat. Beyond conventional mesh partitioning algorithms and the minimum edge cut criterion: Impact on realistic applications. In David H. Bailey et al., editors, *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 611–614. SIAM, 1995.

[55] M. Sharp and C. Farhat. *TOP/DOMDEC, a Totally Object Oriented Program for Visualisation, Domain Decomposition and Parallel Processing. User's Manual.* PGSoft and The University of Colorado, Boulder, Feb. 1994.

[56] D. Vanderstraeten, O. Zone, R. Keunings, and L. Wolsey. Non-deterministic heuristics for automatic domain decomposition in direct parallel finite element calculations. In Richard F. Sincovec et al., editors, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 929–932. SIAM, 1993.

Parallel Iterative Solution Methods for Linear Systems arising from Discretized PDE's

Henk A. Van der Vorst
Mathematical Institute, University of Utrecht
PO Box 80.010
NL-3508 TA Utrecht, the Netherlands
e-mail: vorst@math.ruu.nl

1 Introduction

In these notes we will present an overview of a number of related iterative methods for the solution of linear systems of equations. These methods are so-called Krylov projection type methods and they include popular methods as Conjugate Gradients, Bi-Conjugate Gradients, CGS, Bi-CGSTAB, QMR, LSQR and GMRES. We will show how these methods can be derived from simple basic iteration formulas. We will not give convergence proofs, but we will refer for these, as far as available, to litterature.
Iterative methods are often used in combination with so-called preconditioning operators (approximations for the inverses of the operator of the system to be solved). Since these preconditioners are not essential in the derivation of the iterative methods, we will not give much attention to them in these notes. However, in most of the actual iteration schemes, we have included them in order to facilitate the use of these schemes in actual computations.
For the application of the iterative schemes one usually thinks of linear sparse systems, e.g., like those arising in the finite element or finite difference approximations of (systems of) partial differential equations. However, the structure of the operators plays no explicit role in any of these schemes, and these schemes might also successfully be used to solve certain large dense linear systems. Depending on the situation that might be attractive in terms of numbers of floating point operations.

It will turn out that all of the iterative are parallelizable in a straight forward manner. However, especially for computers with a memory hierarchy (i.e., like cache or vector registers), and for distributed memory computers, the performance can often be improved significantly through rescheduling of the operations. We will discuss parallel implementations, and occasionally we will report on experimental findings.

2 Direct versus Iterative

1. Standard Gaussian elimination leads to fill-in, and this makes the method often expensive. Usually large sparse matrices are related to some grid or network. In a 3D situation this leads typically to a bandwidth $\sim n^{\frac{2}{3}}$ ($= m^2$ and $m^3 = n$, $1/m$ the gridsize).
The number of flops is then typically $\mathcal{O}(nm^4) \sim n^{2\frac{1}{3}}$ [36, 25]. For 2D problems the bandwidth is $\sim n^{\frac{1}{2}}$, so that the number of flops for a direct method then varies like n^2.
If one has to solve many systems with different right-hand sides, then one has to decompose the matrix only once after which the costs for solving each system will vary like $n^{\frac{5}{3}}$ for 3D problems, and like $n^{\frac{3}{2}}$ for 2D problems.

2. For symmetric positive definite systems the error reduction per iteration step of CG is $\sim \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$, with $\kappa = \|A\|_2\|A^{-1}\|_2$ [14, 2, 35].
For discretized second order pde's, over grids with gridsize $\frac{1}{m}$ we typically see $\kappa \sim m^2$. Hence, for 3D problems we have that $\kappa \sim n^{\frac{2}{3}}$, and for 2D problems: $\kappa \sim n$. For an error reduction of ϵ we must have that

$$\left(\frac{1-\frac{1}{\sqrt{\kappa}}}{1+\frac{1}{\sqrt{\kappa}}}\right)^j \approx (1-\frac{2}{\sqrt{\kappa}})^j \approx e^{-\frac{2j}{\sqrt{\kappa}}} < \epsilon.$$

For 3D problems we have that

$$\Rightarrow \quad j \sim -\frac{\log\epsilon}{2}\sqrt{\kappa} \approx -\frac{\log\epsilon}{2}n^{\frac{1}{3}},$$

whereas for 2D problems:

$$j \approx -\frac{\log\epsilon}{2}n^{\frac{1}{2}}.$$

If we assume the number of flops per iteration to be $\sim fn$ (f stands for the number of nonzeros per row of the matrix and the overhead per unknown introduced by the iterative scheme)
$\Rightarrow$ flops per reduction with ϵ:
$\sim -fn^{\frac{4}{3}}\log\epsilon$ for 3D problems,
and $\sim -fn^{\frac{3}{2}}\log\epsilon$ for 2D problems.

Paper presented in an AGARD-FDP-VKI Special Course on "Parallel Computing in CFD", held at the VKI, Rhode-Saint-Genèse, Belgium, from 15-19 May 1995 and 16-20 October 1995 at NASA Ames, United States and published in R-807.

Conclusion: If we have to solve one system at a time, then for large n, or small f, or modest ϵ:

Iterative methods may be preferable.

If we have to solve many similar systems with different right-hand side, and if we assume their number to be so large that the costs for constructing the decomposition of A is relatively small per system, then it seems likely that for 2D problems direct methods may be more efficient, whereas for 3D problems this is still doubtful, since the flops count for a direct solution method varies like $n^{\frac{5}{3}}$, and the number of flops for the iterative solver (for the model situation) varies like $n^{\frac{4}{3}}$.

Example
The above given arguments are quite nicely illustrated by observations made by Horst Simon [74]. He expects that by the end of this century we will have to solve repeatedly linear problems with some 5×10^9 unknowns. For what he believes to be a model problem at that time, he has estimated the CPU time required by the most economic direct method, available at present, as 520,040 years, provided that the computation can be carried out at a speed of 1 TFLOP. On the other hand, he estimates the CPU time for preconditioned conjugate gradients, assuming still a processing speed of 1 TFLOPS, as 575 seconds. Though we should not take it for granted that in particular the preconditioning part can be carried out at that high processing speed (for the direct solver this is more likely), we see that the differences in CPU time requirements are gigantic, indeed (we will come to this point in more detail).
Also the requirements for memory space for the iterative methods are typically smaller by orders of magnitude. This is often the argument for the usage of iterative methods in 2D situations, when flop counts for both classes of methods are more or less comparable.

Remarks:

- With suitable preconditioning we may have $\sqrt{\kappa} \sim n^{\frac{1}{6}}$ and the flops count then becomes

$$\sim -fn^{\frac{7}{6}} \log \epsilon,$$

  see, e.g., [37].

- For classes of problems some methods may even be faster: multigrid, fast poisson solvers.

- Storage considerations are also in favour of iterative methods.

- For matrices that are not positive definite symmetric the situation can be more problematic: it is often difficult to find the proper iterative method or a suitable preconditioner. However, for projection type methods, like GMRES, Bi-CG, CGS, and Bi-CGSTAB we often see that the flops counts vary as for CG.

- Iterative methods can be attractive even when the matrix is dense. Again, in the positive definite symmetric case, if the condition number is $n^{2-2\epsilon}$ then, since the amount of work per iteration step is $\sim n^2$, and the number of iteration steps $\sim n^{1-\epsilon}$, the total work estimate is roughly proportional to $n^{3-\epsilon}$, and this is asymptotically less than the amount of work for Choleski's method, which varies like $\sim n^3$.

The question remains at the moment how well iterative methods can take advantage of modern computer architectures. From Dongarra's linpack benchmark [22] it may be concluded that the solution of a dense linear system can (in principle) be computed with computational speeds close to peak speeds on most computers. This is already the case for systems of, say, order 50000 on parallel machines with as many as 1024 processors.
In sharp contrast with the dense case are computational speeds reported in [24] for the preconditioned as well as the unpreconditioned conjugate gradient method (ICCG and CG, respectively).

In [24] a test problem was taken, generated by discretizing a three-dimensional elliptic partial differential equation by the standard 7-point central difference scheme over a three-dimensional rectangular grid, with 100 unknowns in each direction ($m = 100$, $n = 1,000,000$). The observed computational speeds for several machines (1 processor in each case) are given in Table 1.

3 Basic iteration method

A very basic idea, that leads to many effective iterative solvers, is to to split the matrix of a given linear system in the sum of two matrices, one of which a matrix that would have led to a system that can easily be solved. The most simple splitting we can think of is $A = I-(I-A)$. Given the linear system $Ax = b$, this splitting leads to the well-known Richardson iteration:

$$x_{i+1} = b + (I - A)x_i = x_i + r_i.$$

Multiplication by $-A$ and adding b gives

$$b - Ax_{i+1} = b - Ax_i - Ar_i$$

or

$$r_{i+1} = (I - A)r_i = (I - A)^{i+1}r_0 = P_{i+1}(A)r_0,$$

or, in terms of the error

$$A(x - x_{i+1}) = P_{i+1}(A)A(x - x_0)$$

Table 1: Speed in Megaflops for 50 Iterations of the Iterative Techniques.

Machine	optimized ICCG Mflops	Scaled CG Mflops	Peak Performance Mflops
NEC SX-3/22 (2.9 ns)	607	1124	2750
CRAY Y-MP C90 (4.2 ns)	444	737	952
CRAY 2 (4.1 ns)	96.0	149	500
IBM 9000 Model 820	39.6	74.6	444
IBM 9121 (15 ns)	10.6	25.4	133
DEC Vax/9000 (16 ns)	9.48	17.1	125
IBM RS/6000-550 (24 ns)	18.3	21.1	81
CONVEX C3210	15.8	19.1	50
Alliant FX2800	2.18	2.98	40

$$\Rightarrow \quad x - x_{i+1} = P_{i+1}(A)(x - x_0).$$

In these expressions P_{i+1} is a (special) polynomial of degree $i+1$. Note that $P_{i+1}(0) = 1$.
Results obtained for the standard splitting can be easily generalized to other splittings, since the more general splitting $A = M - N = M - (M - A)$ can be rewritten as the standard splitting $B = I - (I - B)$ for the preconditioned matrix $B = M^{-1}A$. The theory of matrix splittings, and the analysis of the convergence of the corresponding iterative methods, is treated in depth in [90]. We will not discuss this aspect here, since it is not relevant at this stage. Instead of studying the basic iterative methods we will show how other more powerful iteration methods can be constructed as accelerated versions of the basic iteration methods. In the context of these accelarated methods, the matrix splittings become important in another way, since the matrix M of the splitting is often used to *precondition* the given system. That is, the iterative method is applied to, e.g., $M^{-1}Ax = M^{-1}b$. We will return to this later.

From now on we will assume that $x_0 = 0$. This too does not mean a loss of generality, for the situation $x_0 \neq 0$ can through a simple linear transformation $z = x - x_0$ be transformed to the system

$$Az = b - Ax_0 = \tilde{b}$$

for which obviously $z_0 = 0$.

For the simple Richardson iteration it follows that

$$x_{i+1} = r_0 + r_1 + r_2 + \cdots + r_i = \sum_{j=0}^{i} (I - A)^j r_0$$

$$\in \{r_0, Ar_0, \ldots, A^i r_0\} = K^{i+1}(A; r_0).$$

Apparently, the Richardson iteration delivers elements of increasing Krylov subspaces. Including local iteration parameters in the iteration would lead to other elements of the same Krylov subspaces. Let us write such an element still as x_{i+1}. Since $x_{i+1} \in K^{i+1}(A; r_0)$, we have that

$$x_{i+1} = Q_{i+1}(A)r_0,$$

with Q_{i+1} an arbitrary polynomial of degree $i+1$. It follows that

$$\begin{aligned} r_{i+1} &= b - Ax_{i+1} = (I - AQ_{i+1}(A))r_0 \\ &= \tilde{P}_{i+1}(A)r_0, \end{aligned} \tag{3.0a}$$

with, just as in the standard Richardson iteration, $\tilde{P}_{i+1}(0) = 1$.
The Richardson iteration can be characterized by the polynomial $P_{i+1}(A) = (I - A)^{i+1}$.

Note that one almost never computes inverses of matrices, like K^{-1}, explicitly. Instead, vectors like $\tilde{r}_i = K^{-1}b - \tilde{A}x_i = K^{-1}(b - Ax_i)$ are usually computed by solving $\tilde{r}_i$ from $K\tilde{r}_i = b - Ax_i$. The matrix K is often sparse, whereas K^{-1} usually is not, so that this procedure is much more efficient both in CPU-time and in computer memory space.

4 Towards optimal iteration methods

The natural question arises whether we can pick up a better x_{i+1} from the Krylov subspace that is generated by the basic iterative method. One would like to see the x_{i+1} for which $\|x_{i+1} - x\|_2$ is minimal.
E.g., $x_1 \in \{r_0\} \Rightarrow x_1 = \alpha_0 r_0$.

$$\|x - x_1\|_2^2 = (x - \alpha_0 r_0, x - \alpha_0 r_0) =$$

$$= (x, x) - 2\alpha_0(x, r_0) + \alpha^2(r_0, r_0).$$

Minimizing with respect to α_0 gives

$$\alpha_0 = \frac{(r_0, x)}{(r_0, r_0)},$$

and this is not practical, since x is unknown.

The above expression for α_0 suggests that with a different innerproduct the problem might be solvable: $(x, y)_A \equiv (x, Ay)$.

5 Symmetric matrices

If A is symmetric positive definite then this defines a proper innerproduct:

$$(x,y)_A = (y,x)_A,$$

$$(x,x)_A = 0 \Longleftrightarrow x = 0.$$

Now we have that

$$\|x - x_1\|_A^2 = (x - \alpha_0 r_0, x - \alpha_0 r_0)_A$$

$$\Longrightarrow \alpha_0 = \frac{(r_0,x)_A}{(r_0,r_0)_A} = \frac{(r_0,Ax)}{(r_0,Ar_0)}.$$

This looks promising and therefore we will follow that line.
In general we want $\|x_i - x\|_A$ minimal for $x_i \in K^i(A;r_0)$

$$\Rightarrow \quad x_i - x \perp_A K^i(A;r_0)$$

$$\Rightarrow \quad r_i \perp K^i(A;r_0).$$

In particular $r_1 \in \{r_0, Ar_0\}$. Assuming that $r_1 \neq \gamma r_0$ (it is easy to check that in that case r_0 is an eigenvector of A and the process could be stopped since the exact solution has then be obtained after only one iteration step), we see that $\{r_0, r_1\}$ form an orthogonal basis for $K^2(A;r_0)$.
By an induction argument we conclude that when the process does not find the exact solution at or before step i then

$$\{r_0, r_1, \ldots, r_i\}$$

is an orthogonal basis for $K^{i+1}(A;r_0)$.
This leads to the idea to construct an orthogonal basis for the Krylov subspace, a basis of which is generated implicitly by the standard iteration anyway, and then to project $x_i - x$, with respect to the A-innerproduct, onto the Krylov subspace and to determine x_i from that.

We have seen that the r_j form an orthogonal basis for $K^i(A;r_0)$, but the next remarkable property is that they satisfy a 3-term recurrence relation:

(5.0a) $$\alpha_{j+1} r_{j+1} = Ar_j - \beta_j r_j - \gamma_j r_{j-1}.$$

The proof is as follows.
$r_1 \in K^2(A;r_0) \Rightarrow \alpha_1 r_1 = Ar_0 - \beta_0 r_0$
$r_2 \in K^3(A;r_0) \Rightarrow r_2 \in \{r_0, r_1, A^2 r_0\}$
$\Rightarrow r_2 \in \{r_0, r_1, Ar_1\}$

$$\Rightarrow \quad \alpha_2 r_2 = Ar_1 - \beta_1 r_1 - \gamma_1 r_0$$

Now we use an induction argument.

$$r_{j-1} \in K^j(A;r_0) \;\Rightarrow\; Ar_{j-1} \in K^{j+1}(A;r_0)$$

$$= \{r_0, r_1, \ldots, r_j\}$$

$$\Rightarrow \quad \alpha_j r_j = Ar_{j-1} - \sum_{i=0}^{j-1} \delta_i r_i$$

Because we want the new vector r_j to be orthogonal with respect to all previous ones, the constants δ_i are determined by

$$(Ar_{j-1}, r_k) - \delta_k (r_k, r_k) = 0$$

(5.0b) $$(Ar_{j-1}, r_k) = (r_{j-1}, Ar_k)$$

(note that we have used the symmetry of A)

$$= (r_{j-1}, \alpha_{k+1} r_{k+1} + \beta_k r_k + \gamma_k r_{k-1})$$

Here we have used the induction argument for k. Because of the orthogonality it follows that $\delta_k = 0$ for $k = 0, \ldots, j-3$ and hence r_j also satisfies a 3-term recurrence relation.
The values for β_j and γ_j follow from the orthogonality of the residual vectors:

$$\beta_j = (r_j, Ar_j)/(r_j, r_j),$$

and

$$\gamma_j = (r_{j-1}, Ar_j)/(r_{j-1}, r_{j-1}).$$

The value of α_{j+1} determines the proper length of the new residual vector. From the consistency relation (3.0a) we have that each residual can be written as r_0 plus powers of A times r_0. Comparing the coefficient for r_0 in the recurrence relation (5.0a) shows that

$$\alpha_{j+1} + \beta_j + \gamma_j = 0.$$

At the end of this section we will consider the situation where the recurrence relation terminates.

We can view this 3-term recurrence relation slightly different as

$$Ar_j = \gamma_j r_{j-1} + \beta_j r_j + \alpha_{j+1} r_{j+1}$$

If we consider the r_j as being the j-th column of the matrix

$$R_i = (r_0, \ldots, r_{i-1})$$

then the recurrence relation says that A applied to a column of R_i results in the combination of three successive columns, or

$$AR_i = R_i \begin{pmatrix} \ddots & \ddots & & \\ \ddots & \ddots & \gamma_j & \\ & \ddots & \beta_j & \ddots \\ & & \alpha_{j+1} & \ddots & \ddots \\ & & & \ddots & \ddots \end{pmatrix}$$

$$+\alpha_i \begin{pmatrix} 0, 0, \ldots, r_i \end{pmatrix}$$

or

$$AR_i = R_i T_i + \alpha_i r_i e_i^T, \tag{5.0c}$$

in which T_i is an i by i tridiagonal matrix and e_i is the ith canonical vector in $\mathcal{R}^i$.

Since we are looking for a solution x_i in $K^i(A; r_0)$, that vector can be written as a combination of the basis vectors of the Krylov subspace, and hence

$$x_i = R_i y.$$

(Note that y has i components)
Further we have for the x_i, for which the error in A-norm is minimal, that

$$R_i^T(Ax_i - b) = 0$$

$$\Rightarrow \quad R_i^T A R_i y - R_i^T b = 0.$$

Using equation (5.0c) and the fact that r_i is orthogonal with respect to the columns of R_i we obtain

$$R_i^T R_i T_i y = \|r_0\|_2^2 e_1$$

Since $R_i^T R_i$ is a diagonal matrix with diagonal elements $\|r_0\|_2^2$ up to $\|r_{i-1}\|_2^2$ we find the desired solution from

$$T_i y = e_1 \quad \Rightarrow \quad y \quad \Rightarrow \quad x_i = R_i y.$$

Note that so far we have only used the fact that A is symmetric and we have assumed that the matrix T_i is not singular. We will see later that this opens the possibility for several suitable iterative methods, among which the conjugate gradients method. The Krylov subspace method that has been derived here is known as the Lanczos method for symmetric systems [47]. We will exploit the relation between the Lanczos method and the conjugate gradients method for the analysis of the convergence behaviour of the latter method.

Note that for some $j \leq n-1$ the construction of the orthogonal basis must terminate. In that case we have that $AR_{j+1} = R_{j+1}T_{j+1}$. Let y be the solution of the reduced system $T_{j+1}y = e_1$, and $x_{j+1} = R_{j+1}y$. Then it follows that $x_{j+1} = x$, i.e., we have arrived at the exact solution, since $Ax_{j+1} - b = AR_{j+1}y - b = R_{j+1}T_{j+1}y - b = R_{j+1}e_1 - b = 0$ (we have assumed that $x_0 = 0$).

5.1 THE CG-METHOD:

The Conjugate Gradients **CG** method [41] is merely a variant on the above approach, which saves storage and computational effort. For, when solving the projected equations in the above way, we see that we have to save all columns of R_i throughout the process in order to recover the current iteration vectors x_i. This can be done cheaper. If we assume that the matrix A is in addition positive definite then, because of the relation

$$R_i^T A R_i = R_i^T R_i T_i,$$

we conclude that T_i can be transformed by a rowscaling matrix $R_i^T R_i$ into a positive definite symmetric tridiagonal matrix (note that $R_i^T A R_i$ is positive definite for $y \in \mathbb{R}^{i+1}$). This implies that T_i can be LU decomposed without any pivoting:

$$T_i = L_i U_i,$$

with L_i lower unit bidiagonal and U_i upper bidiagonal. Hence

$$x_i = R_i y = R_i T_i^{-1} e_1 = (R_i U_i^{-1})(L_i^{-1} e_1)$$

We concentrate on the factors, placed between parenthesis, separately.

1.

$$L_i = \begin{pmatrix} 1 & & & \\ f_1 & 1 & & \\ & f_2 & \ddots & \\ & & \ddots & \ddots \\ & & & f_{i-1} \quad 1 \end{pmatrix}$$

With $q \equiv L_i^{-1} e_1$ we have that q can be solved from $L_i q = e_1 \Rightarrow f_{i-1} q_{i-2} + q_{i-1} = 0 \Rightarrow q_{i-1}$, in recursive manner.

2. Write $B_i \equiv R_i U_i^{-1}$, then we have that

$$R_i = B_i U_i = \begin{pmatrix} & & \vdots \\ & & \vdots \\ & & \vdots \\ & & \vdots \end{pmatrix} \times$$

$$\begin{pmatrix} d_0 & g_1 & & \\ & d_1 & \ddots & \\ & & \ddots & \\ & & & g_{i-1} \\ & & & d_{i-1} \end{pmatrix}$$

$$\Rightarrow \quad r_{i-1} = g_{i-1}(B_i)_{i-2} + d_{i-1}(B_i)_{i-1}$$

$$\Rightarrow \quad (B_i)_{i-1}.$$

Glueing these two recurrences together we obtain

$$x_i = \begin{pmatrix} & \vdots & \\ & (B_i)_{i-1} & \\ & \vdots & \end{pmatrix} \begin{pmatrix} \vdots \\ \vdots \\ q_{i-1} \end{pmatrix}$$

$$= x_{i-1} + q_{i-1}(B_i)_{i-1}$$

and this is in fact the well-known conjugate gradients method. The name stems from the property that the update vectors $(B_i)_{i-1}$, usually notated as p_{i-1}, are A-orthogonal.
Note that the positive definiteness of A is only exploited as to guarantee the flawless decomposition of the implictly generated tridiagonal matrix T_i. This suggests that the conjugate gradients method may also work for certain non positive definite systems, but then at our own risk [59]. We will later see how other ways of solving the projected system will lead to other well-known methods.

5.1.1 Computational notes

The standard (unpreconditioned) Conjugate Gradient algorithm for the solution of $Ax = b$ can be represented by the following scheme:

x_0= initial guess; $r_0 = b - Ax_0$;
$p_{-1} = 0; \beta_{-1} = 0$;
$\rho_0 = (r_0, r_0)$
for $i = 0, 1, 2, ...$
  $p_i = r_i + \beta_{i-1} p_{i-1}$;
  $q_i = Ap_i$;
  $\alpha_i = \frac{\rho_i}{(p_i, q_i)}$
  $x_{i+1} = x_i + \alpha_i p_i$;
  $r_{i+1} = r_i - \alpha_i q_i$;
  if x_{i+1} accurate enough then quit;
  $\rho_{i+1} = (r_{i+1}, r_{i+1})$;
  $\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
end;

CG is most often used in combination with a suitable splitting $A = K - R$, and then K^{-1} is called the preconditioner. We will assume that K is also positive definite.
Note first that the CG method can be derived for any choice of the innerproduct. In our derivation we have used the standard innerproduct $(x, y) = \sum x_i y_i$, but we have not used any specific property of that innerproduct. Now we make a different choice:

$$[x, y] \equiv (x, Ky).$$

It is easy to verify that $K^{-1}A$ is symmetric positive definite with respect to [,]:

$$\begin{aligned} [K^{-1}Ax, y] &= (K^{-1}Ax, Ky) = (Ax, y) \\ &= (x, Ay) = [x, K^{-1}Ay]. \end{aligned} \tag{5.1a}$$

Hence, we can follow our CG procedure for solving the preconditioned system $K^{-1}Ax = K^{-1}b$, using the new [,]-innerproduct.
Apparently, we now are minimizing

$$[x_i - x, K^{-1}A(x_i - x)] = (x_i - x, A(x_i - x)),$$

which leads to the remarkable (and known) result that for this preconditioned system we still minimize the error in A-norm, but now over a Krylov subspace generated by $K^{-1}r_0$ and $K^{-1}A$.

In the following computational scheme for preconditioned CG, for the solution of $Ax = b$ with preconditioner K^{-1}, we have replaced the [,]-innerproduct again by the familiar standard innerproduct. E.g., note that with $\tilde{r}_{i+1} = K^{-1}Ax_{i+1} - K^{-1}b$ we have that

$$\begin{aligned} \rho_{i+1} &= [\tilde{r}_{i+1}, \tilde{r}_{i+1}] \\ &= [K^{-1}r_{i+1}, K^{-1}r_{i+1}] = [r_{i+1}, K^{-2}r_{i+1}] \\ &= (r_{i+1}, K^{-1}r_{i+1}), \end{aligned}$$

and $K^{-1}r_{i+1}$ is the residual corresponding to the preconditioned system $K^{-1}Ax = K^{-1}b$.

x_0= initial guess; $r_0 = b - Ax_0$;
$p_{-1} = 0; \beta_{-1} = 0$;
Solve w_0 from $Kw_0 = r_0$;
$\rho_0 = (r_0, w_0)$
for $i = 0, 1, 2, ...$
  $p_i = w_i + \beta_{i-1} p_{i-1}$;
  $q_i = Ap_i$;
  $\alpha_i = \frac{\rho_i}{(p_i, q_i)}$
  $x_{i+1} = x_i + \alpha_i p_i$;
  $r_{i+1} = r_i - \alpha_i q_i$;
  if x_{i+1} accurate enough then quit;
  Solve w_{i+1} from $Kw_{i+1} = r_{i+1}$;
  $\rho_{i+1} = (r_{i+1}, w_{i+1})$;
  $\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
end;

Note that this formulation, which is quite popular, has the advantage that the preconditioner needs not to be splitt into two factors, and it is also avoided to backtransform solutions and residuals, as is necessary when one applies CG to $L^{-1}AL^{-1^T}y = L^{-1}b$.

The coefficients α_j and β_j, generated by the above scheme, can be used to build the matrix T_i in the following way:

$$T_i = \begin{pmatrix} \ddots & & \\ \ddots & -\frac{\beta_{j-1}}{\alpha_{j-1}} & \\ \ddots & \frac{1}{\alpha_j} + \frac{\beta_{j-1}}{\alpha_{j-1}} & \ddots \\ & -\frac{1}{\alpha_j} & \ddots \\ & & \ddots \end{pmatrix} \tag{5.1b}$$

Since $\alpha_j > 0$ and $\beta_j > 0$ we see that the above matrix is similar to the following symmetric tridiagonal

matrix:

$$\tilde{T}_i = \begin{pmatrix} \ddots & & \\ \ddots & -\frac{\sqrt{\beta_{j-1}}}{\alpha_{j-1}} & \\ \ddots & \frac{1}{\alpha_j} + \frac{\beta_{j-1}}{\alpha_{j-1}} & \ddots \\ & -\frac{\sqrt{\beta_j}}{\alpha_j} & \ddots \\ & & \ddots \end{pmatrix}.$$

The eigenvalues of the leading i^{th} order minor of this matrix are the Ritz values of the preconditioned matrix $K^{-1}A$ with respect to the i-dimensional Krylov subspace spanned by the first i residual vectors. The Ritz values approximate the (extremal) eigenvalues of the preconditioned matrix increasingly well. These approximations can be used to get an impression of the relevant eigenvalues. They can also be used to construct upperbounds for the error in the delivered approximation with respect to the solution [45, 40]. According to the results in [80] the eigenvalue information can also be used in order to understand or explain delays in the convergence behaviour.

5.1.2 The convergence of Conjugate Gradients

The conjugate gradient method (here with $K = I$) constructs in the i^{th} iteration step an x_i, which can be written as

$$x_i - x = P_i(A)(x_0 - x) \quad \text{(cf. (3.0}a\text{))},$$

such that $\|x_i - x\|_A$ is minimal over all polynomials P_i of degree i, with $P_i(0) = 1$.
Let us denote the eigenvalues and the orthonormalized eigenvectors of A by λ_j, z_j. We write $r_0 = \sum_j \gamma_j z_j$. It follows that

$$r_i = P_i(A)r_0 = \sum_j \gamma_j P_i(\lambda_j) z_j$$

and hence

$$\|x_i - x\|_A^2 = \sum_j \frac{\gamma_j^2}{\lambda_j} P_i^2(\lambda_j).$$

Note that only those λ_j play a role in this process for which $\gamma_j = 0$. In particular, if A happens to be semidefinite, i.e., there is a $\lambda = 0$, then this is no problem for the minimization process as long as the corresponding coefficient γ is zero as well. The situation where γ is small, due to rounding errors, is discussed in [45].
Upperbounds on the error (in A-norm) are obtained by observing that

$$\|x_i - x\|_A^2 = \sum_j \frac{\gamma_j^2}{\lambda_j} P_i^2(\lambda_j) \leq \sum_j \frac{\gamma_j^2}{\lambda_j} Q_i^2(\lambda_j)$$

$$\text{(5.1c)} \qquad \leq \max_{\lambda_j} Q_i^2(\lambda_j) \sum_j \frac{\gamma_j^2}{\lambda_j},$$

for any arbitrary polynomial Q_i of degree i with $Q_i(0) = 1$, where the maximum is taken, of course, only over those λ for which the corresponding $\gamma \neq 0$. When P_i has zeros at all the different λ_j then $r_i = 0$. The conjugate gradients method tries to spread the zeros in such a way that $P_i(\lambda_j)$ is small in a weighted sense, i.e., $\|x_i - x\|_A$ is as small as possible.

We get suitable upperbounds by selecting appropriate polynomials for Q_i. A very well-known upperbound arises by taking for Q_i the i^{th} degree Chebychev polynomial transformed to the interval $[\lambda_{min}, \lambda_{max}]$ and scaled such that its value in 0 is equal to 1.

$$\|x_i - x\|_A^2 \leq \sum \frac{\gamma_j^2}{\lambda_j} T_i^2(\lambda_j)$$

$$\text{(5.1d)} \qquad \leq \max_{\lambda_1, \lambda_n} |T_i^2(\lambda_j)| \, \|x_0 - x\|_A^2,$$

and

$$\text{(5.1e)} \qquad |T_i(\lambda_j)| \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i$$

The purpose of preconditioning is to reduce the condition number κ.

As we have seen the conjugate gradients algorithm is just an efficient implementation of the Lanczos algorithm. The eigenvalues of the implicitly generated tridiagonal matrix T_i are the Ritz values of A with respect to the current Krylov subspace. It is known from Lanczos theory that these Ritz values converge towards the eigenvalues of A and that in general the extremal eigenvalues of A are first well approximated [46, 58, 63]. Furthermore, the speed of convergence depends on how well these eigenvalues are separated from the others (gap ratio) [63]. This helps us to understand the so-called superlinear convergence behaviour of the conjugate gradient method (as well as other Krylov subspace methods). It can be shown that as soon as one of the extremal eigenvalues is modestly well approximated by a Ritz value, the pocedure converges from then on as a process in which this eigenvalue is absent, i.e., a process with a reduced condition number. Note that superlinear convergence behaviour in this connection is used to indicate linear convergence with a factor that is gradually decreased during the process as more and more of the extremal eigenvalues are sufficiently well approximated (for details on this see [80]).

5.1.3 Further references

A more formal presentation of CG, as well as many theoretical properties, can be found in the textbook by Hackbusch [39]. A shorter presentation is given

in [35]. An overview of papers, published in the first 25 years of existence of the method, is given in [34]. Vector processing and parallel computing aspects are discussed in [23] and [57].

5.2 MINRES and SYMMLQ:

When A is not positive definite, but still symmetric, then we can construct an orthogonal basis for the Krylov subspace, as we have seen before. We write the recurrence relations slightly different as

$$AR_i = R_{i+1}\bar{T}_i,$$

with

$$\bar{T}_i = \begin{pmatrix} \ddots & \ddots & & & \\ \ddots & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \\ & & & \ddots & \end{pmatrix} \begin{matrix} \leftarrow\ i\ \rightarrow \\ \\ \\ \\ \\ \end{matrix} \quad \begin{matrix} \uparrow \\ \\ i+1 \\ \\ \downarrow \end{matrix}$$

In this case we have the problem that $(\ ,\ )_A$ does not define an innerproduct. However we can still try to minimize the residual. We look for an

$$x_i \in \{r_0, Ar_0, \ldots, A^{i-1}r_0\}, \quad x_i = R_i\bar{y}$$

$$\|Ax_i - b\|_2 = \|AR_i\bar{y} - b\|_2$$

$$= \|R_{i+1}\bar{T}_i y - b\|_2$$

Now we exploit the fact that $R_{i+1}D_{i+1}^{-1}$, with $D_{i+1} = diag(\|r_0\|_2, \|r_1\|_2, ..., \|r_i\|_2)$, is an orthonormal transformation with respect to the current Krylov subspace:

$$\|Ax_i - b\|_2 = \|D_{i+1}\bar{T}_i y - \|r_0\|_2 e_1\|_2$$

and this final expression can simply be seen as a minimum norm least squares problem.
The element in the $(i+1, i)$ position of $\bar{T}_i$ can be transformed to zero by a simple Givens rotation and the resulting upper bidiagonal system (the other subdiagonal elements being removed in previous iteration steps) can simply be solved, which leads to the so-called MINRES method [60].
Another possibility is to solve the system $T_i y = \|r_0\|_2 e_1$, as in the CG method (T_i is the upper i by i part of $\bar{T}_i$. Other than in CG we cannot rely on the existence of a Choleski decomposition (since A is not positive definite). An alternative is then to decompose T_i by an LQ-decomposition. This again leads to simple recurrences and the resulting method is known as SYMMLQ [60].

5.3 Parallelism and data locality in preconditioned CG:

For successful application of CG one needs that the matrix A is symmetric positive definite. In other short recurrence methods, other properties of A may be desirable, but we will not exploit these properties explicitly in the discussion on parallel aspects.
Most often, the conjugate gradients method is used in combination with some kind of preconditioning. This means that the matrix A can be thought of to be multiplied with some suitable approximation K^{-1} for A^{-1}. Usually, K is constructed as an approximation of A, such that systems like $Ky = z$ are much more easy to solve as $Ax = b$. Unfortunately, a popular class of preconditioners, based upon incomplete factorization of A, do not lend themselves very much for parallel implementation. We will discuss some approaches to obtain more parallelism in the preconditioner in section 9.1. At the moment we will assume that the preconditioner is chosen such that the parallelism in solving $Ky = z$ is comparable with the parallelism in computing Ap, for given p.
For CG it is also required that the preconditioner K is symmetric positive definite. This aspect will play a role in our discussions since it shows how some properties of the preconditioner can be used sometimes to our advantage for an efficient implementation.

The scheme for preconditioned CG is given in Section 5.1.1. Note that in that scheme the updating of x and r can only start after the completion of the innerproduct required for α_i. Therefore, this innerproduct is a so-called synchronization point: all computation has to wait for completion of this operation. One can try to avoid such synchronization points as much as possible, or to formulate CG in such a way that synchronization points can be taken together. We will see such approaches further on.
Since on a distributed memory machine communication is required to assemble the innerproduct, it would be nice if we could proceed with other useful computation while the communication takes place. However, as we see from our CG scheme, there is no possibility to overlap this communication time with useful computation. The same observation can be made for the updating of p, which can only take place after the completion of the innerproduct for β_i. Apart from the computation of Ap and the computations with K, we need to load 7 vectors for 10 vector floating point operations. This means that for this part of the computation only 10/7 floating point operation can be carried out per memory reference in average.

Several authors ([11, 52, 53]) have attempted to improve this ratio, and to reduce the number of synchronization points. In our formulation of CG there are two such synchronization points, namely the com-

putation of both innerproducts.
Meurant [52] (see also [68]) has proposed a variant in which there is only one synchronization point, however at the cost of a possibly reduced numerical stability, and one additional innerproduct. In this scheme the ratio between computations and memory references is about 2.
We show here another variant, proposed by Chronopoulos and Gear [11].

x_0= initial guess; $r_0 = b - Ax_0$;
$q_{-1} = p_{-1} = 0; \beta_{-1} = 0$;
Solve w_0 from $Kw_0 = r_0$;
$s_0 = Aw_0$;
$\rho_0 = (r_0, w_0); \mu_0 = (s_0, w_0)$;
$\alpha_0 = \rho_0/\mu_0$;
for $i = 0, 1, 2, ...$
$p_i = w_i + \beta_{i-1} p_{i-1}$;
$q_i = s_i + \beta_{i-1} q_{i-1}$;
$x_{i+1} = x_i + \alpha_i p_i$;
$r_{i+1} = r_i - \alpha_i q_i$;
if x_{i+1} accurate enough then quit;
Solve w_{i+1} from $Kw_{i+1} = r_{i+1}$;
$s_{i+1} = Aw_{i+1}$;
$\rho_{i+1} = (r_{i+1}, w_{i+1})$;
$\mu_{i+1} = (s_{i+1}, w_{i+1})$;
$\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
$\alpha_{i+1} = \frac{\rho_{i+1}}{\mu_{i+1} - \rho_{i+1}\beta_i/\alpha_i}$;
end i;

In this scheme all vectors need only be loaded once per pass of the loop, which leads to a better exploitation of the data (improved data locality). However, the price is that we need $2n$ flops more per iteration step. Chronopoulos and Gear [11] claim stability, based upon their numerical experiments.
Instead of 2 synchronization points, as in the standard version of CG, we have now only one synchronization point, as the next loop can only be started when the innerproducts at the end of the previous loop have been assembled. Another slight advantage is that these innerproducts can be computed in parallel.
Chronopoulos and Gear [11] propose to further improve the data locality and parallelism in CG by combining s successive steps. Their algorithm is based upon the following property of CG. The residual vectors $r_0, ..., r_i$ form an orthogonal basis (assuming exact arithmetic) for the Krylov subspace spanned by $r_0, Ar_0, ..., A^{i-1}r_0$. When arrived at r_j, the vectors $r_0, r_1, ..., r_j, Ar_j, ..., A^{i-j-1}r_j$ also form a basis for this subspace. Hence, we may combine s successive steps by generating $r_j, Ar_j, ..., A^{s-1}r_j$ first, and then do the orthogonalization and the updating of the current solution with this blockwise extended subspace. This approach leads to a slight increase in flops in comparison with s successive steps of the standard CG, and also one additional matrix vector product is required per s steps.
The main drawback in this approach seems to be the potential numerical instability. Depending on the spectral properties of A, the set $r_j, ..., A^{s-1}r_j$ may tend to converge to a vector in the direction of a dominating eigenvector, or, in other words, may tend to dependence for increasing values of s. The authors claim to have seen successful completion of this approach, with no serious stability problems, for small values of s. Nevertheless, it seems that s-step CG, because of these problems, has a bad reputation (see also [69]). However, a similar approach, suggested by Chronopoulos and Kim [12] for other processes such as GMRES, seems to be more promising. Several authors have pursued this research direction, and we will come back to this in section 7.3.

We consider still another variant of CG, in which there is possibility to overlap all of the communication time with useful computations. This variant is just a reorganized version of the original CG scheme, and is therefore precisely as stable. The key trick in this approach is to delay the updating of the solution vector by one iteration step.
Another advantage over the previous scheme is that no additional operations are required.
It is assumed that the preconditioner K can be written as $K = (LL^T)^{-1}$. Furthermore, it is assumed that the preconditioner has a block structure, corresponding to the gridblocks assigned to the processors, so that communication (if necessary) can be overlapped with computation.

x_0= initial guess; $r_0 = b - Ax_0$;
$p_{-1} = 0; \beta_{-1} = 0; \alpha_{-1} = 0$;
$s = L^{-1} r_0$;
$\rho_0 = (s, s)$;
for $i = 0, 1, 2, ...$
$w_i = L^{-T} s$; (0)
$p_i = w_i + \beta_{i-1} p_{i-1}$; (1)
$q_i = Ap_i$; (2)
$\gamma = (p_i, q_i)$; (3)
$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$; (4)
$\alpha_i = \frac{\rho_i}{\gamma}$; (5)
$r_{i+1} = r_i - \alpha_i q_i$; (6)
$s = L^{-1} r_{i+1}$; (7)
$\rho_{i+1} = (s, s)$; (8)
if r_{i+1} *small enough* then (9)
$x_{i+1} = x_i + \alpha_i p_i$
quit;
$\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
end i;

Now we discuss how this scheme may lead to an efficient parallel scheme, and how local memory (vector registers, cache, ...) can be exploited.

1. All computing intensive operations can be carried out in parallel. Only for the operations (2), (3), (7), (8), (9), and (0), communication between processors is required. We have assumed that the communication in (2), (7), and (0) can be largely overlapped with computation.

2. The communication required for the assembly of the innerproduct in (3) can be overlapped with the update for x (which could have been done in the previous iteration step).

3. The assembly of the innerproduct in (8) can be overlapped with the computation in (0). Also step (9) usually requires information such as the norm of the residual, which can be overlapped with (0).

4. Steps (1), (2), and (3) can be combined: the computation of a segment of p_i can be followed immediately by the computation of a segment of q_i (2), and this can be followed by the computation of a part of the innerproduct in (3). This saves on load operations for segments of p_i and q_i.

5. Depending on the structure of L, the computation of segments of r_{i+1} in (6) can be followed by operations in (7), which can be followed by the computation of parts of the innerproduct in (8), and the computation of the norm of r_{i+1}, required for (9).

6. The computation of β_i can be done as soon as the computation in (8) has been completed. At that moment, the computation for (1) can be started if the requested parts of w_i have been completed in (0).

7. If no preconditioner is used, then $w_i = r_i$, and steps (7) and (0) have to be skipped. Step (8) has to be replaced by $\rho_i = (r_{i+1}, r_{i+1})$. Now we need useful computation in order to overlap the communication for this innerproduct. To this end, one might split the computation in (4) per processor in two parts. The first of these parts are computed in paralell in overlap with (3), while the parallel computation of the other parts is used in order to overlap the communication for the computation of ρ_i.

5.4 Parallel performance of CG:

Some realistic 3D computational fluid dynamics simulation problems, as well as other problems, lead to the necessity to solve linear systems $Ax = b$ with a matrix of very large order, billions of unknowns, say. If not of very special structure, such systems are not likely to be solved by direct elimination methods.
For such very large (sparse) systems we will have to exploit parallelism in combination with suitable solution techniques, like for instance iterative solution methods.
From a parallel point of view CG mimics very well parallel performance properties of a variety of iterative methods such as Bi-CG, CGS, BiCGSTAB, QMR, and others.
In this section we study the performance of CG on parallel distributed memory systems and we report on some supporting experiments on actual existing machines. Guided by our experiments we will discuss the suitability of CG for Massively Parallel Processing systems.

All computational intensive elements in preconditioned CG (updates, innerproducts, and matrix vector operations) are trivially parallelizable for shared memory machines [23], except possibly for the preconditioning step: *Solve* w_{i+1} *from* $Kw_{i+1} = r_{i+1}$. For the latter operation parallelism depends very much on the choice for K. In this section we restrict ourselves to block Jacobi preconditioning, where the blocks have been chosen so that each processor can handle one block independently of the others. For other preconditioners that allow some degree of parallelism see [23].
For a distributed memory machine at least some of the steps require communication between processors: the accumulation of innerproducts and the computation of Ap_i (depending on the non-zero structure of A and the distribution of the non-zero elements over the processors). We consider in some more detail the situation where A is a block-tridiagonal matrix of order N, and we assume that all blocks are of order $\sqrt{N}$:

$$A = \begin{pmatrix} A_1 & D_1 & & \\ D_1 & A_2 & D_2 & \\ & D_2 & \ddots & \ddots \\ & & \ddots & \end{pmatrix},$$

in which the D_i are diagonal matrices, and the A_i are tridiagonal matrices. Such systems occur quite frequently in finite difference approximations in 2 space dimensions. Our discussion can easily be adapted to 3 space dimensions.

5.4.1 Processor configuration and data distribution

For simplicity we will assume that the processors are connected as a 2D grid with $p \times p = P$ processors.
The data have been distributed in a straight forward manner over the processor memories and we have not attempted to fully exploit the underlying grid structure for the given type of matrix in order to reduce communication as much as possible. In fact it will turn out that in our case the communication for the

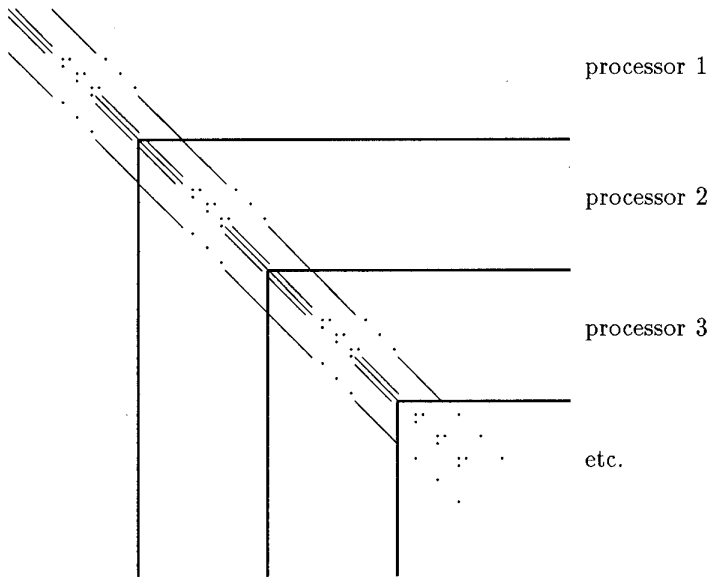


Fig.1: Distribution of A over the processors..

matrix vector product plays only a minor role for matrix systems of large size .
Because of symmetry only the 3 non-zero diagonals in the upper triangular part of A need to be stored, and we have chosen to store successive parts of length N/P of each diagonal in consecutive neighbouring processors. In Figure 1 we see which part of A is represented by the data in the memory of a given processor.

The blocks for block Jacobi are chosen to be the diagonal blocks that are available on each processor, and the various vectors (r_i, p_i, etc.) have been distributed likewise, i.e. each processor holds a section of length N/P of these vectors in its local memory.

5.4.2 Required Communication

matrix vector product It is easily seen for a $2D$ processor grid (as well as for many other configurations, including hypercube and pipeline), that the matrix vector product can be completed with only neighbour-neighbour communication. This means that the communication costs do not increase for increasing values of p. If one follows a domain decomposition way of approach, in which the finite difference discretization grid is subdivided into p by p subgrids (p in x-direction and p in y-direction), then the communication costs are smaller than the computational costs by a factor of $\mathcal{O}(\frac{\sqrt{N}}{p})$.
In [17] much attention is given to this sparse matrix vector product and it is shown that the time for communication can almost completely be overlapped with computational work. Therefore, with adequate coding the matrix vector products do not necessarily lead to serious communication problems, even not for relatively small-sized problems.
On a MEIKO SP1 (located at Utrecht University, this machine has only 4 processors) we have observed, for $N = 90000$, a speed-up by a factor of 1.85 for two processors, and of 1.96 when overlap possibilities are exploited. In both cases we expect, by extrapolating our timing results, a factor of 2 for very large N. According to a naive interpretation of Amdahl's law we might expect a severe degradation in performance for more than two processors. However, if we increase the size of the problem for increasing numbers of processors then the local communication time for the matrix product does not increase so that it does not pose limits on the performance when we increase the value of p.

vector update In our case these operations do not require any communication and we should expect linear speed up when increasing the number of processors P.

inner product For the innerproduct we need global communication for assembly and we need global communication for the distribution of the assembled innerproduct over the processors. For a $p \times p$ processor grid these communication costs are proportional with p. This means that for a constant length of the vectorparts per processor, these communicationcosts will dominate for values of p large enough. This is quite unlike the situation for the matrix vector product and as we will see it may be a severely limiting factor in achieving high speed-ups in a massively parallel environment.

For the MEIKO SP1 we have done some experiments in order to determine the costs of inter proces-

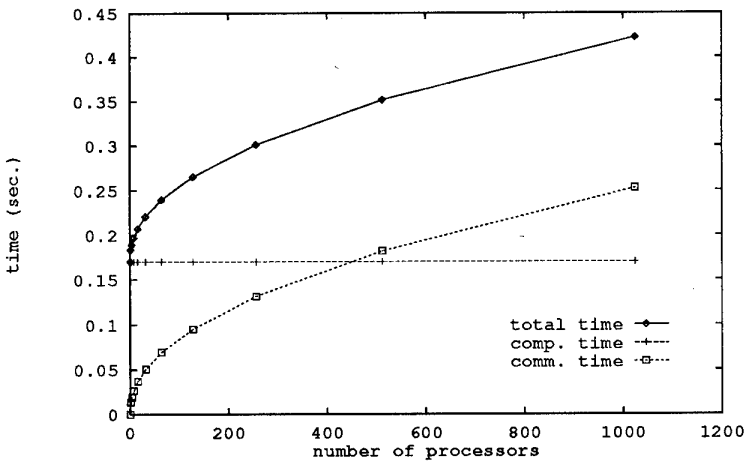


Fig.2: Modelled timings for 1 iteration with CG.

sor communication and for communication. Assuming that the costs for communication (for the innerproducts) grow linearly with the length of the path of communication we have modelled the wall-clock time for 1 iteration with CG, for matrices of order $90000P$, as in Figure 2. Note that we have increased the size of the linear system linearly with the number of processors, which seems realistically since with larger computers one aims to solve larger systems. The value 90000 has been chosen since this is more or less the size of the part of the system that can be kept in the local memory of one processor of our MEIKO machine.

From this Figure we learn that for P slightly larger than 400 the communication costs may be expected to dominate, and eventually they will lead to very low speed-ups (even for systems for which the size is as large as the total memory permits). We also see, that if overlap of communication and computation is possible, then potentially the communication can be hidden for values of P less than 400, but this demands for a reformulation of the CG algorithm. Of course, these expectations are based on a model, but we have also carried out similar experiments on the 512 processor Parsytec GCel-3/512 of the University of Amsterdam [15]. In particular we have observed on that machine that the communication time for the innerproduct increases like $\sqrt{P}$, which just explains the behaviour of our model for the MEIKO-type of architecture.

Our experiments and our modelling approach clearly show that even a method like CG, which might be anticipated to be highly parallel, may suffer severely from the communication overhead due to the required innerproducts. Our study indicates that if we want reasonable speed-up in a massively parallel environment then the local memories should also be much larger when the number of processors is increased in order to accomodate for systems large enough to compensate for the increased global communication costs.

Another approach would be to modify the CG method such that the innerproducts take relatively less time. Many of such approaches have been studied recently. A quite popular approach is to reformulate CG such that the required innerproducts can be computed simultaneously, so that the communication overhead is reduced (the communication required for 2 simultaneous innerproducts is almost the same as for 1 innerproduct). An extreme form of this approach is to reformulate CG so that a number of basis vectors for the search space are computed without making them orthogonal. The orthogonalization is then carried out afterwards, and in this approach most of the communication can be combined. The numerical stability of these approaches is still a point of concern. For an overview and further references see [6]. For some other iterative methods, such as GMRES, this approach can be quite effective as is shown in [17].

Still another approach is to try to more useful computational work per iteration step, so that the communication for the two innerproducts takes relatively less time. One way to do this is to use polynomial preconditioning, i.e., the preconditioner consists of a number of matrix vector products with the matrix A. This may work well in situations where the matrix vector product requires only little (local) communication. Another way is to apply domain decomposition: the given domain is split into P, say, subdomains with estimated values for the solutions on the interfaces. Then all the subproblems are solved independently and in parallel. This way of approximating the solution may be viewed as a preconditioning step in an

iterative method. In this way we do more computational work per communication step. Unfortunately, depending on the problem and on the way of decoupling the subdomains one may need a larger number of iteration steps for larger values of P, which may then, of course, detoriate the overall efficiency of the domain decomposition approach. For more information on this approach we also refer to references given in [6].

If a given architecture permits the overlap of communication with computation, then we may try to reformulate CG in order to create possibilities for overlap. For the (extrapolated) MEIKO this may help for values of P up to about 400. For larger P we will see communication dominating anyhow, but the adverse effects can be lessened. A stable reformulation of CG which has this effect has been described in [20].

6 Unsymmetric problems

There are essentially three different ways to solve unsymmetric linear systems, while maintaining some kind of orthogonality between the residuals:

1. Solve the normal equations $A^TAx = A^Tb$ with conjugate gradients
2. Make all the residuals explicitly orthogonal in order to have an orthogonal basis for the Krylov subspace
3. Construct a basis for the Krylov subspace by a 3-term biorthogonality relation

6.1 Normal Equations:

The first solution seems rather obvious. However, it has severe disadvantages because of the squaring of the condition number. This has as effects that the solution is more susceptible to errors in the right-hand side and that the rate of convergence of the CG procedure is much slower as for a comparable symmetric system with a matrix with the same condition number as A. Moreover, the amount of work per iteration step, necessary for the matrix vector product, is doubled.
There have been made several proposals to improve the numerical stability of this rather robust approach. The most well-known is by Paige and Saunders [61] and is based upon applying the Lanczos method to the auxiliary system

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

Clever execution of this delivers in fact the factors L and U of the LU-decomposition of the tridiagonal matrix that would have been delivered when carrying out the Lanczos procedure with A^TA.
Another approach to improve the numerical stability of this normal equations approach is suggested by Björck and Elfving [8]. They observed that the matrix A^TA is used in the construction of the iteration coefficients through an innerproduct like (p, A^TAp). They simply suggest to replace such an innerproduct by (Ap, Ap).
The use of conjugate gradients in a least squares context, as well as a theoretical comparison with SIRT type methods, is discussed in [81] and [82].

An interesting variant of LSQR is the so-called Craig's method [61]. The easiest way to think of this method is to apply Conjugate Gradients to the system $A^TAx = A^Tb$, with the following choice for the innerproduct

$$[x, y] \equiv (x, (A^TA)^{-1}y),$$

which defines a proper innerproduct if A is of full rank (see section 5.1.1).
First note that the two innnerproducts in CG (as in section 5.1.1 can be computed without inverting A^TA:

$$[p_i, A^TAp_i] = (p_i, p_i),$$

and, assuming that $b \in R(A)$ so that $Ax = b$ has a unique solution x (since A has full rank):

$$\begin{aligned} [r_i, r_i] &= [A^T(Ax_i - b), A^T(Ax_i - b] \\ &= [A^TA(x_i - x), A^T(Ax_i - b)] \\ &= (x_i - x, A^T(Ax_i - b)) \\ &= (Ax_i - b, Ax_i - b) \end{aligned} \tag{6.1a}$$

Apparantly, we are with CG minimizing

$$\begin{aligned} [x_i - x, A^TA(x_i - x)] &= (x_i - x, x_i - x) \\ &= \|x_i - x\|_2^2 \end{aligned} \tag{6.1b}$$

that is, in this approach the Euclidean norm of the error is minimized. Note, however, that the rate of convergence of Craig's method is determined by the condition number of A^TA, so that this method is only attractive if one has a good preconditioner for A^TA.

6.2 FOM and GMRES:

The second approach is to form explicitly an orthonormal basis for the Krylov subspace. Since A is not symmetric we no longer have a 3-term recurrence relation for that purpose and the new basis vector has to be made explicitly orthonormal with respect to all the previous vectors:

$$v_1 = \frac{1}{\|r_0\|_2} r_0,$$

$$h_{i+1,i}v_{i+1} = Av_i - \sum_{j=1}^{i} h_{j,i}v_j.$$

As in the symmetric case this can be exploited in two different ways. The orthogonality relation can either be written as

(6.2a) $$AV_i = V_i H_i + h_{i+1,i} v_{i+1} e_i^T,$$

after which the projected system, with a Hessenberg matrix instead of a tridiagonal matrix as in the symmetric case, can be solved (nonsymmetric CG, GENCG, FOM, Arnoldi's method), or it can be written as

(6.2b) $$AV_i = V_{i+1} \bar{H}_i,$$

after which the projected system, with an $i+1$ by i upper Hessenberg matrix can be solved as a least squares system. In GMRES [72] this is done by the QR method using Givens rotations in order to annihilate the subdiagonal elements in the upper Hessenberg matrix $\bar{H}_i$.
The first approach (based upon (6.2a)) is similar to the conjugate gradient approach (or SYMMLQ), the second approach (based upon (6.2b)) is similar the conjugate directions method (or MINRES).

In order to avoid excessive storage requirements and computational costs for the orthogonalization, GMRES is usually restarted after each m iteration steps. This algorithm is referred to as GMRES(m). Below we give a scheme for GMRES(m) which may be suitable to develop a computer code. It solves $Ax = b$, with a given preconditioner K.

x_0 is an initial guess;
for $j = 1, 2, ...$
  Solve r from $Kr = b - Ax_0$;
  $v_1 = r/\|r\|_2$;
  $s := \|r\|_2 e_1$;
  for $i = 1, 2, ..., m$
    Solve w from $Kw = Av_i$;
  orthogonalization of w
  against v's, by modified
  Gram-Schmidt process
    for $k = 1, ..., i$
      $h_{k,i} = (w, v_k)$;
      $w = w - h_{k,i} v_k$;
    end k;
    $h_{i+1,i} = \|w\|_2$;
    $v_{i+1} = w/h_{i+1,i}$;
    apply $J_1, ..., J_{i-1}$ on $(h_{1,i}, ..., h_{i+1,i})$;
  construct J_i, acting on i–th
  and $(i+1)$–st component
  of $h_{.,i}$, such that $(i+1)$–st
  component of $J_i h_{.,i}$ is 0;
    $s := J_i s$;
    if $s(i+1)$ is small enough then:
    (UPDATE($\tilde{x}, i$); quit);
  end i;
  UPDATE($\tilde{x}, m$);
end j;

In this scheme UPDATE($\tilde{x}, i$) replaces the following computations:

Compute y as the solution of $Hy = \tilde{s}$,
in which the upper i by i triangular
part of H has $h_{i,j}$ as its elements
(in least squares sense if H is singular),
$\tilde{s}$ represents the first i components of s;
$\tilde{x} = x_0 + y_1 * v_1 + y_2 v_2 + ... + y_i v_i$;
s_{i+1} equals $\|b - A\tilde{x}\|_2$;
if this component is not small enough
then $x_0 = \tilde{x}$;
else quit;

Another scheme for GMRES, based upon Householder orthogonalization instead of modified Gram-Schmidt has been proposed in [92]. For certain applications it seems attractive to invest in additional computational work in turn for improved numerical properties: the better orthogonality might save iteration steps.

The eigenvalues of H_i are the Ritz values of $K^{-1}A$ with respect to the Krylov subspace spanned by v_1, ..., v_i. They approximate eigenvalues of $K^{-1}A$ increasingly well for increasing dimension i.

There is an interesting and simple relation between the two different Krylov subspace projection approaches (6.2a), the "FOM" approach, and (6.2b), the "GMRES" approach. The projected system matrix $\bar{H}_i$ is transformed by a Givens rotations to an upper triangular matrix (with last row equal to zero). So, in fact, the major difference between FOM and GMRES is that in FOM the last $((i+1)$-th row is simply discarded, while in GMRES this row is rotated to a zero vector. Let us characterize the Givens rotation, acting on rows i and $i+1$, in order to zero the element in position $(i+1, i)$, by the sine s_i and the cosine c_i. Let us further denote the residuals for FOM with an superscript F and those for GMRES with superscript G. Then the above observations lead to the following results for FOM and GMRES (for details see [72] and [9]).

1. The reduction for successive GMRES residuals is given by

   (6.2c) $$\frac{\|r_k^G\|_2}{\|r_{k-1}^G\|_2} = |s_k| \ .$$

   ([72]: p. 862, Proposition 1)

2. If $c_k \neq 0$ then the FOM and the GMRES residuals are related by

   (6.2d) $$\|r_k^G\|_2 = |c_k| \, \|r_k^F\|_2$$

   ([9]: theorem 5.1)

From these relations we see that when GMRES has a local significant reduction in the norm of the residual (i.e., s_k is small), then FOM gives about the same result as GMRES (since $c_k^2 = 1 - s_k^2$). On the other hand when FOM has a break-down ($c_k = 0$), then the GMRES does not lead to an improvement in the same iteration step.
Because of these relations we can link the convergence behaviour of GMRES with the convergence of Ritz values (the eigenvalues of the "FOM" part of the upper Hessenberg matrix). This has been exploited in [88], for the analysis and explanation of local effects in the convergence behaviour of GMRES.

In order to limit the required amount of memory storage and the amount of flops per iteration step, one often restarts the GMRES method after each m steps. This restarted version is commonly referred to as GMRES(m), while the not-restarted method often is called Full GMRES.

There are various different implementations of FOM and GMRES. Among those equivalent with GMRES are: Orthomin [91], Orthodir [44], Axelsson's method [3] and GENCR [27]. These methods are often more expensive than GMRES per iteration step. Orthomin seems to be still popular, since this variant can be easily truncated (Orthomin(s)), in contrast to GMRES. The truncated or restarted versions of these algorithms are not necessarily mathematically equivalent.
Methods that are mathematically equivalent with FOM are: Orthores [44] and GENCG [13, 93]. In these methods the approximate solutions are constructed such that they lead to orthogonal residuals (which form a basis for the Krylov subspace; analogously to the CG method). A good overview of all these methods and their relations is given in [71].

6.3 Rank-one updates for the Matrix Splitting:

Iterative methods can be derived from a splitting of the matrix, and we have used the very simple splitting $A = I - R$, with $R = I - A$, in order to derive the projection type methods. In [26] it is suggested to update the matrix splitting with information obtained in the iteration process. We will give the flavour of this method here since it turns out that it has an interesting relation with GMRES. This relation is exploited in [89] for the construction of new classes of GMRES-like methods, that can be used as cheap alternatives for the increasingly expensive full GMRES method.

Assume that the matrix splitting in the k-th iteration step is given by $A = H_k^{-1} - R_k$, then we obtain the iteration formula

$$x_k = x_{k-1} + H_k r_{k-1} \quad \text{with} \quad r_k = b - Ax_k.$$

The idea is now to construct H_k by a suitable rank-one update to H_{k-1}:

$$H_k = H_{k-1} + u_{k-1} v_{k-1}^T,$$

which leads to

$$\text{(6.3a)} \quad x_k = x_{k-1} + (H_{k-1} + u_{k-1} v_{k-1}^T) r_{k-1}$$

or

$$\begin{aligned} r_k &= r_{k-1} - A(H_{k-1} + u_{k-1} v_{k-1}^T) r_{k-1} \\ \text{(6.3b)} \quad &= (I - AH_{k-1}) r_{k-1} - A u_{k-1} v_{k-1}^T r_{k-1} \\ &= (I - AH_{k-1}) r_{k-1} - \mu_{k-1} A u_{k-1}. \end{aligned}$$

The optimal choice for the update would have been to select u_{k-1} such that

$$\mu_{k-1} A u_{k-1} = (I - AH_{k-1}) r_{k-1},$$

or

$$\mu_{k-1} u_{k-1} = A^{-1}(I - AH_{k-1}) r_{k-1}.$$

However, A^{-1} is unknown and the best approximation we have for it is H_{k-1}. This leads to the choice

$$\text{(6.3c)} \quad \bar{u}_{k-1} = H_{k-1}(I - AH_{k-1}) r_{k-1}.$$

The constant μ_{k-1} is chosen such that $\|r_k\|_2$ is minimal as a function of μ_{k-1}. This leads to

$$\mu_{k-1} = \frac{1}{\|A\bar{u}_{k-1}\|_2^2} (A\bar{u}_{k-1})^T (I - AH_{k-1}) r_{k-1}.$$

Since v_{k-1} has to be chosen such that $\mu_{k-1} = v_{k-1}^T r_{k-1}$, we have the following obvious choice for it

$$\text{(6.3d)} \quad \bar{v}_{k-1} = \frac{1}{\|A\bar{u}_{k-1}\|_2^2} (I - AH_{k-1})^T A\bar{u}_{k-1}$$

(note that from the minimization property we have that $r_k \perp A\bar{u}_{k-1}$).

In principle the implementation of the method is quite straight forward, but note that the computation of r_{k-1}, $\bar{u}_{k-1}$ and $\bar{v}_{k-1}$ costs 4 matrix vector multiplications with A (and also some with H_{k-1}). This would make the method too expensive for being of practical interest. Also the updated splitting is most likely a dense matrix if we carry out the updates explicitly.
We will now show, still following the lines set forth in [26], that there are orthogonality properties, following from the minimization step, by which the method can be implemented much more efficiently.
We define

1. $c_k = \frac{1}{\|A\bar{u}_k\|_2} A\bar{u}_k$ (note that $r_{k+1} \perp c_k$)
2. $E_k = I - AH_k$

From (6.3b) we have that $r_k = E_k r_{k-1}$, and from (6.3c):

$$A\bar{u}_k = AH_k E_k r_k = \alpha_k c_k$$

or

$$\text{(6.3e)} \qquad \begin{aligned} c_k &= \frac{1}{\alpha_k}(I - E_k)E_k r_k \\ &= \frac{1}{\alpha_k} E_k (I - E_k) r_k \end{aligned}$$

Furthermore (on behalf of (6.3c)):

$$\begin{aligned} E_k &= I - AH_k \\ &= I - AH_{k-1} - A\bar{u}_{k-1}\bar{v}_{k-1}^T \\ &= I - AH_{k-1} - A\bar{u}_{k-1}(A\bar{u}_{k-1})^T \\ &\quad (I - AH_{k-1})\frac{1}{\|A\bar{u}_k\|_2^2} \\ &= (I - c_{k-1}c_{k-1}^T)E_{k-1} \\ \text{(6.3f)} \qquad &= \prod_{i=0}^{k-1}(I - c_i c_i^T)E_0. \end{aligned}$$

We see that the operator E_k has the following effect on a vector. The vector is multiplied by E_0 and then orthogonalized with respect to c_0, ..., c_{k-1}. Now we have from (6.3e) that

$$c_k = \frac{1}{\alpha_k} E_k y_k,$$

and hence

$$\text{(6.3g)} \qquad c_k \perp c_0, ..., c_{k-1}.$$

A consequence from (6.3g) is that

$$\prod_{j=0}^{k-1}(I - c_j c_j^T) = I - \sum_{j=0}^{k-1} c_j c_j^T = I - P_{k-1}$$

and therefore

$$\text{(6.3h)} \qquad P_k = \sum_{j=0}^{k} c_j c_j^T.$$

The actual implementation is based on the above properties. Given r_k we compute r_{k+1} as follows (and we update x_k in the corresponding way):

$$r_{k+1} = E_{k+1} r_k.$$

With $\xi^{(0)} = E_0 r_k$ we first compute (with the c_j from previous steps):

$$E_k r_k = \xi^{(k)} \equiv (I - \sum_{j=0}^{k-1} c_j c_j^T)\xi^{(0)} = \prod_{j=0}^{k-1}(I - c_j c_j^T)\xi^{(0)}.$$

The expression with $\sum$ leads to a Gram-Schmidt formulation, the expression with $\prod$ leads to the Modified Gram-Schmidt variant.
The computed updates $-c_j^T \xi^{(0)} c_j$ for r_{k+1} correspond to updates

$$c_j^T \xi^{(0)} A^{-1} c_j = c_j^T \xi^{(0)} u_j / \|Au_j\|_2$$

for x_{j+1}. These updates are in the scheme, given below, represented by η.

From (6.3c) we know that

$$\bar{u}_k = H_k E_k r_k = H_k \xi^{(k)}.$$

Now we have to make $A\bar{u}_k \sim c_k$ orthogonal w.r.t. c_0, ..., c_{k-1}, and to update $\bar{u}_k$ accordingly. Once we have done that we can do the final update step to make H_{k+1}, and we can update both x_k and r_k by the corrections following from including c_k. The orthogonalization step can be carried out easily as follows. Define $c_k^{(k)} \equiv \alpha_k c_k = AH_k E_k r_k = (I - E_k)E_k r_k$ (see (6.3e)) $= (I - E_0 + P_{k-1}E_0)\xi^{(k)}$ (see (6.3f)) $= AH_0\xi^{(k)} + P_{k-1}(I - AH_0)\xi^{(k)} = c_k^{(0)} + P_{k-1}\xi^{(k)} - P_{k-1}c_k^{(0)}$. Note that the second term vanishes since $\xi^{(k)} \perp c_0, ..., c_{k-1}$.

The resulting scheme for the k-th iteration step becomes:

1. $\xi^{(0)} = (I - AH_0)r_k$; $\eta^{(0)} = H_0 r_k$;
   for $i = 0, ..., k-1$ do
   $\alpha_i = c_i^T \xi^{(i)}$;
   $\xi^{(i+1)} = \xi^{(i)} - \alpha_i c_i$; $\eta^{(i+1)} = \eta^{(i)} + \alpha_i u_i$;
2. $u_k^{(0)} = H_0 \xi^{(k)}$; $c_k^{(0)} = Au_k^{(0)}$;
   for $i = 0, ..., k-1$ do
   $\beta_i = -c_i^T c_k^{(i)}$; $c_k^{(i+1)} = c_k^{(i)} + \beta_i c_i$;
   $u_k^{(i+1)} = u_k^{(i)} + \beta_i u_i$;
   $c_k = c_k^{(k)} / \|c_k^{(k)}\|_2$; $u_k = u_k^{(k)} / \|c_k^{(k)}\|_2$;
3. $x_{k+1} = x_k + \eta^{(k)} + u_k c_k^T \xi^{(k)}$;
   $r_{k+1} = (I - c_k c_k^T)\xi^{(k)}$;

Remarks

1. The above scheme is a Modified Gram Schmidt variant, given in [89], of the original scheme in [26].
2. If we keep H_0 fixed, i.e., $H_0 = I$, then the method is not scaling invariant (the results for $\rho Ax = \rho b$ depend on ρ). In [89] a scaling invariant method is suggested.
3. Note that in the above implementation we have 'only' two matrix vector products per iteration step. In [89] it is shown that in many cases we may also expect comparable converge as for GMRES in half the number of iteration steps.

4. A different choice for $\bar{u}_{k-1}$ does not change the formulas for $\bar{v}_{k-1}$ and E_{k-1}. For each different choice we can derive similar schemes as the one above.

5. From (6.3b) we have

$$r_k = r_{k-1} - AH_{k-1}r_{k-1} - \mu_{k-1}Au_{k-1}.$$

In view of the previous remark we might also make the different choice $\bar{u}_{k-1} = H_{k-1}r_{k-1}$. With this choice, we obtain a variant which is algebraically identical to GMRES (for a proof of this see [89]). This GMRES variant is obtained by the following changes in the previous scheme: Take $H_0 = 0$ (note that in this case we have that $E_{k-1}r_{k-1} = r_{k-1}$, and hence we we may skip part 1 of the above algorithm), and set $\xi^{(k)} = r_k$, $\eta^{(k)} = 0$. In step 2 start with $u_k^{(0)} = \xi^{(k)}$.
The result is a different formulation of GMRES in which we can obtain explicit formulas for the updated preconditioner (i.e., the inverse of A is approximated increasingly well): The update for H_k is $\bar{u}_k c_k^T E_k$ and the sum of these updates gives an approximation for A^{-1}.

6. Also in this GMRES-variant we are still free to select u_k a little bit different. Remember that the leading factor H_{k-1} in (6.3c) was introduced as an approximation for the actually desired A^{-1}. With $\bar{u}_{k-1} = A^{-1}r_{k-1}$, we would have that $r_k = E_{k-1}r_{k-1} - \mu_{k-1}r_{k-1} = 0$ for the minimizing μ_{k-1}. We could take other approximations for the inverse (with respect to the given residual r_{k-1}), e.g., the result vector y obtained by a few steps GMRES applied to $Ay = r_{k-1}$. This leads to the so-called GMRESR family of nested methods (for details see [89]). See also section 6.4. A similar algorithm, named FGMRES, has been derived independently by Saad [70]. In FGMRES the search directions are preconditioned, whereas in GMRESR the residuals are preconditioned. This gives GMRESR direct control over the reduction in norm of the residual. As a result GMRESR can be made robust while FGMRES may suffer from break-down. A further disadvantage of the FGMRES formulation is that this method cannot be truncated, or selectively orthogonalized, as GMRESR can.
In [4] a generalized conjugate gradient method is proposed, a variant of which produces in exact arithmetic identical results as the proper variant of GMRESR, though at higher computational costs and with a classical Gram–Schmidt orthogonalization process instead of the modified process as in GMRESR.

6.4 GMRESR and GMRES$\star$:

By Van der Vorst and Vuik [89] it has been shown how the GMRES-method can be combined (or rather preconditioned) with other iterative schemes. The iteration steps of GMRES (or GCR) are called outer iteration steps, while the iteration steps of the preconditioning iterative method are referred to as inner iterations. The combined method is called GMRES$\star$, where $\star$ stands for any given iterative scheme; in the case of GMRES as the inner iteration method, the combined scheme is called GMRESR[89].
Similar schemes have been proposed recently. In FGMRES[70] the update directions for the approximate solution are preconditioned, whereas in GMRES$\star$ the residuals are preconditioned. The latter approach offers more control over the reduction in the residual, in particular break-down situations can be easily detected and remedied.
In exact arithmetic GMRES$\star$ is very close to the Generalized Conjugate Gradient method[4]; GMRES$\star$, however, leads to a more efficient computational scheme.

The GMRES$\star$ algorithm can be described by the following computational scheme:

x_0 is an initial guess; $r_0 = b - Ax_0$;
for $i = 0, 1, 2, 3, ...$
  Let $z^{(m)}$ be the approximate solution
  of $Az = r_i$, obtained after m steps of
  an iterative method.
  $c = Az^{(m)}$ (often available from the
    iteration method)
  for $k = 0, ..., i-1$
    $\alpha = (c_k, c)$
    $c = c - \alpha c_k$
    $z^{(m)} = z^{(m)} - \alpha u_k$
  $c_i = c/\|c\|_2$; $u_i = z^{(m)}/\|c\|_2$
  $x_{i+1} = x_i + (c_i, r_i)u_i$
  $r_{i+1} = r_i - (c_i, r_i)c_i$
  if x_{i+1} is accurate enough then quit
end

A sufficient condition to avoid break-down in this method ($\|c\|_2 = 0$) is that the norm of the residual at the end of an inner iteration is smaller than the right-hand residual: $\|Az^{(m)} - r_i\|_2 < \|r_i\|_2$. This can easily be controlled during the inner iteration process. If stagnation occurs, i.e. no progress at all is made in the inner iteration, then it is suggested by Van der Vorst and Vuik[89] to do one (or more) steps of the LSQR method, which guarantees a reduction (but this reduction is often only small).

The idea behind this combined iteration scheme is that we explore parts of high-dimensional Krylov subspaces, hopefully localizing the same approximate solution that full GMRES would find over the entire subspace, but now at much lower computational

costs. The alternatives for the inner iteration could be either one cycle of GMRES(m), since then we have also locally an optimal method, or some other iteration scheme, like for instance Bi-CGSTAB. As has been shown by Van der Vorst[87] there are a number of different situations where we may expect stagnation or slow convergence for GMRES(m). In such cases it does not seem wise to use this method.

On the other hand it may also seem questionable whether a method like Bi-CGSTAB should lead to success in the inner iteration. This method does not satisfy a useful global minimization property and large part of its effectiveness comes from the underlying Bi-CG algorithm, which is based on bi-orthogonality relations. This means that for each outer iteration the inner iteration process has to build a bi-orthogonality relation again. It has been shown for the related Conjugate Gradients method that the orthogonality relations are determined largely by the distribution of the weights at the lower end of the spectrum and on the isolated eigenvalues at the upper end of the spectrum[82]. By the nature of these kind of Krylov processes the largest eigenvalues and their corresponding eigenvector components quickly do enter the process after each restart, and hence it may be expected that much of the work is lost in rediscovering the same eigenvector components in the error over and over again, whereas these components may already be so small that further reduction in those directions in the outer iteration is waste of time, since it hardly contributes to a smaller norm of the residual.
This heuristic way of reasoning may explain in part our rather disappointing experiences with Bi-CGSTAB as the inner iteration process for GMRES$\star$.

De Sturler and Fokkema[19] propose to prevent the outer search directions explicitly from being reinvestigated again in the inner process. This is done by keeping the Krylov subspace that is build in the inner iteration orthogonal with respect to the Krylov basis vectors generated in the outer iteration. The procedure works as follows.
In the outer iteration process the vectors c_0, ..., c_{i-1} build an orthogonal basis for the Krylov subspace. Let C_i be the n by i matrix with columns c_0, ..., c_{i-1}. Then the inner iteration process at outer iteration i is carried out with the operator A_i instead of A, and A_i is defined as

$$A_i = (I - C_i C_i^T)A. \tag{6.4a}$$

It is easily verified that $A_i z \perp c_0, ..., c_{i-1}$ for all z, so that the inner iteration process takes place in a subspace orthogonal to these vectors. The additional costs, per iteration of the inner iteration process, are i inner products and i vector updates. In order to save on these costs, one should realize that it is not necessary to orthogonalize with respect to all previous c-vectors, and that "less effective" directions may be dropped, or combined with others. De Sturler and Fokkema[19] suggestions are made for such strategies. Of course, these strategies in cases where we see too little residual reducing effect in the inner iteration process in comparison with the outer iterations of GMRES$\star$.

6.5 Bi-conjugate Gradients:

The third class of methods arises from the attempt to construct a suitable set of basis vectors for the Krylov subspace by a three-term recurrence relation as in (5.0a):

$$\alpha_{j+1} r_{j+1} = A r_j - \beta_j r_j - \gamma_j r_{j-1}. \tag{6.5a}$$

As we have seen in the proof for the orthogonality of such a set of vectors (see section 4), we needed the symmetry of the matrix A. In the nonsymmetric case we need instead of (5.0b) that

$$(Ar_{j-1}, r_k) = (r_{j-1}, A^T r_k) = 0 \quad \text{for} \quad k < j-2.$$

By similar arguments as in the proof for (5.0a) we conclude that (6.5a) can be used to generate a basis r_0,...,r_{i-1} for $K^i(A; r_0)$, such that $r_j \perp K^{j-1}(A^T; r_0)$, or even more general,

$$r_j \perp K^{j-1}(A^T; s_0),$$

since there is no explicit need to generate the Krylov subspace for A^T with r_0 as the starting vector.
If we let the basis vectors s_j for $K^i(A^T; s_0)$ satisfy the same recurrence relation as the vectors r_j, i.e., with identical recurrence coefficients, then we see that

$$(r_k, s_j) = 0 \quad \text{for} \quad k \neq j$$

(by a simple symmetry argument).
Hence, the sets $\{r_j\}$ and $\{s_j\}$ satisfy a *biorthogonality* relation. Now we can proceed in a similar way as in the symmetric case:

$$AR_i = R_i T_i + \alpha_i r_i e_i^T, \tag{6.5b}$$

but now we use the matrix $S_i = [s_0, s_1, ..., s_{i-1}]$ for the projection of the system

$$S_i^T (Ax_i - b) = 0,$$

or

$$S_i^T A R_i y - S_i^T b = 0.$$

Using (6.5b) we find that y_i satisfies

$$S_i^T R_i T_i y = (r_0, s_0) e_1.$$

Since $S_i^T R_i$ is a diagonal matrix with diagonal elements (r_j, s_j), we find, if all these diagonal elements are nonzero, that

$$T_i y = e_1 \quad \Rightarrow \quad x_i = R_i y.$$

This method is known as the Bi-Lanczos method [47]. We see that we are in problems when a diagonal element of $S_i^T R_i$ becomes (near) zero, this is referred to in litterature as a serious (near) breakdown. The way to get around this difficulty is the so-called Look-ahead strategy, which comes down to taking a number of successive basis vectors for the Krylov subspace together and to make them blockwise bi-orthogonal. This has been worked out in detail in [62] and [30], [31], [32].
Another way to avoid break-down is to restart as soon as a diagonal element gets small. Of course, this strategy looks surprisingly simple, but one should realise that at a restart the Krylov subspace, that has been built up so far, is thrown away, which destroys possibilities for faster (i.e., superlinear) convergence.

As has been shown for Conjugate Gradients, the LU decomposition of the tridiagonal system can be updated from iteration to iteration and this leads to a recursive update of the solution vector. This avoids to save all intermediate r and s vectors. This variant of Bi-Lanczos is usually called Bi-Conjugate Gradients, or shortly Bi-CG [28].
Of course one can in general not be certain that an LU decomposition (without pivoting) of the tridiagonal matrix T_i exists, and this may lead also to breakdown of the Bi-CG algorithm. Note that this breakdown can be avoided in the Bi-Lanczos formulation of this iterative solution scheme. It is also avoided in the QMR approach (see Section 5.4.2).
Note that for symmetric matrices Bi-Lanczos generates the same solution as Lanczos, provided that $s_0 = r_0$, and under the same condition Bi-CG delivers the same iterands as CG for positive definite matrices. However, the Bi-orthogonal variants do so at the cost of two matrix vector operations per iteration step.

It is difficult to make a fair comparison between GMRES and Bi-CG. GMRES really minimizes a residual, but at the cost of increasing work for keeping all residuals orthogonal and increasing demands for memory space. Bi-CG does not minimize a residual, but often it has a comparable fast convergence as GMRES, at the cost of twice the amount of matrix vector products per iteration step. However, the generation of the basis vectors is relatively cheap and the memory requirements are limited and modest. Several variants of Bi-CG have been proposed which increase the effectiveness of this class of methods in certain circumstances. These variants will be discussed in coming subsections.

The following scheme may be used for a computer implementation of the Bi-CG method. In the scheme the equation $Ax = b$ is solved with a suitable preconditioner K.

x_0 is an initial guess; $r_0 = b - Ax_0$;
solve w_0 from $Kw_0 = r_0$;
$\tilde{r}_0$ is an arbitrary vector such that $(w_0, \tilde{r}_0) \neq 0$,
usually one chooses $\tilde{r}_0 = r_0$ or $\tilde{r}_0 = w_0$;
solve $\tilde{w}_0$ from $K^T \tilde{w}_0 = \tilde{r}_0$;
$p_{-1} = \tilde{p}_{-1} = 0; \beta_{-1} = 0; \rho_0 = (w_0, \tilde{r}_0)$;
for $i = 0, 1, 2, \ldots$.
  $p_i = w_i + \beta_{i-1} p_{i-1}$;
  $\tilde{p}_i = \tilde{w}_i + \beta_{i-1} \tilde{p}_{i-1}$;
  $z_i = Ap_i$;
  $\alpha_i = \frac{\rho_i}{(\tilde{p}_i, z_i)}$;
  $r_{i+1} = r_i - \alpha_i z_i$;
  $\tilde{r}_{i+1} = \tilde{r}_i - \alpha_i A^T \tilde{p}_i$;
  solve w_{i+1} from $Kw_{i+1} = r_{i+1}$;
  solve $\tilde{w}_{i+1}$ from $K^T \tilde{w}_{i+1} = \tilde{r}_{i+1}$;
  $\rho_{i+1} = (\tilde{r}_{i+1}, w_{i+1})$;
  $x_{i+1} = x_i + \alpha_i p_i$;
  if x_{i+1} is accurate enough then quit;
  $\beta_i = \frac{\rho_{i+1}}{\rho_i}$
end

As with conjugate gradients, the coefficients α_j and β_j, $j = 0, \ldots, i-1$ build the matrix T_i, as given in formula (5.1b). This matrix is, for BiCG, in general not similar to a symmetric matrix. Its eigenvalues can be viewed as Petrov-Galerkin approximations, with respect to the spaces $\{\tilde{r}_j\}$ and $\{r_j\}$, of eigenvalues of A. For increasing values of i they tend to converge to eigenvalues of A. The convergence patterns, however, may be much more complicated and irregular as in the symmetric case.

6.5.1 Another derivation of Bi-CG

An alternative way to derive Bi-CG comes from considering the following symmetric linear system:

$$\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} \hat{x} \\ x \end{pmatrix} = \begin{pmatrix} b \\ \hat{b} \end{pmatrix}, \quad \text{or} \quad B\tilde{x} = \tilde{b},$$

for some suitable vector $\hat{b}$.
If we select $\hat{b} = 0$ and apply the CG-scheme to this system, then we obtain LSQR again. However, if we select $\hat{b} \neq 0$ and apply the CG scheme with the preconditioner

$$\begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix},$$

in the way as is shown in section 4.4.1, then we obtain right away the unpreconditioned Bi-CG scheme for the system $Ax = b$. Note that the CG-scheme can be applied since $K^{-1}B$ is symmetric (but not positive definite) with respect to the bilinear form

$$[p, q] \equiv (p, Kq),$$

which is not a proper innerproduct. Hence, this formulation clearly reveals the two principal weaknesses

of Bi-CG (i.e., the causes for break-down situations). Note that if we restrict ourselves to vectors

$$p = \begin{pmatrix} p_1 \\ p_1 \end{pmatrix},$$

then $[p, q]$ defines a proper innerproduct. This situation arises for the Krylov subspace that is created for B and $\tilde{b}$ if $A = A^T$ and $\hat{b} = b$. If, in addition, A is positive definite then $K^{-1}B$ is positive definite symmetric with respect to the generated Krylov subspace, and we obtain the CG-scheme (as expected). More generally, the choice

$$K = \begin{pmatrix} 0 & K_1 \\ K_1^T & 0 \end{pmatrix},$$

where K_1 is a suitable preconditioner for A, leads to the preconditioned version of the Bi-CG scheme, as given in section 5.4.

The above presentation of Bi-CG was inspired by a closely related presentation of BI-CG in [42]. The latter paper gives a rather untractable reference for the choice of the system $B\tilde{x} = \tilde{b}$ and the preconditioner

$$\begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix}$$

to [43].

6.5.2 QMR

The QMR method [32] relates to Bi-CG in a similar way as MINRES relates to CG. For stability reasons the basis vectors r_j and $\tilde{r}_j$ are normalized (as is usual in the underlying Bi-Lanczos algorithm, see [94]), which leads to other coefficients in the 3-term recursion formulas.
If we group the residual vectors r_j, for $j = 0, ..., i-1$ in a matrix R_i, then we can write the recurrence relations as

$$AR_i = R_{i+1}\bar{T}_i,$$

with

$$\bar{T}_i = \begin{pmatrix} \ddots & \ddots & & \\ \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ & & \ddots & \ddots \\ & & & \ddots \end{pmatrix} \begin{matrix} \\ \uparrow \\ \\ i+1 \\ \\ \downarrow \end{matrix} \quad \begin{matrix} \leftarrow\ i\ \rightarrow \end{matrix}$$

Similar as for MINRES we would like to construct the x_i, with

$$x_i \in \{r_0, Ar_0, \ldots, A^{i-1}r_0\}, \quad x_i = R_i\bar{y}$$

for which

$$\begin{aligned} \|Ax_i - b\|_2 &= \|AR_i\bar{y} - b\|_2 \\ &= \|R_{i+1}\bar{T}_i y - b\|_2 \\ &= \|R_{i+1}D_{i+1}^{-1}\{D_{i+1}\bar{T}_i y - \|r_0\|_2 e_1\}\|_2 \end{aligned}$$

is minimal. However, in this case that would be quite an amount of work since the columns of R_{i+1} are not necessarily orthogonal. Freund and Nachtigal [32] suggest to solve the miniminum norm least squares problem

$$\min_{y \in R^i} \|D_{i+1}\bar{T}_i y - \|r_0\|_2 e_1\|_2. \tag{6.5c}$$

This leads to the simplest form of the QMR method. A more general form arises if the least squares problem (6.5c) is replaced by a weighted least squares problem. No strategies are yet known for optimal weights, however.
In [32] the QMR method is carried out on top of a look-ahead variant of the bi-orthogonal Lanczos method, which makes the method more robust. Experiments suggest that QMR has a much smoother convergence behaviour than Bi-CG, but it is not essentially faster than Bi-CG.

6.5.3 CGS

For the bi-conjugate gradient residual vectors it is well-known that they can be written as $r_j = P_j(A)r_0$ and $\hat{r}_j = P_j(A^T)\hat{r}_0$, and because of the bi-orthogonality relation we have that

$$\begin{aligned} (r_j, \hat{r}_i) &= (P_j(A)r_0, P_i(A^T)\hat{r}_0) \\ &= (P_i(A)P_j(A)r_0, \hat{r}_0) = 0, \end{aligned}$$

for $i < j$.
The iteration parameters for bi-conjugate gradients are computed from innerproducts like the above. Sonneveld observed that we can also construct the vectors $\tilde{r}_j = P_j^2(A)r_0$, using only the latter form of the innerproduct for recovering the bi-conjugate gradients parameters (which implicitly define the polynomial P_j). By doing so, it can be avoided that the vectors $\hat{r}_j$ have to be formed, nor is there any multiplication with the matrix A^T.
The resulting CGS [79] method works in general very well for many unsymmetric linear problems. It converges often much faster than BI-CG (about twice as fast in some cases) and does not have the disadvantage of having to store extra vectors like in GMRES. These three methods have been compared in many studies (see, e.g., [67, 10, 65, 55]).
However, CGS usually shows a very irregular convergence behaviour. This behaviour can even lead to cancellation and a spoiled solution [86]. See also section 6.5.4.

The following scheme carries out the CGS process for the solution of $Ax = b$, with a given preconditioner K:

x_0 is an initial guess; $r_0 = b - Ax_0$;
$\tilde{r}_0$ is an arbitrary vector, such that
$(r_0, \tilde{r}_0) \neq 0$,
e.g., $\tilde{r}_0 = r_0$; $\rho_0 = (r_0, \tilde{r}_0)$;
$\beta_{-1} = \rho_0$; $p_{-1} = q_0 = 0$;
for $i = 0, 1, 2, ...$
  $u_i = r_i + \beta_{i-1} q_i$;
  $p_i = u_i + \beta_{i-1}(q_i + \beta_{i-1} p_{i-1})$;
  solve $\hat{p}$ from $K\hat{p} = p_i$;
  $\hat{v} = A\hat{p}$;
  $\alpha_i = \frac{\rho_i}{(\tilde{r}_0, \hat{v})}$;
  $q_{i+1} = u_i - \alpha_i \hat{v}$;
  solve $\hat{u}$ from $K\hat{u} = u_i + q_{i+1}$
  $x_{i+1} = x_i + \alpha_i \hat{u}$;
  if x_{i+1} is accurate enough then quit;
  $r_{i+1} = r_i - \alpha_i A\hat{u}$;
  $\rho_{i+1} = (\tilde{r}_0, r_{i+1})$;
  if $\rho_{i+1} = 0$ then method fails to converge !;
  $\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
end

In exact arithmetic, the α_j and β_j are the same constants as those generated by BiCG. Therefore, they can be used to compute the Petrov-Galerkin approximations for eigenvalues of A.

6.5.4 Effects of irregular convergence

By very irregular convergence we refer to the situation where successive residual vectors in the iterative process differ in orders of magnitude in norm, and some of these residuals may be even much bigger in norm than the starting residual. We will try to give an impression why this is a point of concern, even if eventually the (updated) residual satisfies a given tolerance. For more details we refer to Sleijpen et al[75, 77].

We will say that an algorithm is *accurate* for a certain problem if the *updated residual* r_j and the *true residual* $b - Ax_j$ are of comparable size for the j's of interest.

The best we can hope for is that for each j the error in the residual is only the result of applying A to the update w_{j+1} for x_j in finite precision arithmetic:

$$r_{j+1} = r_j - Aw_{j+1} - \Delta_A w_{j+1} \tag{6.5d}$$

if

$$x_{j+1} = x_j + w_{j+1}, \tag{6.5e}$$

for each j, where Δ_A is an $n \times n$ matrix for which $|\Delta_A| \preceq n_A \bar{\xi} |A|$: n_A is the maximum number of nonzero matrix entries per row of A, $|B| \equiv (|b_{ij}|)$ if $B = (b_{ij})$, $\bar{\xi}$ is the relative machine precision, the inequality $\preceq$ refers to element-wise $\leq$. In the Bi-CG type methods that we consider, we compute explicitly the update Aw_j for the residual r_j from the update w_j for the approximation x_j by matrix multiplication: for this part, (6.5d) describes well the local deviations caused by evaluation errors.

In the "ideal" case (i.e. situation (6.5d) whenever we update the approximation) we have that

$$\begin{aligned} r_k - (b - Ax_k) &= \sum_{j=1}^{k} \Delta_A w_j \\ &= \sum_{j=1}^{k} \Delta_A (e_{j-1} - e_j), \end{aligned} \tag{6.5f}$$

where the perturbation matrix Δ_A may depend on j and e_j is the approximation error in the jth approximation: $e_j \equiv x - x_j$. Hence,

$$\begin{aligned} &|\|r_k\| - \|b - Ax_k\|| \leq \\ &2k\, n_A \bar{\xi}\, \|\|A\|\| \sum_{j=0}^{k} \|e_j\| \leq \\ &2\,\Gamma \bar{\xi} \sum_{j=0}^{k} \|r_j\|, \end{aligned} \tag{6.5g}$$

where $\Gamma \equiv n_A \, \|\|A\|\| \, \|A^{-1}\|$.

Except for the factor Γ, the last upper-bound appears to be rather sharp. We see that approximations with large approximation errors may ultimately lead to an inaccurate result. Such large local approximation errors are typical for CGS, and Van der Vorst[86] describes an example of the resulting numerical inaccuracy is given. If there are a number of approximations with comparable large approximation errors, then their multiplicity may replace the factor k, otherwise it will be only the largest approximation error that makes up virtually the bound for the deviation.

Example. Figure 3 illustrates nicely the loss of accuracy as described above; for other examples, cf. [86]. The convergence history of the updated residuals (the 'circles': oo) and the true residuals (the solid curve: — –) of CGS is given for the matrix SHERMAN4 from the Harwell-Boeing set of test matrices. Here, as in other figures, the norm of the residuals, on log-scale, is plotted (along the vertical axis) against the number of matrix-vector multiplications (along the horizontal axis). The dotted curve $(\cdots)$ represents the estimated inaccuracy: $2\bar{\xi} \sum_{j \leq i} \|r_j\|$ (here with $\Gamma = 1$; cf. (6.5g)).

We will discuss two approaches that lead to a smoother convergence.

— Approaches to obtain the smoothing effect by adding a few lines to existing codes leave the speed of convergence essentially unchanged. One of these approaches leads to optimal accurate approximations

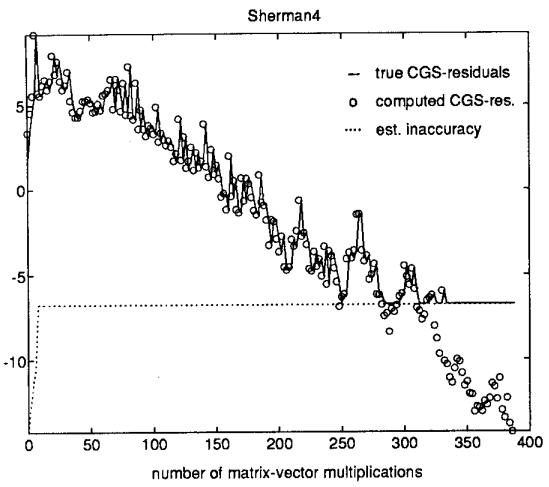


Fig.3: Convergence plot CGS for the true residuals and the updated residuals.

[76] and will be discussed in Section 7. For other ones, we refer to the literature (e.g., [95]).
— In the next section, we concentrate on techniques that really change the convergence: they smooth down and speed up the convergence, and lead to more accurate approximations, all at the same time.

6.5.5 Bi-CGSTAB

Bi-CGSTAB [86] is based on the following observation. Instead of squaring the Bi-CG polynomial, we can construct other iteration methods, by which x_i are generated so that $r_i = \tilde{P}_i(A)P_i(A)r_0$ with other i^{th} degree polynomials $\tilde{P}$. An obvious possibility is to take for $\tilde{P}_j$ a polynomial of the form

$$(6.5h)\; Q_i(x) = (1-\omega_1 x)(1-\omega_2 x)...(1-\omega_i x),$$

and to select suitable constants ω_j. This expression leads to an almost trivial recurrence relation for the Q_i.
In Bi-CGSTAB ω_j in the j^{th} iteration step is chosen as to minimize r_j, with respect to ω_j, for residuals that can be written as $r_j = Q_j(A)P_j(A)r_0$.
The preconditioned Bi-CGSTAB algorithm for solving the linear system $Ax = b$, with preconditioning K reads as follows:

x_0 is an initial guess; $r_0 = b - Ax_0$;
$\bar{r}_0$ is an arbitrary vector, such that
  $(\bar{r}_0, r_0) \neq 0$, e.g., $\bar{r}_0 = r_0$;
$\rho_{-1} = \alpha_{-1} = \omega_{-1} = 1$;
$v_{-1} = p_{-1} = 0$;
for $i = 0, 1, 2, ...$
  $\rho_i = (\bar{r}_0, r_i)$; $\beta_{i-1} = (\rho_i/\rho_{i-1})(\alpha_{i-1}/\omega_{i-1})$;
  $p_i = r_i + \beta_{i-1}(p_{i-1} - \omega_{i-1}v_{i-1})$;
  Solve $\hat{p}$ from $K\hat{p} = p_i$;
  $v_i = A\hat{p}$;
  $\alpha_i = \rho_i/(\bar{r}_0, v_i)$;
  $s = r_i - \alpha_i v_i$;
  if $||s||$ small enough then
    $x_{i+1} = x_i + \alpha_i\hat{p}$; quit;
  Solve z from $Kz = s$;
  $t = Az$;
  $\omega_i = (t, s)/(t, t)$;
  $x_{i+1} = x_i + \alpha_i\hat{p} + \omega_i z$;
  if x_{i+1} is accurate enough then quit;
  $r_{i+1} = s - \omega_i t$;
end

The matrix K in this scheme represents the preconditioning matrix and the way of preconditioning [86]. The above scheme in fact carries out the Bi-CGSTAB procedure for the explicitly postconditioned linear system

$$AK^{-1}y = b,$$

but the vectors y_i and the residual have been back-transformed to the vectors x_i and r_i corresponding to the original system $Ax = b$. Compared to CGS two extra innerproducts need to be calculated.
In exact arithmetic, the α_j and β_j have the same values as those generated by Bi-CG and CGS. Hence, they can be used to extract eigenvalue approximations for the eigenvalues of A (see Bi-CG).
Bi-CGSTAB can be viewed as the product of Bi-CG and GMRES(1). Of course, other product methods can be formulated as well. Gutknecht [38] has proposed BiCGSTAB2, which is constructed as the product of Bi-CG and GMRES(2).

6.5.6 Derivation of Bi-CGSTAB

The polynomial P_i and related polynomials are implicitly defined by the Bi-CG scheme.

Bi-CG:

x_0 is an initial guess; $r_0 = b - Ax_0$;
$\hat{r}_0$ is an arbitrary vector, such that
  $(\hat{r}_0, r_0) \neq 0$, e.g., $\hat{r}_0 = r_0$;
$\rho_0 = 1$;
$\hat{p}_0 = p_0 = 0$;
for $i = 1, 2, 3, ...$
  $\rho_i = (\hat{r}_{i-1}, r_{i-1})$; $\beta_i = (\rho_i/\rho_{i-1})$;
  $p_i = r_{i-1} + \beta_i p_{i-1}$;
  $\hat{p}_i = \hat{r}_{i-1} + \beta_i\hat{p}_{i-1}$;
  $v_i = Ap_i$;
  $\alpha_i = \rho_i/(\hat{p}_i, v_i)$;
  $x_i = x_{i-1} + \alpha_i p_i$;
  if x_i is accurate enough then quit;
  $r_i = r_{i-1} - \alpha_i v_i$;
  $\hat{r}_i = \hat{r}_{i-1} - \alpha_i A^T\hat{p}_i$;
end

From this scheme it is straight forward to show that $r_i = P_i(A)r_0$ and $p_{i+1} = T_i(A)r_0$, in which $P_i(A)$ and $T_i(A)$ are i-th degree polynomials in A. The Bi-CG scheme then defines the relations between these

polynomials:

$$T_i(A)r_0 = (P_i(A) + \beta_{i+1}T_{i-1}(A))r_0,$$

and

$$P_i(A)r_0 = (P_{i-1}(A) - \alpha_i AT_{i-1}(A))r_0.$$

In the Bi-CGSTAB scheme we wish to have recurrence relations for

$$r_i = Q_i(A)P_i(A)r_0.$$

With Q_i as in (6.5h) and the Bi-CG relation for the factor P_i and T_i, it then follows that

$$Q_i(A)P_i(A)r_0 =$$

$$(1 - \omega_i A)Q_{i-1}(A)(P_{i-1}(A) - \alpha_i AT_{i-1}(A))r_0$$

$$= \{Q_{i-1}(A)P_{i-1}(A) - \alpha_i AQ_{i-1}(A)T_{i-1}(A)\}r_0$$

$$-\omega_i A\{(Q_{i-1}(A)P_{i-1}(A) - \alpha_i AQ_{i-1}(A)T_{i-1}(A))\}r_0.$$

Clearly, we also need a relation for the product $Q_i(A)T_i(A)r_0$. This can also be obtained from the Bi-CG relations:

$$Q_i(A)T_i(A)r_0 = Q_i(A)(P_i(A) + \beta_{i+1}T_{i-1}(A))r_0$$

$$= Q_i(A)P_i(A)r_0 + \beta_{i+1}(1 - \omega_i A)Q_{i-1}(A)T_{i-1}(A)r_0$$

$$= Q_i(A)P_i(A)r_0 + \beta_{i+1}Q_{i-1}(A)T_{i-1}(A)r_0$$

$$-\beta_{i+1}\omega_i AQ_{i-1}(A)T_{i-1}(A)r_0.$$

Finally we have to recover the Bi-CG constants ρ_i, β_i, and α_i by innerproducts in terms of the new vectors that we now have generated.
E.g., β_i can be computed as follows. First we compute

$$\tilde{\rho}_{i+1} = (\hat{r}_0, Q_i(A)P_i(A)r_0) = (Q_i(A^T)\hat{r}_0, P_i(A)r_0).$$

By construction $P_i(A)r_0$ is orthogonal with respect to all vectors $U_{i-1}(A^T)\hat{r}_0$, where U_{i-1} is an arbitrary polynomial of degree $i-1$ at most. This means that we have to consider only the highest order term of $Q_i(A^T)$ when computing $\tilde{\rho}_{i+1}$. This term is given by $(-1)^i\omega_1\omega_2\cdots\omega_i(A^T)^i$. We actually wish to compute

$$\rho_{i+1} = (P_i(A^T)\hat{r}_0, P_i(A)r_0),$$

and since the highest order term of $P_i(A^T)$ is given by $(-1)^i\alpha_1\alpha_2\cdots\alpha_i(A^T)^i$, it follows that

$$\beta_i = (\tilde{\rho}_i/\tilde{\rho}_{i-1})(\alpha_{i-1}/\omega_{i-1}).$$

The other constants can be derived similarly.

Note that in our discussion we have focussed on the recurrence relations for the vectors r_i and p_i, while in fact our main goal is to determine x_i. As in all CG-type methods, x_i itself is not required for continuing the iteration, but it can easily be determined as a "sideproduct" by realizing that an update of the form $r_i = r_{i-1} - \gamma Ay$ corresponds to an update $x_i = x_{i-1} + \gamma y$ for the current approximated solution.

By writing r_i for $Q_i(A)P_i(A)r_0$ and p_i for $Q_{i-1}(A)T_{i-1}(A)r_0$, we obtain the following scheme for Bi-CGSTAB (we trust that, with the foregoing observations, the reader will now be able to verify the relations in Bi-CGSTAB). In this scheme we have computed the ω_i so that $r_i = Q_i(A)P_i(A)r_0$ is minimized in 2-norm as a function of ω_i.

6.5.7 Bi-CGSTAB2 and variants

The residual $r_k = b - Ax_k$ in the Bi-Conjugate Gradient method, when applied to $Ax = b$ with start x_0 can be written formally as $P_k(A)r_0$, where P_k is a k-degree polynomial. These residuals are constructed with one operation with A and one with A^T per iteration step. It was pointed out in [79] that with about the same amount of computational effort one can construct residuals of the form $\tilde{r}_k = P_k^2(A)r_0$, which is the basis for the CGS method. This can be achieved without any operation with A^T. The idea behind the improved efficiency of CGS is that if $P_k(A)$ is viewed as a reduction operator in BiCG, then one may hope that the square of this operator will be a twice as powerful reduction operator. Although this is not always observed in practice, one typically has that CGS converges faster than BiCG. This, together with the absence of operations with A^T, explains the success of the CGS method. A drawback of CGS is that its convergence behavior can look quite erratic, that is the norms of the resdiduals converge quite irregularly, and it may easily happen that $\|r_{k+1}\|_2$ is much larger than $\|r_k\|_2$ for certain k (for an explanation of this see [84]).
In [86] it was shown that by a similar approach as for CGS, one can construct methods for which r_k can be interpreted as $r_k = P_k(A)Q_k(A)r_0$, in which P_k is the polynomial associated with BiCG and Q_k can be selected free under the condition that $Q_k(0) = 1$. In [86] it was suggested to construct Q_k as the product of k linear factors $1 - \omega_j A$, where ω_j was taken to minimize locally a residual. This approach leads to the BiCGSTAB method. Because of the local minimization, BiCGSTAB displays a much smoother convergence behavior than CGS, and more surprisingly it often also converges (slightly) faster. One weak point in BiCGSTAB is that we get break-down if an ω_j is equal to zero. One may equally expect negative effects when ω_j is small. In fact, BiCGSTAB can be viewed as the combined effect of BiCG and GCR(1), or GMRES(1), steps. As soon as the GCR(1) part of the algorithm (nearly) stagnates, then the BiCG part in the next iteration step cannot (or only poorly) be constructed.

Another dubious aspect of BiCGSTAB is that the factor Q_k has only real roots by construction. It is well-known that optimal reduction polynomials for matrices with complex eigenvalues may have complex roots as well. If, for instance, the matrix A is real skew-symmetric, then GCR(1) stagnates forever, whereas a method like GCR(2) (or GMRES(2)), in which we minimize over two combined successive search directions, may lead to convergence, and this is mainly due to the fact that then complex eigenvalue components in the error can be effectively reduced.
This point of view was taken in [38] for the construction of the BiCGSTAB2 method. In the odd-numbered iteration steps the Q-polynomial is expanded by a linear factor, as in BiCGSTAB, but in the even-numbered steps this linear factor is discarded, and the Q-polynomial from the previous even-numbered step is expanded by a quadratic $1 - \alpha_k A - \beta_k A^2$. For this construction the information from the odd-numbered step is required. It was anticipated that the introduction of quadratic factors in Q might help to improve convergence for systems with complex eigenvalues, and, indeed, some improvement was observed in practical situations (see also [64]).
However, our presentation suggests a possible weakness in the construction of BiCGSTAB2, namely in the odd-numbered steps the same problems may occur as in BiCGSTAB. Since the even-numbered steps rely on the results of the odd-numbered steps, this may equally lead to unnecessary break-downs or poor convergence. In [78] another, and even simpler approach was taken to arrive at the desired even-numbered steps, without the necessity of the construction of the intermediate BiCGSTAB-type step in the odd-numbered steps. Hence, in this approach the polynomial Q is constructed straight-away as a product of quadratic factors, without ever constructing a linear factor. As a result the new method BiCGSTAB(2) leads only to significant residuals in the even-numbered steps and the odd-numbered steps do not lead necessarily to useful approximations.
In fact, it is shown in [78] that the polynomial Q can also be constructed as the product of ℓ-degree factors, without the construction of the intermediate lower degree factors. The main idea is that ℓ successive BiCG steps are carried out, where for the sake of an A^T-free construction the already available part of Q is expanded by simple powers of A. This means that after the BICG part of the algorithm vectors from the Krylov subspace $s, As, A^2s, ..., A^\ell s$, with $s = P_k(A)Q_{k-\ell}(A)r_0$ are available, and it is then relatively easy to minimize the residual over that particular Krylov subspace. There are variants of this approach in which more stable bases for the Krylov subspaces are generated [77], but for low values of ℓ a standard basis satisfies, together with a minimum norm solution obtained through solving the associated normal equations (which requires the solution of an ℓ by ℓ system. In most cases BiCGSTAB(2) will already give nice results for problems where BiCGSTAB or BiCGSTAB2 may fail. Note, however, that, in exact arithmetic, if no break-down situation occurs, BiCGSTAB2 would produce exactly the same results as BiCGSTAB(2) at the even-numbered steps.

Bi-CGSTAB(2) can be represented by the following algorithm:

x_0 is an initial guess; $r_0 = b - Ax_0$;
$\hat{r}_0$ is an arbitrary vector,
such that $(r, \hat{r}_0) \neq 0$,
e.g., $\hat{r}_0 = r$;
$\rho_0 = 1; u = 0; \alpha = 0; \omega_2 = 1$;
for $i = 0, 2, 4, 6, ...$
$\rho_0 = -\omega_2 \rho_0$
even BiCG step:
$\rho_1 = (\hat{r}_0, r_i); \beta = \alpha\rho_1/\rho_0; \rho_0 = \rho_1$
$u = r_i - \beta u$;
$v = Au$
$\gamma = (v, \hat{r}_0); \alpha = \rho_0/\gamma$;
$r = r_i - \alpha v$;
$s = Ar$
$x = x_i + \alpha u$;
odd BiCG step:
$\rho_1 = (\hat{r}_0, s); \beta = \alpha\rho_1/\rho_0; \rho_0 = \rho_1$
$v = s - \beta v$;
$w = Av$
$\gamma = (w, \hat{r}_0); \alpha = \rho_0/\gamma$;
$u = r - \beta u$
$r = r - \alpha v$
$s = s - \alpha w$
$t = As$
GCR(2)-part:
$\omega_1 = (r, s); \mu = (s, s)$;
$\nu = (s, t); \tau = (t, t)$;
$\omega_2 = (r, t); \tau = \tau - \nu^2/\mu$;
$\omega_2 = (\omega_2 - \nu\omega_1/\mu)/\tau$;
$\omega_1 = (\omega_1 - \nu\omega_2)/\mu$
$x_{i+2} = x + \omega_1 r + \omega_2 s + \alpha u$
$r_{i+2} = r - \omega_1 s - \omega_2 t$
if x_{i+2} accurate enough then quit
$u = u - \omega_1 v - \omega_2 w$
end

For more general BiCGSTAB(ℓ) schemes see [78, 77].
Another advantage of BiCGSTAB(2) over BiCGSTAB2 is in its efficiency. The BiCGSTAB(2) algorithm requires 14 vector updates, 9 innerproducts and 4 matrix vector products per full cycle. This has to be compared with a combined odd-numbered and even-numbered step in BiCGSTAB2, which requires 22 vector updates, 11 innerproducts, and 4 matrix vector products, and with two steps of BiCGSTAB which require 4 matrix vector products, 8

innerproducts and 12 vector updates. The numbers for BiCGSTAB2 are based on an implementation described in [64].
Also with respect to memory requirements, BiCGSTAB(2) takes an intermediate position: it requires 2 n-vectors more than BiCGSTAB and 2 n-vectors less than BiCGSTAB2.

For distributed memory machines the innerproducts may cause communication overhead problems (see, e.g., [16]). We note that the BiCG steps are very similar to conjugate gradient iteration steps, so that we may consider all kind of tricks that have been suggested to reduce the number of synchronization points caused by the 4 innerproducts in the BiCG parts. For an overview of these approaches see [6]. If on a specific computer it is possible to overlap communication with communication, then the BiCG parts can be rescheduled as to create overlap possibillities: 1. the computation of ρ_1 in the even BiCG step may be done just before the update of u at the end of the GCR part.
2. The update of x_{i+2} may be delayed until after the computation of γ in the even BiCG step.
3. The computation of ρ_1 for the odd BiCG step can be done just before the update for x at the end of the even BiCG step.
4. The computation of γ in the odd BiCG step has already overlap possibillities with the update for u.
For the GCR(2) part we note that the 5 innerproducts can be taken together, in order to reduce start-up times for their global assembling. This gives the method BiCGSTAB(2) a (slight) advantage over Bi-CGSTAB. Furthermore we note that the updates in the GCR(2) may lead to more efficient code than for BiCGSTAB, since some of them can be combined.

Our next numerical example illustrates quite nicely the difference in convergence behavior of some of the methods that we have discussed.

Example. We consider an advection dominated 2nd order PDE, with Dirichlet boundary conditions, on the unit cube (this equation was taken from [50]):

$$(6.5i) \qquad -u_{xx} - u_{yy} - u_{zz} + 1000\, u_x = f.$$

The function f is defined by the solution

$$u(x,y,z) = xyz(1-x)(1-y)(1-z).$$

This equation was discretized using $22 \times 22 \times 22$ volumes, resulting in a seven-diagonal linear system of order 10648. In order to make differences between iterative methods more visible, we have here and in our other examples not use any form of preconditioning.

In Figure 4 we see a plot of the convergence history. Bi-CGSTAB almost stagnates, as might be anticipated from the fact that this linear system has eigenvalues with relatively large imaginary parts. Surprisingly, Bi-CGSTAB does even worse than Bi-CG. For

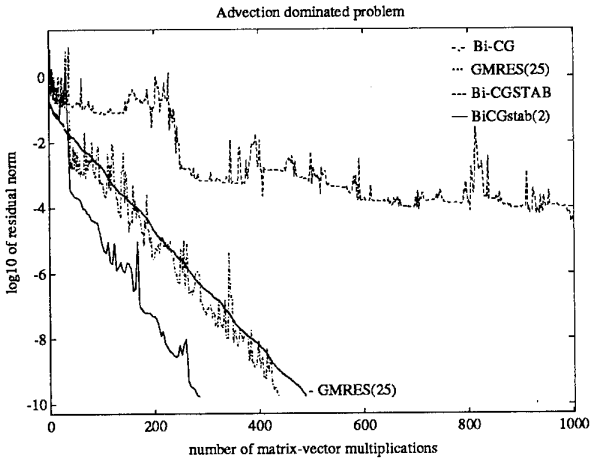


Fig.4: Convergence plot.

this type of matrices this behavior of Bi-CGSTAB is not uncommon and, as we will see in the next subsection, this can be explained by the poor recovery of the Bi-CG iteration coefficients α_k and β_k. Bi-CGstab(2) converges quite nicely and almost twice as fast as Bi-CG. GMRES(25) is about as fast as Bi--CG. Since the GMRES steps are much more expensive, BiCGstab(2) is the most efficient method here.

6.6 Maintaining Convergence:

The BiCGstab methods are designed for smooth convergence, with the purpose to avoid loss of local bi-orthogonality in the underlying Bi-CG process. This is important, since then the convergence of the Bi--CG part is exploited as much as possible. However, local bi-orthogonality may also be disturbed by, for instance, inaccuracies in the Bi-CG coefficients α and β. They are the quotients of scalars $\rho \equiv (r_i, \widehat{r}_0)$ and $\gamma \equiv (Ap, \widehat{r}_0)$ (see the algorithms for BiCGSTAB and BiCGSTAB(2)) and they will be inaccurate if ρ or γ is relatively small (see (6.6b)). The question is, when does this occur and how can it be avoided? Here, we will concentrate on ρ only, but similar arguments apply to γ as well.

As in the introduction of this section, r_i is the residual $r_i = \widetilde{P}_i(A)P_i(A)r_0$ where $\widetilde{P}_i$ is an appropriate polynomial of degree i with $\widetilde{P}_i(0) = 1$. Now, ρ is given by

$$(6.6a) \qquad \rho \equiv \rho_i \equiv (\widetilde{P}_i(A)P_i(A)r_0, \widehat{r}_0).$$

The scalar ρ_i can be small if the underlying Bi-Lanczos process nearly breaks down (i.e. $(\widetilde{P}_i(A)P_i(A)r_0, \widehat{r}_0) \approx 0$ relatively, for any polynomial $\widetilde{P}_i$ of degree i). Also an 'unlucky' choice of $\widetilde{P}_i$ may lead to a small ρ_i (which occurs in Bi-CGSTAB if the GCR(1) part stagnates). Here, we will concentrate on typical Bi-CGSTAB situations. Therefore, we assume that the Bi-Lanczos process itself (and the LU decomposition) does not (nearly) break down.

The relative rounding error ϵ_i in ρ_i can relatively and sharply be bounded by

$$|\epsilon_i| \leq n\overline{\xi}\frac{(|r_i|,|\widehat{r}_0|)}{|(r_i,\widehat{r}_0)|} \leq n\overline{\xi}\frac{\|r_i\|\,\|\widehat{r}_0\|}{|(r_i,\widehat{r}_0)|}. \tag{6.6b}$$

For a small relative error we want to have the expression at the right-hand side as small as possible.

Since the Bi-CG residual $P_i(A)r_0$, here to be denoted by s_i, is orthogonal to $\mathcal{K}_i(A^T;\widehat{r}_0)$ it follows that

$$(r_i,\widehat{r}_0) = \theta_i(A^i s_i,\widehat{r}_0)$$

if

$$\widetilde{P}_i(A) = \theta_i A^i + \theta_{i-1}^{(i)}A^{i-1} + \ldots.$$

Therefore, since $\|\widehat{r}_0\|/|(A^i s_i,\widehat{r}_0)|$ does not depend on $\widetilde{P}_i$, minimizing the right-hand side of (6.6b) is equivalent to minimizing

$$\frac{\|\widetilde{P}_i(A)s_i\|}{|\theta_i|} \tag{6.6c}$$

with respect to all polynomials $\widetilde{P}_i$ of exact degree i with $\widetilde{P}_i(0) = 1$. This minimization problem is solved by the FOM polynomial P_i^{F}, here associated with the initial residual s_i: P_i^{F} is the i^{th} degree polynomial for which $r_i^{\mathrm{F}} = P_i^{\mathrm{F}}(A)s_i$ (cf. Section 6.2). This polynomial is characterized by:

$$P_i^{\mathrm{F}}(A)s_i \perp \mathcal{K}_i(A;s_i) \text{ and } P_i^{\mathrm{F}}(0) = 1.$$

For optimally accurate coefficients, we should select FOM polynomials for our polynomials $\widetilde{P}_i$. However, since the hybrid Bi-CG methods are designed to avoid all the work for the construction of an orthogonal basis, the selection of complete FOM polynomials is out of the question.

For efficiency reasons, we have used products of first degree polynomials in Bi-CGSTAB and products of degree ℓ polynomials in BiCGstab(ℓ). Of course, our arguments can also be applied to such low degree factors. Therefore, suppose that $s = Q_{i-\ell}(A)P_i(A)r_0$ (as BiCGstab(ℓ)) has been computed and that the vectors s, As, ..., $A^\ell s$ are available. The suggestion for BiCGstab(ℓ) to minimize the residual over this particular Krylov subspace is equivalent to selecting a polynomial factor q_i ($Q_i = q_i Q_{i-\ell}$) of exact degree ℓ with $q_i(0) = 1$ such that

$$\|q_i(A)s\| \tag{6.6d}$$

is minimal, while in this situation, for optimal accurate coefficients, we rather would like to minimize

$$\|\!|q_i(A)s|\!\| \tag{6.6e}$$

where, with θ_i such that $q_i(A) = \theta_i A^\ell + \ldots$,

$$\|\!|q_i(A)s|\!\| \equiv \frac{\|q_i(A)s\|}{|\theta_i|}. \tag{6.6f}$$

The GMRES polynomial q_i^{G} of degree ℓ solves (6.6d), the FOM polynomial q_i^{F} solves (6.6e). For small residuals, the FOM polynomial is not optimal:

$$\|q_i^{\mathrm{G}}(A)s\| = |c_i|\,\|q_i^{\mathrm{F}}(A)s\|$$

with c_i as in (6.2d). Similarly, for accurate coefficients, the GMRES polynomial is not optimal [75]:

$$\|\!|q_i^{\mathrm{F}}(A)s|\!\| = |c_i|\,\|\!|q_i^{\mathrm{G}}(A)s|\!\|$$

with the same scalar c_i. For degree 1 factors, as in Bi-CGSTAB, (assuming no preconditioning)

$$c_i = \frac{(s,As)}{\|s\|\,\|As\|}, \tag{6.6g}$$

and c_i is the cosine of the angle between s and As (in the BiCGSTAB algorithm, t represents As).

Clearly, for extremely small $|c_i|$, say $|c_i| \leq \sqrt{\overline{\xi}}$ (in the $\ell = 1$ case, this means that s and As are almost orthogonal), taking GMRES polynomials for the degree ℓ factors will lead to inaccurate coefficients ρ_i, α and β, while FOM polynomials on the other hand will lead to large residuals. In both situations, the speed of convergence will seriously be deteriorated. The same phenomena can be observed when in a consecutive number of sweeps $|c_i|$ is small, but not necessarily extremely small (say, it takes k sweeps before $|c_{i-k}c_{i-k+1}\cdots c_i| \leq \sqrt{\overline{\xi}}$). In other words, the inaccuracies seem to accumulate. This seems to occur quite often in practise. E.g., for linear equation stemming from PDEs with large advection terms, Bi-CGSTAB often stagnates, although all c_i may be larger than, say .1, and none of the ω_i can considered to be relatively small ($\omega_i = c_i\|s\|/\|As\|$).

Both Bi-CGSTAB and BiCGstab(ℓ) are built on top of the same Bi-CG process. At roughly the same computational costs, one sweep of BiCGstab(ℓ) covers the same Bi-CG track as ℓ sweeps of Bi-CGSTAB. In one sweep of BiCGstab(ℓ), GMRES(ℓ) is applied once, in ℓ sweeps of BiCGSTAB, GMRES(1) is applied ℓ times. For two reasons it pays off to use GMRES(ℓ) instead of $\ell\times$GMRES(1):

1. Due the super-linear convergence, one sweep of GMRES(ℓ) may be expected to give a better residual reduction than ℓ times GMRES(1).
2. In ℓ steps of GMRES(1), ℓ small c_i's may contribute to inaccuracies in the coefficients α and β, where GMRES(ℓ) contributes ato this only once.

BiCGstab(ℓ) profits from GMRES(ℓ) by a better residual reduction in the GMRES part and by the faster convergence of a better recovered Bi-CG due to the more stable computations. However, we do not recommend to take ℓ large; $\ell = 2$ or $\ell = 4$ will usually lead already to almost optimal speed of convergence. The computational costs increase slightly

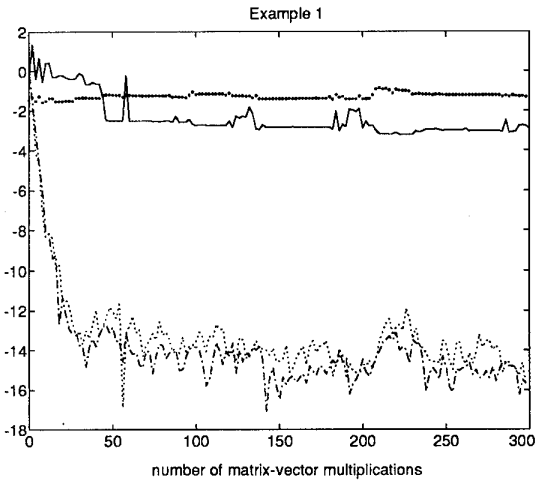


Fig.5: Convergence Bi-CGSTAB.

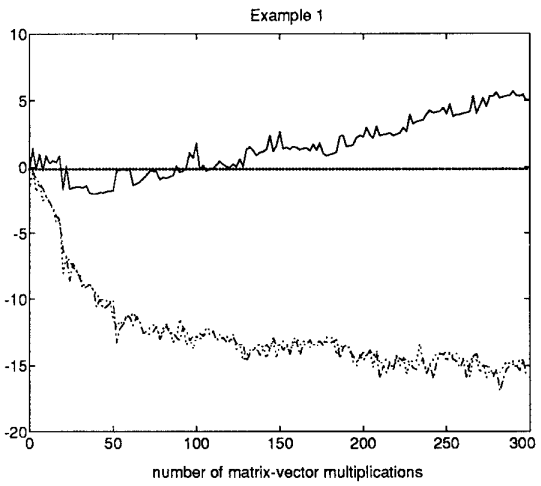


Fig.6: Convergence stabilized Bi-CGSTAB.

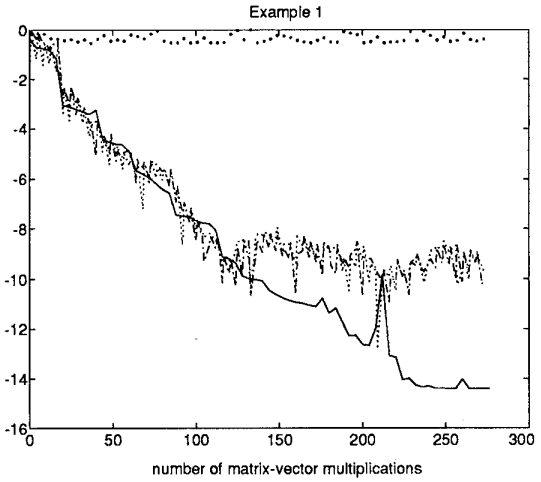


Fig.7: Convergence BiCGstab(2).

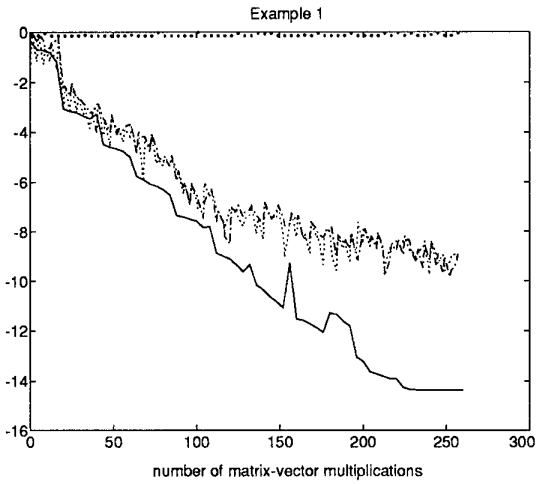


Fig.8: Convergence stab. BiCGstab(2).

by increasing ℓ (i.e. 2ℓ+10 vector updates and ℓ+7 inner products per 4 matrix multiplications), and more vectors have to be stored ($2\ell + 5$ vectors). Moreover, the method is less accurate for larger ℓ due to the fact that intermediate residuals (as r and $r - \omega_1 s$ in the Bi-CGSTAB(2) algorithm) can be large, with similar negative effects as in Section 6.5.4.

For Bi-CGSTAB there is a simple strategy that relaxes the danger of error amplification in consecutive sweeps with small $|c_i|$: replace in the Bi-CGSTAB algorithm the line

'$\omega = (s,t)/(t,t)$'

by the piece of code in Algorithm 1. In this way we limit the size of $|c|$. The constant .7 is rather arbitrarily and may be replaced by any other fixed non-small constant less than 1. Since GMRES(1) reduces well only if $|c_i| \approx 1$ (see (6.2c)), this strategy still profits from a possible good reduction by GMRES(1). A similar strategy that is equally cheap and easy to implement can be applied to BiCGstab(ℓ); see [75] for details.

We give a few numerical examples that demonstrate the cumulative effects of small $|c|$'s and that illustrate the effects of limiting its sizes.

Examples. The figures for the examples display, all on log-scale, the values for each iteration step of
- the residual-norms $||r||$, by solid curves (——);
- the scaled ρ, $\widehat{\rho} \equiv |(r, \widehat{r}_0)|/(||r||\,||\widehat{r}_0||)$ (cf. (6.6b)), by dashed-dotted curves (- · - · -);
- the scaled γ: $\widehat{\gamma} \equiv |(Ap, \widehat{r}_0)|/(||Ap||\,||\widehat{r}_0||)$, by dotted curves (·····);
- $|c|$, resp. $\max(|c|, 0.7)$, by bullets (●●●).

Before describing the examples, we will discuss part of the results.

In the figures 5–16, we see that the scaled ρ and the scaled γ behave similarly (the dashed-dotted - and dotted curves coincide more or less). Further, none of the $|c|$ is extremely small even not in cases where the $\widehat{\rho}$ and $\widehat{\gamma}$ are. The decrease of $\widehat{\rho}$ for values of $\widehat{\rho}$ not in the range of the machine precision ($\geq 10^{-12}$)

$$
\begin{array}{l}
\vdots \\
c = (s,t)/(\|s\|\,\|t\|); \\
\omega = \mathrm{sign}(c)\,\max(|c|, 0.7)\,\|s\|/\|t\|; \\
\vdots
\end{array}
$$

Alg.1: Limiting the size of $|c|$.

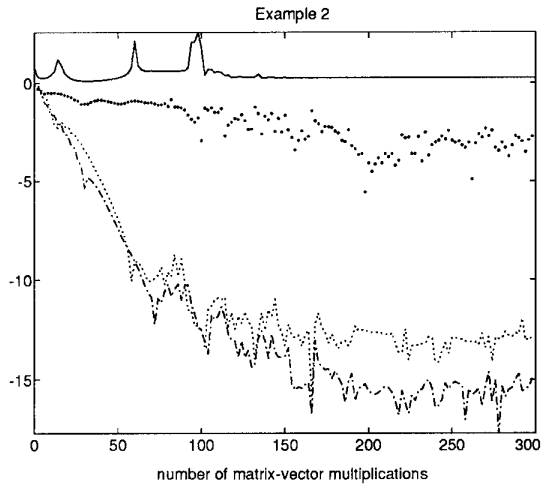


Fig.9: Convergence Bi-CGSTAB.

seems to be proportional to the product of previou: $|c|$'s. In all the examples, the method stagnates if $\widehat{\rho}$' or $\widehat{\gamma}$'s become extremely small, say less than 10^{-12} In these cases, almost all significance of the Bi-CC coefficients α and β will be lost. Limiting the size o. $|c|$ (Algorithm 1) slows down the decrease of $\widehat{\rho}$ and $\widehat{\gamma}$. In the caption of the figures, we used the adjective 'stabilized' to indicate that we used the limiting strategy. Often 'stabilizing' is enough to overcome the stagnation phase, and to lead to a converging process.

Example 1 (Figures 5–8). BiCGstab(2) converges. Although stabilizing Bi-CGSTAB leads to more accurate Bi-CG coefficients in the initial phase of the process, this is apparently not enough to restore full convergence.

Example 2 (Figures 9–12). Increasing ℓ to $\ell = 2$ leads to a slowly converging BiCGstab(2) process (many more than 300 matrix vector multiplications are needed; not shown in the graph). Our simple stabilizing strategy works well here.

Example 3 (Figures 13–16). The combined improvements, stabilizing and increasing ℓ to $\ell = 2$, are necessary for convergence.

For the first example, we have taken the PDE of (6.5i). The righ-hand side f is defined by the solution

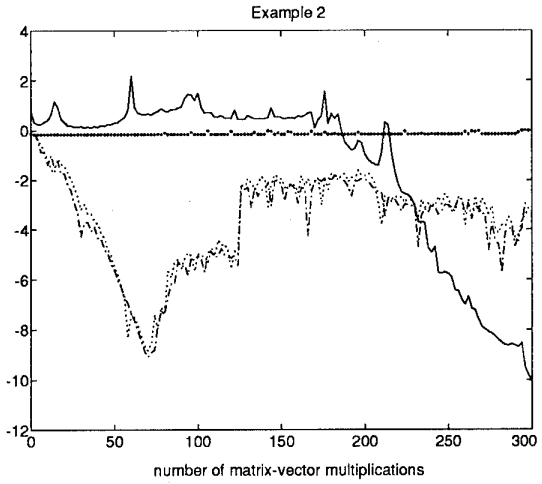


Fig.10: Convergence stabilized Bi-CGSTAB.

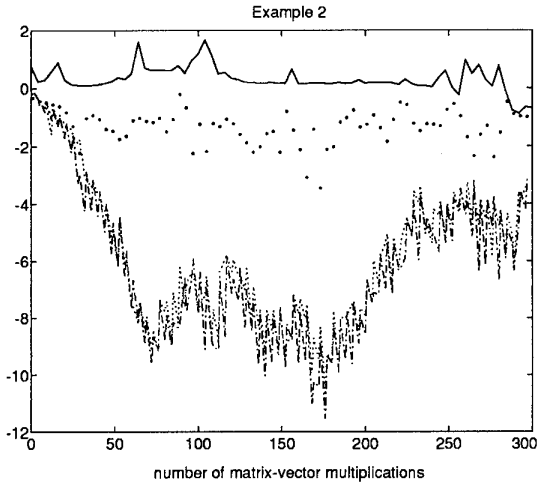


Fig.11: Convergence BiCGstab(2).

$$u(x,y,z) = \exp(xyz)\sin(\pi x)\sin(\pi y)\sin(\pi z).$$

The discretization is with $10 \times 10 \times 10$ finite volumes (no preconditioner has been used).

In the second and third example [33, 70], we have discretized

$$-u_{xx} - u_{yy} + a(xu_x + yu_y) + bu = f$$

on the unit-square with Dirichlet boundary conditions, with 63×63 finite volumes, taking $a = 100$ and $b = -200$, respectively 66×66, $a = 1000$ and $b = 10$ (no preconditioner has been used). The function f is such that the discrete solution is constant 1 (on the grid).

6.7 Generalized CGS:

We have now discussed in some detail the family of BiCGstab(ℓ) methods, but one should not deduce from this that these methods are to be preferred over CGS in all circumstances. We have had very good experiences with CGS in the context of solving nonlinear problems with Newton's method. It turns out

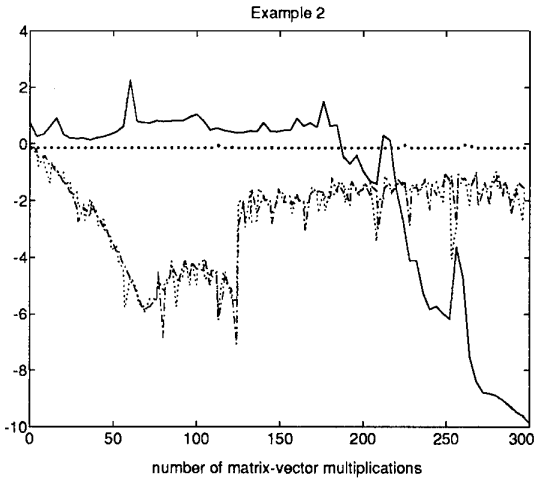


Fig.12: Convergence stab. BiCGstab(2).

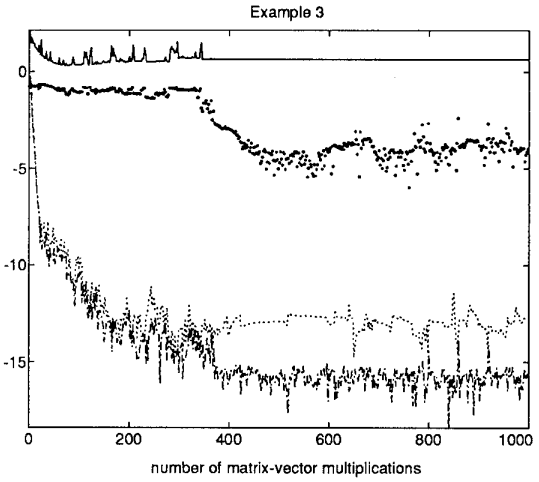


Fig.13: Convergence Bi-CGSTAB.

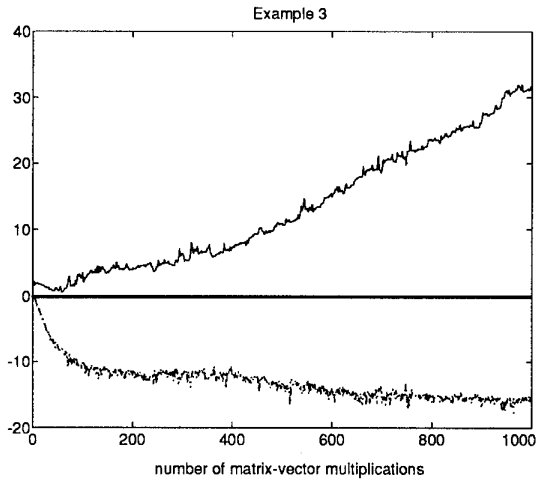


Fig.14: Convergence stabilized Bi-CGSTAB.

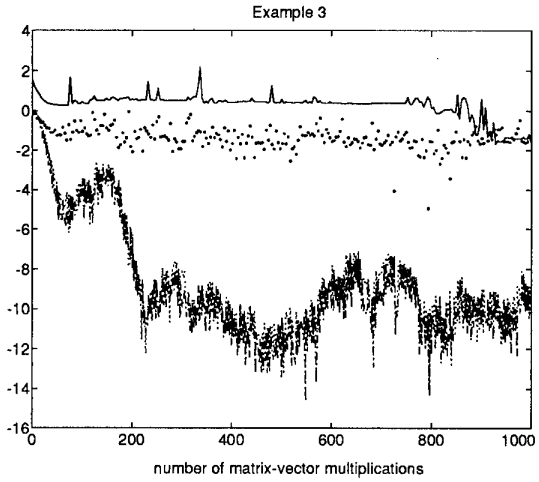


Fig.15: Convergence BiCGstab(2).

that we can exploit some of the presented ideas also to improve on CGS itself.

In the Newton method one has to solve a Jacobian system for the correction. This can be done by any method of choice, e.g., CGS or BiCGstab(ℓ). Often fewer Newton steps are required to solve a non-linear problem accurately when using CGS. Although the BiCGstab methods tend to solve each of the linear systems (defined by the Jacobi matrices) faster, the computational gain in these inner loops does not always compensate for the loss in the outer loop because of more Newton steps.

This phenomenon can be understood as follows. For eigenvalues λ that are extremal in the convex hull of the set of all eigenvalues of A (the Jacobian matrix), the values $P_i(\lambda)$ of the Bi-CG polynomials P_i tend to converge more rapidly towards zero than for eigenvalues λ in the interior. Since CGS squares the Bi-CG polynomials, CGS may be expected to reduce extremely well the components of the initial residual r_0 in the direction of the eigenvectors associated with extremal eigenvalues λ: with reduction factor $P_i(\lambda)^2$. Of course, the value $P_i(\lambda)$ can also be large, specifically for interior eigenvalues and in an initial stage of the process. CGS amplifies the associated components, (which also explains the typical irregular convergence behavior of CGS). The BiCGstab polynomial Q_i does not have this tendency of favoring the extremal eigenvalues. Therefore, the BiCGstab methods tend to reduce all eigenvector components equally well: on average, the "interior components" of a BiCGstab residual r_i are smaller than the corresponding components of a CGS residual $\tilde{r}_i$, while, with respect to the exterior components the situation is the other way around. However, the non-linearity of a non-linear problem seems often to be represented rather well by the space spanned by the "extremal eigenvectors". With respect to this space, and hence with respect to the complete space, Newtons scheme with CGS behaves like an exact Newton scheme.

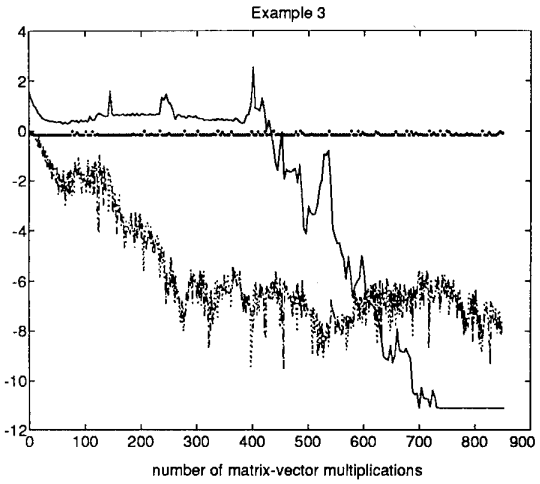


Fig.16: Convergence stab. BiCGstab(2).

We would like to preserve this property when constructing iterative schemes for Newton iterations. Fokkema et al [29] suggest polynomials $\widetilde{P}_i$ that lead to efficient algorithms (small modifications of the CGS algorithm) with a convergence that is slightly smoother, faster, and more accurate, than for CGS, but that still has the property of reducing extremal components quadratically. As a linear solver for isolated linear problems these "generalized CGS" schemes do not seem to have much advantage over BiCGstab(ℓ), but as a linear solver in a Newton scheme for non-linear problems, they often do rather well.

7 RELIABLE UPDATING

In all the Bi-CG related methods we see that the approximation for x and the residual vector r are updated by different vectors, and that the value for x does not influence the further iteration process, whereas the value for r does. In exact arithmetic the updated r is equal to the true residual $b - Ax$, but in rounded arithmetic it is unavoidable that differences between r and $b - Ax$ arise. This means that we may be misled for our stopping criteria, which are usually based upon knowledge of the updated r (and that we may have iterated too far in vain).

In this section we will discuss some techniques that have been proposed recently for the improvement of the updating steps. It turns out that this can be settled by relatively easy means.

Although the techniques in the previous section led to smoother and faster convergence and more accurate approximations the approximation may still not as accurate as possible. Here, we strive for optimal accuracy, i.e the updated r_i should be very close to the values of $b - Ax_i$, while leaving the convergence of the updated r intact.

First, we observe that even if x_m is the exact solution then the residual, computed in rounded arithmetic as $b - Ax_m$, may not be expected to be zero: using the notation of Section 6.5.4,

$$\begin{aligned} \|b - Ax_m\| &\leq \overline{\xi}\left(\|b\| + n_A \left\||A|\right\| \|x_m\|\right) \\ &\leq 2\Gamma\overline{\xi}\,\|b\|. \end{aligned} \tag{7.1a}$$

Therefore, the best we can strive for is an approximation x_m for which the true residual and the updated one differ in order of magnitude by the initial residual times the relative machine precision ($\mathcal{O}(\overline{\xi}\,\|r_0\|)$; recall that we assumed $x_0 = 0$, and hence $r_0 = b$).

Now it becomes also obvious why it is a bad idea to replace the updated residual in each step by the true one. Except from the fact that this would cost an additional matrix vector multiplication in each step, it also introduces errors in the recursions for the residuals. Although these errors may be expected to be small relatively to r_0, they will be large relatively to r_i if $\|r_i\| \ll \|r_0\|$. This perturbs the local bi-orthogonality of the underlying Bi-CG process and it may significantly slow down the speed of convergence. This observation suggests to replace the updated residual by the true one only if the updated residual has the same order of magnitude as the initial residual. However, meanwhile x_i and r_i may have drifted apart, and replacing r_i by $b - Ax_i$ brings in the "error of x_i" in the recursion (bounded as in (6.5g)), and again the speed of convergence may be affected. Although it is a good idea to use true residuals at strategic places, the approximation x_i should first be 'tied' more closely to the updated residual r_i. We can achieve this by updating x_i cumulatively: if $x_i = x_0 + w_1 + \ldots + w_i$ (cf. (6.5d)) then we actually compute x_i in groups as

$$x_i = x_0 + x'_1 + x'_2 + \ldots \tag{7.1b}$$

where, for some decreasing sequence of indices $\pi(1) = 1, \pi(2), \ldots$, x'_j represents the sum of a group;

$$x'_j = w_{\pi(j)} + w_{\pi(j)+1} + \ldots + w_{\pi(j+1)-1}, \text{ etc..}$$

Simultaneously, we compute r_i as

$$r_i = r_0 - Ax'_1 - Ax'_2 - \ldots. \tag{7.1c}$$

In this way we can control the size of the updates for x_i and r_i, and we avoid large errors (cf. (6.5g)): for a proper choice of the $\pi(j)$, the x'_j will be small even if some of the w_j are large.

In the modification of the algorithms that we will propose in Algorithm 2, we kept in mind that we only may allow errors which
(a) are small with respect to the initial residual r_0 (otherwise accuracy will be disturbed) and
(b) are small with respect to the present updated residual r_i (otherwise local bi-orthogonality may be jeopardized).

In Section 3, we have explained that it is no restriction to take $x_0 = 0$, arguing that this situation can be forced simply by a shift: shift x by x_0, and b by Ax_0. This shift can be made explicit in the hybrid Bi-CG algorithms by making three changes:
(i) adding as a last line to the initialization phase

$$x = x_0; \quad x' = 0; \quad b' = r_0;$$

(ii) adding as a last line in the algorithms (just after 'end')

$$x = x + x';$$

(iii) replacing all x_i (and x) by x' (skipping the index i).
Even in rounded arithmetic, this modification will not change the value of any of the vectors and scalars in the computational scheme, except for the x's. Since $x + x'$ is the approximation that we are interested in, one also may want to change the termination criterion. We propose to replace the line

if x is accurate enough then quit;

by

if $\|r_{i+1}\|$ is small enough then quit;

To allow for a more accurate way of updating of the residual and the approximation, we suggest to add another few lines just before 'end' in the algorithm, as is shown in Algorithm 2. We suggest to replace the

```
    ⋮
x = x_0;   x' = 0;   b' = r_0;
for i = 0, 1, 2, ...
        ⋮
        ⋮   Replace all x_i and x by x'.
        ⋮
    if r_{i+1} is small enough then quit;
    set 'compute_res' and 'update_app';
    if 'compute_res' is true
        r_{i+1} = b' - Ax';
        if 'update_app' is true
            x = x + x'; x' = 0; b' = r_{i+1};
        endif
    endif
endfor
x = x + x';
```

Alg.2: For accurate approximations.

updated residual by the true one on strategically chosen steps (we have to explain when the value of the boolean functions *'compute_res'* is true). However, we also suggest to shift the problem once in a while (when the boolean function *'update_app'* is true) in order to let the right-hand decrease (cf. (7.1a)). Here we use the fact that, in exact arithmetic, also these intermediate shifts do not change the iteration parameters and vectors (except for the vectors x). Observe that the updated residual r_{i+1} is replaced by the true residual $b' - Ax'$ of the shifted problem if *'compute_res'* is true.

For this we propose the following strategy.
Update x and b' only if the residual is significantly smaller than the initial residual, while an intermediate residual was larger (cf. (7.1b), (7.1c) and reminder (a)):

$$\begin{aligned} &\textit{'update_app'} = \text{true} \\ (7.1\text{d}) \quad &\text{if} \quad \|r_{i+1}\| \le \|b\|/100 \ \ \& \ \ \|b\| \le \mu \\ &\text{else } \textit{'update_app'} = \text{false}, \end{aligned}$$

where $\mu \equiv \max\|r_i\|$ and the maximum is taken over all residuals since the previous update of x and b' (since the previous *'update_app'* is true).
The bound in (7.1a) suggests that the norm $\|b\|$ of the initial residual should be used as criterion for shifting the problem (*'update_app'* is true if $\|r_{i+1}\| \le \|b\|$ & $\|r_i\| \ge \|b\|$). However, if the process converges irregularly this would lead to many shifts. The relaxed version in (7.1d) turns out to work equally well at less costs.
Compute a true residual whenever *'compute_res'* is true and if a previous residual is larger than the initial residual and significantly larger than the present updated residual:

$$\begin{aligned} &\textit{'compute_res'} = \text{true} \\ (7.1\text{e}) \quad &\text{if} \quad \|r_{i+1}\| \le M/100 \ \ \& \ \ \|b\| \le M \\ &\quad\ \text{or } \textit{'update_app'} \text{ is true} \\ &\text{else } \textit{'compute_res'} = \text{false}, \end{aligned}$$

where $M \equiv \max\|r_i\|$ and the maximum is taken over all residuals since the last computation of the true residual.
Replacing the updated residual by the true one perturbs the recursion for the residuals. If the residual decreases too much since the previous replacement, the perturbation may become large relatively to the present residual (reminder (b)). Therefore, *'compute_res'* may be true more often than *'update_app'*.

We suggest to add the above strategy to an existing code. That means that an additional matrix-vector multiplication has to be performed whenever a true residual has to be computed. The conditions (7.1d) and (7.1e) are chosen as to minimize the number of these additional computations. One also may try to skip a matrix-vector multiplication in one of the preceding lines of the algorithm, which requires some additional care for BiCGstab(ℓ), but which easily can be accomplished for CGS.

If CGS is modified as suggested, then the new lines do not require additional matrix vector multiplications, and there is no need to restrict the number of

computations of true local residuals. For this CG variant, Neumaier [56] suggested places where the x and b' can be updated for accurate approximations update x and b' whenever the residual decreases with respect to the previous 'best residual',

$$\begin{aligned} &\text{'update_app'} = \text{true} \\ &\text{if} \quad \|r_{i+1}\| \leq \|b'\| \\ &\text{else 'compute_res'} = \text{false}. \end{aligned} \tag{7.1f}$$

The modifications according to Neumaier's approach are given in Algorithm 3. Observe that the norm o

$$\begin{array}{l} \vdots \\ x = x_0;\ x' = 0;\ b' = r_0;\ \mu' = \|b'\|; \\ \text{for } i = 0, 1, 2, \ldots \\ \qquad \vdots \\ \qquad \textit{Replace all } x_i \textit{ and } x \textit{ by } x'. \\ \qquad \vdots \\ \textit{Skip the CGS update for } r \\ \textit{together with the MV involved} \\ \textit{in this update. Compute instead} \\ \quad r_{i+1} = b' - Ax';\ \mu = \|r_{i+1}\|; \\ \quad \text{if } \mu \text{ is small enough then quit;} \\ \quad \text{if } \mu \leq \mu' \\ \qquad x = x + x';\ x' = 0; \\ \qquad b' = r_{i+1};\quad \mu' = \mu; \\ \quad \text{endif} \\ \text{endfor} \\ x = x + x'; \end{array}$$

Alg.3: Neumaier's strategy for CGS.

the b' (the residuals with respect to the x) strictly decrease: the Neumaier trick also smoothes convergence (without improving its speed!).

Below, we discuss the effects of our strategies in practise. We illustrate our observations by a simple numerical example.

Example. Figure 17 shows the convergence history of the *true residuals* as produced by standard CGS, and by the modified versions of CGS as suggested above, applied to the SHERMAN4 matrix of the Harwell-Boeing collection (as in the example of Section ??). The dotted curve (···) represents the results for standard CGS. We also applied modified CGS as in Algorithm 2, using the update criterions (7.1d) and (7.1e). The solid curve (——) represents the results for this *simple strategy*, while the dashed-dotted curve (- · - ·) represents the results for Neumaier's strategy in Algorithm 3. On log-scale, the norm of the true residuals $\|b - Ax_i\|$, $\|b - A(x + x')\|$, respectively, is plotted against the number of matrix-vector multiplications. Neumaier's strategy as well our's lead to approximations that are accurate (cf. (7.1a)): comparing $\|r_0\|$

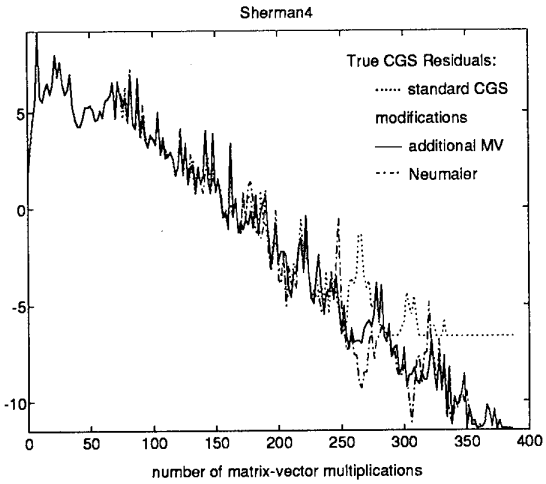


Fig.17: Reliable updates.

with the norm of the smallest true residual, we see that a reduction is obtained by a factor $\approx 10^{-14}$ ($\overline{\xi} = 2.2\,10^{-16}$). Standard CGS does not produce true residuals smaller than $\approx 10^{-9}\|r_0\|$, which is approximately $\overline{\xi} \cdot \max\|r_i\| \approx 2.2\,10^{-16} \cdot 10^{7}\|r_0\|$; cf. (6.5g). Observe that, though the convergence histories do not coincide for residuals less than $\approx 10^{1}$, the speed of convergence is not affected: the modified versions exhibit a rate of convergence that is very similar to the one of the updated residuals in standard CGS as shown in Figure 3.

Experiments for other examples and with other iterative schemes, as Bi-CGSTAB and BiCGstab(ℓ), led to similar conclusions. Although, two observations should be made.

— Quite often the improvements are much more spectacular than for this SHERMAN4 example: CGS may produce intermediate residuals as large as $\|r_0\|/\overline{\xi}$ and none of the digits in the finial approximation of standard CGS will be correct.

— There are some differences between CGS and the BiCGstab methods: (i) as observed above, Neumaier's strategy only works well for CGS, while the simple strategy of Algorithm 2 can always be applied. (ii) Especially for the BiCGstab methods, the simple strategy of Algorithm 2 with update criterions (7.1d) and (7.1e) does not lead to much additional work. The additional computation of a true residual takes place after the process encounters residuals that are (much) larger than the initial residual. Since BiCGstab(ℓ) tends to show much smoother convergence behavior than CGS, for small ℓ, the additional work in these methods is usually much less than for CGS. In the SHERMAN4 example, our strategy for CGS requires 7 additional matrix-vector multiplications ('*compute_res*' is true 7 times) and one special update of the approximation ('*update_app*' is true only once). For BiCGstab(ℓ), $\ell \leq 6$, only 1 additional

matrix-vector multiplication was needed. Neumaier's strategy for CGS does not require additional matrix-vector multiplications (but 364 additional updates for the approximation were needed).

8 Termination Criteria

An important point, when using iterative processes, is to decide when to terminate the process. Popular stopping criteria are based on the norm of the current residual, or on the norm of the update to the current approximation to the solution (or a combination of these norms). More sophisticated criteria have been discussed in litterature.
In [45] a practical termination criterion for the conjugate gradient method is considered. Suppose we want an approximation x_i for the solution x for which

$$\|x_i - x\|_2 / \|x\|_2 \leq \varepsilon,$$

where ε is a tolerance set by the user.
It is shown in [45] that such an approximation is obtained by CG as soon as

$$\|r_i\|_2 \leq \mu_1 \|x_i\|_2 \varepsilon / (1+\varepsilon),$$

where μ_1 stands for the smallest eigenvalue of the positive definite symmetric (preconditioned) matrix A. Of course, in most applications the value for μ_1 will be unknown, but with the iteration coefficients of CG we can build the tridiagonal matrix T_i, and compute the smallest eigenvalue (Ritz value) $\mu_1^{(i)}$ of T_i, which is an approximation for μ_1. In [45] a simple algorithm for the computation of $\mu_1^{(i)}$, along with the CG algorithm, is described, and it is shown that a rather robust stopping criterion is formed by

$$\|r_i\|_2 \leq \mu_1^{(i)} \|x_i\|_2 \varepsilon / (1+\varepsilon).$$

A similar criterion has also been suggested earlier in [40].

A quite different, but much more generally applicable approach has been suggested in [1]. In this approach the approximate solution of an iterative process is regarded as the exact solution of some (nearby) linear system, and computable bounds for the perturbations with respect to the given system are given. A nice overview of termination criteria has been presented in [6]: Section 4.2.

9 Implementation Aspects

For effective use of the given iteration schemes, it is necessary that they can be implemented such that high computing speeds are achievable. It is most likely that high computing speeds will be realized only by parallel architectures and therefore we must see how well iterative methods fit to such computers.

The iterative methods only need a handful of basic operations per iteration step

- Vector updates: in each iteration step the current approximation to the solution is updated by a correction vector. Often the corresponding residual vector is also obtained by a simple update, and we have update formulas as well for the correction vector (or search direction).

- Innerproducts: In many methods the speed of convergence is influenced by carefully constructed iteration coefficients. These coefficients are sometimes known analytically, but more often they are computed by innerproducts, involving residual vectors and search directions, as in the methods discussed in the previous sections.

- Matrix vector products: In each step at least one matrix vector product has to be computed with the matrix of the given linear system. Sometimes also matrix vector products with the transpose of the given matrix are required (e.g., BiCG). Note that it is not necessary to have the matrix explicitly, it suffices to be able to generate the result of the matrix vector product.

- Preconditioning: It is common practice to precondition the given linear system by some preconditioning operator. Again it is not necessary to have this operator in explicit form, it is enough to generate the result of the operator aplied to some given vector. The preconditioner is applied as often as the matrix vector multiply in each iteration step.

For problem sizes large enough the innerproducts, vectorupdates and matrix vector product are easily parallelized and vectorized. The more successful preconditionings, i.e, based upon incomplete LU decomposition, are not easily parallelizable. For that reason one is often satisfied with the use of only diagonal scaling as a preconditioner on highly parallel computers, such as the CM2 [7].

On distributed memory computers we need large grained parallelism in order to reduce synchronization overhead. This can be achieved by combining the work required for a successive number of iteration steps. The idea is to construct first in parallel a straight forward Krylov basis for the search subspace in which an update for the current solution will be determined. Once this basis has been computed, the vectors are orthogonalized, as is done in Krylov subspace methods. The construction as well as the orthogonalization can be done with large grained parallelism, and has sufficient degree of parallelism in it. This approach has been suggested for CG in [11] and for GMRES in [12], [5] and [18]. One of the disadvantages in this approach is that a straight forward basis, of the form $y, Ay, A^2y, ..., A^iy$ is usually very

ill-conditioned. This is in sharp contrast to the optimal condition of the orthogonal basis set constructed by most of the projection type methods and it puts severe limits on the number of steps that can be combined. However, in [5] and [18] ways to improve the condition of a parallel generated basis are suggested and it seems possible to take larger numbers of steps, say 25, together. In [18] the effects of this approach on the communication overhead are studied and compared with experiments done on moderately massive parallel transputer systems.

9.1 Parallelism in the preconditioner:

In this section we consider a number of possibilities to obtain parallelism in the standard Incomplete Choleski preconditioner [51]. The linear systems are supposed to arise from standard finite difference discretisations of second order pde's over rectangular grids in two or three dimensional space.

9.1.1 Overlapping Local Preconditioners

Radicati di Brozolo and Robert [66] suggest to partition the given matrix A in (slightly) overlapping blocks along the main diagonal. Note that a given non-zero entry of A is not necessarily contained in one of these blocks. But experience suggests that this approach is more successful if these blocks cover all the non-zero entries of A. The idea is to compute in parallel local preconditioners for all of the blocks; e.g.,

(9.1a) $$A_n = L_n {D_n}^{-1} U_n - R_n.$$

Then, when solving $Kw = r$ in the preconditioning step, we partition r in (overlapping) parts r_n, according to A_n, and we solve the systems $L_n D_n^{-1} U_n w_n = r_n$ in parallel. Finally we define the elements of w to be equal to corresponding elements of the w_n's in the nonoverlapping parts and to the average of them in the overlapped parts.

Radicati di Brozolo and Robert [66] report on timing results obtained on an IBM3090-600E/VF for GMRES preconditioned by overlapped incomplete LU decomposition for a 2D system of order 32400 with a bandwidth of 360. For p processors ($1 \leq p \leq 6$) they subdivide A in p overlapping parts, the overlap being so large that thses blocks cover all the nonzero entries of A. They found experimentally an overlap of about 360 elements to be optimal for their problem. This approach led to a speedup of roughly p. In some cases a speedup even slightly larger than p was observed, apparantly due to the fact that the parallel preconditioner was slightly more effective than the standard one in those cases.

9.1.2 Repeated Twisted Factorization

Meurant [54] reports on timing results obtained with a CRAY Y-MP/832, using an incomplete repeated twisted block factorization for 2D problems. In his experiments the L of the incomplete factorization has a block structure, i.e., L has alternatingly a block below the diagonal, one above, one below, and it ends with one above the diagonal. For this approach Meurant reports a speedup, for preconditioned CG, close to 6 on the 8-processor CRAY Y-MP. This speedup has been measured relative to the same repeated twisted factorization process executed on one single processor. Meurant also reports an increase in the number of iteration steps, due to this repeated twisting. This implies that the effective speedup with respect to the nonparallel code is only about 4.

9.1.3 Twisted and Nested Twisted Factorization

For 3D problems we have used the blockwise twisted approach [23] in the z- direction, i.e. the (x, y)-planes in the grid were treated in parallel from bottom and top inwards. Over each plane we used the diagonalwise ordering, in order to achieve high vector speeds on each processor.
On a dedicated CRAY X-MP/2 this led, for preconditioned CG, to a reduction by a factor of close to 2 in wall clock time with respect to the CPU time for the nonparallel code on one single processor. For the microtasked code the wall clock time on the 2-processor system was measured for a dedicated system, whereas for the nonparallel code the CPU time was measured on a moderately loaded system. In some situations the speedup was even slightly larger than 2, due to better convergence properties of the twisted incomplete preconditioner.
The effects of these and other orderings on the convergence of preconditioned methods and on the amount of parallelism have been studied in [21].

We can also apply the twisted incomplete factorization in a nested way [83]. For 3D problems this can be exploited by twisting also the blocks corresponding to (x, y) planes in the y-direction. Over the resulting blocks, corresponding to half (x, y) planes, we may apply diagonal ordering in order to fully vectorize the four parallel parts.
By this approach we have been able to reduce the wall clock time by a factor of 3.3, for preconditioned CG, on the 4-processor CONVEX C-240. In this case the total CPU time, used by all of the processors, is roughly equal to the CPU time required for single processor execution [85]. Other then for the experiments on the CRAY X-MP/2, as reported before, we have relied on the autotasking capabilities of the Fortran compiler for the C-240, for all of the code, except for the preconditioning part. Since some statements in the code lead to rather short vector lengths, this may explain partially why the factor 3.3 for the CONVEX C-240 stays well behind the theoretically

expected factor of about 3.9. Another reason might be that we were not completely sure whether our testing machine was executing constantly in stand alone mode during the time of our timing experiments. Even the system itself needs some CPU-time from time to time.

9.1.4 Hyperplane Ordering

For a CYBER 205 it has been reported how to obtain long vectorlengths for certain 3D situations ([23], [73]), and, of course, this approach can also be followed in order to obtain parallelism. This has been done by Berryman et.al. [7] for parallelizing standard ICCG on a Connection Machine CM-2. For a 4K processor machine they report a computational speed of 52.6 Mflops for the (sparse) matrix vector product, while 13.1 Mflops has been realized for the preconditioner, using the hyperplane approach.
This reduction in speed by a factor of 4 makes it attractive to use only diagonal scaling as a preconditioner in some situations, for massively parallel machines like the CM-2. The latter approach has been followed by Mathur and Johnsson [48] for finite element problems.

We have used the hyperplane ordering for preconditioned CG on an ALLIANT FX/4, for 3D systems with dimensions $n_x = 40, n_y = 39$ and $n_z = 30$. For 4 processors this led to a speedup of 2.61, to be compared with a speedup of 2.54 for the CG-process with only diagonally scaling as a preconditioner. The fact that both speedups are quite far below the optimal value of 4, must be attributed to cache effects [85]. These cache effects can be largely removed, when using the reduced system approach suggested by Meier and Sameh [49]. However, for the 3D systems that we have tested sofar, the reduced system approach led, in average, to about the same CPU times as for the hyperplane approach, on Alliant FX/8 and FX/80 computers.

10 *

References

[1] M. Arioli, I. S. Duff, and D. Ruiz. Stopping criteria for iterative solvers. *SIAM J. Matrix Anal. Appl.*, 13:138–144, 1992.

[2] O. Axelsson. Solution of linear systems of equations: iterative methods. In V. A. Barker, editor, *Sparse Matrix Techniques*, Berlin, 1977. Copenhagen 1976, Springer Verlag.

[3] O. Axelsson. Conjugate gradient type methods for unsymmetric and inconsistent systems of equations. *Lin. Alg. and its Appl.*, 29:1–16, 1980.

[4] O. Axelsson and P. S. Vassilevski. A black box generalized conjugate gradient solver with inner iterations and variable-step preconditioning. *SIAM J. Matrix Anal. Appl.*, 12(4):625–644, 1991.

[5] Zhaojun Bai, Dan Hu, and Lothar Reichel. A Newton basis GMRES implementation. Technical Report 91-03, University of Kentucky, 1991.

[6] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM, Philadelphia, PA, 1994.

[7] H. Berryman, J. Saltz, W. Gropp, and R. Mirchandaney. Krylov methods preconditioned with incompletely factored matrices on the CM-2. Technical Report 89-54, NASA Langley Research Center, ICASE, Hampton, VA, 1989.

[8] A. Björck and T. Elfving. Accelerated projection methods for computing pseudo-inverse solutions of systems of linear equations. *BIT*, 19:145–163, 1979.

[9] P. N. Brown. A theoretical comparison of the Arnoldi and GMRES algorithms. *SIAM J. Sci. Statist. Comput.*, 12:58–78, 1991.

[10] G. Brussino and V. Sonnad. A comparison of direct and preconditioned iterative techniques for sparse unsymmetric systems of linear equations. *Int. J. for Num. Methods in Eng.*, 28:801–815, 1989.

[11] A. T. Chronopoulos and C. W. Gear. s-Step iterative methods for symmetric linear systems. *J. on Comp. and Appl. Math.*, 25:153–168, 1989.

[12] A. T. Chronopoulos and S. K. Kim. s-Step Orthomin and GMRES implemented on parallel computers. Technical Report 90/43R, UMSI, Minneapolis, 1990.

[13] P. Concus and G. H. Golub. A generalized Conjugate Gradient method for nonsymmetric systems of linear equations. Technical Report STAN-CS-76-535, Stanford University, Stanford, CA, 1976.

[14] P. Concus, G. H. Golub, and D. P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In J. R. Bunch and D. J. Rose, editors, *Sparse Matrix Computations.* Academic Press, New York, 1976.

[15] G. C. (Lianne) Crone. The conjugate gradient method on the Parsytec GCel-3/512. *to appear in FGCS.*

[16] L. Crone and H. van der Vorst. Communication aspects of the conjugate gradient method on distributed-memory machines. *Supercomputer*, X(6):4–9, 1993.

[17] E. de Sturler. A parallel restructured version of GMRES(m). Technical Report 91-85, Delft University of Technology, Delft, 1991.

[18] E. de Sturler. A parallel variant of GMRES(m). In R. Miller, editor, *Proc. of the fifth Int.Symp. on Numer. Methods in Eng.*, 1991.

[19] E. De Sturler and D. R. Fokkema. Nested Krylov methods and preserving the orthogonality. In N. Duane Melson, T.A. Manteuffel, and S.F. McCormick, editors, *Sixth Copper Mountain Conference on Multigrid Methods*, volume Part 1 of *NASA Conference Publication 3324*, pages 111–126. NASA, 1993.

[20] J. Demmel, M. Heath, and H. van der Vorst. Parallel linear algebra. In *Acta Numerica 1993*. Cambridge University Press, Cambridge, 1993.

[21] Shun Doi. On parallelism and convergence of incomplete LU factorizations. *Appl. Num. Math.*, 7:417–436, 1991.

[22] J. J. Dongarra. Performance of various computers using standard linear equations software in a fortran environment. Technical Report CS-89-85, University of Tennessee, Knoxville, 1990.

[23] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM, Philadelphia, PA, 1991.

[24] Jack J. Dongarra and Henk A. van der Vorst. Performance of various computers using standard sparse linear equations solving techniques. *Supercomputer*, 9(5):17–29, 1992.

[25] I. S. Duff, A. M. Erisman, and J.K.Reid. *Direct methods for sparse matrices.* Oxford University Press, London, 1986.

[26] T. Eirola and O. Nevanlinna. Accelerating with rank-one updates. *Lin. Alg. and its Appl.*, 121:511–520, 1989.

[27] H. C. Elman. *Iterative methods for large sparse nonsymmetric systems of linear equations.* PhD thesis, Yale University, New Haven, CT, 1982.

[28] R. Fletcher. *Conjugate gradient methods for indefinite systems*, volume 506 of *Lecture Notes Math.*, pages 73–89. Springer-Verlag, Berlin–Heidelberg–New York, 1976.

[29] D. R. Fokkema, G.L.G. Sleijpen and H.A. Van der Vorst. Generalized Conjugate Gradient Squared. *Preprint 851*, Dept. Math., University Utrecht, 1994.

[30] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. *SIAM J. Sci. Comput.*, 14:137–158, 1993.

[31] R. W. Freund and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, part 2. Technical Report 90.46, RIACS, NASA Ames Research Center, 1990.

[32] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Num. Math.*, 60:315–339, 1991.

[33] R. W. Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14:470–482, 1993.

[34] G. H. Golub and D.P. O'Leary. Some history of the conjugate gradient and lanczos algorithms: 1948-1976. *SIAM Review*, 31:50–102, 1989.

[35] G. H. Golub and C. F. van Loan. *Matrix Computations.* North Oxford Academic, Oxford, 1983.

[36] G. H. Golub and C. F. van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, 1989.

[37] I. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.

[38] M. H. Gutknecht. Variants of BICGSTAB for matrices with complex spectrum. *SIAM J. Sci. Comput.*, 14:1020–1033, 1993.

[39] W. Hackbusch. *Iterative Lösung großer schwachbesetzter Gleichungssysteme.* Teubner, Stuttgart, 1991.

[40] L. A. Hageman and D. M. Young. *Applied Iterative Methods.* Academic Press, New York, 1981.

[41] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, 49:409–436, 1954.

[42] C. P. Jackson and P. C. Robinson. A numerical study of various algorithms related to the preconditioned Conjugate Gradient method. *Int. J. for Num. Meth. in Eng.*, 21:1315–1338, 1985.

[43] D. A. H. Jacobs. Preconditioned Conjugate Gradient methods for solving systems of algebraic equations. Technical Report RD/L/N 193/80, Central Electricity Research Laboratories, 1981.

[44] K. C. Jea and D. M. Young. Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods. *Lin. Algebra Appl.*, 34:159–194, 1980.

[45] E. F. Kaasschieter. A practical termination criterion for the Conjugate Gradient method. *BIT*, 28:308–322, 1988.

[46] C. Lanczos. An iteration method for the solution of the egenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stand*, 45:225–280, 1950.

[47] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Natl. Bur. Stand*, 49:33–53, 1952.

[48] K. K. Mathur and S. L. Johnsson. The finite element method on a data parallel computing system. Technical Report CS 89-2, Thinking Machines Corporation, 1989. to appear in International Journal of High-Speed Computing.

[49] U. Meier and A. Sameh. The behavior of conjugate gradient algorithms on a multivector processor with a hierarchical memory. Technical Report CSRD 758, University of Illinois, Urbana, IL, 1988.

[50] U. Meier Yang. Preconditioned Conjugate Gradient-Like Methods for Nonsymmetric Linear Systems. *Preprint*, Center for Research and Development, University of Illinois at Urbana-Champaign, 1992.

[51] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math.Comp.*, 31:148–162, 1977.

[52] G. Meurant. The block preconditioned conjugate gradient method on vector computers. *BIT*, 24:623–633, 1984.

[53] G. Meurant. Numerical experiments for the preconditioned conjugate gradient method on the CRAY X-MP/2. Technical Report LBL-18023, University of California, Berkeley, CA, 1984.

[54] G. Meurant. The conjugate gradient method on vector and parallel supercomputers. Technical Report CTAC-89, University of Brisbane, July 1989.

[55] N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen. How fast are nonsymmetric matrix iterations? *SIAM J. Matrix Anal. Appl.*, 13:778–795, 1992.

[56] A. Neumaier. Oral presentation at the Oberwolfach meeting: Numerical Linear Algebra, Oberwolfach, 1994.

[57] J. M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, New York and London, 1988.

[58] C. C. Paige. Computational variants of the Lanczos method for the eigenproblem. *J. Inst. Math. Appl.*, 10:373–381, 1972.

[59] C. C. Paige, B. N. Parlett, and H. A. van der Vorst. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Num. Lin. Alg. with Appl.*, 2:115-134, 1995.

[60] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.

[61] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Soft.*, 8:43–71, 1982.

[62] B. N. Parlett, D. R. Taylor, and Z. A. Liu. A look-ahead Lanczos algorithm for unsymmetric matrices. *Math. Comp.*, 44:105–124, 1985.

[63] Beresford N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Englewood Cliffs, N.J., 1980.

[64] Claude Pommerell. *Solution of large unsymmetric systems of linear equations.* PhD thesis, Swiss Federal Institute of Technology, Zürich, 1992.

[65] Claude Pommerell and Wolfgang Fichtner. PILS: An iterative linear solver package for ill-conditioned systems. Technical Report 91/5, ETH Zürich, 1991.

[66] G. Radicati di Brozolo and Y. Robert. Vector and parallel CG-like algorithms for sparse non-symmetric systems. Technical Report 681-M, IMAG/TIM3, Grenoble, 1987.

[67] G. Radicati di Brozolo and Y. Robert. Parallel conjugate gradient-like algorithms for solving sparse non-symmetric systems on a vector multiprocessor. *Parallel Computing*, 11:223–239, 1989.

[68] Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Sci. Stat. Comput.*, 6:865–881, 1985.

[69] Y. Saad. Krylov subspace methods on supercomputers. Technical report, RIACS, Moffett Field, CA, September 1988.

[70] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14:461–469, 1993.

[71] Y. Saad and M. H. Schultz. Conjugate Gradient-like algorithms for solving nonsymmetric linear systems. *Math. of Comp.*, 44:417–424, 1985.

[72] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[73] J. J. F. M. Schlichting and H. A. van der Vorst. Solving 3D block bidiagonal linear systems on vector computers. *Journal of Comp. and Appl. Math.*, 27:323–330, 1989.

[74] Horst D. Simon. Direct sparse matrix methods. In James C. Almond and David M. Young, editors, *Modern Numerical Algorithms for Supercomputers*, pages 325–444, Austin, 1989. The University of Texas at Austin, Center for High Performance Computing.

[75] G. L. G. Sleijpen and H.A. Van der Vorst. Maintaining convergence properties of BICGSTAB methods in finite precision arithmetic. Technical report, University Utrecht, Department of Mathematics, 1994.

[76] G. L. G. Sleijpen and H.A. Van der Vorst. Reliable updated residuals in hybrid Bi-CG methods. *Preprint Nr. 886*, Dept. Math., University Utrecht, 1994.

[77] G. L. G. Sleijpen, H.A. Van der Vorst, and D. R. Fokkema. Bi-CGSTAB(ℓ) and other hybrid bi-cg methods. *Numerical Algorithms*, 7:75–109, 1994.

[78] G. L. G. Sleijpen and D. R. Fokkema. BICGSTAB(ℓ) for linear equations involving unsymmetric matrices with complex spectrum. *ETNA*, 1:11–32, 1993.

[79] P. Sonneveld. CGS: a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 10:36–52, 1989.

[80] A. van der Sluis and H. A. van der Vorst. The rate of convergence of conjugate gradients. *Numer. Math.*, 48:543–560, 1986.

[81] A. van der Sluis and H. A. van der Vorst. Numerical solution of large sparse linear algebraic systems arising from tomographic problems. In G. Nolet, editor, *Seismic Tomography*, chapter 3, pages 49–83. Reidel Pub. Comp., Dordrecht, 1987.

[82] A. van der Sluis and H. A. van der Vorst. SIRT- and CG-type methods for the iterative solution of sparse linear least-squares problems. *Lin. Alg. and its Appl.*, 130:257–302, 1990.

[83] H. A. van der Vorst. The convergence behavior of some iterative solution methods. In R. Gruber, J. Periaux, and R. P. Shaw, editors, *Proc. of the fifth Int.Symp. on Numer. Methods in Eng.*, 1989. vol 1.

[84] H. A. van der Vorst. The convergence behaviour of preconditioned CG and CG-S in the presence of rounding errors. In O. Axelsson and L. Yu. Kolotilina, editors, *Preconditioned Conjugate Gradient Methods*, Berlin, 1990. Nijmegen 1989, Springer Verlag. Lecture Notes in Mathematics 1457.

[85] H. A. van der Vorst. Experiences with parallel vector computers for sparse linear systems. *Supercomputer*, 37:28–35, 1990.

[86] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13:631–644, 1992.

[87] H. A. van der Vorst. Conjugate gradient type methods for nonsymmetric linear systems. In R. Beauwens and P. de Groen, editors, *Iterative Methods in Linear Algebra*, Amsterdam, 1992. IMACS Int. Symp., Brussels, Belgium, 2-4 April, 1991, North-Holland.

[88] H. A. van der Vorst and C. Vuik. The superlinear convergence behaviour of GMRES. *JCAM*, 48:327–341, 1993.

[89] H. A. van der Vorst and C. Vuik. GMRESR: A family of nested GMRES methods. *Num. Lin. Alg. with Appl.*, 1:369–386, 1994.

[90] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs N.J., 1962.

[91] P. K. W. Vinsome. ORTOMIN: an iterative method for solving sparse sets of simultaneous linear equations. In *Proc.Fourth Symposium on Reservoir Simulation*, pages 149–159. Society of Petroleum Engineers of AIME, 1976.

[92] H. F. Walker. Implementation of the GMRES method using Householder transforma- tions. *SIAM J. Sci. Stat. Comp.*, 9:152–163, 1988.

[93] O. Widlund. A Lanczos method for a class of nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, 15:801–812, 1978.

[94] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, 1965.

[95] L. Zhou and H. F. Walker. Residual smoothing techniques for iterative methods. *SIAM J. Sci. Comput.*, 15:297–312, 1994.

Structured Grid Solvers I

Accurate and Efficient Flow Solvers for 3D Applications on Structured Meshes

Norbert Kroll, Rolf Radespiel, Cord-C. Rossow

Institute for Design Aerodynamics
DLR, Lilienthalplatz 7, 38108 Braunschweig, Germany

SUMMARY

This lecture is devoted to the parallelization of blockstructured grid solvers for industrial applications. It is divided into two parts. Part I describes well established numerical algorithms with emphasis on spatial discretization and time stepping schemes. Attention is focused on the multigrid technique which is one of the most promising approach to improve the efficiency of numerical methods. Finally, several large-scale computations are shown which demonstrate the ability of current blockstructured flow solvers.

Part II of the lecture addresses various aspects of the parallelization of such flow solvers.

LIST OF SYMBOLS

c	speed of sound
$\vec{D}$	dissipative operator
e	internal energy per unit mass
E	specific total energy
$\bar{\bar{F}}^C, \bar{\bar{F}}^V$	inviscid and viscous part of flux tensor $\bar{\bar{F}}$
H	specific total enthalpy
K	heat conductivity
M	Mach number
$\vec{n}$	outward pointing normal
p	pressure
Pr	Prandtl number
$\vec{q}$	vector of cartesian velocities
$\vec{R}$	discrete flux balance
Re	Reynolds number
$\vec{S}$	surface vector
t	time
u,v,w	cartesian velocity components
V	control volume
$\vec{W}$	vector of conserved variables
α	angle of attack
γ	ratio of specific heats
$\tilde{\lambda}$	spectral radius of flux Jacobian
μ	nondimensional viscosity
φ	density

Indices

l	laminar
t	turbulent
∞	free stream
i,j,k	indices of grid node

1. INTRODUCTION

Numerical flow simulations have found their way into the aerodynamic design cycles of aerospace vehicles. Not only do these simulations reduce turn-around time and cost, but they also offer flow parameter variations which are not possible with wind tunnel testing. On the other hand, numerical simulations in aerodynamics are still an engineering challenge. The governing partial differential equations do not always represent a well-posed problem, that is, uniqueness and existence of a solution is usually not proven and it is difficult to formulate suitable initial and boundary conditions. Moreover, the existence of turbulence in the majority of relevant flow problems makes the direct solution of the governing unsteady Navier-Stokes equations impossible because the relevant scales vary too much. The problem may be circumvented by averaging the turbulent motion. This yields the Reynolds-averaged Navier-Stokes equations which can be solved if a turbulence model is provided for closure. Suitable turbulence models have been under investigation over the last 70 years, and the matter is still not solved to satisfaction. However, in the present lecture we will assume that the effect of turbulence can be described by adding a turbulent viscosity and heat conductivity to their laminar counterparts. Even then, flows over aerodynamic configurations display flow phenomena with very different scales and with highly nonlinear behavior. We mention here the laminar and turbulent boundary layers at very high Reynolds numbers and their interaction with shocks as an example. Numerical simulation of such flow problems often converge slowly because the discretized mathematical model is stiff.

Paper presented in an AGARD-FDP-VKI Special Course on "Parallel Computing in CFD", held at the VKI, Rhode-Saint-Genèse, Belgium, from 15-19 May 1995 and 16-20 October 1995 at NASA Ames, United States and published in R-807.

The present lecture describes well established techniques used for numerical simulations of complex aerodynamic flows based on blockstructured meshes. We restrict ourselves to problems with steady mean flow, that is, we want to obtain steady-state solutions of the Euler equations governing inviscid flows and of the Reynolds-averaged Navier-Stokes equations for viscous flows. In this paper attention is focused on the general description of the two major parts of the numerical method. These are the spatial discretization and time stepping algorithms. The parallelization issues of blockstructured flow solvers for industrial applications are treated in the second lecture [1].

With the spatial discretization of the governing equations we seek to obtain accurate solutions with as few as possible discrete points in the flow domain. Care must be taken to resolve all relevant flow phenomena, i.e. smoothly varying regions of inviscid flows, flow discontinuities as shocks and slip lines, and viscous layers which are governed by diffusion. Moreover, numerical analysis and well-known experience show that the choice of the spatial discretization also influences the convergence of the overall method to the desired steady-state.

Possibilities to improve convergence to steady-state solutions by improving or adding numerical techniques has attracted the work force of many researchers over the last 20 years. We will concentrate on one of the most promising approaches, which is called multigrid. The present state of the art in the use of multigrid for the solution of the hyperbolic flow equations with time-stepping schemes is described in detail, analyzed, and demonstrated with a variety of sample calculations.

Finally, we present several large-scale computations which demonstrate the usefulness of the efforts to improve accuracy and convergence of current flow solvers.

2. GOVERNING EQUATIONS

The most general description of the fluid flow is obtained from the time dependent compressible Navier-Stokes equations which express the conservation laws for mass, momentum and energy for viscous fluids. For turbulent flows the so-called Reynolds-averaged Navier-Stokes equations are exploited. They are derived from the Navier-Stokes equations by introducing a time-averaging procedure. The laws of motion are then expressed for the mean, time-averaged, turbulent quantities. By this means the equations for turbulent flows look the same as the equations for laminar flow.

The integral form of the three-dimensional Reynolds-averaged Navier-Stokes equations using nondimensional variables in a cartesian coordinate system can be written as

$$\frac{\partial}{\partial t}\int_V \vec{W}\,dv + \int_{\partial V} \bar{\bar{F}}\cdot\vec{n}\,ds = 0 \qquad (2.1)$$

where

$$\vec{W} = [\rho, \rho u, \rho v, \rho w, \rho E]^T$$

is the vector of conserved quantities with ρ ,u,v,w and E denoting the density, cartesian velocity components and specific total energy, respectively. V denotes an arbitrary control volume fixed in time and space with boundary ∂V and the outer normal $\vec{n}$. The total enthalpy is given by

$$H = E + p/\rho \qquad (2.2)$$

The flux tensor $\bar{\bar{F}}$ may be divided into its inviscid (convective) part $\bar{\bar{F}}^c$ and its viscous part $\bar{\bar{F}}^v$ as

$$\bar{\bar{F}} = \bar{\bar{F}}^c - \bar{\bar{F}}^v \qquad (2.3)$$

with

$$\bar{\bar{F}}^c = \begin{bmatrix} (\rho u)\vec{k}_x & + (\rho v)\vec{k}_y & + (\rho w)\vec{k}_z \\ (\rho u^2 + p)\vec{k}_x & + (\rho uv)\vec{k}_y & + (\rho uw)\vec{k}_z \\ (\rho uv)\vec{k}_x & + (\rho v^2 + p)\vec{k}_y & + (\rho vw)\vec{k}_z \\ (\rho uw)\vec{k}_x & + (\rho vw)\vec{k}_y & + (\rho w^2 + p)\vec{k}_z \\ (\rho uE + up)\vec{k}_x & + (\rho vE + vp)\vec{k}_y & + (\rho wE + wp)\vec{k}_z \end{bmatrix}$$

and

$$\bar{\bar{F}}^v = \begin{bmatrix} 0 & & \\ \sigma_{xx}\vec{k}_x & + \sigma_{xy}\vec{k}_y & + \sigma_{xz}\vec{k}_z \\ \sigma_{yx}\vec{k}_x & + \sigma_{yy}\vec{k}_y & + \sigma_{yz}\vec{k}_z \\ \sigma_{zx}\vec{k}_x & + \sigma_{zy}\vec{k}_y & + \sigma_{zz}\vec{k}_z \\ \psi_x & + \psi_y & + \psi_z \end{bmatrix}$$

with

$$\psi_x = (u\sigma_{xx} + v\sigma_{xy} + w\sigma_{xz} - q_x)\vec{k}_x$$
$$\psi_y = (u\sigma_{yx} + v\sigma_{yy} + w\sigma_{yz} - q_y)\vec{k}_y$$
$$\psi_z = (u\sigma_{zx} + v\sigma_{zy} + w\sigma_{zz} - q_z)\vec{k}_z$$

where $\vec{k}_x$, $\vec{k}_y$, $\vec{k}_z$ denote the cartesian coordinate directions. Assuming that air behaves as calorically perfect gas, the pressure is calculated by the equation of state

$$p = (\gamma - 1)\rho\left(E - \frac{u^2 + v^2 + w^2}{2}\right) \tag{2.4}$$

where γ denotes the ratio of specific heats. The temperature T is given by

$$T = p/\rho\,. \tag{2.5}$$

The elements of the shear-stress tensor and the heat-flux vector are given by the equations for Newtonian fluid

$$\begin{aligned}
\sigma_{xx} &= 2\mu u_x - 2/3\mu\,(u_x + v_y + w_z)\\
\sigma_{yy} &= 2\mu v_y - 2/3\mu\,(u_x + v_y + w_z)\\
\sigma_{zz} &= 2\mu w_z - 2/3\mu\,(u_x + v_y + w_z)\\
\sigma_{xy} &= \sigma_{yx} = \mu\,(u_y + v_x)\\
\sigma_{xz} &= \sigma_{zx} = \mu\,(u_z + w_x)\\
\sigma_{yz} &= \sigma_{zy} = \mu\,(v_z + w_y)\\
q_x &= -K\frac{\partial T}{\partial x}, q_y = -K\frac{\partial T}{\partial y}, q_z = -K\frac{\partial T}{\partial z}.
\end{aligned} \tag{2.6}$$

For laminar flow the nondimensional viscosity μ is assumed to follow the Sutherland law

$$\mu = \frac{\gamma^{1/2} M_\infty}{Re_\infty}\left(\frac{\bar{T}}{\bar{T}_\infty}\right)^{3/2}\frac{\bar{T}_\infty + 110k}{\bar{T} + 110k} \tag{2.7}$$

with M_∞, Re_∞ and $\bar{T}$ denoting the free stream Mach number, Reynolds number and the dimensional temperature, respectively. The heat conductivity K is given by

$$K = \frac{\gamma}{\gamma - 1}\frac{\mu}{Pr} \tag{2.8}$$

with Pr being the Prandtl number.

For turbulent flows, the laminar viscosity μ in eq. (2.7) is replaced by $\mu + \mu_t$ and μ/Pr in eq. (2.8) is replaced by $\mu/Pr + \mu_t/Pr_t$, where the eddy viscosity μ_t and the turbulent Prandtl Pr_t number are provided by a turbulence model. For the transonic airfoil calculations presented in this paper the algebraic turbulence model of Baldwin/Lomax [2] is used.

For hypersonic flow calculations it is assumed that air behaves as reacting air in thermochemical equilibrium. In this case a modified ratio of specific heats is used. Furthermore, the speed of sound is given by

$$c^2 = \left.\frac{\partial p}{\partial \rho}\right|_{e\,=\,\text{const}} + \frac{p}{\rho}\left.\frac{\partial p}{\partial e}\right|_{\rho\,=\,\text{const}} \tag{2.9}$$

where e is the internal energy per unit mass. For the calculation of the effective ratio of specific heats and for the partial derivatives of pressure in eq. (2.9), piecewise analytically defined functions [3] are used. These functions relate the pressure to both, the density and specific internal energy and take into account exitation of vibration and dissociation of O_2 and N_2 molecules. The temperature, viscosity and heat conductivity are similarly computed.

3. SPATIAL DISCRETIZATION SCHEME

The derivation of the conservation laws in integral form only requires the assumption that the density is twice continuously differentiable with respect to time. Therefore, in contrast to the differential form, the integral form of the governing equations does not impose any assumptions on the regularity of the solution. This is extremely important since discontinuities such as shock waves and slip lines occur in most of the relevant flow fields.

The discretization of the integral form of the conservation laws leads to finite element or finite volume methods. This paper focuses on the discussion of the finite volume approximation based on structured computational meshes.

3.1 Finite Volume Approximation

In finite volume methods the flow field is subdivided into a set of non-overlapping cells which cover the whole domain without gaps. On each cell the conservation laws in integral form are applied which also in the discrete formulation ensure the conservation of mass, momentum and energy. In general, the control volumes can have arbitrary shapes. With respect to computational efficiency, however, very often structured hexahedral cells are used for 3D calculations. For practical applications the control volumes are provided by a body-fitted mesh generated by grid generation packages using curvilinear coordinates (see Fig. 1). The only required data concerning the grid are the cartesian coordinates of the vertices. Hence, no global transformation of the governing equations into the curvilinear coordinate system is necessary.

Through the application of the integral form of the Navier-Stokes equations a discrete flux balance is obtained for each control volume which can be used to approximately determine the change of flow quantities with respect to time in particular points. Various finite volume formulations are known in the literature. They differ in the arrangement of control volumes and update points for the flow variables. The most frequently used schemes are the cell-centered, the cell-vertex and the node-centered approach. They are sketched in Fig. 2. For the node-centered and cell-vertex scheme the flow variables are associated with the cell vertices, whereas for the cell-centered scheme they are located at the center of the cell. Each of these schemes has advantages and disadvantages. For example, using a central discretization it can be shown that for stretched or screwed meshes the discretization errors from the cell-centered formulation is larger that those of the node-based

schemes [4]. However, for smooth meshes the spatial accuracy is the same for all schemes. On the other hand, numerical experience has shown that for high speed flows the cell-centered and the node-centered arrangement seem to be more suited, especially in combination with an upwind-biased discretization operator [5].

This paper focuses on the cell-vertex and node-centered formulation. In both cases the spatial discretization leads to an ordinary differential equation for the rate of change of the conservative flow variables in each grid point

$$\frac{d}{dt}\vec{W}_{i,j,k} = -\frac{1}{V_{i,j,k}}\left(\vec{R}^{c}_{i,j,k} - \vec{R}^{v}_{i,j,k}\right) \qquad (3.1)$$

where $\vec{R}^{c}_{i,j,k}$ and $\vec{R}^{v}_{i,j,k}$ represent the approximation of the inviscid and viscous net flux of mass, momentum and energy for a particular control volume arrangement with volume $V_{i,j,k}$ surrounding the grid node (i,j,k). The fluxes can be approximated using either central or upwind discretization operators. While classical central difference schemes perform admirably for inviscid sub-, trans- and even low supersonic flows, problems arise near strong discontinuities in high Mach number flows. Moreover, in recent papers (e.g. [6,7]) it has been pointed out that central schemes show deficiencies in the resolution of viscous flows due to the unsuited scaling of the scalar artificial dissipation implemented in most of the standard methods. This lack of a suitable high-resolution capability has been considered as a major problem of central schemes and has led to the development of a variety of upwind-biased algorithms. These schemes rely on local wave propagation theory for the differencing of the convective terms of the governing equations. This is not only important for capturing flow discontinuities but also it can lead to a high-resolution scheme for viscous flows, provided the linear waves are properly taken into account. In the following, various discretization schemes for the convective terms are discussed. The special merits and shortcomings of each scheme are highlighted. In discretizing the Navier-Stokes equations, virtually all schemes rely on a centered approximation of the viscous fluxes. A brief description is given at the end of this chapter.

3.2 Central Differencing with Scalar Dissipation

The central differencing of the convective terms of the governing equations discussed here is based on the cell-vertex scheme [4,8]. In this formulation the update of the flow variables in grid node (i,j,k) is a function of the discrete flux balances of the surrounding eight cells (see Fig. 3). The term $\vec{R}^{c}_{i,j,k}$ in eq.(3.1) can be expressed as

$$\vec{R}^{c}_{i,j,k} = \vec{G}_{i,j,k} + \vec{G}_{i-1,j,k} + \vec{G}_{i,j-1,k} + \vec{G}_{i-1,j-1,k} + \vec{G}_{i,j,k-1} + \vec{G}_{i-1,j,k-1} + \vec{G}_{i,j-1,k-1} + \vec{G}_{i-1,j-1,k-1} \qquad (3.2)$$

with $\vec{G}_{i,j,k}$ representing the convective flux for the mesh cell with vertices {(i+n,j,k), (i+n,j,k+1), (i+n,j+1,k), i+n,j+1,k+1), n=0,1}. Accordingly, the volume $V_{i,j,k}$ in eq.(3.1) represents the sum of the volumes of the corresponding cells surrounding the node (i,j,k).

The net flux $\vec{G}_{i,j,k}$ is given by the sum of the inviscid fluxes through all cell faces of the particular mesh cell (see Fig. 3.b)

$$\vec{G}_{i,j,k} = \vec{g}^{i}_{i+1,j,k} - \vec{g}^{i}_{i,j,k} + \vec{g}^{j}_{i,j+1,k} - \vec{g}^{j}_{i,j,k} + \vec{g}^{k}_{i,j,k+1} - \vec{g}^{k}_{i,j,k} \qquad (3.3)$$

where the flux through the cell face $S^{i}_{i+1,j,k}$ is evaluated using an arithmetric average of the flux quantities at the vertices. That is

$$\vec{g}^{i}_{i+1,j,k} = \frac{1}{4}\vec{S}^{i}_{i+1,j,k} \cdot \left(\bar{\bar{F}}^{c}_{i+1,j,k} + \bar{\bar{F}}^{c}_{i+1,j+1,k} + \bar{\bar{F}}^{c}_{i+1,j,k+1} + \bar{\bar{F}}^{c}_{i+1,j+1,k+1}\right) \qquad (3.4)$$

where $\vec{S}^{i}_{i+1,j,k}$ denotes the surface vector of cell face $S^{i}_{i+1,j,k}$ calculated by projecting the cell face on the corresponding coordinate surface.

A close inspection of eqs. (3.2) and (3.3) shows, that due to the fact that the fluxes across inner faces cancel, $\vec{R}^{1}_{i,j,k}$ represents the flux balance over a super cell formed by the eight neighboring cells of node (i,j,k). According to [4,8], the scheme is at least first order accurate, if the normal vector on each cell face is a smooth function with respect to grid refinement and if the cell faces do not degenerate to triangles. On smooth meshes the discretization is second-order accurate.

The finite volume discretization based on central averaging is not dissipative, which means that high frequency oscillations in the solution are not damped. In order to avoid these spurious oscillations, dissipative terms have to be explicitly introduced. In most central schemes the well known scalar dissipation model of Jameson et al [9] is implemented. It uses a blend of second and fourth differences of the flow variables. In order to preserve the conservation form of the numerical scheme, the artificial dissipative terms are introduced by adding dissipative fluxes to the semi-discrete system (3.1)

$$\frac{d}{dt}\vec{W}_{i,j,k} = -\frac{1}{V_{i,j,k}}\left(\vec{R}^{c}_{i,j,k} - \vec{R}^{v}_{i,j,k} - \vec{D}_{i,j,k}\right). \qquad (3.5)$$

The dissipative operator $\vec{D}_{i,j,k}$ is defined as

$$\vec{D}_{i,j,k} = \vec{d}_{i+\frac{1}{2},j,k} - \vec{d}_{i-\frac{1}{2},j,k} + \vec{d}_{i,j+\frac{1}{2},k} - \vec{d}_{i,j-\frac{1}{2},k} + \vec{d}_{i,j,k+\frac{1}{2}} - \vec{d}_{i,j,k-\frac{1}{2}} \tag{3.6}$$

where the dissipative flux $\vec{d}_{i+1/2,j,k}$ is given as

$$\vec{d}_{i+\frac{1}{2},j,k} = \alpha_{i+\frac{1}{2},j,k}\left\{\varepsilon^{(2)}_{i+\frac{1}{2},j,k}\left(\vec{W}_{i+1,j,k} - \vec{W}_{i,j,k}\right) - \varepsilon^{(4)}_{i+\frac{1}{2},j,k}\left(\vec{W}_{i+2,j,k} - 3\vec{W}_{i+1,j,k} + 3\vec{W}_{i,j,k} - \vec{W}_{i-1,j,k}\right)\right\}. \tag{3.7}$$

Here, $\varepsilon^{(2)}_{i+1/2,j,k}$ and $\varepsilon^{(4)}_{i+1/2,j,k}$ are adaptive coefficients designed to switch on enough dissipation where it is needed. The coefficient $\alpha_{i+1/2,j,k}$ is chosen such that the dissipative terms have a proper weightage. According to [9], the value is given as an average of the spectral radii of the flux Jacobians associated with the three curvilinear coordinates.

The coefficients $\varepsilon^{(2)}$ and $\varepsilon^{(4)}$ are adapted to the local flow gradients by

$$\varepsilon^{(2)}_{i+1/2,j,k} = k^{(2)}\max(\nu_{i+2,j,k}, \nu_{i+1,j,k}, \nu_{i,j,k}, \nu_{i-1,j,k}) \tag{3.8}$$

$$\varepsilon^{(4)}_{i+1/2,j,k} = \max(0, k^{(4)} - \varepsilon^{(2)}_{i+1/2,j,k}) \tag{3.9}$$

where $\nu_{i,j,k}$ is defined as

$$\nu_{i,j,k} = \frac{|p_{i+1,j,k} - 2p_{i,j,k} + p_{i-1,j,k}|}{|p_{i+1,j,k} + 2p_{i,j,k} + p_{i-1,j,k}|} \tag{3.10}$$

and $k^{(2)}$, $k^{(4)}$ are small constants. Typical values for $k^{(2)}$ and $k^{(4)}$ are 1/2 and 1/64, respectively. The dissipation operators in j-, and k-direction are defined in a similar manner.

The coefficient $\varepsilon^{(2)}$ is proportional to the second difference of pressure and therefore proportional to the square of the mesh size in smooth regions of the flow, while $\varepsilon^{(4)}$ is of order one. Since the operator in eq. (3.7) contains differences of the flow variables which are not divided by the mesh size, the dissipative flux $\vec{d}_{i+1/2,j,k}$ is of third order. However, in regions where the pressure changes rapidly, as in the case of shock waves, the term $\nu_{i,j,k}$ is of order one and with eq. (3.9) the third order difference operator in eq. (3.7) is switched off. The dissipation is then of first order and the central finite volume scheme behaves like a first-order accurate scheme. The sensitivity of the numerical solution with respect to the dissipation parameter has been studied in detail in [10,11]. Since the dissipative fluxes are formed by blended second and fourth differences, the evalutation of these terms near boundaries requires special care. The treatment at boundaries is described in [12] in more detail.

As mentioned above, the dissipation in each coordinate direction is scaled the same by the average of the spectral radii of all flux Jacobians. This leads to excessively large dissipation levels for cells with high-aspect ratios which are often required for accurate and efficient calculations of viscous flows. Therefore, according to Martinelli [13] the scaling factor of the dissipative term is adjusted for each coordinate direction taking into account a varying cell aspect ratio (see also [14]). The scaling function is

$$\alpha_{i+\frac{1}{2},j,k} = \tilde{\lambda}^{i}_{i+\frac{1}{2},j,k}\,\phi_{i+\frac{1}{2},j,k} \tag{3.11}$$

with

$$\phi_{i+\frac{1}{2},j,k} = 1 + \max\left(\left(\frac{\tilde{\lambda}^{j}_{i+\frac{1}{2},j,k}}{\tilde{\lambda}^{i}_{i+\frac{1}{2},j,k}}\right)^{\beta}, \left(\frac{\tilde{\lambda}^{k}_{i+\frac{1}{2},j,k}}{\tilde{\lambda}^{i}_{i+\frac{1}{2},j,k}}\right)^{\beta}\right) \tag{3.12}$$

where

$$\begin{aligned}(\tilde{\lambda}^{i})_{i+\frac{1}{2},j,k} &= \left|\vec{q}\cdot\vec{S}^{i}_{i+\frac{1}{2},j,k}\right| + c\left|\vec{S}^{i}_{i+\frac{1}{2},j,k}\right| \\ (\tilde{\lambda}^{j})_{i+\frac{1}{2},j,k} &= \left|\vec{q}\cdot\vec{S}^{j}_{i+\frac{1}{2},j,k}\right| + c\left|\vec{S}^{j}_{i+\frac{1}{2},j,k}\right| \\ (\tilde{\lambda}^{k})_{i+\frac{1}{2},j,k} &= \left|\vec{q}\cdot\vec{S}^{k}_{i+\frac{1}{2},j,k}\right| + c\left|\vec{S}^{k}_{i+\frac{1}{2},j,k}\right|\end{aligned} \tag{3.13}$$

are the spectral radii of the flux Jacobians in i-, j-, k-direction, respectively. $\vec{q}=[u,v,w]^T$ is the vector of cartesian velocities and c is the speed of sound. $\vec{S}^i$, $\vec{S}^j$, $\vec{S}^k$ are the cell face vectors associated with i-, j-, and k-direction of the body-fitted coordinate system. The use of the maximum function in the definition of ϕ is important for grids where $\tilde{\lambda}^j/\tilde{\lambda}^i$ and $\tilde{\lambda}^k/\tilde{\lambda}^i$ are very large and of same order of magnitude. In this case, if these ratios are summed rather than taking the maximum, too large dissipative terms are obtained, which will degrade the solution. It has been found that for the exponent β the choice β=0.5 yields a robust scheme.

The transonic turbulent flow over the RAE 2822 airfoil is used to demonstrate the capabilities of the method for high Reynolds number turbulent flows. The well-known test case M_∞=0.73, α=2.79° and Re_∞=6.5x10^6 has been considered. The accuracy of the central scheme with scalar dissipation is examined using a variation of the grid density. For this purpose a sequence of a coarse (193x33 points), medium (385x65 points) and fine grid (577x97 points) has been created. A C-grid topology has been selected with a first spacing of 10^{-5} chord lengths away from the wall. The calculations have been carried out with the Baldwin/Lomax turbulence model. The variation of the co-

efficients for lift, pressure drag and friction drag with number of mesh points N is presented in Fig. 4. The influence of the dissipation parameter of the second and fourth difference dissipation operator is also indicated. On coarse meshes, the discretization error is obviously dominated by the artificial dissipation. The integral values show a large variation. The fine meshes allow the extrapolation of the coefficients to their values for an infinitely fine mesh. For the mesh with 385x65 points, the predicted lift is within 1.5 percent, the pressure drag is within 3 counts and the friction drag is within 0.3 count of the extrapolated values. For the fine mesh with 577x97 points, the predicted lift is within 0.5 percent, the pressure drag within 1 count and the skin friction drag within 0.1 count. Fig. 5 shows pressure and skin friction distributions for different grid densities. The experimental values [15] are also included. The main features of the flow are essentially captured on the medium mesh. The differences between medium and fine mesh are small.

In Fig. 6 the transonic flow around the ONERA-M6 wing is considered. The computational domain is discretized using a C-type topology in the streamwise direction and an O-type topology in the spanwise direction with 289x65x45 points. A somewhat coarser mesh has also been used in order to indicate the influence of the grid density for three-dimensional viscous flows. The commonly used test case $M_\infty = 0.84$, $\alpha = 3.06^\circ$ and $Re_\infty = 11 \times 10^6$ has been considered. Here, again the Baldwin/Lomax turbulence model has been used. The pressure distributions along several spanwise stations of the wing are displayed in Fig. 3.6. The results of the fine mesh agree well with those from the coarser mesh and with experimental data [16].

A comprehensive validation of the central cell-vertex scheme with scalar dissipation can be found in [4, 14, 17, 18].

3.3 Central Differencing with Matrix Dissipation

As shown in the literature and indicated by the results above, it is possible to obtain grid-converged solutions for transonic viscous flows with central schemes, provided sufficiently fine meshes are used for the computations. However, for efficiency reasons, especially for 3-D applications, the accuracy of the solution needs to be improved on a given grid, in order to reduce the number of grid points required for obtaining a specified level of accuracy. The major drawback of standard central schemes, as the one presented above, is the scalar form of the artificial viscosity. In this approach the dissipation of each conservation equation is scaled the same. The spectral radius of the flux Jacobian associated with the corresponding coordinate direction is employed as the scaling factor. As suggested by Turkel [19] and Swanson and Turkel [20] the central discretization can be improved by replacing the scalar dissipation by a matrix-valued dissipation using ideas from the concept of upwind schemes. In this case, the dissipation in a particular coordinate direction for each equation is scaled by the specific eigenvalue associated with the corresponding flux Jacobian matrix.

In the case of matrix-valued dissipation the dissipative flux $\vec{d}_{i+1/2,j,k}$ of eq. (3.6) through interface i+1/2 is defined as

$$\vec{d}_{i+\frac{1}{2},j,k} = |A|_{i+\frac{1}{2},j,k} \left\{ \varepsilon^{(2)}_{i+\frac{1}{2},j,k} \left(\vec{W}_{i+1,j,k} - \vec{W}_{i,j,k} \right) - \varepsilon^{(4)}_{i+\frac{1}{2},j,k} \left(\vec{W}_{i+2,j,k} - 3\vec{W}_{i+1,j,k} + 3\vec{W}_{i,j,k} - \vec{W}_{i-1,j,k} \right) \right\}. \quad (3.14)$$

In contrast to eq. (3.7), the differences of the flow quantities are now scaled by a matrix which is given by

$$|A|_{i+\frac{1}{2},j,k} = \left(T |\Lambda_A| (T)^{-1} \right)_{i+\frac{1}{2},j,k} \quad (3.15)$$

with T and $(T)^{-1}$ being the right and left eigenvector matrices of the flux Jacobian A associated with the i-direction of the curvilinear coordinate system. $|\Lambda_A|$ denotes a diagonal matrix, where the elements are the absolute values of the eigenvalues of A. The eigenvalues of A are given by

$$\begin{aligned} \lambda_n &= \vec{q} \cdot \vec{S}^i \quad , n = 1,2,3 \\ \lambda_4 &= \vec{q} \cdot \vec{S}^i + c\left|\vec{S}^i\right| \\ \lambda_5 &= \vec{q} \cdot \vec{S}^i - c\left|\vec{S}^i\right| . \end{aligned} \quad (3.16)$$

The matrices in eq. (3.15) are evaluated at the interface i+1/2 using simple averages of the flow quantities $\vec{W}$ at grid nodes (i,j,k) and (i+1,j,k). According to [19,20], by taking advantage of the special form of the elements of $|A|$, the matrix vector products occurring in eq. (3.14) can be replaced by the products of row and column vectors. This leads to a simpler and more efficient procedure for the evaluation of the dissipative flux. For details see also [21]. The parameter $\varepsilon^{(2)}$ and $\varepsilon^{(4)}$ are essentially the same as in the case of the scalar dissipation. Also here, typical values of the coefficients $k^{(2)}$ and $k^{(4)}$ are 1/2 and 1/64, respectively. Note that, if $|A|$ is replaced by its spectral radius, then the usual scalar dissipation outlined above is obtained.

As can be seen in eqs. (3.14) and (3.15), for each flow equation the dissipation is scaled by the corresponding eigenvalue. In practice, however, the eigenvalues as given in eq. (3.16) can not be used. At stagnation points the eigenvalues λ_1, λ_2 and λ_3 vanish, whereas near sonic lines (M=1) the eigenvalue λ_4 or λ_5 approaches zero. It is well

known, that for a central difference scheme zero artificial viscosity can create numerical difficulties. Therefore, the eigenvalues are limited in the following manner according to [19,20]

$$\begin{aligned} |\tilde{\lambda}_n| &= \max\left(|\lambda_n|, V_l\,\rho(A)\right), n = 1,2,3 \\ |\tilde{\lambda}_4| &= \max\left(|\lambda_4|, V_n\,\rho(A)\right) \\ |\tilde{\lambda}_5| &= \max\left(|\lambda_5|, V_n\,\rho(A)\right) \end{aligned} \tag{3.17}$$

where V_l and V_n are small coefficients which limit the eigenvalues associated with the linear and nonlinear characteristic fields to a minimum value that is a fraction of the spectral radius $\rho(A)$ (largest eigenvalue) of A. The parameters V_l and V_n are determined through numerical experiments such that shocks are captured without spurious oscillations and good convergence behavior is still maintained. Typical values are $0.3 \le V_l \le 0.6$ and $V_n = 0.4$ (see [21]). It should be noted, that in the case of $V_l = V_n = 1$ the scalar form of the artificial dissipation is recovered.

The improvement of the accuracy of the central scheme for a given grid by using a matrix-valued dissipation instead of a scalar dissipation is demonstrated for the turbulent flow around the RAE 2822 airfoil [21]. Fig. 7 shows the comparison of the surface pressure distribution for scalar and matrix dissipation. The calculations have been carried out on C-grids with 160x32, 320x64 and 640x128 cells. It is obvious that the quality of the solution obtained with the scalar artificial viscosity model on the 320x64 and 640x128 cells can already be achieved with the matrix dissipation on the next coarser grid, that is on the 160x32 and 320x64 grid, respectively. This is underlined in Fig. 8 where the skin friction distributions are compared. Fig. 9 shows the variation of the global force coefficients with number of mesh points N=NX*NY. In contrast to the scalar dissipation model, the matrix dissipation provides a second-order scheme which is indicated by the linear dependency of the integral values with respect to the total number of grid points. Also this figure shows that the results calculated with the scalar dissipation model on the 640x128 grid is already obtained on the 320x64 grid by using the matrix dissipation approach. On the other hand, on a given grid the matrix dissipation model requires additional computational costs due to the increased complexity. Furthermore, it shows a degeneration of the convergence behavior to steady state [20,21]. However, for a specified level of accuracy the central scheme with matrix dissipation is more cost-effective than with the scalar dissipation, since coarser grids can be used. Thus, e. g. for the two-dimensional turbulent flow past an airfoil the computational effort could be reduced by a factor of 2-3.

3.4 Flux Difference Splitting

Numerical analysis of high speed flow often involves the resolution of strong shocks producing pressure jumps of considerable strength, complex shock-shock interactions, expansion fans and contact discontinuities as well as regions of highly expanded flow as e. g. on the leeside of reentry vehicles at high angle of attack. For such flows, certain aspects of the numerical methods which perform well for sub- and transonic applications have to be modified, in order to facilitate robust, efficient and accurate calculations. Classical central difference schemes are not well suited to such flows, since they require excessive artificial damping in order to suppress high frequency oscillations which may grow unbounded in the vicinity of strong shocks. This has led to the development of a variety of upwind schemes. These schemes rely on local wave propagation theory for the differencing of the convective terms of the governing equations throughout the domain. Prominent representatives of this class of algorithms are schemes based on the 'Flux Difference Splitting' (e. g. [22,23] and the' Flux Vector Splitting' (e. g. [24,25] concept.

Out of the class of flux difference split methods we have focused on the upwind TVD discretization according to [23,26]. This scheme is based on the approximate Riemann solver of Roe [22] and uses the modified flux approach of Harten [23] for second-order accuracy. Upwinding in multi-dimensions is performed by applying the one-dimensional operator successively in each coordinate direction. In order to implement an approximate Riemann solver in the framework of a node-based finite volume scheme [5,27], control volumes are used which are defined by connecting the cell centers of the original cell (see Fig. 10). The convective flux $\vec{R}^c_{i,j,k}$ for the control volume $V_{i,j,k}$ in eq. (3.1) is then approximated by

$$\begin{aligned} \vec{R}^c_{i,j,k} = {} & \vec{R}^c_{i+\frac{1}{2},j,k} - \vec{R}^c_{i-\frac{1}{2},j,k} + \vec{R}^c_{i,j+\frac{1}{2},k} \\ & - \vec{R}^c_{i,j-\frac{1}{2},k} + \vec{R}_{i,j,k+\frac{1}{2}} - \vec{R}^c_{i,j,k-\frac{1}{2}}. \end{aligned} \tag{3.18}$$

The flux $\vec{R}^c_{i+\frac{1}{2},j,k}$ through cell face i+1/2 is given as

$$\begin{aligned} \vec{R}^c_{i+\frac{1}{2},j,k} = {} & \frac{1}{2}\left(\bar{\bar{F}}_{i+1,j,k} + \bar{\bar{F}}_{i,j,k}\right)\cdot \vec{S}^i_{i+\frac{1}{2},j,k} \\ & + \frac{1}{2}\, T_{i+\frac{1}{2},j,k}\, \vec{Q}_{i+\frac{1}{2},j,k} \end{aligned} \tag{3.19}$$

with T denoting the right eigenvector matrix of the flux Jacobian in the i-direction of the curvilinear coordinate system. Eq. (3.19) separates the inviscid numerical flux into the sum of an averaged term corresponding to central differencing and a dissipative term, which adapts the discretization stencil in accordance with local wave propagation. Ac-

cording to Yee and Harten [23], the n'th component q^n of the flux function $\vec{Q}$, is given as

$$q^n_{i+\frac{1}{2},j,k} = \frac{1}{2}\Psi\left((\lambda_n)_{i+\frac{1}{2},j,k}\right)(h^n_{i+1,j,k} + h^n_{i,j,k}) - \Psi\left((\lambda_n)_{i+\frac{1}{2},j,k} + \gamma^n_{i+\frac{1}{2},j,k}\right)\alpha^n_{i+\frac{1}{2},j,k} \tag{3.20}$$

where λ_n represents the n'th eigenvalue of the transformed flux Jacobian in i-direction, $\alpha^n_{i+\frac{1}{2},j,k}$ denotes the difference of characteristic variables

$$\alpha^n_{i+\frac{1}{2},j,k} = (T)^{-1}_{i+\frac{1}{2},j,k}\left(\vec{W}_{i+1,j,k} - \vec{W}_{i,j,k}\right) \tag{3.21}$$

and

$$\gamma^n_{i+\frac{1}{2},j,k} = \frac{1}{2}\psi\left((\lambda_n)_{i+\frac{1}{2},j,k}\right)\cdot\begin{cases}\dfrac{h^n_{i+1,j,k} - h^n_{i,j,k}}{\alpha^n_{i+\frac{1}{2},j,k}} & \text{if } \alpha^n_{i+\frac{1}{2},j,k} \neq 0 \\ 0 & \text{if } \alpha^n_{i+\frac{1}{2},j,k} = 0.\end{cases} \tag{3.22}$$

The function ψ, often called entropy function, prevents the scheme from violating the entropy condition when the wave speeds, λ_n, vanish. According to Harten, this function is given by

$$\psi(\lambda_n) = \begin{cases}|\lambda_n| & \text{if } |\lambda_n| \geq \delta \\ \dfrac{|\lambda_n|^2 + \delta^2}{2\delta} & \text{if } |\lambda_n| < \delta\end{cases} \tag{3.23}$$

where $0<\delta<0.5$ is a suitable chosen parameter. The term h^n in eqs. (3.20) and (3.21) represents a limiter function which brings the scheme to second order. Many limiter functions have been proposed in the literature (see e. g. [26,28]). In most of our calculations the function

$$h^n_{i,j,k} = \frac{\alpha^n_{i-\frac{1}{2},j,k}\left[\left(\alpha^n_{i+\frac{1}{2},j,k}\right)^2 + \varepsilon\right] + \alpha^n_{i+\frac{1}{2},j,k}\left[\left(\alpha^n_{i-\frac{1}{2},j,k}\right)^2 + \varepsilon\right]}{\left(\alpha^n_{i-\frac{1}{2},j,k}\right)^2 + \left(\alpha^n_{i+\frac{1}{2},j,k}\right)^2 + 2\varepsilon} \tag{3.24}$$

is used where $\varepsilon \geq 0$ is a small constant to prevent division by zero. The quantities at face i+1/2 are evaluated using the Roe averaged state [22] involving the values at grid nodes (i,j,k) and (i+1,j,k). The fluxes through the other cell faces are evaluated in a similar manner.

Setting the limiter h^n identically to zero reduces this method to Roe's first-order flux difference method. It has been shown that the scheme is TVD (Total Variation Diminishing) for one-dimensional nonlinear hyperbolic scalar equations and for linear constant coefficient systems. It is formally second-order accurate except at shocks where due to the limiter the accuracy is reduced to first-order.

For viscous flows the entropy correction, eq. (3.23), has to be carefully designed. The shear layers along solid walls are numerically smeared, if an entropy correction is applied to the eigenvalues associated with the convective waves. On the other hand, if cells with high-aspect ratios are present, additional support for damping in the direction of the long side of a cell is needed in regions of low velocities, such as stagnation points. Therefore, as proposed by Radespiel and Swanson [29], the correction is constructed as a function of the cell aspect ratio. In i-direction the correction for the linear waves, n=1,2,3 (see eq. (3.16)) is defined as

$$\gamma(\lambda^i_n) = \begin{cases}|\lambda^i_n| & \text{if } |\lambda^i_n| \geq \tilde{\delta} \\ \beta\dfrac{|\lambda^i_n|^2 + \tilde{\delta}^2}{2\tilde{\delta}} + (1-\beta)|\lambda^i_n| & \text{if } |\lambda^i_n| < \tilde{\delta}\end{cases} \tag{3.25}$$

and for the acoustic waves, n=4,5 it is given by

$$\gamma(\lambda^i_n) = \begin{cases}|\lambda^i_n| & \text{if } |\lambda^i_n| \geq \tilde{\delta} \\ \dfrac{|\lambda^i_n|^2 + \tilde{\delta}^2}{2\tilde{\delta}} & \text{if } |\lambda^i_n| < \tilde{\delta}.\end{cases} \tag{3.26}$$

The parameter $\tilde{\delta}$ is given according to Müller [30]

$$\tilde{\delta} = \delta\bar{\lambda}^i\left(1 + \max\left(\left(\frac{\tilde{\lambda}^j}{\bar{\lambda}^i}\right)^\omega, \left(\frac{\tilde{\lambda}^k}{\bar{\lambda}^i}\right)^\omega\right)\right) \tag{3.27}$$

where $\tilde{\lambda}^i, \tilde{\lambda}^j, \tilde{\lambda}^k$ are the spectral radii of the flux Jacobians in i-, j-, k-direction, respectively and $0<\omega<1$. The blending coefficient, β, accounts for the cell aspect ratio. It is given as

$$\beta = \max\left(1 - \max\left(\frac{\tilde{\lambda}^i}{\tilde{\lambda}^j}, \frac{\tilde{\lambda}^i}{\tilde{\lambda}^k}\right)\right). \tag{3.28}$$

It has been shown in [29] that a wide range of flow problems can be solved accurately with a single set of parameters, that is $\delta=0.25$ and $\omega=0.3$.

In the following some results obtained with the TVD scheme are shown in order to demonstrate the capability of the method. Firstly, in Figs. 11-12 the accuracy is displayed

for the turbulent transonic flow around the RAE 2822 airfoil. Calculations have been carried out on a coarse (80x16 cells), medium (160x32 cells) and fine grid (320x64) with C-grid topology. Fig. 11 shows the pressure distributions and skin friction distributions along the surface for the three different grids. It is seen that with the TVD scheme a grid-converged solution is obtained. The difference between the medium and fine grid solution is very small. The improved force coefficients obtained with the upwind TVD scheme compared to the classical central scheme of chapter (3.2) is shown in Fig. 12 where lift and drag values are plotted as a function of the inverse of the total number of cells.

Next the laminar flow over the NACA 0012 airfoil at M_∞=25 and α=25° is chosen as a test case to demonstrate that the method is able to handle very strong shock waves and highly expanded flow. Fig. 13 shows the 250x80 mesh. The numerical solution is represented in Figs. 14-17. The streamlines in Fig. 15 feature a large separated flow region with two distinct vortices. The difficulties in resolving this highly separated flow are illustrated by a comparison of the distribution of skin friction and Stanton number along the airfoil obtained from meshes with different fine grids. It is obvious, that the grid with 129x41 mesh points is still too coarse to resolve the separated flow region.

The third viscous test case presented here is the hypersonic laminar flow past a 15° compression ramp. With onflow conditions M_∞=11.68, Re_c=2.47x10^5, T_∞=65K and T_w/T_∞=4.604 it corresponds to case III.4 of the Workshop on Hypersonic Flows for Reentry Problems, Part II, held in Antibes, France, 1991 [31]. Results have been obtained for three successive grids [32]. The Mach contours of the fine grid with 288x224 cells are shown in Fig. 18. The pressure coefficient, skin friction and Stanton number are displayed in Figs. 19-21. It is seen that almost identical solutions are obtained on the medium and fine meshes. Experimental data of Holden [33] are also plotted. The comparison of experimental and theoretical results shows that the calculated separation extent is somewhat larger than the experimental result. The discrepancies may be attributed to the fact that the experimental data contain 3D effects which are not modeled in the computation.

As a last test case, Edney's Type IV shock-interference flow [34] is investigated. This flow problem demands the solver to resolve many rigorous flow features (see Fig. 22) and it points out significant differences in the accuracy and convergence behavior of the numerical methods [35]. Fig. 23 shows the Mach contours for the TVD scheme and the classical central scheme with scalar dissipation. Inviscid results have been obtained for a coarse grid with 60x40 cells and a fine grid with 120x80 cells as shown in Fig. 23. The results demonstrate the superior resolution of the upwind TVD scheme. As anticipated, the additional dissipation required for the central scheme to suppress oscillations near shocks, considerably smears both the impinging shock and the distorted bow shock. The upwind method sharply resolves these features. Moreover, even on the coarse mesh the internal structure of the field is captured including the imbedded shock and terminating normal shock. In contrast to that, the fine grid solution obtained with the central difference scheme still shows a lack of structure.

Many two- and three-dimensional applications [27,29,37] have shown, that the upwind TVD scheme provides an accurate discretization for inviscid and viscous flows. Based on our experience, however, flux difference split methods are of difficult use with respect to robustness and parameter sensitivities for hypersonic flow fields with strong expansions into regions of low pressure and low density as e.g. on the leeside of re-entry vehicles at high angle of attack. Moreover, the extension of flux difference split methods to non-equilibrium flows is rather complex.

3.5 Flux Vector Splitting

Upwind methods based on the flux vector splitting concept have shown to be efficient and robust schemes for inviscid flows. However, often they exaggerate diffusive effects which take place in shear and boundary layers. Consequently, substantial effort has been put on the improvement of flux vector split methods for viscous flows [38,39,40].

A remarkably simple upwind flux vector splitting scheme has been introduced by Liou and Steffen [38,40]. It treats the convective and pressure terms of the flux function separately. The convective quantities are extrapolated to the cell interface in an upwind-biased manner using a properly defined cell face advection Mach number. Accordingly, the scheme is called *Advection Upstream Splitting Method* (AUSM). Results for simple flow problems given by Liou [39,41] have shown that AUSM retains the robustness and efficiency of the flux vector splitting schemes but it achieves the high accuracy attributed to schemes based on the flux difference splitting concept. The computational effort for the flux evaluation is only linearly proportional to the number of unknowns, as in the case of central differencing. Furthermore, the scheme can be easily extended to real gas calculations.The application to various relevant flow problems, however, has shown [36,42,43,44] that the original flux vector splitting method of Liou and Steffen has several deficiencies. It locally produces pressure oscillations in the vicinity of shocks. Furthermore, the scheme has a poor damping behavior for small Mach numbers which leads to spurious oscillations in the solution and affects the ability of the scheme to capture flows aligned with the grid coordinates.

In the present paper several modifications to the original advection upstream splitting method of Liou and Steffen

are proposed which substantially improve the scheme's ability to predict viscous flows accurately. In particular, a hybrid method is introduced which switches from AUSM to van Leer scheme at shock waves. This ensures the well-known sharp and clean shock capturing capability of the van Leer scheme and the high resolution of slip lines and contact discontinuities through AUSM. An adaptive dissipation is introduced in order to achieve sufficient numerical damping in cases of adverse grid situations and flow alignment. Furthermore, the MUSCL implementation for higher-order accuracy is modified to allow a more accurate scaling of the numerical dissipation in boundary layers where the contravariant Mach number is usually small in the wall-normal direction. The improved accuracy of the modified scheme is demonstrated by the calculation of two- and three-dimensional inviscid and viscous flows.

As shown in [39,36], the discrete inviscid flux $\vec{R}^{c}_{i+1/2,j,k}$ through cell face i+1/2 (see eq.(3.18)) can be interpreted as a sum of a Mach number weighted average of the left (L) and right (R) state at the cell face i+1/2 (see Fig. 3.10) and a scalar dissipative term. It reads

$$\vec{R}^{c}_{i+\frac{1}{2},j,k} = \left|\vec{S}^{i}\right|_{i+\frac{1}{2},j,k}\left(\frac{1}{2}M_{i+\frac{1}{2},j,k}\left(\begin{bmatrix}\rho c\\ \rho cu\\ \rho cv\\ \rho cw\\ \rho H\end{bmatrix}_{L}+\begin{bmatrix}\rho c\\ \rho cu\\ \rho cv\\ \rho cw\\ \rho H\end{bmatrix}_{R}\right)-\right.$$

$$\left.-\frac{1}{2}\Phi_{i+\frac{1}{2},j,k}\left(\begin{bmatrix}\rho c\\ \rho cu\\ \rho cv\\ \rho cw\\ \rho H\end{bmatrix}_{R}-\begin{bmatrix}\rho c\\ \rho cu\\ \rho cv\\ \rho cw\\ \rho H\end{bmatrix}_{L}\right)\right)+\begin{bmatrix}0\\ s_x p\\ s_y p\\ s_z p\\ 0\end{bmatrix}_{i+\frac{1}{2},j,k} \qquad (3.29)$$

where

$$\vec{S}^{i}_{i+\frac{1}{2},j,k} = \left[\,(s^{i}_{x}, s^{i}_{y}, s^{i}_{z})\,\right]^{T}_{i+\frac{1}{2},j,k} \qquad (3.30)$$

denotes the surface vector normal to the cell face $i+1/2$. The quantity c represents the speed of sound. $M_{i+1/2,j,k}$ denotes the advection Mach number at the cell face $i+1/2$ which is calculated according to [39] as

$$M_{i+\frac{1}{2},j,k} = M^{p}_{L} + M^{m}_{R} \qquad (3.31)$$

where the split Mach numbers $M^{p/m}$ are defined as [25]

$$M^{p} = \begin{cases} M & \text{if } M \geq 1 \\ \frac{1}{4}(M+1)^2 & \text{if } |M| < 1 \\ 0 & \text{if } M \leq -1 \end{cases} \qquad (3.32)$$

$$M^{m} = \begin{cases} 0 & \text{if } M \geq 1 \\ -\frac{1}{4}(M-1)^2 & \text{if } |M| < 1 \\ M & \text{if } M \leq -1 \end{cases} \qquad (3.32)$$

M_L and M_R denote the Mach number associated with the left and right state, respectively. The advection Mach number is given by

$$M = \frac{1}{\left|\vec{S}^{i}\right|}\frac{((s^{i}_{x}u + s^{i}_{y}v + s^{i}_{z}w))}{c}. \qquad (3.33)$$

The pressure p at cell face $i+1/2$ is calculated in a similar way as

$$p_{i+\frac{1}{2},j,k} = p^{p}_{L} + p^{m}_{L} \qquad (3.34)$$

where $p^{p/m}$ denote the split pressure defined according to [25]

$$p^{p} = \begin{cases} p & \text{if } M \geq 1 \\ \frac{1}{4}p(M+1)^2(2-M) & \text{if } |M| < 1 \\ 0 & \text{if } M \leq -1 \end{cases} \qquad (3.35)$$

$$p^{m} = \begin{cases} 0 & \text{if } M \geq 1 \\ \frac{1}{4}p(M-1)^2(2+M) & \text{if } |M| < 1 \\ p & \text{if } M \leq -1 \end{cases}$$

The definition of the dissipative term Φ determines the particular flux vector splitting formulation. A hybrid scheme is proposed here [45], which combines the van Leer scheme and the scheme of Liou and Steffen (AUSM). It reads

$$\phi_{i+\frac{1}{2},j,k} = (1-\omega)\cdot\phi^{VL}_{i+\frac{1}{2},j,k} + \omega\cdot\phi^{modAUSM}_{i+\frac{1}{2},j,k} \qquad (3.36)$$

with

$$\phi^{VL}_{i+\frac{1}{2},j,k} =$$

$$\begin{cases} \left|M_{i+\frac{1}{2},j,k}\right| & \text{if } \left|M_{i+\frac{1}{2},j,k}\right| \geq 1 \\ \left|M_{i+\frac{1}{2},j,k}\right| + \frac{1}{2}(M_R - 1)^2 & \text{if } 0 \leq M_{i+\frac{1}{2},j,k} < 1 \\ \left|M_{i+\frac{1}{2},j,k}\right| + \frac{1}{2}(M_L + 1)^2 & \text{if } -1 < M_{i+\frac{1}{2},j,k} \leq 0 \end{cases} \quad (3.37)$$

and

$$\phi^{modAUSM}_{i+\frac{1}{2},j,k} = \begin{cases} \left|M_{i+\frac{1}{2},j,k}\right| & \text{if } \left|M_{i+\frac{1}{2},j,k}\right| > \tilde{\delta} \\ \dfrac{\left(M_{i+\frac{1}{2},j,k}\right)^2 + \tilde{\delta}^2}{2\tilde{\delta}} & \text{if } \left|M_{i+\frac{1}{2},j,k}\right| \leq \tilde{\delta} \end{cases} \quad (3.38)$$

where $\tilde{\delta}$ is a small parameter $0 < \tilde{\delta} \leq 0.5$ and ω is a constant $0 \leq \omega \leq 1$.

The above equations clearly show that for a supersonic cell face Mach number the hybrid scheme represents a pure upwind discretization, using either the left or right state for the convective and pressure terms, depending on the sign of the Mach number. For ω=0 the method reduces to the classical van Leer flux vector splitting scheme. In the case of ω=1 and $\tilde{\delta}$=0 the original AUSM developed by Liou and Steffen is recovered. Comparing both fluxes it is obvious that the van Leer scheme is more dissipative than AUSM ($\tilde{\delta}$=0). It has an additional Mach number scaled dissipative term which does not vanish even for M=0. Consequently, the van Leer scheme is more robust but less accurate than the original scheme of Liou and Steffen, especially for viscous flow calculations.

The hybrid flux has been introduced in order to ensure both, the clean and sharp shock resolution of the van Leer scheme and the low diffusive solution of AUSM in smooth regions. This is realized by relating the parameter ω to the second difference of the pressure,

$$\omega = \max(\mu_{i,j,k}\mu_{i+1,j,k}),$$

$$\nu_{i,j,k} = \max\left(1 - \alpha\left|\frac{p_{i-1,j,k} - 2p_{i,j,k} + p_{i+1,j,k}}{p_{i-1,j,k} + 2p_{i,j,k} + p_{i+1,j,k}}\right|, 0\right), \quad (3.39)$$

$$\alpha = 0(5).$$

The value of ω is 1 in smooth regions and switches to 0 in the vicinity of shocks. Moreover, in order to improve the damping behavior of the original AUSM ($\tilde{\delta}$=0) in regions with adverse grid situations and flow alignment, its dissipative term has been modified. As it can be seen in eq.(3.38), controlled dissipation is locally introduced for small advection Mach numbers, preventing the dissipative term from approaching zero as the Mach number tends to zero. In Fig. 24 the dissipative term ϕ is plotted as a function of Mach number. Note, that for simplicity $M_L \sim M_R$ is assumed, which is valid at least in the vicinity of M=0 on a sufficiently fine computational grid.

Accurate and efficient calculations of viscous flows require computational grids with high-aspect ratio cells. Therefore, the dissipation term of the improved AUSM for small advection Mach numbers (eq. (3.38)) has to be properly scaled in order to avoid smearing of the shear layers in wall-normal direction. As mentioned in [42], this is realized by defining the parameter $\tilde{\delta}$ in eq. (3.38) not as a constant but as a function of the cell metric

$$\tilde{\delta}_{i+\frac{1}{2},j,k} = \delta \cdot \beta_{i+\frac{1}{2},j,k} \quad (3.40)$$

where δ is a small constant, $0 \leq \delta \leq 0.5$, and β is a scaling function. It may be given by

$$\beta_{i+\frac{1}{2},j,k} = 1 - \min\left\{\max\left[\frac{\left|\vec{S}^i\right|}{\left|\vec{S}^j\right|}, \frac{\left|\vec{S}^i\right|}{\left|\vec{S}^k\right|}\right], 1\right\}. \quad (3.41)$$

In the above, $\left|\vec{S}^i\right|, \left|\vec{S}^j\right|, \left|\vec{S}^k\right|$ represent the surface areas associated with the i-,j-,k-direction of the body-fitted coordinate system, respectively. The scaling function β in j- and k-direction is defined in a similar way. With this scaling, controlled adaptive dissipation can be introduced, which on the one hand improves the damping behavior of AUSM in adverse grid situations but on the other hand does not spoil the accuracy of the method for boundary layer calculations. It is obvious from eqs. (3.38)-(3.41) that additional dissipation as a function of the grid aspect ratio is fed in only along the long sides of the cell, that is if the cell face area $\left|\vec{S}^i\right|$ is smaller than the areas $\left|\vec{S}^j\right|$ and $\left|\vec{S}^k\right|$. In the contrary, if the cell face area $\left|\vec{S}^i\right|$ is larger than areas $\left|\vec{S}^j\right|$ and $\left|\vec{S}^k\right|$, as typical in wall-normal direction, the original non smearing dissipation of AUSM is recovered.

An alternate scaling function is given by

$$\beta_{i+\frac{1}{2},j,k} = \min\left\{\max\left[\frac{\left|\vec{S}^j\right|}{\left|\vec{S}^i\right|}, \frac{\left|\vec{S}^k\right|}{\left|\vec{S}^i\right|}\right], 1\right\}. \quad (3.42)$$

This function leads to a constant $\tilde{\delta} = \delta$ along the long side of the cell, whereas in the wall-normal direction the dissipative coefficient is weighted by the cell aspect ratio. It is obvious that along the short cell face the dissipation is reduced as the cell aspect ratio increases.

Another possibility for the scaling of the adaptive dissipation is to use the local flow quantities instead of the metric terms. In this case the function β is defined as

$$\beta_{i+\frac{1}{2},j,k} = \min\left\{\max\left[\frac{|\tilde{\lambda}^j|}{|\tilde{\lambda}^i|},\frac{|\tilde{\lambda}^k|}{|\tilde{\lambda}^i|}\right],1\right\} \qquad (3.43)$$

where $\tilde{\lambda}^i,\tilde{\lambda}^j,\tilde{\lambda}^k$ are the spectral radii of the inviscid flux Jacobians in the i-, j-, k-coordinate direction, respectively. The scaling function eq. (3.43) also introduces additional damping in the direction of the long side of the cell. In the wall-normal direction again only a small amount of dissipation is allowed.

The spatial accuracy of the improved flux vector split scheme depends on the determination of the left and right state at cell interfaces. For a first-order scheme the flow quantities at the left and right state are given by their values at the neighboring mesh points, i.e. i,j,k and i+i,j,k, respectively, (see Fig. 10). Higher-order accuracy is obtained with the MUSCL approach in the present work. MUSCL uses extrapolation of flow quantities for the calculation of the left and right states. With this approach several decisions must be taken which affect the ability of the scheme to capture strong shocks and viscous shear layers aligned with the coordinate grids. These are the choice of the flow variables to be extrapolated to the cell face and the choice of the extrapolation function which gives higher-order fluxes in smooth regions of the flow. At discontinuities the function switches to first-order accuracy in order to guarantee shock capturing without spurious oscillation. Here, the van Albada limiter function is chosen according to [46]

$$u_L = u_{i,j,k} + \frac{1}{2}\frac{(\Delta_+^2+\varepsilon)\Delta_- + (\Delta_-^2+\varepsilon)\Delta_+}{\Delta_+^2+\Delta_-^2+2\varepsilon} \qquad (3.44)$$

with

$$\Delta_+ = u_{i+1,j,k} - u_{i,j,k}$$

$$\Delta_- = u_{i,j,k} - u_{i-1,j,k}$$

where u_L denotes the flow quantity u of the left state to be extrapolated to the face i+1/2. The right state, u_R, is evaluated similarly by using the data of points (i,j,k), (i+1,j,k), (i+2,j,k). This limiter function is equivalent to Fromm's scheme in smooth regions of the flow where the gradients squared, Δ_+^2, Δ_-^2, are small compared to ε. In [46] the quantity ε is used in order to suppress limiting of the solution in regions where the flow is nearly constant. This is accomplished by taking

$$\varepsilon = \kappa_1 \Delta x^n \qquad (3.45)$$

where Δx denotes the distance between the grid points i,j,k and i+1,j,k. κ_1 is an empirical constant of O(10) and $2<n<3$. Note, that one can only expect eq. (3.45) to work well when solving the flow equations in their nondimensional form. Eq. (3.45) can be extended to suppress limiting the fluxes within boundary layers. Not only does limiting in the wall-normal direction degrade accuray on coarse meshes but it may also introduce spurious oscillations in the solution as seen in Fig. 25a). Here, we encounter the situation that the cartesian velocity components, u and v, are nonzero but the contravariant velocity component in wall-normal direction is close to zero. Limiting the extrapolation of u and v individually, as it is standard practise in most MUSCL implementations [47], may result in false values for M_L and M_R which define the inherent dissipation of the split flux (eq. (3.29)). This problem is resolved by defining

$$\varepsilon = \max\left[(\kappa_1 \Delta x^n),\kappa_2\left(\Phi^{modAUSM}_{i+\frac{1}{2},j,k} - \bar{M}\right)\right] \qquad (3.46)$$

where κ_2=O(100), $\Phi^{modAUSM}$ is evaluated according to eq. (3.38) with $\tilde{\delta}$ =O(0.1), and $\bar{M}$ is the average of the contravariant Mach numbers at points i,j,k and i+1,j,k. Fig. 3.25a demonstrates that oscillations in wall-normal direction are completely removed by using eq. (3.46) instead of (3.45). Note, that this type of oscillations does not occur in the higher-order results published in [39]. This may be explained by the fact that the viscous test cases selected in [39] used cartesian meshes where the cartesian velocity v is equal to the corresponding contravariant velocity component. For this special case eq.(3.45)is sufficient in order to obtain proper dissipative terms.

It should also be mentioned that the second-order interpolant in eq. (3.44) may be replaced by the third-order formula of [48]. This alternative yields somewhat more accurate results for transonic and supersonic flows but it is less robust for hypersonic flows with strong shocks.

The selection of flow variables for the extrapolation process is described next. Initially, we tried some standard choices, these are the use of primitive or conserved flow variables for extrapolation. It turned out that the latter choice is not robust at transient shock waves, whereas the former tends to support oscillations in stagnation point regions behind strong shocks. Furthermore, either choice does not allow inviscid steady state solutions with constant total enthalpy. Constant total enthalpy in the steady state can be obtained if the energy flux in eq. (3.29) is formed with total enthalpy H being an extrapolated quantity. However, recalculation of the pressure p in eq. (3.29) from a single set of flow variables including H does not yield nonoscillatory fluxes for the momentum equation. Further numerical experiments showed, that extrapolation of the

primitives for mass and momentum fluxes combined with extrapolation of H in order to compute the energy flux results in nonoscillatory flow solutions and superior convergence behavior. This numerical treatment corresponds closely to the underlying design principle of AUSM, which splits the flux vector into an advective and a pressure part.

In the computations of 3D hypersonic flow problems, very strong shocks may occur in regions of strong variations of the grid metrics. For these cases shock resolution is further improved by modifying the limiter function, eq. (3.44), as

$$u_L = u_{i,j,k} + \frac{1}{2}\upsilon_{i,j,k}\frac{(\Delta_+^2+\varepsilon)\Delta_- + (\Delta_-^2+\varepsilon)\Delta_+}{\Delta_+^2+\Delta_-^2+2\varepsilon} \qquad (3.47)$$

with the pressure switch υ, given by eq.(3.39). Additionally, the contravariant Mach numbers, M_L and M_R, are obtained by extrapolation of the contravariant velocity component. More specifically, M_L at cell face i+1/2 is computed by taking

$$M_L = \frac{(q_n)_L}{c_L} \qquad (3.48)$$

where c_L denotes the speed of sound associated with the left state and $(q_n)_L$ is the contravariant velocity which is evaluated with the help of eq.(3.47) and

$$(q_n)_{i,j,k} = \frac{\vec{S}^i_{i+1/2,j,k}\cdot\vec{q}_{i,j,k}}{|\vec{S}^i|_{i+1/2,j,k}} \qquad (3.49)$$

$$\Delta_+ = (\vec{q}_{i+1,j,k} - \vec{q}_{i,j,k})\frac{\vec{S}^i_{i+1/2,j,k}}{|\vec{S}^i|_{i+1/2,j,k}} \qquad (3.50)$$

$$\Delta_- = (\vec{q}_{i,j,k} - \vec{q}_{i-1,j,k})\frac{\vec{S}^i_{i-1/2,j,k}}{|\vec{S}^i|_{i-1/2,j,k}}. \qquad (3.51)$$

Here, $\vec{q} = [u, v, w]^T$ is the vector of cartesian velocities.

In the following, numerical results for inviscid and viscous flows obtained with the improved advection upstream splitting method are presented. Emphasis is put on the method's capability to resolve wall-normal gradients of flow quantities which for instance occur in entropy and boundary layers. As test cases the inviscid flow around a blunt slender cone and viscous 2D flows are selected.

Inviscid calculations around a blunt slender cone [49] at freestream Mach number M_∞=8 and angle of attack α=0° have been carried out. The curved bow shock detached from the blunt nose produces a thick entropy layer in the front part of the configuration which, however, develops to a very thin layer in the rear part. Since the quality of the numerical results strongly depends on the resolution of the entropy layer, computational methods have to be used which accurately predict this flow feature.

The grid used for the calculations is shown in Fig. 26. The C-O topology has been chosen with 161x41x31 grid points in i-,j-,k-direction, respectively [50]. 21 grid points were used to discretize the spherical nose shape in streamwise direction. In i- and j-direction a linear stretching of the grid spacing was introduced. This allows a suitable grid distribution with respect to computational efficiency. The stretching in j-direction provides enough grid points in the near-wall region necessary to resolve the thin entropy layer in the rear part of the configuration. Fig. 27 shows Mach number and pressure contours in the nose region obtained with the improved flux splitting method. The flow field is axi-symmetric since the angle of attack has been set to zero. In order to check the accuracy of the scheme, in Fig. 3.28 the entropy value at the wall is plotted along the body in streamwise direction. Since for inviscid flows the body surface is part of the stagnation streamline, the entropy is constant along the body. Its value is determined through the entropy raise across the normal shock. In Fig. 28 numerical results obtained with the improved AUSM and with the classical van Leer scheme are depicted. In addition, the analytical entropy value at the wall is given. In the front part of the configuration (almost up to 100 nose radii) the error of AUSM is less than 1%. In the rear part, however, the accuracy is decreasing. This may be attributed to the computational grid, which in this part of the configuration is not sufficiently fine to resolve the thin entropy layer as accurately as in the front part. It should be noted that for this calculation the scaling function eq. (3.42) with $\delta = 0.1$ has been used to control the dissipative term. Computations with the other scaling functions or with different parameters δ did not improve the results. As it can be seen in Fig. 28, the classical van Leer scheme produces less accurate results along the whole configuration. This demonstrates that on a given grid the improved flux splitting method is less diffusive compared to the van Leer scheme and therefore more qualified for the accurate resolution of entropy layers.

Several two-dimensional viscous flow problems serve to demonstrate the ability of the new flux vector split scheme to resolve viscous shear layers. We have chosen transonic and hypersonic test cases which are well known from literature.

The first test case is the transonic turbulent flow over the RAE 2822 airfoil at M_∞=0.73, α=2.79°, R_e=65x10^6. The computational grid consists of 320x64 cells. Flow computations were carried out with explicit multi-stage time stepping and multigrid with full coarsening. A typical convergence history is displayed in Fig. 29. Computing time was reduced by full multigrid, that is, coarse-mesh solutions on

grids with 80x16 cells and 160x32 cells were obtained with each 100 multigrid iterations in order to produce the initial solution on the next finer grid. An impression of the overall flow field is provided by Fig. 25b. The improved AUSM yields a clean resolution of the shock and the boundary layers. Fig. 30 compares the distributions of skin friction yielded by AUSM and van Leer scheme under grid refinement. There is a dramatic improvement of resolution visible for the improved AUSM. Not only does the improved resolution of shear layers affect friction drag of the airfoil but also the pressure forces due to viscous/inviscid interaction. This is demonstrated in Fig. 31 where lift and drag values are plotted as a function of the inverse of the total number of cells, N. The results of the high-resolution upwind TVD scheme are included for comparison. The smeared boundary layers of van Leer 's scheme affect the interaction with the shock in that the shock location moves upstream (not shown here). Consequently, lift is underpredicted by van Leer's scheme as compared to AUSM and Upwind TVD. The improved AUSM is best for the prediction of pressure drag whereas AUSM and Upwind TVD do similarly well for skin friction drag. The relatively large values of pressure drag for the upwind schemes on coarse meshes as compared to those for the central differencing plus matrix-valued dissipation given in Fig. 12 are caused by the effect of the flux limiter in the nose region of the airfoil. This effect disappears for subsonic cases when the flux limiting is switched off. The construction of a limiter function which is only active at shocks and does not adversly affect smooth flow regions is still unresolved.

The next viscous 2D test case presented here is the hypersonic laminar flow past a 15° compression ramp. The onflow conditions correspond to case III.4 of the Workshop on Hypersonic Flows for Reentry Problems held in Antibes, 1991 [31]. The grid consists of 288x224 cells. In Fig. 32 the Mach contours are plotted. Results obtained with the second-order TVD scheme and the second-order improved AUSM with scaling eq. (3.42) and δ=0.05 are presented. There are no major differences between the results of the different schemes visible. This statement is supported by the plots of the pressure coefficient the skin friction coefficient and Stanton number along the wall in Fig. 33. Only slight differences occur in the skin friction coefficient and the Stanton number. As in the previous test case, the scaling of the dissipative term in the modified AUSM has no influence on the result on this very fine grid. These calculations demonstrate that the improved flux vector split method predicts viscous flows as accurate as the TVD flux difference splitting scheme. For the viscous test cases presented here almost no differences in the results have been observed for the different scaling functions which have been proposed for a proper scaling of the dissipative term. Compared to the TVD scheme the convergence behavior of the modified AUSM scheme is slightly worse. However, due to the reduced computational effort per time step, the overall efficiency of both methods is comparable. Since in contrast to the TVD scheme the numerical effort of AUSM is proportional to the number of unknowns, substantial reduction of the computational cost can be expected for 3D calculations and also for solutions of flow problems with additional conservation equations. Computations of complex 3D viscous flows over a winged reentry vehicle including deflected control surfaces and multiblock computations of the flow through the slot between different control surfaces (see chapter 5) demonstrated the usefulness of the present discretization for general 3-D applications. AUSM enables us to compute flows with very strong shocks and strong expansions into leeside flow regions, which were impossible with flux difference split methods.

3.6 Viscous Terms

For the approximation of the Navier-Stokes equations all schemes presented in the previous sections rely on the same central discretization of the viscous terms. The viscous fluxes required to determine the solution at point (i,j,k) are approximated using the auxiliary cell shown in Fig. 3.10. They contain first derivatives of the flow variables, which are computed using a local transformation from cartesian coordinates to the curvilinear coordinates ξ, η, ζ [14]. For an arbitrary flow quantity one obtains

$$\phi_x = \frac{\partial \phi}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial \phi}{\partial \eta}\frac{\partial \eta}{\partial x} + \frac{\partial \phi}{\partial \zeta}\frac{\partial \zeta}{\partial x}. \qquad (3.52)$$

The derivatives ϕ_ξ, ϕ_η and ϕ_ζ are evaluated employing central finite differences, whereas the cell face vectors and the volume are used to compute the metric derivatives.

In high Reynolds number flows with thin viscous shear layer the flow gradients in the direction normal to the wall are much larger than those along the wall. This fact allows a simplified approximation of the viscous terms, called thin layer approximation. Using a body fitted mesh, there is one family of grid lines almost parallel to the wall and another one approximately normal to it. If the thin layer is to be resolved accurately and if the number of points is to be kept within a limit which is tolerable to todays supercomputers, highly stretched grids in wall-normal direction are used. On such grids one cannot expect the viscous terms in streamwise direction to be resolved accurately. Therefore, with the thin layer approximation all the viscous contributions arising form gradients in the direction of the quasi-streamwise coordinates are neglected. In all viscous applications shown in this paper the thin layer approximation has been used.

4. EFFICIENT ALGORITHMS FOR THE COMPUTATION OF STEADY-STATE SOLUTIONS

As numerical flow simulations pave their way into the practical aerodynamic design process, the need for efficient methods to solve the equations governing inviscid and viscous flows has become very obvious. Many solvers still used in current aerospace development programs exhibit slow convergence towards the desired steady-state solutions, which leads to high computer costs and long turnaround times. Consequently, there is a substantial amount of research work focused on methods for convergence acceleration. One of the promising approaches is the multigrid method. Multigrid which uses a sequence of successively coarser meshes in order to propagate and damp disturbances throughout the flow field, was initallly invented and analyzed for the solution of elliptic partial differential equations by A. Brandt [51]. Later, the idea was successfully applied to purely hyperbolic or mixed systems of equations in fluid mechanics, even though the mathematical backing of these extensions is still incomplete.

4.1 Multigrid Approach

To set the stage for the discussions of multigrid in subsequent parts of the chapter we first describe the multigrid method and some means to analyze its performance.

4.1.1 Definition of Multigrid Components

The multigrid method deals with a sequence of meshes which differ by their density of grid points. They may be created by successively deleting every second grid line in all coordinate directions. By this, 4-7 coarse meshes are generated for practical flow problems. Here, we will describe an arrangement of a fine mesh with index f and a coarse mesh with index c. The semi discretization of chapter 3 on the fine mesh can be written as

$$\frac{d}{dt}\vec{W}_f = \frac{-\vec{R}_f}{V_f} \tag{4.1}$$

where $\vec{W}_f$ is the solution vector, $\vec{R}_f$ represents the discrete flux balance, and V_f is the discrete volume around the grid point. The fine-mesh solution, $\vec{W}_f$, may be improved by numerically advancing eq. (4.1) in time, which is called smoothing in multigrid terminology. Practical smoothing schemes based on explicit and implicit time stepping are discussed in chapter 4.2. In order to improve the solution on the fine grid with the aid of a coarse grid, a series of steps are carried out as follows.

Both the solution vector and the residual vector are transferred to the coarse mesh. Simple injection

$$\vec{W}_c = \vec{W}_f \tag{4.2}$$

at the coincident grid point is used for transfer of the solution. In order to ensure conservation property for the residual transfer, full weighting according to [51] is applied as

$$I_f^c\vec{R}_f = 8\mu_x^2\mu_y^2\mu_z^2\vec{R}_f \tag{4.3}$$

and μ_x, μ_y, μ_z are the standard averaging operators in curvilinear coordinate directions. Note, that the transferred residual, $\vec{R}_f$, should be based on the most recent solution, $\vec{W}_f$, in order to obtain best efficiency of the overall method. The restricted residual is used to define a forcing function for the coarse mesh

$$\vec{P}_c = I_f^c\vec{R}_f - \vec{R}_c \tag{4.4}$$

as the difference between the restricted residual and the coarse-grid residual calculated with the injected solution. The use of the forcing function eq. (4.4) is necessary if we want to solve eq. (4.1) on the coarse mesh in order to obtain corrections for the solution on the fine mesh. The smoothing scheme is then used to solve

$$\frac{d}{dt}\vec{W}_c = -\left(\vec{R}_c + \vec{P}_c\right)/V_c . \tag{4.5}$$

Note, that during the first numerical update of eq. (4.5) on the coarse mesh the coarse-grid residual $\vec{R}_c$ drops out. This ensures zero corrections from the coarse mesh if the restricted residual from the fine mesh, $I_f^c\vec{R}_f$, vanishes in the steady state.

Execution of one or several time steps on the coarse mesh yields corrections of the form

$$\Delta\vec{W}_c = \vec{W}_c^k - \vec{W}_c^o \tag{4.6}$$

where the superscripts denote the discrete time level. The correction is then transferred to the fine grid which is called prolongation. The prolongation operator is denoted by I_c^f and it contains linear interpolation for most of the results presented in the present lecture (see also chapter 4.3). The total correction on the fine mesh after n time steps on the fine mesh and k time steps on the coarse mesh is

$$\Delta\vec{W}_f = \vec{W}_f^n - \vec{W}_f^o + I_c^f\Delta\vec{W}_c \tag{4.7}$$

4.1.2 Analysis of Model Problem

Von Neumann analysis of a model problem is carried out in order to study the numerical behavior of the multigrid components defined above. Until now, we have used a 2D scalar model of the type

$$\frac{\partial w}{\partial t} + a\frac{\partial w}{\partial x} + b\frac{\partial w}{\partial y} = c\frac{\partial^2 w}{\partial y^2} \quad . \tag{4.8}$$

With appropriate choices of a, b, and c, the model allows to investigate the properties of multidimensional convection dominated problems and also cases, where diffusion takes over.

If one uses uniform spacings, Δx and Δy, for discretization, one can also study the effect of high aspect ratio cells,

$$\Delta x >> \Delta y \tag{4.9}$$

and convection aligned with the grid,

$$a\Delta y >> b\Delta x \quad . \tag{4.10}$$

The scalar model does not allow analysis in situations where the eigenvalues of the inviscid flux Jacobians of the system of flow equations differ due to large differences in the acoustic and convective wave speeds. These differences are typical features of low-speed flow regions and also near sonic lines. The interested reader is referred to Refs. [52-54] for more details about these problems.

We apply semidiscretization for the spatial derivatives on a domain, Ω, which is covered with the fine mesh containing cells with spacings Δx_f and Δy_f and the volume, $V_f = \Delta x_f \Delta y_f$. Defining a time step on the fine mesh, for example,

$$\Delta t_f = \frac{CFL\ V_f}{a\Delta y_f + b\Delta x_f} \tag{4.11}$$

the discrete approximation of eq. (4.8) at point (i,j) reads

$$\Delta t_f \frac{d}{dt}(w_f)_{i,j} = -\frac{\Delta t_f}{V_f} R_{i,j} \quad , \tag{4.12}$$

$$R_{i,j} = a\Delta y_f D_x + b\Delta x_f D_y - c\frac{\Delta x_f}{\Delta y_f} D_{yy} \quad .$$

D_x, D_y and D_{yy} denote the difference operators used to approximate the first and second derivatives of eq. (4.8), respectively. Suppose we want to investigate first-order upwind differencing for the convective terms. Then, we obtain

$$D_x = w_{i,j} - w_{i-j} \quad , \; D_y = w_{i,j} - w_{i,j-1} \tag{4.13}$$

for $a > 0$ and $b > 0$. Difference operators for higher-order discretization may be found in Refs. [55-56].

Assuming a periodic boundary condition, the scalar function, w(x,y,t), can be expressed by a Fourier series

$$w = \sum_i \sum_j \hat{w}_{kl} e^{I(i\theta_x + j\theta_y)} \quad , \tag{4.14}$$

where the Fourier angles, θ_x, θ_y, vary between $-\pi$ and π. In the Von Neumann analysis the behavior of a single mode

$$\hat{w}_{i,j} e^{I(i\theta_x + j\theta_y)} \tag{4.15}$$

is studied and the complete result is obtained by linear superposition.

Inserting eq. (4.15) and (4.13) into eq. (4.12) one obtains the growth of the amplitude of the Fourier mode,

$$\Delta t \frac{d\hat{w}}{dt} = -\hat{Z}\hat{w} \quad ,$$

$$\hat{Z} = \frac{\Delta t}{V}\left[a\,\Delta y\,(I\sin\Phi_x + (1 - \cos\phi_x))\right.$$

$$+ b\Delta x\,(I\sin\phi_y + (1 - \cos\phi_y)) \tag{4.16}$$

$$\left. + 2c\frac{\Delta x}{\Delta y}(1 - \cos\phi)\right] \quad .$$

If the Fourier symbols of a time stepping operator used to solve eq. (4.16) is denoted by $\hat{F}$, one can write eq. (4.16) as

$$\delta w = -\hat{F}\hat{Z}\hat{w}^n \, , \tag{4.17}$$

or

$$\hat{u}^{n+1} = g\hat{u}^n \qquad g = 1 - \hat{F}\hat{Z} \, . \tag{4.18}$$

The Fourier symbols of some selected time stepping schemes used as a smoother for multigrid algorithms are discussed in section 4.2. Any time-stepping scheme to solve the semidiscrete equation (4.12) is linearly stable, if the Fourier mode does not grow in time, that is

$$|g| \leq 1 \, . \tag{4.19}$$

4.1.3 Multigrid Analysis

If a multigrid algorithm is used to solve semidiscrete equation (4.12), the resulting iteration operator becomes a matrix according to Hackbusch [57]. Accordingly, the Fourier transform for analysis of an iteration with multigrid is a matrix with the dimension $2^{l-1} \times 2^{l-1}$ where l denotes the number of grid levels involved. Analysis of this type for fluid mechanics has been published by Mulder [58], Leclercq [59], and Eliasson [60].

As an alternative, Jameson [61] has presented a so-called uniform analysis which simplifies the Fourier transform of

the matrix to a scalar. With the multilevel uniform analysis, fine-grid and coarse-grid corrections are formally computed at all points of the fine grid. Then a nonlinear filter is applied to remove the coarse-grid corrections at fine-grid points not contained in the coarse-grid. The filtering introduces errors in the analysis for the grid points not contained on the coarse grid, that is, it does not allow for the coupling effects due to the interpolation operator in the multigrid method. However, it does offer the advantages of simplicity and easy application to more than two-level schemes. Thus, it allows the rapid comparison of different multigrid algorithms. If a multigrid method is unstable or inefficient according to this analysis, then it is certainly not a reasonable scheme.

In order to apply the uniform scheme analysis one needs the Fourier symbols of the multigrid components. The Fourier symbol of the injection operator from eq. (4.2) is simply 1. The weighted residual transfer operator in 2D,

$$I_f^c = 4\mu_x^2\mu_y^2,$$

has a Fourier transform,

$$\hat{I}_f^c = (1+\cos\Theta_x)(1+\cos\Theta_y). \tag{4.20}$$

As for prolongation, we consider only the mesh points which are contained in the coarse mesh and the fine mesh. Hence, Fourier transform of prolongation is simply 1.

4.2 Smoothing Schemes

This section discusses two selected schemes to iterate the semidiscrete equation

$$\frac{\partial}{\partial t}w + \frac{R}{V} = 0$$

towards its desired steady state solution. The chosen explicit and implicit schemes are characterized by their low operation count and storage requirements. The analysis for 1D and 2D scalar model problems indicates good damping properties of these schemes for high-frequency components of transient errors whereas the long waves which occur on fine coordinate meshes are slowly damped. Therefore, these schemes may be taken as smoother for a multigrid method.

4.2.1 Explicit Multistage Schemes

Explicit multistage schemes are considered here for several reasons. They are simple to program, in particular at boundaries, and for multiblock partitioned computational domains. Moreover, the number of stages and their coefficients can be varied in order to optimize both damping and convection of transient disturbances [61-62]. Finally, the explicit schemes usually do not require start-up procedures. The most simple multistage scheme with p stages reads

$$\begin{aligned} w^{(o)} &= w^n \\ w^{(l)} &= w^{(o)} - \alpha_l \frac{\Delta t}{V} R^{(l-1)} \quad ,l = 1,2,.....,p \\ w^{n+1} &= w^{(p)} \end{aligned} \tag{4.21}$$

One can always represent the change of the Fourier mode, $\hat{w}$, according to eq. (4.16) by substitution of eq. (4.14) into (4.21). This yields the amplification rate, g, as function of the Fourier angles Φ_x, Φ_y. Fig. 34 presents results of a 3-stage scheme and first-order upwind spatial discretization for a 1D convection problem taken from Ref. [62]. High-frequency error modes for $\pi/2 < \Theta_x < \pi$ are well damped whereas the damping for lower frequencies is poor. The Courant number of this scheme is limited to about 1.5. This indicates that transient errors in the solution are convected out of the computational domain at a relatively low rate per time step.

In eq. (4.21) we have assumed that both the central (convective) and dissipative parts of the spatial discretization operator, $\hat{Z}$, are evaluated on each stage of the time stepping scheme. Somewhat more freedom in the design of multistage schemes is gained by evaluating the dissipative parts only at q out of p total stages. Moreover, the dissipation evaluations may be weighted. If one defines C and D as the centrally discretized and dissipative part of the flux approximation, the residual function of weighted multistage schemes is defined as

$$\begin{aligned} R^{(l+1)} &= C(w^{(l)}) + \sum_{m=0}^{l} \gamma_{lm} D(w^{(m)}), \\ \sum_{m=0}^{l} \gamma_{lm} &= 1. \end{aligned} \tag{4.22}$$

One can extend the stability range of the explicit multistage schemes by a simple scalar implicit operator acting on the residuals. For two dimensions, this residual smoothing can be applied in the factored form

$$(1-\beta_x\nabla_x\Delta_x)(1-\beta_y\nabla_y\Delta_y)\bar{\Re}^{(l)} = \Re_r^{(l)} \tag{4.23}$$

where the residual $\Re_r^{(l)}$ is defined as

$$\Re_r^{(l)} = \frac{\Delta t}{V} R^{(l)}, \tag{4.24}$$

∇ and Δ are the normal forward and backward difference operators. The smoothing coefficients, β_x and β_y depend on the Courant numbers in the individual coordinate direc-

tions according to Refs. [63,64]. This implicit procedure allows the explicit stability limit to be increased by a factor of 2 to 3. The performance of a particular 5-stage scheme which was optimized by Tai [65] using weighted residual evaluation and implicit residual smoothing is displayed in Fig. 35. The scheme damps disturbances much better for the long-wave range, as compared to the 3-stage scheme of Fig. 4.1. Note, that this scheme requires only 3 evaluations of dissipative flux terms which usually take the majority of floating point operations. Contours of constant amplification factor over the Fourier angles in ξ and ζ directions for the 2D convection problem with a=b=1 are shown in Fig. 36. Results of the uniform multigrid analysis briefly described in section 4.1 are also included. It is seen that the multigrid scheme acts as to improve damping at lower frequencies. There is enough stability margin of the scheme for Fourier angles $|\Theta| > \bar{\pi}/4$ which indicates that the errors contained in the uniform analysis can be tolerated.

4.2.2 Implicit LU-SSOR Scheme

Multigrid methods based on explicit multi-stage schemes have been shown to yield good convergence rates for both inviscid and viscous flows. As seen in chapter 4.2.1 the principal reason for this is, that the number of stages and the stage coefficients can be tuned such that good high frequency damping is obtained which is necessary for an efficient multigrid process. However, for flow problems which are governed by equations with strong source terms, as for example viscous flows for which turbulence viscosity is determined by multi-equations turbulence models and hypersonic non-equilibrium flows, a severe time-step restriction is imposed on explicit schemes. This leads to slow convergence, even if a multigrid method is used. In order to overcome the time-step restriction, some kind of implicit operator has to be used. Various approaches are known in the literature. Preferable techniques are the point-implicit treatment of source terms or the full implicit treatment of all equations. Thus there is an urgent need to develop implicit multigrid methods.

In the past efficient multigrid methods have been developed in conjunction with implicit schemes (e.g. [66-70]). Various implicit operators have been used as a multigrid driver including factored and unfactored schemes. This report focuses on the investigation of the damping properties, convergence behavior and stability of the implicit LU-SSOR scheme in the framework of a standard multigrid method.

The LU-SSOR scheme (Lower-Upper Symmetric Successive Overrelaxation) became quite popular because of its low numerical effort, efficient implementation on vector computers and reasonable convergence speed. The algorithm belongs to the class of factored schemes and is based on the decomposition of the full implicit operator into lower and upper triangular matrices. The LU-SSOR scheme originally introduced by Yoon and Jameson [69,71] and further developed by Rieger and Jameson [72] and Yoon and Kwak [73] combines the advantages of the LU-factonization and the symmetric Gauss-Seidel relaxation. Recently, Yoon et al [74] and Blazek [70,75] have used the LU-SSOR scheme as an effective smoother for an efficient multigrid scheme. They have shown fast convergence for many inviscid and viscous flow problems including high speed flows.

In the following the LU-SSOR scheme is briefly presented. Details are given in [75]. In general an implicit scheme for the system of ordinary differential equations (4.1) can be formulated as

$$\frac{\Delta\vec{W}}{\Delta t} = -\left(\beta\vec{R}^{n+1} + (1-\beta)\vec{R}^{n}\right) \tag{4.25}$$

with the solution correction

$$\Delta\vec{W} = \vec{W}^{n+1} - \vec{W}^{n} \tag{4.26}$$

and the discrete flux balance

$$\vec{R} = \frac{1}{V}\left(\vec{R}^{c} - \vec{R}^{v}\right) \tag{4.27}$$

where $\vec{R}^{c}$ and $\vec{R}^{v}$ denote the convective and viscous part, respectively. The parameter β ($1/2 \le \beta \le 1$) determines the accuracy of the implicit scheme in time. For $\beta=1/2$ the scheme is second-order accurate, while for all other values the time accuracy drops to first order.

Linearizing eq. (4.25) by

$$\vec{R}^{n+1} = \vec{R}^{n} + \frac{\partial\vec{R}}{\partial\vec{W}}\Delta\vec{W} + 0(\Delta t^{2}) \tag{4.28}$$

and dropping all terms of second- and higher-order, one obtains the general unfactored implicit scheme

$$\left[I + \beta\Delta t\frac{\partial\vec{R}}{\partial\vec{W}}\right]\Delta\vec{W} = -\Delta t\ \vec{R}^{n}. \tag{4.29}$$

For a grid point (i,j,k) the term $\partial\vec{R}/\partial\vec{W}$ is expressed as

$$\left(\frac{\partial\vec{R}}{\partial\vec{W}}\right)_{i,j,k} =$$

$$\frac{\partial \vec{R}^{c}_{i+1/2,j,k}}{\partial \vec{W}} - \frac{\partial \vec{R}^{c}_{i-1/2,j,k}}{\partial \vec{W}} + \frac{\partial \vec{R}^{c}_{i,j+1/2,k}}{\partial \vec{W}} - \frac{\partial \vec{R}^{c}_{i,j-1/2,k}}{\partial \vec{W}} + \frac{\partial \vec{R}^{c}_{i,j,k+1/2}}{\partial \vec{W}} - \frac{\partial \vec{R}^{c}_{i,j,k-1/2}}{\partial \vec{W}} - \frac{\partial \vec{R}^{v}_{i,j+1/2,k}}{\partial \vec{W}} + \frac{\partial \vec{R}^{v}_{i,j-1/2,k}}{\partial \vec{W}} \qquad (4.30)$$

Here it is assumed, that the thin layer approximation of the Navier-Stokes equations is used in which the viscous flux only in the wall-normal direction (j-direction) are taken into account.

The evaluation of the quantities on the right hand side of eq. (4.30) in terms of the flux Jacobians yields

$$\left(\partial \vec{R}/\partial \vec{W}\right)_{i,j,k} =$$

$$\begin{aligned}
&[\,(A^{+}_{i,j,k} + A^{-}_{i+1,j,k}) - (A^{+}_{i-1,j,k} + A^{-}_{i,j,k}) \\
&+ (B^{+}_{i,j,k} + B^{-}_{j+1,k}) - (B^{+}_{i,j-1,k} + B^{-}_{i,j,k}) \\
&+ (C^{+}_{i,j,k} + C^{-}_{i,j,k+1}) - (C^{+}_{i,j,k-1} + C^{-}_{i,j,k}) \\
&- (-H_{i,j,k} + H_{i,j+1,k}) + (-H_{i,j-1,k} + H_{i,j,k})\,]/V_{i,j,k}
\end{aligned} \qquad (4.31)$$

where $A^{\pm}$, $B^{\pm}$, $C^{\pm}$ denote the split matrices of the flux Jacobians A,B,C in i-,j-k-direction, respectively. The matrices with superscript '+' contain only positive and those with superscript '-' only negative eigenvalues. As proposed in [72] they are given by

$$A^{\pm} = \frac{1}{2}(A \pm \omega\, r_A\, 1) \qquad (4.32)$$

with

$$r_A = \max\{|\lambda| : \lambda \text{ eigenvalue of matrix A}\}\,. \qquad (4.33)$$

The factor ω, $\omega \geq 1$, determines the amount of implicit dissipation and hence influences the damping and convergence properties of the scheme.

The terms $B^{\pm}$ and $C^{\pm}$ are defined in the same manner. The matrix H in eq. (4.31) corresponds to a viscous flux Jacobian without the spatial operators. It has been found that in the framework of a finite volume formulation the use of correct metric terms is a critical point. For details the reader is referred to references [75].

Inserting expression (4.31) into equation (4.29) one obtains

$$\begin{aligned}
&\{1 + \beta\frac{\Delta t}{V_{i,j,k}}\,[\,(A^{+}_{i,j,k} + A^{-}_{i+1,j,k}) - (A^{+}_{i-1,j,k} + A^{-}_{i,j,k}) \\
&+ (B^{+}_{i,j,k} + B^{-}_{j+1,k}) - (B^{+}_{i,j-1,k} + B^{-}_{i,j,k}) \\
&+ (C^{+}_{i,j,k} + C^{-}_{i,j,k+1}) - (C^{+}_{i,j,k-1} + C^{-}_{i,j,k}) \\
&- (-H_{i,j,k} + H_{i,j+1,k}) + (-H_{i,j-1,k} + H_{i,j,k})\,]\,\}\,\Delta\vec{W} \\
&\qquad = \Delta t\, \vec{R}^{n}_{i,j,k}
\end{aligned} \qquad (4.34)$$

The LU factorization of the implicit operator of eq. (4.34) then yields

$$(LD^{-1}U)\,\Delta\vec{W} = -\Delta t\, \vec{R}^{n}_{i,j,k} \qquad (4.35)$$

with the factors

$$D = \qquad (4.36)$$

$$\{I + \beta\frac{\Delta t}{V}A^{+}_{i,j,k} - A^{-}_{i,j,k} + B^{+}_{i,j,k} - B^{-}_{i,j,k} + C^{+}_{i,j,k} - C^{-}_{i,j,k} + 2H_{i,j,k}\,]\,\}$$

$$L = \{D - \beta\frac{\Delta t}{V}[A^{+}_{i-1,j,k} + B^{+}_{i,j-1,k} + C^{+}_{i,j,k-1} + H_{i,j-1,k}]\,\}$$

$$U = \{D + \beta\frac{\Delta t}{V}[A^{-}_{i+1,j,k} + B^{-}_{i,j+1,k} + C^{-}_{i,j,k+1} - H_{i,j+1,k}]\,\}\,.$$

The use of splitting according to eq. (4.32) allows a simplified evaluation of the diagonal operator D

$$D = I + \beta\frac{\Delta t}{V}[\,(r_A + r_B + r_{C_{i,j,k}}) + 2H_{i,j,k}]\,. \qquad (4.37)$$

The diagonal dominance of the factors L and U is provided by eq. (4.32). Hence, each factor of the decomposition is diagonally dominant and thus the numerical stability of the inversion process is ensured.

As demonstrated in [75] one iteration of the LU-SSOR scheme is carried out in two steps, a forward and a backward sweep

$$\begin{aligned}
L\Delta\vec{W}^{*} &= -\Delta t\, \vec{R}^{n}_{i,j,k} \\
U\Delta\vec{W} &= D\Delta\vec{W}^{*} \\
\vec{W}^{n+1} &= \vec{W}^{n} + \Delta\vec{W}.
\end{aligned} \qquad (4.38)$$

The sweeps are accomplished along diagonal lines. As a consequence, in comparison to the most other implicit schemes, only a block diagonal matrix for viscous flows - or even a scalar diagonal for inviscid flows - instead of block tri- or pentadiagonal matrices has to be inverted.

block tri- or pentadiagonal matrices has to be inverted. This reduces the numerical effort significantly and it also allows a straight forward vectorization. Furthermore, as shown in [72] the Jacobian matrices can be substituted by fluxes which considerably reduces the number of operations. All in all, the computational expense of the LU-SSOR scheme is comparable to that of an explicit two-stage scheme (see chapter 4.2.1).

In combination with multigrid the LU-SSOR scheme described above is used as smoother on all grid levels. Its damping properties have been investigated in detail by Blazek [75] using single- and a two-grid von Neumann Fourier analysis as described in section 4.1. Central as well as upwind spatial discretizations of the explicit operator (right hand side of eq. (4.29)) have been considered. Figs. 37 and Fig. 38 present contours of the single-grid amplification factor for the central discretization and the second-order upwind discretization for a fully convection dominated model problem. In both cases the influence of the relaxation parameter ω (eq. (4.32)) is shown. For the central discretization (Fig. 37) the best damping is obtained with ω=1 (no overrelaxation). However, it is evident, that compared to the very good damping properties of explicit multi-stage schemes the high frequency modes are only poorly damped with the implicit LU-SSOR scheme. Fig. 38 shows that in the case of upwind discretization the high frequency damping behavior of the LU-SSOR scheme can be significantly improved by a moderately increased relaxation parameter. Further improvement can be achieved by spending two time steps on the fine grid.

Despite the fact that compared to the explicit multi-stage schemes the damping behavior of the implicit LU-SSOR scheme is rather poor, an efficient and robust multigrid method driven by the LU-SSOR scheme can be constructed as demonstrated in [75]. Fig. 39 displays the convergence behavior for the transonic inviscid flow past the NACA 0012 airfoil for M_∞=0.8, α=1.25°. An O-type grid with 160x48 cells has been used and the right hand side has been discretized by central differences. The convergence histories of a single-grid and two different 5-level multigrid schemes are compared. The number in the parentheses denote the number of time steps on each grid, ordered from the finest to the coarsest one. As one can observe, a significant improvement of the convergence is obtained by using the LU-SSOR scheme in combination with multigrid. Note, that the usual multigrid scheme with one time step on each grid was not running stable. This may be due to the poor high frequency damping of the LU-SSOR scheme. Fig. 40 presents the convergence behavior of the implicit multigrid scheme for a hypersonic laminar flow past a 15° compression ramp at a medium Reynolds number (test case III.4 of Antibes workshop see Figs. 32-33). The flow parameters are M_∞=11.68, Re_c=2.47x10^5, T_∞=65k and T_w/T_∞=4.604. A computational grid with 288x224 cells has been used. Fig. 40 displays the convergence histories of different multigrid strategies. As one can observe, the multigrid scheme with two time steps on all grids shows the fastest convergence and requires by far the shortest CPU-time for the same residual level. It is also evident from Fig. 40 that the LU-SSOR scheme is only in combination with multigrid adequate to solve this flow problem.

4.3 Multigrid Strategies

The numerical simulation of high Reynolds number flows requires coordinate meshes with high-aspect ratio cells in order to resolve thin shear layers with a reasonable number of grid points. This renders the discretized flow equations stiff because the spectral radii of the flux Jacobian in wall-normal and tangential coordinate directions are very different. Consequently, convergence to the steady state slows down considerably for such flows if no action is taken to circumvent the problem. Similarly, stiffness occurs in situations where the flow is aligned with the grid lines and hence, the numerical dissipation inherent in modern upwind schemes vanishes. One possibility to cope with stiffness resulting from high-aspect ratios is to use specific multigrid strategies in order to improve damping rates. The semicoarsening method introduced by Mulder [58] is one possible approach. Fig. 41 gives a sketch of the idea for two grid levels. With conventional full coarsening the fine mesh with m x n cells is coarsened to yield a mesh with m/2 x n/2 cells. Figs. 41b-d show schemes with semicoarsening in the different coordinate directions, which use two coarse meshes, m/2 x n, and m x n/2. The various semicoarsening schemes differ in how the corrections on the coarse meshes are assembled and prolongated according to Ref. [76]. The coarse-mesh corrections of the scheme, Fig. 41b, are averaged before adding them to the fine mesh. This is indicated by the numbers at the "up" arrows. Due to this averaging, half of the individual corrections on the coarse meshes is lost.

In order to overcome this deficiency of the semicoarsening scheme, two more variants are considered. For the scheme of Fig. 41c, the solutions on the coarse meshes are computed sequentially. Hence, the corrections obtained on the m/2 x n mesh can be used to update the m x n/2 mesh before time stepping (as indicated by the horizontal arrow). The sequential update of the second coarse mesh allows the full amount of corrections to be passed up to the fine mesh. An interesting compromise between the schemes of Figs. 41b and 41c is displayed in Fig. 41d. Here, only the corrections common to both of the coarse meshes, m/2 x n, and m x n/2, are averaged, whereas the corrections to the modes living either on m/2 x n or on m x n/2 are passed to the fine mesh in full.

For the numerical solution of the Navier-Stokes equations,

the two-level strategies presented in Fig. 41 are extended to multilevel schemes as displayed in Fig. 42 following ideas of Mulder [58]. Suitable coordinate meshes for thin boundary layers exhibit mostly cells with high aspect ratios in the surface-aligned direction. Fig. 43 displays further variants of semicoarsening for these situations which are computationally cheaper than the schemes shown in Fig. 42.

Detailed numerical investigations for various viscous flow problems have been reported in Ref. [76]. A sample result is presented in Figs. 44-45. The flow over a slender forebody is chosen to represent a generic configuration corresponding to an air-breathing high-speed transport. The high Reynolds number requires a mesh with aspect ratios up to 25000. The flow computations where done with boundary layer transition fixed at 2 percent chord. The flow solution shown in Fig. 44 was extensively investigated with respect to both grid convergence and residual convergence. The convergence histories presented in Fig. 45 indicate substantial convergence acceleration by multigrid. The sequential semicoarsening scheme takes 194 cycles and 570s on CRAY-YMP to reduce the averaged residuals by 6 orders of magnitude. The scheme with full coarsening takes 1024 and 1230s whereas the single-mesh code requires 7762 steps and 6190s to achieve the same level. We conclude that suitable multigrid strategies can improve computational efficiency by an order of magnitude for tough flow problems.

5. APPLICATIONS

Applications to complex, three-dimensional configurations are given in this section. To demonstrate the range of applicability, sub-, trans-, and hypersonic flow fields are considered. Since the solution method uses structured, body-fitted meshes, the multiblock approach is employed. Hence, this section begins with an outline of the multiblock concept. The first problem to be presented is concerned with the interaction of a jet with a multi-element wing at subsonic speed. The next application deals with engine-airframe integration for transonic transport aircraft. At last the flow field around a reentry vehicle at hypersonic speeds will be analyzed.

5.1 Multiblock Approach

Using structured, body-fitted meshes the physical domain is decomposed into a set of computational cells by the curvilinear coordinates ξ, η, and ζ, as sketched in Fig. 46 for a three-dimensional wing. The curvilinear coordinates allow the mapping of the physical domain into a computational domain as shown in Fig. 47, where the computational coordinates i, j, k are defined along the curvilinear directions ξ, η, and ζ. With the indices i, j, k each point in the computational domain and his neighbors may directly be identified, and the underlying structure allows an easy implementation of the solution algorithm on vector computers.

For complex configurations in general it is not possible to map the physical domain into one coherent computational domain. Therefore, the physical domain of interest is decomposed into different appropriate regions which are called blocks. Each block is mapped into a separate computational domain, and the flow solver is then repeatedly applied to the different blocks. In order to establish a communication between the blocks, data has to be transferred between adjacent block faces. In the DLR CEVCATS code considered here, the exchange of data is established by using the concept of fictitious points, as sketched for a two-dimensional example in Fig. 48. For a 2D problem the real computational domain ranges from i = 2 to i = imax and j = 2 to j = jmax. This real computational domain is surrounded by a sheet of fictitious cells, as indicated by the dashed lines in Fig. 48. Considering a simple O-mesh around an airfoil, the physical domain may be mapped into the computational domain by introducing a computational cut at i = 2 and i = imax, see Fig. 49. Since in the physical domain the block faces at i = 2 and i = imax are adjacent to each other, the exchange of data can be performed by loading the data of line i = 3 into the line of fictitious points at i = imax+1, and by loading data of i = imax-1 into the line with i = 1, as sketched in Fig. 50. It should be noted that when using a vertex based method, it is not sufficient to transfer only the dependent variables. In order to evaluate the flux balances for the points lying directly on the line of the computational cut, the cartesian coordinates of the adjacent block face have also to be provided for the fictitious points. However, for time-invariant grids this needs only to be done once at the beginning of the computation.

It is well known that there does not exist one optimal grid topology for arbitrary configurations. Each aerodynamic component of an aircraft may have its own natural grid structure, and different configurations call for different block arrangements. Therefore, the part of a computer code which depends on the specific configuration has to be kept to a minimum to allow an easy change of grid topologies. In the CEVCATS code this flexibility is provided by an external logic-file which contains all information about the arrangement of blocks, adjacent block faces, and boundary conditions on these block faces. With the information stored in the logic-file, data in the fictitious cells is updated depending on the boundary conditions specified on the particular block face. In order to allow a high flexibility for complex problems, block faces may be subdivided into arbitrary segments. The logic-file then identifies the size and the type of boundary condition on the segments. The use of the logic-file allows to apply one source code to various kinds of problems without the need to change and recompile the program.

As long as the data of all blocks is stored in the main memory of the computer, the fictitious cells at computational cuts may be updated after each operation inside a block, and sweeping successively through all blocks by consistently updating the block boundaries at coordinate cuts, the block structure can be made invisible for the solution algorithm for explicit time stepping methods. However, in order to enable the computation of problems which exceed the storage capacity of the main memory, the DLR CEVCATS code allows the storage of block data on external high-speed storage devices. In this case only one block at a time is loaded into the main memory, and data of all other blocks is stored on the external devices. Having performed a certain number of operations inside the block, data is unloaded onto the external devices and data of the next block is transferred into the main memory. This strategy theoretically enables the computation of problems with an almost unrestricted number of grid points. The problem with this strategy is the high amount of I/0 operations which arise when the cut boundaries should be consistently updated. This becomes especially important when multigrid acceleration is used. since performing more operations inside a block before switching to the next one introduces a time-lag in the evolution of the solution in different blocks. This time-lag may severely deteriorate the damping properties of the scheme. which are mandatory for good multigrid performance. In the CEVCATS code different strategies for multiblock multigrid have been implemented to allow the best compromise between convergence and I/0 operations, depending on the problem. Without going into the details of the different strategies it may be noted that even in the strategy with the lowest amount of I/0 operations, a sweep through all blocks is completed on one grid level before starting on the next coarser grid. It was found that performing a complete multigrid cycle inside a block before switching to the next block degrades the multigrid performance to that of a code without multigrid acceleration or even inhibits convergence. The application of Full Multigrid may alleviate the problems associated with the time-lag, since the solution which evolved on coarser meshes provides a well conditioned starting solution on the finest mesh, and time-differences between blocks are then already rather small.

Details of the implementation of the multiblock multigrid technique into the CEVCATS code may be found in [77] and [78].

5.2 Interaction of a Jet with a Multi-Element Wing

The influence of a jet on a High-Lift device was investigated. It was assumed that the flow field will be dominated by the momentum of the jet flow, and the solution of the Euler equations was regarded as being sufficient to describe the main flow phenomena. The greatest challenge was to decide on an appropriate grid topology. On the one hand the components of the High-Lift device had to be sufficiently resolved, and on the other hand the jet generator had to be incorporated into the mesh. Therefore, a preliminary two-dimensional study was performed to investigate different grid topologies for multi-element airfoils. In the finally chosen topology all single components were resolved by local O-meshes around each component, and the O-meshes were then embedded into a global H-mesh. The grid was generated with the mesh generation tool MEGACADS [79]. Fig. 51 gives a view of the 2D mesh around the complete multi-element airfoil, and Fig. 52 and Fig. 53 show the mesh topology in the region of the slat and in the region of flap and tab.

Since the CEVCATS code has an option for the computation of two-dimensional flows on block structured grids, the same source code as for the following three-dimensional computations could be used for this preliminary test problem. Fig. 54 shows the pressure distribution computed for $M_\infty = 0.182$ and $\alpha = 10°$. The corresponding distribution of total pressure losses is displayed in Fig. 55. On all components total pressure losses are well below 2%. The convergence history for this case is given in Fig. 56, where a W-cycle with four grid levels had been used.

The described grid topology had proved to be adequate for this problem, and the incorporation of the jet-generator was achieved as sketched in Fig. 57. Fig. 58 shows a view of the symmetry plane of the final grid, where the components of the multi-element wing and the jet-generator are displayed as solid objects. The jet-generator had been resolved by a local polar mesh, and this polar mesh was embedded into the global mesh, as shown in Fig. 59.

First computations were performed at $M_\infty = 0.182$ and $\alpha = 10°$, and the ratio of the total pressure of the jet to the ambient static pressure was chosen to $P_{t\,jet}/P_\infty = 2.0$. At these conditions the Mach number of the jet is close to $M_\infty = 0.9$ at the exit of the jet generator. Fig. 60 shows the Mach number distribution in the symmetry plane. The jet can be identified by the concentration of isolines at the jet boundaries. Due to the numerical viscosity, the boundaries are spread into regions of large gradients instead of being discontinuities. Since for these calculations the basic cell vertex central differencing scheme had been used, the smearing effect of the scalar dissipation is clearly visible. The jet passes very closely beneath the slat, and due to the presence of flap and tab the jet is deflected by nearly 25°. In Fig. 61 the corresponding streamline pattern is displayed. When the jet hits flap and tab, streamlines are running against the main flow direction around the leading edges of flap and tab. Fig. 62 gives an enlargement of the region around the tab. Since the streamlines are following the surfaces of flap and tab, the momentum of this deflected part of the flow leads to a deflection of the total jet. The interac-

tion between jet and flap/tab influences a large region of the flow around the wing. Fig. 63 shows the streamline pattern on the lower wing surface. Hitting flap and tab, fluid diverts in all directions. It takes about five engine diameters apart from the symmetry plane until the main flow direction prevails again. It should be noted that the boundary at the wing tip was modelled by solid wall conditions to simulate the wind tunnel walls.

For the onflow conditions of $M_\infty = 0.147$, $\alpha = 10°$, and a pressure ratio of $P_{t\,jet}/P_\infty = 1.252$, experimental data were available. Sectionwise pressure distributions were measured in the symmetry plane, half an engine diameter apart from the symmetry plane, and one engine diameter apart from the symmetry plane. Figs. 64-66 show a comparison of experimental and computational data. The qualitative agreement between calculation and experiment is quite good, despite the neglection of viscous effects. The influence of the jet on the pressure distribution in different spanwise direction is accurately predicted by the calculation.

5.3 Engine Integration for Transport Aircraft

Engine/airframe integration is a key feature in the design and development of advanced technology aircraft, since the interaction between propulsion system and airframe can have a significant impact on the performance of the aircraft. It is evident that an optimal integration of the propulsion system into the airframe will result in an enhanced performance of the whole aircraft. In order to get a better understanding of the aerodynamic phenomena playing the major roles in the interference process, substantial efforts have been made to simulate interference effects. Besides wind tunnel testing numerical methods are increasingly gaining attention, and the solution of the Euler equations has successfully been used to predict interference effects [80, 81, 82]. However, the flow around modern transonic wings is very sensitive to viscous effects, and neglecting viscosity leads to systematic deviations from experimental results [83]. Therefore, the Navier-Stokes equations have to be solved for an adequate simulation. For complex configurations grid generation becomes a substantial challenge, especially for viscous flows, since the boundary layers on all components have to be resolved. To alleviate the necessary effort and to approach the task of generating a viscous grid for the complete configuration step by step, it therefore seems appropriate to first resolve only the boundary layer on the wing and to treat all other components as in inviscid flow.

In the study to be presented here, the DLR-F6 configuration has been selected as a generic twin-engine transport aircraft configuration. The propulsion system is simulated by axisymmetric throughflow nacelles, and the nacelle position was chosen to give rise to quite strong interference effects. Fig. 67 presents a view of the model in tail-off configuration including the main geometrical dimensions.

Using block-structured methods, an appropriate grid topology has to be chosen. On the one hand different engine systems may have to be realized, and on the other hand the boundary layer around the wing has to be resolved adequately. Here a global H-topology in streamwise direction and an O-topology in spanwise direction have been chosen. Nacelle and pylon have been embedded into this grid by using a local polar subgrid with an H-type topology in streamwise direction. Fig. 68 shows selected grid planes to visualize the spanwise topology of the wing and the nacelle. To resolve the wing boundary layer, an C-grid wrapped around the wing has been integrated into the global H-O topology. Fig. 69 presents the resulting H-C-O topology. The C-block is generated using the surface normal vectors of the wing, and the first distance off the wall is about 1.0×10^{-5}. Fig. 70 shows a grid plane at the pylon location through the nacelle to display the embedded C-grid. It should be noted that in the figures not all grid lines have been displayed to allow a clear presentation. The complete field grid consisted of about 1,200,000 cells, and 14 computational blocks had been used. The number of blocks was not only dictated by topological requirements, but the maximum block size had to be adapted to the limited main memory of the computer.

Experiments for the DLR-F6 model have been carried out in the S2MA wind tunnel of ONERA [83]. Pressure distributions have been measured at eight different wing sections with two of them located closely inboard and outboard of the pylon. Transition was fixed and the Reynolds number was kept constant to $Re = 3.0 \times 10^6$. The results to be presented here are restricted to typical cruise conditions of a transonic transport aircraft at $M_\infty = 0.75$ and $\alpha = 0.98°$. For the computations the flow was assumed to be fully turbulent and the algebraic turbulence model of Baldwin and Lomax [84] was used in the solution of the Reynolds-averaged Navier-Stokes equations.

Fig. 71 shows a comparison of measured and computed pressure distributions at two sections inboard of the pylon, and Fig. 72 gives the comparison for two sections located outboard. The exact location of the sections is given in the sketch in the figures. The shock location predicted by the computation agrees favorably with the experimental data. The interference effects caused by the nacelle are clearly visible by the difference in the pressure distributions just inboard ($y/s = 0.331$) and outboard ($y/s = 0.377$) of the pylon. At $y/s = 0.331$ on the lower wing surface a strong flow acceleration occurs. The computation accurately predicts the corresponding pressure peak. Outboard at $y/s = 0.377$ however, the flow is only accelerated around the pylon leading edge, but then no further acceleration occurs. This difference between inboard and outboard side of the pylon is simulated in agreement with the experiment, indicating

that in this case interference is mainly caused by the displacement effect of pylon and nacelle. Besides this overall agreement, there are still discrepancies in the simulation. Downstream of the shock an overexpansion occurs in the computation, which is not observed in the experiment. Furthermore, the effect of the rear loading is overpredicted by the computation. The reason for these effects is still not clear. On the one hand the wing had a blunt trailing edge which was artificially closed for the computation. On the other hand the grid distortion in the vicinity of the pylon may be too large and lead to a reduction of solution accuracy. Computations of the configuration without nacelle gave better agreement with experimental data [84]. The overexpansion downstream of the shock and the overprediction of the rear loading lead eventually to an overprediction of the spanwise lift distribution. Fig. 73 shows a comparison of measured and calculated spanwise lift distributions. Besides the overprediction of lift, the characteristic discontinuity at the pylon location is accurately predicted by the computation. The computations have been carried out on the CRAY Y-MP computer of DLR, and Fig. 74 presents the convergence history for this case. Full Multigrid has been used with 4 grid levels on the first mesh. The residual could be reduced by 3 orders of magnitude within 150 iterations. The computation required about 6000 seconds of CPU-time on the CRAY Y-MP.

5.4 Aerothermodynamics of Winged Reentry Vehicles

At hypersonic flow conditions the thermal stability of the materials used for the fabrication of the flight vehicle limits the maximum allowed heating of the surfaces. The heating becomes critical during reentry maneuvers at high Mach numbers where peak heating rates occur at the nose of the vehicle, along the leading edges of wing and winglet, and on deflected control surfaces which are necessary to achieve equilibrium in pitching moment.

Fig. 75 shows the European space plane HERMES which is a typical design for personnel transport to orbit and return missions. The critical heat loads on HERMES configuration during reentry have been analyzed using a series of global and local flow solutions which were computed with the DLR multiblock code CEVCATS. The hypersonic flow computations require high resolution of very strong shocks and thin temperature layers near the surfaces. Therefore, the hybrid AUSM scheme described in section 3 was employed for spatial discretization instead of the central-difference scheme used for the transonic flow cases.

Elevon heating and pitching moment coefficients of HERMES were computed with global flow solutions on a grid with 800,000 points shown in Fig. 76 and an additional series of local flow solutions, Fig. 77 with deflected elevons [85]. As the inviscid part of the flow is supersonic in axial direction, the flow variables in the inflow plane of the local computational domain for the rear of HERMES could be obtained from the global flow solutions. Steady-state solutions where obtained with about 300 multigrid cycles. Fig. 78 displays streamlines and Stanton numbers for the rear of the windward side of HERMES (1.0) configuration and 10° deflection of elevon and body flap. The flow conditions correspond to windtunnel tests in ONERA S4MA. A large separation occurs at the hinge line of the deflected controls. The computed Stanton numbers are in good agreement with the wind tunnel data. Some discrepancies occur along the symmetry line which were traced to boundary layer transition in the experiment. Note, that the experimental M_∞=10 data represents the highest Mach number, for which reliable experimental data for the complete configuration can be obtained in Western Europe. However, flight peak heating rates occur at M_∞=25 and an trajectory point of 75 km altitude. At these flow conditions, the Reynolds number is lower than at M_∞=10 and significant chemical reactions take place in the flow due to high temperatures. These reactions were taken into account by assuming air in thermochemical equilibrium in our computations. Fig. 79 displays significant differences in the flow behavior between both flow conditions. The flow separation almost disappears at M_∞=25. However, the heat flux is more sensitive to local flow divergence than at M_∞=10, that is, heating increases largely towards the lateral edges of the deflected elevon. Ref. [85] presents a detailed analysis of the flap heating versus flap efficiency and also the effect of changing flap geometries.

The need for aerodynamic control makes the integration of control surfaces for pitch, roll, and yaw control necessary. This is accomplished by defining the body flap, the elevon and the rudder, according to Fig. 75 . The controls are sized by the requirement of sufficient control surface efficiency at hypersonic speed and the maximum deflection angle allowed to limit aerodynamic heating. A large slot between the rudder and the elevon is thus unavoidable due to the dihedral of the winglet. The winglet of a winged reentry vehicle with aerodynamic control has therefore two edges which are exposed to the incoming flow. These are:

- the leading edge of the winglet with a local maximum of the heat flux which depends on the angle of attack, the geometric angle ψ according to Fig. 75 , and the leading edge radius
- the lower edge of the rudder where an attachment line with a local maximum of the heat flux expected.

Computations of peak heating rates along the attachment lines at the winglet leading edge and the lower edge of the rudder are reported in Ref. [86]. Here, we will only present some surprising results which could not have obtained without extensive use of 3D flow computations. As for the computations of flap heating we have used a series of glo-

bal and local flow solutions to compute attachment line heating. The local flow solutions where necessary to represent the complex geometry of the slot in between elevon and rudder with a two-block computational domain, according to Fig. 80.

Peak heating rates along the heading edges of the wing and winglet of HERMES are presented in Fig. 81. It is seen that there exist a large sensitivity of winglet heating due to angle of attack. At higher angles of attack, the effective sweep of the leading edge increases thereby reducing heat load. Even though peak heat fluxes may be measured in wind tunnel tests at M_∞=10, numerical flow simulations are necessary for trajectory points at higher Mach numbers. Fig. 81 demonstrates that semiempirical correlations in order to collapse peak heating at different flow conditions for simple shapes, i.e. the use of Stanton-Miller numbers of Ref. [87], do not neccessarily work well at the winglet. The differences in Stanton-Miller numbers between wind tunnel and flight condition may be due to increased viscous interaction at the lower Reynolds number, and also, the high temperature chemical effects on local flow angles ahead of the winglet.

A completely different trend is observed for heating along the lower edge of the rudder. Fig. 82 shows that nondimensional heat fluxes reduce by 35% for flight conditions as compared to wind tunnel conditions. Inspection of the computed flow fields shows two flow phenomena which may be responsible for this behavior. Firstly, we observe large flow separations at the lateral edges of the elevon for the wind tunnel conditions which seem to form a modified effective slot shape with more rapid flow expansion, see Fig. 83. Secondly, the thicker boundary layers present in the flow solution for flight conditions, Fig. 84, tend to block the slot and hence, they reduce flow expansion and peak heating rates.

In conclusion we have successfully used 3D flow computations in the aerothermal analysis of winged reentry vehicles. These computations allow detailed understanding of critical flow phenomena and much more accurate transposition from wind tunnel to flight as compared to strategies used for the US-Orbiter twenty years ago. Consequently, uncertainties of data to be used to design the thermal protection system is considerably reduced which improves the weight of space planes.

6. CONCLUSION

Well established algorithms used in current blockstructured Euler/Navier-Stokes solvers for industrial applications have been reviewed. Attention has been focused on various spatial discretization and time stepping schemes. The approach of blockstructured meshes has been discussed in detail. It allows the treatment of complex configurations and forms the basis of parallelization of structured solvers. Special emphasis has been put on the implementation of multigrid within a blockstructured solver. Several large-scale computations have been shown which demonstrate the ability of current blockstructured flow solvers for 3D complex applications.

Acknowledgement

The authors want to thank Jiri Blazek, Olaf Brodersen, Ulrich Herrmann, Dr. Jose Longo and Arno Ronzheimer for their contributions to this report.

7. REFERENCES

[1] Eisfeld, B., Bleecke, H.-M., Kroll, N., Ritzdorf, H., "Parallelization of Block Structured Flow Solvers", AGARD FDP/VKI Special Coarse on Parallel Computing in CFD, VKI, Rhode-Saint-Genése, Belgium 15-19 May, 1995.

[2] Baldwin, B.S., Lomax, H., "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows", AIAA Paper 78-257, 1978.

[3] Mundt, Ch., Keraus, R., Fischer, J., "New Accurate Vectorized Approximations of State Surfaces for the Thermodynamik and Transport Properties of Equilibrium Air", ZFW, No. 15, 1991, pp. 179-184.

[4] Rossow, C.-C., "Berechnung von Strömungsfeldern durch Lösung der Euler-Gleichungen mit einer erweiterten Finite-Volumen Diskretisierungsmethode", DLR-FB 89-38, 1989.

[5] Kroll, N., Rossow, C.-C., "A High Resolution Cell Vertex TVD Scheme for the Solution of the Two- and Three-Dimensional Euler Equations", Lecture Notes in Physics, Vol. 371, Springer 1990, pp. 442-446.

[6] Allmaras, S.R., "Contamination of Laminar Boundary Layers by Artifical Dissipation in Navier-Stokes Solutions", Conference on Numerical Methods in Fluid Dynamics, Reading, UK, April 7-10, 1992.

[7] Swanson, R.C., Turkel, E., "Aspects of a High-Resolution Scheme for the Navier-Stokes Equations", AIAA Paper 93-3372-CP, 1993.

[8] Rossow, C.-C., Kroll, N., Radespiel, R., Scherr, S., "Investigation of the Accuracy of Finite Volume Methods for 2- and 3-Dimensional Flows.", AGARD-CPP-437, 1988, pp. 17.1-11.

[9] Jameson, A., Schmidt, W., Turkel, E., "Numerical Solutions of the Euler Equations by Finite Volume

Methods Using Runge-Kutta Time-Stepping Schemes", AIAA Paper 81-1259, 1981.

[10] Kroll, N., Jain, R.K., "Solution of Two-Dimensional Euler Equations - Experience with a Finite Volume Code", DFVLR-FB 87-41, 1987.

[11] Radespiel, R., "Computation of Two- and Three-Dimensional Sub- and Transonic Flow Fields", CCG-Course F6.03, 29-5-1.6.1989, Braunschweig, Germany.

[12] Kroll, N., Rossow, C.-C., "Foundatons of Numerical Methods for the Solution of Euler Equations", CCG-Course F6.03, 29.5-1.6.1989, Braunschweig, Germany.

[13] Martinelli, L., Jameson, A., "Validation of a Multigrid Method for the Reynolds-Averaged Navier-Stokes Equations", AIAA Paper 88-0414, 1988.

[14] Radespiel, R., Rossow, C.-C., Swanson, R.C., "Efficient Cell-Vertex Multigrid Scheme for the Three-Dimensional Navier-Stokes Equations", AIAA Journal, Vol. 28. No. 8, 1990.

[15] Cook, P.H., Mc Donald, M.A., Firmin, M.C.P., "Aerofoil RAE 2822-Pressure Distributions and Boundary Layer and Wake Meassurements", AGARD-AR-138, 1979.

[16] Schmitt, V., Charpin, F., "Pressure Distributions on the ONERA-M6-Wing at Transonic Mach Numbers", AGARD-AR-138, 1979.

[17] Longo, J.M.A., "Viscous Transonic Flow Simulation around a Transport Aircraft Configuration", DGLR-Jahrestagung, Bremen, 1992.

[18] Longo, J.M.A., "Simulation of Complex Inviscid and Viscous Vortex Flows", IUTAM Symposium on Fluid Dynamics of High Angle of Attack, 1992, Japan.

[19] Turkel, E., "Improving the Accuracy of Central Difference Schemes", 11th International Conference on Numerical Methods in Fluid Dynamics, Springer-Verlag, Lecture Notes in Physics, Vol. 323, 1988, pp. 586-591.

[20] Swanson, R.C., Turkel, E., "On Central-Difference and Upwind Schemes.", Journal of Comput. Phys., Vol. 101, No. 2, 1992, pp. 292-306.

[21] Brodersen, O., "Untersuchung einer Matrix-Dissipation in einem Zelleneckpunkt-Finite-Volumen-Schema zur Lösung der Navier-Stokes-Gleichungen",DLR-FB 92-33, 1992.

[22] Roe, P.L., "Approximate Riemann Solvers, Parameter Vectors and Difference Schemes", Journal of Computational Physics, Volume 43, 1981, pp. 357-372.

[23] Yee, H.C., Harten, A., "Implicit TVD Schemes for Hyperbolic Conservation Laws in Curvilinear Coordinates", AIAA Journal, Vol. 25, 1987, pp. 266-247.

[24] Steger, J.L., Warming, R.F., "Flux Vector Splitting of the Inviscid Gasdynamic Equations with Application to Finite Difference Methods", Journal of Computational Physics, Volume 40, 1981, pp. 263-293, 1981.

[25] Van Leer, B., "Flux-Vector Splitting for the Euler Equations", Lecture Notes in Physics, Vol. 170, 1982, pp. 507-512, Springer-Verlag.

[26] Yee, H.C., "Upwind and Symmetric Shock Capturing Schemes", NASA-TM 89464, 1987.

[27] Schöne, J., Kroll, N., Streit, Th., "Steps Towards an Efficient and Accurate Method Solving the Euler Equations around a Re-Entry Configuration at Supersonic and Hypersonic Speeds", Proceedings of the European Symposium on Hypersonics. ESTEC, ESA SP-318, 1991, pp 115-120.

[28] Yee, H.C., "A Class of High-Resolution Explicit and Implicit Shock Capturing Methods", VKI, Lecture Series LS-04, 1989.

[29] Radespiel, R., Swanson, R.C., "Progress with Multigrid Schemes for Hypersonic Flow Problems", ICASE Report, No. 91-89, 1991.

[30] Müller, B., "Simple Improvements of an Upwind TVD Scheme for Hypersonic Flow", AIAA Paper 89-1977-CP, 1989.

[31] Workshop on Hypersonic Flows for Reentry Problems, Part II, Antibes, France, 1991.

[32] Radespiel, R., "Computation of Hypersonic Flows over 2-D Ramps with a Multigrid Method.", Proceedings of Workshop on Hypersonic Flows for Reentry Problems, Part II., Antibes, April 15-19, 1991.

[33] Holden, M.S., "A Study of Flow Separation in Regions of Shock Wave-Boundary Layer Interaction in Hypersonic Flow", AIAA Paper 78-1169,

1978.

[34] Edney, B., "Anomalous Heat Transfer and Pressure Distributions on Blunt Bodies at Hypersonic Speeds in the Presence of an Impinging Shock", Technical Report 115, The Aeronautical Research Institute of Sweden, Stockholm, February 1968.

[35] Kroll, N., Gaitonde, D., Aftosmis, M., "A Systematic Comperative Study of Several High-Resolution Schemes for Complex Problems in High Speed Flows", AIAA Paper 93-0636, 1963.

[36] Radespiel, R., Poirier, D., Streit, Th., "Computation of Viscous Flows Around HERMES (1.0) at M=10", DLR-IB 129-92/3, 1992.

[37] Hänel, D., Schwane, R., "An Implicit Flux Vector Splitting Scheme for the Computation of Various Hypersonic Flows", AIAA Paper 89-0274, 1989.

[38] Liou, M.-S., Steffen, Ch., "A New Flux Splitting Scheme", Journal of Computational Physics, Vol. 107, No. 1, 1993, pp. 23-29.

[39] Coquel, F., Liou, M.-S., "Stable and Low Diffusive Hybrid Upwind Splitting Methods", Proceedings of the 1st European CFD Conference, Brussels, 1992.

[40] Liou, M.-S., "On a New Classl of Flux Splittings", Lecture Notes in Physics, Vol. 414, 1992, pp. 115-119, Springer Verlag.

[41] Kroll, N., Herrmann, U., Radespiel, R., "Discretization Properties in Hypersonic Flows", DLR-IB 129-92/28, 1993.

[42] Radespiel, R., Kroll, N., "Extension of the Navier-Stokes Code CEVCATS to Hypersonic Euilibrium Flows", DLR-IB 129-92/21, 1992.

[43] Bergamini, L., Cinnella, P., "A Comparison of "New" and "Old" Flux-Splitting Schemes for the Euler Equations", AIAA Paper 93-0876, 1993.

[44] Wada, Y., Liou, M.-S., "A Flux Splitting Scheme wiht High-Resolution and Robustness for Discontinuities", AIAA Paper 94-0083, 1994.

[45] Kroll, N., Radespiel, R., "An Improved Flux Vector Split Discretization Scheme for Viscous Flows", DLR-FB 129-93/53, 1993.

[46] Ventkatakristinan, V., "On the Accuracy of Limiters and Convergence to Steady State Solutions", AIAA Paper 93-0880, 1993.

[47] Van Leer, B., "Upwind Difference Methods for Aerodynamics Governed by the Euler Equations", In Lectures in Applied Mathematics, (B. Engmist, S. Osher, R. Sommerville eds.), Vol. 22, Part II, 1985, pp. 327-336.

[48] Krist, S.L., Thomas, J.L., Sellers, W.L, Kjelgaard, S.O., "An Embedded Grid Formulation Applied to a Delta Wing", AIAA Paper 90-0429, 1990.

[49] Stetson, K.F., Thompson, E.R, Donaldson, J.C., Siler, L.G., "Laminar Boundary-Layer Experiments on a Cone at Mach 8, Part 2: Blunt Cone", AIAA Paper , "Engineering Transition Prediction for a Hypersonic Axisymmetric Boundary Layer" AIAA Paper 93-5114, 1993.

[50] Stilla, J., "Engineering Transition Prediction for a Hypersonic Axisymmetric Boundary Layer", AIAA Paper 93-5114, 1993.

[51] Brandt, A., "Guide to Multigrid Development", Multigrid Methods I, Lecture Notes in Mathematics, No. 960, 1981.

[52] Lee, D.-Y., Van Leer, B., "Progress in Local Precondioning of the Euler and Navier-Stokes Equations", AIAA-93-3328, 1993.

[53] Turkel, E., "Review on Preconditioning Methods for Fluid Dynamics", ICASE Report No. 92-47, (unpublished).

[54] Godfrey, A.G., "Steps Toward a Robust Preconditioning", AIAA-94-0520, 1994.

[55] Blazek, J., Kroll, N., Radespiel, R., Rossow, C.-C., "Upwind Implicit Residual Smoothing Method for Multistage Schemes", AIAA-91-1533, 1991.

[56] Blazek, J., "Verfahren zur Beschleunigung der Lösung der Euler- und Navier-Stokes-Gleichungen bei stationären Über- und Hypserschallströmungen", Dissertation Braunschweit, Technical University, to appear 1994.

[57] Hackbush, W., "Multi-Grid Methods and Applications", Springer Verlag, 1985.

[58] Mulder, W.A., "A New Multigrid Approach to Convection Problems", Journal of Computational Physics, Vol. 83, 1989, pp. 303-323.

[59] Leclerq, M.-P., "Resolution Des Equations D'Euler par Des Methods Multigrilles Conditions Aux Lim-

ites En Regime Hypersonique", Doctoral Thesis, Speciality: Applied Mathematics, L'Universite De Saint-Etienne, France, April 1990.

[60] Eliasson, P., "Dissipation Mechanisms and Multigrid Solutions in a Multiblock Solver for Compressible Flow", Dissertation at the KTH, Stockholm, 1993.

[61] Jameson, A., "Multigrid Algorithms for Compressible Flow Calculations", MAE Report 1743, Princeton University, Text of Lecture given at 2nd European Conference on Multigrid Methods, Cologne, October 1985.

[62] Van Leer, B., Tai, C.-H., Powell, K.G., "Design of Optimally Smoothing Multi-Stage Schemes for the Euler Equations, AIAA Paper 89-1933-CP, 1989.

[63] Radespiel, R, Rossow, C.-C., Swanson, R.-C., "An Efficient Cell-Vertex Multigrid Scheme for the Three Dimensional Navier-Stokes Equations", AIAA Journal, Vol. 28, No. 8, 1990, pp. 1464-1472.

[64] Swanson, R.C., Turkel, E., White, J.A., "An Effective Multigrid Method for High-Speed Flows", Fifth Copper Mountain Conference on Multigrid Methods, Colorado, March 31 - April 5, 1991.

[65] Tai, C.-H., private communication, 1990.

[66] Thomas, J.L., "An Implicit Multigrid Scheme for Hypersonic Strong-Interaction Flowfields", 5th Copper Mountain Conference on Multigrid Methods, Denver, USA, 1991.

[67] Schröder, W., Hänel, D., "An Unfactored Implicit Scheme with Multigrid Acceleration for the Solution of the Navier-Stokes Equations", Journal of Computers and Fluids, No. 15, 1987, p. 315.

[68] Caughey, D., "Diagonal Implicit Multigrid Algorithm for the Euler Equations", AIAA Journal, Vol. 26, 1988, pp. 841-851.

[69] Yoon, S., Jameson, A., "Lower-Upper Implicit Schemes with Multiple Grids for the Euler Equations", AIAA Journal, Vol. 7, 1987, pp. 929-935.

[70] Blazek, J., "A Multigrid LU-SSOR Scheme for the Solution of Hypersonic Flow Problems", AIAA 94-0062, 1994.

[71] Yoon, S., Jameson, A, "An LU-SSOR Scheme for the Euler and Navier-Stokes Equations", AIAA Journal, Vol. 26, 1988, pp. 1025-1026.

[72] Rieger, H., Jameson, A., "Solution of Steady Three-Dimensional Compressible Euler and Navier-Stokes Equations by an Implicit LU-Scheme", AIAA Paper 88-0619, 1988.

[73] Yoon, S., Kwak, D., "An Implicit Three Dimensional Navier-Stokes Solver for Compressible Flow", AIAA Paper 91-1555, 1991.

[74] Yoon, S., Chang, L., Kwak, D., "Multigrid Convergence of an Implicit Symmetric Relaxation Scheme", AIAA Paper 93-3357-CP, 1993.

[75] Blazek, J., "Investigations of the Implicit LU-SSOR Scheme", DLR-FB 129-93/51, 1993.

[76] Radespiel, R., Swanson, R.-C., "Progress with Multigrid Schemes for Hypersonic Flow Problems", ICASE Report No. 91-89, 1991.

[77] Atkins, H.L., "A Multi-Block Multigrid Method for the Euler- and Navier-Stokes Equations for Three-Dimensional Flows", AIAA Paper 91-0101, 1991.

[78] Rossow, C.-C., "Efficient Computation of Inviscid Flow Fields Around Complex Configurations Using a Multiblock Multigrid Method", Communications in Applied Numerical Methods, Vol. 8, 1992, pp. 735-747.

[79] Ronzheimer, A, Brodersen, O., Rudnik, R., Findling, A., Rossow, C.-C., "A New Interactive Tool for the Management of Grid Generation Processes Around Arbitrary Configurations", To be published at: 4th International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Swansea, GB, 6-8th April, 1994.

[80] Hoheisel, H., Kiock, R.; Rossow, C.-C., Ronzheimer, A., Baumert, W., Capdevila, H., "Aspects of Theoretical and Experimental Investigations on Airfram/Engine Integration Problems", ICAS-Congress, Paper 90-2.7.3, 1990.

[81] Rill, S., Becker, K., "Simulation of Transonic Flow over Twin-Jet Transport Aircraft", Journal of Aircraft, Vol. 29, 1992, pp. 640-653.

[82] Rivoire, V., Vigneron, V., "CFD Environment for Aerodynamic Design at Aerospatiale Avions", Cray Channels, Vol. 15, No. 2, 1993, pp. 5-7.

[83] Rossow, C.-C., Godard, J.-L., Hoheisel, H., Schmidt, V., "Investigation of Propulsion Airfram Integration Interference Effects on a Transport Air-

craft Configuration,", AIAA Paper 92-3097, 1992.

[84] Brodersen, O., Rossow, C.-C., "Calculation of Interference Phenomena for a Transport Aircraft Configuration Considering Viscous Effects", European Forum Conference on Recent Developments on Applications in Aeronautical CFD, Bristol, GB, September 1993.

[85] Longo, J.M.A., Radespiel, R., "Analysis of Aerodynamic Flap Efficiency and Flap Heating of a Winged Reentry Vehicle", DGLR Jahrestagung 1993, Sept. 28-Oct.1, Göttingen, to appear in DGLR Jahrbuch 1993.

[86] Herrmann, U., Radespiel, R., Longo, J.M.A., "Critical Flow Phenomena on the Winglet of Winged Reentry Vehicles", AIAA 94-0629, 1994.

[87] Miller, C.G., Micol, J.R., Gnoffo, P.A., Wilder, S.E., "Heat Transfer Distributions on Biconics at Incidence in Hypersonic-Hypervelocity He, N_2, N_z, Air, and CO_2 Flows", NASA TM 84667, 1983.

8. FIGURES

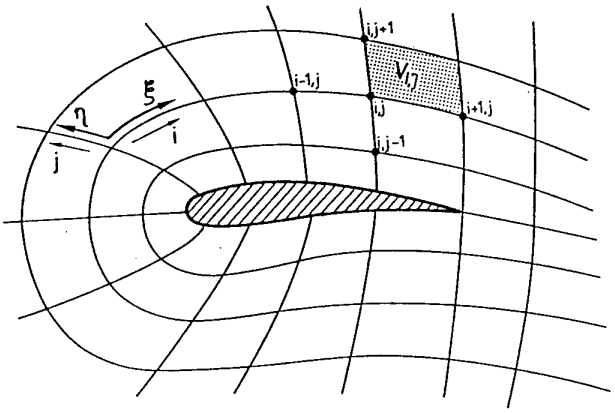
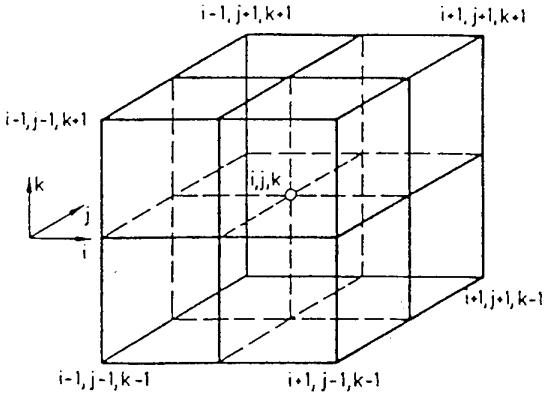


Fig. 1 Body-fitted grid



a) control volume

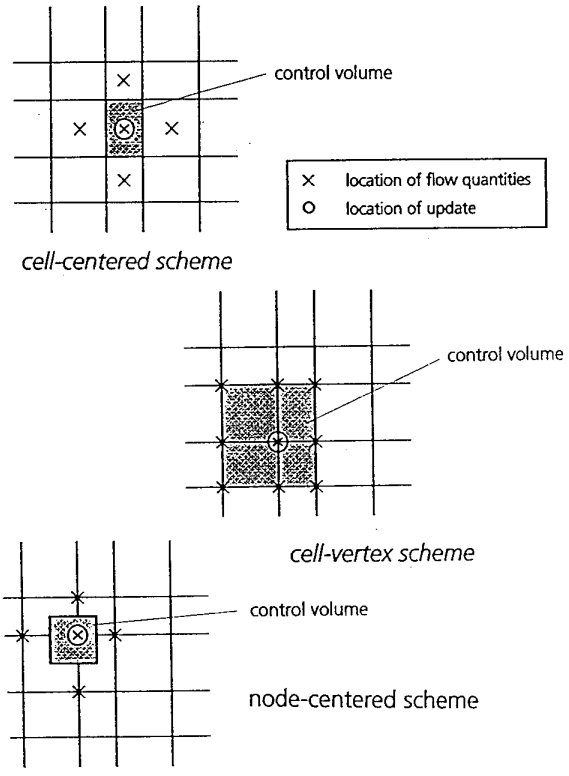
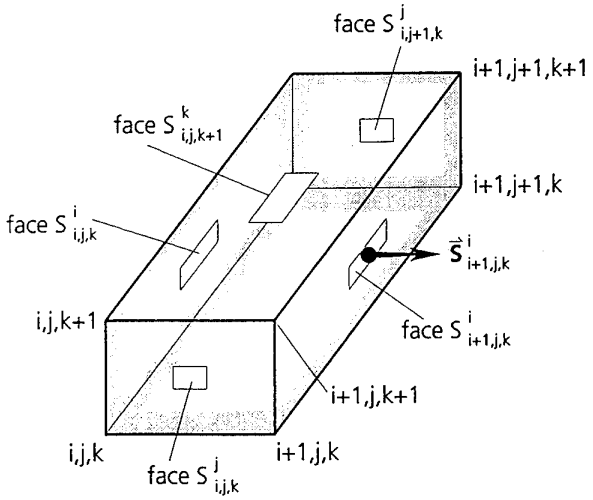


Fig. 2 Control volume arrangements for finite volume schemes



b) metric definition of a computational cell

Fig. 3 Cell-vertex scheme

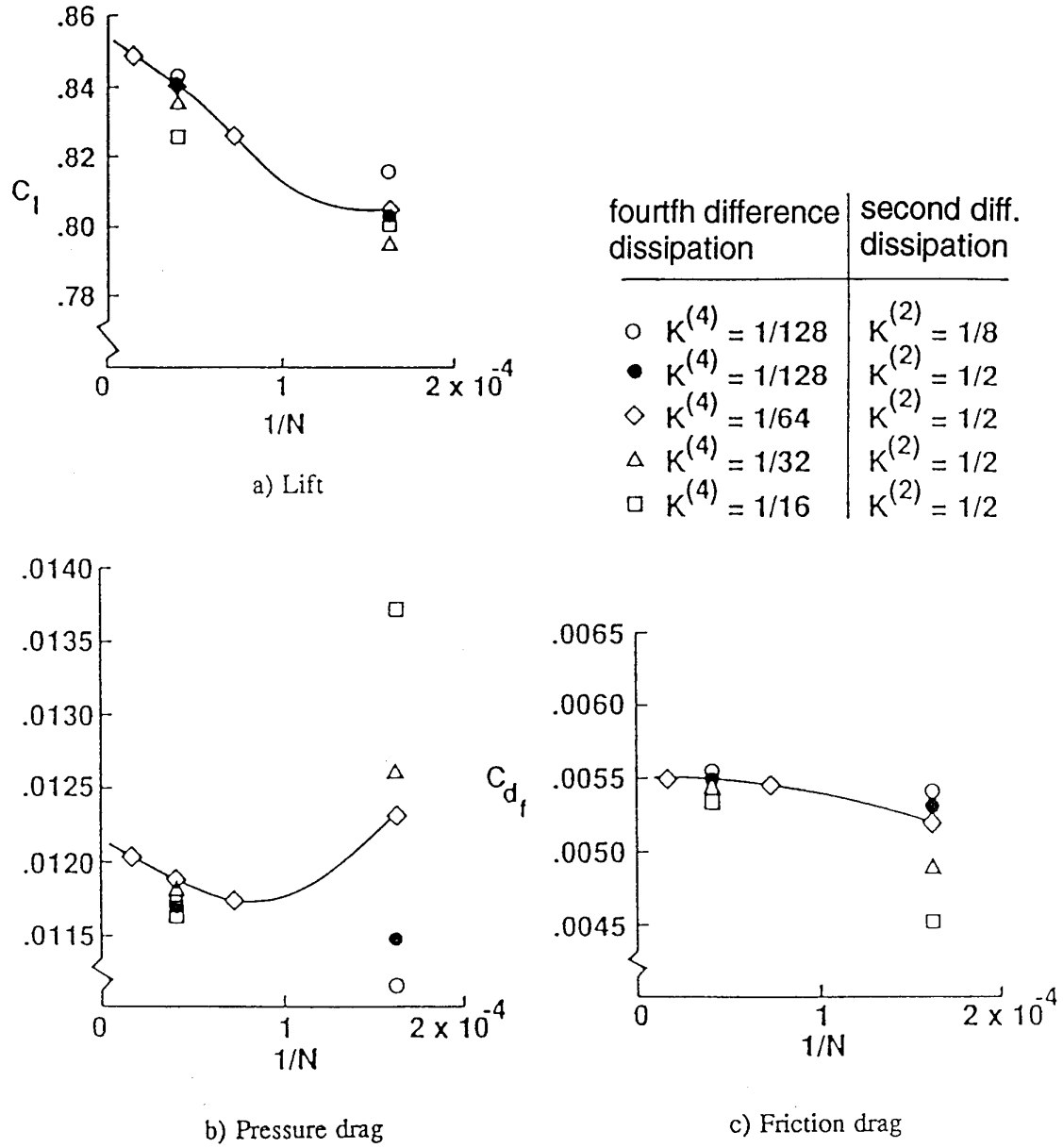


Fig. 4 Influence of grid density and artificial viscosity on global force coefficients for flow around RAE 2822 airfoil (M_∞ =0.73, α=2.790°, Re_∞ =6.5x10^6), central scheme with scalar dissipation

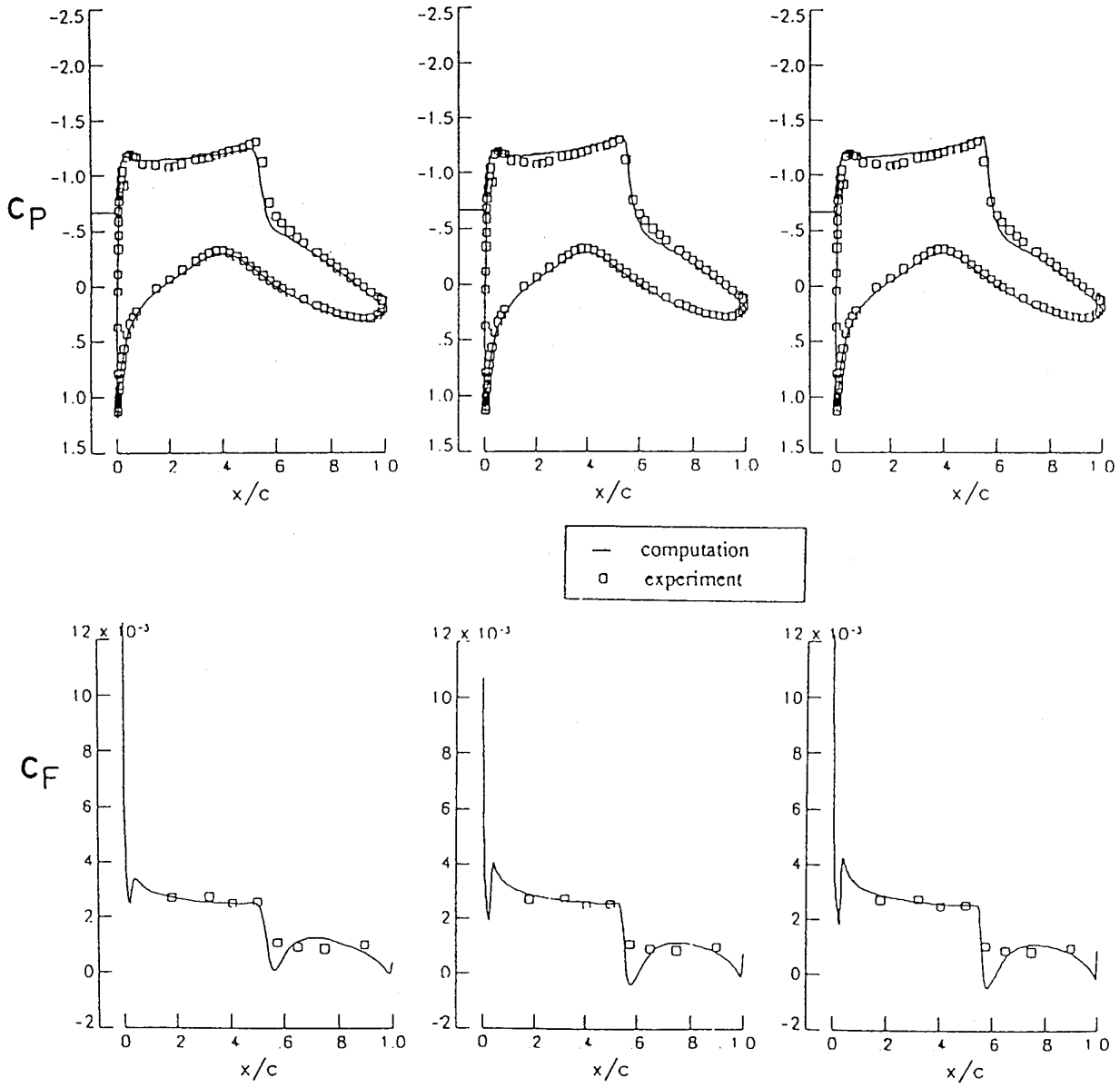


Fig. 5 Influence of grid density on distributions of pressure and skin friction along RAE 2822 airfoil (M_∞ =0.73, α=2.79°, Re_∞ =6.5x10^6), central scheme with scalar dissipation

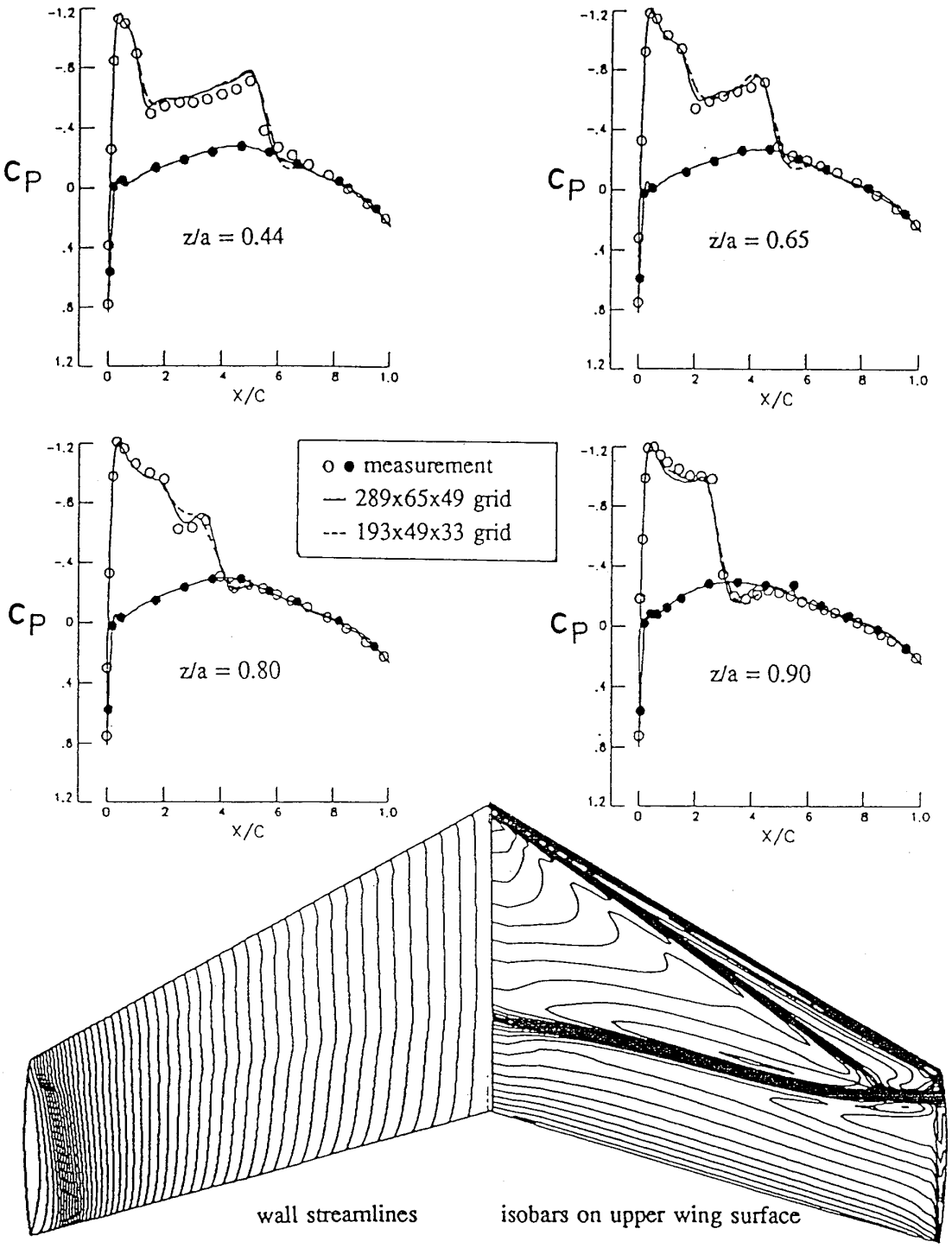


Fig. 6 Surface pressure distributions and wall streamline for ONERA-M6 wing (M_∞ =0.84, α=3.06°, Re_∞ =11x10^6), central scheme with scalar dissipation

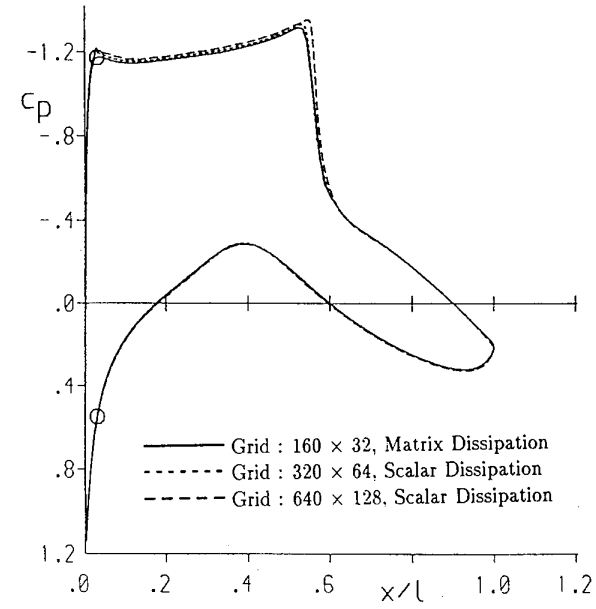


Fig. 7 Pressure distributions along RAE 2822 airfoil (M_∞ =0.73, α=2.79°, Re_∞ =6.5x10^6), comparison of central schemes with scalar and matrix-valued dissipation

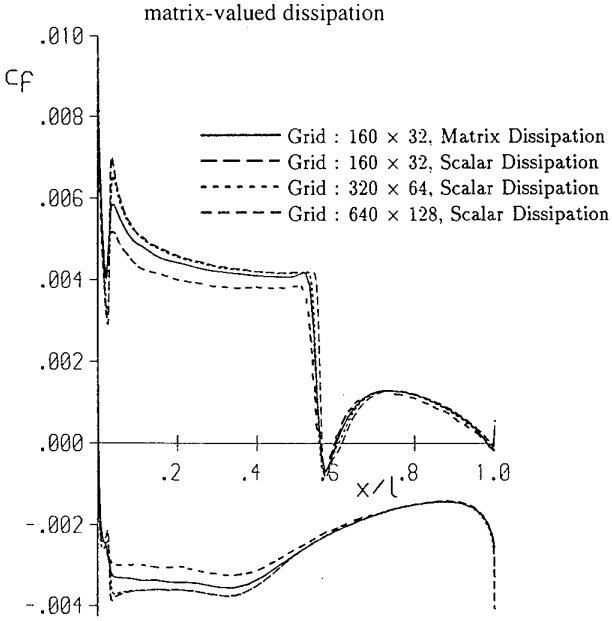


Fig. 8 Skin friction distributions along RAE 2822 airfoil (M_∞ =0.73, α=2.79°, Re_∞ =6.5x10^6), comparison of central schemes with scalar and matrix-valued dissipation

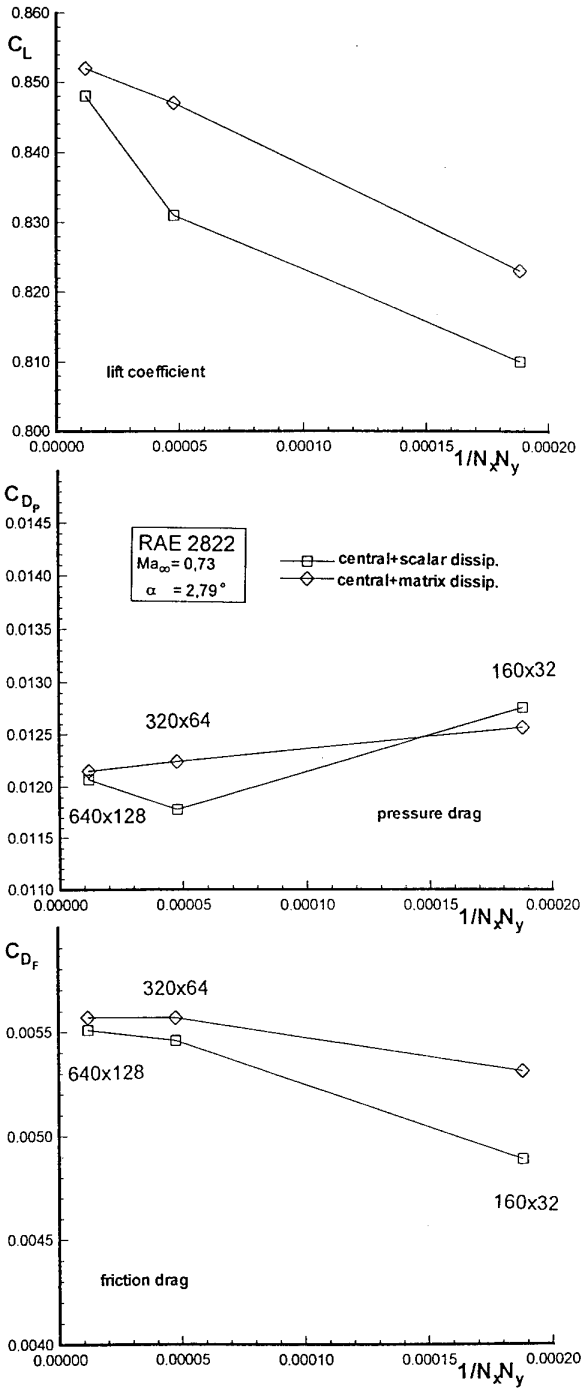


Fig. 9 Influence of grid density on global force coefficients for flow around RAE 2822 airfoil (M_∞ =0.73, α=2.79°, Re_∞ =6.5x10^6), comparison of scalar and matrix-valued dissipation

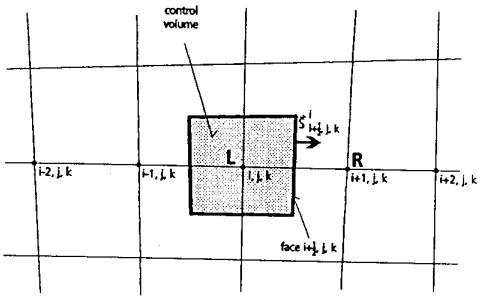


Fig. 10 Control volume for node-centered schemes

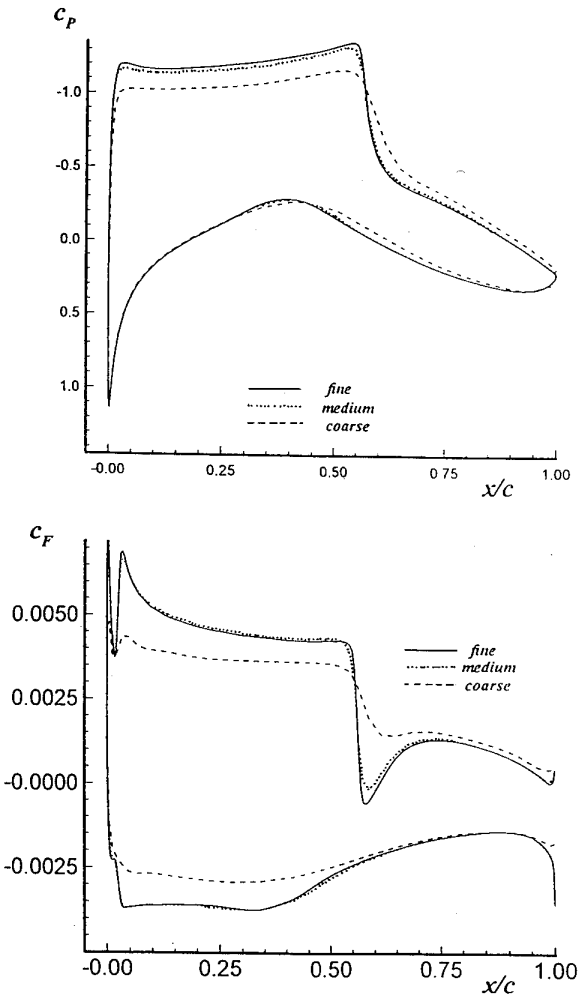


Fig. 11 Influence of grid density on distributions of pressure and skin frictions along RAE 2822 airfoil (M_∞ =0.73, α=2.79°, Re_∞ =6.5x10^6), upwind TVD scheme

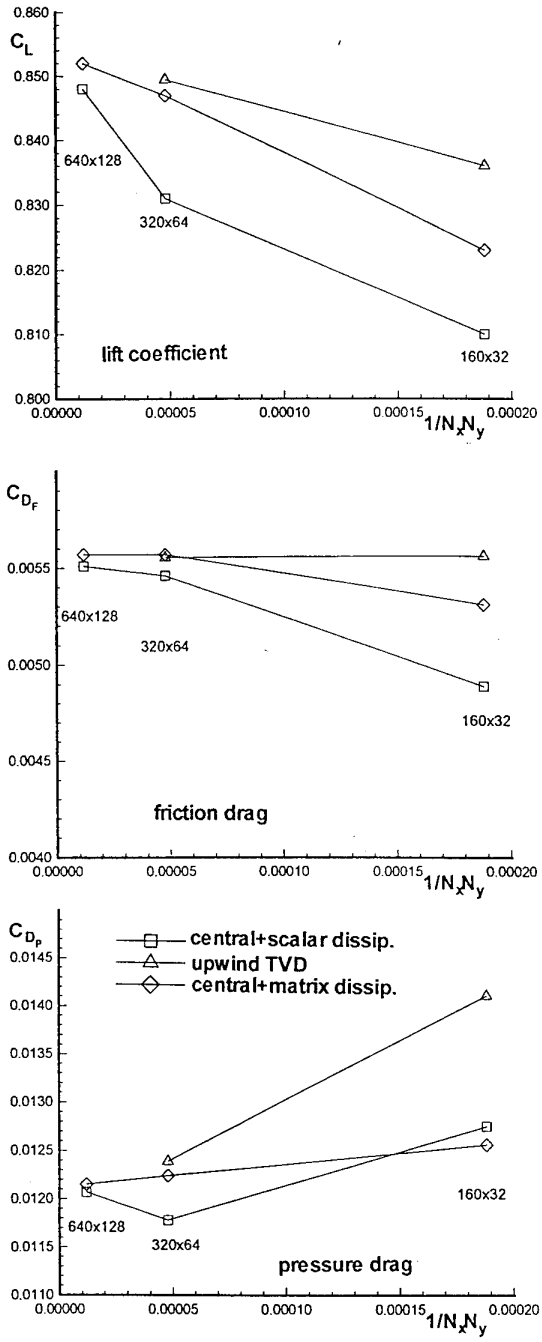


Fig. 12 Influence of grid density on global force coefficients for flow around RAE 2822 airfoil (M_∞ =0.73, α=2.79°, Re_∞ =6.5x10^6), upwind TVD scheme

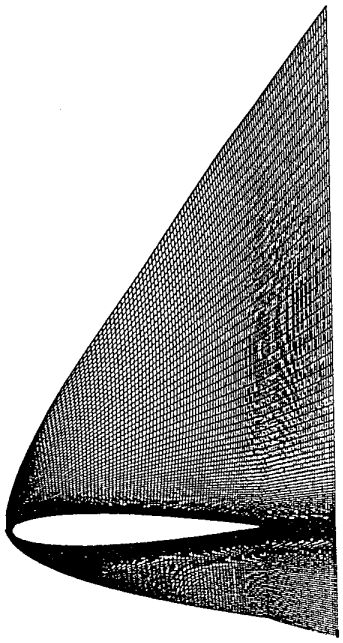


Fig. 13 Coordinate mesh with 256x80 cells for NACA 0012 airfoil at M_∞ =25, α=30°

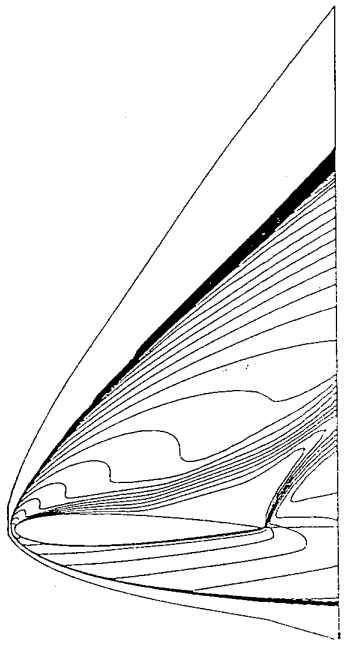


Fig. 14 Mach contours for viscous flow around NACA 0012 airfoil (M_∞ =25, α=30°)

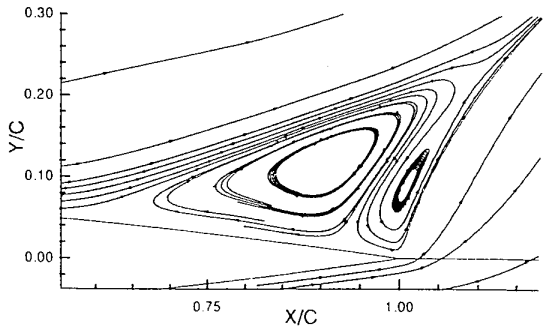


Fig. 15 Streamlines in separated flow region, NACA 0012 airfoil (M_∞ =25, α=30°), upwind TVD scheme

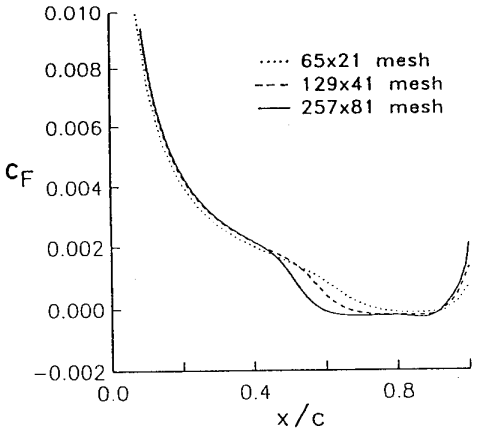


Fig. 16 Skin friction distribution along NACA 0012 airfoil (M_∞ =25, α=30°), upwind TVD scheme

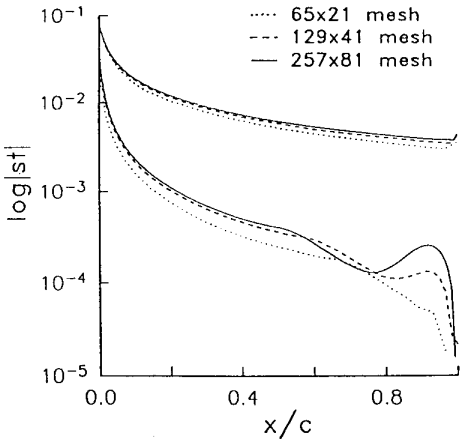


Fig. 17 Distribution of Stanton number along NACA 0012 airfoil (M_∞ =25, α=30°), upwind TVD scheme

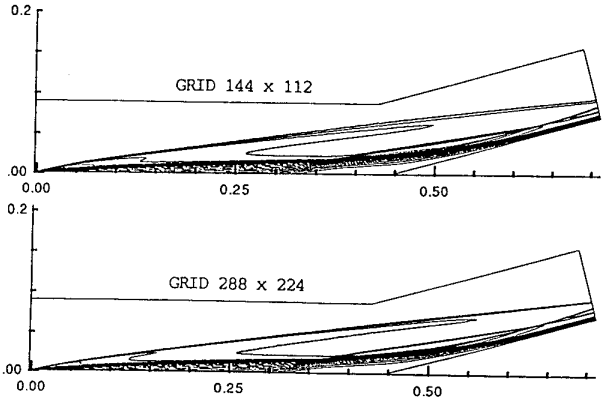


Fig. 18 Mach contours (ΔM=0.5) for 15° compression ramp (M_∞ =11.68, Re_c=2.47x10^5), upwind TVD scheme

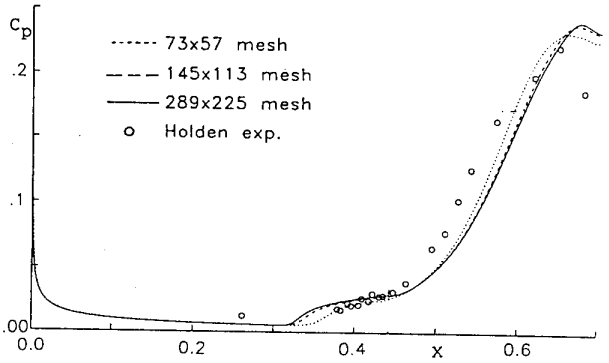


Fig. 19 Influence of grid density on pressure coefficient for 15° compression ramp (M_∞ =11.68, Re_c=2.47x10^5), upwind TVD scheme

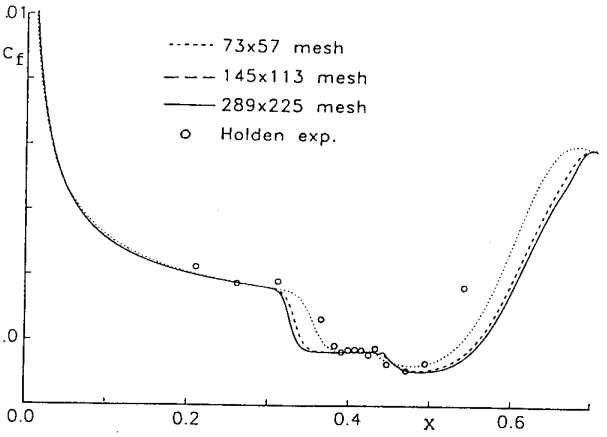


Fig. 20 Influence of grid density on skin friction coefficient for 15° compression ramp (M_∞ =11.68, Re_c=2.47x10^5), upwind TVD scheme

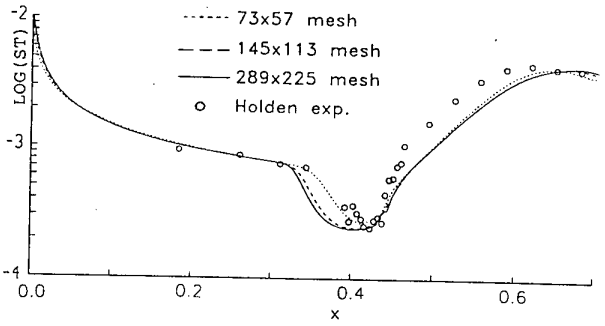


Fig. 21 Influence of grid density on Stanton number for 15° compression ramp (M_∞ =11.68, Re_c=2.47x10^5), upwind TVD scheme

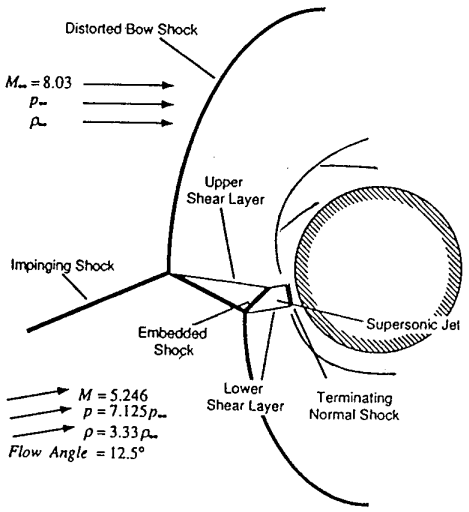


Fig. 22 Schematic of Type IV shock-shock interaction of Edney

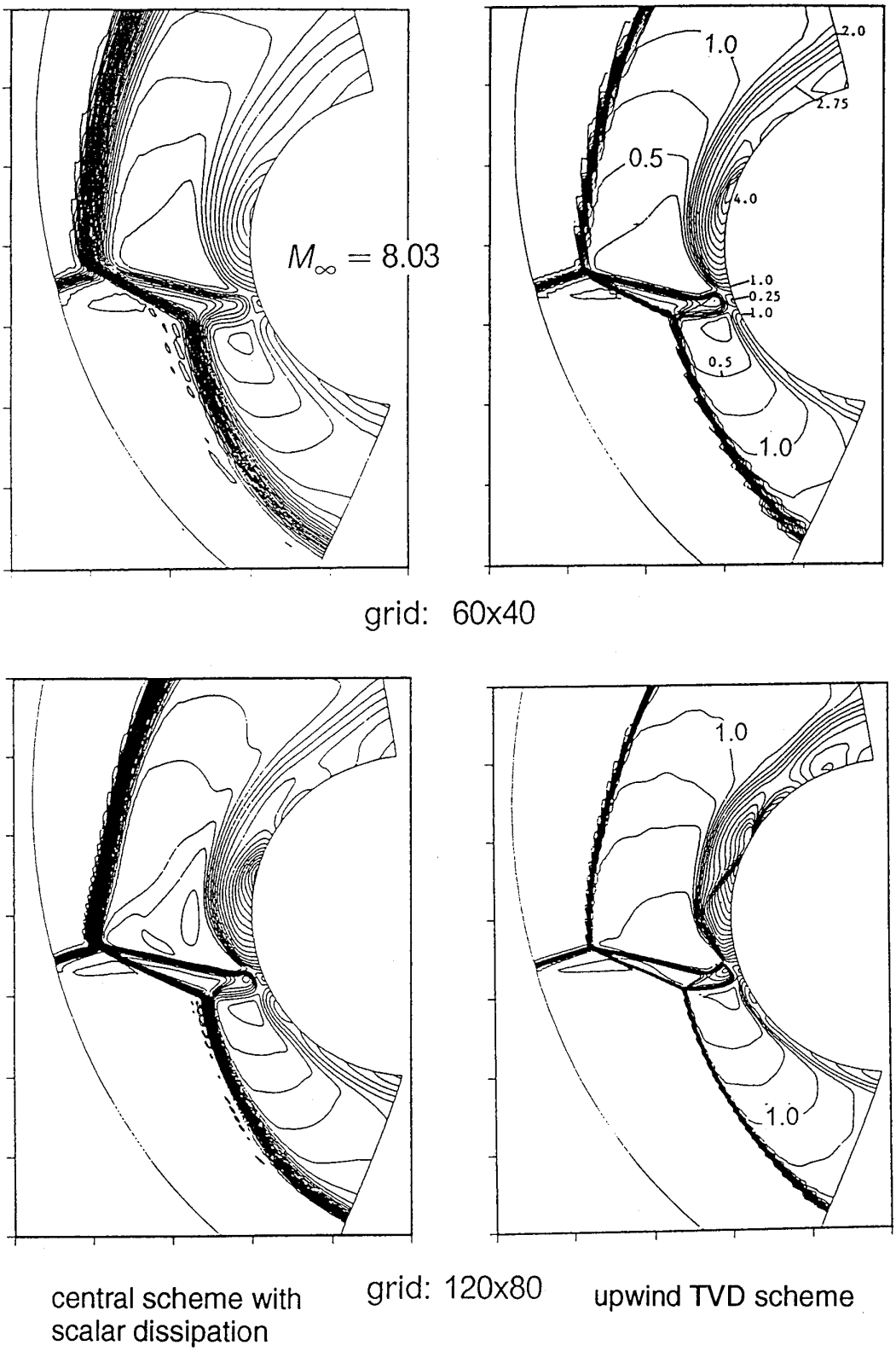


Fig. 23 Mach contour plots for Type IV interaction, comparison of central scheme with scalar dissipation and upwind TVD scheme, coarse and fine mesh

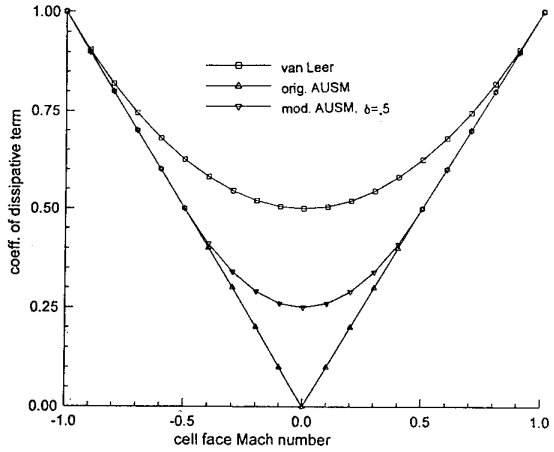


Fig. 24 Dissipation coefficient for flux vector split methods for Mach number close to zero

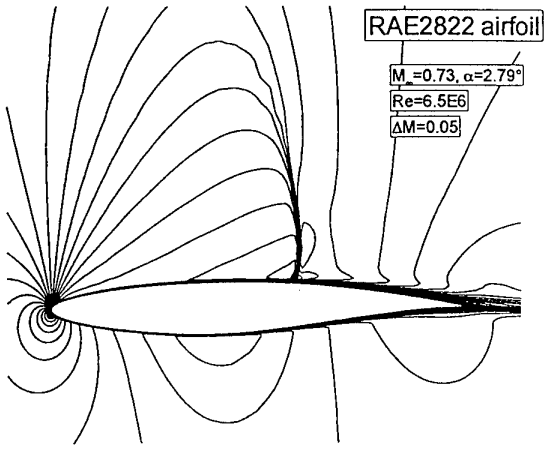


Fig. 25 b) Mach contours for RAE 2822 airfoil (M_∞ =0.73, α=2.79°, Re_∞ =6.5x10^6), improved AUSM

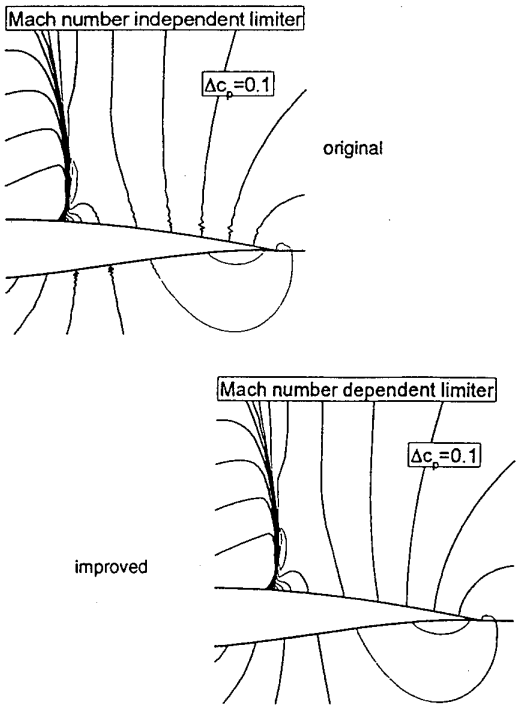


Fig. 25 a) Pressure contours for RAE 2822 airfoil (M_∞ =0.73, α=2.79°, Re_∞ =6.5x10^6), original and improved AUSM

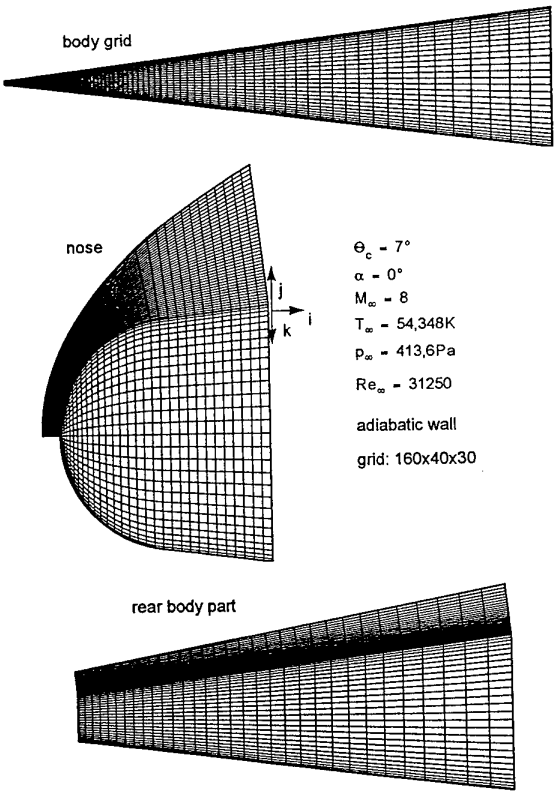


Fig. 26 Grid around blunt slender cone with spherical nose shape

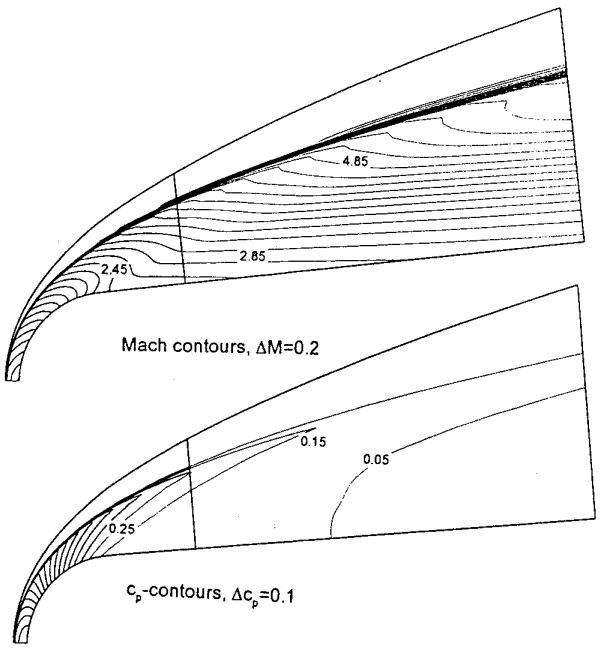


Fig. 27 Mach contours and pressure contours for blunt cone (M_∞ =8, α=0°), improved AUSM

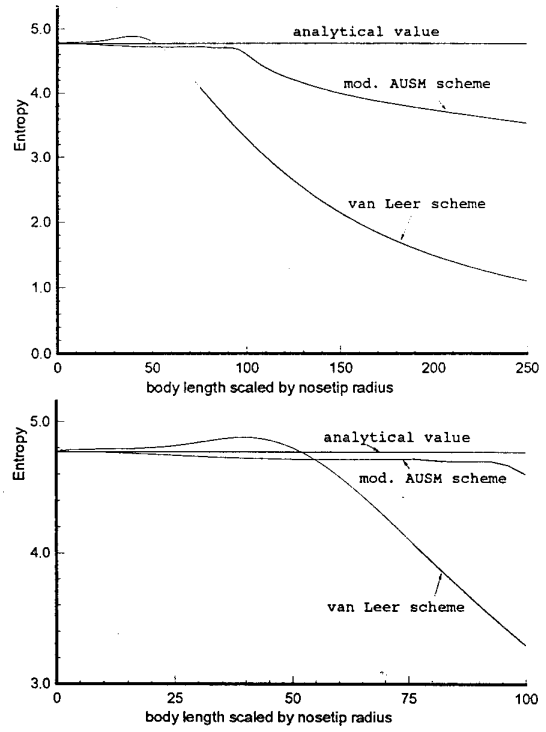
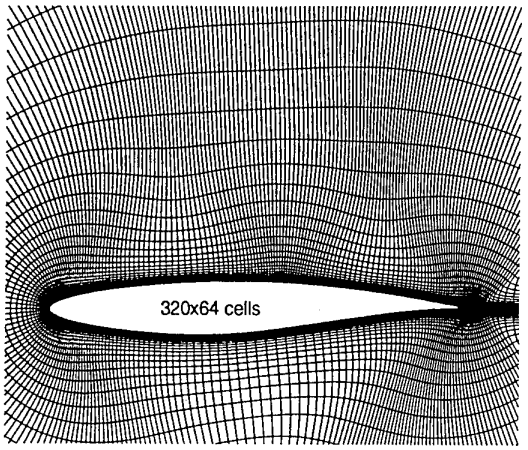


Fig. 28 Entropy value along the body, blunt cone (M_∞ =8, α=0°), comparison of van Leer scheme and improved AUSM



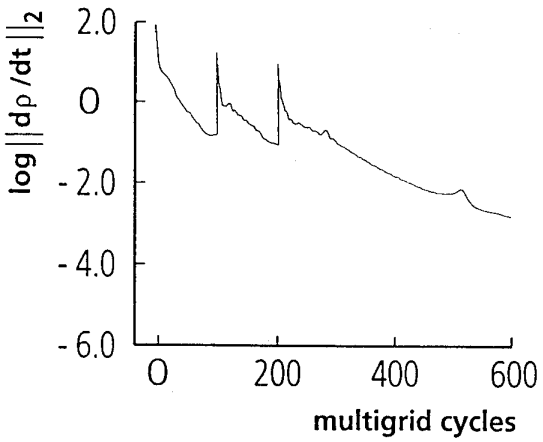
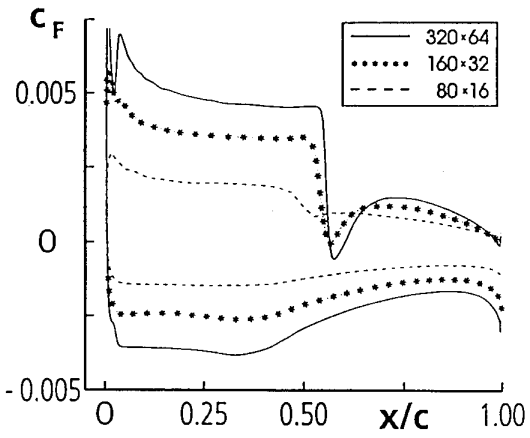
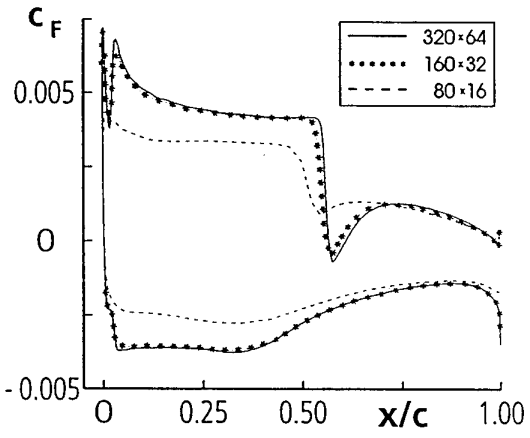


Fig. 29 Convergence history for improved AUSM for flow around RAE 2822 airfoil (M_∞ =0.73, α=2.79°, Re_∞ =6.5x10^6)

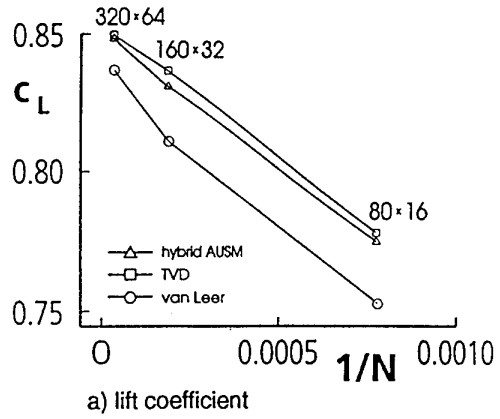


a) van Leer flux vector splitting

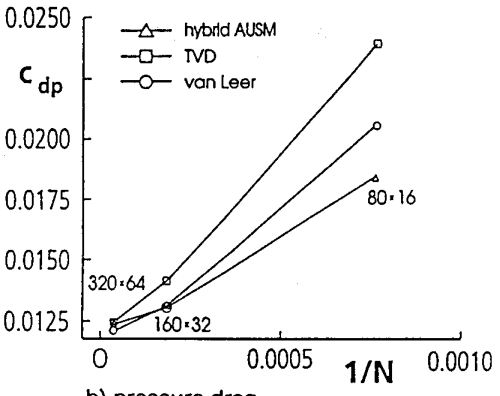


b) improved AUSM

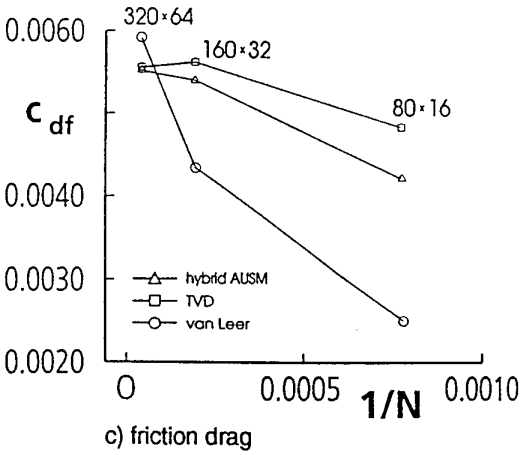
Fig. 30 Distributions of skin friction along RAE 2822 airfoil (M_∞=0.73, α=2.79°, Re_∞ =6.5x10^6), comparison of van Leer scheme and improved AUSM



a) lift coefficient

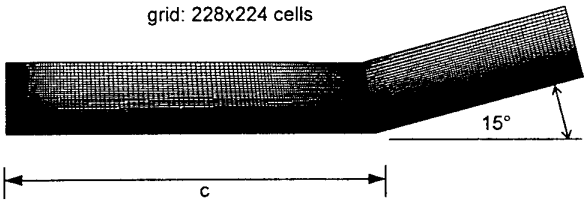


b) pressure drag



c) friction drag

Fig. 31 Grid convergence of force coefficients for RAE 2822 airfoil (M_∞=0.73, α=2.79°, Re_c=6.5x10^6), comparison of different upwind schemes



Grid for 15° compression ramp

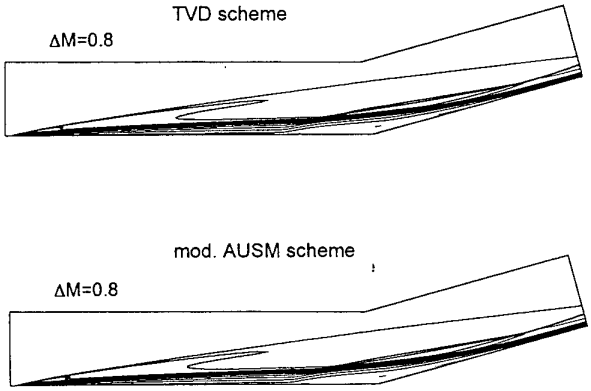


Fig. 32 Grid and Mach contours for 15° compression ramp (M_∞ =11.68, Re_c=2.47x10^5), comparison of upwind TVD and improved AUSM

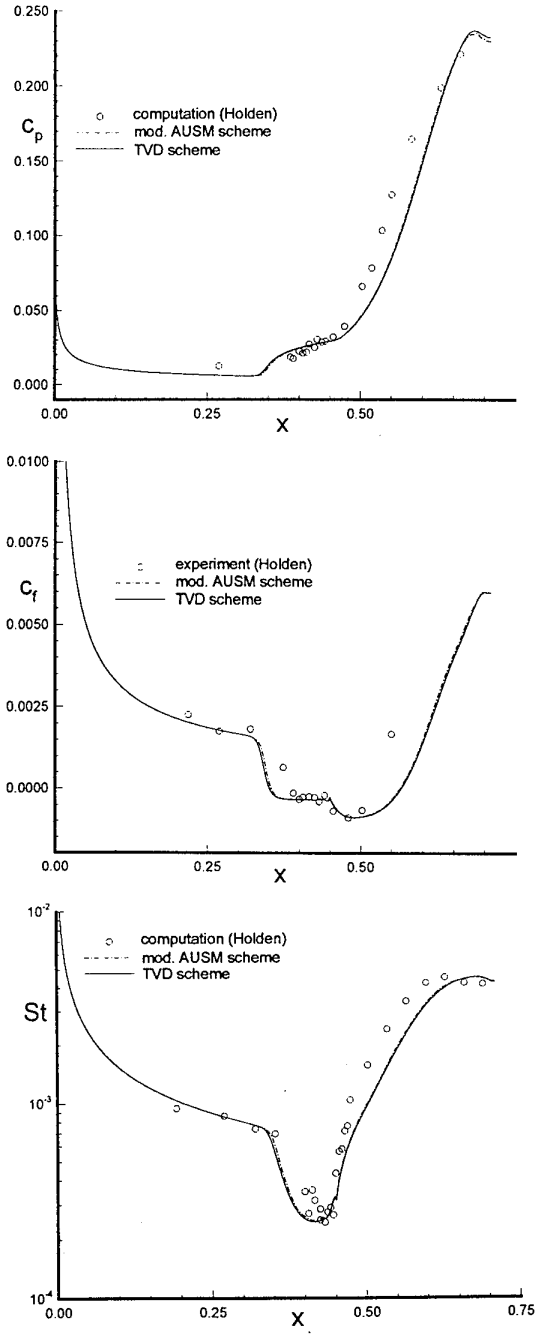


Fig. 33 Pressure coefficient, skin friction and Stanton number for 15° compression ramp (M_∞ =11.68, Re_c=2.47x10^5), comparison of upwind TVD and improved AUSM

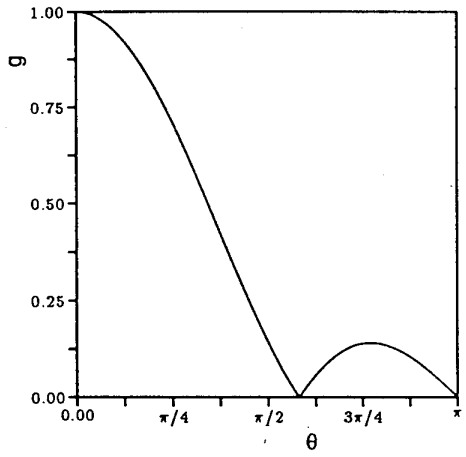


Fig. 34 Amplification factor for 1D convection problem, 3-stage scheme, CFL = 1.5, coefficients: 0.1481, 0.4, 1.0, according to [59]

$\gamma_{00} = 1,$

$\gamma_{10} = 1, \quad \gamma_{11} = 0,$

$\gamma_{20} = \Gamma_3, \quad \gamma_{21} = 0, \quad \gamma_{22} = \bar{\gamma}_3,$

$\gamma_{30} = \Gamma_3, \quad \gamma_{31} = 0, \quad \gamma_{32} = \bar{\gamma}_3, \quad \gamma_{33} = 0,$

$\gamma_{40} = \Gamma_3\Gamma_5, \quad \gamma_{41} = 0, \quad \gamma_{42} = \bar{\gamma}_3\Gamma_5, \quad \gamma_{43} = 0, \quad \gamma_{44} = \bar{\gamma}_5,$

$\Gamma_3 = (1 - \bar{\gamma}_3), \ \Gamma_5 = (1 - \bar{\gamma}_5), \ \bar{\gamma}_3 = 0.56$, and $\bar{\gamma}_5 = 0.44$.

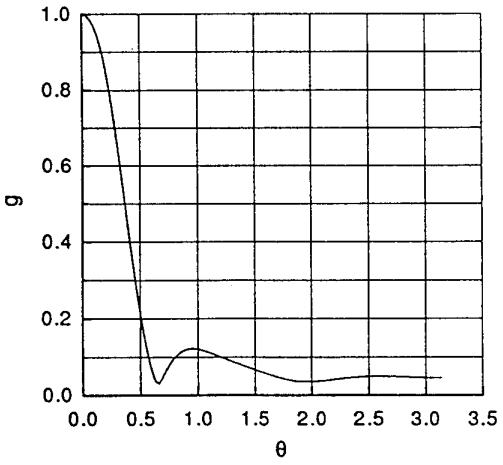
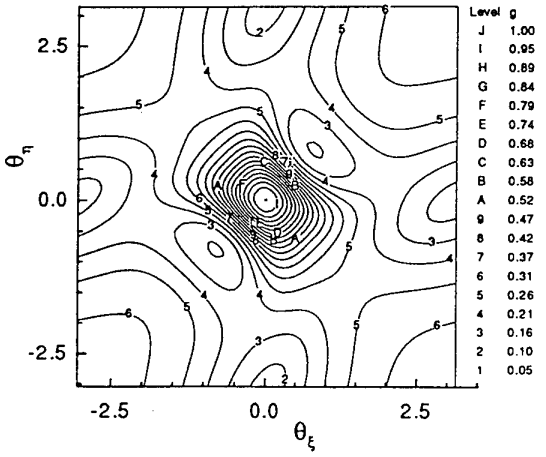
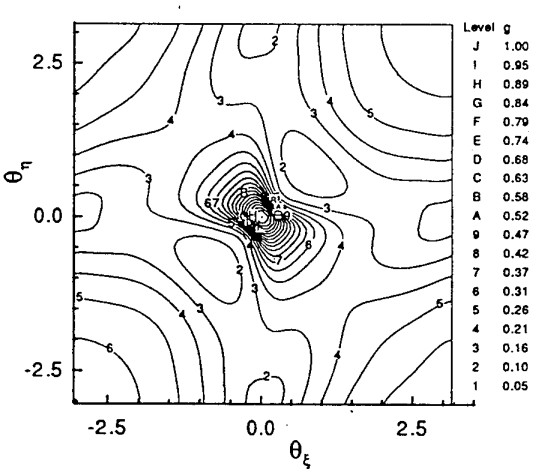


Fig. 35 Amplification factor for 1D convection problem, 5-stage scheme with 3 evaluations of weighted dissipation and residual smoothing, CFL = 5.0, $\beta = 1.0$, coefficients: 0.2742, 0.2067, 0.5020, 0.5142, 1.0

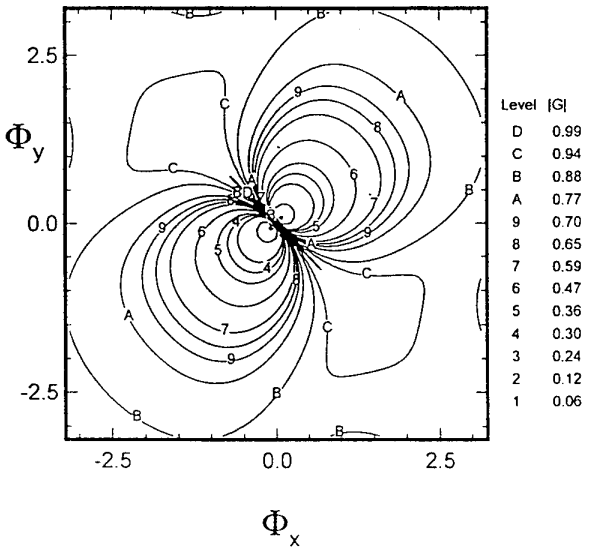


(a) One level, CFL = 5.0

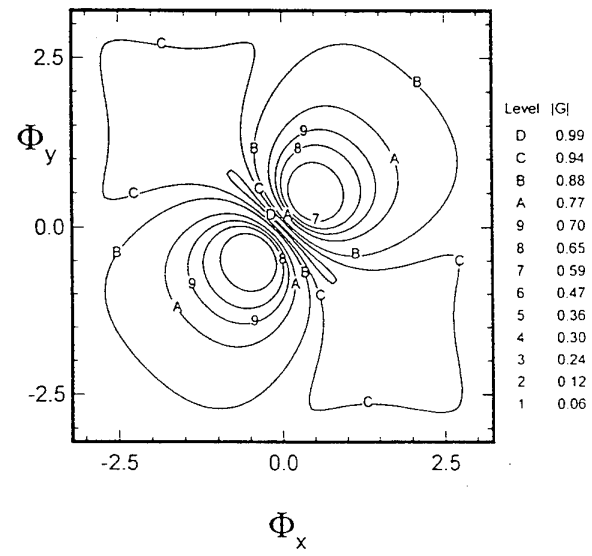


(b) Two levels, full coarsening, CFL = 5.0

Fig. 36 Contour plots of amplification factor for 2D convection problem and all aspect ration = 1. Five stage scheme with 3 evaluations of weighted dissipation and implicit residual smoothing

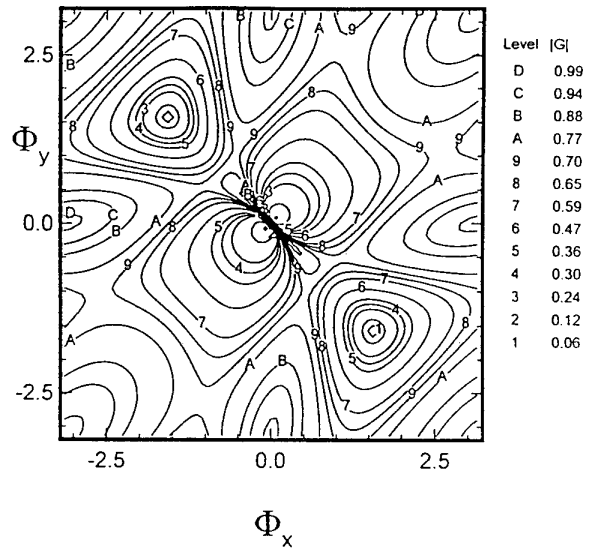


(a) implicit dissipation coefficient $\omega = 1.0$

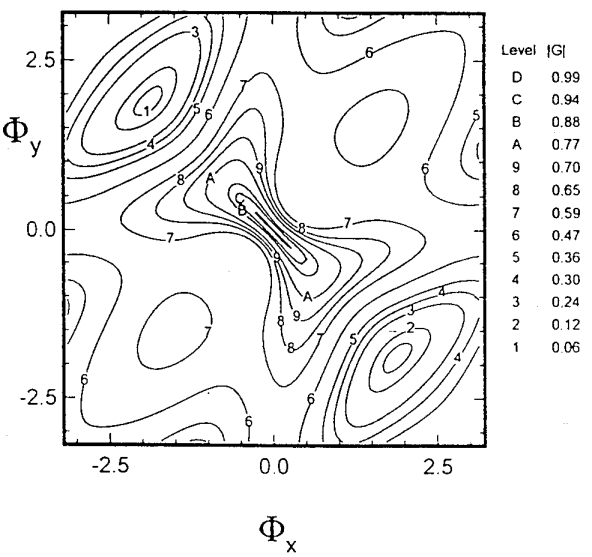


(b) implicit dissipation coefficient $\omega = 1.3$

Fig. 37 Contour plots of the amplification factor for LU-SSOR single-grid scheme with central discretization of the explicit operator



(a) implicit dissipation coefficient $\omega = 1.0$



(b) implicit dissipation coefficient $\omega = 1.3$

Fig. 38 Contour plots of the amplification factor for LU-SSOR single-grid scheme with second-order upwind spatial discretization of the explicit operator

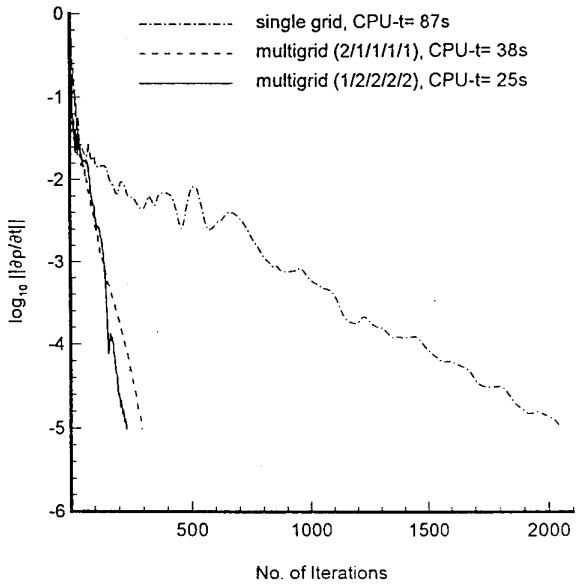


Fig. 39 Convergence histories for LU-SSOR scheme for inviscid flow around NACA 0012 airfoil (M_∞=0.8, α=1.25°), numbers in parentheses indicate the number of time steps on each grid starting from the fines one

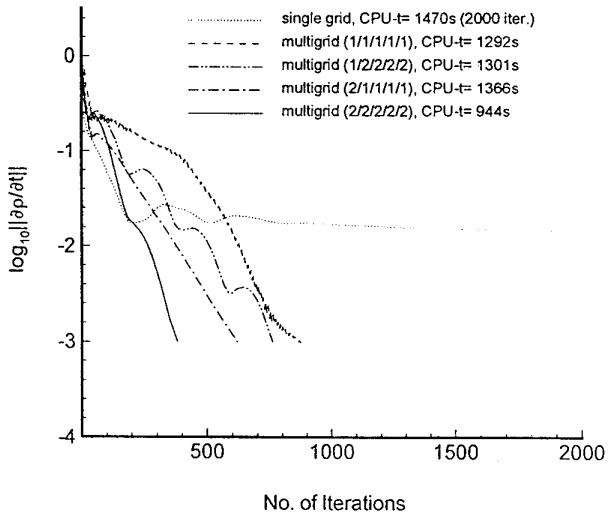


Fig. 40 Convergence histories for LU-SSOR scheme for viscous flow over 15° compression ramp, numbers in pharantheses indicate the number of

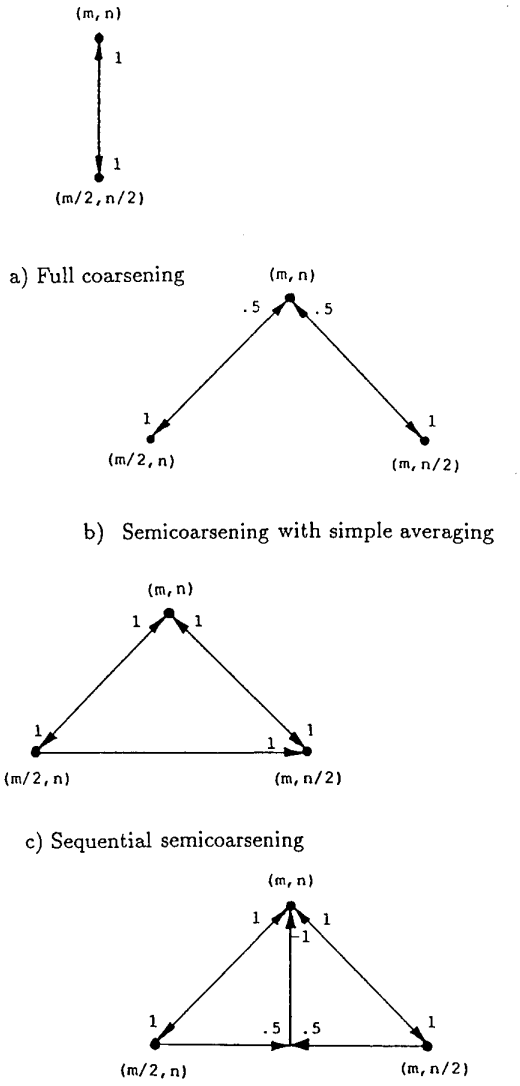
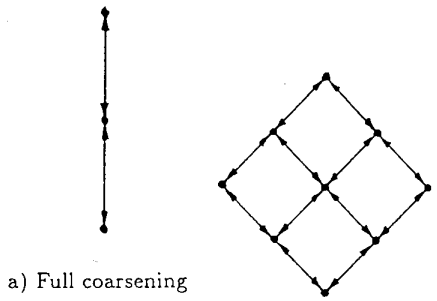
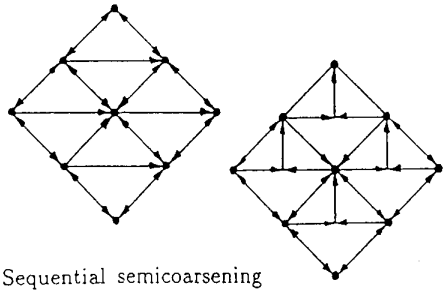


Fig. 41 Two-level multigrid schemes



a) Full coarsening

b) Semicoarsening with simple averaging



c) Sequential semicoarsening

d) Semicoarsening with selective averaging

Fig. 42 Multilevel schemes

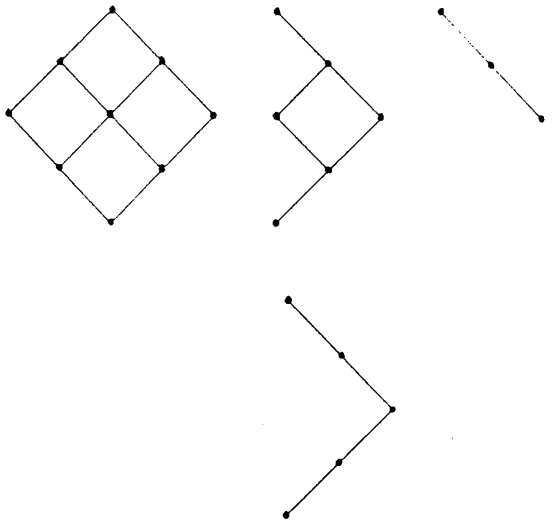
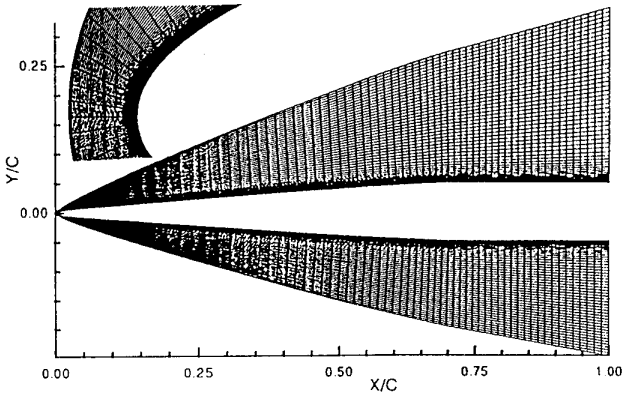
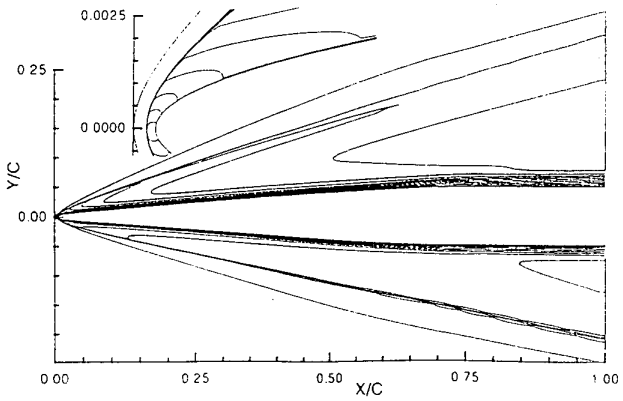


Fig. 43 Semicoarsening with selection of coarse meshes



Coordinate mesh for forebody with 256x96 cells



Mach contours for turbulent forebody

Fig. 44 Two-dimensional turbulent flow over forebody, $M_\infty = 7$, $\alpha = 5°$, $Re = 200 \times 10^6$, adiabatic wall

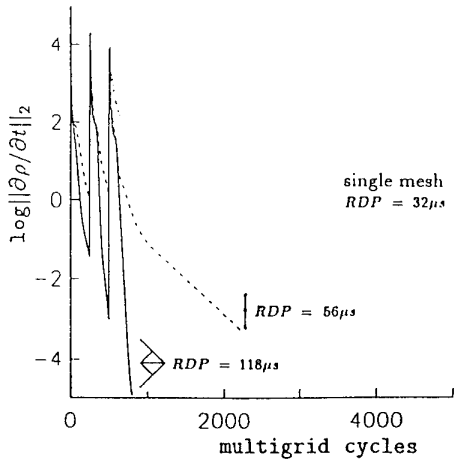


Fig. 45 Convergence histories for single grid and multigrid flow computations of turbulent forebody

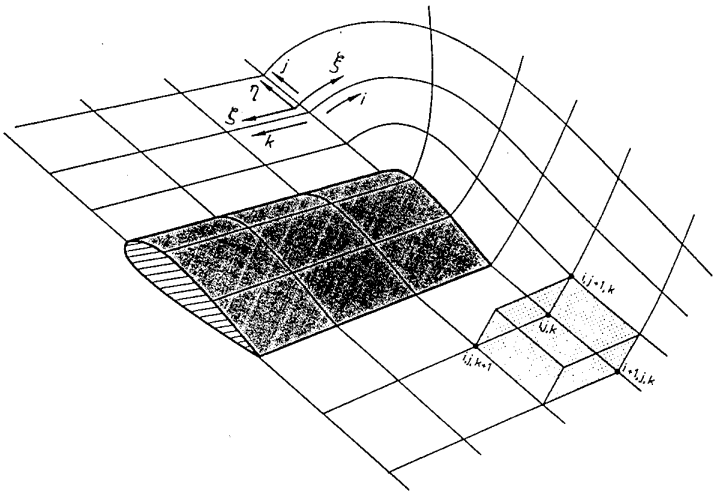


Fig. 46 Structured, three-dimensional, body-fitted mesh

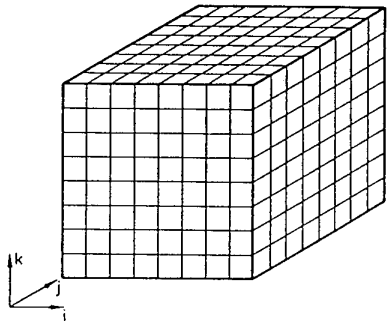


Fig. 47 Three-dimensional computational domain

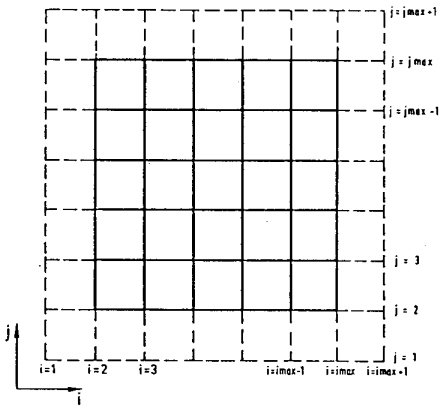


Fig. 48 Concept of fictitious points

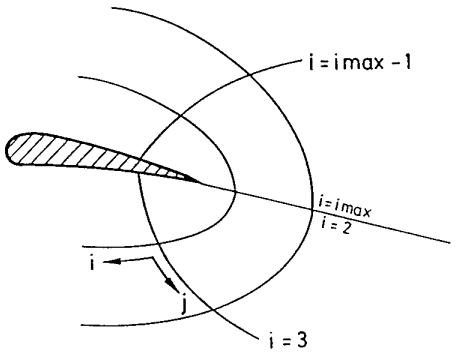


Fig. 49 Computational cut

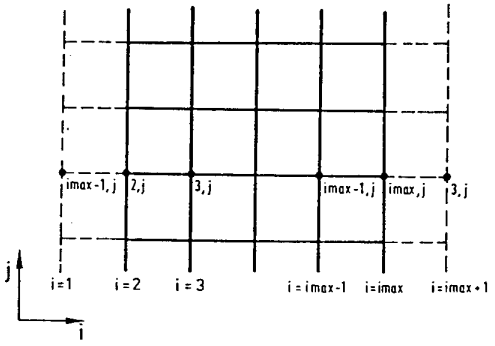


Fig. 50 Data exchange at a computational cut

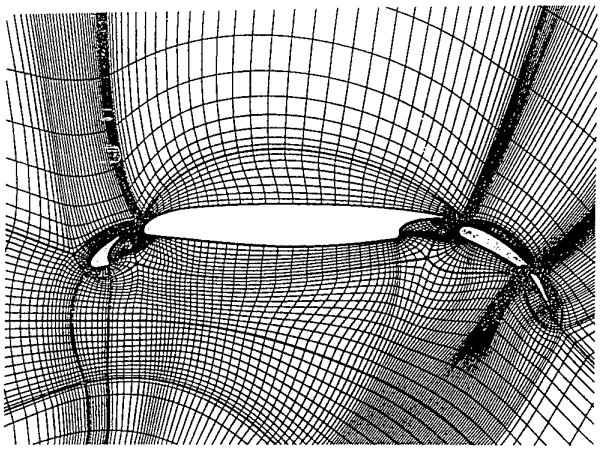


Fig. 51 Computational mesh around multi-element airfoil

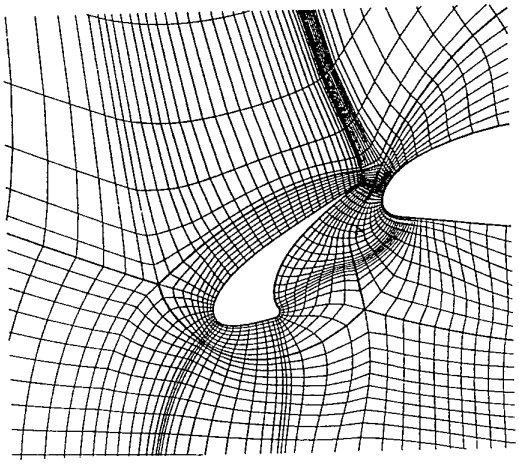


Fig. 52 Enlarged view of the slat-region

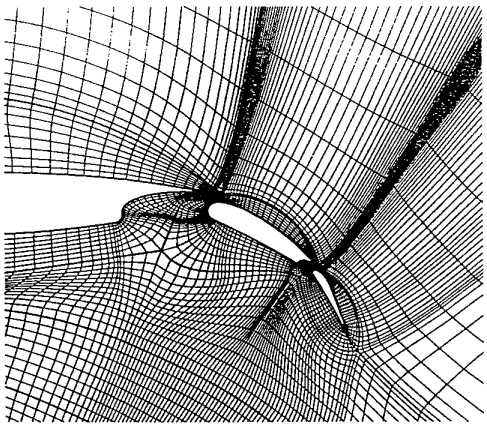


Fig. 53 Enlarged view of region at flap and tab

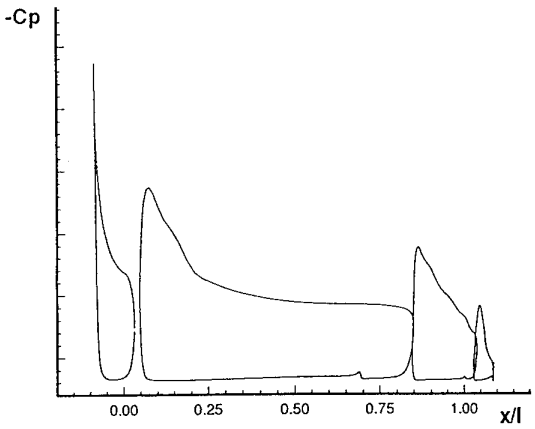


Fig. 54 Pressure distribution on multi-element airfoil at M_∞ =0.182, α=10°

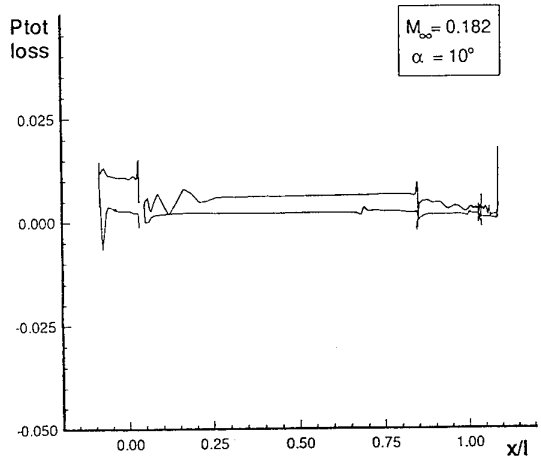


Fig. 55 Total pressure losses on multi-element airfoil at M_∞ =0.182, α=10°

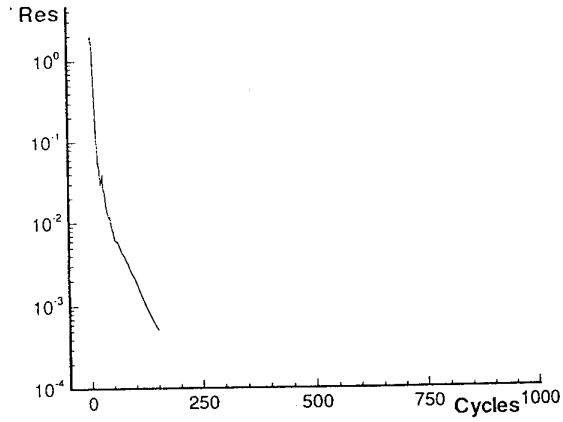


Fig. 56 Convergence history for multi-element airfoil computation at M_∞ =0.182, α=10°

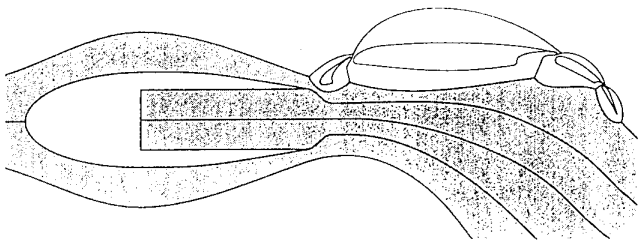


Fig. 57 Grid topology and block boundaries for incorporation of jet-generator

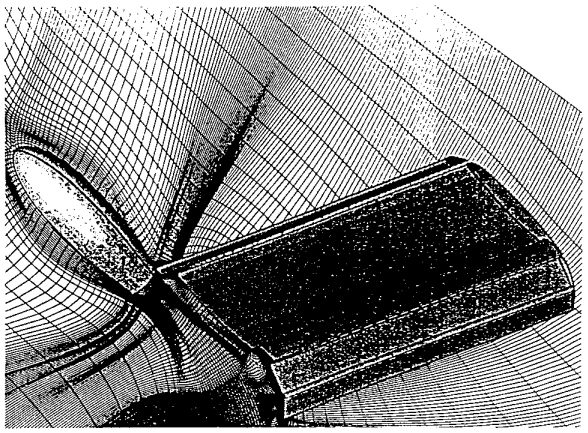


Fig. 58 View of grid symmetry-plane, jet-generator, and multi-element wing

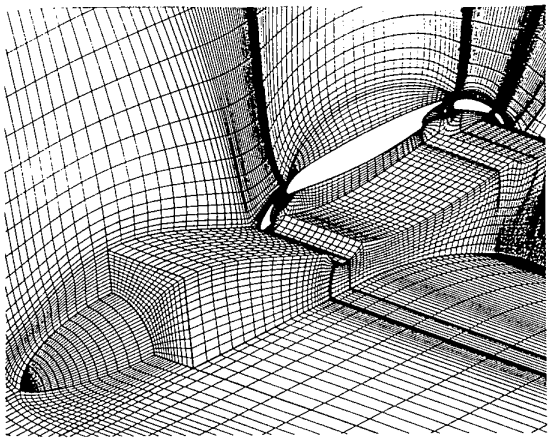


Fig. 59 Embedding of the local, polar mesh around jet-generator into the global mesh around multi-element wing

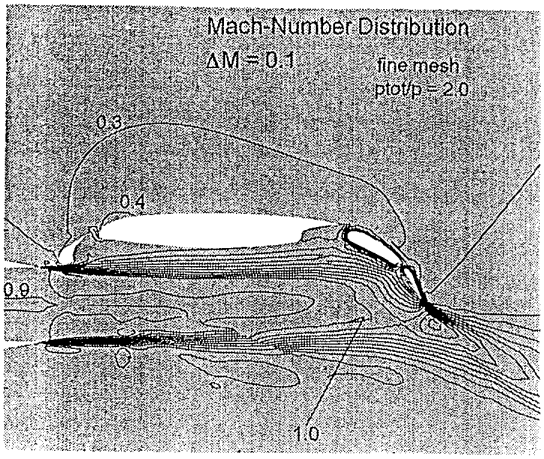


Fig. 60 Mach number distribution in the symmetry-plane at M_∞ =0.182, α=10°, P_{tjet}/ P_∞ =2.0

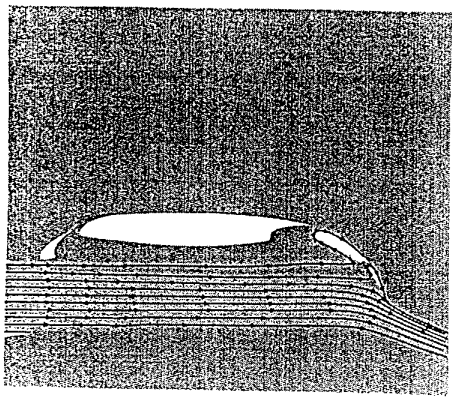


Fig. 61 Streamline pattern in the symmetry plane at M_∞ =0.182, α=10°, P_{tjet}/ P_∞ =2.0

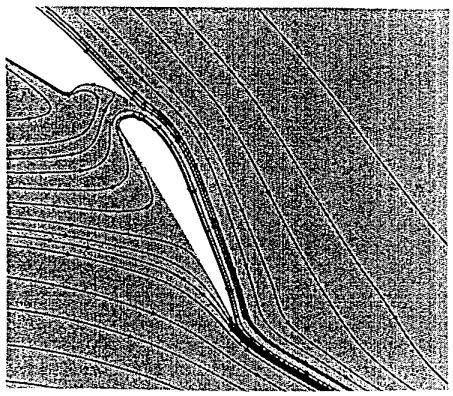


Fig. 62 Enlargement of the streamline pattern in the symmetry-plane for the region around the tab at M_∞ =0.182, α=10°, P_{tjet}/ P_∞ =2.0

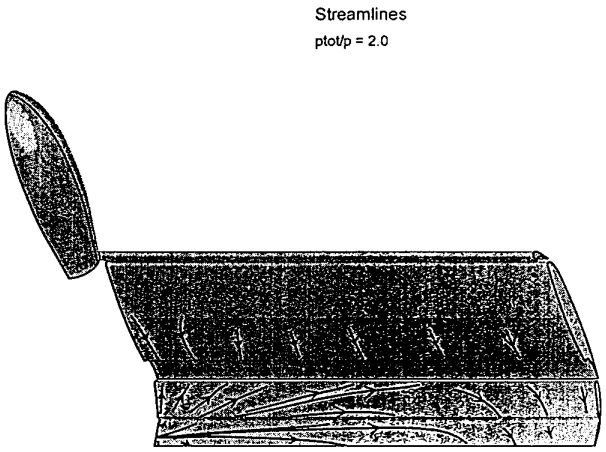


Fig. 63 Streamline pattern on the lower wing surface at M_∞ =0.182, α=10°, P_{tjet}/ P_∞ =2.0

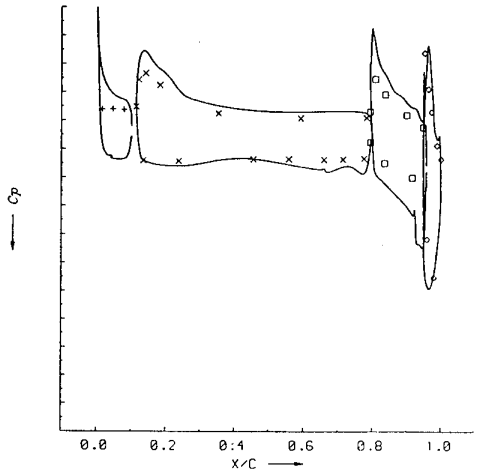


Fig. 65 Comparison of measured and calculated pressure distributions in a plane half an engine diameter apart from the symmetry-plane at M_∞ =0.147, α=10°, P_{tjet}/ P_∞ =1.252

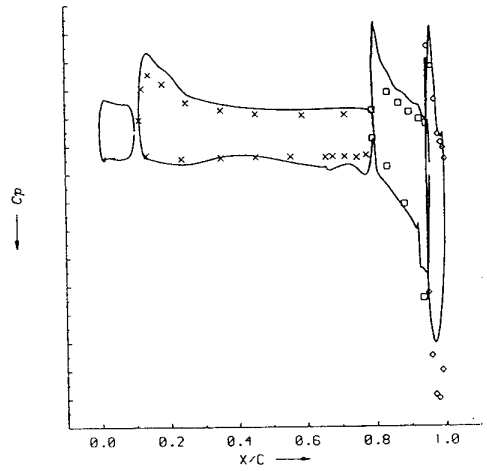


Fig. 64 Comparison of measured and calculated pressure distributions in the symmetry-plane at M_∞ =0.147, α=10°, P_{tjet}/ P_∞ =1.252

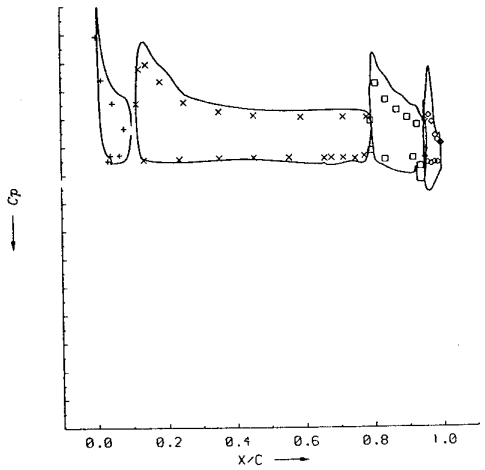


Fig. 66 Comparison of measured and calculated pressure distributions in a plane one engine diameter apart from the symmetry-plane at M_∞ =0.147, α=10°, P_{tjet}/ P_∞ =1.252

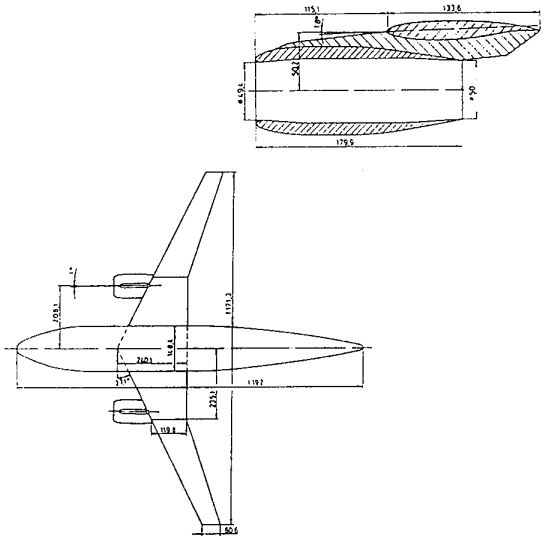


Fig. 67 DLR-F6 configuration

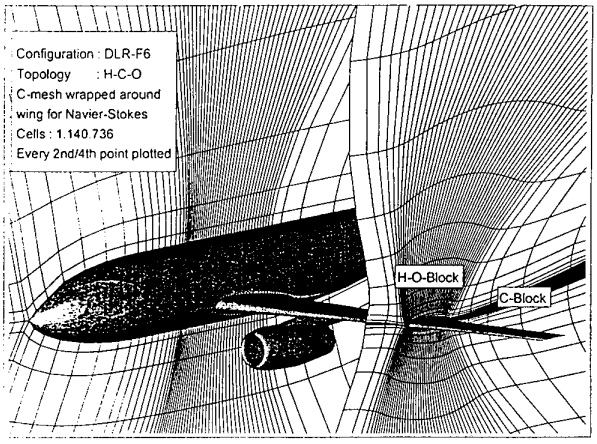


Fig. 68 Three-dimensional grid with selected grid planes

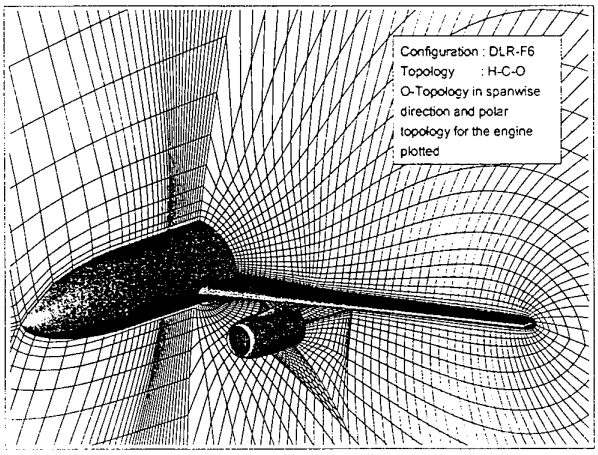


Fig. 69 H-C-O grid topology for consideration of viscous effects

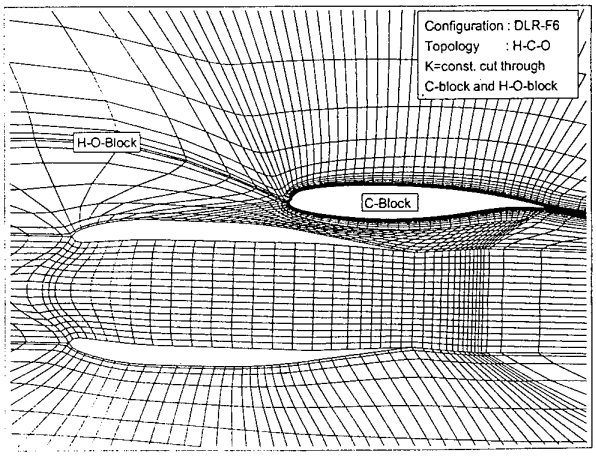


Fig. 70 Embedded C-grid at pylon location

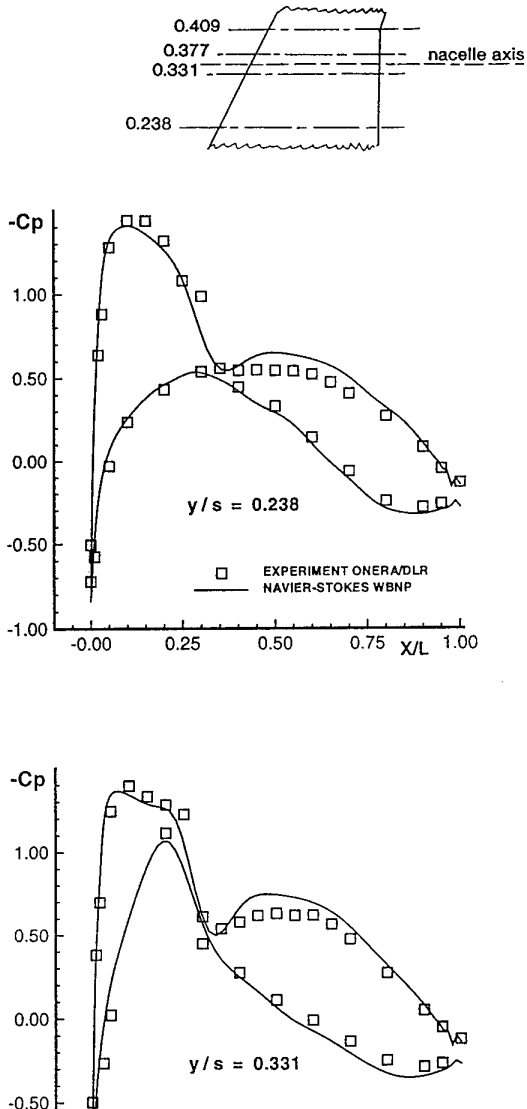


Fig. 71 Comparison of measured and calculated pressure distributions in two inboard sections at M_∞ =0.75, α=0.98°, Re=3.0x10^6

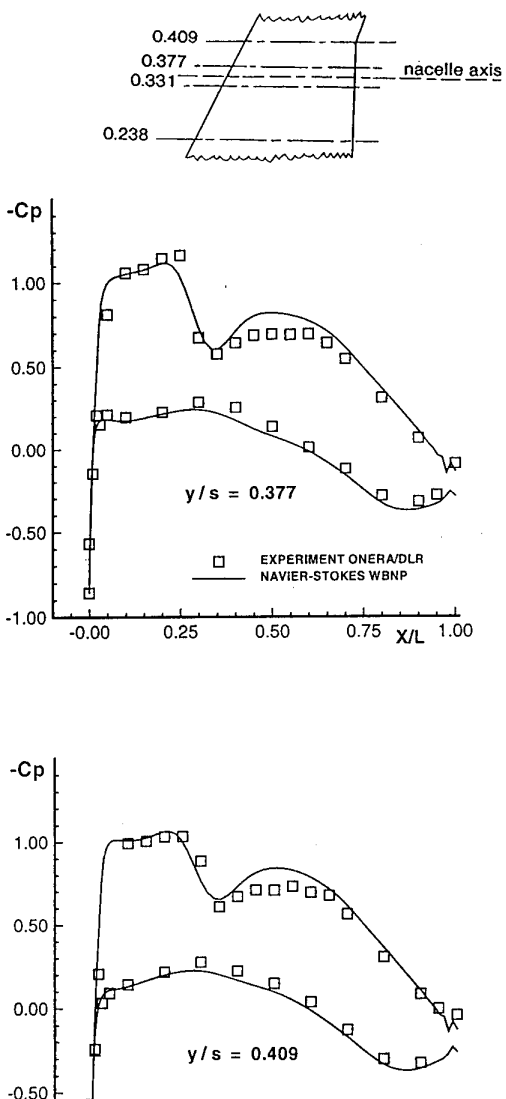


Fig. 72 Comparison of measured and calculated pressure distributions in two outboard sections at M_∞ =0.75, α=0.98°, Re=3.0x10^6

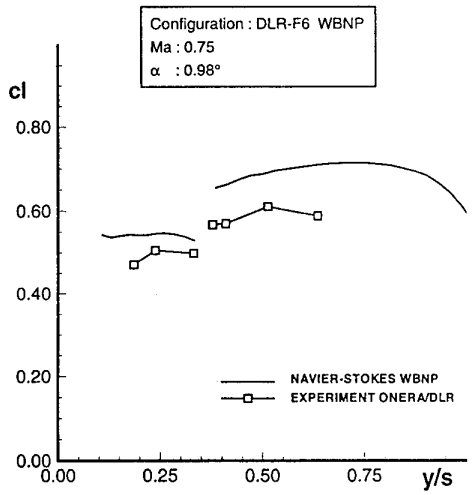


Fig. 73 Comparison of measured and calculated spanwise lift distribution at M_∞ =0.75, α=0.98°, Re=3.0x10^6

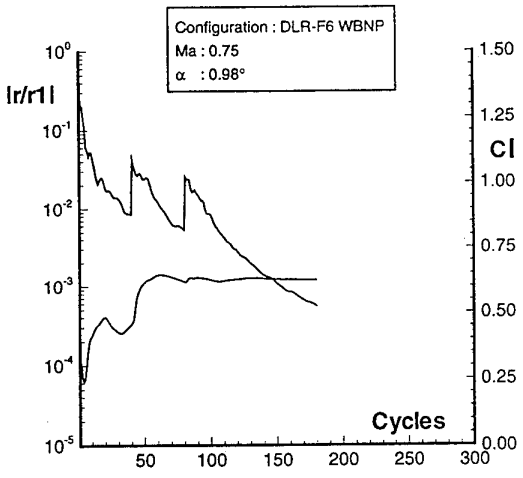


Fig. 74 Convergence history

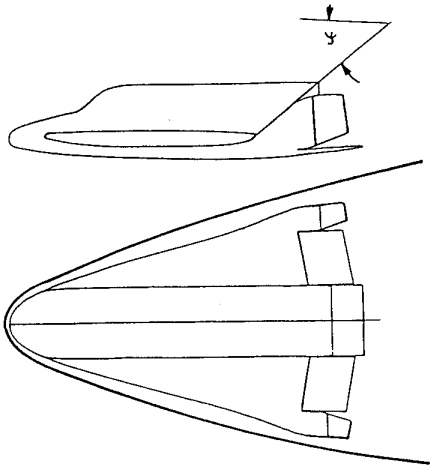


Fig. 75 Configuration HERMES (1.0) with elevon, winglet and rudder

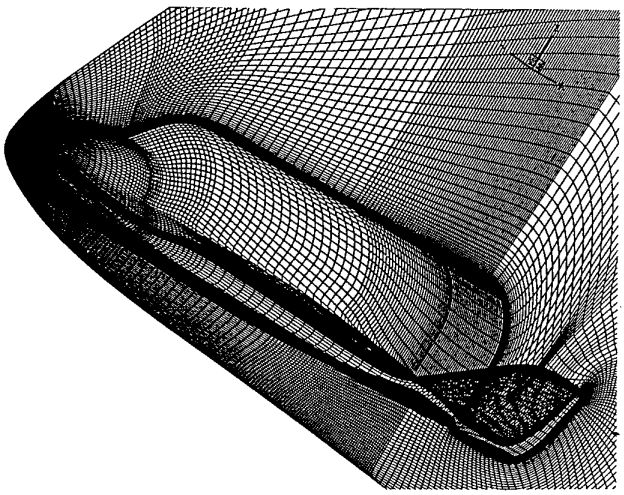


Fig. 76 Discretized HERMES reentry vehicle with 144x96-64 cells, surface grid, plane of symmetry and outflow grid plane

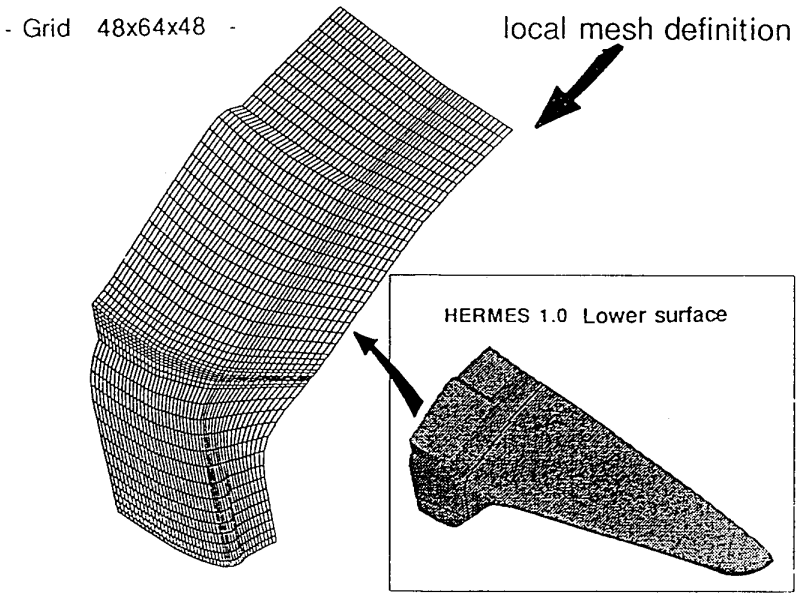
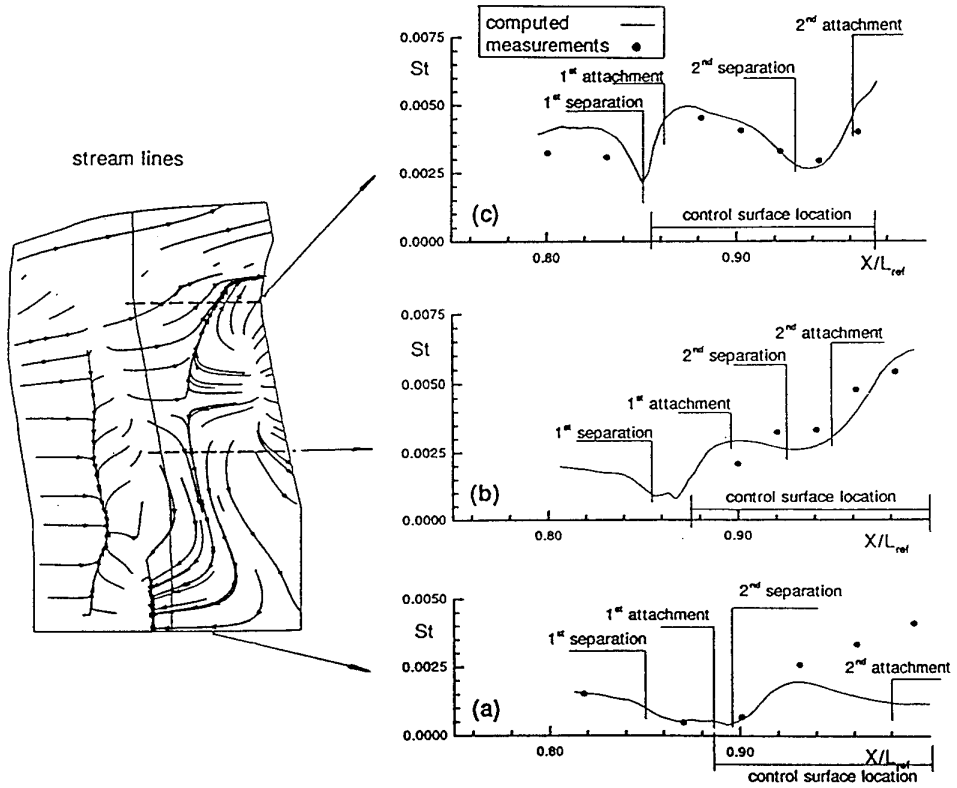


Fig. 77 Definition of local grid around windward side of deflected control surfaces



$M_\infty = 10, \alpha = 30^\circ, T_w = 293\ K, Re = 2.1\ 10^6$, calorically perfect gas

Fig. 78 Wall streamlines and heat flux distributions on deflected body flap and elevon of HERMES (1.0) configuration

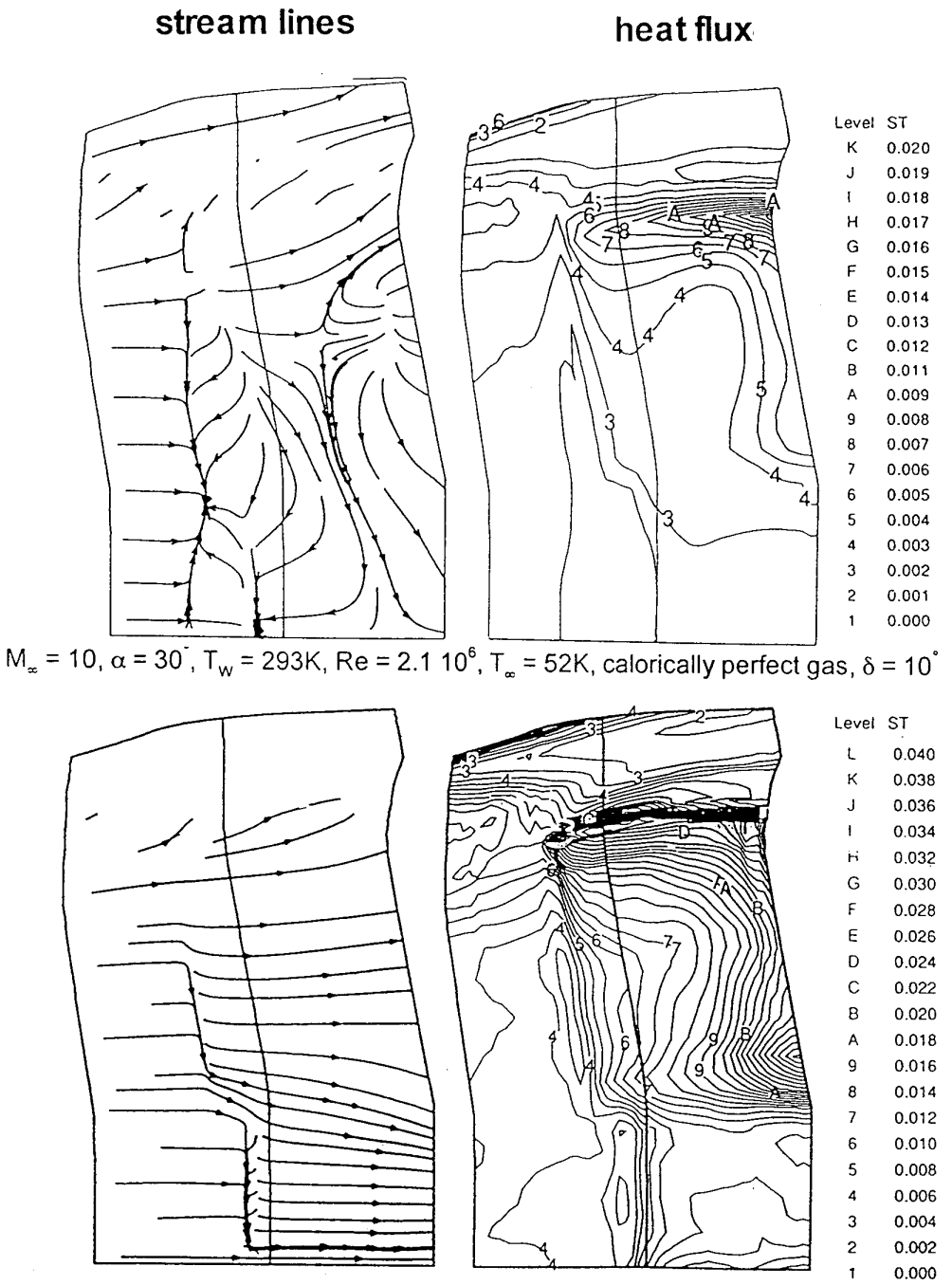


Fig. 79 Effect of inflow conditions on wall streamlines and heat fluxes for deflected controls of HERMES (1.0) configuration

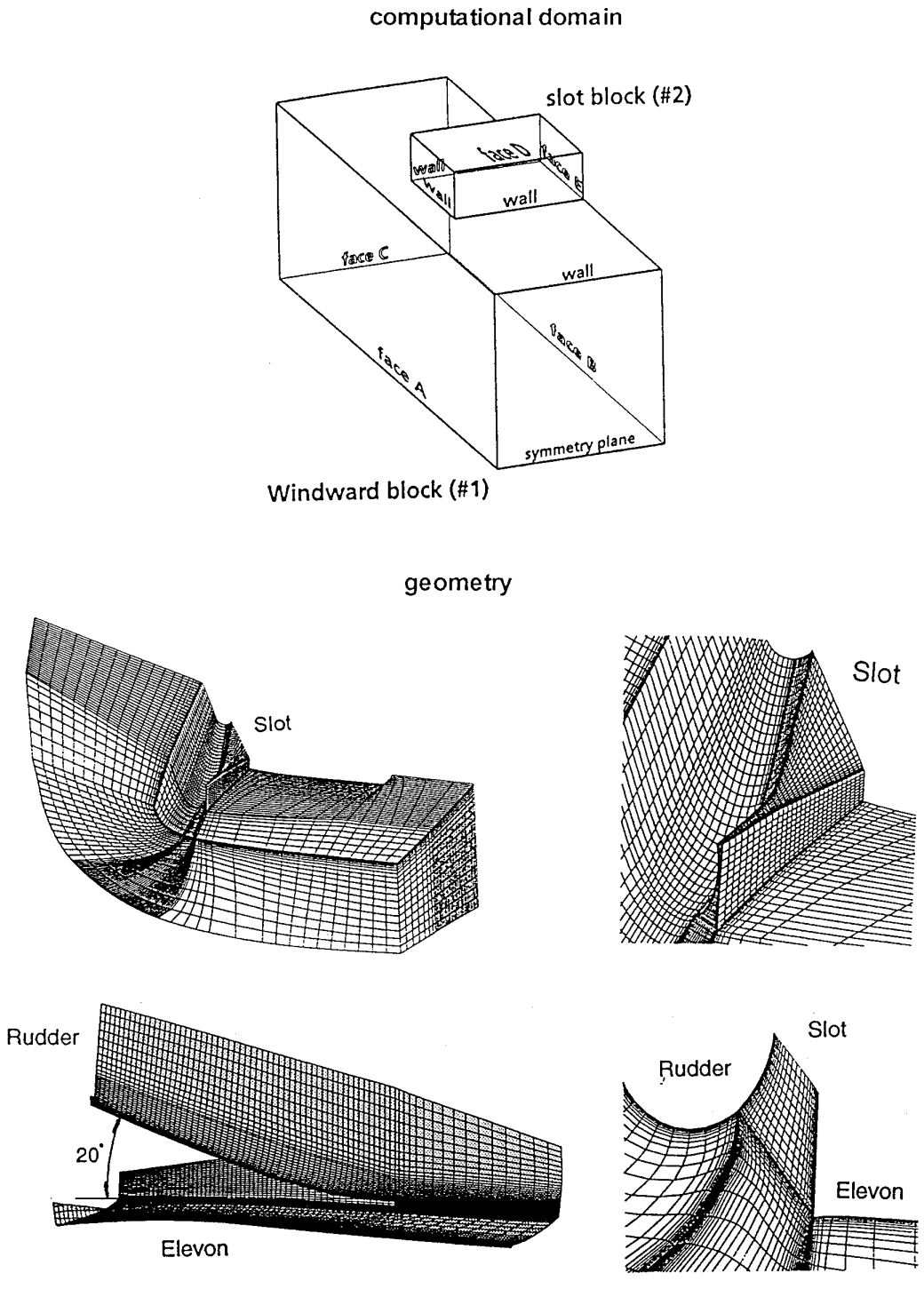
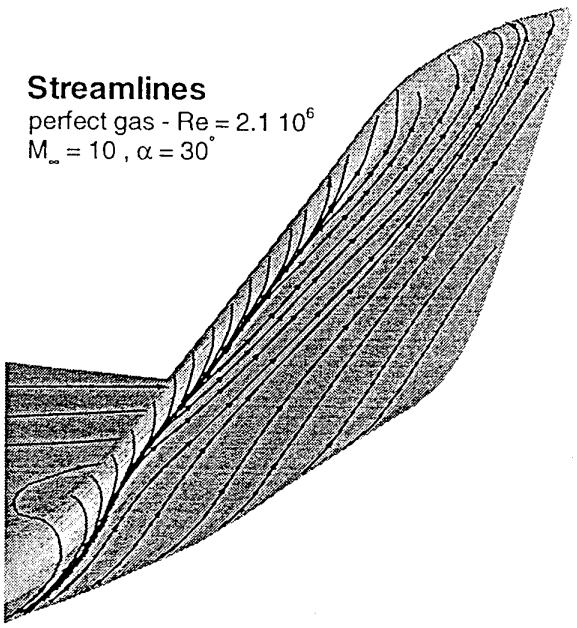
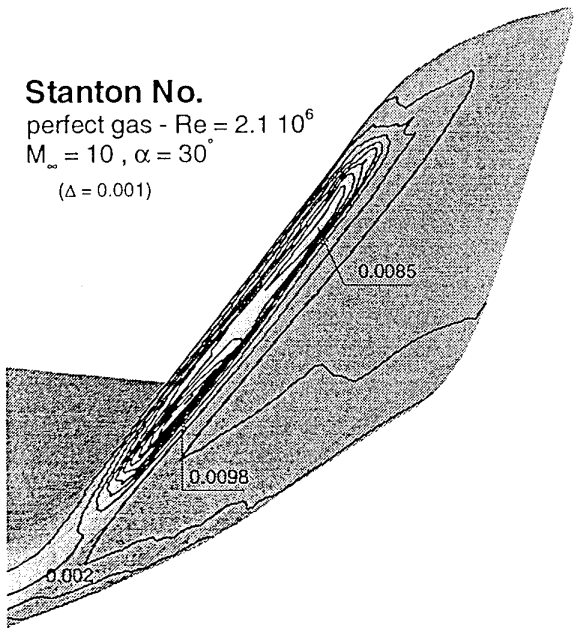


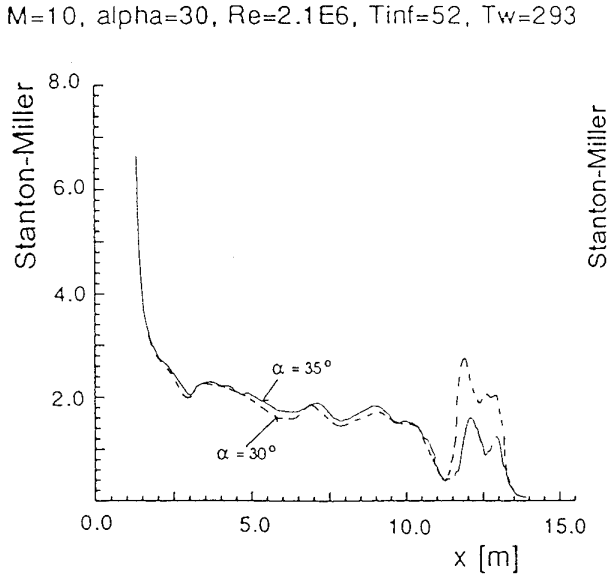
Fig. 80 Discretization of domain around slot inbetween elevon and rudder of HERMES



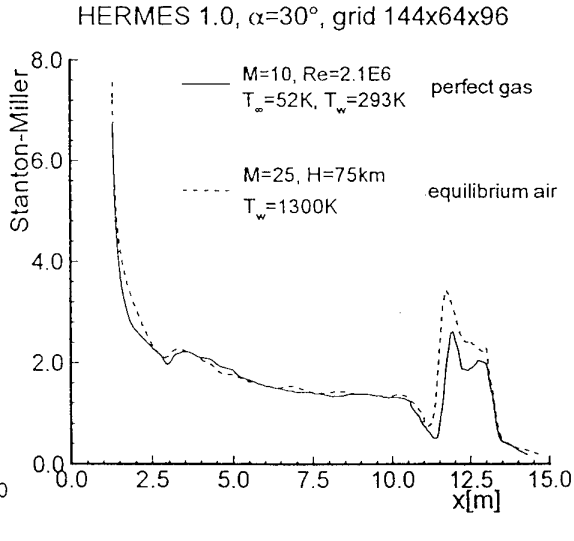
Streamlines at the winglet leading edge



Stanton contours for Ma=10, α=30° on the winglet



Influence of angle of attack on heat transfer along leading edges of wing and winglet



Heat flux along the leading edge at free flight and windtunnel conditions

Fig. 81 Flow solution around the winglet of HERMES and distribution of peak heating rates along the leading edges of wing and winglet

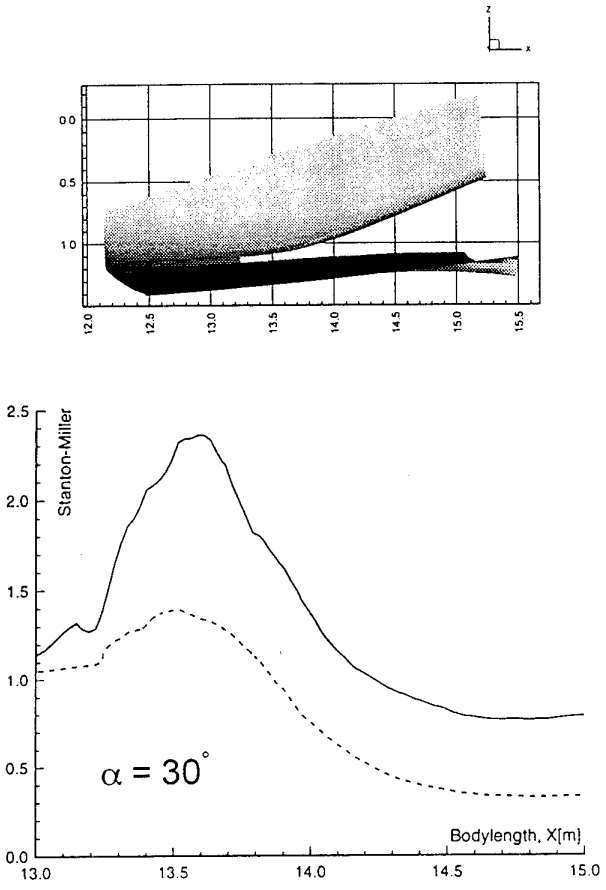


Fig. 82 Peak heating along lower edge of rudder of HERMES (1.0) configuration

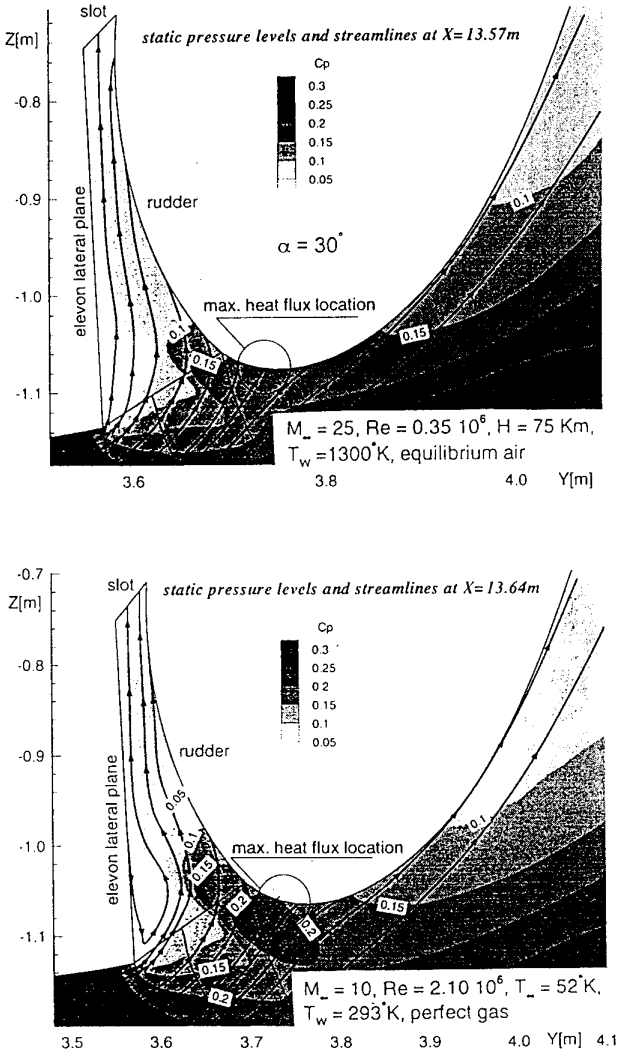


Fig. 83 Two-dimensional streamlines and pressure contours in two intersections x = const through the slot

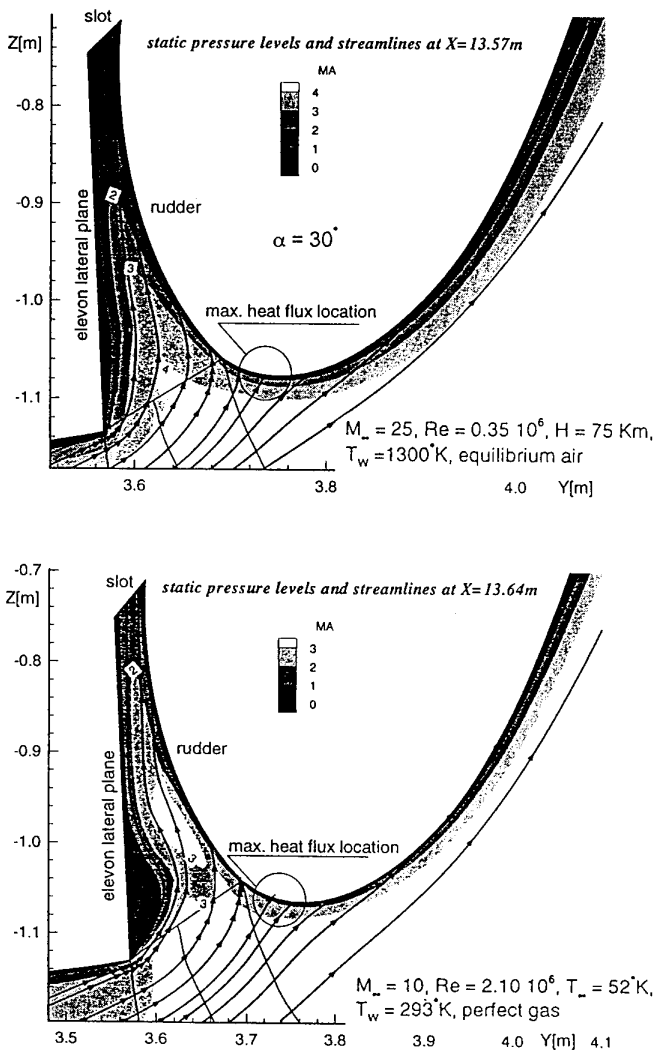


Fig. 84 Two-dimensional streamlines and Mach number contours

Structured Grid Solvers II
Parallelization of Block Structured Flow Solvers

B. Eisfeld, H.-M. Bleecke, N. Kroll, H. Ritzdorf*

Institute for Design Aerodynamics
DLR, Braunschweig, Germany

*Institute SCAI
GMD, St. Augustin, Germany

SUMMARY

This paper reviews some general considerations on the parallelization of large block structured flow solvers for production use. Parallelization is therefore not treated as an isolated subject of research, but as a tool to increase the computational power for the user and as integral part of the developmental environment of a CFD code. As an example the parallelization of the FLOWer code using the portable communications library CLIC-3D is given. Results of benchmark tests obtained on various computer hardware architectures demonstrate today's possibilities of parallel processing in CFD applications.

LIST OF SYMBOLS

a	start-up time for communication
b	bandwidth for communication
C_p	specific heat at constant pressure
$\vec{D}$	vector of artificial dissipative fluxes
E	total energy
E	speed-up
$\bar{\bar{F}}$	flux tensor
f	ratio of operations which cannot perform concurrently
H	total enthalpy
k	heat transfer coefficient
N_B	number of blocks
N_p	number of processors
$\vec{n}$	outward pointing unit normal vector
n	message length
P_{rel}	relative performance
Pr	Prandtl number
p	pressure
$\vec{R}$	residual vector
S	speed-up
T	temperature
t	wall clock time
$\vec{u}$	velocity vector
u	velocity in x-direction
V	volume
v	velocity in y-direction
$\vec{W}$	vector of conservative variables
w	velocity in z-direction
γ	ratio of specific heats
μ	viscosity
ρ	density
σ	normal stress components
τ	shear stress components
ϕ	components of the energy dissipation function

Indices

alg	algorithmic
comm	communication
i	inviscid
ijk	discrete point
l	laminar
R	reference
t	turbulent
v	viscous
x	in x-direction
y	in y-direction
z	in z-direction
∞	at infinity

Paper presented in an AGARD-FDP-VKI Special Course on "Parallel Computing in CFD", held at the VKI, Rhode-Saint-Genèse, Belgium, from 15-19 May 1995 and 16-20 October 1995 at NASA Ames, United States and published in R-807.

1. INTRODUCTION

Reviewing the topics of computer applications of the last few years an increasing interest in parallel processing is observed , in CFD as well as in other engineering disciplines, since experts predict the TFLOP/s computer until the end of the century being a parallel architecture [1, 2]. Therefore, parallel computing has become subject of basic research, carried out by mathematicians, computer scientists, engineers and other scientists dealing with a large variety of aspects. In literature one can find benchmark results for studying different hardware architectures [3], discussions on fast communication protocols [4] or considerations on computer languages supporting parallel processing, e. g. [5, 6, 7]. Others demonstrate that they have parallelized their special application program and that it is working reasonably well on different platforms, e. g. [8, 9, 10].

The paper presented here will touch all these areas, but not in detail, because it shall be devoted to the major goal of all parallelization effort made in CFD: The increase of compute power, in order to either reduce the response time for a given problem or to extend the problem size to be solved.

It should be kept in mind that an engineer applying a large CFD-code in general is not interested in details of the computer his program is running on, but in details of the solution he can obtain, i. e. in the aerodynamics of the problem he is investigating on. Therefore, in this paper parallelization is considered as a tool improving the capabilities of numerical research in aerodynamics, not as a field of research for its own sake.

From this point of view the question must be asked, whether parallelization is always useful and when should it be applied? The answer is, that the usefulness of parallelization depends on the program to be dealt with. The improvement in run time to be obtained by any acceleration technique can never exceed the run time currently needed to solve a typical problem, and an automatic parallelization is only possible on those few machines where auto-parallelizing compilers are available. Therefore, a certain amount of parallelization effort has to be considered, if one does not want to restrict oneself to a special hardware environment, such that the gain is highest when parallelizing programs for large applications.

Secondly it is questionable to parallelize algorithms which guarantee a high parallel efficiency but converge slowly. Such programs clearly show an excellent acceleration by exploiting many CPUs, but probably reveal longer response times than sequentially running algorithms which converge much faster.

Therefore, only for large CFD-codes that employ the most efficient numerical techniques, the improvement due to parallelization will be the greatest, and this paper will deal especially with this class of programs. Furthermore it is restricted to block structured codes, i. e. to solvers which work on structured grids which are split into smaller, interconnected subdomains which can be treated separately of each other. As described previously [11], this is a standard technique, in order to allow computations of flow fields around complex geometries for which no structured grid can be generated as one logically rectangular block for mathematical reasons.

Such software usually is the historic product of many scientists throuthout a long period and is applied by a number of different users, so that parallelization cannot be treated as an isolated problem, but has to meet general requirements.

After identifying some of them in the next section discussing their influence, it is dealt with strategies for the parallelization of CFD-codes which depend as well as on hard- and software aspects of the computer as on the type of program. As an example for the parallelization of a large structured flow solver, the parallelization of the FLOWer code is described in the following section. This program has evolved from the previously described DLR standard flow solver CEVCATS [11] and is developed in cooperation with the German national research center for computer science GMD and the German aeronautical industry as a multi purpose flow solver. Benchmark results obtained on a variety of different parallel computers are demonstrating the success of the approach chosen and the potential of parallel processing in realistic applications.

2. REQUIREMENTS FOR THE PARALLELIZATION OF LARGE CFD-CODES

2.1 Portability

As already mentioned in the introduction, large CFD-codes are applied by a variety of users, since otherwise the costs for their development could not be accepted. Of course it cannot be guaranteed that all these users are working on the same platform, neither parallel nor sequential. Moreover the life time of such programs exceeds that of today's computers by far, so that portability is an essential demand for any application program in industrial use.

For sequentially running codes this problem can be circumvented by restricting the implementation to standardized languages for which compilers exist on any machine, e. g. ANSI-C or Fortran 77 (Fortran 90 is still problematic, since compilers do not exist for as many computers as for Fortran 77). Furthermore it is possible to exclude dangerous programming techniques which are allowed by the language standard, but which might not work correctly on every target platform, by rigid application of programming standards.

For parallel programs things are much more difficult. Of

course, all techniques for guaranteeing portability in sequential mode still apply, but this is not sufficient, since the communication between different processes has to be portable, too. Up to now, each manufacturer of parallel computers employs his own proprietary communication system being generally incompatible with that of others. The MPI-standard for message passing systems [12] has been established about one year ago, but still implementations are hardly available, so that it is not yet guaranteeing portability.

In the contrary the PVM communication system [13] is widely spread, but since it is public domain software it might be dangerous to base large application programs on it. In case of severe problems nobody would be responsible for trouble shooting, and applications are urgent most often.

A third possibility to obtain portability as far as message passing systems are concerned is the PARMACS library [14] which is a commercial product that has been implemented on a large variety of parallel computers. A defined path to MPI is guaranteed, when this system has become a real standard, but the popularity of PARMACS is clearly restricted to European users.

Even if a decision has been made for one system or another, still the problem remains that parallel computers might not be available to any user, i. e. one should seek for the possibility to run the same program on sequential as well as on parallel computers.

2.2 Consideration of Development Effort

The development of large CFD-codes which are able to treat large problems and complex flow situations takes a long time and necessitates the experience of many scientists in order to establish an efficient, accurate and robust solver. Furthermore the users usually have been working with those programs for a long time, too, so that they are familiar with its behavior and experienced in the interpretation of its numerical results.

Therefore, parallelization must not result in the complete re-implementation of the flow solver, but is restricted to modifications of the given code, as far as large application programs are considered.

2.3 Parallelization Effort

As already pointed out, parallelization is only a means of high performance computing, i. e. as any other acceleration technique its efficiency decides about its worthiness for the user. Unfortunately any larger gain in efficiency is only possible by increasing the developmental effort, in order to gain it. The latter is clearly restricted for economical reasons, since the parallelization costs must not exceed the reduction of computational costs for an institution or an industrial business as a whole.

Therefore a parallelization strategy has to be applied guaranteeing sufficient acceleration with as little effort as possible.

3. PARALLELIZATION STRATEGIES

3.1 Parallel Architectures and Parallelism in Structured Grid Solvers

Since expectations head towards some TFLOP/s peak performance by parallel processing, a variety of different architectures has been developed attempting to step further into this direction, but it is not yet clear which design is going to succeed. Generally one distinguishes two classes of parallel computers: shared memory machines where all CPUs are coupled by a common memory (Cray C90) and distributed memory machines where each processor has its own memory unit. In this case the nodes are coupled by an interconnecting network either between the CPUs (IBM SP2) or between the memory units (KSR 1). Latest developments attempt to combine both types by clustering together several processors around one shared memory and connecting these clusters via network (NEC SX-4).

Looking on the design of large structured grid solvers, they reveal different levels of inherent parallelism to be exploited. First of all on statement level, operations could be performed concurrently, e. g. one addition and one multiplication at a time on super scalar processors. Secondly the grid structure implies a parallelism of data, such that operations on different grid points could be carried out independently which is already known from vector processors. Last but not least large structured grid solvers are multi block codes for grid generation reasons. These blocks characterize the coarse grain parallelism of programs considered here, since the different blocks could be treated concurrently.

Comparing machine architecture and code structure with each other one finds out, that different platforms fit to a different level of inherent parallelism. Fine grain parallelism on statement level is already exploited by single processor machines, data parallelism seems to be best suited for shared memory computers, whereas coarse grain parallelism based on the block structure corresponds best with a distributed memory architecture. Therefore, computers combining all three features might be best suited for structured grid solvers, but until they are available one has to investigate the possibilities of exploiting data parallelism and multi block parallelism separately. This leads to the question of how to perform communication between processors.

3.2 Communication Models

According to the different machine architectures there exist different types of communication models which support ei-

ther data or multi block parallelism. Nevertheless these models are not restricted to the corresponding computer architecture and moreover their implementations are generally incompatible with each other.

3.2.1 Parallelizing Languages

There exist attempts to describe data parallelism already by the programming language such as high Performance Fortran or Vienna Fortran. However, these systems have not yet reached a widely accepted standardization level, such that portability is hardly guaranteed for the moment. This could be overcome by current developments incorporating parallel communication within objects of existing object oriented programming languages like C++ [5, 6, 7], but one major drawback remains: Any large solver not yet implemented in such a language would have to be completely rewritten which will clearly not be acceptable for the reasons mentioned in the last section.

3.2.2 Compiler Directives and Autotasking

Another data parallel approach which makes parallelization more feasible for the programmer is to use directives telling the compiler which sections of the code can be treated concurrently, e.g. where loops incorporate data parallel structures. This method has got the great advantage that an existing code basically remains unchanged and that there exist analyzing tools at least on some machines, making suggestions about where to place such directives.

The problem is, that this procedure has to be repeated on each platform again, since compiler directives are naturally machine dependent. Furthermore, experiments employing autoparallelizing compilers have revealed that best efficiencies were always achieved by putting in these directives manually increasing the parallelization effort [15].

The autotasking approach only assumes that only data incorporate parallelism, i. e. only array data can be treated independently of each other, so that good efficiencies can only be expected from highly vectorizable programs. This assumption will generally hold for structured grid solvers, but depends on the block size which might be low for grid generation reasons and which becomes definitely low on coarse grids of multigrid algorithms. The advantage of this method is, that it is definitely portable, since parallelization is carried out automatically.

On virtual shared memory machines, i.e. distributed memory computers which are programmed as if they had a global shared memory, efficiency decreases, because data have to be transferred by global communication.

Last but not least compiler directives are spread all over the code such that any algorithmic development cannot be separated from the parallel machine where the code is running on.

3.2.3 Message Passing

The typical communication model corresponding to coarse grain parallelism is message passing where the programmer is responsible himself for all types of communication between the different processes. This means the programmer explicitly must tell the program when and where to send or receive data respectively which of course is increasing the parallelization effort. The advantage of this type of communication model is its efficiency, since data transfer takes place only, when needed. Moreover all operations can be performed in parallel, independent of vectorization features.

Of course portability is still a problem, because of the vendors implementing proprietary systems, but as pointed out in the last section, there already exist widely spread systems and the MPI-standard allowing an acceptable degree of portability today.

On the contrary to data parallel communication models, the message passing technique can be treated independently from all algorithmical considerations as far as single blocks are concerned. Each block is treated the same way in the parallel mode as in the sequential mode, and all communication takes place outside the block algorithm.

3.3 Guidelines for the Parallelization of Block Structured Flow Solvers

In the following four rules will be given and explained which have proven to lead to an efficient parallelization while meeting the objectives on large block structured flow solvers for industrial use. Of course they should not be understood as the eternal laws of parallelization, but they have successfully be applied for parallelizing at least two solvers of this category, i.e. the FLOWer code and the NSFLEX-code [16].

3.3.1 Grid Partitioning as Parallelization Strategy

This method is based on the idea of splitting a given grid into smaller subdomains which can be treated independently of each other. The arising intersections between the different blocks are treated as boundaries with a special cut condition. In general there exists an overlap region at those cuts where data are copied to from the corresponding neighboring block. As an example figure 1 shows schematically the partitioning of a two-dimensional domain around an airfoil into four subdomains.

This technique is chosen, since it is an approach of relative simplicity. Furthermore, this strategy is agreed to be the most efficient one [17, 18], when solving partial differential equations as it is done by flow solvers. From a more practical point of view, this method has got the great advantage of being already well established in sequential structured grid solvers, since the multi block technique is nothing else but grid partitioning.

The main difference between a parallel and a sequential code then is, that the exchange of boundary data between neighboring blocks has to be replaced by sending and receiving procedures. Another slight difference concerns global operations involving all blocks, e.g. the computation of the overall residual which has to be realized by global communication techniques. Therefore, applying grid partitioning as basic strategy is an approach that leads straight forward to parallelization while keeping a sequentially proven algorithm widely unchanged.

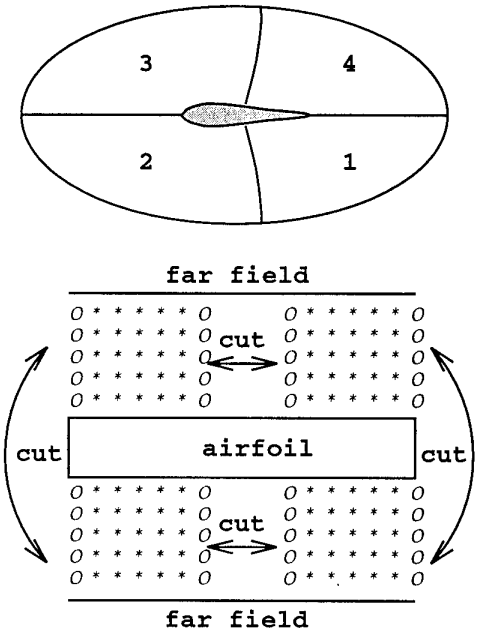


Fig. 1 Schematic multi block decomposition of the flow field around a generic transport aircraft.

3.3.2 Separation of Computation and Communication

Only keeping to this rule strictly will allow the development of algorithms independently from hardware aspects.

This feature is necessary with respect to the conditions under which large block structured codes are usually developed. There are several scientists, engineers or programmers working on the same software, and one cannot assume that all of them are sharing the same parallel super computer for development purpose, i.e. for testing, and debugging instead of high performance computing. Separating all communication operations from the algorithmical parts therefore allows the integration of developments carried out on simple workstations without problems.

Furthermore from software engineering reasons it must be aimed at a high degree of modularity of the program design which enables a coordinated development by a group of researchers. Any intermixing of communication and computation would therefore contradict to this basic principle of software realization.

Last but not least the portability problem becomes much more feasible to handle, when all communication procedures are concentrated within separate units of the program. Even if communication systems are not compatible with each other, the effort for porting a program to another parallel platform is reduced, since only defined modules have to be modified or exchanged respectively.

3.3.3 Communication by Message Passing

The decision for the message passing programming model evolves quite naturally from the things said above. As has been shown, this type of communication corresponds to coarse grain parallelism, and that is exactly what is represented by the grid partitioning strategy or multi block technique.

Additionally, one gets the highest efficiency, since parallelism is not restricted to the vectorizable parts of the code. One should never forget that it is high performance computing what is aimed at by parallelization. Another advantage is what programmers might fear for the increase of implementation effort: communication has to be realized by explicit calls of system routines for sending and receiving data and so on. Therefore, the message passing routines already form some type of library which exists independently of any application program, such that separating communication from computation becomes a simple task. One only has to concentrate all these routine calls within distinct modules of the program.

Finally, the application of message passing does not exclude the possibilities of data parallel communication models as far as compiler directives are concerned. Since message passing is only touching the block structure of a flow solver, there still remains the inherent data parallelism within each block. Therefore a combination of techniques involving message passing for the inter block communication and data parallel directives within each block might be thought of, especially with respect to future multi level architectures. Nevertheless drawbacks and advantages of such an approach would have to be assessed after practical experiences have been made.

3.3.4 Use of a Communication Library

Returning back to pure message passing and what has been said about its features, it is only one step further to demand for a library realizing all necessary communication in a parallel code. Remembering the last subsections this would only be a more detailed guideline concluding what has been already said, but it is more than that.

What is thought about, is a high level library incorporating the whole functionality involving communication in block structured programs, e.g. an exchange of boundary data at block interfaces. Since all these functionalities must have been realized already in sequential mode, ideally within separate modules, portability between sequential and parallel computers is no problem any more. One only has to either link different libraries or call different subroutines depending on the architecture.

Additionally, such a library can be developed in almost complete independence of the calling CFD-solver, such that specialists on parallel computing could be employed for its implementation guaranteeing a high degree of reliability. The application programmer on the other hand is relieved from any basic considerations on parallelism. He only must be familiar with the interfaces to the library routines, the functionality of which he already knows from his sequential experience.

Therefore, although the effort of realizing such a library is high, the parallelization costs for the application program are low, and, since a library can be re-used again and again by different codes, its implementation is worthwhile. This approach is not a vision for the future fairly to be reached, but has already become reality, and will be described within the next section.

4. THE COMMUNICATIONS LIBRARY CLIC-3D

At GMD this approach has been followed with the creation of the GMD communications library CLIC („Communications Library for Industrial Codes", former versions are known as the GMD Comlib). The target applications are PDE solvers on regular and block-structured grids, as they result from finite difference or finite volume discretizations. In particular, the library supports parallel multigrid applications. For this class of applications it turned out that, while the numerics differ widely, the communication sections are quite similar in many programs, depending only on the underlying problem geometry. As a consequence of the high level abstraction, the CLIC library is useful only for the application class for which it was designed.

The development of the CLIC library started at GMD in 1986 with the definition and implementation of routines for 2- and 3-dimensional logically rectangular grids. It followed the implementation of routines for 2-dimensional block-structured grids. The routines for 3-dimensional block-structured grids are currently developed in the project POPINDA. The routines support vertex-oriented as well as cell-centered discretization schemes.

POPINDA is a German national project, funded by the German Federal Ministry for Education, Science, Research and Technology (BMBF). Its central goal is to provide the utilization of highly parallel systems for aerodynamic production codes. The parallel codes being developed in the project are based on highly efficient numerical algorithms (multigrid). They will allow more accurate simulations, which are indispensable due to increased economic, ecological and technical requirements.

The aim in the development of CLIC is to make programming for complex geometries as easy as for a single cube and to provide high level library routines for all communication tasks. The CLIC user interface provides the application program with all required information about the problem geometry.

The CLIC library is based on the PARMACS message passing system [14] and, thus, is designed for a host-node (master-slave) model. A host process starts the distributed application, performs the input and output and data transfers with the node processes. The host process does not participate in the grid computations; this is performed by the node processes. As a consequence the user application is separated in a host program and a node program, as illustrated by figure 2.

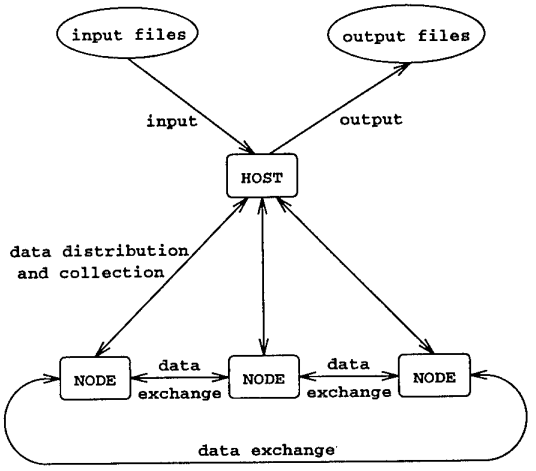


Fig. 2 Host-node-structure of the parallel FLOWer code.

In the host program of a 3-dimensional block-structured application, the same input parameters are read as in the sequential user program. Then, CLIC-routines read in the description of the block-structured grid, create the node processes, distribute the blocks in a load-balanced way to the allocated node processors and distribute the input parameters to the node processes. Another routine reads the grid coordinates and sends them to the corresponding node processes. After the data is distributed to the node processes, the host program usually calls a CLIC-routine which waits for output generated by the node processes and writes that output to the corresponding output units.

Each node process executes the node program which is

very similar to the sequential user program without reading the input data. The input data is transferred by CLIC-routines, which receive data containing the essential block information of blocks, together with global information passed by the host program. The grid coordinates are also received by a library routine. It should be noted that a node process receives the information and grid coordinates only for the blocks for which the node process performs grid computations. A schematic flow chart of the host and node process cooperation is given in figure 3.

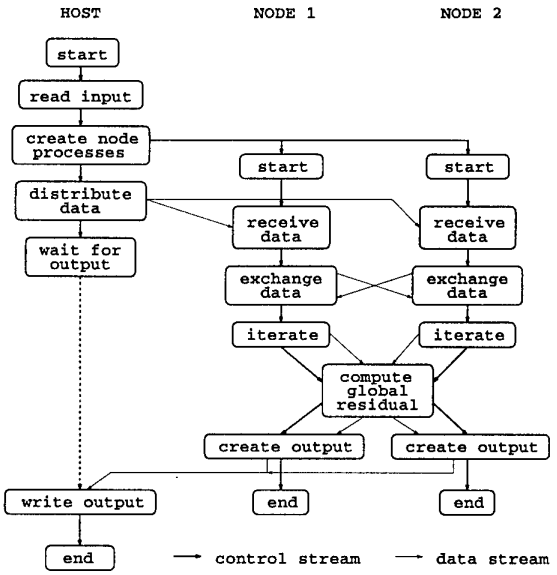


Fig. 3 Schematic flow chart of host and node process execution supported by the CLIC communications library.

Library routines also analyze the block-structure; i. e. for each segment edge and segment point, the adjoining blocks and the number of coinciding grid cells are determined and the edge or point is topologically classified. If the segment edge or point is part of the physical boundary, the physical boundary conditions of all adjoining blocks are also determined. In addition, the grid coordinates can be examined, and geometrical singularities such as block faces which collapse to a single point can be detected. All that data can be inquired and may be used in the user program, for example in the discretization of irregular grid points or physical boundary points.

This data may be important for the user program, however, it is essential for the CLIC library to correctly update the overlap regions (to exchange the boundary data) of neighboring blocks and to optimize this update procedure. An optimization of this update procedure is significant for the parallel efficiency, because the corresponding CLIC-routine is generally called most of all and is the most crucial routine especially on coarse grids of multigrid algorithms.

An example for such an optimization of the update procedure is regular corners of 8 blocks; a straightforward technique to update the overlap regions is to send and receive messages over all faces edges and corners of a block; thus, 26 messages (6 faces, 12 edges, 8 corners) have to be sent and received for each block in such a regular case. However, in such regular cases, the number of messages to update the overlap regions can be decreased to 6 by the technique as follows: in the first step, all blocks exchange their data with neighbor blocks in I-direction (1 message per block face); in the second and third step, all blocks exchange their data with neighbor blocks in J- and K-direction, respectively, but now including the already updated overlap regions. This technique is illustrated by figure 4 for a two-dimensional example.

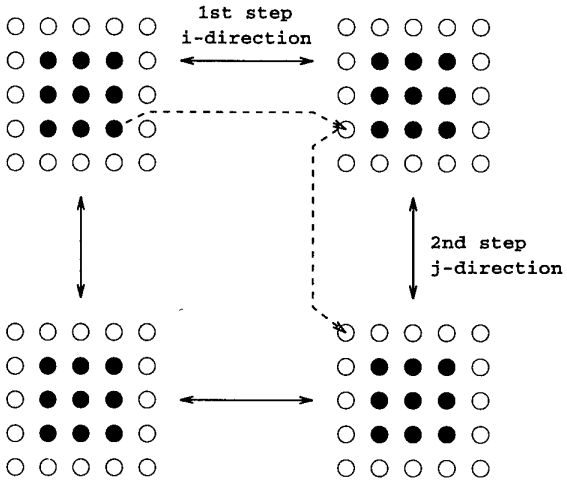


Fig. 4 CLIC exchange stragegy.

So, the data resulting from the analysis of the block-structure is used to optimize the update of the overlap regions. Since it is too expensive to optimize this update sequence and to determine the areas which have to be sent to neighbor blocks within each update, these tasks are performed only once by CLIC-routines in an initialization routine of the user program. Within the solution process of the user program, the update of the overlap regions of all blocks is then performed by the call of a single CLIC-routine. In that call, the user specifies the number of the multigrid levels and can choose the number of grid functions to be simultaneously exchanged.

Among other tasks, the CLIC library performs also the computation of global values (for example global residu-

als) and the output to files and standard output which is generated by the node processes. In the next year, the library will be extended to adaptive block-structures (i. e. hierarchies of block-structures). This will include routines which create and manage adaptively refined new grid levels, which perform a load-balanced dynamic mapping and which perform all data re-distribution required during adaptive multigrid algorithms.

An important fact for the development and management of user programs is that there is also a sequential version of the 3-dimensional block-structured CLIC library. Thus, a user program can be sequentially executed with the same interfaces as in the parallel case.

5. PARALLELIZATION OF THE FLOWer CODE

The development of the FLOWer code was initiated within the parallelization project POPINDA. The program has directly evolved from the DLR-CEVCATS code [11] and is further developed in close cooperation of the DLR and the German aerospace industry, i.e. DASA.

As the DLR-CEVCATS code, the FLOWer code is written in standard Fortran 77 for portability reasons and operates on block structured grids. Therefore, it allows computations of flows around complex aircraft geometries as illustrated by figure 5. Furthermore effort is made, in order to push all FLOWer development towards the design of a multi purpose standard code for a wide area of complex applications. Since many different departments of various institutions of research and industry are involved, the future FLOWer code must cover all of their aerodynamical problems reaching from incompressible flows to hypersonics.

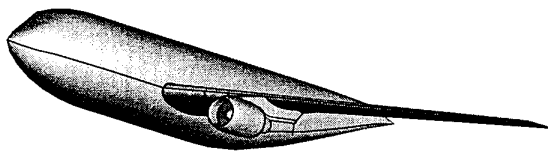


Fig. 5 Generic aircraft configuration consisting of wing, body, engine and pylon

5.1 Numerical Method

The FLOWer code is solving the Euler- or Navier-Stokes equations in conservative form written as

$$\frac{\partial}{\partial t}\int_V \vec{W}\,dV + \int_{\partial V} \bar{\bar{F}}\cdot\vec{n}\,dS = 0$$

where $\vec{W}$ denotes the vector of conservative variables

$$\vec{W} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}$$

and $\bar{\bar{F}}$ is the flux tensor which can be split into an inviscid and a viscid part:

$$\bar{\bar{F}} = \bar{\bar{F}}_i + \bar{\bar{F}}_v$$

with the inviscid flux tensor being defined by

$$\bar{\bar{F}}_i = \begin{pmatrix} \rho u & \rho v & \rho w \\ \rho u^2 + p & \rho uv & \rho uw \\ \rho uv & \rho v^2 + p & \rho vw \\ \rho uw & \rho vw & \rho w^2 + p \\ \rho uH & \rho vH & \rho wH \end{pmatrix}$$

and the viscid flux tensor by

$$\bar{\bar{F}}_v = \begin{pmatrix} 0 & 0 & 0 \\ \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \\ \phi_x & \phi_y & \phi_z \end{pmatrix}$$

The components of the energy dissipation function are:

$$\phi_x = u\sigma_x + v\tau_{xy} + w\tau_{xz} - k\frac{\partial T}{\partial x}$$

$$\phi_y = u\sigma\tau_{xy} + v\sigma_y + w\tau_{yz} - k\frac{\partial T}{\partial y}$$

$$\phi_z = u\tau_{xz} + v\tau_{yz} + w\sigma_z - k\frac{\partial T}{\partial z}$$

For the non-dimensional pressure and temperature the following relations hold

$$p = \rho(\gamma-1)\left(E - \frac{\vec{u}^2}{2}\right)$$

$$T = \frac{p}{\rho}$$

The elements of the viscous stress tensor are given by Newton's law of skin friction, i. e.

$$\sigma_x = -2\mu\frac{\partial u}{\partial x} + \frac{2}{3}\mu\vec{\nabla}\cdot\vec{u}$$

$$\sigma_y = -2\mu\frac{\partial v}{\partial y} + \frac{2}{3}\mu\vec{\nabla}\cdot\vec{u}$$

$$\sigma_z = -2\mu\frac{\partial w}{\partial x} + \frac{2}{3}\mu\vec{\nabla}\cdot\vec{u}$$

$$\tau_{xy} = -\mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)$$

$$\tau_{yz} = -\mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right)$$

$$\tau_{xz} = -\mu\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)$$

This formulation is further simplified by applying a thin shear layer approximation such that gradients in streamwise direction, i. e. along quasi streamwise grid lines, are neglected [19].

The system is closed by the relations for the transport coefficients

$$\mu = \mu_l + \mu_t$$

$$k = C_p\left(\frac{\mu_l}{Pr_l} + \frac{\mu_t}{Pr_t}\right)$$

where the laminar viscosity μ_l is given by Sutherlands's formula

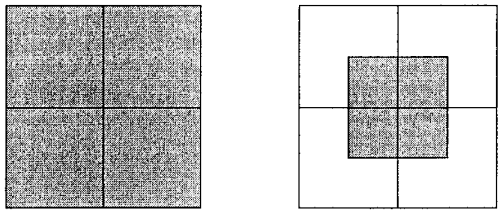
$$\frac{\mu_l}{\mu_\infty} = \left(\frac{T}{T_\infty}\right)^{3/2}\frac{T_\infty + 110K}{T + 110K}$$

and the turbulent viscosity μ_t being computed from the algebraic Baldwin-Lomax model [20].

These equations are discretized in space by the method of lines resulting in a system of ordinary differential equations involving each hexaeder of the structured grid

$$\frac{d}{dt}\vec{U}_{ijk} + \frac{1}{V_{ijk}}\int_{\partial V}\bar{\bar{F}}_{ijk}\cdot\vec{n}dS = 0.$$

The discretization is central, but it can be switched between a cell vertex and a node centered scheme (figure 6).



cell vertex node centered

Fig. 6 Discretization stars of the FLOWer code

Therefore, an artificial dissipation term due to Jameson et al. [21] is added damping high frequency oscillations and allowing a sufficiently sharp resolution of shock waves in the flow field.

The resulting system of equations then reads

$$\frac{d}{dt}\vec{U}_{ijk} + \frac{1}{V_{ijk}}\left(\vec{R}_{ijk} - \vec{D}_{ijk}\right) = 0$$

with $\vec{R}_{ijk}$ being the vector of the residuals of convective and viscous fluxes and $\vec{D}_{ijk}$ the vector of the artificial dissipative fluxes respectively.

The time integration is carried out by an explicit, hybrid Runge-Kutta scheme involving multiple stages [22]. The convergence to steady state is further accelerated by the techniques of local time stepping and an implicit smoothing of the residuals obtained within a Runge-Kutta stage. For Euler computations there exists a possibility of driving the solution to steady state faster by exploiting the demand for constant enthalpy [23].

Alternatively, a two stage implicit LU-scheme has been implemented only recently and is currently tested for the final integration.

Both iteration techniques are embedded into a powerful multigrid algorithm [24]. Depending on the user input data standard single grid computations are as well possible as a successive grid refinement, simple multigrid or full multigrid algorithms. As is illustrated in [11], high convergence rates can be obtained, using this technique.

A more detailed description of the algorithms used can be found once again by Kroll et al. [11].

5.2 Block Structure

Since grids around complex geometries cannot be generated as one logically rectangular block, the FLOWer code is block structured. That means that the domain is split into regions for each of which the generation of a structured grid is possible. Figure 7 is showing schematically such a grid around a transport aircraft. The program then treats the blocks more or less independently from each other which

can only be done properly by exchanging data of the current solution at block interfaces before each time step.

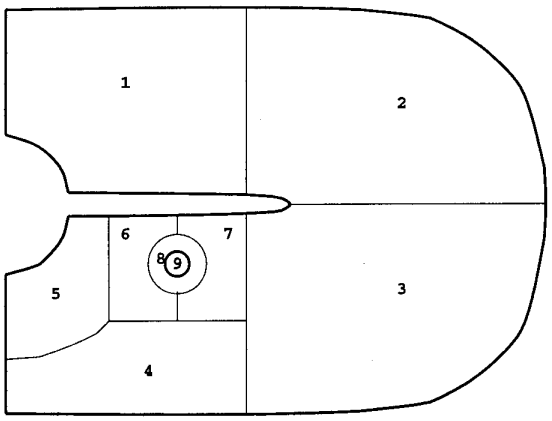


Fig. 7 Schematic multiblock decomposition of the flow field around a generic transport aircraft.

Therefore, each block is surrounded by one or two layers of dummy cells respectively which are used for the formulation of boundary conditions. At block intersections these cells correspond with those of their neighboring block and carry the solution of the points there. This technique has already been illustrated by figure 1 where a 2D example of the block structure around an airfoil is given involving one dummy layer.

The overlap width at such intersections decides about the order of accuracy that could be obtained at boundaries, such that the FLOWer code allows two dummy layers on demand by the user, in order to keep the accuracy at block intersections unchanged at second order in space.

This number of dummy layers is necessary for computing the artificial dissipation terms at cuts correctly. Since these involve central fourth differences in space, each grid point needs a support of two further vertices to either side. Therefore, grid points located on the intersection of two blocks need information on data of two layers of points from their neighbor, i. e. two dummy layers, in order to compute the artificial dissipation there exactly as if there were no cut.

That inaccuracies at block interfaces may influence the solution is demonstrated by figures 8 and 9, where the pitchingmoment of an oscillating NACA 0012 airfoil is plotted versus the angle of attack [29]. When involving only one layer of dummy cells, the multiblock solution deviates from that obtained by a single block computation. Re-establishing second order accuracy at the cuts by adding the second dummy layer, these differences vanish.

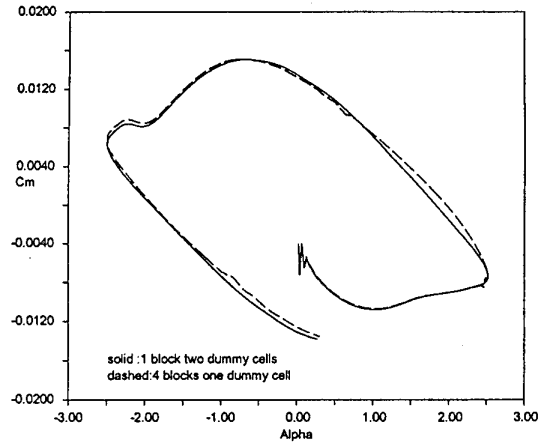


Fig. 8 Pitching moment of an oscillating airfoil versus angle of attack. Comparison of single block and multiblock solution with one dummy layer at cuts [29].

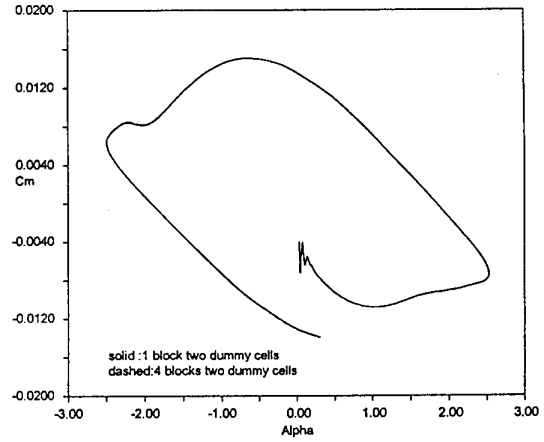


Fig. 9 Pitching moment of an oscillating airfoil versus angle of attack. Comparison of single block and multiblock solution with two dummy layers at cuts [29].

Since exchange of data between blocks creates an additional overhead with respect to single block computations, the FLOWer code inhabits different strategies for this procedure varying by the frequency of exchange during one iteration [25]. They are sketched in figure 10.

The first approach contains a complete exchange at block boundaries before each Runge-Kutta stage and before the computations of the residuals for the forcing functions of the multigrid algorithm.

The second possibility is, to update the block interface before each complete Runge-Kutta iteration step and again before computing the residuals for the forcing functions on the coarse grid.

Finally, a third strategy is carrying out the data exchange only once per grid level before the Runge-Kutta iteration step. All three techniques differ as far as the convergence behavior is concerned and in the memory needed, because the less exchange is performed the less data have to be stored intermediately.

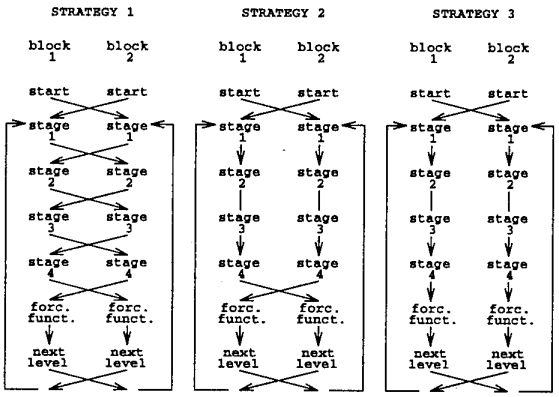


Fig. 10 Strategies for exchange of data at block interfaces.

The complete procedure is realized as an in-core solver as well as an off-core solver locating all block data on an external storing device. Therefore, large problems usually exceeding the main memory capacity as a whole can be solved.

5.3 Parallelization of the FLOWer code

As already pointed out, the parallelization of the FLOWer code followed the guidelines which were explained above. Therefore, parallelization meant integration of calls of the CLIC-library at distinct locations within the code. This leads to the structure of software layers sketched in figure 8 which illustrates how parallelization and portability are achieved at once.

Since the CLIC-library is based on the PARMACS message passing interface, there are two different programs necessary for a parallel run, called host and node program (figure 2). This feature was used, in order to establish a possibility for applying the FLOWer code as well as on parallel as on sequential computers:

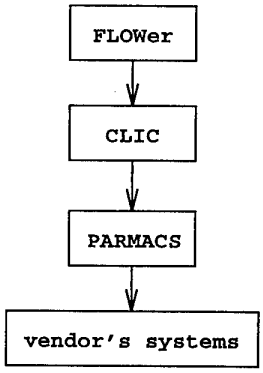


Fig. 11 Software layers of the parallelized FLOWer code.

The host program is only needed in parallel mode and performs the I/O-operations. It creates and starts the node processes and distributes the initial data correspondingly, i. e. grid coordinates and global control data. During the program run the host process receives the convergence information from all nodes and prints it to the standard output. At the end it collects the solution data from the nodes and writes them to the specified output files.

All parallel output operations performed by the host process are completely hidden from the user, since they are driven from the node process. There is only one call of the CLIC-library necessary, in order to initiate the communication between the host and the nodes, the rest is carried out automatically.

The node program contains the complete sequential flow solver. There is only one parameter to be specified by the user which switches between routines of the CLIC-library and standard sequential procedures. Therefore, the parallel mode differs essentially only in four points from the sequential mode:

- Input read operations are replaced by reception of input data from the host process.
- Global operations involving all blocks of the given block structure are performed by the CLIC instead of within do loops over all blocks.
- The exchange of data at block interfaces is carried out fully automatically within distinct CLIC-routines. There is no intermediate storing or reordering of data necessary any more.
- Write statements for putting out data are replaced by parallel write operations of the CLIC consisting of an initialization, an output format, output data and a termination procedure.

These differences are all only slight additions to the sequential program, such that the advantages of the applied guidelines described above become quite clear:

- The program is fully portable, since the FLOWer code and the CLIC-library are portable for themselves.
- All modifications do not touch the numerical algorithm, so that users and developers can keep their well known environment.
- The parallelization effort is extremely low, when assuming an existing communications library CLIC.

6. RESULTS

After integrating the CLIC calls into the FLOWer program structure, several test computations and benchmarks have been carried out, in order to demonstrate the success of the approach chosen and for investigating the potential of parallelization of a real application code. These will be reported on in the following.

6.1 Test Cases

Two different test cases were defined for comprehensive studies of the performance of computers and networks representing typical problems in aerodynamics while still remaining simple.

The first problem to be solved was the inviscid flow around a non-swept wing consisting of NACA0012 airfoils at a free stream Mach number of M = 0.6 and an incidence of $\alpha = 0°$. For this case two different grids were generated, a coarse one consisting of 160 x 32 x 8 cells and a fine one consisting of 320 x 64 x 16 cells. Both gridsmainly were subdivided into 1, 4 and 8 blocks of equal size as shown in figure 12. This subdivision was driven further for the fine grid, giving a 16 block and a 32 block case. Each computation consisted of 100 multigrid W-cyles involving three mesh levels, where the wall clock time was measured between the start of the initialization of the solution and the end of the iterations.

The second test case was the DLR-F4 wing-body combination, a generic, Airbus like aircraft given in figures 13 and 14. Here, the inviscid flow was computed at a free stream Mach number of M = 0.75 at $\alpha = 0°$ incidence. The C-grid consists of 256 x 40 x 40 cells and is blocked along the C-lines into 1, 4 and 8 blocks of equal size respectively. Each computation consisted of 35 multigrid W-cycles involving four mesh levels. The time measurement was carried out as described above.

In both cases each block was mapped to one processor.

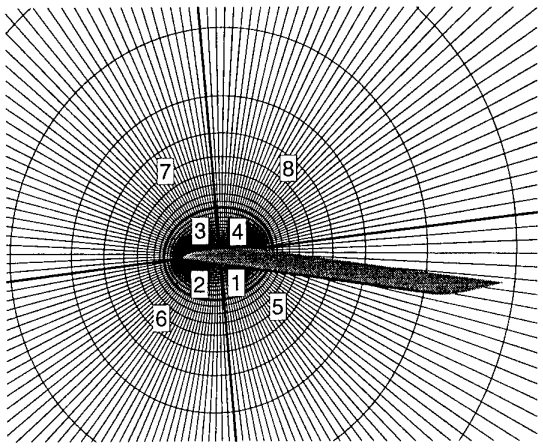


Fig. 12 Block structure for the NACA 0012 wing test cases

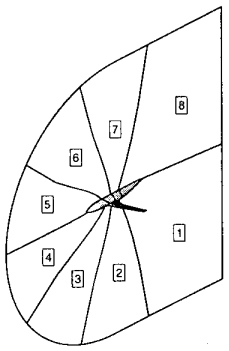


Fig. 13 Block structure of the DLR-F4 wing-body combination.

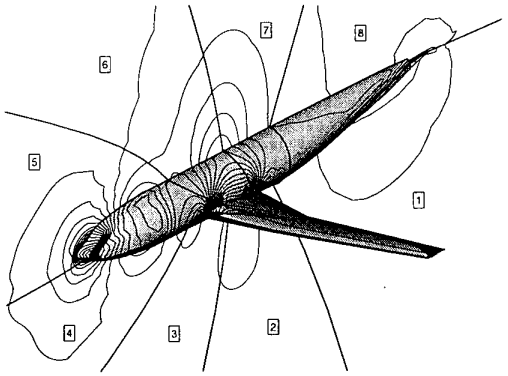


Fig. 14 Iso-Mach-contours on the DLR-F4 wing-body combination, M = 0.75, $\alpha = 0\,^{o}$.

6.2 Methods of Performance Assessment

Different quantities have been evaluated, especially for the NACA 0012 test case, in order to assess various parallel and sequential computers. This procedure is necessary, if one wants to get information on the real performance of a computer, since a restriction to only one criterion could possibly give a wrong impression of a computer's abilities. Therefore, some characteristic values are defined in the following.

6.2.1 Speed-up

The speed-up gives the value of acceleration obtained by employing several CPUs for a problem of a given size. It is defined as

$$S = \frac{t(N_p=1)}{t(N_p)}$$

which is the ratio of wall clock times needed by one processor and by N_p processors. This is usually compared with the number of processors used which is called linear speed-up. The true speed-up is always deviating from the linear one, because employing several CPUs is always creating an overhead for communication.

Since the blocking creates an additional overhead for computing block interfaces multiply (once per block) by the FLOWer code, an algorithmic speed-up is defined as

$$S_{alg} = N_B \cdot \frac{t(N_B=1)}{t(N_B)}$$

which is the ratio of computing times for the one block case and the N_B block case on a single processor multiplied with the number of processors which could be employed, i. e. the number of blocks. This value gives the algorithmically possible speed-up for a given problem.

6.2.2 Efficiency

The efficiency of a parallelization denotes the degree up to which the theoretically linear speed-up is reached, i. e.

$$E = \frac{S}{N_P}.$$

Because of the algorithmic overhead to be expected by the blocking, an algorithmic efficiency can be defined by

$$E_{alg} = \frac{S}{S_{alg}}$$

showing the degree up to which the parallel code reaches the algorithmically possible speed-up. Therefore, the standard efficiency is a global indicator for the degree a parallel code is exploiting a given machine, whereas the algorithmic efficiency characterizes the quality of the parallelization itself.

6.2.3 Relative Performance

Since speed-up and efficiency are both related to measurements on the same computer, a comparison between different machines is mandatory for a true assessment of a parallel program. Therefore, one can define a relative performance

$$P_{rel} = \frac{t_R}{t}$$

which is the ratio of the computing time on a reference processor (usually a Cray C90) and the time needed on the benchmark machine. This value allows comparisons even between parallel and sequential computers, given that the program will perform on all platforms.

6.2.4 Concluding Remarks

Assessing computers using the above definitions is still problematical and has to be done with great caution. Speed-up and efficiency as isolated values do not say anything about the quality of a computer or the program, since they lack any information about the absolute time needed. For example figure 15 shows the speed-up obtained for a two-dimensional computation of the flow around a NACA 0012 airfoil. On 160 blocks a speed-up of 125 was reached on a Parsytec GCel. A great value, but the time still exceeded that one obtained on only four processors of an IBM SP1 [28]

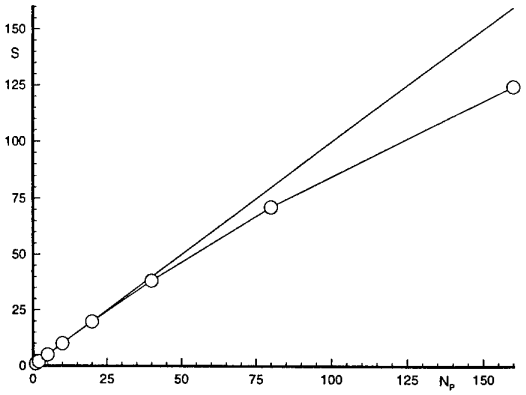


Fig. 15 Speed-up versus processor number on Parsytec GCel for a two-dimensional NACA 0012 airfoil [28].

Furthermore both quantities, as defined here, are related to the solution of a problem of a given size employing an increasing number of processors. Therefore, it is necessary that the problem can be solved completely on a single CPU of a machine for computing these values. But parallelization is done for solving future problems exceeding today's single processor capabilities.

Therefore, all results given here can only be taken as a general information on today's abilities of parallel computers with respect to sequential machines. In addition, it is the potential of parallel processing in CFD which can be indicated.

6.3 Comparison of Computer Performance

The NACA 0012 wing test case has been used for comparative performance measurements on a number of computers of different architecture. The results are given as histograms in figures 16 and 17 showing the relative time needed with respect to that one measured on a C90 single processor.As one can see in figure 16, the workstations tested cannot compete even with older vector computers as the Cray Y-MP, as far as performance is concerned. Their application is therefore restricted to research and development duties.

Within the class of vector computers, the NEC SX-3, which is the DLR working horse is clearly the strongest machine outperforming the reference Cray C90 processor. On this machine a sustained performance above 1 GFLOP/s was achieved.

Switching to figure 17, one sees that older parallel computers, i. e. the CM5 and the Intel Paragon, need many processors, i. e. blocks, in order to reach a high performance. For the case tested here consisting of eight blocks, they can only compete with single processor workstations, since their node CPUs are too weak. Another problem showed to be their little main memory, such that the large test case could not be computed on them using less than 32 processors. Since the results for the coarse grid were already disappointing, this calculation was not carried out.

Somewhat more promising are the results obtained on an IBM SP1. On eight nodes using the fastest communication system available one can almost reach the performance of older generation vector processors as the Cray Y-MP.

More recent distributed memory parallel computers which have been tested then, revealed that they are able to compete even with today's vector machines. If the problem is sufficiently large, the C90 single processor performance can almost be reached employing 32 nodes of a NEC Cenju-3. Using the same number of CPUs on an IBM SP2 the C90 single processor is already outperformed.

A special case is the result of the J916, since it is a shared memory vector computer. As one can see in this case, eight

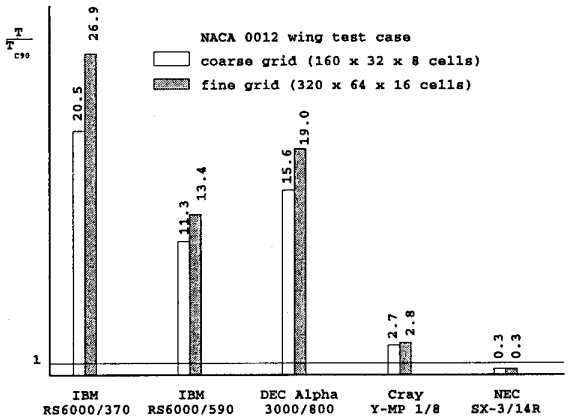


Fig. 16 Relative performance of sequential computers obtained for the NACA 0012 wing test case (160x32x8cells) with respect to a Cray C90 single processor.

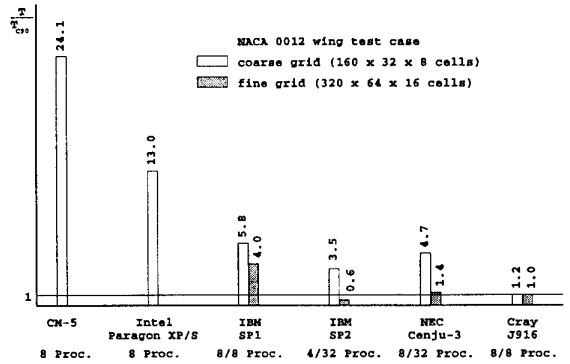


Fig. 17 Relative performance of parallel computers obtained for the NACA 0012 wing test case (160x32x8cells) with respect to a Cray C90 single processor.

computing nodes are already sufficient for reaching the C90 single processor performance.

In figure 18 the development of the relative performance is plotted versus the respective processor number involved for the most powerful parallel computers tested. As one can see, the highest performance is achieved on the computers with the most powerful single processors. On the other hand, the more powerful the computers are the worse their scalability becomes, i. e. the less steep is the slope of the corresponding curve.

This clearly indicates that on all these computers the performance is mainly gained by an increase in the single processor performance, whereas the network cannot keep track. The better scalability of the less powerful parallel

computers is therefore not gained by an improved network speed, but by weaker processors leading to a better balance of both components.

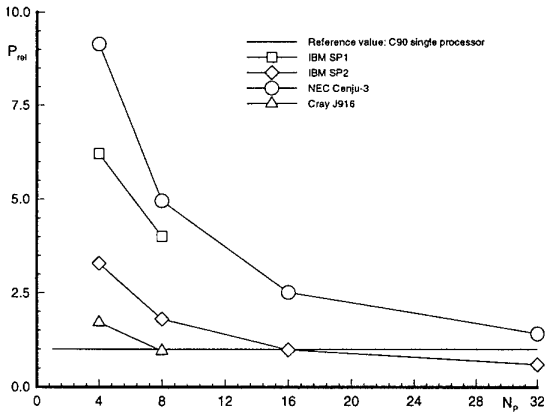


Fig. 18 Development of the relative performance versus processor number with respect to a C90 single processor.
Test case: NACA 0012 wing, 320x64x16 cells.

6.4 Influence of the Communication System

It is clear that the performance of parallel computers is mainly influenced by the communication system including hardware and software aspects. This effect was studied computing both test cases, the NACA 0012 wing and the DLR-F4 wing-body combination, on an IBM SP1 using different communication systems available there. The results of the measurements are given as speed-up versus number of processors in figures 19 to 21.

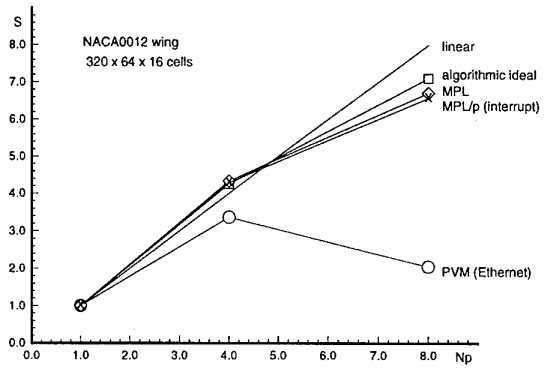


Fig. 20 Speed-up versus processor number on IBM SP1. Test case: NACA 0012 wing, fine grid (320x64x16 cells).

There have been tested five different communication systems:

- PVM using Ethernet
- PVM using the IBM High Performance Switch
- MPL (POE)
- MPL/p (euih) in default configuration
- MPL/p (euih) with interrupt control.

Figures 19 to 21 clearly indicate that PVM using an Ethernet connection is not suited for the CFD problems treated here, i. e. workstation clusters with an Ethernet connection are definitely not suitable for replacing a true parallel computer at least for the FLOWer code.

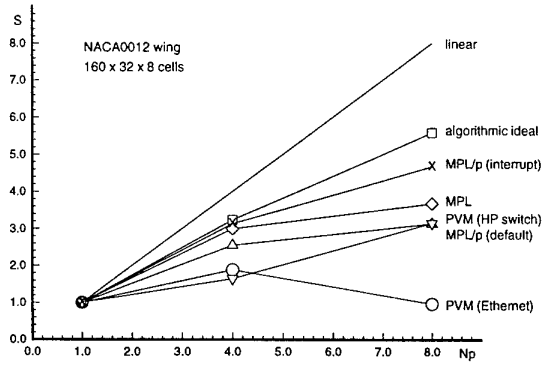


Fig. 19 Speed-up versus processor number on IBM SP1. Test case: NACA 0012 wing, coarse grid (160x32x8 cells).

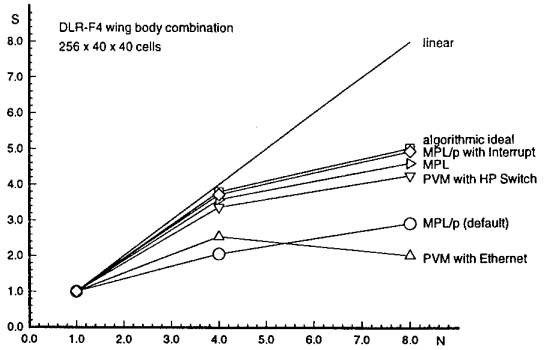


Fig. 21 Speed-up versus processor number on IBM SP1. Test case: DLR-F4 wing-body combination (256x40x40 cells).

The main reason for the speed down on eight nodes is the low performance of the Ethernet as can be seen from the improvement using PVM with the high performance switch. Nevertheless there is still too much software overhead within the communication which is drastically reduced by applying the IBM proprietary systems.

With the fastest systems the algorithmically ideal speed-up is reached up to an acceptable degree depending on the problem size. One can clearly perceive an increase of the speed-up when increasing the work load per processor, i. e. the block size.

For the larger NACA 0012 wing test case even a super linear speed-up was obtained (figure 20) which is due to a paging effect. Indeed, the one block case exceeded the main memory capacity of a single CPU, such that this is a typical case where parallel processing becomes advantageous while speed-up measurements are questionable.

Another observation is, that the algorithmically ideal speed-up considerably deviates from the linear one due to the algorithmic overhead because of multiple computations on block interfaces. This overhead is reduced of course, when increasing the block size per processor, as can be seen from a comparison of both NACA 0012 wing test cases.

On the other hand this overhead is remarkably increasing, when involving a fourth multigrid level, as is done for the DLR-F4 wing-body combination. Since it is W-cycles which are performed, much more time is spent on coarse grids where the ratio of boundary points to field points is getting worse. As it seems, the FLOWer code behavior there is dominated by the corresponding algorithmic overhead at the inter block boundaries and not by the increasing communication activity, because the algorithmic ideal is reached to a high degree indicating an excellent parallelization efficiency.

6.5 Comparison of Communication Models

Finally it is possible to compare the efficiencies of different communication models using the FLOWer code on shared memory computers. Therefore, measurements carried out on a Cray C916 and a Cray J916 computer using on the one hand the CLIC-library, i. e. exploiting coarse grain parallelism by message passing, and on the other hand using the auto-parallelizing compiler distributing parallel data to different processors. In the latter case the CLIC library was replaced by a dummy library, and no additional compiler directives were used. The message passing solutions were obtained for the multi block cases, whereas the auto-parallelizer worked on the single block problems.

The results of the measurements are given in figures 22 to 25 as speed-up versus number of processors for the two NACA 0012 wing test cases. What can be seen, is that only for the coarse grid problem the message passing approach is working slightly worse than the auto-parallelization approach, although it is creating a considerable algorithmical overhead at block interfaces as pointed out above. For the fine grid problem the parallelization via CLIC not only is competitive on the Cray C916, but even outperforms the auto-parallelization on the Cray J916.

Furthermore the scalability of the loop based data parallel approach is rather poor which is indicated by the strong non-linear deviation of the speed-up from the linear one. Using message passing this deviation is higher for small processor numbers, but remains almost linear, at least until eight CPUs, such that it is to be expected that this approach is performing better for large processor numbers.

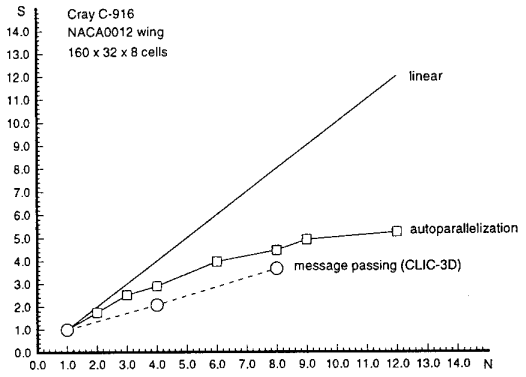


Fig. 22 Speed-up versus processor number on Cray C916. Test case: NACA 0012 wing, coarse grid (160x32x8 cells).

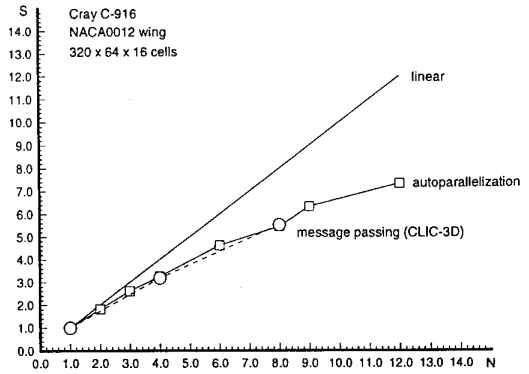


Fig. 23 Speed-up versus processor number on Cray C916. Test case: NACA 0012 wing, fine grid (320x64x16 cells).

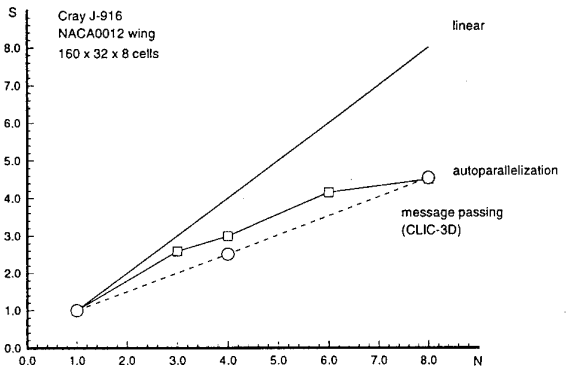


Fig. 24 Speed-up versus processor number on Cray J916. Test case: NACA 0012 wing, coarse grid (160x32x8 cells).

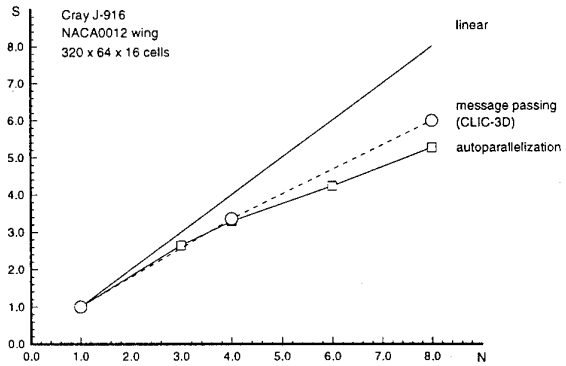


Fig. 25 Speed-up versus processor number on Cray J916. Test case: NACA 0012 wing, fine grid (320x64x16 cells).

The reasons for that interesting behavior might be the following [26]:

First of all, employing the CLIC-library creates some software overhead necessary for the operations involved in the communication. In addition, the algorithmic overhead due to multiple computations at block boundaries further decreases the parallel efficiency to be obtained by the FLOWer code. This explains the somewhat expected behavior for small processor numbers, that a vendor's specific strategy outperforms a portable one.

On the other hand the auto-parallelizing approach is restricted to the distribution of array data within loops to different processors. Of course there remains a body of operations outside of loops, i. e. scalar operations. These are excluded from the parallelization using an auto-parallelizing compiler, but of course take part in the coarse grain parallelization based on the block structure. Therefore, the number of operations which cannot be performed in parallel is higher for the data parallel approach than for the message passing approach. Due to Amdahl's law [27]

$$S = \frac{N_P}{1 + f \cdot (N_P - 1)}$$

where f is the portion of operations which cannot perform concurrently, this must lead to a higher speed-up theoretically to be obtained by the parallelization via CLIC, since the value of f is smaller there reducing the denominator of the above expression.

Another reduction of the speed-up gained by auto-parallelization is caused by small load imbalances which are indicated by the small wiggles of the speed-up curves for that approach. Depending on the strategy chosen for the distribution of concurrently processed data and depending on the number of array data to be treated, it can hardly be avoided that there will be processors computing slightly more data than others reducing further the efficiency. On the contrary the blocks of the test cases for the message passing parallelization were of equal size guaranteeing an ideal load balancing for that strategy.

What can be observed further, is that both techniques perform better with an increase of the problem size which is due to an increase of the vector length in either approach. But in the message passing solution additionally the communication and the algorithmic overhead becomes less important, since the local ratio of boundary data to field data is getting better. Therefore, the message passing efficiency becomes less dependent on the processor number, when increasing the problem size while keeping the work load constant per CPU.

The figures 22 to 25 clearly indicate that message passing is superior to auto-parallelizing compilers for sufficiently large blocks and for a sufficiently balanced ratio of communication to computational power. The latter is demonstrated by the Cray J916 results where the single CPU is much weaker than that one of a Cray C916, but where the message passing speed-ups are always above the corresponding ones on the Cray C916. Therefore, the crossover point for the message passing approach is reached earlier than on the C916 machine.

Of course these conclusions are only valid, when no additional compiler directive are put into the code for tuning, but this was excluded, for the reasons given with the guidelines for parallelization.

6.6 Massive Parallelism

It is clear that the benefits of parallelism will be greatest, when applying a higher number of processors assuming a sufficiently powerful network. There exist attempts to employ hundreds or thousands of CPUs working on the same problem at an overall performance of about 1 TFLOP/s. Therefore investigations with a two-dimensional code were carried out, in order to study the effects occuring in a massively parallel environment [28].

There were standard computations carried out on a Parsytec GCel for the flow field around a NACA 0012 airfoil at M = 0.8 and $\alpha = 1.25^{o}$ on an O-grid of 320 x 64 cells. The mesh was split according to different strategies in up to 160 blocks of equal size. The block structures varied with respect to the direction the grid was split, i. e. the mesh was subdivided in the normal direction j into 1, 2, 4 and 8 blocks keeping the number of blocks constant in the circumferential direction i at 1, 5, 10 or 20 blocks respectively. The results of these computations are shown in figure 26 where the obtained efficiencies are plotted versus the respective number of processors.

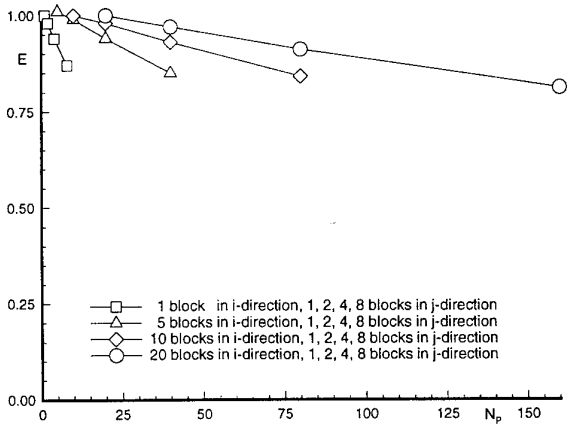


Fig. 26 Efficiency versus processor number for different block structures for the 2d NACA 0012 airfoil on Parsytec GCel.

As one can see, the efficiency varies remarkably between the different strategies applied. Moreover, it is essentially determined by the number of blocks in the normal direction j. The efficiency values for 8 blocks in j direction differ only between 85% and 81%, although the total number of blocks, i. e. processors, covers a range from 8 to no less than 160.

The reason for that behavior is found, when thinking about the communication pattern in the different cases. The time needed for communicating a set of data is usually given by the following linear relation

$$t_{comm} = a + bn$$

where a is the start-up time needed for initialization, b the bandwidth and n the number of data to be transferred. The latter usually is proportional to the number of points at a block interface.

In the case here the number of messages sent per block depends only on the number of neighbors. This value varies slightly from 2 (1 block in any direction) to 4 (4 or 8 blocks in any direction) in the worst case, but this does not differ between a blocking in i- and in j-direction. What counts, is that the blocks resulting from a splitting in the normal direction j always have longer edges along j than along i in the range considered, such that the length of at least part of the messages is always longer in j- than in i-direction (figure 27). Therefore, for this test case it is always profitable to achieve a given number of blocks by slicing the grid in the circumferential direction instead of a blocking in normal direction.

Therefore, when blocking a problem for parallelization purpose one should not only think of the load balancing problem, i. e. to produce equally sized subdomains, but also of an optimum grid partitioning with respect to the communication pattern.

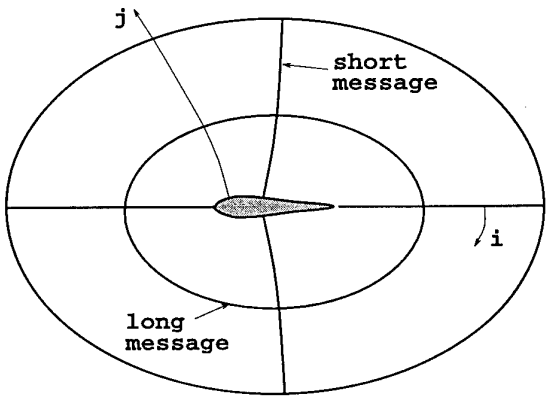


Fig. 27 Schematic block structure around the 2d NACA 0012 airfoil.

7. CONCLUSIONS

It has been shown that parallelization is an interesting method for accelerating large CFD solvers for production use, but for this class of programs parallelization cannot be treated in isolation. Moreover the requirements for portability, conservation of the effort spent for the numerical development in sequential mode and reduction of the paral-

lelization effort have to be met.Therefore, guidelines, how to proceed, are given which have been proven to lead to a parallel code fulfilling these industrial objectives on software.

The strategy suggested is based on grid partitioning using message passing for communication, since this technique corresponds to the well known multiblock approach in sequential programs. All functionalities involving communication between parallel nodes should be concentrated within a high level library guaranteeing portability and simplifying the parallelization task.

The communications library CLIC which is currently developed at the GMD within the POPINDA-project is such a library. Based on the portable message passing interface PARMACS it is supporting any block structured program.

As an example the parallelization of the FLOWer code is described which is developed for production use in aerodynamics. It is demonstrated that the chosen approach using the CLIC library allows this program to run on computers of any architecture ranging form single processor workstations up to shared and distributed memory parallel machines.

Comparisons of performance data obtained with the FLOWer code show that modern parallel computers are already able to reach the single processor performance of a Cray C90 processor employing a moderate number of nodes.

Studies on different communication systems demonstrate that the communication performance clearly determines the potential of parallel processing. As it comes out, workstation clusters connected by Ethernet are definitely not suitable for replacing true parallel computers, at least for CFD applications of the FLOWer code.

A comparison of different parallelization techniques on shared memory computers reveals that the portable message passing approach suggested is not necessarily inferior to vendor's auto-parallelizing compilers. It was demonstrated that only for small processor numbers the FLOWer code performs worse using the CLIC-library, but the scalability features of the message passing communication model appeared to be generally better than that of the data parallel model involving an auto-parallelizer.

Finally, studies on the behaviour of different block structures reveal a strong influence of the grid partitioning on the resulting communication amount yielding remarkable differences of the efficiency to be obtained in a parallel run.

8. OUTLOOK

Further development is to be carried out for the future, in order to improve the parallel behavior of the FLOWer code. Major effort will have to spent on the reduction of the algorithmical overhead at block intersections for increasing the absolute speed-up rates.

Furthermore investigations on the parallelization features of the program have to be devoted to Navier-Stokes computations, since up to now only Euler results have been studied.

Finally the integration of a local grid refinement has to be done within the research project POPINDA involving as well the FLOWer code as the communications library CLIC. Additional features of this library will be realized in the near future, i. e. an automatic load balancing and a special detection and treatment of mesh singularities.

9. ACKNOWLEDGEMENTS

The work reported on here has been funded by the German Ministry of Research within the parallelization project POPINDA involving the following organizations:

Daimler Benz Aerospace Airbus Bremen, Daimler Benz Aerospace DASA-LM Manching/Ottobrunn, Deutsche Forschungsanstalt für Luft- und Raumfahrt Braunschweig, Gesellschaft für Mathematik und Datenverarbeitung St. Augustin, IBM Wissenschaftliches Zentrum Heidelberg.

The authors want to thank all contributors of these institutions.

In addition they authors have to thank the CRAY and NEC corporations for carrying out several benchmarks with the FLOWer code on their machines.

10. LITERATURE

1. Holst, T. L., Salas, M., D., Claus, R. W., "The NASA Computational Aerosciences Program - Toward Teraflops Computing", AIAA-92-0558, 1992

2. Agarwal, R. K., "Parallel Computers and Large Problems in Industry" in "Computational Methods in Applied Sciences", Elsevier Science Publishers, 1992

3. Bailey, D. H., Barszcz, E., Dagum, L., Simon, H., "The NAS Parallel Benchmarks: Review and Current Results in "Supercomputer 1994", K. G. Saur Verlag, 1994

4. Haverkort, B. R., "High-Speed Networks for the Interconnection of Clusters of Workstations" in "Praxisorientierte Parallelverarbeitung", Carl Hanser Verlag, 1994

5. Lüpke, S., "Accelerated Access to Shared Distributed Arrays on Distributed Memory Systems by Access Objects" in "Lecture Notes in Computer Science", Springer-Verlag, 1994

6. Devloo, Ph. R. B., Fezoui, L., Lacire, St., "Object Oriented Programming Applied to Massively Parallel

Computing: A C++ Interface to the Connenction Machine" in "Numerical Methods in Engineering", Elsevier Science Publishers, 1992

7. Devloo, Ph., R., B., Filho, J. S. R. A., "On the Development of Finite Element Program Based on the Object Oriented Programming Filosophy" in "Numerical Methods in Engineering", Elsevier Science Publishers, 1992

8. Quintana, G. Vidal, A. M., "Parallel Householder Factorization on the Supernode Multicomputer" in "Numerical Methods in Engineering", Elsevier Science Publishers, 1992

9. Deshpande, M. Jinzhang, F., Merkle, Ch. L, Deshpande, A., "Implementation of a Parallel Algorithm on a Distributed Network", AIAA-93-0058, 1993

10. DrikakisS, D., Schreck, E., "Development of Parallel Implicit Navier-Stokes Solvers on MIMD Multi-Processor Systems", AIAA-93-0062, 1993

11. Kroll, N., Radespiel, R., Rossow, C.-C., "Accurate and Efficient Flow Solvers for 3D Applications on Structured Meshes", AGARD FDP/VKI Special Course on Parallel Computing in CFD, 1995

12. Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard", University of Tennessee, 1994

13. Sunderam, V. S., "PVM, a framework for parallel distributed computing" in "Concurrency, Practice and Experience", Vol. 2(4), pp. 315 - 339, 1990

14. Hempel, R., Hoppe, H.-C., Supalov, A., "PARMACS 6.0 Library Interface Specification", GMD St. Augustin, 1992

15. Gerndt, M., "Automatic Parallelization of a Crystal Growth Simulation Program for Distributed-Memory Systems" in "Proceedings of HPCN '94 Europe", Springer-Verlag, 1994

16. Schmatz, M. A., "Hypersonic Three-Dimensional Navier-Stokes Calculations for Equilibrium Gas", AIAA-89-2183, 1989

17. Keyes, D. E, "Domain Decomposition: A Bridge Between Nature and Parallel Computers", ICASE Report No. 92-44, 1992

18. Lonsdale, G., Schüller, A., "Multigrid efficiency for complex flow simulations on distributed memory machines", Parallel Computing 19, pp 23-32, 1993

19. Radespiel, R., Rossow, C., "A Cell Vertex Finite Volume Scheme for the Two-Dimensional Navier-Stokes Equations", DFVLR-IB 129-87/40, 1987

20. Baldwin, B. S., Lomax, H., "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows", AIAA-78-257, 1978

21. Jameson, A., Schmidt, W., Turkel, E., "Numerical simulation of the Euler equations by finite volume methods using Runge-Kutta time stepping schemes", AIAA-81-1259, 1981

22. Rossow, C.-C., "Berechnung von Strömungsfeldern durch Lösung der Euler-Gleichungen mit einer erweiterten Finite-Volumen Diskretisierungsmethode", DLR-FB 89-38, 1989

23. Kroll, N., Jain, R. K., "Solution of the Two-Dimensional Euler Equations - Experience with a Finite Volume Code", DFVLR-FB 87-41, 1987

24. Atkins, H., "A Multiple-Block Multigrid Method for the Solution of the Three-Dimensional Euler and Navier-Stokes Equations", DLR-FB 90-45, 1990

25. Rossow, C.-C., "Efficient Computation of Inviscid Flow Fields Around Complex Configurations Using a Multiblock Multigrid Method", Communications in Applied Numerical Methods 8, pp 737-747, 1992

26. Vogelsang, R., private communication

27. Amdahl, G., "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities", in AFIPS Conference proceedings, vol. 30 Atlantic City NJ, 1967, pp 483-485, cited in Bräunl, T., "Parallel Programmierung", Vieweg Verlag1993

28. Roll, B., Eisfeld, B., "Parallelisiserung des 2D-Eulercodes CEVCATS auf Parsytec GCel und IBM SP1", to be published as DLR-IB.

29. Pahlke, K., private communication

Parallel Automated Adaptive Procedures for Unstructured Meshes

M.S. Shephard, J.E. Flaherty, H.L. de Cougny, C. Ozturan, C.L. Bottasso and M.W. Beall
Scientific Computation Research Center
Rensselaer Polytechnic Institute
Troy, NY 12180–3590
USA

Summary

Consideration is given to the techniques required to support adaptive analysis of automatically generated unstructured meshes on distributed memory MIMD parallel computers. The key areas of new development are focused on the support of effective parallel computations when the structure of the numerical discretization, the mesh, is evolving, and in fact constructed, during the computation. All the procedures presented operate in parallel on already distributed mesh information. Starting from a mesh definition in terms of a topological hierarchy, techniques to support the distribution, redistribution and communication among the mesh entities over the processors is given, and algorithms to dynamically balance processor workload based on the migration of mesh entities are given. A procedure to automatically generate meshes in parallel, starting from CAD geometric models, is given. Parallel procedures to enrich the mesh through local mesh modifications are also given. Finally, the combination of these techniques to produce a parallel automated finite element analysis procedure for rotorcraft aerodynamics calculations is discussed and demonstrated.

Contents

1. Introduction
2. Parallel Control of Evolving Meshes
   - 2.1 Mesh Data Structure to Support Geometry-Based Automated Adaptive Analysis
   - 2.2 Partition Communication and Mesh Migration
     - 2.2.1 Requirements of PMDB and Related Efforts
     - 2.2.2 Distributed Mesh Model and Notation Used
     - 2.2.3 Data Structures
     - 2.2.4 Mesh Migration
     - 2.2.5 Scalability of Mesh Migration and Extensions
   - 2.3 Dynamic Load Balancing of Adaptively Evolving Meshes
     - 2.3.1 Geometry-Based Dynamic Balancing Procedures
     - 2.3.2 Topologically-Based Dynamic Balancing Procedures
3. Parallel Automatic Mesh Generation
   - 3.1 Introduction
   - 3.2 Background and Meshing Approach
   - 3.3 Sequential Region Meshing
     - 3.3.1 Underlying Octree
     - 3.3.2 Template Meshing of Interior Octants
     - 3.3.3 Face Removal
   - 3.4 Parallel Constructs Required
     - 3.4.1 Octree and Mesh Data Structures
     - 3.4.2 Multiple Octant Migration
     - 3.4.3 Dynamic Repartitioning
   - 3.5 Parallel Region Meshing
     - 3.5.1 Underlying Octree
     - 3.5.2 Template Meshing of Interior Octants
     - 3.5.3 Face Removal
4. Parallel Mesh Enrichment
   - 4.1 Local Retriangulation Tools
     - 4.1.1 Edge Swapping
     - 4.1.2 Edge Removal
     - 4.1.3 Multi-Face Removal
     - 4.1.4 Triangulation Optimization Using Local Retriangulation Tools
   - 4.2 Refinement
     - 4.2.1 Subdivision Patterns
     - 4.2.2 Generalized Bisection
     - 4.2.3 Alternate Bisection
     - 4.2.4 Delaunay Insertion
     - 4.2.5 Splitting
     - 4.2.6 Refinement Using Full Set of Subdivision Patterns
   - 4.3 Derefinement
   - 4.4 Complete Mesh Adaptation Procedure
   - 4.5 Parallelization of Mesh Adaptation
     - 4.5.1 Derefinement
     - 4.5.2 Triangulation Optimization
     - 4.5.3 Refinement
5. Parallel Adaptive Analysis Procedures
   - 5.1 Structure of a Parallel Adaptive Analysis Procedure
   - 5.2 Finite Element Code for Rotorcraft Aerodynamics
     - 5.2.1 Finite Element Formulation
     - 5.2.2 Boundary Conditions for Hovering Rotors
     - 5.2.3 Subsonic and Transonic Hovering Rotors
   - 5.3 Effectiveness of Parallel Adaptive Analysis Procedures
6. Closing Remarks
7. Acknowledgment
8. References

Paper presented in an AGARD-FDP-VKI Special Course on "Parallel Computing in CFD", held at the VKI, Rhode-Saint-Genèse, Belgium, from 15-19 May 1995 and 16-20 October 1995 at NASA Ames, United States and published in R-807.

Nomenclature

Notation used to describe models and topological entities within the models

- ${}_v\Omega$ Domain associated with model $v, v = g, p$ or m where g signifies the geometric model, p signifies the partition model, and m signifies the mesh model.
- $\overline{{}_v\Omega}$ Closure of domain associated with the model $v, v = g, p$ or m
- ${}_vT_i^d$ Topological entity i from model v of dimension d, $d = 0$ is a vertex, $d = 1$ is an edge, $d = 2$ is a face, $d = 3$ is a region.
- ${}^k_vT_i^d$ Indicates the kth use of the topological entity ${}_vT_i^d$. Use entities uniquely identify how entities are used in non-manifold models. The simplest case of uses arises from the fact that a face can be bounding two regions. One face use is associated with each region.
- ${}^{\pm}_vT_i^d$ The $\pm$ indicates a directional use of the topological entity ${}_vT_i^d$ as defined by its ordered definition in terms of lower order entities. A $+$ indicates use in the same direction, a $-$ indicates use in the opposite diection.
- $\partial({}_vT_i^d)$ Boundary of topological entity ${}_vT_i^d, v = g, p$ or m
- $\overline{{}_vT_i^d}$ Closure of topological entity defined as $({}_vT_i^d \cup \partial({}_vT_i^d))$, $v = g, p$ or m
- $\sqsubset$ Classification symbol used to indicate the association of one or more entities from one model, typically m or p, with a higher model, typically p or g

Groups of topological entities used in the definition of topological adjacencies

- $\{{}_vT^d\}^{(n)}$ Unordered group of n topological entities of dimension d
- $\lfloor {}_vT^d \rfloor^{(n)}$ Ordered group of n topological entities of dimension d
- $[{}_vT^d]^{(n)}$ Cyclicly ordered group of n topological entities of dimension d
- $\langle {}_vT^d \rangle^{(n)}$ Group of n topological entities of dimension d without order specified
- $\langle {}_vT^d \rangle_i^{(n)}$ The ith entity in a group of n topological entities of dimension d

Notation used to describe adjacency relationships for topological entities

- $\varphi\langle {}_vT^d \rangle^{(n)}$ Set of n topological entities of dimension d adjacent to, or contained in φ. φ may be an entity, ${}_vT_i^{d_i}$, or a group of entities, $\langle {}_vT^d \rangle$

Examples of adjacency groups

- $v\{{}_vT^d\}$ All model entities of dimension d in model v
- $v\{{}_vT^d\}_i$ The ith entity of dimension d in model v in the group. Note that $v\langle {}_vT^d \rangle_i = {}_vT_j^d$
- ${}_vT_i^{d_i}\{{}_vT^{d_j}\}^{(n)}$ The n entities of dimension d_j adjacent to entity ${}_vT_i^{d_i}$

Adjacency relationships are evaluated left to right. For example ${}_vT_i^3\{{}_vT^0\}\{{}_vT^3\}$ is found by first finding the group defined by $\varphi = {}_vT_i^3\{{}_vT^0\}$ and then by defining the group $\varphi\{{}_vT^3\}$

1. Introduction

Adaptive techniques provide the promise of reliably solving many complex flow problems to the desired level of accuracy. The computational requirements of these solution processes can only be met by scalable parallel computers. The development of effective parallel algorithms for adaptive techniques is challenging due to the irregular nature of adaptive discretizations and the constant modification of the discretization. These notes discuss the techniques required to support automated adaptive analysis on distributed memory MIMD parallel computers.

Three assumptions underlying the techniques presented are (i) the parallel computation algorithms assume a partitioning of the mesh onto the processors, (ii) the meshes are unstructured, and (iii) the mesh generation and enrichment processes interact directly with a geometric definition of the domain being analyzed as it exists in a CAD system. These assumptions have a defining influence on the procedures developed. The most critical of the assumptions is the direct link to the CAD definition of the domain which allows the adaptive procedures to solve the problem over the intended domain, not some approximation based on an initial mesh. The results of our adaptive CFD calculations clearly demonstrate that adaptive results in which the mesh enrichments do not improve the geometric approximation often yield no improvement in the solution accuracy. This is because the adaptive procedure is obtaining a better solution to the wrong problem.

A key aspect to supporting calculations on adaptively evolving mesh is the data structure used to describe the mesh and support its evolution during the adaptive analysis process. When the analyses are performed on parallel computers, capabilities must be available to support the communications between the partitions of the mesh assigned to various processors. As the mesh is adapted, partition work load becomes unbalanced, therefore procedures must be available to effectively modify the mesh partitions to regain load balance for the next computational step. Chapter 2 of these notes presents a set of data structures and algorithms for the effective parallel control of evolving meshes.

The demand for continuously larger meshes indicates the need for the development of efficient parallel automatic mesh generators which can operate directly from the geometric representations housed in CAD systems. Chapter 3 of these notes discusses the issues of automatic mesh generation from solid models and presents an algorithm

for parallel mesh generation. Although the mesh enrichments dictated by an adaptive analysis can be satisfied through remeshing by the automatic mesh generator, the computational cost and need to project parameters between meshes indicates the desire to employ alternative mesh enrichment techniques when possible. Chapter 4 presents a set of local mesh modification procedures for the effective refinement and coarsening of meshes.

Given a set of parallel procedures for controlling mesh partitions, for the generation and enrichment of the mesh, the remaining ingredient of the automated adaptive analysis is the adaptive solver. Consistent with the other component procedures presented in these notes, it is assumed that the solver operates on an unstructured mesh which has been partitioned to the various processors of the parallel computer. Under this assumption, adaptive finite volume and finite element solvers are most appropriate. Chapter 5 presents the structure of such a solver. The specific solver discussed is a finite element based procedure which builds directly on the parallel mesh control tools of the earlier sections.

2. Parallel Control of Evolving Meshes

Central to the parallel automated adaptive analysis procedures considered here are tools to control the mesh and its distribution among the processors as the meshes are generated and analyzed. These tools must be able to maintain load balance as the mesh evolves during the computations in such a manner that the interprocessor communications are kept as small as possible. Its is also critical that these procedures operate in parallel and scale as the problem size grows so they do not become the bottleneck in the parallel computation process.

The tools required to support parallel automated adaptive analysis include:

1. data structures and operators to support the model representations employed
2. interprocessor communication control mechanisms
3. mechanisms to effectively move portions of the discrete models generated to various processors so load balance can be maintained
4. techniques to partition the mesh among the processors so the load is balanced and communications are minimumized
5. techniques to up-date the mesh partitions to regain load balance which was lost due to mesh modifications

The minimum data structures needed for an automated adaptive analysis are (i) a problem definition, in terms of a geometric model and analysis attributes, and (ii) a mesh, which the discrete representation used by the analysis procedures. The next section describes a general structure, based on boundary representations, for the problem definition and the mesh. This same form of structure is used to support the partition model used by the partition operators, mesh migration procedures and dynamic load balancing procedures. In additional to these data structures, several procedures described employ tree structures to support searching and spatial enumeration. The mesh partition procedures described in section 2.2 are designed to effectively collect groups of mesh entities for migration and, using the interprocessor communication operators, transfer the information and update all local data structures as needed.

A number of algorithms have been developed to partition a given mesh to a set of processors. The interested reader is referred to references [4, 20, 21, 56, 80] for more information. The current document focuses on procedures to update an existing set of mesh partitions after the mesh has been modified by a mesh adaptation procedure. Section 2.3 presents two classes of procedures for this purpose.

2.1. Mesh Data Structure to Support Geometry-Based Automated Adaptive Analysis

The classic unstructured mesh data structure of a set of node point coordinates and element connectivities is not sufficient for supporting automated adaptive analysis. Richer structures are required to support adaptive mesh enrichment procedures and to provide the links to the original domain definition needed by critical functions, including ensuring that the automatic mesh generator has produced a valid discretization of the given domain. A number of alternative mesh data structures have been proposed for various forms of mesh adaptation. Instead of describing and comparing these structures, a general data structure based on a hierarchy of topological entities is given.

The goal of an analysis process is to solve a set of partial differential equations over the geometric domain of interest, ${}_g\Omega$. Generalized numerical analysis procedures employ a discretized version of this domain in terms of a mesh. Since the mesh domain, $\overline{{}_m\Omega}$ may not be identical to the original geometric domain, $\overline{{}_g\Omega}$, and/or various procedures, such as automatic mesh generation, adaptive mesh refinement and element stiffness integration need to understand the relationship of the mesh to the geometric model, it is critical to employ a representational scheme which maintains the relationships between these two models. Although a number of schemes are possible for defining a geometric domain [58], the most advantageous for the current purposes are boundary-based schemes in which the geometric domain to be analyzed is represented as

$$\overline{{}_g\Omega}(g\{{}_gS\}, g\{{}_gT\}) \tag{1}$$

where $g\{{}_gS\}$ represents the information defining the shape of the entities which define the domain and $g\{{}_gT\}$ represents the topological types and adjacencies[1] of the

[1] In the context of a domain representation, adjacencies are the relationships among topological entities which bound each other. For example, the edges that bound a face, is a commonly used topological adjacency.

entities which define the domain. In addition to being unique, the use of topological entities and their adjacencies provides a convenient abstraction for defining the relationship of different models of the same domain. Boundary representations also allow the convenient specification, with respect to the geometric domain, of the analysis attributes of material properties, loads, boundary conditions and initial conditions [72, 75]. An additional advantage of boundary representations is the fact that current computer aided design systems support a boundary representation of the domains defined within them. This allows the effective combination of these packages with automatic mesh generation. A final advantage of recent boundary representations are their ability to properly represent the non-manifold geometric domains commonly used for analysis processes [89, 32].

Since individual volume finite elements will be limited to simple regions, bounded by simply connected faces, consideration of the topological entities for a model can focus on the basic 0 to *d* dimensional topological entities, which for the three-dimensional case (*d*=3) are:

$$v\{{}_vT\} = \{v\{{}_vT^0\}, v\{{}_vT^1\}, v\{{}_vT^2\}, v\{{}_vT^3\}\} \quad (2)$$

where $v\{{}_vT^d\}$, $d = 0, 1, 2, 3$ are respectively the set of vertices, edges, faces and regions defining the primary topological elements of the domain[2].

Critical to the understanding of the relationship of the mesh with the geometric domain is the concept of classification of a derived model to its parent model [66, 67].

Definition: Mesh Classification Against the Geometric Domain — The unique association of a topological mesh entity of dimension d_i, ${}_mT_i^{d_i}$, *to a topological geometric domain entity of dimension* d_j, ${}_gT_j^{d_j}$, *where* $d_i \leq d_j$, *is termed classification and is denoted*

$${}_mT_i^{d_i} \sqsubset {}_gT_j^{d_j} \quad (3)$$

where the classification symbol, $\sqsubset$, *indicates that the left hand entity, or set, is classified on the right hand entity.*

Multiple ${}_mT_i^{d_i}$ can be classified on a ${}_gT_j^{d_j}$. A mesh region, ${}_mT_i^3$, is classified in the domain region, ${}_gT_j^3$, in which it lies. A mesh face, ${}_mT_i^2$, is classified in the domain region, ${}_gT_j^3$, in which it lies, or on the domain face, ${}_gT_j^2$, on which it lies. A mesh edge, ${}_mT_i^1$, is classified in the domain region, ${}_gT_j^3$, in which it lies, on the domain face, ${}_gT_j^2$, on which it lies, or on the domain edge, ${}_gT_j^1$, on which it lies. Finally, a mesh vertex, ${}_mT_i^0$, is classified in the domain region, ${}_gT_j^3$, in which it lies, on the domain face, ${}_gT_j^2$, on which it lies, on the domain edge, ${}_gT_j^1$, on which it lies, or on the domain vertex, ${}_gT_j^0$, on which it lies. Mesh entities are always classified with respect to the lowest order object entity possible.

Any automated adaptive analysis must consider both the geometric domain representation, $\overline{{}_g\Omega}(g\{{}_gS\}, g\{{}_gT\})$, and the mesh representations, $\overline{{}_m\Omega}(m\{{}_mS\}, m\{{}_mT\})$ where $m\{{}_mS\}$ is limited to pointwise information at specific locations obtained by interrogation of the geometric model representation. Since the mesh representation lacks the complete geometric shape information of the geometric domain representation, that shape information must be accessed during various operations such as integrating elements to the true geometry, or placing new nodes defined by adaptive refinement on the true boundary of the domain.

Classification of the mesh against the geometric domain is central to (i) ensuring that the automatic mesh generator has created a valid mesh [66, 67], (ii) transferring analysis attribute information to the mesh [75], (iii) supporting h-type mesh enrichments, and (iv) integrating to the exact geometry as needed by high order elements.

In addition to the mesh representation, it is often desirable to consider other derived representations of the domain. The one of central importance to the parallel adaptive analysis is the processor representation, $\overline{{}_p\Omega}$. This representation is an intermediate representation between that of the mesh and the geometric domain. Therefore, its topological entities can be classified against the geometric domain. Since the mesh is the lowest order representation, its entities can be classified against both the geometric domain and the processor representation.

An additional representation employed in the parallel mesh generation procedure, and one set of parallel adaptive procedures, is an octree representation. Since tree representations are derived to support specific searching operations, or spatial enumerations, they vary dramatically from the topological hierarchies used to define the geometric domain and mesh. Structures of these types will be described as they are used in specific algorithms.

The adjacencies of various order mesh topological entities and their classification with respect to the higher order models are used to support a great number of the operations required by a parallel automated adaptive analysis. Therefore, it is important that they can be quickly determined. Clearly, if the adjacencies of each order entity against all other entities were stored, all possible adjacency information would be readily available. This approach would be highly wasteful with respect to the amount of data storage required. On the other hand, storing only a minimal number of adjacencies could require extensive searches and sorts to determine other specific adjacencies. An examination of the specific adjacencies used by the various algorithmic operations provides guidance as to the minimum number of adjacencies needed. For example references [6, 13, 30, 38] define adjacencies used in specific finite volume and finite element procedures. Since the procedures considered here must support the highly demanding, from the view point of topological adjacencies, automatic mesh generation procedures,

[2] Proper consideration of general geometric domains requires consideration of the loop and shell topological entities, and, in the case of non-manifold models, use entities for the vertex, edge, loop, and face entities [89]. We will introduce any of these additional entities only as needed.

and any form of adaptive analysis on conforming unstructured meshes[3], all adjacencies are either stored, or can be quickly determined through a set of local traversals and sorts which are not a function of the mesh size. One set of relationships that can effectively meet these requirements is to maintain adjacencies between entities one order apart. Figure 1 graphically depicts this set of relationships as well as the classification with respect to the geometric domain representation.

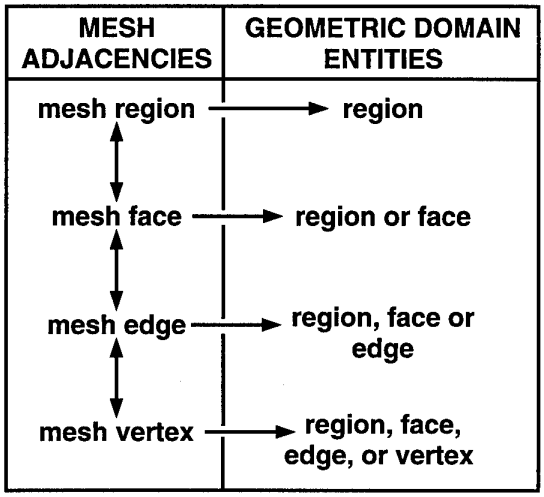


Figure 1. Mesh topological adjacencies and classification information

Since there are natural orderings for several of the adjacencies which prove useful to the operations performed, the forms of adjacencies employed are: an unordered list of n entities adjacent to entity ϑ signified by $\vartheta\{{}_vT^d\}^{(n)}$, a linear list of n entities adjacent to entity ϑ signified by $\vartheta\lfloor{}_vT^d\rfloor^{(n)}$, and a cyclic list of n entities adjacent to entity ϑ signified by $\vartheta\lceil{}_vT^d\rceil^{(n)}$. Specific entities also store directional knowledge of how that entity is used in the specific adjacency. In these cases the left superscript, $\pm$, on the entity, ${}^{\pm}_vT^d_i$, indicates a directional use of the topological entity ${}_vT^d_i$ as defined by its ordered definition in terms of lower order entities. A + indicates use in the same direction, while a − indicates a use in the opposite direction.

The specific downward adjacencies stored are:
For mesh regions

$$ {}_mT^3_i\{{}^{\pm}_mT^2\}^{(n)} \tag{4}$$

which indicates the faces bounding the mesh region, where $n = 4$ for a tetrahedron, $n = 6$ for a hexahedron, etc.

[3] A conforming mesh is one where all mesh entities exactly match. For example, a situation where the mesh edge bounding one mesh face has two mesh edges from another mesh face lying exactly on top of it is not allowed. Although possible to extend the procedures presented here to support those situations, they will not be considered in the present document.

For mesh faces

$$ {}_mT^2_i\lceil{}^{\pm}_mT^1\rceil^{(n)} \tag{5}$$

which defines the loop of edges that bound the face, where $n = 3$ for a triangular face and , $n = 4$ for a quadrilateral face.
For mesh edges

$$ {}_mT^1_i\lfloor{}_mT^0\rfloor^{(2)} \tag{6}$$

which indicates the two vertices that bound the edge.
The specific upward adjacencies stored are:
For mesh vertices

$$ {}_mT^0_i\{{}_mT^1\}^{(n)} \tag{7}$$

which indicates the edges the vertex is on the boundary of.
For mesh edges

$$ {}_mT^1_i\{{}_mT^2\}^{(n)} \tag{8}$$

which indicates the faces the edge partly bounds.
For mesh faces

$$ {}_mT^2_i\lfloor{}_mT^3\rfloor^{(2)} \tag{9}$$

which indicates the zero, one, or two regions the face partly bounds.
An alternative set of adjacencies which can directly meet the needs of many applications is to maintain the same downward adjacencies and store only the single upward adjacency from the vertices to the highest order entities using them. In the case of a manifold mesh in 3–D this upward adjacency would be

$$ {}_mT^0_i\{T^3\} \tag{10}$$

which are the regions that the vertex bounds. In the case of general non-manifold models, it is the upward adjacencies form the vertices to any mesh entity it bounds which itself is not bounded by a higher order entity. In this case the adjacency relationship is a bit more complex being

$$ {}_mT^0_i\Big\{{}_mT^3, {}_mT^2, {}_mT^1 \;\Big|\; |{}_mT^2\lfloor{}_mT^3\rfloor| = 0, \; |{}_mT^1\{{}_mT^2\}| = 0\Big\} \tag{11}$$

This set includes the regions the vertex bounds, the faces the vertex bounds which do not bound any regions, and the edges the vertex bonds which do not bound any faces.

2.2. Partition Communication and Mesh Migration

Adaptive unstructured meshes on distributed memory computers require data structures which provide efficient queries for various entity and processor adjacency information as well as fast updates for changes in the mesh. The requirements for sequential implementations of hp-adaptive finite element methods can be satisfied by the SCOREC mesh database just given. For parallel applications, we first enumerate the major requirements of a distributed memory mesh environment. These requirements are met by the distributed mesh environment Parallel Mesh Database (PMDB) that is then described.

2.2.1 Requirements of PMDB and Related Efforts

A parallel mesh database must:

- Provide a common interface and a single library for all the mesh related applications, namely, mesh generation, mesh refinement/coarsening and finite element analysis.
- Provide a full spectrum of adjacency relations among shared entities on different processors.
- Provide a general purpose mesh migration algorithm which will facilitate arbitrary mobility of mesh entities on processors. Additionally, the update procedures for data structures should be scalable after migration.
- Support meshes generated on *non-manifold* models. In a non-manifold representation the surface area around a given point on a surface might not be flat in the sense that the neighborhood of the point need not be a simple two-dimensional disk [89]. Figure 2 shows examples of meshes on non-manifold geometric models. Just as the mesh data structures, the PMDB can handle the situations in which mesh entities attach to vertex contacts. This specifically requires the ability for such entities to be migrated with no loss of information, and that the vertex at the contact can be a shared partition boundary entity.

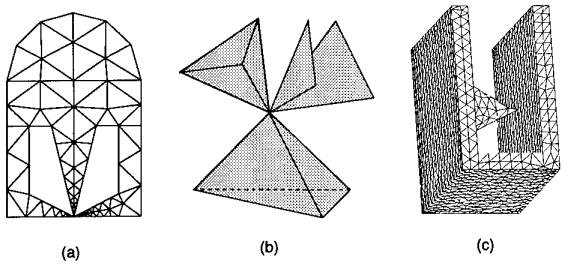


Figure 2. Example meshes handled by PMDB library

The early parallel and distributed memory implementations of finite element methods such as [51] involved static meshes and used the data parallel SIMD computing systems such as CM2. The ease of programming static and regular problems using the data parallel model led the compiler writers to incorporate this model in high performance Fortran compilers. The analysis for generating communication primitives for irregular references found in unstructured meshes could not be done at compile time. Therefore, runtime systems such as the PARTI primitives [63] were designed which would compute these references prior to entering a loop where the actual computations are done. If the distribution of the mesh changes, then all the references have to be recomputed. Since limited analysis can be done at the level of references only, the data parallel Fortran compilers soon proved to be weak for handling the dynamically changing mesh data structures of adaptive applications. This weakness has directed other researchers to design distributed mesh environments providing functionalities for refinement, coarsening, migration and load balancing.

A heuristic which has been the by product of high performance Fortran compilers is the *owner computes paradigm* [11][95]. This heuristic was used as a rule for letting the processor which owns a data item to perform the computations which define it. This paradigm is also used in other contexts such as parallel linear solvers provided by PetSc [31] which requires the designation of owners of the rows. A variation of this paradigm is used in implementing the current mesh migration algorithm.

Williams' Distributed Irregular Mesh Environment (DIME) project [90] can be considered as one of the earliest distributed unstructured mesh environments. This initial version was restricted to two dimensional meshes and could not handle non-manifold models and surface meshes such as a torus. The newer version DIME++ [93] implemented in C++ provides support for three dimensional elements.

DIME uses a hash table to implement *voxel databases* [92] which store a global key associated with an entity. This key is the geometric centroid of the entity. The coordinates of the centroid are converted to integer hash table index by dividing it with a user supplied tolerance. We show in sections 2.2.2 and 2.2.3 that explicit generation of global key by computing and storing the centroid is not necessary. When elements are migrated in DIME, new voxel entries are packaged into a message and the message is passed from processor to processor in a ring until each has seen the message. Each processor takes the voxel entry and checks if a match is found in the hash table. If found, then this implies that the entity is shared and the off-processor address is stored. Note that Williams uses the notion of *secretary* points which correspond to the owner of shared entities in PMDB. Even though the secretary points are used in computing the scalar products, they are not utilized in the implementation of an efficient update procedure after migration. Since the new voxels are passed in a ring of all processors, the update procedure has a fixed cost dependent on the number of processors.

Vidwans et al. [85] present a procedure to migrate tetrahedral elements between face adjacent and *sender-receiver-disjoint* processors. The sender-receiver-disjoint requirement necessitates processors involved in migration to be paired as either a sender or a receiver. This pairing process is carried out as part of their divide and conquer dynamic load balancing algorithm. Since a face can be shared by no more than two processors and a processor migrates to its face-adjacent processor, the shared face identification is readily available. Hence Vidwans et al. does not need use global identification numbers. A disadvantage of sender-receiver disjoint migration is that elements cannot be piped by a receiver processor to other processors in the same cycle of migration. This can lead to memory problems whereby a receiving processor obtains a large number of elements and has to store them before it can pass them onto other processors.

The `Tiling` system developed by Devine [18] is the first distributed environment to support hp-adaptive analysis and provides migration routines for regularly structured two dimensional meshes which can be hierarchically refined. Each tiling element stores pointers to neighboring four elements with partition boundary elements pointing to a ghost-element data which acts as a buffer during communication. The elements are assigned a unique id at the beginning and after refinements. The elements with unique ids are maintained in a balanced AVL tree [68] to allow efficient insertion and deletion during migration. The `Tiling` system supports only rectangular elements as the basic entity and the notion of shared entities like edges is implicit.

2.2.2 Distributed Mesh Model and Notation Used

The distributed mesh is viewed analogous to the modeling of non-manifold geometric objects. Figure 3 shows the hierarchical classification of the global mesh entities ${}_mT_i^d$, the processor model entities ${}_pT_i^{d'}$ and geometric model entities ${}_gT_i^{d''}$. Given the set of mesh entities $\{{}_mT\}$, a partitioning at the d_m dimension level divides the mesh into n_p parts, ${}_pT_{p_k}^{d_m}$, each of which is assigned to a processor with id $p_k = 0, \ldots, n_p - 1$. As a result of partitioning, some of the entities with dimension $d < d_m$ will be *shared* by more than one processor. The d_m-dimensional entity will be held by only one processor. Hence in general, partitioning with $d_m > 0$ defines a one-to-many relation from a mesh entity ${}_mT_i^d$ to its uses ${}_m^kT_i^d$ where $k \leq min(\Delta({}_mT_i^d), n_p)$. Here Δ defines the degree of an entity, i.e. given the dimension d of an entity, Δ is the number of $d+1$ dimensional entities which use it.

Since the procedures in a distributed memory environment operate on private local processor address space, we refer to each entity use ${}_m^kT_i^d$ in the global model as ${}_m^{(p_k,a_k)}T_i^d$ or in shorthand notation (p_k, a_k). The tuple (p_k, a_k) stands for the use of an entity by processor p_k at local address a_k. In the algorithm descriptions presented later this tuple is also called a *link* particularly if it is stored on a different processor than p_k.

For the implementation of owner computes paradigm, one of the processors holding a given entity ${}_mT_i^d$ is designated as the owner of that entity. In the distributed processor address space, we distinguish the owned entities as (p_o, a_o). Therefore, a partitioning in this case defines a one-to-one and onto mapping of global mesh entities onto the owned distributed mesh entities: Note that the inverse of this mapping exists and hence the pair (p_o, a_o) can serve as a global key of a distributed entity.

The uses of the shared entities are mapped onto the owner entity by a many-to-one relation :

$$\Phi : (p_k, a_k) \mapsto (p_o, a_o) \tag{12}$$

Figure 3 shows the relationship between the geometric model entities ${}_gT_i^{d''}$, the global mesh entities ${}_mT_i^d$ and the processor model. Given the uses (p_k, a_k) of an entity distributed over processors p_k, an agreement can be reached among these processors on whether they hold the identical entity by computing the ownership using the function Φ.

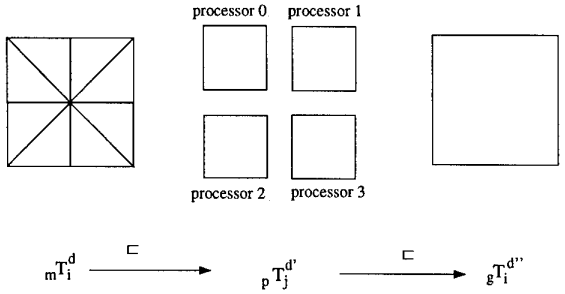


Figure 3. The relationship between the mesh model, processor model and the geometric model

2.2.3 Data Structures

PMDB data structures were designed to provide full variety of adjacency information. At the micro level of a partition boundary entity, one should be able to get all the uses or links of an entity on other processors. Each partition boundary entity stores all the uses on other processors as a linked list. This is shown in Figure 4. Note that one of the processors holding a shared entity is marked as an owner of that entity. The bold edges and vertices indicate the owners of the shared entities. This ownership information can be used in the implementation of the owner computes rule, for example, during link updates in mesh migration or scalar product computation in an iterative linear solver.

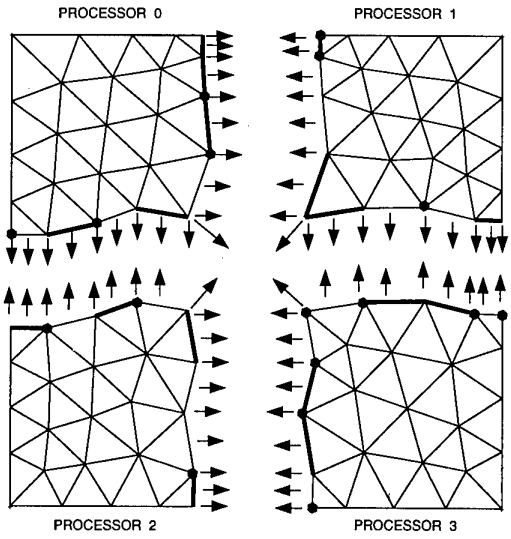


Figure 4. PMDB inter processor links and entity ownership

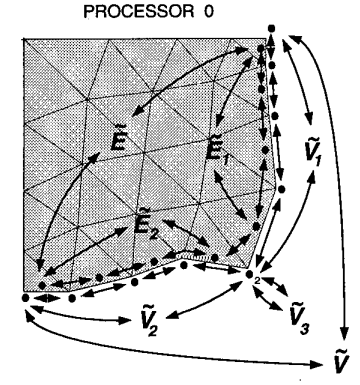
Since each processor stores the uses (p_k, a_k) on all the processors that hold a shared entity, the ownership can be computed as a function of these uses. An example of an ownership function Φ given in equation 12 is to choose the processor which has the tuple (p_k, a_k)

as the minimum. The other alternative is to let the owner regenerate the ownership. Whereas the former method can be done locally, the latter method needs communication of ownership information from the owner to the holders.

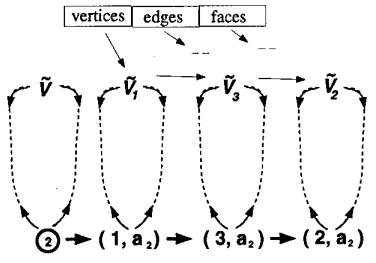
Note that the ownership information provides a global key for identifying an entity uniquely over all processors. Since the pair (p_o, a_o) *is the global key*, there is no need to generate and store a separate key as Williams [92] does by computing the centroid of the entity. On a processor, at the level of entities, the sets of entities that are on the partition boundary or adjacent to a specific processor are organized in doubly linked lists which provide constant insertion and deletion. Figure 5(a) shows the organization of the partition boundary entities. The lists can be traversed to get partition boundary entities shared among processors. For example, the set of all partition boundary vertices given by;

$$\tilde{V} = \left\{ {}_mT_i^0 : \exists {}_pT_j^d \; s.t. \; {}_mT_i^0 \sqsubset {}_pT_j^d \wedge \right. \\ \left. 0 \le d \le 1 \wedge \left| {}_pT_j^d \{ {}_pT_k^2 \} \right| > 1 \right\} \tag{13}$$

can be enumerated by the data structure. The set of all partition boundary edges $\tilde{E}$ which can be similarly defined, is also readily available.



(a)



(b)

Figure 5. Doubly linked structures of partition boundary entities ; global view (a) and partition boundary entity view (b)

In addition, adjacent processors based on various entity connectivity as well as the number of entities adjacent to the processor are maintained by storing this information in a linked list. Figure 5(b) shows the structure of the vertex adjacent processors and the doubly linked lists attached to it.

The list of partition boundary vertices $\tilde{V}_{p_k}$ adjacent to a particular processor p_k can be given by:

$$\tilde{V}_{p_k} = \left\{ {}_mT_i^0 : \exists {}_pT_j^d \; s.t. \; {}_mT_i^0 \sqsubset {}_pT_j^d \wedge \right. \\ \left. 0 \le d \le 1 \wedge {}_pT_{p_k}^2 \subseteq {}_pT_j^d \{ {}_pT_k^2 \} \right\} \tag{14}$$

which is directly accessible from the data structures.

2.2.4 Mesh Migration

Analogous to the owner computes rule, the mesh migration procedure of PMDB uses an *owner updates rule* to collect and update any changes to the links on partition boundaries after moving entities among processors. The migration of a set of mesh entities from a given processor to destination processors proceeds in three stages. Firstly, sender processors migrate the mesh entities to receiver processors. Secondly, the senders and receiver processors report the deletions or new addresses of migrated mesh entities to owner processors. In the last stage, the owner processors inform the affected processors about the updates in links. The processing which is done in the first stage is proportional to the number of mesh entities being migrated, whereas in the second and third stages, it is proportional to union of boundary of the migrated mesh entities. The migration procedure is given in Figure 6 and the detailed steps of the algorithm are explained below:

Senders to Receivers: These steps are responsible for sending the raw mesh data from the sender to the receiver processors. The mesh entities in $\{ {}_mT_{s_i}^{d_i} \}$ to be sent are packed into messages together with the data attached to the entities. The entities on the union of the boundary of the migrated mesh entities are also found, since any possible link updates will be limited to these.

In the case of all tetrahedral mesh in 3D space, the migrated boundary is given by faces which have exactly one migrated region targeted to the same processor attached to it. This applies to two dimensional meshes also with the migrated boundary enclosed by edges having exactly one face on its side which is being migrated to the same processor. Finding the migrated boundary for three dimensional meshes which contain both tetrahedral and dangling faces as shown in Figure 2(b) requires additional work. In this case if dangling faces are being migrated then migrated boundary cannot be derived by just checking the edges in the manner that is done for 2D meshes. Additionally, the vertices must be checked to see if they are used by any edge which is not being migrated.

The migrated internal entities can be deleted immediately since they cannot be referred to again by any processor. The migrated boundary entities cannot be deleted immediately, since if they happen to be owned by the processor, they will act as a fixed point where all the shared entity uses will be collected later.

procedure mesh_migrate (P_s,$\{{}_mT^{d_i}_{s_i}\}$,P_r,$\{{}_mT^{d_i}_{r_i}\}$)
input: P_s: destination processors.
$\{{}_mT^{d_i}_{s_i}\}$: sets of regions to be migrated
output: P_r: source processors
$\{{}_mT^{d_i}_{r_i}\}$: sets of regions received
begin
/* 1. senders and receivers to owners */
1 Pack the mesh $\{{}_mT^{d_i}_{s_i}\}$ to be sent.
2 Find the migrated boundary.
3 Delete migrated internal entities
4 Pack the owners' uses corresponding to migrated boundary
5 Send packed submeshes and uses to P_{s_i}
6 Receive packed submeshes and uses from P_{r_i}
7 Unpack the submeshes to get $\{{}_mT^{d_i}_{r_i}\}$

/* 2. senders and receivers to owners */
8 Establish usage of both sent and received migrated boundary entities.
9 Pack local uses of migrated boundary and owners uses to be sent to owner processors P_o
10 Send packed local and owner uses to owner processors.
11 Receive packed uses from senders and receivers.

/* 3 owners to affected */
12 Owners update use lists by inserting/deleting received local uses into/from use lists pointed to by owner uses and generate new ownerships.
13 Pack updated uses list of entities to be sent to affected processors P_a.
14 Send updated use lists and ownership to owner processors.
15 Receive updated uses list and ownership from owner processors.
16 P_a update use lists and ownership.
17 Delete unused sent migrated boundary entities.
end

Figure 6. Mesh Migration Algorithm

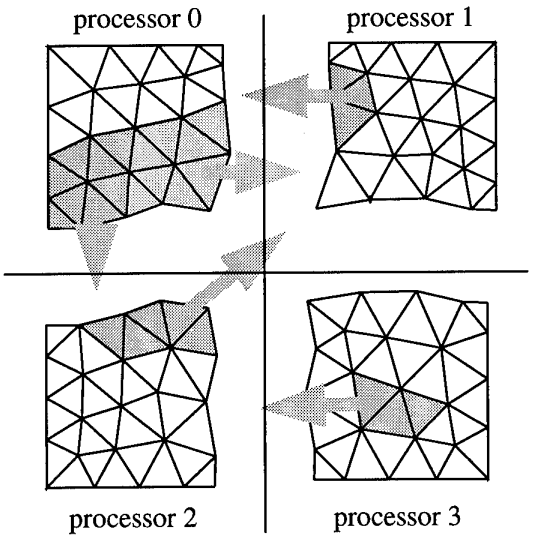
Once the packed submesh has been received, the processors unpack it and insert it into the mesh $p_k\{{}_mT\}$ held by the processor p_k. It is also possible that when more than one submesh arrives from different processors, they all might share some common entities. Figure 7 shows an example of such a case. As shown processors 0 and 2 both migrate to processor 1. Among the migrated entities are those which are shared by both 0 and 2. In such a case, these commonly shared entities, once unpacked, should not be unpacked for the subsequent received submeshes which also contain them and comes from a different processor. This process is achieved by inserting the unpacked migrated boundary entities into a red-black tree [68] which has guaranteed logarithmic access for each inserted entity. A key is needed to represent the entity in the red-black tree. This key can be either a global key or the readily available (p_o, a_o) tuple which was discussed earlier. Currently, PMDB version 3.1 by default generates global numbers after mesh is refined. The global numbers can be used for debugging and also provides a readily available equation number for linear equation solvers which assemble the global matrix. A future version of PMDB will make the global number generation optional in order to save memory for applications which do not need it.

Senders and Receivers to Owners: These steps operate only on the sent and received migrated boundary entities. These entities are tested to see if they are used by ${}_pT$ on processor p. Determining the use on processor p of a $d-$dimensional entity requires determining if that entity is part of the boundary of a $d+1$ dimensional entity on processor p. The entity hierarchy data structures of SCOREC mesh database readily provide this d to $d+1$ dimensional entity adjacency relationship. If the entity is used, its use (p, a) is packed and identified by the (p_o, a_o) use to be sent to owner processor. If the entity is not used $(p, null)$ is packed. Once packed, this information is sent to the owner processors. The overall complexity of these steps is proportional to the size of the sent and received migrated boundaries.

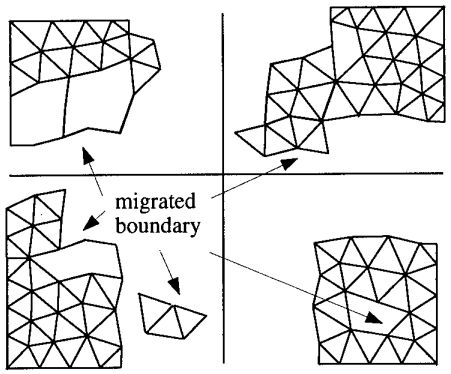
Owners to Affected Processors: Owners receive updates targeting a particular entity (p_o, a_o) it owns. If a use (p, a) is received, it is inserted in the list of uses of the shared entity at address a_o. If $(p, null)$ is received, the use (p, a) is deleted from the list of uses at address a_o. Once all the updates are completed, the ownership of these entities are regenerated. The updated links are then packed and sent to the affected processors. The affected processors receive these uses and update the corresponding local shared entities' list of uses. At this point, the migrated boundary entities can be deleted and mesh migration completes.

Computing Number of Receives: The steps $5-6, 10-11$ and $14-15$ implement non-blocking sends and receives. Each processor needs to know how many messages are being sent to it by other processors so that it can post a corresponding number of receive statements. A simple way to compute the number of receives is by first having each processor initialize a vector r of length n_p and to set r_p to 1 if a message will be sent to processor p and 0 otherwise. A follow-up sum scan operation can then be executed by all the processors resulting in each location r_p containing the number of receives. This procedure has $O(n_p log n_p)$ run time complexity and requires a message of length n_p to be communicated during the combine operation. Whereas this scheme will be efficient for small n_p, it is nevertheless non-scalable. The DIME environment, for example, makes use of the crystal_router [24] which provides a scheme for this problem by utilizing $log(n_p)$ message exchanges across the dimensions of the hypercube multiprocessors.

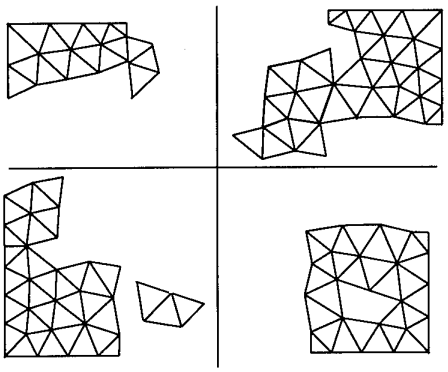
Considering the fact that each processor p usually sends to a small number s_p of processors, a scalable strategy is desirable for large n_p. We can make this scheme scalable by making use of the radix sort routine [7]. Since the processor ids are in the range $0, \ldots, n_p - 1$,



(a)



(b)



(c)

Figure 7. Example showing steps of mesh migration

this problem can be solved by sorting $T_s = \sum_{p=0}^{n_p-1} s_p$ keys each of length $log(n_p)$ bits. Before applying the radix sort, the keys are balanced by moving them such that each processor has T_s/n_p[4]. The balancing can be done, by first scan summing s_p to get the total number of messages and assigning a rank to each send. The target processor and the location on that processor can then be computed knowing the rank of the send and the total number of sends. The complexity of such a procedure will be $O(max_p\{s_p\}+(T_s/n_p)\cdot log^2 n_p)$ which is polylogarithmic in n_p when s_p is small.

We illustrate this with an example when $s_p = 1$. Consider the following scenario with the intended sends in the first row of Table 1. x is assigned the value 8 which equals n_p and indicates the processors which will not send any messages to anyone.

	0	1	2	3	4	5	6	7
send to processor	x	x	7	4	x	x	0	4
sort	0	4	4	7	x	x	x	x
mark end	1	0	1	1				
segment sum	1	1	2	1				
number of recvs	1		2	1				
ranges of no-recvs	[1,3]		[5,6]					

Table 1 Example showing computation of number of receives

After sorting the sends, the duplicate sends are now contiguous. We can mark the end of duplicate sends by communicating and comparing sends with the right neighbor processor. A *segment-sum* [7] can then be performed to count the number of duplicates. The processor marked at the end of the segment has the number of receives for the corresponding target processor. Note, however, that even though we can now inform the receiving processors of the number of messages, we still need to make sure that each processor posting a receive will be matched by a corresponding send by some processor. This can be done by letting all processors post receive messages which can be satisfied by guaranteed sends. As a result, we need to send messages to non-receivers that they will be receiving no messages. The last row computes the ranges of ids of non-receiving processors which can be computed again by a neighbor communication. Given this contiguous range, the processors can be sent a message by parallel recursive bisection strategy in logarithmic time. At the end, all processors will know how many messages they will be receiving.

In the above scheme, the complexity is dominated by the sorting scheme. A more elaborate scheme in the reference [41] provides radix sorting in $log(n_p)$. We also remark that currently PMDB uses the simple $O(n_p log n_p)$ procedure since the largest number of processors used is 64, a number too small to make the scalable version worthwhile to use.

[4] assuming wlog that T_s is divisible by n_p

2.2.5 Scalability of Mesh Migration and Extensions

In the mesh migration procedure presented above, the amount of communication involved is proportional to the volume of submeshes in the first stage of the algorithm and to the surface of submeshes during link updates in the second and third stages. As a result, if each processor migrates to a small number of processors, such as its neighbors, then we expect that the migration will scale as the number of processors is increased. Various tests have been performed to demonstrate scalability of migration. The data involving the maximum number of regions migrated by a processor, the total number of regions migrated by all processors, the time taken, and the throughput, that is, the number of regions sent by a single processor per second are plotted against the number of processors used.

Test 1: In the first test, we let each processor exchange a slice on its partition boundary with its neighbors. This test is a realistic representative of the migration patterns that occur in iterative dynamic load balancers since regions near partition boundaries are migrated in clusters to the neighborhood of a heavily loaded processor. Another application that performs this kind of migration is mesh coarsening [10]. Figure 8 shows the example mesh that was used before (a) and after migration (b). Figure 9(a) plots the maximum number of regions sent by a processor and (b) shows the wall time taken. From these plots, we see that execution time is proportional to the number of regions sent irrespective of the number of processors.

Figure 10 on the other hand plots the total number of regions sent by all processors. As the number of processors are increased the total number of regions at partition boundaries increases. Hence even though overall more regions have been moved, the time is proportional to the maximum sent by a single processor. This behavior demonstrates that when processors migrate to a small number of neighbors, the migration procedure scales well. Figure 10(b) plots the throughput attained.

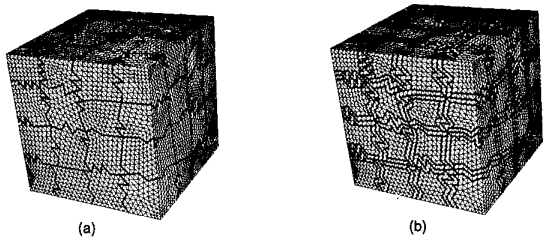


Figure 8. Neighborhood migration test; before boundary exchange (a), after boundary exchange (b)

Test 2: In the second test, we let each processor hold 2500 regions corresponding to a partition of the box mesh and migrate all its regions randomly targeted to s processors with $s = 1, \ldots, 2^i, \ldots, n_p - 1$. The plots of time taken for migration and the throughput per processor is shown in Figure 11. The plot in (a) shows that as the number of processors is increased, the time taken grows

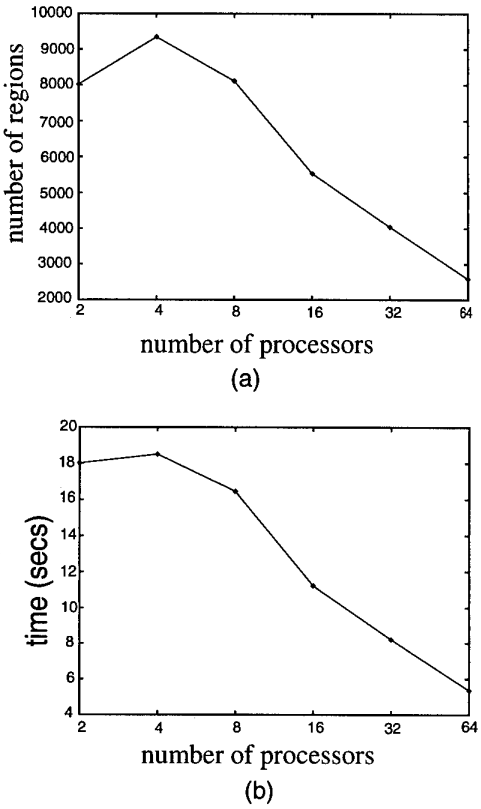


Figure 9. Neighborhood migration test for box ; maximum number of regions migrated by a processor (a), wall time (b)

slowly. In particular, if we look at the $s = 1$ case, we see a flat curve between 32 and 64 processors. The number of processors has been doubled, yet the execution time remains the same. As s is increased the execution time growth is larger as expected, since the number of total migrations is increased. In particular, if $s = n_p - 1$, we have all-to-all migration. Note that, there is a pronounced drop in the throughput as shown in Figure 11(b) between the cases $s = 1$ and 2. For example, with $n_p = 48$, the throughput is 519 regions for $s = 1$ and drops to 309 regions at $s = 2$. The major cause of this drop is not the mere increase in s, but rather the fact that when regions are assigned random destination, the union of the migrated boundary of the mesh entities being sent becomes proportional to the number of regions sent. In the case of $s = 1$, the migrated boundary is proportional to the surface of the mesh entities sent. As a result, since the cost of stages 2 and 3 of the mesh migration algorithm is dependent on the size of migrated boundary, these stages contribute greatly to the drop in cases $s > 1$. The sets of regions which are migrated in practice are clustered locally and hence the migrated boundary size is rarely proportional to the volume being sent. Therefore, higher throughput rates can be attained for larger s as is evident from Test 1 above.

This section discussed the data structures and the migration routines used in the PMDB library. PMDB library cur-

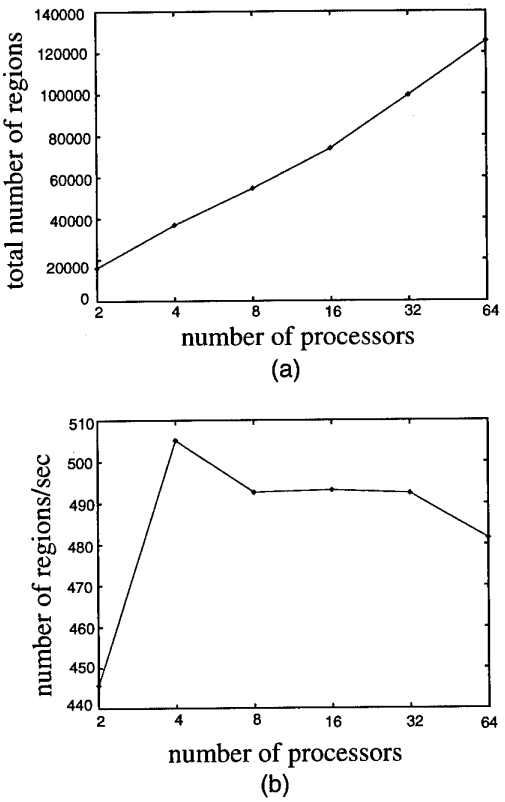


Figure 10. Neighborhood migration test for box mesh ; total number of regions migrated by all processors (a), throughput per processor (b)

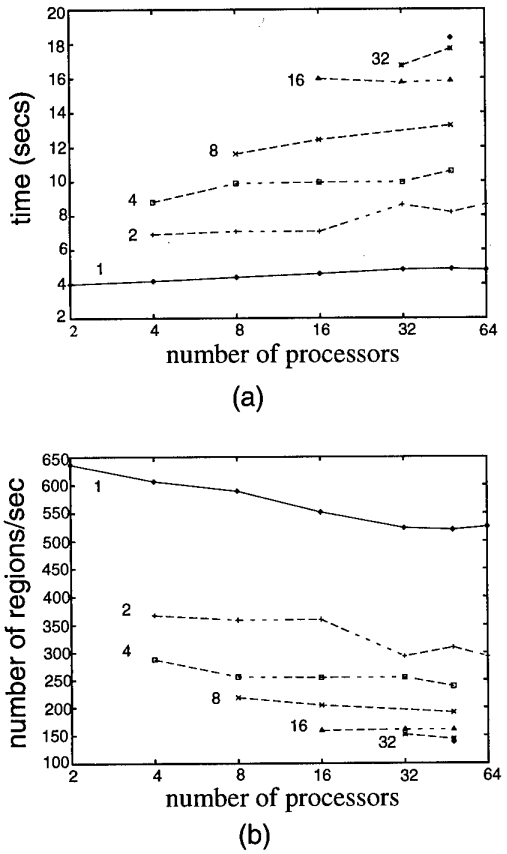


Figure 11. Migrating to s processor ; wall time in seconds (a), throughput per processor (b)

rently supports triangular and tetrahedral meshes. However, the data structures and the mesh migration procedures easily extend to other types of elements such as quads, bricks or mixed meshes. Further fine tunings are also possible which can reduce memory requirements and improve the throughput of the migration procedure by, for example, generating ownership corresponding to the target processor for entities on migrated model boundary.

2.3. Dynamic Load Balancing of Adaptively Evolving Meshes

The evolving nature of an adaptive discretization introduces load imbalance into the solution process. Therefore, it is critical that the load be dynamically rebalanced as the adaptive calculation proceeds. The current repertoire of partitioning and dynamic redistribution heuristics for unstructured meshes can be classified into three main categories given as follows:

The most popular category involves *Recursive Bisection (RB)* techniques which repeatedly split the mesh into two-submeshes. *Coordinate RB* methods bisect the elements by their spatial coordinates. If the axis of bisection is Cartesian, then it is called *Orthogonal RB* [4]. If the axes are chosen to be along the principal axis of the moment of inertia matrix, then it is called *Inertial RB*. *Spectral RB* is another method which utilizes the properties of the *Laplacian matrix* [22] of the mesh connectivity graph and bisects it according to the eigenvector corresponding to the second smallest eigenvalue of this matrix [56].

The least popular choice for partitioning meshes is the *probabilistic* methods which include simulated annealing and genetic algorithms. These methods require many iterations and are expensive to use as mesh partitioning methods [91].

Iterative Local Migration techniques have been the target of recent attention due to their potential for dynamically balancing adaptive meshes which change incrementally. These techniques exchange load between neighboring processors to improve the load balance and/or decrease the communication volume. The definition of processor neighborhood can either be the hardware link or the connectivity of the split domains. The *cyclic pairwise exchange* [33] pairs processors connected by a hardware link and exchanges the nodes of the mesh to improve the communication. Leiss/Reddy [43] on the other hand uses the hardware link as the neighborhood to transfer work from heavily loaded to less loaded processors. The `Tiling` system [18] uses and extends the Leiss/Reddy algorithm to the case where the neighborhood is defined by the connectivity of the split domains. The algorithm of Lohner et al. [50] exchanges elements between subdomains according to a deficit difference function which reflects the imbalance between an element and its neighbors. The procedure by Vidwans et al. [85] uses a divide

and conquer approach to pair processors and uses connectivity as well as coordinate information to decide which elements to migrate.

A disadvantage of the common implementations RB methods is they start with the entire mesh on a single processor and partition from there. Two problems with this approach in a parallel adaptive calculation are (i) the time required to gather the distributed mesh together on a single processor, and (ii) the fact that after the mesh has been adapted, it may have grown to the point that it can not fit on a single processor. These problems can be alleviated it the mesh remains distributed during the repartitioning process. The next subsection discusses a parallel implementation of *Inertial Recursive Bisection* that operates on a distributed mesh.

RB methods operate on the whole mesh and compute the direct destination for each element. Because of this, it is possible that RB methods may require complete remapping of the elements at the end. On the other hand, iterative local migration techniques propagate the excess load by local transfers to other processors. A disadvantage of iterative local migration techniques is that many iterations may be required to regain global balance and hence elements reach their final destination after many local transfers rather than directly. In particular, when elements are migrated, the full element data involving connectivity and local attached data are communicated. For parallel repartitioners based on coordinate bisection, only the centroids and region pointers need to be communicated during a parallel sorting phase. As a result this class of repartitioners may have better performance on machines in which the communication between any pair of processors is distance-independent.

Subsection 2.3.2 presents an iterative load balancing procedure based on the Leiss/Reddy heuristic of requesting load from the most heavily loaded neighbor. The performance of this procedure is compared with repartitioning by the parallel distributed inertia recursive bisection algorithm.

2.3.1 Geometry-Based Dynamic Balancing Procedures

Geometry-based dynamic balancing (or repartitioning) relies here on the Inertial Recursive Bisection (IRB) method [50] which is a variation of the more classic Orthogonal Recursive Bisection (ORB) [4]. ORB is a recursive process that bisects a set of entities by considering the median of the set of corresponding centroids with respect to a given coordinate axis. As ORB is recursively called, the choice of coordinate axis is circularly permuted (x,y,z,x, etc). Unlike ORB, IRB considers the inertial coordinate system (origin is at the center of gravity and the three axes are the principal axes of inertia) for the set of entities to be bisected. In three dimensions, the determination of the three principal axes of inertia is an eigenvalue problem of order 3. Once the inertial coordinate system is defined, the coordinates of the centroids are transformed and the cut is made at the median with respect to the first coordinate. This first coordinate is the "key" that the sorting algorithm described later in this section works on.

The main assumption for performing repartitioning in parallel is that the entities are distributed. It is also assumed that there is no reason for the number of entities stored on processor to be uniform across processors. The result of this repartitioning will be an equal number of entities per processor. It should be noted that, in this context, the goal of repartitioning is equivalent to the goal of dynamic load balancing [15, 55, 73, 54, 43, 85]. The key algorithm in IRB (and ORB) is the determination of the median for a given set of doubles (referred to as "keys") [68]. With respect to this paper, the "keys" are the first coordinates, in the inertial frame, of the entities to be bisected. The method used here is to sort the "keys" and then pick the entry at the middle of the sorted list. In this case, efficiently performing IRB in parallel can be reduced to efficiently sorting in parallel [34]. From the conclusions of the paper by Blelloch et al. [8] which compares different parallel sorting algorithms (Batcher's bitonic sort, radix sort, and sample sort), it appears that the sample sort algorithm is the fastest of the three for large data sets. Therefore, a parallel sample sort algorithm has been implemented in order to efficiently support IRB.

Given a set of n "keys" distributed on p processors ($n >> p$), a sample sort algorithm consists of three main steps:

1. $p-1$ splitters (or pivots) are chosen among the n "keys"
2. Each key is routed to the processor corresponding to the bucket the "key" is in
3. Keys are sorted within each bucket (no communication)

The goal of step 1 is to split the set of "keys" into p parts (buckets) as evenly as possible and as efficiently as possible. The $p-1$ splitters which are implicitly sorted (say with respect to increasing value) are labeled from 1 to $p-1$. All distributed "keys" below splitter 1 belong to bucket 0, all distributed "keys" between splitter i ($0 < i < p-1$) and splitter $i+1$ belong to bucket i, and all distributed "keys" above splitter $p-1$ belong to bucket $p-1$. Processor i ($0 \le i < p$) is responsible for the bucket labeled i. In step 2, assuming the $p-1$ splitters have been found and broadcasted to all processors, any distributed "key" can tell in which bucket it belongs and is rerouted to the processor that is responsible for that bucket. At this point, any processor has knowledge of all "keys" that belong to the bucket it has been assigned to. Step 3 can be performed using any efficient sequential sorting algorithm, like quicksort [68]. It is clear that the parallel efficiency of the sample sort algorithm depends on the sizes of the buckets. Parallel efficiency is maximal when the sizes of the buckets are near equal. A sampling method is used to obtain "good" splitters. Given the n input "keys", ps "keys" (s is an integer ≥ 1 called the over sampling ratio) are selected at random and sorted typically sequentially. The entries in the sorted list of ranks s, $2s$, ... , $(p-1)s$ are the $p-1$ splitters. The bound for bucket expansion (ratio of maximum bucket size to average) is

given in the paper by Blelloch et al [8]. In practice, the over sampling ratio should be such that the sorting to find the splitters (which is done serially) does not become a bottleneck for the global parallel sample sort algorithm. For the purpose of the presented repartitioning technique, the over sampling ratio is chosen such that *ps* is of the order of *n/p* (*n/p* being of the order of the number of "keys" to sort in step 3).

The following pseudo-code shows the process of repartitioning using IRB in parallel. It is assumed that the entities are already distributed on processors. A statement of the form **for** ($i = 0$; $i < n$; i++) { ... } indicates a loop that is executed as long as the loop variable i, initially set to 0 ($i = 0$) and incremented by 1 upon completion of each pass (i++), has a value less than n ($i < n$) [40]. Each processor executes the following pseudo-code (MIMD):

1. Associate each entity with a "key" structure consisting of:
   a. 3 doubles for the coordinates of the entity's centroid with respect to the current inertial coordinate system (initially with respect to original coordinate system)
   b. 1 integer that indicates on which processor the actual entity is stored
   c. 1 pointer to the entity
   d. 1 integer that indicates the destination processor for the entity
2. **for** (*step* = 0 ; *step* < $\log_2 p$; *step* ++) {
   a. Split the p processors into 2^{step} processor sets (each set is of cardinality $p' = p/2^{step}$)
   b. Balance the load such that each processor has approximately the same number of keys (reroute the keys accordingly)
   c. Get center of gravity, find the three principal axes of inertia, and apply transformation to the keys
   d. Get $p' - 1$ splitters among the keys
   e. Depending on the position with respect to the splitters, determine in which bucket (processor) each key goes (reroute the keys accordingly)
   f. Sort the keys (no communication)
   g. Depending on the position with respect to the median, determine in which bucket (processor) each key goes (reroute the keys accordingly)
   h. Free the processor sets

   }
3. The destination processor is set to the processor the key is currently in
4. Reroute all keys to the originating processors
5. Migrate entities according to the destination processor stored at the key level

Steps 2.b through 2.g are done independently on each processor set. Once all keys have been sorted in the processor set (at the end of step 2.f), the median (key that splits the set of keys into two subsets of same cardinality) is easily obtained. Any key that is before the median is placed (if not already there) on a processor with a low rank (0 to $p'/2 - 1$) and any key that is after the median is placed (if not already there) on a processor with a high rank ($p'/2$ to $p' - 1$). This guarantees that any key stored on a processor set is smaller that any key in the next processor set. Figure 12 is a graphical depiction of steps 2.b through 2.g in the case when p' equals two. At each step, the array of keys (distributed across the processors in the set) is represented by a horizontal line which is cut to show how it is currently distributed. The symbol < indicates that the keys in the array are not sorted if above the processor cutter, it also indicates that any key in the left processor's array is smaller than any key in the right processor's array. If there is no such symbol, the keys are not sorted yet.

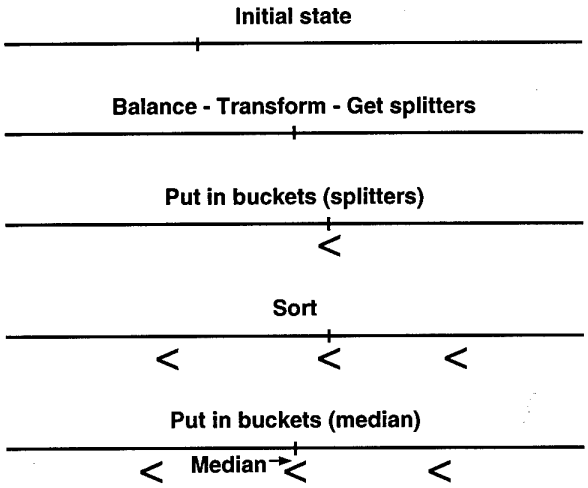


Figure 12. Graphical description of the repartitioning algorithm (2–processor set)

Figure 13 shows a randomly distributed mesh (approximately 35,000 elements) and the resulting dynamically repartitioned mesh for eight processors. Figure 14 shows timings (wall-clock seconds on IBM sp-2) for that particular mesh on 2, 4, 8, and 16 processors. The processor assignment timing corresponds to steps 1 to 4 (decision making). The migration timing corresponds to step 5. It should be noted that a randomized mesh as the initial state is a worst-case scenario for the migration part of the repartitioning procedure. Past four processors, the time spent decreases as the number of processors increases, which is a good indication of scalability. It is conjectured that the "abnormal" speed with two processors is due to the fact that (i) the only processor set ever used is the full set of processors and (ii) there is some performance degradation when more than one processor set is defined.

2.3.2 Topologically-Based Dynamic Balancing Procedures

Tree Based Load Balancing Algorithm The `Tiling` system which uses the Leiss/Reddy approach calculates the load averages utilizing the immediate neighborhood.

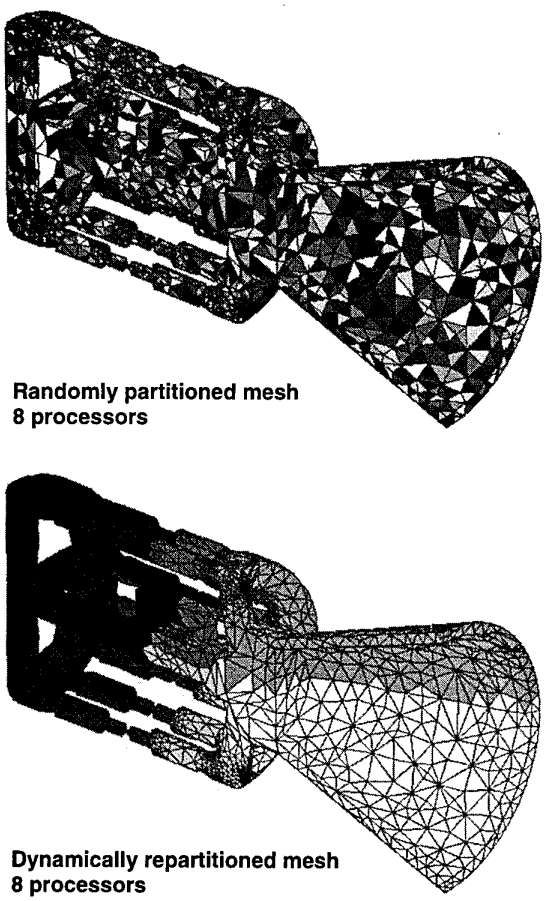


Figure 13. Dynamic repartitioning on a randomly distributed mesh

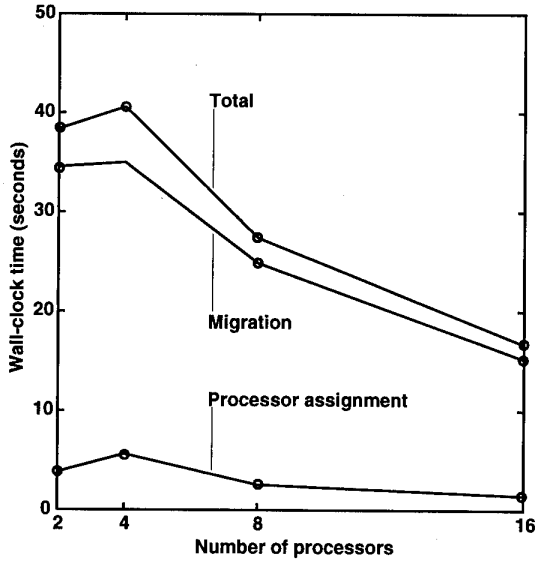


Figure 14. Timings for dynamic repartitioning

To incorporate more global information and to direct load transfers, we view the processor requests for load from heavily loaded processors as forming a forest of trees.

Figure 15(a) shows an example of requests that can be formed. Given this hierarchical arrangement of processors as the nodes of trees, we balance the trees as shown in Figure 15(b) and iteratively repeat the process until the load distribution converges to optimal load balance within a user supplied tolerance. The full algorithm is given in Figure 16. The procedure details are given as follows.

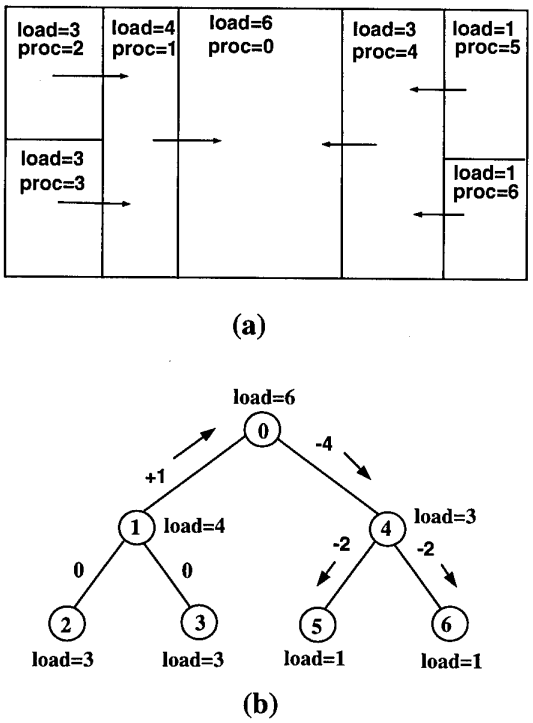


Figure 15. Load balancing example; load request (a) load migration on the tree (b)

procedure `tree_load_balance`(tol_{load}, max_{iter})
in tol_{load}imbalance load tolerance
in max_{iter} : maximum number of iterations
begin
1 iter = 0
2 **while** (max. load difference > tol_{load}) **and** (iter < max_{iter}) **do**
3 iter = iter + 1
4 Compute neighboring load differences.
5 Request load from neighbor processor having largest load difference (creates processor trees).
6 Linearize processor trees.
7 Compute amounts of load migration.
8 Select and migrate load.
9 **endwhile**
end

Figure 16. Tree based dynamic load balancing procedure

Steps of the procedure The steps of balancing the forest of trees are repeated until convergence is achieved. Assuming that load transfer occurs when there is a load difference of at least two units, Leiss/Reddy's algorithm

has worst case imbalance of $d/2$ where d is the diameter of the network. In such a converged state, all the processors have a load difference of one with its neighbors. This configuration forms a staircase load distribution. For our `tree_load_balance`, a staircase will not be the worst case distribution, but rather a forest of trees each of which has a maximum height of two and load difference of one. In such a state, the worst case imbalance will be $d/4$. This kind of imbalance can be tolerated on a coarse grain machine. For example, a 100K mesh on 64 processors will imply a worst case of 1.02% imbalance.

In step 4, load differences are computed by having each processor send its load value to its neighbors and correspondingly receive load values from its neighbors.

Step 5 invokes the Leiss/Reddy load request process. Since each processor can receive requests from multiple processors, but can only request from a single processor, a forest of trees is formed.

In step 6, the trees are linearized for efficient scan operations. One possible linearization is given by Euler Tour [34]. This however requires $2(|T|-1)$ links where $|T|$ denotes the number of vertices on a tree. We use the depth-first-links [41][83] which use between $|T|$ and $2(|T|-1)$ number of links.

Step 7 computes the amounts of load migrations on the tree using logarithmic scan operations on the linearized tree. Let $load_mig_i$ denote amount of load that will be migrated into or out of a tree node i which represents a processor. Let also T_i denote the subtree with node i as the root of the subtree and $load(T_i)$ be the sum of loads of nodes in this subtree. The amount of load migration is then calculated as

$$load_mig_i = load(T_i) - avg_load(T) * |T_i|$$

with $avg_load(T) = load(T)/|T|$ representing the average load on the tree when balanced. Given $load_mig_i$, the direction of load migrations can be found as

$$\begin{aligned} load_mig_i &= 0, \text{ do nothing with parent,} \\ &< 0, \text{ get load from parent,} \\ &> 0, \text{ send load to parent.} \end{aligned}$$

Having calculated the directions of load migration, step 8 migrates the elements on the partition boundary in a slice by slice manner until $load_mig_i$ of them has been transferred. Each slice of elements forms a peeling of the partition boundary and are selected by choosing elements which touch the boundary by any one of their vertices. Figure 17 shows the element selection criteria for migration.

Examples In this section we plot various statistics that show the performance of the load balancing on refined meshes and compare them with coordinate based repartitioning.

Test 1: In this test a `patella` mesh is refined manually in the center of the model. At the beginning, the mesh has 8497 tetrahedrons. After refinement, the number of tetrahedrons increases to 15329. In Figure 4, we plot the convergence history of the `tree_load_balance` for the 16 processor run. The scheme converges in 3 iterations.

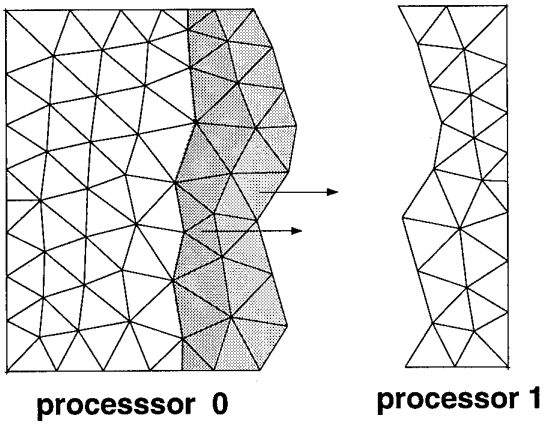


Figure 17. Selecting elements for migration

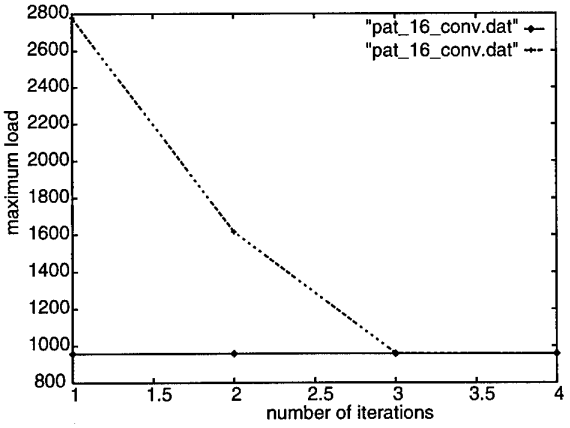


Figure 18. The convergence history for load balance

In Figure 19, we plot the times taken for the `tree_load_balance`, the moment of inertia parallel recursive bisection routine and the parallel inertia repartitioner. The parallel recursive bisection starts with the whole mesh on one processor and recursively splits it in parallel. The repartitioner employs a parallel sort routine. The parallel repartitioner outperforms the other two strategies. There are various reasons why the inertia repartitioner outperforms the load balancer. Firstly, the performance of the load balancer is greatly affected by the distances between the heavily loaded and underloaded processors. For example, suppose there is imbalance due to refinement at the corner of a model on one of the processors. In such a case, to propagate the excess load to the rest of the processors by local neighborhood transfers, one can see that the number of steps needed will be at least the size of the diameter of partition graph. This is because the load will be transferred one step at a time under the iterative load balancer. For other types of distributions in which the distance between the lightly loaded to heavily loaded processors is small and there is a

high frequency of load imbalances, the load balancer will have better performance. The repartitioner bypasses the effects of distance by directly sending load from heavily loaded to lightly loaded processors. On an architecture such as the IBM-SP2, in which communication cost is independent of the distance between the processors and hence the same between any pair of processors, the repartitioner will be advantageous since it directly sends the load to its final destination. The load balancer will be disadvantageous since it will incur expensive latency cost during many local transfers it performs.

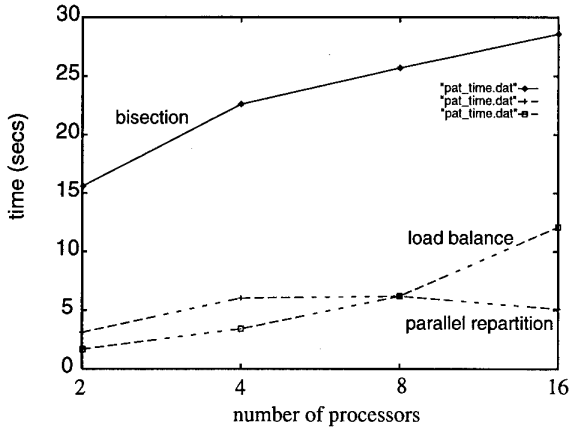


Figure 19. Time taken for load balance, parallel repartitioning and bisection

Finally, Figure 20 shows the quality of the partitions produced in terms of maximum and total percentage of faces cut. The load balancer's element selection criteria for migration dictates the quality of the partitions. The criteria currently used can be improved by incorporating coordinate information to selection decision.

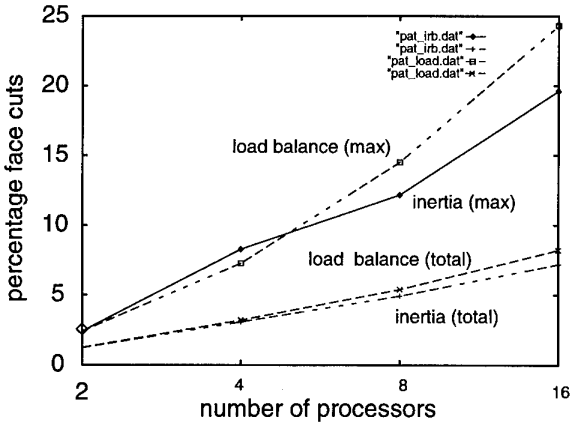


Figure 20. The total and maximum number of cut faces for each method

Test 2: In this test, various statistics are reported for the adaptively refined `onera-m6` wing mesh during an actual CFD analysis on 32 processors. At the beginning the mesh has 85567 tetrahedrons. Three stages of adaptive refinements are performed during which the number of tetrahedrons increase to 131000, 223501 and finally to 388837. Figure 21 shows the convergence history of the iterative load balancer. In all cases of load balancing after refinement, the imbalance reduces to less than 4% during the first 8 iterations and takes far more number of iterations to reduce this imbalance further to 0% imbalance. One need not run the `tree_balance` to full convergence. It can be stopped when a reasonable imbalance is achieved.

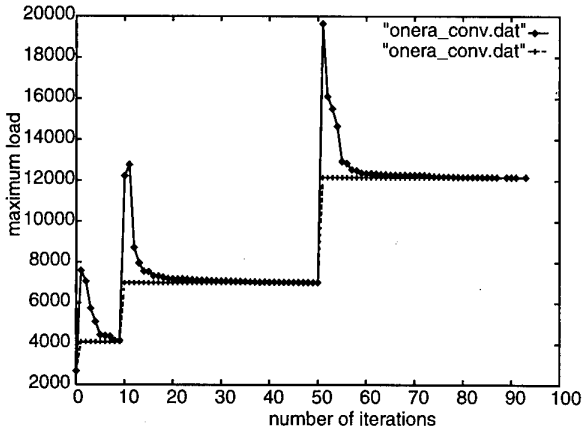


Figure 21. The convergence history for load balance during three stages of refinement

Table 2 shows the execution time comparisons between the `tree_load_balance` and the parallel moment of inertia partitioner. In all cases, moment of inertia outperforms `tree_load_balance` for the same reasons which was explained in Test 1.

refinement	1st		2nd		3rd	
percent imbalance	1.7	0	3.8	0	1.9	0
tree_balance (sec)	73	85	127	210	189	283
inertia partition (sec)		21		48		128

Table 2 Execution times (in seconds) for `tree_load_ balance` and inertia partitioner

Finally, Table 3 shows partition quality comparisons between the `tree_load_balance` and the moment of inertia partitioner. The percentage of maximum number and the total number of cut faces are given for both `tree_load_balance` and the inertia partitioner.

refinement	1st		2nd		3rd	
percent cuts	max	total	max	total	max	total
tree_balance	24	10	26	11	25	10
inertia partition	26	7	25	7	19	6

Table 3 Maximum and total percentage of cut faces for `tree_load_balance` and inertia partitioner

3. Parallel Automatic Mesh Generation

3.1. Introduction

The development of automatic mesh generation techniques for complex three-dimensional configurations has been an active area of research for over a decade [26, 78]. The introduction of these mesh generation procedures has removed a major bottleneck in the application of finite element and finite volume analysis techniques. The introduction of scalable parallel computers is allowing the solution of ever larger models. It is now common to see meshes of several million elements solved on these computers, with the ability to solve on meshes of hundreds of millions of elements coming in the near future. As mesh sizes become this large, the process of mesh generating on a serial computer becomes problematic both in terms of time and storage. Therefore, parallel mesh generation procedures that operate on the same computer, and using similar structures, as the parallel analysis procedures must be developed

With recent advances in the efficiency of automatic mesh generators which create well over two million elements per hour on a workstation [88], one may question the need for the parallel generation of meshes. The obvious answer is that as the problem size grows, the solution process on parallel computers will continue to scale by the addition of more processors. However, mesh generation on a single processor will not scale, therefore becoming the computational bottleneck. A second critical reason for parallel mesh generation is the shortage of memory on a sequential machine when dealing with very large meshes. On a parallel machine, the memory problem is addressed by distributing the mesh over a number of processors, each of which stores its own portion of the mesh.

Efficient parallel algorithms require a balance of work load among the processors while maintaining interprocessor communication at a minimum. Key to determining and distributing the work load and controlling communications is knowledge of the structure of the calculations and communications. In the finite element analysis process, the mesh and its connectivity naturally provide the required structure. The ability to maintain efficiency is compromised when the structure and, therefore, work load and communications is altered as is the case in parallel adaptive finite element analysis [15, 55, 73, 54]. Parallel mesh generation is even more complex to effectively control since the only structure known at the start of the process is that of the geometric model which has no discernible relationship to the work load needed to generate the mesh. On the other hand, the more useful structure to discern work load and control communications is the mesh which is only fully known at the end of the process. The lack of initial structure and ability to accurately predict work load during the meshing process underlies the selection of algorithmic procedures in the parallel mesh generation procedure presented here. In particular, the procedure employs an octree decomposition of the domain to control the meshing process. The octree structure supports the distribution or redistribution of computational effort to processors.

3.2. Background and Meshing Approach

To date, there has been limited attention given to parallel automatic mesh generation algorithms. Löhner et al [49] have parallelized a two-dimensional advancing front procedure which starts from a pre-triangulated model boundary. The approach taken is to subdivide (partition) the domain (with the help of a background grid) and distribute the sub-domains to different processors for triangulation. The interior of subdomains are meshed independently. Then, the inter-subdomain regions are meshed using a coloring technique to avoid conflicts. Finally, the "corners" between more than two processors are meshed following the same basic strategy. A "one master-many slaves" paradigm has been chosen to drive the parallel procedures. This approach has been extended to three dimensions with some modifications [79]. A load balancing phase follows the initial domain splitting (at the background grid level). The interface gridding incorporates mechanisms (i) to avoid degradation of performance by using fine grain parallelism and (ii) to reduce the number of processors when there is too much communication overhead. Results show scalability of the method.

Saxena and Perruchio [64] describe a parallel Recursive Spatial Decomposition (RSD) scheme which discretizes the model into a set of octree cells. Interior and boundary cells are meshed by either using templates or element extraction (removal) schemes in parallel. The algorithmic procedure they employ to create these octant level meshes requires no communication between octants. The main difficulty for this meshing approach is to guarantee that a boundary octant can always be meshed regardless of the complexity of the model. Robust loop building algorithms which include possible tree refinement to resolve invalid configurations are in general difficult to parallelize [76]. Parallel results have been simulated on a sequential machine.

The parallel mesh generator presented here builds upon previous work on sequential octree-based mesh generators [66, 76, 77], parallel adaptive finite element analysis procedures [15, 55, 73], and parallel mesh generation [16]. It meshes three-dimensional non-manifold objects following the hierarchy of topological entities. That is, the model edges are meshed first, the model faces are meshed second, and the model regions are meshed last. The current discussion focuses on the octree-based region meshing procedure.

Figure 22 graphically depicts the basics of the present mesh generator. The first step in meshing a model region is to develop a variable level octree which reflects the mesh control information and is consistent with the triangulation on the boundary of the model region. Octants containing mesh entities classified on the boundary of the model region to be meshed are constructed to be approximately of the same size as the mesh entities they contain. A one level difference on octants sharing one

or more edges is enforced during this process to control smoothness of the mesh gradations. Once the octree is generated, the octants are classified as *interior*, *outside*, or *boundary*. Those classified as *outside* receive no further consideration. Some *interior* octants are reclassified *boundary* if they are too close to mesh entities classified on the boundary of the model region (*boundary-interior*). The purpose of this reclassification is to avoid the complexities caused when *interior* octant mesh entities (coming from the application of templates) are too close to the boundary and may lead to the creation of poorly shaped elements in that neighborhood. *Interior* octants are meshed using templates. Face removal procedures are then used to connect the boundary triangulation to the interior octants. Figure 23 graphically describes a face removal in a two-dimensional setting.

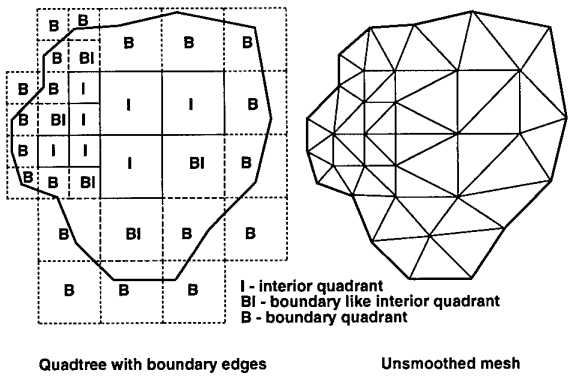


Figure 22. Graphical depiction of the basics of the presented mesh generator

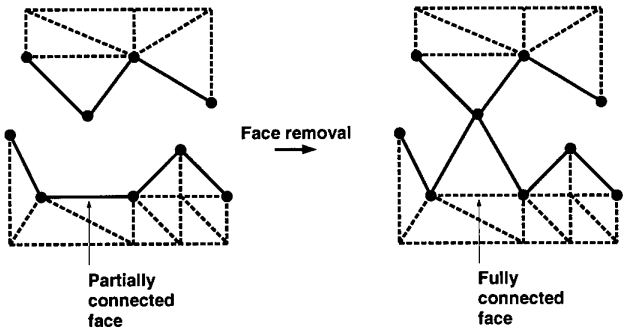


Figure 23. Face removal (2–D setting)

3.3. Sequential Region Meshing

As indicated above, the starting point for the region meshing process is a completely triangulated surface. The surface triangulation must satisfy the conditions of topological compatibility and geometric similarity [67] with respect to the model faces. The region meshing process consists of the three steps of (i) generation of the underlying octree, (ii) template meshing of interior octants, and (iii) face removal to connect the given surface triangulation to the *interior* octants.

Mesh faces to which tetrahedral elements will eventually be connected are referred to as partially connected faces. They are basically missing one connected tetrahedron in the manifold case, and one or two in non-manifold situations. Initially, the mesh faces classified on the model boundary are the partially connected mesh faces. Once templates have been applied, that is, at the start of face removal, the interior mesh faces connected to exactly one tetrahedron are also partially connected mesh faces. In the remainder of this discussion, the current set of partially connected mesh faces will be referred to as the front. During face removal, tetrahedra are connected to these faces, therefore eliminating them. Any non-existing face of a newly created tetrahedra, referred to as a new face, is a partially connected face until it is eliminated. The face removal process is complete when there are no partially connected mesh faces remaining.

3.3.1 Underlying Octree

The octree is built over the given surface mesh to (i) help in localizing the mesh entities of interest, and (ii) provide support for the use of fast octant meshing templates. Proper localization is achieved by having each terminal octant reference any partially connected mesh face which is either totally or partially inside its volume. This information is used to efficiently guarantee the correctness of the face removal technique. The octree building process can be decomposed into: (i) root octant building, (ii) octree building, (iii) level adjustment, (iv) assignment of partially connected mesh faces to terminal octants, and (v) terminal octant classification.

The root octant is such that the given surface mesh is contained within it. It is cubic in order to avoid the creation of unnecessary stretched tetrahedra coming from the application of meshing templates on stretched octants (assuming isotropy is desirable in the resulting mesh).

The terminal octants are constructed to be approximately the same size as any partially connected mesh face associated with them in order to ensure appropriate element sizes and gradations. This is done by visiting each mesh vertex in the initial surface mesh, computing the average size of the connected mesh edges, and refining the octree until any terminal octant around that vertex is at a level corresponding to that average size. The level of the octant is given by:

$$octlev = \log_2 \left(\frac{rootlength}{size} \right) \qquad (16)$$

where $rootlength$ is the length of the root octant and $size$ is the size of the mesh entity (defined here as the average length of the bounding edges). It should be noted that this procedure does not theoretically ensure a match in size between every terminal octant and the partially connected mesh faces it knows about.

To ensure a smooth gradation between octant levels, no more than one level of difference is allowed between terminal octants that share an octant edge. Application of this rule can possibly lead to refinement of some

terminal octants past the level that was set by the partially connected mesh faces in their volumes.

Once the tree is completed, partially connected mesh faces are assigned to terminal octants. Given a mesh face, terminal octants that should know about it can be separated into two groups: (i) those that are in the path of each bounding mesh edge (obtained by intersecting line segments with axis aligned solid boxes) and (ii) those whose octant edges are in the path of the mesh face (obtained by intersecting line segments with triangles).

Any terminal octant which knows at least one partially connected mesh face is classified *boundary*. Terminal octants classified *boundary* separate *interior* terminal octants from *outside* terminal octants. At this point, it should be noted that the *interior* of the model can be made of several model regions. One octant corner of a *boundary* terminal octant is then classified either *interior* or *outside* by firing a ray toward a corner of the root octant. Considering the partially connected mesh face closer to the octant corner among the ones that intersect the ray, the classification corresponding to the model region on the side of the mesh face facing the octant corner is given to the octant corner [76]. If there is no intersection, the octant corner is classified *outside*. In case the intersection is on the boundary of the partially connected mesh face, no decision can be taken and a ray to another corner of the root is fired. The classification of the octant corner is then propagated to any neighboring terminal octant (in a recursive way) which has not been classified yet. The process of classifying an octant corner and propagating its classification continues until all terminal octants have been classified.

After the basic octant classification process, *interior* terminal octants can exist which have boundary entities arbitrarily close to surface triangles in *boundary* octants. Since poorly shaped elements can result when these entities are too close, some *interior* terminal octants are reclassified as *boundary*. If an *interior* terminal octant is too close to a partially connected mesh face, it is reclassified *boundary*. In this discussion, distances between two entities are always considered relative, that is, the actual distance should be divided by the average size of the entities involved. In this particular case, the relative distance between a partially connected mesh face and an octant is equal to the absolute distance divided by the average size of the octant (its length) and mesh face. The threshold for closeness is set to 1.0, which basically guarantees that there is at least a one-element buffer between *interior* terminal octants and surface triangles.

3.3.2 Template Meshing of Interior Octants

Terminal octants classified *interior* are meshed using (i) meshing templates or (ii) fast meshing procedures when a template is not available. Examination of the number of templates required for all cases and the distribution of template usage indicates that octants with eight, nine, thirteen, and seventeen vertices cover over 90% of the interior octants. All the eight, nine, thirteen, and seventeen vertex octant configurations can be meshed by six templates (Fig. 24) with the correct rotations applied. The remaining interior octants are then quickly meshed using a fast procedure which accounts for the fact that the octant is a rectangular prism. One very fast option is to create an interior vertex and to create the correct connections to it [94].

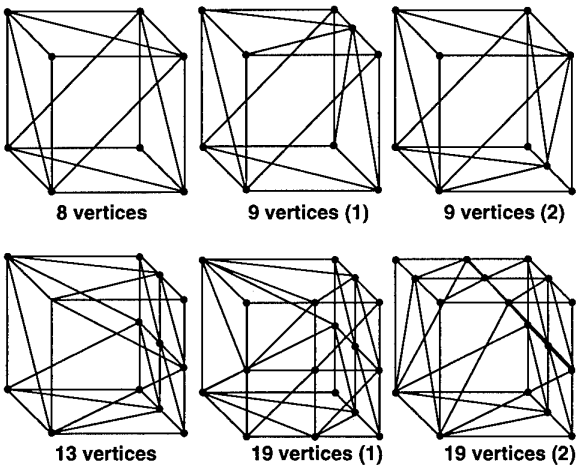


Figure 24. Terminal octant meshing templates available: one eight vertex case, two nine vertex cases, one thirteen vertex case, and two seventeen vertex cases

3.3.3 Face Removal

Given a partially connected mesh face, a face removal consists of connecting it to a mesh vertex. Since the volume to be meshed consists of the space between the given surface triangulation and the interior octree, the vertex used is usually an existing one. However, in some situations, it is desirable to create a new vertex. The choice of the target vertex (existing or new) must be such that the created element is of good quality and its creation does not lead to poor (in terms of shape) subsequent face removals in that neighborhood.

The following pseudo-code indicates how the target vertex is selected for a given partially connected mesh face to be removed. Detailed explanation for the key steps is given in the next paragraphs of the section. In this pseudo-code and any other thereafter, **break** forces an exit from a loop, **return** forces an exit from the function or routine (in other words, the function terminates), and text between /* and */ denotes a comment [40].

1. Collect set of potential target vertices from tree neighborhood
2. Reorder target vertices with respect to decreasing shape measure (for the element to be created)
3. Initialize:
   a. *dist_lim* = α
   b. *target_vert* = 0
   c. *max_min_dist* = 0.0

4. **for** each potential target vertex *vert* {
   a. Perform preliminary check on acceptability. If not acceptable, **continue**
   b. If the new element contains any mesh vertex belonging to the front, **continue**
   c. If the new element intersects any existing mesh entity, **continue**
   d. Evaluate how close the new element is to existing mesh entities (compute relative minimum distance *min_dist*)
   e. **if** (*min_dist* ≥ *dist_lim*) {
      - *target_vert* = *vert*
      - *max_min_dist*= *min_dist*
      - **break**

      }
   f. **else if** (*min_dist* > *max_min_dist*) {
      - *target_vert* = *vert*
      - *max_min_dist* = *min_dist*

      }

   }
5. **if** (*max_min_dist* ≥ *dist_lim*) **return**
6. **if** *target_vert* == 0 {
   a. Create a new vertex *vert* at the best position for the partially connected mesh face to be removed
   b. *target_vert* = *vert*

   }
7. **else** { /* Consider creating a new vertex */
   a. Create a new vertex *vert* at the best position for the partially connected mesh face to be removed
   b. Evaluate closeness of new element to existing mesh entities (*min_dist*)
   c. **if** (*min_dist* > *max_min_dist*) *target_vert* = *vert* /* Better to create a new vertex */

   }

The neighborhood of an entity is defined as a tree neighborhood of a given order. Given a mesh entity, a tree neighborhood of order 0 consists of all terminal octants that know about the entity (have the entity or part of it within their volumes). A tree neighborhood of order *n* (*n* > 0) consists of a tree neighborhood of order *n-1* to which is added all terminal octants that neighbor any octant corner of any terminal octant in the tree neighborhood of level *n-1*. The set of potential target vertices is obtained via the partially connected mesh faces in the tree neighborhood of the appropriate order for the face in consideration. The set of potential target vertices should be as small as possible (for efficiency reasons) but should not be missing the best target (with respect to both shape of new element and closeness to nearby existing mesh entities) assuming all mesh vertices of the front were considered. A tree neighborhood of order 0 is clearly not enough while a tree neighborhood of order 1 is adequate when the terminal octants have approximately the same sizes as the partially connected mesh faces they know about.

It is of interest to be able to discard potential target vertices as early as possible for purpose of efficiency. A potential target is kept only if it satisfies one of the three following conditions (types):

1. connects to a bounding vertex of the face to be removed through a mesh edge of the front. This allows for the removal of partially connected mesh faces other than the face in consideration (not in all cases) and therefore leads to a reduction of the size of the front (guaranteeing convergence of the method)
2. is positioned inside the sphere centered at the best position (with respect to shape) for the fourth vertex of the face to be removed with a radius the size of the face to be removed. This avoids the creation of a stretched element with respect to the face in consideration.
3. any of the three bounding vertices of the face to be removed are positioned inside the sphere of any of the partially connected mesh faces connected to the target vertex. This allows for the creation of a stretched element with respect to the face in consideration which is not stretched with respect to partially connected mesh faces connected to the target.

Figure 25 shows potential target vertices of type 1, 2, and 3 for the face to remove.

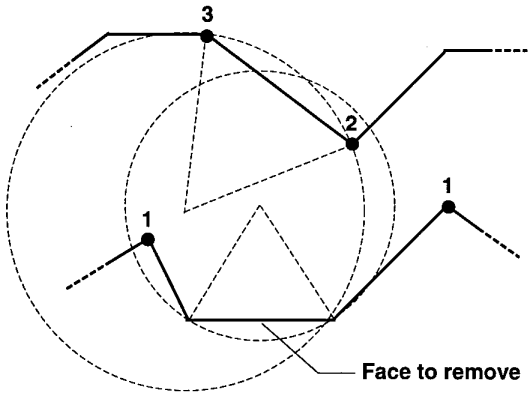


Figure 25. The three types of potential target vertices (2–d setting)

Given a potential target vertex, one has to make sure that any new mesh entity (resulting from the creation of the new mesh region) does not intersect an existing mesh entity. The creation of a new mesh region may result in the creation of a new mesh vertex, up to three new mesh edges, and up to three new mesh faces. New mesh edges are checked for intersection against nearby partially connected mesh faces. Given a virtual new mesh edge, the nearby partially connected mesh faces are obtained through the tree neighborhood of order 0 (of the new edge). If no intersection is detected, new mesh faces are checked for intersection against nearby front mesh edges. Given a virtual new mesh face, nearby front mesh edges

are obtained through the partially connected mesh faces in the tree neighborhood of order 0 (of the new face). Because any terminal octant knows about the partially connected mesh faces in its volume, considering a tree neighborhood of order 0 guarantees that no intersection can be missed.

The closeness of the new mesh region to existing mesh entities is evaluated by considering the minimum relative distance between any new mesh entity and nearby existing mesh entities. The relative distance is defined as the absolute distance divided by the average size of the mesh entities involved. The nearby mesh entities are obtained through a tree neighborhood of order 1 of the new entity being tested. It is important to note that nearby existing mesh entities in a tree neighborhood of order 1 may not be in a tree neighborhood of order 0. On the other hand, nearby existing mesh entities cannot be missed with a tree neighborhood of order 1. If there is a new vertex, distances between the new vertex and nearby existing partially connected mesh faces are considered. For any new mesh edge, distances between the new edge and nearby existing front mesh edges are considered. If the point on the new edge corresponding to the distance (that is, closest to the nearby existing front mesh edge) corresponds to an existing bounding mesh vertex, the distance is discarded. In that case, it means that the nearby existing front mesh edge is close to another existing mesh entity and not to a new mesh entity. Also, for any new face, distances between the new face and nearby existing front mesh vertices are considered. Again, distances are discarded if the point on the new mesh face corresponding to the distance (that is, closest to the nearby existing front mesh vertex) is actually on an existing bounding mesh vertex or edge. The three different cases are shown in Figure 26. The threshold α corresponds to what is considered acceptable in terms of closeness when creating a new element. Experimentation led to the use of a value of 0.2 for α.

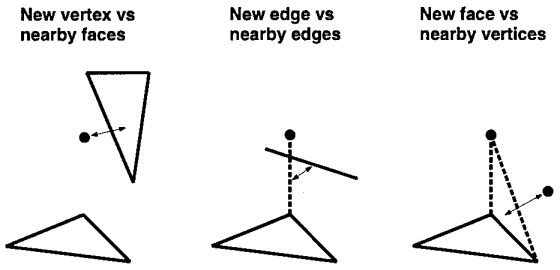


Figure 26. Evaluation of relative minimum distance between new entities and nearby existing mesh entities

If a new vertex needs to be created, its location must be such that the new element is well-shaped, and neither causes intersection nor is too close to nearby existing mesh entities. The initial location for the new vertex is at the position which creates the best shaped element for the face to be removed. This location is on the perpendicular to the face passing through the centroid. If the current location causes the new element to intersect nearby existing mesh entities, a new location is considered on the normal half-way from the current location, and so on, until a valid location is found. In order not to be too close to existing mesh entities, the final location is considered conservatively half-way from the current location.

Figure 27 graphically depicts a face removal in a two-dimensional setting. There are four target vertices ordered (1, 2, 3, and 4) with respect to increasing shape measure of the element to be created. Target vertex 1 is rejected since the new element is too close to an existing mesh entity (vertex 3). Target vertex 2 is rejected since the new element intersects existing mesh entities. Target vertex 3 is therefore accepted.

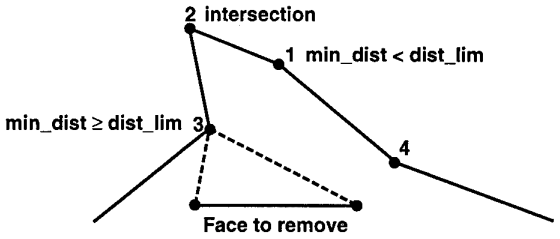


Figure 27. Potential target vertices and best face removal (2–d setting)

3.4. Parallel Constructs Required

3.4.1 Octree and Mesh Data Structures

The two main data structures are the mesh and octree data structures. The mesh data structure (sequential) and parallel mesh data base (PMDB) both described above are used here to support the presented mesh generator. The octree data structure is on top of the mesh data structure. To gather a tree neighborhood or all terminal octants in the path of a mesh entity (new vertex, edge, or face), any processor must be able to effectively determine to which processor any given terminal octant is assigned. This information is easily available when each processor has full knowledge of the basic octree in terms of structure and processor assignment. This is the approach currently implemented. Although the size of the tree is small compared to that of the mesh and this tree information can easily be copied to each processor, this approach does not scale indefinitely. Any terminal octant stores links to on-processor partially connected mesh faces and off-processor partially connected mesh faces totally or partially within its volume. Octree neighboring information (like finding terminal octants neighboring an octant face, edge, or corner) is obtained through tree traversals (logarithmic complexity).

Techniques that maintain only portions of the tree on individual processors while providing tree neighboring information efficiently are currently under investigation. It is of interest to be able to retrieve tree neighboring information without having to communicate. If communication is allowed during a neighboring information request, some processors will have to interrupt and be involved in the

request, which certainly can degrade the overall performance if not done carefully. An easy solution is to make sure that all processors participate in the request (soft synchronization). On a sequential machine, performing tree traversals to obtain neighboring information, typically, getting all terminal octants that neighbor an octant entity (face, edge, or corner) can be avoided if octant face neighboring terminal octants are stored. The limited increase in data storage is well worth the constant time complexity for getting neighboring information. In a parallel setting, it is difficult to conceive such a scheme without having to communicate between processors.

3.4.2 Multiple Octant Migration

When the mesh generation process comes to a point when no face removal can be applied (face removals are not applied when needed tree neighborhoods are not fully on processor), the tree and associated mesh is repartitioned. The migration of octants is key to repartitioning once decisions concerning new destinations of terminal octants (classified *boundary*) have been made. Multiple octant migration itself relies on the multiple migration of partially connected mesh faces and/or mesh regions (described above). Note that multiple mesh region migration is also used in the final repartitioning at the region level once the mesh has been fully generated.

Any processor can send any number of terminal octants to another processor. When a terminal octant is migrated from one processor to another, the partially connected mesh faces not connected to any mesh region (these are the mesh faces remaining from the given surface triangulation) owned by the octant and/or the mesh regions that are bounded by at least one partially connected mesh face owned by the octant are migrated as well. An octant owns a mesh entity when it knows about it (has it within its volume) and has its centroid within its volume. Note that a partially connected mesh face not known by the octant may be migrated as part of a mesh region if that region is bounded by another partially connected mesh face whose owner is the octant. Also, if a mesh region is bounded by more than one partially connected mesh face known to the octant to be migrated (up to four), the ownership is arbitrarily dictated by the first partially connected mesh face to be processed (from the list of partially connected mesh faces known to the octant). Figure 28 shows a two-dimensional example of the mesh regions to be migrated within an octant. When the multiple octant migration completes, the processor is informed of the octants it has received. For each received octant, a list of associated mesh entities is also given, basically the partially connected mesh faces and/or mesh regions that were sent.

The primary complexity that arises when migrating octants and associated mesh information is the absence of a global labeling system for the mesh entities. Each processor employs a local labeling for the hierarchy of mesh entities that it is assigned. The interprocessor mesh adjacency links maintain the required knowledge of the adjacent mesh entities on neighboring processors. Although the mesh data for a partially connected face is on one

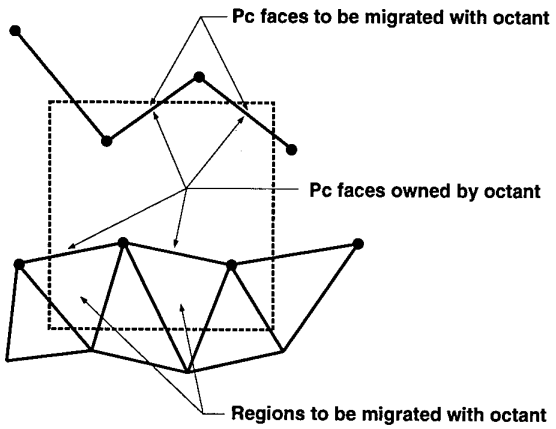


Figure 28. Octant migration

processor, the octants which refer to that face may be on multiple processors. Since the face removal procedure must perform geometric checks on all partially connected faces known to that octant, the time required to perform these operations would be greatly increased if the required information had to be fetched from neighboring processors. To eliminate this requirement, each partially connected face known to an octant will either be a pointer to face, when the face is actually on-processor, or a set of three coordinates when the face is stored off-processor. Although this approach avoids interprocessor communications, it complicates the process of updating references to partially connected mesh faces on and off-processor when octants are migrated. Concerning the update of processor assignment at the octant level, since the tree structure is currently stored on all processors, a broadcast is performed to all processors indicating the fact that octants have been relocated.

3.4.3 Dynamic Repartitioning

Dynamic repartitioning enables redistribution of the load among processors as evenly as possible at key stages of the mesh generation process. These key stages are:

1. at the beginning of template meshing,
2. at the beginning of each face removal step, and
3. at completion of the mesh generation process.

Repartitioning for stages 1 and 2 is done at the terminal octant level (1 with respect to terminal octants classified *interior* and 2 with respect to terminal octants classified *boundary*). Repartitioning for stage 3 is performed at the mesh region level. The strategy is identical for both cases, only the process of migrating differs. The methods used here are geometry-based dynamic balancing (repartitioning) procedures which are described in section 2.3.1.

3.5. Parallel Region Meshing

3.5.1 Underlying Octree

At this point in time, the octree is built sequentially on a single processor (processor 0). Since a sequential octree building can become a bottleneck when dealing with

very large meshes, techniques to build the tree in parallel are currently considered. A distributed tree can be constructed in parallel as long as operators to subdivide and migrate octants are available. Octant migration guarantees that the tree can be well distributed at any stage during the building process, which is important memory wise. Those operators are key to the problem since they update possible inter-processor links in a distributed tree.

3.5.2 Template Meshing of Interior Octants

Once all terminal octants have been properly classified, the terminal octants classified *interior* are partitioned. The parallel application of templates is a straight forward process in which there is no communication required during the process of creating the octant level meshes. It should be noted that the application of templates to octants sharing the same octant face implicitly lead to the same octant face triangulation. The finite elements generated in these octants are loaded into the processor mesh data structure. The interprocessor communication required at the end of this step is for the updating of inter-processor mesh entity links for mesh entities created on the boundaries of interior octants which are on processor boundaries. The cost for the application of templates is small compared to the cost of performing face removals. Therefore, the parallel efficiency of parallel region meshing is dictated primarily by the face removal part only.

3.5.3 Face Removal

Parallel face removal is an iterative process where each iteration consists of three steps:

1. Tree repartitioning at the terminal octant (classified *boundary*) level,
2. Face removal step, and
3. Reclassification of terminal octants from *boundary* to *meaningless*

The goal of step 1 is to make sure that all processors will have an equal amount of work to perform during step 2. It is difficult to predict how much work or, more precisely, how many face removals (step 2) any processor will perform and the total amount of effort for a particular face removal. However, a terminal octant classified *boundary* is a good unit of work load since the set of all terminal octants classified *boundary* approximately corresponds to the domain still to be meshed. The difficulty of performing face removals in parallel resides in the fact that any face removal requires the knowledge of tree neighborhoods. Tree neighborhoods of order 0 or 1 are needed at different steps of the removal of a given mesh face. If, at any point during the face removal, a tree neighborhood is not fully on-processor, the face removal is aborted and the next mesh face is considered for removal. Once all possible face removals have been performed on processor, some terminal octants classified *boundary* which used to know about partially connected mesh faces (on or off-processor) are reclassified *meaningless*. Because those octants no longer cover any portion of the domain still to be meshed, they are now useless (for the purpose of face removals) and will therefore not influence the next repartitioning.

Figure 29 depicts the first iteration on a simplistic example. In the left-side picture, terminal octants classified *boundary* have been partitioned and each of them is assigned to a processor (0 to 3). The right-hand side picture shows the current mesh after all possible face removals have been performed on processors. Shaded areas represent the domain still to be meshed.

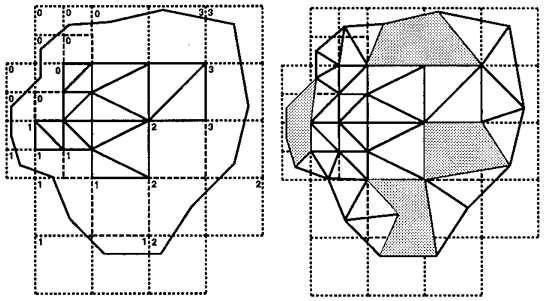


Figure 29. Parallel face removal (2–d setting)

The process of performing face removals and repartitioning the tree continues until there are no more partially connected mesh faces in the mesh. Define the efficiency of the face removal stage as being the ratio of the number of performed face removals to the number of attempted face removals. After a few iterations, the efficiency of the face removal stage can be very low because information required to perform face removals is almost always off-processor. When more than half of the processors have an efficiency below some given threshold (25%), the processor set is reduced (by half).

Since migration of terminal octants only deals with those classified *boundary* and only worries about mesh regions bounded by partially connected mesh faces, it is very likely that the final mesh will be scattered across processors with no real structure. It is therefore necessary to repartition in parallel the distributed mesh using IRB at the mesh region level with the original full set of processors. Figure 30 shows the whole process of parallel face removal on four processors. The first 8 pictures display the currently partially connected mesh faces after the terminal octants classified *boundary* have been repartitioned. Note that iterations 1, 2, 3, and 4 use all four processors, iterations 5, 6, and 7 use two processors, and iteration 8 uses one processor. The final picture displays the final three-dimensional repartitioned mesh on four processors.

Tables 4 and 5 show speed-ups for up to four processors for the *connecting rod* and *blade* models, respectively (final repartitioned meshes on four processors are shown in Fig. 31 and Fig. 32, respectively). Tables 6, 7 and 8 show speed-ups for up to eight processors for the *onera wing*, *mechanical part*, and *mechanical part 2* models, respectively (final repartitioned meshes on eight processors are shown in Fig. 33, 34, and Fig. 35, respectively). The number of mesh regions created indicated in the captions corresponds to parallel face removal only and does

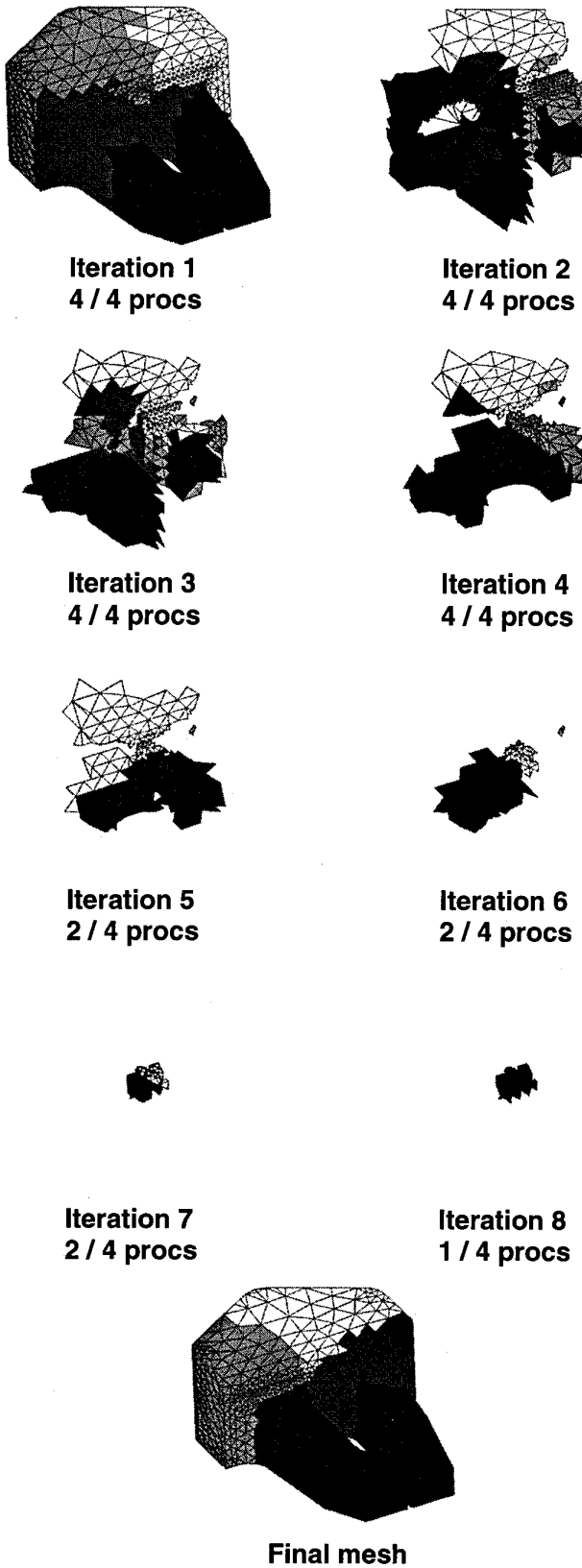


Figure 30. Successive face removal iterations and final repartitioned mesh for *chicklet*

Procs	1	2	4
Iterations	1	5	7
Face removal speedup	1.0	1.9	3.3
Total speedup	1.0	1.8	2.9

Table 4 Face removal statistics for *connecting rod* (35,000 mesh regions created by face removals — 70,000 total)

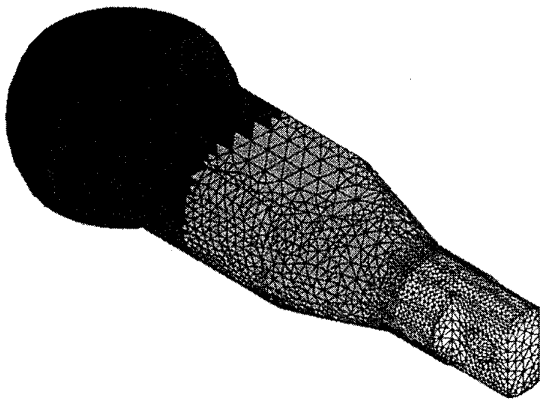


Figure 31. Final repartitioned mesh for *connecting rod* (4 processors)

Procs	1	2	4
Iterations	1	5	8
Face removal speedup	1.0	2.0	3.2
Total speedup	1.0	1.9	2.8

Table 5 Face removal statistics for *blade* (60,000 mesh regions created by face removals — 90,000 total)

not include template meshing. Face removal speed-up indicates speed-up for step 2 of the parallel face removal procedure. Total speed-up indicates speed-up for all steps (1, 2, and 3). In that case, the first repartitioning (iteration 1) is not counted since it can be considered an initial partitioning step. Note that the time taken to perform the first repartitioning depends on the size of the problem and not the number of processors. The speed-up is by definition set to 1.0 for the run with the smallest number of processors. The results show good speed-ups as long as the size of the problem is adequate with the number of processors on hand.

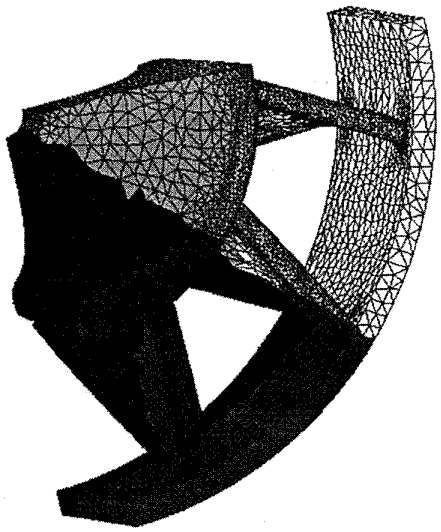


Figure 32. Final repartitioned mesh for *blade* (4 processors)

Procs	2	4	8
Iterations	4	7	11
Face removal speedup	1.0	1.9	2.8
Total speedup	1.0	1.8	2.6

Table 6 Face removal statistics for *onera wing* (90,000 mesh regions created by face removals — 220,000 total)

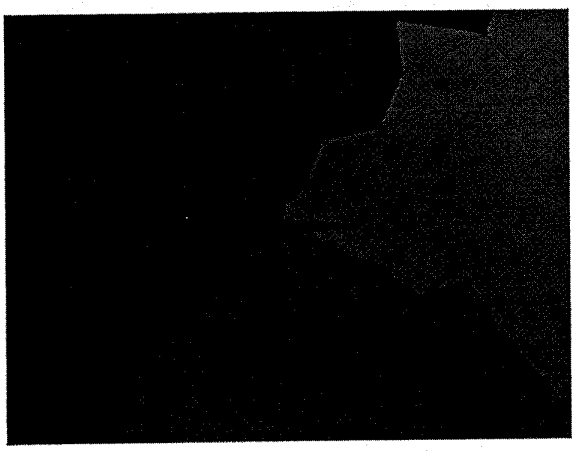


Figure 33. Final repartitioned mesh for *onera wing* (8 processors)

4. Parallel Mesh Enrichment

4.1. Local Retriangulation Tools

Local retriangulation techniques have been used to transform locally non-Delaunay triangulations of a set of points into Delaunay triangulations [35], generate Geometric triangulations of models with faceted boundaries [28] (boundary recovery), optimize existing triangulations [25, 17], etc. The local retriangulation tools presented in this section do not delete or create vertices. The mesh entity splitting presented in the refinement section creates a vertex. The edge collapsing presented in the derefinement section deletes a vertex. Local retriangulation tools are used here to optimize triangulations (locally or globally) and to help in "snapping" refinement mesh vertices to the model boundary (if required).

Procs	2	4	8
Iterations	4	7	12
Face removal speedup	1.0	2.0	3.2
Total speedup	1.0	1.9	3.0

Table 7 Face removal statistics for *mechanical part* (120,000 mesh regions created by face removals – 230,000 total)

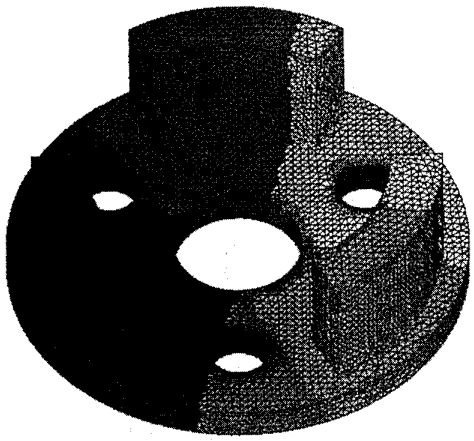


Figure 34. Final repartitioned mesh for *mechanical part* (8 processors)

Procs	2	4	8
Iterations	4	7	11
Face removal speedup	1.0	2.0	3.4
Total speedup	1.0	1.9	3.2

Table 8 Face removal statistics for *mechanical part 2* (125,000 mesh regions created by face removals — 240,000 total)

Figure 35. Final repartitioned mesh for *mechanical part 2* (8 processors)

A few definitions related to triangulation quality relevant to local retriangulation tools are now given:
Triangulation quality: *If each mesh entity ${}_mT_i^{d_i}$ of a triangulation ${}_m\Omega$ is associated with a quality measure q_i, the quality of the triangulation is defined as* $Q = \min(q_i)$
Triangulation acceptability: *Given a quality threshold q_l, a triangulation ${}_m\Omega$ is acceptable with respect to triangulation quality if* $Q > q_l$.
Triangulation comparison: *A triangulation ${}_m\Omega_i$ of a set of points is considered better with respect to triangulation quality than another triangulation ${}_m\Omega_j$ of the same set of points if* $Q_i > Q_j$.

4.1.1 Edge Swapping

In two and three dimensions, a swapping step is performed after inserting a new node into the triangulation to transform a locally non-Delaunay triangulation into a Delaunay one. Aside from the refinement issue, it is a method to incrementally build a Delaunay triangulation of a set of points.

Swapping relies on the general result given by Lawson which states that a set of *n+2* points in R^n may be triangulated in at most two ways [42]. In two dimensions, there are two ways to triangulate a strictly convex quadrilateral. Edge swapping consists of switching diagonals for the quadrilateral resulting from the union of the two connected triangles (if convex). In three dimensions, there are two ways to triangulate a strictly convex triangular hexahedron containing five and only five points (the five apices of the triangular hexahedron). Joe provided a set of workable swappable configurations for the three-dimensional case [35]. If a mesh face ${}_mT_1^2$ is not locally optimal (does not satisfy the Delaunay criterion) and corresponds to one of the two situations on the left side of figure 36, it is swapped. If $[{}_mT_4^0, {}_mT_5^0] \cap {}_mT_1^2 \neq \emptyset$ ($[{}_mT_4^0, {}_mT_5^0]$ being the line segment spanning from ${}_mT_4^0$ to ${}_mT_5^0$), the triangular hexahedron initially containing two tetrahedra is retriangulated with three. If $[{}_mT_4^0, {}_mT_5^0] \cap (S_1 \cup S_2 \cup S_3) \neq \emptyset$ (where the S_i's are plane sectors appearing shaded in figure 36) and $\exists\ {}_mT_1^3 \mid \{{}_mT_2^0, {}_mT_3^0, {}_mT_4^0, {}_mT_5^0\} \in \partial({}_mT_1^3)$, the triangular hexahedron initially containing three tetrahedra is retriangulated with two.

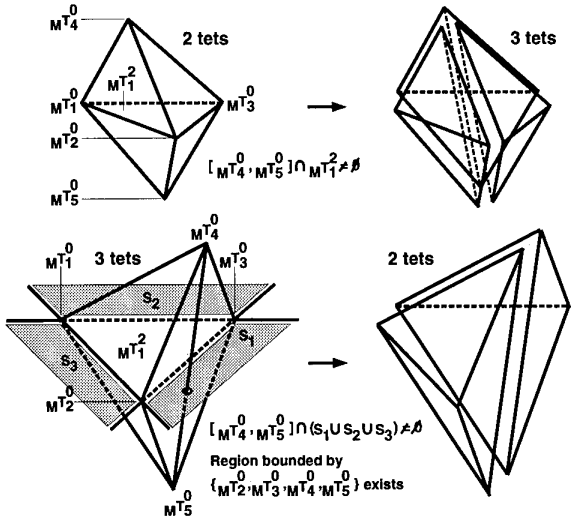


Figure 36. 2–to-3 and 3–to-2 swaps in three dimensions

These swaps, commonly referred to as 2–to-3 and 3–to-2, are suited for Delaunay triangulations and by extension for regular triangulations [19]. Referring to Fig. 36, if $[{}_mT_4^0, {}_mT_5^0] \cap (S_1 \cup S_2 \cup S_3) \neq \emptyset$ and $\forall {}_mT_i^3\ \{{}_mT_2^0, {}_mT_3^0, {}_mT_4^0, {}_mT_5^0\} \notin \partial({}_mT_i^3)$ or $[{}_mT_4^0, {}_mT_5^0] \cap ({}_mT_1^2 \cup S_1 \cup S_2 \cup S_3) = \emptyset$, there is no possible swap. When dealing with Delaunay triangulations (or regular triangulations), theoretical results indicate that non swappable faces (in Joe's sense) are not critical. However, when dealing with any other criterion, non swappable faces (in Joe's sense) may be critical. The other non swappable configuration from figure 36 which corresponds to $[{}_mT_4^0, {}_mT_5^0] \cap ({}_mT_1^2 \cup S_1 \cup S_2 \cup S_3) = \emptyset$ consists of four tetrahedra bounded by $\{{}_mT_1^0, {}_mT_2^0, {}_mT_4^0, {}_mT_5^0\}$, $\{{}_mT_2^0, {}_mT_3^0, {}_mT_4^0, {}_mT_5^0\}$, $\{{}_mT_1^0, {}_mT_2^0, {}_mT_3^0, {}_mT_4^0\}$, and $\{{}_mT_1^0, {}_mT_2^0, {}_mT_3^0, {}_mT_5^0\}$, respectively [35]. It is clear that there is no other way of triangulating this convex hull. The ideas presented by Brière de l'Isle and George [17] about edge removal enable the extension of the classic 3–to-2 swap [35, 19].

4.1.2 Edge Removal

Brière de l'Isle and George [17] have proposed an edge removal technique as part of an algorithm to optimize the quality of a given mesh. It can also be used as part of a scheme to recover the faceted boundary of a model [28]. A mesh edge ${}_mT_1^1 \sqsubset {}_gT_1^3$ which is bounded by vertices ${}_mT_1^0$ and ${}_mT_2^0$ can be eliminated by retriangulating the polyhedron of all connected tetrahedrons. The polyhedron is retriangulated by: (i) triangulating the polygon

of all mesh vertices of the polyhedron which are neither ${}_mT_1^0$ nor ${}_mT_2^0$, and (ii) connecting the new mesh faces to ${}_mT_1^0$ and ${}_mT_2^0$ (Fig. 37).

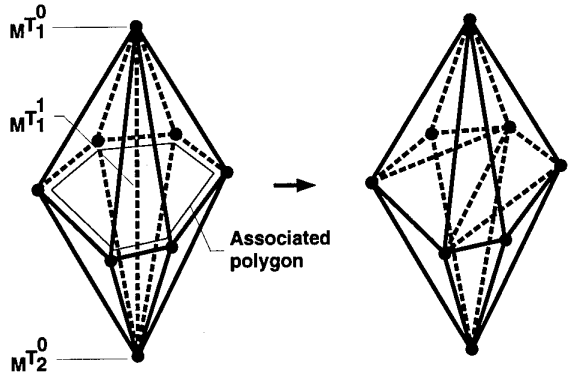


Figure 37. Edge removal

If the polyhedron originally consisted of *m* tetrahedra, the associated polygon has *m* sides and *m* apices. The number of possible retriangulations is $N_m = \sum_{i=3}^{m} N_{i-1}N_{m+2-i}$ with $N_2 = 2$ [27, 17]. Clearly, if the polyhedron is not convex, some possible retriangulations have to be discarded. The number of different triangles when considering all retriangulations is $NT_m = m(m-1)(m-2)/6$ [17]. Table 9 shows computed values of N_m and NT_m for m = 3 to 9 [17].

m	3	4	5	6	7	8	9
N_m	1	2	5	14	42	132	429
NT_m	1	4	10	20	35	56	84

Table 9 Number of possible retriangulations and different triangles as *m* (number of connected tetrahedra) increases

Since N_m grows rapidly, Brière de l'Isle and George have chosen m = 9 as an upper limit for their edge removal scheme. It should be noted that those retriangulations represent all possible triangulations of the polyhedron that do not have ${}_mT_1^1$ but are only a subset of all possible retriangulations of the polyhedron.

Given a mesh edge ${}_mT_1^1$ connected to more than one mesh region, edge removal consists of retriangulating the polyhedron $pol({}_mT_1^1)$ of all mesh regions connected to ${}_mT_1^1$ in such a way that ${}_mT_1^1$ is not present in the retriangulation (Fig. 37). When edge removal is topologically possible, the mesh edge is said to be topologically removable. An edge removal is positive (negative) if the retriangulation of the polyhedron is better (worse, respectively) than the original triangulation, in other words, the variation of the local triangulation quality is positive (negative, respectively). A brief description of the algorithm to remove an edge in the context of Geometric triangulation optimization follows [17]:

1. Determine quality Q_{org} of triangulation of $pol({}_mT_1^1)$
2. Get the associated polygon as an ordered list of vertices
3. Consider among all possible retriangulations of $pol({}_mT_1^1)$ those that are better than the original one and keep track of highest ($max(Q_{new})$)
4. If $max(Q_{new})$ exists:
   a. Delete all mesh regions connected to ${}_mT_1^1$ to form a polyhedral cavity
   b. Retriangulate such that new quality is $max(Q_{new})$

The initial triangulation of the polyhedron for a mesh edge classified in model region is such that there are:

1. *m* mesh regions,
2. *m* interior (with respect to the polyhedron) mesh faces, and
3. *1* interior mesh edge connected to *m* mesh regions.

The resulting triangulation is such that there are:

1. *2m-4* mesh regions,
2. *m-2* interior mesh faces, and
3. *m-3* interior mesh edges each connected to *4* mesh regions.

A local retriangulation tool like edge removal is typically used to remove an undesirable mesh region from a triangulation. Since a mesh region has six edges, there are six possibilities to remove the mesh region using edge removals. It is sometimes of interest to have more ways to remove that mesh region. Beside edge collapsing and mesh entity splitting, the procedure that reverses the edge removal process can be used to attempt to remove the mesh region. This new procedure is described in the next section and is called multi-face removal.

4.1.3 Multi-Face Removal

Multi-face removal is a procedure that reverses edge removal, in other words, it considers a configuration that could have resulted from edge removal and obtain the starting configuration. When applied to a single mesh face, it is the classic 2–to-3 swap [35, 19].

Given a simply connected set of mesh faces $\{{}_mT_i^2\}$ such that any mesh face in the set connects (through a mesh region) to ${}_mT_1^0$ on one side and ${}_MT_2^0$ on the other side, the polyhedron $pol(\{{}_mT_i^2\})$ is defined by the union of all mesh regions that connect to a mesh face in $\{{}_mT_i^2\}$. Multi-face removal retriangulates the polyhedron $pol(\{{}_mT_i^2\})$ such that all mesh faces in $\{{}_mT_i^2\}$ are removed. As for the edge removal, a multi-face removal is positive (negative) if the new triangulation is better (worse, respectively) than the original one. Multi-face removal is topologically possible if (i) the deletion of all mesh regions in $pol(\{{}_mT_i^2\})$ does not lead to the deletion of a mesh vertex, and (ii) the set of mesh edges peripheral to $\{{}_mT_i^2\}$ constitutes a single loop that does not touch itself. Figure 38 illustrates cases of multi-face removals that are not topologically possible. When a multi-face removal is topologically possible, the set of mesh faces

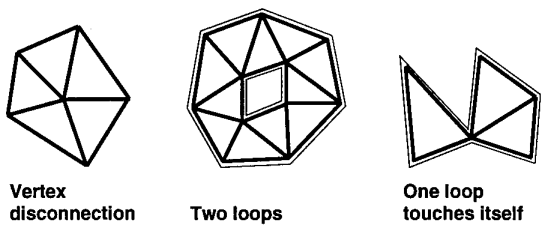


Figure 38. Cases when the topological state of the set of mesh faces $\{{}_mT_i^2\}$ prevents multi-face removal

$\{{}_mT_i^2\}$ (as well as any mesh face in the set) is said to be topologically removable.

Since the goal of the presented optimization algorithm is to get rid of undesirable mesh regions, the input to the multi-face removal procedure is a mesh region ${}_mT_1^3$ and a bounding mesh face ${}_mT_1^2$ from which the simply connected set of mesh faces is constructed. The description of the algorithm follows:

1. Get vertex ${}_mT_1^0$ opposite ${}_mT_1^2$ in ${}_mT_1^3$ (Fig 39.a)
2. Get region ${}_mT_2^3$ on other side of ${}_mT_1^2$
3. Get the vertex ${}_mT_2^0$ opposite ${}_mT_1^2$ in ${}_mT_2^3$ (Fig 39.b)
4. Gather all pairs of face-connected mesh regions such that one mesh region connects to ${}_mT_1^0$ and the other connects to ${}_mT_2^0$. Keep track of the mesh faces in-between pairs of mesh regions ($\{{}_mT_i^2\}$). The set of gathered mesh regions defines $pol(\{{}_mT_i^2\})$ (Fig 39.c)
5. If retriangulation would create invalid elements, do not perform removal
6. Compute quality of initial triangulation Q_{org}
7. Compute quality Q_{new} of triangulation that would result from connecting all boundary faces of $pol(\{{}_mT_i^2\})$ to ${}_mT_1^0$
8. If $Q_{new} < Q_{org}$, do not perform removal
9. Delete the mesh regions in $pol(\{{}_mT_i^2\})$ to form a polyhedral cavity
10. Connect all faces of polyhedral cavity to ${}_mT_1^0$ (Fig 39.d)

The initial triangulation of the polyhedron (Fig 39.c) is such that there are:

1. *m* mesh regions (note that *m* is an even number),
2. *3m/2–2* interior (with respect to the polyhedron) mesh faces, and
3. *m/2–1* interior mesh edges each connected to *4* mesh regions.

The resulting triangulation (Fig 39.d) is such that there are:

1. *m/2+2* mesh regions,
2. *m/2+2* interior mesh faces, and
3. *1* interior mesh edge connected to *m/2+2* mesh regions.

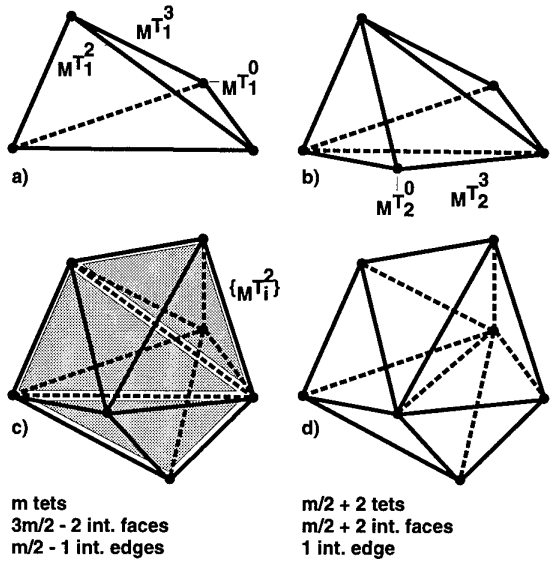


Figure 39. Multi-face removal in three dimensions

4.1.4 Triangulation Optimization Using Local Retriangulation Tools

The goal of the optimization algorithm is to improve the quality of Geometric triangulations with respect to a given criterion (e.g., element shape). The optimization procedure described here makes use of the local retriangulation tools described above, namely edge removal and multi-face removal. Other local retriangulation tools which change the number of mesh vertices like mesh entity splitting, edge collapsing, and even local remeshing are not incorporated into this specific optimization procedure. Also, smoothing techniques (vertex repositioning) [23, 14] are not addressed. In this discussion, triangulation optimization can be used over the whole triangulation or locally over a sub-triangulation resulting from adaptive enrichments such as refinement and derefinement.

The optimization procedure is region based, that is, it looks for mesh regions that are not acceptable (quality below q_l) and attempts to remove them from the triangulation with local retriangulation tools. Given a non acceptable mesh region ${}_mT_1^3$, one can potentially remove that mesh region from the triangulation by considering edge removal with respect to any of its four bounding edges or multi-face removal with respect to any of its four bounding faces. The optimization algorithm is described as follows:

1. Initialize queue *Qu* of non acceptable mesh regions (quality below q_l)
2. If (*Qu* empty) or (there is no edge removal or multi-face removal that can successfully be applied to any mesh region in *Qu*), end
3. Pop a region from *Qu*
4. Consider which edge removal (with respect to any bounding edge) or multi-face removal (with respect to any bounding face) gives the best quality improvement of the corresponding polyhedron

5. If either an edge removal or a multi-face removal has been performed:
   a. Remove from Qu any deleted region
   b. Enqueue any new non acceptable mesh region

   Else (re)enqueue mesh region
6. Goto step 2

The process terminates when either the queue is empty or no local retriangulation can be applied to any mesh region in the queue. It terminates in a finite number of steps. This is easily proven by examining the criterion for locally retriangulating. A domain is locally retriangulated only if the quality of the new triangulation of the domain is strictly greater than the original one. Assume the above process does not terminate, the quality of the global triangulation would improve indefinitely which cannot be. If the queue is empty when the program terminates, the resulting geometric triangulation is acceptable and the goal of the optimization procedure has been met. If the queue is not empty when the procedure terminates, neither local transformation procedure (edge removal or multi-face removal) could be applied to any of the mesh regions in the queue.

The optimization of a triangulation using local retriangulation techniques leads to a local optimum. Depending in which order the local transformation procedures are applied, different local optima can be reached. Also, local retriangulation procedures may have to be applied even if they are negative. It is impossible to say whether those local optima are far or close to the global optimum. It is conjectured that a global optimum cannot be reached with local retriangulation techniques. However, in practice, these local retriangulation techniques often improve the quality of a triangulation.

4.2. Refinement

Refinement algorithms have been decomposed into three groups, depending on which technique they are based: i) subdivision patterns [2, 6, 57, 48, 9, 38], ii) bisection (generalized [59, 60, 45, 44] and alternate [3]), and iii) insertion in a Delaunay context [87] or by mesh entity splitting [53, 30, 46]. The following sections describe these known schemes and introduce a new procedure which considers a full set of subdivision patterns, therefore allowing the possibility of no over-refinement. A set of definitions is given prior to the description of the refinement algorithms:

Conformity: *An n-dimensional triangulation* ${}_m\Omega$ *is conforming if the intersection of any two non disjoint elements is a common d-dimensional geometric entity with* $0 \leq d < n$. It is assumed here that conformity is a requirement. Figure 40 illustrates the definition with a two-dimensional example.

Triangulation refinement sequence: *The ordered set* $\{{}_m\Omega_1, {}_m\Omega_2, .., {}_m\Omega_N\}$ *is a triangulation refinement sequence if* $\forall\ i \in [1, N-1]$ ${}_m\Omega_{i+1}$ *is obtained by selectively refining* ${}_m\Omega_i$.

Nesting: *A triangulation* ${}_m\Omega_i$ *is nested into a triangulation* ${}_m\Omega_j$ *if any element of* ${}_m\Omega_i$ *is fully inside one element of* ${}_m\Omega_j$.

Refinement stability: *A refinement scheme is stable if all interior angles of all triangulations in the sequence* $\{{}_m\Omega_1, {}_m\Omega_2, .., {}_m\Omega_N\}$ *are bounded from below and above as N goes to infinity.*

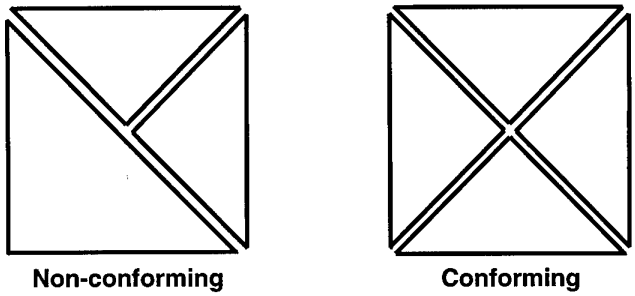


Figure 40. Non-conforming and conforming triangulations in two dimensions

4.2.1 Subdivision Patterns

In the two-dimensional case, two subdivision patterns are commonly used: i) regular 1:4 (each child triangle is similar to the parent) and ii) "green" 1:2 (Fig. 41). Bank and Sherman [2] use a 1:4 subdivision scheme to refine elements. Any element with two or three non-conforming vertices is 1:4 subdivided (iteratively). At this stage, all elements can not have more than one non-conforming vertex. A clean-up phase which "green" subdivides any remaining non-conforming element completes the process. For the next refinement iteration, if an element resulting from a "green" subdivision is marked for refinement, the parent element is reinstated and 1:4 subdivided (Fig. 42). This ensures an angle is not divided more than once.

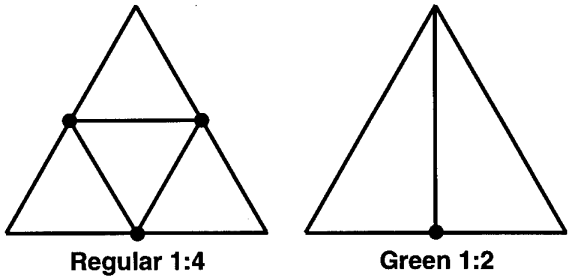


Figure 41. Classic element subdivision patterns in two dimensions

In three dimensions, given a mesh region, subdivision patterns are applied depending on the number of marked edges. The set of available subdivision patterns varies. Biswas and Strawn [6], Rausch et Al. [57], and Löhner and Baum [48] have the 1:2, 1:4, and 1:8 subdivision schemes (Fig. 43). Bornemann et Al. [9] have the 1:2 ("green I"), "green II", 1:4 ("green III"), and 1:8 subdivision schemes. The "green II" scheme corresponds to the case where there are 2 non-conforming edges for the element. Kallinderis and Vijayan [38] use the 1:2, 1:4, 1:8, and a centroidal node subdivision schemes. In the centroidal node subdivision scheme, a vertex is created

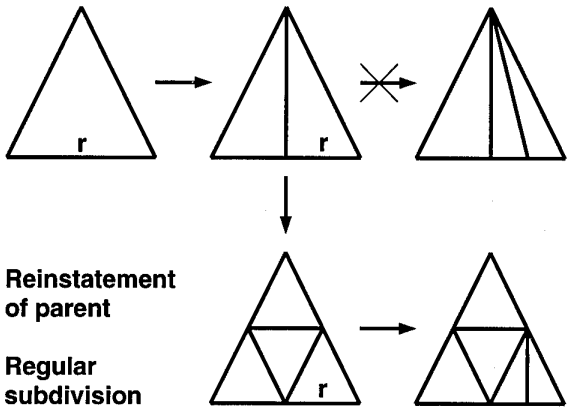


Figure 42. Reinstatement of parent element followed by regular subdivision in two dimensions

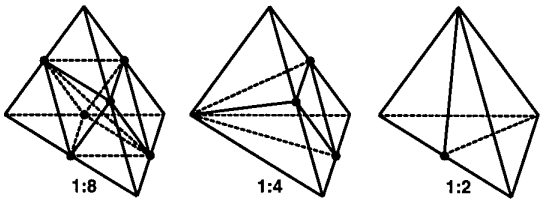


Figure 43. Classic element subdivision patterns in three dimensions

at the centroid of the element and the element is split accordingly. If a marking pattern does not correspond to a predefined configuration, it is upgraded to the closest one. The process terminates in a finite number of steps. It has been shown that the regular 1:8 subdivision scheme is stable as long as the proper (shortest) inner diagonal is chosen [29, 5]. Algorithms based on subdivision patterns are stable if irregular child elements (not resulting from regular subdivision) are never further subdivided, in other words, parents of those are reinstated and subdivided with the 1:8 subdivision scheme prior to any further subdivision [57, 48, 9]. Note that ${}_m\Omega_i$ is always nested into ${}_m\Omega_1$ but ${}_m\Omega_{i+1}$ may not be nested into ${}_m\Omega_i$ due to the possible reinstatement of parents ($i \geq 2$). Any refinement scheme based on subdivision patterns which does not have all possible subdivision patterns and/or reinstates some parent elements prior to further subdivision will in general over-refine, that is, produce more refinement than requested by the adaptive procedure. Also, using subdivision patterns which add a centroidal vertex when not actually needed will over-refine as well.

4.2.2 Generalized Bisection

In two dimensions, an element is refined by bisecting its longest edge (two–triangle algorithm) [59]. Elements with non-conforming edges are subdivided following the patterns of Figure 44. The process terminates in a finite number of steps. Following the results of Rosenberg and Stenger [61] and Stynes [82] about longest edge bisection, the scheme is stable, furthermore, interior angles are always greater than one half of the lowest angle in the initial triangulation ${}_m\Omega_1$ [59].

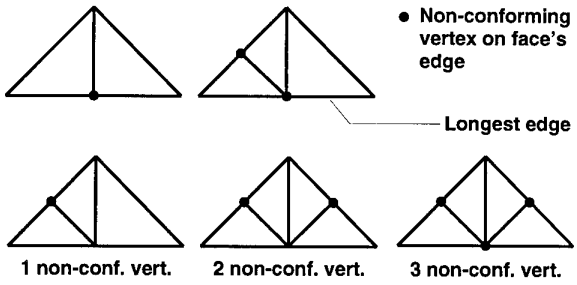


Figure 44. Non-conforming elements and their triangulations in two dimensions

This method of subdivision along the longest edge has been extended to three dimensions [60]. Elements to be refined are bisected along their longest edges. Non-conforming elements are subdivided along their longest edges in a recursive fashion. Unlike the two-dimensional case, an element that needs refinement or is non-confoming must be bisected at its longest edge.

This scheme guarantees nesting. In two dimensions, following the longest edge bisection results of Rosenberg and Stenger [61] and Stynes [82], the scheme is stable, furthermore, interior angles are always greater than one half of the lowest angle in the initial triangulation ${}_m\Omega_1$ [59]. In three dimensions, to this point in time, no one has yet presented a proof of the stability of the scheme probably because (i) the longest edge in a mesh region is not necessarily opposite the largest dihedral angle and (ii) the sum of all dihedral angles of a mesh region is not constant. However, the scheme seems to be "experimentally" stable. Because the non-conformity can propagate, this scheme will in general over-refine.

Joe [45] has proven that the infinite bisection of a tetrahedron is stable using generalized bisection on a mapped special tetrahedron. Note that this result does not prove that generalized bisection in the real space is stable. Liu and Joe [44] have presented a stable refinement algorithm that makes use of this result. In ${}_m\Omega_1$, for each element, a bisected edge is uniquely chosen (this does not mean that all elements will be subdivided). Elements that need to be subdivided are bisected along their bisected edges. When an element is subdivided into two elements, the bisected edges for the two new elements are imposed according to rules given in [44]. Once all elements that need refinement have been subdivided, there may be some non-conforming elements in the triangulation. The process of subdividing elements continues until there are no more non-conforming elements in the mesh. At this point, the scheme guarantees nesting, is stable, and will in general over-refine. After all levels of refinement have been applied, local transformations [35] are applied to further improve the quality of the final mesh. It should be noted that if local transformations are applied after each refinement iteration, a priori control of stability is lost. From experimental results given in [44], this scheme over-refines less than the scheme by Rivara and Levin [60] especially as the number of refinement levels becomes high. As a

remark, this scheme appears very similar to the alternate bisection scheme [3] presented in the next section.

4.2.3 Alternate Bisection

This approach has been presented by Bänsch [3]. In ${}_m\Omega_1$, refinement edges are chosen for each element (a good choice is the longest edge). Note that choosing a refinement edge for each element does not mean that all elements will be subdivided. Elements that need to be subdivided are bisected along their refinement edges. When an element is subdivided into two elements, the refinement edges are topologically imposed on the two new elements according to Fig. 45. A conforming step subdivides elements with non-conforming edges.

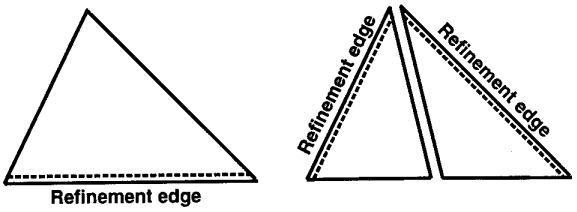


Figure 45. Alternate bisection in two dimensions

This scheme extends to three dimensions. In ${}_m\Omega_1$, each face is given a refinement edge (e.g., longest edge). For each element in ${}_m\Omega_1$, it is then assumed there is at least one common refinement edge for two adjacent faces, called a global refinement edge. In particular, this assumption holds if the longest edge of each face in ${}_m\Omega_1$ is chosen as the refinement edge. Each element to be refined is bisected along its global refinement edge. The refinement edges on the four new faces resulting from bisection of the two parent faces are imposed as in the two-dimensional case and the refinement edge on the interior face is chosen according to rules given in [3]. The two new elements are then guaranteed to have a global refinement edge. Elements with non-conforming edges are bisected until no non-conforming elements remain.

This scheme guarantees nesting and is stable [69, 3]. Because the non-conformity can propagate, this scheme will in general over-refine.

4.2.4 Delaunay Insertion

Inserting a new node into a Delaunay triangulation can be done using, for example, Watson's algorithm [87]. All elements which contain the new node (in terms of circumcircle in two dimensions or circumsphere in three) are deleted to form a point-convex polyhedral cavity. New elements are created by connecting the boundary of the cavity to the new node (Fig. 46). The new triangulation is guaranteed to be Delaunay.

4.2.5 Splitting

The insertion of a point into a triangulation can be done by splitting the mesh entities the new point falls on. In three dimensions, a mesh region can be split into four new regions, a mesh face into three new faces, and a mesh edge into two new edges. Mesh entity

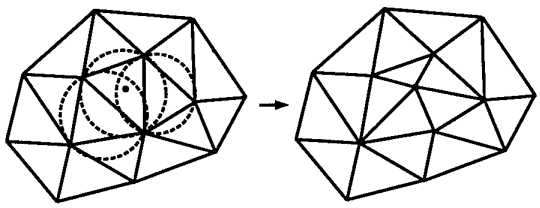


Figure 46. Watson's algorithm in two dimensions

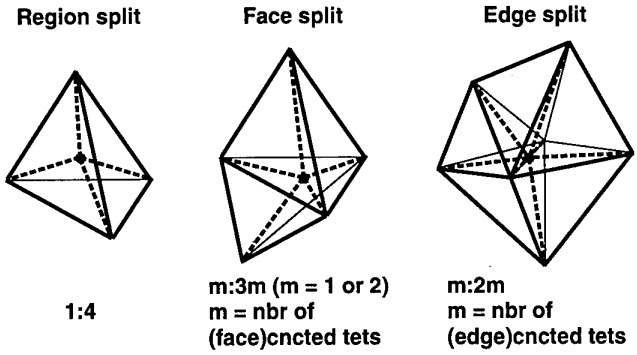


Figure 47. Mesh entity splitting in three dimensions

splitting retriangulates a polyhedron by adding a vertex and connecting the boundary faces of the polyhedron to the new vertex. In the case of a mesh region, the polyhedron is the mesh region itself. In the case of a mesh face or a mesh edge, the polyhedron is built from the union of all mesh regions connected to the face or edge, respectively. Figure 47 displays the three types of split in three dimensions and indicates for each one of them the change in number of mesh regions.

This technique can be used to add vertices into a given triangulation. For instance, if the error indicator is edge-based, any marked mesh edge is split. Mesh entity splitting guarantees nesting, is not stable, and will not over-refine if the mesh entities that are marked for refinement are the only ones to be split.

For a given mesh region to refine, Golias and Tsiboukis [30] split its longest edge, which leads to the refinement of all tetrahedra connected to the edge. At this point, the scheme guarantees nesting, is not stable, and does not artificially refine. Then, Delaunay transformations and node relaxation (repositioning) techniques are applied to improve the quality of the resulting triangulation (nesting is lost). The Delaunay transformations used are:

1. exchange of interface faces (in Fig. 36 upper-left, when the faces bounded by $\{{}_mT_1^0, {}_mT_2^0, {}_mT_4^0\}$ and $\{{}_mT_1^0, {}_mT_2^0, {}_mT_5^0\}$ are classified on the same model face, a 2–to-2 swap which is a degenerate case of the 2–to-3 swap can be applied),
2. 2–to-3 and 3–to-2, and
3. local transformation of tetrahedron (the above three transformations are applied recursively to the tetrahedron under consideration, then the tetrahedron's neighbors, etc).

Muthukrishnan et al. [53] sort mesh regions that are to be refined with respect to increasing length of their longest edges. The first region to be refined is the one at the end of the list. Before splitting the longest edge, the regions connected to the edge are examined. If a connected region has a longest edge different from the edge to be split, it is put in the list of regions to be refined at the appropriate rank. After the split, the list is updated. This refinement scheme is actually identical to the scheme described by Rivara and Levin [60] and therefore has the same properties. It is followed by a node repositioning procedure (nesting is lost).

Lo [46] sorts (in an approximate way) the mesh edges marked for refinement with respect to increasing length. The mesh edge at the end of the list is split and the list is updated. This scheme is different from the one by Rivara and Levin [60] and Muthukrishnan et al. [53] since only edges marked for refinement will be split. At this point, the scheme guarantees nesting, is not stable, and does not artificially refine. It is followed by a triangulation optimization procedure which makes use of node repositioning and local transformations (nesting is lost). These local transformations are:

1. 2–to-3,
2. 3–to-2, and
3. 4–to-4 which is an edge removal when there are four mesh regions connected to an edge.

4.2.6 Refinement Using Full Set of Subdivision Patterns

Refinement is performed by marking appropriate mesh edges for refinement and applying subdivision patterns to each mesh region. Each mesh region has from zero to six marked edges. Subdivision patterns for each possible configuration of marked edges have been developed in order to annihilate any over-refinement. There are ten possible patterns which are as follows (Fig. 48):

1. 1–edge: this is the classic 1:2 subdivision pattern (one template)
2. 2–edge (this is also the Green II in [9]):
   a. One face has two marked edges (two templates)
   b. All faces have one marked edge (one template)
3. 3–edge:
   a. One face has three marked edges: this is the classic 1:4 subdivision pattern (one template)
   b. Two faces have two marked edges (four templates)
   c. Three faces have two marked edges (eight templates)
4. 4–edge:
   a. One face has three marked edges (four templates)
   b. All faces have two marked edges (sixteen templates)
5. 5–edge (four templates)
6. 6–edge: this is the classic 1:8 subdivision pattern (one template)

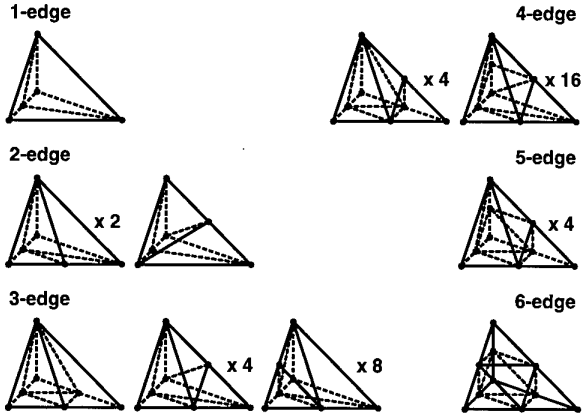


Figure 48. Subdivision patterns in three dimensions

When only the 1:2, 1:4, and 1:8 subdivision patterns are used, there is no possible triangulation incompatibility at the face level, in other words, the subdivision patterns on both sides of a face with either one or three marked edges will always match (at the face level). Inclusion of all the refinement types requires explicit consideration of triangulation compatibility at the face level. If a face with two and only two marked edges has been triangulated due to the subdivision of one region using that face, the template used to subdivide the other region must match the face triangulation. Since there are a priori two ways to triangulate a face with two marked edges (Fig. 49), any pattern which has N faces with two and only two marked edges needs 2^N templates.

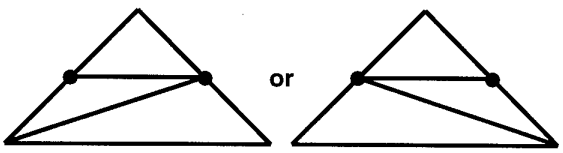


Figure 49. The two ways to triangulate a mesh face with two marked edges

As is, this refinement scheme is not stable since it is possible, and likely, that an angle (solid) will be bisected more than once when multiple refinements are applied in the same areas. However, it can be made stable at the price of some over-refinement. Assuming the quality of the initial triangulation ${}_m\Omega_1$ is Q_1, stability requires that for any subsequent triangulation ${}_m\Omega_i$ $(i > 1)$ its quality Q_i is such that $Q_i \geq q_l$ with $q_l = \alpha Q_1$ where α is some constant. Given a mesh region with at least one marked edge but fewer than six, the template corresponding to the number of marked edges is applied and the optimization procedure (with q_l as the threshold) is applied locally to the subdivided mesh region. If the optimization procedure is successful, nothing else has to be done for that mesh

region. However, if the optimization procedure is unsuccessful, the situation that existed prior to the application of the template is recovered and the subdivision pattern is upgraded by marking an additional edge. This process is repeated until the optimization procedure is successful or the number of marked mesh edges reaches six. Recall that the application of the 1:8 template (with shortest inner diagonal) does not affect the stability of the refinement scheme (neutral) [29, 5]. Another approach is to directly upgrade to six marked mesh edges and apply the 1:8 template. It should be noted that as soon as a subdivision pattern is upgraded, neighboring mesh regions have to be reprocessed for subdivision.

In the context of curved model boundary, vertices resulting from refinement that are classified on the model boundary need to be "snapped" to the appropriate model entity. For example, when studying the flow around an airfoil, it is critical to be able to snap refinement vertices to the airfoil (especially at the leading edge) in order to make sure the resulting flow corresponds to the actual airfoil geometry. Difficulties may arise as moving a vertex to its destination target can generate invalid elements especially when the triangulation is rather coarse. Figure 50 illustrates this problem in two dimensions when the triangulation around a circular hole is selectively refined. If the snapping of a refinement vertex causes a mesh region to be invalid or of poor quality, the local retriangulation tools described above can be used to attempt to remove that mesh region. This process is repeated until the refinement vertex can be snapped. Note that other local retriangulation techniques, such as edge collapsing, mesh entity splitting, and local remeshing can be applied to these situations. It is possible that local retriangulation tools may not succeed in snapping all refinement vertices, however, it is believed that local remeshing will permit all snappings. Efforts are under way to complete the appropriate algorithms. A mesh adaptation procedure should in theory not only be stable (with respect to triangulation quality) but also capable of snapping all refinement vertices classified on the model boundary to the proper model entities. This new requirement lessens the importance of stability and justifies the presented mesh adaptation procedure which makes use of local retriangulation tools to optimize the current triangulation and snap refinement vertices.

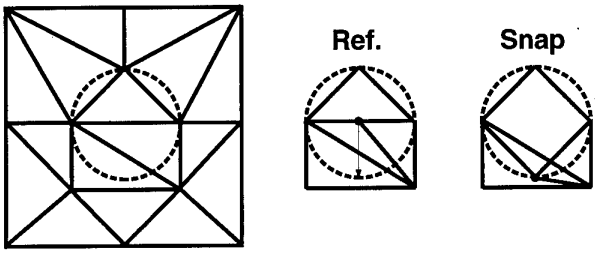


Figure 50. Snapping refinement vertex to the model boundary in two dimensions can generate geometric invalidity

4.3. Derefinement

Schemes that use subdivision patterns or bisection for refinement can derefine by simply reversing the refinement process [6, 59, 48, 38, 57]. To illustrate this concept, consider the methodology employed by Biswas and Strawn [6] which is representative of such derefinement schemes. If two sibling edges (same parent edge) are marked for derefinement, they are replaced by the parent edge and all parent elements sharing the parent edge are reinstated. The procedure described for refinement and/or conformity can then be applied to the set of elements that have been reinstated. Figure 51 shows a simple example of derefinement. In Figure 51.a, edges marked with a "d" are to be derefined. Once the parent edges have been reinstated, any parent edge with at least one child edge not marked "d" is marked "r" (Fig. 51.b). The refinement procedure described earlier is then applied to produce the final derefined triangulation (Fig. 51.c). It should be noted that Biswas and Strawn [6] perform refinement and derefinement simultaneously. Derefinement has been separated only in the scope of the present paper. In order to efficiently reinstate parent entities, parent elements and edges are stored resulting in an overhead estimated at 15% of total memory requirements in Biswas and Strawn's case. It should be noted that any triangulation in the sequence cannot be coarser than the first one.

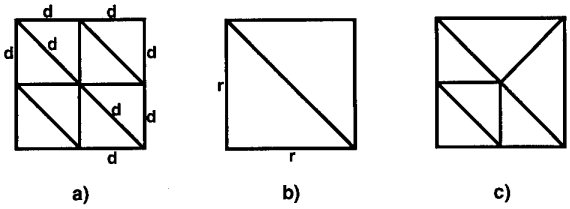


Figure 51. Derefinement example

Derefinement is performed here by using a local retriangulation technique that deletes a vertex: edge collapsing. A mesh edge is derefined by collapsing it to one of its end vertices. A description of the algorithm follows (see also Fig. 52 for a graphical description):

1. Check if edge collapsing is topologically possible. If it is possible, one end vertex is the collapsed vertex (${}_mT_1^0$) while the other is the target vertex (${}_mT_2^0$)
2. Check if edge collapsing is geometrically possible
3. Delete all mesh regions connected to ${}_mT_1^0$, which produces a polyhedral cavity
4. Connect the faces of the polyhedral cavity to ${}_mT_2^0$ to form new mesh regions

Since edge collapsing locally modifies a Geometric (valid) triangulation [67, 77], one has to make sure the validity of the triangulation is not violated by the modification (this check refers to step 1 of the algorithm). Since any mesh entity is classified against the model, it is always possible to predict such violations. Figure 53 contains the pseudo-code to check if a mesh edge can be collapsed to one of its end vertices. It returns TRUE

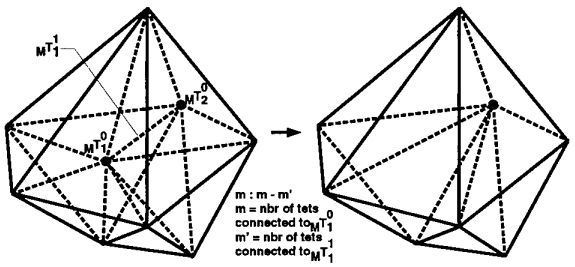


Figure 52. Edge collapsing in three dimensions

if the mesh edge can be collapsed (FALSE otherwise). Figure 54 illustrates graphically some of the cases where edge collapsing is not possible which are pointed out in the pseudo-code.

Before physically collapsing the edge, the geometry of the mesh regions to be created can be predicted exactly (this check refers to step 2 of the algorithm). The volumes of the new mesh regions can be computed by considering all mesh regions which are connected to ${}_mT_1^0$ but not connected to ${}_mT_1^1$ and virtually moving ${}_mT_1^0$ to ${}_mT_2^0$. Since the computation of the volume of a mesh region always consider the bounding vertices in a certain order, the (virtual) movement of one of its bounding vertices is valid only if the new volume is positive. Therefore, one can always tell beforehand if the to-be created mesh regions are invalid. The quality of the to-be created mesh regions can be predicted as well. If the quality of the to be created elements is not good enough with respect to some predetermined threshold, the derefinement of the edge need not be performed. This is important in order to guarantee the stability of the refinement/derefinement scheme. Also, assuming both end vertices are candidates to be the target vertex, the target vertex that would create the "better" triangulation of the two is chosen.

4.4. Complete Mesh Adaptation Procedure

The actual implementation of the mesh adaptation scheme uses the following steps:

1. Derefinement using edge collapsing as described above
2. Global optimization with $q_l = Q_1$
3. Refinement using full set of subdivision patterns without consideration for stability
4. Refinement vertex snapping (to the model boundary)
5. Global optimization with $q_l = Q_1$

So far, problems due to the non-stability of the implemented refinement scheme have not appeared. If they happen, the refinement can be made stable as described above at the price of some over-refinement.

4.5. Parallelization of Mesh Adaptation

Today's CFD computations are costly both in CPU time and memory. For big enough problems, the flow solver cannot be run on a classic scalar workstation for which performance and memory are limited. For large-scale analysis of fluid flows, it is necessary to use a parallel flow solver. Since the mesh adaptation is an integral part

```
Get bounding vertices ({}_mT_1^0 ⊏ {}_gT_1^{d_1^0}, {}_mT_2^0 ⊏ {}_gT_2^{d_2^0})
of edge {}_mT_1^1 ⊏ {}_gT_1^{d_1^1}
if d_1^0 = d_2^0
    if {}_gT_1^{d_1^0} = {}_gT_2^{d_2^0}
        if d_1^0 = 3 return TRUE (ok to collapse)
    else return FALSE (cannot collapse) (Fig. 54.a)
else
    if d_1^0 = 3 or d_2^0 = 3 return TRUE (target vertex is the one classified on lower order model entity)
    At this point, the two mesh vertices are classified on model boundary
    if d_1^1 = 3 return FALSE
    Switch (if necessary) {}_mT_1^0 and {}_mT_2^0 so that d_1^0 > d_2^0 (from now on, target vertex will be {}_mT_2^0 if collapsing is possible)
    if {}_gT_1^{d_1^1} ≠ {}_gT_1^{d_1^0} return FALSE (Fig. 54.b)
At this point, the two vertices are classified on model boundary and the edge is classified on the model entity of higher order
for each pair of mesh edges ({}_mT_2^1 ⊏ {}_gT_2^{d_2^1}, {}_mT_3^1 ⊏ {}_gT_3^{d_3^1}) that connect to {}_mT_1^0 and {}_mT_2^0 respectively and connect to each other
    if d_2^1 = 3 or d_3^1 = 3 continue
    if d_2^1 = d_3^1
        if {}_gT_2^{d_2^1} ≠ {}_gT_3^{d_3^1} return FALSE
        else if d_2^1 = 1 return FALSE
    At this point, the two edges are classified on same model face or one is classified on model face and the other is classified on the model face's boundary
    Switch (if necessary) {}_mT_2^1 and {}_mT_3^1 so that d_2^1 > d_3^1
    Find face {}_mT_1^2 ⊏ {}_gT_1^{d_1^2} bounded by {{}_mT_1^1, {}_mT_2^1, {}_mT_3^1}
    if {}_mT_1^2 does not exist, return FALSE
    if {}_gT_1^{d_1^2} ≠ {}_gT_2^{d_2^1} return FALSE (Fig. 54.c)
for each pair of mesh faces ({}_mT_1^2 ⊏ {}_gT_1^{d_1^2}, {}_mT_2^2 ⊏ {}_gT_2^{d_2^2}) that connect to {}_mT_1^0 and {}_mT_2^0 respectively and connect to each other by a mesh edge
    if d_1^2 = 2 and d_2^2 = 2 return FALSE (Fig. 54.d)
    if ({}_mT_1^2, {}_mT_2^2) do not bound a mesh region, return FALSE
return TRUE
```

Figure 53. Pseudo-code for checking topological validity for edge collapsing

of the flow solver, it must be running in parallel as well in order not to become a bottleneck.

4.5.1 Derefinement

If a mesh edge ${}_mT_1^1$ is marked for derefinement, it is attempted to be collapsed. If the polyhedron $pol({}_mT_1^0)$ is on processor p_i, the edge collapsing is performed on p_i. If $pol({}_mT_1^0)$ is not fully on p_i, the missing mesh regions are requested from the appropriate processors. When all processors are done traversing their lists of mesh edges, the processors that have received requests send (migrate)

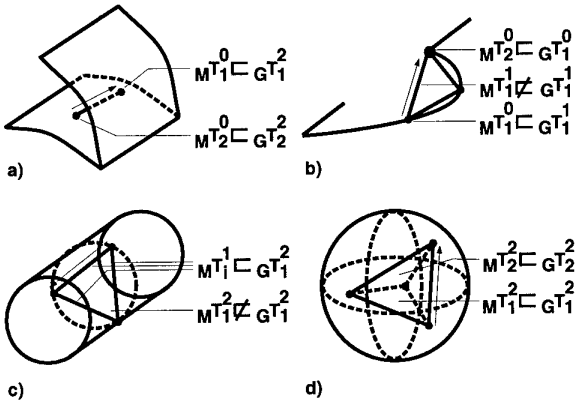


Figure 54. Some cases when edge collapsing in three dimensions is not possible due to topology

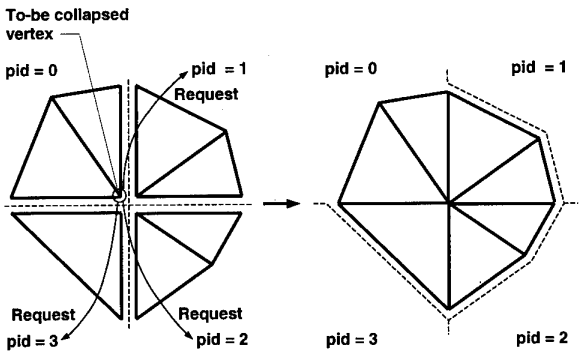


Figure 55. Mesh migration to support parallel distributed derefinement

the requested mesh regions. In Figure 55, processor p_0 requests mesh regions from processors (p_1, p_2, p_3) and the requested mesh regions are migrated. If there is conflict, the processor with lowest p_i has priority. On the next iteration, it is the processor with highest p_i that will have priority. This switching is done to prevent too much load imbalance at completion. The process of traversing the list of mesh edges and sending/receiving requests continues until all marked mesh edges have been collapsed (more exactly, have been attempted to be collapsed). Because mesh regions are migrated, it is possible that the processors are not well balanced after the derefinement step. The triangulation is therefore submitted to a load balancing step (at the region level) before going further. Figure 56 shows the speed-ups for a triangulation of approximately 85,000 elements where 50% of the mesh edges are derefined (the resulting triangulation has approximately 46,000 elements).

4.5.2 Triangulation Optimization

Assuming the current triangulation is partitioned, each processor p_i ($0 \leq i < n_p$) optimizes its own partition (${}_pT_{p_i}$) considering a global quality threshold q_l. As processor p_i pops a mesh region from its queue Qu_i, two situations may occur:

1. All polyhedra to be considered for edge removal and multi-face removal are fully on p_i (that is, all mesh regions of all polyhedra belong to p_i), in that case, the proper local retriangulation tool can be applied (if needed) and any new mesh region of quality below q_l is pushed in Qu_i
2. At least one polyhedron is not fully on p_i, in that case, p_i requests for each polyhedron (concerning edge removal and multi-face removal) any mesh region that is not on p_i and push back the mesh region in Qu_i

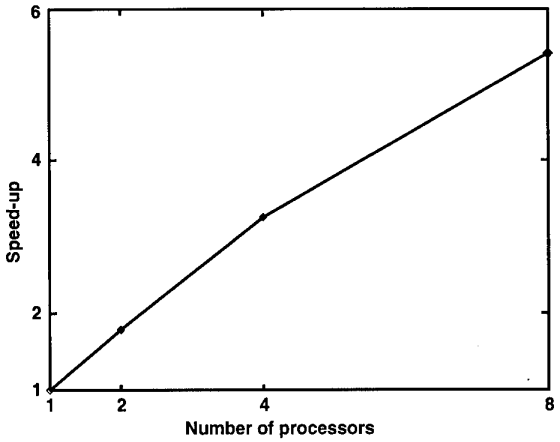


Figure 56. Speed-ups for derefinement (85,000 elements — 50% edges derefined)

The mesh region popping process continues until Qu_i is empty or stuck (does not change). Clearly, requests concerning mesh regions that have been deleted since are cancelled. After a synchronization step, all processors examine the requests they have received and send (migrate) the appropriate mesh regions to the appropriate processors. If a mesh region is requested by several processors, the processor with lowest p_i has priority and will be granted the mesh region. On the next iteration, it is the processor with highest p_i that will have priority. This switching is done to prevent too much load imbalance at completion. Each processor p_i adds to its queue Qu_i any new mesh region of quality below q_l that it has received and restarts popping mesh regions. The combined process of emptying the queue and migrating requested mesh regions terminates when all queues Qu_i ($0 \leq i < n_p$) are empty or stuck and there is no mesh region to migrate. Because mesh regions are migrated, it is possible that the processors are not well balanced after the optimization step. The triangulation is therefore submitted to a load balancing step (at the region level) before going further. Figure 57 shows the speed-ups for a triangulation of approximately 85,000 elements.

4.5.3 Refinement

Any mesh face on some partition boundary with at least one marked mesh edge is triangulated using two-dimensional subdivision patterns (Fig. 58). Since two sibling mesh faces (physically identical mesh faces on two neighboring procs) have same orientation, it is guaranteed that the application of these templates will produce physically identical triangulations (in terms of child faces).

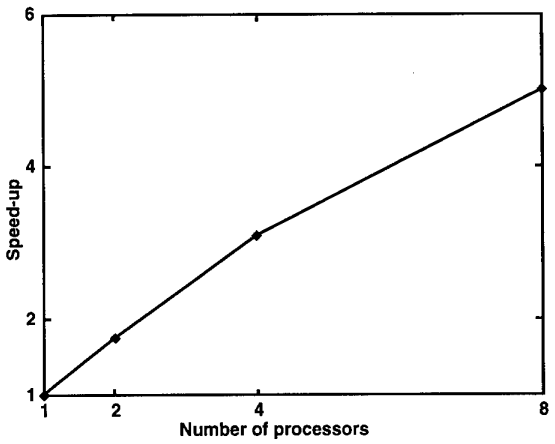


Figure 57. Speed-ups for the optimization procedure (85,000 elements)

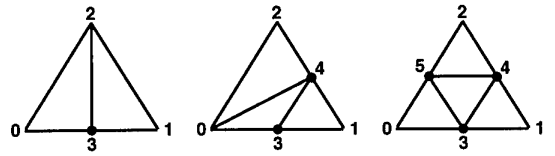


Figure 58. Subdivision patterns at the mesh face level

Once all mesh faces on the partition boundary are subdivided, links for all new mesh entities are updated. Then, each processor can apply the three-dimensional templates on any mesh region with at least one marked edge (as described above) without any communication.

Once all appropriate mesh regions have been subdivided, the refinement vertices which are classified on the model boundary need to be snapped to the corresponding model entity. Since snapping makes use of the local retriangulation tools, the technique to parallelize that process is similar to the one used to parallelize the derefinement and optimization steps. All processors iterate on a two step process: (i) (sequential) vertex snapping along with requests for missing mesh regions, and (ii) sending of requests and migration of requested mesh regions until all refinement vertices have been attempted to be snapped. At the end of the refinement step, the processors may not be well balanced for two reasons: (i) refinement is selective, and (ii) mesh regions have been migrated (due to snapping). Therefore, a load balancing step is applied before going further. Figure 59 shows speed-ups for the refinement procedure on 36,000 elements when 20% of the mesh edges are refined (resulting triangulation has 88,000 elements).

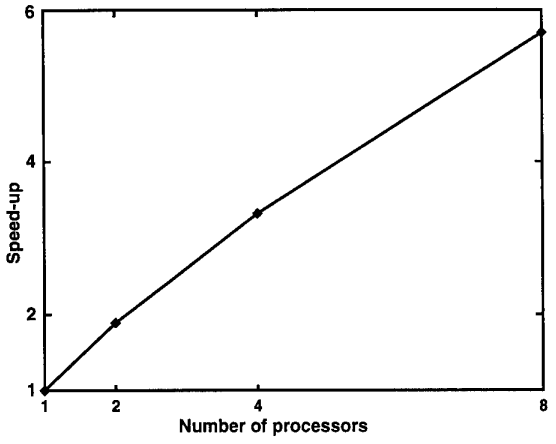


Figure 59. Speed-ups for parallel refinement

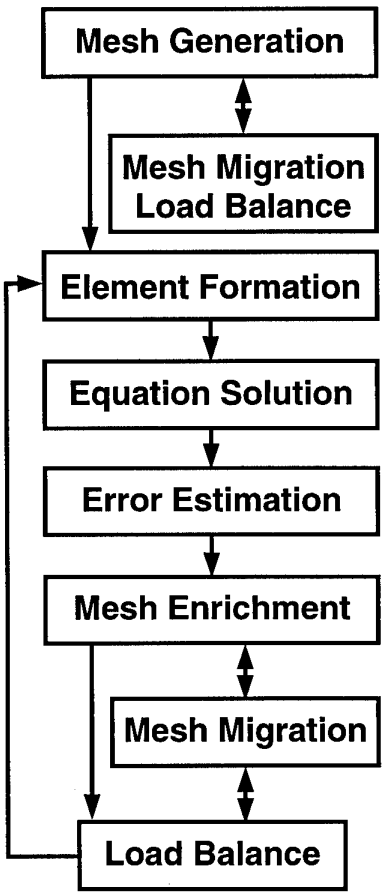


Figure 60. Components of a parallel adaptive analysis procedure

5. Parallel Adaptive Analysis Procedures

5.1. Structure of a Parallel Adaptive Analysis Procedure

Although the most computationally intensive operations in an adaptive analysis are of the same type as those of a fixed mesh analysis, an adaptive analysis must use more general structures which effectively account for the evolution of the discretization. The structure of a parallel adaptive analysis procedure follows directly from the procedures used for the parallel control of evolving meshes presented in the previous sections. Figure 60 presents an overall flow chart of a parallel automated adaptive analysis procedure.

Two main processing phases naturally emerge in the finite element method, the "form phase", where the local finite element arrays at the sub-domain level are generated, and the "solve phase", where the global problem is solved. Parallel implementation of the form phase is straightforward, in the sense that it can be performed in parallel with no communication among the processing nodes.

On the other hand, the efficient scalable realization of the solve phase is a non trivial task. Current Multiple Instruction/Multiple Data (MIMD) computers tie together independent processors using a high speed switch under a message passing paradigm. The resulting system incorporates relatively powerful individual processors with large local memories. The communication bandwidth between processors remains well below that of the individual processor to memory bandwidth resulting in significant cost for interprocessor communication compared to local computation. This type of architecture has significant impact on the design of parallel algorithms for the solution of large linear systems. Such algorithms must amortize any communication costs over large amounts of simultaneous parallel computation. Additionally, the large local data space is still only a fraction of the global memory space and data cannot be highly duplicated over multiple processors if full advantage is to be taken of the available memory. Under these constraints, two different algorithm classes become attractive for the solution of linear systems, Krylov space based iterative solvers and domain decomposition techniques.

In the remainder of this subsection a brief review of the Krylov space based GMRES procedure used in the rotorcraft aerodynamics discussed in subsequent subsections is given. Readers interest in more information on Krylov space based domain decomposition methods are referred to the chapters of this report by van der Vorst and Farhat, respectively.

Given the non–symmetric linear system $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$, the Generalized Minimal Residual (GMRES) algorithm of Saad and Schultz [62] attempts to find the approximate solution $\mathbf{p}_0 + \mathbf{z}$, $\mathbf{z}$ being in the Krylov space $K = (\mathbf{r}_0, \mathbf{A} \cdot \mathbf{r}_0, \ldots, \mathbf{A}^{k-1} \cdot \mathbf{r}_0)$ and $\mathbf{r}_0 = \mathbf{b} - \mathbf{A} \cdot \mathbf{p}_0$. $\mathbf{z}$ is the solution of the minimization problem $\min_{\mathbf{z} \in K} \| \mathbf{b} - \mathbf{A} \cdot (\mathbf{p}_0 + \mathbf{z}) \|$, which is solved by means of the QR algorithm. The GMRES algorithm obtains an orthonormal basis of K by means of a Gram–Schmidt procedure which involves matrix–vector multiplications and dot products. These operations represent the computer intensive part of the algorithm. In general, all Krylov methods can be written in terms of these two basic kernels. It is therefore important to devise efficient ways of performing distributed matrix-vector and dot product operations in parallel.

The matrix–vector multiplications necessitate the exchange of data through the inter–processor boundaries. In order to overlap communication and computation for efficiency reasons, these operations can be realized following a four step procedure on each processing node: *(i)* send data relative to the inter-processor boundaries to each neighboring processor, *(ii)* perform computations involving only data relative to nodes that lie within the internal volume of the partition, *(iii)* receive data relative to the inter–processor boundaries from all the neighbors, *(iv)* perform computations involving only data relative to nodes lying on the inter–processor boundaries.

For the implementation of the dot product operations, nodes that lie on the inter–processor boundaries are randomly split, so that two partitions that share an internal boundary are assigned only a subset of the nodes of that internal boundary. Each processing node then performs the dot product involving nodes contained in its internal volume and its subset of nodes on the partition boundaries. Global sum of the local results at the processor level yields the global dot product result.

The minimization problem in the GMRES algorithm can be written in terms of an upper Hessenberg matrix, whose entries are essentially the results of the dot products performed during the orthogonalization procedure. At the end of the Gram–Schmidt procedure, each processing node has then complete knowledge of the upper Hessenberg matrix and it is therefore able to perform the solution of the minimization problem independently with no communication. It should be remarked that the size of the Hessenberg matrix is the size of the Krylov space employed, typical values for the applications here considered being around 5–30. The computer intensive SAXPY operations needed in order to update the solution of the linear system are consequently performed in parallel with no communication. Once convergence is achieved in the iterative linear solver, each processing node has complete knowledge of the incremental solution at the current Newton or time step, and it is therefore able to update the current state completely independently, without any inter–processor communication.

It should be noted that the GMRES algorithm, like all other Krylov methods, does not need to operate on the system matrix by itself, but just needs to compute products of this jacobian matrix with a given vector. One can take advantage of this feature, and develop a matrix—free version of the algorithm [37, 36] in which the matrix–vector products are approximated with a finite difference stencil. This has the advantage of avoiding the storage of the tangent matrix, thus realizing a substantial saving of computer memory at the cost of additional on–processor computations. In the matrix–free version of the algorithm, matrix–vector multiplications of the form $\mathbf{A}(\mathbf{f}) \cdot \mathbf{u}$ are approximated by means of a finite difference of residuals $\mathbf{b}$ as

$$\mathbf{A}(\mathbf{f}) \cdot \mathbf{u} = \frac{\mathbf{b}(\mathbf{f}) - \mathbf{b}(\mathbf{f} + \varepsilon \mathbf{u})}{\varepsilon},$$

where $\mathbf{f}$ is the vector of the field variable nodal values and ε is a perturbation parameter which is computed minimizing the truncation error, which results from truncating the Taylor expansion, and the cancellation error, which is a consequence of operating in finite precision arithmetic.

The addition of preconditioners to the solution strategy is a necessary ingredient for the successful application of

Krylov space solvers. However, they can complicate their parallelization, leading to increased data communications or the need for a global ordering. However, depending upon the underlying problem, local preconditioners may prove adequate to assure convergence in a reasonable number of iterations, or the preconditioner may be calculated one time, stored, and used repeatedly.

5.2. Finite Element Code for Rotorcraft Aerodynamics

This section presents an parallel adaptive procedure for the automated aerodynamic analysis of helicopter rotors based on the procedures discussed in this paper. Adaptive analyses on unstructured discretizations represent an effective and accurate method to address the complex physical phenomena that characterize rotorcraft systems. The problem of the accurate numerical simulation of these phenomena has recently stimulated a vigorous research effort in the scientific community, certainly prompted by the fact that rotor–body interactions, transonic effects, wake effects and blade stall, all have a major impact on the performance, stability and noise characteristics of helicopter rotors.

One of the most important characteristics and distinguishing features of the software presented here is that all the different phases of the analysis, namely the mesh partitioning, the finite element solution, the error indication, the mesh adaptation and the subsequent load balancing, are realized without leaving the parallel environment. In contrast with other procedures that perform only part of the analysis in parallel, as for example just the finite element solution phase, our approach has the advantage of making better use of the power of a distributed memory architecture, leading to an integrated software environment, reducing the i/o and avoiding the bottlenecks that are always present when one tries to solve certain phases of the analysis in serial, especially when very large problems are addressed.

This integrated approach to the parallel adaptive solution of PDE's has lead us to select the message passing paradigm as our method of choice for the parallel programming. This is in contrast with the trend shown by some recent publications [36, 39, 52], where parallel finite element methodologies on fixed meshes have been developed based on data parallel techniques. In fact, we believe that the software development is more easily accomplished in a message passing programming model when one has to deal with adaptive strategies and mesh modification techniques. With the idea of developing a uniform software environment, we have used portable message passing protocols in each stage of the analysis. The implementation has been carried out using the message passing library standard MPI [1] and it has been tested on IBM SP–1 and SP–2 systems.

The procedure developed employs a stabilized finite element formulation which is valid for forward flight and for hovering rotor problems, as well as for general unsteady and steady compressible flow problems. The linear algebra is solved by means of a scalable implementation of the standard and matrix–free GMRES algorithms. Simple techniques are used for estimating regions of high error with the purpose of driving the adaptive procedures.

Techniques to effectively handle the far–field and symmetry boundary conditions for a hovering rotor are considered. Results are presented to demonstrate the ability of the parallel adaptive procedures to solve rotorcraft aerodynamics problems.

Consideration is also given to measures of efficiency and scalability of the parallel adaptive procedures that have been developed. The importance of these measures are demonstrated.

5.2.1 Finite Element Formulation

The initial/boundary value problem can be expressed by means of the Euler equations in quasi–linear form as

$$\mathbf{U}_{,t} + \mathbf{A}_i \cdot \mathbf{U}_{,i} = \mathbf{E}, \qquad (i = 1, \ldots, n_{sd}) \tag{17}$$

plus well posed initial and boundary conditions. In equation (17), n_{sd} is the number of space dimensions, while $\mathbf{U} = \rho(1, u_1, u_2, u_3, e)$ are the conservative variables, $\mathbf{A}_i \cdot \mathbf{U}_{,i} = \mathbf{F}_{i,i}$ where $\mathbf{F}_i = \rho u_i(1, u_1, u_2, u_3, e) + p(0, \delta_{1i}, \delta_{2i}, \delta_{3i}, u_i)$ is the Euler flux, and $\mathbf{E} = \rho(0, b_1, b_2, b_3, b_i u_i + r)$ is the source vector. In the previous expressions, ρ is the density, $\mathbf{u} = (u_1, u_2, u_3)$ is the velocity vector, e is the total energy, p is the pressure, δ_{ij} is the Kronecker delta, $\mathbf{b} = (b_1, b_2, b_3)$ is the body force vector per unit mass and r is the heat supply per unit mass.

The Time–Discontinuous Galerkin Least–Squares finite element method is used in this effort [70, 71]. The TDG/LS is developed starting from the symmetric form of the Euler equations expressed in terms of the entropy variables $\mathbf{V}$ and it is based upon the simultaneous discretization of the space–time computational domain. A least–squares operator and a discontinuity capturing term are added to the formulation for improving stability without sacrificing accuracy. The TDG/LS finite element method takes the form

$$\begin{aligned}
&\int_{Q_n} \left(-\mathbf{W}^h_{,t} \cdot \mathbf{U}(\mathbf{V}^h) - \mathbf{W}^h_{,i} \cdot \mathbf{F}_i(\mathbf{V}^h) + \mathbf{W}^h \cdot \mathbf{E}(\mathbf{V}^h)\right) \mathrm{d}Q \\
&+ \int_{\mathcal{D}(t_{n+1})} \mathbf{W}^{h^-} \cdot \mathbf{U}(\mathbf{V}^{h^-})\, \mathrm{d}\mathcal{D} - \int_{\mathcal{D}(t_n)} \mathbf{W}^{h^+} \cdot \mathbf{U}(\mathbf{V}^{h^-})\, \mathrm{d}\mathcal{D} \\
&+ \int_{P_n} \mathbf{W}^h\, \mathbf{F}_i(\mathbf{V}^h) \cdot \mathrm{d}\mathbf{P} \\
&+ \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \left(\mathcal{L}\mathbf{W}^h\right) \cdot \tau \left(\mathcal{L}\mathbf{V}^h\right) \mathrm{d}Q \\
&+ \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \nu^h \hat{\nabla}_\xi \mathbf{W}^\mathbf{h} \cdot \mathrm{diag}\,[\tilde{\mathbf{A}}_0] \hat{\nabla}_\xi \mathbf{V}^\mathbf{h}\, \mathrm{d}\mathbf{Q} = 0.
\end{aligned} \tag{18}$$

Integration is performed over the space–time slab Q_n, the evolving spatial domain $\mathcal{D}(t)$ of boundary $\Gamma(t)$ and the surface P_n described by $\Gamma(t)$ as it traverses the time interval $I_n =]t_n, t_{n+1}[$. $\mathbf{W}^h$ and $\mathbf{V}^h$ are suitable spaces

for test and trial functions, while τ and ν^h are appropriate stabilization parameters. $\tilde{\mathbf{A}}_0 = \partial \mathbf{U}/\partial \mathbf{V}$ is the metric tensor of the transformation from conservation to entropy variables. Refer to [70, 71] for additional details on the TDG/LS finite element formulation.

Two different three dimensional space–time finite elements have been implemented. The first is based on a constant in time interpolation, and, having low order of time accuracy but good stability properties, it is well suited for solving steady problems using a local time stepping strategy. The second makes use of linear–in–time basis functions and, exhibiting a higher order temporal accuracy, is well suited for addressing unsteady problems, such as, for example, forward flight. In these cases, moving boundaries are handled by means of the space–time deformed element technique [84].

For efficiently solving hover problems a formulation starting from the Euler equations written in a rotating frame is included in the program. This allows treatment of a hovering rotor as a steady problem when the unsteadiness in the wake can be neglected, thus allowing the use of the less computationally expensive constant–in–time formulation.

Assuming that the axis of rotation is coincident with the z axis and that the angular velocity is Ω, the compressible Euler equations in a rotating frame can be expressed in terms of the absolute flow variables $\mathbf{U}$ as

$$\mathbf{U}_{,t} + (\mathbf{A}_i - v_i \mathbf{I}) \cdot \mathbf{U}_{,i} = \mathbf{E} + \mathbf{E}_G, \qquad (19)$$

where $v_1 = -\Omega y$, $v_2 = \Omega x$, $v_3 = 0$ and $\mathbf{E}_G$ can be defined as

$$\mathbf{E_G} = \mathbf{C}\mathbf{U} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \Omega & 0 & 0 \\ 0 & -\Omega & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{U}, \qquad (20)$$

or, in terms of entropy variables, $\mathbf{E}_G = \tilde{\mathbf{C}}\mathbf{V}$, $\tilde{\mathbf{C}} = -\rho T \mathbf{C}$. Clearly, by the nature of the gyroscopic terms, we have that $\mathbf{C}^T = -\mathbf{C}$.

We remark that the rotating frame formulation of the compressible Euler equations in terms of absolute flow variables is formally equivalent to a change of variables (modification of the jacobians $\mathbf{A}_i$ into $\mathbf{A}_i - v_i \mathbf{I}$) plus the introduction of a source term $\mathbf{E}_G$.

From the formulation expressed in equation (19), a TDG/LS finite element formulation can be easily constructed along the lines of equation (18). In an inertial frame, a definition of τ that results in full upwinding on each mode of the system [70] is given by

$$\tau = \tilde{\mathbf{A}}_0^{-1} \left(\hat{\mathbf{A}}_\xi^T \mathrm{diag}(\tilde{\mathbf{A}}_0^{-1}) \hat{\mathbf{A}}_\xi \tilde{\mathbf{A}}_0^{-1} \right)^{-\frac{1}{2}}, \qquad (21)$$

where

$$\hat{\mathbf{A}}_\xi = \left(\frac{\partial \xi_0}{\partial x_0} \tilde{\mathbf{A}}_0, \frac{\partial \xi_1}{\partial x_i} \tilde{\mathbf{A}}_i, \ldots, \frac{\partial \xi_{n_{sd}}}{\partial x_i} \tilde{\mathbf{A}}_i \right)$$

and ξ_i are the local element coordinates, x_0 and ξ_0 referring to the time dimension. In a rotating frame, we redefine $\hat{\mathbf{A}}_\xi$ as

$$\hat{\mathbf{A}}_\xi = \left(\tilde{\mathbf{C}}, \frac{\partial \xi_0}{\partial x_0} \tilde{\mathbf{A}}_0, \frac{\partial \xi_1}{\partial x_i} (\tilde{\mathbf{A}}_i - v_i \tilde{\mathbf{A}}_0), \ldots, \frac{\partial \xi_{n_{sd}}}{\partial x_i} (\tilde{\mathbf{A}}_i - v_i \tilde{\mathbf{A}}_0) \right).$$

Solution to (21) can be obtained based upon the eigenproblem

$$\left(\hat{\mathbf{A}}_\xi^T \mathrm{diag}(\tilde{\mathbf{A}}_0^{-1}) \hat{\mathbf{A}}_\xi - \lambda^2 \tilde{\mathbf{A}}_0^{-1} \right) \cdot \mathbf{T}_i = 0. \qquad (22)$$

The eigenproblem is simplified by means of a similarity transformation $\mathbf{S}$ that diagonalizes $\mathbf{A}_1$ and $\mathbf{A}_2$ and symmetrizes $\mathbf{A}_3$ [86]. However, the term arising from $\mathbf{E}_G$ remains non–symmetric. We have implemented both the non–symmetric and a symmetric form obtained by dropping the contribution of $\mathbf{E}_G$ from (22) and have found that for the hovering rotors that we have studied in our numerical simulations, the symmetric form gives results indistinguishable from those of the non–symmetric form at a lower computational cost.

Discretization of the weak form implied by the TDG/LS method leads to a non–linear discrete problem, which is solved iteratively using a quasi–Newton approach. At each Newton iteration, a non–symmetric linear system of equations is solved using the GMRES algorithm. We have developed scalable parallel implementations of the preconditioned GMRES algorithm and of its matrix–free version [37, 36]. This latter algorithm approximates the matrix–vector products with a finite difference stencil with the advantage of avoiding the storage of the tangent matrix, thus realizing a substantial savings of computer memory at the cost of additional on–processor computations. Preconditioning is achieved by means of a nodal block–diagonal scaling transformation.

In this work we have implemented a simple error indicator based on the norm of the gradient of the flow variables and a slightly more sophisticated one [47] for linear elements which takes the basic form

$$e_i = \frac{h^2 \mid \text{Second Derivative of } \Psi \mid}{h \mid \text{First Derivative of } \Psi \mid + \varepsilon \mid \text{Mean Value of } \Psi \mid},$$

where e_i is the error indicated at node i, h is a mesh size parameter, Ψ is the solution variable being monitored, ε is a tuning parameter. The second derivative of Ψ is computed using a variational recovery technique.

The edge values of the error indicator are computed by averaging the corresponding two nodal values. These edgewise error indicator values are then used for driving the mesh adaptation procedure. Appropriate thresholds are supplied for the error values, so that the edge is refined if the error is higher than the maximum threshold, while the edge is collapsed if the error is less than the minimum threshold.

5.2.2 Boundary Conditions for Hovering Rotors

The imposition of the correct far–field boundary conditions is a critical issue in the analysis of hovering rotors, when one wants to give an accurate representation of the hovering conditions within a finite computational domain. For determining the inflow/outflow far–field conditions we have adopted the methodology suggested by Srinivasan *et al.* [81], where the 1–D helicopter momentum theory is used for determining the outflow velocity due to the rotor wake system. The inflow velocities at the remaining portion of the far–field are determined considering the rotor as a point sink of mass, for achieving conservation of mass and momentum within the computational domain.

Another important condition that must be considered for the efficient simulation of hovering rotors is the periodicity of the flow field. This allows consideration of a reduced computational domain given by the angle of periodicity $\psi = 2\pi/n_b$, n_b being the number of rotor blades.

The introduction of the periodicity conditions in the rotating wing flow solver has been implemented treating them as linear 2–point constraints applied via transformation as part of the assembly process. This approach has the double advantage of being easily parallelizable and of avoiding the introduction of Lagrange multipliers. On the other hand, it requires the mesh discretizations on the two symmetric faces of the computational domain to match on a vertex by vertex basis. Since this is not directly obtainable with the currently used unstructured mesh generator, a mesh matching technique has been developed for appropriately modifying an existing discretization.

In order to simplify the discussion, define one of the symmetric model faces as "master" and the other as "slave". The face discretization of the slave model face is deleted from the mesh, together with all the mesh entities connected to it. The mesh discretization of the master model face is then rotated of the symmetry angle ψ about the axis of rotation and copied onto the slave model face, yielding the required matching face discretizations. The matching procedure is then completed filling the gap between the new discretized slave face and the rest of the mesh using a face removal technique followed by smoothing and mesh optimization.

The imposition of the constraints can be formalized in the following manner. Consider the partition of the unknowns $\mathbf{V}$ in internal ($\mathbf{V}_i$), master ($\mathbf{V}_m$) and slave ($\mathbf{V}_s$), as

$$\mathbf{V} = (\mathbf{V}_i, \mathbf{V}_m, \mathbf{V}_s).$$

The slave unknowns $\mathbf{V}_s$ can be expressed symbolically as functions of the master unknown $\mathbf{V}_m$ as

$$\mathbf{V}_s = \mathbf{G} \cdot \mathbf{V}_m$$

or, for the j–th master–slave pair of nodes as

$$\mathbf{V}_s^j = \mathbf{G}^j \cdot \mathbf{V}_m^j,$$

where

$$\mathbf{G}^j = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}, \tag{23}$$

$\mathbf{R}$ being the rotation tensor associated with the rotation of the symmetry angle ψ about the axis of rotation.

The minimal set of unknowns $\bar{\mathbf{V}} = (\mathbf{V}_i, \mathbf{V}_m)$ is related to the redundant set $\mathbf{V}$ by

$$\mathbf{V} = \mathbf{\Gamma} \cdot \bar{\mathbf{V}} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{G} \end{bmatrix} \cdot \bar{\mathbf{V}}. \tag{24}$$

The unconstrained linearized discrete equations of motion read

$$\mathbf{J} \cdot \Delta\mathbf{V} = \mathbf{r},$$

where $\mathbf{J}$ is the tangent matrix and $\mathbf{r}$ is the residual vector. Applying the transformation $\mathbf{\Gamma}$ to the unconstrained system yields the constrained reduced system

$$\mathbf{\Gamma}^T\mathbf{J}\mathbf{\Gamma} \cdot \Delta\bar{\mathbf{V}} = \mathbf{\Gamma}^T \cdot \mathbf{r}. \tag{25}$$

Refer to [74] for implementation details of this technique.

5.2.3 Subsonic and Transonic Hovering Rotors

Caradonna and Tung [12] have experimentally investigated a model helicopter rotor in several subsonic and transonic hovering conditions. These experimental tests have been extensively used for validating CFD codes for rotating wing analysis. The experimental setup was composed of a two–bladed rotor mounted on a tall column containing the drive shaft. The blades had rectangular planform, square tips and no twist or taper, made use of NACA0012 airfoil sections and had an aspect ratio equal to six.

Figure 61 shows the experimental and numerical values of the pressure coefficients at different span locations for three subsonic test cases investigated by Caradonna and Tung, namely $\theta_c = 0^o$ and $M_t = 0.520$, $\theta_c = 5^o$ and $M_t = 0.434$, $\theta_c = 8^o$ and $M_t = 0.439$. The agreement with the experimental data is good at all locations, including the section close to the tip. Only two pressure distributions are presented for each case for space limitations, however similar correlation with the experimental data was observed at all the available locations. Relatively crude meshes have been employed for all the three test cases, with the coarsest mesh of only 101,000 tetrahedra being used for the $\theta_c = 0^o$ case, and the finest of 152,867 tetrahedra for the $\theta_c = 8^o$ test problem.

The analysis was performed on 32 processing nodes of an IBM SP–2. Reduced integration was used for the interior elements for lowering the computational cost, while full integration was used at the boundary elements for better resolution of the airloads, especially at the trailing edge of the blade. The GMRES algorithm with block–diagonal preconditioning was employed, yielding an average number of GMRES iterations to convergence of about 10. The analysis was advanced in time using one single Newton

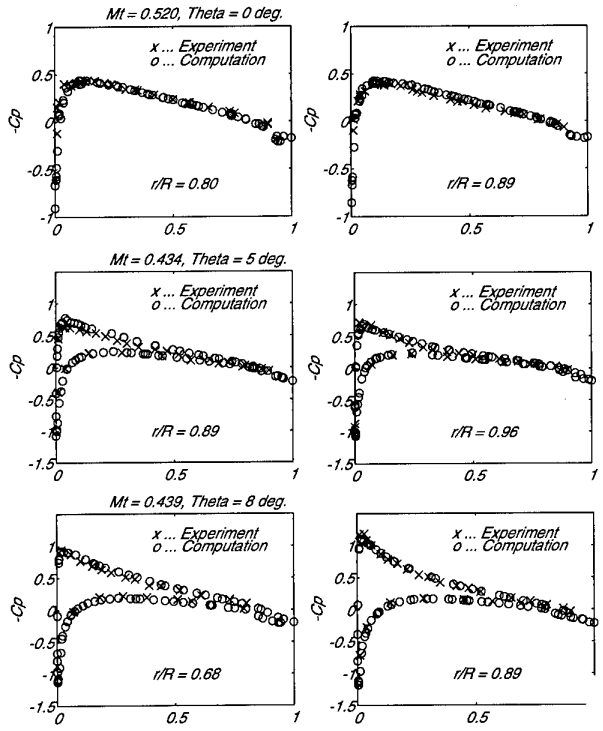


Figure 61. Computed and experimental pressure coefficients on the blade at different span locations, for the three subsonic cases $\theta_c = 0^o$, $M_t = 0.520$; $\theta_c = 5^o$, $M_t = 0.434$; $\theta_c = 8^o$, $M_t = 0.439$.

iteration per time step and a local time stepping strategy denoted by CFL numbers ranging from 10 at the beginning of the simulation to 20 towards convergence yielding a reduction in the energy norm of the residual of almost four orders of magnitude in 50 to 60 time steps The symmetric form of the least–squares stabilization was employed, and the discontinuity capturing operator was not activated.

Figure 62 shows the experimental and numerical values of the pressure coefficients for a transonic case denoted by $\theta_c = 8^o$ and $M_t = 0.877$. The first two plots of Figure 62 present the pressure distributions obtained using an initial crude grid consisting of 142,193 tetrahedra. Three levels of adaptivity were applied to this grid in order to obtain a sharper resolution of the tip shock, yielding a final mesh characterized by 262,556 tetrahedra. The pressure distributions obtained with the adapted grid are shown in the third and fourth plots of the same picture. Note that the smearing present in the first two plots and due to the numerical viscosity introduced in the formulation with the purpose of stabilizing it, has disappeared. Consistently with the nature of the Euler equations, the shocks appear as jumps and are resolved in only one or two elements. Note also the appearance of the analytically predicted overshoot just aft of the shock which is typical of the transonic Euler solutions.

The effect of the adaptation of the mesh on the resolution of the shock is clearly demonstrated in Figure 63, where

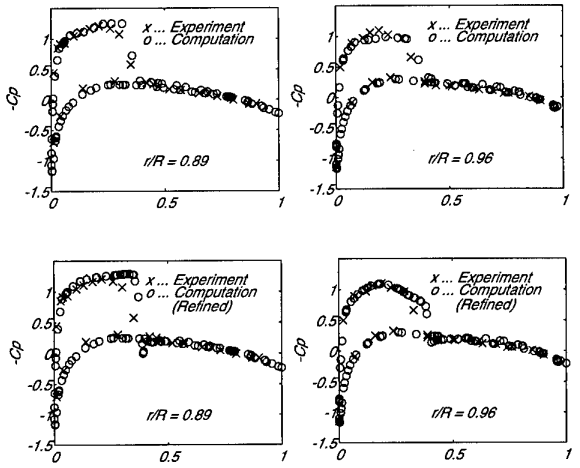


Figure 62. Computed and experimental pressure coefficients on the blade, at two different span locations close to the tip, $\theta_c = 8^o$, $M_t = 0.877$. Top two plots: initial coarse 142,193 tetrahedron grid. Bottom two plots: adapted (three levels) final 262,556 tetrahedron grid.

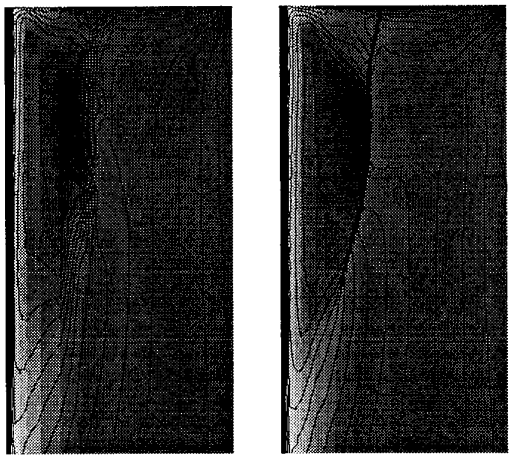


Figure 63. Density isocontour plots on the upper surface of the blade tip, $\theta_c = 8^o$, $M_t = 0.877$. At left: initial coarse grid. At right: final adapted grid.

the density isocontour plots at the upper tip surface are presented for the initial and adapted meshes. The effect noted in Figure 62 can be more fully appreciated here.

The parallel adaptive analysis was conducted on 32 processing nodes with the GMRES algorithm, using once again reduced integration for the interior elements and full integration at the boundary elements. The symmetric form of the least–squares stabilization was employed, together with the discontinuity capturing term for improved shock confinement. After partitioning of the initial coarse mesh using the IRB algorithm, the simulation was performed for 60 implicit time steps with CFL condition equal to 10 in the initial 20 steps and equal to 15 for the remaining steps. The results gathered at convergence were used for computing an error indicator based on den-

sity and Mach number, which was employed for driving the parallel adaptation of the mesh. For the new vertices created by the adaptation process, the solution was projected from the coarser mesh using simple edge interpolation. The solution obtained in this way was used for restarting the analysis, which was advanced for 60 time steps with a CFL number of 15. Similarly, a second adaptation was performed, yielding the final mesh for which another 40 time steps were performed at a CFL of 20, until convergence in the energy norm of the residual. The average number of GMRES cycles per time step throughout the analysis was 8.

Figure 64 shows the mesh at the upper face of the blade tip, before and after refinement. The different grey levels indicate the different subdomains, i.e. elements assigned to the same processing node are denoted by the same level of grey. Note the change in the shape of the partitions from the initial to the final mesh, change generated by the mesh migration procedure for re–balancing the load after the refinement procedure has modified the discretization. Note also how the mesh nicely follows the shock.

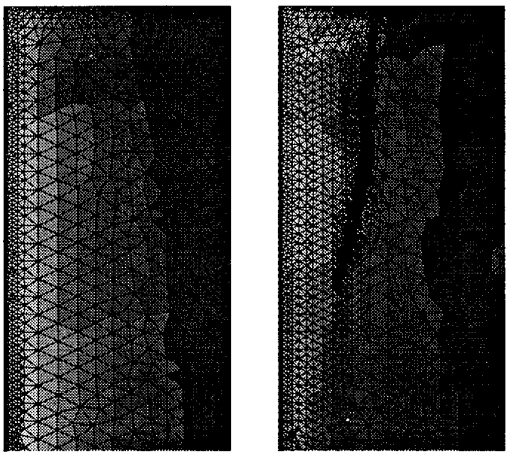


Figure 64. Meshes with partitions on the upper surface of the blade tip, $\theta_c = 8^o$, $M_t = 0.877$. At left: initial coarse grid with IRB partitions. At right: final adapted grid with partitions obtained by migration.

5.3. Effectiveness of Parallel Adaptive Analysis Procedures

The evaluation of the efficiency and performance of a parallel adaptive analysis is a task complicated by the numerous aspects that must be considered. In the following we will try to address at least some of them with the help of a classical problem in CFD, namely that of the Onera M6 wing in transonic flight, that we have used in the early stages of development of our code for validation purposes. This wing has been studied experimentally by Schmitt and Charpin [65] and it has been employed by numerous researchers for validating both structured and unstructured flow solvers. The wing is characterized by an aspect ratio of 3.8, a leading edge sweep angle of 30^o, and a taper ratio of 0.56. The airfoil section is an Onera D symmetric section with 10% maximum thickness–to–cord ratio.

We consider a steady flow problem characterized by an angle of attack $\alpha = 3.06^o$ and a value of $M = 0.8395$ for the freestream Mach number. In such conditions, the flow pattern around the wing is characterized by a complicated double–lambda shock on the upper surface of the wing with two triple points.

We first address the scalability of the parallel solver on a fixed mesh, i.e. we analyze the speed–ups attained by the code using one single mesh and varying the number of processing nodes. This is a classical measure of efficiency, and it is important to show that the implemented procedure performs well with respect to it before measuring other properties that are more pertinent to an adaptive analysis.

The simulation was performed using a mesh consisting of 128,172 tetrahedra, using the matrix–free GMRES algorithm with reduced integration of the interior elements and full integration of the boundary elements. A local time stepping strategy was employed with one single Newton iteration per time step, using a CFL condition of 5 in the first 20 time steps and a CFL equal to 10 for another 80 time steps, attaining a drop in the residual of three orders of magnitude. The mesh was partitioned using a parallel implementation of the IRB algorithm. The time for partitioning, even if small when compared with the time needed for achieving convergence in the finite element analysis, is not considered in the following. The analysis was run on 4, 8, 16, 32, 64, 128 processors of an IBM SP–2 and the results are presented in Figure 65 in terms of the inverse of the wall clock time versus the number of processing nodes. The highly linear behavior of the parallel algorithm shows the excellent characteristics of scalability of the code.

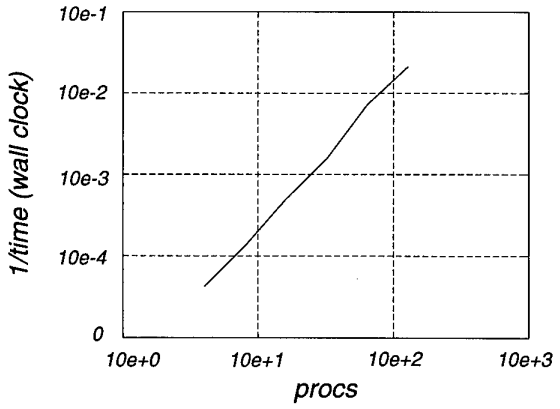


Figure 65. Parallel efficiency evaluated at fixed mesh for the Onera M6 wing in transonic flight. 128,172 tetrahedra, IRB partitions.

The same problem was then adaptively solved in order to more accurately resolve the complicated features of the flow. An initial coarse mesh of 85,567 tetrahedra was partitioned with the IRB algorithm on 32 processing nodes

and the analysis was carried on to convergence as previously explained. The results obtained were then used for computing an error indicator based on density and Mach number, which was employed for performing a first level of refinement, bringing the mesh to 131,000 tetrahedra. The solution was projected on the new vertices using a simple edge interpolation technique, and the analysis was then performed on the refined mesh for 80 time steps at a CFL number of 10. Similarly, other two levels of refinement followed by subsequent analysis were performed, obtaining an intermediate 223,499 tetrahedron mesh and a final 388,837 tetrahedron mesh.

Figure 66 shows the density isocontour plots on the upper surface of the wing corresponding to the initial and the final mesh discretizations. Note that the forward shock is barely visible in the results obtained with the initial coarse mesh, the aft shock presents significant smearing and the lambda shock located at the tip of the wing is not resolved. As expected, considerable improvement in the resolution of the shocks can be observed when mesh adaptation is employed.

Figure 67 shows the initial and final meshes. Once again, elements assigned to the same subdomains are denoted by the same grey level. For the final mesh, the partitions shown are those obtained with the iterative load balancing algorithm.

The fact that the analysis is conducted in parallel doesn't modify the convergence characteristics of a classical h refinement technique, such as the one considered here. However, while in a serial environment essentially only the accuracy of the solution versus the size of the problem and its computational cost enter into the picture, in a parallel environment other factors must be considered. In particular, we consider here the evolution during the analysis of two fundamental parameters: *(i)* the surface–to–volume ratio for the subdomains, and *(ii)* the number of neighbors of each subdomain. The first of these two parameters essentially dominates the volume of communication in terms of the size of the messages to exchange, while the second parameter dominates the number of messages that each processor must send and receive.

In a parallel adaptive environment, the issue is then: given certain repartitioning algorithms, which is the quality of the partitions that they produce compared to their relative cost? It is well known that certain classes of partitioning algorithms, such as the Spectral Bisection method, produce very high quality partitions. However, the cost associated with spectrally bisecting increasingly larger meshes during an adaptive analysis would be prohibitive. Therefore in this work we consider two relatively low cost approaches to the problem, the previously mentioned parallel IRB repartitioning and the iterative load migration scheme.

Two distinct runs were made, the only difference between them being the repartitioning strategy adopted. In both cases, all the stages of the analysis —initial IRB partitioning, flow solution, error sensing, adaptation and load balancing— were performed automatically in parallel on 32 processing nodes, i.e. without ever leaving the parallel environment. The load balancing algorithm was activated three times during the adaptation of each of the meshes, after the refinement, after the snapping of the newly generated vertices to the curved boundaries of the model and after the local retriangulation[5]. At every call, the algorithm was requested to perform only approximately eight migration iterations, yielding a maximum out of balance number of elements per processing node equal to one at the end of each refinement level. This strategy allows better efficiency of the various stages of the adaptive al-

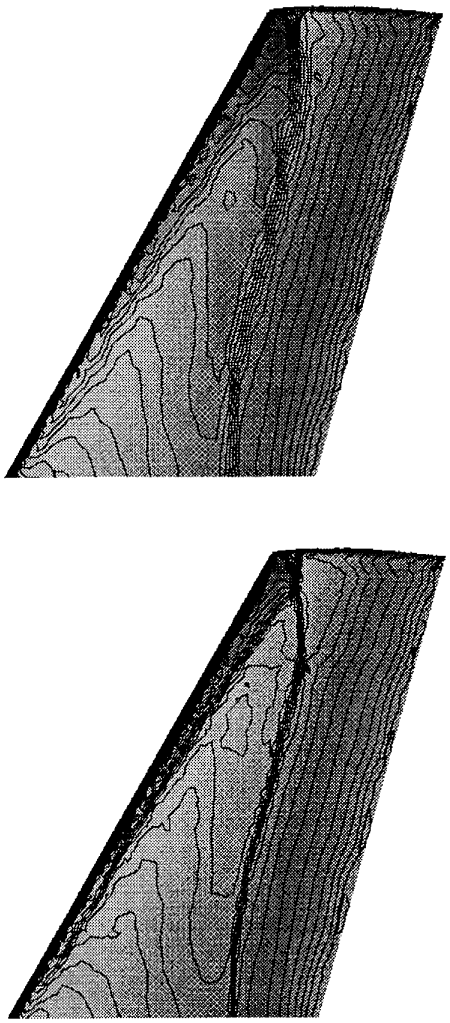


Figure 66. Onera M6 wing in transonic flight, $\alpha = 3.06^o$, $M = 0.8395$. Density isocontour plots for the initial and final meshes.

[5] We remark that in the current implementation, snapping can also cause load imbalance since it makes use of local triangulation.

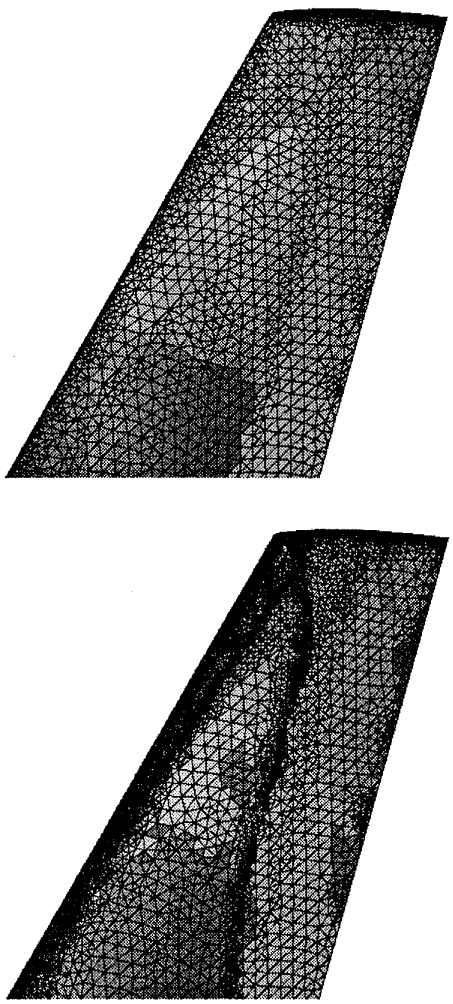


Figure 67. Onera M6 wing in transonic flight, $\alpha = 3.06^o$, $M = 0.8395$. Initial and final meshes. Grey levels indicate processor assignment.

gorithm that can then operate on balanced or nearly balanced meshes. This "incremental" rebalancing capability represents a nice advantage of the iterative load balancing scheme over other algorithms. The parallel repartitioning algorithm was instead activated just once at the end of each adaptive step.

The meshes obtained during the two previously mentioned parallel adaptive simulations of the Onera M6 wing were analyzed for gathering data on the overall performance of the analysis. Figure 68 reports plots of the boundary faces and neighbor statistics. The quantities plotted are defined as:

(i) Surface–to–volume measures:

$$S_{\max} = \max_i (\text{Boundary Faces}_i/\text{Faces}_i), .$$

$$S_{\text{glob}} = \text{Boundary Faces/Faces}.$$

(ii) Neighbor measures:

$$N_{\max} = \max_i (\text{Neighbors}_i/(\text{Procs} - 1)),$$

$$N_{\text{avrg}} = (\sum_i \text{Neighbors}_i/(\text{Procs} - 1))/\text{Procs}.$$

All these quantities are reported in Figure 68 versus the number of tetrahedra in the mesh at a certain adaptive level normalized by the number of tetrahedra in the initial mesh. The solid line represents the values of the parameters obtained for the parallel adaptive analysis where the iterative mesh migration procedures were employed. The dashed line corresponds to the parallel adaptive analysis where the refined meshes were repartitioned after each adaptive step using the parallel IRB algorithm.

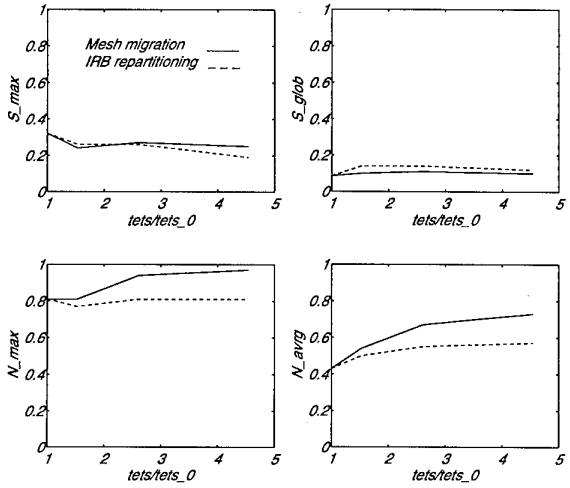


Figure 68. Boundary faces and neighbor statistics for the parallel–adaptive analysis of the Onera M6 wing in transonic flight using the mesh migration and IRB rebalancing schemes.

From the analysis of the first two plots at the top of Figure 68, it is clear that the migration procedures implemented in this work control very effectively the surface–to–volume ratios, which in fact remain constant and fairly similar to the ones obtained with the IRB partitioning for the whole simulation. On the other hand, the second two plots of the same figure show that the number of neighbors of each subdomain tends to increase with the number of adaptive steps performed. A more detailed analysis shows that in general each subdomain is connected by a significant amount of mesh entities (vertices, faces, edges) only with a reduced number of neighbors, while it shares a very limited number of mesh entities with the other neighbors. We are currently investigating ways of removing such small contact area interconnections, in order to achieve a better control on the number of neighbors.

The different partition statistics provided by the two rebalancing algorithms and shown in the previous figure

clearly have an impact on the performance of the flow solver. For example, the ratio of the wall clock timings for the flow solutions performed on the final adapted mesh was found to be 0.83, in favor of the repartitioning algorithm. It should be pointed out that this is not an objective measure of efficiency of the rebalancing strategy, in the sense that it depends on the algorithm used for the flow solution. On the contrary, $S_{\max}$, S_{glob}, $N_{\max}$ and N_{avrg} are objective measures.

The two approaches were also compared in terms of relative wall clock timing cost. The repartitioning algorithm outperformed the migration scheme at each adaptive step. The ratio of the iterative migration wall clock time to the rebalancing wall clock time was found to be 4.07 at the first level (131,000 tetrahedron mesh), 4.41 at the second (223,l499 tetrahedron mesh) and 2.21 at the third (388,837 tetrahedron mesh).

These preliminary test results seem to indicate that the iterative load migration scheme tends to be more computationally expensive than the parallel IRB algorithm, and at the same time does not yield the same quality of the partitions, at least with the currently implemented heuristics. However, it must not be forgotten that these tests are certainly not as exhaustive as one might desire for ruling in favor of one approach over the other. Moreover, it is clear that this result is partially due to the low cost of the IRB partitioning, and comparing the migration scheme with other more expensive partitioning algorithms might lead to opposite conclusions. For example, if an algorithm with better control over the number of neighbors could be devised, then the migration scheme used in conjunction with a high quality initial partition (such as the one provided by a spectral partitioning) could yield an overall better performance than a repartitioning scheme. A more complete analysis of the relative merits of the two approaches will be the subject of future work.

6. Closing Remarks

This paper has presented progress made to date on the development of parallel automated adaptive analysis procedures for unstructured meshes which operate on distributed memory MIMD computers. The procedures presented allow for the reliable analysis, through the use of automated adaptive analysis, of large problems which can only be supported by the computational power of parallel computers. Specific emphasis was placed on the techniques needed to effectively support evolving meshes such that computational load balance was maintained throughout the simulation process.

7. Acknowledgment

The authors would like to acknowledge the support of NASA Ames Research Center under grants NAG 2–832 and NCC 2–9000, the Army Research Office under grant DAAH04–93–G-0003, and the National Science Foundation under grant DMS-93–18184.

We would also like to acknowledge the input of several others that contributed to the material presented in this document. They are Saikat Dey, Pascal Frey, Rao Garimella, Marcel Georges, Ramamoorthy Ravichandran and Wes Turner.

8. References

[1] Message passing interface forum, document for a standard message-passing interface. Technical Report CS-93-214, University of Tennessee, November 1993.

[2] R. E. Bank and A. H. Sherman. An adaptive multilevel method for elliptic boundary value problems. *Computing*, 26:91–105, 1981.

[3] E. Bänsch. Local mesh refinement in 2 and 3 dimensions. *Impact of Comp. in Sc. and Engng.*, 3:181–191, 1991.

[4] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, C-36(5):570–580, May 1987.

[5] J. Bey. *Analyse und Simulation eines Konjugierte Gradienten Verfahrens mit einem Multilevel Präkonditionierer zur Lösung Dreidimensionaler Elliptischer Randwertprobleme für Massiv Parallel Rechner*. PhD thesis, RWTH, Aachen, 1991.

[6] R. Biswas and R. Strawn. A new procedure for dynamic adaptation of three-dimensional unstructured grids. In *31st Aero. Sci. Meet.*, 1993.

[7] G. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38:1526–1538, 1989.

[8] G. Blelloch, C. Leiserson, B. Maggs, C. Plaxton, S. Smith, and M. Zagha. A comparison of sorting algorithms for the connection machine cm-2. *ACM*, pages 3–16, 1991.

[9] F. Bornemann, B. Erdmann, and R. Kornhuber. Adaptive multilevel methods in three space dimensions. *Int. J. Numer. Meth. Engng.*, 36:3187–3203, 1993.

[10] C. L. Bottasso, H. L. de Cougny, M. Dindar, J. E. Flaherty, C. Ozturan, Z. Rusak, and M. S. Shephard. Compressible aerodynamics using a parallel adaptive time-discontinuous Galerkin least-squares finite element method. In *12th AIAA Applied Aerodynamics Conference*, number 94-1888, Colorado Springs, CO, June 20-22, 1994. American Institute for Aeronautics and Astronautics.

[11] D. Callahan and K. Kennedy. Compiling programs for distributed-memory multiprocessors. *Journal of Supercomputing*, 2:151–169, October 1988.

[12] F. Caradonna and C. Tung. Experimental and analytical studies of a model helicopter rotor in hover. Technical Report USAAVRADCOM TR-81-A-23, 1981.

[13] S. D. Connell and D. G. Holmes. 3-dimensional unstructured adaptive multigrid scheme for the euler equations. *AIAA J.*, 32:1626–1632, 1994.

[14] H. L. de Cougny. Automatic generation of geometric triangulations based on octree/Delaunay techniques. Master's thesis, Civil and Environmental Engineering, Scientific Computation Research Center, Rensselaer Polytechnic Institute,Troy, NY 12180-3590, May 1992. SCOREC Report # 6-1992.

[15] H. L. de Cougny, K. D. Devine, J. E. Flaherty, R. M. Loy, C. Ozturan, and M. S. Shephard. Load balancing for the parallel solution of partial differential equations. *Applied Numerical Mathematics*, 16:157–182, 1994.

[16] H. L. de Cougny, M. S. Shephard, and C. Ozturan. Parallel three-dimensional mesh generation on distributed memory mimd computers. *Engineering with Computers*, 1995. submitted.

[17] B. E. de l'Isle and P. L. George. Optimization of tetrahedral meshes. INRIA, Domaine de Voluceau, Rocquencourt BP 105 Le Chesnay France, 1993.

[18] K. M. Devine. *An adaptive HP-finite element method with dynamic load balancing for the solution of hyperbolic conservation laws on massively parallel computers*. PhD thesis, Computer Science Dept, Rensselaer Polytechnic Institute, Troy, New York, 1994.

[19] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. In *8th Annual Comp. Geometry*, pages 6–43, 1992.

[20] C. Farhat. A simple and efficient automatic FEM domain decomposer. *Computers and Structures*, 28:579–602, 1988.

[21] C. Farhat and M. Lesoinne. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *Int. J. Numer. Meth. Engng.*, 36:745–764, 1993.

[22] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Math. J.*, 23:298–305, 1973.

[23] D. A. Field. Laplacian smoothing and Delaunay triangulations. *Comm. Appl. Num. Meth.*, 4:709–712, 1987.

[24] G. C. Fox and W. Furmanski. Communication algorithms for regular convolutions and matrix problems on the hypercube. In M. T. Heath, editor, *Conference on Hypercube Multiprocessors*, pages 223–238, Philadelphia, 1986. SIAM.

[25] W. H. Frey and D. A. Field. Mesh relaxation: A new technique for improving triangulations. *Int. J. Numer. Meth. Engng.*, 31:1121–1133, 1991.

[26] P. L. George. *Automatic Mesh Generation*. John Wiley and Sons, Ltd, Chichester, 1991.

[27] P. L. George. Generation de maillages par une methode de type voronoi partie 2: Le cas tridimensionnel. Technical Report INRIA 1664, INRIA, Domaine de Voluceau, Rocquencourt BP 105 Le Chesnay France, 1992.

[28] P. L. George, F. Hecht, and E. Saltel. Fully automatic mesh generator for 3d domains of any shape. *Impact of Comp. in Sc. and Engng.*, 2:187–218, 1990.

[29] M. E. Go Ong. *Hierarchical Basis Preconditioners for Second Order Elliptic Problems in Three Dimensions*. PhD thesis, Univ. of California, Los Angeles CA, 1989.

[30] N. Golias and T. Tsiboukis. An approach to refining three-dimensional tetrahedral meshes based on delaunay transformations. *Int. J. Numer. Meth. Engng.*, 37:793–812, 1994.

[31] W. Gropp. Simplified linear equation solvers users manual. Technical Report ANL-98/8-REV 1, Mathematics and Computer Science Division, Argonne National Laboratory, 1993.

[32] E. L. Gursoz, Y. Choi, and F. B. Prinz. Vertex-based representation of non-manifold boundaries. In M. J. Wozny, J. U. Turner, and K. Priess, editors, *Geometric Modeling Product Engineering*, pages 107–130. North Holland, 1990.

[33] S. W. Hammond. *Mapping Unstructured Grid Computations to Massively Parallel Computers*. PhD thesis, Computer Science Dept., Rensselaer Polytechnic Institue, Troy, 1991.

[34] J. JaJa. *An introduction to Parallel Algorithms*. Addison Wesley, Reading Mass., 1992.

[35] B. Joe. Three-dimensional triangulations from local transformations. *SIAM J. Sci. Stat. Comp.*, 10(4):718–741, 1989.

[36] Z. Johan. *Data Parallel Finite Element Techniques for Large-Scale Computational Fluid Dynamics*. PhD thesis, Stanford University, July 1992.

[37] Z. Johan, T. J. R. Hughes, K. K. Mathur, and S. L. Johnsson. A data parallel finite element method for computational fluid dynamics on the connection machine system. *Comp. Meth. Appl. Mech. Engng.*, 99:113–134, 1992.

[38] Y. Kallinderis and P. Vijayan. Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes. *AIAA J.*, 31(8):1440–1447, August 1993.

[39] J. G. Kennedy, M. Behr, V. Kalro, and T. E. Tezduyar. Implementation of implicit finite element methods for incompressible flows on the cm-5. In *Army High-Performance Computing Research Center*, number 94-017, University of Minnesota, 1994.

[40] B. W. Kernigham and D. M. Ritchie. *The C programming Language*. Prentice Hall, Inc., 1990.

[41] C. P. Kruskal, L. Rudolph, and M. Snir. Efficient parallel algorithms for graph problems. *Algorithmica*, 5:43–64, 1990.

[42] C. L. Lawson. Properties of n-dimensional triangulations. *Computer Aided Geometric Design*, 3(4):231–246, 1986.

[43] E. Leiss and H. Reddy. Distributed load balancing: Design and performance analysis. Technical Report Vol. 5, W. M. Keck Research Computation Laboratory, 1989.

[44] A. Liu and B. Joe. Quality local refinement of tetrahedral meshes based on bisection. in press, February 1994.

[45] A. Liu and B. Joe. On the shape of tetrahedra from bisection. *Math. of Comp.*, 63(207):141–154, July 1994.

[46] S. H. Lo. 3d mesh refinement in compliance with a specified node spacing function. Civil Engng. Dept., The U. of Hong Kong, submitted to IJNME - 1995.

[47] R. Löhner. An adaptive finite element scheme for transient problems in cfd. *Comp. Meth. Appl. Mech. Engng.*, 61:323–338, 1987.

[48] R. Löhner and J. D. Baum. Numerical simulation of shock interaction with complex geometry three-dimensional structures using a new adaptive h-refinement scheme on unstructured grids. In *28th Aero. Sci. Meet.*, 1990.

[49] R. Löhner, J. Camberos, and M. Merriam. Parallel unstructured grid generation. *Comp. Meth. Appl. Mech. Engng.*, 95:343–357, 1992.

[50] R. Löhner and R. Ramamurti. A parallelizable load balancing algorithm. In *Proc. of the AIAA 31st Aerospace Sciences Meeting and Exhibit*, 1993.

[51] K. Mathur and S. L. Johnsson. Data structures and algorithms for the finite element method on a data parallel supercomputer. *International Journal of Numerical Methods in Engineering*, 29:881–908, 1990.

[52] K. K. Mathur and S. L. Johnson. The finite element method on a data parallel computing system. *Int. J. High Speed Comp.*, 1:29–44, 1989.

[53] S. N. Muthukrishnan, P. S. Shiakolas, R. V. Nambiar, and K. L. Lawrence. A simple algorithm for the adaptive refinement of three dimensional problems with tetrahedral meshes. Mech. Engng. Dept., The Univ. of Texas at Arlington, Arlington, TX 76019, 1993.

[54] C. Ozturan. *Distributed Environment and Load Balancing for Adaptive Unstructured Meshes*. PhD thesis, Rensselaer Polytechnic Institute, Troy NY, August 1995.

[55] C. Ozturan, H. L. de Cougny, M. S. Shephard, and J. E. Flaherty. Parallel adaptive mesh refinement and redistribution on distributed memory machines. *Comp. Meth. Appl. Mech. Engng.*, 119:123–137, 1994.

[56] A. Pothen, H. Simon, and K. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, July 1990.

[57] R. D. Rausch, J. T. Batina, and H. T. Y. Yang. Spatial adaptation procedures on tetrahedral meshes for unsteady aerodynamic flow calculations. In *31st Aero. Sci. Meet.*, 1993.

[58] A. A. G. Requicha and H. B. Voelcker. Solid modeling: Current status and research directions. *IEEE Computer Graphics and Applications*, 3(7):25–37, 1983.

[59] M.-C. Rivara. Selective refinement/derefinement algorithms for sequences of nested triangulations. *Int. J. Numer. Meth. Engng.*, 28:2889–2906, 1989.

[60] M.-C. Rivara. A 3-D refinement algorithm suitable for adaptive and multi-grid techniques. *Communications in Applied Numerical Methods*, 8:281–290, 1992.

[61] I. G. Rosenberg and F. Stenger. A lower bound on the angles of triangles constructed by bisecting the longest side. *Math. of Computation*, 29(130):390–395, April 1975.

[62] Y. Saad and M. Schultz. A generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sc. Stat. Comp.*, 7:856–869, 1986.

[63] J. Saltz, R. Crowley, R. Mirchandaney, and H. Berryman. Run-time scheduling and execution of loops on message passing machines. *Journal of Parallel and Distributed Computing*, 8(2):303–312, 1990.

[64] M. Saxena and R. Perucchio. Parallel fem algorithms based on recursive spatial decompositions - I. automatic mesh generation. *Computers and Structures*, 45:817–831, 1992.

[65] V. Schmitt and F. Charpin. Pressure distributions on the onera m6 wing at transonic mach numbers. Technical Report R-702, AGARD, 1982.

[66] W. J. Schroeder and M. S. Shephard. A combined octree/Delaunay method for fully automatic 3-D mesh generation. *Int. J. Numer. Meth. Engng.*, 29:37–55, 1990.

[67] W. J. Schroeder and M. S. Shephard. On rigorous conditions for automatically generated finite element meshes. In J. Turner, J. Pegna, and M. Wozny, editors, *Product Modeling for Computer-Aided Design and Manufacturing*, pages 267–281. North Holland, 1991.

[68] R. Sedgewick. *Algorithms in C*. Addison-Wesley Publishing Company, 1990.

[69] E. G. Sewell. *Automatic Generation of Triangulations for Piecewise Polynomial Approximation*. PhD thesis, Purdue Univ., West Lafayette IN, 1972.

[70] F. Shakib. *Finite Element Analysis of The Compressible Euler and Navier-Stokes Equations*. PhD thesis, Stanford University, 1988.

[71] F. Shakib, T. J. R. Hughes, and Z. Johan. A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier Stokes equations. *Comp. Meth. Appl. Mech. Engng.*, 89:141–219, 1991.

[72] M. S. Shephard. The specification of physical attribute information for engineering analysis. *Engineering with Computers*, 4:145–155, 1988.

[73] M. S. Shephard, C. L. Bottasso, H. L. de Cougny, and C. Ozturan. Parallel adaptive finite element analysis of fluid flows on distributed memory computers. In *Recent Developments in Finite Element Analysis*, pages 205–214. Int. Center for Num. Meth. in Engng., Barcelona, Spain, 1994.

[74] M. S. Shephard, S. Dey, and M. K. Georges. Automatic meshing of curved three-dimensional domains: Curving finite elements and curvature-based mesh control. In *Proceedings of the IMA Summer Program Modeling Mesh Generation and Adaptive Numerical Methods for Partial Differential Equations, July 6-23, 1993*. Springer Verlag, 1994.

[75] M. S. Shephard and P. M. Finnigan. Toward automatic model generation. In A. K. Noor and J. T. Oden, editors, *State-of-the-Art Surveys on Computational Mechanics*, pages 335–366. ASME, 1989.

[76] M. S. Shephard and M. K. Georges. Automatic three-dimensional mesh generation by the Finite Octree technique. *Int. J. Numer. Meth. Engng.*, 32(4):709–749, 1991.

[77] M. S. Shephard and M. K. Georges. Reliability of automatic 3-D mesh generation. *Comp. Meth. Appl. Mech. Engng.*, 101:443–462, 1992.

[78] M. S. Shephard and N. P. Weatherill, editors. *Int. J. Numer. Meth. Engng.*, volume 32. Wiley-Interscience, Chichester, England, 1991.

[79] A. Shostko and R. Löhner. Three-dimensional parallel unstructured grid generation. *Int. J. Numer. Meth. Engng.*, 38:905–925, 1995.

[80] H. D. Simon. Partitioning of unstructured meshes for parallel processing. *Comput. Sys. Engng.*, 2:135–148, 1991.

[81] G. Srinivasan, V. Raghavan, and E. Duque. Flowfield analysis of modern helicopter rotors in hover by Navier-Stokes method. In *International Technical Specialist Meeting on Rotorcraft Acoustics and Rotor Fluid Dynamics*, Philadelphia, PA, 1991.

[82] M. Stynes. On faster convergence of the bisection method for all triangles. *Math. of Computation*, 35(152):1195–1201, October 1980.

[83] B. K. Szymanski and A. Minczuk. A representation of a distribution power network graph. *Archiwum Elektrotechniki*, 27(2):367–380, 1978.

[84] T. E. Tezduyar, M. Behr, S. Mittal, and J. Liou. A new strategy for finite element computations involving moving boundaries and interfaces - the deforming-spatial-domain/space time procedure: I. the concept and preliminary tests. *Comp. Meth. Appl. Mech. Engng.*, 94:339–351, 1992.

[85] A. Vidwans, Y. Kallinderis, and Venkatakrishnan. Parallel dynamic load-balancing algorithm for three-dimensional adaptive unstructured grids. *AIAA Journal*, 32(3):497–505, March 1994.

[86] R. F. Warming, R. M. Beam, and B. J. Hyett. Diagonalization and simultaneous symmetrization of the gas-dynamic matrices. *Math. of Comp*, 29:1037–1045, 1975.

[87] D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer J.*, 24(2), 1981.

[88] N. P. Weatherill and O. Hassan. Efficient three-dimensional delaunay triangulation with automatic point creation and imposed boundary constraints. *Int. J. Numer. Meth. Engng.*, 37:2005–2039, 1994.

[89] K. J. Weiler. The radial-edge structure: A topological representation for non-manifold geometric boundary representations. In M. J. Wozny, H. W. McLaughlin, and J. L. Encarnacao, editors, *Geometric Modeling for CAD Applications*, pages 3–36. North Holland, 1988.

[90] R. Williams. *DIME: Distributed Irregular Mesh Environment*. Supercomputing Facility, California Institute of Technology, 1990.

[91] R. D. Williams. Performance of dynamic load balancing algorithms for unstructured grid calculations. Technical Report C3P913, Pasadena, 1990.

[92] R. D. Williams. Voxel databases: A paradigm for parallelism with spatial structure. *Concurrency*, 4:619–636, 1992.

[93] R. D. Williams. Dime++: A parallel language for indirect addressing. Technical Report CCSF-34, Caltech Concurrent Supercomputing Facilities, Pasadena, June 1993.

[94] M. A. Yerry and M. S. Shephard. Automatic three-dimensional mesh generation by the modified-octree technique. *Int. J. Numer. Meth. Engng.*, 20:1965–1990, 1984.

[95] H. Zima and B. M. Chapman. Compiling for distributed memory systems. Technical Report ACPC/TR 92-17, Austrian Center for Parallel Computation, University of Vienna, 1992.

Parallel CFD Algorithms on Unstructured Meshes

Timothy J. Barth[1]
Advanced Algorithms and Applications Branch
NASA Ames Research Center
Moffett Field, CA 94035
USA

Introduction and Overview

The intent of these notes is to review several basic algorithms and procedures used in computational fluid dynamics (CFD) with emphasis on algorithms suitable to parallel computing environments. In particular, we will concentrate on numerical methods in CFD which require the formation and solution of large sparse linear systems of algebraic equations. These matrices will arise from the discretization of the Navier-Stokes equations which govern compressible fluid flow. From this point of view, a large portion of these notes addresses algorithms used in the formation, manipulation, and solution of sparse matrices on serial and parallel computers.

Chapter 1 begins by considering the task of ordering (numbering) vertices of an unstructured mesh. Good vertex orderings can greatly improve the efficiency and memory storage required in many sparse matrix algorithms. For example, techniques for iterative matrix solution sometimes exploit incomplete matrix factorizations. The quality of these factorizations usually depends on the ordering of matrix unknowns and consequently mesh vertices. Next, we review the mesh partitioning problem. Three simple procedures for decomposing an arbitrary triangulated domain into a specified number of subdomains are discussed. Each subdomain may then be placed on an individual processor of the parallel computer. Communication between processors is accomplished using message packet exchanges. This computational model places demands on the partitioning algorithms so that computational work is evenly distributed (balanced) while requiring minimal communication among processors.

In Chapter 2 we turn to the compressible Navier-Stokes equations. These equations represent conservation principles for mass, momenta, and energy of a Newtonian fluid. In high speed aerodynamic applications, the effects of turbulence are very important and must either be accurately computed or approximately modeled. This increases the difficulty and complexity of solving the Navier-Stokes equations. In the present applications, a one-equation turbulence model equation is added to the basic time-averaged Navier-Stokes equations. The resulting system of coupled integral equations are discretized using a finite-volume technique based on linear least squares reconstruction. This yields a system of nonlinear coupled algebraic equations which are solved via Newton iteration. The most difficult task in Newton's method is the solution of the resulting sequence of large sparse linear matrix problems. Iterative methods based on preconditioned bi-conjugate gradient and generalized minimum residual iterations are considered. Numerical examples are then shown to demonstrate the convergence characteristics of the uniprocessor algorithm.

Chapter 3 focuses on domain decomposed variants of the uniprocessor CFD algorithm given in Chapter 2. As a starting point, the Schwarz domain decomposition algorithm for elliptic equations is reviewed. This technique requires the isolated solution of subdomain problems. Next we derive the well-known relationship between convergence rate of the Schwarz algorithm and overlap of subdomains. This analysis reveals that special care must be taken to insure that the domain decomposition procedure does not become ill-conditioned as the number of subdomains is increased. The Schwarz algorithm can also be applied to the solution of nonelliptic equations. Computations of inviscid and viscous fluid flow are shown to demonstrate the favorable effect of increasing subdomain overlap on convergence of the Schwarz algorithm. In these computations, each subdomain is independently solved

Paper presented in an AGARD-FDP-VKI Special Course on "Parallel Computing in CFD", held at the VKI, Rhode-Saint-Genèse, Belgium, from 15-19 May 1995 and 16-20 October 1995 at NASA Ames, United States and published in R-807.

using the Newton algorithm given in Chapter 2. An alternative to the conventional domain decomposition procedure is the Newton-Krylov technique with the overlapping Schwarz method used to precondition the underlying global matrix problems. Inviscid and viscous computations are shown to demonstrate the efficiency of this method.

Finally, Chapter 4 presents some selected computations performed on the IBM SP2 parallel computer located at NASA Ames.

Chapter 1

Graph Ordering and Partitioning Algorithms for CFD

In this section we review a few basic graph algorithms which are frequently used in numerical computations performed on parallel computers.

1.1 Graph Ordering

The particular ordering of mesh (graph) vertices can sometimes alter the amount of computational effort and memory storage required in solving sparse matrix problems. In sparse matrix $L-U$ factorization, the number of fill elements produced during factorization is dependent on the ordering of equations. Good ordering algorithms attempt to reduce the number of fill elements produced during factorization. Similarly, the quality of inexact factorizations such as incomplete Cholesky and incomplete $L-U$ factorization also depends on the ordering of matrix unknowns. Reordering vertices can also lead to improved processor efficiency by reducing "cache misses", an important consideration for computations performed on workstation class computers. In parallel computation, ordering algorithms can be used as means for partitioning a mesh among processors of the computer. This last consideration will be addressed in a later section.

In this section we review the Cuthill-McKee [CM69] ordering algorithm. This popular algorithm is simple yet surprisingly effective. Other popular ordering strategies which deserve attention but are not discussed here include the minimum degree algorithm [GL81] and Rosen's algorithm [Ros68] for bandwidth reduction. We begin a discussion of the Cuthill-McKee algorithm by simply stating the procedure.

Algorithm: Graph ordering, Cuthill-McKee.

Step 1. Find vertex with lowest degree. This is the *root* vertex.

Step 2. Find all neighboring vertices connecting to the root by incident edges. Order them by increasing vertex degree. This forms level 1.

Step 3. Form level k by finding all neighboring vertices of level $k-1$ which have not been previously ordered. Order these new vertices by increasing vertex degree.

Step 4. If vertices remain, go to step 3.

The heuristics behind the Cuthill-McKee algorithm are very simple. In the graph of a matrix, neighboring vertices must have numberings which are near by, otherwise they will produce entries in the matrix with large band width. The idea of sorting elements among a given level is based on the heuristic that vertices with high degree should be given indices as large as possible so that they will be as close as possible to vertices of the *next* level generated. Figures 1.1 and 1.2 show the dramatic improvement in matrix bandwidth achieved using the Cuthill-McKee algorithm.

Studies of the Cuthill-McKee algorithm have shown that the fill characteristics of a matrix during $L-U$ decomposition can be greatly reduced by reversing the ordering of the Cuthill-McKee algorithm, see George [Geo71]. This

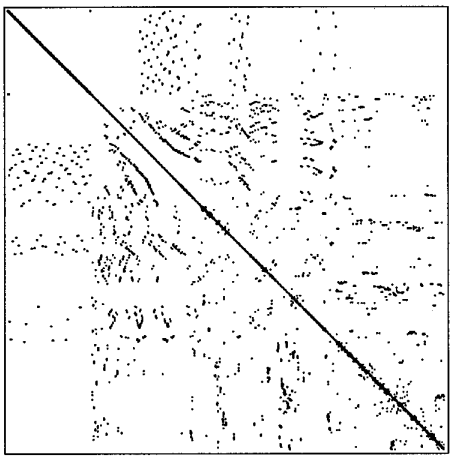


Figure 1.1: Nonzero matrix elements produced by a Laplacian discretization (left) on the triangulated domain shown in Figure 1.4.

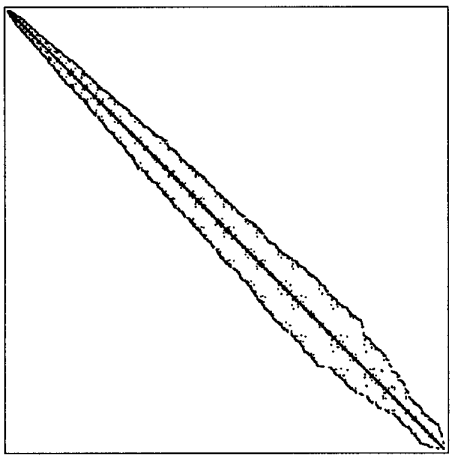


Figure 1.2: Nonzero matrix elements after Cuthill-McKee reordering (right).

amounts to a renumbering given by

$$k \rightarrow n - k + 1$$

where n is the size of the matrix. While this does not change the bandwidth of the matrix, it can dramatically reduce the fill that occurs in Cholesky or $L - U$ matrix factorization when compared to the original Cuthill-McKee ordering.

1.2 Graph Bisection and Mesh Partitioning

An efficient partitioning of a mesh for distributed memory computation is one that ensures an even distribution of computational workload among the processors and minimizes the amount of time spent in interprocessor communications. The former requirement is termed *load balancing.* For if the load were not evenly distributed, some processors will have to sit idle at synchronization points waiting for other processors to catch up. The second requirement comes from the fact that communication between processors takes time and it is not always possible to hide this latency in data transfer. The actual cost of communication can often be accurately modeled by the linear relationship:

$$Cost = \alpha + \beta m$$

where α is the time required to initiate a message, β is the rate of data-transfer between two processors and m is the message length. For n messages, the cost would be

$$Cost = \sum_{n} (\alpha + \beta m_n).$$

This cost can be reduced in two ways: (1) reduce the number of messages n, (2) reduce the size of each message m. Consider the partitioning shown in Figure 1.3. The left figure requires 3 pairwise communication messages of length 5 while the right figure requires 4 pairwise messages of length 2 and 2 pairwise messages of length 1. The choice of partitioning depends critically on the hardware dependent constants α and β.

In practice, it is difficult to partition an unstructured mesh while simultaneously minimizing the number and length of messages. In the

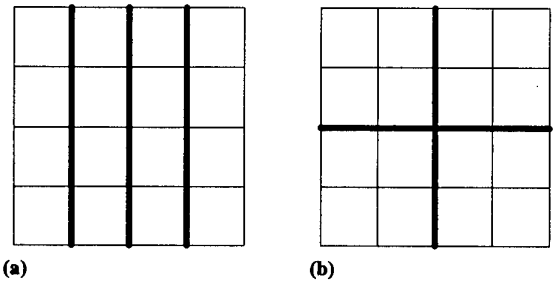


Figure 1.3: (a) Mesh partitioning with minimized number of messages, (b) Mesh with minimized message length.

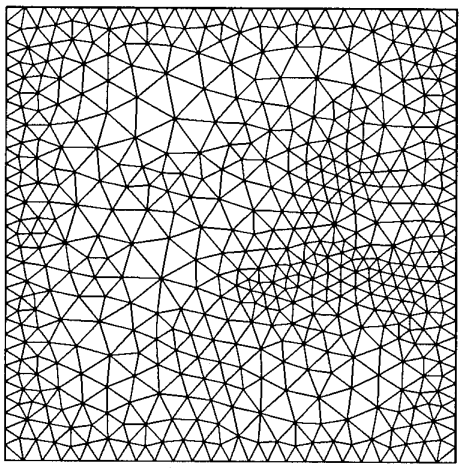


Figure 1.4: Typical triangulation for a square-shaped domain.

following paragraphs, a few of the most popular partitioning algorithms which approximately accomplish this task will be discussed. All the algorithms discussed below: coordinate bisection, Cuthill-McKee, and spectral partitioning are discussed in the paper by Venkatakrishnan, Simon, and Barth [VSB92]. This paper evaluates the partitioning techniques within the confines of an explicit, unstructured finite-volume Euler solver. Spectral partitioning has been extensively studied by Simon [Sim91] for other applications. Although we restrict our discussion to partitioning planar triangulations, all of the algorithms discussed below extend naturally to arbitrary cell complexes and higher space dimensions.

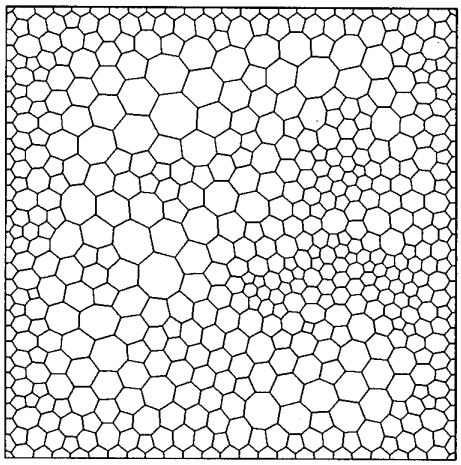


Figure 1.5: Geometric dual of previous triangulation for a square-shaped domain.

In the following sections, we consider mesh partitioning via recursive application of graph bisection. The mesh is first divided into two sub-meshes of nearly equal size. Each of these sub-meshes is subdivided into two more sub-meshes and the process in repeated until the desired number of partitions p is obtained (p is a integer power of 2). In many applications it makes sense to partition mesh cells such that partition boundaries correspond to edges of the mesh. This can be viewed as a vertex partitioning of the graph dual to the cell complex, see for example Figures 1.4 and 1.5. In this way, dual graph vertices are associated with mesh cells.

1.2.1 Recursive Coordinate Bisection

In the coordinate bisection algorithm, graph vertex *coordinates* are sorted either horizontally or vertically depending of the current level of the recursion. A separator is chosen which balances the number of vertices. Vertices are then 2-colored depending on which side of the separator they are located.

Figure 1.6 shows the recursive coordinate bisection of a multi-element airfoil geometry. In this example, the dual graph of the triangulation has

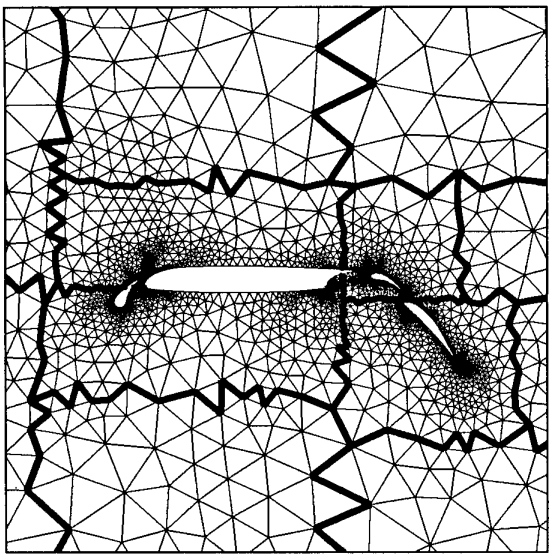


Figure 1.6: Recursive coordinate bisection partitioning of multi-element airfoil mesh.

been used for partitioning with dual graph vertices assigned the centroid coordinates of cells in the triangulation plane. The recursive coordinate partitioning is very efficient to create but gives sub-optimal performance on parallel computations owing to the long message lengths than can routinely occur.

1.2.2 Recursive Cuthill-McKee Bisection

The Cuthill-McKee algorithm described earlier can also be used for recursive mesh partitioning. In this case, the Cuthill-McKee level structure is used to 2-color vertices of the graph. A separator is chosen either at the median of the level structure ordering or at the level set boundary *closest* to the median. This latter technique has the desired effect of reducing the number of disconnected sub-graphs that occur during the recursive partitioning. Figure 1.7 shows a Cuthill-McKee partitioning for the multi-element airfoil mesh. The Cuthill-McKee ordering tends to produce long boundaries because of the way that the ordering is propagated through a mesh. The number of communication messages required to exchange boundary information tends to be higher using the Cuthill-McKee algorithm when compared to the coordinate bisection algorithm. The results shown in [VSB92] for multi-element airfoil grids indicate an overall performance on parallel computations which is slightly worse than the coordinate bisection technique.

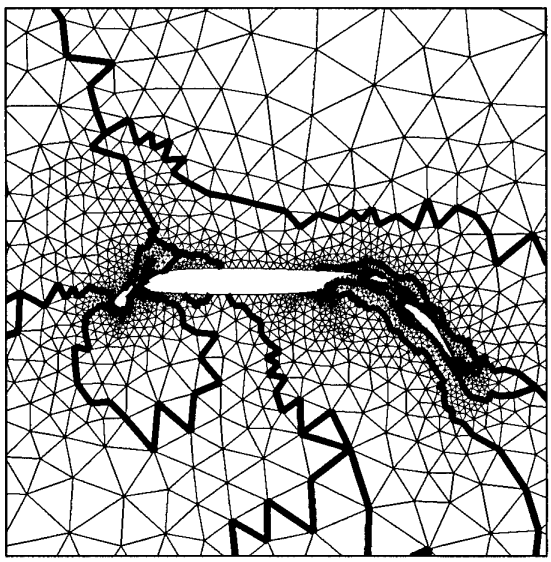


Figure 1.7: Recursive Cuthill-McKee bisection partitioning of multi-element airfoil mesh.

1.2.3 Recursive Spectral Bisection

The last partitioning algorithm considered is the spectral bisection algorithm [PSL90] [Sim91] [VSB92] [BS93] [HL95]. This algorithm determines a 2-color bisection of a graph such that the *cut-weight*, W_c, is approximately minimized. The cut-weight of a graph is defined as the sum of edge weights for all edges with vertices of disjoint color. For simplicity, we will consider unweighted (unit edge weight) graphs. The problem of minimizing the cut-weight of a graph subject to the constraint that the number of vertices is balanced is related to a simpler problem in graph bisection which is known to be *np*-hard [GJS76]. The spectral bisection algorithm can be seen as an algorithm for approximately solving

this *np*-hard combinatorial problem by solving a continuous (hopefully nearby) problem. The algorithm consists of the following steps:

Algorithm: Spectral Graph Bisection.

Step 1. Calculate the matrix $\mathcal{L}$ associated with the Laplacian of the graph.
Step 2. Calculate the eigenvalues and eigenvectors of $\mathcal{L}$.
Step 3. Order the eigenvalues by magnitude, $\lambda_1 \leq \lambda_2 \leq \lambda_3 ... \lambda_n$.
Step 4. Determine the smallest nonzero eigenvalue, λ_f and its associated eigenvector $\mathbf{x}_f$ (the Fiedler vector).
Step 5. Sort elements of the Fiedler vector.
Step 6. Choose a divisor at the median of the sorted list and 2-color vertices of the graph which correspond to elements of the Fielder vector less than or greater than the median value.

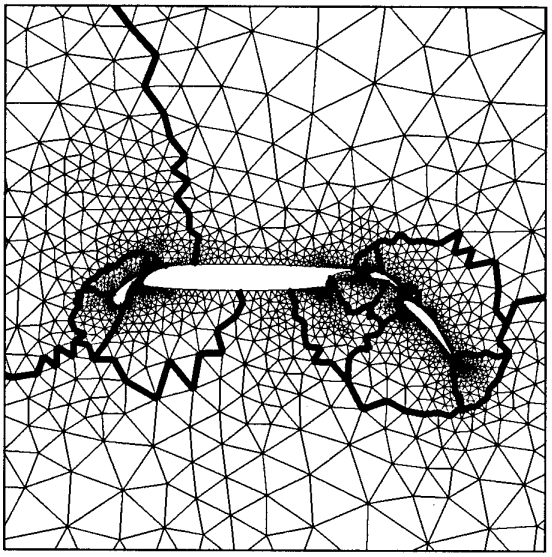


Figure 1.8: Recursive spectral bisection partitioning of multi-element airfoil mesh.

The spectral partitioning of the multi-element airfoil is shown in Figure 1.8. In [VSB92] it was observed that superior performance was attained for parallel flow field computations using spectral partitioning. The cost of the spectral partitioning is high even using a Lanczos algorithm to compute the eigenvalue problem. Recently, this cost has been reduced by the use of a multilevel Lanczos algorithm as discussed in [BS93].

The spectral partitioning exploits a peculiar property of the "second" eigenvector of the Laplacian matrix associated with a graph. Consider a the graph $G = (V, E)$ consisting of n vertices and m edges. The Laplacian matrix of a graph $\mathcal{L} \in \Re^{n \times n}$ is given by

$$\mathcal{L} = -\mathcal{D} + \mathcal{A}.$$

where $\mathcal{A} \in \Re^{n \times n}$ is the standard adjacency matrix

$$\mathcal{A}_{ij} = \begin{cases} 1 & e(v_i, v_j) \in G \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

and $\mathcal{D}$ is a $n \times n$ diagonal matrix with entries equal to the degree of each vertex, $\mathcal{D}_i = d(v_i)$. Alternatively, the Laplacian of a graph can be written in terms of the rectangular incidence matrix $\mathcal{C} \in \Re^{n \times m}$

$$\mathcal{C}_{il} = \begin{cases} -1 & \text{if } v_i \text{ is the origin of edge } l \\ 1 & \text{if } v_i \text{ is the destination of edge } l \\ 0 & \text{otherwise} \end{cases}. \tag{1.2}$$

Using the incidence matrix, the Laplacian of the graph is given by

$$\mathcal{L} = \mathcal{C}\mathcal{C}^T \tag{1.3}$$

Multiplication of $\mathcal{C}^T$ times a vector $x \in \Re^n$ is equivalent to differencing vertex values of x across each edge so that

$$\sum_{e(v_i,v_j) \in E} (x_j - x_i)^2 = x^T \mathcal{C}\mathcal{C}^T x = x^T \mathcal{L} x \tag{1.4}$$

This provides an easy way to verify the symmetry and positive semi-definiteness of $\mathcal{L}$. Also from the above definitions, it should be clear that rows of $\mathcal{L}$ each sum to zero. Define the summation vector $s \in \Re^n$, $s = [1, 1, 1, ...]^T$. By construction we have that $\mathcal{L}s = 0$. This means that at least one eigenvalue is zero with s as an eigenvector. To understand the spectral bisection algorithm, define a partitioning vector $p \in \Re^n$ which 2-colors the vertices of a graph

$$p = [+1, -1, -1, +1, +1, ..., +1, -1]^T \tag{1.5}$$

depending on the sign of elements of p and the one-to-one correspondence with vertices of the graph, see for example Figure 1.9. A critical observation is that the cut-weight can be expressed in terms of the partitioning vector p and the Laplacian of the graph by the following easily verified formula

$$W_c = \frac{1}{4} p^T \mathcal{L} p. \tag{1.6}$$

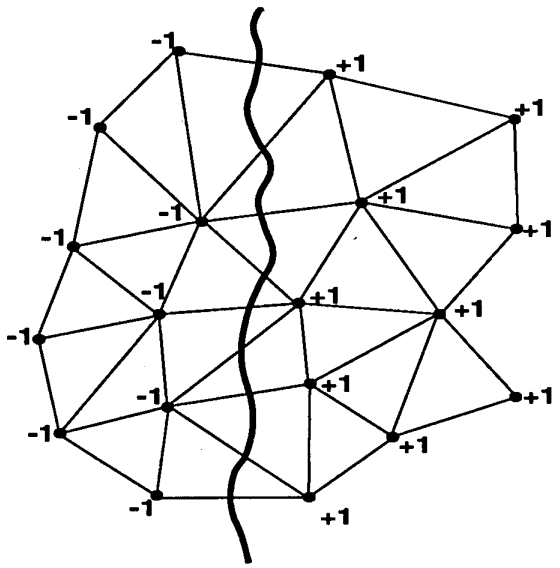


Figure 1.9: Arbitrary graph with 2-coloring showing separator and cur edges (left)

The objective of the spectral bisection algorithm is to determine a balanced 2-color partitioning of each connected component of the graph such that the number of edges cut by the partition boundary (the cut-weight) is approximately minimized.
Using the cut-weight formula, we can succinctly state the discrete bisection problem:

Discrete Bisection Problem (np-hard)

$$\begin{aligned} &\text{minimize } \frac{1}{4} p^T \mathcal{L} p \quad (\text{minimize cut} - \text{weight}) \\ &\text{subject to} \\ &p^T s = 0 \qquad (\text{balanced partitioning}) \\ &p^{(i)} = \pm 1 \qquad (\text{discrete partitioning vector}) \end{aligned} \tag{1.7}$$

In the spectral bisection method the discrete np-hard problem is replaced by a simpler continuous minimization problem. The constraint that p take on integer values ± 1 is removed and replaced with a normalization condition on a continuous partitioning vector.

Continuous Bisection Problem ($x \in \Re^n$)

$$\begin{aligned} &\text{minimize } \frac{1}{4} x^T \mathcal{L} x \quad \begin{pmatrix} \text{minimize continuous} \\ \text{cut} - \text{weight} \end{pmatrix} \\ &\text{subject to} \\ &x^T s = 0 \qquad (\text{balanced partitioning}) \\ &x^T x = n \qquad (\text{normalization}) \end{aligned} \tag{1.8}$$

After solving the continuous bisection problem (exactly), the partitioning vector p is obtained using discrete approximation:

$$p^{(i)} = \text{sign}(x^{(i)}). \qquad (\text{discrete approximation}) \tag{1.9}$$

It is the replacement of the discrete partitioning vector by a continuous counterpart followed by discrete approximation which makes the spectral bisection algorithm approximate.

The solution to the continuous bisection problem has a well-known (exact) solution in terms of the eigenvector associated with the first nonzero eigenvalue. To show this consider the spectral decomposition of $\mathcal{L}$,

$$\mathcal{L} = \sum_{i=1}^{n} \lambda_i \, y_i y_i^T, \quad 0 \le \lambda_i \le \lambda_j \quad i < j \tag{1.10}$$

where $\lambda_i \in \Re$ and $y_i \in \Re^n$ denote the eigenvalues and orthonormal eigenvectors of $\mathcal{L}$. For ease of exposition, assume the graph consists of a single connected component and let λ_2 denote the first nonzero eigenvalue. The cut-weight of the continuous problem is given by

$$4W_c = x^T \mathcal{L} x = \sum_{i=2}^{n} \lambda_i (y_i^T x)^2. \tag{1.11}$$

Since the orthonormal eigenvectors completely span all space in $\Re^n$ and $y_1 = s$, we can expand x (suitably normalized) in terms of the remaining eigenvectors vectors

$$x = n^{1/2} \sum_{i=2}^{n} \beta_i y_i = n^{1/2} \left((1-\alpha)^{1/2} y_2 + \alpha^{\frac{1}{2}} \sum_{i=3}^{n} \sigma_i y_i \right) \tag{1.12}$$

with $\sum_{i=2}^{n} \beta_i^2 = 1$ and $\sum_{i=3}^{n} \sigma_i^2 = 1$. A direct computation yields

$$4W_c = n(1-\alpha)\lambda_2 + n\alpha \sum_{i=3}^{n} \sigma_i^2 \lambda_i \qquad (1.13)$$

which is minimized when $\alpha = 0$ so that the solution

$$x = n^{1/2} y_2$$

satisfies the continuous bisection problem with a lower bound cut-weight estimate of

$$W_c = \frac{n\lambda_2}{4}.$$

Figure 1.10 shows contours of the second eigenvector for a multi-element airfoil mesh.

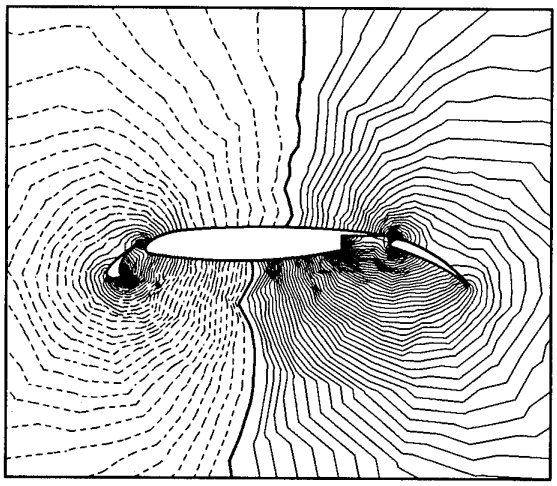


Figure 1.10: Contours of Fiedler Vector for Spectral Partitioning. Dashed lines are less than the median value (right).

1.3 Graph Quadrisection and Higher Order Partitionings

One complaint commonly leveled against recursive bisection algorithms is that they are too greedy and lack "look ahead" properties. For example, if the goal is to construct a partitioning containing 4 subdomains then a more optimal partitioning might be possible by considering all four partitions simultaneously when determining the cut-weight of the graph rather than cut-weights for pairwise bisections. This has prompted generalizations [HL95] of the spectral bisection algorithm which require more than one eigenvector. The spectral quadrisection algorithm by Henrickson and Leland [HL95] uses the first two eigenvectors, y_2 and y_3, associated with nonzero eigenvalues. The algorithm then considers orthogonal combinations subject to a rotation angle θ

$$\begin{aligned} x_2 &= y_2 \cos\theta + y_3 \sin\theta \\ x_3 &= -y_2 \sin\theta + y_3 \cos\theta. \end{aligned}$$

Again using discrete approximation, $p_2^{(i)} = \text{sign}(x_2^{(i)})$, $p_3^{(i)} = \text{sign}(x_3^{(i)})$, partitioning vectors are calculated from which quadrants are assigned $\{(+1,+1),(-1,+1),(-1,-1),(+1,-1)\}$. The angle θ is determined by minimizing the distance between x and p

$$\text{minimize} \sum_{i=1}^{n} \left((x_2^{(i)})^2 - 1\right)^2 + \left((x_3^{(i)})^2 - 1\right)^2 \qquad (1.14)$$

This has the effect of finding continuous solutions that are nearby the desired discrete solution. The results shown in [HL95] are very promising and show a definite improvement over the standard spectral bisection algorithm (which is already considered to be quite good). The technique extends naturally to higher order partitionings.

Chapter 2

A Uniprocessor CFD Algorithm

2.1 Basic Flow Equations

We consider the standard compressible Navier-Stokes equations in integral form for a domain Ω with bounding surface $\partial\Omega$

$$\frac{\partial}{\partial t}\int_\Omega \mathbf{u}\, dV + \int_{\partial\Omega}(\mathbf{f}\cdot\mathbf{n})dS = \int_{\partial\Omega}(\mathbf{g}\cdot\mathbf{n})dS \quad (2.1)$$

where $\mathbf{u}$ represents the vector of conserved variables, $\mathbf{f}$ and $\mathbf{g}$ the inviscid and viscous flux vectors respectively. In $\Re^d$ the vectors are written in Einstein summation form ($j = 1, 2, \ldots, d$):

$$\mathbf{u} = \begin{pmatrix} \rho \\ \rho u_j \\ E \end{pmatrix} \quad (2.2)$$

$$\mathbf{f} = \begin{pmatrix} \rho u_i \\ \rho u_i u_j + \delta_{ij} p \\ u_i(E+p) \end{pmatrix} \hat{1}_i,\ \mathbf{g} = \begin{pmatrix} 0 \\ \tau_{ij} \\ u_k \tau_{jk} - \kappa q_j \end{pmatrix} \hat{1}_j \quad (2.3)$$

with viscous stresses given by

$$\tau = \lambda\left(\frac{\partial u_i}{\partial x_i}\right) + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \quad (2.4)$$

and Fourier heat transfer given by $q = -\kappa\nabla T$. Finally, an ideal gas is assumed $p = \rho RT = (\gamma - 1)\left(E - \frac{1}{2}\rho(u^2 + v^2)\right)$.

2.1.1 Surface Boundary Conditions

At solid walls with no permeability and/or no slip boundary conditions the inviscid flux reduces to the following form:

$$\mathbf{f}(\mathbf{u};\mathbf{n})_s = \mathbf{f}_s \cdot \mathbf{n} = \begin{pmatrix} 0 \\ n_j p \\ 0 \end{pmatrix}.$$

An analysis of the viscous flux reveals the following limiting form for no slip surfaces:

$$\mathbf{g}(\mathbf{u}, \nabla\mathbf{u}; \mathbf{n})_s = \mathbf{g}_s \cdot \mathbf{n} = \begin{pmatrix} 0 \\ \mu\nabla u_j \cdot \mathbf{n} \\ \kappa\nabla T \cdot \mathbf{n} \end{pmatrix}.$$

The last entry in the viscous flux vanishes for adiabatic flow. These conditions can be enforced weakly. In addition, for viscous flow the strong condition can be applied that the velocity vector vanish at the surface.

2.1.2 Far Field Boundary Conditions via Characteristic Projectors

Let $A(\mathbf{u};\mathbf{n})$ denote the flux Jacobian matrix directed along the normal vector $\mathbf{n}$

$$A(\mathbf{u};\mathbf{n}) = \frac{d\mathbf{f}}{d\mathbf{u}} \cdot \mathbf{n} \quad (2.5)$$

and define the characteristic projector matrices

$$P^{\pm} = \frac{1}{2}[I \pm \mathrm{sign}(A)]. \quad (2.6)$$

The far field inviscid flux is computed from

$$\mathbf{f}_\infty = \mathbf{f}(\overline{\mathbf{u}}), \qquad \overline{\mathbf{u}} = P^+\mathbf{u} + P^-\mathbf{u}_\infty \quad (2.7)$$

so that the boundary condition satisfies the well-known Friedrichs [Fri58] strong solution admissibility condition for symmetric hyperbolic problems:

$$P^-\overline{\mathbf{u}} = P^-P^+\mathbf{u} + P^-P^-\mathbf{u}_\infty = P^-\mathbf{u}_\infty. \quad (2.8)$$

For viscous flow, the inviscid flux (2.7) can be combined with a weak Neumann condition for the viscous flux. Alternatively, strong Dirichlet conditions can be imposed as dictated by the physical problem.

2.2 Turbulence

In addition to the basic Navier-Stokes equations, we model the effects of turbulence on the mean flow equations using an eddy viscosity turbulence model. In a report with Baldwin [BB90], we proposed a single equation turbulence transport model with the specific application to unstructured meshes in mind. This model was subsequently modified by Spalart and Allmaras [SA92] to improve the predictive capability of the model for wakes and shear-layers as well as to simplify the model's dependence on distance to solid walls. In the present computations, the Spalart model is solved in a form fully coupled to the Navier-Stokes equations. The one-equation model for the viscosity-like parameter $\tilde{\nu}$ is written

$$\frac{D\tilde{\nu}}{Dt} = \frac{1}{\sigma}\left[\nabla\cdot((\nu+\tilde{\nu})\nabla\tilde{\nu}) + c_{b2}(\nabla\tilde{\nu})^2\right] - c_{w1}f_w\left(\frac{\tilde{\nu}}{d}\right)^2 + c_{b1}\tilde{S}\tilde{\nu}. \quad (2.9)$$

In the Spalart model the kinematic eddy viscosity is given by

$$\nu_t = \tilde{\nu} f_{v1} \quad (2.10)$$

and requires the following closure functions and constants

$$\begin{aligned}
\tilde{S} &= |\omega| + \frac{\nu\tilde{\nu}}{\kappa^2 d^2} f_{v2} \\
f_{v1} &= \frac{\chi^3}{\chi^3 + c_{v1}^3} \\
f_{v2} &= 1 - \frac{\chi}{1+\chi f_{v1}} \\
r &= \frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2} \\
g &= r + c_{w2}(r^6 - r)
\end{aligned}$$

with ω the fluid vorticity, d the distance to the closest surface, and the constants $c_{b1} = 0.1355$, $c_{b2} = 0.622$, $c_{v1} = 7.1$, $c_{w1} = 3.24$, $c_{w2} = 0.3$, $c_{w3} = 2.0$, $\kappa = .41$, $\sigma = 2./3.$. The model also includes an optional term for simulating transition to turbulence.

2.3 The Spatial Discretization Algorithm

The flow equations are discretized in space using a finite-volume method. In this technique the solution domain is tessellated into a number of smaller subdomains ($\Omega = \cup\Omega_i$). Each subdomain serves as a control volume in which mass, momentum, and energy are conserved. In the present application, the control volumes are formed from a median dual obtained from the triangulation, see Figure 2.1.

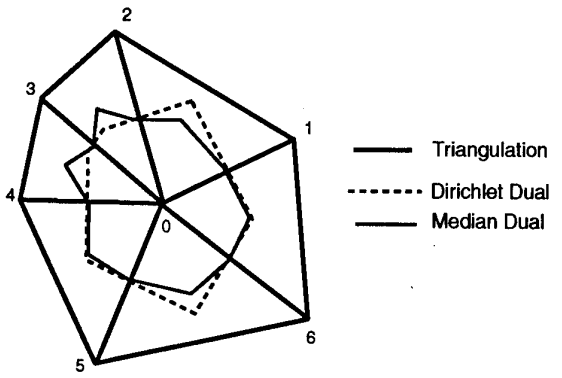


Figure 2.1: Local triangulation with Dirichlet and median duals.

Fundamental to the finite-volume method is the definition of the integral cell average. Componentwise, the integral cell average is defined in each subdomain as:

$$\overline{u}_i = \frac{1}{V_i}\int_{\Omega_i} u\, dV$$

where $V_i = \int_{\Omega_i} dV$. The integral conservation law can then be rewritten in the following form:

$$\frac{\partial}{\partial t}(\overline{\mathbf{u}}V) + \int_{\partial\Omega}(\mathbf{f}\cdot\mathbf{n})dS = \int_{\partial\Omega}(\mathbf{g}\cdot\mathbf{n})dS. \quad (2.11)$$

The integral cell averages are the basic unknowns (degrees of freedom) in the scheme. The task at hand is to evaluate the flux integral given these cell averages of the solution. The basic solution process is summarized in the following steps and further details are given in [Bar91] [BJ89]:

Reconstruction

Given integral averages of the solution in each control volume, reconstruct a piecewise polynomial which approximates the behavior of the solution in each control volume.

Flux Quadrature

From the piecewise polynomial description of the solution, approximate the flux integral in (2.11) by numerical quadrature. Because the piecewise polynomials are not continuous at control volume boundaries, special flux functions are employed which are functions of multiple solution states. Those flux functions which can be characterized as some approximate and/or exact solution of the Riemann problem of gasdynamics result in upwind biased approximations. Present computations utilize Roe's approximate Riemann solver [Roe81].

Evolution

Given a numerical approximation to the flux integral, evolve the system in time using any class of implicit or explicit schemes. This results in new integral cell averages of the solution. The solution process can then be repeated.

It is important to realize that for steady-state calculations, the spatial accuracy of the scheme depends solely on the reconstruction and flux quadrature steps. Moreover, the use of cell averages can be replaced by pointwise values of the solution associated with each control volume. In our application, we place the solution unknowns at mesh vertices. As we will see, this can greatly simplify the reconstruction step. Unfortunately, schemes based on these reconstructed polynomials are subject to the generation of spurious oscillations near discontinuities and regions of high solution gradient unless additional measures are taken which limit extremum behavior of the reconstructed polynomial. These measures are the basis for the class of MUSCL schemes developed by van Leer [vL79]. This framework of reconstruction followed by monotonicity enforcement extends naturally to unstructured meshes in higher dimensions and sufficient conditions required by the reconstructed polynomial to guarantee monotonicity are generally known, see for example [Bar94].

2.3.1 Linear Least-Squares Reconstruction

Consider a vertex v_0 and suppose that the solution varies linearly over the support of adjacent neighbors of the mesh. In this case, the change in vertex values of the solution along an edge $e(v_i, v_0)$ can be calculated by

$$(\nabla u)_0 \cdot (\mathbf{r}_i - \mathbf{r}_0) = u_i - u_0$$

where $\mathbf{r}$ denotes the spatial position vector. This equation represents the scaled projection of the gradient along the edge $e(v_i, v_0)$. A similar equation could be written for all incident edges subject to an arbitrary weighting factor. The result is the following matrix equation, shown here in two dimensions:

$$\begin{bmatrix} w_1\Delta x_1 & w_1\Delta y_1 \\ \vdots & \vdots \\ w_n\Delta x_n & w_n\Delta y_n \end{bmatrix} \begin{pmatrix} u_x \\ u_y \end{pmatrix} = \begin{pmatrix} w_1(u_1 - u_0) \\ \vdots \\ w_n(u_n - u_0) \end{pmatrix}$$

or in symbolic form $\mathcal{L}\ \nabla u = \mathbf{f}$ where

$$\mathcal{L} = \begin{bmatrix} \overrightarrow{L}_1 & \overrightarrow{L}_2 \end{bmatrix}$$

in two dimensions. Exact calculation of gradients for linearly varying u is guaranteed if any two row vectors $w_i(\mathbf{r}_i - \mathbf{r}_0)$ span all of 2 space. This implies linear independence of $\overrightarrow{L}_1$ and $\overrightarrow{L}_2$. The system can then be solved via normal equations

$$\begin{bmatrix} \overrightarrow{V}_1 \\ \overrightarrow{V}_2 \end{bmatrix} \begin{bmatrix} \overrightarrow{L}_1 & \overrightarrow{L}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The row vectors $\overrightarrow{V}_i$ are given by

$$\overrightarrow{V}_1 = \frac{l_{22}\overrightarrow{L}_1 - l_{12}\overrightarrow{L}_2}{l_{11}l_{22} - l_{12}^2}, \quad \overrightarrow{V}_2 = \frac{l_{11}\overrightarrow{L}_2 - l_{12}\overrightarrow{L}_1}{l_{11}l_{22} - l_{12}^2} \tag{2.12}$$

with $l_{ij} = (\overrightarrow{L}_i \cdot \overrightarrow{L}_j)$.

Note that reconstruction of N independent variables in $\Re^d$ implies $\binom{d+1}{2} + dN$ inner product sums. Since only dN of these sums involves the solution variables themselves, the remaining sums could be precalculated and stored in computer memory. Using the edge data structure, the calculation of inner product sums can

be calculated for *arbitrary* combinations of polyhedral cells. In all cases linear functions are reconstructed exactly.

Algorithm: Weighted Least-Squares Gradient Calculation

For $k = 1, n(e)$	Loop through edges
$j_1 = e^{-1}(k, 1)$	Edge origin
$j_2 = e^{-1}(k, 2)$	Edge destination
$\Delta x = w(k) \cdot (x(j_2) - x(j_1))$	Weighted Δx
$\Delta y = w(k) \cdot (y(j_2) - y(j_1))$	Weighted Δy
$l_{11}(j_1) = l_{11}(j_1) + \Delta x \cdot \Delta x$	l_{11} orig sum
$l_{11}(j_2) = l_{11}(j_2) + \Delta x \cdot \Delta x$	l_{11} dest sum
$l_{12}(j_1) = l_{12}(j_1) + \Delta x \cdot \Delta y$	l_{12} orig sum
$l_{12}(j_2) = l_{12}(j_2) + \Delta x \cdot \Delta y$	l_{12} dest sum
$\Delta u = w(k) \cdot (u(j_2) - u(j_1))$	Weighted Δu
$f_1(j_1) += \Delta x \cdot \Delta u$	$\overrightarrow{L}_1 f$ sum
$f_1(j_2) += \Delta x \cdot \Delta u$	
$f_2(j_1) += \Delta y \cdot \Delta u$	$\overrightarrow{L}_2 f$ sum
$f_2(j_2) += \Delta y \cdot \Delta u$	
Endfor	

For $j = 1, n(v)$ Process vertices dividing by det
$det = l_{11}(j) \cdot l_{22}(j) - l_{12}^2(j)$
$u_x(j) = (l_{22}(j) \cdot f_1(j) - l_{12}(j) \cdot f_2(j))/det$
$u_y(j) = (l_{11}(j) \cdot f_2(j) - l_{12}(j) \cdot f_1(j))/det$
Endfor

This formulation provides freedom in the choice of weighting coefficients, w_i. These weighting coefficients can be a function of the geometry and/or solution. Classical approximations in one dimension can be recovered by choosing geometrical weights of the form $w_i = 1./|\Delta \mathbf{r}_i - \Delta \mathbf{r}_0|^t$ for values of $t = 0, 1, 2$. Data dependent choices are discussed in [Bar94].

2.4 Exact and Approximate Newton Methods

In this section we consider implicit solution strategies for the upwind discretization scheme described in the previous section. Define the solution vector

$$\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, ..., \mathbf{u}_N]^T$$

and similarly the residual vector $\mathbf{R}$ for all mesh vertices. The basic scheme is written as

$$D\mathbf{U}_t = \mathbf{R}(\mathbf{U}) \tag{2.13}$$

where D is a positive diagonal matrix. Performing a backward Euler time integration, equation (2.13) is rewritten as

$$D(\mathbf{U}^{n+1} - \mathbf{U}^n) = \Delta t\ \mathbf{R}(\mathbf{U}^{n+1})$$

where n denotes the iteration (time step) level. Linearizing the right-hand-side of the preceding equation in time produces the following form:

$$D(\mathbf{U}^{n+1} - \mathbf{U}^n) = \Delta t \left[\mathbf{R}(\mathbf{U}^n) + \frac{d\mathbf{R}^n}{d\mathbf{U}}(\mathbf{U}^{n+1} - \mathbf{U}^n)\right] \tag{2.14}$$

By rearrangement of terms, we arrive at the delta form of the backward Euler scheme

$$\left[\frac{D}{\Delta t} - \frac{d\mathbf{R}^n}{d\mathbf{U}}\right](\mathbf{U}^{n+1} - \mathbf{U}^n) = \mathbf{R}(\mathbf{U}^n). \tag{2.15}$$

Note that for large time steps, the scheme becomes equivalent to Newton's method. In practice the diagonal entries are locally scaled as a exponential function of the norm of the residual

$$\frac{D_i}{\Delta t} = \frac{\mathrm{cfl}_i}{\mathrm{cfl}_{max}}, \qquad \mathrm{cfl}_{max} = f(\|\mathbf{R}(\mathbf{U})^n\|)$$

so that when $\|\mathbf{R}(\mathbf{U})\| \to 0$, $\mathrm{cfl}_{max} \to \infty$ and the scheme approaches Newton's method. It should be emphasized that by using this strategy, the scheme is technically an approximate Newton method which becomes exact only in the final few iterations of the computation.

The following two sections present examples which demonstrate the convergence characteristics of Newton's method for inviscid and viscous fluid flow problems. In viewing these examples, the reader can assume that each matrix problem required in the Newton scheme is solved "exactly." In reality, these matrix problems are solved iteratively to a user specified tolerance. The topic of solving the linear algebra problem will be discussed in detail in later sections. The test case examples are presented at this time so that they may be used in the remaider of these notes for comparison purposes.

2.4.1 Test Case 1: Inviscid Flow Past a Multi-Element Airfoil

As a first test case, inviscid Euler flow is computed about a multi-element airfoil geometry as shown in Figure 2.2.

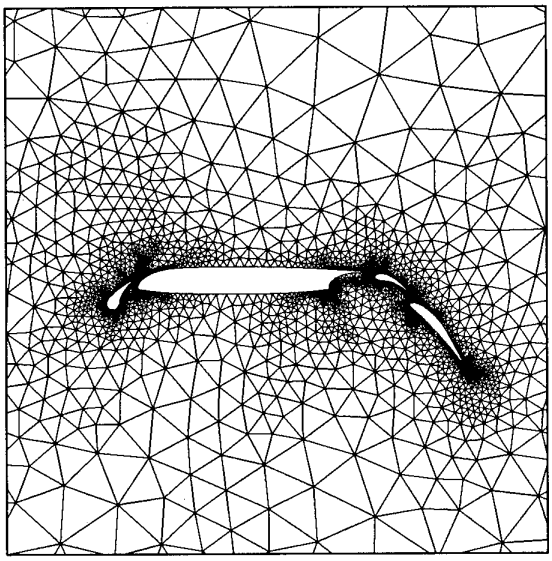


Figure 2.2: Multi-element airfoil mesh, 4900 vertices.

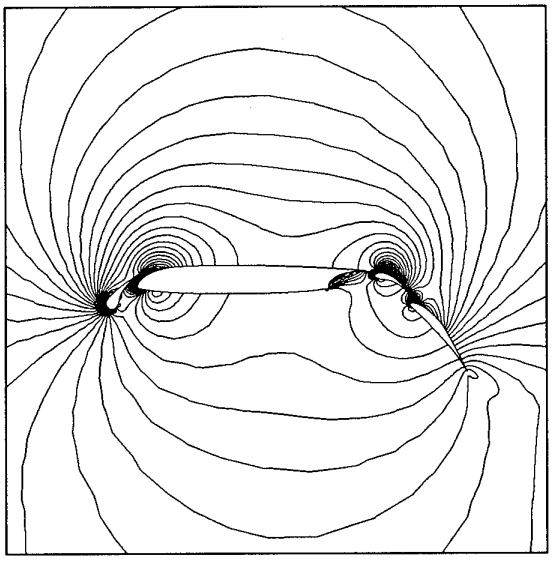


Figure 2.3: Solution isomach contours about multi-element geometry, $M_\infty = 0.2$, $\alpha = 2.0°$.

The mesh contains approximately 4900 mesh vertices. Subsonic flow conditions are imposed ($M_\infty = 0.2$) with a 2° free stream angle of attack. Figures 2.3 - 2.5 show Mach number contours, surface pressure coefficient, and convergence history for the calculation. An initial time step was chosen for the calculation which corresponds to an effective local CFL number of approximately 50, but over the next 10 iterations the effective CFL number quickly reaches 10^8. This test case will be used extensively in Chapter 3 when evaluating parallel solution strategies.

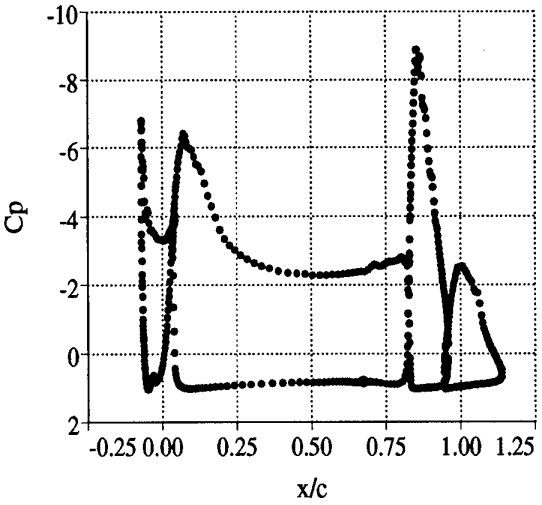


Figure 2.4: Surface pressure coefficient computed from multi-element geometry.

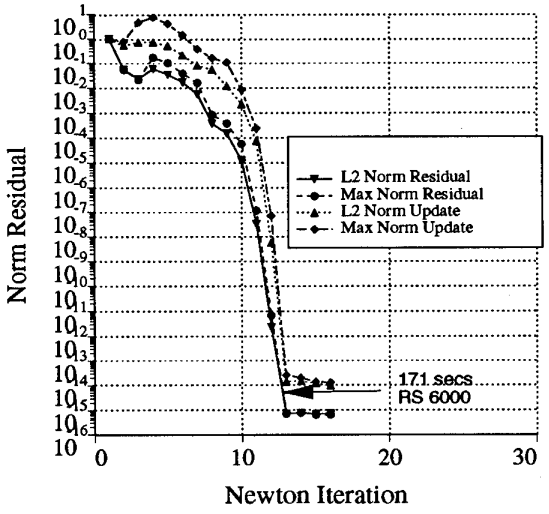


Figure 2.5: Numerical solution convergence history.

2.4.2 Test Case 2: Viscous Flow Past a Multi-Element Airfoil

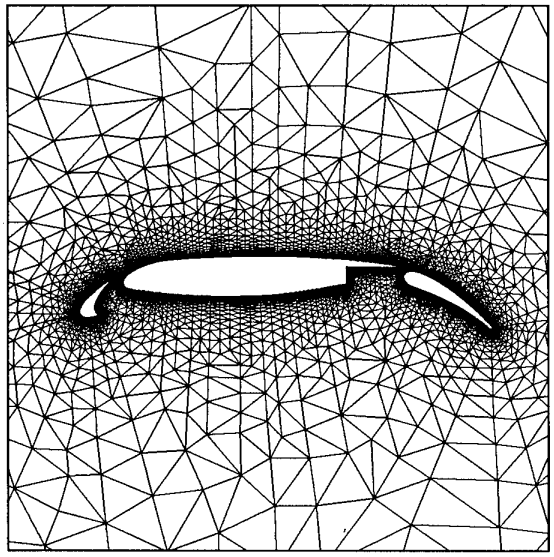


Figure 2.6: Multi-element airfoil triangulation, 22,000 vertices.

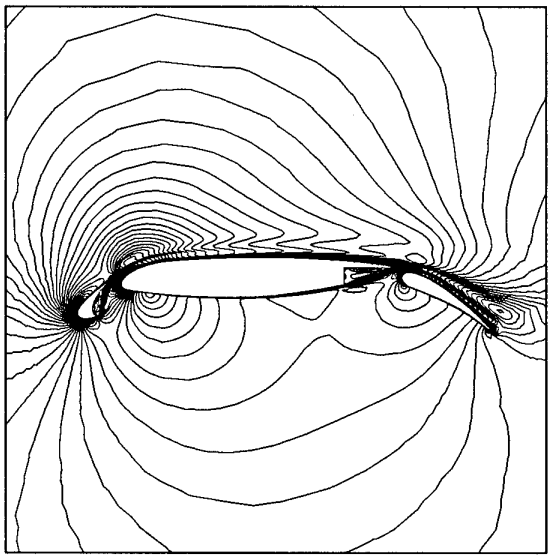


Figure 2.7: Multi-element airfoil solution isomach contours, $M_\infty = 0.2$, $\alpha = 16.0°$, $Re = 9.0$ million.

As a second test case, viscous flow with turbulence is computed about the multiple-element airfoil geometry. This geometry has been triangulated using the Steiner triangulation algorithm described in [Bar95], see Figure 2.6. The mesh contains approximately 22,000 vertices with cells near the airfoil surface attaining aspect ratios greater than 1000:1. This example provides a demanding test case for CFD algorithms. The experimental flow conditions are $M_\infty = .20$, $\alpha = 16°$, and a Reynolds number of 9 million. Experimental results are given in [VDMG92] and computed results are shown in Figure 2.7. Even though the wake passing over the main element is not well resolved, the surface pressure coefficient shown in Figure 2.8 agrees quite well with experiment. The convergence history in Figure

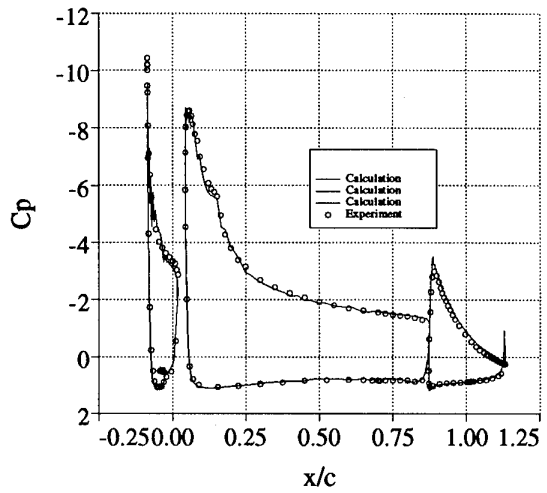


Figure 2.8: Comparison of computational and experimental surface pressure coefficients.

2.9 shows that roughly twice as many iteration steps are needed for the viscous turbulent flow calculation when compared to the inviscid flow computation of Test Case 1. This seems to be typical for aerodynamic high lift computations. This is contrasted by single element airfoil computations which show very little difference in the number of iterations needed when computing inviscid and viscous flow. This test case will also be used extensively in subsequent chapters for evaluating various solution strategies.

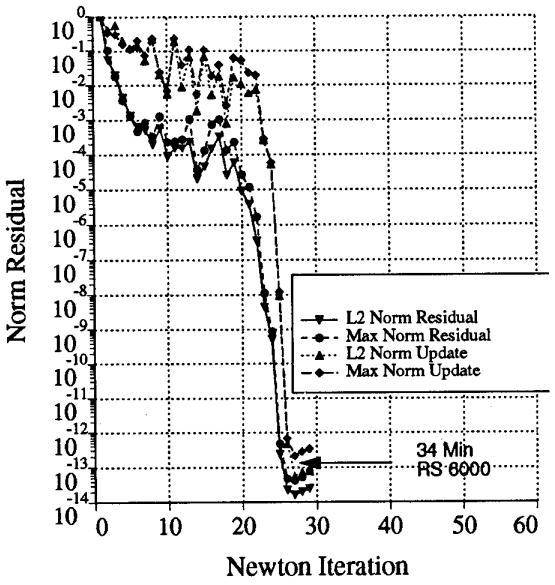


Figure 2.9: Solution convergence history for Case 2 computation.

2.4.3 Storage Requirements

It is worthwhile to assess the computer memory requirements for storing sparse matrices obtained from discretizations on simplicial meshes (triangulations). In practice we will be solving systems of l coupled equations so that each nonzero entry of the matrix is actually a small $l \times l$ block. The schemes discussed in previous sections require data from distance-one neighbors in the graph (mesh). In addition, the higher order accurate schemes require distance-two neighbors in building the scheme. First consider the situation in which the scheme requires only distance-one neighbors. The number of nonzero entries in each row of the matrix is related to the number of edges incident to the vertex associated with that row. Or equivalently, each edge $e(v_i, v_j)$ will guarantee nonzero entries in the i-th column and j-th row and similarly the j-th column and i-th row. In addition nonzero entries will be placed on the diagonal of the matrix. From this counting argument we see that the number of nonzero block entries, nnz, in the matrix is exactly twice the number of edges plus the number of vertices, $2E+N$ (approximately $7N$ in 2-d). Using a similar counting argument we obtain the following approximate requirements for storing distance-one and distance-two neighboring information as a sparse matrix: Note that the entries of the sparse matrix associated with Newton's method are actually small 5×5 and 6×6 blocks in two and three dimensions respectively. At first glance, this storage requirement appears prohibitively large. While this may be true to some extent today, the memory capacity of computers is expanding at a rapid rate. It is quite reasonable to expect that in the foreseeable future sufficient memory will be available for solving most problems of engineering interest. Even so, it is possible to reduce, and in some cases eliminate, the explicit storage of the Jacobian matrix without compromising the favorable convergence characteristics of Newton's method. These techniques will be discuss in subsequent sections.

Table 2.1: Storage Estimates for Sparse Matrices

Dim.	nnz (Distance-1)	nnz (Distance-2)
2	$7N$	$19N$
3	$14N$	$55N$

2.4.4 Calculating Analytic Jacobian Derivatives

In this section we address the task of computing Jacobian derivatives for Newton's method. In the following section we consider the related task of multiplying an arbitrary vector by the Jacobian matrix.

A major task in the overall calculation of the Jacobian derivatives for the finite-volume discretization is the linearization of the numerical flux vector with respect to the two solution states, e.g. given the Roe flux function [Roe81]

$$\mathbf{h}(\mathbf{u}^R, \mathbf{u}^L; \mathbf{n}) = \frac{1}{2}\left(\mathbf{f}(\mathbf{u}^R, \mathbf{n}) + \mathbf{f}(\mathbf{u}^L, \mathbf{n})\right) \quad (2.16)$$

$$- \frac{1}{2}|A(\mathbf{u}^R, \mathbf{u}^L; \mathbf{n})|\left(\mathbf{u}^R - \mathbf{u}^L\right) \quad (2.17)$$

we require the Jacobian terms $\frac{d\mathbf{h}}{d\mathbf{u}^R}$ and $\frac{d\mathbf{h}}{d\mathbf{u}^L}$. Exact analytical expressions for these terms are available [Bar87]. In constructing the Jacobian matrix for the entire scheme it is useful to conceptualize the finite-volume scheme in *composition form*:

$$\mathbf{R}(\mathbf{U}) = \mathcal{L}_1(\mathcal{L}_2(\mathbf{U})) \tag{2.18}$$

with $\mathcal{L}_1$ representing the flux quadrature and accumulation step and $\mathcal{L}_2$ representing the data reconstruction step. In this form, each operator requires distance-1 information. The Jacobian matrix can then be written as

$$\frac{d\mathbf{R}}{d\mathbf{U}} = \frac{d\mathcal{L}_1}{d\mathcal{L}_2}\frac{d\mathcal{L}_2}{d\mathbf{U}} \tag{2.19}$$

with the critical observation that the Jacobian matrix can be calculated as the sparse product of two matrices. This could potentially be an expensive task, but because of the special form of $\mathcal{L}_1$ and $\mathcal{L}_2$, the resulting sparse product produces at most distance-2 fill and can be computed at reasonable cost.

2.4.5 Exact and Approximate Jacobian Matrix-Vector Products

Consider the standard matrix equation $A\mathbf{x} - \mathbf{b} = 0$. As we will see, iterative matrix solution algorithms for this problem such as the generalized minimum residual method (GMRES) and the stablized bi-conjugate gradient method (Bi-CGSTAB) both require the computation of matrix-vector products of the form $A\mathbf{p}$ for some arbitrary $\mathbf{p}$ vector. In the approximate Newton algorithm

$$A = \left[\frac{D}{\Delta t} - \frac{d\mathbf{R}}{d\mathbf{U}}\right] \tag{2.20}$$

so that a major computation in the matrix-vector product $A\mathbf{p}$ is the computation of Jacobian derivatives in the direction of $\mathbf{p}$ (a Fréchet derivative)

$$\overline{A}\mathbf{p} = \frac{d\mathbf{R}}{d\mathbf{U}}\mathbf{p}. \tag{2.21}$$

Several possible strategies exist for computing the needed Fréchet derivatives:

Sparse Matrix-Vector Multiply

The most straightforward strategy is to analytically compute and store the Jacobian matrix using a compressed storage scheme designed for sparse matrices. This strategy has the added benefit that a copy of the matrix can also be used as a preconditioner for the iterative solver. In addition, the explicit storage also permits the formation of the transposed matrix problem which is often encountered in optimization procedures coupled with Newton's method. Obviously, a detraction of this approach is the large storage requirement.

Approximate Fréchet Derivatives

An alternative to the analytic calculation of Fréchet derivatives is to approximate them using a finite-difference approximation [Joh92] [BS94] [EW94]. The required Fréchet derivative is a limiting form of the difference approximation

$$\frac{d\mathbf{R}}{d\mathbf{U}}\mathbf{p} = \lim_{\epsilon \to 0} \frac{\mathbf{R}(\mathbf{U} + \epsilon\mathbf{p}) - \mathbf{R}(\mathbf{U})}{\epsilon}.$$

The primary concern with this approach is the accuracy of derivatives and the optimal choice for ϵ. If derivatives are not computed accurately then methods such as GMRES or Bi-CGSTAB iteration may stall or fail. Using a forward difference approximation, ϵ must be carefully chosen. In general it is insufficient to choose ϵ as a constant such as the square root of machine precision. Johan [Joh92] also mentions this fact and gives some analysis for choosing ϵ but this analysis assumes that $\mathbf{R}(\mathbf{u})$ is well scaled. A common choice for ϵ is given by

$$\epsilon = \delta_0 + \delta_1 \frac{\|\mathbf{U}\|}{\|\mathbf{p}\|} \tag{2.22}$$

with suitably chosen constants δ_0 and δ_1. An alternative is to use higher order accurate formulas such as central differencing at double the computational cost.

The clear attraction of this approach is the low memory requirement. On the other hand, the numerical computation of Fréchet derivatives does not produce a matrix approximation which

can be used to precondition the system. Lastly, for situations requiring the solution of the transposed matrix problem, there does not appear to be a Fréchet-like technique for constructing the matrix-vector product

$$\left[\frac{d\mathbf{R}}{d\mathbf{U}}\right]^T \mathbf{p}$$

using numerical difference approximations. We consider this a serious shortcoming of the method.

Exact Product Forms

In this section we will present a technique for constructing matrix-vector products which is an exact calculation of the Fréchet derivative. Extension to systems and the inclusion of diffusion terms are also handled using this technique.

Let $G(E,V)$ denote the triangulation in 2-d or 3-d with n vertices and m edges. Next recall the definition of the incidence matrix given in Equation (1.2):

$$C_{il} = \begin{cases} -1 & \text{if } v_i \text{ is the origin of edge } l \\ 1 & \text{if } v_i \text{ is the destination of edge } l \\ 0 & \text{otherwise} \end{cases} . \tag{2.23}$$

Let $\mathbf{h} = \mathbf{h}(\mathbf{u}^L, \mathbf{u}^R; \mathbf{n})$ denote the numerical flux function as defined by Equation 2.17. For a system of l coupled differential equations, the Jacobian matrix entries are actually small $l \times l$ blocks. For ease of exposition, we tacitly treat these small blocks as scalar entries. Under these simplifications, the desired matrix-vector product is given by

$$\frac{d\mathbf{R}}{d\mathbf{U}}\mathbf{p} = \mathcal{C}^T\left[\left[\frac{d\mathbf{h}}{d\mathbf{u}^L}\right]\left[\frac{d\mathbf{u}^L}{d\mathbf{u}}\right] + \left[\frac{d\mathbf{h}}{d\mathbf{u}^R}\right]\left[\frac{d\mathbf{u}^R}{d\mathbf{u}}\right]\right]\mathbf{p} \tag{2.24}$$

where $\left[\frac{d\mathbf{h}}{d\mathbf{u}}\right] \in \Re^{m\times m}$ with nonzero diagonal elements, and $\left[\frac{d\mathbf{u}^L}{d\mathbf{u}}\right] \in \Re^{m\times n}$. If we do not incorporated monotonicity enforcement into the reconstruction procedure then a considerable simplification occurs in the calculation of matrix-vector products. The main idea is given in the following almost trivial lemma.

Lemma: Let $v = \mathcal{R}(U) = \mathcal{R}(u_1, u_2, ..., u_n)$ denote an arbitrary order reconstruction operator. If $\mathcal{R}$ depends linearly on u_i then

$$\frac{dv}{du}p = \mathcal{R}(p).$$

Proof: Linearity implies that

$$v = \mathcal{R}(u_1, u_2, ..., u_n) = \sum_{i=1}^{n} \alpha_i u_i$$

so that $\frac{dv}{du_i} = \alpha_i$. The desired result follows immediately

$$\frac{dv}{du}p = \sum_{i=1}^{n} \frac{dv}{du_i}p_i = \sum_{i=1}^{n} \alpha_i p_i = \mathcal{R}(p).$$

This lemma suggests the following procedure for calculation of matrix-vector products.

$$\frac{d\mathbf{R}}{d\mathbf{U}}\mathbf{p} = \mathcal{C}^T\left[\left[\frac{d\mathbf{h}}{d\mathbf{u}^L}\right]\mathcal{R}^L(\mathbf{p}) + \left[\frac{d\mathbf{h}}{d\mathbf{u}^R}\right]\mathcal{R}^R(\mathbf{p})\right] \tag{2.25}$$

This amounts to a reconstruction of the vectors $\mathbf{p}^L$ and $\mathbf{p}^R$ from $\mathbf{p}$ using the same reconstruction operator used in the residual computation. Next, the linearized form of the flux function is computed:

$$\mathbf{h}_{lin} = \frac{d\mathbf{h}}{d\mathbf{u}^L}\mathbf{p}^L + \frac{d\mathbf{h}}{d\mathbf{u}^R}\mathbf{p}^R.$$

Finally, the linearized fluxes are assembled using the same procedure as the residual vector assembly. In actual calculations, the conservative flow variables are not reconstructed, thereby necessitating that a change of variable transformation be embedded in the formulation. This is not a serious complication.

Equation (2.25) does not reveal how to construct the transposed matrix-vector product

$$\left[\frac{d\mathbf{R}}{d\mathbf{U}}\right]^T \mathbf{p}.$$

But by introducing addition matrices, we can derive the required equation. In addition, the following forms allow the incorporation of monotonicity limiting in the reconstruction process,

although we have not done so here. Define $\mathcal{A}, \mathcal{S}^+, \mathcal{S}^- \in \Re^{n \times m}$

$$\text{If } e(v_i, v_j) \in G(E,V), \text{ then}$$

$$\mathcal{A}_{ie} = \mathcal{S}^+_{ie} = 1, \ \mathcal{A}_{je} = \mathcal{S}^-_{je} = 1$$

and zero otherwise. In addition, define the diagonal $m \times m$ matrices containing weighted edge geometry $[\Delta x]$ and $[\Delta y]$ as well as the $n \times n$ diagonal matrices $[D_{uv}]$ containing pointwise determinant values for the least squares solution. Using these matrices the left and right reconstructed states obtained by least squares reconstruction are given by

$$\begin{aligned}
\mathbf{u}^L = & \left[[\mathcal{S}^-]^T \right. \\
& + \left[\frac{\Delta x}{2}\right] [\mathcal{S}^-]^T \left[[D_{xx}]\, A\, [\Delta x] + [D_{xy}]\, A\, [\Delta y] \right] \mathcal{C}^T \\
& + \left[\frac{\Delta y}{2}\right] [\mathcal{S}^-]^T \left. \left[[D_{yx}]\, A\, [\Delta x] + [D_{yy}]\, A\, [\Delta y] \right] \mathcal{C}^T \right] \mathbf{U} \\
\mathbf{u}^R = & \left[[\mathcal{S}^+]^T \right. \\
& - \left[\frac{\Delta x}{2}\right] [\mathcal{S}^+]^T \left[[D_{xx}]\, A\, [\Delta x] + [D_{xy}]\, A\, [\Delta y] \right] \mathcal{C}^T \\
& - \left[\frac{\Delta y}{2}\right] [\mathcal{S}^+]^T \left. \left[[D_{yx}]\, A\, [\Delta x] + [D_{yy}]\, A\, [\Delta y] \right] \mathcal{C}^T \right] \mathbf{U}.
\end{aligned}$$

From these formulas the transposed matrix-problem problem is easily calculated

$$\begin{aligned}
\left[\frac{d\mathbf{R}}{d\mathbf{U}}\right]^T \mathbf{p} = & \left[[\mathcal{S}^-] \right. \\
& + \mathcal{C} \left[[\Delta x]\, A^T\, [D_{xx}] + [\Delta y]\, A^T\, [D_{xy}] \right] [\mathcal{S}^-] \left[\frac{\Delta x}{2}\right] \\
& + \mathcal{C} \left[[\Delta x]\, A^T\, [D_{yx}] + [\Delta y]\, A^T\, [D_{yy}] \right] [\mathcal{S}^-] \left[\frac{\Delta y}{2}\right] \\
& \left. \left[\frac{d\mathbf{h}}{d\mathbf{u}^L}\right]^T \mathcal{C}^T \mathbf{p} + \left[[\mathcal{S}^+] \right. \right. \\
& - \mathcal{C} \left[[\Delta x]\, A^T\, [D_{xx}] + [\Delta y]\, A^T\, [D_{xy}] \right] [\mathcal{S}^+] \left[\frac{\Delta x}{2}\right] \\
& - \mathcal{C} \left[[\Delta x]\, A^T\, [D_{yx}] + [\Delta y]\, A^T\, [D_{yy}] \right] [\mathcal{S}^+] \left[\frac{\Delta y}{2}\right] \\
& \left. \left. \left[\frac{d\mathbf{h}}{d\mathbf{u}^R}\right]^T \mathcal{C}^T \mathbf{p}. \right. \right.
\end{aligned} \tag{2.26}$$

Just as equations (2.26) have an implementation using an edge data structure (one would *never* store the connectivity as $\mathcal{A}$ or $\mathcal{C}$ in dense matrix form), the transposed equation has an implementation using an edge data structure for the calculation of $\left[\frac{d\mathbf{R}}{d\mathbf{U}}\right]^T \mathbf{p}$. For example, the matrix operation $\mathcal{A}^T v$ performs a *gather and sum* of the two edge vertex values of v for each and every edge. The matrix operation $\mathcal{A}w$ performs a *scatter and accumulate* of an edge quantity w to the two edge vertices locations for each and every edge. Similar edge operations exist for the incidence matrix $\mathcal{C}$. Thus we have constructed a technique for matrix-vector products based on elementary edge operations which also permits constructing the transposed matrix-vector product. The ability to write the entire algorithm in terms of a sequence of edge operations makes the parallel implementation straightforward.

2.4.6 Solving the Matrix Problem

The next task is to solve the large sparse linear system of the form

$$A\mathbf{p} - \mathbf{b} = 0$$

produced by Newton's method. Owing to the nonsymmetry of A, we consider solving this system using the generalized minimum residual method (GMRES) of Saad and Schultz [SS86] and the stabilized bi-conjugate gradient method of Van der Vorst [vdV92]. Both algorithms are outlined in Table 2.2. The paragraphs given below briefly describe the methods but for a full description we defer to the lectures of Prof. Van der Vorst.

The GMRES Algorithm

The GMRES algorithm explicitly forms an orthogonal basis $[\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k]$ from the Krylov sequence $[\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0]$ using a modified Gram-Schmidt orthogonalization procedure. Using this orthogonal basis, GMRES iterates are computed

$$\mathbf{p}_k = \mathbf{p}_0 + \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \ldots + \alpha_k \mathbf{v}_k \tag{2.27}$$

by minimizing the residual norm

$$\|A\mathbf{p}_k - \mathbf{b}\|. \tag{2.28}$$

The algorithm requires $k+1$ vector inner products, $k+1$ **SAXPY** operations, and k matrix-vector multiplies for iteration k. Thus as k increases, the storage increases linearly and the computation quadratically. To avoid the storage and computation demands imposed by large matrices, Saad and Schultz devised a variant, GMRES(k), in which the GMRES algorithm is restarted every k steps. The proper choice of k is problem dependent and a strong function of the choice and quality of preconditioning.

The Bi-CGSTAB Algorithm

The stabilized bi-conjugate gradient method (Bi-CGSTAB) is a short recurrence method designed for nonsymmetric matrices. Roughly speaking, Bi-CGSTAB combines the basic bi-conjugate gradient method with GMRES(1). The inclusion of the GMRES(1) steps is intended to smooth the irregular convergence behavior of the basic Bi-CG method. The Bi-CGSTAB method requires 4 vector inner products, 6 **SAXPY** operations, and 2 matrix-vector products for each iteration.

Matrix Preconditioning

Practical experience has shown that the success or failure of the GMRES and Bi-CGSTAB algorithm hinges heavily on the choice of matrix preconditioning. In left preconditioned form, the matrix problem becomes

$$P(A\mathbf{p} - \mathbf{b}) = 0. \quad \text{(left preconditioned)} \tag{2.29}$$

An alternative is the right preconditioned system

$$APP^{-1}\mathbf{p} - \mathbf{b} = 0. \quad \text{(right preconditioned)} \tag{2.30}$$

If available, the optimal choice of P (left or right) is clearly A^{-1} or its $L-U$ factors. In this instance the underlying matrix problem is trivially solved in one step. More generally, we consider finding a *nearby* preconditioning matrix such that $\kappa(AP) < \kappa(A)$, i.e. AP is better conditioned than A alone.

In the present applications, we consider a preconditioning matrix based the incomplete lower-upper (ILU) factorization of the matrix A. ILU preconditioning is a popular and robust preconditioning procedure for use in iterative matrix solvers. ILU factorization is a modification to the standard Gaussian elimination for which the nonzero fill pattern is either preimposed or determined dynamically based on the size or location of fill elements. In this way the amount of storage required can be specified and in some instances minimized. Technical aspects of ILU factorization such as existence and spectral properties have been proven for M-matrices, but the general applicability is much broader and well documented in the literature. The triangular solves required in the application of ILU preconditioning generally give the method global support. This is usually considered a favorable characteristic of the method.

The finite-volume scheme with high order data reconstruction suggests two possible matrices suitable for incomplete factorization.

1. Distance-1 matrix preconditioning. Construct the preconditioning matrix from the Jacobian matrix associated with the lower (first) order accurate discretization of the flow equations. This matrix involves distance-1 neighbors in the triangulation. Matrix-vector products are computed "exactly" using the Jacobian matrix associated with the full second order accurate scheme.

2. Distance-2 matrix preconditioning. Use the Jacobian matrix of the entire second order accurate scheme for both matrix-vector products and preconditioning.

Algorithm: Preconditioned GMRES(k)

For $l = 1, m$	m restart iterations
$\mathbf{v}_0 := \mathbf{b} - A\mathbf{p}_0$	initial residual
$\beta := \|\mathbf{r}_0\|_2$	initial residual norm
$\mathbf{v}_1 := \mathbf{r}_0/\beta$	define initial Krylov
For $j = 1, k$	inner iterations
$\mathbf{y}_j := P\mathbf{v}_j$	preconditioning
$\mathbf{w} := A\mathbf{y}_j$	matrix-vector prod
For $i = 1, j$	Gram-Schmidt
$h_{i,j} := (\mathbf{w}, \mathbf{v}_i)$	.
$\mathbf{w} := \mathbf{w} - h_{i,j}\mathbf{v}_i$	.
End For	
$h_{j+1,j} := \|\mathbf{w}\|_2$	
$\mathbf{v}_{j+1} := \mathbf{w}/h_{j+1,j}$	define Krylov vector
End For	
$\mathbf{z} := \min_{\hat{\mathbf{z}}} \|\beta\mathbf{e}_1 - H\hat{\mathbf{z}}\|_2$	least squares solve
$\mathbf{p} := \mathbf{p}_0 + \sum_{i=1}^{m} \mathbf{y}_i\mathbf{z}_i$	approximate solution
If $\|\beta\mathbf{e}_1 - H\mathbf{z}\|_2 \leq \epsilon$ exit	convergence check
$\mathbf{p}_0 := \mathbf{p}$	restart
End For	

Algorithm: Preconditioned Bi-CGSTAB

$\mathbf{r}_0 := \mathbf{b} - A\mathbf{p}_0$	initial residual
$\tilde{\mathbf{r}} := \mathbf{r}_0$	
For $i = 1, m$	m total iterations
$\rho_{i-1} := \tilde{\mathbf{r}}^T\mathbf{r}_{i-1}$	
If $\rho_{i-1} = 0$ **(Breakdown)**	method fails
If $i = 1$	
$\mathbf{y}_i := \mathbf{r}_{i-1}$	
Else	
$\beta_{i-1} := (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})$	
$\mathbf{y}_i := \mathbf{r}_{i-1} + \beta_{i-1}(\mathbf{y}_{i-1} - \omega_{i-1}\mathbf{v}_{i-1})$	
Endif	
$\tilde{\mathbf{y}} := P\mathbf{y}_i$	preconditioning
$\mathbf{v}_i := A\tilde{\mathbf{y}}$	matrix-vector prod
$\alpha_i := \rho_{i-1}/\tilde{\mathbf{r}}^T\mathbf{v}_i$	
$\mathbf{s} := \mathbf{r}_{i-1} - \alpha_i\mathbf{v}_i$	
If $\|\mathbf{s}\|_2 < \epsilon$	check tolerance
$\mathbf{p}_i := \mathbf{p}_{i-1} + \alpha_i\tilde{\mathbf{y}}$	
Exit	
Endif	
$\tilde{\mathbf{s}} := P\mathbf{s}$	preconditioning
$\mathbf{t} := A\tilde{\mathbf{s}}$	matrix-vector prod
$\omega_i := \mathbf{t}^T\mathbf{s}/\mathbf{t}^T\mathbf{t}$	
$\mathbf{p}_i := \mathbf{p}_{i-1} + \alpha_i\tilde{\mathbf{y}}_i + \omega_i\tilde{\mathbf{s}}$	
$\mathbf{r}_i := \mathbf{s} - \omega_i\mathbf{t}$	
If $\|\mathbf{r}_i\|_2 < \epsilon$ Exit	
If $\omega_i = 0$ **(Breakdown)**	method fails
End For	

Table 2.2: GMRES and Bi-CGSTAB Algorithms for Nonsymmetric Matrices

Performance of GMRES and Bi-CGSTAB for Case 1 and Case 2 Test Problems

The test problems given in Sections 2.4.1 and 2.4.2 provide representative matrices for evaluating the GMRES and Bi-CGSTAB algorithms. In evaluating the iterative methods we construct approximate Newton matrices corresponding to flow CFL numbers of 10^3 and 10^8. In addition, distance-1 and distance-2 preconditioning matrices are used to accelerate the algorithms. Figures 2.10-2.11 graph the convergence histories for the GMRES and Bi-CGSTAB algorithms in solving matrix problems produced from the inviscid flow problem given in Section 2.4.1. Since the matrix-vector products and preconditioning solves dominate the iterative calculation, convergence histories are plotted against the number of matrix-vector products required. Each GMRES iteration requires one matrix-vector product while each Bi-CGSTAB iteration requires two products. The GMRES algorithm is clearly adversely affected by the distance-1 preconditioning. For this case the distance-1 preconditioned system requires roughly twice as many iterations as the distance-2 preconditioned system. These graphs also show the somewhat erratic convergence behavior of the Bi-CGSTAB method. Even so, the Bi-CGSTAB appears to require a similar number of matrix-vector products when compared to the GMRES algorithm.

The second test case given in Section 2.4.2 is more revealing. Matrices have been taken from this test case corresponding to CFL numbers of 10^3 and 10^8. Computations show a definite degradation in convergence for both methods using the distance-1 preconditioning, see Figures 2.12-2.13. In fact for CFL = 10^8, the convergence is unacceptably slow. In general we find that when using the distance-1 preconditioning matrix, an optimal CFL number exists for convergence and efficiency which is large but not infinite.

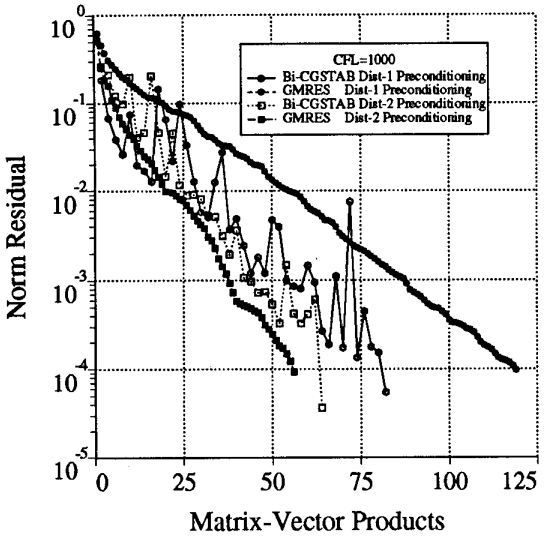


Figure 2.10: Case 1 (Inviscid Flow) matrix solution convergence histories for the $GMRES(20)$ and $Bi-CGSTAB$ algorithms at $CFL = 10^3$ using ILU(0) distance-1 and distance-2 preconditioning matrices

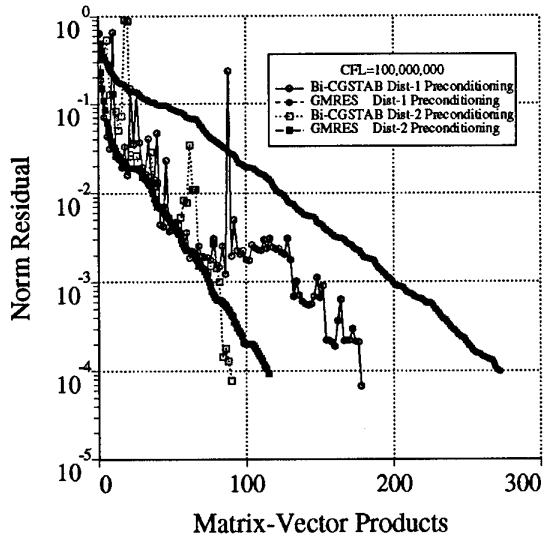


Figure 2.11: Case 1 (Inviscid Flow) matrix solution convergence histories for the $GMRES(20)$ and $Bi-CGSTAB$ algorithms at $CFL = 10^8$ using ILU(0) distance-1 and distance-2 preconditioning matrices

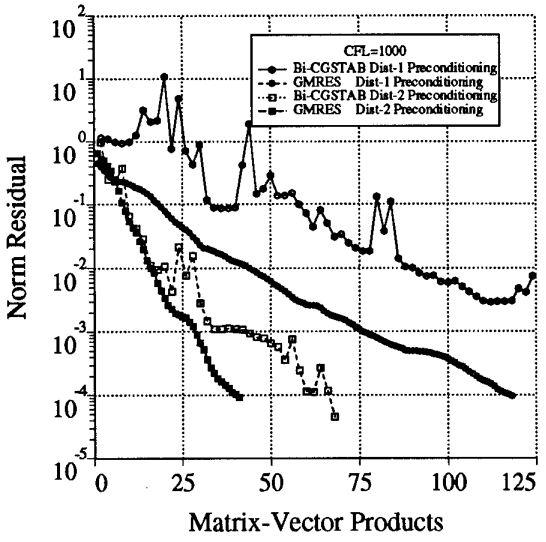


Figure 2.12: Case 2 (Viscous Flow) matrix solution convergence histories for the $GMRES(30)$ and $Bi-CGSTAB$ algorithms at $CFL = 10^3$ using ILU(0) distance-1 and distance-2 preconditioning matrices

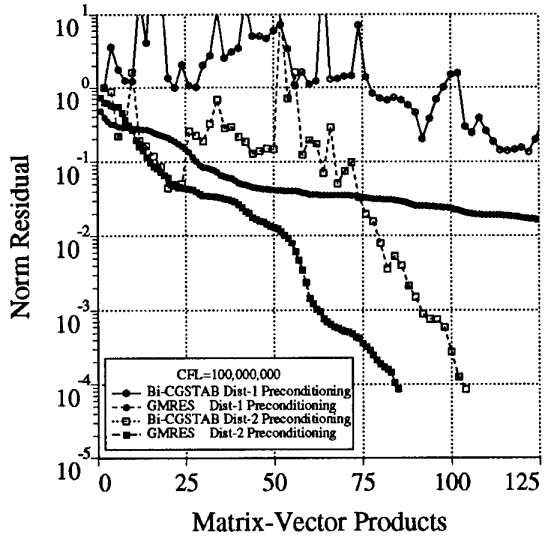


Figure 2.13: Case 2 (Viscous Flow) matrix solution convergence histories for the $GMRES(30)$ and $Bi-CGSTAB$ algorithms at $CFL = 10^8$ using ILU(0) distance-1 and distance-2 preconditioning matrices

Chapter 3

Parallel Algorithms

3.1 Additive and Multiplicative Schwarz Procedures

In this section we review the Schwarz theory for elliptic problems. Beginning with analysis of the two point boundary value problem, we derive the exact theory governing the alternating Schwarz method introduced in 1869 by Schwarz [Sch69]. Next we consider the discrete Schwarz procedure and mention some known results concerning the method for unstructured meshes.

3.1.1 The Model Two Point BVP

Consider two point boundary value problem on the interval $x \in [0, L]$

$$\begin{aligned} -u''(x) &= f(x) \\ u(0) = u(L) &= 0 \end{aligned} \tag{3.1}$$

which has the solution

$$u(x) = \int_0^L g_0(x;\xi) f(\xi) d\xi \tag{3.2}$$

in terms of the Green's function $g_0(x;\xi)$ defined on that interval. Equation (3.1) implies that for $0 < \alpha < \beta < 1$ the following Dirichlet problems:

$$\begin{aligned} -u''(x) &= f(x), \quad x \in [0, \beta L] \\ u(0) = 0, &\quad u(\beta L) = u(\beta L) \end{aligned} \tag{3.3}$$

$$\begin{aligned} -u''(x) &= f(x), \quad x \in [\alpha L, L] \\ u(\alpha L) = u(\alpha L), &\quad u(L) = 0 \end{aligned} \tag{3.4}$$

Let $g_1(x;\xi)$ and $g_2(x;\xi)$ denote Green's functions on the intervals $[0, \beta L]$ and $[\alpha L, L]$ respectively. Using these Green's functions we obtain the solution of (3.3) for $x \in [0, \beta L]$:

$$u(x) = \frac{x}{\beta L} u(\beta L) + \int_0^{\beta L} g_1(x;\xi) f(\xi) d\xi \tag{3.5}$$

and (3.4) for $x \in [\alpha L, L]$:

$$u(x) = \frac{L - x}{(1-\alpha)L} u(\alpha L) + \int_{\alpha L}^{L} g_2(x;\xi) f(\xi) d\xi \tag{3.6}$$

In the following paragraphs we consider the additive and multiplicative Schwarz algorithms which utilize (3.3) and (3.4).

The Additive Schwarz Algorithm

The basic idea in additive Schwarz domain decomposition is to consider the iteration

$$\begin{aligned} -U''(x) &= f(x), \quad 0 < x < \beta L \\ U^{k+1}(0) = 0, &\; U^{k+1}(\beta L) = V^k(\beta L) \end{aligned} \tag{3.7}$$

$$\begin{aligned} -V''(x) &= f(x), \quad \alpha L < x < L \\ V^{k+1}(\alpha L) = U^k(\alpha L), &\; V^{k+1}(L) = 0 \end{aligned} \tag{3.8}$$

From Equations (3.5) and (3.6) it follows that

$$\begin{aligned} U^{k+1}(x) &= \frac{x}{\beta L} V^k(\beta L) + \int_0^{\beta L} g_1(x;\xi) f(\xi) d\xi \\ V^{k+1}(x) &= \frac{L - x}{(1-\alpha)L} U^k(\alpha L) + \int_{\alpha L}^{L} g_2(x;\xi) f(\xi) d\xi \end{aligned}$$

where the interval of validity is understood. Next define the error functions

$$\begin{aligned} d^{k+1}(x) &= U^{k+1}(x) - u(x) = \left(\frac{x}{\beta L}\right) e^k(\beta L) \\ e^{k+1}(x) &= V^{k+1}(x) - u(x) = \left(\frac{L - x}{L(1-\alpha)}\right) d^k(\alpha L). \end{aligned} \tag{3.9}$$

Clearly the error behaves linearly and of the form

$$d^k(x) = \frac{a^k}{\beta L}x, \quad e^k(x) = \frac{b^k}{L}\left(\frac{L-x}{1-\alpha}\right) \quad (3.10)$$

Substitution into (3.9) yields

$$a^{k+1} = \left(\frac{1-\beta}{1-\alpha}\right)b^k, \quad b^{k+1} = \left(\frac{\alpha}{\beta}\right)a^k \quad (3.11)$$

so that

$$a^{k+2} = \left(\frac{(1-\beta)\alpha}{(1-\alpha)\beta}\right)a^k, \quad b^{k+2} = \left(\frac{(1-\beta)\alpha}{(1-\alpha)\beta}\right)b_k. \quad (3.12)$$

From this we obtain a fundamental result in domain decomposition concerning the additive Schwarz algorithm:

Theorem 3.1 *Consider the additive Schwarz iteration given by (3.7), (3.8). There exists a constant $C = C(U^0, V^0)$ such that*

$$\sup_{0\le x\le \beta L} |U^{2k} - u(x)| \le C\left(\frac{(1-\beta)\alpha}{(1-\alpha)\beta}\right)^k \quad (3.13)$$

$$\sup_{\alpha L\le x\le L} |V^{2k} - u(x)| \le C\left(\frac{(1-\beta)\alpha}{(1-\alpha)\beta}\right)^k. \quad (3.14)$$

The proof follows directly from (3.12) and (3.9). ∎

The Multiplicative Schwarz Algorithm

Two subdomain multiplicative Schwarz algorithm differs from the additive Schwarz algorithm only in that the subdomains are updated sequentially, i.e.

$$\begin{aligned} &-U''(x) = f(x), \quad 0 < x < \beta L \\ &U^{k+1}(0) = 0, U^{k+1}(\beta L) = V^k(\beta L) \end{aligned} \quad (3.15)$$

$$\begin{aligned} &-V''(x) = f(x), \quad \alpha L < x < L \\ &V^{k+1}(\alpha L) = U^{k+1}(\alpha L), V^{k+1}(L) = 0 \end{aligned} \quad (3.16)$$

Following a similar analysis to the previous section we obtain the following result concerning the multiplicative Schwarz algorithm:

Theorem 3.2 *Consider the multiplicative Schwarz iteration given by (3.15), (3.16). There exists a constant $C = C(U^0, V^0)$ such that*

$$\sup_{0\le x\le \beta L} |U^k - u(x)| \le C\left(\frac{(1-\beta)\alpha}{(1-\alpha)\beta}\right)^k \quad (3.17)$$

$$\sup_{\alpha L\le x\le L} |V^k - u(x)| \le C\left(\frac{(1-\beta)\alpha}{(1-\alpha)\beta}\right)^k. \quad (3.18)$$

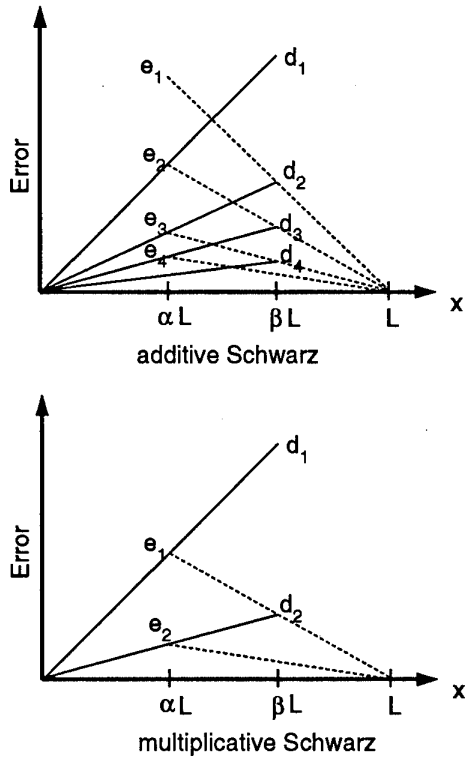


Figure 3.1: Comparison of the 2 subdomain additive Schwarz (top) and multiplicative Schwarz (bottom) algorithms for the two point BVP.

The theory clearly shows the favorable consequences of increased overlap. Figure 3.1 graphs the error functions $d(x)$ and $e(x)$ for the two domain additive and multiplicative Schwarz algorithms. As predicted by the theory, the multiplicative algorithm converges at a rate twice that of the additive algorithm. Next, consider the situation in which both subdomains are of equal size with overlap distance δ. Some simple

algebra yields the following simple relationship for the convergence parameter:

$$\frac{(1-\beta)\alpha}{(1-\alpha)\beta} = \left(\frac{1-\frac{\delta}{L}}{1+\frac{\delta}{L}}\right)^2$$

which is graphed in Figure 3.2. But as Figure 3.3

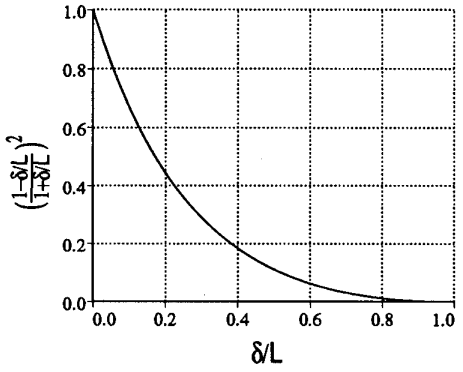


Figure 3.2: Convergence parameter $\frac{1-\frac{\delta}{L}}{1+\frac{\delta}{L}}$ for the two subdomain Schwarz iteration with equal subdomain size and overlap δ/L.

shows, the algorithm deteriorates as the number of subdomains increases. This effect will be quantified in the next section.

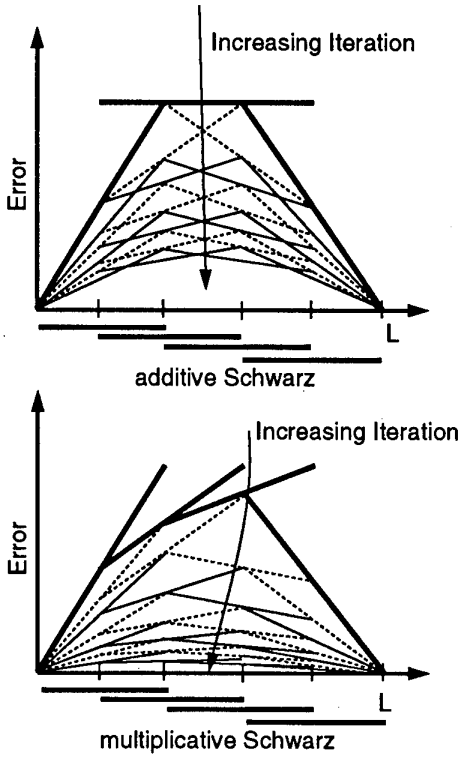


Figure 3.3: Comparison of the 4 subdomain additive Schwarz (top) and multiplicative Schwarz (bottom) algorithms for the two point BVP.

3.1.2 The Discrete Schwarz Theory

In this section we review the Schwarz theory for discrete systems using the notation given in Chan and Mathew [CM94]. Consider the symmetric positive definite linear system

$$Au = f \tag{3.19}$$

obtained from the 2-D discretization of an elliptic equation on the domain Ω which consists of two overlapping subdomains Ω_1 and Ω_2 such that $\Omega = \Omega_1 \cup \Omega_2, \Omega_1 \cap \Omega_2 \neq \emptyset$. Let I_1 and I_2 denote the set of interior mesh vertices contained in Ω_1 and Ω_2 respectively. The total number of esh vertices is n and the number of interior vertices in I_1 and I_2 is n_1 and n_2. Next define the zero extension matrices $R_i^T \in \Re^{n \times n_i}$ for each subdomain such that for $x_i \in \Re^{n_i}$

$$(R_i^T x_i)_k = \begin{cases} (x_i)_k & k \in I_i \\ 0 & \text{otherwise} \end{cases} \tag{3.20}$$

The local subdomain matrices are then given by $A_i = R_i A R_i^T$. The discrete form of the alternating Schwarz procedure produces the following sequence of iterates

$$\begin{aligned} u^{k+1/2} &= u^k + R_1^T A_1^{-1} R_1 (f - Au^k) \\ u^{k+1} &= u^{k+1/2} + R_2^T A_2^{-1} R_2 (f - Au^{k+1/2}) \end{aligned} \tag{3.21}$$

Defining the matrices

$$P_i = R_i^T A_i^{-1} R_i A \tag{3.22}$$

the convergence is governed by the iteration matrix $(I - P_2)(I - P_1)$. This motivates the term *multiplicative* Schwarz iteration. Similarly, the sequence of iterates

$$\begin{aligned} u^{k+1/2} &= u^k + R_1^T A_1^{-1} R_1 (f - Au^k) \\ u^{k+1} &= u^{k+1/2} + R_2^T A_2^{-1} R_2 (f - Au^k) \end{aligned} \tag{3.23}$$

produces the combined *additive* Schwarz scheme

$$\begin{aligned} u^{k+1} &= u_k + \left(R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2\right)(f - Au^k) \\ &= M_{as}^{-1}(f - Au^k) \end{aligned} \tag{3.24}$$

so that the convergence is governed by the sum $M_{as}^{-1}A = P_1 + P_2$. The additive Schwarz algorithm is appealing in parallel computation since each subdomain solve can be done in parallel. Unfortunately the performance of the algorithm deteriorates as the number of subdomains increases. This effect was observed in Figure 3.3. Let H denote the characteristic size of each subdomain, δ the overlap distance, and h the mesh spacing. The condition number κ of $M_{as}^{-1}A$ is given in the following theorem:

Theorem 3.3 *There exists a constant C independent of H and h such that*

$$\kappa(M_{as}^{-1}A) \leq CH^{-2}\left(1 + \left(\frac{H}{\delta}\right)^2\right). \tag{3.25}$$

Proof: Given in [DW89] [DW92]. ∎

This theorem describes the deterioration as the number of subdomains increases (and H decreases). With some additional work this deterioration can be removed by the introduction of a *global, coarse subspace* and a global restriction matrix $R_H \in \Re^{n_c \times n}$. The *two level* additive Schwarz matrix for p subdomains becomes

$$M_{as}^{-1} = R_H^T A_H^{-1} R_H + \sum_{i=1}^{p} R_i^T A_i^{-1} R_i. \tag{3.26}$$

Under the assumption of "generous overlap" we have the following result:

Theorem 3.4 *There exists a constant C independent of H and h such that*

$$\kappa(M_{as}^{-1}A) \leq C\left(1 + \left(\frac{H}{\delta}\right)\right). \tag{3.27}$$

Proof: See [DW89] [DW92] and Chan and Zou [CZ93]. ∎

3.1.3 Interface Substructuring

A number of algorithms exist in domain decomposition which exhibit superior conditioning to that given in Equation (3.27) while still maintaining parallel scalability. The lectures by Professor Farhat will describe the two-level Finite Element Tearing and Interconnectivity (FETI) method which has

$$\kappa(M^{-1}A) \leq C\left(1 + \log^2(H/h)\right) \tag{3.28}$$

conditioning properties for self-adjoint equations on meshes with element size h. In this method, Lagrange multipliers are introduced to couple subdomains and ensure continuity at interface boundaries.

Other interface strategies begin by ordering matrix unknowns in each subdomain first followed by interface unknowns as shown in Figure 3.4. This matrix ordering can be represented by

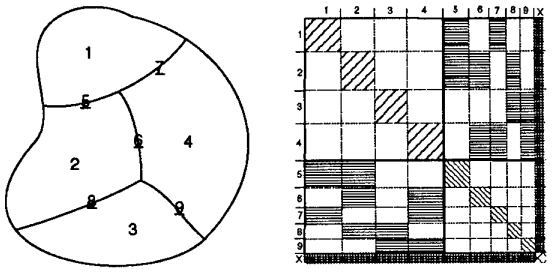


Figure 3.4: Arbitrary domain with subdomains $1-4$ and interfaces $5-9$.

the following partitioned matrix equation:

$$\begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

Next consider the 2×2 inverse

$$A^{-1} = \begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix},$$

where

$$\begin{aligned} C_1 &= A_1^{-1} + A_1^{-1}A_2 S^{-1} A_3 A_1^{-1} \\ C_2 &= -A_1^{-1}A_2 S^{-1} \\ C_3 &= -S^{-1}A_3 A_1^{-1} \\ S &= A_4 - A_3 A_1^{-1} A_2 \\ C_4 &= S^{-1}. \end{aligned}$$

In practice, we do not require the explicit inverse of the matrix A but rather the ability to apply A^{-1} to an arbitrary vector. In this context the Schur complement strategy requires the ability to solve matrix systems for A_1 and the Schur complement matrix S. Observe that the matrix A_1 consists of decoupled subdomain matrices which can be solved independently (in parallel). The Schur complement matrix provides a form of global coupling of interface unknowns. In Smith [Smi92], the Schur complement form of the equations is considered together with a vertex, edge, and face space decomposition thus producing a coarse-fine space algorithm with

$$\kappa(M^{-1}A) \leq C\,(1+\log(H/\delta)). \qquad (3.29)$$

conditioning. A complete discussion of these advanced techniques is beyond the scope of these notes. Further details and references can be found in Chan and Mathew [CM94] and Tallec [Tal94].

3.1.4 Boundary Condition Admissibility and Hyperbolic Equations

Consider the model d-dimensional hyperbolic differential equation

$$\lambda(x)\cdot\nabla u(x) = f(x), \quad x \in \Re^d \qquad (3.30)$$

in a domain Ω with boundary Γ and boundary normal $\mathbf{n}$. Next subdivide Γ into segments Γ^- and Γ^+ associated with incoming ($\lambda\cdot\mathbf{n} < 0$) and outgoing (($\lambda\cdot\mathbf{n} > 0$)) characteristics. The admissibility condition requires that

$$\begin{aligned} u &= u|_\Omega, \quad \lambda\cdot\mathbf{n} > 0 \\ u &= u|_\infty, \quad \lambda\cdot\mathbf{n} \leq 0 \end{aligned} \qquad (3.31)$$

In terms of the characteristic projectors $p^\pm = \frac{1}{2}[1+\text{sign}(\lambda\cdot\mathbf{n})]$ the boundary condition becomes

$$u = p^+u|_\Omega + p^-u|_\infty. \qquad (3.32)$$

Next consider a two subdomain overlapped problem, $\Omega = \Omega_1 \cup \Omega_2$ with $\Omega_1 \cap \Omega_2 \neq \emptyset$. The hyperbolic form of the equations suggests the following multiplicative Schwarz type algorithm

$$\begin{aligned} &\lambda\cdot\nabla u_1^{k+1} = f \\ &\text{with boundary conditions} \\ &u_1^{k+1} = p^+u_1^{k+1}|_{\Omega_1} + p^-u|_\infty, \quad \Gamma\cap\Gamma_1 \\ &u_1^{k+1} = p^+u_1^{k+1}|_{\Omega_1} + p^-u_2^{k}, \quad \Gamma_1\cap\Omega_2 \end{aligned} \qquad (3.33)$$

followed by

$$\begin{aligned} &\lambda\cdot\nabla u_2^{k+1} = f \\ &\text{with boundary conditions} \\ &u_2^{k+1} = p^+u_2^{k+1}|_{\Omega_2} + p^-u|_\infty, \quad \Gamma\cap\Gamma_2 \\ &u_1^{k+1} = p^+u_2^{k+1}|_{\Omega_2} + p^-u_1^{k+1}, \quad \Gamma_2\cap\Omega_1. \end{aligned} \qquad (3.34)$$

An additive Schwarz-type algorithm can be similarly posed [Qua90]. In the next section, we will show that numerical schemes based on upwind differencing naturally inherit the admissibility condition so that Dirichlet overlap conditions can be imposed even in the hyperbolic limit.

3.1.5 Numerical Admissibility for Hyperbolic Equations

Consider the model advection equation

$$u_t + c(x)u_x = 0, \quad 0 \leq x \leq L \qquad (3.35)$$

together with the prototypical differencing scheme

$$(u_j)_t + \frac{h_{j+1/2} - h_{j-1/2}}{\Delta x} = 0, \quad j = 0,1,...,N \qquad (3.36)$$

defined on the mesh $x_j = j\Delta x$ with $\Delta x = L/N$. Next consider the following conventional upwind flux function for the interface position located midway between x_j and x_{j+1}

$$h_{j+1/2} = \frac{1}{2}c_{j+1/2}(u_{j+1}+u_j) - \frac{1}{2}|c_{j+1/2}|(u_{j+1}-u_j). \qquad (3.37)$$

After some manipulation, this flux can be placed in the following revealing form

$$h_{j+1/2} = c^{j+1/2}(p^+_{j+1/2}u_j + p^-_{j+1/2}u_{j+1}) \qquad (3.38)$$

with $p^{\pm}_{j+1/2} = \frac{1}{2}[1 + \text{sign}(c_{j+1/2})]$. If we impose a Dirichlet condition on u at $x = L$ for $c(L) > 0$ this would normally lead to an ill-posed hyperbolic problem. But by using (3.37) we see that numerically the ill-posed data is ignored by the scheme. More generally we make the following observation:

The use of upwind flux functions such the Roe's flux function [Roe81] permits the numerical overspecification of boundary data. In the strong solution limit, the characteristic nature of the flux function correctly ignores all ill-posed boundary data.

This remarkable property greatly simplifies the implementation of Schwarz schemes for hyperbolic equations. The only complication that arises concerns higher order schemes in which the flux formula takes a slightly more general form

$$h_{j+1/2} = \frac{1}{2}c_{j+1/2}(u^L + u^R)_{j+1/2} - \frac{1}{2}|c_{j+1/2}|(u^R - u^L)_j \tag{3.39}$$

where u^L and u^R denote states obtained from higher order reconstruction and extrapolation. For example the extrapolation formulas

$$\begin{aligned} u^L_{j+1/2} &= u_j + (\delta_x u)_j \frac{\Delta x}{2} \\ u^R_{j+1/2} &= u_{j+1} - (\delta_x u)_{j+1} \frac{\Delta x}{2} \end{aligned} \tag{3.40}$$

would require having values of the solution gradient $\delta_x u$ at subdomain boundaries. A one-sided approximation could be made but would lead to an inconsistency in residuals at mesh vertices distance-1 from subdomain boundaries. The alternative is to compute numerical gradients in each subdomain followed by at exchange at subdomain boundaries as shown in Figure 3.5. When gradient data is exchanged in this way, the final solution obtained in each subdomain will be identical to a single domain computation.

3.1.6 Numerical Results for Additive and Multiplicative Schwarz Iteration

The following paragraphs present sample Schwarz calculations for the inviscid and viscous flow test cases given in Sections 2.4.1 and 2.4.2. In these calculations, we will consider overlap distances of 1,2, and 3 as partially depicted in Figures 3.6 and 3.7. Note that we have not included a "coarse space" correction to the Schwarz method. Consequently, we expect to see a degradation in the method as the number of partitions increases.

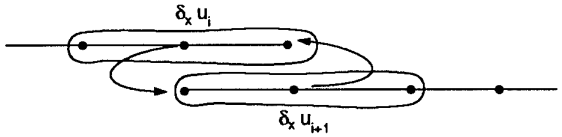


Figure 3.5: Strategy for exchanging boundary gradients prior to flux computation.

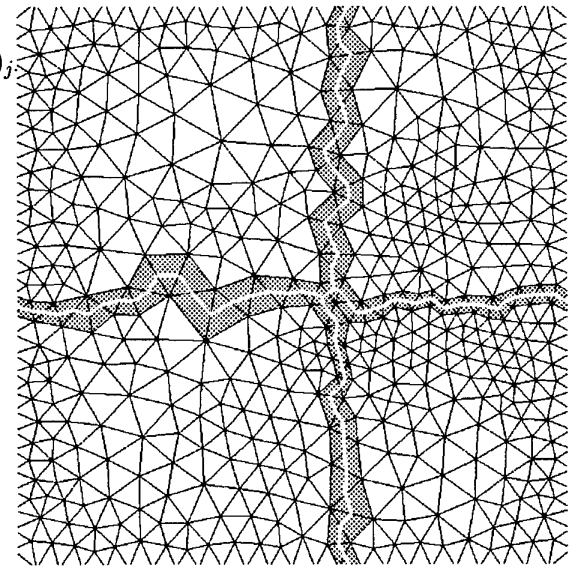


Figure 3.6: A triangulated and partitioned domain exhibiting distance-1 overlap

Test Case 1 (Inviscid Flow)

Using the mesh and flow conditions given in Section 2.4.1 inviscid flow was computed about the multi-element airfoil geometry. Figures 3.8-3.11 graph convergence histories for multiplicative and additive Schwarz iterations.

Each graph contains data for overlap distances

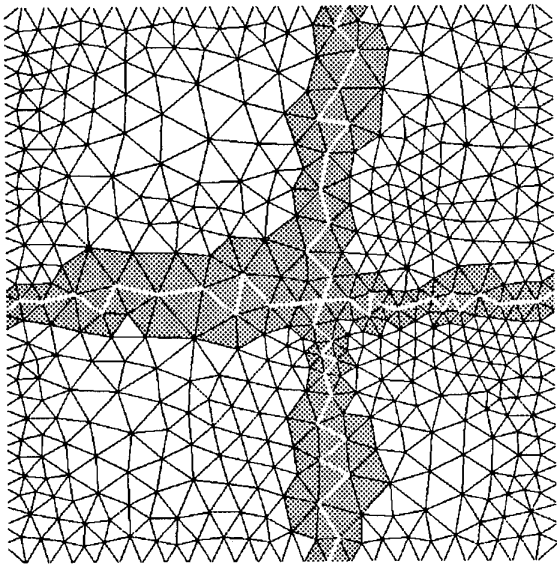


Figure 3.7: A triangulated and partitioned domain exhibiting distance-2 overlap

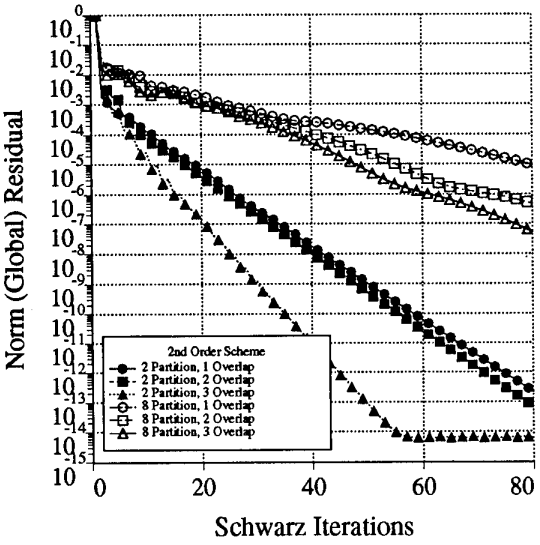


Figure 3.9: Case 1 Inviscid Flow. Variation in Overlap For Multiplicative Schwarz with 2nd order discretization.

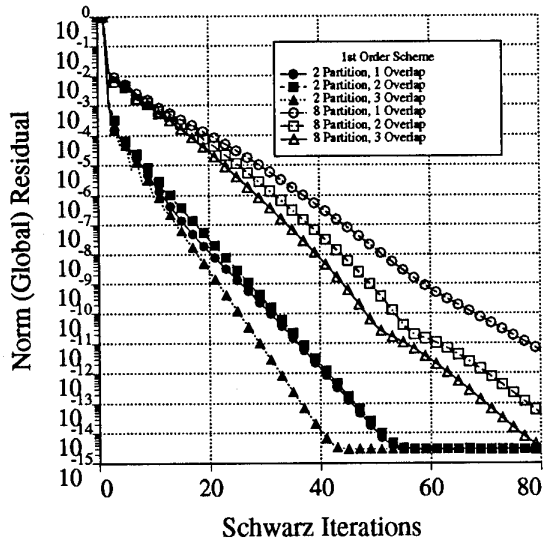


Figure 3.8: Case 1 Inviscid Flow. Variation in Overlap For Multiplicative Schwarz with 1st order discretization.

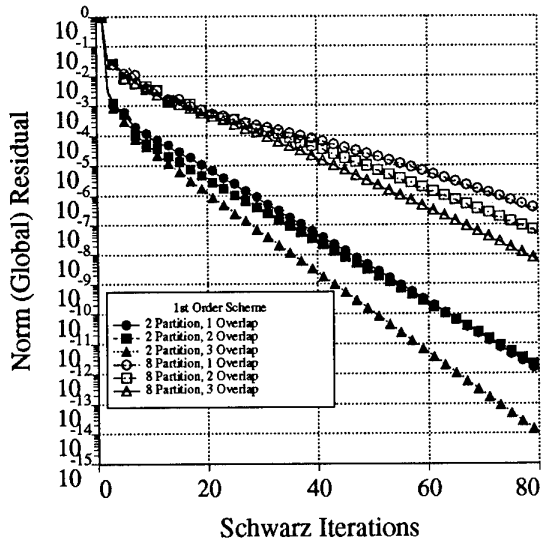


Figure 3.10: Case 1 Inviscid Flow. Variation in Overlap For Additive Schwarz with 1st order discretization.

of 1,2, and 3 and domains partitioned into 2 and 8 subdomains. Using a variant of the uniprocessor algorithm described Chapter 2, each subdomain problem is solved "exactly". In reality, each subdomain problem need only be solved to some reasonable level of convergence. This results in a tremendous savings in computation time. The graphs shown on the right represent

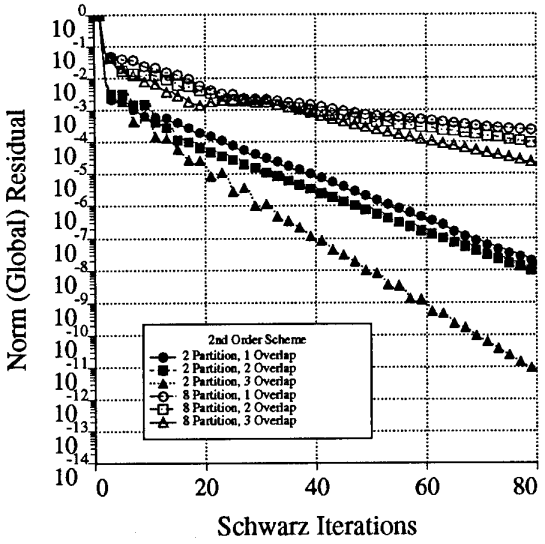


Figure 3.11: Case 1 Inviscid Flow. Variation in Overlap For Additive Schwarz with 2nd order discretization.

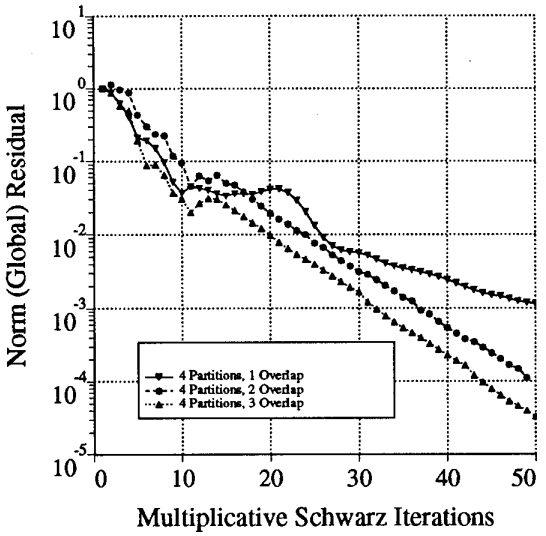


Figure 3.12: Residual history for Case 2 viscous airfoil using multiplicative Schwarz iterations on 4 partitions.

computations using the second order accurate finite-volume scheme with linear reconstruction. The graphs shown on the left represent computations using a first order accurate finite-volume scheme which only requires distance-1 data on the triangulation. The basic trends show a noticeable degradation in the convergence rate with increased partitioning and a mild improvement with increased overlap.

Test Case 2 (Viscous Turbulent Flow)

Using the mesh and flow conditions given previously in Section 2.4.2 viscous turbulent flow was computed about the multi-element airfoil geometry. Figure 3.12 graphs convergence histories for computations using distance 1,2, and 3 overlap on a 4 subdomain partitioning. The improvement with increased overlap is rather slight. In Figures 3.14-3.18 the mesh and Mach number contours for a subdomain near the leading edge of the main element are shown for Schwarz iterations 1,3,5,7, and 40. Note that at iteration 7 the solution visually appears quite close to its final value. Even so, the number of iterations required to reach a comfortable level of convergence is relatively large when compared to the uniprocessor Newton scheme.

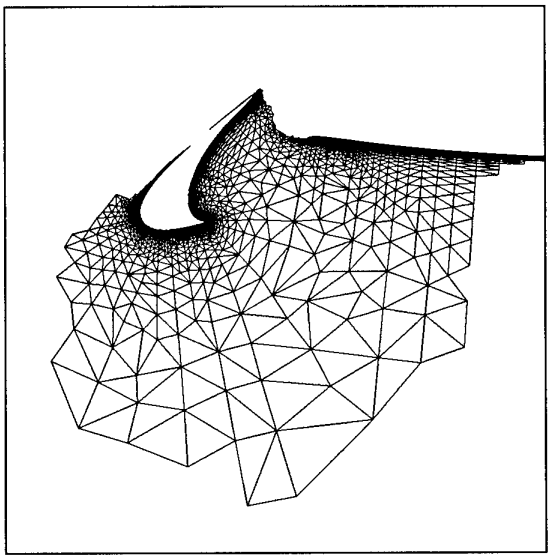


Figure 3.13: Mesh partition for multi-element viscous flow computation.

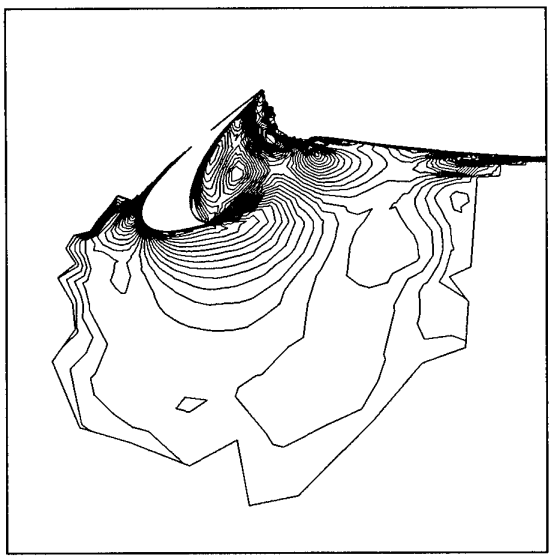


Figure 3.14: Snapshot of solution isomach contours at iteration 1.

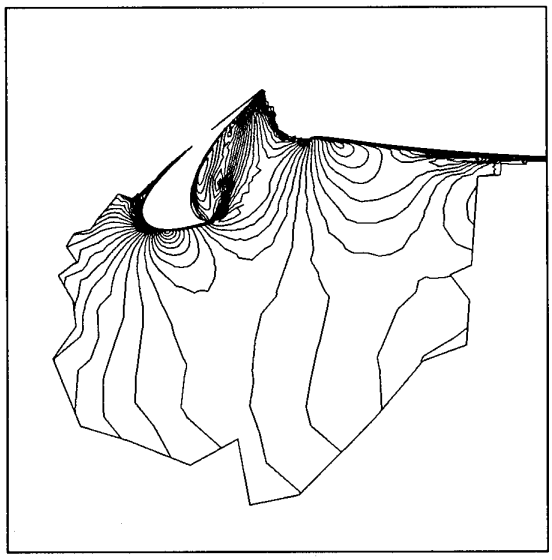


Figure 3.15: Solution isomach contour snapshots at iteration 3.

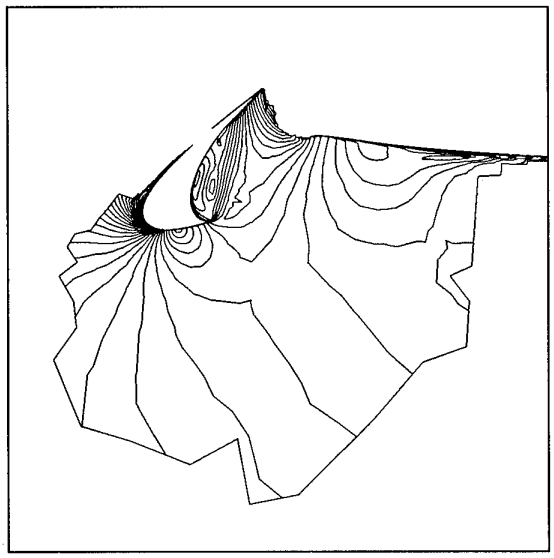


Figure 3.16: Solution isomach contour snapshots at iteration 5.

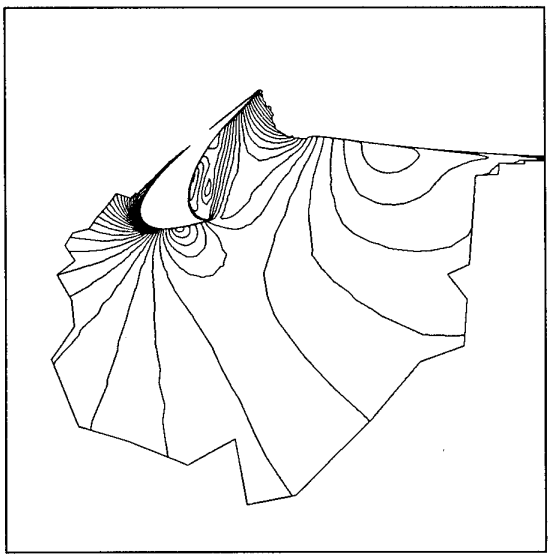


Figure 3.17: Solution isomach contour snapshots at iteration 7.

3.2 Newton's Method with Schwarz Preconditioning

Next we consider using the additive Schwarz precedure to precondition the GMRES and Bi-CGSTAB methods. When used as a preconditioner, some flexibility and compromises can be made which can lead to reduced execution times:

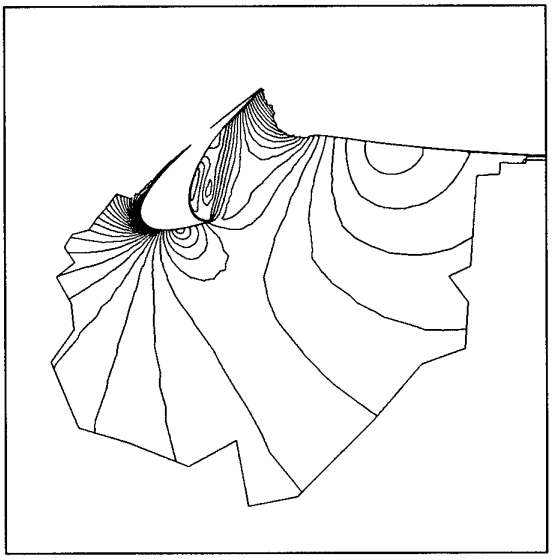


Figure 3.18: Solution isomach contour snapshots at final iteration.

1. Increased Sparsity Preconditioner. This is a common technique for higher order discretization methods. In the present second order finite-volume discretization, the Jacobian matrix contains distance-2 nonzero entries. *For purposes of preconditioning only*, the Jacobian matrix associated with a lower (first) order discretization is used.

2. Inexact Matrix Restriction. Exact matrix restriction performs the task of extracting local submatrices (a gather, scatter operation)

$$A_i = R_i A R_i^T. \qquad (3.41)$$

In some parallel implementations not all data for this calculation is processor resident. This implies communication overhead if an exact computation is to be achieved. In the next section a 3-D parallel implementation is described in which the mesh contains no overlap, yet through communication the scheme performs Schwarz preconditioning exactly equivalent distance-2 overlap. This is sometimes referred to as "implicit" overlap. Using implicit overlap, a compromise is possible in the formation of subdomain preconditioning matrices by neglecting off-processor contributions to on-processor matrix elements. We refer to this as "inexact matrix restriction."

3.2.1 Case 1 (Inviscid flow) Numerical Results.

Using the mesh and flow conditions given in Section 2.4.1, inviscid flow is computed on a 4 subdomain mesh using the Schwarz-type preconditioning of GMRES(20) iterations. All computations were performed using a Newton matrix corresponding to a CFL number of approx 10^8. Figures 3.19 and 3.20 demonstrate the viability of using inexact matrix restriction. In addition, this figure shows the degradation in convergence due to the use of a lower order accurate preconditioning matrix and the enhancement in convergence with increased overlap. Figure 3.21 shows the mild effect of increasing the number of mesh subdomains (4,8,16).

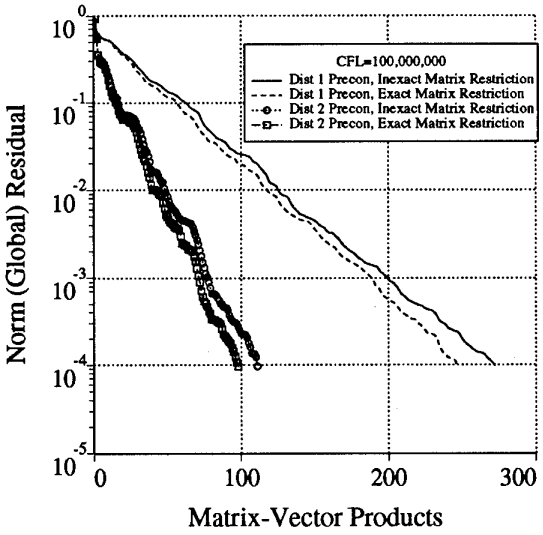


Figure 3.19: Convergence of GMRES(20). Effect of boundary fill strategies on 4 partition mesh with unit overlap.

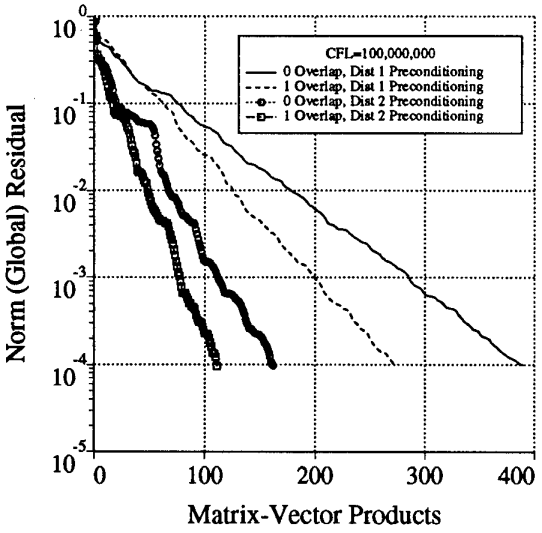


Figure 3.20: Convergence of GMRES(20). Effect of increasing overlap.

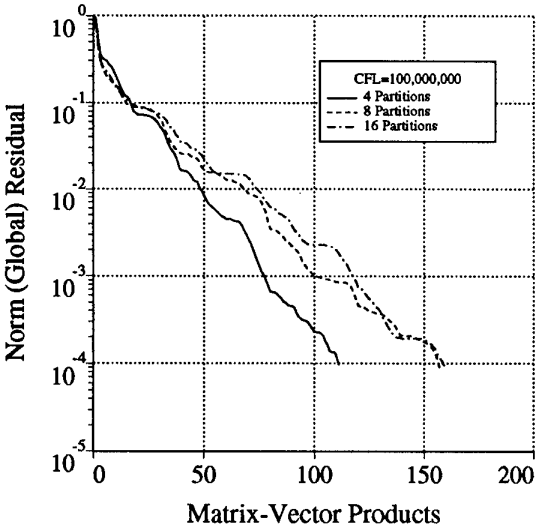


Figure 3.21: Convergence of GMRES(20). Effect of increasing number of partitions, unit overlap, distance 2 preconditioning

3.2.2 Case 2 (Viscous flow) Numerical Results.

Using the mesh and flow conditions given in Section 2.4.2, viscous turbulent flow is computed on a 4 subdomain mesh using the Schwarz-type preconditioning and GMRES(30) iterations. All computations were performed using a Newton matrix corresponding to a CFL number of 10^8. Figure 3.22 shows the resulting convergence histories for the GMRES calculation using single and 4 partitions as well as distance-1 and distance-2 preconditioning matrices. The distance-2 preconditioning works very well for this problem. The distance-1 preconditioning initially reduces the matrix residual norm rapidly but then reverts to a much slower rate of convergence.

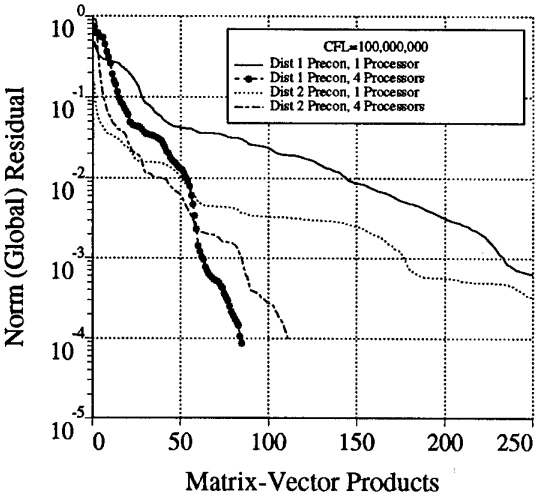


Figure 3.22: Case 2 (viscous flow). Convergence of GMRES(30). Effect of increasing number of partitions, unit overlap

3.3 Some 3-D Computations on the IBM SP2

Our current platform for parallel computation at the NASA Ames Research Center is the IBM SP2 computer. The current configuration consists of 160 rack-mounted IBM 590 workstations with total memory capacity exceeding 20 gigabytes. Each processor has a peak theoretical speed of 250 megaflops. For these computations a single processor attains a sustained speed of about 55 megaflops. The processors are interconnected

via a fast network switch with measured bandwidth of approximately 33 megabytes/second and a measured latency of about 45 microseconds. For maximum portability MPI message passing protocol has been chosen for implementation of the parallel Newton algorithms.

The overall strategy in the parallel implementation is to reduce the entire algorithm to a sequence of steps requiring only distance-one information on the triangulation. The greatly simplifies the implementation of the algorithm while still replicating uniprocessor results. The implementation contains several algorithmic elements. Each of these elements will be described in the following sections and elucidated using the realistic example of fluid flow over a multiple-component wing geometry. The wing geometry, symmetry plane mesh, and Mach contours at a midspan cutting plane are shown in Figure 3.23.

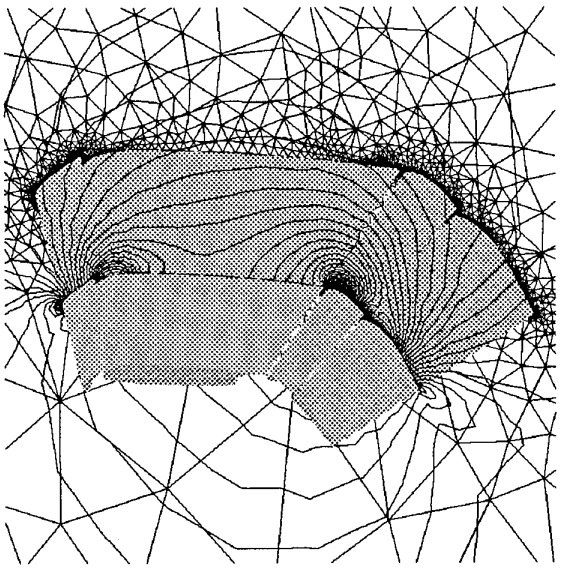


Figure 3.23: Inviscid flow $M_\infty = .20, \alpha = 0$ over a multiple-component wing geometry (600,000 degrees of freedom).

Mesh Partitioning

In the 3-D parallel algorithm, the mesh is *a priori* partitioned into N nonoverlapping subdomains, each of which resides on one of N processors. More precisely, mesh volumes (tetrahedra, hexahedra, prisms, etc) lie entirely on a given partition, triangulation vertices are repeated on partition boundaries, and control volumes for the finite-volume scheme span partition boundaries. The situation is depicted in Figure 3.24. In general we find that the spectral partitioning method outperforms the others but at a higher partitioning cost. Figure 3.25 shows the mesh subdivisions (bold lines) induced by a spectral partitioning on the midspan cutting plane.

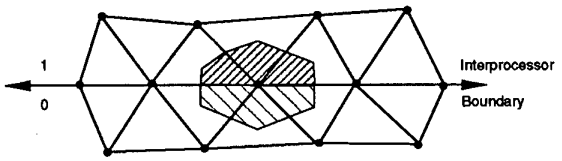


Figure 3.24: Portion of mesh spanning partition boundary showing control volume subdivision.

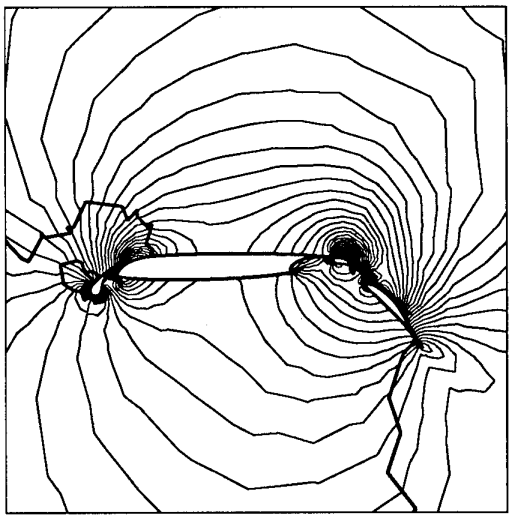


Figure 3.25: Mach Contours and Partition Boundary (bold lines).

Computation of the Explicit Residual and Reconstruction Gradients

For control volumes completely contained within a single partition domain, the calculation of the

residual is identical to the uniprocessor computation. For the control volumes subdivided by partition boundaries, integral conservation implies that

$$\int_{\Omega_0+\Omega_1} (\mathbf{f}\cdot\mathbf{n})\, dS = \int_{\Omega_0} (\mathbf{f}\cdot\mathbf{n})\, dS + \int_{\Omega_1} (\mathbf{f}\cdot\mathbf{n})\, dS. \tag{3.42}$$

In Figure 3.24, Ω_0 and Ω_1 correspond to the control volume portions on processors 0 and 1 respectively such that $\Omega = \Omega_0 \cup \Omega_1$. Therefore residuals can be computed on a processor-by-processor basis followed by an exchange and sum of residuals on interprocessor boundaries. This yields results identical to that obtained on a uniprocessor mesh. The least-squares reconstruction technique also extends in a similar way if a bit mask is assigned to all edges in the mesh so that edges lying on processor boundaries contribute only once to the accumulation formulas.

The GMRES Algorithm

The GMRES algorithm requires three basic operations: vector inner products, matrix-vector products, and preconditioning. The parallel implementation of each of these is described below. In our implementation of GMRES all processors solve the same small least-squares problem. This redundancy is of minor consequence.

Vector Inner Products

Redundancy of boundary vertices in vector inner products is eliminated with a mask bit preassigned to each vertex. The actual inner product is calculated by a local masked inner product followed by a global summation reduction (*MPI_REDUCE*).

Matrix-Vector Products

Previously, we discussed several strategies for computing matrix-vector products in the uniprocessor case. If Fréchet approximate derivatives are used, then the procedure is straightforward and uses exactly the same communication steps needed in computing the explicit residual. If exact matrix-vector products are desired, we store only the matrix associated with the distance-one neighbors on the triangulation and compute the remaining terms in a matrix-free way as discussed in Chapter 2.

Processor Local ILU(0) Preconditioning

Our preconditioning matrix for the GMRES solver is based on the Jacobian matrix of the first-order accurate spatial discretization. This matrix has nonzero entries placed at distance-one locations in the connectivity graph. In a departure from the uniprocessor code, we compute and store on a processor nonzero entries in the matrix associated with mesh vertices residing on that processor. As a second step, the diagonal matrix blocks corresponding to interprocessor boundary vertices are exchanged and summed. This yields diagonal block entries in the resulting processor local matrix that are identical to the corresponding uniprocessor matrix. At the cost of increased interprocessor boundary vertex communication, all processor local matrix entries could be made identical to the uniprocessor matrix entries. The processor local matrix is ILU factored and used for preconditioning GMRES iterations. If these matrix entries are retained then a unique value must be obtained from a linear combination of the multiple computed values. Our experience has shown that the local processor preconditioning does not significantly impact the effectiveness of the ILU preconditioning. In Figure 3.26, we show the convergence of GMRES(12) with local ILU(0) preconditioning on 16, 32, and 64 processors for the multiple-component wing calculation at a CFL number of about 20000.

Keep in mind that this departure from the uniprocessor algorithm only affects the GMRES convergence and not the convergence of Newton's method. Figure 3.27 shows the convergence history of the Newton scheme for the first and second order accurate spatial discretization schemes.

3.3.1 Scalability

The scalability of the current parallel algorithm on the IBM SP2, while not excellent is certainly acceptable. This is particularly true since the parallel algorithm retains the favorable qualities

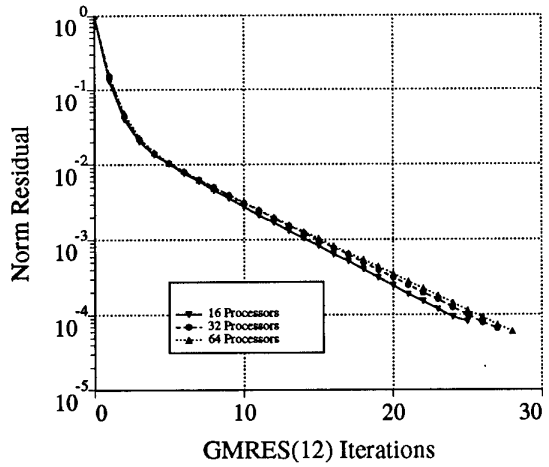


Figure 3.26: GMRES Iterations (restarts) required.

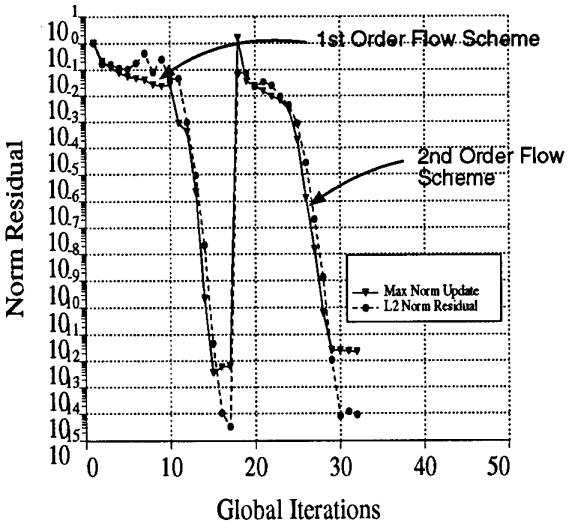


Figure 3.27: Convergence History for Inviscid Multiple Component Wing Case (16 Processors)

of the uniprocessor algorithm, such as Newton-like convergence. Furthermore, because the parallel algorithm makes very few compromises in implementing the uniprocessor algorithm, the primary contribution to the degradation of scalability is the time taken by interprocessor communication. This implies that the scalability would be better on parallel computers with faster interprocessor communication. Figure 3.28 shows

Table 3.1: Wallclock Time in Minutes for the Multiple-Component Wing Calculation

# Procs	First Order Accurate Scheme	Second Order Accurate Scheme
16	50.0	176.0
32	31.0	103.0
64	15.6	55.0

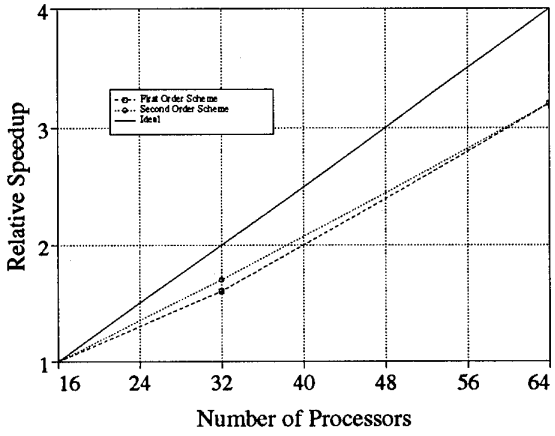


Figure 3.28: Relative speedup of parallel computations using 16, 32, and 64 processor. Each speedup is normalized by the 16 processor value.

the relative speedup of computations on the IBM SP2 for 16, 32, and 64 processors, for both the first order and second order schemes. Once again, the problem being solved is the inviscid flow about a multiple component wing, as described above. The speedups are normalized by the 16 processor value, since the memory requirements made 16 processors a minimum requirement to run the problem. The Table 3.1 shows the total wallclock time in minutes taken by the runs corresponding to Figure 3.28. In each case the second order scheme takes roughly 3.5 times as long as the first order scheme.

3.3.2 3-D Parallel Computation Results

In this section we will present results for a turbulent viscous computation about the multiple component wing described above. The Mach number of the run is 0.2, with a Reynolds number of 5 Million. The angle of attack is 8°. For this run, 64 processors are used. To minimize storage, the Jacobian matrices are stored using single precision (32 bits on the SP2), although all floating point operations are still performed in double precision.

The tetrahedral mesh about the body has roughly 400,000 vertices and over 2,000,000 tetrahedra. Because of the need to resolve the turbulent boundary layer, the mesh is highly stretched near the wall, with cell aspect ratios of more than 10,000. The mesh on the centerline plane is seen in Figure 3.29.

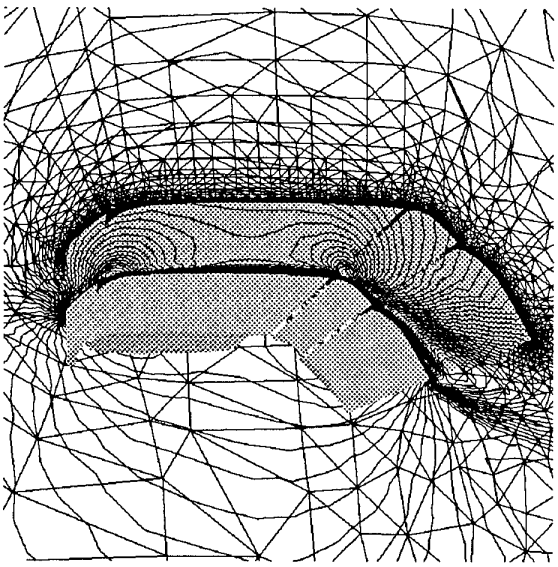


Figure 3.29: Viscous turbulent flow ($M_\infty = .20, \alpha = 8°$, $Re = 5$ Million) over a multiple-component wing. Mach contours are shown on the midspan cutting plane.

The Spalart and Allmaras turbulence model [SA92] is used to simulate the effect of turbulence on the mean flow equations. Although the basic flow equations are solved using linear reconstruction, the turbulence model equation is solved using only first order advection. This is a common procedure used to increase robustness, even in structured mesh codes. The turbulence model is fully coupled with the flow equations in computing the Jacobians. This insures that Newton's method is approached at large timesteps even for turbulent computations.

Figure 3.30 shows the resulting Mach contours on a cutting plane placed at approximately midspan. Note the qualitative agreement between these results and those obtained by the corresponding two-dimensional computation shown in Chapter 2, albeit at a different angle of attack.

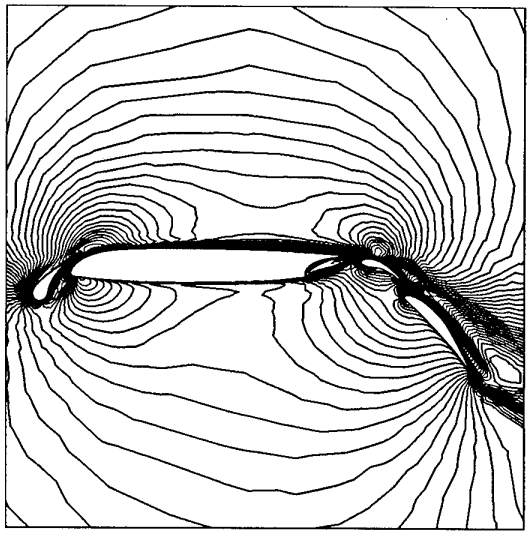


Figure 3.30: Mach Contours on the midspan cutting plane.

In Figure 3.31, contours of the eddy viscosity-like turbulence parameter $\tilde{\nu}$ defined earlier are depicted on the mid-span cutting plane. Note the high levels generated downstream of the main wing element over the aft flaps.

Presently, this computation takes about 10 minutes per step, and about 80 steps to converge to steady-state (a relatively large number for Newton's method). This is due to the slow development of turbulence over the wing. This situation is likely to improve in the near future, as we refine our technique for approaching steady-state and compute on a sequence of

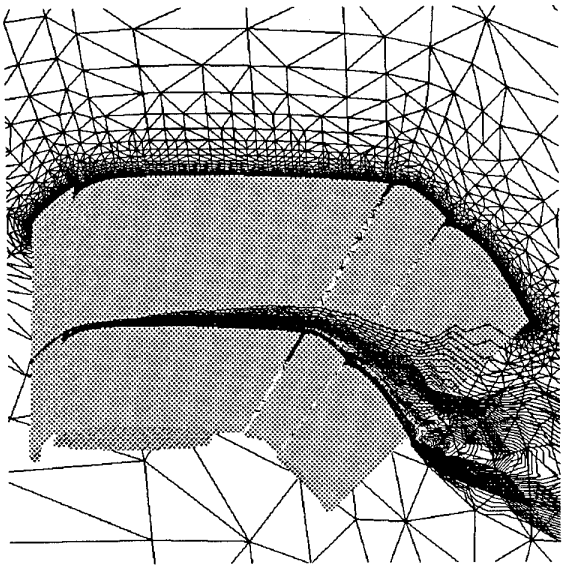


Figure 3.31: Turbulence quantity contours showing the buildup of turbulence over the aft flap.

coarser meshes to accelerate the removal of the initial transient.

Bibliography

[Bar87] T. J. Barth. *Analysis of Implicit Local Linearization Techniques for TVD and Upwind Algorithms.* Technical Report AIAA 87-0595, Reno, NV, 1987.

[Bar91] T. J. Barth. *A Three-Dimensional Upwind Euler Solver of Unstructured Meshes.* Technical Report AIAA 91-1548, Honolulu, Hawaii, 1991.

[Bar94] T. J. Barth. *Aspects of Unstructured Grids and Finite-Volume Solvers for the Euler and Navier-Stokes Equations*, March 1994. von Karman Institute Lecture Series 1994-05.

[Bar95] T. J. Barth. *Steiner Triangulation for Isotropic and Stretched Elements.* Technical Report AIAA 95-0213, Reno, NV, 1995.

[BB90] B. S. Baldwin and T. J. Barth. *A One-Equation Turbulence Transport Model for High Reynolds Number Wall-Bounded Flows.* Technical Report TM-102847, NASA Ames Research Center, Moffett Field, CA, August 1990.

[BJ89] T. J. Barth and D. C. Jespersen. *The Design and Application of Upwind Schemes on Unstructured Meshes.* Technical Report AIAA 89-0366, Reno, NV, 1989.

[BS93] S.T. Barnard and H.D. Simon. A fast multilevel implementation of recursive spectral bisection. In *Proc. 6h SIAM Conf. Parallel Proc. for Sci. Comp.*, pages 711–718. SIAM, 1993.

[BS94] P. Brown and Y. Saad. Convergence Theory of Nonlinear Newton-Krylov Algorithms. *SIAM J. Optimization.*, 4:297–330, 1994.

[CM69] E. Cuthill and J. McKee. Reducing the Band Width of Sparse Symmetric Matrices. *Proc. ACM Nat. Conference*, pages 157–172, 1969.

[CM94] T. Chan and Tarek P. Mathew. *Domain Decomposition Algorithm.* Technical Report CAM 94-2, UCLA Department of Mathematics, January 1994.

[CZ93] T. Chan and J. Zou. *Additive Schwarz Domain Decomposition Methods for Elliptic Problems on Unstructured Meshes.* Technical Report CAM 93-40, UCLA Department of Mathematics, December 1993.

[DW89] M. Dryja and O.B. Widlund. Some domain decomposition algorithms for elliptic problems. pages 273–291. Iterative Methods for Large Linear Systems, 1989.

[DW92] M. Dryja and O.B. Widlund. Additive schwarz methods for elliptic finite element problems in three dimensions. Fifth Conf. on Domain Decomposition Methods for Partial Differential Equations, 1992.

[EW94] S. Eisenstat and H. Walker. Globally Convergent Inexact Newton Meth-

ods. *SIAM J. Optimization.*, 4:393–422, 1994.

[Fri58] K.O. Friedrichs. Symmetric positive linear differential equations. *Comm. Pure and Appl. Math.*, 11:333–418, 1958.

[Geo71] J.A. George. Computer implementation of the finite element method. Technical Report STAN-CS-71-208, Computer Science Dept., Stanford University, Stanford, CA, 1971.

[GJS76] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoret. Comput. Sci.*, 1:237–267, 1976.

[GL81] J. A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1981.

[HL95] B. Hendrickson and R. Leland. An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations. *SIAM J. Sci. Stat. Comput.*, 16(2):452–469, 1995.

[Joh92] Z. Johan. *Data Parallel Finite Element Techniques for Large-Scale Computational Fluid Dynamics.* PhD thesis, Stanford University, Department of Mechanical Engineering, 1992.

[PSL90] A. Pothen, H. D. Simon, and K. P. Liou. Partitioning Sparse Matrices with Eigenvectors of Graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[Qua90] A. Quateroni. *Domain Decomposition Method for the Numerical Solution of Partial Differential Equations.* Technical Report UMSI90/246, Supercomputer Institute, University of Minnesota, 1990.

[Roe81] P. L. Roe. Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. *J. Comput. Phys.*, 43, 1981.

[Ros68] R. Rosen. Matrix Band Width Minimization. pages 585–595. Proceedings of the ACM National Conference, ACM, 1968.

[SA92] P. Spalart and S. Allmaras. *A One-Equation Turbulence Model for Aerodynamic Flows.* Technical Report AIAA 92-0439, Reno, NV, 1992.

[Sch69] H.A. Schwarz. Uber einige abbildungensaufgaben. *J. Reine Angew. Math.*, 70:105–120, 1869.

[Sim91] H. D. Simon. *Partitioning of Unstructured Problems for Parallel Processing.* Technical Report RNR-91-008, NASA Ames Research Center, Moffett Field, CA, 1991.

[Smi92] B. F. Smith. An optimal domain decomposition preconditioner for the finite element solution of linear elasticity. *SIAM J. Sci. Stat. Comput.*, 13:364–378, 1992.

[SS86] Y. Saad and M. H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM J. Sci. Stat. Comp.*, 7(3):856–869, 1986.

[Tal94] P. Le Tallec. Domain decomposition methods in computational mechanics. *Comp. Mech. Adv.*, pages 1–220, 1994.

[VDMG92] W. Valarezo, C. Dominik, R. McGhee, and W. Goodman. *High Reynolds Number Confguration Development of a High-Lift Airfoil.* Technical Report AGARD Meeting In High-Left Aerodynamics 10-01, 1992.

[vdV92] H. van der Vorst. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM J. Sci. Statist. Comput.*, 13:631–644, 1992.

[vL79] B. van Leer. Towards the Ultimate Conservative Difference Schemes V. A Second Order Sequel to Godunov's Method. *J. Comput. Phys.*, 32, 1979.

[VSB92] V. Venkatakrishnan, H. D. Simon, and T. J. Barth. A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids. *J. Supercomput.*, 6:117–137, 1992.

HIGH PERFORMANCE SIMULATION OF COUPLED NONLINEAR TRANSIENT AEROELASTIC PROBLEMS

Charbel Farhat
Department of Aerospace Engineering Sciences
and Center for Aerospace Structures
University of Colorado at Boulder
Boulder, CO 80309-0429, U. S. A.

SUMMARY

Aeroelasticity studies the mutual interaction between aerodynamic and elastic forces for an aerospace vehicle. A flexible aircraft structure immersed in a flow is subjected to surface pressures induced by that flow. If the incident flow or boundary conditions are unsteady, these pressures become time-dependent. Moreover, structural dynamic motions induced by these pressures in turn change the boundary conditions of the flow. The accurate prediction of aeroelastic phenomena such as divergence and flutter is essential in the design of high performance and safe aircrafts. This prediction requires solving *simultaneously* the coupled fluid and structural equations of motion. Therefore, numerical aeroelastic simulations are in general resource intensive. They belong to the family of Grand Challenge engineering problems, and as such, can benefit from the parallel processing technology. This paper highlights some important aspects of nonlinear computational aeroelasticity. These include a three-field arbitrary Lagrangian-Eulerian (ALE) finite element/volume formulation for coupled transient aeroelastic problems, a rigorous derivation of geometric conservation laws (GCLs) for flow problems with moving boundaries and *unstructured deformable* meshes, the design of a family of staggered procedures for the efficient solution of the coupled fluid/structure partial differential equations, and fast parallel domain decomposition solvers. The derivations of the GCLs are presented for ALE based finite volume formulations as well as ALE based stabilized finite element methods. The impact of these GCLs on the numerical algorithms used for time-integrating the semi-discrete equations governing the structural and fluid mesh motions is also discussed. The solution of the governing three-field equations with mixed implicit/implicit and explicit/implicit staggered procedures are analyzed with particular reference to accuracy, stability, subcycling, distributed computing, I/O transfers, and parallel processing. A general and flexible framework for implementing the partitioned analysis of coupled transient aeroelastic problems with non-matching fluid/structure interfaces on heterogeneous and/or parallel computational platforms is also described. This framework and the staggered solution procedures are demonstrated with examples ranging from the numerical investigation on an iPSC-860 massively parallel processor of the instability of flat panels with infinite aspect ratio in supersonic airstreams, to the solution on the Paragon XP/S, Cray

Paper presented in an AGARD-FDP-VKI Special Course on "Parallel Computing in CFD", held at the VKI, Rhode-Saint-Genèse, Belgium, from 15-19 May 1995 and 16-20 October 1995 at NASA Ames, United States and published in R-807.

T3D and IBM SP2 parallel systems of three-dimensional wing response problems in the transonic regime.

1. INTRODUCTION

Aeroelasticity is the study of the effect of aerodynamic forces on elastic bodies. Because these effects have a great impact on performance and safety issues, aeroelasticity has rapidly become one of the most important considerations in aircraft design. The basic mechanism of a fluid/structure interaction phenomenon can be simply explained as follows. The aerodynamic forces acting on an aircraft depend critically on the attitude of its lifting body with respect to the flow, which in turn depends on the flexibility of the aircraft. Therefore, the elastic deformations of a structure play an important role in determining its external loading. Since the magnitude of the aerodynamic forces cannot be known until the elastic deformations are first determined, it follows that the external load cannot be evaluated until the coupled aeroelastic problem is solved.

In general, aeroelastic problems are divided into: (a) *stability*, and (b) *response* problems. Each of these two classes can be further classified into *steady-state* or *static* problems in which the inertia forces may be neglected, and *unsteady*, or *dynamic*, or *transient* problems which are characterized by the interplay of all of the aerodynamic, elastic, and inertia forces. Throughout this paper, we focus exclusively on dynamic aeroelasticity problems.

If one notes that the external aerodynamic forces acting on an aircraft structure increase rapidly with the flight speed, while the internal elastic and inertial forces remain essentially unchanged, one can easily imagine that there may exist a critical flight speed at which the structure becomes unstable. Such instability may cause excessive structural deformations and may lead to the destruction of some components of the aircraft. Panel or wing *flutter*, which is a sustained oscillation of panels or wings caused by the high-speed passage of air along the panel or around the wing, is an example of such instability problems. *Buffeting*, which is the unsteady loading of a structure by velocity fluctuations in the oncoming flow, is another important example. Because of the potentially disastrous character of these phenomena, aircraft flutter and buffeting speeds must be well outside the flight envelope. In many cases, this requirement is the determining factor in the design of wings and tail surfaces.

An aeroelastic response problem can associate with a stability problem. For example, if a control surface of an aircraft is displaced, or a turbulence in the flow is encountered, the response to be found may be the motion, the deformation, or the stress state induced in the elastic body of the aircraft. When the response of the structure to such an input is finite, the structure is stable and flutter will not occur. When the structure flutters, its response to a finite disturbance is unbounded. However, an aeroelastic response problem can also associate with a performance rather than a stability problem. For example, it is well-known that for transonic flows, small variations in incidence may lead to considerable changes in the pressure distribution, shock position, and shock strength. It is also well-known that there are some margins within the Mach number and incidence that can be varied around the design condition of a supercritical airfoil without a serious deterioration of the favorably low-drag property of the shock-free flow condition [1]. Determining whether an oscillating airfoil is within or outside these margins requires determining its aeroelastic response.

Past literature on aeroelasticity is mostly devoted to linear models where the motion of a gas or a fluid past a structure, the deformation and vibration of that structure, and more importantly the interaction phenomenon

itself are described with linear mathematical concepts [2,3]. Even experimental results are often interpreted by assuming a linear behavior of the physical model. However, just as swimming in a pool is a prerequisite for swimming in an ocean, understanding linear aeroelasticity problems is essential for solving nonlinear ones. Next, we summarize the linear theory of aeroelasticity.

1.1. Linear Theory of Aeroelasticity

The fundamental assumptions behind the linear formulation and solution of transient aeroelastic problems are

- the structure is elastic.
- it undergoes a *harmonic* motion with *small displacement amplitudes.*
- the flow can be approximated by a linearized theory.

Under the above conditions, given a free-stream Mach number M_∞, the aerodynamic forces acting on an aircraft elastic structure immersed in an unsteady flow can be written as

$$\mathbf{F}(t) = -\lambda(\mathbf{A}_1\mathbf{q}(t) + \mathbf{A}_2\dot{\mathbf{q}}(t)) + \mathbf{F}_0(t) \quad (1)$$

where $\lambda(\mathbf{A}_1\mathbf{q}(t) + \mathbf{A}_2\dot{\mathbf{q}}(t))$ represents the aerodynamic forces generated by the transient motion of the flexible structure, and $\mathbf{F}_0(t)$ represents the unsteady aerodynamic forces that would have been generated if the aircraft had a rigid rather than elastic structure. Here, t denotes time, λ is the dynamic pressure, $\mathbf{A}_1$ and $\mathbf{A}_2$ denote the linear aerodynamic operators accounting for the surrounding flow and computed for a given M_∞ and a unit dynamic pressure, and the time-dependent vector $\mathbf{q}(t)$ represents the discretized structural displacements. Because these displacements are assumed to have small amplitudes, the governing equations of dynamic equilibrium of the aircraft elastic structure can be written as

$$\mathbf{M}\ddot{\mathbf{q}}+\mathbf{D}\dot{\mathbf{q}}+\mathbf{K}\mathbf{q} = -\lambda(\mathbf{A}_1\mathbf{q}(t)+\mathbf{A}_2\dot{\mathbf{q}}(t))+\mathbf{F}_0(t) \quad (2)$$

where a dot superscript denotes a time derivative, and $\mathbf{M}$, $\mathbf{D}$, and $\mathbf{K}$ are respectively the symmetric positive mass, damping, and stiffness matrices associated with the discretized structure — for example, but not necessarily, via finite elements. Eq. (2) above can be rearranged as follows

$$\mathbf{M}\ddot{\mathbf{q}}+(\mathbf{D}+\lambda\mathbf{A}_2)\dot{\mathbf{q}}+(\mathbf{K}+\lambda\mathbf{A}_1)\mathbf{q} = \mathbf{F}_0(t) \quad (3)$$

If the flow is steady, $\mathbf{F}_0$ does not vary with time, and the solution of the above problem can be decomposed into a steady and unsteady components

$$\mathbf{q}(t) = \mathbf{q}^s + \mathbf{q}^u(t) \quad (4)$$

where $\mathbf{q}^s$ is solution of

$$(\mathbf{K} + \lambda\mathbf{A}_1)\mathbf{q}^s = \mathbf{F}_0 \quad (5)$$

and $\mathbf{q}^u(t)$ is solution of

$$\mathbf{M}\ddot{\mathbf{q}}^u+(\mathbf{D}+\lambda\mathbf{A}_2)\dot{\mathbf{q}}^u+(\mathbf{K}+\lambda\mathbf{A}_1)\mathbf{q}^u = 0 \quad (6)$$

Eq. (5) is the governing equation for static aeroelasticity, where the central problem is the effect of elastic deformation on the lift distribution over lifting surfaces such as airplane wings and tails. At higher speeds, the effect of elastic deformation can become important enough to cause a wing to become unstable, to render a control surface ineffective, or even worse to reverse the sense of control. The first phenomenon is known as *divergence*, and the last as *aileron reversal.* Mathematically, the divergence speed can be obtained from the investigation of the values of λ for which the matrix $(\mathbf{K} + \lambda\mathbf{A}_1)$ becomes singular. On the other hand, Eq. (6) is the governing equation of aeroelastic dynamic stability (or instability). The flutter dynamic pressure corresponds to the critical value λ^{cr} beyond which Eq. (6) has a solution $\mathbf{q}^u(t)$ that grows continuously in time. This value λ^{cr} of the dynamic pressure defines

the stability limit of the solution of Eq. (6). Beyond this critical value, the elastic structure will continuously extract energy from the surrounding flow and become dynamically unstable. For dynamic pressure values below λ^{cr}, the structure will release energy to the surrounding flow which will act as a damper.

If the flow is unsteady, Eq. (3) becomes the governing equation for the dynamic aeroelastic response problem, and its homogeneous counterpart

$$\mathbf{M}\ddot{\mathbf{q}} + (\mathbf{D} + \lambda\mathbf{A}_2)\dot{\mathbf{q}} + (\mathbf{K} + \lambda\mathbf{A}_1)\mathbf{q} = 0 \quad (7)$$

becomes the governing equation for the aeroelastic dynamic stability problem. Note that each of Eq. (3) and Eq. (7) represents a system of n coupled second-order differential equations, where n is the size of the square matrices $\mathbf{M}$, $\mathbf{D}$, $\mathbf{K}$, $\mathbf{A}_1$ and $\mathbf{A}_2$, and is equal to the number of structural degrees of freedom (d.o.f.) introduced in the computational structural model. For a detailed structural wing model or a complete aircraft configuration, n can be as large as a hundred thousand, and therefore solving directly Eq. (3) for the aeroelastic response $\mathbf{q}(t)$ or Eq. (7) for the flutter dynamic pressure λ^{cr} becomes a formidable task. For this reason, Eq. (3) and/or Eq. (7) are usually projected onto an m-dimensional subspace ($m << n$) represented by its basis $\mathbf{\Psi}_m = [\psi_1, \psi_2, ..., \psi_m]$. This basis is called a *modal* basis because each column vector ψ_j is an eigenvector of the generalized symmetric eigenvalue problem

$$\mathbf{K}\psi = \omega^2\mathbf{M}\psi \quad (8)$$

and therefore each ψ_j is a *mode shape* of the structure. The above generalized symmetric eigenvalue problem admits n eigenpairs $\{\omega_j^2, \psi_j\}_{j=1}^{j=n}$ where ω_j is the *circular frequency* associated with the mode shape ψ_j. This problem arises when the conservative structural system

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{K}\mathbf{q} = 0 \quad (9)$$

is considered, and harmonic solutions of the form $\mathbf{q}(t) = \psi e^{i\omega t}$ are sought. Here and throughout this section, i denotes the complex number satisfying $i^2 = -1$. If the ψ_j eigenvectors are mass normalized, from Eq. (8) and the symmetry properties of $\mathbf{M}$ and $\mathbf{K}$, it follows that

$$\mathbf{\Psi}_m^T\mathbf{K}\mathbf{\Psi}_m = \mathbf{\Omega}_m^2 = \begin{bmatrix} \omega_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \omega_m^2 \end{bmatrix} \quad (10)$$

$$\mathbf{\Psi}_m^T\mathbf{M}\mathbf{\Psi}_m = \mathbf{I}_m$$

where the superscript T designates the transpose operation, and $\mathbf{I}_m$ denotes the $m \times m$ identity matrix. Hence, projecting $\mathbf{q}(t)$ onto the modal basis $\mathbf{\Psi}_m$

$$\mathbf{q}(t) = \mathbf{\Psi}_m\mathbf{y}(t) \quad (11)$$

substituting the above expression in Eq. (3), premultiplying that equation by $\mathbf{\Psi}_m^T$, and exploiting the relationships given in Eqs. (10) leads to the modal equations of equilibrium

$$\mathbf{I}_m\ddot{\mathbf{y}} + (\mathbf{D}_m + \lambda\mathbf{A}_{2_m})\dot{\mathbf{y}} + (\mathbf{\Omega}_m^2 + \lambda\mathbf{A}_{1_m})\mathbf{y} = \mathbf{F}_{0_m}(t) \quad (12)$$

where

$$\begin{aligned} \mathbf{D}_m &= \mathbf{\Psi}_m^T\mathbf{D}\mathbf{\Psi}_m \\ \mathbf{A}_{1_m} &= \mathbf{\Psi}_m^T\mathbf{A}_1\mathbf{\Psi}_m \\ \mathbf{A}_{2_m} &= \mathbf{\Psi}_m^T\mathbf{A}_2\mathbf{\Psi}_m \\ \mathbf{F}_{0_m}(t) &= \mathbf{\Psi}_m^T\mathbf{F}_0(t) \end{aligned} \quad (13)$$

and $\mathbf{y}(t)$ is known as the vector of generalized or modal coordinates. If the so-called Rayleigh structural damping is used ($\mathbf{D} = a\mathbf{M} + b\mathbf{K}$, $a \geq 0$, $b \geq 0$), or a modal damping is assumed for the structure, $\mathbf{D}_m$ also becomes a diagonal matrix. However, $\mathbf{A}_{1_m}$ and $\mathbf{A}_{2_m}$ are in general $m \times m$ full matrices.

In summary, even though projecting Eq. (3) and/or Eq. (7) onto the modal basis $\mathbf{\Psi}_m$

does not completely uncouple the n second-order differential equations because of the presence of the aerodynamic operators $\mathbf{A}_1$ and $\mathbf{A}_2$, this procedure is still attractive because it reduces the number of coupled ordinary differential equations to be solved from n to $m << n$.

If an aeroelastic response problem is investigated, Eq. (12) is usually solved for $\mathbf{y}(t)$ using a numerical time integration algorithm. Then, the structural displacement field $\mathbf{q}(t)$ is recovered by making use of Eq. (11). However, it should be noted that Eq. (11) can also be written as

$$\mathbf{q}(t) = \mathbf{\Psi}_m \mathbf{y}(t) = \sum_{r=1}^{r=m<<n} \psi_r y_r(t) \quad (14)$$

which highlights the fact that $\mathbf{q}(t)$ is a *truncated* modal solution of the original Eq. (3). Aside from time discretization errors, the accuracy of such a solution depends on the importance of the contributions to the exact response of the structure of the truncated mode shapes or eigenvectors. In other words, it depends on the load distribution of the aircraft and the frequency content of the aeroelastic response of the structure. For wing flutter problems, the behavior of the structure is often dominated by low frequency dynamics, and therefore is well represented by the first few modes. In that case, only the first few eigenvectors ψ_j are usually kept in the modal basis $\mathbf{\Psi}_m$, and the truncated modal superposition method delivers an accurate solution of the dynamic aeroelastic response problem.

On the other hand, if an aeroelastic dynamic stability problem is investigated, the homogeneous form of Eq. (12) is solved for the flutter dynamic pressure λ^{cr}. One methodology for obtaining λ^{cr} goes as follows. Let V_∞ denote the free-stream velocity (flight speed), and ρ_∞ the free-stream air density. We have

$$\lambda = \frac{1}{2}\rho_\infty V_\infty^2 \quad (15)$$

When the structure undergoes a harmonic motion characterized by a circular frequency $\bar{\omega}$, the linear aerodynamic operators $\mathbf{A}_1$ and $\mathbf{A}_2$ become a function of the reduced frequency $\bar{k}$

$$\begin{aligned} \mathbf{A}_1 &= \mathbf{A}_1(\bar{k}) \\ \mathbf{A}_2 &= \mathbf{A}_2(\bar{k}) \\ \bar{k} &= \frac{\bar{\omega}}{V_\infty} \end{aligned} \quad (16)$$

If structural damping is neglected, seeking a solution of the homogeneous form of Eq. (12) of the form

$$\begin{aligned} \mathbf{y}(t) &= \bar{\mathbf{y}} e^{i\bar{\omega}t} \\ \bar{\omega} &= \omega(1 + i\alpha) \quad |\alpha| << 1 \end{aligned} \quad (17)$$

leads to

$$\begin{aligned} [-\bar{\omega}^2 \mathbf{I}_m + \mathbf{\Omega}_m^2 + \frac{1}{2}\rho_\infty V_\infty^2 \mathbf{A}_m(\frac{\bar{\omega}}{V_\infty})]\, \bar{\mathbf{y}} &= 0 \\ \mathbf{A}_m &= \mathbf{A}_1(\frac{\bar{\omega}}{V_\infty}) + i\bar{\omega}\mathbf{A}_2(\frac{\bar{\omega}}{V_\infty}) \end{aligned} \quad (18)$$

Note that the first of Eqs. (17) can be rearranged as

$$\mathbf{y}(t) = \bar{\mathbf{y}} e^{-\alpha\omega t} e^{i\omega t} \quad (19)$$

which shows that the homogeneous form of Eq. (12) will have a stable solution if and only if all of the solutions $\bar{\omega}$ of Eq. (18) have a positive real part $\alpha \geq 0$. Therefore, $\alpha = 0$ is the stability limit, and the sought after flutter dynamic pressure λ^{cr} corresponds to the critical value V_∞^{cr} of the flight speed, or the critical value $\bar{k}^{cr} = \bar{\omega}/V_\infty^{cr}$ of the reduced frequency, for which Eq. (18) admits a real solution $\bar{\omega} = \omega(1 + i \times 0) = \omega$.

From the second of Eqs. (17), it follows that

$$\frac{1}{\bar{\omega}^2} \approx \frac{1}{\omega^2} - i\frac{2\alpha}{\omega^2} \quad (20)$$

Hence, substituting Eq. (20) into Eq. (18), making use of the third of Eqs. (16), and exploiting the assumption $|\alpha| << 1$ finally gives

$$\begin{aligned} \mathbf{Z}_m(\bar{k})\,\bar{\mathbf{y}} &= (\frac{1}{\omega^2} - i\frac{2\alpha}{\omega^2})\,\bar{\mathbf{y}} \\ \mathbf{Z}_m(\bar{k}) &= \mathbf{\Omega}_m^{2\,-1}(\mathbf{I}_m - \frac{1}{2}\frac{\rho_\infty}{k^2}\mathbf{A}_m(\bar{k})) \end{aligned} \tag{21}$$

which shows that $(1 - i2\alpha)/\omega^2$ is a complex eigenvalue of a matrix $\mathbf{Z}_m$ that, for a fixed free-stream air density ρ_∞, depends only on the reduced frequency $\bar{k}$. Therefore, the flutter dynamic pressure $\lambda^{cr} = \rho_\infty V_\infty^{cr^2}/2$ can be found by sweeping over the values of $\bar{k}$, and solving for each $\bar{k}$ the eigenvalue problem (21). Among all possible critical values of the reduced frequency $\bar{k}^{cr}$ for which a real eigenvalue $1/\omega^{2^{cr}}$ is found — and therefore for which α vanishes — the flutter speed is given by the smallest value $V_\infty^{cr} = \omega^{cr}/k^{cr}$, and the flutter dynamic pressure by the corresponding $\lambda^{cr} = \rho_\infty V_\infty^{cr^2}/2$. This procedure is known as the "k" method, or the "k-sweeping" method. It is implemented in many industrial codes (see, for example, [4]). It is accurate when the structure is less than 10% damped. When the structure has a higher percentage of damping, other methods such as the "p-k" method [5] can be used for finding the flutter dynamic pressure λ^{cr}. Such methods are in general more expensive than the "k" method and are beyond the scope of this paper.

At this point, the reader should recall that the linear aerodynamic operators $\mathbf{A}_1$ and $\mathbf{A}_2$ are computed for a specified free-stream Mach number, and therefore V_∞^{cr} is also computed for a specified M_∞ (and a specified free-stream air density ρ_∞). This implies that for each value of M_∞, there exists a critical free-stream speed of sound $c_\infty^{cr} = V_\infty^{cr}/M_\infty$, and that a curve $c_\infty^{cr} = c_\infty^{cr}(M_\infty)$ can be determined. The intersection of this curve with the horizontal line $c_\infty^{cr} = 320m/s$ gives the critical free-stream Mach number M_∞^{cr}.

So far, the derivation of the linear aerodynamic operators $\mathbf{A}_1$ and $\mathbf{A}_2$ has not been discussed. It has only been stated that a linearized flow theory and a harmonic motion of the structure with small displacement amplitudes are assumed. More precisely, $\mathbf{A} = \mathbf{A}_1 + i\bar{\omega}\mathbf{A}_2$ can be computed using the doublet-lattice method [6] in the subsonic regime, and the potential gradient method [7], or the harmonic gradient method [8], or the piston theory [2] in the supersonic regime. In all cases, the flow is assumed to be inviscid, irrotational, and isentropic. In the transonic regime, the mixed subsonic-supersonic flow patterns and shock waves are such that there are no reliable theoretical means for predicting the unsteady aerodynamic forces. In that case, the linear aeroelasticity theory simply breaks down. This is most unfortunate because of the current renewed interest in transonic flight for both military (F-16) and civil aircraft.

Besides transonic flights, there are many other important cases where the linear aeroelastic theory cannot be used for predicting the dynamic response or stability of an aircraft. These include, to name only a few, problems where the structure undergoes large displacements and/or rotations — as an example, we note that the maximum upward deflection of the wing of the B52 bomber is 22 feet [2] — parachute dynamics, bluff body oscillators, airfoil oscillations in separated flow, buffeting, and high-G and high angle of attack maneuvers such as those performed by the X-31 aircraft. Some of these and related problems are discussed in [9] where emphasis is placed on the fundamental understanding of the nonlinear theory of interaction, others are still unresolved. The pressing need for solving and understanding all of these problems is the main motivation for designing a reliable nonlinear transient aeroelastic numerical simulation capability.

1.2. Formulation of Coupled Nonlinear Aeroelastic Problems

Here, the structure is no longer restricted to a harmonic motion with small displacement amplitudes. In principle, there is also no reason to confine its constitutive modeling to that of an elastic material. However, while aircraft structures can undergo large displacements and rotations, they seldom experience large strains. Therefore, the nonlinear modeling of the structural behavior can be limited to the proper accounting of nonlinear geometric effects without a serious loss of generality.

More importantly, the aerodynamic forces acting on the structure are no longer predicted here by the use of a linear aerodynamic operator because of the important limitations associated with such an approach and discussed at the end of Section 1.1. Rather, these unsteady forces are determined from the solution of the compressible Euler equations when viscous effects are neglected, and the solution of the compressible Navier-Stokes equations otherwise. Furthermore, no restriction is imposed on the nature of the fluid/structure coupling, at least in principle. This coupling is numerically modeled by suitable fluid/structure interface boundary conditions. Clearly, this means that the methodology described here for simulating nonlinear transient aeroelastic problems is based on the simultaneous solution of the governing nonlinear fluid and structure equations, and as such, is computationally intensive and can benefit from parallel processing.

One difficulty in handling numerically the fluid/structure coupling stems from the fact that the structural equations are usually formulated with material (Lagrangian) co-ordinates, while the fluid equations are typically written using spatial (Eulerian) co-ordinates. Therefore, a straightforward approach to the solution of the coupled fluid/structure dynamic equations requires moving at each time-step at least the portions of the fluid grid that are close to the moving structure. This can be appropriate for small displacements of the structure but may lead to severe grid distorsions when the structure undergoes large motion. Several different approaches have emerged as an alternative to partial regridding in transient aeroelastic computations, among which we note the arbitrary Lagrangian/Eulerian (ALE) formulation [10–12], the co-rotational approach [13,14], dynamic meshes [15] which are closely related to ALE concept, interpolation based methods [16], and space-time formulations [17]. All of these approaches treat a computational aeroelastic problem as a coupled two-field problem.

However, a moving mesh (Fig. 1) can also be viewed as a pseudo-structural system with its own dynamics [18], and therefore, the coupled transient aeroelastic problem can be formulated as a *three-* rather than two-field problem: the fluid, the structure, and the dynamic mesh (Fig. 2). The semi-discrete equations governing this three-way coupled problem can be written as follows:

$$\begin{aligned} \frac{\partial}{\partial t}(\mathbf{V}(x,t)\ \mathbf{W}(x,t)) + \mathbf{F}^c(\mathbf{W}(x,t),x,\dot{x}) & \\ = \mathbf{R}(\mathbf{W}(x,t)) & \\ \mathbf{M}\frac{d^2\mathbf{q}}{dt^2} + \mathbf{f}^{int}(\mathbf{q}) = \mathbf{f}^{ext}(\mathbf{W}(x,t),x) & \\ \widetilde{\mathbf{M}}\frac{d^2\mathbf{x}}{dt^2} + \widetilde{\mathbf{D}}\frac{d\mathbf{x}}{dt} + \widetilde{\mathbf{K}}\mathbf{x} = \widetilde{\mathbf{K}}_c\ \mathbf{q} & \end{aligned} \tag{22}$$

where x is the *displacement or position, depending on the context of the sentence* of a moving fluid grid point, $\mathbf{W}$ is the fluid state vector, $\mathbf{V}$ results from the finite element/volume discretization of the fluid equations, $\mathbf{F}^c$ is the vector of convective ALE fluxes that depend on the fluid grid velocity, $\mathbf{R}$ is the vector of diffusive fluxes, $\mathbf{q}$ is as before the structural displacement vector, $\mathbf{f}^{int}$ denotes the vector of internal structural forces that is equal to $\mathbf{Kq}$ in the linear case, $\mathbf{f}^{ext}$ the vector of external forces acting on

the structure, $\mathbf{M}$ is the finite element mass matrix of the structure, $\widetilde{\mathbf{M}}$, $\widetilde{\mathbf{D}}$, and $\widetilde{\mathbf{K}}$ are fictitious mass, damping, and stiffness matrices associated with the fluid moving grid (Fig. 3) and constructed to avoid any parasitic interaction between the fluid and its grid, or the structure and the moving fluid grid [18], and $\widetilde{\mathbf{K}}_c$ is a transfer matrix that describes the action of the motion of the structural side of the fluid/structure interface on the fluid dynamic mesh [19]. For example, $\widetilde{\mathbf{M}} = \widetilde{\mathbf{D}} = 0$, and $\widetilde{\mathbf{K}} = \widetilde{\mathbf{K}}^R$ where $\widetilde{\mathbf{K}}^R$ is a rotation matrix corresponds to a rigid mesh motion of the fluid grid around an oscillating airfoil, and $\widetilde{\mathbf{M}} = \widetilde{\mathbf{D}} = 0$ includes as particular cases the spring-based mesh motion scheme introduced in [15] and the continuum based updating strategy advocated by several investigators (see, for example, [17]).

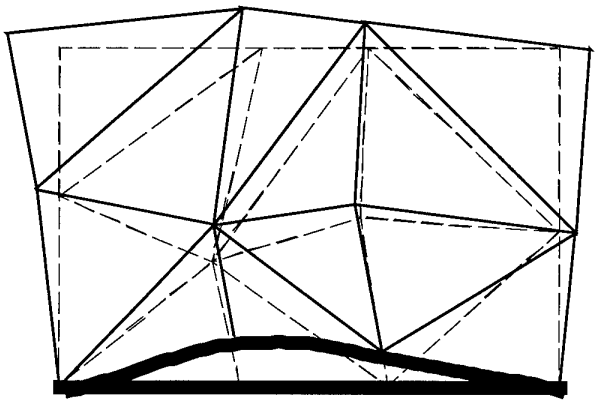


Fig. 1. Moving and deforming fluid grid

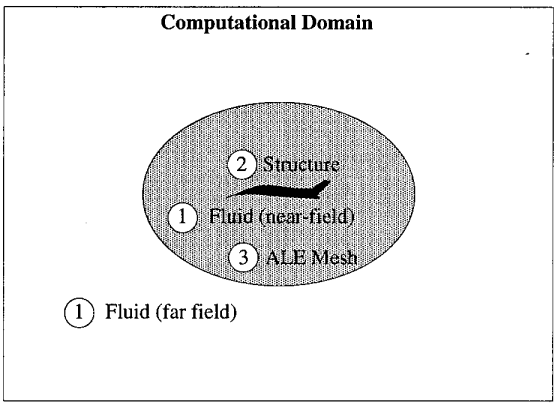


Fig. 2. Three-field formulation

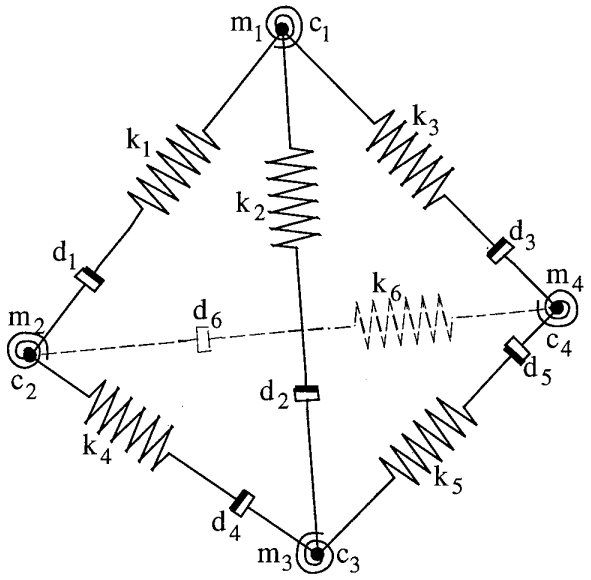


Fig. 3. A pseudo-structural tetrahedron in a fluid mesh

The first of Eqs. (22) is derived in details in Section 2. The second of Eqs. (22) is the standard nonlinear structural dynamics equation of equilibrium. The notation $\mathbf{f}^{ext}(\mathbf{W}(t), \mathbf{x})$ is used to remind the reader that the external forces acting on the structure include, among others, the aerodynamic forces that are computed from the knowledge of the fluid state vector $\mathbf{W}$ and the motion and deformation of the surface of the structure, which in turn controls the motion $\mathbf{x}(t)$ of the fluid grid. Hence, Eqs. (22) are fully coupled.

1.3. Objectives and outline of this paper

Each of the three components of the three-way coupled problem described by Eqs. (22) has different mathematical and numerical properties, and distinct software implementation requirements. For Euler and Navier-Stokes flows, the fluid equations are nonlinear. The structural equations and the semi-discrete equations governing the pseudo-structural fluid grid system may be linear or nonlinear. The matrices resulting from a linearization procedure are in general symmetric for the structural problem, but they

are typically unsymmetric for the fluid problem. Morevoer, the nature of the coupling in Eqs. (22) is implicit rather than explicit, even when the fluid mesh motion is ignored. The fluid and the structure interact only at their interface, via the pressure and viscous forces, and the motion of the physical interface. However, for Euler and Navier-Stokes compressible flows, the pressure variable cannot be easily isolated neither from the fluid equations nor from the fluid state vector **W**. Consequently, the numerical solution of Eqs. (22) via a fully coupled monolithic scheme is computationally challenging and software-wise unmanageable.

Alternatively, Eqs. (22) can be solved via a *partitioned* analysis or a *staggered* procedure [20–23]. This approach offers several appealing features including the ability to use well established discretization and solution methods within each discipline, simplification of software development efforts, and preservation of software modularity.

Traditionally, nonlinear transient aeroelastic problems have been solved via the simplest possible partitioned analysis whose cycle can be described as follows: a) advance the structural system under a given pressure load, b) update the fluid mesh accordingly, and c) advance the fluid system and compute a new pressure load [15,16,24–27]. Occasionally, some investigators have advocated the introduction of a few predictor/corrector iterations within each cycle of this three-step staggered integrator in order to improve accuracy [28], especially when the fluid equations are nonlinear and treated implicitly [29]. However, more efficient staggered solution procedures can and should be devised.

The main objective of this paper is to present a computational framework for the massively parallel solution of the three-way coupled Eqs. (22) that is being developed at the University of Colorado by the author and his coworkers. This is certainly not to imply that we are the only research group working on this problem. However, we believe that our computational framework includes many innovative ideas and unique capabilities that are worthy discussing. For this purpose, the remainder of this paper is organized as follows.

At the heart of nonlinear transient aeroelastic simulations is the computation of unsteady flow problems with moving boundary conditions and dynamic unstructured meshes. In this paper, we do not discuss the state-of-the-art of unsteady flow solvers, especially that their status seems to be far from satisfactory [30]. For this specific topic, we refer the reader to references [30,31]. However, we focus in Section 2 on the important issues of geometric conservation laws (GCLs) which, in the presence of dynamic meshes, impose important constraints on the algorithms employed for time-integrating the semi-discrete equations governing the fluid and dynamic mesh motions. In particular, we address the problem of satisfying both displacement and velocity continuity constraints between the structure and fluid mesh motions at the fluid/structure interface, and the impact of this problem on the accuracy and stability of the time-integrator selected for predicting the aeroelastic structural response. In Section 3, we present a broad family of staggered solution procedures where the fluid flow can be integrated using either an implicit or an explicit scheme, and the structural response is advanced using an implicit one. We address important issues pertaining to numerical stability, subcycling, accuracy vs. speed trade-offs, implementation on heterogeneous computing platforms, and inter-field as well as intra-field parallel processing. Next, we describe in Section 4 our particular two- and three-dimensional unsteady flow solvers. In Section 5, we discuss the solution of the structural dynamics equations. Because our goal is to handle linear as well as nonlinear structural dynamics problems, we opt for a direct time integration method

rather than the restrictive modal superposition approach. We describe a substructure based nonlinear time integration implicit algorithm that features second-order accuracy and unconditional stability. For scalability purposes, we also adopt as a linearized solver the substructure based preconditioned conjugate gradient FETI method [32,33] equipped with the projection scheme presented in [34] for solving iteratively and efficiently systems with repeated right hand sides. In general, the fluid and structure meshes have two independent representations of the physical fluid/structure interface, and do not necessarily match at that interface. We discuss this and other related issues in Section 6 where we also describe "Matcher" [35], a program for generating in parallel the data structures needed for handling arbitrary and non-conforming fluid/structure interfaces in aeroelastic computations. In Section 7, we turn to the solution of the equations governing the dynamic motion of the fluid grid. In Section 8, we describe a unified and portable approach for parallel fluid/structure computations that is based on the mesh partitioning paradigm. We also briefly discuss the controversial topic of what constitutes a good mesh partition for parallel processing. In Section 9, we illustrate our framework for computational dynamic aeroelasticity with examples ranging from the numerical investigation on an iPSC-860 massively parallel processor of the instability of flat panels with infinite aspect ratio in supersonic airstreams, to the solution on the Paragon XP/S, Cray T3D and IBM SP2 parallel systems of three-dimensional wing response problems in the transonic regime. Finally, we conclude this paper in Section 10.

REMARK 1: Some of the content of this paper is based on recent publications by the author and his co-workers. These publications are indicated between [ ] at the beginning of each section and wherever is appropriate.

2. GEOMETRIC CONSERVATION LAWS [19,36]

As stated earlier, the matrices $\widetilde{\mathbf{K}}^c$ and $\widetilde{\mathbf{K}}$ that appear in the third of Eqs. (22) are designed to enforce continuity between the grid motion and the structural displacement and/or velocity at the moving fluid/structure boundary $\Gamma_{F/S}(t)$

$$\begin{aligned} x(t) &= q(t) \quad \text{on } \Gamma_{F/S}(t) \\ \dot{x}(t) &= \dot{q}(t) \quad \text{on } \Gamma_{F/S}(t) \end{aligned} \tag{23}$$

The first of Eqs. (22) involves both the position and velocity of the underlying fluid dynamic mesh. These entities are usually obtained from the solution of the second and third of Eqs. (22), and optionally from the use of a predictor. When selecting a method for integrating the fluid equations, it is desirable to choose one that preserves the trivial solution of a uniform flow field (in the absence of other boundary conditions, a uniform flow field is a solution of the Navier-Stokes equations). In this section, we show that this property is verified only when the numerical scheme chosen for solving the fluid equations and the algorithm constructed for updating the mesh position and velocity satisfy a certain condition. We refer to this condition as the Geometric Conservation Law (GCL) because: (a) it can be identified as integrating exactly the area or volume swept by the boundary of a cell in a finite volume formulation, and (b) its principle is similar to the GCL condition that was first pointed out in [37] for structured grids and finite difference schemes. In the present work, we derive the conditions imposed by the GCL in terms of an appropriate choice of integration points in time, and a consistent scheme for updating the grid point velocities. This is in contrast with previous works [38,39] where the GCL was addressed in terms of *averaged* normal or velocity coefficients for moving finite volume cells. The

approach exposed herein for deriving and satisfying a GCL is deemed more general than those previously discussed in the literature. For example, it recovers the results of the normal averaging algorithm recently proposed in [38] for finite volume discretizations, and applies as well to finite element methods that are not covered by this normal averaging procedure.

Throughout this section, we consider flow computations using unstructured moving meshes. We focus on the Euler equations, because in our formulation the viscous terms are not explicitly affected by the mesh motion. We derive several GCL conditions for these problems, and discuss their various algorithmic implications. We consider first the case where the finite volume method is chosen for the spatial approximation of the flow equations, and the ALE formulation is used for handling dynamic meshes. Then, we analyze the cases where the finite element method is employed for spatial discretization, and the moving mesh is treated with either a space-time or an ALE formulation, respectively. In particular, we show that space-time finite element methods always satisfy the fundamental geometric conservation law. We investigate the consequences of the GCL condition on the temporal integration of the structural equations of motion. Most importantly, we address the problem of satisfying both displacement and velocity continuity equations between the structure and fluid mesh at the fluid/structure interface, without violating the GCL. Finally, we highlight the importance of the GCL with an illustration of its effect on the computation of the transient aeroelastic response of a flat panel in transonic flow.

2.1. The Finite Volume Method with an ALE Formulation

Let $\Omega(t) \subset \mathcal{R}^n$ $(n = 2,\ 3)$ be the flow domain of interest and $\Gamma(t)$ be its moving and deforming boundary. We introduce a mapping function between $\Omega(t)$ where time is denoted by t and the grid point coordinates by x, and a reference configuration $\Omega(0)$ where time is denoted by θ and the grid point coordinates by ξ as follows

$$x = x(\xi, \theta); \quad t = \theta \tag{24}$$

The conservative form of the equations describing Euler flows can be written in arbitrary Lagrangian-Eulerian (ALE) form as

$$\begin{aligned} \frac{\partial(JW)}{\partial t}|_\xi + J\nabla_x.\mathcal{F}^c(W,\dot{x}) &= 0 \\ \mathcal{F}^c(W,\dot{x}) &= \mathcal{F}(W) - \dot{x}W \end{aligned} \tag{25}$$

where $J = det(dx/d\xi)$ is the jacobian of the frame transformation $\xi \rightarrow x$, W denotes the fluid conservative variables, $\mathcal{F}^c$ denotes the convective ALE fluxes, and $\dot{x} = \frac{\partial x}{\partial \theta}|_\xi$ is the ALE grid velocity that may be different from the fluid velocity and from zero.

The finite volume method for unstructured meshes relies on the discretization of the computational domain into control volumes or cells C_i constructed around the vertices S_i, with boundaries denoted by ∂C_i, and normals to these boundaries denoted by ν_i.

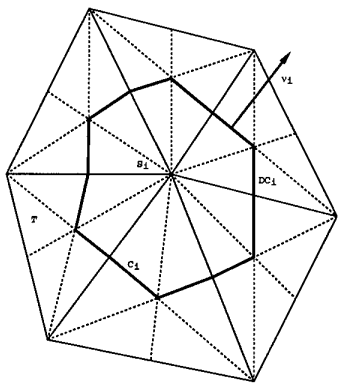


Fig. 4. Control volume
(unstructured two-dimensional mesh)

Eq. (25) can then be integrated over the control cells. In an ALE formulation, these cells move and deform in time. First, integration is

performed over a reference cell in the ξ space as follows

$$\int_{C_i(0)} \frac{\partial(JW)}{\partial t}|_\xi \, d\Omega_\xi + \int_{C_i(0)} J\nabla_x.\mathcal{F}^c(W,\dot{x}) \, d\Omega_\xi = 0 \tag{26}$$

In the above equation, the partial time derivative is evaluated at constant ξ; hence, it can be moved outside of the integral sign to obtain

$$\frac{d}{dt}\int_{C_i(0)} W \, J \, d\Omega_\xi + \int_{C_i(0)} \nabla_x.\mathcal{F}^c(W,\dot{x}) \, J \, d\Omega_\xi = 0 \tag{27}$$

Switching back to the time-varying cells, Eq. (27) above can be rewritten as

$$\frac{d}{dt}\int_{C_i(t)} W \, d\Omega_x + \int_{C_i(t)} \nabla_x.\mathcal{F}^c(W,\dot{x}) \, d\Omega_x = 0 \tag{28}$$

Finally, integrating by parts the last term yields the governing integral equation

$$\frac{d}{dt}\int_{C_i(t)} W \, d\Omega_x + \int_{\partial C_i(t)} \mathcal{F}^c(W,\dot{x}).\nu_i \, d\sigma = 0 \tag{29}$$

In a finite volume method, the flux through the cell boundary $\partial C_i(t)$ is usually evaluated via a flux splitting approximation [40] as follows

$$F_i^c(\mathbf{W},\mathbf{x},\dot{\mathbf{x}}) = \sum_j \int_{\partial C_{i,j}(x)} (\mathcal{F}_+^c(W_i,\dot{x}) + \mathcal{F}_-^c(W_j,\dot{x})).\nu_i \, d\sigma \tag{30}$$

where $\partial C_{i,j}$ is the intersection between the boundaries of cells C_i and C_j, W_i denotes the average value of W over the cell C_i, $\mathbf{W}$ is the vector formed by the collection of W_i, and $\mathbf{x}$ is the vector of the time-dependent grid point positions. The numerical flux functions $\mathcal{F}_+^c$ and $\mathcal{F}_-^c$ are designed to make the resulting system stable. An example of such functions can be found in [41]. For consistency, these numerical fluxes must verify

$$\mathcal{F}_+^c(W,\dot{x}) + \mathcal{F}_-^c(W,\dot{x}) = \mathcal{F}^c(W,\dot{x}) \tag{31}$$

Thus, the resulting discrete equation is

$$\boxed{\frac{d}{dt}(V_i W_i) + F_i^c(\mathbf{W},\mathbf{x},\dot{\mathbf{x}}) = 0} \tag{32}$$

where

$$V_i = \int_{C_i(t)} d\Omega_x \tag{33}$$

is the area for two-dimensional flow problems, and the volume for three-dimensional flow problems, of cell C_i. Collecting all Eqs. (32) into a single system yields

$$\boxed{\frac{d}{dt}(\mathbf{VW}) + \mathbf{F}^c(\mathbf{W},\mathbf{x},\dot{\mathbf{x}}) = 0} \tag{34}$$

where $\mathbf{V}$ is the diagonal matrix of the cell areas, $\mathbf{W}$ is the vector containing all state variables W_i, and $\mathbf{F}^c$ is the collection of the fluxes F_i^c. This also completes the derivation of the first of Eqs. (22).

2.1.1. The Geometric Conservation Law

Let Δt and $t^n = n\Delta t$ denote respectively the chosen time-step and the n-th time-station. Integrating Eq. (32) between t^n and t^{n+1} leads to

$$V_i(\mathbf{x}^{n+1})W_i^{n+1} - V_i(\mathbf{x}^n)W_i^n + \int_{t^n}^{t^{n+1}} F_i^c(\mathbf{W},\mathbf{x},\dot{\mathbf{x}})dt = 0 \tag{35}$$

The most important issue in the solution of the first of Eqs. (22) via an ALE method is the proper evaluation of $\int_{t^n}^{t^{n+1}} F_i^c(\mathbf{W},\mathbf{x},\dot{\mathbf{x}})dt$ in Eq. (35). In particular, it is crucial to establish where the fluxes must be integrated: on the mesh configuration at $t = t^n$ ($\mathbf{x}^n$), on that at

$t = t^{n+1}$ ($\mathbf{x}^{n+1}$), or in between these two configurations. The same questions arise as to the choice of the mesh velocity vector $\dot{\mathbf{x}}$.

Clearly, a proposed numerical algorithm for computing the quantity $\int_{t^n}^{t^{n+1}} F_i^c(\mathbf{w}, \mathbf{x}, \dot{\mathbf{x}})dt$ involving general and arbitrary time dependent fluid state vectors and mesh configurations cannot be acceptable unless it conserves the state of a uniform flow. Let W^* denote a given uniform state of the flow. Substituting $W_k^n = W_k^{n+1} = W^*$ in Eq. (35) gives

$$(V_i^{n+1} - V_i^n)W^* + \int_{t^n}^{t^{n+1}} F_i^c(\mathbf{W}^*, \mathbf{x}, \dot{\mathbf{x}})\ dt = 0 \tag{36}$$

where $\mathbf{W}^*$ is the vector of the state variables when $W_k = W^*$ for all k. From Eq. (30), it follows that

$$\begin{aligned} F_i^c(\mathbf{W}^*, \mathbf{x}, \dot{\mathbf{x}}) \ &= \\ &\sum_j \int_{\partial C_{i,j}(x)} (\mathcal{F}_+^c(W^*, \dot{x}) + \mathcal{F}_-^c(W^*, \dot{x})).\nu_i\ d\sigma \\ &= \int_{\partial C_i(\mathbf{X})} (\mathcal{F}(W^*) - \dot{x}W^*).\nu_i\ d\sigma \end{aligned} \tag{37}$$

Given that the integral on a closed boundary of the flux of a constant function is identically zero

$$\int_{\partial C_i(\mathbf{X})} \mathcal{F}(W^*).\nu_i\ d\sigma = 0 \tag{38}$$

it follows that

$$F_i^c(\mathbf{W}^*, \mathbf{x}, \dot{\mathbf{x}}) = -\int_{\partial C_i(\mathbf{X})} \dot{x}W^*.\nu_i\ d\sigma \tag{39}$$

Hence, substituting Eq. (39) into Eq. (36) yields

$$\begin{aligned} &(V_i(\mathbf{x}^{n+1}) - V_i(\mathbf{x}^n))W^* \\ &- (\int_{t^n}^{t^{n+1}} \int_{\partial C_i(\mathbf{X})} \dot{x}.\nu_i\ d\sigma\ dt)W^* = 0 \end{aligned} \tag{40}$$

which can be rewritten as

$$\boxed{(V_i(\mathbf{x}^{n+1}) - V_i(\mathbf{x}^n)) = \int_{t^n}^{t^{n+1}} \int_{\partial C_i(\mathbf{X})} \dot{x}.\nu_i\ d\sigma\ dt} \tag{41}$$

Eq. (41) above defines a geometric conservation law (GCL) that must be verified by any proposed ALE mesh updating scheme. This law states that the change in area (volume) of each control volume between t^n and t^{n+1} must be equal to the area (volume) swept by the cell boundary during $\Delta t = t^{n+1} - t^n$. Therefore, the updating of $\mathbf{x}$ and $\dot{\mathbf{x}}$ cannot be based on mesh distorsion issues alone when using ALE solution schemes.

The assumption that the numerical method performs exactly the integration of Eq. (38) is referred to in [39] as the Surface Conservation Law (SCL). Satisfying of this condition is necessary for flow computations on static meshes and is not specific to dynamic ones. Therefore, we do not discuss this condition in this section any further and refer the reader to [39] for additional details.

2.1.2. Implications of the GCL

From the analysis presented in the previous section, it follows that an appropriate scheme for evaluating $\int_{t^n}^{t^{n+1}} F_i^c(\mathbf{W}^*, \mathbf{x}, \dot{\mathbf{x}})dt$ in Eq. (36) is a scheme that respects the GCL (41). Note that once a mesh updating scheme is given, the left hand side of Eq. (41) is always exactly computed. Hence, a proper method for evaluating $\int_{t^n}^{t^{n+1}} F_i^c(\mathbf{W}^*, \mathbf{x}, \dot{\mathbf{x}})dt$ is a method that obeys the GCL and therefore computes exactly the right hand side of Eq. (41)— that is, $\int_{t^n}^{t^{n+1}} \int_{\partial C_i(\mathbf{X})} \dot{x}.\nu_i\ d\sigma\ dt$.

2.1.3. The Two-Dimensional Case

Given that in two dimensions ∂C_i is the union of segments, it suffices to consider the integration of $\dot{x}.n$ along a segment $[ab]$ with a normal n

$$I_{[ab]} = \int_{t^n}^{t^{n+1}} \int_{[ab]} \dot{x}.n \; ds dt \qquad (42)$$

Let x_a and x_b denote the instantaneous positions of two connected vertices a and b (Fig. 5). The position of any point on the edge $[ab]$ during the time-interval $[t^n, t^{n+1}]$ can be parametrized as follows

$$\begin{aligned} x(t) &= \alpha x_a(t) + (1-\alpha) x_b(t) \\ \dot{x}(t) &= \alpha \dot{x}_a(t) + (1-\alpha)\dot{x}_b(t) \\ \alpha &\in [0,\ 1] \quad t \in [t^n,\ t^{n+1}] \end{aligned} \qquad (43)$$

where

$$\begin{aligned} x_a(t) &= \delta(t) x_a^{n+1} + (1-\delta(t)) x_a^n \\ x_b(t) &= \delta(t) x_b^{n+1} + (1-\delta(t)) x_b^n \end{aligned} \qquad (44)$$

and $\delta(t)$ is a real function that satisfies

$$\delta(t^n) = 0; \qquad \delta(t^{n+1}) = 1 \qquad (45)$$

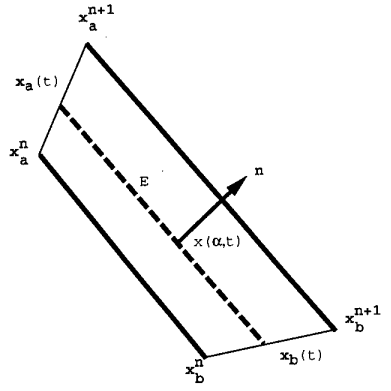


Fig. 5 Parametrization of an edge in a two-dimensional space

Substituting Eqs. (43,44) into Eq. (42) yields

$$\begin{aligned} I_{[ab]} &= \int_{t^n}^{t^{n+1}} \int_0^1 (\alpha \dot{x}_a + (1-\alpha)\dot{x}_b).n \; l \; d\alpha dt \\ &= \int_{t^n}^{t^{n+1}} \frac{1}{2}(\dot{x}_a + \dot{x}_b).n \; l \; dt \\ &= \int_{t^n}^{t^{n+1}} \frac{1}{2}(\dot{x}_a + \dot{x}_b) H(x_a - x_b) \; dt \\ &= \int_{t^n}^{t^{n+1}} \frac{1}{2}(\dot{x}_a + \dot{x}_b) H(\delta(t)(x_a^{n+1} - x_b^{n+1}) \\ &\qquad + (1-\delta(t))(x_a^n - x_b^n)) \; dt \end{aligned} \qquad (46)$$

where l is the length of edge $[ab]$, and $H = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$. The mesh velocities $\dot{x}_a$ and $\dot{x}_b$ can be obtained from the differentiation of Eq. (44).

$$\begin{aligned} \dot{x}_a &= \dot{\delta}(t)(x_a^{n+1} - x_a^n) \\ \dot{x}_b &= \dot{\delta}(t)(x_b^{n+1} - x_b^n) \end{aligned} \qquad (47)$$

and $I_{[ab]}$ can be finally written as

$$\begin{aligned} I_{[ab]} &= \frac{1}{2} \int_{t^n}^{t^{n+1}} \dot{\delta}((x_a^{n+1} - x_a^n) + (x_b^{n+1} - x_b^n)) \\ &\qquad H(\delta(x_a^{n+1} - x_b^{n+1}) + (1-\delta)(x_a^n - x_b^n)) \; dt \\ &= \frac{1}{2} \int_0^1 ((x_a^{n+1} - x_a^n) + (x_b^{n+1} - x_b^n)) \\ &\qquad H(\delta(x_a^{n+1} - x_b^{n+1}) + (1-\delta)(x_a^n - x_b^n)) \; d\delta \end{aligned} \qquad (48)$$

Clearly, the integrand of $I_{[ab]}$ is linear in δ. Therefore, $I_{[ab]}$ can be exactly computed using the *midpoint rule*, provided that Eq. (47) holds — that is

$$\dot{x} = \dot{\delta}(t)(x^{n+1} - x^n) = \frac{\Delta\delta}{\Delta t}(x^{n+1} - x^n) \qquad (49)$$

which in view of Eq. (45) can also be written as

$$\boxed{\dot{x} = \frac{x^{n+1} - x^n}{\Delta t}} \qquad (50)$$

In summary, the GCL derived herein shows that for two-dimensional problems, the integrand of $\int_{t^n}^{t^{n+1}} F_i^c(\mathbf{W}, \mathbf{x}, \dot{\mathbf{x}}) \; dt$ in Eq. (35)

must be evaluated at the midpoint configuration, and that this integral must be computed as follows

$$\int_{t^n}^{t^{n+1}} F_i^c(\mathbf{W},\mathbf{x},\dot{\mathbf{x}})\, dt = \Delta t F_i^c(\mathbf{W}^k,\mathbf{x}^{n+\frac{1}{2}},\dot{\mathbf{x}}^{n+\frac{1}{2}})$$
$$\mathbf{W}^{n+\frac{1}{2}} = \frac{\mathbf{W}^n + \mathbf{W}^{n+1}}{2}$$
$$\mathbf{x}^{n+\frac{1}{2}} = \frac{\mathbf{x}^n + \mathbf{x}^{n+1}}{2}$$
$$\dot{\mathbf{x}}^{n+\frac{1}{2}} = \frac{\mathbf{x}^{n+1} - \mathbf{x}^n}{\Delta t} \tag{51}$$

where the superscript k depends on the time discretization of the flow equation.

2.1.4. The Three-Dimensional Case

In a three-dimensional space, the boundary of each cell is polygonal and can be decomposed into a set of non overlapping triangular facets. Similarly to the two-dimensional case, let $I_{[abc]}$ denote the flux crossing the facet $[abc]$

$$I_{[abc]} = \int_{t^n}^{t^{n+1}} \int_{[abc]} \dot{x}.n\, d\sigma dt \tag{52}$$

Let x_a, x_b and x_c denote the instantaneous positions of three connected vertices a, b and c. The position of any point on the facet can be parametrized as follows (see Fig. 6)

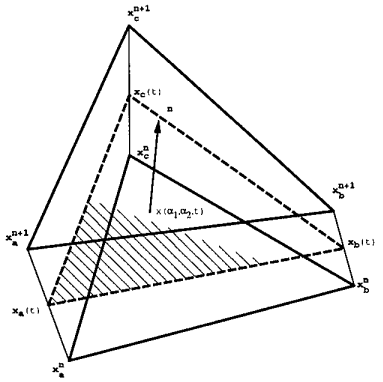


Fig. 6. Parametrization of a facet in a three-dimensional space

$$x = \alpha_1 x_a(t) + \alpha_2 x_b(t) + (1-\alpha_1-\alpha_2)x_c(t)$$
$$\dot{x} = \alpha_1 \dot{x}_a(t) + \alpha_2 \dot{x}_b(t) + (1-\alpha_1-\alpha_2)\dot{x}_c(t)$$
$$\alpha_1 \in [0,\ 1];\quad \alpha_2 \in [0,\ 1-\alpha_1];\quad t \in [t^n,\ t^{n+1}] \tag{53}$$

where

$$x_a(t) = \delta(t)x_a^{n+1} + (1-\delta(t))x_a^n$$
$$x_b(t) = \delta(t)x_b^{n+1} + (1-\delta(t))x_b^n \tag{54}$$
$$x_b(t) = \delta(t)x_b^{n+1} + (1-\delta(t))x_b^n$$

and $\delta(t)$ is given in (45). Substituting the above parametrization in (52) we obtain

$$I_{[abc]} = \int_{t^n}^{t^{n+1}} \int_0^1 \int_0^{1-\alpha_1} (\alpha_1 \dot{x}_a + \alpha_2 \dot{x}_b + (1-\alpha_1-\alpha_2)\dot{x}_c).n\ |x_{ac} \wedge x_{bc}|\ d\alpha_2\ d\alpha_1\ dt$$
$$= \int_{t^n}^{t^{n+1}} \frac{1}{6}(\dot{x}_a + \dot{x}_b + \dot{x}_c).(x_{ac} \wedge x_{bc})\ dt$$
$$= \int_{t^n}^{t^{n+1}} \frac{1}{6}\dot{\delta}(\Delta x_a + \Delta x_b + \Delta x_c).(x_{ac} \wedge x_{bc})\ dt$$
$$= \int_0^1 \frac{1}{6}(\Delta x_a + \Delta x_b + \Delta x_c).(x_{ac} \wedge x_{bc})\ d\delta \tag{55}$$

with

$$x_{ac} = x_a - x_c;\quad x_{bc} = x_b - x_c$$
$$\Delta x_a = x_a^{n+1} - x_a^n;\quad \Delta x_b = x_b^{n+1} - x_b^n \tag{56}$$
$$\Delta x_c = x_c^{n+1} - x_c^n$$

Noting that

$$x_{ac} \wedge x_{bc} = (\delta x_{ac}^{n+1} + (1-\delta)x_{ac}^n) \wedge (\delta x_{bc}^{n+1} + (1-\delta)x_{bc}^n) \tag{57}$$

is a quadratic function of δ, the integrand of $I_{[abc]}$ is clearly quadratic in δ and therefore can be exactly computed using a *2-point integration rule*, provided that Eq. (50) is used to compute $\dot{x}$.

Hence, the proper method for evaluating $\int_{t^n}^{t^{n+1}} F_i^c(\mathbf{W},\mathbf{x},\dot{\mathbf{x}})\ dt$ that respects the GCL (41) in the three-dimensional case is

$$\begin{aligned}
&\int_{t^n}^{t^{n+1}} F_i^c(\mathbf{W},\mathbf{x},\dot{\mathbf{x}})\ dt \\
&\quad = \frac{\Delta t}{2}(F_i^c(\mathbf{W}^{k1},\mathbf{x}^{m1},\dot{\mathbf{x}}^{n+\frac{1}{2}}) \\
&\qquad + F_i^c(\mathbf{W}^{k2},\mathbf{x}^{m2},\dot{\mathbf{x}}^{n+\frac{1}{2}})) \\
&m1 = n + \frac{1}{2} - \frac{1}{2\sqrt{3}} \\
&m2 = n + \frac{1}{2} + \frac{1}{2\sqrt{3}} \\
&\mathbf{W}^{n+\eta} = \eta\mathbf{W}^{n+1} + (1-\eta)\mathbf{W}^n \\
&\mathbf{x}^{n+\eta} = \eta\mathbf{x}^{n+1} + (1-\eta)\mathbf{x}^n \\
&\dot{\mathbf{x}}^{n+\frac{1}{2}} = \frac{\mathbf{x}^{n+1} - \mathbf{x}^n}{\Delta t}
\end{aligned} \tag{58}$$

where the superscripts $k1$ and $k2$ depend on the time discretization of the flow equation.

2.1.5. Recovery of the Averaged-Normals Method

In [38], the convected flux accross the facet $I_{[abc]}$ is computed using

$$\begin{aligned}
I_{[abc]} &= \frac{1}{3}(\Delta x_a + \Delta x_b + \Delta x_c).\eta \\
\eta &= \frac{1}{6}(x_{ac}^n \wedge x_{bc}^n + x_{ac}^{n+1} \wedge x_{bc}^{n+1} \\
&\quad + \frac{1}{2}(x_{ac}^n \wedge x_{bc}^{n+1} + x_{ac}^{n+1} \wedge x_{bc}^n))
\end{aligned} \tag{59}$$

while the evaluation of Eq. (52) using the two-point rule gives

$$\begin{aligned}
I_{[abc]} &= \frac{1}{3}(\Delta x_a + \Delta x_b + \Delta x_c).\eta^* \\
\eta^* &= \frac{1}{4}(x_{ac}^{m1} \wedge x_{bc}^{m1} + x_{ac}^{m2} \wedge x_{ac}^{m2})
\end{aligned} \tag{60}$$

Expanding η^*, we obtain

$$\begin{aligned}
\eta^* &= \frac{1}{4}(((\frac{1}{2} + \frac{1}{2\sqrt{3}})x_{ac}^n + (\frac{1}{2} - \frac{1}{2\sqrt{3}})x_{ac}^{n+1}) \\
&\quad \wedge ((\frac{1}{2} + \frac{1}{2\sqrt{3}})x_{bc}^n + (\frac{1}{2} - \frac{1}{2\sqrt{3}})x_{bc}^{n+1}) \\
&\quad + ((\frac{1}{2} - \frac{1}{2\sqrt{3}})x_{ac}^n + (\frac{1}{2} + \frac{1}{2\sqrt{3}})x_{ac}^{n+1}) \\
&\quad \wedge ((\frac{1}{2} - \frac{1}{2\sqrt{3}})x_{bc}^n + (\frac{1}{2} + \frac{1}{2\sqrt{3}})x_{bc}^{n+1})) \\
&= \frac{1}{2}(\frac{1}{3}x_{ac}^n \wedge x_{bc}^n + \frac{1}{3}x_{ac}^{n+1} \wedge x_{bc}^{n+1} \\
&\quad + \frac{1}{6}(x_{ac}^n \wedge x_{bc}^{n+1} + x_{ac}^{n+1} \wedge x_{bc}^n))
\end{aligned} \tag{61}$$

which shows that the proposed GCL (20) recovers the same results as the averaged-normals method proposed in [38] for the finite volume discretization of flow equations with moving meshes.

2.2. The Stabilized Finite Element Method with a Space-Time Formulation

2.2.1. Semi-Discretization

Time-integration in space-time finite element methods is derived in a different manner than what has been presented so far. Space-time finite element methods contain the time integration formula in the chosen shape functions. These methods are basically weighted residual formulations that perform an integration in space and time of the product of the Euler equations and an appropriate weighting function. Stabilization is usually required for the spatial approximation [42]. In this section, we focus on the stabilized Least-Square/Galerkin method and time-discontinuous shape functions.

Let $0 = t^0 < t^1 < \ \ldots\ < t^N = T$ be a partition of the time-interval $I =]0,T[$, and I_n be the subinterval $]t^n, t^{n+1}[$. A *space-time slab* is defined in $I_n \times R^d$, where d designates the spatial dimension, as follows

$$Q_n = \{(t,\Omega(t)) \mid t \in I_n\} \tag{62}$$

with boundary

$$P_n = \{(t, \Gamma(t)) \mid t \in I_n\} \qquad (63)$$

For each space-time slab, the spatial domain is subdivided into n_{el} elements $\Omega_n^e(t)$, $e = 1, ..., n_{el}$ (see Fig. 7). The following notational convention is adopted

$$W(t_{\pm}^n) = \lim_{\epsilon \to 0^{\pm}} W(t^n + \epsilon) \qquad (64)$$

Given some finite element spaces $\mathcal{S}_n^h$ and $\mathcal{V}_n^h$, the space-time (discontinuous) Least-Square/Galerkin method for solving the Euler flow equations goes as follows

Find $W^h \in \mathcal{S}_n^h$ such that for all $V^h \in \mathcal{V}_n^h$

$$\int_{Q_n} \left(V^h W_{,t}^h - V_{,i}^h F_i^c(W^h) \right) \, dQ + \int_{\Omega(t_+^n)} V^h(t_+^n)(W^h(t_+^n) - W^h(t_-^n)) \, d\Omega + \sum_{e=1}^{n_{el}} \int_{Q_e^n} (\mathcal{L}W^h)\mu(\mathcal{L}V^h) \, dQ = \int_P^n V^h F_i^c n_i \, dP \qquad (65)$$

where $\mathcal{L} = \frac{\partial}{\partial t} + \sum_{k=1}^{k=3} \frac{\partial \mathcal{F}_k}{\partial W} \frac{\partial}{\partial x_k}$, and μ is a stabilization parameter.

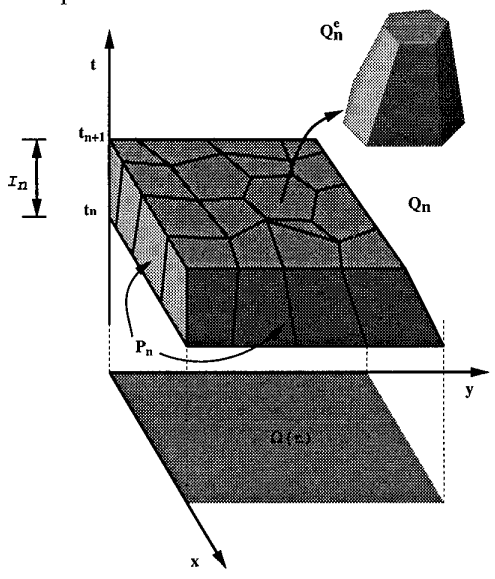


Fig. 7. Space-time slabs

2.2.2. The Geometric Conservation Law

Provided that the spatial integration scheme can compute exactly the following quantities

$$\int_\Omega V^h \, d\Omega \qquad \int_\Omega V_{,i}^h \, d\Omega = 0 \qquad (66)$$

it follows that $W = W^*$ is always a solution of Eq. (65) Hence, a space-time stabilized finite element method always satisfies the GCL. This is certainly an advantage. However, space-time finite element methods are rather computationally expensive.

2.3. The Stabilized Finite Element Method with an ALE Formulation

2.3.1. Semi-Discretization

The stabilized finite element method with an ALE formulation can be derived by multiplying Eq. (25) by a weighting function $V^h(\xi)$, integrating over $\Omega(0)$, and adding a *consistent* stabilization term $S(V^h, W)$ to obtain

$$\int_{\Omega(0)} V^h \left(\frac{\partial (JW)}{\partial t}|_\xi J \nabla_x . \mathcal{F}^c(W, \dot{x}) \right) d\Omega_\xi + S(V^h, W) = 0 \qquad (67)$$

For example, $S(V^h, W)$ can be selected as

$$S(V^h, W) = \sum_{el=1}^{n_{el}} \int_{\Omega^e(t)} \left(\frac{\partial \mathcal{F}_i}{\partial W} V_{,i} \right) \mu \left(\frac{\partial W}{\partial t} + \nabla_x \mathcal{F}(W) \right) d\Omega_x \qquad (68)$$

Consistency requires that S vanishes when W is solution of the Euler equations. Integrating by

parts Eq. (67) and exploiting $\frac{\partial V^h}{\partial t}|_\xi = 0$ leads to

$$\frac{d}{dt}\int_{\Omega(t)} V^h W d\Omega_x - \int_{\Omega(t)} V^h_{,i}\mathcal{F}^c_i(W,\dot{x})d\Omega_x + S(V^h,W) = 0 \tag{69}$$

Integrating the above equation between t^n and t^{n+1} yields

$$\int_{\Omega(t^{n+1})} V^h W d\Omega_x - \int_{\Omega(t^n)} V^h W d\Omega_x - \int_{t^{n+1}}^{t^n}\int_{\Omega(t)} V^h_{,i}\mathcal{F}^c_i(W,\dot{x})d\Omega_x\, dt + \int_{t^{n+1}}^{t^n} S(V^h,W)\, dt = 0 \tag{70}$$

2.3.2. The Geometric Conservation Law

Substituting a constant field $W = W^*$ in Eq. (70) leads to

$$\int_{\Omega(t^{n+1})} V^h W^*\, d\Omega_x - \int_{\Omega(t^n)} V^h W^*\, d\Omega_x - \int_{t^n}^{t^{n+1}}\int_{\Omega(t)} V^h_{,i}\mathcal{F}^c_i(W^*,\dot{x})\, d\Omega_x\, dt + \int_{t^{n+1}}^{t^n} S(V^h,W^*)\, dt = 0 \tag{71}$$

At this point, it is essential to assume that the consistency of S is preserved in its discrete counterpart (at least for a uniform field), and therefore the last term in the above equation is identically zero. From Eq. (68) it can be observed that the least-square term identifies pointwise with zero, and hence the assumption is satisfied independently of the integration rule. One can also reasonably assume that the first and second terms of the above equation can be computed exactly, and that the evaluation of any term of the form

$$\int_\Omega V^h_{,i}\beta_i\, d\Omega_x = 0 \tag{72}$$

where β_i are constants, can also be carried out exactly. Indeed, the latter condition is desirable not only for ALE computations, but also for flow computations using fixed meshes. Violating this condition will introduce artificial fluxes throughout the mesh. Therefore, this condition is the finite element form of the Surface Conservation Law introduced in [39]. For example, if the weighting functions V^h are linear polynomials over each element, $V^h_{,i}$ is constant over each element and a single point integration rule will yield an exact integration formula, provided that the area/volume of the element is computed exactly.

Consequently, provided the SCL is satisfied, and for weighting functions that are zero on the boundary, it follows that

$$\int_\Omega V^h_{,i}\mathcal{F}_i(W^*)\, d\Omega_x = 0 \tag{73}$$

Hence, Eq. (71) can be rewritten as

$$\int_{\Omega(t^{n+1})} V^h W^*\, d\Omega_x - \int_{\Omega(t^n)} V^h W^*\, d\Omega_x - \int_{t^n}^{t^{n+1}}\int_{\Omega(t)} V^h_{,i}\dot{x}_i W^* d\Omega_x\, dt = 0 \tag{74}$$

and can be simplified to

$$\boxed{\int_{\Omega(t^{n+1})} V^h\, d\Omega_x - \int_{\Omega(t^n)} V^h\, d\Omega_x - \int_{t^n}^{t^{n+1}}\int_{\Omega(t)} V^h_{,i}\dot{x}_i d\Omega_x\, dt = 0} \tag{75}$$

Eq. (75) establishes the geometric conservation law for the stabilized finite element method with an ALE formulation.

2.3.3. Implications of the GCL

In order to find the appropriate formula for integrating exactly the last term of the above GCL,

we proceed as follows. First, we introduce the function

$$G(T) = \int_{t^n}^{T} \int_{\Omega(t)} V_{,i}^h \dot{x}_i \; d\Omega_x dt \qquad (76)$$

and note that this function can also be written as

$$\begin{aligned} G(T) \; &= \int_{\Omega(T)} V^h(\xi(x,T)) \; d\Omega_x \\ &- \int_{\Omega(t^n)} V^h(\xi(x,t^n)) \; d\Omega_x \end{aligned} \qquad (77)$$

From the differentiation of Eqs. (76,77) it follows that

$$\begin{aligned} \frac{d}{dT} G(T) &= \int_{\Omega(T)} V_{,i}^h \dot{x}_i \; d\Omega_x \\ &= \frac{d}{dT} \int_{\Omega(T)} V^h(T) \; d\Omega_x \end{aligned} \qquad (78)$$

Hence, the appropriate formula for integrating exactly the last term in Eq. (75) and satisfying the GCL is the one which computes exactly $\int_{t^n}^{t^{n+1}} \frac{d}{dT} \int_{\Omega(T)} V^h(T) \; d\Omega_x$.

2.3.4. The Two-Dimensional Case

Let N_k be some arbitrary mapping functions between the current and reference configurations. We have

$$\begin{aligned} x_1 &= \sum_{k=1}^{k_{max}} N_k(\xi) x_{k_1} \\ x_2 &= \sum_{k=1}^{k_{max}} N_k(\xi) x_{k_2} \end{aligned} \qquad (79)$$

where the subscripts 1 and 2 designate the two different coordinates, and the subscripts k refer to the nodal vertices of the element. The jacobian J of the above transformation is given by

$$J = det \begin{pmatrix} \frac{\partial x_1}{\partial \xi_1} & \frac{\partial x_2}{\partial \xi_1} \\ \frac{\partial x_1}{\partial \xi_2} & \frac{\partial x_2}{\partial \xi_2} \end{pmatrix} = det(\frac{\partial N_k}{\partial \xi_i} x_{k_j}) \qquad (80)$$

where summation is assumed over repeated indices, and x_{k_i} are given by

$$\begin{aligned} x_{k_i}(T) &= (\delta(T) x_{k_i}^{n+1} + (1 - \delta(T)) x_{k_i}^n) \\ &\text{for } k = 1..k_{max}; \quad \text{i=1,2} \end{aligned} \qquad (81)$$

Here, $\delta(t)$ satisfies the conditions given in Eq. (45). This form shows that the matrix involved in the computation of J is a linear function of δ, and therefore J is a quadratic function of δ that can be written as

$$J(T,\xi) = J_0(\xi) + \delta J_1(\xi) + \delta^2 J_2(\xi) \qquad (82)$$

The function G can now be rewritten as

$$\begin{aligned} G(T) \; &= \\ &\sum_{e=1}^{n_{el}} \int_{\Omega^e(0)} (J_0(\xi) + \delta J_1(\xi) + \delta^2 J_2(\xi)) V^h(\xi) \; d\xi \end{aligned} \qquad (83)$$

Therefore, the following conclusions can be made

- $G(T)$ is quadratic in $\delta(T)$, and since

$$\begin{aligned} &\int_{t^n}^{t^{n+1}} \frac{d}{dT} G(T) dT \\ &= \int_{t^n}^{t^{n+1}} \dot{\delta}(T) \frac{d}{d\delta} G(\delta(T)) dT \\ &= \int_0^1 \frac{d}{d\delta} G(\delta(T)) d\delta \end{aligned} \qquad (84)$$

$\frac{d}{d\delta} G(\delta(T))$ is linear in δ and hence, the GCL condition will be satisfied if the midpoint rule is used for the integration of the last term in Eq. (75).

2.3.5. The Three-Dimensional Case

Similarly to the two-dimensional case, the mapping between a reference and current element configuration can be written as

$$\begin{aligned} x_1 &= \sum_{k=1}^{k_{max}} N_k(\xi) x_{k_1} \\ x_2 &= \sum_{k=1}^{k_{max}} N_k(\xi) x_{k_2} \\ x_3 &= \sum_{k=1}^{k_{max}} N_k(\xi) x_{k_3} \end{aligned} \tag{85}$$

and its jacobian J is given by

$$J = det(\frac{\partial N_k}{\partial \xi_j} x_{k_j}) \tag{86}$$

Following the same reasoning as in the two-dimensional case, the following conclusions can be made

- $G(T)$ is cubic in $\delta(T)$, $\frac{d}{d\delta}G(\delta(T))$ is quadratic in δ, and therefore the GCL condition will be satisfied if the two-point rule is used for the integration of the last term in Eq. (75).

2.3.6. Integration Formulae

As discussed above, the integrand of the last term of the geometric conservation law (75) can be linear or quadratic. For a linear integrand, the midpoint rule will perform an exact integration. For a quadratic integrand, the two-point rule must be employed. In all cases, Eq. (78) holds only if $\dot{x}$ is computed in a manner that is compatible with the deformation of $\Omega(\delta)$— that is, if it is obtained by derivation of Eq. (81). Recalling that we are interested in computing

$$\int_{t^n}^{t^{n+1}} \int_{\Omega(t)} -V_{,i}^h \dot{x}_i \; d\Omega_x \; dt \tag{87}$$

and making the change of variable suggested in Eq. (81) and that implies $\dot{x}_i = \dot{\delta}(x_i^{n+1} - x_i^n)$, we obtain

$$\begin{aligned} &\int_0^1 \int_{\Omega(t)} -V_{,i}^h \dot{\delta}(x_i^{n+1} - x_i^n) d\Omega_x \; \frac{1}{\dot{\delta}} d\delta \\ &= \int_0^1 \int_{\Omega(t)} -V_{,i}^h (x_i^{n+1} - x_i^n) d\Omega_x \; d\delta \\ &= \Delta t \int_0^1 \int_{\Omega(t)} -V_{,i}^h \frac{(x_i^{n+1} - x_i^n)}{\Delta t} d\Omega_x \; d\delta \end{aligned} \tag{88}$$

This in turn implies that the mesh velocity $\dot{\mathbf{x}}$ must be computed as follows

$$\dot{\mathbf{x}} = \frac{\mathbf{x}^{n+1} - \mathbf{x}^n}{\Delta t} \tag{89}$$

In summary, the following formulae apply

- two-dimensional flow problems:

$$\boxed{\begin{aligned} &\int_{t^n}^{t^{n+1}} \int_{\Omega(t)} - V_{,i}^h \mathcal{F}_i^c(W^h, \dot{x}) \; d\Omega_x \; dt \\ &\qquad = \Delta t \mathbf{H}(\mathbf{W}^k, \mathbf{x}^{n+\frac{1}{2}}, \dot{x}^{n+\frac{1}{2}}) \\ &\mathbf{H}(\mathbf{W}, \mathbf{x}, \dot{\mathbf{x}}) = \\ &\qquad \int_{\Omega(\mathbf{x})} - V_{,i}^h \mathcal{F}_i^c(W^h(\mathbf{W}), \dot{x}(\dot{\mathbf{x}})) \; d\Omega_x \\ &\qquad \mathbf{x}^{n+\frac{1}{2}} = \frac{\mathbf{x}^{n+1} + \mathbf{x}^n}{2} \\ &\qquad \dot{\mathbf{x}}^{n+\frac{1}{2}} = \frac{\mathbf{x}^{n+1} - \mathbf{x}^n}{\Delta t} \end{aligned}} \tag{90}$$

where the superscript k depends on the time discretization of the flow equation.

• three-dimensional flow problems:

$$
\begin{aligned}
&\int_{t^n}^{t^{n+1}} \int_{\Omega(t)} - V_{,i}^h \mathcal{F}_i^c(W^h, \dot{x})\, d\Omega_x\, dt \\
&\qquad = \frac{\Delta t}{2}(\mathbf{H}(\mathbf{W}^{k_1}, \mathbf{x}^{m_1}, \dot{x}^{n+\frac{1}{2}}) \\
&\qquad\quad + \mathbf{H}(\mathbf{W}^{k_2}, \mathbf{x}^{m_2}, \dot{x}^{n+\frac{1}{2}})) \\
&\mathbf{H}(\mathbf{W}, \mathbf{x}, \dot{\mathbf{x}}) = \\
&\qquad \int_{\Omega(\mathbf{X})} V_{,i}^h \mathcal{F}_i^c(W^h(\mathbf{W}), \dot{x}(\dot{\mathbf{x}}))\, d\Omega_x \\
&\qquad m_1 = n + \frac{1}{2} - \frac{1}{2\sqrt{3}} \\
&\qquad m_2 = n + \frac{1}{2} + \frac{1}{2\sqrt{3}} \\
&\qquad t^{n+\zeta} = \zeta t^{n+1} + (1-\zeta) t^n \\
&\qquad \mathbf{W}^{n+\zeta} = \zeta \mathbf{W}_-^{n+1} + (1-\zeta)\mathbf{W}_+^n \\
&\qquad \mathbf{x}^{n+\zeta} = \zeta \mathbf{x}^{n+1} + (1-\zeta)\mathbf{x}^n \\
&\qquad \dot{\mathbf{x}}^{n+\frac{1}{2}} = \frac{\mathbf{x}^{n+1} - \mathbf{x}^n}{\Delta t}
\end{aligned}
\tag{91}
$$

where the superscripts $k1$ and $k2$ depend on the time discretization of the flow equation.

2.4. Impact of the GCL on the Temporal Solution of Aeroelastic Problems

The most remarquable implication of the GCL condition is the constraint it imposes on the mesh velocity computation, independently of the integration formula for the flow equations

$$\dot{x}^{n+\frac{1}{2}} = \frac{x^{n+1} - x^n}{\Delta t} \tag{92}$$

This formula is intuitive and has been "naturally" used by several investigators independently from any geometric conservation law (see, for example, [15]). However, when sophisticated time-integrators are used for the structure and/or the mesh equations, neither the computed mesh velocities $\dot{\mathbf{x}}^{n+\frac{1}{2}}$ nor the computed structural velocites on the fluid/structure interface are guaranteed to obey $\dot{\mathbf{x}}^{n+\frac{1}{2}} = \frac{x^{n+1}-x^n}{\Delta t}$. In that case, satisfying the GCL requires

- using the mesh velocity $\dot{\mathbf{x}}^{n+\frac{1}{2}}$ computed by the time-integrator, only for evaluating $\mathbf{x}^{n+1}$.
- using the mesh velocity $\dot{\mathbf{x}}^{n+\frac{1}{2}} = \frac{\mathbf{x}^{n+1}-\mathbf{x}^n}{\Delta t}$ in the evaluation of the fluid fluxes.

This means that it is not always possible to respect the continuity of both the displacement and velocity fields on the fluid structure boundary as prescribed by Eqs. (23) without violating the GCL. For example, if the displacement continuity condition $x(t) = q(t)$ is enforced at the fluid/structure interface $\Gamma_{F/S}$, — and that is usually the case — respecting the GCL implies computing a mesh velocity field on $\Gamma_{F/S}$ that is equal to

$$\dot{x}^{n+\frac{1}{2}} = \frac{x^{n+1} - x^n}{\Delta t} = \frac{q^{n+1} - q^n}{\Delta t} \quad \text{on} \quad \Gamma_{F/S} \tag{93}$$

In that case, satisfying also the velocity continuity condition $\dot{x}(t) = \dot{q}(t)$ on $\Gamma_{F/S}$ requires that

$$\dot{q}^{n+\frac{1}{2}} = \frac{q^{n+1} - q^n}{\Delta t} \quad \text{on} \quad \Gamma_{F/S} \tag{94}$$

which is not enforced by all structural time-integrators. Therefore, it is not always possible to satisfy the continuity between both the displacement and the velocity of the structure, and those of the fluid mesh at the fluid/structure interface, without violating the GCL.

Unfortunately, a discontinuity between the velocity of the structure and that of the fluid mesh at the fluid/structure interface can perturb the energy exchange between the fluid and the structure. However, it can be shown that when the implicit midpoint rule is used for advancing the structure and the displacement condition $x(t) = q(t)$ is enforced on $\Gamma_{F/S}$ using a staggered algorithm, both continuity equations (23) can be enforced without violating the GCL. The proof goes as follows.

Given some initial conditions $\mathbf{q}^0$ and $\dot{\mathbf{q}}^0$, suppose that the mesh motion is initialized such

that the following holds on the fluid/structure interface

$$x^{-\frac{1}{2}} = q^0 - \frac{\Delta t}{2}\dot{q}^0 \quad \text{on } \Gamma_{F/S} \tag{95}$$

Also suppose that at each time-station t^n, the continuity of the velocity field is enforced on the fluid/structure boundary

$$\dot{x}^n = \dot{q}^n \quad \text{on } \Gamma_{F/S} \tag{96}$$

If the midpoint rule is used for time-integrating the structural equations of motion, and the dynamic fluid mesh is updated consistently with the GCL as in Eq. (92), it can be proved by induction that

$$x^{n-\frac{1}{2}} = q^n - \frac{\Delta t}{2}\dot{q}^n \quad \text{on } \Gamma_{F/S} \tag{97}$$

Indeed, the above relation holds at $n = 0$. Assuming it holds at n, it follows that

$$\begin{aligned} x^{n+\frac{1}{2}} &= x^{n-\frac{1}{2}} + \Delta t\dot{x}^n \\ &= q^n - \frac{\Delta t}{2}\dot{q}^n + \Delta t\dot{q}^n \\ &= q^n + \frac{\Delta t}{2}\dot{q}^n \end{aligned} \tag{98}$$

Since the midpoint rule algorithm applied to the structural equations implies

$$q^{n+1} - q^n = \frac{\Delta t}{2}(\dot{q}^n + \dot{q}^{n+1}) \tag{99}$$

it follows that

$$x^{n+\frac{1}{2}} = q^{n+1} - \frac{\Delta t}{2}\dot{q}^{n+1} \tag{100}$$

which completes the proof by induction of Eq. (97).

Now, a staggered algorithm for solving the coupled Eqs. (22) can be described as follows

1) using the mesh displacement $\mathbf{x}^{n-\frac{1}{2}}$, and the mesh velocity $\dot{\mathbf{x}}^n$ that matches the structural velocity $\dot{\mathbf{q}}^n$ on $\Gamma_{F/S}$, update the mesh as follows

$$\mathbf{x}^{n+\frac{1}{2}} = \mathbf{x}^{n-\frac{1}{2}} + \Delta t\dot{\mathbf{x}}^n \tag{101}$$

2) using $\mathbf{x}^{n-\frac{1}{2}}$, $\mathbf{x}^{n+\frac{1}{2}}$ and $\dot{\mathbf{x}}^n$, update the fluid state vector $\mathbf{W}^{n+\frac{1}{2}}$ in a manner that satisfies the GCL

3) using the pressure computed from $\mathbf{W}^{n+\frac{1}{2}}$, compute $\mathbf{q}^{n+1}$ and $\dot{\mathbf{q}}^{n+1}$ using the midpoint rule

Defining $\mathbf{x}^n$ as

$$\mathbf{x}^n = \frac{\mathbf{x}^{n-\frac{1}{2}} + \mathbf{x}^{n+\frac{1}{2}}}{2} \tag{102}$$

and substituting Eq. (101) into Eq. (102) leads to

$$\mathbf{x}^n = \mathbf{x}^{n-\frac{1}{2}} + \frac{\Delta t}{2}\dot{\mathbf{x}}^n \tag{103}$$

which in view of Eqs. (97,96) yields

$$x^n = q^n \quad \text{on } \Gamma_{F/S} \tag{104}$$

and demonstrates that, when the midpoint rule is used for time-integrating the structure and a proper staggered procedure is used for solving the coupled fluid/structure problem, the continuity of both the displacement and velocity fields ca be enforced on $\Gamma_{F/S}$ without violating the GCL.

2.5. Numerical Example

In order to highlight the impact of the GCL on coupled aeroelastic computations, we consider here the simulation of the two-dimensional transient aeroelastic response of a flexible panel in a transonic regime. The panel is represented by its cross section that is assumed to have a unit length and a uniform thickness and Young modulus, and to be clamped at both ends. This rectangular cross section is discretized into plane strain 4-node elements with perfect aspect ratios. The two-dimensional flow domain around the panel is discretized into triangles, and the Euler equations are used for this computation. The free stream Mach number is set

to $M_\infty = 0.8$, and a slip condition is imposed at the fluid/structure boundary. Further details on the specifics of this simulation are deferred to Section 9.

Initially, a steady-state flow is computed around the panel at $M_\infty = 0.8$. Next, this flow is perturbed via an initial displacement of the panel that is proportional to its second fundamental mode, and the subsequent panel motion and flow evolution are computed using one of the staggered explicit/implicit fluid/structure procedures described in the following section. Two computed histories of the lift using the same time-step are reported in Fig. 8 for the case where the GCL is violated by updating the mesh velocity field at the fluid/structure interface via a higher-order scheme than that given in Eq. (92), and in Fig. 9 for the case where the GCL is respected. Clearly, this example demonstrates the impact of the GCL on aeroelastic computations as it shows that violating this law leads to undesirable spurious oscillations in the lift prediction.

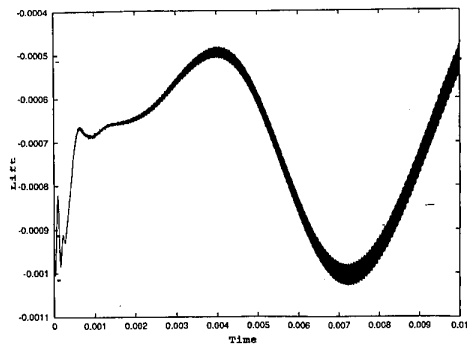


Fig. 8. Lift history when the GCL is violated

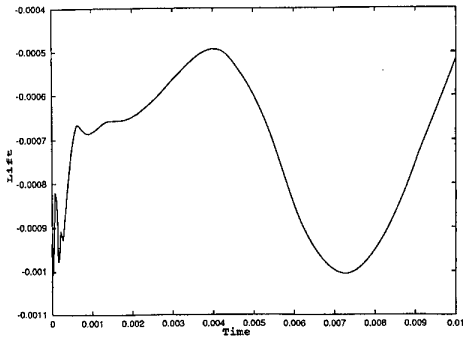


Fig. 9. Lift history when the GCL is obeyed

3. A FAMILY OF STAGGERED SOLUTION PROCEDURES [23,44,45]

In Section 1, we have shown that in the linear theory, the flutter speed of an aircraft can be obtained directly from the solution of an eigenvalue problem. In the nonlinear theory, predicting whether an aircraft will flutter or not for a given set of flight conditions is determined by computing the solution of Eqs. (22), and establishing numerically whether this solution grows continuously in time or not. In other words, a linear aeroelastic dynamic stability problem can be solved without computing explicitly the response of the structure, but a nonlinear aeroelastic dynamic stability problem is typically solved by simulating a set of corresponding nonlinear response problems. Hence, transient nonlinear aeroelastic investigations are in general computationally intensive. For example, establishing the transonic flutter boundary of an aircraft for a given set of aeroelastic parameters requires about 30 aeroelastic response analyses, which clearly demonstrates the need for a fast capability for solving Eqs. (22). Such a capability requires not only powerful supercomputers, but also powerful computational methodologies and algorithms.

One approach for solving the three-way coupled aeroelastic problem described in Eqs. (22) is known as the "monolithic augmentation" approach where, as specific problems arise, a

large-scale single computer program — for example, a finite element structural analysis code — is expanded to house more interaction effects — for example, fluid/structure interaction. Such an approach poses several difficulties, most of which are related to the fact that each of the three components of the three-way coupled aeroelastic problem described in Eqs. (22) has different mathematical and numerical properties, and distinct software implementation requirements. Some of these difficulties have been mentioned in Section 1, others are summarized in [20]. In our opinion, the monolithic augmentation approach is unattractive because once it is implemented, it cannot easily accommodate neither new or improved problem formulations, nor future advances within any of the computational fluid and/or structural dynamics disciplines.

Alternatively, the solution of Eqs. (22) can be obtained through a staggered procedure in which separate fluid and structural analysis programs — often called *field analyzers* [20] — execute and exchange data. Such an approach is also known as partitioned analysis. It offers several appealing features, including the ability to use well established discretization and solution methods within each discipline, simplification of software development efforts, reuse of existing and validated code, accommodation of future single discipline improvements, and preservation of software modularity. Traditionally, nonlinear transient aeroelastic problems have been solved via the simplest possible staggered procedure where the separate fluid and structural analysis programs execute in a strictly sequential fashion, and exchange strictly *interface-state data* such as pressures and velocities at *each single time-step* (see, for example, [15,16,24–27]). The objective of this section is to overview a broader family of more powerful staggered solution procedures that address some important issues related to numerical stability, subcycling, accuracy vs. speed trade-offs, implementation on heterogeneous computing platforms, and inter-field as well as intra-field parallel processing.

3.1. Preliminaries

Of course, the global performance of a partitioned analysis for solving the time-dependent Eqs. (22) depends on the local performances of the fluid and structural field analyzers. But more importantly, the global performance also depends on the stability and accuracy properties of the staggered solution procedure itself. For a given prescribed accuracy, the more stable a staggered algorithm is, the larger is the allowable *coupled time-integration step*, and therefore the faster is the total solution time. Hence, our primal goal is to construct partitioned analysis procedures for Eqs. (22) with superior stability properties.

REMARK 2: The reader is reminded that the stability properties of a staggered solution algorithm depend, among other things, on the stability properties of the field analyzers. However, it is also well-known that using an unconditionally stable time integration algorithm in each field analyzer does not guarantee the unconditional stability of the overall staggered solution algorithm.

Because the aeroelastic response of a structure is often dominated by low frequency dynamics, we consider only implicit schemes for time-integrating the structural displacement field. However, we consider both explicit and implicit time-integrators for advancing the fluid field, as both approaches are popular in computational fluid dynamics. On the other hand, we also note that time-accurate implicit and unstructured flow solvers seem to be less available than their explicit counterparts. In the

sequel, we refer to a partitioned analysis procedure as an explicit/implicit one if an explicit time-accurate flow solver is employed, and as an implicit/implicit one if an implicit flow solver is used. In the implicit/implicit case, our goal is to devise an unconditionally stable staggered algorithm, or at least a partitioned procedure that allows a relatively large time step. In the explicit/implicit case, our objective is to design a staggered solution algorithm whose stability limit is not worse than that of the underlying explicit flow solver. These are not trivial tasks because coupling effects can restrict the stability limits of the independent field time-integrators.

Next, we make the following observations

- linear and nonlinear transient fluid /structure interaction problems have one particularity: they possess a wide variety of self-excited vibrations and instabilities. We have already mentioned the flutter problem. Another example of a dynamic instability is that of the vibrations due to Von Kármán vortices [43]. If the frequency of the structure loading caused by the vortices is close or equal to the natural frequency of the body, then a resonance effect is present and large amplitudes of vibrations result. Therefore, when it comes to analyzing the numerical stability of a proposed staggered algorithm for time-integrating fluid/structure interaction problems, it is essential to consider the case where the coupled system is physically stable — that is, when Eqs. (22) have a solution that does not grow indefinitely in time.
- when the structure undergoes small displacements, the fluid mesh can be frozen and "transpiration" fluxes can be introduced at the fluid side of the fluid/structure boundary to account for the motion of the structure. In that case, the nonlinear transient aeroelastic problem simplifies from a three- to a two-field coupled problem.
- most fluid/structure instability problems can be analyzed by investigating the response of the coupled system to a perturbation around a steady state. If the response is an amplification of the initial perturbation, it is an indication that the system is unstable. If it is a dissipation of the initial perturbation, it means that the system is stable. This suggests that aeroelastic stability or instability problems can be investigated by linearizing the flow around an equilibrium position $\mathbf{W}_0$, and analyzing the response of the fluid/structure system to a perturbation.

Based on the above observations, the authors of reference [23] have constructed a simplified but relevant aeroelastic "test" problem where the coupled fluid/structure system is always physically stable. They have also presented a mathematical framework for analyzing the accuracy and stability properties of staggered procedures applied to the solution of their test problem. Subsequently, this test problem was also shown to be a good model problem for the complex nonlinear aeroelastic systems that we are interested in solving [23,18,19]. In the test problem, the structure is assumed to remain in the linear regime, and the flow is linearized around an equilibrium position of the fluid state vector denoted here by W_0. The semi-discrete equations governing this coupled aeroelastic model problem are given by (see [23] for details)

$$\begin{pmatrix} \dot{\delta\mathbf{W}} \\ \dot{\mathbf{Q}} \end{pmatrix} = \begin{pmatrix} \mathbf{A}^* & \mathbf{B} \\ \mathbf{C} & \mathbf{D}^* \end{pmatrix} \begin{pmatrix} \delta\mathbf{W} \\ \mathbf{Q} \end{pmatrix}$$
$$\begin{pmatrix} \delta\mathbf{W} \\ \mathbf{Q} \end{pmatrix}_{(t=0)} = \begin{pmatrix} \delta\mathbf{W} \\ \mathbf{Q} \end{pmatrix}_0 \tag{105}$$

where $\delta\mathbf{W}$ is the perturbed fluid state vector, $\mathbf{Q} = \begin{pmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{pmatrix}$ is the structure state vector, $\mathbf{A}^*$ results from the spatial discretization of the flow equations, $\mathbf{B}$ is the matrix induced by the transpiration fluxes at the fluid/structure boundary $\Gamma_{F/S}$, $\mathbf{C}$ is the matrix that transforms the fluid pressure on $\Gamma_{F/S}$ into prescribed structural forces, and $\mathbf{D}^* = \begin{pmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{D} \end{pmatrix}$, where as before, $\mathbf{M}$, $\mathbf{D}$, and $\mathbf{K}$ are the structural mass, damping, and stiffness matrices.

In [19], the aeroelastic model problem described in Eqs. (105) has been extended to include the mesh motion of the fluid grid, and therefore to truly represent the three-way coupled aeroelastic problem governed by Eqs. (22). More importantly, reference [19] discusses a methodology for considering a staggered solution procedure that was designed for solving the three-field equivalent of the model problem (105), and extending it to the case of nonlinear transient aeroelastic problems such as those governed by Eqs. (22).

In this section, we overview a family of partitioned analysis procedures for solving the nonlinear transient coupled Eqs. (22). These algorithms are based on the mathematical results established in [23,19], and have recently been described in [44,45]. Rather than discussing mathematical proofs and details that can be found in [23,19], we emphasize important computational and implementational issues pertaining to accuracy, stability, distributed computing, I/O transfers, subcycling, and parallel processing.

3.2. Explicit/implicit partitioned procedures

3.2.1. ALG0: the basic explicit/implicit staggered solution procedure

In order not to obscure the following discussion by the complex notation needed for three-dimensional viscous flows, we focus here, without any loss of generality, on the case of two-dimensional Euler flows discretized by the finite volume method. For three-dimensional inviscid flows, Eqs. (58) should be used instead of Eqs. (51). For finite element and/or space-time discretizations, Eqs. (51) should be replaced by the appropriate equations derived in Section 2.

From the results established in Section 2, it follows that the semi-discrete equations governing the three-way coupled aeroelastic problem can be written in that case as

$$\begin{aligned}
&\mathbf{V}(\mathbf{x}^{n+1})\mathbf{W}^{n+1} - \mathbf{V}(\mathbf{x}^{n})\mathbf{W}^{n} \\
&+\Delta t \mathbf{F}^{c}(\mathbf{W}^{k},\, \mathbf{x}^{n+\frac{1}{2}},\, \dot{\mathbf{x}}^{n+\frac{1}{2}}) \;=\; 0 \\
&\qquad\qquad \mathbf{x}^{n+\frac{1}{2}} \;=\; \frac{\mathbf{x}^{n} + \mathbf{x}^{n+1}}{2} \\
&\qquad\qquad \dot{\mathbf{x}}^{n+\frac{1}{2}} \;=\; \frac{\mathbf{x}^{n+1} - \mathbf{x}^{n}}{\Delta t} \\
&\qquad \mathbf{M}\ddot{\mathbf{q}}^{n+1} + \mathbf{f}^{int}(\mathbf{q}^{n+1}) \;=\; \mathbf{f}^{ext}(\mathbf{x}^{n+1}, \mathbf{W}^{n+1}) \\
&\widetilde{\mathbf{M}}\ddot{\mathbf{x}}^{n+1} + \widetilde{\mathbf{D}}\dot{\mathbf{x}}^{n+1} + \widetilde{\mathbf{K}}\mathbf{x}^{n+1} \;=\; \widetilde{\mathbf{K}}_{c}\, \mathbf{q}^{n+1}
\end{aligned} \tag{106}$$

where the superscript k depends on the time discretization of the fluid flow equations.

In many aeroelastic investigations such as wing flutter problems, first a steady flow is computed around a structure in equilibrium. Next, the structure is perturbed via an initial displacement and/or velocity and the aeroelastic response of the coupled fluid/structure system is analyzed. This suggests that a natural sequencing for the staggered time-integration of Eqs. (106) is

1. perturb the structure via some initial conditions.
2. update the fluid grid to conform to the new structural boundary.
3. advance the flow with the new boundary conditions.

4. advance the structure with the new pressure load.

5. repeat from step 2 until the objective of the simulation is reached.

An important feature of partitioned solution procedures is that they allow using existing single discipline software modules. In our work, we have been particularly interested in reusing the massively parallel explicit flow solver described in [46–49] for two-dimensional problems, and a variant for three-dimensional applications. Therefore, we consider here the case where the semi-discrete fluid equations are integrated with a 3-step variant of the explicit Runge-Kutta algorithm. Of course, other explicit time-integrators can be equally employed. On the other hand, the aeroelastic response of a structure is often dominated by low frequency dynamics. Hence, the structural equations are most efficiently solved by an implicit time-integration scheme. For example, we select to time-integrate the structural motion with the implicit midpoint rule because it allows enforcing both continuity Eqs. (23) while still respecting the GCL (see Section 2). Consequently, we propose the following explicit/implicit solution algorithm for solving the three-field coupled problem (106).

Given a steady flow and initial structural conditions

1. Update the dynamic fluid grid

$$\text{Solve } \widetilde{\mathbf{M}}\ddot{\mathbf{x}}^{n+1} + \widetilde{\mathbf{D}}\dot{\mathbf{x}}^{n+1} + \widetilde{\mathbf{K}}\mathbf{x}^{n+1} = \widetilde{\mathbf{K}}_c\ \mathbf{q}^n$$

$$\mathbf{x}^{n+\frac{1}{2}} = \frac{\mathbf{x}^n + \mathbf{x}^{n+1}}{2}$$

$$\dot{\mathbf{x}}^{n+\frac{1}{2}} = \frac{\mathbf{x}^{n+1} - \mathbf{x}^n}{\Delta t}$$

2. Advance the fluid system using RK3

$$W_i^{n+1^{(0)}} = W_i^n$$

$$W_i^{n+1^{(k)}} = \frac{V_i(\mathbf{x}^n)}{V_i(\mathbf{x}^{n+1})} W_i^{n+1^{(0)}} + \frac{1}{V_i(\mathbf{x}^{n+1})} \frac{1}{4-k} \Delta t F_i^c(W^{(k-1)}, \mathbf{x}^{n+\frac{1}{2}}, \dot{\mathbf{x}}^{n+\frac{1}{2}})$$

$$k = 1,\ 2,\ 3$$

$$W_i^{n+1} = W_i^{n+1^{(3)}}$$

3. Advance the structure using the midpoint rule

$$\mathbf{M}\ddot{\mathbf{q}}^{n+1} + \mathbf{f}^{int}(\mathbf{q}^{n+1}) = \mathbf{f}^{ext}(\mathbf{x}^{n+1}, \mathbf{W}^{n+1})$$

$$\mathbf{q}^{n+1} = \mathbf{q}^n + \frac{\Delta t}{2}(\dot{\mathbf{q}}^n + \dot{\mathbf{q}}^{n+1})$$

$$\dot{\mathbf{q}}^{n+1} = \dot{\mathbf{q}}^n + \frac{\Delta t}{2}(\ddot{\mathbf{q}}^n + \ddot{\mathbf{q}}^{n+1})$$

(107)

In the sequel, we refer to the above explicit/implicit staggered solution procedure as ALG0. It is graphically depicted in Fig. 10. Extensive numerical simulations using this algorithm have shown that its stability limit is governed by the critical time-step of the explicit fluid solver, and therefore is not worse than that of the underlying fluid explicit time-integrator.

The 3-step Runge-Kutta algorithm is third-order accurate for linear problems and second-order accurate for nonlinear ones. The mid-point rule is second-order accurate. A simple Taylor expansion shows that the partitioned analysis procedure ALG0 is first-order accurate when applied to the linearized Eqs. (105). When applied to Eqs. (106), its accuracy depends on the solution scheme selected for solving the mesh equations. As long as the time-integrator applied to the last of Eqs. (106) is consistent, ALG0 is guaranteed to be at least first-order accurate.

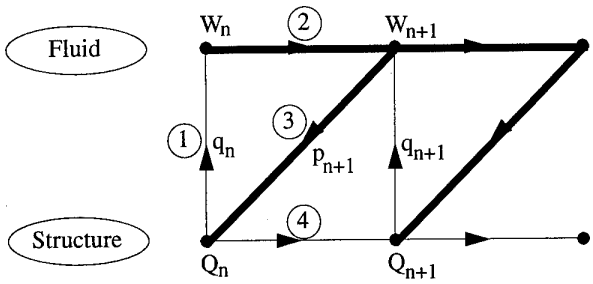


Fig. 10. ALG0: the basic staggered algorithm

3.2.2. ALG1: subcycling

The fluid and structure fields have often different time scales. For problems in aeroelasticity, the fluid flow usually requires a smaller temporal resolution than the structural vibration. Therefore, if ALG0 is used to solve Eqs. (106), the coupling time-step Δt_c will be typically dictated by the stability time-step of the fluid system Δt_F and not the time-step $\Delta t_S > \Delta t_F$ that meets the accuracy requirements of the structural field.

Using the same time-step Δt in both fluid and structure computational kernels presents only minor implementational advantages. On the other hand, subcycling the fluid computations with a factor $n_{S/F} = \Delta t_S / \Delta t_F$ can offer substantial computational advantages, including

- savings in the overall simulation CPU time, because in that case the structural field will be advanced fewer times.
- savings in I/O transfers and/or communication costs when computing on a heterogeneous platform, because in that case the fluid and structure kernels will exchange information fewer times.

However, the computational advantages highlighted above are effective only if subcycling does not restrict the stability region of the staggered algorithm to values of the coupling time-step Δt_c that are small enough to offset these advantages. In [23], it is shown that for the linearized problem (105), the straightforward conventional subcycling procedure — that is, the scheme where at the end of each $n_{S/F}$ fluid subcycles only the interface pressure computed during the last fluid subcycle is transmitted to the structure — lowers the stability limit of ALG0 to a value that is less than the critical time-step of the fluid explicit time-integrator. On the other hand, it is also shown in [23] that when solving Eqs. (105), the stability limit of ALG0 can be preserved if

- the deformation of the fluid mesh between t^n and t^{n+1} is evenly distributed among the $n_{S/F}$ subcycles.
- at the end of each $n_{S/F}$ fluid subcycles, the average of the interface pressure field $\overline{p}_{\Gamma_{F/S}}$ computed during the subcycles between t^n and t^{n+1} is transmitted to the structure rather than the last computed pressure.

Hence, we propose the following explicit/implici fluid-subcycled partitioned procedure for solving Eqs. (106).

1. Build $\{q^{n^{(s)}}\}_{s=1}^{s=n_{S/F}} \;/\; q^{n^{(n_{S/F})}} = q^n$

$\overline{p}_{\Gamma_{F/S}} = 0$

For $s = 1, \ldots, n_{S/F}$ {

Update in stages the dynamic fluid grid

$$Solve\ \widetilde{\mathbf{M}}\ddot{\mathbf{x}}^{n+1^{(s)}} + \widetilde{\mathbf{D}}\dot{\mathbf{x}}^{n+1^{(s)}} + \widetilde{\mathbf{K}}\mathbf{x}^{n+1^{(s)}} = \widetilde{\mathbf{K}}_c\ \mathbf{q}^{n^{(s)}}$$

$$\mathbf{x}^{n+\frac{1}{2}^{(s)}} = \frac{\mathbf{x}^{n^{(s)}} + \mathbf{x}^{n+1^{(s)}}}{2}$$

$$\dot{\mathbf{x}}^{n+\frac{1}{2}^{(s)}} = \frac{\mathbf{x}^{n+1^{(s)}} - \mathbf{x}^{n^{(s)}}}{\Delta t}$$

2. Advance the fluid system using RK3

$$W_i^{n+1^{(0)^{(s)}}} = W_i^{n^{(s)}}$$

$$W_i^{n+1^{(k)^{(s)}}} = \frac{V_i(\mathbf{x}^{n^{(s)}})}{V_i(\mathbf{x}^{n+1^{(s)}})} W_i^{n+1^{(0)^{(s)}}} + \frac{\Delta t F_i^c(W^{(k-1)^{(s)}},\ \mathbf{x}^{n+\frac{1}{2}^{(s)}},\ \dot{\mathbf{x}}^{n+\frac{1}{2}^{(s)}})}{(4-k)V_i(\mathbf{x}^{n+1^{(s)}})}$$

$$k = 1,\ 2,\ 3$$

$$W_i^{n+1^{(s)}} = W_i^{n+1^{(3)^{(s)}}}$$

$$\overline{p}_{\Gamma_{F/S}} = \overline{p}_{\Gamma_{F/S}} + p_{\Gamma_{F/S}}^{n+1^{(s)}}\ \}$$

$$\overline{p}_{\Gamma_{F/S}}^{n+1} = \overline{p}_{\Gamma_{F/S}}/n_{S/F}$$

3. Adv. struc. with midpoint rule

$$\mathbf{M}\ddot{\mathbf{q}}^{n+1} + \mathbf{f}^{int}(\mathbf{q}^{n+1}) = \mathbf{f}^{ext}(\overline{p}_{\Gamma_{F/S}}^{n+1})$$

$$\mathbf{q}^{n+1} = \mathbf{q}^n + n_{S/F}\frac{\Delta t}{2}(\dot{\mathbf{q}}^n + \dot{\mathbf{q}}^{n+1})$$

$$\dot{\mathbf{q}}^{n+1} = \dot{\mathbf{q}}^n + n_{S/F}\frac{\Delta t}{2}(\ddot{\mathbf{q}}^n + \ddot{\mathbf{q}}^{n+1})$$

(108)

In the sequel, we refer to the explicit/implicit fluid-subcycled staggered solution procedure presented here as ALG1. It is graphically depicted in Fig. 11. Extensive numerical experiments have shown that for medium values of $n_{S/F}$, the stability limit of ALG1 is governed by the critical time-step of the explicit flow solver. However, experience has also shown that there exists a maximum subcycling factor beyond which ALG1 becomes numerically unstable.

From the theory developed in [23] for the linearized Eqs. (105), it follows that ALG1 is first-order accurate, and that as one would have expected, subcycling amplifies the fluid errors by the factor $n_{S/F}$.

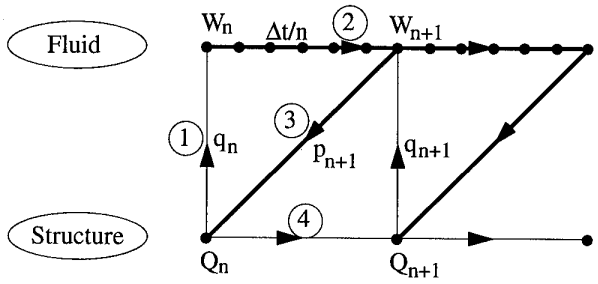


Fig. 11. ALG1: subcycling

3.2.3. ALG2-ALG3: inter-field parallelism

ALG0 and ALG1 are inherently sequential. In both partitioned analysis procedures, the fluid system must be updated before the structural system can be advanced. Of course, ALG0 and ALG1 allow intra-field parallelism (parallel computations within each discipline), but they inhibit inter-field parallelism. Advancing the fluid and structural systems simultaneously is appealing because it can reduce the total simulation time.

A simple variant ALG2 of ALG1 — or ALG0 if subcycling is not desired — that allows inter-field parallel processing is given next.

1. Build $\{q^{n^{(s)}}\}_{s=1}^{s=n_{S/F}} \;/\; q^{n^{(n_{S/F})}} = q^n$

$\overline{p}_{\Gamma_{F/S}} = 0$

For $s = 1, \ldots, n_{S/F}$ {

Update in stages the dynamic fluid grid

$$Solve\ \widetilde{\mathbf{M}}\ddot{\mathbf{x}}^{n+1^{(s)}} + \widetilde{\mathbf{D}}\dot{\mathbf{x}}^{n+1^{(s)}} + \widetilde{\mathbf{K}}\mathbf{x}^{n+1^{(s)}} = \widetilde{\mathbf{K}}_c\ \mathbf{q}^{n^{(s)}}$$

$$\mathbf{x}^{n+\frac{1}{2}^{(s)}} = \frac{\mathbf{x}^{n^{(s)}} + \mathbf{x}^{n+1^{(s)}}}{2}$$

$$\dot{\mathbf{x}}^{n+\frac{1}{2}^{(s)}} = \frac{\mathbf{x}^{n+1^{(s)}} - \mathbf{x}^{n^{(s)}}}{\Delta t}$$

2. Advance the fluid system using RK3

$$W_i^{n+1^{(0)^{(s)}}} = W_i^{n^{(s)}}$$

$$W_i^{n+1^{(k)^{(s)}}} = \frac{V_i(\mathbf{x}^{n^{(s)}})}{V_i(\mathbf{x}^{n+1^{(s)}})} W_i^{n+1^{(0)^{(s)}}} + \frac{\Delta t F_i^c(W^{(k-1)^{(s)}},\ \mathbf{x}^{n+\frac{1}{2}^{(s)}},\ \dot{\mathbf{x}}^{n+\frac{1}{2}^{(s)}})}{(4-k)V_i(\mathbf{x}^{n+1^{(s)}})}$$

$$k = 1,\ 2,\ 3$$

$$W_i^{n+1^{(s)}} = W_i^{n+1^{(3)}(s)}$$

$$\overline{p}_{\Gamma_{F/S}} = \overline{p}_{\Gamma_{F/S}} + p_{\Gamma_{F/S}}^{n+1^{(s)}}\}$$

$$\overline{p}_{\Gamma_{F/S}}^{n+1} = \overline{p}_{\Gamma_{F/S}}/n_{S/F}$$

3. Adv. struc. with midpoint rule

$$\mathbf{M}\ddot{\mathbf{q}}^{n+1} + \mathbf{f}^{int}(\mathbf{q}^{n+1}) = \mathbf{f}^{ext}(\overline{p}_{\Gamma_{F/S}}^{n})$$

$$\mathbf{q}^{n+1} = \mathbf{q}^n + n_{S/F}\frac{\Delta t}{2}(\dot{\mathbf{q}}^n + \dot{\mathbf{q}}^{n+1})$$

$$\dot{\mathbf{q}}^{n+1} = \dot{\mathbf{q}}^n + n_{S/F}\frac{\Delta t}{2}(\ddot{\mathbf{q}}^n + \ddot{\mathbf{q}}^{n+1})$$

(109)

Clearly, the fluid and structure kernels can run in parallel during the time-interval $[t_n,\ t_{n+n_{S/F}}]$. Inter-field communication or I/O transfer is needed only at the beginning of each time-interval.

The basic steps of ALG2 are graphically depicted in Fig. 12. The theory developed in [23] shows that for the linearized Eqs. (105), ALG2 is first-order accurate, and parallelism in ALG2 is achieved at the expense of amplified errors in the fluid and structure responses.

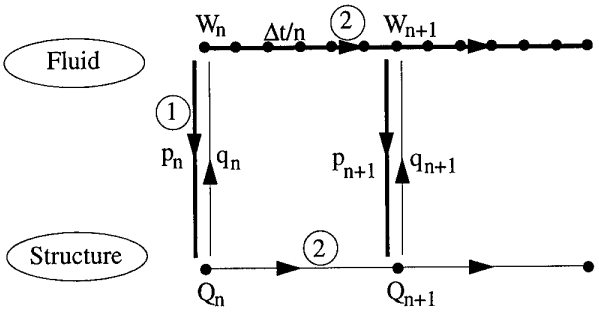


Fig. 12. ALG2: subcycling and inter-field parallelism

In order to improve the accuracy of the basic parallel time-integrator ALG2, we propose to exchange correction type information between the fluid and structure kernels at half-step in the following specific manner (ALG3).

$$
\begin{aligned}
&\overline{p}_{\Gamma_{F/S}} = 0 \\
&For\ s = 1,\ ...,\ \frac{n_{S/F}}{2} - 1 \\
&\quad \{ \\
&\quad Solve\quad \widetilde{\mathbf{M}}\ddot{\mathbf{x}}^{n+1^{(s)}} + \widetilde{\mathbf{D}}\dot{\mathbf{x}}^{n+1^{(s)}} + \widetilde{\mathbf{K}}\mathbf{x}^{n+1^{(s)}} = \widetilde{\mathbf{K}}_c\ \mathbf{q}^{n^{(s)}} \\
&\quad \mathbf{x}^{n+\frac{1}{2}^{(s)}} = \frac{\mathbf{x}^{n^{(s)}} + \mathbf{x}^{n+1^{(s)}}}{2};\quad \dot{\mathbf{x}}^{n+\frac{1}{2}^{(s)}} = \frac{\mathbf{x}^{n+1^{(s)}} - \mathbf{x}^{n^{(s)}}}{\Delta t} \\
&\quad W_i^{n+1^{(0)(s)}} = W_i^{n^{(s)}} \\
&\quad W_i^{n+1^{(k)(s)}} = \frac{V_i(\mathbf{x}^{n^{(s)}})}{V_i(\mathbf{x}^{n+1^{(s)}})} W_i^{n+1^{(0)(s)}} + \frac{\Delta t F_i^c(W^{(k-1)^{(s)}},\ \mathbf{x}^{n+\frac{1}{2}^{(s)}},\ \dot{\mathbf{x}}^{n+\frac{1}{2}^{(s)}})}{(4-k)V_i(\mathbf{x}^{n+1^{(s)}})} \qquad k = 1,\ 2,\ 3 \\
&\quad W_i^{n+1^{(s)}} = W_i^{n+1^{(3)(s)}};\quad \overline{p}_{\Gamma_{F/S}} = \overline{p}_{\Gamma_{F/S}} + p_{\Gamma_{F/S}}^{n+1^{(s)}} \\
&\quad \} \\
&\overline{p}_{\Gamma_{F/S}}^{n+\frac{1}{2}} = \overline{p}_{\Gamma_{F/S}}/(n_{S/F}/2) \\
&\mathbf{M}\ddot{\bar{q}}^{n+1} + \mathbf{f}^{int}(\bar{\mathbf{q}}^{n+1}) = \mathbf{f}^{ext}(\overline{p}_{\Gamma_{F/S}}^{n+\frac{1}{2}}) \\
&\bar{\mathbf{q}}^{n+1} = \mathbf{q}^n + n_{S/F}\frac{\Delta t}{2}(\dot{\mathbf{q}}^n + \dot{\bar{\mathbf{q}}}^{n+1});\quad \dot{\bar{\mathbf{q}}}^{n+1} = \dot{\mathbf{q}}^n + n_{S/F}\frac{\Delta t}{2}(\ddot{\mathbf{q}}^n + \ddot{\bar{\mathbf{q}}}^{n+1}) \\
&\overline{p}_{\Gamma_{F/S}} = 0 \\
&For\ s = \frac{n_{S/F}}{2},\ ...,\ n_{S/F} \\
&\quad \{ \\
&\quad Solve\quad \widetilde{\mathbf{M}}\ddot{\mathbf{x}}^{n+1^{(s)}} + \widetilde{\mathbf{D}}\dot{\mathbf{x}}^{n+1^{(s)}} + \widetilde{\mathbf{K}}\mathbf{x}^{n+1^{(s)}} = \widetilde{\mathbf{K}}_c\ \bar{\mathbf{q}}^{n+1^{(s)}} \\
&\quad \mathbf{x}^{n+\frac{1}{2}^{(s)}} = \frac{\mathbf{x}^{n^{(s)}} + \mathbf{x}^{n+1^{(s)}}}{2};\quad \dot{\mathbf{x}}^{n+\frac{1}{2}^{(s)}} = \frac{\mathbf{x}^{n+1^{(s)}} - \mathbf{x}^{n^{(s)}}}{\Delta t} \\
&\quad W_i^{n+1^{(0)(s)}} = W_i^{n^{(s)}} \\
&\quad W_i^{n+1^{(k)(s)}} = \frac{V_i(\mathbf{x}^{n^{(s)}})}{V_i(\mathbf{x}^{n+1^{(s)}})} W_i^{n+1^{(0)(s)}} + \frac{\Delta t F_i^c(W^{(k-1)^{(s)}},\ \mathbf{x}^{n+\frac{1}{2}^{(s)}},\ \dot{\mathbf{x}}^{n+\frac{1}{2}^{(s)}})}{(4-k)V_i(\mathbf{x}^{n+1^{(s)}})} \qquad k = 1,\ 2,\ 3 \\
&\quad W_i^{n+1^{(s)}} = W_i^{n+1^{(3)(s)}};\quad \overline{p}_{\Gamma_{F/S}} = \overline{p}_{\Gamma_{F/S}} + p_{\Gamma_{F/S}}^{n+1^{(s)}} \\
&\quad \} \\
&\overline{p}_{\Gamma_{F/S}}^{n+1} = \overline{p}_{\Gamma_{F/S}}/(n_{S/F}/2) \\
&\mathbf{M}\ddot{q}^{n+1} + \mathbf{f}^{int}(q^{n+1}) = \mathbf{f}^{ext}(\overline{p}_{\Gamma_{F/S}}^{n+1}) \\
&\mathbf{q}^{n+1} = \mathbf{q}^n + n_{S/F}\frac{\Delta t}{2}(\dot{\mathbf{q}}^n + \dot{\mathbf{q}}^{n+1});\quad \dot{\mathbf{q}}^{n+1} = \dot{\mathbf{q}}^n + n_{S/F}\frac{\Delta t}{2}(\ddot{\mathbf{q}}^n + \ddot{\mathbf{q}}^{n+1})
\end{aligned}
\tag{110}
$$

Algorithm ALG3 is illustrated in Fig. 13. The first-half of the computations is identical to that of ALG2, except that the fluid system is subcycled only up to $t^{n+\frac{n_{S/F}}{2}}$, while the structure is advanced in one shot up to $t^{n+n_{S/F}}$. At $t^{n+\frac{n_{S/F}}{2}}$, the fluid and structure kernels exchange pressure, displacement and velocity information. In the second-half of the computations, the fluid system is subcycled from $t^{n+\frac{n_{S/F}}{2}}$ to $t^{n+n_{S/F}}$ using the new structural information, and the structural behavior is recomputed in parallel using the newly received pressure distribution. Note that the first evaluation of the structural state vector can be interpreted as a prediction step, and the second as a correction step.

It can be shown that when applied to the linearized Eqs. (105), ALG3 is first-order accurate and reduces the errors of ALG2 by the factor $n_{S/F}$, at the expense of one additional communication step or I/O transfer during each coupled cycle (see [23] for a detailed error analysis).

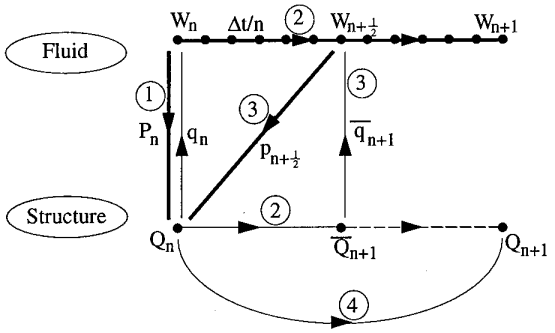


Fig. 13. ALG3: subcycling, inter-field parallelism and improved accuracy

3.3. Implicit/implicit staggered algorithms

Clearly, the partitioned analysis procedures ALG0, ALG1, ALG2 and ALG3 can be equally employed with an implicit flow solver. However, it is shown in [23] that in order for these partitioned procedures to be unconditionally stable, not only an unconditionally stable implicit flow solver must be used, but also an interface coupling operator must be exchanged between the structure and fluid field analyzers. For further details on this topic, we refer the reader to [23].

3.4. Implementation of subcycling

We have pointed out in Section 3.2.2 that when subcycling is desired, the deformation of the fluid mesh between t^n and t^{n+1} should not be entirely applied during the first fluid subcycle, but evenly distributed across all subcycling stages. There are many ways this can be accomplished, including the following one.

At the beginning of time-step t^{n+1}, the fluid code has access to the component of the structural state vector $(q^n,\ \dot{q}^n)_{\Gamma_{F/S}}$ that relates to the degrees of freedom located at the fluid/structure interface. The objective of any mesh updating strategy is to exploit this information and compute a fluid mesh displacement x^{n+1} that satisfies the continuity Eqs. (23)

$$x^{n+1} = q^n; \quad \dot{x}^{n+1} = \dot{q}^n \quad \text{on} \quad \Gamma_{F/S} \qquad (111)$$

We note that the difference in the superscripts between the left and right hand sides of Eqs. (111) is due to the staggered nature of the solution scheme, and that the second of Eqs. (111) should be enforced only if it does not violate the GCL. Using Eqs. (111) as prescribed boundary values, the pseudo-structural equations of motion of the dynamic fluid grid can be solved to obtain an updated fluid mesh displacement x^{n+1}. The details of this particular computation are discussed in Section 7. Then, at every subcycling stage, a new set of prescribed grid boundary displacements can be

generated for computing the subcycled mesh position $x_{\Gamma_{F/S}}^{n+1^{(s)}}$ as follows

$$\begin{aligned} x_{\Gamma_{F/S}}^{n+1^{(0)}} &= q_{\Gamma_{F/S}}^{n-1} \\ x_{\Gamma_{F/S}}^{n+1^{(s)}} &= I^d(x_{\Gamma_{F/S}}^{n+1^{(s-1)}},\ q_{\Gamma_{F/S}}^{n},\ \dot{q}_{\Gamma_{F/S}}^{n},\ \Delta t) \\ x_{\Gamma_{F/S}}^{n+1^{(n_{F/S})}} &= q_{\Gamma_{F/S}}^{n} \end{aligned} \tag{112}$$

where I^d is an interpolation scheme of order $d = 0,\ 1,\ 2$. More specifically, I^d is defined by

$$\begin{aligned} x_{\Gamma_{F/S}}^{n+1^{(s)}} &= q_{\Gamma_{F/S}}^{n} \quad (d=0) \\ x_{\Gamma_{F/S}}^{n+1^{(s)}} &= x_{\Gamma_{F/S}}^{n+1^{(s-1)}} \\ &\quad + \frac{(q_{\Gamma_{F/S}}^{n} - x_{\Gamma_{F/S}}^{n+1^{(s-1)}})}{n_{F/S}-s+1} \quad (d=1) \\ x_{\Gamma_{F/S}}^{n+1^{(s)}} &= x_{\Gamma_{F/S}}^{n+1^{(s-1)}} \\ &\quad + \frac{2q_{\Gamma_{F/S}}^{n} - 2x_{\Gamma_{F/S}}^{n+1^{(s-1)}}}{n_{F/S}-s+1} \\ &\quad - \frac{(n_{F/S}-s+1)\Delta t \dot{q}_{\Gamma_{F/S}}^{n}}{n_{F/S}-s+1} \\ &\quad + \frac{\dot{q}_{\Gamma_{F/S}}^{n} - (n_{F/S}-s+1)\Delta t}{(n_{F/S}-s+1)^2} \\ &\quad - \frac{q_{\Gamma_{F/S}}^{n} + x_{\Gamma_{F/S}}^{n+1^{(s-1)}}}{(n_{F/S}-s+1)^2} \quad (d=2) \end{aligned} \tag{113}$$

In summary, at each subcycle a new set of prescribed boundary displacements are computed for the dynamic fluid grid, and the equations of motion of the corresponding pseudo-structural system are solved in order to update the positions of the remaining grid points. In practice, we have found that $d = 1$ is the best choice for the interpolation scheme. Among other things, this choice does not require transmitting any structural velocity information to the fluid computational kernel. In all cases, the fluid mesh velocity $\dot{x}$ must be computed via Eq. (50) in order to satisfy the GCL.

4. THE FLOW SOLVER [46—49]

So far, no restriction has been imposed on the nonlinear flow solver technology, except for the requirement of satisfying the GCL. Hence, flow solvers based on an ALE finite volume/element discretization or a space-time finite element formulation can be equally employed within the computational framework presented in this paper for solving nonlinear transient aeroelastic problems. The GCLs for all of these approximation methods have been presented in Section 2.

In our case, we have opted for a mixed finite element/volume ALE formulation based on unstructured triangular meshes in two-dimensional problems, and unstructured tetrahedra in three-dimensional ones. This approach combines a Galerkin centered approximation for the viscous terms, and a Roe upwind scheme [50] for the computation of the convective fluxes. Higher order accuracy is achieved through the use of a piecewise linear interpolation method that follows the principle of the MUSCL (Monotonic Upwind Scheme for Conservative Laws) procedure [51–53]. The corresponding ALE flow solvers are the result of a collaboration with the Projet Sinus at INRIA Sophia-Antipolis and are overviewed below.

4.1. Spatial discretization

The conservative form of the equations describing viscous flows can be written in ALE form as

$$\begin{aligned} \frac{\partial(JW)}{\partial t}|_\xi + J\nabla_x.\mathcal{F}^c(W,\dot{x}) &= \frac{1}{Re}\nabla_x\mathcal{R}(W) \\ \mathcal{F}^c(W,\dot{x}) &= \mathcal{F}(W) - \dot{x}W \end{aligned} \tag{114}$$

where, $\mathcal{R}$ denotes the diffusive fluxes, Re is the Reynolds number, and as for the case of Euler flows and Eqs. (25), $J = det(dx/d\xi)$ is the jacobian of the frame transformation $\xi \rightarrow x$, W denotes the fluid conservative variables, $\mathcal{F}^c$ denotes the convective ALE fluxes, and $\dot{x} = \frac{\partial x}{\partial \theta}|_\xi$

is the ALE grid velocity that may be different from the fluid velocity and from zero.

The boundary $\Gamma(t)$ of the flow domain is partitioned into a wall boundary $\Gamma_w(t)$ corresponding to the fluid/structure interface boundary $\Gamma_{F/S}$, and an infinity boundary $\Gamma_\infty(t)$

$$\Gamma(t) \;=\; \Gamma_w(t) \cup \Gamma_\infty(t) \tag{115}$$

Let U_w and T_w denote the wall velocity and temperature. On the wall boundary $\Gamma_w(t)$, a no-slip and a temperature Dirichlet conditions are imposed

$$U \;=\; U_w; \;\; T \;=\; T_w \tag{116}$$

and no boundary condition is specified for the density. Hence, the total energy per unit of volume and the pressure on the wall are given by

$$p = (\gamma - 1)\rho C_v T_w; \;\; E = \rho C_v T_w + \frac{1}{2}\rho \parallel U_w \parallel^2 \tag{117}$$

For external flows around aircraft structures, the viscous effects are assumed to be negligible at infinity, and therefore a uniform free-stream state vector W_∞ is imposed on $\Gamma_\infty(t)$.

Following the procedure described in details in Section 2.1, Eqs. (114) can be transformed into

$$\frac{d}{dt}\int_{C_i(t)} W \; d\Omega_x + \int_{\partial C_i(t)} \nabla \mathcal{F}^c(W, \dot{x}) \; d\Omega_x = \int_{C_i(t)} \frac{1}{Re}\nabla R(W) \; d\Omega_x \tag{118}$$

Integrating Eq. (118) by parts leads to

$$\begin{aligned} &\frac{d}{dt}\int_{C_i(t)} W \; d\Omega_x \\ &+ \sum_j \int_{\partial C_{i,j}(x)} \mathcal{F}^c(W_i, W_j, \dot{x}).\nu_i \; d\sigma \\ &+ \int_{\partial C_i(t) \cap \Gamma(t)} \mathcal{F}^c(\overline{W}, \dot{x}).\nu_i \; d\sigma \\ &= -\frac{1}{Re} \sum \int \mathcal{R}(W).\nabla \phi_i^\Delta \; d\Omega_x \end{aligned} \tag{119}$$

where ϕ_i^Δ is the finite element shape function at vertex S_i in element Δ (a triangle in two-dimensional problems, a tetrahedron in three-dimensional ones), and $\overline{W}$ is the specified value of W at the boundaries. The second term of Eq. (119) corresponds exactly to the term $F_i^c(\mathbf{W}, \mathbf{x}, \dot{\mathbf{x}})$ introduced in Section 2.1. Each component of this term can be written as

$$\begin{aligned} \Phi_{ij} \;&=\; \int_{\partial C_{i,j}(x)} \mathcal{F}^c(W_i, W_j, \dot{x}).\nu_i \; d\sigma \\ &=\; \Phi(W_i, W_j, x, \dot{x}) \end{aligned} \tag{120}$$

While various upwind algorithms can be used for computing Φ_{ij}, Roe's scheme [50] is chosen here. Following the MUSCL approach introduced by Van Leer [51], second-order accuracy is achieved by computing the numerical fluxes at interpolated values of the fluid state vector on the interface between cells C_i and C_j as follows

$$\begin{aligned} \Phi_{ij} \;&=\; \Phi(W_{ij}, W_{ji}, x, \dot{x}) \\ W_{ij} \;&=\; W_i + \frac{1}{2}(\nabla W)_i.vector(S_i S_j) \\ W_{ji} \;&=\; W_j - \frac{1}{2}(\nabla W)_j.vector(S_i S_j) \end{aligned} \tag{121}$$

For three-dimensional problems, the gradient of W at a vertex S_i is computed from

$$V_i(\nabla W)_i \;=\; \sum_{\Delta, S_i \in \Delta} \frac{volume(\Delta)}{4} \sum_{k=1}^{k=4} W_k.\nabla \phi_k^\Delta \tag{122}$$

In practice, the interpolation implied by Eqs. (121) is performed on the physical instead of the the conservative variables. Optional limiters are also implemented following the approaches discussed in [52,53].

The numerical viscous fluxes are computed using a classical Galerkin method.

4.2. Temporal solution

The explicit kernel of our two-dimensional flow solver uses the 3-step Runge-Kutta algorithm discussed in Section 3.2. On the other hand, the explicit module of our three-dimensional flow solver employs the predictor-corrector scheme suggested by Hancock and presented by Van Leer. This scheme has a lower stability limit than the 3-step Runge-Kutta algorithm but is significantly more economical.

The implicit versions of our two- and three-dimensional flow solvers employ a first-order accurate backward Euler time-integration scheme. To solve the system of linearized equations arising at each time-step, we have recently developed a multilevel, overlapping domain decomposition preconditioned Krylov-Schwarz iterative method [54]. Numerical experiments have shown that this and other members of the family of Krylov-Schwarz algorithms are highly scalable and highly parallelizable. (The concept of scalability is discussed in Section 5). More importantly, the convergence of these methods does not degenerate when the linearized system becomes highly nonsymmetric and possibly indefinite, which occurs, for example, in the case of high-angle of attack and/or high Mach number.

4.3. Parallelization

The mesh partitioning paradigm is used for parallelizing both two-dimensional and three-dimensional flow solvers. This paradigm is discussed in details in Section 8.

5. THE STRUCTURAL DYNAMICS ANALYZER [74,76,34,81,57]

There is no question that the finite element method is the most popular method for solving arbitrary structural problems such as those governed by the second of Eqs. (22). However, with the advent of parallel processing, many of the computational modules of this powerful method are being constantly revisited for improvement in performance.

Nonlinear transient finite element problems in structural mechanics are characterized by the semi-discrete equations of dynamic equilibrium

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{f}^{int}(\mathbf{q}) = \mathbf{f}^{ext} \qquad (123)$$

where, as before, $\mathbf{M}$ is the mass matrix, $\mathbf{q}$ is the vector of nodal displacements, a dot superscript indicates a time derivative, $\mathbf{f}^{int}$ is the vector of internal nodal forces, and $\mathbf{f}^{ext}$ is the vector of external nodal forces. In many low and medium frequency dynamics applications such as transient aeroelasticity, Eq. (123) is most efficiently solved using an implicit time-integration scheme. In that case, a nonlinear algebraic system of equations is generated at each time-step. The Newton-Raphson method and its numerous variants collectively known as "Newton-like" methods are the most popular strategies for solving these nonlinear algebraic problems. All of these algorithms require the solution of a linear algebraic system of equations of the form

$$\begin{aligned} \mathbf{K}^*(\mathbf{q}_{n+1}^{(k)})\ \Delta\mathbf{q}_{n+1}^{(k+1)} &= \mathbf{r}^*(\mathbf{q}_{n+1}^{(k)}) \\ \Delta\mathbf{q}_{n+1}^{(k+1)} &= \mathbf{q}_{n+1}^{(k+1)} - \mathbf{q}_{n+1}^{(k)} \end{aligned} \qquad (124)$$

where the subscript n refers to the n-th time step, the superscript k refers to the k-th nonlinear iteration within the current time step, $\mathbf{K}^*$ is a time-dependent symmetric positive approximate tangent matrix that includes both mass and stiffness contributions, and $\Delta\mathbf{q}_{n+1}^{(k+1)}$ and $\mathbf{r}^*(\mathbf{q}_{n+1}^{(k)})$ are respectively the vector of nodal displacement increments and the vector of out-of-balance nodal forces (dynamic residuals).

With the advent of parallel processing, domain decomposition (or substructure) based direct and iterative algorithms have become increasingly popular for the solution of finite element systems of equations of the form given

in Eq. (123). Indeed, domain decomposition provides a higher level of concurrency than parallel global algebraic paradigms, and is simpler to implement on most parallel computational platforms [57]. In general, the subdomain (or substructure) equations are solved using a direct skyline or sparse factorization based algorithm, while both direct and iterative schemes have been proposed for the solution of the interface problem [58–62]. When the reduced system of equations is solved directly, the overall domain decomposition algorithm becomes a direct frontal or multifrontal method [63,64], and its success becomes contingent on finding a good mesh partition and/or reordered system that can achieve an optimal balance between minimizing fill-in and increasing the degree of parallelism [65–68]. When the interface problem is solved iteratively — usually, via a preconditioned conjugate gradient (PCG) algorithm— the overall domain decomposition method becomes a genuine iterative solver whose success hinges on two important properties: *numerical* scalability, and *parallel* scalability. A domain decomposition based iterative method is said to be numerically scalable if the condition number κ after preconditioning does not grow or grows "weakly" with the ratio of the subdomain size H and the mesh size h (Fig. 14), that is

$$\kappa = O\left(1 + log^{\beta}\left(\frac{H}{h}\right)\right) \qquad (125)$$

with a small constant β. Numerous authors have proved Eq. (125) with $\beta = 2$ for various domain decomposition methods (see, for example, [62,69,70] and references therein).

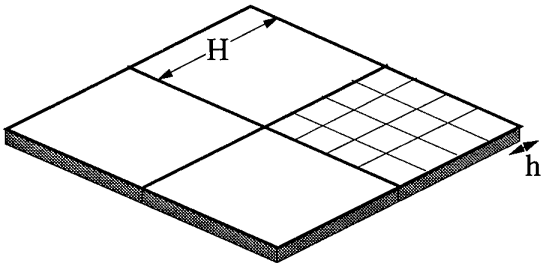


Fig. 14. Subdmomain size H and mesh size h

It is well known that in order to achieve (125), a domain decomposition method must involve a *coarse problem* with a few d.o.f. per subdomain, that must be solved at each iteration to propagate the error globally and accelerate convergence. Parallel scalability characterizes the ability of an algorithm to deliver larger speedups for a larger number of processors. In particular, parallel scalability is necessary for massively parallel processing.

The practical implications of a condition number after preconditioning such as that described in Eq. (125) are

- suppose that a given mesh is fixed, one processor is assigned to every subdomain, and the number of subdomains (which varies as $1/H$) is increased in order to increase parallelism. In that case, h is fixed and H is decreased. From Eq. (125), it follows that the bound on the condition number decreases and therefore *the number of iterations for convergence is expected to decrease with an increasing number of subdomains.* In particular, for a numerically scalable domain decomposition algorithm characterized by Eq. (125), increasing the number of subdomains decreases the amount of work per processor and per iteration, without increasing the number of iterations for convergence.
- on most distributed memory parallel processors, the total amount of available memory increases with the number of processors. When solving a certain class of problems on such parallel hardware, it is customary to define in each processor a constant subproblem size, and to increase the total problem size with the number of processors. In that case, h and H are decreased, but the ratio H/h is kept constant. From Eq. (125), it follows

that a numerically scalable domain decomposition algorithm can solve larger problems with the same number of iterations as smaller ones, simply by increasing the number of subdomains. However, the presence of the coarse problem may limit parallel scalability for a large number of processors.

- When H/h increases, that is, the number of elements assigned to a subdomain increases, the condition number will increase only slightly. Without this property, the condition number may be too large to be practical for subdomains of a size that we wish to work with. If there are only a few substructures, the conjugate gradient algorithm might still converge quickly for some domain decomposition methods because of the presence of gaps in the spectrum of the preconditioned operator; however, for large number of subdomains, the spectrum tends to fill in, and the number of iterations tends to increase [32].

The Finite Element Tearing and Interconnecting (FETI) method developed originally for the solution of self-adjoint elliptic partial differential equations is a numerically scalable domain decomposition method [60,61,32]. This method was shown to outperform direct skyline solvers and several popular iterative algorithms on both sequential and parallel computing platforms [60,73]. It has recently been extended for dynamics problems [74,33] and biharmonic partial differential equations such as those encountered in plate and shell problems [75]. For structural mechanics problems, the condition number of the unpreconditioned FETI interface problem is known to grow asymptotically as [32]

$$\kappa = O\left(\frac{H}{h}\right) \qquad (126)$$

As was observed numerically in [32,57] and proved mathematically in [70,75], for elasticity problems discretized using plane stress/strain and/or brick elements, the condition number of the FETI interface problem preconditioned with a subdomain based Dirichlet operator [32,57] varies as

$$\kappa = O\left(1 + log^{\beta}\left(\frac{H}{h}\right)\right), \quad \beta \leq 3. \qquad (127)$$

For shell and plate problems, this condition number varies as [75]

$$\kappa = O\left(1 + log^{2}\left(\frac{H}{h}\right)\right)$$

The conditioning results (126–128) highlight the numerical scalability of the FETI method with respect to both the mesh size h and the number of subdomains. The parallel scalability of this domain decomposition method — that is, its ability to achieve larger speedups for larger number of processors — has also been demonstrated on current massively parallel processors for several realistic structural problems [57,71,72].

The beauty of the FETI method resides in the fact that it is much more than an algebraic solver. Many complex structural systems such as airplanes are constructed by assembling a set of substructures such as the wing, fuselage, spars and ribs, tail, that are designed by different teams of engineers. The global behavior of such structures is often predicted by "gluing" together the individual substructure analyses. In such cases, the submeshes associated with the substructures may have nonconforming discrete interfaces, mainly because: (a) the corresponding substructures can have different resolution requirements, (b) the submeshes are often designed by different analysts, and (c) these submeshes may be designed using incompatible finite element models. Whether the substructure interfaces are matching or not, the FETI method provides a powerful means for solving such assembly problems [76–78]. In essence, the FETI method is at the same time

a domain decomposition and a domain integration method, and lends itself naturally to parallelism. We overview it next in the context of structural dynamics.

5.1. The Transient FETI Method

Let Ω denote the volume of the structure to be analyzed, and $\{\Omega^s\}_{s=1}^{s=N_s}$ denote a partitioning (tearing) of Ω into N_s non-overlapping substructures (Fig. 15). We denote by Γ_I^s the interface boundary of Ω^s. We use an irreducible displacement formulation inside the subdomains, and independently defined Lagrange multipliers on the substructure interfaces to join them.

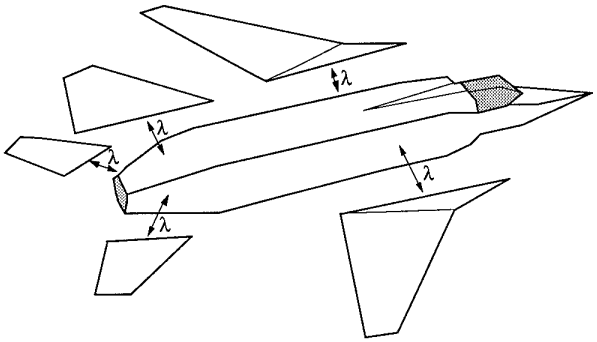


Fig. 15. Partitioning of a structure

For each substructure, the finite element nonlinear equations of dynamic equilibrium can be written as

$$\mathbf{M}^s\ddot{\mathbf{q}}^s + \mathbf{f}^{int^s}(\mathbf{q}) = \mathbf{f}^{ext^s} - \mathbf{B}^{s^T}\boldsymbol{\lambda} \qquad (128)$$

where $\mathbf{B}^s$ is a boolean matrix with entries equal to -1, 0, $+1$ that extracts from a substructure quantity those components that are related to the interface boundary Γ_I^s, and $\boldsymbol{\lambda}$ (not to be confused with its previous use for the dynamic pressure in Section 1.1) is the vector of Lagrange multipliers representing the traction forces needed for enforcing on the substructure interfaces the continuity of the displacement field

$$\sum_{s=1}^{s=N_s} \mathbf{B}^s\mathbf{q}^s = 0 \qquad (129)$$

Using the notation of Eq. (124), the linearization of Eqs. (128,129) around $\mathbf{q}_{n+1}^s$ can be formulated as

$$\begin{aligned} \mathbf{M}^s\Delta\ddot{\mathbf{q}}_{n+1}^{s(k+1)} + \mathbf{K}^s\Delta\mathbf{q}_{n+1}^{s(k+1)} &= \mathbf{r}_{n+1}^{*s(k)} - \mathbf{B}^{s^T}\boldsymbol{\lambda}_{n+1}^{(k+1)} \\ s &= 1, \ ..., \ N_s \\ \sum_{s=1}^{s=N_s} \mathbf{B}^s\Delta\mathbf{q}_{n+1}^{s(k+1)} &= 0 \end{aligned} \qquad (130)$$

where $\mathbf{K}^s$ denotes here the subdomain *tangent* stiffness matrix. Eqs. (130) are known as differential/algebraic equations (DAEs). They are more difficult to solve than the usual ordinary differential equations [79].

5.2. Implicit Time-Integration

Let $\Delta\mathbf{v}_{n+\frac{1}{2}}^{(k+1)} = \left[\Delta\mathbf{v}_{n+\frac{1}{2}}^{1(k+1)} \ \ ... \ \ \Delta\mathbf{v}_{n+\frac{1}{2}}^{N_s(k+1)}\right]$ denote the momentum increment at iteration $k+1$ and at the midpoint between steps n and $n+1$, and let $\mathbf{M} = [\mathbf{M}^1 \ \ ... \ \ \mathbf{M}^{N_s}]$. We have

$$\Delta\mathbf{v}_{n+\frac{1}{2}}^{(k+1)} = \mathbf{M}\,\Delta\dot{\mathbf{q}}_{n+\frac{1}{2}}^{(k+1)} \qquad (131)$$

In [76], it was shown that the following time-integration algorithm for solving the DAEs (13) is *second-order accurate and unconditionally stable*

1. Solve:

$$(\mathbf{M}^s + \frac{\Delta t^2}{4}\mathbf{K}^s)\Delta\mathbf{q}_{n+\frac{1}{2}}^{s(k+1)} = \frac{\Delta t^2}{4}(\mathbf{r}_{n+\frac{1}{2}}^{*s(k)} - \mathbf{B}^{sT}\boldsymbol{\lambda}^{(k+1)}) \quad s = 1, ..., N_s$$

$$\sum_{s=1}^{s=N_s} \mathbf{B}^s \Delta\mathbf{q}_{n+\frac{1}{2}}^{s(k+1)} = 0$$

2. Update:

$$\begin{aligned} \mathbf{q}_{n+\frac{1}{2}}^s &= \mathbf{q}_{n+\frac{1}{2}}^{s(k)} + \Delta\mathbf{q}_{n+\frac{1}{2}}^{s(k+1)} \\ \dot{\mathbf{v}}_{n+\frac{1}{2}} &= \mathbf{f}_{n+\frac{1}{2}}^{ext} - \mathbf{f}^{int}(\mathbf{q}_{n+\frac{1}{2}}) \\ \mathbf{q}_{n+1}^s &= 2\mathbf{q}_{n+\frac{1}{2}}^s - \mathbf{q}_n^s \\ \mathbf{v}_{n+1}^s &= \mathbf{v}_n^s + \Delta t\dot{\mathbf{v}}_{n+\frac{1}{2}}^s \end{aligned} \tag{132}$$

The computational cost of the above implicit time-integration algorithm is dominated by the cost incurred at each time step for the solution of a constrained system of the form

$$\begin{aligned} \mathbf{K}^{*s}\mathbf{q}^s &= \mathbf{g}^s - \mathbf{B}^{sT}\boldsymbol{\lambda} \quad s = 1, ..., N_s \\ \sum_{s=1}^{s=N_s} \mathbf{B}^s\mathbf{q}^s &= 0 \end{aligned} \tag{133}$$

where a simplified notation has been used, and $\mathbf{K}^{*s}$ is given by

$$\mathbf{K}^{*s} = \mathbf{M}^s + \frac{\Delta t^2}{4}\mathbf{K}^s \tag{134}$$

After some algebraic manipulations, Eqs. (133) above can be rewritten as

$$\mathbf{F}_I\boldsymbol{\lambda} = \mathbf{d} \tag{135}$$

where $\mathbf{F}_I$ and $\mathbf{d}$ are given by

$$\mathbf{F}_I = \sum_{s=1}^{s=N_s} \mathbf{B}^s\mathbf{K}^{*s^{-1}}\mathbf{B}^{sT}; \quad \mathbf{d} = \sum_{s=1}^{s=N_s} \mathbf{B}^s\mathbf{K}^{*s^{-1}}\mathbf{g}^s \tag{136}$$

Note that the FETI domain decomposition method transforms the original *primal* problem described in Eq. (124) into a *dual* interface problem. The dual interface operator $\mathbf{F_I}$ is in general symmetric positive semi-definite. It has interesting spectral properties [32,57,74] which induce a superconvergent behavior of a PCG algorithm applied to the solution of Eq. (135). The parallelization of a conjugate gradient scheme applied to the solution of the dual interface problem is straight forward, because $\mathbf{F_I}$ is a sum of independent substructure operators. All CG related computations can be performed in parallel on a substructure-by-substructure basis.

5.3. The FETI PCG Parallel Algorithm

We have developed two preconditioners for the FETI method: (1) a numerically optimal Dirichlet preconditioner that can be written as

$$\overline{\mathbf{F}}_I^{D^{-1}} = \sum_{s=1}^{s=N_s} \mathbf{B}^s \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{K}_{bb}^{*s} - \mathbf{K}_{ib}^{*sT}\mathbf{K}_{ii}^{*s^{-1}}\mathbf{K}_{ib}^{*s} \end{bmatrix} \mathbf{B}^{sT} \tag{137}$$

where the subscripts i and b designate here internal and interface boundary unknowns, respectively, and (2) a numerically efficient "lumped" preconditioner that lumps the Dirichlet operator on the subdomain interface unknowns

$$\overline{\mathbf{F}}_I^{L^{-1}} = \sum_{s=1}^{s=N_s} \mathbf{B}^s \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{K}_{bb}^{*s} \end{bmatrix} \mathbf{B}^{sT} \tag{138}$$

Unlike $\overline{\mathbf{F}}_I^{D^{-1}}$, the preconditioner $\overline{\mathbf{F}}_I^{L^{-1}}$ is not mathematically optimal. However, it is more economical than $\overline{\mathbf{F}}_I^{D^{-1}}$, and has often proved to be more efficient [32,57].

Let $\overline{\mathbf{F}_I^{-1}}$ denote either the Dirichlet or lumped preconditioner, and let $\mathbf{G}_I$ denote the matrix collecting the traces on the substructure interfaces of the the rigid body modes $\mathbf{R}^s$ of the N_f floating substructures Ω^s —- that is, the substructures without enough boundary conditions to prevent the local tangent stiffness matrices $\mathbf{K}^s$ from being singular

$$\mathbf{G}_I = [\mathbf{B}^1\mathbf{R}^1 \quad ... \quad \mathbf{B}^{N_f}\mathbf{R}^{N_f}] \qquad (139)$$

Using $\mathbf{G}_I$, we introduce the projection operator

$$\mathbf{P} = \mathbf{I} - \mathbf{G}_I(\mathbf{G}_I^T\mathbf{F}_I\mathbf{G}_I)^{-1}\mathbf{G}_I^T\mathbf{F}_I \qquad (140)$$

where $\mathbf{I}$ denotes the identity matrix. In three-dimensional problems, each floating substructure can have up to 6 rigid body modes. Therefore, $\mathbf{G}_I^T\mathbf{F}_I\mathbf{G}_I$ is a square matrix of size equal at most to $6 \times N_f \leq 6 \times N_s$ (6 unknowns per floating substructure).

The transient FETI PCG algorithm [33] for solving Eq. (135) goes as follows

1. Initialize

$$\boldsymbol{\lambda}^0 = \mathbf{G}_I\,(\mathbf{G}_I{}^T\mathbf{F}_I\mathbf{G}_I)^{-1}\,\mathbf{G}_I^T\mathbf{d}$$
$$\mathbf{r}^0 = \mathbf{d} - \mathbf{F}_I\boldsymbol{\lambda}^0$$

2. Iterate k = 1, 2, ... until convergence

$$\textit{Project}\quad \mathbf{w}^{k-1} = \mathbf{P}^T\,\mathbf{r}^{k-1}$$
$$\textit{Precond.}\quad \mathbf{z}^{k-1} = \overline{\mathbf{F}_I^{-1}}\mathbf{w}^{k-1}$$
$$\textit{Project}\quad \mathbf{y}^{k-1} = \mathbf{P}\,\mathbf{z}^{k-1}$$
$$\zeta^k = \mathbf{y}^{k-1^T}\mathbf{w}^{k-1}/\mathbf{y}^{k-2^T}\mathbf{w}^{k-2}$$
$$\mathbf{p}^k = \mathbf{y}^{k-1} + \zeta^k\mathbf{p}^{k-1}$$
$$\nu^k = \mathbf{y}^{k-1^T}\mathbf{w}^{k-1}/\mathbf{p}^{k^T}\mathbf{F}_I\mathbf{p}^k$$
$$\boldsymbol{\lambda}^k = \boldsymbol{\lambda}^{k-1} + \nu^k\mathbf{p}^k$$
$$\mathbf{r}^k = \mathbf{r}^{k-1} - \nu^k\mathbf{F}_I\mathbf{p}^k$$

(141)

The application of the projection operator $\mathbf{P}$ defined in (140) means that a coarse problem of the form $(\mathbf{G}_I{}^T\mathbf{F}_I\mathbf{G}_I)\mathbf{x} = \mathbf{b}$ must be solved twice within each FETI iteration. It was shown in [32] that this coarse problem has the expected beneficial effect of coupling all substructure computations and propagating the error globally, so that the condition number of the preconditioned interface problem can be bounded as a function of H/h and independently of the number of substructures, which ensures the numerical scalability of the FETI method.

For shell and plate problems, the definition of $\mathbf{R}^s$ is slightly modified to include not only the substructure rigid body modes, but also the substructure "corner" modes [75]. Otherwise, the remainder of the FETI algorithm remains essentially the same.

5.4. Optimization for Problems with Multiple/Repeated R.H.S.

One of the many reasons why numerical scalability is desirable is that increasing the number of subdomains is the simplest means for increasing the degree of parallelism of a domain decomposition based PCG algorithm. As illustrated in the previous paragraph, this optimal property is usually achieved via the introduction in a domain decomposition method of a coarse problem (or coarse grid, by analogy with multigrid methods) that relates to the original problem and that must be solved at each global CG iteration. Direct methods are often chosen for solving the coarse problem despite the fact that they are difficult to implement on a massively parallel processor and do not parallelize as well as iterative schemes. Therefore in many cases, a numerically scalable domain decomposition method loses its appeal because of its lack of parallel scalability. One way to restore parallel scalability is to solve iteratively the coarse

problem, for example using a CG scheme. However, because the coarse problem is embedded in an outer iterative loop, this approach raises the question of how to solve iteratively and efficiently a system with a constant matrix and repeated right-hand sides. Finding an answer to this question also extends the range of applications of domain decomposition based iterative methods to design problems, eigenvalue problems, and several other applications where multiple and repeated right-hand sides always arise and challenge iterative solvers. Such examples include nonlinear transient aeroelastic simulations where the structure is assumed to remain in the linear regime. In that case, the left hand side $\mathbf{F}_I$ of Eq. (135) remains constant in time, but its right hand side (r.h.s.) $\mathbf{d}$ varies in time.

The iterative solution of systems with multiple and/or repeated right-hand sides has been previously addressed in [80], and recently in [34,81]. Here, we overview the CG based methodology for solving such problems that uses the same data structures as those employed in domain decomposition methods without a coarse grid and which was first presented in [34,81]. The basic idea is related to that analyzed in [80]. However, the algorithm we have developed is different, simpler, and easier to parallelize than that described in [80].

Since $\mathbf{G}_I^T\mathbf{F}_I$ appears twice in the expression of the projector $\mathbf{P}$ (140), we first construct $\mathbf{Q} = \mathbf{F}_I\mathbf{G}_I$. Suppose that the solution of the first encountered coarse problem $(\mathbf{G}^T\mathbf{Q})\mathbf{x}^1 = \mathbf{b}^1$ has been obtained after n^1 CG iterations. Let $\mathbf{S}^1$ denote the space of the $(\mathbf{G}^T\mathbf{Q})$-orthogonal search directions generated during these n^1 CG iterations. If explicit re-orthogonalization is implemented in the CG algorithm [57], ${\mathbf{S}^1}^T(\mathbf{G}^T\mathbf{Q})\mathbf{S}^1$ is guaranteed to be a diagonal matrix. In order to compute the solution of the next coarse problem $(\mathbf{G}^T\mathbf{Q})\mathbf{x}^2 = \mathbf{b}^2$, we proceed as follows

Step 1. we project the problem $(\mathbf{G}^T\mathbf{Q})\mathbf{x}^2 = \mathbf{b}^2$ onto $\mathbf{S}^1$ and solve the trivial diagonal system ${\mathbf{S}^1}^T(\mathbf{G}^T\mathbf{Q})\mathbf{S}^1\mathbf{y}^{2^0} = {\mathbf{S}^1}^T\mathbf{b}^2$. Then, we perform a matrix-vector multiplication to obtain $\mathbf{x}^{2^0} = \mathbf{S}^1\mathbf{y}^{2^0}$. In [34], it is argued that $\mathbf{x}^{2^0}$ is an optimal startup value for $\mathbf{x}^2$ because: (a) it minimizes $\mathbf{x}^T(\mathbf{G}^T\mathbf{Q})\mathbf{x}/2 - \mathbf{x}^T\mathbf{b}^2$ over $\mathbf{S}^1$, and (b) it is inexpensive to compute. Note that the n^1 nonzero entries of the diagonal matrix ${\mathbf{S}^1}^T(\mathbf{G}^T\mathbf{Q})\mathbf{S}^1$ are readily available from the CG solution of the previous coarse problem $(\mathbf{G}^T\mathbf{Q})\mathbf{x}^1 = \mathbf{b}^1$. Therefore, these entries can be stored and need not be re-computed.

Step 2. next, we apply the CG algorithm to the solution of $(\mathbf{G}^T\mathbf{Q})\mathbf{x}^2 = \mathbf{b}^2$ after it has been modified to: (a) accept $\mathbf{x}^{2^0}$ as a startup solution, and (b) perform the explicit orthogonalization of the new search directions and $\mathbf{S}^1$ with respect to $(\mathbf{G}^T\mathbf{Q})$.

The solution of all subsequent coarse problems is carried out using the same two-step procedure outlined above. Essentially, the space of previous search directions is constantly enriched with the most recently computed ones, and orthogonalization with respect to $(\mathbf{G}^T\mathbf{Q})$ is always performed. The storage requirements associated with this methodology are minimal because it is applied to coarse and therefore small problems (see [34] for further details). Because full precision is required for the solution of all coarse problems, the solution of the first one typically converges in a number of iterations equal to the size of the matrix $(\mathbf{G}^T\mathbf{Q})$ — that is, the total number of substructure rigid body modes — and all subsequent coarse problems are solved in *zero iteration*, using only the optimal startup value.

Clearly, the methodology outlined above for solving iteratively and efficiently a system

of equations with a constant matrix and repeated right-hand sides is equally applicable to any (symmetric) system of the form $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is of a relatively small size. In particular, it is applicable to Eq. (135) since the size of $\mathbf{F}_I$ is equal to the number of interface d.o.f., and that number is usually less than 30% of the total number of structural d.o.f. Therefore, this methodology can be used for solving nonlinear transient aeroelasticity problems where the structure remains in the linear regime. In that case, $\mathbf{F}_I$ is the same at all time-steps, and $\mathbf{d}$ varies in time with the pressure associated with the unsteady flow.

As an example, we apply the methodology described above to the solution of the repeated systems arising from the linear transient analysis using an implicit time-integration scheme of the three-dimensional stiffened wing of a High Speed Civil Transport (HSCT) aircraft (Fig. 16). The structure is modeled with 6,204 triangular shell elements, 456 beam elements, and includes 18,900 d.o.f. The finite element mesh is partitioned into 32 subdomains with excellent aspect ratios using TOP/DOMDEC [82]. The size of the interface problem is 3,888 — that is, 20.57% of the size of the global problem. The transient analysis is carried out on a 32-processor iPSC-860 system. After all of the usual finite element storage requirements are allocated, there is enough memory left to store a total number of 360 search directions. This number corresponds to 9.25 % of the size of the interface problem.

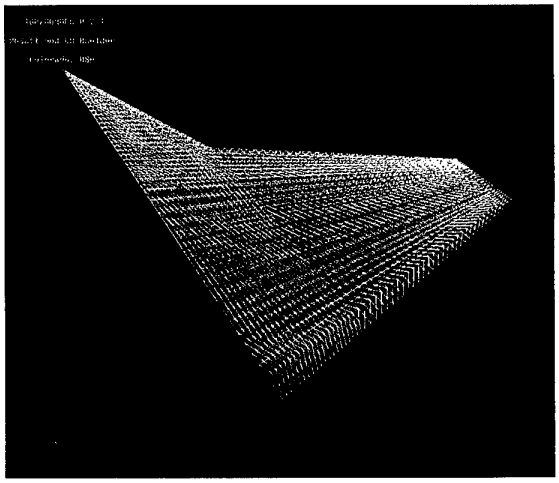


Fig. 16. HSCT stiffened wing

Using the transient FETI method, the system of equations arising at the first time step is solved in 30 iterations and 7.75 seconds CPU. After 5 time steps, 89 search directions are accumulated and only 10 iterations are needed for solving the fifth linear system of equations (Fig. 17). After 45 time steps, the total number of accumulated search directions is only 302 — that is, only 7.76% of the size of the interface problem, and superconvergence is triggered: all subsequent time steps are solved in 2 or 3 iterations (Fig. 17) and in less than 0.78 second CPU (Fig. 18).

When a parallel skyline direct solver is applied to the above problem, the factorization phase consumes 60.5 seconds CPU, and at each time step the pair of forward/backward substitutions requires 10.65 seconds on the same 32 processor iPSC-860. Therefore, the solution methodology described herein is clearly an excellent alternative to repeated forward/backward substitutions on distributed memory parallel processors.

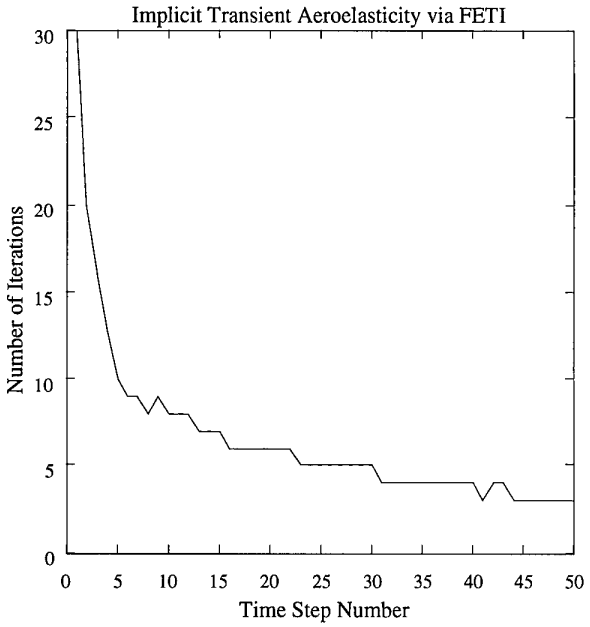


Fig. 17. Convergence rate history

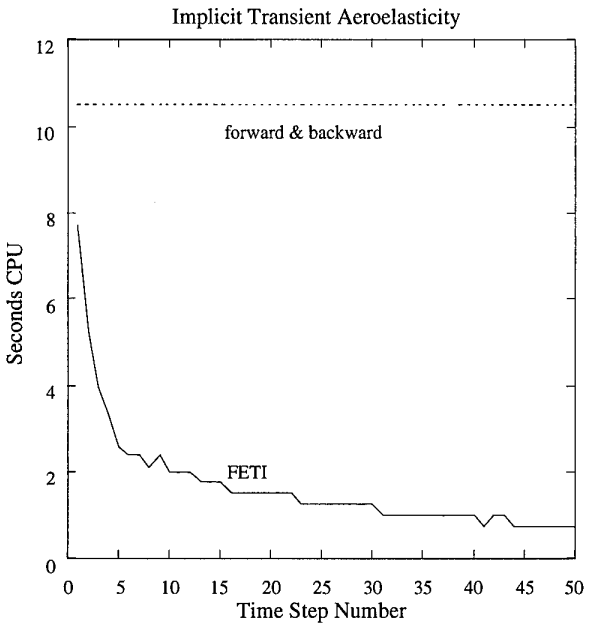


Fig. 18. CPU history

6. NON-MATCHING INTERFACE BOUNDARIES [35]

All four partitioned analysis procedures discussed in Section 3 require exchanging interface-data only between the field analyzers. More precisely, the structure expects to receive the values of the flow pressure and viscous stresses at the fluid/structure interface boundary $\Gamma_{F/S}$, and convert them into a structural load. Similarly, the fluid expects to receive from the structure the displacement and/or velocity of the interface boundary $\Gamma_{F/S}$, and use them to update the position of the dynamic fluid mesh. This exchange is performed at every time-step, or as required by the subcycling algorithm.

In general, the fluid and structure meshes have two independent representations of the physical fluid/structure interface. When these representations are identical — for example, when every fluid grid point on $\Gamma_{F/S}$ is also a structural node and *vice-versa* — the evaluation of the pressure forces and the transfer of the structural motion to the fluid mesh are trivial operations. However, analysts usually prefer to

- use a fluid mesh and a structural model that have been independently designed and validated.
- refine each mesh independently from the other.

Hence, most realistic aeroelastic simulations will involve handling fluid and structural meshes that are incompatible at their interface boundaries (Fig. 19). In [35], we have addressed this issue and proposed a preprocessing "matching" procedure that does not introduce any other approximation than those intrinsic to the fluid and structure solution methods. This procedure can be summarized as follows.

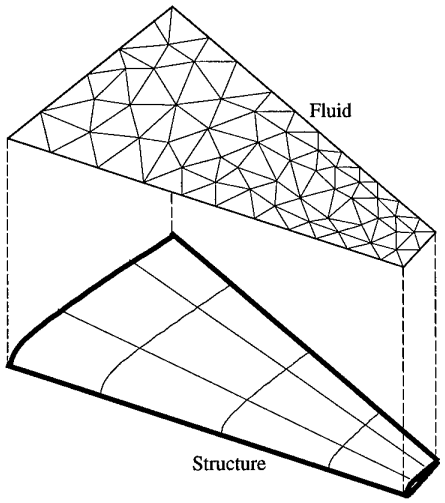


Fig. 19. Incompatible fluid and structure meshes

The nodal forces induced by the fluid pressure on the "wet" surface of a structural element e can be written as:

$$f_i^{ext} = \int_{\Omega^{(e)}} N_i(-p\nu + (\tau\nu)\bar{\nu})\, d\sigma \qquad (142)$$

where $\Omega^{(e)}$ denotes the geometrical support of the wet surface of the structural element e, p is the pressure field, τ is the tensor of viscous stresses, ν is the unit normal to $\Omega^{(e)}$, $\bar{\nu}$ is a tangent to the plane of $\Omega^{(e)}$, and N_i is the shape function associated with node i in element e. Most if not all finite element structural analysis codes evaluate the integral in Eq. (142) via a quadrature rule

$$f_i^{ext} = \sum_{g=1}^{g=n_g} w_g N_i(X_g)(-p(X_g)\nu + (\tau(X_g)\nu)\bar{\nu}) \qquad (143)$$

where w_g is the weight of the Gauss point X_g. Hence, a structural analysis code needs to know only the values of the pressure field at the Gauss points of its wet surface. This information can be easily made available once every Gauss point of a wet structural element is paired with a fluid cell (Fig. 20). It should be noted that in Eq. (143), X_g are not necessarily the same Gauss points as those used for stiffness evaluation. For example, if a high pressure gradient is anticipated over a certain wet area of the structure, a larger number of Gauss points can be used for the evaluation of the pressure forces f_i^{ext} on that area.

On the other hand, when the structure moves and/or deforms, the motion of the fluid grid points on $\Gamma_{F/S}$ can be prescribed via the regular finite element interpolation

$$x(S_j) = \sum_{k=1}^{k=wne} N_k(\mathcal{X}_j) q_k^{(e)} \qquad (144)$$

where S_j, wne, $\mathcal{X}_j$, and q_k denote respectively a fluid grid point on $\Gamma_{F/S}$, the number of wet nodes in its "nearest" structural element e, the natural coordinates of S_j in $\Omega^{(e)}$, and the structural displacement at the k-th node of element e. From Eq. (144), it follows that each fluid grid point on $\Gamma_{F/S}$ must be matched with one wet structural element (Fig. 21).

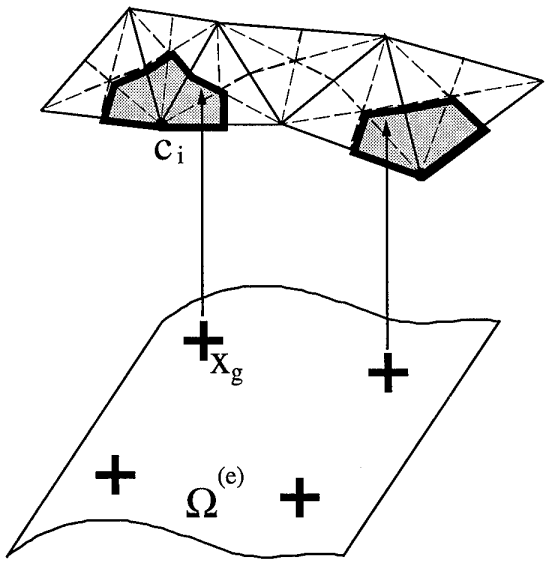


Fig. 20. Gauss-point—fluid cell pairing

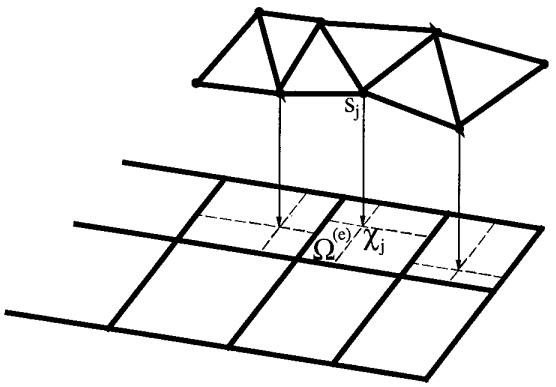


Fig. 21. Fluid grid point—wet structural element pairing

Given a fluid grid, a structural analysis model, and a discrete description of the fluid/structure interface, the Matcher program described in [35] generates all the data structures needed for evaluating the quantities described in Eqs. (143,144). If parallel data structures — for example, data structures associated with mesh partitions of the fluid and structure grids (see Section 8) — are fed as input, Matcher outputs parallel data structures that allow a painless implementation of the interface-data exchange between the field analyzers and are fully compatible with the intrinsic parallel data structures of the fluid and structural analysis programs. In general, the pairing of fluid and structure entities does not change in time. Therefore, Matcher is run as a preprocessor. The pairing data structures are generated only once, prior to any aeroelastic computation.

Finally, we note that Matcher is written in a message-passing style. Therefore, this software is portable to any parallel computing platform that supports a PVM- or MPI-like communication library. Of course, it also runs on sequential computers. For a complete aircraft configuration where the fluid mesh contained 439272 tetrahedra, 77279 vertices, and was partitioned into 32 subdomains, and the structural model contained 7520 triangular shell elements, 3841 nodes, and was partitioned into 16 subdomains, Matcher generated the desired fluid/structure pairing data structures in 327 seconds CPU on a 32-processor iPSC-860 system [35].

7. THE MESH MOTION SOLVER [44]

At the beginning of each step of the chosen staggered solution procedure, the dynamic fluid grid must be updated to conform to the most recently computed configuration of the structure. In general, this is done in two steps

Step 1. first, the points lying on the interface boundary $\Gamma_{F/S}$ are adjusted to match (in the sense defined in Section 6) the newly computed or predicted position of the surface of the structure. This defines a prescribed displacement vector $\mathbf{x}_{\Gamma_{F/S}}$.

Step 2. next, the remainder of the fluid grid points are repositioned accordingly to the prescribed values of $\mathbf{x}_{\Gamma_{F/S}}$. This completes the computation of the new mesh displacement vector $\mathbf{x}$.

Several procedures have been proposed in the literature for implementing the above two steps. Most of them can be summarized as viewing the fluid domain or its grid as a pseudo-structural continuous or discrete system. For example, in the discrete approach, either or all of the following can be done (see Fig. 3)

- lumping a fictitious mass at each vertex of the fluid mesh.
- introducing a fictitious dashpot at each edge connecting two vertices.
- attaching a fictitious spring on each edge connecting two vertices.

Similarly, a pseudo-structural continuous system can be generated with fictitious distributed structural properties. In both cases, the motion of the constructed pseudo-structural system is governed by

$$\widetilde{\mathbf{M}}\ddot{\mathbf{x}} + \widetilde{\mathbf{D}}\dot{\mathbf{x}} + \widetilde{\mathbf{K}}\mathbf{x} = \widetilde{\mathbf{K}}_c \mathbf{x}_{\Gamma_{F/S}} \qquad (145)$$

where $\widetilde{\mathbf{M}}$, $\widetilde{\mathbf{D}}$, and $\widetilde{\mathbf{K}}$ are the fictitious mass, damping, and stiffness matrices associated with the dynamic fluid mesh, and $\widetilde{\mathbf{K}}_c$ is the component of the fictitious stiffness matrix that represents the coupling between the fluid points lying on $\Gamma_{F/S}$ and the others. Eq. (126) above is integrated in time until the steady-state equilibrium displacement $\mathbf{x}$ is reached. This solution procedure can be speeded up by constructing $\widetilde{\mathbf{M}}$ and $\widetilde{\mathbf{D}}$ as follows

$$\widetilde{\mathbf{M}} = a\widetilde{\mathbf{K}}, \qquad \widetilde{\mathbf{D}} = b\widetilde{\mathbf{K}} \tag{146}$$

and selecting the two scalars a and b so that the governing equations of motion (126) are *critically damped.* In that case, the equilibrium solution $\mathbf{x}$ is reached in a few time-steps.

Alternatively, $\widetilde{\mathbf{M}}$ and $\widetilde{\mathbf{D}}$ can be set to zero, and the new position of the dynamic fluid mesh can be computed via the solution of the static problem

$$\widetilde{\mathbf{K}}\mathbf{x} = \widetilde{\mathbf{K}}_c \mathbf{x}_{\Gamma_{F/S}} \tag{147}$$

This approach is often referred to as Batina's network of edge-springs [15]. However, it should be noted that attaching a lineal spring on the edge connecting two vertices of a tetrahedron prevents these two vertices from colliding during the mesh deformation, but does not prevent a vertex from interpenetrating the facet of a tetrahedron. To prevent such a detrimental interpenetration that is more likely to happen when the structure undergoes large motions, torsional springs must also be added at the mesh vertices, and their stiffnesses must be carefully calibrated.

In this work, the pseudo-structural system associated with the unstructured dynamic fluid mesh is constructed with lineal and torsional springs only ($\widetilde{\mathbf{M}} = \widetilde{\mathbf{D}} = 0$). Each fictitious lineal spring attached to an each edge connecting two fluid grid points S_i and S_j is attributed the following stiffness

$$k_{ij} = \frac{1}{||S_i S_j||_2} \tag{148}$$

The grid points located on the upstream and downstream boundaries are held fixed. At each time-step t^{n+1}, the new position of the interior grid points is determined from the solution of Eq. (147) via a two-step iterative procedure. First, the displacements of the interior grid points are predicted by extrapolating the previous displacements at time-steps t^n and t^{n-1} in the following manner

$$\mathbf{\Delta x} = 2\mathbf{\Delta}^{\mathbf{n}}\mathbf{x} - \mathbf{\Delta}^{\mathbf{n-1}}\mathbf{x} \tag{149}$$

where $\mathbf{\Delta}^{\mathbf{n}}\mathbf{x} = \mathbf{x}^{\mathbf{n+1}} - \mathbf{x}^{\mathbf{n}}$. Next, the predicted values are corrected with a few explicit Jacobi relaxations as follows

$$\mathbf{\Delta}^{\mathbf{n+1}}\mathbf{x}_{\mathbf{i}} = \frac{\sum_{\mathbf{j}} \mathbf{k}_{\mathbf{ij}} \mathbf{\Delta x}_{\mathbf{j}}}{\sum_{\mathbf{j}} \mathbf{k}_{\mathbf{ij}}} \tag{150}$$

Finally, the position of the fluid grid points at t^{n+1} is computed from

$$\mathbf{x}^{\mathbf{n+1}} = \mathbf{x}^{\mathbf{n}} + \mathbf{\Delta}^{\mathbf{n+1}}\mathbf{x} \tag{151}$$

8. A UNIFIED PARALLELIZATION STRATEGY [99]

8.1. The Mesh Partitioning and Message-Passing Paradigms

In addition to numerical efficiency and parallel scalability, portability should be a major concern, especially for production codes. With the proliferation of computer architectures, it is essential to adopt a programming model that does not require rewriting thousands of lines of code — or even worse, altering the architectural foundations of a code — every time a new parallel processor is acquired. Here, we are neither referring to differences between programming languages, nor to differences between the multitude of parallel extensions to a specific

programming language. We are more concerned about the impact of a given parallel hardware architecture on the finite element software design, and sometimes, on the solution algorithm itself. For example, a data parallel code written for the CM-2 or CM-5 machines would require major rehauling before it can be adapted to an iPSC computer. A parallel-do-loop based code can be easily ported across different true shared memory multiprocessors, but may require substantial modifications before it can run successfully on some distributed memory systems.

Based on our "hands on" experience with a dozen of different parallel processors, we can argue that the mesh partitioning and message-passing paradigms lead to the most portable software design for parallel computational mechanics. Essentially, the underlying mesh is assumed to be partitioned into several submeshes, each defining a subdomain. The same "old" serial code can be executed within every subdomain. The "gluing" or assembly of the subdomain results can be implemented in a separate software module. Clearly, this is an object-oriented approach that is best programmed in C++, but which can also be programmed in FORTRAN or any other language. This approach enforces data locality and therefore is suitable for all parallel hardware architectures. Note that in this context, "message-passing" refers to the assembly phase of the subdomain results. However, it does not imply that messages have to be explicitly exchanged between the subdomains. For example, message-passing can be implemented on a shared memory multiprocessor as a simple access to a shared buffer, or as a duplication of one buffer into another one. Moreover, the message-passing programming model produces software modules that are easy to maintain, because except for the gluing procedures, the subdomain code can be made identical to that of a workstation version.

In many cases, expensive parallel processors are affordable because some simulations can substitute for experimental studies that would take much longer and cost much more to carry out. However, there are also other cases where current parallel processors are simply too expensive, so that a network of relatively inexpensive workstations is preferred. Obviously, a message-passing based software can be quickly adapted to a cluster of workstations, for example, using a PVM-like communication tool.

Therefore, all of our flow solvers, structural analyzers, and mesh motion solvers are designed to work with mesh partitions, and are written in a message-passing style. Consequently, their performance is not only machine dependent, but sometimes also mesh partition dependent.

Research in mesh partitioning has focused so far on the automatic generation of subdomains with minimum interface points. In this section, we address this issue and emphasize other aspects of the partitioning problem including the fast generation of large-scale mesh decompositions on conventional workstations, the optimization of initial decompositions for specific kernels such as parallel frontal solvers and domain decomposition based iterative methods, More specifically, we overview a two-step partitioning paradigm for tailoring generated mesh partitions to specific applications, and a simple mesh contraction procedure for speeding up the optimization of initial mesh decompositions. We discuss what defines a good mesh partition for a given problem, and show that the methodology summarized here can produce better mesh partitions than the well celebrated multilevel Recursive Spectral Bisection algorithm, and yet be an order of magnitude faster. We illustrate the combined two-step partitioning and contraction methodology with examples from structural mechanics and fluid dynamics problems, and highlight its impact on the total solution time of realistic applications on current massively parallel processors. In particular, we show that the minimum

interface size criteria does not have a significant impact on a reasonably well parallelized application, and highlight other criterion which can have a significant impact.

8.2. The Greedy and RSB Algorithms: Two Extremes

It is often argued and demonstrated that if unstructured computational mechanics problems are to be efficiently solved on distributed-memory parallel computers, their data structures must be partitioned and distributed across the processors in a way that maximizes load balance and minimizes interprocessor communication [46,83]. However, research in mesh partitioning algorithms has mostly focused on the second issue — that is, on minimizing interprocessor communication costs only, and the number of interface points in a mesh partition, or the number of edge cuts in its corresponding graph, has rapidly become the main "acceptance test" for a proposed mesh decomposer.

While several mesh partitioning algorithms have already been presented and/or discussed in the literature [83–88], two radically different schemes have particularly attracted the attention of the user and developer communities: the Greedy algorithm [57,84,85], and the Recursive Spectral Bisection algorithm [83,88,93].

The Greedy (GR) mesh partitioning algorithm was first proposed in [89] and applied to the parallel solution of finite element structural equations on the iPSC-1 system. This mesh decomposition scheme is referred to as the Greedy algorithm because it essentially "bites" into the mesh in order to construct the subdomains. It exploits only the mesh connectivity information, which makes it the fastest partitioning algorithm we know about. In general, the GR algorithm tends to generate mesh partitions that are characterized by reasonable subdomain aspect ratios and a relatively small number of interface points. On a few occasions, this algorithm has been misrepresented [90], perhaps, because one statement is unfortunately missing in the Fortran code given in [84]. This statement is the one which forces every subdomain to start with an element attached to the previously computed interface. The GR algorithm enjoys a relatively large user community because of its high performance/price ratio. For example, it is capable of partitioning a three-dimensional unstructured mesh containing 439272 tetrahedra and 77279 vertices into 64 subdomains with 25906 interface points, in less than 15 seconds on a Crimson Silicon Graphics workstation. Recently, some interesting variants of the basic GR algorithm have been proposed [91,92].

The Recursive Spectral Bisection (RSB) graph partitioning algorithm was first proposed in [88]. This scheme is at the same time the least intuitive mesh decomposer, and the partitioning algorithm that has most attracted the attention of the parallel computing community. Unlike the Greedy algorithm which is simple and has no underlying theory, the RSB scheme is sophisticated and relies on a relatively well understood graph theory. More specifically, the RSB algorithm is derived from a graph bisection strategy based on the computation of the Fiedler vector — that is, the second eigenvector of the Laplacian matrix of the graph associated with the given problem [88]. Thanks to the multilevel strategy described in [93] for extracting the Fiedler vector, the computational requirements of this partitioning scheme are no longer overwhelming, even on a simple workstation. However, the multilevel RSB algorithm is still more expensive than most other partitioning schemes. For example, when applied to the decomposition into 64 subdomains of the same three-dimensional mesh containing 439272 tetrahedra and 77279 vertices, it consumes 707 seconds on a Crimson Silicon Graphics workstation and generates a mesh partition with 21139 interface points. This mesh partition has 18.40% less interface points than the decomposition generated by the Greedy algorithm, but costs 48.07 times more CPU time to

generate. Depending on the target parallel application, such an improvement at such a price may or may not be interesting. Recently, a parallel version of the RSB algorithm has been implemented on the CM-5 [94]. This version has certainly improved the computational feasibility of the RSB partitioner.

REMARK 3: Throughout this section, RSB designates the multilevel Recursive Spectral Bisection algorithm. In particular, all performance results reported for RSB applications correspond to the fast multilevel scheme described in [93], and release 2.1 of the code as integrated in TOP/DOMDEC [82].

Minimizing interprocessor communication costs in general, and the number of interface points in a mesh partition in particular, is a reasonable objective to "prioritize" when the target parallel application:

a) involves communication essentially between neighboring subdomains. This is typically the case for explicit time-integration (or pseudo time-integration) schemes, and some basic iterative solvers such as the conjugate gradient or Jacobi preconditioned conjugate gradient methods.

b) has a computational complexity that can be simply related to mesh entities such as, for example, nodes, and/or edges, and/or elements, and/or cells. In that case, load balancing can be reasonably well achieved by requiring that each subdomain contain the same number of such entities. In the event of heterogeneous meshes, a weighting factor can be attributed to each basic entity and the number of mesh entities per subdomain can be modified accordingly. Most importantly, load balancing in that case does not significantly interfere neither with the minimum edge cut aspect of a graph partitioner, nor with the practical implementation of the corresponding mesh decomposer.

c) uses a solution methodology whose performance is insensitive to the characteristics of a mesh partition such as, for example, the subdomain aspect ratio or the subdomain interconnectivity.

It is our experience that when conditions a), b) and c) are met, the GR and RSB algorithms generate excellent mesh partitions for parallel processing. Therefore, we have consistently used both algorithms for the subset of our parallel applications that can be described by the above a), b) and c) points.

However, many important parallel applications do not fit the profile implied by the a), b) and c) points. For example, frontal sparse solvers which are popular in finite elements and structural mechanics [63–66,95] require mesh partitions that do not significantly inflate the operation count of their sequential counterparts. This particular issue relates more to the subdomain local frontwidths than to the subdomain interface sizes. Moreover, controlling the load balance of these direct solvers is not in general as simple as distributing equally some basic mesh entities across the desired subdomains.

Optimal domain decomposition based iterative solvers are another class of parallel applications whose scalability is not governed by interprocessor communication costs only [57]. These solution algorithms are interesting on massively parallel processors when their number of iterations for convergence does not grow (or grows weakly) with the number of subdomains. Their effectiveness is determined by their convergence rate and not by their amount of communication. In particular, optimal non-overlapping domain decomposition based iterative solvers require mesh partitions that have as perfect subdomain aspect ratios (close to unity) as possible. Sometimes, fulfilling this requirement leads to mesh partitions with larger interfaces than otherwise possible.

This is well demonstrated below for the structural High Speed Civil Transport wing finite element model containing 3150 nodes. For this problem, the 32-subdomain mesh partition generated by the RSB algorithm has 707 interface nodes and an average subdomain aspect ratio $AR = 0.39$ (Fig. 22). The 32-subdomain mesh partition generated by the methodology described in this section and shown in Fig. 23 has 808 interface nodes, but an average subdomain aspect ratio $AR = 0.62$. When the FETI domain decomposition based iterative solver presented in Section 5 is applied to the structural wing problem, it converges in 47 iterations and 11.93 seconds when using the RSB mesh partition on a 32-processor iPSC-860. On the other hand, it converges in 30 iterations and 7.75 seconds when using the mesh partition with larger interface but improved subdomain aspect ratio [96]. Hence, one should question whether the minimum interface size is not after all an overemphasized mesh partitioning criterion.

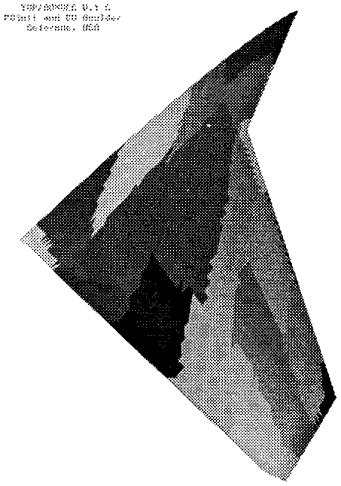


Fig. 22. 32-subdomain mesh partition for an HSCT wing structural model (RSB)

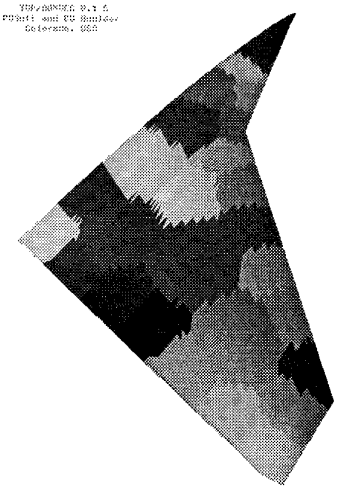


Fig. 23. 32-subdomain mesh partition (optimized subdomain aspect ratio)

8.3. Nomenclature

Throughout this section, the following nomenclature is used:

E	set of edges of the dual graph of the mesh
P	partitioning vector: $P_i = k$ means that the mesh entity i belongs to subdomain k.
C	cost function to be optimized
LBF	load balance factor
L	computational load of a given application
N_p	number of processors
N_s	number of subdomains
N_e	number of elements in the mesh
N_n	number of nodes in the mesh
N_d	number of degrees of freedom in the model
N^k	number of some specific mesh entities in subdomain k (including its interface boundary)
N_{me}	number of macro elements in a contracted mesh
N_I	number of interface points of a mesh partition
$N^{best,k}$	number of mesh entities in subdomain k that yields an optimal load balance

d	spatial dimension of the problem
x_{ijk}	i-th coordinate of the j-th node in subdomain k
$\bar{x}_{ik}$	i-th coordinate of the center of gravity of subdomain k

8.4. Two-Step Partitioning and Retrofitting

For all our computational mechanics parallel applications, we have adopted the two-step mesh partitioning paradigm that was first introduced in [97,98], then refined in [99], and which consists in

Step 1) generating an initial mesh decomposition via either a suboptimal but fast partitioning algorithm, or an algorithm that is known to produce mesh partitions that are reasonably well suited for the target parallel application.

Step 2) formulating the application specific requirements as a cost function C, and optimizing it by readjusting the initial subdomain interfaces. This step can also be described as a *retrofitting* procedure.

The Greedy algorithm is very fast because its complexity grows as $O\ (N_e \times N_s)$. Moreover, it produces mesh partitions that are reasonably well-suited for most parallel computational methods. Hence, the GR algorithm is ideally suited for generating an initial decomposition in Step 1.

In Step 2, a cost function representing the decomposition requirements of the target parallel application must first be formulated. A sample list of cost functions to optimize is given below:

Interface size: $C_1 = \frac{1}{2}|\{(i,j) \in E/P_i \neq P_j\}|$. Here, the size of the interface is defined as the number of edges in E whose vertices belong to two different subdomains. This cost function may not govern all parallel applications but is certainly helpful in all cases.

Load imbalance: $C_2 = \sum_{k=1}^{k=N_s} [N^k - N^{best,k}]^2$. When a parallel application has a computational complexity that can be simply related to mesh entities such as, for example, nodes, and/or edges, and/or elements, and/or cells, the computational load L can be easily estimated and $N^{best,k}$ can be set prior to the decomposition to $N^{best,k} = L/N_p$. Otherwise, $N^{best,k}$ is unknown *a priori*. It can have a different value in every subdomain k, and is adaptively evaluated by the optimization algorithm.

Subdomain aspect ratio: $C_3 = \sum_{k=1}^{k=N_s} (\sum_{i=1}^{i=d} \sum_{j=1}^{j=N_n} (x_{ijk} - \bar{x}_{ik})^2)$. This cost function has been shown to play a pivotal role in the convergence rate of optimal domain decomposition based preconditioned conjugate gradient methods [96].

In practice, the performance of a parallel application is often governed by several distinct factors. Therefore, one should consider in general the following weighted cost function:

$$C = \sum \alpha_i C_i \qquad (152)$$

where C_i is a cost function representing one specific issue — for example, C_i could be anyone of the cost functions listed above — and α_i is the weight attributed to that issue. In that case, optimizing C corresponds to finding the best possible "compromise" mesh partition. Unfortunately, we do not have yet an automatic mechanism for selecting the weight coefficients α_i. For this task, we rely on our understanding of the focus parallel application, and experience with the target parallel processor.

After a cost function is formulated, the decomposition is optimized by readjusting only the subdomain interfaces. More specifically, only the mesh entities that are attached to the interface are examined for possible exchange between the subdomains. Therefore, the computational complexity of the optimization process

is proportional to the interface size and not to the number of elements in the mesh. In collaboration with the Université Catholique de Louvain, we have implemented three different schemes for optimizing a given cost function.

Simulated Annealing (SA) [100]. This algorithm uses a monotonically decreasing "temperature" as control variable for the outer iterations. For a fixed temperature, a number of mesh entities are proposed for transfer to a neighboring subdomain — in the sequel, we refer to this step as a "move". The acceptance of a move is dictated by a probabilistic decision which depends on the difference in cost between making the move or ignoring it. The optimization process ends when the temperature is sufficiently low and no further moves are accepted. In the inner loop, moves are chosen randomly. The probability of acceptance of bad moves decreases with temperature.

Tabu Search (TS) [101]. This scheme stores in a *tabu* list a specified number of recently accepted moves. In the inner loop, several moves outside the tabu list are proposed, and the move with the highest positive or negative gain is accepted. In the outer loop, the last accepted move replaces the oldest move in the tabu list. Therefore, if this algorithm escapes a local minimum, it cannot use the same path in the solution space to reach this minimum again.

Stochastic Evolution (SE) [102]. The main difference between this algorithm and Simulated Annealing is in the evolution of the control variable and the selection of the moves. At each outer iteration, all interface elements are proposed for a move in a predefined order. The temperature decreases rapidly, thereby decreasing the probability of accepting bad moves, until the solution reaches a local minimum of the cost function. At this point, the temperature is reset to its initial value. In general, this algorithm behaves as a series of fast SA processes where the solution jumps from one local minimum to another.

A quality/speed trade-off can be applied to each of the above optimization schemes by "tuning" a few control parameters [98].

There is at least one compelling reason for having more than one optimization algorithm at hand. In some cases, the initial mesh partition generated in Step 1 can get entrapped in a local minimum at the first step of an optimization scheme, in which case Step 2 does not improve the original decomposition. One can hope that switching to another optimization algorithm pulls the solution out of that local minimum. Everytime we have encountered this problem for SA, we were able to resolve it by switching to TS.

In order to illustrate the two-step methodology described above and highlight its potential, we consider the partitioning of two three-dimensional fluid dynamics unstructured meshes into 64 and 128 subdomains. The first mesh, FALC, is designed for the simulation of external Euler flows around a Falcon aircraft. It contains 439272 tetrahedra and 77279 vertices. The second mesh, MUFF, is designed for the simulation of internal viscous flows inside a car muffler (Fig. 24). It contains 237963 tetrahedra and 43592 vertices. Here, we assume that the objective is to generate mesh partitions with equal number of tetrahedra and minimum number of interface points. Hence, the load balance factor can be written in this case as follows

$$LBF = \frac{\text{average}_k \; N_e^k}{\max_k \; N_e^k} \qquad (153)$$

More complex objectives are discussed in Section 8.6.

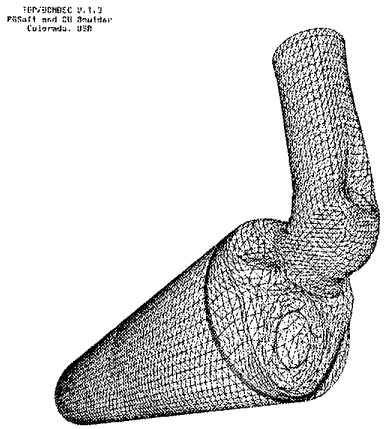


Fig. 24. Three-dimensional discretization of the flow domain inside a car muffler

First, the GR and multilevel RSB algorithms are used to partition the FALC mesh into 64 subdomains, and the MUFF mesh into 128 subdomains. Following the recommendation given in [93], the computational size of the lowest level is set to 300 for the RSB scheme. Next, the two-step methodology is applied to generate similar mesh partitions. The GR algorithm is selected for Step 1, and the SA optimization scheme for Step 2. For both meshes, the cost function is defined as $C = 0.5 \times C_1 + 0.5 \times C_2$, and the parameters N^k and $N^{best,k}$ are set to $N^k = N_e^k$ and $N^{best,k} = N_e/N_p$. The characteristics of the resulting mesh partitions are summarized in Tables 1 and 2. All computations are performed on a Crimson Silicon Graphics workstation.

The results reported in Table 1 show that for the FALC mesh, RSB outperforms GR for the imposed objective. The mesh partition produced by RSB has 18.40% less interface points than that delivered by GR, but costs 48.07 times more CPU time to generate. On the other hand, the two-step partitioning methodology with GR as an initial decomposer outperforms RSB for the same objective. The mesh partition generated by GR and optimized by SA has 9.83% less interface points than that delivered by RSB and costs 1.97 times *less* CPU time to produce. For the MUFF mesh, the results reported in Table 2 show that GR outperforms RSB for the imposed objective. More specifically, GR produces a mesh partition with 2.53% less interface points than RSB does and 115.40 times faster. The two-step partitioning methodology with GR as an initial decomposer outperforms both RSB and GR, is significantly cheaper than RSB, but is also significantly more expensive than GR.

Table 1
Partitioning of the FALC mesh: $N_e = 439272$ — $N_s = 64$
$C = 0.5 \times C_1 + 0.5 \times C_2$
SGI/Crimson

Scheme	Optimizer	N_I	LBF	CPU Step 1	CPU Step 2	CPU Total
RSB	None	21 139	0.999	707.10 s.	0.00 s.	707.10 s.
GR	None	25 906	0.999	14.71 s.	0.00 s.	14.71 s.
GR	SA	19 060	0.999	14.71 s.	342.76 s.	357.47 s.

Table 2
Partitioning of the MUFF mesh: $N_e = 237963$ — $N_s = 128$
$C = 0.5 \times C_1 + 0.5 \times C_2$
SGI/Crimson

Scheme	Optimizer	N_I	LBF	CPU Step 1	CPU Step 2	CPU Total
RSB	None	17 810	0.999	791.69 s.	0.00 s.	791.69 s.
GR	None	17 358	0.999	6.86 s.	0.00 s.	6.86 s.
GR	SA	14 934	0.996	6.86 s.	551.72 s.	558.58 s.

For the above two examples, we have used GR as an initial decomposer in order to keep the total partitioning costs as low as possible. However, if preprocessing costs are not an issue, RSB can also be used in Step 1. For the MUFF mesh, the two-step method with RSB as an initial decomposer generates a mesh partition with 14252 interface points and consumes 1212.49 s. CPU (791.69 s. (Step 1) + 420.80 s. (Step 2)). This particular example shows that when an initial mesh partition is slightly better than another one, its optimized version is not necessarily better than the optimized version of that other one.

Also note that for the above problems, all algorithms including the two-step methodology deliver mesh partitions with perfect load balance factors.

We are particularly interested in fast and good partitioning algorithms because we would like to be able to inspect — possibly interactively — a few mesh decompositions before selecting one for a target parallel application. The examples reported above highlight the potential of the two-step methodology for generating excellent mesh partitions. However, the optimization step is not as fast as we would like it to be. Next, we present a contraction procedure for speeding up the optimization process in Step 2.

8.5. An Efficient Contraction Procedure

The idea of contracting a mesh before partitioning it is not new. Apparently, it was first proposed in [93] for reducing the costs of the RSB partitioning scheme, and in [103] for storage optimization purposes. The contraction approach presented in [93] is based on the concept of maximal independent sets. The contraction approach proposed herein is based on the Greedy algorithm and our experience with this heuristic. Our main objective is to speed up the optimization process in Step 2 of the partitioning methodology. Our main strategy goes as follows.

First, the mesh is recursively coarsened using an $O(N_e)$ Greedy-based contraction procedure until its size reaches a user specified value, say $N_{me} = 5000$ macro-elements. An initial decomposition is performed on the coarse mesh using preferably a fast mesh partitioning algorithm. This decomposition is followed by a few smoothing iterations using one of the three optimization schemes introduced in Section 3. Next, the obtained coarse partition is mapped onto the original and finer mesh, and another optimization is performed on the fine level. When more than one level of contractions is needed to reach the specified number of macro-elements N_{me}, coarse-to-fine mapping and optimization are performed at every intermediate level.

More specifically, the contraction step is implemented as follows. Given a starting element, a fixed-size cluster is constructed by agglomerating neighboring elements in a recursive manner. This cluster defines a macro-element in the contracted mesh. At the beginning, the starting element is selected among the peripheral elements. Later, it is selected among those elements which neighbor existing clusters. The contraction ends when all elements are attributed to a cluster. In practice, we have found that 5 elements is a good choice for the size of a cluster. However, fewer or more elements can sometimes define a cluster for connexity purposes.

The impact of the contraction procedure described above on the two-step partitioning methodology is highlighted in Tables 3 and 4 for the FALC and MUFF meshes, respectively.

Table 3
Partitioning of the FALC mesh: $N_e = 439272$ — $N_s = 64$
$C = 0.5 \times C_1 + 0.5 \times C_2$
Effects of the contraction procedure
SGI/Crimson

Scheme	Optimizer	N_I	LBF	CPU Contr.	CPU Step 1	CPU Step 2	CPU Total
RSB	None	21 139	0.999	0.00 s.	707.10 s.	0.00 s.	707.10 s.
GR	None	25 906	0.999	0.00 s.	14.71 s.	0.00 s.	14.71 s.
GR	SA	19 060	0.999	0.00 s.	14.71 s.	342.76 s.	357.47 s.
Contr. + GR	Contr.+ SA	16 160	0.999	6.65 s.	0.08 s.	38.36 s.	45.09 s.

Table 4
Partitioning of the MUFF mesh: $N_e = 237963$ — $N_s = 128$
$C = 0.5 \times C_1 + 0.5 \times C_2$
Effects of the contraction procedure
SGI/Crimson

Scheme	Optimizer	N_I	LBF	CPU Contr.	CPU Step 1	CPU Step 2	CPU Total
RSB	None	17 810	0.999	0.00 s.	791.69 s.	0.00 s.	791.69 s.
GR	None	17 358	0.999	0.00 s.	6.86 s.	0.00 s.	6.86 s.
GR	SA	14 934	0.996	0.00 s.	6.86 s.	551.72 s.	558.58 s.
Contr. + GR	Contr. + SA	12 792	0.999	3.02 s.	0.12 s.	143.70 s.	146.84 s.

For the FALC mesh and 64 subdomains, the contraction procedure is shown to reduce the cost of Step 2 by a full order of magnitude. In that case, the two-step partitioning method with GR as an initial decomposer produces a mesh partition with 23.55% less interface nodes than that generated by RSB, and is 15.68 times faster than RSB.

For the MUFF mesh and 128 subdomains, the two-step partitioning method with graph contraction and GR as an initial decomposer produces a mesh partition that has 28.17% less interface nodes than the RSB partition, and is 5.39 times faster than RSB.

The performance results reported in Tables 3 and 4 also show that the proposed contraction procedure not only speeds up the two-step partitioning method, but also results in better mesh decompositions. Indeed, the contracted mesh represents the structure of the original grid, and the optimization of its decomposition tends to improve the global structure of the desired mesh partition by moving several elements simultaneously. When the mesh is not contracted, the global structure of the mesh partition remains identical to that of the initial decomposition because the probability of transferring large amounts of elements between the initial subdomains is usually low.

As mentioned earlier, a quality/speed trade-off can be applied to each of the three optimization schemes by "tuning" some of their control parameters. An example of such trade-off is illustrated in Table 5 for the FALC mesh and various number of subdomains. From the results reported in this table, it follows that, for the cost function $C = 0.5 \times C_1 + 0.5 \times C_2$, the two-step partitioning method with contraction can generate even better mesh partitions when the optimization algorithm is allowed to run longer in Step 2. Note that even in that case, the two-step method is still significantly cheaper than the multilevel RSB algorithm. For example, it can generate a 64-subdomain partition for the FALC mesh with 14 613 interface points only in 161.04 seconds, whereas the RSB scheme consumes 707.10 seconds to generate a 64-subdomain mesh partition with 21 139 interface points (see Table 3). This amounts to an almost twice better mesh partition at a quarter of the price. The performance results summarized in Table 5 also show that the complexity of the two-step partitioning method with contraction is sublinear with the number of subdomains.

Table 5
Partitioning of the FALC mesh: $N_e = 439272$ — $N_s = 64$
$C = 0.5 \times C_1 + 0.5 \times C_2$
Two-step partitioning method with contraction
Initial decomposer = GR - optimization scheme = SA
Computational complexity of Step 2 - quality/speed trade-offs
SGI/Crimson

N_s	CPU Step 1	N_I (QUALITY)	N_I (SPEED)	CPU Step 2 (QUALITY)	CPU Step 2 (SPEED)
2	0.02 s.	1834	2371	12.80 s.	6.19 s.
4	0.03 s.	4291	4853	32.90 s.	12.72 s.
8	0.03 s.	5997	6903	51.19 s.	17.40 s.
16	0.04 s.	8240	9413	94.09 s.	27.01 s.
32	0.06 s.	11414	12682	124.40 s.	29.93 s.
64	0.08 s.	14613	16160	161.04 s.	38.44 s.
128	0.16 s.	18740	20520	207.30 s.	47.78 s.

In the remainder of this section, we use ex-

clusively GR for all initial decompositions. We show that in all cases, the two-step partitioning methodology with the contraction procedure described herein is a cheaper and better alternative to the multilevel RSB algorithm.

8.6. Highlights

The two-step decomposition methodology and the contraction procedure described in this section are available in the TOP/DOMDEC [82] interactive software package for mesh partitioning and parallel processing. Here, we illustrate these two methodologies with examples from computational structural mechanics and fluid dynamics, and highlight their impact on the parallel solution time of these problems on an iPSC-860 multiprocessor and a Convex Meta Series system.

Mesh partitioning algorithms are often evaluated and/or benchmarked by simply assessing and/or comparing the characteristics of the mesh partitions they generate (interface size, theoretical load balance factors, ...). Such an approach is at best incomplete. The ultimate goal of a mesh partitioning algorithm is to reduce, if possible, the parallel CPU time of the target parallel application. Hence, mesh partitioning algorithms should be benchmarked by comparing their impact on problem solving. Here, we consider three classes of applications: the solution of a set of semi-discrete differential equations via an explicit time-integration scheme, the solution of a system of sparse linear equations via a domain decomposition based iterative algorithm, and the solution of a system of sparse linear equations via a frontal method.

8.6.1. Explicit Time-Marching

First, we consider a stress wave propagation problem in a line-pinched plate with a circular hole. The plate is discretized using 47680 4-node shell elements and 48235 nodes (Fig. 25). The corresponding number of equations is 233939. For this problem, the semi-discrete finite element equations of dynamic equilibrium are time-integrated using the explicit central difference scheme. Four different mesh partitions are generated for parallel computations on a 64-processor iPSC-860 system. The characteristics of these decompositions are summarized in Table 6 where $N_I^{min,k}$, $N_I^{average,k}$, $N_I^{max,k}$, and N_I denote respectively the minimum, average, and maximum number of interface nodes per subdomain, and the total number of interface nodes in the mesh partition. Given that the parallel performance of the central difference scheme — and most explicit time-integration algorithms — is governed by load balancing and communication costs, the cost function $C = 0.5 \times C_1 + 0.5 \times C_2$ and the parameters $N^k = N_n^k$ and $N^{best,k} = \sum_{k=1}^{k=N_s} N^k / N_p$ are used for this application.

For the above problem and 64 subdomains, the interface size of the mesh partition generated by the RSB scheme is 14 % smaller than that of the mesh partition produced by the GR algorithm. On the other hand, the two-step partitioning methodology without contraction reduces the interface size of the GR decomposition by 29%, and with contraction it reduces it by 36% (see Table 6).

Table 6
Partitioning of the plate mesh: $N_e = 47680$ - $N_n = 48235$ - $N_d = 233939$ - $\boxed{N_s = 64}$
$C = 0.5 \times C_1 + 0.5 \times C_2$

Scheme	Optimizer	Contraction	$N_I^{min,k}$	$N_I^{average,k}$	$N_I^{max,k}$	N_I	$\frac{N_I^{GR}}{N_I}$
RSB	None	No	74	108	159	3433	1.14
GR	None	No	56	124	286	3912	1.00
GR	SA	No	53	101	149	3039	1.29
GR	SA	Yes	52	98	144	2876	1.36

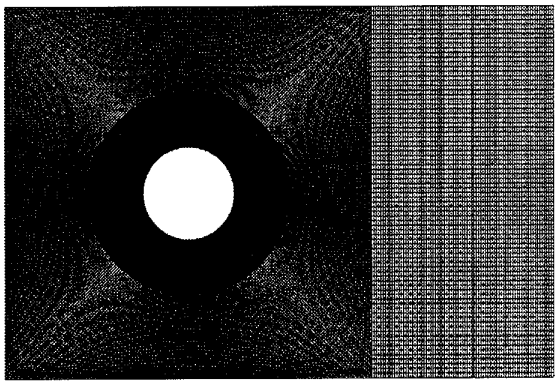


Fig. 25. Finite element discretization of a plate with a circular hole

The performance results on a 64-processor iPSC-860 system of the transient analysis of the plate problem are reported in Table 7 for 2000 integration time-steps, and the four generated 64-subdomain mesh partitions. Throughout the remainder of this section, T_{comm} and T_{sol} denote respectively the communication time, and the total solution time for the target parallel application.

Table 7
Explicit central difference
Plate mesh: $N_e = 47680$ - $N_n = 48235$ - $N_d = 233939$ - $\boxed{N_s = 64}$
Solution time for 2000 time-steps on an $\boxed{\text{iPSC-860/64}}$

Scheme	Optimizer	Contraction	T_{comm}	T_{sol}	$\frac{N_I^{GR}}{N_I}$	$\frac{T_{comm}{}^{GR}}{T_{comm}}$	$\frac{T_{sol}{}^{GR}}{T_{sol}}$
RSB	None	No	115.28 s.	706.91 s.	1.14	1.20	1.03
GR	None	No	138.34 s.	728.12 s.	1.00	1.00	1.00
GR	SA	No	101.72 s.	693.45 s.	1.29	1.36	1.05
GR	SA	Yes	98.81 s.	693.41 s.	1.36	1.40	1.05

Clearly, the results reported in Table 7 show that the communication costs of the explicit central difference time-integration algorithm are directly related to the number of interface nodes (for this problem, it turns out that all generated mesh partitions have a similar average number of neighboring subdomains). However, these results also indicate that for this class of parallel applications, there is little to

gain by searching for the "perfect" mesh partition with the least number of interface nodes. For example, the two-step mesh decomposition algorithm with contraction reduces the interface size and communication costs of the GR partition by factors equal to 1.36 and 1.40, respectively, but improves the total CPU time corresponding to the GR partition by 5% only. Hence, it would seem that the applications for which one has legitimate reasons to prioritize the minimization of the interface nodes are the least sensitive to the size of the subdomain interfaces. Of course, such a statement assumes that the given parallel processor is reasonably fast in communication, and that the size of the problem to be solved justifies the chosen number of subdomains or processors.

One could argue that the above conclusions hold only for two-dimensional problems where the subdomain interfaces are topologically one-dimensional, but not necessarily for three-dimensional problems where the subdomain interfaces are topologically two-dimensional, and the average number of neighbors for a given subdomain is higher. For this reason, we investigate next the parallel performance of the explicit central difference algorithm applied to the evaluation of the linear transient response of a three-dimensional engine nozzle subjected to a sudden pressure burst. The nozzle is discretized into 12800 8-node brick elements, 15579 nodes and 46701 active degrees of freedom (Fig. 26). Four different mesh partitions are generated for parallel computations on a 64-processor iPSC-860 system. The characteristics of these decompositions are summarized in Table 8. As for the previous example, the cost function is set to $C = 0.5 \times C_1 + 0.5 \times C_2$, N^k is set to $N^k = N_n^k$, and $N^{best,k} = \sum_{k=1}^{k=N_s} N^k / N_p$ is adopted.

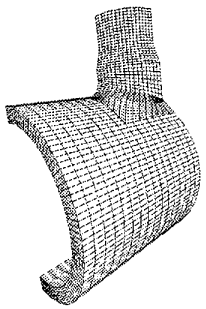


Fig. 26. Three-dimensional finite element discretization of a nozzle

Table 8
Partitioning of the nozzle mesh: $N_e = 12800$ - $N_n = 15579$ - $N_d = 46701$ - $\boxed{N_s = 64}$
$C = 0.5 \times C_1 + 0.5 \times C_2$

Scheme	Optimizer	Contraction	$N_I^{min,k}$	$N_I^{average,k}$	$N_I^{max,k}$	N_I	$\frac{N_I^{GR}}{N_I}$
RSB	None	No	116	185	272	5401	1.12
GR	None	No	129	212	316	6068	1.00
GR	TS	No	134	191	259	5494	1.10
GR	TS	Yes	120	177	220	5079	1.19

For the nozzle problem and 64 subdomains, the mesh partition generated by the RSB scheme has 1.12 times less interface nodes than that produced by the GR algorithm. The two-step

partitioning methodology with contraction reduces the interface size of the GR decomposition by a factor equal to 1.19. Note that reducing the total number of interface nodes also seems to improve the interface load balancing factor $ILBF = N_I^{average,k}/N_I^{max,k}$. For example, $ILBF = 0.67$ only for the 64-subdomain mesh partition generated by the GR algorithm, while $ILBF = 0.80$ for that produced by the two-step decomposition methodology.

Table 9 reports the CPU time on a 64-processor iPSC-860 system of a 2000 time-step transient analysis of the engine nozzle using the various 64-subdomain mesh partitions.

Table 9
Explicit central difference
Nozzle mesh: $N_e = 12800$ - $N_n = 15579$ - $N_d = 46701$ - $\boxed{N_s = 64}$
Solution time for 2000 time-steps on an $\boxed{\text{iPSC-860/64}}$

Scheme	Optimizer	Contraction	T_{comm}	T_{sol}	$\frac{N_I^{GR}}{N_I}$	$\frac{T_{comm}{}^{GR}}{T_{comm}}$	$\frac{T_{sol}{}^{GR}}{T_{sol}}$
RSB	None	No	136.40 s.	338.00 s.	1.12	1.09	1.07
GR	None	No	149.08 s.	362.00 s.	1.00	1.00	1.00
GR	TS	No	139.46 s.	346.00 s.	1.10	1.07	1.05
GR	TS	Yes	129.63 s.	335.00 s.	1.19	1.15	1.08

Before commenting on the performance results summarized in Table 9, it is worthwhile noting that the iPSC-860 computer used for this application has only 8 Mbytes of memory per processor. The smallest number of processors on this machine that is a power of two and can meet the storage requirements of this three-dimensional dynamics application is 64. From Table 8, it follows that 32% to 39% of the nodes of a 64-subdomain mesh partition of the nozzle mesh are interface nodes. Hence, the hardware configuration of this iPSC-860 and the memory requirements of the nozzle dynamics problem are such that the computational and communication requirements of this target parallel application are not particularly well balanced. This is reflected in the performance results summarized in Table 9 which show that 38% to 41% of the total CPU time is spent in communication. To some extent, this situation is typical of three-dimensional finite element problems solved on small memory massively parallel processors. In Table 9, it is shown that RSB improves the communication time over GR by a factor equal to 1.09, and the two-step partitioning methodology with contraction improves the communication time over GR by a factor equal to 1.15. These factors are consistent with those describing the reduction of the number of interface nodes. However, for the enhanced mesh partitions, the total CPU time is only 7% to 8% better than that corresponding to the GR partition, which is also consistent with the distribution of the total simulation time between computation and communication.

In summary, minimizing the number of interface nodes of a mesh partition does improve the total CPU time of this class of parallel applications, but not by impressive factors. Stated differently, unless communication costs are overwhelming — in which case parallel processing is not necessarily attractive — any reasonable mesh partition is suitable for this type

of parallel applications. This fact is rarely recognized in the parallel processing literature.

8.6.2. Domain Decomposition Based Iterative Solvers

Here, we focus on the solution of the system of equations arising from the finite element static analysis of an elastic bearing under a distributed surface load. The finite element model of this three-dimensional structure contains 9600 8-node brick elements and 33075 degrees of freedom (Fig. 27). The optimal domain decomposition based FETI iterative solver (see Section 5) is used for parallel computations on a 64-processor iPSC-860 system. Three 64-subdomain mesh partitions are generated using RSB, GR, and the two-step mesh partitioning method with $C = 0.5 \times C_2 + 0.5 \times C_3$, $N^k = N_e^k$ and $N^{best,k} = N_e/N_p$. The characteristics of these mesh partitions and the corresponding performance results of the FETI solver are reported in Table 10 where AR and N_{itr} denote respectively the average subdomain aspect ratio and the number of FETI iterations for convergence.

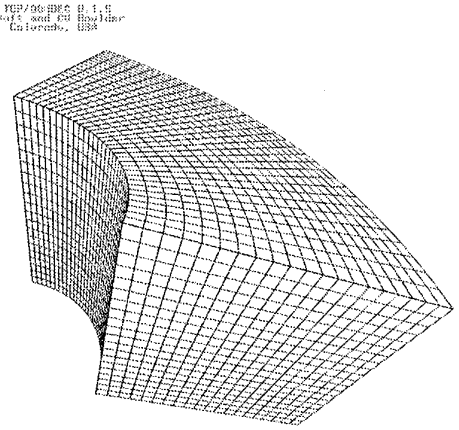


Fig. 27. Finite element discretization of an elastic bearing

For this application, it is clear that the size of the interface problem does not control neither the communication time nor the total CPU time of the domain decomposition solver. In particular, note that for the above problem there is no correlation between N_I and the communication costs per FETI iteration. This is essentially because the communication costs of this application are dominated by those associated with global dot products and some other full matrix linear algebra on a coarse grid problem. On the other hand, the results reported in Table 10 clearly demonstrate the importance of the subdomain aspect ratio for this class of applications. The two-step mesh decomposition method with contraction improves the subdomain aspect ratio of the mesh partitions generated by GR and RSB by a factor equal to 1.7, which reduces the number of FETI iterations by a factor equal to 1.5, and the total solution time by a factor equal to 1.4.

Table 10
Optimal FETI solver
Bearing mesh: $N_e = 9600$ - $N_d = 33075$ - $\boxed{N_s = 64}$
Effect of the subdomain aspect ratio

Scheme	Optimizer	Contraction	N_I	AR	T_{comm}/N_{itr}	N_{itr}	T_{sol}
RSB	None	No	5 426	0.50	0.37 s.	45	36.09 s.
GR	None	No	5 032	0.52	0.37 s.	43	35.17 s.
GR	SA	Yes	4 430	0.84	0.40 s.	30	25.77 s.

8.6.3. Parallel Frontal Solvers

The problem of computing the steady-state flow of an incompressible Oldroyd fluid in a two-cam mixing apparatus arises in polymer processing applications. This problem is governed by a set of mixed elliptic/hyperbolic nonlinear partial differential equations. Here, we consider such a problem and the flow domain depicted in Fig. 28. Its finite element discretization contains 1217 elements only, but generates 26082 equations. At each Newton iteration, these equations are solved with the frontal direct solver described in [99].

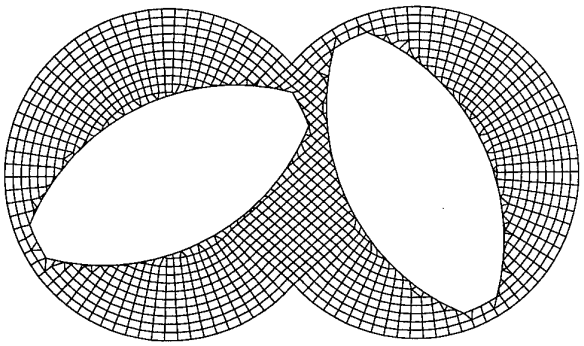


Fig. 28. Discretization of the flow domain in a two-cam mixing apparatus

Among all parallel applications, the frontal direct solver is perhaps the most challenging one for mesh partitioning. Ideally, this algorithm requires a mesh partition where: (a) each subdomain frontwidth is smaller or equal to the frontwidth of the global problem, (b) the computational load is perfectly balanced, and (c) the subdomain interfaces have a minimum and equal number of nodes. Criterion (a) should be emphasized, because trading computational efficiency for parallelism is not always a winning strategy. Enforcing criterion (b) is a seriously difficult task, because the computational load per subdomain cannot be derived a priori from the computational complexity of the global problem. Criterion (c) attempts at minimizing the communication and storage requirements associated with the elimination of the interface unknowns.

Here, four 8-subdomain mesh partitions are generated for parallel computations on an 8-processor Convex Meta Series system, using GR, RSB, and the two-step mesh partitioning method with both GR and RSB as initial decomposers. For this application, the cost function to be optimized is set to $0.5 \times C_1 + 0.5 \times C_2$. However, note that in this case $N^{best,k}$ cannot be determined a priori. Let FR^k and $FR^{max,k}$ denote respectively the variable subdomain frontwidth and its maximum value. During the optimization (or retrofitting) process, $N^{best,k}$ is computed so that $N^{best,k} \times FR^{max,k^2}$ is the same in all subdomains.

The characteristics of all four mesh partitions and the corresponding performances of the parallel frontal solver are reported in Table 11 where $EFRLBF = \text{average}_k(N_e^k \times FR^{k^2})/\max_k(N_e^k \times FR^{k^2})$ is the estimated computational load balance factor, $T^k_{internal}$ is the

parallel CPU time associated with the elimination of the subdomain *internal* unknowns, and T_{sol} is the total parallel solution time. An internal renumbering scheme [32] is used in every subdomain for minimizing fill-in.

Table 11
Parallel frontal solver
Polymer flow mesh: $N_e = 1217$ - $N_d = 26082$ - $\boxed{N_s = 8}$
Effects of the subdomain frontwidth and load balancing

Scheme	Opt.	N_I	$FR^{average,k}$	$EFRLBF$	$\frac{T^{average,k}_{internal}}{T^{max,k}_{internal}}$	T_{sol}
RSB	None	97	282.87	0.53	0.60	135.96 s.
RSB	SA	88	269.38	0.83	0.85	80.01 s.
GR	None	139	393.75	0.47	0.63	230.53 s.
GR	SA	85	252.50	0.67	0.66	102.48 s.

The ability of $EFRLBF$ to predict the computational load balance of the parallel frontal solver is well illustrated in Table 11. Also, the suitability of the selected cost function and the effectiveness of the optimization algorithm are well demonstrated. For example, the run-time load balance factor for the RSB mesh partition is equal to 0.53, while that of the optimized RSB partition is equal to 0.83. The net result of the optimization process is a speedup factor in the solution time equal to 1.69. For the GR partition, the net result of the retrofitting step is a speedup factor equal to 2.25. Note also that for the above problem, the mesh partition that leads to the fastest parallel solution of the linearized equations is neither the one with the minimum number of interface nodes, nor that with the minimum subdomain frontwidth, but the mesh partition with the best predicted load balance factor — and it also turns out to be the mesh partition with the best run-time load balance factor.

10. APPLICATIONS AND PERFORMANCE RESULTS

Here, we demonstrate the aeroelastic computational methodology described in the previous sections with the numerical investigation of the instability of flat panels with infinite aspect ratio in supersonic airstreams, and the solution of three-dimensional wing response problems in the transonic regime. All flow computations are performed using the Euler equations and the explicit solver.

10.1. Two-Dimensional Aeroelastic Supersonic Computations

10.1.1. Problem Definition

The flat panel with infinite aspect ratio considered here (Fig. 29) is assumed to have a length $L = 0.5m$, a uniform thickness $h = 1.35 \times 10^{-3}\ m$, a Young modulus $E = 7.728 \times 10^{10}\ N/m^2$, a Poisson ratio $\mu = 0.33$, a density $\rho = 2710\ Kg/m^3$, and to be clamped at both ends. Its rectangular cross section is discretized into 1111×3 plane strain 4-node elements. This fine discretization — which generates 3333 elements with perfect aspect ratios and 4448 nodes — is not needed for accuracy; we have designed this structural mesh only because we are also interested in assessing some computational and I/O performance issues.

The two-dimensional flow domain above the panel is discretized into 32568 triangles and

16512 vertices. A slip condition is imposed at the fluid/structure boundary. Because the fluid and structural meshes are not compatible at their interface (Fig. 30), the Matcher software [35] is used to generate in a single preprocessing step the data structures required for transferring the pressure load to the structure, and the structural deformations at the upper surface of the panel to the fluid.

We consider several supersonic flows at different Mach numbers and discuss the performances of the partitioned analysis procedures ALG0, ALG1, ALG2, and ALG3. Whenever subcycling is used, the I^1 interpolation scheme is used to prescribe the motion of the fluid grid points on $\Gamma_{F/S}$.

10.1.2. Computational Platform

All computations are performed on an iPSC-860 parallel processor using double precision arithmetic. The fluid and structure solvers are implemented as separate programs that communicate via the intercube communication procedures described in [104].

10.1.3. Assessment of the Partitioned Procedures

In order to illustrate the relative merits of the partitioned procedures ALG0, ALG1, ALG2 and ALG3, we consider first two different series of transient aeroelastic simulations at Mach number $M_\infty = 1.90$ that highlight

- the relative accuracy of these algorithms for a fixed subcycling factor $n_{S/F}$.
- the relative speed of these algorithms for a fixed level of accuracy, on both sequential and parallel computational platforms.

In all cases, 64 processors are allocated to the fluid system, and 2 processors are assigned to the structural solver. Initially, a steady-state flow is computed above the panel at $M_\infty = 1.90$ (Fig. 31), speed at which the panel described above is not supposed to flutter. Then, the aeroelastic response of the coupled system is triggered by a displacement perturbation of the panel along its first mode (Fig. 32).

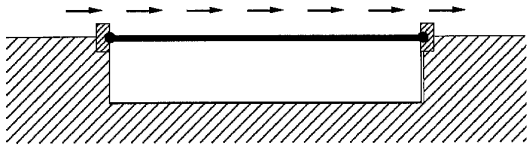


Fig. 29. A flat panel with infinite aspect ratio

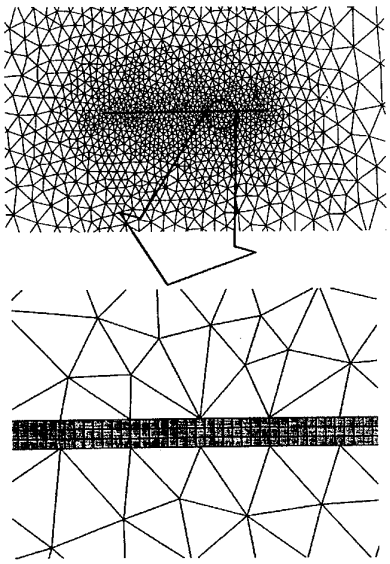


Fig. 30. Mesh incompatibility

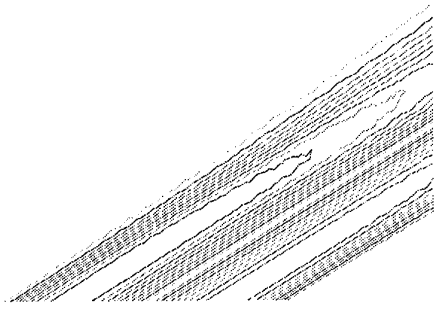


Fig. 31. Pressure isovalues for the steady-state flow solution ($M_\infty = 1.90$)

Fig. 32. Initial perturbation of the panel displacement field

First, the subcycling factor is fixed to $n_{S/F} = 30$, and the lift coefficient is computed using the time-step $\Delta t = 3.9 \times 10^{-6}$ corresponding to the stability limit of the explicit flow solver in the absence of coupling with the structure. The obtained results are depicted in Fig. 33 for the first 4102 time-steps. For $n_{S/F} = 30$, ALG1 and ALG3 exhibit essentially the same accuracy. In the long run, their amplitude and phase errors are less important than those of ALG2. Clearly, this highlights the superiority of ALG3 which, despites its inter-field parallelism and unlike ALG2, is capable of delivering the same accuracy as the sequential algorithm ALG1.

Next, the relative speed of the focus partitioned solution procedures is assessed by comparing their CPU performance for a certain level of accuracy dictated by ALG0. It turns out that in order to meet the accuracy requirements of ALG0, ALG1 and ALG3 can use a subcycling factor as large as $n_{S/F} = 10$, but ALG2 can subcycle only up to $n_{S/F} = 5$ (Fig. 34).

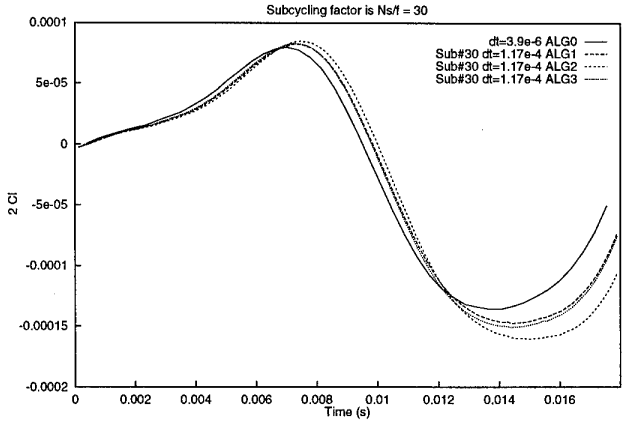


Fig. 33. Lift coefficient history for $n_{S/F} = 30$

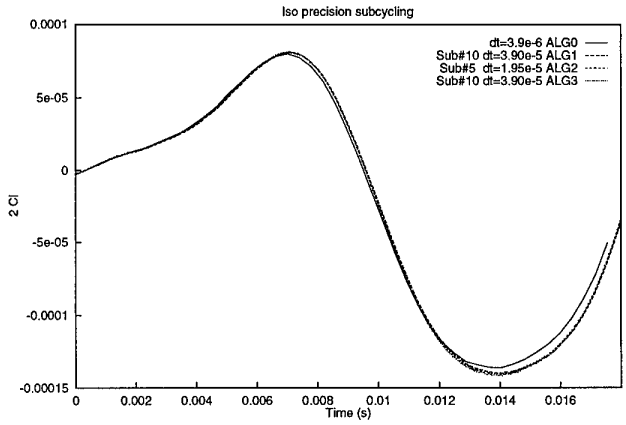


Fig. 34. Lift coefficient history for a fixed level of accuracy

The performance results measured on the iPSC-860 are reported in Table 12 where ICC denotes the intercube communication time. Note that ICC is measured in the fluid kernel and includes idle time when the flow and structural communications do not overlap.

Table 12. Performance results on the iPSC-860

Fluid: 64 processors	Structure: 2 processors			
Elapsed time for 4102 fluid time-steps				
Algorithm	Fluid	Structure	Fluid-Wait+ICC	Total CPU
ALG0	2617.23 s.	1267.93 s.	1283.10 s.	3900.33 s.
ALG1 ($n_{S/F} = 10$)	2625.11 s.	126.67 s.	127.90 s.	2753.01 s.
ALG2 ($n_{S/F} = 5$)	2643.57 s.	253.34 s.	1.67 s.	2645.24 s.
ALG3 ($n_{S/F} = 10$)	2603.56 s.	253.23 s.	1.37 s.	2604.93 s.

From the results reported in Table 12, the following observations can be made

- the fluid computations dominate the simulation time. This is partly because the structural model is simple in this case, and a linear elastic behavior is assumed for the panel.
- considering that the iPSC-860 has 128 processors and that only clusters of 2^n processors can be defined on this machine, allocating 64 processors to the fluid and 2 processors to the structure achieves the minimum possible inter-field load imbalance for this coupled problem.
- the effect of subcycling on intercube communication costs is clearly demonstrated. Because the flow solution time is dominating, the effect of subcycling on the total CPU time is less important for ALG2 and ALG3 which feature inter-field parallelism in addition to intra-field multiprocessing, than for ALG1 which features intra-field parallelism only (note that ALG1 with $n_{S/F} = 1$ is identical to ALG0).
- ALG2 and ALG3 allow a perfect overlap of inter-field communications, which reduces intercube communication and idle time to less than 0.3% of the amount corresponding to ALG0.
- The superiority of ALG3 over ALG2 is not clearly demonstrated for this problem because of the simplicity of the structural model and the subsequent load imbalance between the fluid and structure computations.

10.1.4. Panel Flutter

The classical and analytical solution of the instability problem of flat panels with infinite aspect ratio in supersonic airstreams assumes a shallow shell theory for the structure and a linearized formulation for the flow problem (piston theory). Within this analytical approach, the dynamics of the focus coupled fluid/structure system are governed by a fourth-order partial differential equation [2, page 419], and the flutter condition is obtained by analyzing the roots of the corresponding characteristic equation. For the panel described the beginning of this section, the classical linear theory predicts flutter at the critical Mach number $M_\infty^{cr} = 1.98$. The objective of this Section is to validate the aeroelastic simulation capability presented in this paper by reproducing the theoretical critical Mach number for the given panel. Note that in order to compare the analytical and finite element approaches, the coefficients of the shallow shell equation described in [2, page 419]

must be computed to represent the same equation as that corresponding to the finite element model used in this paper.

Four different runs at $M_\infty = 2.0$, $M_\infty = 2.05$, $M_\infty = 2.095$, and $M_\infty = 2.13$ are performed using ALG3. For each run, a steady-state flow is first computed. Then, a displacement perturbation of the panel along its first mode (Fig. 32) is imposed, and the aeroelastic response of the coupled system is computed. The predicted time histories of the lift coefficient are depicted in Fig. 35 for all four cases.

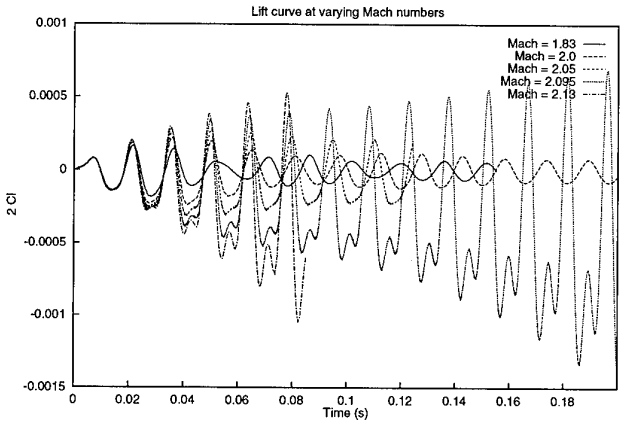


Fig. 35. Flutter analysis

From the results reported in Fig. 35, it follows that the flutter speed predicted by our formulation verifies $2.05 \leq M_\infty^{cr} \leq 2.095$. Hence, this flutter speed is 4.5 % higher than that predicted by the piston theory. This is a rather good agreement, given that the piston theory and the computational approach presented herein do not share exactly the same approximations.

Finally, we report in Fig. 36 the history of the accumulated external energy at $M_\infty = 2.095$ for both the fluid and structural systems. At this speed, the panel is clearly shown to extract energy from the fluid, and therefore to flutter. Note that Fig. 36 also highlights the quality of the matching performed by Matcher: the amount of external energy extracted by the structure is shown to be equal to that lost by the fluid, as it should be.

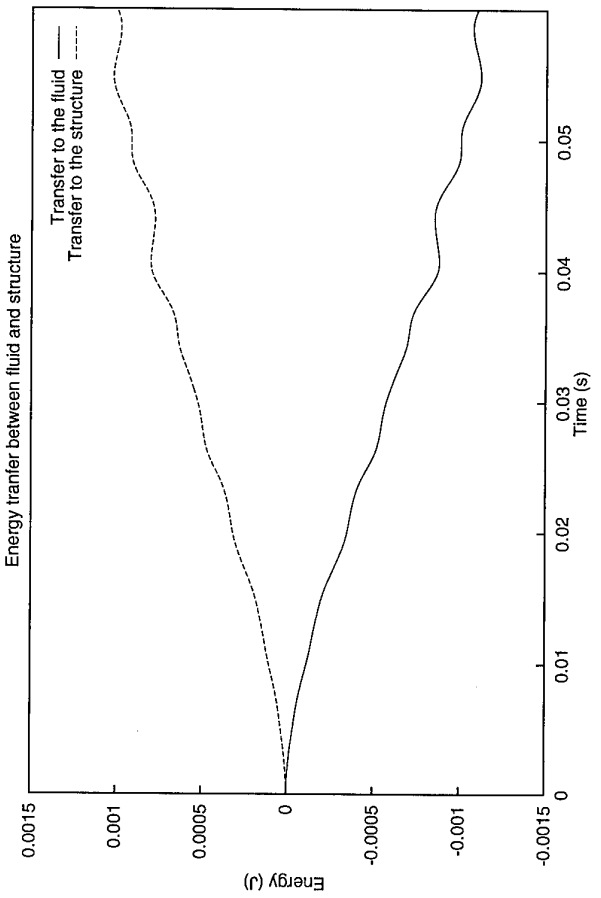


Fig. 36. Accumulated external energy ($M_\infty = 2.095$)

10.2. Three-Dimensional Aeroelastic Transonic Computations

10.2.1. Problem Definition

Next, we consider transient aeroelastic reponse problems associated with a simple structural model of the ONERA M6 wing.

The wing is represented by an equivalent plate model discretized in 1071 triangular plate elements, 582 nodes, and 6426 degrees of freedom (Fig. 37). Four meshes $M1 - -M4$ are designed for the discretization of the three-dimensional flow domain around the wing. The characteristics of theses meshes are given in Table 13 where N_{ver}, N_{tet}, N_{fac}, and N_{var} denote

respectively the number of vertices, tetrahedra, facets (edges), and fluid variables. A partial view of the discretization of the flow domain is shown in Fig. 38.

Table 13
Characteristics of meshes $M1 - -M4$

Mesh	N_{ver}	N_{tet}	N_{fac}	N_{var}
$M1$	15460	80424	99891	77300
$M2$	31513	161830	201479	157565
$M3$	63917	337604	415266	319585
$M4$	115351	643392	774774	576755

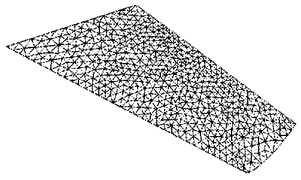


Fig. 37. Finite element plate model of the wing

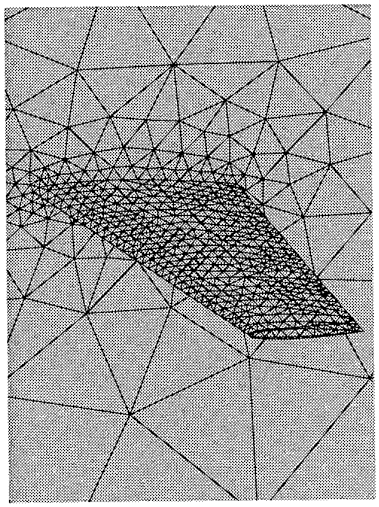


Fig. 38. Partial view of the fluid mesh $M1$ on the skin of the ONERA M6 wing

The sizes of the fluid meshes $M1 - -M4$ have been tailored for parallel computations on respectively 16 ($M1$), 32 ($M2$), 64 ($M3$), and 128 processors ($M4$) of a Paragon XP/S and a Cray T3D systems. In particular, the sizes of these meshes are such that the processors of a Paragon XP/S machine with 32 Mbytes per node would not swap when solving the corresponding flow problems.

Here again, the fluid and structural meshes are not compatible at their interface. Matcher [35] is used to generate in a single preprocessing step the data structures required for transferring the pressure load to the structure, and the structural deformations to the fluid.

10.2.2. Computational Platforms

All computations are performed on an iPSC - 860, and/or a Paragon XP/S, and/or a Cray T3D, and/or an IBM SP2 computers using double precision arithmetic. Message passing is carried out via NX on the Paragon XP/S multiprocessor, PVM T3D on the Cray T3D system, and MPI on the IBM SP2 parallel processor. The fluid and structure solvers are implemented as separate programs that communicate via the intercube communication procedures described in [104].

10.2.3. Parallel Performance of the FlowSolver

The performance of the parallel flow solver is assessed with the computation of the steady state of a flow around the given wing at a Mach number $M_\infty = 0.84$ and an angle of attack $\beta = 3.06$ degrees (Fig. 39) . The CFL number is set to 0.9. The four meshes $M1 - -M4$ are decomposed in respectively 16, 32, 64, and 128 *overlapping* subdomains using TOP/DOMDEC [82]. The motivations for employing overlapping subdomains and the impact of this computational strategy on parallel performance are discussed in [49]. The CPU timings in seconds are reported in Tables 14-16 for the first 100 iterations on a Paragon XP/S machine (128 processors), a Cray T3D system (128 processors), and an IBM SP2 computer (32 processors), respectively. In these tables, N_p, N_{var},

T^{loc}_{comm}, T^{glo}_{comm}, T_{comp}, T_{tot} and $mflops$ denote respectively the number of processors, the number of variables (unknowns) to be solved, the time elapsed in short range interprocessor communication between neighboring subdomains, the time elapsed in long range global interprocessor communication, the computational time, the total simulation time, and the computational speed in millions of floating point operations per second. Typically, short range communication is needed for assembling various subdomain results such as fluxes at the subdomain interfaces, and long range interprocessor communication is required for reduction operations such as those occurring in the the evaluation of the stability time-steps and the norms of the nonlinear residuals. Because message-passing is also used for synchronization, the reported communication timings include any idle-time due to load imbalance. We also note that we use the same fluid code for steady state and aeroelastic computations. Hence, even though we are benchmarking in Tables 14–16 a steady state computation with a local time stepping strategy, we are still timing the kernel that evaluates the global time-step in order to reflect its impact on the unsteady computations that we perform in aeroelastic simulations such as those that are discussed next. The $mflop$ rates reported in Tables 14-16 are computed in a strict manner: they exclude all the redundant computations associated with the overlapping subdomain regions.

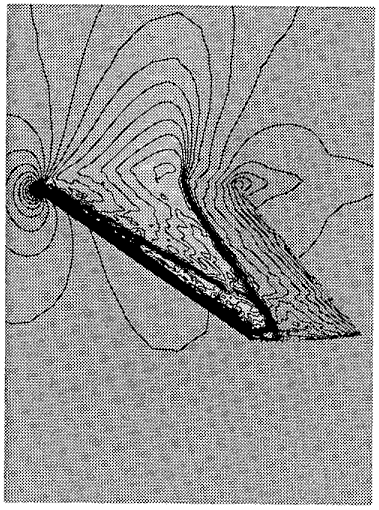


Fig. 39. Steady-state Mach lines (ONERA M6 wing — mesh $M4$)

Table 14
Performance of the parallel flow solver on the Paragon XP/S system (16—128 processors)

100 iterations — CFL = 0.9

Mesh	N_p	T^{loc}_{comm}	T^{glo}_{comm}	T_{comp}	T_{tot}	$mflops$
$M1$	16	2.0 s.	40.0 s.	96.0 s.	138.0 s.	84
$M2$	32	4.5 s.	57.0 s.	98.5 s.	160.0 s.	145
$M3$	64	7.0 s.	90.0 s.	103.0 s.	200.0 s.	240
$M4$	128	6.0 s.	105.0 s.	114.0 s.	225.0 s.	401

Table 15
Performance of the parallel flow solver on the Cray T3D system (16—128 processors)

100 iterations — CFL = 0.9

Mesh	N_p	T^{loc}_{comm}	T^{glo}_{comm}	T_{comp}	T_{tot}	$mflops$
$M1$	16	1.6 s.	2.1 s.	87.3 s.	91.0 s.	127
$M2$	32	2.5 s.	4.1 s.	101.4 s.	108.0 s.	215
$M3$	64	3.5 s.	7.2 s.	100.3 s.	111.0 s.	433
$M4$	128	3.0 s.	7.2 s.	85.3 s.	95.5 s.	945

Table 16
Performance of the parallel flow solver on the IBM SP2 system (4—32 processors)

100 iterations — CFL = 0.9

Mesh	N_p	T^{loc}_{comm}	T^{glo}_{comm}	T_{comp}	T_{tot}	$mflops$
$M1$	4	0.8 s.	0.4 s.	70.8 s.	72.0 s.	160
$M2$	8	1.1 s.	0.6 s.	73.3 s.	75.0 s.	308
$M3$	16	1.4 s.	0.7 s.	78.9 s.	81.0 s.	594
$M4$	32	2.0 s.	1.0 s.	79.0 s.	82.0 s.	1102

The reader can easily verify that the number of processors assigned to each mesh is such that N_{var}/N_p is almost constant. This means that larger numbers of processors are attributed to larger meshes in order to keep each local problem within a processor at an almost constant size. For such a benchmarking strategy, parallel scalability of the flow solver on a target parallel processor implies that the total solution CPU time should be constant for all meshes and their corresponding number of processors. This is clearly not the case for the Paragon XP/S system. On this machine, short range communication is shown to be inexpensive, but long range communication costs are reported to be important. This is certainly due to the latency of the Paragon XP/S parallel processor which is an order of magnitude slower than that of the Cray T3D system. Another possible source of global communication time increase is the load imbalance between the processors since message passing is also used for synchronization. However, this does not seem to be significant on the T3D and SP2 parallel processors.

On the other hand, parallel scalability is well demonstrated for the Cray T3D and IBM SP2 systems. The results reported in Tables 15 and 16 show that all computations using meshes $M1--M4$ and the corresponding number of processors consume almost the same total amount of CPU time. For 128 processors, the Cray T3D system is shown to be more than twice faster than the Paragon XP/S machine. The difference appears to be strictly in long range communication as the computational time is reported to be almost the same on both machines. However, most impressive is the fact that an IBM SP2 with 32 processors only is shown to be three times faster than a 128 - processor Paragon XP/S, and faster than a Cray T3D with 128 processors.

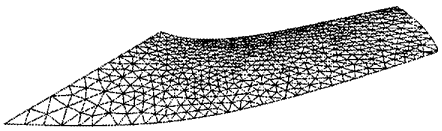


Fig. 40. Initial perturbation of the structure

10.2.4. Performance of the Partitioned Analysis Procedures

As in the two-dimensional application, we consider first two different series of transient aeroelastic simulations at Mach number $M_\infty = 0.84$ that highlight

- the relative accuracy of these coupled solution algorithms for a fixed subcycling factor $n_{S/F}$.
- the relative speed of these coupled solution algorithms for a fixed level of accuracy.

In all cases, mesh $M2$ is used for the flow computations, 32 processors of an iPSC-860 system are allocated to the fluid solver, and 4 processors of the same machine are assigned to the structural code. Initially, a steady-state flow is computed around the wing at $M_\infty = 0.84$, Mach number at which the wing described above is not supposed to flutter. Then, the aeroelastic response of the coupled system is triggered by a displacement perturbation of the wing along its first mode (Fig. 40).

First, the subcycling factor is fixed to $n_{S/F} = 10$ then to $n_{S/F} = 30$, and the lift is computed using a time-step corresponding to the stability limit of the explicit flow solver in the absence of coupling with the structure. The obtained results are depicted in Fig. 41 and Fig. 42 for the first half cycle.

Fig. 41. Lift history for the first half cycle ($n_{S/F} = 10$)

Fig. 42. Lift history for the first half cycle ($n_{S/F} = 30$)

The superiority of the parallel fluid-subcycled ALG3 solution procedure is clearly demonstrated in Fig. 41 and Fig. 42. For $n_{S/F} = 10$, ALG3 is shown to be the closest to ALG0, which is supposed to be the most accurate since it is sequential and non-subcycled. ALG1 and ALG2 have comparable accuracies. However, both of these algorithms exhibit a significantly more important phase error than ALG3, especially for $n_{S/F} = 30$.

Next, the relative speed of the partitioned solution procedures is assessed by comparing their CPU time for a certain level of accuracy dictated by ALG0. For this problem, it turned out that in order to meet the accuracy requirements of ALG0, the solution algorithms ALG1 and ALG2 can subcycle only up to $n_{S/F} = 5$, while ALG3 can easily use a subcycling factor as large as $n_{S/F} = 10$. The performance results measured on an iPSC-860 system are reported in Table 17 for the first 50 coupled time-steps. In this table, ICWF and ICWS denote the intercode communication timings measured respectively in the fluid and structural kernels; these timings include idle and synchronization (wait) time when the fluid and structural communications do not completely overlap. For programming reasons, ICWS is monitored together with the evaluation of the pressure load.

Table 17. Performance results on the iPSC-860

	Fluid: 32 processors		Structure: 4 processors			
	Elapsed time for 50 fluid time-steps					
Alg.	Fluid Solver	Fluid Motion	Struc. Solver	ICWS	ICWF	Total CPU
ALG0	177.4 s.	71.2 s.	33.4 s.	219.0 s.	384.1 s.	632.7 s.
ALG1	180.0 s.	71.2 s.	16.9 s.	216.9 s.	89.3 s.	340.5 s.
ALG2	184.8 s.	71.2 s.	16.6 s.	114.0 s.	0.4 s.	256.4 s.
ALG3	176.1 s.	71.2 s.	10.4 s.	112.3 s.	0.4 s.	247.7 s.

From the results reported in Table 17, the following observations can be made:

- the fluid computations dominate the simulation time. This is partly because the structural model is again simple in this case, and a linear elastic behavior is assumed. However, by allocating 32 processors to the fluid kernel and 4 processors to the structure code, a reasonable load balance is shown to be achieved for ALG0.
- during the first 50 fluid time-steps, the CPU time corresponding to the structural solver does not decrease linearly with the subcycling factor $n_{S/F}$ because of the initial costs of the FETI reorthogonalization procedure designed for the efficient iterative solution of implicit systems with repeated right hand sides (see Section 5).
- the effect of subcycling on intercube communication costs is clearly demonstrated. The impact of this effect on the total CPU time is less important for ALG2 and ALG3 which feature inter-field parallelism in addition to intra-field multiprocessing, than

for ALG1 which features intra-field parallelism only (note that ALG1 with $n_{S/F} = 1$ is identical to ALG0), because the flow solution time is dominating.

- ALG2 and ALG3 allow a certain amount of overlap between inter-field communications, which reduces intercube communication and idle time on the fluid side to less than 0.001% of the amount corresponding to ALG0.

Most importantly, the performance results reported in Table 17 demonstrate that subcycling and inter-field parallelism are desirable for aeroelastic simulations even when the flow computations dominate the structural ones, because these features can significantly reduce the total simulation time by minimizing the amount of inter-field communications and overlapping them. For the simple problem described herein, the parallel fluid-subcycled ALG2 and ALG3 algorithms are more than twice faster than the conventional staggered procedure ALG0.

10. CONCLUSIONS

In this paper, we have highlighted some key elements of the solution of large-scale three-dimensional nonlinear aeroelastic problems on high performance computational platforms. We have described a three-field arbitrary Lagrangian-Eulerian (ALE) finite volume/element formulation for the coupled fluid/structure problem, presented geometric conservation laws for three-dimensional flow problems with moving boundaries and unstructured and deformable meshes, and discussed the solution of the corresponding coupled semi-discrete equations with partitioned analysis procedures. In particular, we have presented a family of mixed explicit/implicit staggered solution algorithms, and discussed them with particular reference to accuracy, stability, subcycling, and parallel processing. We have described a general framework for the solution of coupled aeroelastic problems on heterogeneous and/or parallel computational platforms, and illustrated it with two- and three-dimensional applications on an iPSC-860, a Paragon XP/S, and a Cray T3D massively parallel systems. We have shown that even when the flow computations dominate the total CPU time of a coupled aeroelastic simulation, subcycling and inter-field parallelism are desirable as they can significantly speedup the total solution time.

ACKNOWLEDGEMENTS

The author acknowledges partial support by the National Science Foundation under Grant ASC-9217394, partial support by RNR NAS at NASA Ames Research Center under Grant NAG 2-827, and partial support by CMB at the NASA Langley Research Center under Grant NAG-1536427. He wishes to thank Po-Shu Chen, Stéphane Lanteri, Michel Lesoinne, Serge Piperno, and Paul Stern for their help in this research effort.

REFERENCES

[1] H. Tijdeman and R. Seebass, Transonic flow past oscillating airfoils, Ann. Rev. Fluid Mech. 12 (1980) 181–222.

[2] R. L. Bisplinghoff and H. Ashley, Principles of aeroelasticity, Dover Publications, Inc., 1962.

[3] Y. C. Fung, An introduction to the theory of aeroelasticity, Dover Publications, Inc., 1969.

[4] R. Dat and J. L. Meurzec, Sur les calculs de flottement par la method dite du "balayage" en frequence reduite, La Recherche Aerospatiale 133, Nov. Dec. 1969.

[5] C. Bon, M. Géradin, and J. P. Grisval, Private communication, 1993.

[6] E. Albano and W. P. Rodden, A doublet-lattice method for calculating lift distribution on oscillating surfaces in subsonic flow, AIAA J. 7 (1969) 279–285.

[7] W. P. Jones and K. Appa, Unsteady supersonic aerodynamic theory for interfering surfaces by the method of potential gradient, NASA CR 2898, October 1977.

[8] P. C. Chen and D. D. Liu, A harmonic gradient method for unsteady supersonic flow calculations, AIAA 830887 CP, May 1983.

[9] E. H. Dowell and M. Llgamov, Studies in Nonlinear Aeroelasticity, Springer-Verlag, 1988.

[10] J. Donea, An arbitrary Lagrangian-Eulerian finite element method for transient fluid-structure interactions, Comput. Meths. Appl. Mech. Engrg. 33 (1982) 689–723.

[11] T. J. R. Hughes, W. K. Liu and T. K. Zimmermann, Lagrangian-Eulerian finite element formulation for incompressible viscous flows, U.S.-Japan Seminar on Interdisciplinary Finite Element Analysis, Cornell Univ., Ithaca, NY, Aug. 7–11 (1978).

[12] T. Belytschko and J. M. Kennedy, Computer models for subassembly simulation, Nucl. Eng. Design 49 (1978) 17–38.

[13] O. A. Kandil and H. A. Chuang, Unsteady vortex-dominated flows around maneuvering wings over a wide range of mach numbers, AIAA Paper No. 88-0317, AIAA 26th Aerospace Sciences Meeting, Reno, Nevada, January 11–14, 1988.

[14] C. Farhat and T. Y. Lin, Transient aeroelastic computations using multiple moving frames of reference, AIAA Paper No. 90-3053, AIAA 8th Applied Aerodynamics Conference, Portland, Oregon, August 20–22, 1990.

[15] J. T. Batina, Unsteady Euler airfoil solutions using unstructured dynamic meshes, AIAA Paper No. 89-0115, AIAA 27th Aerospace Sciences Meeting, Reno, Nevada, January 9–12, 1989.

[16] G. P. Guruswamy, Time-accurate unsteady aerodynamic and aeroelastic calculations of wings using Euler equations, AIAA Paper No. 88-2281, AIAA 29th Structures, Structural Dynamics and Materials Conference, Williamsburg, Virginia, April, 18–20, 1988.

[17] T. Tezduyar, M. Behr and J. Liou, A new strategy for finite element computations involving moving boundaries and interfaces-The deforming spatial domain/space-time procedure: I. The concept and the preliminary numerical tests, Comput. Meths. Appl. Mech. Engrg. 94 (1992) 339–351.

[18] M. Lesoinne and C. Farhat, Stability analysis of dynamic meshes for transient aeroelastic computations, AIAA Paper No. 93-3325, 11th AIAA Computational Fluid Dynamics Conference, Orlando, Florida, July 6–9, 1993.

[19] M. Lesoinne, Mathematical analysis of three-field numerical methods for aeroelastic problems, Ph. D. Thesis, The University of Colorado at Boulder, December 1994.

[20] K. C. Park and C. A. Felippa, Partitioned analysis of coupled systems, in: Computational Methods for Transient Analysis, T. Belytschko and T. J. R. Hughes, Eds., North-Holland Pub. Co. (1983) 157–219.

[21] T. Belytschko, P. Smolenski and W. K. Liu, Stability of multi-time step partitioned integrators for first-order finite element systems, Comput. Meths. Appl. Mech. Engrg. 49 (1985) 281–297.

[22] C. Farhat, K. C. Park and Y. D. Pelerin, An unconditionally stable staggered algorithm for transient finite element analysis of coupled thermoelastic problems, Comput. Meths. Appl. Mech. Engrg. 85 (1991) 349–365.

[23] S. Piperno, C. Farhat and B. Larrouturou, Partitioned procedures for the transient solution of coupled aeroelastic problems, Comput. Meths. Appl. Mech. Engrg., (in press).

[24] C. J. Borland and D. P. Rizzetta, Nonlinear transonic flutter analysis, AIAA Journal (1982) 1606–1615.

[25] V. Shankar and H. Ide, Aeroelastic computations of flexible configurations, Comput. & Struc. 30 (1988) 15–28.

[26] R. D. Rausch, J. T. Batina and T. Y. Yang, Euler flutter analysis of airfoils using unstructured dynamic meshes, AIAA Paper No. 89-13834, 30th Structures, Structural Dynamics and Materials Conference, Mobile, Alabama, April 3–5, 1989.

[27] M. Blair, M. H. Williams and T. A. Weisshaar, Time domain simulations of a flexible wing in subsonic compressible flow, AIAA Paper No. 90-1153, AIAA 8th Applied Aerodynamics Conference, Portland, Oregon, August 20–22, 1990.

[28] T. W. Strganac and D. T. Mook, Numerical model of unsteady subsonic aeroelastic behavior, AIAA Journal 28 (1990) 903–909.

[29] E. Pramono and S. K. Weeratunga, Aeroelastic computations for wings through direct coupling on distributed-memory MIMD parallel computers, AIAA Paper No. 94-0095, 32nd Aerospace Sciences Meeting & Exhibit, Reno, January 10-13, 1994.

[30] V. Venkatakrishnan, A perspective of unstructured grid flow solvers, ICASE Report No. 95-3, NASA Langley Research Center, February 1995.

[31] J. W. Edwards and J. B. Malone, Current status of computational methods for transonic unsteady aerodynamics and aeroelastic applications, Comput. Sys. Engrg. 3 (1992) 545–569.

[32] C. Farhat, J. Mandel and F. X. Roux, Optimal convergence properties of the FETI domain decomposition method, Comput. Meths. Appl. Mech. Engrg. 115 (1994) 367–388.

[33] C. Farhat, P. S. Chen and J. Mandel, A scalable Lagrange multiplier based domain decomposition method for implicit time-dependent problems, Internat. J. Numer. Meths. Engrg., (in press).

[34] C. Farhat, L. Crivelli and F. X. Roux, Extending substructure based iterative solvers to multiple load and repeated analyses, Comput. Meths. Appl. Mech. Engrg. 117 (1994) 195–209.

[35] N. Maman and C. Farhat, Matching fluid and structure meshes for aeroelastic computations: a parallel approach, Comput. & Struc. 54 (1995) 779–785.

[36] M. Lesoinne and C. Farhat, Geometric conservation laws for aeroelastic computations using unstructured dynamic meshes, AIAA Paper 95-1709, 12th AIAA Computational Fluid Dynamics Conference, San Diego, California, June 19-22, 1995.

[37] P. D. Thomas and C. K. Lombard, Geometric conservation law and its application to flow computations on moving grids, AIAA J. 17 (1979) 1030–1037.

[38] B. NKonga, H. Guillard, Godunov type method on non-structured meshes for three-dimensional moving boundary problems, Comput. Meths. Appl. Mech. Engrg. 113 (1994) 183–204.

[39] H. Zhang, M. Reggio, J.Y. Trépanier and R. Camarero, Discrete form of the GCL for moving meshes and its implementation in CFD schemes, Computers in Fluids 22 (1993) 9–23.

[40] J. Steger and R. F. Warming, Flux vector splitting for the inviscid gas dynamic with applications to finite-difference methods, Journ. of Comp. Phys. 40 (1981) 263–293.

[41] W.K. Anderson, J.L. Thomas and C.L. Rumsey, Extension and application of flux-vector splitting to unsteady calculations on dynamic meshes, AIAA Paper No 87-1152-CP, 1987.

[42] L.P. Franca, S.L. Frey and T.J.R. Hughes, Stabilized finite element methods: I. Application to the advective-diffusive model, Comput. Meths. Appl. Mech. Engrg. 95 (1992) 253–276.

[43] H. Schlichting, Boundary layer theory, Fourth edition, McGraw-Hill, New York, 1960.

[44] C. Farhat, M. Lesoinne and N. Maman, Mixed explicit/implicit time integration of coupled aeroelastic problems: three-field formulation, geometric conservation and distributed solution, Internat. J. Numer. Meths. Fluids, (in press).

[45] C. Farhat, M. Lesoinne, P. S. Chen and S. Lantéri, Parallel heterogeneous algorithms for the solution of three-dimensional transient coupled aeroelastic problems, AIAA Paper 95-1290, AIAA 36th Structural Dynamics Meeting, New Orleans, Louisiana, April 10-13, 1995.

[46] C. Farhat, S. Lantéri and L. Fezoui, Mixed finite volume/finite element massively parallel computations: Euler flows, unstructured grids, and upwind approximations, in Unstructured Scientific Computation on Scalable Multiprocessors, ed. by P. Mehrotra, J. Saltz, and R. Voigt, MIT Press (1992) 253–283.

[47] C. Farhat, L. Fezoui, and S. Lantéri, Two-dimensional viscous flow computations on the Connection Machine: unstructured meshes, upwind schemes, and massively parallel computations, Comput. Meths. Appl. Mech. Engrg. 102 (1993) 61–88.

[48] S. Lantéri and C. Farhat, Viscous flow computations on MPP systems: implementational issues and performance results for unstructured grids, in Parallel Processing for Scientific Computing, ed. by R. F. Sincovec *et. al.*, SIAM (1993) 65–70.

[49] C. Farhat and S. Lantéri, Simulation of compressible viscous flows on a variety of MPPs: computational algorithms for unstructured dynamic meshes and performance results, Comput. Meths. Appl. Mech. Engrg. 119 (1994) 35–60.

[50] P. L. Roe, Approximate riemann solvers, parameters vectors and difference schemes, J. Comp. Phys. 43 (1981) 357–371.

[51] B. Van Leer, Towards the ultimate conservative difference scheme V: a second-order sequel to Goudonov's method, J. Comp. Phys. 32 (1979) 361–370.

[52] A. Dervieux, Steady Euler simulations using unstructured meshes, Von Kármán Institute Lecture Series, 1985.

[53] L. Fezoui and B. Stoufflet, A class of implicit upwind schemes for Euler simulations with unstructured meshes, J. Comp. Phys. 84 (1989) 174–206.

[54] X.-C. Cai, C. Farhat, and M. Sarkis, Schwarz preconditioners and implicit methods for compressible flows problems on unstructured meshes, Eighth International Conference on Domain Decomposition Methods for Partial Differential Equations, AMS, 1995, (in press).

[55] X.-C. Cai and O.B. Widlund, Multiplicative Schwarz algorithms for nonsymmetric and indefinite elliptic problems, SIAM J. Numer. Anal. 30 (1993) 936–952.

[56] X.-C. Cai, W. D. Gropp, and D. E. Keyes, A comparison of some domain decomposition and ILU preconditioned iterative methods for nonsymmetric elliptic problems, Numer. Lin. Alg. Applics 1 (1994) 477–504.

[57] C. Farhat and F. X. Roux, Implicit parallel processing in structural mechanics, Computational Mechanics Advances 2 (1994) 1–124.

[58] C. Farhat and E. Wilson, A new finite element concurrent computer program architecture, Internat. J. Numer. Meths. Engrg. 24 (1987) 1771–1792.

[59] P. E. Bjordstad and O. B. Widlund, Iterative methods for solving elliptic problems on regions partitioned into substructures, SIAM J. of Num. Anal. 23 (1986) 1097–1120.

[60] C. Farhat, A Lagrange multiplier based divide and conquer finite element algorithm, J. Comput. Sys. Engrg. 2 (1991) 149–156.

[61] C. Farhat and F. X. Roux, A method of finite element tearing and interconnecting and

its parallel solution algorithm, Internat. J. Numer. Meths. Engrg. 32 (1991) 1205–1227.

[62] J. H. Bramble, J. E. Pasciak, and A. H. Schatz, The construction of preconditioners for elliptic problems by substructuring, I, Math. Comp. 47 (1986) 103–134.

[63] I. S Duff, Parallel implementation of multifrontal schemes, Parallel Computing 3 (1986) 193–204.

[64] J. W. H. Liu, The multifrontal method for sparse matrix solution: theory and practice, SIAM Review 34 (1992) 82–109.

[65] R. E. Benner, G. R. Montry and G. G. Weigand, Concurrent multifrontal methods: shared memory, cache, and frontwidth issues, Int. J. Supercomp. Appl. 1 (1987) 26–44.

[66] M. Lesoinne, C. Farhat and M. Géradin, Parallel/vector improvements of the frontal method, Internat. J. Numer. Meths. Engrg. 32 (1991) 1267–1282.

[67] A. Pothen and C. Sun, A mapping algorithm for parallel sparse matrix factorization, SIAM J. Sci. Comput. 4 (1993) 1253–1257.

[68] A. Pothen, E. Rothberg, H. Simon and L. Wang, Parallel sparse Cholesky factorization with spectral nested dissection ordering, RNR-094-011, NASA Ames Research Center, May 1994.

[69] M. Dryja and O. B. Widlund, Domain decomposition algorithms with small overlap, SIAM J. Sci. Comput. 15 (1994) 604–620.

[70] J. Mandel and R. Tezaur, Convergence of a substructuring method with Lagrange multipliers, Numerische Mathematik, (in press).

[71] C. Farhat, Optimizing substructuring methods for repeated right hand sides, scalable parallel coarse solvers, and global/local analysis, in: D. Keyes, Y. Saad and D. G. Truhlar, eds., Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering, SIAM (1995) 141–160.

[72] C. Farhat, P. S. Chen and P. Stern, Towards the ultimate iterative substructuring method: combined numerical and parallel scalability, and multiple load cases, J. Comput. Sys. Engrg. 117 (1994) 195–209.

[73] C. Farhat and F.X. Roux, An unconventional domain decomposition method for an efficient parallel solution of large-scale finite element systems, SIAM J. Sc. Stat. Comp. 13 (1992) 379–396.

[74] C. Farhat, L. Crivelli and F. X. Roux, A transient FETI methodology for large-scale parallel implicit computations in structural mechanics, Internat. J. Numer. Meths. Engrg. 37 (1994) 1945–1975.

[75] J. Mandel and C. Farhat, The FETI method for plate problems, Comput. Meths. Appl. Mech. Engrg., (in press).

[76] C. Farhat, L. Crivelli and M. Géradin, On the spectral stability of time integration algorithms for a class of constrained dynamics problems, AIAA Paper 93-1306, AIAA 34th Structural Dynamics Meeting, 1993.

[77] C. Farhat and M. Géradin, Using a reduced number of Lagrange multipliers for assembling parallel incomplete field finite element approximations, Comput. Meths. Appl. Mech. Engrg. 97 (1992) 333–354.

[78] C. Farhat and M. Géradin, On a component mode synthesis method and its application to incompatible substructures, Comput. & Struc. 51 (1994) 459–473.

[79] L. Petzold, Differential/algebraic equations are not ODE's, SIAM J. Sci. Stat. Comput. 3 (1982) 367-384.

[80] Y. Saad, On the Lanczos method for solving symmetric linear systems with several right-hand sides, Math. Comp. 48 (1987) 651–662.

[81] C. Farhat and P. S. Chen, Tailoring domain decomposition methods for efficient parallel coarse grid solution and for systems with many right hand sides," Contemporary Mathematics 180 (1994) 401–406.

[82] C. Farhat, S. Lanteri and H. D. Simon, TOP/DOMDEC, A software tool for mesh partitioning and parallel processing, J. Comput. Sys. Engrg. (in press).

[83] H. D. Simon, Partitioning of unstructured problems for parallel processing, Comput. Sys. Engrg. 2 (1991) 135–148.

[84] C. Farhat, A simple and efficient automatic FEM domain decomposer, Comput. & Struct. 28 (1988) 579–602.

[85] C. Farhat and M. Lesoinne, Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics, Internat. J. Numer. Meths. Engrg. 36 (1993) 745–764.

[86] J. G. Malone, Automated mesh decomposition and concurrent finite element analysis for hypercube multiprocessors computers, Comput. Meths. Appl. Mech. Engrg. 70 (1988) 27–58.

[87] J. Flower, S. Otto and M. Salama, Optimal mapping of irregular finite element domains to parallel processors, in: A. K. Noor, ed., Parallel Computations and Their Impact on Mechanics, The American Society of Mechanical Engineers, AMD-Vol. 86 (1987) 239–252.

[88] A. Pothen, H. Simon and K. P. Liou, Partitioning sparse matrices with eigen vectors of graphs, SIAM J. Mat. Anal. Appl. 11 (1990) 430–452. (1990)

[89] C. Farhat, E. Wilson and G. Powell, Solution of finite element systems on concurrent processing computers, Engrg. with Comput. 2 (1987) 157–165.

[90] A. I. Khan and B. H. V. Topping, Subdomain generation for parallel finite element analysis, Comput. Sys. Engrg. 4 (1993) 473–488.

[91] P. Ciarlet and F. Lamour, An efficient low-cost greedy graph partitioning heuristic, UCLA CAM Report 94-1.

[92] P. Ciarlet and F. Lamour, Recursive partitioning methods and greedy partitioning methods: a comparison on finite element graphs, UCLA CAM Report 94-9 (also submitted to Internat. J. High Speed Computing).

[93] S. T. Barnard and H. D. Simon, A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems, Concurrency: Practice and Experience 6 (1994) 101–107.

[94] J. Zdenek, K. K. Mathur, S. L. Johnsson and T. J. R. Hughes, An efficient communication strategy for finite element methods on the Connection Machine CM-5 system, Comput. Methds. Appl. Mech. Engrg. 113 (1994) 363–387.

[95] O. Zone, D. Vanderstraeten, P. Henriksen, and R. Keunings, A parallel direct solver for implicit finite element problems based on automatic domain decomposition, Proc. Int. Conf. on Massively Parallel Processing Applications and Development, L. Dekker (Ed.), Elsevier, (in press).

[96] C. Farhat, N. Maman and G. Brown, Mesh partitioning for implicit computations via iterative domain decomposition: impact and optimization of the subdomain aspect ratio," Internat. J. Numer. Meths. Engrg. 38 (1995) 989–1000.

[97] D. Vanderstraeten, R. Keunings and C. Farhat, Optimization of mesh partitions and impact on parallel CFD, Proceedings Parallel CFD'93, Paris, France, May 10-12 (1993).

[98] D. Vanderstraeten and R. Keunings, Optimized partitioning of unstructured computational grids, Internat. J. Numer. Meths. Engrg. 38 (1995) 433-450.

[99] D. Vanderstraeten, C. Farhat, P. S. Chen, R. Keunings, and O. Zone, A retrofit and contraction based methodology for the fast generation and optimization of mesh partitions: beyond the minimum interface size criterion, Comput. Meths. Appl. Mech. Engrg., (in press).

[100] S. Kirkpatrick, C. Gelatt and M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[101] F. Glover, C. McMillan and B. Novick, Interactive decision software and computer graphics for architectural and space planning, Ann. Opns. Res. 5 (1985) 557–573.

[102] Y. G. Saab and V. B. Rao, Combinatorial optimization by stochastic evolution, IEEE Trans. C.A.D. 10 (1991) 525–535.

[103] T. N. Bui and C. Jones, A heuristic for reducing fill-in in sparse matrix factorization, Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, Virginia, (1993) 445–452.

[104] E. Barszcz, Intercube communication on the iPSC/860, Scalable High Performance Computing Conference, Williamsburg, April 26-29, 1992.

REPORT DOCUMENTATION PAGE

1. Recipient's Reference	2. Originator's Reference	3. Further Reference	4. Security Classification of Document
	AGARD-R-807	ISBN 92-836-1025-3	UNCLASSIFIED/ UNLIMITED

5. Originator Advisory Group for Aerospace Research and Development
North Atlantic Treaty Organization
7 rue Ancelle, 92200 Neuilly-sur-Seine, France

6. Title
Parallel Computing in CFD

7. Presented at/sponsored by
AGARD-FDP-VKI Special Course at the VKI, Rhode-Saint-Genèse, Belgium, 15-19 May 1995 and 16-20 October 1995 at NASA Ames, United States.

8. Author(s)/Editor(s)	9. Date
Multiple	October 1995
10. Author's/Editor's Address	**11. Pages**
Multiple	352

12. Distribution Statement There are no restrictions on the distribution of this document. Information about the availability of this and other AGARD unclassified publications is given on the back cover.

13. Keywords/Descriptors

Fluid dynamics	Parallel computing
Computational fluid dynamics	Compressible flow
Algorithms	Incompressible flow
Differential equations	Demain decomposition algorithms
Fluid flow	Partitioning techniques

14. Abstract

Lecture notes for the AGARD Fluid Dynamics Panel (FDP) Special Course on "Parallel Computing in CFD" have been assembled in this report. The aim and scope of this Course was to present and discuss the latest in advances and future trends in the application of parallel computing to solve computationally intensive problems in CFD. Topics in this lecture series focus on the increasingly sophisticated types of architectures now available, and how to exploit these architectures by appropriate algorithms for the simulation of fluid flow. Some of the subjects discussed are: parallel algorithms for computing compressible and incompressible flow; domain decomposition algorithms and partitioning techniques; and parallel algorithms for solving linear systems arising from the discretized partial differential equations.

The material assembled in this report was prepared under the combined sponsorship of the AGARD Fluid Dynamics Panel, the Consultant and Exchange Program of AGARD, and the von Kármán Institute (VKI) for Fluid Dynamics.

AGARD

NATO OTAN

7 RUE ANCELLE • 92200 NEUILLY-SUR-SEINE

FRANCE

Télécopie (1)47.38.57.99 • Télex 610 176

DIFFUSION DES PUBLICATIONS

AGARD NON CLASSIFIEES

Aucun stock de publications n'a existé à AGARD. A partir de 1993, AGARD détiendra un stock limité des publications associées aux cycles de conférences et cours spéciaux ainsi que les AGARDographies et les rapports des groupes de travail, organisés et publiés à partir de 1993 inclus. Les demandes de renseignements doivent être adressées à AGARD par lettre ou par fax à l'adresse indiquée ci-dessus. *Veuillez ne pas téléphoner.* La diffusion initiale de toutes les publications de l'AGARD est effectuée auprès des pays membres de l'OTAN par l'intermédiaire des centres de distribution nationaux indiqués ci-dessous. Des exemplaires supplémentaires peuvent parfois être obtenus auprès de ces centres (à l'exception des Etats-Unis). Si vous souhaitez recevoir toutes les publications de l'AGARD, ou simplement celles qui concernent certains Panels, vous pouvez demander à être inclu sur la liste d'envoi de l'un de ces centres. Les publications de l'AGARD sont en vente auprès des agences indiquées ci-dessous, sous forme de photocopie ou de microfiche.

CENTRES DE DIFFUSION NATIONAUX

ALLEMAGNE
Fachinformationszentrum,
Karlsruhe
D-76344 Eggenstein-Leopoldshafen 2

BELGIQUE
Coordonnateur AGARD-VSL
Etat-major de la Force aérienne
Quartier Reine Elisabeth
Rue d'Evere, 1140 Bruxelles

CANADA
Directeur, Services d'information scientifique
Ministère de la Défense nationale
Ottawa, Ontario K1A 0K2

DANEMARK
Danish Defence Research Establishment
Ryvangs Allé 1
P.O. Box 2715
DK-2100 Copenhagen Ø

ESPAGNE
INTA (AGARD Publications)
Pintor Rosales 34
28008 Madrid

ETATS-UNIS
NASA Headquarters
Code JOB-1
Washington, D.C. 20546

FRANCE
O.N.E.R.A. (Direction)
29, Avenue de la Division Leclerc
92322 Châtillon Cedex

GRECE
Hellenic Air Force
Air War College
Scientific and Technical Library
Dekelia Air Force Base
Dekelia, Athens TGA 1010

ISLANDE
Director of Aviation
c/o Flugrad
Reykjavik

ITALIE
Aeronautica Militare
Ufficio del Delegato Nazionale all'AGARD
Aeroporto Pratica di Mare
00040 Pomezia (Roma)

LUXEMBOURG
Voir Belgique

NORVEGE
Norwegian Defence Research Establishment
Attn: Biblioteket
P.O. Box 25
N-2007 Kjeller

PAYS-BAS
Netherlands Delegation to AGARD
National Aerospace Laboratory NLR
P.O. Box 90502
1006 BM Amsterdam

PORTUGAL
Força Aérea Portuguesa
Centro de Documentação e Informação
Alfragide
2700 Amadora

ROYAUME-UNI
Defence Research Information Centre
Kentigern House
65 Brown Street
Glasgow G2 8EX

TURQUIE
Millî Savunma Başkanliği (MSB)
ARGE Dairesi Başkanliği (MSB)
06650 Bakanliklar-Ankara

Le centre de distribution national des Etats-Unis ne détient PAS de stocks des publications de l'AGARD.
D'éventuelles demandes de photocopies doivent être formulées directement auprès du NASA Center for AeroSpace Information (CASI) à l'adresse ci-dessous. Toute notification de changement d'adresse doit être fait également auprès de CASI.

AGENCES DE VENTE

NASA Center for
AeroSpace Information (CASI)
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
Etats-Unis

ESA/Information Retrieval Service
European Space Agency
10, rue Mario Nikis
75015 Paris
France

The British Library
Document Supply Division
Boston Spa, Wetherby
West Yorkshire LS23 7BQ
Royaume-Uni

Les demandes de microfiches ou de photocopies de documents AGARD (y compris les demandes faites auprès du CASI) doivent comporter la dénomination AGARD, ainsi que le numéro de série d'AGARD (par exemple AGARD-AG-315). Des informations analogues, telles que le titre et la date de publication sont souhaitables. Veuiller noter qu'il y a lieu de spécifier AGARD-R-nnn et AGARD-AR-nnn lors de la commande des rapports AGARD et des rapports consultatifs AGARD respectivement. Des références bibliographiques complètes ainsi que des résumés des publications AGARD figurent dans les journaux suivants:

Scientific and Technical Aerospace Reports (STAR)
publié par la NASA Scientific and Technical
Information Division
NASA Headquarters (JTT)
Washington D.C. 20546
Etats-Unis

Government Reports Announcements and Index (GRA&I)
publié par le National Technical Information Service
Springfield
Virginia 22161
Etats-Unis
(accessible également en mode interactif dans la base de
données bibliographiques en ligne du NTIS, et sur CD-ROM)

Imprimé par le Groupe Communication Canada
45, boul. Sacré-Cœur, Hull (Québec), Canada K1A 0S7

AGARD

NATO OTAN

7 RUE ANCELLE • 92200 NEUILLY-SUR-SEINE

FRANCE

Telefax (1)47.38.57.99 • Telex 610 176

DISTRIBUTION OF UNCLASSIFIED

AGARD PUBLICATIONS

AGARD holds limited quantities of the publications that accompanied Lecture Series and Special Courses held in 1993 or later, and of AGARDographs and Working Group reports published from 1993 onward. For details, write or send a telefax to the address given above. *Please do not telephone.*

AGARD does not hold stocks of publications that accompanied earlier Lecture Series or Courses or of any other publications. Initial distribution of all AGARD publications is made to NATO nations through the National Distribution Centres listed below. Further copies are sometimes available from these centres (except in the United States). If you have a need to receive all AGARD publications, or just those relating to one or more specific AGARD Panels, they may be willing to include you (or your organisation) on their distribution list. AGARD publications may be purchased from the Sales Agencies listed below, in photocopy or microfiche form.

NATIONAL DISTRIBUTION CENTRES

BELGIUM
Coordonnateur AGARD — VSL
Etat-major de la Force aérienne
Quartier Reine Elisabeth
Rue d'Evere, 1140 Bruxelles

CANADA
Director Scientific Information Services
Dept of National Defence
Ottawa, Ontario K1A 0K2

DENMARK
Danish Defence Research Establishment
Ryvangs Allé 1
P.O. Box 2715
DK-2100 Copenhagen Ø

FRANCE
O.N.E.R.A. (Direction)
29 Avenue de la Division Leclerc
92322 Châtillon Cedex

GERMANY
Fachinformationszentrum
Karlsruhe
D-76344 Eggenstein-Leopoldshafen 2

GREECE
Hellenic Air Force
Air War College
Scientific and Technical Library
Dekelia Air Force Base
Dekelia, Athens TGA 1010

ICELAND
Director of Aviation
c/o Flugrad
Reykjavik

ITALY
Aeronautica Militare
Ufficio del Delegato Nazionale all'AGARD
Aeroporto Pratica di Mare
00040 Pomezia (Roma)

LUXEMBOURG
See Belgium

NETHERLANDS
Netherlands Delegation to AGARD
National Aerospace Laboratory, NLR
P.O. Box 90502
1006 BM Amsterdam

NORWAY
Norwegian Defence Research Establishment
Attn: Biblioteket
P.O. Box 25
N-2007 Kjeller

PORTUGAL
Força Aérea Portuguesa
Centro de Documentação e Informação
Alfragide
2700 Amadora

SPAIN
INTA (AGARD Publications)
Pintor Rosales 34
28008 Madrid

TURKEY
Millî Savunma Başkanliği (MSB)
ARGE Dairesi Başkanliği (MSB)
06650 Bakanliklar-Ankara

UNITED KINGDOM
Defence Research Information Centre
Kentigern House
65 Brown Street
Glasgow G2 8EX

UNITED STATES
NASA Headquarters
Code JOB-1
Washington, D.C. 20546

The United States National Distribution Centre does NOT hold stocks of AGARD publications.
Applications for copies should be made direct to the NASA Center for AeroSpace Information (CASI) at the address below.
Change of address requests should also go to CASI.

SALES AGENCIES

NASA Center for
AeroSpace Information (CASI)
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
United States

ESA/Information Retrieval Service
European Space Agency
10, rue Mario Nikis
75015 Paris
France

The British Library
Document Supply Centre
Boston Spa, Wetherby
West Yorkshire LS23 7BQ
United Kingdom

Requests for microfiches or photocopies of AGARD documents (including requests to CASI) should include the word 'AGARD' and the AGARD serial number (for example AGARD-AG-315). Collateral information such as title and publication date is desirable. Note that AGARD Reports and Advisory Reports should be specified as AGARD-R-nnn and AGARD-AR-nnn, respectively. Full bibliographical references and abstracts of AGARD publications are given in the following journals:

Scientific and Technical Aerospace Reports (STAR)
published by NASA Scientific and Technical
Information Division
NASA Headquarters (JTT)
Washington D.C. 20546
United States

Government Reports Announcements and Index (GRA&I)
published by the National Technical Information Service
Springfield
Virginia 22161
United States
(also available online in the NTIS Bibliographic
Database or on CD-ROM)

Printed by Canada Communication Group
45 Sacré-Cœur Blvd., Hull (Québec), Canada K1A 0S7

ISBN 92-836-1025-3