Institut National de Recherche
en Informatique et en Automatique

Neuvième Conférence Internationale sur les METHODES DE CALCUL SCIENTIFIQUE ET TECHNIQUE

Ninth International Conference on COMPUTING METHODS IN APPLIED SCIENCES AND ENGINEERING

29 Janvier - 2 Février 1990
January 29 - February 2, 1990

PARIS (France)

Edition Provisoire
Preprints

UNITÉS DE RECHERCHE

INRIA-LORRAINE INRIA-RENNES

INRIA-ROCQUENCOURT INRIA-SOPHIA ANTIPOLIS

ÉDITEUR

INRIA-Domaine de Voluceau-Rocquencourt-BP 105 /78153 Le Chesnay Cedex

TABLE DES MATIERES
TABLE OF CONTENTS

PROGRES DU LOGICIEL
ADVANCES IN SOFTWARE

MODELISATION NUMERIQUE
NUMERICAL MODELS

SIMULATION NUMERIQUE EN PHYSIQUE ET CHIMIE
COMPUTATIONAL PHYSICS AND CHEMISTRY

APPLICATIONS DES SUPERCALCULATEURS
SUPERCOMPUTER APPLICATIONS

METHODES NUMERIQUES
NUMERICAL METHODS

PREFACE

Le présent volume contient les textes des communications présentées à la Neuvième Conférence Internationale sur les Méthodes de Calcul Scientifique et Technique.

Les actes de cette Conférence reflètent l'évolution du calcul scientifique au cours de ces dernières années, à savoir :

1/ Au plan méthodologique

Un recours accru aux calculateurs vectoriels et parallèles, impliquant la nécessité de repenser l'algorithmique numérique et les logiciels permettant de la mettre en *oeuvre sur ces nouvelles machines.*

2/ Au plan des applications

La résolution des problèmes complexes issus de la simulation numérique en hypersonique, en micro-électronique, en chimie quantique, en combustion ou associés au développement en cours concernant les stations spatiales habitées.

3/ Au plan des mathématiques appliquées

Le développement de méthodes nouvelles telles que les ondelettes, dont le domaine d'application est sans cesse croissant, en particulier grâce à l'interaction des disciplines scientifiques, suscitée par des manifestations telles que cette Conférence.

Nous espérons que les participants bénéficieront des communications présentées lors de cette rencontre, et des possibilités de communiquer directement avec les conférenciers et autres spécialistes présents à cette Conférence. Bien entendu, nous souhaitons qu'ils y trouvent de nouveaux outils leur permettant de résoudre, de façon plus efficace, les problèmes scientifiques difficiles, *rencontrés dans le cadre de leurs activités professionnelles.*

Les Organisateurs.

FOREWORD

The present volume contains the texts of the oral communications presented at the Ninth International Conference on Computing Methods in Applied Sciences and Engineering.

The proceedings of this Conference demonstrate the evolution undertaken by scientific computing during these last years :

1/ At the methodological level

An increasing use of vector and parallel machines implying the necessity of redefining numerical algorithms and softwares in view of computations on these new machines.

2/ At the application level

The solution of complicated problems on hypersonic flow, microelectronic, quantum chemistry, combustion, space station programs...

3/ At the applied mathematical level

The development of new methods, such as wavelets, whose field of applications is steadily necessary, thanks, in particular, to the interaction taking place at scientific meetings, such as the present one.

We hope that the participants will benefit from the communications presented at this Conference and will have the possibility of communicating directly with the speakers and other specialists of the field. We also hope that these scientists will find new tools allowing them to solve with more efficiency the difficult problems they encounter in their professional life.

The Organizers

SIMULATION NUMERIQUE EN MECANIQUE DES FLUIDES

COMPUTATIONAL FLUID DYNAMICS

HYPERSONIC FLOW SIMULATIONS USING DSMC

By

James N. Moss

NASA Langley Research Center
Hampton, VA 23665-5225

Ninth International Conference on
Computing Methods in Applied Sciences and Engineering

Paris, France
January 29 - February 2, 1990

HYPERSONIC FLOW SIMULATIONS USING DSMC

James N. Moss
NASA Langley Research Center
Hampton, VA 23665-5225

Abstract

A review of the direct simulation Monte Carlo (DSMC) method of Bird is presented. The DSMC method provides the capability of simulating real gas flows in the rarefied flow regime. Recent developments and applications of the method for hypersonic flows are reported for both ground-based tests and during entry. Results obtained using both axisymmetric and three-dimensional codes are included.

Nomenclature

$\vec{c}$	velocity
c_r	magnitude of the relative velocity (speed) between two molecules
C_D	drag coefficient
C_H	heat-transfer coefficient, $2\ q/\rho_\infty V_\infty^3$
C_i	species mass fraction
C_L	lift coefficient
C^*	$\mu^* T_\infty / \mu_\infty T^*$
f	normalized velocity distribution function in velocity space
F_N	number of physical molecules represented by each simulated molecule
Kn	Knudsen number, λ/ℓ
K_r^2	Cheng's parameter, $\rho_\infty R_N / \mu_\infty V_\infty C^*$
ℓ	characteristic dimension
L/D	lift-to-drag ratio
n	number density
$\tilde{n}$	nondimensional density rise in shock wave, $(\rho - \rho_1)/(\rho_2 - \rho_1)$
N	number of simulated molecules in a computational cell
p	pressure
q	heat flux
R_N	nose radius
t	time
Δt	flow time step
T	temperature
V_c	volume of computational cell
V_∞	freestream velocity
x,y,z	Cartesian coordinates
α	angle of incidence
δ	shock wave thickness
η	coordinate normal to body surface
λ	mean-free path
μ	viscosity
ρ	density
σ	total collision cross section

Abbreviations

AFE	Aeroassist Flight Experiment
ASTV	Aeroassisted Space Transfer Vehicle
DSMC	direct simulation Monte Carlo
FM	free molecular
NASP	National Aero-Space Plane
NTC	no time counter
TSS-2	Tethered Satellite System - 2
VHS	variable hard sphere
VSL	viscous shock-layer

Introduction

With the commitment of several nations to expand the current space transportation capabilities through the use of transatmospheric winged vehicles and aeroassisted space transfer vehicles (ASTV's), attention is now being seriously focused on the aerothermodynamics of vehicles at very high altitudes. In this rarefied flow regime, the molecular mean-free path in the gas becomes significant when compared with either a characteristic distance over which important flow-property changes take place or when compared with the size of the object creating the flow disturbance. Since the flow is hypersonic, the flow disturbance that envelopes the space vehicle will be a nonequilibrium flow; that is, one in which nonequilibrium[1-2] exists among the various energy modes (translational and internal), the chemistry, and radiation[3-5] for the more energetic flows. For such flows, the shock-wave thickness is significant[6-9] (shown in Fig. 1 is the location in altitude-velocity space where the shock wave δ_s is 10 percent of the shock standoff for a 10-cm nose radius), and the flowfield disturbance created by the vehicle is very much larger than that experienced under continuum flow conditions. At the surface of the vehicle, the gas-surface interactions are very important[10-11] as they significantly influence the aerodynamic forces and heating that the vehicle experiences.

To describe such flows, one must acknowledge the discrete nature of the flow. Because of the limitations of the continuum description, as expressed by the Navier-Stokes equations, to simulate rarefied flows and the difficulties of solving the Boltzmann equation, which acknowledges the discrete nature of the flow, direct physical simulation methods have been developed over the last three decades for modeling rarefied effects. These developments have generally been concerned with the direct simulation Monte Carlo (DSMC) method[12-17] and, to a lesser extent, with the Molecular Dynamics method.[17-18] The direct simulation Monte Carlo (DSMC) method of Bird is the most used method today for simulating rarefied flows in an engineering context. The DSMC method takes advantage of the discrete structure of the gas and provides a direct physical simulation as opposed to a numerical solution of a set of model equations. This is accomplished by developing phenomenological models of the relevant physical events. Phenomenological models have been developed and implemented in the DSMC procedure to account for translational, thermal, chemical, and radiative nonequilibrium effects. The present discussion will review the general features of the DSMC method, the numerical requirements for obtaining meaningful results, the modeling used to simulate high-temperature gas effects, and applications of the method to calculate the flow about various configurations under hypersonic low-density conditions. Results obtained using both axisymmetric and three-dimensional codes are included.

Rarefied Hypersonic Flows

A wide range of engineering studies associated with current and projected space vehicles are concerned with the aerothermodynamics of low-density gas flows. Flows of particular interest can arise from interactions between two or more of the following events: the vehicle itself, the ambient atmosphere, exhaust plumes from upper stage motors or control motors, and other emitted gases from material outgassing and waste gas venting. Studies concerned with these interactions are receiving added impetus by the Space Shuttle Orbiter flights, the commitment of several nations to pursue the goal of transatmospheric flight with hypersonic slender vehicles, space experiments, technology demonstration programs such as the Aeroassist Flight Experiment (AFE) vehicle and the Tethered Satellite System-2 (TSS-2), the projected space station, and aeroassisted space transfer vehicles (ASTV's). (See Fig. 1.) On-orbit and high-altitude flight applications occur under conditions where the effects of rarefaction can be very significant in terms of the development of the flowfield structure that envelopes a vehicle or spacecraft and the momentum and energy transport to its surface. The degree of rarefaction is conventionally expressed through an overall Knudsen number defined by

$$Kn = \lambda_\infty / \ell \tag{1}$$

where λ_∞ is the mean free path in the undisturbed gas and ℓ is some typical dimension of the flow. Bird[1] has suggested that a more precise quantification of rarefaction effects should be based on a local Knudsen number defined as

$$Kn = (\lambda/\alpha)\ (\partial\alpha/\partial x) \tag{2}$$

where λ is the local mean free path, x is a linear dimension, and α is a macroscopic flow variable such as density, velocity, or temperature. When the value of the local Knudsen number approaches 0.1, the continuum formulation as modeled by the Navier-Stokes equations becomes suspect, since the Chapman-Enskog expressions for viscosity, heat conduction, and diffusion coefficients are in error. In fact, the Chapman-Enskog expressions become virtually unusable when the local Knudsen number exceeds 0.2. The ranges of validity, in terms of local Knudsen number, of the equations that describe a gas flow as a continuum or as a set of discrete particles are shown in Fig. 2.[1]

Even though the Boltzmann equation is the classical formulation for describing a gas as a set of individual particles, this equation has remained intractable to analytical and conventional numerical solution for space-related applications. Applications have been largley restricted to a perfect monatomic gas where the flow is steady and one dimensional. The restriction to relatively simple flows is primarily due to the computational requirements of any numerical method that has to work in phase space. The addition of chemical reactions would mean that the Boltzmann equation would be difficult to formulate, let alone solve! Futhermore, almost all space-related applications involve flows with at least two spatial dimensions and a three-dimensional distribution function in velocity space. This leads to a five-dimensional grid, and direct numerical solutions can hardly be contemplated.

Because of the limited prospects of direct numerical solutions of the Boltzmann equation for practical space applications, an alternate approach has been developed. The alternative to a formal numerical solution is to take advantage of the discrete structure of the gas and conduct a direct physical simulation of the flow using the computer. The DSMC method is such an alternative, and is the method today that is most readily applied to hypersonic flow problems in the transitional flow flight regime, that is, flow problems bounded by the continuum and the free molecular flow flight regimes.

DSMC Method

The DSMC method[12-15] is a technique for the computer modelling of a real gas by thousands of simulated molecules. The velocity components and position coordinates of these molecules are stored in the computer and are modified with time as the molecules are concurrently followed through representative collisions and boundary interactions in simulated physical space.

For a simple dilute gas, the assumptions used in implementing the DSMC method are consistent with the assumptions underlying the nonlinear Boltzmann equation

$$\frac{\partial(nf)}{\partial t} + \vec{c} \cdot \nabla(nf) = L(f,f), \qquad (3)$$

where $f(\vec{c}, t, \vec{r})$ is the molecular distribution function, $\vec{r}$ the position vector, t the time, $\vec{c} = (u, v, w)$ the particle velocity, and $n(\vec{r},t)$ the macroscopic particle number density. In equation (3) the left side is the particle movement or convection operator, and L(f,f) is the collision operator representing changes in f due to binary collisions between molecules. In DSMC no actual use of equation (3) is made, and the phase space (i.e., $\vec{r}$, $\vec{c}$) information is carried directly by an ensemble of a few thousand sample molecules. These simulation particles or molecules exist within a framework of cells and at any instant are assumed to represent a sample from f within the phase space being considered. Starting from some initial configuration, DSMC computes the evolution of the sample ensemble through a sequence of discretized time intervals $j\,\Delta t$ (where $j = 1,2,\ldots$). Under certain conditions on Δt, the two sides of equation (3) may be decoupled, and each may be simulated alternately in the time sequence. During the convection simulation, each particle moves in a free trajectory in Δt and interacts with any boundary encountered according to prescribed boundary strategies. For the simulation of the right side of equation (3), a statistical interpretation of L(f,f) leads to a simple algorithm for the calculation of sample collisions during Δt in each of a set of spatial cells which represent the spatial component of phase space. Statistical estimates of the macroscopic fluid properties or the surface properties (pressure, etc.) represent the "solution" to the flow problem and are obtained by averaging the contributions of individual particles as they pass through cells or strike the body surface during the calculation.

The assumptions of a dilute gas (mean spacing between molecules is much larger than the mean particle diameter) and molecular chaos are common to both the DSMC method and the Boltzmann equation. The consistency between

the two was demonstrated at an early stage of development[19] through the derivation of the Boltzmann equation from the DSMC method, and the relationship between the two has since been investigated in some detail by Nanbu.[20] The objective of the simulation should be to obtain a valid physical model of the real gas flow, and while this is generally equivalent to a solution of the Boltzmann equation, the DSMC procedures often go beyond the limitations of the Boltzmann formulation. For example, simulations have included some dense gas effects[21] such as ternary collisions. Morever, the method is routinely applied to problems involving chemical reactions[1-5] and to a lesser extent problems with thermal radiation.[3-5] These effects are also beyond the current formulations of the Boltzmann equation.

Computational Aspects

The principal computational tasks associated with the DSMC method are movement of the molecules, indexing molecules into cells, molecular collisions, and sampling of flowfield and surface quantities. (See Fig. 3.) The molecules used in the analyses are simulated molecules, each of which represents a very large number (on the order of 10^{15}) of physical molecules. Thus, scaling the density by a very large factor has the effect of substantially reducing the number of molecular trajectories and molecular collisions that must be calculated. Remember, however, that the physical velocities, molecular size (diameter or cross section), and internal energies are preserved in the simulation.

Another major aspect of the DSMC method is the uncoupling of the molecular motion and molecular collisions. The validity of this dichotomy is assured by requiring that the computational time step be small when compared with the real physical collision time [$\Delta t < (n\sigma\bar{c}_r)^{-1}$]. In addition to the time discretization, a cell structure is required for two purposes: first, the selection of potential collision pairs and second, the sampling of flow properties. The simulation becomes more exact as the cell size and time step tend to approach zero.

The cell dimensions must be small in comparison with the scale length of the macroscopic flow gradients. The simulated molecules in the cell are then regarded as representative of the real molecules at the location of the cell, and the relative location of the molecules within the cell is disregarded in the selection of collision partners. It is well established[2] that the cell size must be small in comparison with the local mean free path in regions of large gradients. For problems with large density variations, the use of variable cell sizes assists in resolving the flow gradients and also minimizing the computational requirements provided that the flow is steady. Since the flow is always calculated as an unsteady flow starting from some initial specified state (usually a uniform freestream or vacuum), any steady flow becomes the large time state of the unsteady flow. For boundary conditions where the flow is steady, the overall computational effort can be substantially reduced by subdividing the flowfield into an arbitrary number of units (regions) where the time step Δt and the scaling factor F_N (the number of physical molecules represented by each simulated molecule) remain constant within a region, but can vary from region to

region. Of course, such simulations are not time consistent solutions, but they provide steady state solutions with a substantial reduction in computational requirements. The combination of subdividing the flowfield into regions along with the use of variable cell sizes provides the flexibility to substantially reduce the total number of molecules used in the simulation and also resolve the flow gradients. Recall that in the DSMC method of Bird, the procedures are specified such that the computational time is linearly dependent on the number of molecules.

There is some lower limit on the number of simulated molecules per cell because the cell-sampled density is used in the procedures for establishing the collision rate. It is desirable to have the number of molecules per cell as large as the order of ten. The other function of the cell besides sampling is the selection of collision pairs. During this process, it is desirable to reduce the mean separation distance of the collision pairs (pairs are selected without regard to position within the cell) and thereby minimize the smearing of gradients. Bird[15] has recently addressed this requirement by introducing the option of subdividing the sampling cell into an arbitrary number of sub-cells for the selection of collision pairs. The sub-cells are chosen to contain on average two or three molecules, so that all collisions approach the "nearest-neighbor" ideal. Should there be only one molecule in a sub-cell, the potential collision partner is selected from an adjacent sub-cell within the cell. With this procedure, the molecular sampling is still done on a cell basis, while the collision pairs are selected within the sub-cells. The computing time penalty associated with these additional procedures is negligible.

The basis for much of the comments on the DSMC method have centered on the way in which a representative set of collisions are selected for each cell at each discrete flow time step Δt so that the appropriate collision frequency is maintained. Prior to 1988, the collision sampling technique that had been recommended by Bird[12] was the "time-counter" method where advantage is taken of the fact that the computational time is linearly proportional to the number of molecules. The collision pairs are accepted with probability proportional to the product of the magnitude of the relative velocity $\vec{c}_r$ and the total collision cross section σ. [This is accomplished by normalizing the $c_r\sigma$ product by the maximum value that has ever occurred within the particular cell and then using an acceptance - rejection procedure (see Appendix D, Ref. 12) to accept or reject the collision pair that was selected at random.] For each collision pair selected, a "cell time" is advanced by

$$2/(N \langle n \rangle \sigma c_r) \qquad (4)$$

for a simple gas (see page 121 of Ref. 12 for the corresponding expression for inverse power law molecules), where N is the number of simulated molecules in the cell, and $\langle n \rangle$ is the time-averaged (steady flow) or ensemble-averaged (unsteady flow) number density in the physical flow. Sufficient collisions are calculated to keep the cell collision time concurrent with the flow time.

In Ref. 17, Bird introduces a replacement for the time-counter method. The new method is called the "no time counter" or NTC method, and he

strongly recommends it for all applications. A problem with the time-counter method is that the acceptance of an unlikely collision (one with a very small value for σc_r) can advance the cell time by an interval that is much larger than the flow time step Δt and the overall collision rate can be distorted. This is particulary true when the number of simulated molecules N in a cell is small.

The NTC method is obtained by modifying the "direct" or Kac method. In the direct method, all possible pairs in each cell are considered, and the probability of collision within the time step is equal to the ratio of the volume swept out by the cross section (moving with the relative velocity) to the cell volume V_c. The disadvantage is that the computation time is very nearly proportional to the square of the number of molecules. The direct method can be modified by reducing the number of sampled pairs by some factor and increasing the collision probabilities by the same factor. If the factor is such that the maximum collision probability of any pair is unity, the number of pairs to be sampled is

$$0.5\ N\ \langle N \rangle\ F_N\ (\sigma\ c_r)_{max}\ \Delta t / V_c \tag{5}$$

and the collision probability for each selection is

$$(\sigma\ c_r)\ /\ (\sigma\ c_r)_{max} \tag{6}$$

The selection criterion of equation (6) is identical to that used in the time counter method. The only change with this method is that the number of collision pair selections is given deterministically [Equation (5)] rather than probabilistically through the operation of the time counter. This removes the undesirable correlation[17] between unlikely collisions and collision time intervals that can occur when the cell time counter is advanced well beyond the current flow time.

Molecular Models

During the development and extension of the DSMC method, there has been a remarkable increase in the gas complexity for which numerical simulations are possible. The modeling has advanced from a simple hard sphere model to models that include inelastic effects such as rotation, vibration, chemical reactions, electronic excitation, and radiation. The routines used to compute the molecular interactions may be exercised millions of times during the course of a simulation, and it is essential for them to be brief. In developing a model and its numerical algorithm, a careful balance has to be struck between the realism of the physical representation and the computational efficiency.

For monatomic gases, the variable hard sphere (VHS) model is recommended[14,15] for engineering calculations. This model was selected based on the accumulated experience that the effects of molecular models can be correlated with the variation of the differential cross section of the molecules, which is a function of the relative velocity in the collision. The VHS model is essentially a hard sphere with a diameter that varies as some inverse power of the relative velocity in the collision. This is the simplest model that is capable of modeling the viscosity coefficient of real

molecules. A discussion of the VHS model and its overall applicability relative to other interaction models (both inverse power models to which the VHS model belongs and models which incorporate long range attractive forces, i.e., the Lennard-Jones (6-12), Buckingham exp-6, and Morse models) is given in Ref. 22.

Reference 15 gives a brief summary of the modeling currently implemented to describe internal degrees of freedom (rotation and vibration), chemical reactions, electronic excitation, and radiation. Reference 3 outlines in detail the modeling that is currently implemented for describing the effects of partial ionization, electronic excitation, and thermal radiation--effects that become important for very high entry velocities such as experienced by ASTV's in the transitional flow regime.

Surface Interactions

As discussed by Harvey[22], for most applications the distribution of molecules reflected from engineering surfaces appears to correspond closely to the diffuse and fully thermally accommodated pattern. Evidence contrary to this has been reported[23] based on observations from upper atmospheric flight measurements. These observations are now in question based on the results of recent DSMC simulations.[24-26] The DSMC simulations show that transitional effects persist at higher altitudes than had been assumed in the interpretation of the flight measurements. As a consequence, it appears that the aerodynamic characteristics can be explained in terms of a diffuse interaction with full thermal accommodation when transitional effects are included rather than resorting to a combination of diffuse and specular interactions while assuming free molecular flow.

The effects of deviation from the diffuse model with full thermal accommodation have been studied with DSMC calculations by applying the Maxwell boundary condition--a linear combination of fully accommodated diffuse and specular reflections--which is physically unrealistic. Numerous attempts[22] have been made to find more satisfactory ways of predicting the gas-surface interaction which range from simple empirical to complex quantum lattice models. In general, none of these models perform well, and most are too complicated for inclusion in DSMC calculations. Other than the Maxwell model, the only alternative that has been tested in a simulation is a modification of the Nocilla drifting Maxwellian model.[11]

Comparisons with Experiments

When the DSMC calculations are performed carefully (particular attention is given to the numerical requirements of cell size and time step and to the interaction modelling of the viscosity coefficient of the real molecules), the method appears to yield results that agree very precisely with experiments.[27-33] For example, Harvey and associates[27-28] have made numerous comparisons between experiments and DSMC results primarily for hypersonic nitrogen flows where internal translational energy exchange is a feature of importance. Results of comparisons for surface forces, heat transfer, and flowfield profiles (density and rotational temperature) have been reported for flow about sharp as well as blunt configurations.

In Ref. 30, Fiscko and Chapman calculated the one-dimensional shock-wave structure for argon and showed that the calculated shock density thickness was in good agreement with the measured values of Alsmeyer.[34] A more recent study[31-33] also investigated the shock wave structure of argon (Mach 7) and helium (Mach 1.59, 20, and 25) flows using the DSMC method. In these studies, the comparisons between calculation and experiment are done at a very fundamental level in that the comparisons are made for the molecular-velocity distribution function in the shock wave. The measurements (Fig. 4) of the molecular velocities inside a hypersonic normal shock wave, where the gas experiences rapid changes in macroscopic properties, show a highly nonequilibrium molecular motion (translational nonequilibrium) and a bimodal velocity distribution in the direction parallel to the flow. For the experimental conditions, the DSMC method of Bird provided accurate quantitative prediction of the molecular motion. These calculations and comparisons with measurements represent an important, detailed test of the DSMC method for elastic interatomic collisions.

Figure 4 presents a comparison of the calculated and experimental velocity distributions at three locations (n denotes the fraction of the density rise across the shock wave) within the normal shock wave. Shown are the velocity distributions both parallel and normal to the flow for Mach 25 helium. Since the flow was produced in a free-jet expansion, the freestream was not in equilibrium (temperature perpendicular to the flow was about 1.1 K while that parallel to the flow was 2.2 K), and this fact was accounted for in the simulation. The calculated results shown in Fig. 4 used the Maitland-Smith intermolecular potential[35] with a distance parameter of 2.976 Å and a well depth of 10.9 K. Similar results using the VHS model for Mach 20 helium are presented in Ref. 31.

The previously cited comparisons are examples of recent efforts to validate the DSMC technique by experiment. There have also been several studies at hypersonic flight conditions where qualitative validation is attempted by comparing with either limited flight measurements (Refs. 2, 25, and 36,) or with continuum methods (Refs. 2, 7, 8, and 37-41) for moderate to small Knudsen number flows.

Application of DSMC to Hypersonic Flows

This section will briefly summarize the results of five application studies that have been performed at the NASA Langley Research Center using the DSMC method. Four of the studies are concerned with flight applications (blunt slender bodies, Shuttle Orbiter, TSS-2, and AFE) that correspond to the conditions shown in Fig. 1, whereas the fifth has been conducted for future comparisons with measurements performed in a hypersonic wind tunnel.

Blunt Slender Body Calculations

Flows about cylindrically blunted wedges (2-D) and spherically blunted cones with body half angles of 0°, 5°, and 10° were calculated[7-8] with the DSMC method for entry conditions (V_∞ = 7.5 km/s, altitude range of 110 to 70 km). For a nose radius of 2.54 cm, the transitional flow effects persist below 70 km and are important in defining the heating to the leading edges of slender vehicles such as NASP. This is demonstrated by comparing

the DSMC calculations with continuum calculations using a viscous shock-layer (VSL) method. Both the DSMC and VSL calculations include a five-species reacting air gas model, a constant wall temperature of 1,000 K, and a finite catalytic wall. Results from this study are presented in Figs. 5-7. Figures 5 and 6 show the effect of the body configuration (2-D versus axisymmetric) on flowfield composition and stagnation-point heating rate. The extent of the flowfield disturbance is greater for the 2-D flow than the axisymmetric case. This impacts the amount of dissociation in the flow as is clearly demonstrated in Fig. 5 where the maximum atomic mass fractions are shown for both configurations as a function of altitude.

The stagnation point heat-transfer coefficient as a function of Cheng's parameter ($\rho_\infty R_N/\mu_\infty U_\infty C^*$) is presented in Fig. 6 for both the 2-D and axisymmetric bodies. Qualitatively, the results are what one would expect: an increase from the small value near the continuum regime to a value of unity as the free-molecule limit is approached. For the range of freestream conditions considered, the 2-D heat-transfer coefficient is always lower than the corresponding axisymmetric value.

The continuum VSL calculations were made for the 5° cone using the no-slip boundary conditions. Calculations at altitudes of 50, 60, 70, and 80 km were made with the overlap between the VSL and DSMC calculations being 70 and 80 km. Figure 6 shows the extent of the agreement between the two methods for stagnation point heat transfer. The continuum results begin to depart significantly from the DSMC data above 70 km, and the same is true for drag, which is not shown. If slip boundary conditions had been used in the VSL calculations, better agreement at the more rarefied conditions would have occurred.

Even through the computed stagnation point heat-transfer rates differ by only 15 percent at 70 km, there are significant differences between the predicted flowfield structure along the stagnation streamline, particulary downstream of the stagnation region. Figure 7 presents a comparison of the density profiles along the stagnation streamline. With respect to the VSL data, the DSMC calculations show that the upstream influence of the body is more than three times that predicted by the VSL calculation. The shock-layer thickness calculated with the VSL method is only about two freestream mean-free paths ($\lambda_\infty = 9.0X10^{-4}$ m) in thickness. The DSMC results are qualitatively what one would expect, since a freestanding normal shock wave is about five mean-free paths in thickness. (Note that the data points shown for the DSMC calculation are only a partial set, particulary near the wall.) The differences shown in the density adjacent to the surface can be explained in part by the temperature jump (537 K) calculated with the DSMC method and some differences in gas composition adjacent to the wall.

Aerodynamics of Shuttle Orbiter

Accurate predictions of aerothermal loads during entry can be very important for the design and development of hypersonic space vehicles. A portion of the reentry for these vehicles takes place in the transitional flow flight regime where the various nonequilibrium effects become important in establishing the thermal and aerodynamic response of these vehicles. In order to simplify the computational requirements, the aerothermal loads for vehicles such as the Space Shuttle Orbiter are often approximated[42-43] with

a flat plate at incidence for the free-molecular flow regime. For the transitional flow regime, empirical approximations are normally used to calculate these loads.[42-44] It has been observed from the Space Shuttle's flight experiments that measured values of lift-drag ratio are considerably higher than the free-molecular flow calculations at altitudes of 160 km and above where the flow regime was previously believed to be free-molecular. This discrepancy was thought to be due to specular reflection of some fraction of the molecules at the surface. As early as 1985, it was recognized[45] that transitional effects rather than specular gas-surface interaction might be influencing the interpretation of the flight measurements; however, no calculations were available to establish this hypothesis.

Recent direct simulation Monte Carlo (DSMC) calculations[24-25] of the rarefied flow past flat plates at incidence were the first to show that the transitional effects persist for the Space Shuttle Orbiter even at altitudes (160 km and above) where the flow had previously been considered as free-molecular. For the calculations of Ref. 25, two 12-m flat plates at 40° incidence were used to simulate the freestream Knudsen number of the Space Shuttle Orbiter during entry. One plate had zero thickness, and the second had a thickness of 0.5 m and a blunted leading edge (nose radius = 0.5 m). Both plates were 12 m in length, which corresponds to the mean aerodynamic cord of the Shuttle Orbiter's wings. DSMC calculations were made for an altitude range of 200 to 100 km at 7.5 km/s using a five-species reacting air model.

The surface temperature is assumed to be constant along the surface and equal to the wall radiative equilibrium value on the windward side (evaluated with free-molecular heating and a surface emittance of 0.09). Also, the wall is assumed to be diffuse with full thermal accommodation and to promote recombination of the oxygen and nitrogen atoms.

Figure 8 presents the calculated lift and drag coefficients for the flat plate as a function of freestream Knudsen number along with the corresponding free-molecular results. These results show the expected variation in the transitional flow regime. The drag coefficient increases and the lift coefficient decreases substantially with increasing rarefaction; both approach the free-molecular limit.

Figure 9 presents the lift-drag ratio as a function of freestream Knudsen number for both plates. Even at a Knudsen number of 16 (altitude = 200 km), the lift-to-drag ratio has not attained the free-molecular value. Results such as these have important implications for the interpretation of flight measurements used to deduce aerodynamic coefficients under rarefied conditions. At altitudes of 160 km and above, the conventional procedure[42-44] has been to interpret the flight measurements using the free-molecular-flow calculations. Such procedures have been used to establish what fraction of the gas-surface interaction is specular. As the fraction of specular reflection increases, the lift-drag ratio also increases for a given incidence angle. Since these two separate effects both produce increased lift-drag ratio, interpretation of flight measurements must account for the transitional effects.

Tethered Satellite Flowfield Characterization

The Tethered Satellite System-2 (TSS-2) is being proposed as a cooperative effort of the National Aeronautics and Space Administration of

the United States and the Agenzia Spaziale Italiana of Italy. For this mission, the Shuttle Orbiter would be used to demonstrate a tethered satellite system in a downward deployment and retrieval of a 500-kg, 1.6-m-diameter spacecraft attached to the end of a 100-km tether. The tethered spacecraft could reach downward into the outer atmosphere of the Earth to altitudes of 130 km for TSS-2 and later perhaps to 90 km. One of the objectives of the TSS-2 mission is to conduct hypersonic research in the transitional flow regime.

Initial calculations using the DSMC method have been made by Wilmoth[46] for a 1.6-m-diameter sphere. Figures 10 and 11 show selected contours of nondimensional density and temperature, respectively. These results are for a 130-km altitude and a freestream velocity of 7.5 km/s. The gas is a five-species reacting air model, and the wall temperature is constant at 350 K. At 130-km, the freestream Knudsen number based on spacecraft diameter is 4.8. The flow is in the transitional regime, since the drag coefficient is only 95 percent of the free-molecular value of 2.1. In addition, there is negligible dissociation occurring at this altitude; however, there is clear evidence of thermal diffusion (not shown) where the concentration of atomic oxygen (from the freestream) decreases near the surface and the concentration of the more massive oxygen and nitrogen molecules increases. As discussed in Ref. 47, thermal diffusion acts to concentrate the heavy gas in the cooler regions of the flow (adjacent to the surface). Calculations such as these are useful in defining the range of flow parameters for which measurements could be made. Furthermore, the potential flight measurments would be useful in validating the computational tools.

Transitional Flow about the AFE

A side view of the AFE vehicle is shown in Fig. 12. The aerobrake is an elliptically blunted elliptic cone raked off at the base and fitted with a skirt-type afterbody. The three-dimensional configuration has a base length of 4.25 m.

Figure 13 shows the computational grid used to simulate the 3-D flow for the 120-km-altitude case. In this figure, both cells and subcells are shown on the outer freestream boundary. (For the present three-dimensional application, the cells are deformed hexahedra, and each cell is further divided into tetrahedral subcells.) However, on the plane of symmetry, only the cell structure is drawn for clarity. Only the forebody and the experimental carrier are included in the calculation, since the solid rocket motor is ejected during entry near 130 km.

Reference 26 describes in some detail the highly nonequilibrium flow that surrounds the AFE vehicle at these high-altitude conditions (100 to 200 km) and the resulting surface pressure and heat transfer distributions. The results of this study show that dissociation is important at 110-km altitude and below (a five-species gas model was used) and that the flow approaches the free-molecular limit very gradually at higher altitudes. Even at 200 km, the flow is not completely collisionless. This is clearly evident in Fig. 14 where the lift-to-drag ratio is presented at selected altitudes for an angle of incidence of 0° (using the present coordinate system shown in Fig. 12). Figure 14 also shows the calculated free-molecule and modified Newtonian results, along with experimental wind-tunnel data. The experiments were conducted at the NASA Langley Research Center Mach 10 air and Mach 6 CF_4 (freon) wind tunnels using high-fidelity models.[48]

Clearly, the DSMC results approach the free-molecule limit very slowly at higher altitudes, and even at an altitude of 200 km, the flow is not completely collisionless. Prior to this study, it was generally acknowledged that free-molecule flow existed for the AFE vehicle for altitudes near the 150 km, but this study shows that the transitional effects are significant at these altitudes and influence the overall aerodynamic coefficients.

Figure 14 also contains the results of the Lockheed bridging formula which empirically connects the axial and normal aerodynamic force coefficients between the continuum and free-molecule limits. This is accomplished with a sine-square function by assuming continuum flow at a Knudsen number $Kn_\infty = 0.01$ and free-molecule flow at $Kn_\infty = 10$, which corresponds to altitudes of 90 and 150 km, respectively. The bridging formula results are plotted to show the general trend even though they are erroneous for the conditions considered in the present study.

Rarefied Flow about a Delta Wing

In the study of Ref. 49, a general three-dimensional (3-D) DSMC code is used to simulate a rarefied flow about a delta wing (Fig. 15). As shown in this figure, the top of the wing is flat, the bottom is V-shaped, and the edges are rounded with a constant radius, R = 0.0013 m. The shape of the nose from the side view is elliptical although it appears sharp from the top view. The origin of the coordinate system is located at the tip of the nose, and the x axis is parallel to the top surface and is normal to the base plane. The top surface is inclined 30° away from the freestream. The vertical midplane (at z = 0) is a plane of symmetry; thus computations have been made for half of the geometry. Figure 16(a) shows a perspective view for the 3-D computational grid used in this study, and Fig. 16(b) shows the grid structure at the aft end. The body-fitted grid has a total of 5,280 cells.

The flow simulated in this study is a wind-tunnel experiment in which the flowfield freestream conditions are $T_\infty = 13.32$ K, $V_\infty = 1503$ m/s, $M_\infty = 20.2$, and $\rho_\infty = 1.729 \times 10^{-5}$ kg/m^3. According to the VHS collision model (with $T_{ref} = 300$ K, $d_{ref} = 4.07 \times 10^{-10}$ m, and the temperature exponent of the viscosity coefficient of 0.75), the calculated freestream mean free path and viscosity are 0.00159 m and 1.9×10^{-6} N•s/m^2, respectively. Hence, the overall Knudsen number (based on the body length) is 0.0159, and the freestream Reynolds number per meter is 14,000. The body surface is specified to be at a uniform temperature of 620 K. Full thermal accommodation and diffuse reflection are assumed for the gas-surface interaction. Simulations are performed using a nonreacting gas model with one chemical species (N_2) while considering energy exchange between translational and internal (rotational and vibrational) modes.

The computations are performed for a total of 9,000 time steps. A stationary state was reached around 1,000 time steps, and after that, flowfield samples are taken every other time step. Hence, the time-averaged flowfield results presented in this study are based on sample sizes derived from 4,000 samplings.

Figure 17 presents the density flowfield contours along the symmetry plane [17(a)] and at a cross-sectional plane located at the 80-percent cord location [17(b)], and the same contour levels are shown in both parts. The flowfield results are obtained at the centroids of the cells and hence do not extend to the boundaries (one half cell from the boundaries). Accordingly, the flowfield contours for the symmetry plane are actually one half-cell from the symmetry plane. (See Ref. 49 for similar information regarding flowfield contours of Mach number, various temperatures, and surface quantities.)

This study represents one of the first applications of the 3-D DSMC method to a flow about a relatively sharp-nosed body. The computations indicate that the leeside flow is attached, and the results will be compared with the leeside density profiles and total body measurements (Allegre[50] and his co-workers at CNRS in France) when available. Further computations related to downstream effects are needed to have a better understanding of the overall flowfield structure.

Future Research Activities

The previous examples of application of the DSMC method to hypersonic external flow problems represent only a very limited portion of the wide spectrum of current applications. Examples of other applications are spacecraft contamination[51-52] resulting from the expansion of gases out of a rocket nozzle, the study and characterization of pumping devices,[53-54] and studies in materials processing[55] concerning thin-film vapor deposition. Because of the central importance of codes in predicting rarefied flows, future activities will focus on developing codes that are computationally more efficient and easier to use, improving on the existing physical modeling, and performing experiments that can be used to validate existing modeling or provide the data base essential for new models such as that needed for gas-surface interactions and energy exchange mechanisms.

Efforts by a number of researchers are currently being made to implement means of faster execution time while minimizing storage requirements. A major problem of the DSMC method has been the large amounts of computing time required for relatively simple problems. More attention has recently been focused on ways of reducing the computing time, including development of new algorithms that take advantage of current supercomputer architectures.[56-57] An approach that is currently being pursued by a number of researchers is the application of parallel processing.[58-59] Reductions of up to twelvefold using 16 processors[58] and up to sixteenfold using 32 processors[59] have been reported.

Most of the computational time required by existing DSMC codes is taken up by the analytical geometry associated with the description of complex flows. A new flow definition system[60] is under development that cuts this time by a factor of twenty. Consequently, the potential for major reduction in the execution time of the DSMC method is real, particularly with the synergism associated with improved algorithms, vectorization, and parallel processing.

In the area of physical model development, Refs. 61 and 62 are examples of recent studies concerned with the modeling of various physical processes (energy exchange between the translational and internal modes in Ref. 61 and the way in which electric field effects are modeled in Ref. 62). The needs for experimental work in rarefied gas dynamics as it relates to DSMC validation are reviewed by Muntz in Ref. 16.

Concluding Remarks

A review of the DSMC method with regard to what it is, the capability that it provides for analyzing transitional flows, applications that focus on hypersonic external flows, and current and future research activities has been presented. Major accomplishments have occurred in the past five years in developing the capability to predict highly nonequilibrium reacting flowfields along with the effects of ionization and radiation. Considerable efforts are now being focused on improving its computational efficiency and on code validation. The preliminary results of both activities are extremely encouraging as indicated in the present review. With more efficient codes, the range of applications and problem complexity will increase. Consequently, it is important that experiments be performed that provide information from which various aspects of the DSMC method can be validated.

References

[1]Bird, G. A., "Low-Density Aerodynamics," Progress in Astronautics and Aeronautics: Thermophysical Aspects of Re-entry Flows, edited by J. N. Moss and C. D. Scott, Vol. 103, 1986, pp. 3-24.

[2]Moss, J. N. and Bird, G. A., "Direct Simulation of Transitional Flow for Hypersonic Reentry Conditions," Progress in Astronautics and Aeronautics: Thermal Design of Aeroassisted Orbital Transfer Vehicles, edited by H. F. Nelson, Vol. 96, 1985, pp. 113-139.

[3]Bird, G. A., "Nonequilibrium Radiation During Re-entry at 10 km/s," AIAA Paper 87-1543, June 1987.

[4]Moss, J. N., Bird, G. A., and Dogra, V. K., "Nonequilibrium Thermal Radiation for an Aeroassist Flight Experiment Vehicle," AIAA Paper 88-0081, January 1988.

[5]Moss, J. N. and Price, J. M., "Direct Simulation of AFE Forebody and Wake Flow with Thermal Radiation," Progress in Astronautics and Aeronautics: Rarefied Gas Dynamics: Theoretical and Computational Techniques, edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 118, 1989, pp. 413-431.

[6]Chapman, D. R., Fiscko, K. A., and Lumpkin, F. E., "A Fundamental Problem in Computing Radiating Flow Fields with Thick Shock Waves," SPIE Proceeding on Sensing, Discrimination and Signal Processing, and Superconducting Materials and Instrumentation, Vol. 879, 1988, pp. 106-112.

[7]Cuda, V. and Moss, J. N., "Direct Simulation of Hypersonic Flows Over Blunt Wedges," AIAA Journal of Thermophysics and Heat Transfer, Vol. 1 April 1987, pp. 97-104.

[8]Moss, J. N. and Cuda, V., "Nonequilibrium Effects for Hypersonic Transitional Flows," AIAA Paper 87-0404, January 1987.

[9]Bird, G. A., "Direct Simulation of Typical AOTV Entry Flows," AIAA Paper 86-1310, June 1986.

[10]Allegre, J. Raffin, M., and Gottesdiener, L., "Slip Effects on Supersonic Flowfields Around NACA 0012 Airfoils," Proceedings of the 15th International Symposium on Rarefied Gas Dynamics, edited by V. Boff and C. Cercignani, Vol. 1, 1986, pp. 548-557.

[11]Hurlbut, F. C., "Sensitivity of Hypersonic Flow Over a Flat Plate to Wall/Gas Interaction Models Using DSMC," AIAA Paper 87-1545.

[12]Bird, G. A., Molecular Gas Dynamics, Clarendon Press, Oxford, 1976.

[13]Bird, G. A., "Monte Carlo Simulation of Gas Flows," Annual Reviews of Fluid Mechanics, Vol. 10, edited by M. D. Van Dyke, J. V. Wehausen, and J. L. Lumley, Annual Reviews Inc., Palo Alto, CA, 1979, p. 11

[14]Bird, G. A., "Monte Carlo Simulation in an Engineering Context," AIAA Progress in Astronautics and Aeronautics: Rarefied Gas Dynamics, Vol. 74, Part 1, edited by S. S. Fisher, AIAA, New York, 1981, pp.

[15]Bird, G. A., "Direct Simulation of Gas Flows at the Molecular Level," Communications in Applied Numerical Methods, Vol. 4, 1988, pp. 165-172.

[16]Muntz, E. P., "Rarefied Gas Dynamics," Annual Reviews of Fluid Mechanics, Vol. 21, edited by J. L. Lumley, M. D. Van Dyke, and H. L. Reed, 1989, pp. 387-417.

[17]Bird, G. A., "Perception of Numerical Methods in Rarefied Gasdynamics," Progress in Astronautics and Aeronautics: Rarefied Gas Dynamics: Theoretical and Computational Techniques, edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 118, 1989, pp. 221-226.

[18]Alder, B. J., and Wainwright, T. E., "Molecular Dynamics by Electronic Computer," Transport Processes in Statistical Mechanics, edited by I. Prigogine, Interscience, New York, 1958, pp. 97-131.

[19]Bird, G. A., "Direct Simulation and the Boltzmann Equation," The Physics of Fluids, Vol. 13, No. 11, Nov. 1970, pp.2676-2681.

[20]Nanbu, K., "Theoretical Basis of the Direct Simulation Monte Carlo Method," Proceedings of the 15th International Symposium on Rarefied Gas Dynamics, edited by V. Boffi and C. Cercignani, Vol. 1, 1986, pp. 369-383.

[21]Bird, G. A., "Direct Molecular Simulation of a Dissociating Diatomic Gas," Journal of Computational Physics, Vol. 25, Dec. 1977, pp. 353-365.

[22]Harvey, J. K., "Inelastic Collision Models for Monte Carlo Simulation Computation," Progress in Astronautics and Aeronautics: Rarefied Gas Dynamics: Theoretical and Computational Techniques, edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 117, 1989, pp. 3-24.

[23]Blanchard, R. C. and Larman, K. T., "Rarefied Aerodynamics and Upper Atmospheric Flight Results from the Orbiter High Resolution Accelerometer Package Experiment," AIAA Paper 87-2366, 1987.

[24]Dogra, V. K., Moss, J. N., and Price, J. M., "Rarefied Flow Past a Flat Plate at Incidence," Progress in Astronautics and Aeronautics: Rarefied Gas Dynamics: Theoretical and Computational Techniques, edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 118, 1989, pp. 567-581.

[25]Dogra, V. K. Moss, J. N., "Hypersonic Rarefied Flow About Plates at Incidence," AIAA Paper 89-1712, June 1989.

[26]Celenligil, M. C., Moss, J. N., and Blanchard, R. C., "Three-Dimensional Flow Simulation about the AFE Vehicle in the Transitional Regime," AIAA Paper 89-0245, January 1989.

[27]Harvey, J. K., "Direct Simulation Monte Carlo Method and Comparison with Experiment," Progress in Astronautics and Aeronautics: Thermophysical Aspects of Re-Entry Flows, edited by J. N. Moss and C. D. Scott, Vol. 103, 1986, pp. 25-43.

[28]Harvey, J. K., Celenligil, M. C., Dominy, R. G., and Gilmore, M. R., "A Flat-Ended Circular Cylinder in Hypersonic Rarefied Flow," AIAA Paper 89-1709, June 1989.

[29]Taylor, J. C., Moss, J. N., and Hassan, H. A., "Study of Hypersonic Flow Past Sharp Cones," AIAA Paper 89-1713, June 1989.

[30]Fiscko, K. A., and Chapman, D. R., "Comparison of Burnett, Super-Burnett, and Monte Carlo Solutions for Hypersonic Shock Structure," Progress in Astronautics and Aeronautics: Rarefied Gas Dynamics: Theoretical and Computational Techniques, edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 117, 1989, pp. 374-395.

[31]Pham-Van-Diep, G. C. and Erwin, D. A., "Validation of MCDS by Comparison of Predicted with Experimental Velocity Distribution Functions in Rarefied Normal Shocks," Progress in Astronautics and Aeronautics: Rarefied Gas Dynamics: Theoretical and Computational Techniques, edited by E.P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 118, 1989, pp. 271-283.

[32]Erwin, D. A., Muntz, E. P., and Pham-Van-Diep, G., "A Review of Detailed Comparisons Between Experiments and DSMC Calculations in Nonequilibrium Flows," AIAA Paper 89-1883, June 1989.

[33]Pham-Van-Diep, G., Erwin, D., and Muntz, E. P., "Nonequilibrium Molecular Motion in a Hypersonic Shock Wave," Science, Vol. 245, August 1989, pp. 624-626.

[34]Alsmeyer, H., "Density Profiles in Argon and Nitrogen Shock Waves Measured by the Absorption of an Electron Beam," Journal of Fluid Mechanics, Vol. 74, 1976, pp. 497-513.

[35]Maitland, G. C., Rigby, M., Smith, E. B., and Wakeham, W., "Intermolecular Forces," Clarendon Press, Oxford, 1981.

[36]Bird, G. A., "Computation of Electron Density in High Altitude Re-Entry Flows," AIAA Paper 89-1882, June 1989.

[37]Cheng, H. K., Lee, C. J., Wong, E. Y., and Yang, H. T., "Hypersonic Slip Flows and Issues on Extending Continuum Model Beyond the Navier-Stokes Level," AIAA Paper 89-1663, June 1989.

[38]Gokcen, T. and MacCormack, R., "Nonequilibrium Effects for Hypersonic Transitional Flows Using Continuum Approach," AIAA Paper 89-0461, Jan. 1989.

[39]Chrusciel, G. T. and Pool, L. A., "Knudsen-Layer Properties for a Conical Afterbody in Rarefied Hypersonic Flows," Progress in Astronautics and Aeronautics: Rarefied Gas Dynamics: Theoretical and Computational Techniques, edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 118, 1989, pp. 464-475.

[40]Gupta, R. N. and Simmonds, A. L., "Hypersonic Low-Density Solutions of the Navier-Stokes Equations with Chemical Nonequilibrium and Multicomponent Surface Slip," AIAA Paper 86-1349, June 1986.

[41]Gnoffo, P. A., Gupta, R. N., and Shinn, J. L., "Conservation Equations and Physical Models for Hypersonic Air Flows in Thermal and Chemical Nonequilibrium," NASA TP-2867, February 1989.

[42]Blanchard, R. C., "Rarefied Flow Lift-to-Drag Measurements of the Shuttle Orbiter," 15th Congress of International Council of Aeronautical Sciences, Paper ICAS-86-2.10.1, London, England, September 7-12, 1986.

[43]Blanchard, R. C., Hendrix, M. K., Fox, J. C., Thomas, D. J., and Nicholson, J. Y., "Orbital Acceleration Reserch Experiment," Journal of Spacecraft and Rockets, Vol. 24, No. 6, November-December, 1987.

[44]Aerodynamic Design Data Book, Volume 1, Orbiter Vehicle 102, SD72-SH-0060, Volume IM, November 1980, Space Division, Rockwell International.

[45]Blanchard, R. C. and Rutherford, J. F., "Shuttle Orbiter High Resolution Accelerometer Package Experiment: Preliminary Flight Results," Journal of Spacecraft and Rockets, Vol. 22, No. 4, July-August 1985, pp. 474-480.

[46]Wood, G. M., Wilmoth, R. G., Carlomagno, G. M., and deLuca, L., "Proposed Aerothermodynamic Experiments in Transition Flow Using the NASA/ASI Tethered Satellite System-2, AIAA 90-0536, January 1990.

[47]Bird, G. A., "Thermal and Pressure Diffusion Effects in High Altitude Flows," AIAA Paper 88-2732, June 1988.

[48]Wells, W. L., "Wind Tunnel Preflight Test Program for Aeroassist Flight Experiment," AIAA Paper 87-2367-CP, August 1987.

[49]Celenligil, M. C. and Moss, J. N., "Direct Simulation of Hypersonic Rarefied Flow About a Delta Wing," AIAA Paper 90-0143, January 1990.

[50]Allegre, J., Private Communications, Laboratoire d'Aerothermique de CNRS, 4 ter Route des Gardes, F92190 Meudon, France.

[51]Hueser, J. E., Melfi, L. T., Bird, G. A., and Brock, F. J., "Rocket Nozzle Lip Flow by Direct Simulation Monte Carlo," AIAA Journal of Spacecraft and Rockets, Vol. 23, July-August 1986, pp. 363-367.

[52]Bird, G. A., "Influence of Local Configuration on the Back Flow from Small Thruster Rockets," AIAA Paper 90-0147, Janaury 1990.

[53]Usami, M., Fujimoto, T., and Kato, S., "Monte Carlo Simulation on Mass Flow Reduction due to Roughness of a Slit Surface," Progress in Astronautics and Aeronautics: Rarefied Gas Dynamics: Space Related Studies, edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 116, 1989, pp. 283-297.

[54]Nanbu, K., Watanabe, Y., Igarashi, S., Dettleff, G., and Koppenwallner, G., "Effectiveness of a Parallel Plate Arrangement as a Cryogenic Pumping Device," Progress in Astronautics and Aeronautics Rarefied Gas Dynamics: Space-Related Studies, edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 116, 1989, pp. 283-297.

[55]Watanabe, Y., Nanbu, K., and Igarashi, S., "Angular Distributions of Molecular Flux Effusing from a Cylindrical Crucible Partially Filled with Liquid," Progress in Astronautics and Aeronautics: Rarefied Gas Dynamics: Space-Related Studies, edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 116, 1989, pp. 283-297.

[56]McDonald, J. D. and Baganoff, D., "Vectorization of a Particle Simulation Method for Hypersonic Rarefied Flow," AIAA Paper 88-2735, June 1988.

[57]Feiereisen, W. J. and McDonald, J. D., "Three-Dimensional Discrete Particle Simulation of an AOTV," AIAA Paper 89-1711, June 1989.

[58]Wilmoth, R. G., "Direct Simulation Monte Carlo Analysis on Parallel Processors," AIAA Paper 89-1666, June 1989.

[59]Furlani, T. R. and Lordi, J. A., " A Comparison of Parallel Algorithm for the Direct Simulation Monte Carlo Method II: Applications to Exhaust Plume Flowfields," AIAA Paper 89-1167, June 1989

[60]Bird, G. A., "Private communications, Department of Aeronautical Engineering, J07, University of Sydney, N.S.W., 2006, Australia.

[61]Boyd, I. D., "Direct Simulation of Rotational and Nonequilibrium," AIAA Paper 89-1880, June 1989.

[62]Carlson, A. B. and Hassan, H. A., "Direct Simulation of Reentry Flows with Ionization," AIAA Paper 90-0144, January 1990.

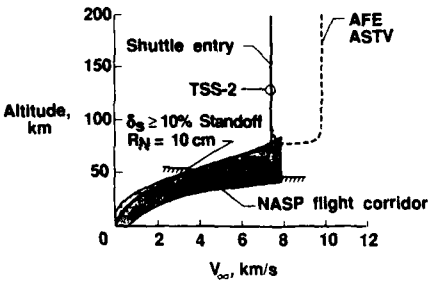


Fig. 1 Hypersonic flow environment.

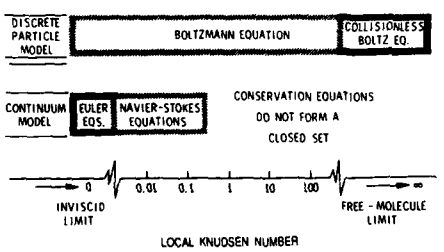


Fig. 2 Local Knudsen number limits on math models.

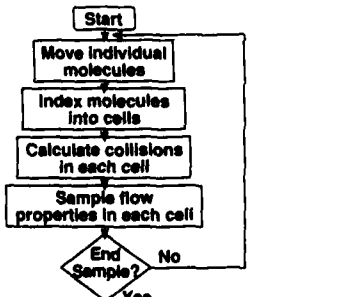
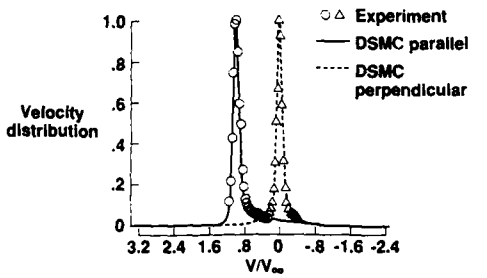
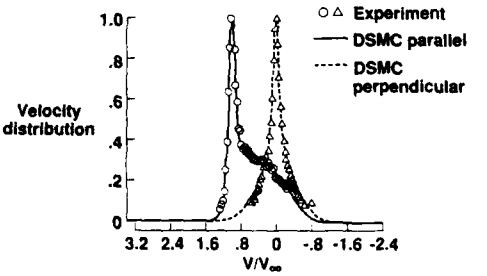


Fig. 3 Flow chart for DSMC method.



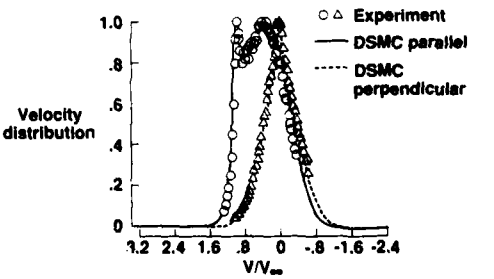
(a) $\tilde{n}$ = 0.065.

Fig. 4 Comparison of molecular velocity distributions[33] (normal shock, Mach 25, helium).



(b) $\tilde{n}$ = 0.285.

Fig. 4 Continued



(c) $\tilde{n}$ = 0.565.

Fig. 4 Concluded.

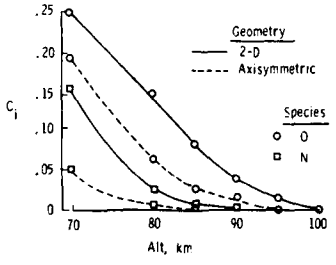


Fig. 5 Calculated maximum atomic mass fractions along stagnation streamline (V_∞ = 7.5 km/s and R_N = 2.54 cm).

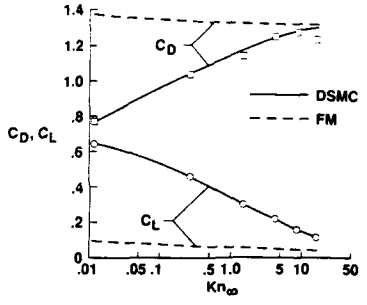


Fig. 8 Calculated lift and drag coefficients for a flat plate (V_∞ = 7.5 km/s, α = 40°).

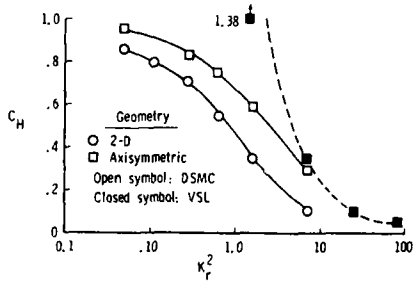


Fig. 6 Stagnation heat-transfer coefficient versus Cheng's rarefaction parameter.

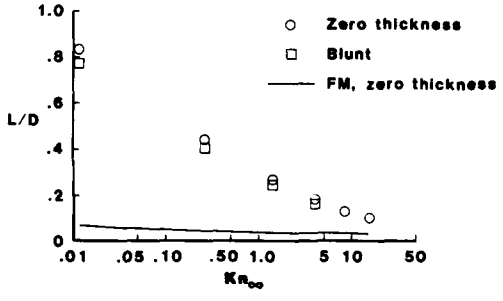


Fig. 9 Effect of rarefaction on aerodynamic characteristics (V_∞ = 7.5 km/s, α = 40°)

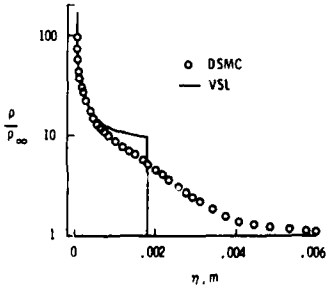


Fig. 7 Calculated stagnation streamline density (Alt = 70 km, V_∞ = 7.5 km/s, R_N = 2.54 cm, Axisy.).

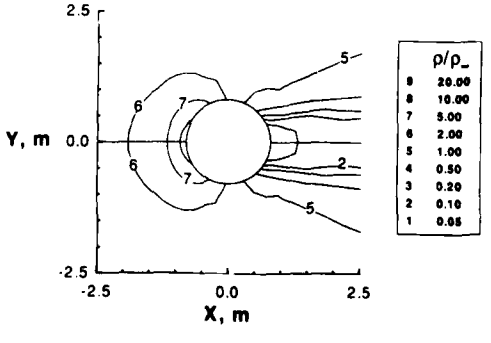


Fig. 10 Calculated density contours about a 1.6-m-diameter sphere (Alt = 130 km, V_∞ = 7.5 km/s, $\rho_\infty = 7.6\times10^{-9}$ kg/m^3).

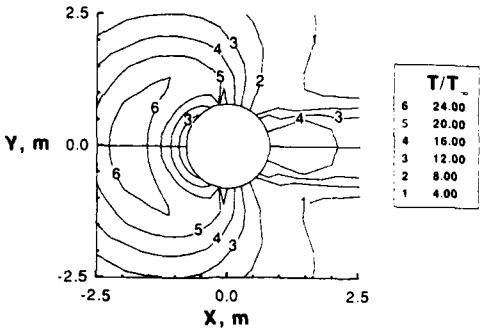


Fig. 11 Calculated temperature contours about a 1.6-m-diameter sphere (Alt = 130 km, V_∞ = 7.5 km/s, T_∞ = 432 K).

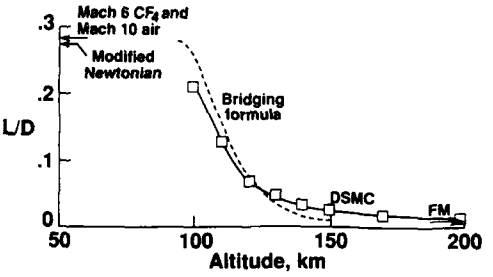


Fig. 14 Lift-to-drag ratio variation with altitude for AFE entry conditions.

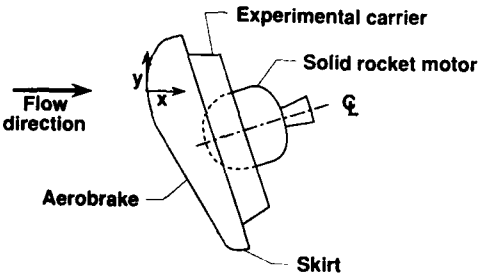


Fig. 12 The AFE vehicle.

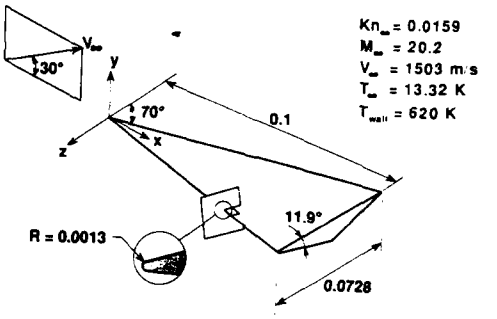


Fig. 15 Schematic of the delta wing (dimensions in meters).

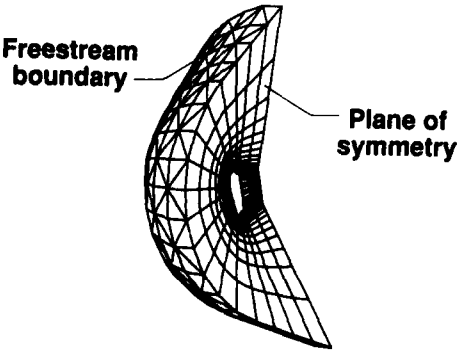
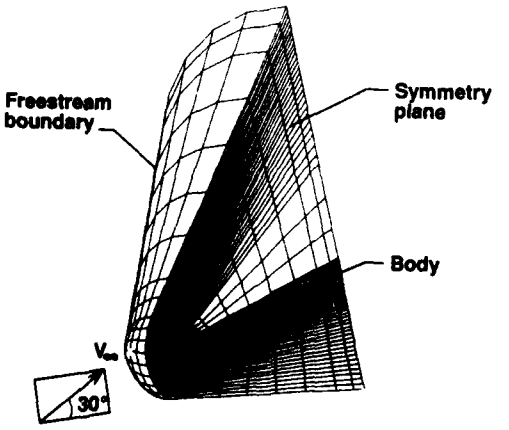
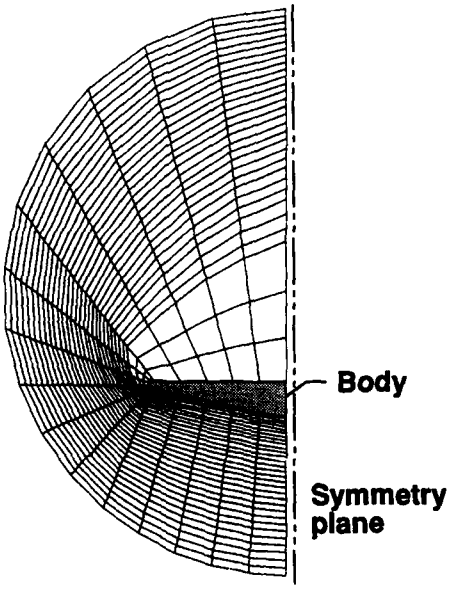


Fig. 13 Computational grid (Alt = 120 km).



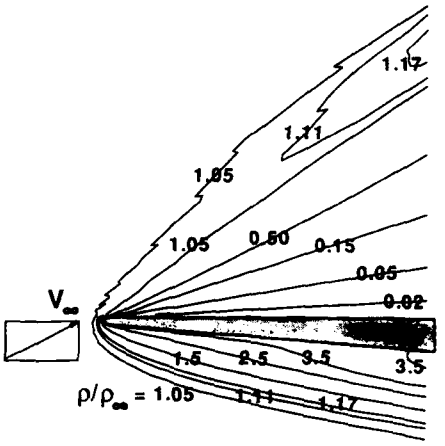
(a) Perspective view.

Fig. 16 Three-dimensional computational grid.



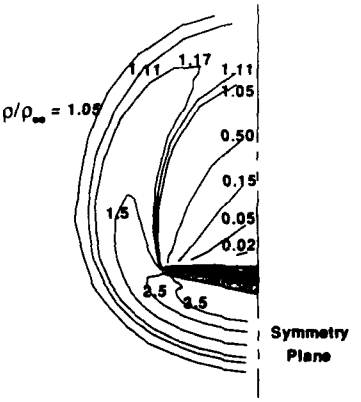
(b) Cross section at aft end.

Fig. 16 Concluded.



(a) Near the symmetry plane.

Fig. 17 Flowfield density contours (Nitrogen, V_∞ = 1.5 km/s, ρ_∞ = 1.7x10^{-5} kg/m^3).



(b) At cross section x = 0.08 m.

Fig. 17 Concluded.

RECENT PROGRESS IN REACTIVE FLOW COMPUTATIONS

B. LARROUTUROU
CERMICS, INRIA, Sophia-Antipolis, 06560 VALBONNE, FRANCE

1. INTRODUCTION

We report in this paper on some recent progress concerning a family of approximation schemes for the numerical simulation of a multi-dimensional flow of a reactive perfect or real gaseous mixture.

All these schemes basically employ a second-order accurate *upwind* approximation, using slope limiters in order to give an oscillation-free solution. Moreover, they operate on any (and in particular on a possibly unstructured) finite-element mesh. Thus, these schemes generalize for multi component flows the finite–element upwind schemes developed for a single gas by A. Dervieux and L. Fezoui [10], [15], in the spirit of the MUSCL (Monotonic Upwind Schemes for Conservation Laws) methodology of Van Leer [40].

The schemes improvements discussed below are in particular related to the coupling of the mass fractions equations with the basic hydrodynamic equations. In particular, we derive a family of schemes which have the property of preserving the maximum principle (and in particular the positivity) for the mass fractions of all species in the gaseous mixture.

The way this coupling between the mass fractions equations and the gas dynamics equations is actually taken into account in the discrete approximation is based on the study of a generalized Riemann problem for one-dimensional multi–component gas dynamics; this Riemann problem is discussed in Section 2. Then Section 3 is devoted to the discussion of the basic multi-component approximation schemes in one space dimension, while the extension to multi-dimensional flows is described in Section 4.

These schemes have revealed to provide robust and accurate solutions of many reactive flows at various regimes, ranging from highly subsonic (Mach number of the order of 10^{-3}) to hypersonic (Mach number of the order of 20) reactive flows, in simple or complex geometries, and in two or three space dimensions. As an illustration we present and discuss several numerical examples in Section 5.

2. THE MULTI-COMPONENT EULER EQUATIONS

2.1. The equations

As a first step, we consider the one-dimensional inviscid flow of a non-reactive mixture of N species Σ_1, Σ_2 ... Σ_N. The governing equations for this flow express the conservation of momentum and of total energy and the conservation of mass for each component. They take the form (see e.g. [43]):

$$\begin{cases} (\rho u)_t + (\rho u^2 + p)_x = 0 \ , \\ \mathcal{E}_t + [u(\mathcal{E} + p)]_x = 0 \ , \\ (\rho Y_k)_t + (\rho u Y_k)_x = 0 \quad \text{for} \ \ 1 \leq k \leq N \ , \end{cases} \tag{2.1}$$

where ρ is the mixture density, u is the mixture velocity (which is also the velocity of each species, since we neglect here molecular diffusion), p is the total pressure in the mixture, $\mathcal{E}$ is the total energy per unit volume and Y_k is the mass fraction of species Σ_k (that is, ρY_k is the separate density of species Σ_k, and $\sum_{k=1}^{N} Y_k = 1$).

For the sake of simplicity, we first assume that each species Σ_k obeys the perfect gas laws, and in particular has constant specific heats at constant volume and pressure C_{vk} and C_{pk}. We will also denote γ_k the ratio $\gamma_k = \dfrac{C_{pk}}{C_{vk}}$, and M_k the molecular weight of species Σ_k. The total pressure p is then given by Dalton's law as:

$$p = \sum_{k=1}^{N} \rho Y_k \frac{R}{M_k} T \ , \tag{2.2}$$

where R is the universal gas constant and T is the temperature of the mixture (the same for all species). Considering that the N species may have different specific heats of formation h_k^0 , we write the total energy $\mathcal{E}$ as (see e.g. [5], [22], [43]):

$$\mathcal{E} = \sum_{k=1}^{N} \left(\frac{1}{2} \rho Y_k u^2 + \rho Y_k C_{vk} T + \rho Y_k h_k^0 \right) \ . \tag{2.3}$$

Since the temperature does not appear in the conservation relations (2.1), we can eliminate it in (2.2)-(2.3) and consider that, in (2.1), the pressure p is given by the following relation, which is deduced from (2.2)-(2.3) and Mayer's relation $M_k(C_{pk} - C_{vk}) = R$:

$$p = (\gamma - 1) \left(\mathcal{E} - \frac{1}{2} \rho u^2 - \sum_{k=1}^{N} \rho Y_k h_k^0 \right) \ . \tag{2.4}$$

Here, γ is the local ratio of the specific heats of the mixture:

$$\gamma = \frac{(C_p)_{mixture}}{(C_v)_{mixture}} = \frac{\sum_k Y_k C_{pk}}{\sum_k Y_k C_{vk}} = \frac{\sum_k Y_k C_{vk}\gamma_k}{\sum_k Y_k C_{vk}} . \tag{2.5}$$

Remark 1: In fact, several of the numerical methods presented below also apply to mixtures of real gases. We will not consider this case in detail below, referring to e.g. [12], [18], and in particular to [23] where a general presentation is given for mixtures where the relation giving the total pressure has the form:

$$p = p\left(\mathcal{E} - \frac{1}{2}\rho u^2, \rho Y_k\right) , \tag{2.6}$$

which is in particular the case if all components in the mixture satisfy Boyle's or Mariotte's law (see [23]). •

2.2. The Riemann problem

For the sake of simplicity, we will from now on restrict our attention to the case of a mixture made of only two species Σ_1 and Σ_2; but all results presented below can be straightforwardly extended to mixtures consisting of any number of components N.

Thus, we rewrite (2.1) as:

$$\begin{pmatrix} \rho u \\ \mathcal{E} \\ \rho Y_1 \\ \rho Y_2 \end{pmatrix}_t + \begin{pmatrix} \rho u^2 + p \\ u(\mathcal{E} + p) \\ \rho u Y_1 \\ \rho u Y_2 \end{pmatrix}_x = 0 . \tag{2.7}$$

System (2.7) can be rewritten in a simpler equivalent form. Simply denoting Y the mass fraction Y_1 of the first species and E the sum of the kinetic and thermal energy per unit volume:

$$E = \mathcal{E} - \sum_{k=1}^{2} \rho Y_k h_k^0 , \tag{2.8}$$

we get:

$$\begin{pmatrix} \rho \\ \rho u \\ E \\ \rho Y \end{pmatrix}_t + \begin{pmatrix} \rho u \\ \rho u^2 + p \\ u(E + p) \\ \rho u Y \end{pmatrix}_x = 0 , \tag{2.9}$$

where the first equation expresses the conservation of mass for the mixture and has the usual form of the continuity equation for a single fluid, and where:

$$p = (\gamma - 1)\left(E - \frac{1}{2}\rho u^2\right) , \tag{2.10}$$

with:

$$\gamma = \frac{YC_{v1}\gamma_1 + (1-Y)C_{v2}\gamma_2}{YC_{v1} + (1-Y)C_{v2}} . \tag{2.11}$$

We will use the classical notations W and F for the vectors of the conservative variables and of the fluxes:

$$W = \begin{pmatrix} \rho \\ \rho u \\ E \\ \rho Y \end{pmatrix} = \begin{pmatrix} W^1 \\ W^2 \\ W^3 \\ W^4 \end{pmatrix} , \quad F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ u(E+p) \\ \rho u Y \end{pmatrix} = \begin{pmatrix} F^1 \\ F^2 \\ F^3 \\ F^4 \end{pmatrix} . \tag{2.12}$$

Then, we have the two following simple results (which are shown in e.g. [1], [2], [23]):

Proposition 1:

The flux vector F is an homogeneous function of degree 1 of W. •

Proposition 2:

If the specific heat ratio γ_k of each species in the mixture satisfies the inequality:

$$\gamma_k > 1 , \tag{2.13}$$

then the system (2.9) is hyperbolic. •

The proof of these results is straightforward. Proposition 1 simply follows from the observation that $\gamma = \gamma(W)$ is homogeneous of degree 0. And the assumption (2.13) is needed in order to insure that $\gamma(W) > 1$ for any W since the last equality in (2.5) shows that the local value of γ is a linear convex combination of the γ_k's.

Let us simply make precise here that the eigenvalues of the 4×4 Jacobian matrix $A(W) = \dfrac{DF}{DW}$ are:

$$\lambda_1 = u - c , \quad \lambda_2 = u , \quad \lambda_3 = u , \quad \lambda_4 = u + c , \tag{2.14}$$

where the sound speed c has the usual expression:

$$c = \sqrt{\frac{\gamma p}{\rho}} , \tag{2.15}$$

but with the local value (2.11) of γ.

Remark 2: The developed expression of the matrix $A(W)$ will be useful in the sequel. A straightforward calculation shows that:

$$A(W) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{(\gamma-3)}{2}u^2 - YX & (3-\gamma)u & \gamma-1 & X \\ \frac{(\gamma-1)}{2}u^3 - uH - uYX & H-(\gamma-1)u^2 & \gamma u & uX \\ -uY & Y & 0 & u \end{pmatrix} , \tag{2.16}$$

where $H = \dfrac{E+p}{\rho}$ is the specific enthalpy of the mixture, and where:

$$X = \frac{p}{\gamma-1}\,\frac{C_{v1}C_{v2}(\gamma_1-\gamma_2)}{\rho[YC_{v1}+(1-Y)C_{v2}]^2} = \frac{C_{v1}C_{v2}(\gamma_1-\gamma_2)T}{YC_{v1}+(1-Y)C_{v2}} \; . \bullet \tag{2.17}$$

Remark 3: It is shown in [23] that, for gases obeying Boyle's law, Proposition 1 still holds; that is, the pressure p in (2.6) is then an homogeneous function of degree 1. •

Remark 4: Proposition 2 can also be extended to real gas mixtures. For a two-component mixture, setting $\rho' = \rho Y$ and writing (2.6) under the form:

$$p = p\left(E - \frac{1}{2}\rho u^2, \rho, \rho'\right) = p(\epsilon, \rho, \rho') \; , \tag{2.18}$$

one can show that the system (2.9) is hyperbolic if the quantity

$$c^2 = p_\rho + Y p_{\rho'} + p_\epsilon \frac{\epsilon}{\rho} + p_\epsilon \frac{p}{\rho} \tag{2.19}$$

is always positive. The eigenvalues of the Jacobian matrix $A(W)$ are then again given by (2.14), with the sound speed c given by (2.19). Moreover, a particularly nice simplification arises when the homogeneity property holds (see Remark 3): then, the sound speed again has its usual expression

$$c = \sqrt{\frac{\gamma p}{\rho}} \; , \tag{2.20}$$

the definition of γ being extended from perfect-gas mixtures to real-gas mixtures by the following relation (which uses the partial derivative of (2.18)):

$$\gamma = p_\epsilon + 1 \; . \tag{2.21}$$

We refer to [23] for the details. •

Since system (2.9) is hyperbolic, it makes now sense to examine the Riemann problem for this system. Introducing two states W_L and W_R, we consider the problem:

$$\begin{cases} W_t + F(W)_x = 0 \quad \text{for } x \in I\!R\,, \quad t \geq 0\,, \\ W(x,0) = \begin{cases} W_L & \text{if } x < 0\,, \\ W_R & \text{if } x > 0\,. \end{cases} \end{cases} \tag{2.22}$$

When trying to solve this problem, a first important question concerns the genuine nonlinearity or the degeneracy of the characteristic fields (see [24]). As in the single-component case, the answer is here that the first and fourth characteristic fields are genuinely non linear, whereas the characteristic fields associated with the eigenvalue u are linearly degenerate (see [23]). Exactly as in the case of the single-component Euler equations, it is then possible, by analysing the shock or rarefaction waves associated with the non linear characteristic fields to completely solve the Riemann problem (2.22) for any left and right state W_L and W_R. Referring to [1], [2], [23] for the details, we simply describe here the structure of the solution of (2.22).

This exact solution $W^{\mathcal{R}}$ is of course self-similar (i.e. $W^{\mathcal{R}}(x,t)$ only depends on the ratio $\frac{x}{t}$), and consists, as in the single-component case, of four constant states $W_{(1)}$, $W_{(2)}$, $W_{(3)}$, $W_{(4)}$ separated by shocks, rarefaction waves or a contact discontinuity. More precisely, as shown on Figure 1, $W_{(1)} = W_L$ and $W_{(2)}$ are separated by a wave associated with the first characteristic field, that is with the eigenvalue $\lambda_1 = u - c$, either a 1-shock or a 1-rarefaction wave; $W_{(2)}$ and $W_{(3)}$ are separated by a contact discontinuity (associated with the eigenvalue u); $W_{(3)}$ and $W_{(4)} = W_R$ are separated by a 4-wave, associated with $\lambda_4 = u + c$. Also, the pressure p and the velocity u are continuous across the contact discontinuity. Last but not least, the mass fraction Y remains constant across the 1-wave and the 4-wave (whatever these waves are, shocks or rarefactions) and only varies accross the contact discontinuity. This fact has important consequences. Indeed, γ is constant on each side of the contact discontinuity. Thus, on the left side of the discontinuity the mixture has the composition of the state W_L, and behaves as a single perfect gas whose specific heat ratio is $\gamma_L = \gamma(W_L)$. Analogous conclusions hold for the right side of the contact discontinuity.

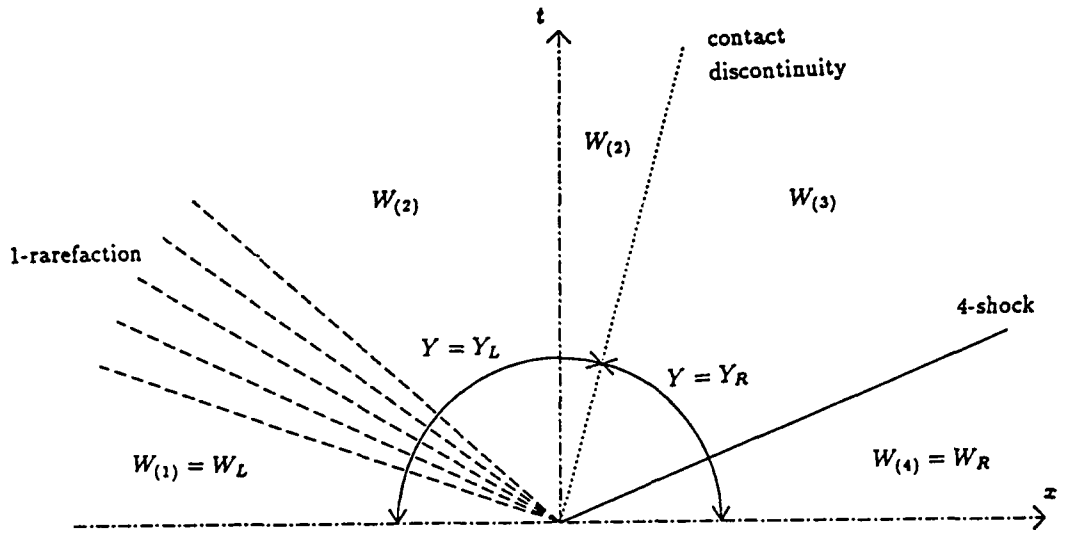


Figure 1: The solution of the multi-component Riemann problem (2.22).

For $\sigma \in \mathbb{R}$ and $t \geq 0$, we will denote $\mathcal{W}(\sigma; W_L, W_R)$ (or sometimes simply $\mathcal{W}(\sigma)$) the value of $W^{\mathcal{R}}(\sigma t, t)$, which is independent of t. We will need below the following property of $\mathcal{W}$:

Proposition 3:

For any states W_L and W_R, the following equality holds:

$$F^4[\mathcal{W}(0)] = F^1[\mathcal{W}(0)] \times \begin{cases} Y_L & \text{if } F^1[\mathcal{W}(0)] > 0 \,, \\ Y_R & \text{if } F^1[\mathcal{W}(0)] < 0 \,. \end{cases} \bullet \tag{2.23}$$

Proposition 3 is proved in [20], [21]. The proof essentially consists in showing that $F^1[\mathcal{W}(0)]$ and the speed of the contact discontinuity have the same sign.

Remark 5: Solving the Riemann problem (2.22) for real-gas mixtures, with the equation of state (2.18), is more difficult. If some convexity property is assumed for the pressure law (2.18), one can again show that the first and fourth characteristic fields are genuinely non linear, and the Riemann problem can be solved exactly (see [9], [31]). Without such an assumption, the genuine non linearity may fail, and there exists for the moment no general procedure to solve (2.22). In all cases, one can show that the characteristic field remains linearly degenerate, and that, in the exact solution of (2.22),

the mass fraction Y remains constant everywhere except at the contact discontinuity associated with the eigenvalue u (see [23]). •

3. ONE-DIMENSIONAL MULTI-COMPONENT UPWIND SCHEMES

Let us now consider the numerical solution of an initial value problem associated with system (2.9):

$$\begin{cases} W_t + F(W)_x = 0 & \text{for } x \in I\!R\,, \quad t \geq 0\,, \\ W(x,0) = W^0(x) & \text{for } x \in I\!R\,. \end{cases} \tag{3.1}$$

We will restrict our attention in this section to explicit, three-point, first-order accurate schemes written in conservative form. In other words, using very classical notations, we consider numerical schemes of the form:

$$\frac{W_j^{n+1} - W_j^n}{\Delta t} + \frac{\phi_{j+1/2} - \phi_{j-1/2}}{\Delta x} = 0\,, \tag{3.2}$$

where the numerical flux $\phi_{j+1/2}$ is evaluated using a "numerical flux function" Φ:

$$\phi_{j+1/2} = \Phi(W_j^n, W_{j+1}^n)\,. \tag{3.3}$$

(we write $\phi_{j+1/2}$ instead of $\phi_{j+1/2}^n$ for simplicity).

The knowledge of the numerical flux function Φ defines the scheme under consideration. Below, we will consider two types of flux functions, based on approximate Riemann solvers (also known as Godunov-type schemes) and on flux-vector splitting techniques.

3.1. Multi-component approximate Riemann solvers

In this section, we will consider two well-known existing approximate Riemann solvers which have been developed for the single-component Euler equations, namely the Roe and Osher approximate Riemann solvers, and present for each of them two different generalizations to multi-component problems, which we will refer to as the fully-coupled and the weakly-coupled generalizations. We will conclude the section by discussing the relative merits and drawbacks of both extensions.

3.1.1. The fully-coupled multi-component Roe scheme

The extension of Roe's scheme [29] to the two-component system (3.1) has been derived in [1] and [10]. The two-component numerical flux function has the form:

$$\Phi(W_L, W_R) = \frac{F(W_L) + F(W_R)}{2} + \frac{1}{2} \mid \tilde{A} \mid (W_L - W_R) \,, \tag{3.4}$$

where $\tilde{A} = \tilde{A}(W_L, W_R)$ is a diagonalisable matrix satisfying Roe's property:

$$F(W_L) - F(W_R) = \tilde{A}(W_L - W_R) \,, \tag{3.5}$$

and where the matrix $|\tilde{A}|$ is defined as follows: the diagonalisation of $\tilde{A}$ being written as $\tilde{A} = T\Lambda T^{-1}$ with $\Lambda = Diag[\mu_1, \mu_2 \cdots \mu_n]$, we set $|\tilde{A}| = T|\Lambda|T^{-1}$ with $|\Lambda| = Diag[|\mu_1|, |\mu_2| \cdots |\mu_n|]$.

To construct this matrix $\tilde{A}$, one introduces the state $\tilde{W} = (\tilde{\rho}, \tilde{\rho}\tilde{u}, \tilde{E}, \tilde{\rho}\tilde{Y})^T$, known as "Roe's average of W_L and W_R"; this state is defined by the relations:

$$\tilde{u} = \frac{u_L\sqrt{\rho_L} + u_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \,, \quad \tilde{H} = \frac{H_L\sqrt{\rho_L} + H_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \,, \tag{3.6}$$

as in the single-component case, and:

$$\tilde{Y} = \frac{Y_L\sqrt{\rho_L} + Y_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \,, \tag{3.7}$$

(determining $\tilde{\rho}$ is not necessary). But in this two-component context, unless both species in the mixture have the same specific heat ratio $\gamma_1 = \gamma_2$ (that is unless $\gamma = \gamma(W)$ is a constant), the flux Jacobian matrix $A(\tilde{W})$ evaluated at this Roe-averaged state $\tilde{W}$ does not satisfy property (3.5). Therefore, the matrix $\tilde{A}$ is to be chosen different from $A(\tilde{W})$ (but close to the latter since we want the extension to reduce to the usual Roe scheme when both species are the same). The result given in [1], [10] is the following:

$$\tilde{A} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \dfrac{(\tilde{\gamma}-3)}{2}\tilde{u}^2 - \tilde{Y}\tilde{X} & (3-\tilde{\gamma})\tilde{u} & \tilde{\gamma}-1 & \tilde{X} \\ -\tilde{u}\tilde{H} + \dfrac{(\tilde{\gamma}-1)}{2}\tilde{u}^3 - \tilde{u}\tilde{Y}\tilde{X} & \tilde{H} - (\tilde{\gamma}-1)\tilde{u}^2 & \tilde{\gamma}\tilde{u} & \tilde{u}\tilde{X} \\ -\tilde{u}\tilde{Y} & \tilde{Y} & 0 & \tilde{u} \end{pmatrix} , \tag{3.8}$$

where $\tilde{\gamma} = \gamma(\tilde{W})$, but where $\tilde{X}$ is not equal to $X(\tilde{W})$ given by (2.17): in order to insure that (3.5) holds, one has to choose (we refer to [1] for the detailed algebra):

$$\tilde{X} = \frac{C_{v1}C_{v2}(\gamma_1 - \gamma_2)\tilde{T}}{\tilde{Y}C_{v1} + (1-\tilde{Y})C_{v2}} , \tag{3.9}$$

with:

$$\tilde{T} = \frac{T_L\sqrt{\rho_L} + T_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \neq T(\tilde{W}) . \tag{3.10}$$

3.1.2. The fully-coupled multi-component Osher scheme

The extension of Osher's scheme [28] to multi-component flows is straightforward (exactly in the same way as we have seen in Section 2.2 that the extension of exact Riemann solvers is straightforward), and has been done by Abgrall and Montagné [2]. The extended scheme is defined by the numerical flux function:

$$\Phi(W_L, W_R) = \frac{F(W_L) + F(W_R)}{2} + \int_{W_L}^{W_R} | A(W) | dW , \tag{3.11}$$

where the integration is carried out on a path connecting W_L and W_R in the state-space. As in the single-component case, the integration path is piecewise parallel to the right eigenvectors of the flux Jacobian matrix A, and the evaluation of the integral in (3.11) relies on the knowledge of the Riemann invariants associated with each eigenvectors. These Riemann invariants are given in e.g. [2], [23], and the problem of evaluating the integral in (3.11) has many similarities with the analogous problem in the single-component case, since Y (and therefore γ) is constant along those pieces of the integration path which are parallel to the first or to the last eigenvector. We refer the reader to [2] for the details.

3.1.3. The weakly-coupled multi-component Roe and Osher schemes

This weakly-coupled extension, proposed in [20], [21], is based on the property (2.23) of the exact solution of the multi-component Riemann problem (2.22).

In order to introduce it, let us first recall that in the Godunov-type schemes, the numerical flux $\phi_{j+1/2}$ is seen as an approximation of the flux $F[\mathcal{W}(0; W_j^n, W_{j+1}^n)]$ of the exact solution of the Riemann problem constructed with the neighbour states W_j^n and W_{j+1}^n. One even has the equality, $\phi_{j+1/2} = F[\mathcal{W}(0, W_j^n, W_{j+1}^n)]$, in the original Godunov method [19].

With this idea in mind, we define the weakly-coupled multi-component extension as follows. The first three components of $\phi_{i+1/2}$ (that is, the density, momentum and energy discrete fluxes) are evaluated using a classical single-component scheme (Roe or Osher scheme), with a "frozen γ". This means that, for the evaluation of the first three components of $\phi_{j+1/2}$ at time t^n, one uses the flux of the usual (i.e. single-component) Roe or Osher scheme computed using a frozen value $\gamma_{j+1/2} = \gamma\left(\frac{Y_j^n + Y_{j+1}^n}{2}\right)$.

Beside this, one evaluates the fourth component of the numerical flux from the following relation, which mimics (2.23):

$$\phi_{j+1/2}^4 = \phi_{j+1/2}^1 \times \begin{cases} Y_j^n & \text{if } \phi_{j+1/2}^1 > 0 \,, \\ Y_{j+1}^n & \text{if } \phi_{j+1/2}^1 < 0 \,. \end{cases} \tag{3.12}$$

Beside its simplicity, this approach has the major advantage of preserving the maximum principle for the mass fraction. The result shown in [21] is the following:

Proposition 4:

Under the following CFL-like conditions:

$$\frac{\Delta t}{\Delta x}\left[\frac{\max(\phi_{i+1/2}^1, 0)}{\rho_i^n} - \frac{\min(\phi_{i+1/2}^1, 0)}{\rho_{i+1}^n}\right] \leq 1 \,, \tag{3.13}$$

the weakly-coupled multi-component schemes defined above preserve the maximum principle for the mass fraction Y: for all i and $n \geq 0$:

$$\min_j Y_j^0 \leq Y_i^n \leq \max_j Y_j^0 \,. \bullet \tag{3.14}$$

The proof of Proposition 4 essentially relies on the fact that weakly-coupled schemes use the same discrete mass fluxes for all discrete mass conservation equations (the continuity equation and the species equation).

Remark 6: It is even shown in [21] that the one-dimensional weakly-coupled multi-component Roe or Osher schemes preserve the mass fraction monotonicity. It is also observed there that condition (3.13) is not more restrictive than the usual CFL condition when an upwind scheme is used to evaluate the first three components of $\phi_{j+1/2}$. •

3.1.4. Fully-coupled versus weakly-coupled multi-component schemes

Several detailed comparisons between the fully-coupled and the weakly-coupled approaches have been carried out, for both Roe and Osher schemes, in one or two space dimensions (see [6], [7], [21]). As one could expect, these comparisons have shown that:

* The weakly-coupled approach gives more accurate values for the computed mass fractions Y. In particular, the fully-coupled approach does not always preserve the inequalities $0 \leq Y \leq 1$: computed mass fractions values between -5.10^{-2} and 0, or between 1 and 1.05 are sometimes obtained with the fully-coupled schemes.

* The weakly-coupled schemes are easier to implement and cheaper than the fully-coupled schemes. This is particularly true for Roe scheme when the number N of gaseous species in the mixture is large, since the fully-coupled Roe scheme then involves $N + 2 \times N + 2$ matrices.

* When γ does not actually depend on the mixture composition (that is, if both species have the same specific heat ratio $\gamma_1 = \gamma_2$), the fully-coupled and the weakly-coupled schemes exactly give the same results for the computed hydrodynamical variables ρ, u, p. On the opposite, when γ genuinely depends on Y, the computed values of the hydrodynamical variables ρ, u, p are slightly less accurate with the weakly-coupled Roe scheme than with the fully-coupled Roe scheme. This could be expected since the former approach uses a frozen γ for evaluating the "Euler fluxes" ϕ^1, ϕ^2, ϕ^3, whereas the latter takes the variations of γ into account (the partial derivatives $\frac{\partial \gamma}{\partial W^i}$ appear in the matrix $\tilde{A}$ (3.8)). However, this drawback of the weakly-coupled approach vanishes with a slight modification of the scheme (aiming at taking the variations of γ into account in the evaluation of the Euler fluxes instead of using a frozen γ; we refer to [6] for the details).

Therefore, the weakly-coupled method, which has both advantages of being cheaper and of preserving the maximum principle for the mass fraction, should be preferred to the fully-coupled approach. This is in particular the case without any modification of the computation of the Euler fluxes in those cases where the specific heat ratio γ is constant (which, as said above, happens if $\gamma_1 = \gamma_2$, but also if the variable Y, instead of being the mass fraction of a species in the mixture, represents any other quantity which is simply convected by the flow).

3.2. Multi-component flux vector splitting

The best-known flux vector splitting techniques developed for the single-component Euler equations, namely the methods of Steger and Warming [34] and of Van Leer [41] have been extended to multi-component mixtures in [23]. Since the extension of the Steger and Warming splitting is straightforward (because the multi-component flux

vector is still an homogeneous function of the conservative variables, as in the single-component case), we will simply consider here the differentiable flux vector splitting of Van Leer.

As in the single-component case, the numerical flux function of the multi-component Van Leer scheme has the form:

$$\Phi(W_L, W_R) = F_+(W_L) + F_-(W_R) \,, \tag{3.15}$$

where, for any W:

$$F(W) = F_+(W) + F_-(W) \,, \tag{3.16}$$

and where F_+ is defined by the following expressions:

* if $u \geq c = \sqrt{\frac{\gamma p}{\rho}}$, $F_+(W) = F(W)$;

* if $-c \leq u \leq c$,

$$F_+(W) = \begin{pmatrix} F_+^1 \\ F_+^2 \\ F_+^3 \\ F_+^4 \end{pmatrix} = \begin{pmatrix} \frac{\rho}{4c}(u+c)^2 \\ F_+^1 \left(u - \frac{u-2c}{\gamma} \right) \\ \frac{\gamma^2}{2(\gamma^2-1)} \frac{(F_+^2)^2}{F_+^1} \\ Y F_+^1 \end{pmatrix} \,, \tag{3.17}$$

(the first three components of F_+ have the same expression as in the single-component case, but now with the non constant coefficient γ);

* if $u \leq -c$, $F_+(W) = 0$.

As in the single-component case, we have here the property that if all characteristic wave speeds associated with the state W are positive (resp. negative), then $F_+(W) = F(W)$ (resp. $F_-(W) = F(W)$). Moreover, the following result, which says that this flux splitting can be used to define a stable conservative scheme, is proved in [23] (the analogous result for the single-component case was proved in [41]):

Proposition 5:

If the specific heat ratio γ_k of each species in the mixture satisfies the inequality $1 < \gamma_k < 3$, then all eigenvalues of the Jacobian matrix $\frac{DF_+}{DW}$ (resp. $\frac{DF_-}{DW}$) are real and positive (resp. negative). •

Lastly, we can notice that, exactly as the weakly-coupled Roe and Osher schemes of the previous section, this multi-component Van Leer scheme uses for the discrete species equations the same discrete mass fluxes $F_{\pm}^{1}$ as for the continuity equation. This allows us to prove that this scheme also preserves the maximum principle for the mass fraction (see [21] for the details):

Proposition 6:

Under the following CFL-like conditions:

$$\frac{\Delta t}{\Delta x} \frac{u_j^2 + c_j^2}{2c_j} \leq 1 \, , \tag{3.18}$$

the multi-component Van Leer scheme defined above preserves the maximum principle for the mass fraction Y: for all i and $n \geq 0$:

$$\min_j Y_j^0 \leq Y_i^n \leq \max_j Y_j^0 \; . \; \bullet \tag{3.19}$$

4. EXTENSIONS

In this section, we briefly give an idea of how the above one-dimensional multi-component upwind schemes can be extended to second-order accuracy, to implicit time-stepping, and to multi-dimensional reactive flows. We refer to the bibliography for the details.

4.1. One-dimensional extensions

4.1.1. Second-order accuracy

Starting from the previous first-order accurate schemes, the second-order spatial accuracy is obtained by using piecewise linear variables instead of piecewise constant variables, following the MUSCL approach of Van Leer [39], [40]. To obtain schemes which are second-order accurate in space but remain first-order accurate in time, the method involves three steps:

(a) At each time step, starting from the values W_i^n, one first evaluates slopes s_i^n for all variables which are chosen to be piecewise linear. Several choices are possible at this stage (for instance, one can choose either the conservative variables ρ, ρu, E, ρY or the "physical variables" ρ, u, p, Y to vary linearly in each

computational cell; see [11]); here, we take Y (and not ρY) as a piecewise linear variable.

(b) Slope limiters are then used in order to avoid the creation of new extrema; here again different strategies exist to evaluate the limited slopes (see e.g. [11], [30], [32], [38], [39]). Essentially, the slopes are limited in order to avoid the creation of new extrema, i.e. they are constrained such that, taking the variable Y as an example, we have:

$$\min_{|j-i|\leq 1} Y_j^n \leq Y_i^n \pm \frac{\Delta x}{2} s_i^n \leq \max_{|j-i|\leq 1} Y_j^n \,. \tag{4.1}$$

(c) The limited slopes are then used to evaluate cell-interface values $W^n_{i+1/2,\pm}$ (one sets $Y^n_{i+1/2,-} = Y_i^n + \frac{\Delta x}{2} s_i^n$, $Y^n_{i-1/2,+} = Y_i^n - \frac{\Delta x}{2} s_i^n$), and the solution is advanced in time according to relation (1.5), where we now take:

$$\phi_{i+1/2} = \Phi(W^n_{i+1/2,-}, W^n_{i+1/2,+}) \,. \tag{4.2}$$

This construction *(a)-(c)* can be applied to any of the schemes presented in the preceding sections, by using in (4.2) the corresponding numerical flux function Φ. In particular, it is shown in [21] that the result of Proposition 4 still holds: the weakly-coupled second-order accurate schemes preserve the maximum principle for the mass fractions, provided that limited slopes are used for the mass fraction Y itself and not for ρY.

The extension to a second-order time-accurate scheme can also be done in the usual way using a predictor-corrector formulation (see e.g. [11], [15], [21]).

4.1.2. Implicit time-stepping

All the above described explicit schemes can be made implicit, following the lines of [10], [17], [36] (these works deal with the single-component case). To give the idea, one uses instead of the explicit formulation (3.2) the following (linearized implicit) conservative formulation:

$$\frac{W_j^{n+1} - W_j^n}{\Delta t} + \frac{\phi^{n+1}_{j+1/2} - \phi^{n+1}_{j-1/2}}{\Delta x} = 0 \,, \tag{4.3}$$

where the numerical flux $\phi^{n+1}_{j+1/2}$ has the form:

$$\phi^{n+1}_{j+1/2} = \Phi(W_j^n, W_{j+1}^n) + \frac{\partial \Phi}{\partial W_j}(W_j^{n+1} - W_j^n) + \frac{\partial \Phi}{\partial W_{j+1}}(W_{j+1}^{n+1} - W_{j+1}^n) \,, \tag{4.4}$$

the quantities $\frac{\partial \Phi}{\partial W_j}$ and $\frac{\partial \Phi}{\partial W_{j+1}}$ being the Jacobian matrix (or an approximate Jacobian matrix) of the numerical flux function $\Phi(W_j, W_{j+1})$ (see [10], [17], [36]).

Since no particular difficulty appears in extending these implicit formulations from the single-component case to the multi-component case (we refer to [13] for the implicit fully-coupled multi-component Roe scheme, and to [18] for the implicit multi-component Van Leer scheme), we simply add now some comments on the implicit weakly-coupled multi-component schemes. For these schemes, (3.12) simply becomes:

$$\phi^{4,n+1}_{i+1/2} = \phi^{1,n+1}_{i+1/2} \times \begin{cases} Y_i^{n+1} & \text{if } \phi^{1,n+1}_{i+1/2} > 0 \ , \\ Y_{i+1}^{n+1} & \text{if } \phi^{1,n+1}_{i+1/2} < 0 \ . \end{cases} \tag{4.5}$$

Therefore the solution of a linear system is required at each time step to evaluate the mass fractions Y_i^{n+1}; it is shown in [21] that the matrix of this linear system is an M-matrix (see e.g. [42]), and that the properties of the explicit weakly-coupled schemes still hold here, without any restriction on the time step:

Proposition 7:

For any value of the time step Δt, the implicit weakly-coupled schemes defined by (4.5) preserve the maximum principle for the mass fraction Y: for all i and $n \geq 0$:

$$\min_j Y_j^0 \leq Y_i^n \leq \max_j Y_j^0 \ . \ \bullet \tag{4.6}$$

4.2. Extension to two-dimensional reactive flows

Let us now turn to multi-dimensional flows, with the description of diffusive and reactive effects. To present this extension, we will consider the explicit simulation of a two-dimensional laminar inviscid flow of a mixture of two reactive species Σ_1 and Σ_2 (the extension to three-dimensional flows and to viscous flows can be done along the same lines; see [16], [32], [36], [37]). Thus, we consider the following system of equations:

$$\begin{cases} \rho_t + (\rho u)_x + (\rho v)_y = 0 \ , \\ (\rho u)_t + (\rho u^2 + p)_x + (\rho uv)_y = 0 \ , \\ (\rho v)_t + (\rho uv)_x + (\rho v^2 + p)_y = 0 \ , \\ E_t + [u(E+p)]_x + [v(E+p)]_y = \overrightarrow{\nabla}.(\lambda \overrightarrow{\nabla} T) + \Omega_T + \sum_{k=1}^{2} \overrightarrow{\nabla}.(\rho D C_{pk} T \overrightarrow{\nabla} Y_k) \ , \\ (\rho Y)_t + (\rho u Y)_x + (\rho v Y)_y = \overrightarrow{\nabla}.(\rho D \overrightarrow{\nabla} Y) + \Omega_Y \ , \end{cases} \tag{4.7}$$

with:

$$\begin{cases} E = \sum_{k=1}^{2} \rho Y_k C_{v_k} T + \frac{1}{2}\rho(u^2+v^2) \, , \\ p = \sum_{k=1}^{2} \dfrac{\rho Y_k R T}{M_k} \, . \end{cases} \tag{4.8}$$

Our notations are classical : u and v are the components of the mixture velocity $\overrightarrow{U}$, E is again the sum of the internal and kinetic energies per unit volume, λ is the mixture thermal conductivity, D is the molecular diffusion coefficient of species Σ_1. Lastly, the source terms Ω_T and Ω_Y represent the contribution of the chemical reactions to the energy and mass fraction equations. We will assume below that the quantities λ , ρD, C_{pk}, and C_{vk} are constant.

As said in the introduction, we consider mixed finite-element / finite-volume two-dimensional extensions of the upwind schemes presented above, in the spirit of the methods developed by A. Dervieux and L. Fezoui [10], [15] for a single gas (but of course all above one-dimensional schemes can also be extended to structured finite-volume multi-dimensional meshes). To make it precise, let us first rewrite system (4.7)-(4.8) under the following form, separating the time-dependent, convective, diffusive and reactive terms:

$$\hat{W}_t + \hat{F}(\hat{W})_x + \hat{G}(\hat{W})_y = \hat{P}(\hat{W}, \hat{W}_x)_x + \hat{Q}(\hat{W}, \hat{W}_y)_y + \hat{S}(\hat{W}) \, , \tag{4.9}$$

where:

$$\begin{cases} \hat{W} = (\rho, \rho u, \rho v, E, \rho Y)^T \, , \\ \hat{F}(\hat{W}) = (\rho u, \rho u^2 + p, \rho uv, u(E+p), \rho uY)^T \, , \\ \hat{G}(\hat{W}) = (\rho v, \rho uv, \rho v^2 + p, v(E+p), \rho vY)^T \, , \end{cases} \tag{4.10}$$

$$\begin{cases} \hat{P}(\hat{W}, \hat{W}_x) = (0,0,0, \lambda T_x + \sum_{k=1}^{2} \rho D C_{pk} T (Y_k)_x, \rho D Y_x)^T \, , \\ \hat{Q}(\hat{W}, \hat{W}_y) = (0,0,0, \lambda T_y + \sum_{k=1}^{2} \rho D C_{pk} T (Y_k)_y, \rho D Y_y)^T \, , \end{cases} \tag{4.11}$$

$$\hat{S}(\hat{W}) = (0,0,0, \Omega_T, \Omega_Y)^T \, . \tag{4.12}$$

Then we introduce a (possibly unstructured) finite-element triangulation of the computational domain. In order to derive a finite-volume formulation, we consider a dual partition of the domain in control volumes or cells : a cell C_i is constructed around each vertex S_i by means of the medians of the neighbouring triangles, as shown on Figure 2.

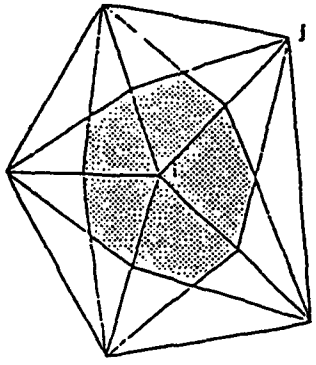


Figure 2: The control volume C_i.

Integrating (4.9) on the control volume C_i, we get:

$$\int\int_{C_i} \hat{W}_t + \int_{\partial C_i} (\hat{F}\nu_i^x + \hat{G}\nu_i^y) = \int_{\partial C_i} (\hat{P}\nu_i^x + \hat{Q}\nu_i^y) + \int\int_{C_i} \hat{S} \ , \qquad (4.13)$$

where $\overrightarrow{\nu_i} = (\nu_i^x, \nu_i^y)$ is the outward unit normal on ∂C_i. It now remains to specify how the four integrals in (4.13) are evaluated.

The time derivative and source terms integrals are approximated using a mass-lumped approximation:

$$\int\int_{C_i} \hat{W}_t = \frac{\hat{W}_i^{n+1} - \hat{W}_i^n}{\Delta t} \ . \ area(C_i) \ , \quad \int\int_{C_i} \hat{S} = \hat{S}(W_i^n) \ . \ area(C_i) \ . \qquad (4.14)$$

In addition to its simplicity, the mass-lumped approximation has two advantages: first, it allows us to employ an explicit time integration scheme, which is no longer possible when the consistent non diagonal finite-element mass matrix is used; moreover, the mass-lumped approximation of the heat equation may preserve the positivity of the unknowns, while a consistent finite-element formulation does not (see e.g. [8]).

Next, we have to consider the integral of the diffusive fluxes in (4.13). In view of the definitions (4.11) of $\hat{P}$ and $\hat{Q}$, this integral reduces to expressions like:

$$\int_{\partial C_i} \overrightarrow{\nabla} T . \overrightarrow{\nu_i} \ , \quad \int_{\partial C_i} \overrightarrow{\nabla} Y . \overrightarrow{\nu_i} \ , \quad \text{and} \quad \int_{\partial C_i} T \overrightarrow{\nabla} Y . \overrightarrow{\nu_i} \ . \qquad (4.15)$$

To evaluate these terms, we consider here that the integrands are constant in each triangle τ of the triangulation. More precisely, we consider that, in a triangle τ with vertices S_j $(1 \leq j \leq 3)$, we have:

$$\nabla T|_\tau = \sum_{j=1}^{3} T_j \nabla \phi_j \ , \quad \nabla Y|_\tau = \sum_{j=1}^{3} Y_j \nabla \phi_j \ , \tag{4.16}$$

where ϕ_j is the P1 finite-element basis function associated to vertex S_j and, for the last term in (4.15):

$$T|_\tau = \frac{1}{3} \sum_{j=1}^{3} T_j \ . \tag{4.17}$$

Then the diffusive term in (4) takes the value:

$$\int_{\partial C_i} (P \nu_i^x + Q \nu_i^y) = \sum_{\tau \in T(i)} \overrightarrow{\begin{pmatrix} P_\tau \\ Q_\tau \end{pmatrix}} \cdot \int_{\partial C_i \cap \tau} \overrightarrow{\nu_i} \ , \tag{4.18}$$

where P_τ and Q_τ are the constant values of P and Q in the triangle τ. It is easy to check that (4.16)-(4.18) is equivalent to a classical P1 finite-element discretization of the diffusive terms.

Lastly, we come to the approximation of the second integral in (4.13), which is based of course on the one-dimensional multi-component upwind schemes presented in the previous sections. Indeed, the system $\hat{W}_t + \hat{F}(\hat{W})_x + \hat{G}(\hat{W})_y = 0$ is a nonlinear hyperbolic system of conservation laws, which means that, for any $(\alpha, \beta) \in \mathbb{R}^2$, the matrix $\alpha \dfrac{\partial \hat{F}}{\partial \hat{W}} + \beta \dfrac{\partial \hat{G}}{\partial \hat{W}}$ has five real eigenvalues:

$$\begin{cases} \lambda_1 = \alpha u + \beta v - \sqrt{\alpha^2 + \beta^2}\, c \ , \\ \lambda_2 = \lambda_3 = \lambda_4 = \alpha u + \beta v \ , \\ \lambda_5 = \alpha u + \beta v + \sqrt{\alpha^2 + \beta^2}\, c \ , \end{cases} \tag{4.19}$$

with $c = \sqrt{\dfrac{\gamma p}{\rho}}$, and a complete set of real eigenvectors. Thus, we can extend all approximations defined in Section 3 for the one-dimensional flux vector F to the flux vector $\alpha \hat{F} + \beta \hat{G}$ (in other words, we use here the rotational invariance of the multi-component Euler equations). For instance, given two values $\hat{W}_L$ and $\hat{W}_R$ of $\hat{W}$, and a vector $\overrightarrow{\eta} = (\eta^x, \eta^y)$, we define a fully-coupled two-dimensional Roe numerical flux function $\hat{\Phi}$ by:

$$\hat{\Phi}(\hat{W}_L, \hat{W}_R, \overrightarrow{\eta}) = \frac{1}{2}[\mathcal{F}_\eta(\hat{W}_L) + \mathcal{F}_\eta(\hat{W}_R)] + \frac{1}{2} |\tilde{A}_\eta| (\hat{W}_R - \hat{W}_L) \ . \tag{4.20}$$

In this expression, we have set $\mathcal{F}_\eta(\hat{W}) = \eta^x \hat{F}(\hat{W}) + \eta^y \hat{G}(\hat{W})$, and $\tilde{A}_\eta = \tilde{A}_\eta(\hat{W}_L, \hat{W}_R)$ is a diagonalisable matrix satisfying Roe's property:

$$\mathcal{F}_\eta(\hat{W}_L) - \mathcal{F}_\eta(\hat{W}_R) = \tilde{A}_\eta(\hat{W}_R - \hat{W}_L) \,, \tag{4.21}$$

(the matrix $\tilde{A}_\eta$ is deduced from $\tilde{A}$ in (3.8)).

To evaluate the second integral in (4.13), we first write it in the form:

$$\int_{\partial C_i} (F\nu_i^x + G\nu_i^y) = \sum_{j \in \mathcal{K}(i)} \int_{\partial C_{ij}} (F\nu_i^x + G\nu_i^y) \,, \tag{4.22}$$

where $\mathcal{K}(i)$ is the set of neighbouring nodes of S_i, and where $\partial C_{ij} = \partial C_i \cap \partial C_j$. Then, defining the vector $\overrightarrow{\nu_{ij}} = (\nu_{ij}^x, \nu_{ij}^y)$ by:

$$\nu_{ij}^x = \int_{\partial C_{ij}} \nu_i^x \;, \quad \nu_{ij}^y = \int_{\partial C_{ij}} \nu_i^y \;, \tag{4.23}$$

we obtain a first-order accurate upwind approximation of the convective fluxes (4.22) by:

$$\int_{\partial C_i} (F\nu_i^x + G\nu_i^y) = \sum_{j \in \mathcal{K}(i)} \hat{\Phi}(\hat{W}_i, \hat{W}_j, \overrightarrow{\nu_{ij}}) \,, \tag{4.24}$$

with $\hat{\Phi}$ defined in (4.20).

This extension to two space dimensions can of course be used also for the weakly-coupled schemes. Then, the first four components of $\hat{\Phi}(\hat{W}_i, \hat{W}_j, \overrightarrow{\nu_{ij}})$ are evaluated using a single-component scheme (with γ frozen if need be), and the fifth component is given by:

$$\hat{\Phi}^5(\hat{W}_i, \hat{W}_j, \overrightarrow{\nu_{ij}}) = \hat{\Phi}^1(\hat{W}_i, \hat{W}_j, \overrightarrow{\nu_{ij}}) \;\times\; \begin{cases} Y_i & \text{if } \hat{\Phi}^1(\hat{W}_i, \hat{W}_j, \overrightarrow{\nu_{ij}}) > 0 \,, \\ Y_j & \text{if } \hat{\Phi}^1(\hat{W}_i, \hat{W}_j, \overrightarrow{\nu_{ij}}) < 0 \,. \end{cases} \tag{4.25}$$

It is straightforward to check that this two-dimensional weakly-coupled method still preserves the maximum principle for the mass fraction. Also, transforming a multi-dimensional Euler code into a multi-dimensional multi-component code using (4.25) is very easy and cheap.

We do not present in detail here how the limited second-order extension of this first-order numerical fluxes is derived. The main task in this derivation is to choose the slopes limiters, a problem which is obviously less simple in the present context of

two-dimensional unstructured grids than in one space dimension. We refer to e.g. [10], [11], [15], [32] for more details.

5. NUMERICAL EXAMPLES

We now briefly present three different numerical illustrations of the above methods, all dealing with two-dimensional reactive flows.

5.1. Flame propagation in a closed vessel

To show the ability of these methods to operate on unstructured triangulations, we first consider two examples of flame propagation problems where adaptive highly non uniform grids are used.

The first example is taken from [26], where a dynamic mesh refinement prodedure is used to follow the propagating flame. Referring to [4], [25], [26] for the details, we simply mention here the basic features of this adaptive procedure. It uses a multi-level triangular finite-element mesh with a filiation hierarchy between two consecutive levels. This procedure dynamically refines and unrefines the mesh, using refinement decisions based on some refinement criterion. When the adaption routine is called, it starts from an original coarse mesh, makes refinement decisions at that level (level 0), and creates a new mesh (of level 1) by local element division; then new decisions are made on this new mesh, and the mesh of level 2 is created, and so on...

We consider an experiment where a flame is ignited at the middle of the top wall of a square chamber filled with a combustible mixture initially at rest, and propagates downwards in the chamber. Figure 3 shows the five-level computational mesh and the temperature contours at two early stages of the flame propagation (we refer to [26] for more details on this computation).

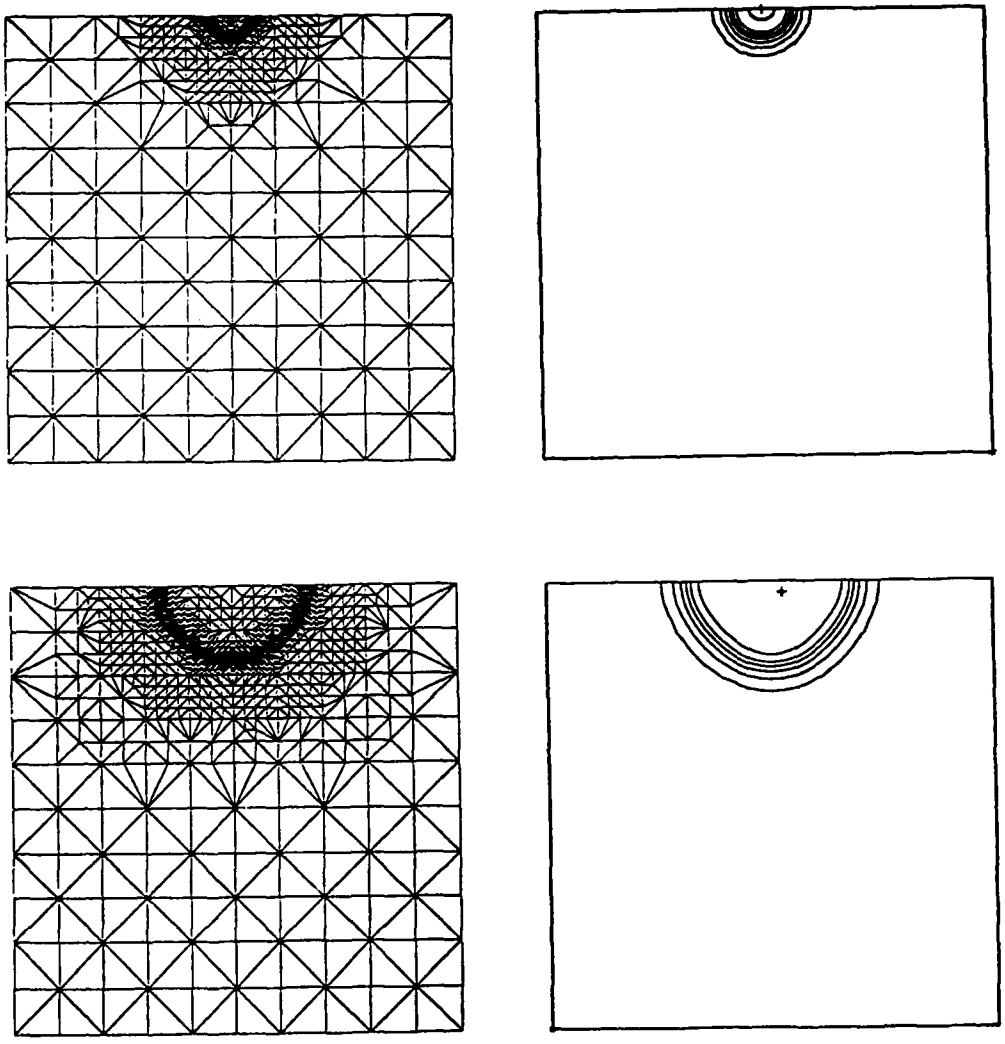


Figure 3: (from [26]) Unsteady flame propagation on a five-level dynamically adapted mesh: triangulation and isotherms at two different time levels.

The numerical method behaves well, although the original unrefined mesh is very coarse, which makes the adapted mesh highly non uniform. One remaining difficulty, which we are currently investigating, is that, although the (second-order accurate) weakly-coupled multi-component Roe scheme of Section 3.1.3 is used for the convective terms approximation, the computed mass fractions of all species in the mixture are not always in the interval $[0, 1]$. This is due to the (unavoidable) presence of obtuse angles in the adapted triangulations, and of diffusive terms in the governing equations: it is indeed well-known that the classical triangular finite-element approximation of the heat equation is not positivity-preserving if obtuse angles exist in the triangulation [8]. We are therefore in the strange situation where we are able to preserve the mass fraction positivity for the convective terms, but not for the dissipative terms...

The next numerical example, taken from [27], again concerns a flame propagating in a closed vessel. Here, the physical parameters have been chosen in order to reproduce numerically the so-called tulip flame instability which has been investigated experimentally in [35]; in particular, the flame is now thinner, and the Mach number of the flow is 10^{-3}. Also, a different mesh adaptation procedure is used: we employ here the line-by-line adaption algorithm of [3] (an example of the adapted mesh, corresponding to the last but one time level of Figure 4, is shown on Figure 5; as one can see on Figure 4, this pseudo-one-dimensional adaptive procedure is used only once the flame has reached the horizontal walls of the rectyangular chamber). A second-order accurate fully-coupled implicit Roe scheme is used in this computation, where the tulip instability is actually observed, in very good agreement with the experimental results of [35] (see [27]).

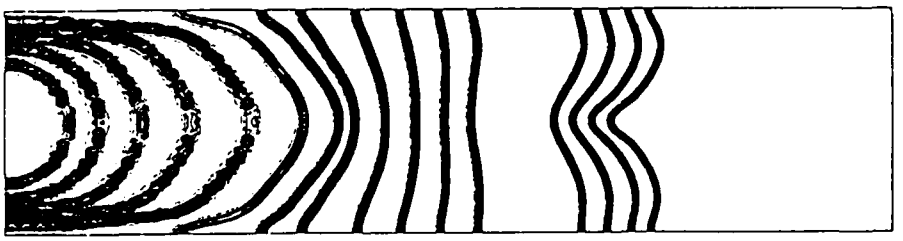


Figure 4: (from [27] Tulip flame instability: flame history (reaction rate contours at successive time levels).

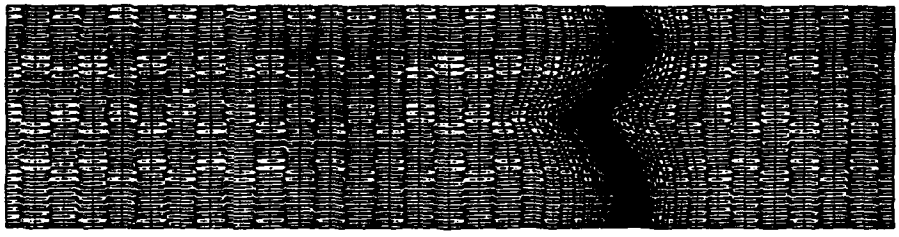


Figure 5: (from [27]) Tulip flame instability: adapted computational mesh.

5.2. Hypersonic reactive flow with non equilibrium chemistry

Our last example, taken from [18], concerns an inviscid hypersonic flow around a double ellipse, with non equilibrium air chemistry and vibrational equilibrium (i.e. the equation of state includes vibrational terms). The free-stream Mach number is equal to 25; we refer to [12], [18] and the references therein for the details. Figure 6 shows the steady-state Mach number contours, computed using a variant of the first-order accurate multi-component Van Leer scheme of Section 3.2 in which an approach using an "equivalent-γ" is employed to define locally the parameter γ. This calculation was made with two steady mesh refinements to cluster points in the detached shock and canopy shock regions. Again, the upwind finite-element method proves to be very robust, and oscillatory-free results are obtained.

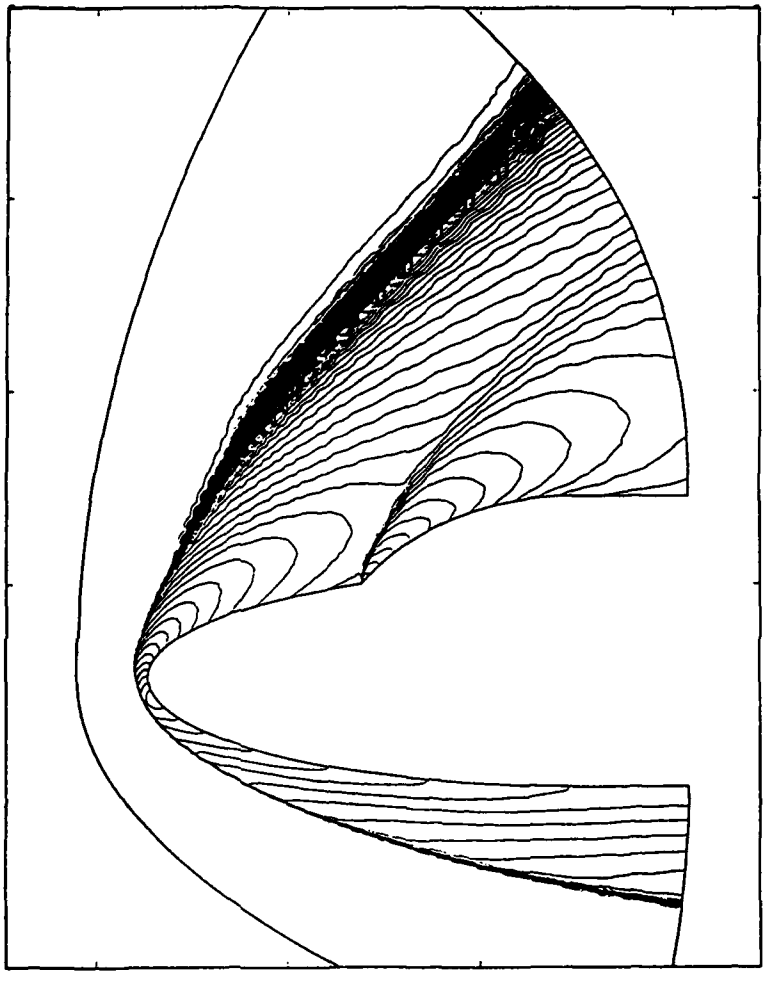


Figure 6: (from [18]) Inviscid non equilibrium air flow around a double ellipse.

ACKNOWLEDGEMENTS

I would like to warmly thank here all colleagues of INRIA who have participated in the studies presented in this paper.

REFERENCES

[1] R. ABGRALL, "Généralisation du schéma de Roe pour le calcul d'écoulements de mélanges de gaz à concentrations variables", La Recherche Aérospatiale, pp. 31-43, **6**, (1988).

[2] R. ABGRALL & J. L. MONTAGNE, "Généralisation du schéma d'Osher pour le calcul d'écoulements de mélanges de gaz à concentrations variables et de gaz réels", La Recherche Aérospatiale, to appear.

[3] F. BENKHALDOUN & B. LARROUTUROU, "Explicit adaptive calculations of wrinkled flame propagation", Int. J. Num. Meth. Fluids, **7**, pp. 1147-1158, (1987).

[4] F. BENKHALDOUN, P. LEYLAND & B. LARROUTUROU, "Dynamic mesh adaption for unsteady nonlinear phenomena - Application to flame propagation", Numerical grid generation in computational fluid mechanics 88, Sengupta Haüser Eiseman & Thompson eds., pp. 977-986, Pineridge Press, Swansea, (1988).

[5] J. D. BUCKMASTER & G. S. S. LUDFORD, "Theory of laminar flames", Cambridge Univ. Press, Cambridge, (1982).

[6] D. CHARGY, "Simulation numérique d'écoulements réactifs transsoniques", Thèse, in preparation.

[7] D. CHARGY, R. ABGRALL, L. FEZOUI & B. LARROUTUROU, "Comparisons of several numerical schemes for multi-component one-dimensional flows", INRIA Report, to appear.

[8] P. G. CIARLET & P. A. RAVIART, "Maximum principle and uniform convergence for the finite-element method", Comp. Meth. Appl. Mech. Eng., **2**, pp. 17-31, (1973).

[9] P. COLELLA & H. M. GLAZ, "Efficient solution algorithms for the Riemann problem for real gases", J. Comp. Phys., **59**, pp. 264-289, (1985).

[10] A. DERVIEUX, "Steady Euler simulations using unstructured meshes", Partial differential equations of hyperbolic type and applications, Geymonat ed., pp. 33-111, World Scientific, Singapore, (1987).

[11] A. DERVIEUX, L. FEZOUI & B. LARROUTUROU, "Upwind numerical methods for compressible flows", to appear.

[12] J. A. DESIDERI, N. GLINSKY & E. HETTENA, "Hypersonic reactive flow computations", Computers and Fluids, (1990), to appear.

[13] G. FERNANDEZ, "Simulation numérique d'écoulements réactifs à petit nombre de Mach", Thèse, Université de Nice, (1989).

[10] G. FERNANDEZ & B. LARROUTUROU, "Hyperbolic schemes for multi-component Euler equations", Nonlinear hyperbolic equations - Theory, numerical methods and applications, Ballmann & Jeltsch eds., pp. 128-138, Notes on Numerical Fluid Mechanics, **24**, Vieweg, Braunschweig, (1989).

[15] L. FEZOUI, "Résolution des équations d'Euler par un schéma de Van Leer en éléments finis", INRIA Report 358, (1985).

[16] L. FEZOUI, S. LANTERI, B. LARROUTUROU & C. OLIVIER, "Résolution numérique des équations de Navier-Stokes pour un fluide compressible en maillage triangulaire", INRIA Report 1033, (1989).

[17] L. FEZOUI & B. STOUFFLET, "A class of implicit upwind schemes for Euler simulation with unstructured meshes", J. Comp. Phys., **84**, (1), pp. 174-206, (1989).

[18] N. GLINSKY, L. FEZOUI, M. C. CICCOLI & J. A. DESIDERI, "Non-equilibrium hypersonic flow computations by implicit second-order upwind finite elements", Eighth GAMM conf. on numerical methods in fluid mechanics, Delft, (1989).

[19] S. K. GODUNOV, "A difference scheme for numerical computation of discontinuous solutions of equations of fluid dynamics", Math. Sbornik, **47**, pp. 271-306, (1959) (in russian).

[20] B. LARROUTUROU, "Problème de Riemann et approximation numérique pour les équations d'Euler multi-espèces", C. R. Acad. Sci. Paris, **309**, Série I, pp. 893-896, (1989).

[21] B. LARROUTUROU, "How to preserve the mass fraction positivity when computing compressible multi-component flows", INRIA Report 1080, (1989).

[22] B. LARROUTUROU, "Introduction to combustion modelling", Springer Series in Computational Physics, (1990), to appear.

[23] B. LARROUTUROU & L. FEZOUI, "On the equations of multi-component perfect or real gas inviscid flow", "Non linear hyperbolic problems", Carasso Charrier Hanouzet Joly eds., Lecture Notes in Mathematics, Springer Verlag, Heidelberg, (1989).

[24] P. D. LAX, "Hyperbolic systems of conservation laws and the mathematical theory of shock waves", CBMS regional conference series in applied mathematics, 11, SIAM, Philadelphia, (1972).

[25] N. MAMAN, "Méthodes de maillage adaptatif pour des problèmes de propagation de flammes laminaires", Thèse, Université de Toulouse, (1990), in preparation.

[26] N. MAMAN & B. LARROUTUROU, "On the use of dynamical finite-element mesh adaptation for 2D simulation of unsteady flame propagation", to appear.

[27] B. N'KONGA, H. GUILLARD & B. LARROUTUROU, "Numerical investigations of the tulip flame instability - Comparisons with experimental results", to appear.

[28] S. OSHER & F. SOLOMON, "Upwind schemes for hyperbolic systems of conservation laws", Math. Comp., 38, (158), pp. 339-374, (1982).

[29] P. L. ROE, "Approximate Riemann solvers, parameters vectors and difference schemes", J. Comp. Phys., 43, pp. 357-372, (1981).

[30] P. L. ROE, "Some contributions to the modelling of discontinuous flows", Proc. AMS/SIAM Seminar, San Diego, (1988).

[31] J. M. ROQUEJOFFRE, L. FEZOUI & B. LARROUTUROU, "Solveurs de Riemann pour gaz réels", to appear.

[32] P. ROSTAND & B. STOUFFLET, "Finite-volume Galerkin methods for viscous gas dynamics", INRIA Report 863, (1988).

[33] J. SMOLLER, "Shock waves and reaction-diffusion equations", Springer Verlag, New-York, (1983).

[34] J. L. STEGER & R. F. WARMING, "Flux vector splitting for the inviscid gas dynamic equations with applications to finite-difference methods", J. Comp. Phys., 40, (2), pp. 263-293, (1981).

[35] W. STEINERT, D. DUNN-RANKIN & R. F. SAWYER, "Influence of chamber length and equivalence ratio on flame propagation in a constant volume duct", Lawrence Berkeley Laboratory Report, (1982).

[36] H. STEVE, "Schémas implicites linéarisés décentrés pour la résolution des équations d'Euler en plusieurs dimensions", Thèse, Université d'Aix-Marseille I, (1988).

[37] B. STOUFFLET, J. PERIAUX, F. FEZOUI & A. DERVIEUX, "Numerical simulation of 3-D hypersonic Euler flow around space vehicle using adapted finite elements", AIAA paper 87-0560, (1987).

[38] P. K. SWEEBY, "High resolution schemes using flux limiters for hyperbolic conservation laws", SIAM J. Num. Anal., **21**, pp. 995-1011, 1984).

[39] B. VAN LEER, "Towards the ultimate conservative difference scheme II - Monotonicity and conservation combined in a second-order scheme", J. Comp. Phys., **14**, pp. 361-370, (1974).

[40] B. VAN LEER, "Towards the ultimate conservative difference scheme III - Upstream centered finite-difference schemes for ideal compressible flow", J. Comp. Phys., **23**, pp. 263-275, (1977).

[41] B. VAN LEER, "Flux-vector splitting for the Euler equations", Eighth international conference on numerical methods in fluid dynamics, Krause ed., pp. 507-512, Lecture notes in physics, **170**, Springer-Verlag, (1982).

[42] R. S. VARGA, "Matrix iterative analysis", Prentice Hall, (1962).

[43] F. A. WILLIAMS, "Combustion theory", second edition, Benjamin Cummings, Menlo Park, (1985).

TWO AND THREE DIMENSIONAL COMPRESSIBLE FLUID FLOW COMPUTATIONS WITH UNSTRUCTURED GRIDS

A. DESCAMPS, M.P. LECLERCQ and B. STOUFFLET

Avions Marcel Dassault - Bréguet Aviation
78 quai Marcel Dassault 92214 Saint-Cloud (France)

January 3, 1990

Abstract

This paper presents recent developments of the numerical tools used to simulate hypersonic fluid flows motivated by the Hermes shuttle project. The approach relies on the use of unstructured meshes combined with upwind approximations, especially the treatment of reactive gas problems is emphasized. Numerical results in two and three dimensions demonstrate the capacities of the methodology.

1 Introduction

The design of a new generation of space vehicles such as the advanced space transportation system (NASP) or the European space shuttle program (HERMES) requires the development of efficient numerical flow solvers taking into account adapted thermodynamical and chemical modelizations. During the atmosphere reentry of such vehicles at high velocity and at high altitude, dissociation, ionization and excitation of internal energy modes of air have to be considered. The gas is no more a perfect gas and thermal and chemical nonequilibrium models have to be included in the set of equations. When the caracteristic times of these phenomena are small enough compared to the fluid motion caracteristic time, all the processes are at equilibrium with their reverse processes and in this case, one has only to replace the perfect gas law by a general gas law.

The goal of this paper is to discuss the extension of an Euler flow solver developped in the past [14] using unstructured meshes to handle equilibrium and/or nonequilibrium reactive flow simulations. For perfect gas computations, we chose the Osher Riemann solver as upwind scheme combined with an unstructured grid MUSCL-type approximation [4]. This solver has proved to be very robust and free of any tuning parameters. Our aim is to keep these properties for real gas simulations by deriving an adapted generalization of this scheme.

Recently, Abgrall and Montagné [1], Larrouturou and Fézoui [3] have shown that for a mixture of perfect gas components (conservation of mass of each component is expressed), the computation of Riemann invariants (which is the keypoint to evaluate the numerical flux with the Osher solver) is possible and is a straightforward extension of the single perfect gas situation. The mixture constitution is only changed in the contact discontinuity and remains constant along the first and third subpathes as well as the ratio of specific heats. Unfortunatly, for a mixture of only thermally perfect gases (when specific heat coefficients are variables and functions of temperature), no simple analytic expression of Riemann invariants is available. Nevertheless, it seems reasonable to advocate the use of approximate Riemann solvers; this approach has been investigated in [1] for chemical equilibrium flows.

This paper presents at first a possible generalized Osher Riemann solver for a mixture of thermally perfect gases with nonequilibrium chemical assumption. Then, a simple generalized Osher solver is proposed when chemical equilibrium is assumed. The following section recalls the methodology of the basic Euler solver in which the generalized Riemann solvers will be included. Numerical experiments in two and three dimensions of hypersonic flow simulations on adapted non structured meshes are presented to validate and illustrate the possibilities of the reactive flow solvers.

2 Chemical non-equilibrium reactive flows

We consider a gas of mixture of N chemical species excluding ionized atoms or molecules and electrons. In this section, we assume that chemical reactions are in non-equilibrium (what is called Finite Rate Chemistry) and that vibrationnal excitation of molecules is in equilibrium with the translationnal one, that means that the system is characterized by a single temperature. At this point, we make no assumption on the use of possible algebraic equations (conservation of atoms, of mixing proportions..) to reduce the number of species appearing in the system of conservation laws. We intend to design an extension of the Osher approximate Riemann solver to treat such a gas using similar ideas of the approaches referenced in the introduction.

2.1 Thermodynamic model

The conservation equations in 1-D are given as follows:

$$\begin{cases} \frac{\partial}{\partial t}\rho + \frac{\partial}{\partial x}(\rho u) = 0 \\ \frac{\partial}{\partial t}(\rho u) + \frac{\partial}{\partial x}(\rho u^2 + p) = 0 \\ \frac{\partial}{\partial t}e + \frac{\partial}{\partial x}((e+p)u) = 0 \\ \frac{\partial}{\partial t}(\rho Y_i) + \frac{\partial}{\partial x}(\rho u Y_i) = \Omega_i \quad \text{for} \quad i = 1, \ldots, N-1 \end{cases}$$

where ρ denotes the density, u the velocity, p the pressure, e the total energy by unit volume, Y_i, Ω_i the mass fraction and the production rate of the ith species ($i = 1, \ldots, N$). We have the identity $\sum_i Y_i = 1$.

In term of conserved quantities $\mathbf{W}$, the flux can be written as:

$$\mathbf{W} = \begin{pmatrix} \rho \\ m \\ e \\ \rho_i \end{pmatrix} \quad ; \quad \mathbf{F}(\mathbf{W}) = \begin{pmatrix} m \\ \frac{m^2}{\rho} + p \\ (e+p)\frac{m}{\rho} \\ \rho_i \frac{m}{\rho} \end{pmatrix} \quad ; \quad \mathbf{\Omega}(\mathbf{W}) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \Omega_i \end{pmatrix}$$

where $m = \rho u$ is the momentum and $\rho_i = \rho Y_i$ is the ith *species density*.

We denote ϵ the specific internal energy and $\tilde{\epsilon} = \rho\epsilon$ the energy by unit volume:

$$e = \frac{1}{2}\rho u^2 + \rho\epsilon$$

So, let h be the total enthalpy by unit volume and λ the specific enthalpy:

$$h = \frac{1}{2}\rho u^2 + \rho\lambda$$

The pressure is a function of density ρ, of specific internal energy ϵ and of mass fractions Y_i :

$$p = p(\rho, \tilde{\epsilon}, (\rho_i)_{i=1,\ldots,N-1})$$

Considering a mixing of perfect gases, the relation between pressure, density and temperature is given by:

$$p = \rho R T \sum_{i=1}^{N} \frac{Y_i}{M_i} \tag{1}$$

where R is the universal constant of gases, M_i the molecular weight of species i. The temperature T arising in (1) is then computing by the following nonlinear expression of enthalpy:

$$\lambda = \epsilon + \frac{p}{\rho} = \sum_{i=1}^{N} Y_i \, h_{f_i}^0 + \sum_{i=1}^{N} Y_i \, C_{p_i}(T) \, T$$

or by

$$\epsilon = \sum_{i=1}^{N} Y_i \, h_{fi}^0 \; + \; \sum_{i=1}^{N} Y_i \, C_{v_i}(T) \, T = \sum_{i=1}^{N} Y_i \, f_i(T) \tag{2}$$

where h_{fi}^0 is the formation enthalpy at temperature T_0, C_{p_i} the specific heat coefficient at constant pressure and C_{v_i} at constant volume of species i.

We will denote in the sequel:

$$C_v = \sum_{i=1}^{N} Y_i f_i'(T) \quad \text{and} \quad R = \mathcal{R} \sum_{i=1}^{N} \frac{Y_i}{M_i}$$

The Jacobian matrix of the flux with respect to $\mathbf{W}$ can be easily computed. This matrix is diagonalizable with eigenvalues $u - c, u, u + c$. The expression of the sound speed is found to be

$$c^2 = \frac{\partial p}{\partial \rho} \; + \; \left(\frac{\tilde{\epsilon} + p}{\rho} \right) \frac{\partial p}{\partial \tilde{\epsilon}} + \sum_{i=1}^{N-1} Y_i \frac{\partial p}{\partial \rho_i} \tag{3}$$

This expression can be simplified in the considered context by studying the partial derivatives.

$$\begin{cases} p \;\; = \;\; p(\rho, \tilde{\epsilon}, (\rho_i)_{i=1,\ldots,N-1}) \;\; = \;\; \mathcal{R} T \sum_{i=1}^{N} \frac{\rho_i}{M_i} \\ \tilde{\epsilon} \;\; = \;\; \sum_{i=1}^{N} \rho_i f_i(T) \end{cases}$$

By considering the identity $\rho_N = \rho - \sum_{i=1}^{N-1} \rho_i$, these two equations become:

$$\begin{cases} p \;\; = \;\; \mathcal{R} T \frac{\rho}{M_N} + \sum_{i=1}^{N-1} \rho_i \left(\frac{1}{M_i} - \frac{1}{M_N} \right) \\ \tilde{\epsilon} \;\; = \;\; \rho f_N(T) + \sum_{i=1}^{N-1} \rho_i (f_i(T) - f_N(T)) \end{cases} \tag{4}$$

By deriving both equations (4) with respect to ρ, we get:

$$\begin{cases} \frac{\partial p}{\partial \rho} \;\; = \;\; \rho R \frac{\partial T}{\partial \rho} \; + \; \frac{\mathcal{R}}{M_N} T \\ \frac{\partial T}{\partial \rho} \;\; = \;\; \frac{-f_N(T)}{\rho C_v} \end{cases}$$

By deriving now both equations (4) with respect to $\tilde{\epsilon}$, we obtain:

$$\begin{cases} \frac{\partial p}{\partial \tilde{\epsilon}} \;\; = \;\; \rho R \frac{\partial T}{\partial \tilde{\epsilon}} \\ \frac{\partial T}{\partial \tilde{\epsilon}} \;\; = \;\; \frac{1}{\rho C_v} \end{cases}$$

and with respect to ρ_j:

$$\begin{cases} \frac{\partial p}{\partial \rho_j} \;\; = \;\; \rho R \frac{\partial T}{\partial \rho_j} \; + \; \mathcal{R} \left(\frac{1}{M_j} - \frac{1}{M_N} \right) T \\ \frac{\partial T}{\partial \rho_j} \;\; = \;\; \frac{(f_N - f_j)}{\rho C_v} \end{cases}$$

If we insert these identities in the expression of the speed of sound (3), we obtain the simplified equation:

$$c^2 = \left(1 + \frac{R}{C_v}\right) \frac{p}{\rho}$$

We then notice that in the case of non-equilibrium chemical reactive gas, the expression of the speed of sound leads to the definition of a coefficient $\tilde{\gamma}$ given by:

$$\tilde{\gamma} = 1 + \frac{R}{C_v}$$

This relation can also be achieved for a general multi-temperature thermo-chemical nonequilibrium system (see [10]). Furthermore, since temperature is an homogeneous function of W of degree one, so is the coefficient $\tilde{\gamma}$.

2.2 Flux Jacobian matrix properties

In order to simplify our study, we restrict the following analysis to the model that we will use in the applications. The gas of mixture that we are interested in is dissociated air constituted of five species O, N, NO, O_2 and N_2 subject to chemical reactions. Mass fractions $(Y_i)_{i=1,..,5}$ verify two algebraic relations; the first one is simply due to their definition:

$$\sum_{i=1}^{5} Y_i = 1$$

already used in the previous section. As the air is supposed to be a mixture of 79% nitrogen and 21% oxygen, the conservation of species gives the second relation:

$$\frac{1}{21}\left(\frac{Y_1}{m_1} + \frac{Y_3}{m_3} + 2\frac{Y_4}{m_4}\right) = \frac{1}{79}\left(\frac{Y_2}{m_2} + \frac{Y_3}{m_3} + 2\frac{Y_5}{m_5}\right)$$

where m_i is the molecular weight of the ith species.

The composition of the mixture is then entirely determined by the evolution equations of mass fractions of three of the components. We will keep the equations corresponding to the three first ones: O, N, NO .

The Jacobian matrix of the flux has the following expression:

$$\mathbf{A}(\mathbf{W}) = \frac{d\mathbf{F}(\mathbf{W})}{d\mathbf{W}} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -u^2 + p_\rho & 2u + p_m & p_e & p_{\rho_1} & p_{\rho_2} & p_{\rho_3} \\ u(p_\rho - h) & h + up_m & u(1+p_e) & up_{\rho_1} & up_{\rho_2} & up_{\rho_3} \\ -uY_1 & Y_1 & 0 & u & 0 & 0 \\ -uY_2 & Y_2 & 0 & 0 & u & 0 \\ -uY_3 & Y_3 & 0 & 0 & 0 & u \end{pmatrix}$$

where the partial derivatives of the pressure are given by:

$$p_e = \tilde{\gamma} - 1 \quad ; \quad p_m = -(\tilde{\gamma} - 1)u$$

$$p_{\rho_i} = -(\tilde{\gamma} - 1)\left(h_i^0 + C_{v_i}(T)T + \sum_{j=4}^{5} \frac{\partial \rho_j}{\partial \rho_i}\left(C_{v_j}(T)T\right)\right) + \sum_{j=4}^{5} \frac{\partial \rho_j}{\partial \rho_i}\frac{R}{m_j}T + \frac{R}{m_i}T.$$

As said in the previous section, this matrix is diagonalizable with three distinct eigenvalues:

$$\lambda_1(\mathbf{W}) = u - c\ , \quad \lambda_i(\mathbf{W}) = u \ \ (i = 2,...,5)\ , \quad \lambda_6(\mathbf{W}) = u + c$$

where c is the frozen speed of sound given by

$$c^2 = \tilde{\gamma}\frac{p}{\rho}$$

The right eigenvector matrix can be written as

$$\mathbf{R}(\mathbf{W}) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ u-c & u & u & u & u & u+c \\ h-uc & h-\frac{c^2}{\tilde{\kappa}} & u^2-\frac{\chi_1}{\tilde{\kappa}} & u^2-\frac{\chi_2}{\tilde{\kappa}} & u^2-\frac{\chi_3}{\tilde{\kappa}} & h+uc \\ Y_1 & Y_1 & Y_1 & 0 & 0 & Y_1 \\ Y_2 & Y_2 & 0 & Y_2 & 0 & Y_2 \\ Y_3 & Y_3 & 0 & 0 & Y_3 & Y_3 \end{pmatrix}$$

with $\tilde{\kappa} = \tilde{\gamma} - 1$ and $\chi_i = p_\rho + Y_i p_e$ for $i = 1,2,3$.

2.3 Riemann invariants

Since the system of conservation laws is an hyperbolic system, it is important to determine the property of each characteristic field (degeneracy or genuine nonlinearity). For that, we compute the sign of the following quantities:

$$\begin{aligned} \nabla\lambda_1(\mathbf{W})\cdot\mathbf{R}_1(\mathbf{W}) &= -\frac{c}{2\rho}(\tilde{\gamma}+1) \\ \nabla\lambda_i(\mathbf{W})\cdot\mathbf{R}_i(\mathbf{W}) &= 0 \qquad\qquad i = 2,\ldots,5 \\ \nabla\lambda_6(\mathbf{W})\cdot\mathbf{R}_6(\mathbf{W}) &= \frac{c}{2\rho}(\tilde{\gamma}+1) \end{aligned}$$

We deduce from these quantities that the caracteristic fields associated to λ_1 et λ_6 are genuinely non-linear while those associated to the eigenvalues λ_i for $i = 2,\ldots,5$ are linearly degenerated. The variation of the eigenvalue λ_1 (resp.λ_6) along the associated path is monotone. It means that only one sonic point at more exists where the sign of λ_1 (resp. λ_6) changes.

A Riemann invariant associated to the ith eigenvector is a function Φ obtained by solving the following equation:

$$\nabla\Phi(\mathbf{W})\cdot\mathbf{R}_i(\mathbf{W}) = 0 \tag{5}$$

So, for the genuinely nonlinear fields, we respectively get:

$$\frac{\partial\Phi}{\partial\rho} + \frac{\partial\Phi}{\partial m}(u-c) + \frac{\partial\Phi}{\partial e}(h-uc) + \sum_{j=1}^{3}\frac{\partial\Phi}{\partial\rho_j}Y_j = 0$$

$$\frac{\partial\Phi}{\partial\rho} + \frac{\partial\Phi}{\partial m}(u+c) + \frac{\partial\Phi}{\partial e}(h+uc) + \sum_{j=1}^{3}\frac{\partial\Phi}{\partial\rho_j}Y_j = 0$$

For $i = 1$ or $i = 6$, mass fractions Y_j are Riemann invariants, which can be easily verified. For $i = 2,\ldots,5$, the velocity u and pressure p are still Riemann invariants as in the perfect gas case. The other invariants have not an a priori analytic expression in the considered

context. In the case where the specific heat coefficients C_{v_i} (and then C_{p_i}) are constant (do not depend on temperature), Abgrall and Montagné [1], Fézoui and Larrouturou [8] have shown that the invariants can be computed and are the classic ones with the parameter $\tilde{\gamma}$ as defined previously. Consequently, we will formally use in our case (where the specific heat coefficients are functions of temperature) these expressions of Riemann invariants that are only now approximate Riemann invariants. This choice is of course not at all unique but provides simple expressions that will be used to define an extension of the Osher Riemann solver. This approximation does not affect the consistency of the scheme. The expressions of the Abgrall-Montagné Riemann invariants are given in Table 1.

$u-c$	u	$u+c$
$\Phi_1^1 = \frac{p}{\rho^{\tilde{\gamma}}}$		$\Phi_1^3 = \frac{p}{\rho^{\tilde{\gamma}}}$
$\Phi_2^1 = \frac{\rho_i}{\rho}$	$\Phi_1^2 = u$	$\Phi_2^3 = \frac{\rho_i}{\rho}$
$\Phi_3^1 = u + \frac{2}{\tilde{\gamma}-1}c$	$\Phi_2^2 = p$	$\Phi_3^3 = u - \frac{2}{\tilde{\gamma}-1}c$

Table 1: Riemann invariants

2.4 Extension of the Osher Riemann solver

To compute the flux separating two states $\mathbf{W}_l$ and $\mathbf{W}_r$, Osher et al. [11] proposed a numerical flux function that can be written in a condensed form as

$$\Phi(\mathbf{W}_l, \mathbf{W}_r) = \frac{1}{2}\left[\mathbf{F}(\mathbf{W}_l) + \mathbf{F}(\mathbf{W}_r)\right] - \frac{1}{2}\int_{\mathbf{W}_l}^{\mathbf{W}_r} |\mathbf{A}(\mathbf{W})| d\mathbf{W}$$

where $\quad |\mathbf{A}| = \mathbf{A}^+ - \mathbf{A}^-$.

Instead of using the exact path of the Riemann problem between $\mathbf{W}_l$ and $\mathbf{W}_r$, Osher proposed a path consisting of two rarefaction waves (corresponding to the two genuinely non-linear fields) and a contact discontinuity (corresponding to the linearly degenerated fields). For that, the integration path between the two states $\mathbf{W}_l$ and $\mathbf{W}_r$ is decomposed in three simple subpathes

$$\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$$

the director vectors of Γ_i belonging to the eigenspace associated to the eigenvalue λ_i . Eigenvalues can be ordered in two manners as $u-c$, u , $u+c$ (natural order) or as $u+c$, u , $u-c$ (order adopted by Osher et al.). We will use hereafter the last one that leads to the following parametrization:

- First subpath Γ_1 corresponding to a rarefaction wave is defined by $\frac{d\mathbf{W}(s)}{ds} = \mathbf{R}_6(\mathbf{W}(s))$.
- Second one Γ_2 is defined by $\frac{d\mathbf{W}(s)}{ds} = \sum_{i=2}^{5} \alpha_i \mathbf{R}_i(\mathbf{W}(s))$.

- and the last one Γ_3 corresponding also to a rarefaction wave is defined by $\frac{d\mathbf{W}(s)}{ds} = \mathbf{R}_1(\mathbf{W}(s))$.

We denote by $\mathbf{W}_{1/3}$, $\mathbf{W}_{2/3}$ the intersection states of the subpathes (respectively intersection of Γ_1 and Γ_2 and intersection of Γ_2 and Γ_3) and by $\overline{\mathbf{W}}_1$ and $\overline{\mathbf{W}}_3$ the two sonic points i.e. the points of Γ_1 (resp. Γ_3) where λ_1 (resp. λ_6) are equal to zero. The integration over the subpathes can be given in a simple form as follows.

- Along Γ_2, the eigenvalue $\lambda_2 = u$ remains constant equal to $u_{1/3}$. The corresponding integral is independent on the choice of parameters α_i and has the expression:

$$\int_{\Gamma_2} |\mathbf{A}(\mathbf{W})| d\mathbf{W} = \int_{\mathbf{W}_{1/3}}^{\mathbf{W}_{2/3}} |\mathbf{A}(\mathbf{W})| d\mathbf{W} = sign(\lambda_2(\mathbf{W}_{1/3})) \left[\mathbf{F}(\mathbf{W}_{2/3}) - \mathbf{F}(\mathbf{W}_{1/3})\right]$$

- The contribution on subpath Γ_1 is given by:

$$\int_{\Gamma_1} |\mathbf{A}(\mathbf{W})| d\mathbf{W} = sign(\lambda_6(\mathbf{W}_l)) \left[\mathbf{F}(\overline{\mathbf{W}}_1) - \mathbf{F}(\mathbf{W}_l)\right] + sign(\lambda_6(\mathbf{W}_{1/3})) \left[\mathbf{F}(\mathbf{W}_{1/3}) - \mathbf{F}(\overline{\mathbf{W}}_1)\right]$$

- The contribution on subpath Γ_3 is given by:

$$\int_{\Gamma_3} |\mathbf{A}(\mathbf{W})| d\mathbf{W} = sign(\lambda_1(\mathbf{W}_{2/3})) \left[\mathbf{F}(\overline{\mathbf{W}}_3) - \mathbf{F}(\mathbf{W}_{2/3})\right] + sign(\lambda_1(\mathbf{W}_r)) \left[\mathbf{F}(\mathbf{W}_r) - \mathbf{F}(\overline{\mathbf{W}}_3)\right]$$

The integral of the numerical flux function is at this point completely defined. We have now to precise how to compute the different states $\mathbf{W}_{1/3}$, $\mathbf{W}_{2/3}$, $\overline{\mathbf{W}}_1$ and $\overline{\mathbf{W}}_3$. For that, we use the approximate Riemann invariants $\Phi^i_{1,2,3}$ defined previously and given in Table 1; since they are assumed to verify (5), they remain constant on their corresponding subpath Γ_i. The following relations are found:

- For the wave corresponding to $u + c$:

$$\frac{p_l}{\rho_l^{\tilde{\gamma}_l}} = \frac{p_{1/3}}{\rho_{1/3}^{\tilde{\gamma}_l}}, \quad Y_l = Y_{1/3}, \quad u_l - \frac{2}{\tilde{\gamma}_l - 1} c_l = u_{1/3} - \frac{2}{\tilde{\gamma}_l - 1} c_{1/3} \tag{6}$$

- For the wave corresponding to $u - c$:

$$\frac{p_r}{\rho_r^{\tilde{\gamma}_r}} = \frac{p_{2/3}}{\rho_{2/3}^{\tilde{\gamma}_r}}, \quad Y_r = Y_{2/3}, \quad u_r + \frac{2}{\tilde{\gamma}_r - 1} c_r = u_{2/3} + \frac{2}{\tilde{\gamma}_r - 1} c_{2/3} \tag{7}$$

- For the contact discontinuity, velocity and pressure remain constant which give the following identities:

$$u_{1/3} = u_{2/3}, \quad p_{1/3} = p_{2/3} \tag{8}$$

We obtain at the end the following nonlinear scalar equation:

$$\frac{2c_l}{\tilde{\gamma}_l - 1} x^{\frac{\tilde{\gamma}_l - 1}{2}} + \frac{2c_r}{\tilde{\gamma}_r - 1} \left(\frac{p_l}{p_r}\right)^{\frac{\tilde{\gamma}_r - 1}{2\tilde{\gamma}_r}} x^{\left(\frac{\tilde{\gamma}_l - 1}{2} + \frac{\tilde{\gamma}_r - \tilde{\gamma}_l}{2\tilde{\gamma}_r}\right)} = u_r - u_l + \frac{2c_r}{\tilde{\gamma}_r - 1} + \frac{2c_l}{\tilde{\gamma}_l - 1}$$

where $x = \frac{\rho_{1/3}}{\rho_l}$.

This equation has been already discussed in the previous work of Abgrall and Montagné [1] who analyzed adapted resolution methods. We solve it by adequate dichotomy. The other unknowns are simple functions of x :

$$\begin{cases} \rho_{1/3} &= \rho_l\, x \\ u_{1/3} &= u_l - \dfrac{2}{\tilde{\gamma}_l - 1}\, c_l \\ p_{1/3} &= p_l\, x^{\tilde{\gamma}_l}\left[1 - x^{\frac{\tilde{\gamma}_l - 1}{2}}\right] \end{cases} \qquad \begin{cases} \rho_{2/3} &= \left(\dfrac{p_l}{p_r}\right)^{\frac{1}{\tilde{\gamma}_r}} \rho_r\, x^{\frac{\tilde{\gamma}_l}{\tilde{\gamma}_r}} \\ u_{2/3} &= u_{1/3} \\ p_{2/3} &= p_{1/3} \end{cases}$$

Sonic points are determined by expressing the corresponding eigenvalue is equal to zero and that Riemann invariants remain constant on the subpath:

$$\begin{cases} \overline{u}_1 + \overline{c}_1 = 0 \\ \overline{u}_1 - \dfrac{2}{\tilde{\gamma}_l - 1}\overline{c}_1 = u_l - \dfrac{2}{\tilde{\gamma}_l - 1} c_l \\ \dfrac{\overline{p}_1}{\overline{\rho}_1^{\tilde{\gamma}_l}} = \dfrac{p_l}{\rho_l^{\tilde{\gamma}_l}} \end{cases} \qquad \begin{cases} \overline{u}_3 - \overline{c}_3 = 0 \\ \overline{u}_3 + \dfrac{2}{\tilde{\gamma}_r - 1}\overline{c}_3 = u_r + \dfrac{2}{\tilde{\gamma}_r - 1} c_r \\ \dfrac{\overline{p}_3}{\overline{\rho}_3^{\tilde{\gamma}_r}} = \dfrac{p_r}{\rho_r^{\tilde{\gamma}_r}} \end{cases}$$

We notice that intermediate and sonic points are determined through density, velocity, pressure and mass fractions. Temperature is then computed by the state of law. The last quantity needed to compute the corresponding flux is energy which is determined by equation (2).

3 Chemical equilibrium reactive flows

It is of interest in applications to consider the limit case of local chemical equilibrium assumption. For many applications concerning the computation of reentry flow problems, this approximation is valid. In this case, mass fractions are given by assuming equilibrium $\mathbf{\Omega}(\mathbf{W}) = 0$. We then have:

$$Y_i = Y_i(\rho, \rho\epsilon) \implies p = p(\rho, \rho\epsilon)$$

In order to connect the variables p and c to the independent variables ρ and ϵ, it is of common use to introduce the nondimension variables

$$\overline{\gamma} = 1 + \frac{p}{\tilde{\epsilon}} \quad \text{et} \quad \tilde{\gamma} = \frac{\rho c^2}{p}\ .$$

We still have the relation

$$p = (\overline{\gamma} - 1)\rho\epsilon$$

This coefficient is not equal to the ratio of specific heat coefficients of the gas. The speed of sound c involved in the expressions of the eigenvalues $u - c, u, u + c$ of the Jacobian matrix of the system is given now by the differential formula:

$$c^2 = \frac{\partial p}{\partial \rho} + \left(\frac{\tilde{\epsilon} + p}{\rho}\right)\frac{\partial p}{\partial \tilde{\epsilon}}$$

As in the nonequilibrium case, we intend to define an extension of the Osher Riemann solver. We describe now in this case our approach

The field associated to the second eigenvalue is still linearly degenerated with Riemann invariants u and p. As in the previous section, we do not have analytic expressions of the Riemann invariants of the first and third characteristic fields. To evaluate the integrals of the Osher solver, we propose an approximate computation of them to define the path of integration as previously. We choose to compute the Riemann invariants as the ones given in Table 1 with the parameter $\tilde{\gamma}$ taken equal to $(\tilde{\gamma}(\mathbf{W}_l)+\tilde{\gamma}(\mathbf{W}_r))/2$. No nonlinear equation has to be solved in this case. Intersection and sonic points of the subpathes are known through density, velocity and pressure. In order to compute the corresponding energy, we assume that the parameter $\overline{\gamma}$ is constant along the first and third subpathes. In this methodology, we notice that the chemistry routine which gives pressure and speed of sound for one couple of data $(\rho, \rho\epsilon)$ is called only twice at each flux evaluation i.e. for the states $\mathbf{W}_l$ and $\mathbf{W}_r$.

4 Spacial Approximation

We present in this section the main features of a high order approximation of the two-dimension Euler equations relying on an upwind formulation on an unstructured mesh in conjunction with Total Variation Diminushing (TVD) properties. The extension to the three dimension equations is straitforward.

Let $\mathcal{T}_g$ be a triangulation of the computational domain $\mathcal{D}$ with boundary Γ. We can write the Euler system in a conservative form such as:

$$\frac{\partial \mathbf{W}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{W}) = \mathbf{\Omega}(\mathbf{W})$$

The complete formulation can be found in [4] and is based on a Green formula:

$$\left\{ \begin{array}{l} \text{Find } \mathbf{W}_g \in (\mathcal{V}_g)^m \quad \text{such as} \quad \forall N_{g_i} \in \mathcal{T}_g, \\ \displaystyle\int_{C_{g_i}} \frac{\partial \mathbf{W}_g}{\partial t}\, dx + \int_{\partial C_{g_i}} \mathbf{F}(\mathbf{W}_g)\cdot \mathbf{n}_{g_i}\, d\sigma + \int_{\partial C_{g_i}\cap\Gamma} \mathbf{F}(\mathbf{W}_g)\cdot \mathbf{n}_\Gamma\, d\sigma = \int_{C_{g_i}} \mathbf{\Omega}(\mathbf{W})\, dx \\ \mathbf{R}_{g_i} \stackrel{\text{def}}{=} -\displaystyle\int_{\partial C_{g_i}} \mathbf{F}(\mathbf{W}_g)\cdot \mathbf{n}_{g_i}\, d\sigma - \int_{\partial C_{g_i}\cap\Gamma} \mathbf{F}(\mathbf{W}_g)\cdot \mathbf{n}_\Gamma\, d\sigma \end{array} \right. \tag{9}$$

where $\mathcal{V}_g = \{V_g \in C^0(\mathcal{D})\ ;\ V_g \text{ is linear on each triangle }\}$ and $\mathbf{R}_g$ denotes the residual. The cell C_{g_i} is defined for each vertex $N_{g_i} \in \mathcal{T}_g$, as the union of the subtriangles which have N_{g_i} as vertex and result from the subdivision of each triangle of $\mathcal{T}_g$ by means of the median planes as shown on Figure 1. The vectors $\mathbf{n}_{g_i}$ and $\mathbf{n}_\Gamma$ designe outward normals of respectively the cell C_{g_i} and the domain boundary Γ.

The scheme will be completely defined if we now precise which approximation is used to compute the left hand-side integral in (9). In order to do this, the boundary ∂C_{g_i} of the cell C_{g_i} is splitted in panels $\partial S_{g_{ij}}$, joining the segment $[N_{g_i}, N_{g_j}]$ to the centroids of the triangle having N_{g_i} and N_{g_j} as common vertices.

Let us give the following notations :

$$\mathbf{F}_{ij}(\mathbf{W}_g) = \mathbf{F}(\mathbf{W}_g)\cdot \int_{\partial S_{g_{ij}}} \mathbf{n}_{g_i}\, d\sigma \quad \text{and} \quad \mathbf{P}_{ij}(\mathbf{W}_g) = \frac{\partial \mathbf{F}}{\partial \mathbf{W}}(\mathbf{W}_g)\cdot \int_{\partial S_{g_{ij}}} \mathbf{n}_{g_i}\, d\sigma$$

An upwinding is introduced in the computation of the convection term through the numerical flux function Φ of a first-order accurate upwind scheme by :

$$\int_{\partial S_{g_{ij}}} \mathbf{F}(\mathbf{W}_g)\cdot \mathbf{n}_i\, d\sigma = \mathbf{H}_{ij}^{(1)} = \Phi_{F_{ij}}(\mathbf{W}_{g_i}, \mathbf{W}_{g_j})$$
$$\text{where} \quad \mathbf{W}_{g_i} = \mathbf{W}_g(N_{g_i}) \quad \text{and} \quad \mathbf{W}_{g_j} = \mathbf{W}_g(N_{g_j})$$

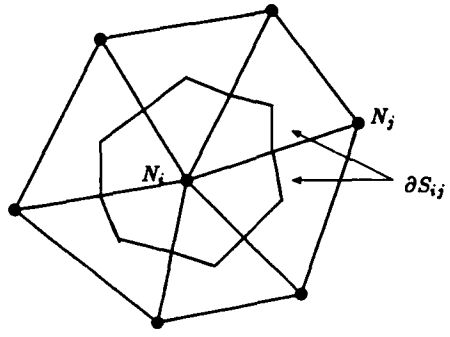


Figure 1: cell C_i

The numerical flux function used in this scheme is the Osher's approximate Riemann solver [11] in the nonreactive case or its extension as described in the first two sections when chemical effects are considered; this scheme has been chosen because of its robustness and its parameter-free implementation. The numerical integration with the upwind scheme, as described previously, leads to approximations which are only first-order accurate. A second-order accurate MUSCL-like [9] extension can be defined without changing the approximation space:

$$\left\{ \begin{array}{l} \text{Find } \mathbf{W}_g \in (\mathcal{V}_g)^m \quad \text{such that} \\ \displaystyle\int_{C_{g_i}} \frac{\partial \mathbf{W}_g}{\partial t}\, dx + \sum_{j \in K(i)} \mathbf{H}_{ij}^{(2)} + \int_{\partial C_{g_i} \cap \Gamma} \mathbf{F}(\mathbf{W}_g) \cdot \mathbf{n}_\Gamma \, d\sigma = \int_{C_{g_i}} \mathbf{\Omega}(\mathbf{W}_g)\, dx \end{array} \right. \tag{10}$$

where $K(i)$ is the set of neighbours of vertex N_{g_i}, and $\mathbf{H}_{ij}^{(2)} = \mathbf{\Phi}_{F_{ij}}(\mathbf{W}_{g_{ij}}, \mathbf{W}_{g_{ji}})$.

The arguments $\mathbf{W}_{g_{ij}}$ and $\mathbf{W}_{g_{ji}}$ are values at the interface $\partial S_{g_{ij}}$ which have been interpolated by using upwind gradients as described below.

We define the downstream and upstream triangles $T_{g_{ij}}$ and $T_{g_{ji}}$ for each segment $[N_{g_i}, N_{g_j}]$ as shown on Figure 2. Let the centered gradient be $\nabla \mathbf{W}_{g_{ij}} = \nabla \mathbf{W}_g \mid_{T_{g_{ij}}^\beta}$ where $T_{g_{ij}}^\beta$ is one of the triangles having N_{g_i} and N_{g_j} as vertices.

A good procedure in term of accuracy is to use limiters on characteristic variables. We compute these variables by the transformation taken at midpoint of the segment. If we denote by $\mathbf{\Pi}_{ij}$ the transformation matrix corresponding to $\mathbf{P}_{ij}((\mathbf{W}_{g_i} + \mathbf{W}_{g_j})/2)$, then the values at interface needed to compute the flux $\mathbf{H}_{ij}^{(2)}$ are given by :

$$\mathbf{W}_{g_{ij}} = \mathbf{W}_{g_i} + \mathbf{\Pi}_{ij}\ \mathbf{Lc}_{ij}\ \mathbf{\Pi}_{ij}^{-1} \left(\frac{1-\kappa}{4} \nabla \mathbf{W}_g \mid_{T_{g_{ij}}} + \frac{1+\kappa}{4} \nabla \mathbf{W}_{g_{ij}} \right) \cdot \overrightarrow{N_{g_i} N_{g_j}}$$

where $\mathbf{Lc}_{ij}$, $\mathbf{Lc}_{ji}$ are the diagonal limiting matrices introduced to reduce numerical oscillations of the solution and to provide some kind of monotonicity property. In all computations, we use the Van Albada limiter [2] associated to Fromm scheme corresponding to $\kappa = 0$, combining a certain monotonocity property and second-order accuracy [13].

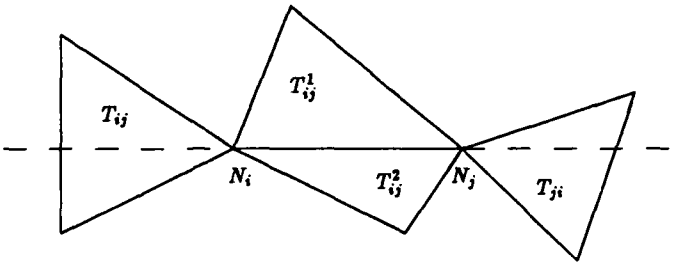


Figure 2: downstream and upstream triangles for the segment $[N_i, N_j]$

Boundary conditions

Boundary integrals over Γ of (10) are computed in order to take into account the physical boundary conditions.

- At inflow and outflow boundaries, the integral is evaluated with a flux-splitting applied between exterior data and interior values of the solution.
- On the body, the slip condition is prescribed; the boundary integral can then be written as a pressure integral term:

$$\int_{\partial C_{g_i} \cap \Gamma} \mathbf{F}(\mathbf{W}) \cdot \mathbf{n}_\Gamma \, d\sigma = \int_{\partial C_{g_i} \cap \Gamma} \begin{pmatrix} 0 \\ \hat{p}_i \, \mathbf{n}_\Gamma \\ 0 \\ \mathbf{0}_{R^N} \end{pmatrix} d\sigma$$

  where N_{g_i} is a node on the body and $\hat{p}_i$ has to be determined.

  In terms of Riemann problem, the boundary problem is under-defined since only the left state (taken equal to $\mathbf{W}_{g_i}$) and the wall speed are known. Nevertheless, it is easy to verify that there is no 3-wave and that the contact discontinuity is confounded with the wall [6]. The 1-wave can be either a shock ($\mathbf{u}_i \cdot \mathbf{n}_\Gamma > 0$) or a rarefaction ($\mathbf{u}_i \cdot \mathbf{n}_\Gamma \leq 0$) wave.

  In case of perfect gases, the so-called 1/2-Riemann problem can be solved analytically and allows the use of the Godunov flux.

  To take into account reactive effects, we will use the modified Osher flux (§2.4) since the apparition of a 1-shock wave would lead to a non-linear equation to be solved. Using formula (7), we obtain:

$$\hat{p}_i = p_i \left[1 - \frac{\tilde{\gamma}_i - 1}{2} \frac{\mathbf{u} \cdot \mathbf{n}_\Gamma}{c_i} \right]^{\frac{2\tilde{\gamma}_i}{\tilde{\gamma}_i - 1}}$$

5 Numerical treatment of source terms

Source terms Ω_i are evaluated according to the model of 17 chemical reactions (dissociation and exchange reactions) given in [12] and prescribed as basic model at the Hermes Workshop [5].

The source terms are treated implicitely in order to remove restrictive timesteps limitations due to the chemical part. The scheme is written as

$$\frac{\mathbf{W}_{g_j}^{n+1} - \mathbf{W}_{g_j}^{n}}{\Delta t} = \frac{1}{\text{area}(C_{g_i})} \mathbf{R}_{g_j}^{n} + \mathbf{\Omega}(\mathbf{W}_{g_j}^{n+1})$$

As in Désidéri et al. [3], the term $\mathbf{\Omega}(\mathbf{W}_{g_j}^{n+1})$ is linearized in the following manner:

$$\mathbf{\Omega}(\mathbf{W}_{g_j}^{n+1}) = \mathbf{\Omega}(\mathbf{W}_{g_j}^{n}) + \frac{\partial \mathbf{\Omega}}{\partial \mathbf{W}}(\mathbf{W}_{g_j}^{n}) \, (\mathbf{W}_{g_j}^{n+1} - \mathbf{W}_{g_j}^{n})$$

which gives

$$\left(Id - \Delta t \frac{\partial \mathbf{\Omega}}{\partial \mathbf{W}}(\mathbf{W}_{g_j}^{n}) \right) \delta \mathbf{W}_{g_j} = \frac{\Delta t}{\text{area}(C_{g_j})} \mathbf{R}_{g_j}^{n} + \Delta t \, \mathbf{\Omega}(\mathbf{W}_{g_j}^{n})$$

avec

$$\delta \mathbf{W}_{g_j} = \mathbf{W}_{g_j}^{n+1} - \mathbf{W}_{g_j}^{n}$$

This formulation leads to the resolution of a 3×3 system.

6 Numerical results

We present a set of typical computations obtained applying the methodology detailed below to illustrate the capacities of the solver for both equilibrium and nonequilibrium real gas simulations. For equilibrium flow simulations, we use the tabulated thermochemical model developped by Vancamberg [15]. All the computations have been performed by an explicit four-stage time stepping scheme allowing the use of Courant number of 1.8.

The equilibrium flow solver is going to be validated in 2D during the Workshop [5] with computations around a double ellipse and we only present here three dimensionnal experiments. The first result presented is the computation of the inviscid with local equilibrium chemical assumption around a forebody of the European space shuttle Hermes using the methodology described above at a freestream Mach number of 10 and at an angle of attack of 30^0. The iso-Mach number lines in the symmetry plane are diplayed in Figure 3 and 4. The two pictures on each Figure correspond respectively to a computation on an initial coarse grid and on a final one obtained after 4 successive adaptive refinements. On Figure 4, we can clearly notice that the canopy shock is well captured on the final mesh. This shows that refinement is mandatory for such computations to compute accurate solutions on reasonable grids. Then a computation of equilibrium flow around the complete Hermes using the same methodology corresponding to a freestream Mach number of 25 at an angle of attack of 30^0 and an altitude of 76000 meters is presented. Two adaptive mesh refinement procedures have been successively applied based on a criterion related to the gradient of Mach number. The initial surface mesh is presented in Figure 5. The first mesh is made of 13770 nodes and 74659 elements and the final one of 27338 nodes and 146343 elements. Surface iso-Mach number lines are shown on Figure 6 and iso-Mach number lines in the symmetry vertical plane in Figure 7 for the solution obtained on the final mesh. A comparison of the solutions obtained on both meshes is displayed on Figure 8 and 9 through a presentation of the iso-Mach number lines in a horizontal plane crossing the shuttle winglets. One can easily notice that adaptive mesh refinement has been active in the winglet regions and ine the shock capture. The validation of the equilibrium reactive gas solver has been performed by a numerical simulation of hypersonic flows around an Aeroassisted Orbital Transfer Vehicle geometry for which an accurate numerical prediction

of the aerodynamic moments is required for stability. The global final adapted mesh consists of 9041 nodes and 45817 elements obtained after two refinement procedures corresponding to a computation at Mach number of 10, no angle of attack and at 75000 meters altitude. Figure 10 presents the iso-Mach number lines in the symmetry plane and Figure 11 the pressure coefficient lines. Maximum temperature at stagnation point is equal to $4246K$.

We then present results concerning nonequilibrium chemical flows. The thermodynamic condidered in these applications is the one proposed in the hypersonic Workshop [5] where specific heat coefficients are given for molecules by

$$C_{v_i}(T) = \frac{\mathcal{R}}{m_i}\left(\frac{5}{2} + \frac{\theta_i^v/T}{\exp\left(\theta_i^v/T\right) - 1}\right)$$

Firstly a validation of the solver is performed on a shock tube problem proposed by Montagné defined by two states at rest corresponding to a density of 0.066, a temperature of $4390K$ on the left and a density of 0.030, a temperature of $1378K$ on the right. Results concerning density, Mach number, pressure and temperature are presented on Figure 12. No oscillations appear and the obtained solution is monotone. There is still some numerical dissipation when compared with other available results revealed by the insufficient sharpness of the rarefaction wave. This is certainly due to the limiting procedure which is applied on the primitive variables and not on the characteristic ones. The pressure remains constant in the contact discontinuity without any oscillation as it should be.

The computation of the hypersonic flow around a double ellipse has been performed with the nonequilibrium flow solver. The considered case is the one proposed in the aforementionned Workshop [5] at a freestream Mach number of 25 and at an angle of attack of 30^0. The two-dimensionnal mesh made of 4257 nodes is shown on Figure 13. Iso-Mach number lines are presented on Figure 14 ($\Delta M = .25$). The two shocks are well captured but the spreading of the upper part of the bow shock indicates the necessity of using adaptive refinement in this region. Next Figures (15, 16 and 17) show the wall values of the mass fractions of species produced by chemical reactions respectively NO, N and O. We notice that along the wall the amounts of O and NO are nearly constant. The mass fraction of N decreases along the double ellipse wall and is weakly affected by the canopy shock. Wall values of temperature, pressure coefficient and Mach number are displayed respectively on Figures 18, 19 and 20. The temperature at stagnation point in less than $10000K$. These results are in good agreement with other available results, for exemple in [7]; this is quite satisfactory because the mesh which is employed is undeniably coarse.

7 Conclusion

This paper has presented an extension of an inviscid perfect gas flow solver to take in account reactive real gases. Emphasis has been put on the capacity of the method to compute hypersonic flows around 2D and 3D geometries. We have not at all addressed here the efficiency of the solvers; development of implicit versions are under investigation.

8 Acknowledgments

This work has been performed in the Numerical Analysis Group of Dr J. Périaux who is acknowledged for his constant encouragements and motivation in the accomplishment of the work. The authors would like to thank Dr A. Dervieux from INRIA Sophia-Antipolis for

valuable discussions during this research. A close cooperation with the group of J.A. Désidéri, specially with E. Hettena and N. Glinsky, and with R. Abgrall who initiated the extension of Osher Riemann solver to nonequilibrium flows, made the realization of this work possible. C. Carasso and H. Gilquin from University of Saint-Etienne are also acknowledged.

References

[1] R. Abgrall and J.L. Montagné. Généralisation du schéma d'osher pour le calcul d'écoulements de mélanges de gaz à concentrations variables et de gaz réels. *La recherche aérospatiale*, 4:1–13, 1989.

[2] G.D. van Albada, B. van Leer, and W.W. Roberts. A comparative study of computational methods in cosmic gas dynamic. *Astron. Astrophys.*, 108:16–84, 1982.

[3] J.A. Désidéri, N. Glinsky, and E. Hettena. Hypersonic reactive flow computations. *Computers and Fluids*, to appear 1990.

[4] L. Fezoui and B. Stoufflet. A class of implicit upwind schemes for Euler simulations with unstructured meshes. *Journal of Computational Physics*, 84(1):174–206, september 1989.

[5] INRIA - GAMNI-SMAI. Workshop on hypersonic flow for reentry problems. Antibes(France), january 1990.

[6] H. Gilquin. *Analyse numérique d'un problème hyperbolique multidimensionnel en dynamique des gaz avec frontières mobiles.* thèse de doctorat de 3$^{\text{ième}}$ cycle, Université de Saint-Etienne, mai 1984.

[7] N. Glinsky, L. Fézoui, M.C. Ciccoli, and J.A. Désidéri. Non-equilibrium hypersonic flow computations by implicit second-order upwind finite-elements. In *8th GAMM Conference on Numerical Methods in Fluid Mechanics*, September 1989.

[8] B. Larrouturou and L. Fézoui. On the equations of multi-component perfect or real gas inviscid flow. In *Non linear hyperbolic problems, Carasso, Charrier, Hanouzet and Joly eds., Lecture Notes in Mathematics, Springer-Verlag*, 1989.

[9] B. van Leer. Toward the ultimate conservative difference scheme III. *Journal of Computationnal Physics*, 23:263–275, 1977.

[10] Y. Liu and M. Vinokur. Nonequilibrium flow computations. I. An analysis of numerical formulations of conservation laws. *Journal of Computational Physics*, 83:373–397, 1989.

[11] S. Osher and F. Salomon. Upwind difference schemes for hyperbolic systems of conservative laws. *Math. Comp.*, 38:339–374, 1988.

[12] C. Park. On convergence of chemically reacting flows. *AIAA paper*, (85-0247), 1985.

[13] S.P. Spekreijse. *Multigrid solution of the steady Euler equations.* thesis, C.W.I., Amterdam, 1987.

[14] B. Stoufflet, J. Periaux, L. Fezoui, and A. Dervieux. Numerical simulations of 3-D hypersonic Euler flows around space vehicles using adapted finite elements. *AIAA paper*, (87-0560), 1987.

[15] P. Vancamberg. Experimental and theoretical tests for the prediction of aerodynamical moments of hermes in hypersonic flight. ICAS,London, 1986.

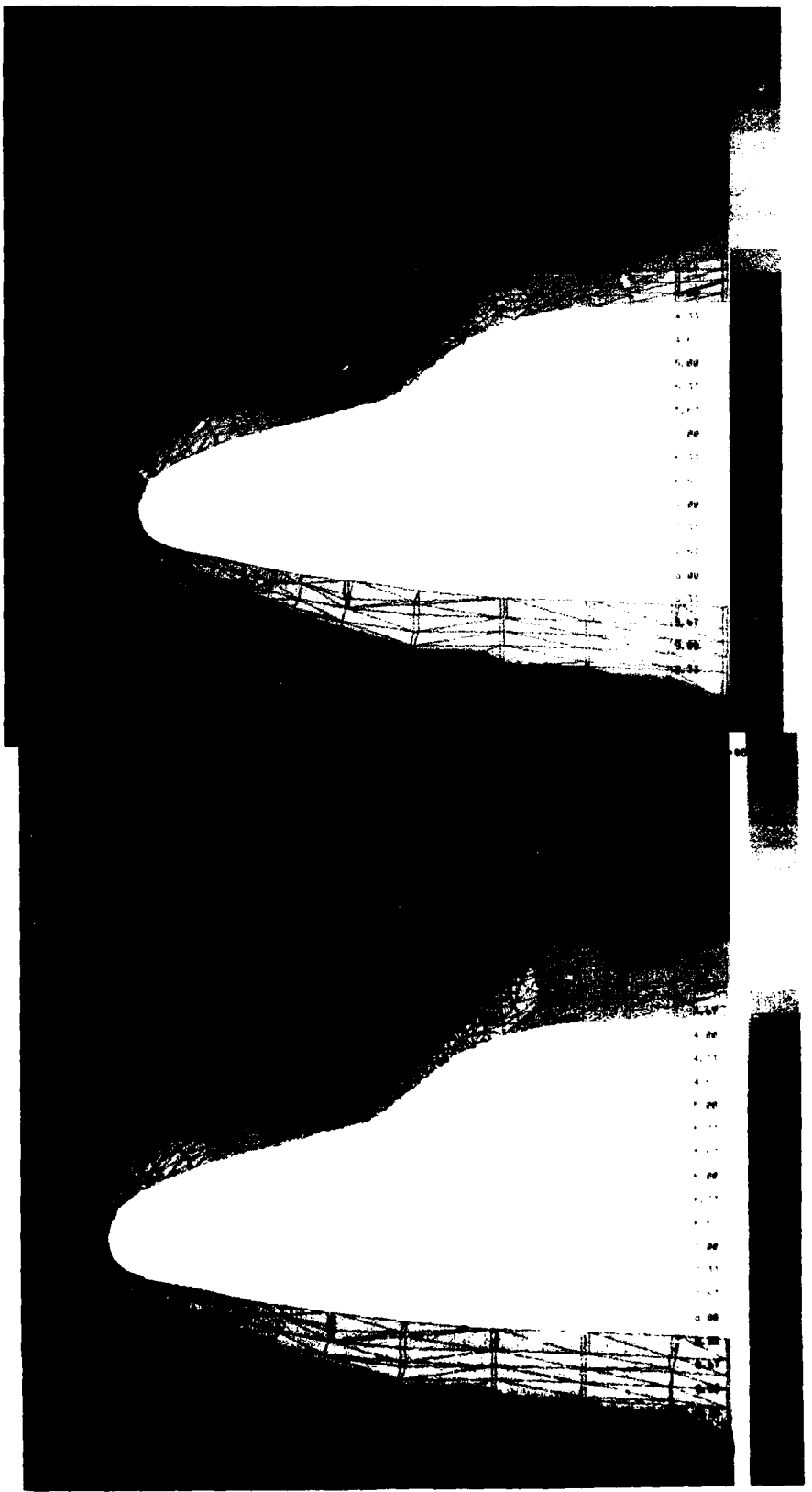


Figure 3

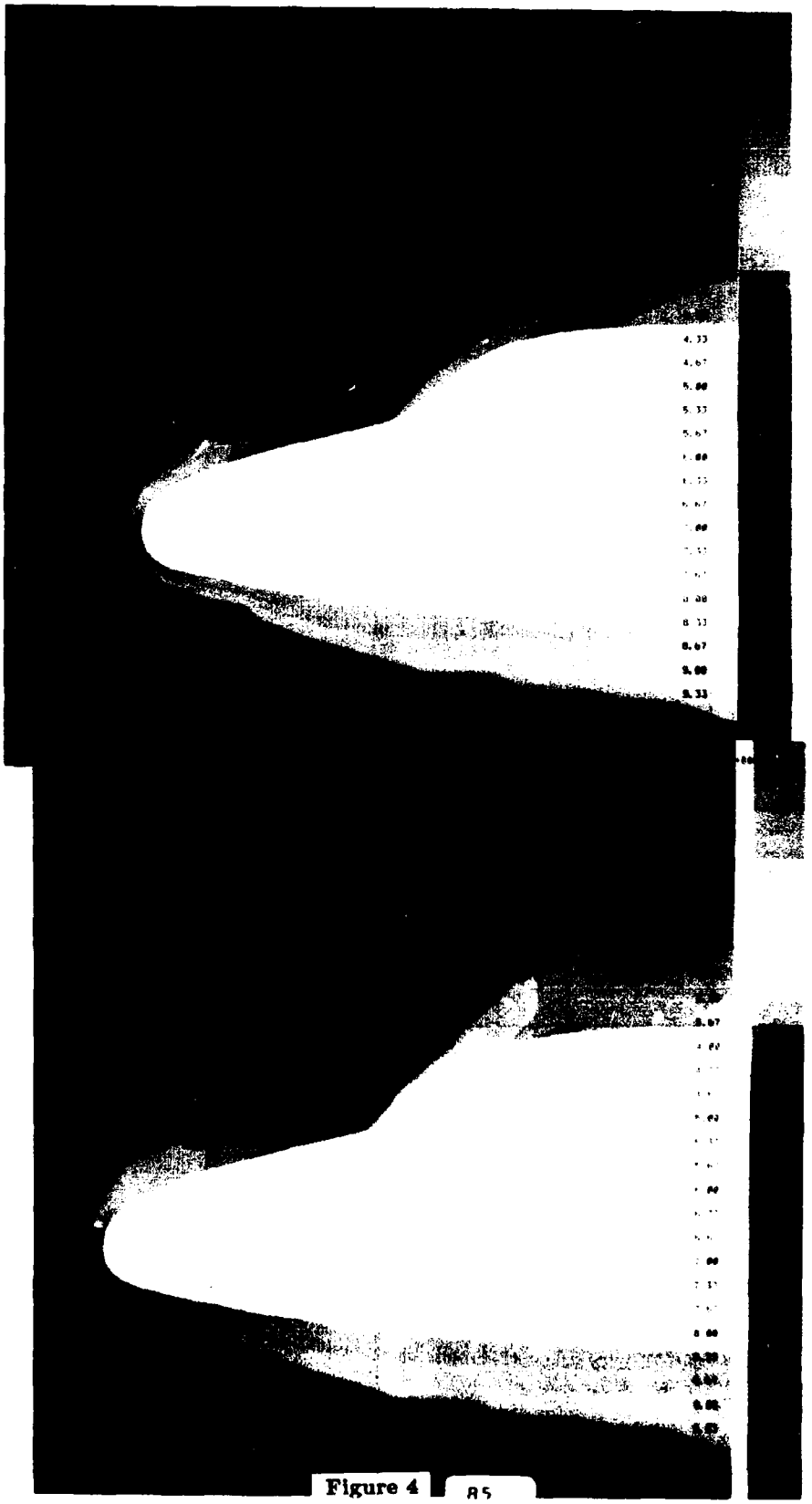


Figure 4

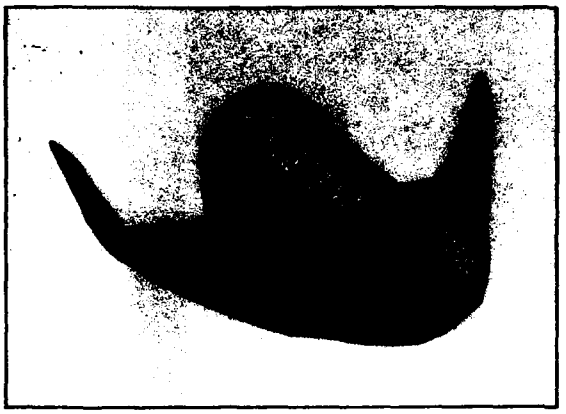


Figure 5

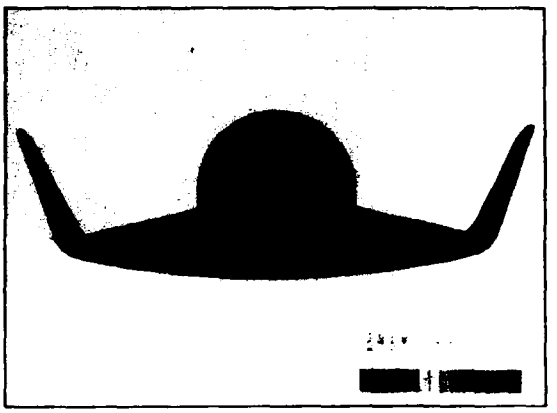


Figure 6

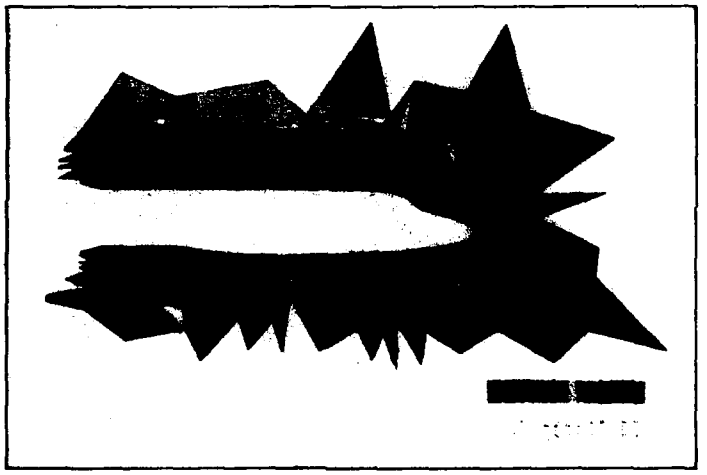


Figure 7

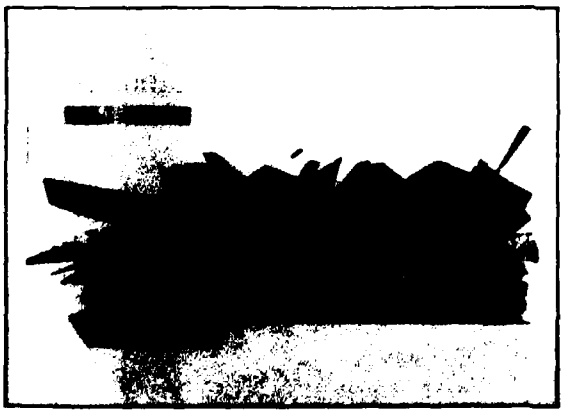


Figure 8

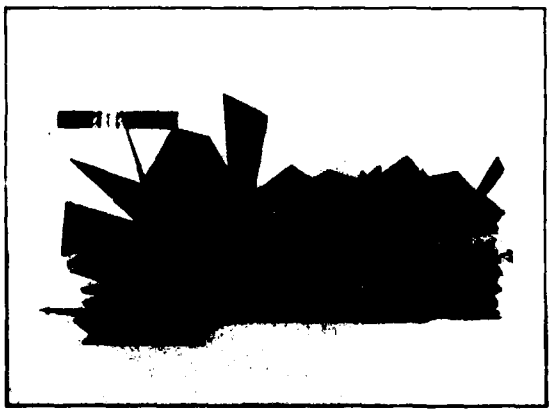


Figure 9

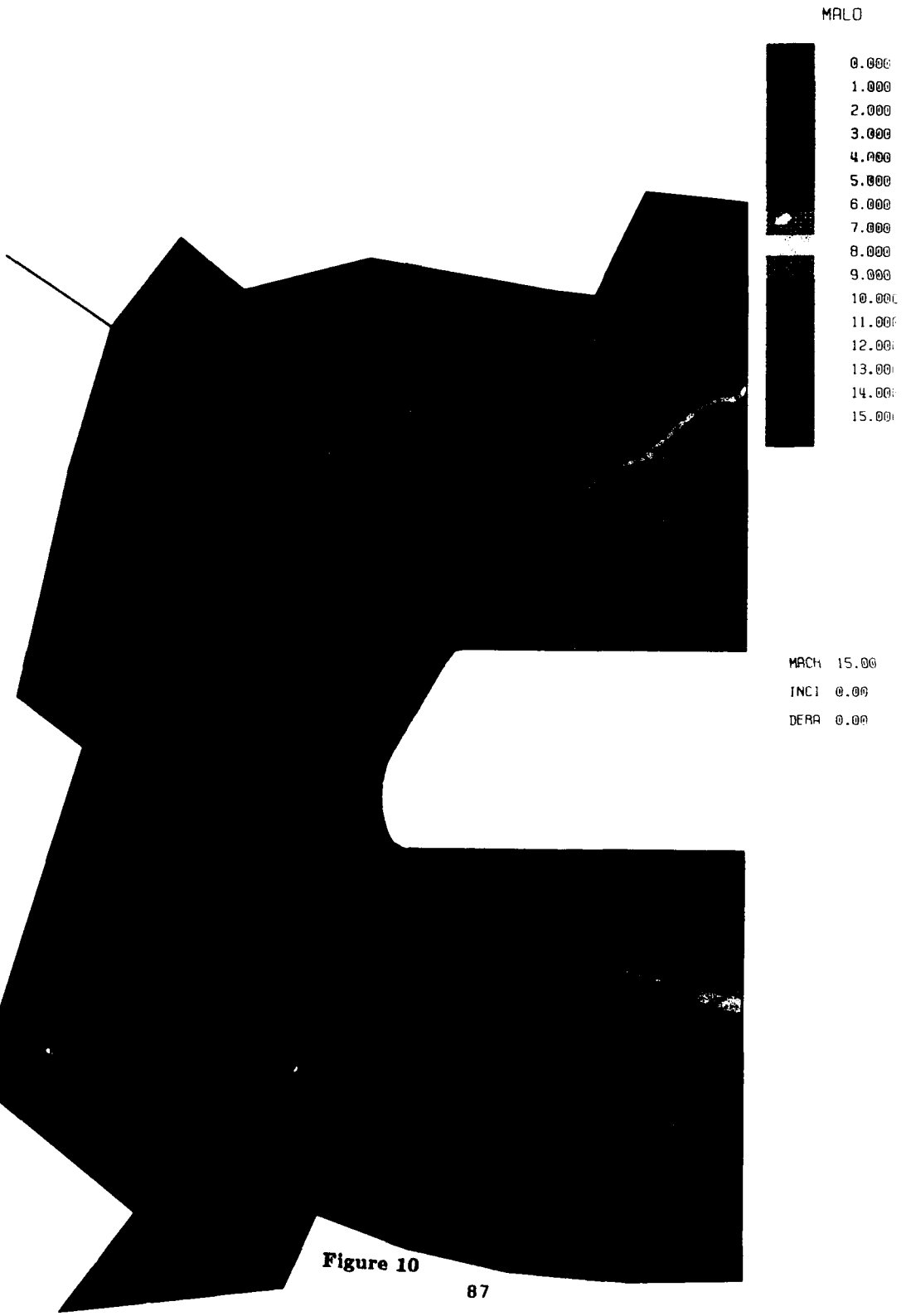

MALO
0.000
1.000
2.000
3.000
4.000
5.000
6.000
7.000
8.000
9.000
10.000
11.000
12.000
13.000
14.000
15.000
MACH 15.00
INCI 0.00
DERR 0.00

Figure 10

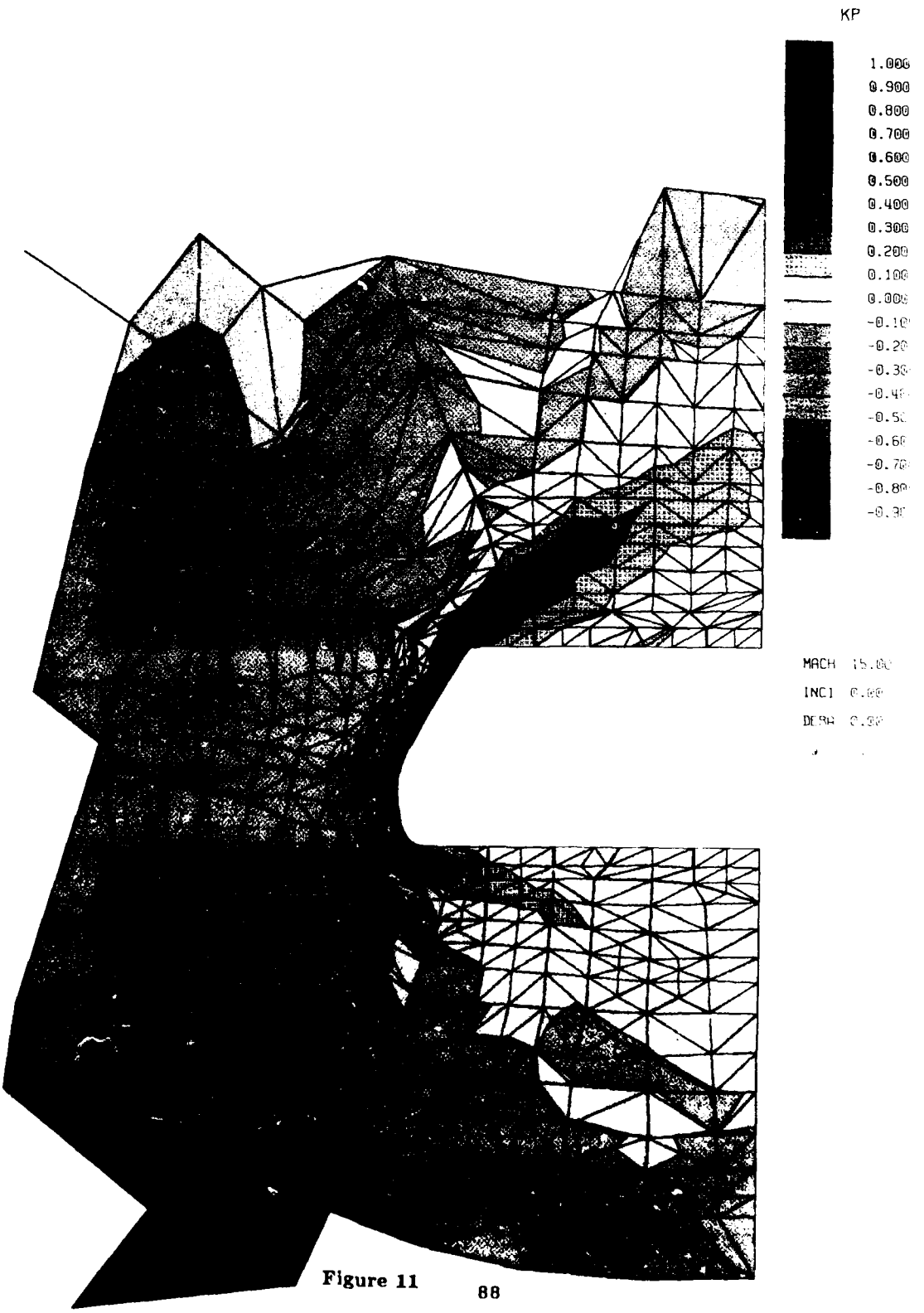

KP
1.000
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
-0.10
-0.20
-0.30
-0.40
-0.50
-0.60
-0.70
-0.80
-0.90
MACH 15.00
INCI 0.00
DEGR 0.00

Figure 11

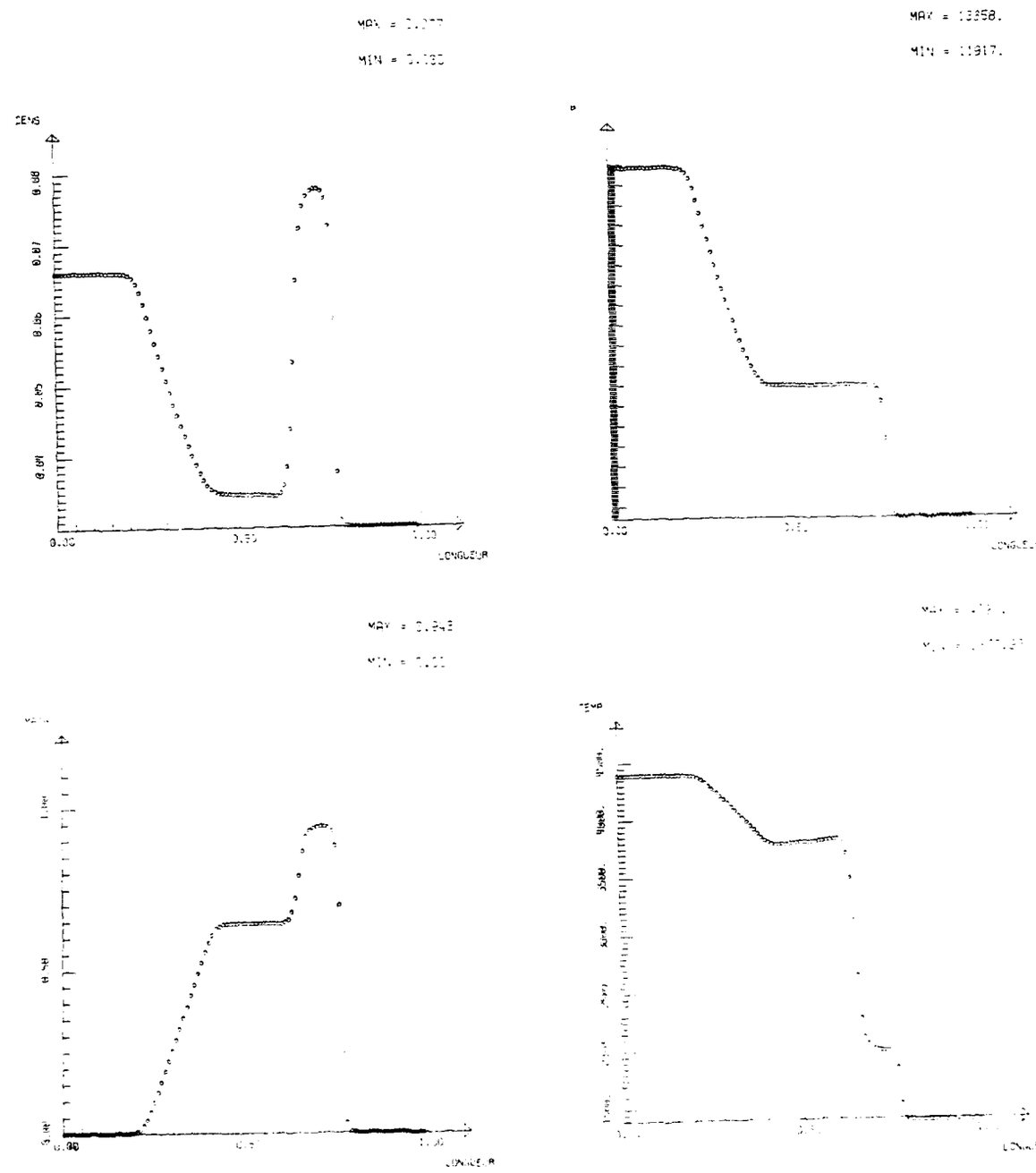


Figure 12

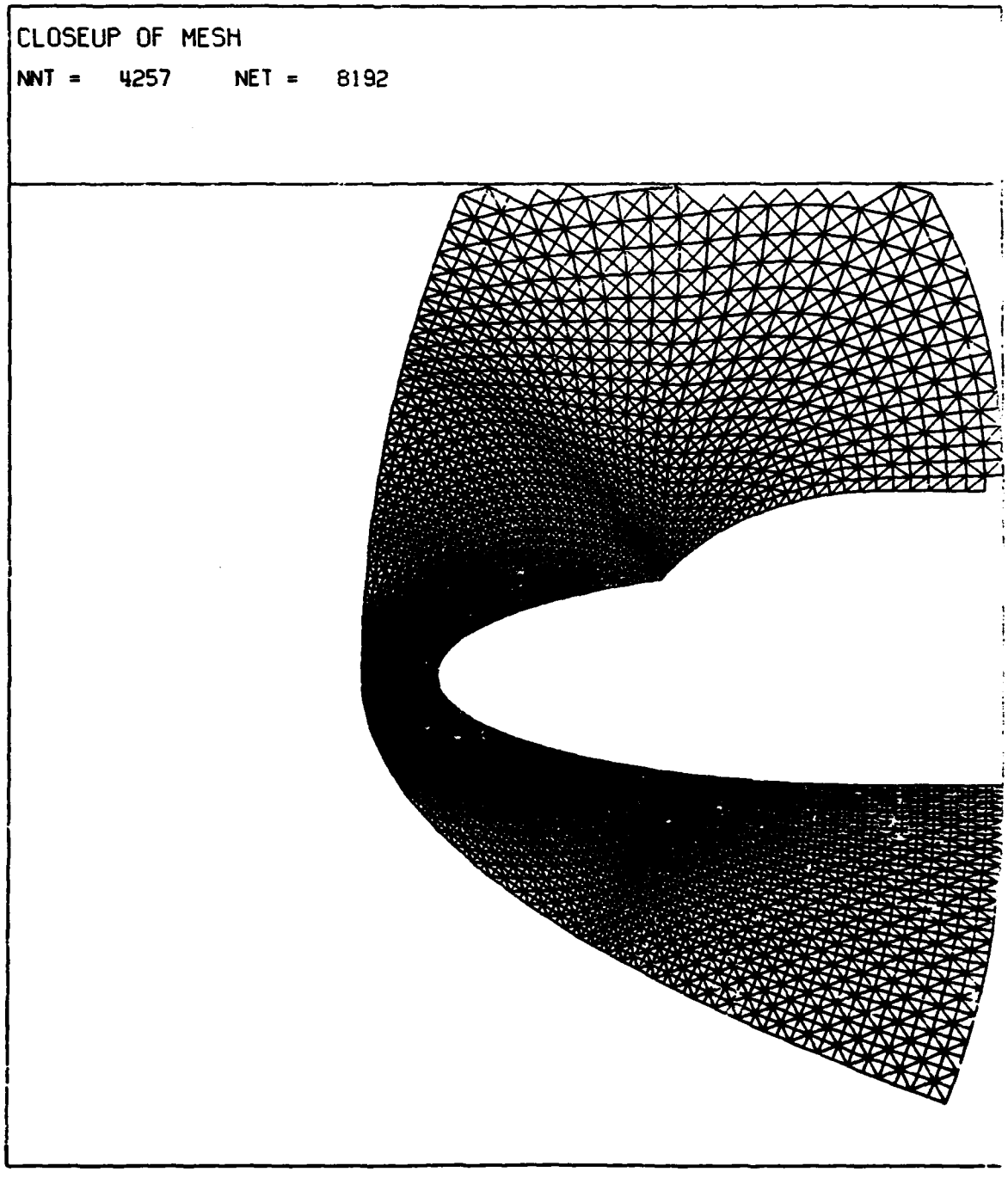


Figure 13

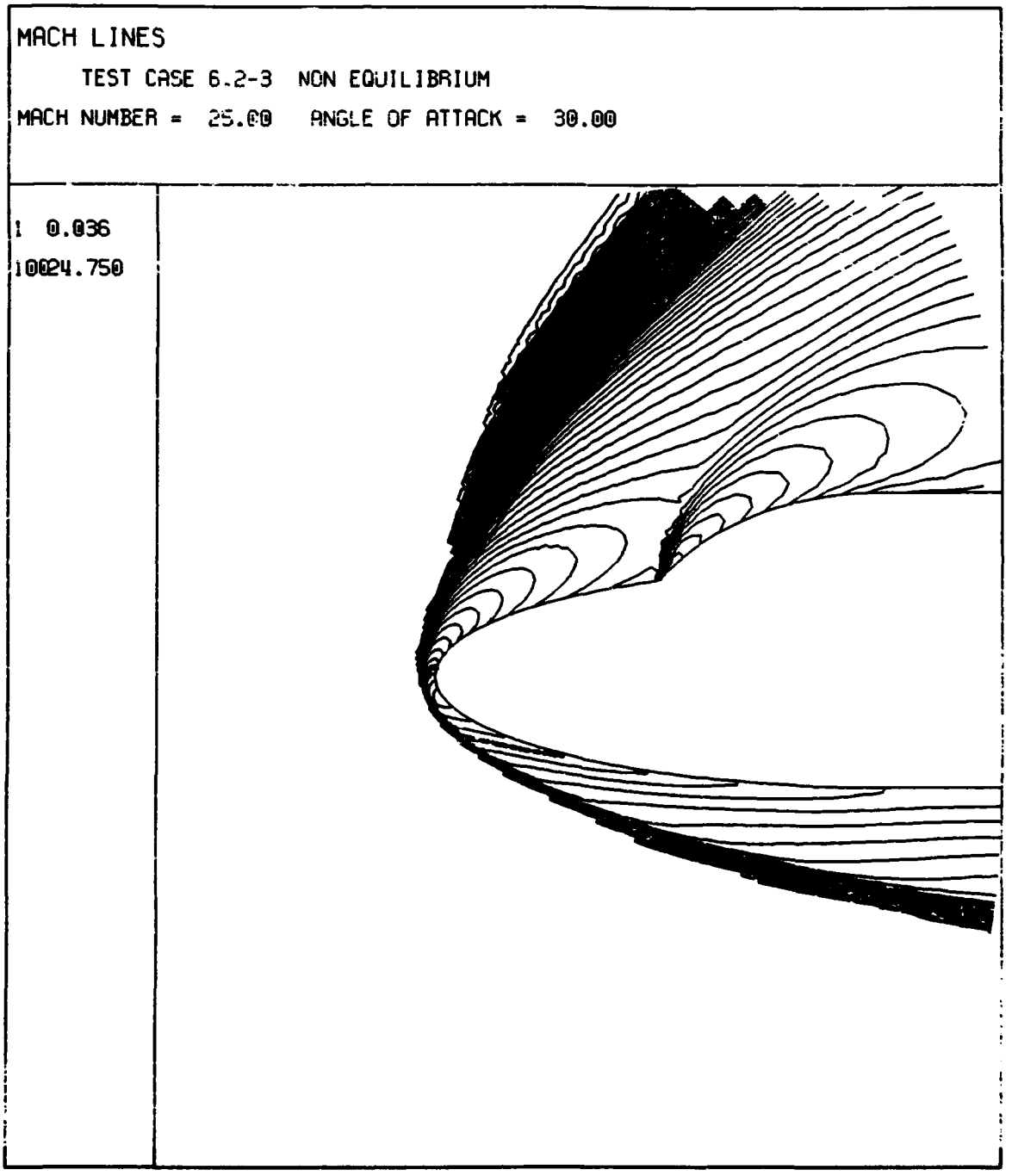

MACH LINES
TEST CASE 6.2-3 NON EQUILIBRIUM
MACH NUMBER = 25.00 ANGLE OF ATTACK = 30.00
1 0.036
10024.750

Figure 14

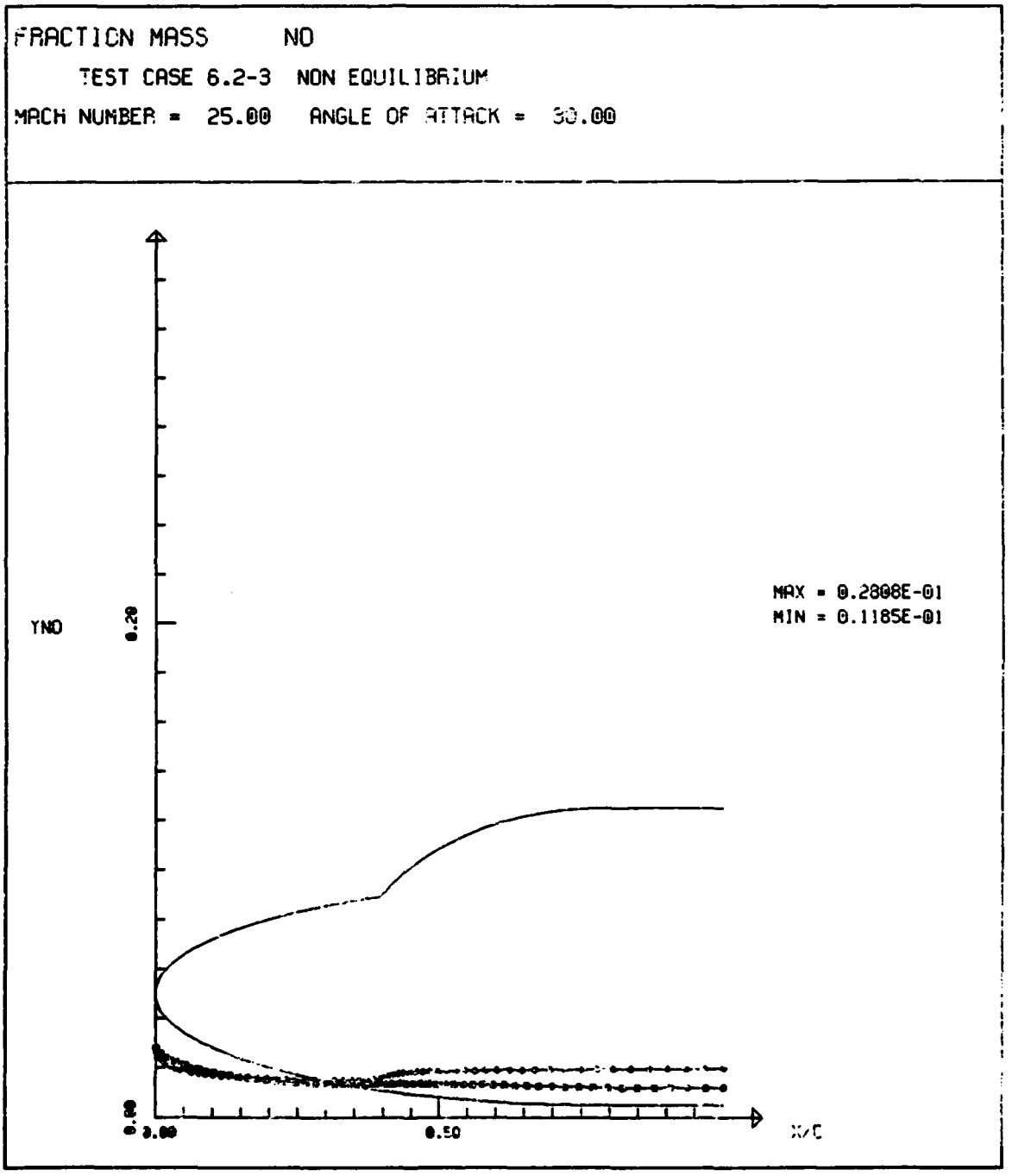

FRACTION MASS NO
TEST CASE 6.2-3 NON EQUILIBRIUM
MACH NUMBER = 25.00 ANGLE OF ATTACK = 30.00
YNO
0.20
0.00
0.00
0.50
X/C
MAX = 0.2808E-01
MIN = 0.1185E-01

Figure 15

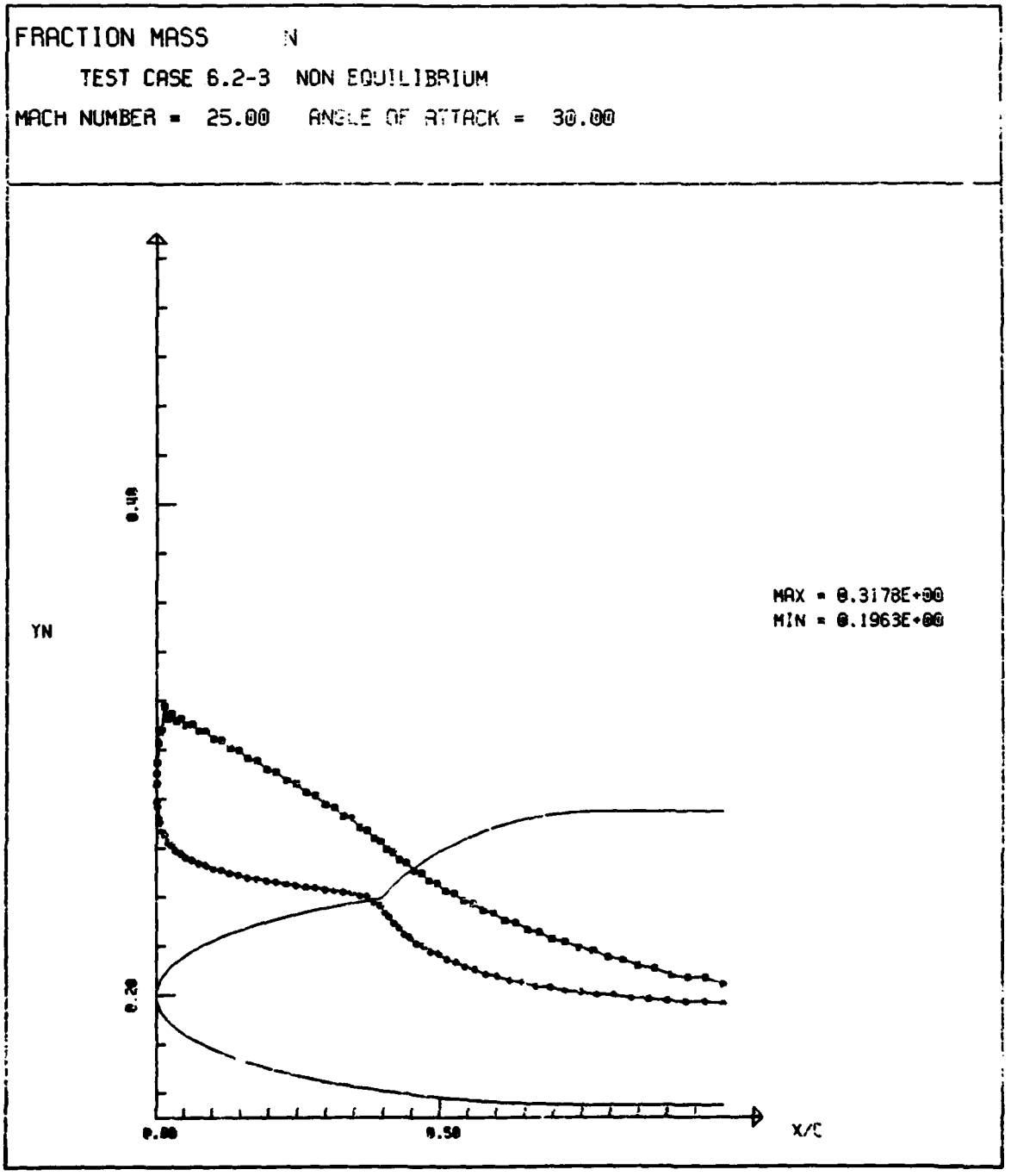

FRACTION MASS N
TEST CASE 6.2-3 NON EQUILIBRIUM
MACH NUMBER = 25.00 ANGLE OF ATTACK = 30.00
YN
0.40
0.20
MAX = 0.3178E+00
MIN = 0.1963E+00
0.00
0.50
X/C

Figure 16

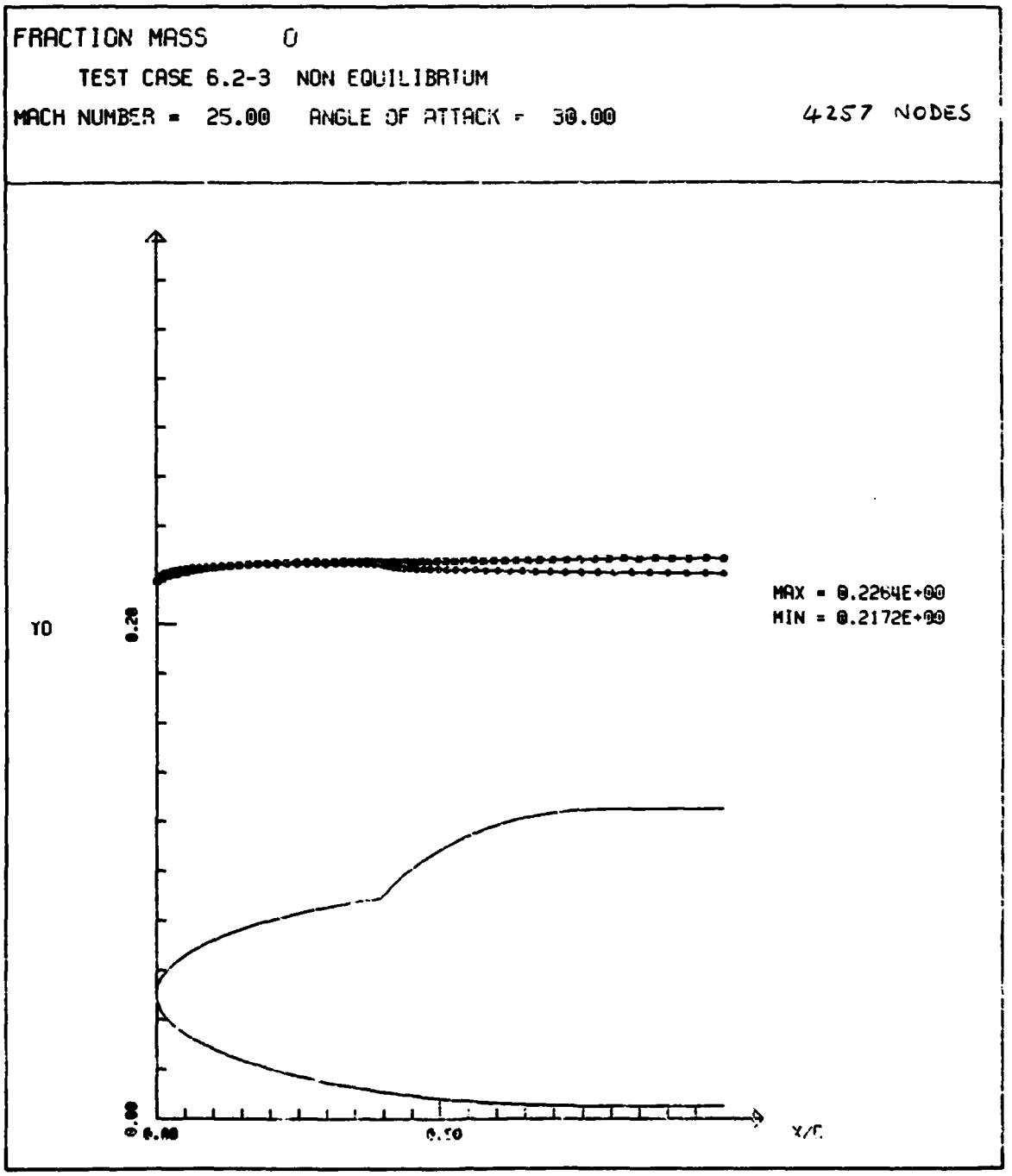

FRACTION MASS 0
TEST CASE 6.2-3 NON EQUILIBRIUM
MACH NUMBER = 25.00 ANGLE OF ATTACK = 30.00
4257 NODES
YO
0.20
0.00
0.00
0.50
X/L
MAX = 0.2264E+00
MIN = 0.2172E+00

Figure 17

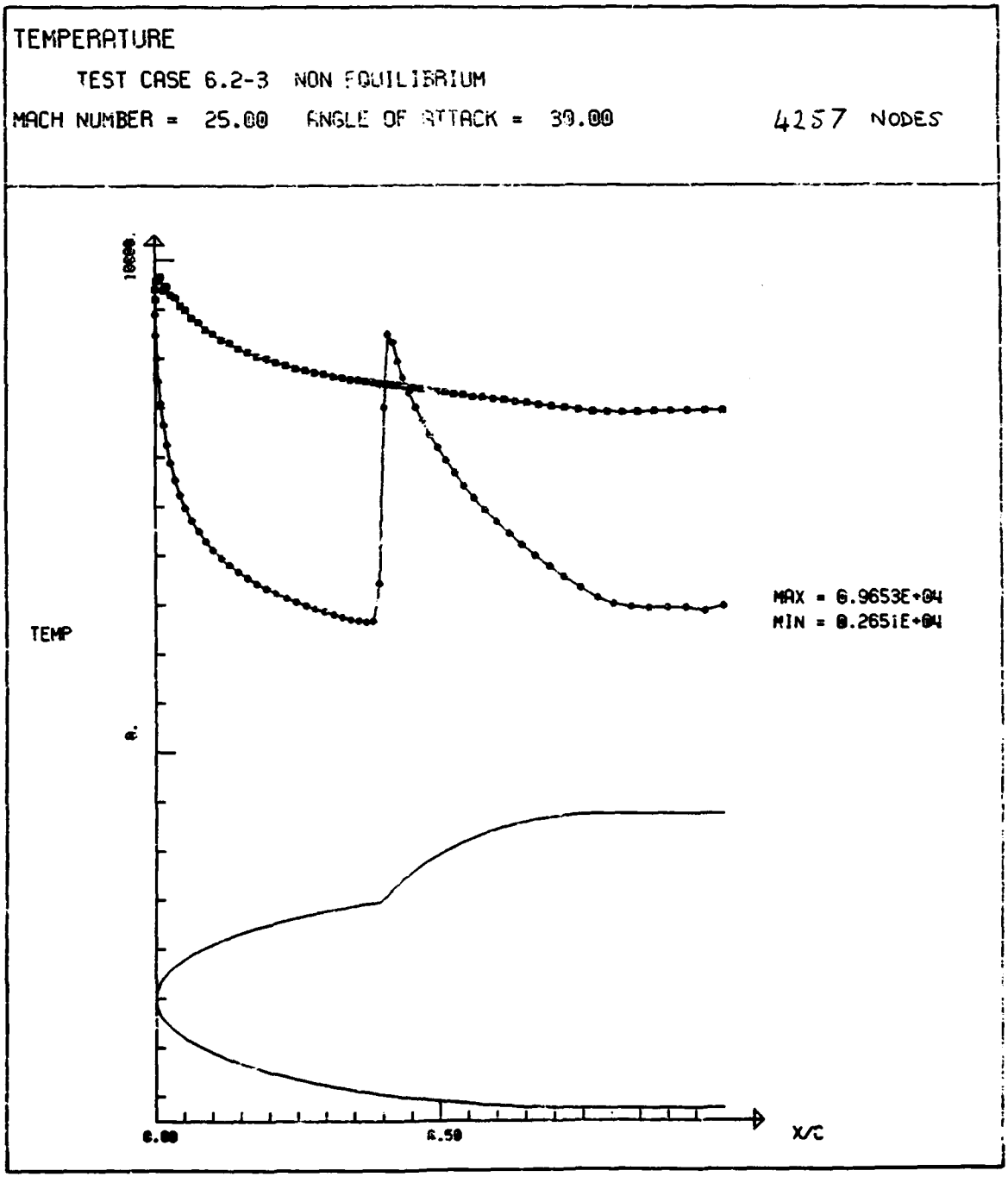

TEMPERATURE
TEST CASE 6.2-3 NON EQUILIBRIUM
MACH NUMBER = 25.00 ANGLE OF ATTACK = 30.00
4257 NODES
TEMP
MAX = 6.9653E+04
MIN = 8.2651E+04
0.00
0.50
X/C

Figure 18

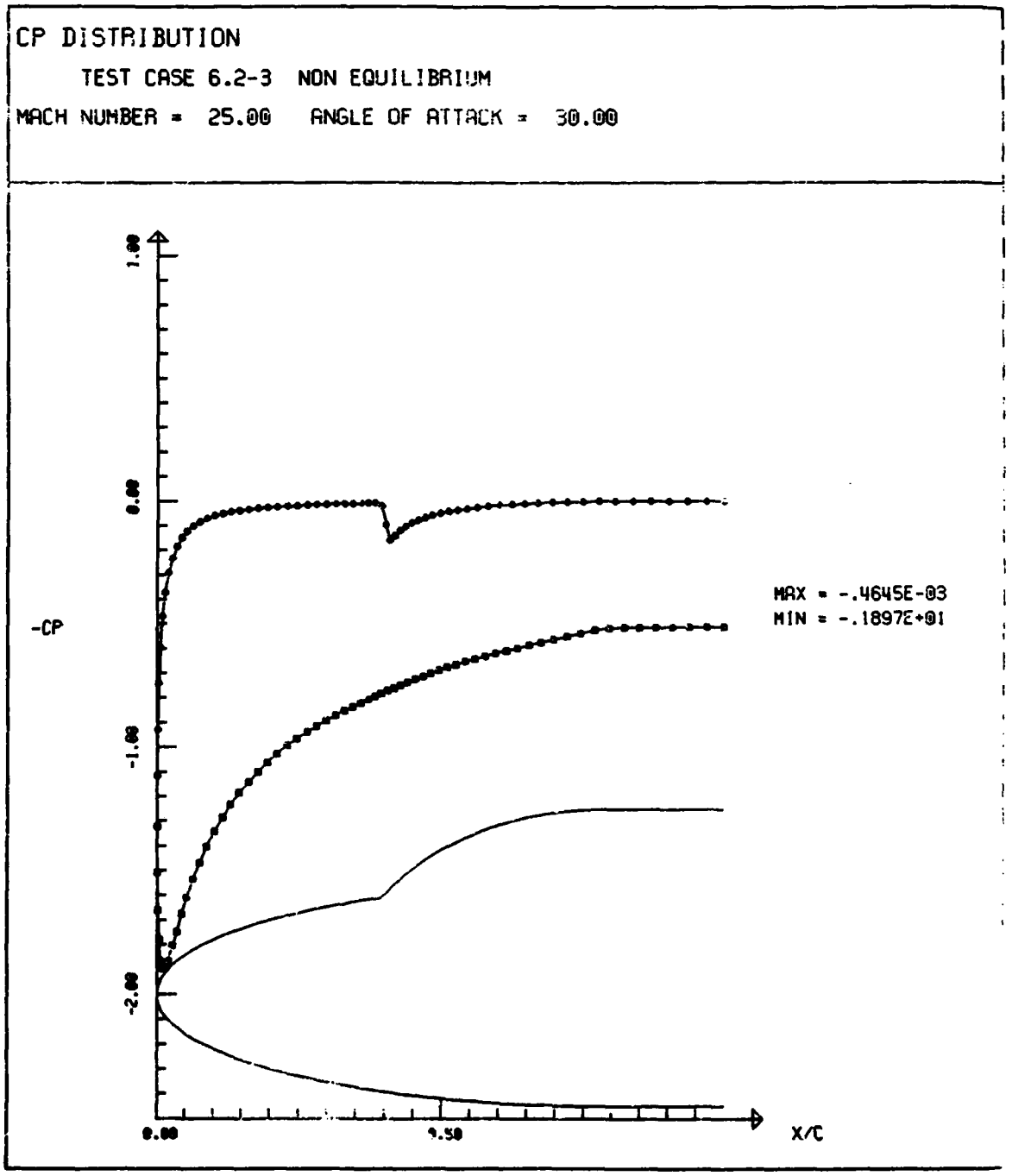

CP DISTRIBUTION
TEST CASE 6.2-3 NON EQUILIBRIUM
MACH NUMBER = 25.00 ANGLE OF ATTACK = 30.00
-CP
1.00
0.00
-1.00
-2.00
0.00
0.50
X/C
MAX = -.4645E-03
MIN = -.1897E+01

Figure 19

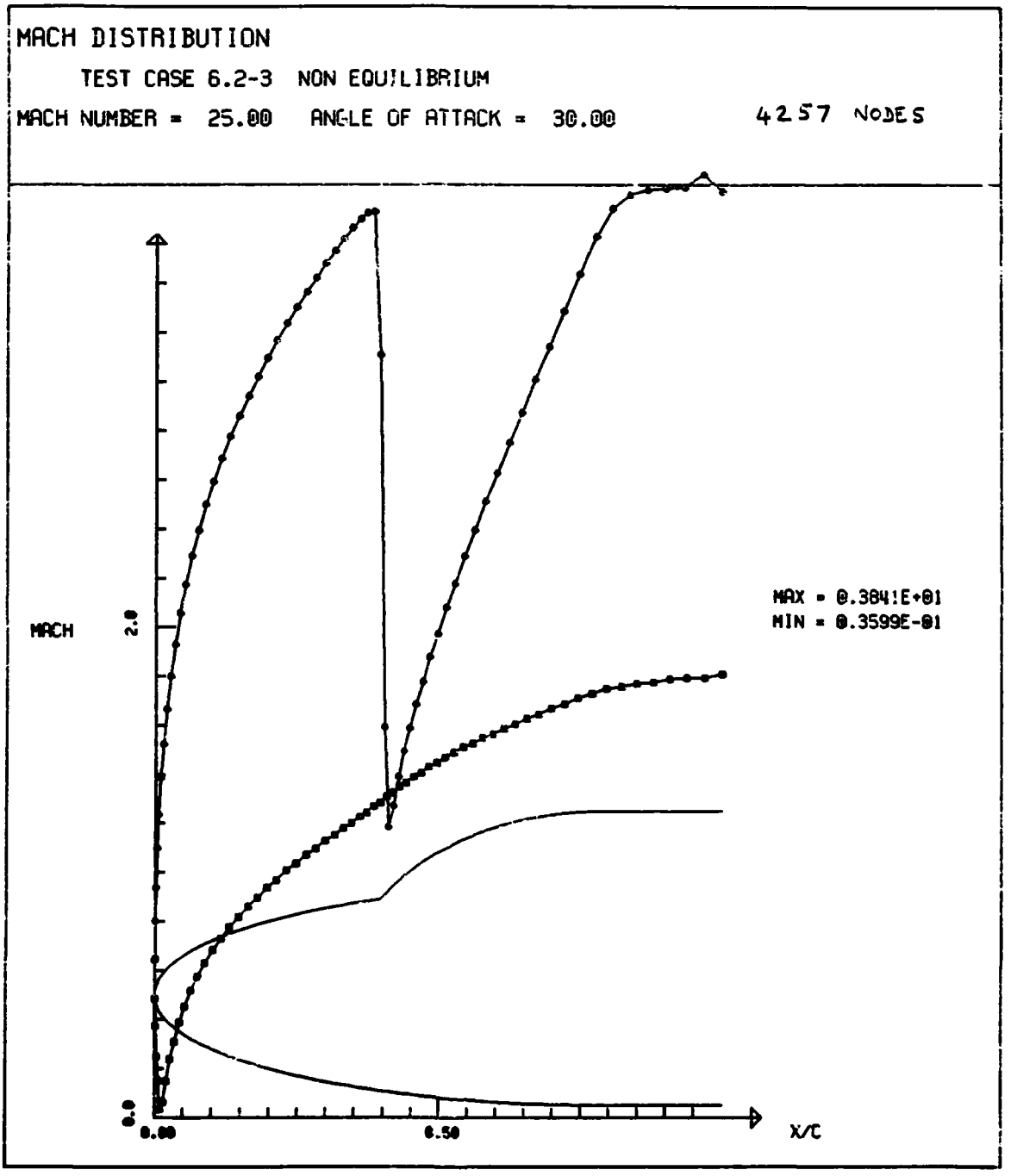

MACH DISTRIBUTION
TEST CASE 6.2-3 NON EQUILIBRIUM
MACH NUMBER = 25.00 ANGLE OF ATTACK = 30.00
4257 NODES
MACH
2.0
0.0
0.00
0.50
X/C
MAX = 0.3841E+01
MIN = 0.3599E-01

Figure 20

Compressible Flow Algorithms on Structured/Unstructured Grids

by

Robert W. Walters

Virginia Polytechnic Institute and State University
Blacksburg, Virginia

An invited lecture presented at the
Ninth International Conference on
Computing Methods in Applied Sciences and Engineering
January 29 - February 2, 1990
Paris, France

Compressible Flow Algorithms on Structured/Unstructured Grids

Robert W. Walters
Department of Aerospace and Ocean Engineering
Virginia Polytechnic Institute and State University

Abstract

Various techniques for implementing upwind flux-split schemes for the Euler and Navier-Stokes equations on unstructured meshes are reviewed. The development of a space-marching technique on hybrid structured/unstructured meshes is presented. In addition, time integration algorithms on unstructured grids with an emphasis on convergence acceleration to the steady-state are compared. An m-stage Jameson style explicit Runge-Kutta scheme is used as a baseline comparison. Implicit schemes discussed include a highly vectorizable skyline sparse matrix solver, hybrid explicit/implicit time advancement schemes, and various relaxation strategies. Mesh adaptation techniques are also discussed. Results in both two- and three-dimensions are presented including a supersonic inlet calculation with complex wave interactions and a space-marching, inviscid simulation on an unstructured mesh about a high speed reconnaissance aircraft.

Nomenclature

a	speed of sound
c	mass concentration
D_{12}	binary diffusion coefficient
e	internal energy per unit mass
$\tilde{e}$	equilibrium portion of internal energy
e_n	nonequilibrium portion of internal energy
e_0	total internal energy per unit mass
$\dot{e}_n$	nonequilibrium energy production rate
$\hat{F}, \hat{G}, \hat{H}$	inviscid flux vectors
$\hat{F}_v \hat{G}_v, \hat{H}_v$	viscous flux vectors
h	enthalpy per unit mass
h_0	total enthalpy per unit mass
I	identity matrix
J	Jacobian of coordinate transformation
k	thermal conductivity
L, U	LU decomposition matrices
p	pressure
q	velocity magnitude

Q	vector of conserved variables
R	residual for time integration algorithms
t	time
T	temperature
u, v, w	cartesian components of velocity
$\tilde{u}$, $\tilde{v}$, $\tilde{w}$	contravariant components of velocity
V	cell volume
$\dot{w}$	chemical production rate
W	vector of production rates
α	weighting coefficient
$\nabla(*)$	gradient$(*)$
Δ	finite difference operator
μ	dynamic viscosity
ξ, η, ζ	generalized space coordinates
$\tilde{\xi}$, $\tilde{\eta}$, $\tilde{\zeta}$	direction cosines
ρ	density

Introduction

There has been a clear trend in recent years toward the development of algorithms for computational fluid dynamic (CFD) simulations that have significant flexibility in modeling problems with complex geometries and/or complex physics. This has led several researchers away from structured (or logical) indexing schemes for addressing mesh elements to generalized indexing schemes frequently referred to as unstructured techniques [1-7].

This paper discusses some of the research that has taken place during the past two years at VPI&SU concerning algorithm development on unstructured and hybrid (structured/unstructured) grids for compressible flow simulations. Three primary areas have been investigated and will be briefly discussed. They are: 1) implicit time integration schemes, 2) mesh adaptation, and 3) space-marching methods on hybrid grids. The work presented herein has been heavily influenced by contributions from many researchers including, but not limited to, the efforts of A. Jameson, R. Löhner, P. Roe, B. Van Leer, T. Barth, B. Stouflett, B. Grossman, and J. L. Thomas and their co-workers. In the authors opinion, this work represents a combination of some of the best ideas presented by these people.

The following sections provide a brief discussion of the governing equations considered along with a description of cell-vertex and cell-centered spatial discretizations. Various time integrations schemes are presented and compared for a simple transonic test problem. Results with and without mesh adaptation for a supersonic inlet are shown. Finally, results from a novel space-marching method applied to a high speed reconnaissance aircraft are discussed.

Governing Equations

The equations of motion of interest in this paper are the full Navier-Stokes (FNS) equations and subsets thereof including the Thin-Layer Navier-Stokes (TLNS), Parabolized Navier-Stokes (PNS), and the Euler equations. The integral form of these equations may be written in the common form:

$$\frac{\partial}{\partial t}\int\int\int_V Q dV + \oint_s \vec{F}\cdot\hat{n} ds = \int\int\int_V W dV \tag{1}$$

where Q is the vector of dependent variables, W is a source term, and $\vec{F}\cdot\hat{n}$ represents the flux of mass, momentum, and energy out of the control volume V through the surface, S, with $\hat{n}$ an outward unit normal vector from S. The algorithms discussed here always directly discretize the integral form of the governing equations, but it is frequently convenient for discussion purposes to rewrite (1) in the differential form

$$\frac{\partial Q}{\partial t} + \frac{\partial(\hat{F}-\hat{F}_v)}{\partial\xi} + \frac{\partial(\hat{G}-\hat{G}_v)}{\partial\eta} + \frac{\partial(\hat{H}-\hat{H}_v)}{\partial\zeta} = W \tag{2a}$$

where

$$Q = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \vdots \\ \rho_N \\ \rho u \\ \rho v \\ \rho w \\ \rho_1 e_{n_1} \\ \vdots \\ \rho_M e_{n_M} \\ \rho e_0 \end{pmatrix}, \qquad W = \begin{pmatrix} \dot{w}_1 \\ \dot{w}_2 \\ \vdots \\ \vdots \\ \dot{w}_N \\ 0 \\ 0 \\ 0 \\ \rho_1 \dot{e}_{n_1} + e_{n_1}\dot{w}_1 \\ \vdots \\ \rho_M \dot{e}_{n_M} + e_{n_M}\dot{w}_M \\ 0 \end{pmatrix} \tag{2b}$$

For generality, a source term, W, due to chemical reactions and nonequilibrium thermodynamics has been included. A thorough discussion of this formulation can be found in [8]. The common perfect gas form is a simple subset of this more general case. The vectors $\hat{F}$, $\hat{G}$, $\hat{H}$ represent the inviscid and pressure terms and $\hat{F}_v$, $\hat{G}_v$, $\hat{H}_v$ contain the shear stress and heat flux contributions. As an example, for an N-species, finite-rate, chemically reacting flow in which M of the species are considered to be in vibrational nonequilibrium,

one may write

$$\widehat{F} = \frac{|\nabla\xi|}{J}\begin{pmatrix} \rho_1\tilde{u} \\ \rho_2\tilde{u} \\ \vdots \\ \vdots \\ \vdots \\ \rho_N\tilde{u} \\ \rho u\tilde{u} + \tilde{\xi}_x p \\ \rho v\tilde{u} + \tilde{\xi}_y p \\ \rho w\tilde{u} + \tilde{\xi}_z p \\ \rho_1 e_{n_1}\tilde{u} \\ \vdots \\ \rho_M e_{n_M}\tilde{u} \\ \rho\tilde{u}h_0 \end{pmatrix}, \quad \widehat{F}_v = \frac{|\nabla\xi|^2}{J}\begin{pmatrix} \rho D_{12}c_{1\xi} \\ \rho D_{12}c_{2\xi} \\ \vdots \\ \vdots \\ \vdots \\ \rho D_{12}c_{N\xi} \\ \mu u_\xi + \mu\tilde{u}_\xi\tilde{\xi}_x/3 \\ \mu v_\xi + \mu\tilde{u}_\xi\tilde{\xi}_y/3 \\ \mu w_\xi + \mu\tilde{u}_\xi\tilde{\xi}_z/3 \\ \rho D_{12}c_{1\xi}e_{n_1} + k_{n_1}T_{1_\xi} \\ \vdots \\ \rho D_{12}c_{1\xi}e_{n_M} + k_{n_M}T_{M_\xi} \\ \Theta \end{pmatrix}$$

where

$$\Theta = \left[\frac{\mu(q^2)_\xi}{2} + kT_\xi + \sum_{j=1}^{M} k_{n_j}T_{j_\xi} + \rho\sum_{i=1}^{N} h_i D_{12}c_{i\xi}\right] + \frac{\mu\tilde{u}_\xi\tilde{u}}{3}$$

and

$$q^2 = u^2 + v^2 + w^2$$

$$\tilde{u}_\xi \equiv \tilde{\xi}_x u_\xi + \tilde{\xi}_y v_\xi + \tilde{\xi}_z w_\xi$$

The other flux vectors can be written in a similar manner. In the above, some simplifying assumptions have been made, e.g. the use of a binary diffusion coefficient as opposed to a multi-component diffusion model. However, the particular choice of a chemistry and thermodynamics model is up to the user. Discussions of various models and their practical applications can be found in [9-10].

Spatial Discretization

An approach that has become popular during the past few years for discretizing hyperbolic conservation laws is the so-called upwind technique in which the numerics attempt to model the physics by differencing the characteristic information independently. One advantage of such a formulation is the increased robustness of codes that incorporate this technology, particularly in the high-speed regime. There are two general classes of upwind methods, flux vector splitting (FVS) and flux difference splitting (FDS). Among the FVS schemes, the Steger-Warming [11] and Van Leer [12] splittings are the best known. Roe's scheme [13] is by far the most popular FDS technique. The splittings were originally developed for the one-dimensional flow of a perfect gas and have been extended to three-dimensional generalized coordinates and to thermo-chemical nonequilibrium flows by several researchers including the author [c.f. 8].

Upwind techniques can be implemented on structured, unstructured, or hybrid meshes (i.e., grids that contain features of both). Both cell-centered and cell-vertex discretizations can also be developed and in a variety of ways. In this paper, only one cell-centered approach and one cell-vertex method will be discussed in two dimensions on a triangular mesh followed by a three-dimensional technique on a hybrid mesh. The latter approach has found particular utility for high speed space-marching simulations.

The cell-centered method shown in Fig. 1 stores the dependent variables at the centroid of each triangle, the edges of the triangle define the faces of the control volume.

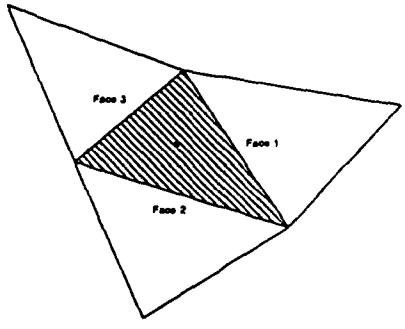


Figure 1. Typical control volume for a cell-centered technique.

The cell-vertex scheme depicted in Fig. 2 stores the conserved variables at the vertices of the triangles. The control volume is formed by connecting the centroids of the triangles surrounding each vertex to the midpoints of the edges. This makes each vertex the approximate cell center of the control volume created around it.

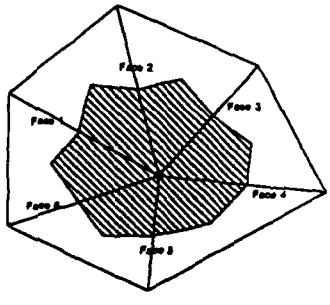


Figure 2. Typical control volume for a cell-vertex technique.

It should be noted that these techniques can be applied to both triangles and quadrilaterals in two-dimensions, and to tetrahedra and hexagons in three-dimensions and they are independent of the manner in which the individual mesh elements and control volumes are addressed (i.e. structured or unstructured). Moreover, these methods can be applied to even more general elements and control volumes such as the one shown in Fig. 3.

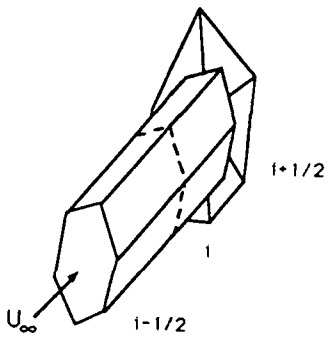


Figure 3. Control volume on a structured/unstructured mesh.

For space-marching applications, one needs not only a direction for which the inviscid field is entirely supersonic, but also a mesh that contains control surfaces which can be used to march the solution in space. General three-dimensional unstructured grids do not inherently contain such surfaces whereas it is easy to construct structured meshes that meet this requirement. Thus, space-marching algorithms on structured meshes have been developed and are popular because the computing times required for space-marching calculations are much less than that required by standard global iteration techniques. However, the requirement of a logical indexing scheme results in a loss of geometric modeling flexibility which is a severe drawback of the structured grid techniques.

The control volume shown in Fig. 3 has been obtained by constructing a hybrid structured/unstructured three-dimensional grid. It consists of two-dimensional unstructured triangular cell faces in each cross-flow plane which have been stacked together in the streamwise (supersonic) direction thus giving rise to mesh elements that are five-sided prisms. This grid can be used for developing space-marching methods since each plane can be a surface on which the Mach number based on the local contravariant component of velocity normal to all of the cell faces is supersonic. The advantage of such a formulation is that it combines the increased numerical efficiency of space-marching algorithms with the geometric flexibilty of an unstructured indexing scheme. One can also construct other types of hybrid meshes for marching applications that use different elements to form the base grid (e.g. tetrahedra) as long as surfaces can be constructed for advancing the solution in the streamwise direction.

With any of these discretizations, one may replace the integral form of the governing equations by the semi-discrete approximation

$$V_i \frac{\partial \langle Q_i \rangle}{\partial t} = V_i W_i - \sum_{j=\kappa(i)} F_{i,j} \Delta s_{i,j} \equiv R_i \tag{3}$$

where $\langle Q_i \rangle$ is the volume average of Q in the i^{th} control volume, V_i is the control volume, $F_{i,j}$ is the flux out of the element i though face j, $\Delta s_{i,j}$ is the area of the j^{th} cell face of volume i, and $\kappa(i)$ is a list of neighboring cells.

In order to apply an upwind scheme, it is necessary to obtain two distinct fluid dynamic states on each side of a cell face, frequently referred to as the left and right states. For first order accuracy, the left and right states may be obtained by assuming a piecewise constant distribution of the state variables within the control volumes. Thus, one simply obtains the left and right states from the either the cell-centered or cell-vertex volume-averaged values immediately adjacent to the cell face at which the numerical flux is sought depending on the scheme employed.

In order to increase the spatial accuracy, a piecewise linear distribution of the data may be assumed. From this reconstruction, more accurate left and right states may be determined and their values limited in such a way that no new extrema are generated in an effort to prevent spurious oscillations in the vicinity of discontinuities. This linear distribution of the cell averaged flow variables can be represented by

$$Q(x,y) = Q(x_0,y_0) + \nabla Q \cdot \vec{r} \tag{4}$$

where $\vec{r}$ is the vector from the cell center (x_0,y_0) to any point (x,y) in the cell, and ∇Q represents the solution gradient in the cell. Note that this equation is simply the first-order accurate Taylor approximation plus a higher-order correction. For each cell, since the solution gradient ∇Q is constant, it can be computed from

$$\nabla Q_A = \frac{1}{S_\Omega} \oint_\Omega Q \, \hat{n} \, d\Omega$$

where S_Ω is the area contained in the path of integration. For the cell-centered case, the path chosen passes through the centroids of the cells B,C, and D which surround cell A, as indicated in figure 4.

For the vertex scheme, the path connects the neighboring vertices B,C,D,E,F, and G as shown in figure 5. Both paths ensure exact calculation of ∇Q_A when Q varies linearly.

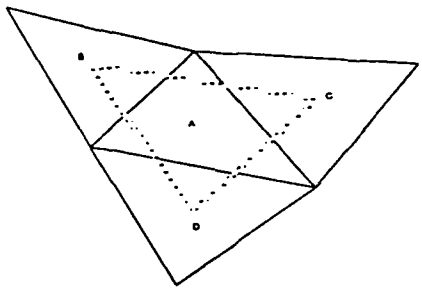


Figure 4. Integration path for cell-centered gradient calculation

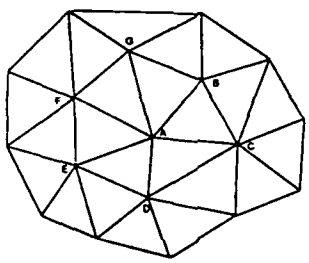


Figure 5. Integration path for cell-vertex gradient calculation

One can consider a limited version of the linear function about the centroid of cell A

$$Q(x,y)_A = Q(x_0,y_0)_A + \Phi_A \nabla Q_A \cdot \vec{r}_A, \qquad \Phi \ \epsilon \ [0,1]$$

In order to find the value of Φ_A, a monotonicity principle is enforced on the unlimited quantities $Q_{A_i} = Q(x_i,y_i)_A$ calculated in (4) at the faces of cell A. It requires that the values computed at the faces must not exceed the maximum and minimum of neighboring cell values including the value in cell A, i.e.,

$$Q_A^{min} \leq Q_{A_i} \leq Q_A^{max}$$

where $Q_A^{min} = min(Q_A, Q_{neighbors})$ and $Q_A^{max} = max(Q_A, Q_{neighbors})$ Φ can now be

calculated for each face j of cell A as

$$\Phi_{A_j} = \begin{cases} min\left(1, \dfrac{Q_A^{max} - Q_A}{Q_j - Q_A}\right), & \text{if } Q_j - Q_A > 0 \\ min\left(1, \dfrac{Q_A^{min} - Q_A}{Q_j - Q_A}\right), & \text{if } Q_j - Q_A < 0 \\ 1 & \text{if } Q_j - Q_A = 0 \end{cases}$$

with $\Phi_A = min(\Phi_{A_j})$, where $j = \kappa(1), \kappa(2), \kappa(3), ..., \kappa(max)$. For a more in depth discussion of this higher-order accurate scheme, see Barth [3].

Temporal Discretization

In order to obtain a steady-state solution, the governing equations must be integrated in time. Seven time integration methods have been considered : a four stage explicit Runge-Kutta, a four stage Runge-Kutta with implicit residual smoothing, point Jacobi, point Gauss-Seidel, a block Gauss-Seidel type relaxation, a fully implicit LU decomposition, and a hybrid scheme which combines Runge-Kutta and LU decomposition. Jameson style Runge-Kutta is used here which can be written as:

$$\begin{aligned} Q^{(0)} &= Q^{(N)} \\ Q^{(1)} &= Q^{(0)} + \alpha_1 \frac{\Delta t}{V} R(Q^{(0)}) \\ Q^{(2)} &= Q^{(0)} + \alpha_2 \frac{\Delta t}{V} R(Q^{(1)}) \\ Q^{(3)} &= Q^{(0)} + \alpha_3 \frac{\Delta t}{V} R(Q^{(2)}) \\ Q^{(4)} &= Q^{(0)} + \alpha_4 \frac{\Delta t}{V} R(Q^{(3)}) \\ Q^{(N+1)} &= Q^{(4)} \end{aligned}$$

where $R(Q)$ is the right-hand side of (3) and α are weights. Convergence to the steady-state can be accelerated by using a local time-stepping technique in which the maximum permissible time step for each individual cell in the flow field is used, as dictated by local stability analysis. In addition, the Runge-Kutta scheme can be accelerated by applying implicit residual smoothing at every stage of the time integration. Residual smoothing is essentially a Laplacian filtering of the numerical values of the residuals. After every stage a new value of the residual is obtained from

$$\bar{R}_i = R_i + \epsilon \nabla^2 \bar{R}_i$$

where $\bar{R}_i$ is the Laplacian filtered value of R_i. The undivided Laplacian, $\nabla^2 \bar{R}_i$, can be represented on an unstructured mesh as:

$$\nabla^2 \bar{R}_i = \sum_{j=\kappa(i)} (\bar{R}_j - \bar{R}_i)$$

The resulting implicit equation for $\bar{R}_i$ is solved here by Jacobi iteration. Typically, two Jacobi iterations were performed with $\epsilon = 0.5$.

The point Jacobi, point Gauss-Seidel, block Gauss-Seidel, and LU decomposition schemes that have been studied utilize the Euler implicit time integration algorithm as a common starting point which can be represented in delta form as

$$V\frac{\Delta Q}{\Delta t} = R^{N+1}$$

where $\Delta Q = Q^{N+1} - Q^N$. The equation can be linearized and written as

$$A\Delta Q = R^N$$

where

$$A = \left(\frac{V}{\Delta t}I - \frac{\partial R^N}{\partial Q}\right).$$

The matrix is generally large and sparse and has a variable bandwidth. The linear system can be an approximate or exact linearization of the residual, R. Many relaxation schemes for solving the linear problem rewrite A as

$$A = M + D + N$$

where M is a lower triangular matrix, D is a diagonal matrix, and N is an upper triangluar matrix. With a point Jacobi method, the block matrices on the diagonal are inverted and multiplied by the right hand side to obtain

$$\Delta Q = D^{-1}R^N$$

To implement a point Gauss-Seidel method, the off-diagonal terms of A are multiplied by the current approximation to ΔQ and are subtracted from the residual. Point Gauss-Seidel can be written for $i = 1, ..., n$ as

$$\Delta Q_i^{(1)} = D_{i,i}^{-1}\left[R_i^N - \sum_{j=1}^{i-1} M_{i,j}\Delta Q_j^{(1)} - \sum_{j=i}^{n} N_{i,j}\Delta Q_j^{(0)}\right]$$

The superscripts on ΔQ refer to the inner iteration number of the Gauss-Seidel method on the linear system. Typically, $\Delta Q^{(0)}$ is an initial guess for the Gauss-Seidel solver and $\Delta Q^{(1)}$ is used to update Q^N to Q^{N+1}. Due to the recursive nature of the point Gauss-Seidel algorithm, complete vectorization of this method is not possible. Point Gauss-Seidel can be made symmetrical by sweeping through the list of vertices in the opposite direction before updating Q. The symmetrical point Gauss-Seidel can be written for $i = n, ..., 1$ as

$$\Delta Q_i^{(2)} = D_{i,i}^{-1}\left[R_i^N - \sum_{j=1}^{i} M_{i,j}\Delta Q_j^{(1)} - \sum_{j=i+1}^{n} N_{i,j}\Delta Q_j^{(2)}\right]$$

with $\Delta Q^{(2)}$ being used to update Q.

In order to perform the block Gauss-Seidel type relaxation used here, it is first useful to renumber the cells to get as many of the elements of A as possible into tridiagonal form. A typical matrix (associated with the transonic channel flow problem discussed later) is shown in figure 6 for a 1005 element mesh. The matrix can be subdivided into several subsections (denoted by the blocks) of variable length. A close up of the third section shows the essentially tridiagonal form. Each subsection is then solved by

$$T\Delta Q = R(Q^N, Q^{N+1})$$

where T denotes a tridiagonal submatrix, and the residual becomes a function of Q^N and Q^{N+1}. Since the blocks are independent of each other, the inversion of these submatrices can be vectorized over the number of blocks. When the matrix A is divided into subsections of variable length, a minimum allowable length is imposed. The inversion procedure is then vectorized for the elements in every block up to the minimum number of elements allowed, the remainder of the elements are then computed in scalar mode.

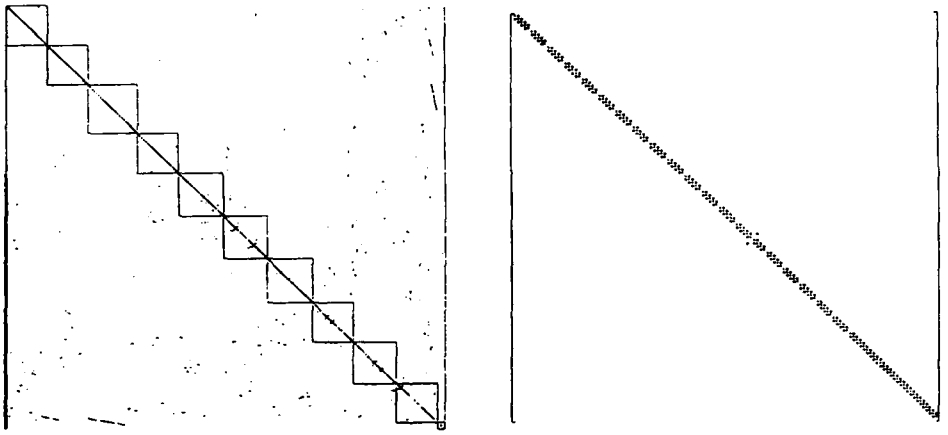


Figure 6. Structure of the linear system and close-up of the 3^{rd} block

Once the tridiagonal matrices have been inverted, the values of ΔQ are solved for sequentially over each section of the matrix A. The elements to the left of the diagonal on a forward sweep (or the elements to the right of the diagonal on a reverse sweep) through the matrix are included implicitly in the solution procedure. After the values for ΔQ of a subsection are calculated, the residual for the next subsection is computed using these new updated values resulting in a non-linear update of the residual.

The LU decomposition method solves the linear system exactly. A renumbering scheme (currently reverse Cuthill-McGee) is used as a preprocessor for the LU decomposition. The solution procedure involves factoring A into the product of a lower triangular and upper triangular matrix (L and U) such that

$$[LU]\Delta Q = R^N$$

and then the system $L\Delta Q^* = R^N$ is solved by forward substitution and $U\Delta Q = \Delta Q^*$ by backward substitution. This procedure reduces to Newton's method as $\Delta t \to \infty$, and as a result it exhibits quadratic convergence to the solution of the non-linear system of equations under certain restrictions. A simple change to this scheme which has proven to be more efficient involves freezing the LU decomposition in time and performing only forward and backward substitutions to advance the solution vector Q to the steady state. This approach can be represented by

$$\Delta Q^{N+K} = -(U^{-1}L^{-1})^N R(Q^{N+K}, Q^{N+K+1})$$

where N represents the time at which the LU is frozen, and $N + K$ is the current time step.

It has been found that for many problems, a scheme involving the combination of Runge-Kutta initially and LU decomposition with frozen decompostion elements as the steady-state is approached is practical. During the initial transients, LU decomposition is not efficient because the originial estimate of the solution is typically far from the steady state solution and the work associated with the inversions at the onset cannot be justified.

Mesh Generation, Adaptation, and Graphical Interface

The procedure used for generating triangular elements about an arbitrary configuration is an advancing front method discussed by Löhner [4]. This technique requires input of stretching parameters α, δ, and s. The direction α denotes the direction of stretching, with δ being the element size at right angles to α and $s\delta$ being the element size in the direction of stretching. These parameters are given in the context of a coarse background grid of triangular elements which cover the solution domain. They are first used to place nodes along the boundaries of the computational region. The sides connecting these nodes form the initial generation front. Elements are added by interpolating the stretching parameters from the background grid, the entire front is updated, and the process repeated. Once the entire domain has been triangulated, a smoothing routine is performed to improve the quality of the generated mesh.

The grid generation process described requires input of a background mesh and stretching parameters. In general, a very simple background mesh with no streching is input for computing the initial grid. The initial grid will then become the background grid for the first mesh regeneration. This process requires utilizing a measure of the error in the computed solution on the initial grid to obtain the parameters α, δ, and s. An improved

mesh can then be generated with the initial grid as a background grid along with these computed parameters.

If the density ρ is chosen as the variable used to determine the error, then an estimate for the root mean square value of the local error can be given by

$$E_i^{RMS} = \left(\frac{\int_0^{h_i} E_i^2 dx}{h_i} \right)^{\frac{1}{2}} \propto h_i^2 \| \frac{d^2 \hat{\rho}}{dx^2} \|_i$$

where h_i is the local element size, and $\hat{\rho}$ denotes the computed solution. A criterion for a uniform value of the error over the entire domain would then be

$$h_i^2 \| \frac{d^2 \hat{\rho}}{dx^2} \|_i = constant$$

which suggests that the value δ on the new mesh should be computed so that

$$\delta_i^2 \| \frac{d^2 \hat{\rho}}{dx^2} \|_i = constant$$

For multi-dimensional problems, this approach is employed in each direction separately.

This mesh regeneration procedure is generally performed several times in order to obtain a well refined mesh in as little of CPU time as possible. At each remeshing stage the solution on the old grid is interpolated onto the newly generated grid. This is accomplished by simply using the values of the nearest point on the old mesh as the values at each point on the new mesh. This is not conservative, nor is it extremely accurate, but it is very fast, and the error introduced is dominated by high frequencies and hence will be damped out quickly. It should be emphasized that though the intermediate interpolation is not conservative, the solution on the final mesh **will be conservative**. This process is repeated a fixed number of times before turning the remeshing off and converging the solution to the steady-state on the final mesh.

It is very difficult to predict when and how many remeshing stages will be required. Therefore, a graphics interface has been developed to allow the user to have a certain degree of control in guiding the flow solver. For the purpose of portability, all of the device dependent graphics subroutines have been separated from the main flow solver. Currently, three separate graphic libraries have been developed: a Sun GKS, an Iris4D, and a DI-3000 library. The DI-3000 graphics package is device independent and will work on any computer with the DI-3000 software installed such as the NASA Cray-2 supercomputers Voyager and Navier.

When the flow solver is implemented, the user has the ability to execute in either automatic or interactive mode. If the automatic mode is chosen, the solver will execute without user interaction. If interactive mode is chosen, the solver will pause for input just before each remeshing step is to take place. The user then has the ability to view the current solution and mesh, and then decide whether to remesh or continue computing on

the current mesh. In this way, the user is able to guide the solver avoiding unneccessary or premature remeshing.

Computational Results

Transonic Channel Flow

Convergence rates for an inviscid flow over a transonic ($M_\infty = 0.85$) circular arc in a channel on a 1005 element mesh for first order accurate cell-centered and cell-vertex schemes are compared. Figure compares the norm of the residual versus Cray-YMP CPU time for various time integration strategies using both a cell-centered and a cell-vertex finite volume discretization. For the cell-centered calculations, the plot compares the Runge-Kutta, the LU decomposition, the tridiagonal block Gauss-Seidel type relaxation, and the hybrid Runge-Kutta/LU methods. Results are also given in this plot for both the LU decomposition and the tridiagonal algorithms utilizing a frozen Jacobian matrix. The results clearly demonstrate the utility of using the combined Runge-Kutta/LU strategy over the other schemes.

For the cell-vertex calculations, the plot compares the Runge-Kutta, the Runge-Kutta with residual smoothing, the point Jacobi, and the point Gauss-Seidel schemes. The point Jacobi and point Gauss-Seidel methods also have the capability to reuse the Jacobian matrices and to alternate the sweep direction. The fastest method in this case is the symmetric point Gauss-Seidel method with Jacobian reuses. This method compares similarly to the block Gauss-Seidel scheme from the previous plot. A more thorough discussion of these and other convergence acceleration methods can be found in [7].

Supersonic Inlet

The purpose of this test case is to demonstrate the utility of remeshing for a problem with complex wave interactions. Figure 8 shows the first order adaptive remeshing sequence beginning on the initial 85 element uniform mesh (figure 8(a)) and finishing on the 4211 element mesh (figure 8(i)). The first few grids in the sequence develop the initial shock, expansion and reflection and the latter remeshes better define these phenomenon. Figure 9 shows the higher order remeshing sequence which starts with the final first order solution and ends on a mesh with 6247 elements. The higher order remeshes make a significant contribution in refining all of the shocks, especially those toward the exit.

The remeshing strategy took approximately the same CPU time (700 sec) as the solution computed on the 6263 element uniform mesh shown in figure 10. Since the adapted mesh has more elements near the shocks, and since these elements are stretched in the direction of the shocks, the pressure contours for this case display much better resolution than the solution on the uniform mesh.

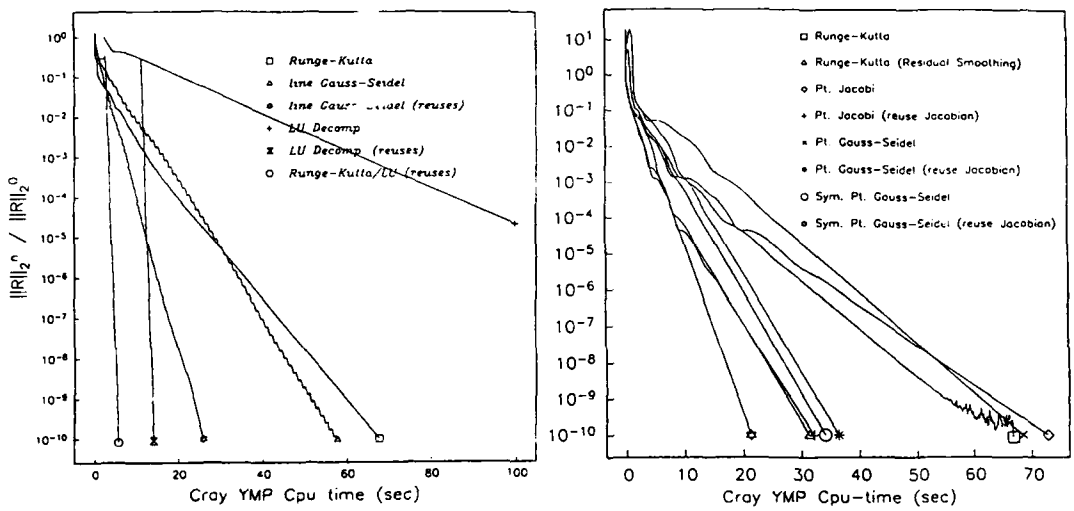


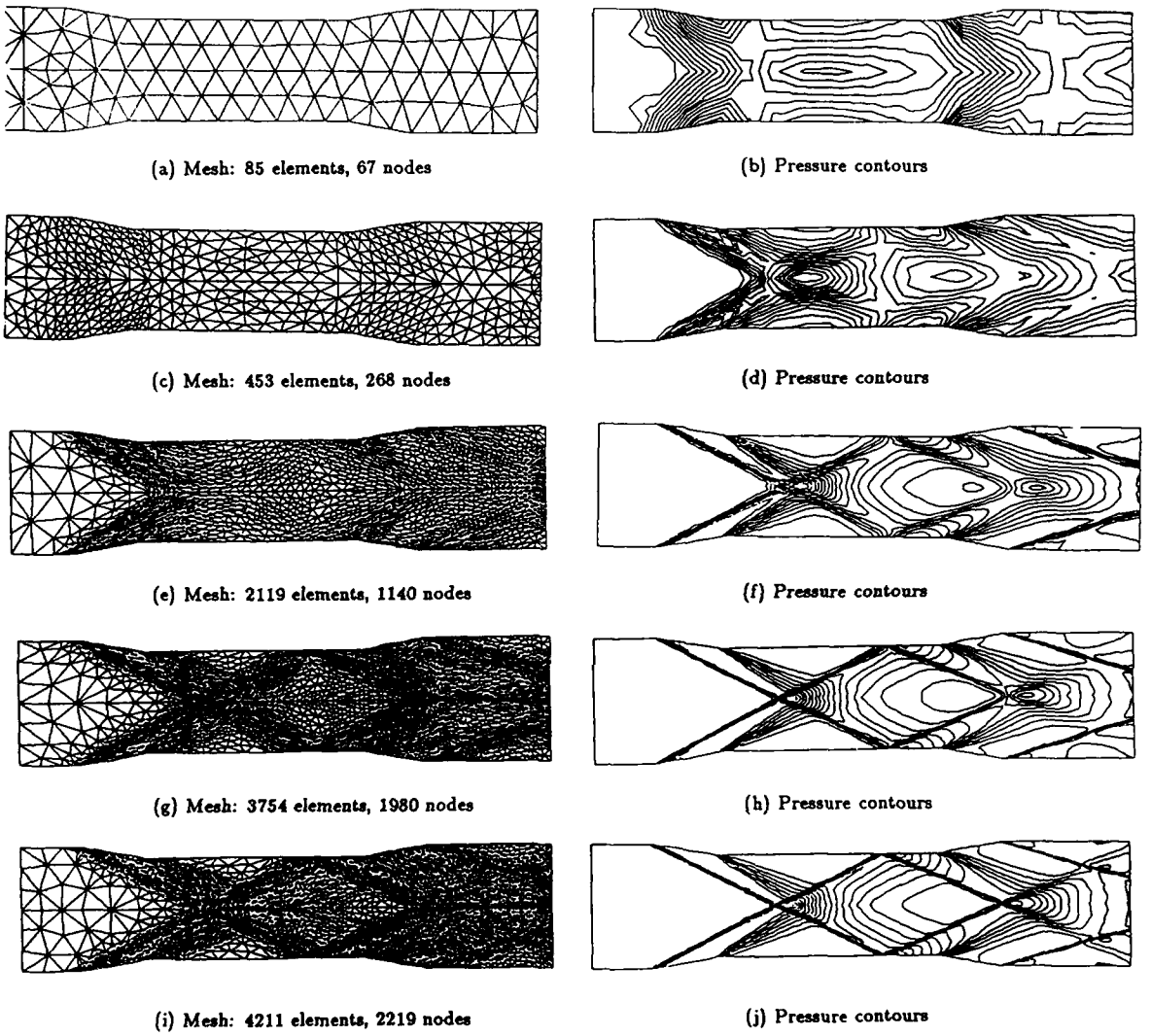
Figure 7. Residual versus CPU time for a transonic channel flow.

Model SR71 Aircraft

As an example of the use of a space-marching method on a structured/unstructured mesh and to demonstrate the geometric flexibility of this algorithm, the solution about a simplified model of the SR71 reconaissance aircraft has been calculated. This model includes a region of multiple elements in the streamwise direction at the start of the engine inlets, as well as multiple vertical tails. Both of these geometries cause difficulties with a structured discretization, while they impose no restrictions with an unstructured discretization. The solution was calculated in a Mach 3.5 free stream at a 0° angle of incidence relative to the root chord. This solution was obtained on a grid with a total of 42 cross flow planes. Figure 11 shows the structured nature of the grid on the surface of the body and contours of pressure in the exit plane. Also shown are shaded pressure contours along the body surface and exit plane along with the unstructured triangular grid in this plane. It is noteworthy that this calculation has been performed on a wide range of computers including an IRIS-4D graphics workstation since both the memory and CPU time required by this approach are minimal. Other examples and comparison with both theory and experiment with this space-marching method are presented in [5].

Concluding Remarks

The development of algorithms on unstructured and hybrid meshes for compressible flow simulations has become a popular research subject in addition to the development of the actual grid generators. This attention is due to the significant improvements that are possible by these approaches. Rapid and robust three-dimensional grid generation



(a) Mesh: 85 elements, 67 nodes

(b) Pressure contours

(c) Mesh: 453 elements, 268 nodes

(d) Pressure contours

(e) Mesh: 2119 elements, 1140 nodes

(f) Pressure contours

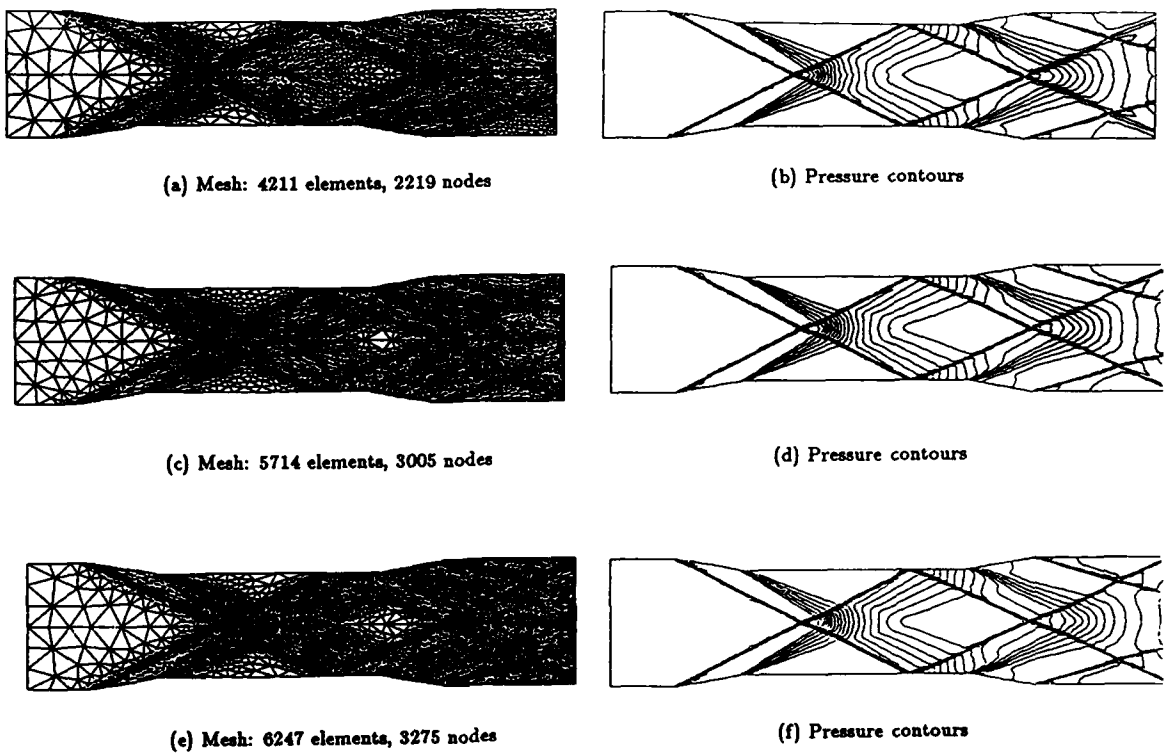
(g) Mesh: 3754 elements, 1980 nodes

(h) Pressure contours

(i) Mesh: 4211 elements, 2219 nodes

(j) Pressure contours

Figure 8. Grids and pressure contours from a 1^{st} order accurate remeshing sequence.



(a) Mesh: 4211 elements, 2219 nodes

(b) Pressure contours

(c) Mesh: 5714 elements, 3005 nodes

(d) Pressure contours

(e) Mesh: 6247 elements, 3275 nodes

(f) Pressure contours

Figure 9. Grids and pressure contours from a 2^{nd} order accurate remeshing sequence.

appears to be a primary topic requiring further attention although a few research groups have demonstrated impressive results. In general, however, the CFD community does not appear to have this technology well in hand although significant resources are now being put into this effort. The flow solvers for unstructured meshes are progressing rapidly and do not appear to be hendering the transfer of technology to the user community.

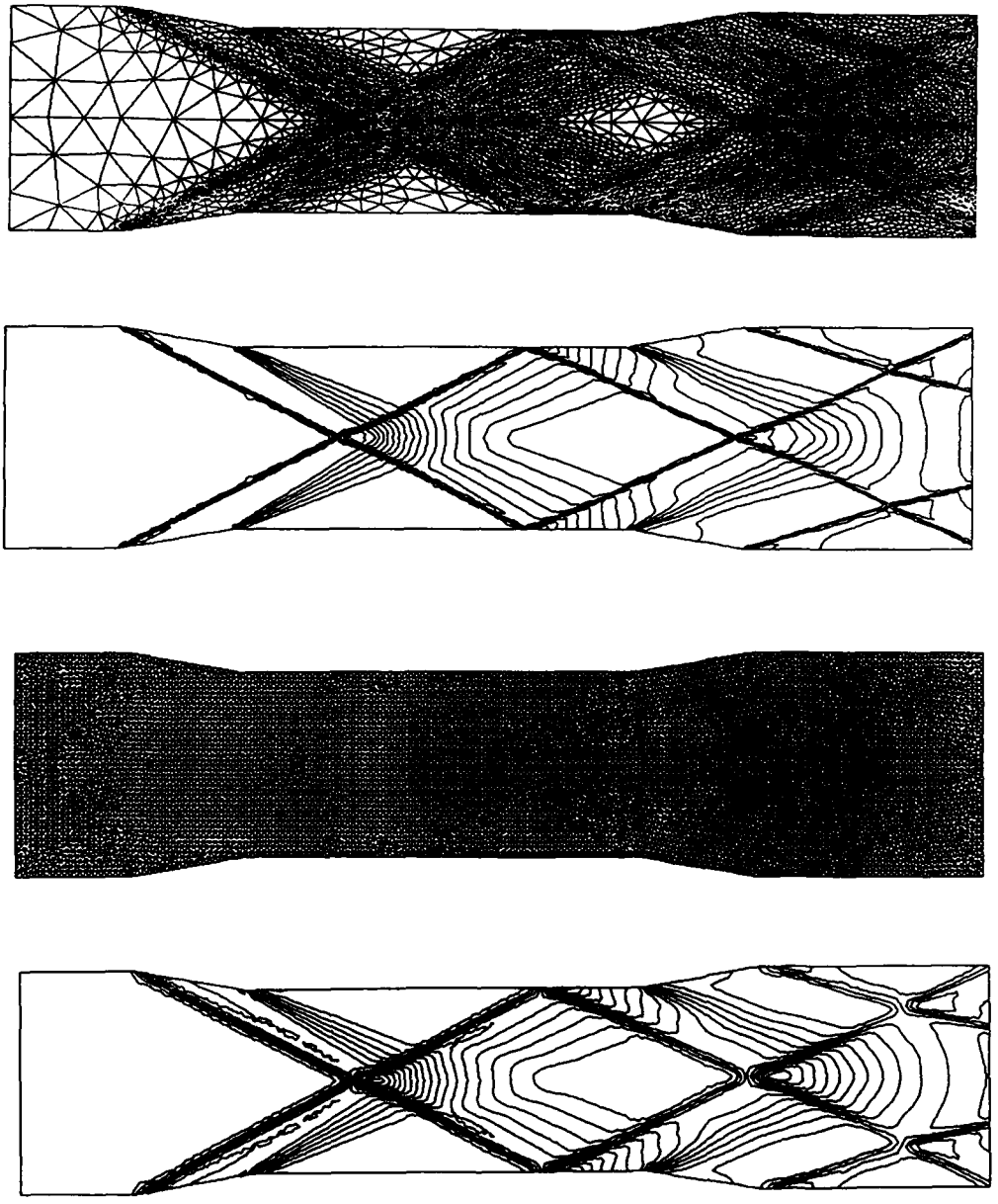


Figure 10. Comparison of uniform and adapted meshes and pressure contours.

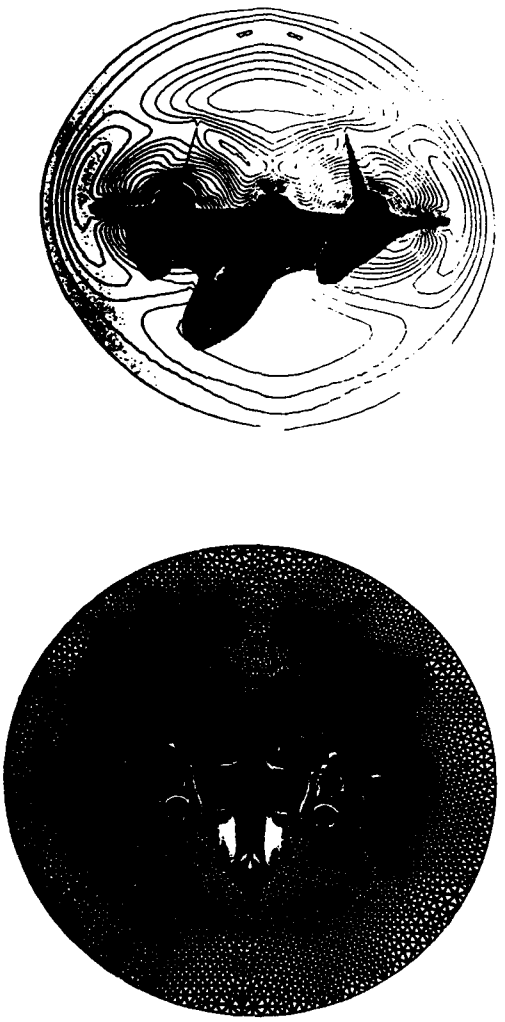


Figure 11. SR71 surface grid, exit plane grid and pressure contours.

References

1. Mavriplis, D. and Jameson, A., "Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes", ICASE Report 87-53, July 1987.
2. Stoufflet, B., Périaux, J., Fezoui, F., Dervieux, A., "Numerical Simulation of 3-D Hypersonic Euler Flows Around Space Vehicles using Adapted Finite-Elements", AIAA Paper 87-0560, 1987.
3. Barth, T.J. and Jespersen D.C., "The Design and Application of Upwind Schemes on Unstructured Meshes", AIAA Paper 89-0336, January 1989.
4. Löhner, R., K. Morgan, J. Peraire and M. Vahdati, "Finite Element Flux-Corrected Transport (FEM-FCT) for the Euler and Navier-Stokes Equations", *Int. J. Num. Meth. Fluids*, no. 7, pp. 1093-1109, 1987.
5. McGrory, W.D., Walters, R.W., and Löhner, R., "A Three-Dimensional Space Marching Algorithm for the Solution of the Euler Equations on Unstructured Grids," AIAA Paper 90-0014, January, 1990.
6. Slack, D.C., Walters, R.W., and Löhner, R., "An Interactive Adaptive Remeshing Algorithm for the Two-Dimensional Euler Equations," AIAA Paper 90-0331, January, 1990.
7. Whitaker, D.L., Slack, D.C., and Walters, R.W., "Solution Algorithms for the Two-Dimensional Euler Equations on Unstructured Meshes", AIAA Paper 90-0697, January 1989.
8. Walters, R.W., Cinnella, P., and Slack, D.C., "Characteristic Based Algorithms for Flows in Thermo-Chemical Nonequilibrium," AIAA Paper 90-0393, January, 1990.
9. Cinnella, P. "Flux-Split Algorithms for Flows with Nonequilibrium Chemistry and Thermodynamics," Ph.D. Dissertation, VPI&SU, December, 1989.
10. Candler, G. "The Computation of Weakly Ionized Hypersonic Flows in Thermo-Chemical Nonequilibrium", Ph.D.Dissertation, Stanford University, 1988.
11. Steger, J. L. and Warming, R. F., *J. Comput. Phys.*, **40**, 263 (1981).
12. Van Leer, B., in *Lecture Notes in Physics,* **170**, (Springer-Verlag, Berlin, 1982), p. 507.
13. Roe, P.L., "Characteristic Based Schemes for the Euler Equations", *Annual Review of Fluid Mechanics*, Vol. 18, 1986, pp.337-365.

IMPLICIT CENTERED METHODS FOR INVISCID AND VISCOUS HYPERSONIC FLOWS

K. KHALFALLAH, G. LACOMBE and A. LERAT

Laboratoire de Simulation Numérique en Mécanique des Fluides

E.N.S.A.M. Paris, France.

SUMMARY We plan to show that the use of space-centered approximations can provide an accurate and efficient way to compute compressible flows with shocks, even at large Mach and Reynolds numbers.

First, we shall present the basic centered method we use for solving the Euler equations. It is a 2 time-level implicit finite-volume method which is conservative, second-order accurate and always linearly stable in any number of space dimensions. When applied to transonic aerodynamics, it gives non-oscillatory solutions with sharp shock profiles (over one or two mesh cells) - see |1|.

Then, we shall describe two modifications of this Euler solver that we have recently investigated for the calculation of hypersonic flows. The first one is the addition of a local entropy correction |2| which preserves second-order accuracy and unconditional stability. This correction enforces a discrete entropy inequality at steady-state (proved for a multidimensional hyperbolic system of conservation laws with a convex entropy, in a general structured mesh). The second modification of the basic Euler solver concerns the introduction of a third time-level to improve the robustness of the method without altering the approximation at steady-state.

Finally, we shall consider the extension to the Navier-Stokes equations with a particular attention to stability and convergence rate questions.

Numerical applications to hypersonic problems will be presented for inviscid and high Reynolds laminar flows.

|1| A. LERAT and J. SIDES - "Efficient solution of the steady Euler equations with a centered implicit method", in Numerical Methods for Fluid Dynamics III, K.W. Morton and M.J. Baines Eds, Clarendon Press-Oxford (1988), p. 65-86.

|2| K. KHALFALLAH and A. LERAT - "Correction d'entropie pour des schémas numériques approchant un système hyperbolique", C.R. Acad. Sc. Paris, 308 II (1989), p. 815-820.

NUMERICAL SIMULATION OF TRANSITION TO TURBULENCE IN FREE-SHEAR LAYERS

M. **LESIEUR**, P. COMTE, Y. FOUILLET,
M.A. GONZE and X. NORMAND

Institut de Mécanique de Grenoble

BP 53 X - 38041 Grenoble-Cedex, France

Ninth International Conference
on
COMPUTING METHODS
IN APPLIED SCIENCES
AND ENGINEERING

29 January - 2 February 1990, PARIS (France)

Abstract

We present unsteady numerical resolutions of 2D or 3D Navier-Stokes equations, in order to study the transition to turbulence in various free-shear layers. The following cases are envisaged:
a) A two-dimensional mixing-layer with periodic boundary conditions in the flow direction (*temporal* mixing layer). The numerical code uses finite differences methods.
b) Two-dimensional spatially-developing mixing layers, wakes and jets.
c) A two-dimensional compressible mixing layer.
d) A three-dimensional incompressible temporal mixing layer and wake (direct and large-eddy simulations, pseudo-spectral methods).

In all the cases, turbulence develops from a random perturbation of small amplitude and broad-band spectrum superposed upon a basic flow.

In the two-dimensional case, it is shown that the coherent structures develop from the Kelvin-Helmholtz instability. They undergo successive pairings, are shown to be unpredictable, and possess in the case of the mixing layer a broad-band spatial spectrum of slope comprised between k^{-3} and k^{-4}.

The compressible calculations show an inhibition of the instability above a convective Mach number of 0.6, in good agreement with earlier experiments and calculations.

In the incompressible three-dimensional case, direct-numerical simulations at low Reynolds numbers allow to show how *hairpin vortices* are strained longitudinally between the big rollers in both cases of the mixing layer and the wake. High Reynolds numbers can be reached with the aid of a spectral subgrid-scale eddy-viscosity. It is shown in this case that the above coherent structures survive, and that the kinetic energy cascades towards the subgrid scales along a Kolmogorov $k^{-5/3}$ spectrum.

1 Introduction

In the study of turbulent free-shear flows, there has been during the last 15 years a growing interest brought to *coherent structures*, that is, structures having a recognisable shape for times much longer than their turn-overtime[1]. These coherent structures exist in particular in mixing layers (Brown and Roshko, 1974), where they appear as spiralling vortices[2], jets and wakes (Perry et al., 1982), They are extremely common in aeronautics, for instance after the detachment of a boundary layer, or in separated flows. They play an important role in combustion, where they determine the flame fronts, and in accoustics, they are largely responsible for the generation of

[1] In my definition of the coherent structures, I require also that they should be unpredictable.
[2] hereafter called Kelvin-Helmholtz vortices

noise. They are also found in the atmosphere (cyclonic or anticyclonic perturbations), in the ocean (mesoscale eddies), and in planetary atmospheres (Jupiter's Great Red Spot or Neptune's Dark Blue spot). An important characteristic of these coherent structures is their highly mixing and unpredictable characters: for these reasons they are an essential component of the turbulence.

More recently, another type of coherent structure has been discovered in free-shear flows, namely three-dimensional hairpin-shaped[3] longitudinal vortices (Breidenthal, 1981, Bernal and Roshko, 1986). These structures seem to play an important role during the transition process to small-scale turbulence.

The coherent structures have been first discovered in the experiments. However, the increasingly fast development of computational fluid dynamics allows now spectacular progresses in the understanding of the role and dynamics of coherent structures in turbulent shear flows.

2 The two-dimensional free-shear layers

2.1 Numerical methods

We solve numerically in a rectangular domain the two-dimensional Navier-Stokes equation

$$\frac{D}{Dt}\omega = \nu\nabla^2\omega \tag{1}$$

where

$$\omega = -\nabla^2\psi(x,y,t) \tag{2}$$

is the vorticity, and ψ the stream function of the flow. The D/Dt operator is the Lagrangian derivative following the fluid motion, given by

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + J(.,\psi) \quad , \tag{3}$$

where $J\,(A,B) = (\partial A/\partial x)(\partial B/\partial y) - \partial B/\partial x)(\partial A/\partial y)$ is the Jacobian operator. The initial basic flow (in a periodic calculation) or inflow (in a spatially-growing calculation) may be a hyperbolic-tangent velocity profile (mixing layer), a $(1/\cosh^2 y)$ profile (plane jet) or a gaussian deficit velocity profile (wake). To these profile, we superpose initially (in the periodic calculations) or at each time step (in the spatially-growing calculations) a random perturbation of weak amplitude (white noise), modulated in the y direction by a gaussian function of width δ_i. In fact, this study

[3] Hairpin structures are also found in the boundary layer both during the transition to turbulence (Klebanoff et al., 1962) and in developed turbulence (Kline et al., 1967). They will not be considered here.

concerns both the *natural* transition to turbulence[4], and also the generation of coherent structures in a developed shear-layer, due to the instability of the mean-velocity profile. Boundary conditions at the lateral boundaries are of the free-slip type. Numerical methods consist of finite-differences on a regular grid, and are described in Comte et al. (1988) and Comte (1989). The Jacobian in eq. (3) is calculated using Arakawa's scheme, which conserves kinetic energy and enstrophy in the domain. In the spatially-developing calculations, the outflow boundary condition is of the Sommerfeld type.

The diffusion of a passive scalar θ, called here *temperature*, is simultaneously studied. The latter satisfies an equation analogous to (1)

$$\frac{D}{Dt}\theta = \kappa \nabla^2 \theta \tag{4}$$

where κ is the molecular diffusivity. The Prandtl number ν/κ will be taken equal to 1. The initial temperature profile is identical to the basic velocity profile, and allows to visualize the flow structures in the same manner as a numerical dye would do in the experiments.

Calculations are done here at Reynolds numbers (based on the initial vorticity thickness) of 1000 or 500. Notice that, in a previous study, we did also calculations where the dissipative term $\nu\nabla^2\omega$ in (1) was replaced by $-\nu_1\nabla^4\omega$ (see Lesieur et al., 1988). This modification is made frequently in two-dimensional turbulence studies related to oceanography and meteorology, and shifts the dissipative effects towards the smallest scales close to the calculation mesh Δx. However, the results are not substantially different at the Reynolds numbers considered here, as far as the coherent-structure dynamics is concerned.

2.2 Coherent-structures dynamics

Before looking at the results of the calculation, it is of interest to recall the main results of the linear-instability theory applied to the periodic free-shear flows. In a fluid of uniform density ρ_0, we consider the stability of a parallel flow of components $\bar{u}(y), 0, 0$, upon which is superposed a small perturbation. This perturbation is assumed to be two-dimensional in the (x, y) plane, with a stream function $\tilde{\psi}(x, y, t)$ of the form

$$\tilde{\psi} = \epsilon\, \Phi(y)\, \exp i\, \alpha(x - ct) \quad , \tag{5}$$

corresponding to a perturbed velocity field of components $\tilde{u} = \partial\tilde{\psi}/\partial y, \tilde{v} = -\partial\tilde{\psi}/\partial x, 0$, with:

$$\tilde{u} = \epsilon\, \frac{d\Phi}{dy}\, \exp i\, \alpha(x - ct), \tilde{v} = -\epsilon\, i\alpha\Phi(y)\, \exp i\, \alpha(x - ct) \quad . \tag{6}$$

$\epsilon << 1$ is a small non-dimensional parameter. α is real, and is the spatial longitudinal wave number of the perturbation: this is a *temporal analysis*, by opposition to a *spatial analysis* where α

[4] where the white noise models the residual incoming turbulence in an experimental apparatus, which injects energy in all the unstable modes

is complex. In this temporal study, c is complex, of real and imaginary parts c_r and c_i. αc_r is the phase-speed of the perturbation, while αc_i is its temporal-growth rate. Notice that, within the linear-instability analysis, it is not necessary to consider the growth of three-dimensional perturbations, which are always less amplified than two-dimensional perturbations (Squire's theorem, see Drazin and Reid, 1981). This is, however, no longer true in compressible flows: for instance, the temporal-compressible mixing layer admits, at a Mach number $M_c = U/c > 0.6$ (U is half the velocity difference), oblique waves which are more unstable than the two-dimensional waves, as shown by Sandham and Reynolds (1989).

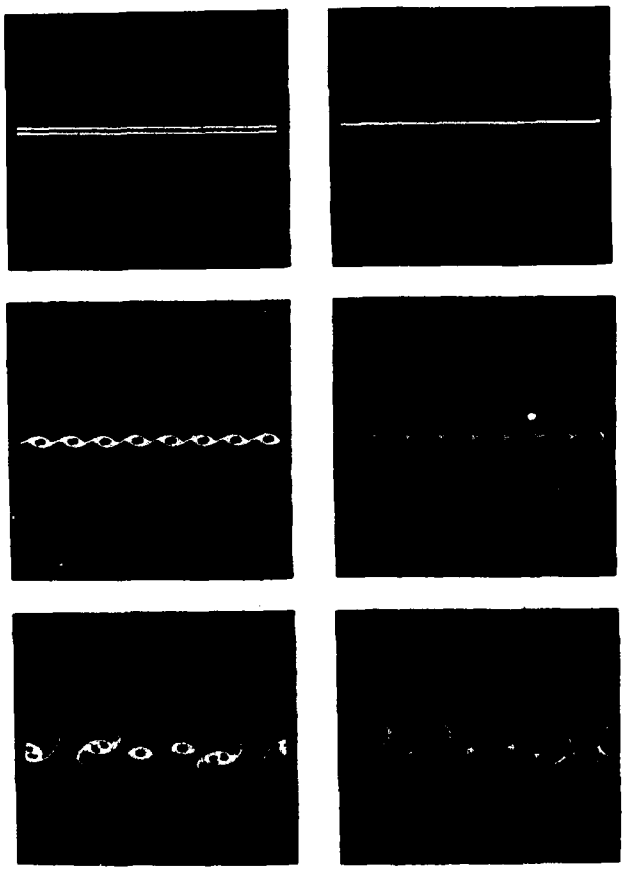


Figure1: two-dimensional temporal incompressible mixing-layer; simultaneous evolution of the vorticity (left) and temperature (right) fields (from Comte, 1989).

We assume that the total velocity field (basic flow + pertubation) is two-dimensional (no

z-dependance). Its vorticity is

$$\omega = -\frac{d\bar{u}}{dy} + \tilde{\omega} \; ; \quad \tilde{\omega} = -\nabla^2 \tilde{\psi} \tag{7}$$

The vorticity equation (1) is thus expanded to the first order with respect to ϵ. We assume that the basic flow is a solution of the Navier-Stokes equation. Therefore, it is easily obtained:

$$\frac{\partial \tilde{\omega}}{\partial t} + \bar{u}(y)\frac{\partial \tilde{\omega}}{\partial x} + \tilde{v}\frac{d\bar{\omega}}{dy} = \nu \nabla^2 \tilde{\omega} \quad , \tag{8}$$

which can be written as (assuming that $\alpha \neq 0$):

$$[\bar{u}(y) - c]\,(\frac{d^2\Phi}{dy^2} - \alpha^2\Phi) - \frac{d^2\bar{u}}{dy^2}\,\Phi = -\frac{i\nu}{\alpha}\,(\frac{d^2}{dy^2} - \alpha^2)^2\,\Phi \quad , \tag{9}$$

which is the traditional form of the Orr-Sommerfeld equation. This equation can be solved numerically, for various basic velocity profiles: for the hyperbolic-tangent mixing layer, corresponds to a given Reynolds number a range of unstable wave numbers α. The amplification rate αc_i is maximum for a certain wave number α_a, called the most-amplified mode: it is this mode which will emerge the first when the initial perturbation is a white noise, or a random perturbation of broad spatial spectrum. This is a very efficient way of selecting the associated spatial wave length $\lambda_a = 2\pi/\alpha_a$. When the Reynolds number exceeds a value ~ 30 (for the mixing layer) and ~ 100 (for the jet or the wake), λ_a is Reynolds number independant, and scales on δ_i , the initial vorticity thickness. Therefore, the vortical layer of width δ_i will first oscillate, then roll up, by vorticity induction, and form a street of either Kelvin-Helmholtz vortices (in the mixing-layer case) or a Karman street (in the wake or jet case).

Figure 1 shows the simultaneous evolution of the vorticity and temperature fields, in a mixing-layer calculation involving initially 8 fundamental eddies: the eddies, once formed, undergo successive pairings, in which they turn around each other and amalgamate. One verifies also that the temperature wraps around the vorticity concentrations. These images present striking resemblances with experimental visualizations of the mixing between two chemically-reacting flows, done by Koochesfahani and Dimotakis (1986). Another important remark concerns the unpredictable character of the pairing interaction, and the fact (when looking at the subsequent evolution of the layer) that eddies of different size will preferentially pair: this may be responsible for structures involving three eddies, when two eddies finishing to merge will pair with a third one.

Figure 2 shows an incompressible spatially growing mixing layer once a statistically stationary regime is established. The visual spreading rate is found to be in good agreement with the experiments on natural[5] mixing layers. Figure 3 shows the vorticity field in a spatially growing jet.

[5] that is, unforced

Figure2: numerical simulation of an incompressible spatially growing mixing layer: passive scalar contours.

We have studied in Staquet et al. (1985) and Lesieur et al. (1988) the longitudinal[6] spectra of kinetic energy and passive temperature in a two-dimensional periodic mixing layer: it was shown that the kinetic energy spectrum, initially formed of a peak at the fundamental mode α_a, and of its harmonics, develops an inertial range at the end of the first pairing. The slope of this range is comprised between k^{-3} and k^{-4}.

Figure3: numerical simulation of an incompressible spatially growing plane jet: vorticity field (courtesy P. Alexandre, I.M.G., 1989).

3 Two-dimensional compressible periodic mixing layer

We have developed a finite-differences numerical code allowing to solve the full compressible Navier-Stokes equations. The spatial scheme is a centered second-order scheme, and the temporal

[6] in the x direction

one is a third-order Runge-Kutta scheme with reduced storage. Here (see also Normand et al., 1989), this code is applied to a two-dimensional temporal mixing layer of uniform temperature initially. It was shown by Papamoschou and Roshko (1988) that the relevant parameter to describe the spreading of the layer is the convective Mach number, equal to $M_c = (U_1 - U_2)/2c$ in this simplified isothermal case. Calculations show that there is an inhibition of the Kelvin-Helmholtz instability above $M_c = 0.6$: eddies hardly roll up and they merge without spiralling about each other, as it is the case in the incompressible case. This seems to be related to the sharp transition in the mixing-layer structure found experimentally by Papamoschou and Roshko (1989) at this Mach number. Since, as already stressed, the three-dimensional instabilities grow faster above $M_c = 0.6$, it is necessary to develop three-dimensional computations in order to understand the structure of a turbulent compressible mixing layer at high Mach numbers.

4 Three-dimensional direct and large-eddy simulations

We have used pseudo-spectral numerical methods in order to simulate both the incompressible three-dimensional temporal mixing layer (with free-slip boundary conditions on the upper and lower faces of the computational box), and a gaussian wake[7] in a periodic box. The Navier-Stokes equation in Fourier space may be written as:

$$\frac{\partial}{dt}\hat{\underline{u}}(\vec{k},t) = \Pi(\vec{k}) \circ F[F^{-1}(\hat{\underline{u}}) \times F^{-1}(\hat{\underline{\omega}})] - [\nu + \nu_t(k|k_c)]\ k^2 \hat{\underline{u}}(\vec{k},t) \quad , \tag{10}$$

where $\hat{\underline{u}}(\vec{k},t)$ is the velocity field, and $\Pi(\vec{k})$ the projector on the plane perpendicular to $\vec{k}$. F stands for a Fast Fourier transform operator. The incompressibility writes:

$$\vec{k}.\hat{\underline{u}}(\vec{k},t) = 0 \quad . \tag{11}$$

The term $\nu_t(k|k_c)$ is a spectral eddy-viscosity, which will be used for large-eddy simulations (see below). It is zero for direct-numerical simulations.

A passive scalar $\hat{\theta}(\vec{k},t)$ satisfies the equation

$$\frac{\partial}{dt}\hat{\theta}(\vec{k},t) = -i\vec{k}.F[F^{-1}(\hat{\theta})F^{-1}(\hat{\underline{u}})] - [\kappa + \kappa_t(k|k_c)]\ k^2 \hat{\theta}(\vec{k},t) \quad , \tag{12}$$

where a spectral eddy-conductivity $\kappa_t(k|k_c)$ will be used also for large-eddy simulations.

The résolutions are of 48^3 collocation points for the calculations done on the I.M.G. Alliant VFX40 machine, and 128^3 points on the C.C.V.R. Cray 2 machine. The graphics are done on the Alliant, using the FLOSIAN[8] software we have developed in Grenoble.

[7] that is, developing from a perturbed gaussian velocity profile

[8] FLow SImulation ANalysis

4.1 Direct-Numerical simulations

4.1.1 *The mixing layer*

The calculation domain in physical space is a rectangular parallelepiped of sides L_x, L_y et L_z. One assumes periodicity in the x and z directions. The resolution is 128^3 for a calculation involving 4 fundamental eddies ($L_x = 4\lambda_a$). One superposes to the basic flow a three-dimensional isotropic wide-band perturbation of kinetic energy $< u'^2 >= 10^{-4}U^2$, whose spectrum peaks at α_a. Visualizations of the scalar[9] show that a hairpin-shaped vortex of spanwise wave length $\lambda = 2\lambda_a$ is formed in the braids, with the same topology as that observed experimentally by Bernal and Roshko (1986). The same structures have been found in the direct-numerical simulations involving two eddies done by Metcalfe et al. (1987). The origin of these hairpin coherent structures is still under discussion. A possible explanation, proposed by Lasheras and Choi (1988), comes from the straining between two Kelvin-Helmholtz rollers of vortex filaments perturbed about the stagnation line. We expect to show more about the vorticity fields corresponding to these structures during the conference.

4.1.2 *The plane wake*

The calculation is done at a resolution of 48^3 points. The initial flow is a gaussian profile, perturbed by the same perturbation as above. Figure 4 shows the passive scalar during the formation of the Karman street. Later on longitudinal vortices appear, in the outer edge of the Karman eddies. The vorticity contours will be shown at the conference.

Figure4: passive scalar contour in the three-dimensional wake calculation.

[9] This work is in progress. Up to now, only colour slides of the scalar field during the pairing of primary vortices are available.

4.2 Large-Eddy simulations

4.2.1 *Spectral eddy-coefficients*

We use the concept of spectral eddy-viscosity and eddy-conductivity in order to model the subgridscales corresponding to $k > k_c$, k_c being the cutoff wave number (see Lesieur, 1987). Using the ***non-local interactions theory*** of isotropic turbulence two-point closures, it may be shown that the kinetic energy flux between k and the subgridscales can be represented with the aid of the eddy-viscosity (Kraichnan, 1976, Lesieur and Schertzer, 1978):

$$\nu_t(k|k_c) = \frac{1}{15}\int_{k_c}^{\infty} \theta_{kpq}[5E(p) + p\frac{\partial E}{\partial p}]dp \quad , \tag{13}$$

for $k << k_c$. $E(k$ is the kinetic energy spectrum, and θ_{kpq} a time characteristic of the triple-correlations relaxation. In the same way, the eddy-conductivity may be written as

$$\kappa_t(k|k_c) = \frac{2}{3}\int_{k_c}^{\infty} \theta^T_{kpq} E(p)dp \quad . \tag{14}$$

If k_c lies in a $k^{-5/3}$ Kolmogorov spectrum, it is found:

$$\nu_t(k|k_c) = 0.267[\frac{E(k_c)}{k_c}]^{1/2} \quad , \tag{15}$$

$$\kappa_t(k|k_c) = 0.445[\frac{E(k_c)}{k_c}]^{1/2} \quad . \tag{16}$$

It is these quantities which are used in eqs. (10) and (12).

When the method is applied to isotropic decaying three-dimensional turbulence, kinetic energy spectra close to $k^{-5/3}$ are obtained in the neighbourhood of k_c, and the kinetic energy decays like $t^{-1.37}$ (see Lesieur et al., 1989), in good agreement with the predictions of the statistical EDQNM theory (see Lesieur, 1987).

The eddy Prandtl number $\nu_t(k|k_c)/\kappa_t(k|k_c)$, where $\kappa_t(k|k_c)$ is the spectral eddy-diffusivity, calculated using the spectral temperature transfers, is, from EDQNM calculations, taken constant and equal to 0.6. In fact, it was recently shown by Lesieur and Rogallo (1989) and Lesieur et al. (1989), on the basis of a direct determination, that it increases with k between the values 0.2 and 0.8. But this variation has no incidence on the following calculations, where the passive temperature is used only as a numerical dye to visualize the coherent structures.

4.2.2 *The mixing layer*

The calculation involves two fundamental eddies, with a resolution of $64 \times 32 \times 32$. Figure 5 shows the interface[10] when the primary instability starts to grow. It indicates that a strong spanwise

[10] characterised by the isothermal surface of zero value initially

instability develops as well. This spanwise instability could be due to the straining between the fundamental eddies of vortex lines perturbed initially in the spanwise direction by the random perturbation. This would lead to hairpin vortices by the mechanism proposed by Lasheras and Choi (1988), already mentioned.

Figure5: large-eddy simulation of the mixing layer; scalar field at the beginning of the roll up.

Figure 6 shows the same surface at the end of the roll up. Figure 7 shows at the same time the primary vorticity ω_z, corresponding to an iso-value $2U/\delta_i$. Notice that this value is the maximum vorticity you can get in two dimensions, due to the vorticity advection by the motion, and its molecular dissipation. Notice that, on Figure 6, the dark thin longitudinal lines indicate the longitudinal vorticity ω_x, for the same iso-value as ω_z. It indicates that longitudinal streaks of intense vorticity link the billows, as in the above direct numerical simulations.

Figure6: large-eddy simulation of the mixing layer; scalar field at the end of the roll up; in dark is shown the intense longitudinal vorticity.

Figure7: large-eddy simulation of the mixing layer; spanwise vorticity (same time as Figure 6).

Finally, Figure 8 shows during the pairing the three-dimensional spatial spectra of the passive temperature, the kinetic energy, and the three components of velocity. It indicates how, starting from initial spectra exponentially decaying at high wave numbers, a cascade has developed towards the small scales (with a slope which is fairly close to $k^{-5/3}$). This is in very good agreement with the experimental measurements of these spectra. Finally, and when the turbulence in the small scales has developed, the variances of the three velocity components are found to be:

$$< u'^2 >= 0.11U^2$$

$$< v'^2 >= 0.07U^2$$

$$< w'^2 >= 0.08U^2$$

again in fairly good agreement with the experiments.

It seems thus that this spectral large-eddy simulation code is a very good tool which allows to describe the whole transitional process towards developed turbulence, both for predicting the right statistics and displaying the correct primary and secondary coherent structures.

Aknowledgements

This work has been supported by D.R.E.T. under contract 88/150, and by CNES/Avions Marcel Dassault. Part of the calculations were done on a grant of the Centre de Calcul Vectoriel pour la Recherche. The Institut de Mécanique de Grenoble is sponsored by the C.N.R.S., I.N.P.G. and U.J.F.

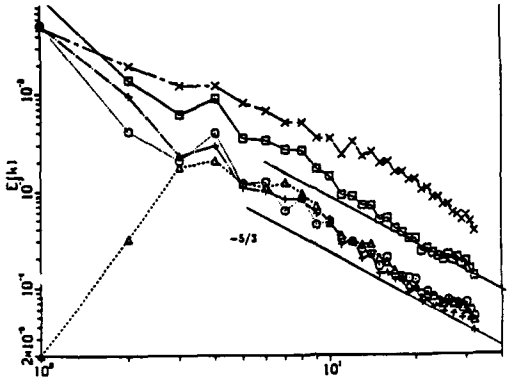


Figure8: large-eddy simulation of the mixing layer; three-dimensional spatial spectra of (from top to bottom) the passive temperature, the kinetic energy, and the three velocity components (longitudinal, transverse and spanwise).

References

Bernal, L.P. et Roshko, A., 1986, "Streamwise vortex structure in plane mixing layers", J. Fluid Mech., **170**, pp 499-525.

Breidenthal, R., 1981, "Structure in turbulent mixing layers and wakes using a chemical reaction", J. Fluid Mech., **109**, pp 1-24.

Brown, G.L. et Roshko, A., 1974, "On density effects and large structure in two-dimensional mixing layers", J. Fluid Mech., **64**, pp 775-816.

Comte, P., Lesieur, M., Laroche, H. et Normand, X., 1988, "Numerical simulations of turbulent plane shear layers", in *Turbulent Shear Flows VI, Toulouse*, Springer-Verlag, pp 360-380.

Comte, P., 1989, Doctoral dissertation, Institut National Polytechnique de Grenoble.

Drazin, P.G. and Reid, W.H., 1981, *Hydrodynamic instability*, Cambridge University Press, pp 233-237.

Klebanoff, P.S., Tidstrom, K.D. and Sargent, L.M., 1962, "The three-dimensional nature of boundary layer instability", J. Fluid Mech., **12** , pp 1-34.

Kline, S.J., Reynolds, W.C., Schraub, F.A. and Runstadler, P.W., 1967, "The structure of turbulent-boundary layers", J. Fluid Mech., **30**, pp 741-773.

Koochesfahani, M.M. and Dimotakis, D.E., 1986, "Mixing and chemical reactions in a turbulent liquid mixing layer", J. Fluid Mech., **170**, pp 83-112.

Kraichnan, R.H., 1976, "Eddy-viscosity in two and three dimensions", J. Atmos. Sci., **33**, pp

1521-1536.

Lasheras, J.C. and Choi, H., 1988, "Three-dimensional instability of a plane free shear layer: an experimental study of the formation and evolution of streamwise vortices", J. Fluid Mech., **189**, pp 53-86.

Lesieur M., 1987, *Turbulence in Fluids*, Kluwer.

Lesieur, M. , Staquet, C. , Le Roy, P. et Comte, P. , 1988, "The mixing layer and its coherence examined from the point of view of two-dimensional turbulence", J. Fluid Mech., **192**, pp 511-534.

Lesieur, M. et Rogallo, R., 1989, "Large-Eddy simulation of passive scalar diffusion in isotropic turbulence", Phys. Fluids A, **1**, pp 718-722.

Lesieur, M., Métais, O. and Rogallo, R., 1989, "Etude de la diffusion turbulente par simulation des grandes échelles", C.R. Acad. Sci., Paris, Ser. *II*, **308**, pp 1395-1400.

Metcalfe, R.W., Orszag, S.A., Brachet, M.E., Menon, S. et Riley, J., 1987, "Secondary instability of a temporally growing mixing layer", J. Fluid Mech., **184**, pp 207-243.

Normand, X., Fouillet, Y. and Lesieur, M., 1988, "How much are the small scales of high Mach number turbulence affected by compressibility?", preprint, I.M.G.

Papamoschou, D. and Roshko, A., 1988, "The compressible turbulent shear layer: an experimental study", J. Fluid Mech., **197**, pp 453-477. 1988

Perry, A., Chong, M.S. et Lim, T.T., 1982, in *Vortex motions*, pp 106-121, Friedr. Vieweg. and Sohn eds.

Sandham, N. and Reynolds, W.C., 1989, "The compressible mixing layer: linear theory and direct simulation", AIAA paper 89-0371.

Staquet, C., Métais, O. and Lesieur, M., 1985, "Etude de la couche de mélange et de sa cohérence du point de vue de la turbulence bidimensionnelle", C.R. Acad. Sci., Paris, Ser II,**300**, pp 833-838.

Far field boundary conditions for problems in fluid dynamics

by
Bertil Gustafsson
Department of Scientific Computing
Uppsala University
Uppsala, Sweden

Abstract

When solving flow problems in unbounded domains it is necessary to introduce artificial boundaries. If the flow is smooth in the far field and there are no significant viscous effects, it is rather well known how to construct boundary conditions such that accurate solutions are obtained. However, sometimes the computational domain cannot be extended far enough. For example, when computing the flow around a solid body, the boundary may intersect the wall, thereby cutting through the boundary layer. In that layer the gradients are very large for large Reynolds-numbers, and the usual linearized equations are no longer valid.

We shall analyze a few model problems in order to get an understanding of the behaviour of the solutions depending on the boundary conditions. In particular, we shall discuss the procedure of using the inviscid conditions as the basic set and then add viscous conditions of derivative type. In general the errors introduced in this way are small provided the given data at the boundaries are accurate. If such data are not available, a common procedure is to extrapolate all variables. We shall prove that this in general introduces large errors. However, in the case with a boundary layer, the situation is more favorable. We shall prove that large gradients along the boundary actually helps to keep the error small.

We shall also present a number of numerical experiments which confirm the theoretical results.

ARCHITECTURES DES SUPERCALCULATEURS

SUPERCOMPUTER ARCHITECTURES

NEC Supercomputer SX-3 Series

Architecture and Software

A. Iwaya
Senior Program Manager of
EDP Product Planning Division
33-1, Shiba 5-chome, Minato-ku,
Tokyo 108, JAPAN

Abstract

This paper presents a brief review of the NEC SX-3 Series Architecture and Software from the view point of NEC philosophy for the SX-3 development. In particular, the importance of single processor power is stressed even though this system support parallel processing.

1. Introduction

In April, 1989 NEC announced SX-3 Series consisting of 7 model configurations, with maximum performance ranging from 1.37 G Flops for the processor Model 11, to 22 G Flops for the 4 processor Model 44. The SX-3 is the first Japanese supercomputer employed a multiprocessor configuration.

2. SX-3 Development Philosophy

The objectives of Supercomputer is to offer users the problem solution. In other words, to offer the capability to solve the large scale scientific problems with minimal cost. It has to offer the shortest turn-around time or response-time for the given problem.

Response time may consist CPU time, I/O time and communication time. These have to be offered in well balanced manner, however, the most important one is considered to be CPU time.

Minimal CPU time is realized by the appropriate choise of the architecture and the device technology. The criteria of consideration for the choise are as follows;

. The selected architacture has to offer minimal CPU time for the wide range of application.

. The architecture is also appropriate one from reliability, easy-to-use and upgradability view point.

. The high-speed device technology in a given periods of time and the balance between the architecture and the device technology have to be considered.

In the early stage of SX-3 development, the followings were observed and then decided considering the above mentioned criteria.

. To develop the fastest single processor is very important, in particular, the scalar performance of single processor is never degraded even if the cost is considerably high.

. The importance of powerful scalar processor is never too stressed. Because even if the vectorization or parallelization ratio of the program is say over 99% and remaining 1% is to be processed by the slower scalar processor, then the total performance is severely degraded.

. Even more, these exists a lot of large application programs whose vectorization or parallelization ratio is rather lower.

- In the present and foreseenable future, SIMD type of application is considered to be dominant over MIMD type application, therefore the vector processing support has priority to the parallel processing support. Note that 'microtasking' is considered to be a kind of vector processing.
- Yet, there exist some applications which require MIMD operation and also the user site where throughput is important, therefore multi-processor support is also necessary as second priority.
- In case of multi-processor system, the shared memory type architecture is appropriate, considering the ease-of-use and the continuation from single processor system point of view.
- For the device technology, Silicon has to be continuously used because devices such as GaAs and JJ is yet in infant stage and silicon technology has enough room to further speed enhancement, and it is stable and economical.

The observation described above strongly affected the choice of architecture and technology for SX-3 development. What have to be stressed here is that NEC strongly favors the system which has the most powerful processors by the faster VLSI technology and never supports the system which have many processors by relatively slower technology as general purpose supercomputers.

3. SX-3 technology

NEC continues to use silicon VLSI technology for achieving the most powerful scalar as well as vector and parallel processing capability.

Using high-speed Silicon bipolar CML (current mode logic) circuitry, the CML-LSI logic elements of the SX-3 have gate delays of 70 picoseconds and contain up to 20,000 gates per chip. The high speed 40 Kbit RAM chips used in the vector registers contain 7,000 gates of logic and have access times of 1.6 nanoseconds.

Main memory chips are 256 Kbit Bi-CMOS static RAMs with access times of 20 ns, and the extended memory is integrated with 1 Mbit MOS dynamic RAMs.

Up to 100 high speed LSIs (2,000,000 gates) are contained on each 22.5 x 22.5 cm ceramic package. Due to the enormous number of input/output terminals to connect, four signal wire layers were employed on the ceramic board. The density of the chip layout is further augmented with the use of polymide insulation which enables faster signal delay times, resulting in very fast signal propagation.

4. SX-3 Architecture

As mentioned above, our basic approach to realize a high-speed computer system is to enhance a single processor performance to the ultimate, and then to combine those ultra high-speed processors constituting a multiprocessor system.

Figure 1 shows a maximum configuration of the SX-3 system, and Table 1 shows the system specifications together with those of the SX-2A, the top end model of the former SX series. In a maximum configuration, four arithmetic processors share a common main memory unit which has a capacity of up to 2 GBytes. The shared memory system and a small number of high-speed processors to realize a multiprocessor system give users the ease of use and easy programming environments, because they don't need to care about the memory allocation algorithm, different from the distributed memory system, and don't need to augment the degree of parallelism to fully utilize the hardware capability.

In Figure 1, the Control Processor (CP) performs I/O management. On the other hand, the Arithmetic Processor (AP), which has internally a scalar and vector unit, is an ultra high-speed Fortran engine and executes all the user codes compiled by Fortran compiler and major supervisory operations.

The Control Processor Memory (CPM) with a capacity of up to 256 MBytes is used as large I/O buffer. The Main Memory Unit (MMU) is a large and fast memory for the execution of user and supervisory programs running on the Arithmetic Processors. To transfer a large amount of vector data quickly and smoothly, the MMU is divided into a maximum of 1,024 independent banks, that is 1,024 way interleaved system is employed.

The Extended Memory Unit (XMU) is a large capacity semiconductor memory unit ranging from 1 GBytes to 16 GBytes, and works as a very high-speed virtual disk unit. The XMU, which has a transfer speed of 2.75 GBytes/sec, is used for temporary/permanent disk files, disk cache buffer and job swapping files.

The SX-3 can configure up to four I/O processor with an aggregate transfer speed of 1 GBytes/sec. Each I/O processor has up to 64 channels through which various peripherals such as disk units, cartridge library, laser printers, optical disk units are connected.

SX-3 AP Scalar Unit

Each AP Scalar Unit consists of a full complement of 64 bit floating and fixed point arithmetic pipelines. Each AP scalar unit has 128 general purpose, 64 bit, registers. The scalar unit issues both scalar and vector instructions. The machine instruction set is based on the RISC (Reduced Instruction Set Computing) concept.

Numeric Representations

"Standard Range" is recommended when greater precision is desired; it is compatible with IBM single, double, and quad precision floating point representations. "Extended Range" is recommended when a large numeric range is required; it is compatible with CRAY single and double precision floating point representations. The format can be selected at compile time by use of a compiler switch.

Instruction Chaining

Full instruction chaining is supported. Chaining is an advanced form of pipelining which allows either related or unrelated vector instructions to begin execution before previous vector instructions complete, even though they may use the same registers or pipelines.

SX-3 Series Main Memory

Main Mamory is interleaved up to 1024 ways. 80 Gigabytes/second of concurrent sequential vector load/store, constant stride vector load/store, vector gather/scatter, scalar cache load, scalar store, I/O, and XMU transfer are supported on the SX-3.

The XMU consists of up to 16 Gigabytes of 1 Megabit DRAM (Dynamic Random Access Memory). Transfer speed to and from Main Memory is 2.75 Gigabytes/second.

Reliability

Traditional supercomputers have been designed for the sole purpose of high speed execution. As a result, certain reliability and error checking features, common to lesser machines, have been sacrificed.

Most recent designs have put greater emphasis on system stability and reliability. As an example, reliability of the SX-3 Series is supported by BID (Built In Diagnostics). BID are implemented in hardware; the scope of BID includes over 10,000 error indications within the system.

Computational circuits are continuously monitored by modulo-3 verification and dual circuit confirmation. If an unrecoverable error is detected, the faulty unit is automatically stopped. A hardware implemented, automated fault dictionary is referenced immediately upon detection of the unrecoverable error. Therefore, the maintenance engineers know which modules to replace within seconds of the error detection.

6. The SUPER-UX Operating System

The primary operating system for all models of the SX-3 Series is SUPER-UX, a supercomputer enhanced operating system based on AT&T System V UNIX. Some of the major extensions added to support the requirements of supercomputing include the (a) Intelligent I/O Accelerator Subsystem, (b) Supercomputer File System, and (c) BATCH processing facilities.

Commitments for ongoing SUPER-UX development are to maintain compatibility with AT&T UNIX, IEEE 1003.1 (POSIX), relevant future standards as they are adopted, and to continue enhancing both performance and usability for the supercomputer community.

IAS (Intelligent I/O Accelerator System)

IAS is an operating system extension which intelligently buffers I/O data through a series of multi-level data caches. Sophisticated algorithms eliminate data thrashing while providing automated, user independent, high performance I/O, including transparent parallel I/O.

SFS (Supercomputer File System)

SFS is a file system extension which is optimized for large scientific data files. This compares to the standard System 5 file system (S5FS) which is designed for rather small data quantities, such as programs and shell scripts. SFS allocates file space in units of disk tracks or cylinders; S5FS allocates by sector units (512 bytes). Both file systems are fully, transparently supported within SUPER-UX.

BATCH Processing and Networks

One of the shortcomings of UNIX is the lack of batch processing capability. Since one of the primary environments of supercomputing is the batch workload, an implementation of NQS has been ported and coupled with enhanced scheduling and job control functionality.

A distributed production environment is further supported by NFS and TCP/IP networking capability, the latter including the telnet and ftp facilities. Future extensions will include OSI and FDDI networking products.

7. Compilers

FORTRAN is the primary programming language in scientific computing, and as such, is the basis of providing automated vector and parallel processing capabilities. The FORTRAN compiler, FORTRAN77/SX is based on the advanced vectorizing compiler used for the SX-2A. Enhancements are added to provide state-of-art automated parallel processing. Vectorization features include loop collapsing, automatic statement interchange, index migration, automatic inlining, and automatic loop unrolling. Automated parallel processing is initially implemented by microtasking techniques.

The C compiler, C/SX, will feature automatic vectorization and automated inlining. It will be suitable for applications development as well as systems and utility generation.

8. Summary

The SX-3 is the first Japanese machine to be able to use parallel processing to either increase overall throughput of multiple jobs and reduce the turn around time of a single job. However, NEC's basic design approach for high-speed processing is to pursue the ultimate scalar performance of a single processor as well as vector processing capability.

It should also be noted that one of the major technological progresses, which contributed to the realization of this type of multiprocessor system is in Silicon VLSI and high-density packaging technologies.

Fig. 1 SX-3 SERIES System Configuration

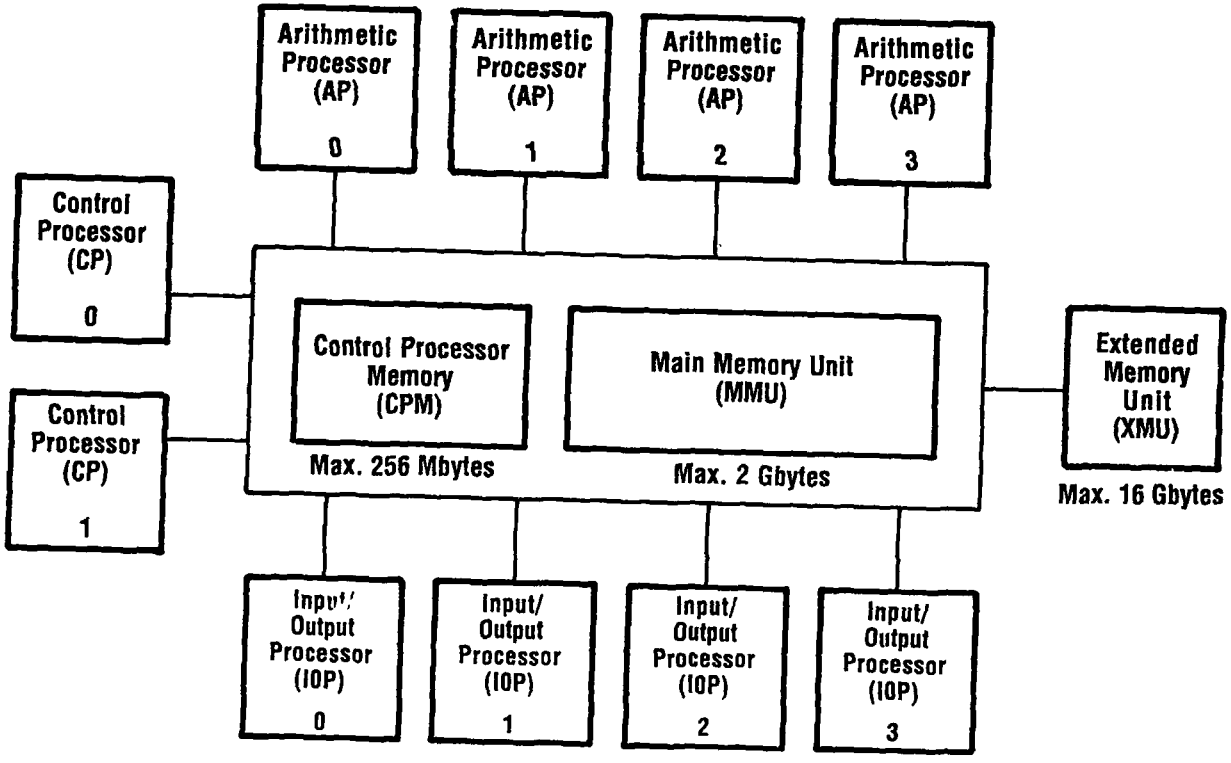
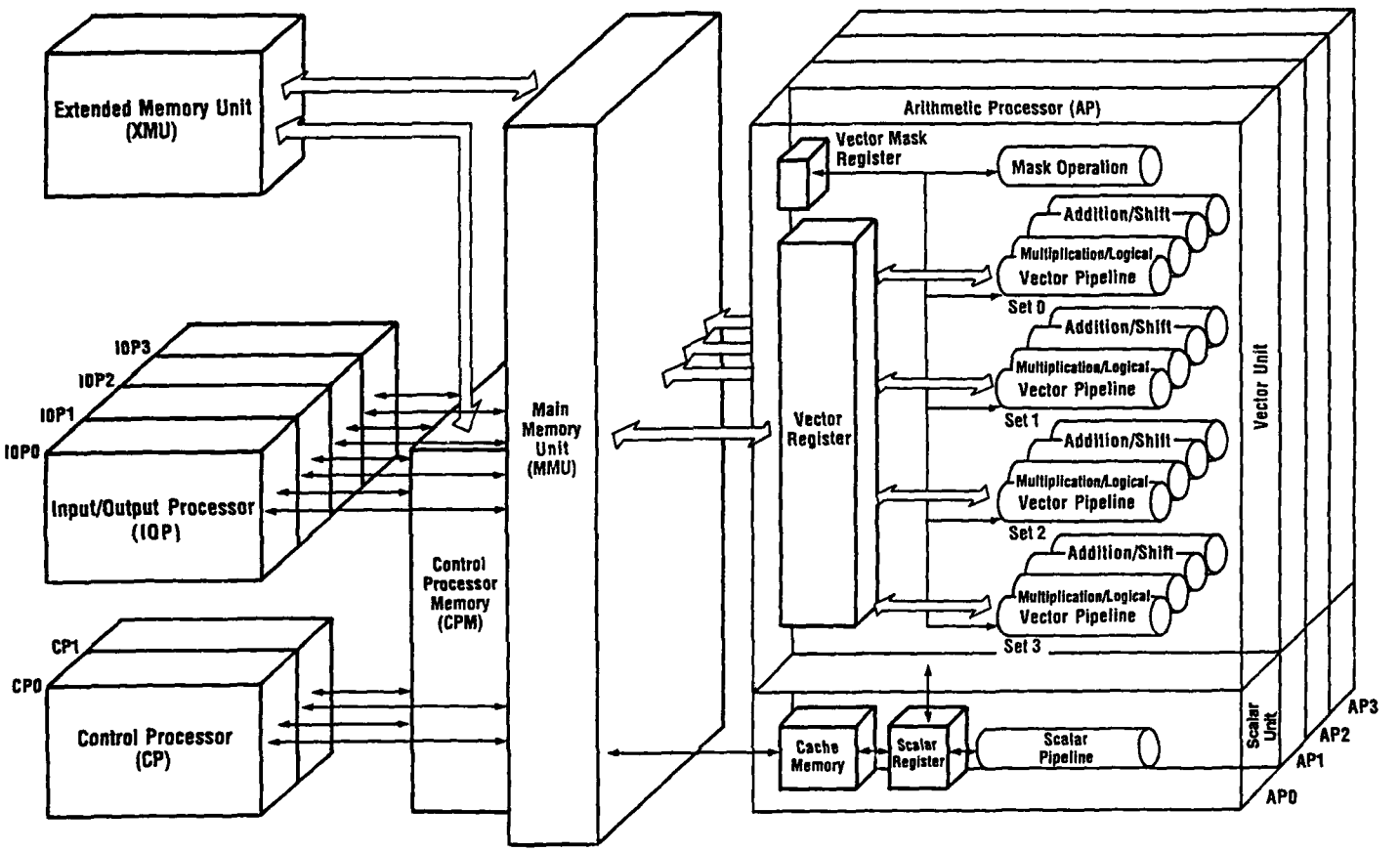


Fig. 2 SX-3 SERIES Arithmetic Processor Configuration



LARGE SCALE APPLICATIONS OF TRANSPUTERS : ACHIEVEMENT AND PERSPECTIVE

D.J. Wallace,
Physics Department, University of Edinburgh

Abstract

This paper gives an overview of large scale applications of transputers in the context of the Edinburgh Concurrent Supercomputer Project. This is built around a Meiko Computing Surface, with presently some 400 floating-point transputers and 1.7 Gbytes of memory. The first part of the paper gives an overview of the Project's origins and status and describes experience gained in providing a multi-user service. The second part gives examples of applications which are able to exploit effectively this processing power. Tools which facilitate the use of the machine for large scale computation and visualisation are also briefly described.

1 Project Origins

Work at Edinburgh on the use of parallel computers for Physics applications began in 1980, on the ICL Distributed Array Processor at Queen Mary College. This initiative was spearheaded by Stuart Pawley, whose main interest was in molecular dynamics, but the work rapidly expanded into lattice field theory as we appreciated the potential of this fine-grain SIMD machine. We were successful in acquiring a dedicated machine at Edinburgh in 1982, through funding from the Science and Engineering Research Council and an agreement for software development between ICL and the Edinburgh University Computing Service; a second DAP was donated by ICL in 1983. The machines had to be decommissioned in 1987, on the replacement of the ICL hosts which acted as the University mainframe resource. At that time some 180 publications had resulted from the work spanning many application areas; a summary of the work can be found in [1].

The anticipated replacement of the ICL host machines, and the now large community dependent on high performance computing obliged us to look for alternative sources of it. We were convinced then that the only way we would have access to the required power with the budgets we might expect was through exploiting novel architecture parallel machines. We already had a relationship with Inmos who were supporting a student in parallel computing for high energy physics, and were fortunate to obtain one of the earliest Meiko Computing Surfaces in April 1986, with the support of the Department of Trade and Industry and the Computer Board. This demonstrator system consisted of 40 T414 transputers each with 1/4 Mbyte, along with a display system, and was file served and networked through a microVAX host. The reliability of this system, the imminent loss of the DAPs and a survey of existing parallel machines formed the cornerstone of the proposal for the Edinburgh Concurrent Supercomputer (ECS). The proposal, in collaboration with Meiko, sought some £3.4M from the SERC, DTI and Computer Board to fund a machine built around 1024 T800 transputers (with floating-point capability) each with 1 Mbyte of memory, to provide an electronically reconfigurable multi-user resource. After some three months the three funding agencies agreed in principle to consider joint funding of the machine. Phase 1 funding for the machine infrastructure and compute resource of 200 T800s, each with 4 Mbytes, was secured during 1987, multi-user service for code development was established in September 1987,

Department of Trade and Industry	£1144K
Computer Board	£575K
SERC (Science and Nuclear Physics Boards)	£400K
Meiko Limited (excl. personnel)	£779K
Scottish Development Agency	£12K
Industrial Affiliation etc.(cash and kind)	£800K
Edinburgh Univ. 3 Comp. Officers, infrastr., plus cash of	£102K

Table 1: Funding support for the ECS as at December 1989

and the first T800 compute resource installed at the end of that year.

2 Present Status

2.1 Funding

The present level of commitment to the Project is shown in Table 1. The initial DTI contribution funded the infrastructure for the machine, including cabinets, inter-cabinet link boards, 32 host boards for code development, 4 display systems and some 5 Gbytes of memory. A second phase of DTI support has enabled us to expand the compute resource with another 195 T800s and 900 Mbytes of memory. The Computer Board and SERC support includes some £600K for hardware, which provided the Phase 1 T800 compute resource, and a frame grabber and fast I/O system. Their contributions also contain some funds for software, maintenance, and two to three people for up to five years. Meiko have also contributed very significantly in discount, maintenance and software; in addition they site two people at Edinburgh and have considerable in-house software activity to meet the Project requirements. The University has committed £102K in cash as well as three computing officers for service management and support. At the time of writing (December 1989), there are a total of 15 personnel in the core of the Project (i.e. excluding specific application teams); Meiko support and the use of the income from the industrial affiliation scheme in supporting 10 of these people have been crucial factors in the successful launch of the Project. It is anticipated that substantial recurrent funding from the Science and Engineering Research Council and from the Computer Board will enable the establishment of a parallel computing centre early in 1990 incorporating work on two AMT DAPs and departmental shared memory resources, as well as the ECS Project.

2.2 Hardware Configuration

General information about the transputer, the Computing Surface and the occam language can be found in [2]. The machine organisation is shown schematically in figure 1. The computational model is that of a network of workstations and file servers. It is realised by a transparent communication spine which is also based on transputers and off which hang the various resources of the machine. The microVAX host of the original demonstrator system is now retained as one of the file servers, and to provide a VMS environment. The user can also file serve off a number of Hewlett-Packard disks running a Berkeley 4.2 BSD

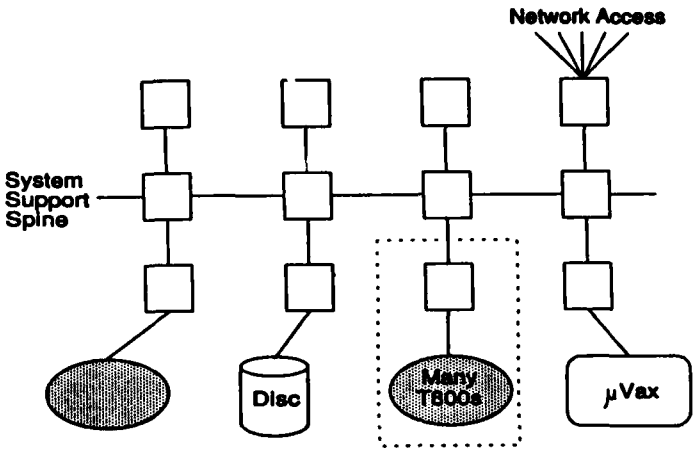


Figure 1: Schematic diagram of Computing Surface organisation

file system. Access is provided by direct lines and the facility is networked with ethernet and X.25 connections. To provide multi-user capability the compute resource is divided into domains; the size and number of these is controlled by the system manager, and may be changed (exploiting the software reconfiguration) for example to match day and night time user needs. At present there are typically some 30 domains on the service machine; a separate machine for system development can support a further 8 users. At login, the user is presented with a menu of domains; she boots an available domain and connects to a file server and has then a personal parallel machine. Back-up is provided by Exabyte cartridges.

2.3 Operational Aspects

Experience at Edinburgh in the offering of a multi-user service on novel architectures is based on the two mainframe-hosted DAPs between 1982 and 1987 and on the ECS since 1987. Although these machines are of very different architectures, the operational problems of supplying and maintaining a multi-user service are very similar. The problems are principally of two distinct types: those of actual day-to-day supply of the service, and those of providing effective support to the users of the service.

The first class of problems is mainly concerned with the fair allocation of machine resource. Such problems as the amounts of on-line and off-line file space can be critical as all the experience gained shows that the amounts of disk requirements may be phenomenal both in total size and in their rate of growth. The gift of disks under the Hewlett-Packard Affiliation agreement has therefore been particularly valuable. Another major problem is that of ensuring access for long runs, which are frequently required and can prove to be a scheduling problem.

The experience at Edinburgh is that there is almost always pressure on the facility, no matter how much parallel resource is available. In the ECS we have two opposing requirements: to partition the resource so that many users may access it at the same time, and to provide the largest domains possible for a fewer number of users for long production runs.

The second class of problems concerns offering a reasonable level of user support. Because of the different nature of the coding strategy and structure, typically a very high level of specific expertise is required of any user support staff. At this stage in parallel computing software development, such staff have to be specialists, while traditionally user support personnel in University Computing Centres have had a more widely based and general background. This problem will of course become less acute as parallel programming environments improve.

3 System and Utilities

A stand-alone multi-user system MMVCS has been in use since September 1987. It provided the user with the occam programming system, OPS, and UNIX[1] file serving utilities. The full UNIX-like system MEiKOS was released into the Service Machine in October. Among the work performed on the development machine is a porting of communications software and the AT&T System V.3 utilities. There are C and Fortran compilers for single transputers and a range of utilities is available or under development at Meiko; Meiko's parallel programming environment for Fortran and C code with message passing, CS-tools, has been used in a number of projects over the summer. Standard packages which have been ported include GKS.

There has been considerable effort at Edinburgh to develop utilities which provide greater flexibility and ease of porting of codes to the Computing Surface. The cornerstone of this effort is the development of fast topology independent adaptive message passing systems [3,4]. The utility, called Tiny, explores the transputer configuration at run-time and sets up point-to-point communications and broadcasts. Code does not have to be recompiled to run on different configurations. The harness also has fault tolerant capabilities; provided a booting path is available through each transputer, efficient routing between pairs will be set up even if some of the links are defective - they will simply not be utilised in setting up the routing tables. The utility can be called from Fortran or C as well as used in an occam program. Various flags permit the user to specify whether data must arrive in the same order as it was sent etc., and the size of the buffers can be varied to match the application requirements. The system has particularly good characteristics under heavy load; for example if messages arriving on link 0 cannot be forwarded on link 1 because the latter is blocked, other messages arriving on link 0 can be passed on by links 2 or 3 if these are available.

A recent major development has been the extension of this utility to provide deadlock-free communication. The approach is based on recursively casting spanning trees in the processor graph, followed by a reconstructive phase which repairs excessive damage. The method finds the shortest distance solutions for regular graphs such as grids, and for irregular topologies the mean interprocessor communication distance is only modestly increased, for example by around 25% for a random transputer graph of 256 nodes. It should be said that in practice the original Tiny has only rarely been known to deadlock unless the user has written an incorrect program or insufficient buffers have been provided; the deadlock-free Tiny is important however both as a matter of principle and for safety-critical applications.

The start-up latency for the utility is 17 microseconds on both the send and receive processors, and the through-routing CPU overhead time is 19 microseconds per node, with a realised bandwidth of around 1.4 Mbytes per second per link. These figures compare

[1]UNIX is a trade mark of AT&T Bell Laboratories

favourably with the iPSC/2 for communication over up to three links, particularly when one bears in mind that we are comparing hardware and software through-routing capability, and that they refer to a lightly loaded network.

From the user point of view, utilities like Tiny are important because they assist flexible and portable code development. It is already the basis for a number of other topology independent utilities for example for task farming and 3-d graphics. It has also been used to explore the properties of irregular graphs [5] which have many attractive features, including mean interprocessor distance and diameter which increase only logarithmically with the number of (fixed-valency) nodes, and are very close to the optimal bound. Such graphs provide a framework for shared memory emulation on distributed memory machines, following the work of Valiant [6].

Although the implementation does not utilise random graphs, it should also be mentioned here that Linda has also been implemented on the ECS [7]. The kernel has recently been rewritten in C using Tiny [8] and tools will soon be in place to allow C-Linda programs to be developed and run on any topology; it is also intended that a Prolog interpreter with added predicates will be ported to provide a Prolog-Linda implementation.

4 Node Performance

Reasonable performance on a single node is obviously an important prerequisite for supercomputer performance across an array. We summarise here experience gained in a range of applications.

For well-structured Fortran or C code which is floating-point intensive, benchmarks for single precision give up to 0.6 or 0.7 Mflops per node. A number of applications written in occam are running at in excess of 1 Mflops per node. Some comparisons relative to workstations are given in the examples of applications below.

To achieve maximum performance with minimum effort, BLAS1 routines have been written in assembler for a single T800 [D. Roweth and L.J. Clarke, unpublished]. The table below illustrates the performance obtained in these routines.

routine	Mflops	routine	Mflops
saxpy	1.17	daxpy	0.72
sdot	1.17	ddot	0.78
snorm	1.58	dnorm	1.05
sscal	0.78	dscal	0.49
ssum	1.35	dsum	1.03

5 Applications

There are currently over 300 registered users of the facility. The second edition of the Project Directory [9] gives a snapshot of around 100 projects under way as of September 1989. The following sections therefore contain only a few examples of some of these projects, to illustrate areas where it would not have been possible to do the work without the ECS facility. The topics fall into three broad categories: visualisation and image processing, simulation and control, and interactive design.

5.1 Visualisation and Image Processing

The four display systems and frame grabber facility support a wide range of activities in these areas, including constructive solid geometry, and NMR and other medical imaging techniques. The existence of the 3-d graphics utility will further expand this work. We focus brief comments on two specific examples.

5.1.1 Radiosity

Radiosity can produce 3-d visualisation of very high quality. The method is based on (i) dividing the surfaces in the scene into N patches and determining line of sight visibility between pairs of patches, (ii) solving the N simultaneous equations for the brightness of each patch, and (iii) rendering the scene using z-buffering or ray-casting. Rendering the scene from different view-points requires only stage three, and changing the lighting only stages two and three.

This algorithm has been implemented at Edinburgh to run on an array of arbitrary size [10]. The natural parallelism in the calculations of the form-factors (the coefficients relating the contribution of the illumination of one patch due to the light from another) and in the iteration of the matrix of these form factors to determine the solution enables the method to run effectively on hundreds of transputers.

5.1.2 Fractal Landscapes

Fractals are well known to produce realistic and spectacular landscapes, and some of the larger Hollywood special effects studios such as Lucas Film are starting to use them in films. The conventional approach constructs the surface from random numbers starting from a plane.

An undergraduate project in Computer Science [M. White, unpublished] was undertaken to extend this method to construct fractal planets. The project was originally conceived as a serial program to run on a SUN workstation. However, as the student discovered, the method has enormous computing requirements. To obtain results, a last minute decison had to be made to move to the ECS. The porting took two weeks. The resulting C code ran 20 times faster on one transputer than on a SUN 3/60 (without floating point support), and the level of natural parallelism in the ray-casting phase enabled the largest domain available on the ECS (130 T800s) to be used efficiently. The most complicated picture took one hour on this domain, and would have taken more than 2500 hours (more than 111 days) on the SUN. Would-be space travellers will be interested to know that these hospitable planets with their oceans and continents can be visited and explored on the screen.

5.2 Simulation and Control

Simulation in scientific and engineering problems offers great scope for parallelism. The use of data-parallelism (geometric decomposition) is rather well understood now (see for example [11] and references therein), in particular the conditions under which the application can be scaled up to run at the same speed on arbitrary numbers of processors. Simply stated, it is sufficient that the dimension of the connectivity of the computer be at least as large as the spatial dimension of the problem - usually two or three. Thus it would

be theoretically advantageous to have at least six links on transputers to ensure scaling in three-dimensional problems, although we have not encountered serious problems in practice with present hardware limitations (and hardware through-routing seems in any event a more important target). We focus here on four examples.

5.2.1 Lattice Field Theory

Applications in High Energy Physics are outwith the concerns of this Meeting, but some comments are in order here, because they are traditionally amongst the first to be mounted on high-performance parallel machines, and although they are not communications-intensive, they provide a more relevant and convincing benchmark than the Mandelbrot set. In brief they involve simulations on a four-dimensional lattice (approximating space and time), in which the variables are the elementary particles of the theory (quarks, gluons, electrons etc.). Because the spin-half particles are Fermions and obey the Pauli exclusion principle, special algorithms must be developed. These algorithms typically involve a sparse matrix inversion at each time-step of the simulation. This matrix has as one of its indices the space-time lattice points; on a 16^4 lattice this implies a 65000×65000 matrix. Effort at Edinburgh (for a review see [12] and references therein) has been focused on the phase transitions resulting from the interactions of Fermionic particles, and on the possibility of models unifying the known fundamental forces without the need for the elusive Higgs scalar particle. These codes can be run on any size of domain; on 130 T800s they deliver more than 100Mflops.

5.2.2 High Temperature Superconductors

Superconductivity is another area of science where Fermions (electrons) play a central role. This field has returned to prominence with the discovery of materials that superconduct above liquid nitrogen temperatures. The properties of these materials are not yet well understood, and computer simulation is potentially an important tool.

Jones and Yeung at Queen Mary College have implemented a parallel variational Monte Carlo method to study a model of these materials which may be both superconducting and magnetic [13]; the method was developed and tested on a small local machine and benchmarking and production runs done on the ECS. The algorithm achieves near-linear scale-up which indicates that it will run with acceptable efficiencies on arrays of hundreds or more T800s. To date the program has been successfully run on a 22×22 lattice, probably the largest of any attempted so far. This implementation on transputers makes possible the study of the physics for a range of practically interesting parameters.

5.2.3 Neural Network Models

The 'wetware' components in the nervous system have typical timescales of the order of milliseconds, in contrast to the nanoseconds of semiconductor hardware, and our remarkable neural processing capability is due in part at least to its massive parallelism. Most neural network models reflect this parallelism and are amenable to study on parallel computers. Activity on the ECS covers many aspects, including general pattern processing and optimisation.

The most widely studied network for practical applications is the layered network, which is trained by error correction methods (back-propagation) so that the desired processing is

achieved for given input data with known outputs (for example, in medical diagnosis, the input could be symptoms and the output likely diagnosis and suggested treatment). The key idea is that the net should capture the intrinsic correlations of the data, so that its capability generalises to new data in the domain in which it has been trained. A simulator has been developed on the ECS [14] which enables the user to specify the number of layers in the network, the number of nodes in each layer, the nature of the connectivity between layers, etc. This is then automatically mapped down on to the transputer array. The simulator has been used at Edinburgh for studies of content addressable memory and storage properties, for exploring training strategies, for protein secondary structure prediction and for texture discrimination. In the last case, the simulator runs at about 10 Mflops on an array of 17 transputers; on larger problems, the performance should scale up. External use includes in the oil and defence industries, and in assessment for credit scoring in the banking industry.

Neural networks also provide a framework for tackling optimisation problems. An example of current interest includes the use of the method of analogue neurons for image restoration in the framework of Geman and Geman. Interest in optimisation problems has evolved also into the study of genetic algorithms, in particular for the optimisation of transputer array topology for some application. For further examples and references on ECS work in these areas see [15].

5.2.4 Chemical Process Simulation and Control

In most chemical plants and all oil refineries distillation is an important operation. Much attention is devoted to controlling distillation columns efficiently, since these are one of the largest energy sinks in the process. Problems affecting efficient control include: columns consisting of many stages which may be slow to respond to feedback control actions; and stringent specifications on final product purity. If computer simulation of the process is fast enough, an efficient control plan can be designed and implemented to keep the column at the desired production specifications.

The implementation on the ECS involves a chain of transputers for each column, each transputer being responsible for a module or plate in the column. In the simulation, the dynamic evolution of the column's state is interactively displayed. Two control policies have been examined: feedforward control with change of composition of feed; and product changeover, i.e. switch of production from product A to product B with the minimum 'off-spec' production.

The conclusions from this work by McKinnel and Ponton [16] are that modestly-sized transputer-based systems are now sufficiently powerful and cost-effective to be considered for dedicated use 'on-line' for this problem.

5.3 Design: Optimisation in stressed membrane suface structures

An important Engineering application is being undertaken by Moncrieft and Topping at Heriot Watt University [17,18]. This concerns the optimisation of cutting patterns for tension structures such as the Schlumberger Headquarters in Cambridge. The aim is to improve current methods for optimising stress distributions across the surface. The conventional non-linear modelling approach starts from some arbitrary surface and progressively relaxes this to an equilibrium configuration; this requires specifying a desired surface stress distribution and an appropriate topology. The real difficulty is that one must define a cutting pattern

for flat cloths from which the real surface will be fabricated. One is therefore faced with optimising the cutting pattern by varying the planar cloth geometry. This is just one example of an important class of non-linear optimisation problems in which new design points are iteratively determined by displacement along a search vector by a specific amount.

In practice a hierarchical CAD data structure is used, in which the surface patches are defined in terms of discrete space curves. These curves are in turn defined using lists of atomic control points, which are the variables over which the optimisation is performed. The non-linearity of the problem requires that the gradients for the determination of the search vector must be calculated numerically. Having determined a search vector, a linear search must be preformed along this direction to determine a good step length. The gradients of the objective and constraint functions with respect to each control point can be calculated independently, as can the trial step lengths, so that the whole calculation can be done efficiently by task farming. Since in real structures there may be hundreds of variables, and this is by far the most time-consuming part of the computation, the potential for parallel computation is vast.

The method has been implemented on the ECS, and near-linear speed-up observed. The use of the ECS has enabled interactive design to replace overnight batch runs on the Edinburgh University mainframe.

6 Concluding Remarks

In this paper we have presented a snapshot in the development of a large transputer array facility, with emphasis on a number of applications where performance is a crucial factor in the feasibility and success of the problem. The range of work underlines the potential of massively parallel machines in supercomputer applications. We have also stressed the importance of the development of tools and environments to facilitate ease of porting and future portability. For distributed memory machines, we are still at the beginning of this process, but already it is clear that sufficient progress has been made to establish the scientific value and commercial viability of this technology. In drawing an overall perspective, one would conclude that the two key factors for its continued commercial success are tools for efficient porting of large C and Fortran codes, and competitive microprocessor development, incorporating commensurate communications and routing capabilities.

Acknowledgements

The Edinburgh Concurrent Supercomputer is a collaborative project with Meiko Limited, supported by major grants from the Department of Trade and Industry, the Computer Board and SERC. It is a great pleasure to acknowledge the work of many colleagues without whom the ECS project would not have existed, and to thank them for stimulating discussions and for help in the preparation of this paper. I apologise to those whose work I was unable to contain in it. This is an edited version of a paper prepared for the Conference on Applications of Transputers, Liverpool, August 1989 [19]; I thank the organisers and publishers for their permission to use the article in this way.

References

1. KC Bowler, AD Bruce, RD Kenway, GS Pawley, DJ Wallace and A McKendrick, Scientific Computation on the Edinburgh DAPs, University of Edinburgh Report, December

1987.

2. KC Bowler, RD Kenway, GS Pawley and D Roweth, An Introduction to Occam 2 Programming, Chartwell-Bratt, Bromley 1987.

3. MG Norman and S Wilson, The TITCH User Guide.

4. LJ Clarke, The Tiny User Guide, available from the ECS Project, Edinburgh University Computing Service.

5. D Prior, NJ Radcliffe, MG Norman and LJ Clarke, What Price Regularity?, Concurrency: Practice and Experience, to appear.

6. LG Valiant, Optimally Universal Parallel Computers, in Scientific Applications of Multiprocessors, eds RJ Elliott and CAR Hoare, Prentice Hall International Series in Computer Science 1989, p. 17.

7. JP Dourish, Implementing a Parallel Lisp System on a Transputer Array, ECS Project Report, 1989.

8. N MacDonald, ECS Newsletter 9, available from the ECS Project, Edinburgh University Computing Service.

9. J Wexler and GV Wilson, Edinburgh Concurrent Supercomputer Project Directory, 1989.

10. D Prior, An Architecture that Exploits Parallelism in Radiosity Calculations, in Proceedings BCS Conference on Image Processing, 1989.

11. GC Fox, MA Johnson, GA Lyzenga, SW Otto, JK Salmon and D Walker, Solving Scientific Problems on Concurrent Processors, Prentice Hall, New Jersey, 1988.

12. SP Booth, KC Bowler, DJ Candlin, RD Kenway, BJ Pendleton, AM Thornton, DJ Wallace, J Blair-Fish and D Roweth, Large Scale Applications of Transputers in HEP: the Edinburgh Concurrent Supercomputer Project, in Proc. Conf. Computing in High Energy Physics, Oxford, 1989.

13. RB Jones and W Yeung, Variational Studies of the 2-D Hubbard Model on Transputer Arrays, Queen Mary College preprint, 1989.

14. GD Richards, Implementation of Back-propagation on a Transputer Array, Proc. 8th Technical Meeting of the Occam User Group, ed J Kerridge, IOS Amsterdam 1988, p. 173.

15. MG Norman, NJ Radcliffe, GD Richards, FJ Smieja, DJ Wallace, JF Collins, SJ Hayward and BM Forrest, Neural Network Applications in the Edinburgh Concurrent Supercomputer Project, in Proc. NATO Adv. Study Inst. on Neural Computing, Les Arcs, 1989.

16. R McKinnel and J Ponton, Use of a MIMD Parallel Computer for Simulation and Control of Distillation, Chemical Engineering preprint, University of Edinburgh, 1989.

17. E Moncrieff and BHV Topping, Membrane Structure Cutting Pattern Generation, Proc. 1st Int. Oleg Korensky Conf. on Tension Structures, Institution of Structural Engineers, London, 1988.

18. E Moncrieff and BHV Topping, The Optimisation of Stressed Membrane Structures using Parallel Computational Techniques, in Optimization in Civil Engineering, NATO ASI Series, Kluwer, 1989, to appear.

19. DJ Wallace, Supercomputing with Transputers, in Proceedings, Applications of Transputers (Liverpool, 1989), eds. TL Freeman and C Phillips, IOS, Amsterdam, 1990, to appear.

Compiler Optimizations for Superscalar Computers

Monica S. Lam

Computer Systems Laboratory
Stanford University
Stanford, CA 94305

Lam@cs.stanford.edu

Abstract

Capable of executing multiple scalar operations per cycle, a superscalar architecture can parallelize not just vectorizable programs, but also code containing recurrences and data dependent control flow. This paper presents an overview of the compiler optimizations that are crucial in harnessing the computation power of superscalar machines. These optimizations include high-level loop transformations to find parallelism and improve the efficiency of caches, software pipelining and hierarchical reduction techniques for scheduling instructions, and modulo variable expansion for assigning registers.

Recent advances in hardware technology have made superscalar architectures amenable to single-chip implementations. The combination of cheap hardware to provide a high raw computing power and sophisticated compiler technology to effectively use the parallelism can produce extremely low-cost, high-performance workstations that are easily accessible to the general scientific and engineering community.

1. Introduction

A superscalar computer is a uniprocessor that can execute two or more scalar operations in parallel. The operations are individually specified in the object code; this is distinct from vector machines which expand vector instructions into series of parallel operations. The parallelism of a vector instruction is defined for each vector machine at machine design time; on a superscalar machine, a parallel execution schedule is created uniquely for each program, by either hardware or software. As a result, superscalar machine organizations are more versatile and effective in using the hardware resources in the system.

Superscalar machines existed long before the term was coined. The IBM Stretch [5], the CDC 6600 [24] and the IBM 360/91 [2] are all superscalar architectures that can execute multiple operations in parallel. These machines all implement a sequential instruction set with hardware that schedules the instructions dynamically. Besides hardware, software has also been used for instruction scheduling. Epitomizing the class of superscalar machines that rely on software for scheduling instructions is the

VLIW (Very Long Instruction Word) architecture [13]. Each wide instruction word explicitly specifies the operations to be executed in parallel. Examples of such machines include the Multiflow's Trace machines [8], the Carnegie Mellon's processors for the Warp systolic array [3] and the Cydrome's Cydra 5 [9]. The recent hardware technology advances have made software scheduled superscalar architectures amenable to single-chip implementations. A follow-on of the Warp processor, the Carnegie Mellon and Intel's iWarp processor integrates high-performance computation and systolic communication in a single component [6]. The Intel's i860 is a single-chip microprocessor that can perform up to 100 million floating-point per seconds (MFLOPS) using a dual-instruction word format [17].

The development of the recent superscalar architectures presents an exciting prospect to the engineering and scientific community. As technology improves, the superscalar processor performance is expected to grow. The superscalar processor provides a more flexible form of instruction parallelism in a low-cost package. The impact is that high computing power can be easily provided in a low-cost desktop workstation that is widely accessible to engineers and scientists. The high-level of integration also makes these scalar processors a useful building block for large-scale multiprocessing, thus delivering an aggregate computation bandwidth higher than ever before.

The parallelism of a superscalar machine may be managed in hardware or software. The hardware approach schedules the instructions dynamically, thus hiding parallelism from the architecture. The instruction set architecture can therefore be made compatible with that of an existing sequential machine. Run-time scheduling, however, requires more hardware logic, which may result in a slower clock cycle or longer latency in instruction execution. In the software approach, the parallelism is exposed at the architecture level, and the compiler is responsible for specifying the parallel operations to execute. By analyzing the entire program statically, the compiler can exploit higher level program semantics and rearrange the code globally to derive a better schedule.

To harness the raw computing speeds of software-scheduled superscalar processor in applications, compiler technology is crucial. The compiler hides the parallelism from the programmer, so the programmer can develop applications easily using a high-level sequential language. This approach has the additional advantage that the same sequential programs can now easily be ported to other current and future machine architectures.

In this paper, we first describe the characteristics of the superscalar architecture and the issues in compiling code for such machines. We then present a set of compiler optimizations, showing how the functionality of the processor can be used in programs. We then close with a discussion on the performance of these superscalar machines.

2. Superscalar Architectures

Common to all superscalar processors is the presence of parallel and/or pipelined functional units. Like any machine that employs parallelism and pipelining, a program running on a superscalar seldom achieves the peak computation rate of the machine. If a superscalar processor has n functional units, or a functional unit with n pipeline stages, n independent operations must be present at all times to utilize the

machine fully. If no parallelism is found, the machine may operate at 1/*n*th of the peak rate. Therefore, for a superscalar to be effective, it is important that the scheduler can find enough independent operations to execute in parallel.

Before we discuss the scheduling techniques, let us first take a look at the fundamental limit the hardware imposes on the execution speed of a program. Even if there are enough independent operations, the full computation power of a superscalar may not be brought to bear on an application because of specialization. The processor typically consists of a set of specialized functional units, some memory access units, possibly different arithmetic units, and an instruction branch control unit. For example, a program that requires no multiplications will not be able to take advantage of the multiplication unit on the processor.

The hardware of a system is typically designed such that the distribution of the computational units matches the distribution of operations in a typical program. From the statistics of a large set of numerical applications [18], we have observed that there are about as many floating-point arithmetic operations as memory operations. About 60% of the memory operations are read operations, and about 70% of the floating-point operations are additions. On a machine that can execute one memory read, one memory write, one floating-point addition, and one floating-point multiplication in a single cycle, the adder is often the critical resource and is followed by the memory read unit.

Besides the utilization of the functional units in a processor, it is also important to consider the memory subsystem. To support a high computation bandwidth, a processor must also have a similarly powerful memory subsystem. For a vector machine, the more restricted mode of operation permits the use of vector registers and efficient block transfers between the memory subsystem and the registers. Being able to support a less regular form of parallelism, a superscalar architecture requires a more flexible memory system. The concept of memory hierarchy has been shown to be useful in reducing the average access latencies for general-purpose machines. A cache can also reduce the number of memory accesses which can be important in a multiprocessor environment.

Unfortunately, a cache sometimes behaves rather poorly for numerical code. Because of the large data set used, data brought into the cache may be flushed out before they are reused. The cache hit rate can fluctuate widely depending on, for example, whether a matrix operand is in the cache. This may greatly affect the overall speed obtained due to the large difference between cache and memory speeds. While a cache is normally transparent to compilers for general-purpose programs, it is beneficial to optimize the cache behavior in superscalar compilers.

In many ways, a superscalar compiler faces similar issues as those of a vectorizing compiler. The compiler must extract parallelism from sequential programs and try to use the parallel, specialized functional units effectively. The compiler must also manage the cache; this is analogous to the management of vector registers in vectorizing compilers. Though the issues are similar, a superscalar machine presents new challenges to compiler optimization. The parallelism must be managed at the scalar operation level and the parallelism exploitable is not regular like vector instructions.

3. Overview of Compiler Techniques

There are two levels of compiler optimization: the loop level and the instruction level. The loop level involves higher level transformations on the loop structure. These transformations are useful both for bringing parallelism to the innermost loop as well as improving data locality. This high-level restructuring prepares the loop for low-level instruction scheduling.

The instruction level optimization consists of instruction scheduling and register assignment techniques. The scheduling problem is to find the shortest instruction schedule that satisfies the constraints imposed by the machine resources and the program semantics. In particular, since most of the computation time is spent on innermost loops, it is important to schedule such loops efficiently. *Software pipelining* is a scheduling technique that exploits the repetitive nature of innermost loops to generate highly efficient code for processors with parallel, pipelined functional units [19, 22, 25]. Another code scheduling technique used with software pipelining is *hierarchical reduction*, a technique that abstracts control constructs as operations in a basic block, so the same scheduling algorithms can be applied to within and across basic blocks. For example, using hierarchical reduction, software pipelining can be applied to all innermost loops, including those containing conditional statements. Hierarchical reduction makes it possible to obtain a consistent performance improvement for many more programs. Interacting with code scheduling is register assignment. When the same register is assigned to different variables, their uses must be serialized, thus constraining the parallelism in the computation. Therefore, the register assignment must also be considered hand-in-hand with instruction scheduling.

In the following, we first present an overview of the analysis techniques necessary to support both loop level and instruction level parallelism. We then discuss each of the optimizations: loop level transformations, software pipelining, hierarchical reduction and register assignment.

Program semantics produces two kinds of constraints: control dependences and data dependences. A conditional branch instruction must first be executed to determine the instruction to execute next. This sequencing constraint is known as *control dependence*. An operation cannot execute until all its operands are produced. This sequencing constraint is known as *true data dependence*. To ensure that a read operation always reads the latest value produced, the order of the write operations on the same location must also be observed. This sequencing constraint is known as *output dependence*. Furthermore, since a data location may hold different values at different times, a value must not be overwritten before its use. This form of data dependence is known as *anti-dependence*.

The compiler must first extract dependence constraints from the program. The analysis algorithms are similar to those previously used for vectorizing and concurrentizing compilers. The control dependence can either be obtained through analysis of the flow graph [11], or simply retained from the syntactic control structure of the program [16]. For data dependence, since array references are very common in numerical code, it is important to determine if two array references can refer to the same location, and thus may share a dependence relationship between them. Various dependence tests have been proposed for disambiguating between array references whose indices are an affine function of loop indices [1, 4, 27].

The dependence information was used previously only for source-to-source loop transformations. For a superscalar machine, this information is used at both the loop and instruction level. In the compiler currently developed at Stanford, data dependence is captured in an intermediate representation that supports loop level transformations, and this same information can be used in the code generation phase.

4. Loop Level Transformations

High level code transformations are useful in bringing parallelism into the innermost loop, as well as improving the efficiency of the caches. Consider the simple example of a matrix multiplication:

```
FOR i := 0 TO n-1 DO
   FOR j := 0 TO n-1 DO
       FOR k := 0 TO n-1 DO
           C[i,j] := A[i,k]*B[k,j]+C[i,j];
```

The result of one addition is used by the addition in the next iteration of the loop. The additions must therefore execute sequentially; with an n-stage pipelined adder, an iteration takes at least n clocks. The multiplications, being independent, can execute in parallel with the additions. (Unlike a vector machine, a superscalar machine can execute some instructions in parallel even for recurrences.) To further increase the utilization of the machine, the compiler must perform higher level transformations so as to expose more parallelism in the innermost loop to the instruction scheduler. In this example, if the inner two loops are interchanged as follows:

```
FOR i := 0 TO n-1 DO
   FOR k := 0 TO n-1 DO
       FOR j := 0 TO n-1 DO
           C[i,j] := A[i,k]*B[k,j]+C[i,j];
```

The iterations in the innermost loop are now independent; as many iterations as necessary can execute in parallel to fully utilize the hardware resources of the machine. Therefore, when the innermost loop does not contain enough parallel operations to keep the hardware resources busy, high level transformations, similar to those used in vectorizing and parallelizing compilers, should be applied.

For superscalar machines with caches, high level transformations can also be used to improve overall performance by reducing the cache miss rate. Consider a machine whose cache is relative small in comparison with the matrix size. The objective of the optimization is to minimize memory accesses by reusing data in the cache as much as possible. In the optimized program above, the innermost loop accesses rows `k` and `i` of matrices `B` and `C`, respectively. The same row of `C` is used in the next outer loop, but the `B` data will not be reused until the next iteration in the outermost loop. If the data size is large compared to the cache, even the `C` data may not be in the cache, let alone the `B` data. Maximum reuse is obtained if we can block, or tile, the computation as follows:

```
FOR ii := 0 TO n-1 BY b DO
   FOR kk := 0 TO n-1 BY b DO
       FOR jj := 0 TO n-1 BY b DO
           FOR i := ii TO min(ii+b-1, n) DO
               FOR k := kk TO min(kk+b-1, n) DO
                   FOR j := jj TO min(jj+b-1, n) DO
                       C[i,j] := A[i,k]*B[k,j]+C[i,j];
```

Each of the matrix elements brought into the cache is reused `b` times before it is removed from the cache. The value of `b` is chosen to maximize the cache utilization.

Previous research on data locality has provided ways to predict the cache behavior of a loop nest. Gannon et al. [14] use *uniformly generated references* to find where locality exists in a nesting of loops. They also discuss how to choose which array elements should go into the cache for a given loop. Porterfield [21] estimates cache behavior for a loop nest assuming that the cache uses the least recently used replacement policy, and may block a loop if the cache cannot hold all the data in an iteration. Gannon et al.'s and Porterfield's estimates can be used to evaluate the data locality of entire loop nests obtained by different sets of transformations.

Loop transformations beneficial to data locality and parallelism for superscalar machines include loop interchange, reversal, skewing and tiling. Wolf and I have developed an efficient algorithm to search through the space of these transformations and generates code that displays data locality and parallelism in the innermost loops [26]. We reduce the optimization problem to placing the maximum number of loops identified to carry locality in the innermost tile. Using this goal and the legality considerations of tiling, we can significantly prune the search space to find the best set of transformations. How tiling improves data locality has been illustrated by the example above. The conditions that made tiling legal in the first place guarantee both coarse and fine grain parallelism within a tiled loop. Therefore, by tiling the loops, we generate code that exhibits both data locality and parallelism.

5. Software Pipelining

After performing the high-level transformations, the compiler can then apply the instruction level optimizations. The basic technique for obtaining parallelism is software pipelining. Let us introduce the concept of software pipelining by way of an example. Suppose we have a machine that can perform one load, one store, and initiate a 7-stage pipelined floating operation in one instruction, and suppose the code we want to execute is:

```
FOR i := 1 TO n DO
   A[i] := A[i]+1.0;
```

Assume for now that we can generate the addresses for the loads and stores in parallel with the rest of the computation; the specifics of this topic will be discussed in Section 7. The most compact instruction sequence to execute a single iteration of this loop is given in Figure 5-1. The operation `BLoop 1` branches back to label `1` if there are more iterations to execute. The schedule is sparse due to the heavy pipelining in the data path. (For machines with hardware interlocks, the `nop` instructions are used only at code scheduling time; they are omitted when the code is emitted.) If we simply iterate this schedule, the throughput of the loop is only 1 iteration every 9 clock ticks, and no resources are used more than 1/9th of the time.

```
1: LD
   FADD
   nop
   nop
   nop
   nop
   nop
   nop
   ST    BLoop 1
```

Figure 5-1: Object code for one iteration in example program

Techniques such as trace scheduling [12] depend on loop unrolling to generate enough parallel instructions to schedule. Suppose the loop body of the example is unrolled 9 times, the optimal schedule of the body of the unrolled loop is shown in Figure 5-2. (This instruction sequence assumes that the number of iterations is divisible by 9.) Each row in the figure corresponds to operations in an instruction, and each column corresponds to the computation of one iteration of the loop in the source program. Unrolling the loop 9 times improves the throughput to 9 iterations every 17 clocks. From the figure, it is clear that unrolling an additional iteration will only lengthen the schedule by one clock. This can be kept up until the iterations run out. A loop unrolled u times will have a throughput rate of $u/(u+8)$ iterations per clock, while the ideal throughput is 1 iteration per clock.

```
1:  LD
    FADD  LD
          FADD  LD
                FADD  LD
                      FADD  LD
                            FADD  LD
                                  FADD  LD
                                        FADD  LD
    ST                                  FADD  LD
          ST                                  FADD
                ST
                      ST
                            ST
                                  ST
                                        ST
                                              ST    Bloop 1
```

Figure 5-2: Optimal schedule for nine iterations

Although the schedule improves as we unroll more iterations, code expansion limits the degree of unrolling. Unrolling can therefore overlap only a small finite number of iterations; all the unrolled iterations must complete before the program branches back to another set of contiguous iterations. On a vector machine, such a loop maps directly into a vector instruction; a vector instruction can continually overlap operations from successive iterations to deliver a throughput of one iteration per clock cycle.

Software pipelining can achieve the same kind of performance obtained with vector instructions by continually overlapping operations from different iterations of a loop. The software pipelined program for the example above is shown in Figure 5-3. Code generated by software pipelining is compact. The code in the figure assumes that there are at least nine iterations in the loop. The first eight instructions constitute the *prolog* where more and more iterations of the loop start to execute. The *steady state* is reached after eight instructions, and is repeated until all iterations have been initiated. In the steady state, nine iterations are in progress at the same time, with one iteration starting up and one finishing off every clock. On leaving the steady state, the iterations currently in progress are completed in the *epilog*, the 10th through 17th instructions. This program achieves the *optimal* computation time by executing n iterations in $n+8$ clock ticks, where n is the number of iterations in the loop.

Software pipelining is different from loop unrolling in that a source iteration may span one or more iterations in the object code. If the machine contains pipelined functional units, the pipeline stages need not be emptied at iteration boundaries. In the example above, seven additions initiated in seven different

```
      LD
      FADD  LD
            FADD  LD
                  FADD  LD
                        FADD  LD
                              FADD  LD
                                    FADD  LD
                                          FADD  LD
                                                FADD  LD     BLoop 1
1:    ST                                                   FADD
            ST
                  ST
                        ST
                              ST
                                    ST
                                          ST
                                                ST
                                                      ST
```

Figure 5-3: Program of a software pipelined loop

iterations execute in parallel. The hardware pipelines are filled and drained only once on entering and exiting the loop, respectively. Software pipelining is especially beneficial for machines with high degrees of parallelism and specialization. The results are that optimal throughput can be achieved, and achieved with an extremely compact program.

5.1. The Problem

In this section, we first concentrate on the scheduling of loops containing a single basic blocks. Extending software pipelining to other loops is discussed with hierarchical reduction in the next section. The primary goal of software pipelining is to maximize the throughput in executing the iterations; it does not matter if the execution time of individual iterations is lengthened. Its secondary goal is to keep the code size down. In other words, the schedule must have a short steady state so that it can be captured in a relatively succinct code sequence. The problem is thus formulated as finding a common schedule for all iterations of the source loop, such that successive iterations are initiated with a constant interval, and the objective is to minimize this interval. In the example above, the schedule of an iteration is given in Figure 5-1, and the *iteration initiation interval* is one.

Software pipelining was originally derived from a technique for scheduling hardware pipelines, where the problem was formulated as inserting delays between hardware units to increase the overall throughput of the system [20]. New input is accepted by the hardware pipeline at regular periodic intervals. The software analog is to schedule operations within an iteration such that the iterations can be pipelined to yield optimal throughput.

Software pipelining has been used in compilers for several different architectures. The algorithm was first used in the ESL polycyclic architecture [22]. The polycyclic machine uses a specialized crossbar to simplify the scheduling problem for a subset of loops [23]. The same concept is also implemented in Cydrome's Cydra 5 [9]. Software pipelining is also used in the compiler for the FPS-164 machine [25]. The FPS-164 does not have any specialized support for software pipelining, and software heuristics are used to schedule the loops. We improved upon the FPS heuristics, especially in the algorithm for

scheduling recurrences, and implemented them in our compilers [7, 19] for the Carnegie Mellon's Warp and iWarp machines. Eisenbeis et al. applied software pipelining to the problem of scheduling vector instructions, and implemented a compiler that generates software pipelined vector code for the Cray-2 architecture [10].

Let us first describe some of the fundamental limits in scheduling a loop. There are two kinds of constraints: resource and precedence constraints.

Resource Constraints. Suppose a machine has $m(r)$ units of resource r, and an iteration of a loop requires $n(r)$ units of resource r, then a pipelined loop cannot execute faster than the rate of at most one iteration every

$$\max_r \left\lceil \frac{n(r)}{m(r)} \right\rceil$$

cycles. This equation reconfirms the notion that it is harder to fully utilize highly specialized functional units and the computation rate is limited by the resource with the highest demand.

In software pipelining, we must ensure that the resource commitment in each clock cycle of the steady state does not exceed the available resources. The resource usage of the steady state can be represented by a *modulo resource reservation table* whose ith entry contains the sum of the resources used in cycles $i, i+s, i+2s, \ldots$ of the schedule of an iteration, where s is the initiation interval of the loop.

Precedence constraints. While recurrences limit the throughput of the computation, a superscalar, unlike a vector machine, can often still find some parallelism in such loops. Consider the following example:

```
FOR i := 1 to 100 DO
    a := a + 1.0;
```

We must first read a before we write back into a in the same iteration, which in turn must precede the read operation in the next iteration. The flow graph representing the above example, assuming a seven-staged addition, is shown in Figure 5-4. Each edge is labeled by the number of iterations the dependence crosses and the delay between them. As shown in the figure, inter-iteration data dependences may introduce cycles into the precedence constraint graph. The precedence constraints in Figure 5-4 impose a delay of 9 clock ticks between load operations from consecutive iterations. That is, loops cannot execute at a rate greater than one iteration every 9 clocks.

We define the minimum delay, d, and minimum iteration difference, p, of a path to be the sum of the minimum delays and minimum iteration differences of the edges in the path, respectively. If we let c denote a cycle in the graph, the rate at which the iterations can be executed is one iteration every

$$\max_c \left\lceil \frac{d(c)}{p(c)} \right\rceil$$

cycles.

The maximum of the two bounds determined by resource and precedence considerations establishes a lower bound on the initiation interval. Therefore, a schedule that pipelines with an initiation interval

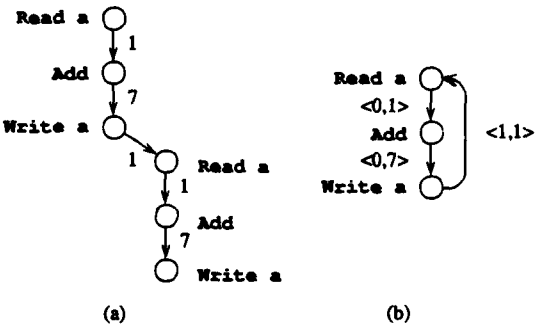


Figure 5-4: (a) Delays between operations from two iterations, and (b) precedence graph

meeting the lower bound is optimal. Empirical results show that this lower bound can indeed be met in many cases [18].

5.2. The Algorithm

The problem of finding the optimal software pipeline schedule is NP-complete. For acyclic graphs, the scheduling problem is tractable if operations execute in unit time and use only one resource. The polycyclic architecture [22] and the Cydra 5 architecture [9] use a specialized, rather expensive crossbar to provide exactly that property. All functional units of a polycyclic machine are interconnected through a crossbar. This crossbar has storage at every crosspoint to serve as a dedicated buffer for each pair of functional units. Therefore, there is never any contention in reading or writing data. Each operation thus consumes only one explicitly scheduled resource. For acyclic graphs, the minimum initiation interval is given by the bound discussed above and an optimal schedule can easily be found. However, the problem remains NP-complete for cyclic graphs even if operations use one unit of resource and execute in one unit time.

Without the specialized hardware to support software pipelining, both the FPS and the Warp/iWarp compilers use software heuristics. The algorithms used for scheduling acyclic graphs are similar, but the cyclic graph scheduling algorithm is significantly improved in our Warp/iWarp compilers. The algorithm for acyclic graphs is as follows: First, establish a lower and an upper bound on the initiation interval. The lower bound is calculated from the resource and precedence constraints; the upper bound can be found by the schedule of a single loop iteration. Next, find the smallest initiation interval. Simple linear search is used in our Warp/iWarp compiler because empirical results show that a schedule meeting the lower bound can often be found. The algorithm first sets the target of the initiation interval to be the lower bound value, and attempts to find a pipelinable schedule for the target initiation interval using the method described below. If the attempt fails, this process is reiterated by increasing the target initiation interval by one clock tick at a time.

The basic algorithm used to find a software pipelinable schedule for a target initiation is list scheduling. In list scheduling, the precedence constraints are applied first to determine the earliest slot in which an operation can be scheduled. The scheduler then goes on to try to satisfy the resource constraints; the

modulo resource reservation table defined above is used to determine if there is a resource conflict. The scheduler tries to schedule the operation in successive time slots until one that can accommodate its resource requirement is found. If s is the target initiation interval, and s attempts to satisfy the resource constraints fail, by the definition of modulo resource usage, this operation cannot fit into the schedule built so far. When this happens, the attempt to find a schedule for the given initiation interval is aborted and the scheduling process is repeated with a greater interval value.

As in the case of acyclic graphs, the main scheduling step for cyclic graphs is iterative. For each target initiation interval, the strongly connected components are first scheduled individually. The original graph is then reduced by representing each strongly connected component as a single vertex: the resource usage of the vertex represents the aggregate resource usage of its components, and edges connecting nodes from different connected components are represented by edges between the corresponding vertices. This reduced graph is acyclic, and the acyclic graph scheduling algorithm can then be applied.

Two main concepts are used in the algorithm for scheduling the strongly connected components. First, the precedence constraints are preprocessed so that the scheduler can easily determine the legal time span in which any node can be scheduled. Second, the order in which the instructions are scheduled is designed such that when the target initiation interval value is increased, the chance for success also improves. This is important because it would be futile if the scheduling algorithm simply retried the same schedule that failed.

A large set of evaluation data on the Warp/iWarp machine indicates that provably optimal schedules can often be found [18]. This shows that software pipelining does not require expensive hardware support. The code generated is compact; the body of a software pipelined loop is even shorter than the unoptimized code.

6. Hierarchical Reduction

The *hierarchical reduction* technique is designed to make software pipelining applicable to all innermost loops, including those containing conditional statements. The proposed approach schedules the program hierarchically, starting with the innermost control constructs. As each construct is scheduled, the entire construct is reduced to a simple node representing all the scheduling constraints of its components with other constructs. This node can then be scheduled just like a simple node within the surrounding control construct. The scheduling process is complete when an entire program is reduced to a single node.

The use of the construct structure exploits high-level control dependence knowledge [11] to increase the opportunity for code motion. As an example of the kind of code motions achievable with this technique, consider the following program:

```
FOR i := 0 to n DO
BEGIN
      statement 1;
      IF c THEN statement 2 ELSE statement 3;
      statement 4;
END
```

Although statement 4 comes after the conditional statement, it is not control dependent upon the result of the condition c. Once the program decides to execute another iteration, it can execute statements 1 and 4 in any order that satisfies the data dependences. For example, an operation in statement 4 can be executed before the conditional statement. The hierarchical reduction algorithm first schedules the THEN and ELSE parts of the conditional statement, and represents the entire construct with a single node that inherits the union of the scheduling constraints for each of the two parts of the conditional statement. The entire construct is then scheduled with statements 1 and 4. Operations corresponding to statements 1 and 4 may be reordered, they may also execute in parallel with the THEN and ELSE components of the conditional statement. At code emission time, any code scheduled in parallel with the conditional statement is duplicated in both the THEN and ELSE parts.

This control dependence knowledge when combined with software pipelining can produce surprisingly efficient code. The loop termination test for the next iteration can be performed immediately after the decision to execute the current iteration. This test can move past all the conditional branches in the body of the loop. In this way, hierarchical reduction exposes many more parallel operations for scheduling.

Hierarchical reduction also minimizes the penalty of short vectors, or loops with small number of iterations. The prolog and epilog of a loop can be overlapped with scalar operations outside the loop; the epilog of a loop can be overlapped with the prolog of the next loop; lastly, software pipelining can be applied even to an outer loop. In summary, hierarchical reduction makes it possible to exploit parallelism in a much larger set of applications. It allows loops containing conditional statements to be software pipelined, and it finds parallelism within loop bodies that are too long to pipeline.

7. Modulo Variable Expansion

If traditional register assignment were performed before code scheduling, then the reuse of registers for different variables would significantly reduce the potential parallelism in the code. This is because the objective of register assignment is to use as few registers as possible. A register is recycled in the shortest amount of time, thus creating many more data dependences that need to be observed. Cooperation is therefore required between code scheduling and register assignment in a superscalar compiler. Proposed strategies include combining register assignment with scheduling [15], and postponing register assignment until after scheduling [18]. The latter approach simplifies the compiler design by separating scheduling and register assignment into two different phases. The drawback, however, is that there may not be enough registers and code needs to be inserted to spill values to memory.

There is one form of register reuse that can greatly inhibit parallelization, and that is the use of the same register for the same variable in different iterations of a loop. To illustrate this point, let us use the same example:

```
FOR i:= 0 TO n DO
   A[i] := A[i]+1.0;
```

For the sake of simplicity, here we assume that a floating-point addition takes only two clocks. The object code for one iteration, complete with register assignment, is as follows.

```
#   R1 preloaded with address of A
#  FR7 preloaded with 1.0

LD   FR1,(R1)
FADD FR1,FR7
nop
ST   FR1,(R1)
ADD  R1,R1,4
```

The register assignment prevents this vectorizable loop from executing in parallel. The register FR1 cannot be loaded with the next input until after its last use in the previous iteration. Similarly, the register R1 cannot be incremented until the last store operation is performed. Anti-dependences force the write operations to follow all the read operations of the old values; consequently, the computation must execute serially.

Modulo variable expansion is a register assignment technique that eliminates these anti-dependences. The following is the result of applying the combination of software pipelining and modulo variable expansion to the example above.

```
      #  R1 preloaded with address of A
      # FR7 preloaded with 1.0

      LD   FR1,(R1)
      FADD FR1,FR7   ADD  R2,R1,4
    1:               LD   FR2,(R2)
      ST   FR1,(R1)  FADD FR2,FR7   ADD  R1,R2,4
                                    LD   FR1,(R1)
                     ST   FR2,(R2)  FADD FR1,FR7   ADD  R2,R1,4  BLoop 1
                                    nop
                                    ST   FR1,(R1)
```

To eliminate the anti-dependence constraint, the second iteration uses a different set of registers, R2 and FR2, and can thus overlap with the first. The third iteration, on the other hand, can reuse the set in the first iteration. In fact, every other iteration can use the same set of registers, making the code identical every two consecutive iterations. The length of the steady state is just twice the initiation interval and the loop body is therefore still very compact.

We call this optimization of assigning several registers to a loop variable modulo variable expansion. In vectorizing compilers, scalar variables are expanded into arrays so that each iteration refers to a different array element, making the loop vectorizable. Modulo variable expansion takes advantage of the flexibility of superscalar machines, and reduces the number of registers allocated to a variable by reusing the same location in non-overlapping iterations.

A tradeoff can be made between the degree of loop unrolling and the number of registers used. For the Warp machine which contains a relatively large number of registers, minimizing the degree of unrolling is a better choice [19]. Eisenbeis et al., on the other hand, minimizes register usage because their target machine, Cray-2, has only eight vector registers [10].

8. Performance of Superscalar Machines

Having functional units that can be explicitly controlled by software, a superscalar processor is more versatile than a vector machine. The parallelism on a vector machine is restricted to the set of vector instructions, and, if chaining is supported, parallelism between vector instructions that use different functional units. Using software pipelining to schedule a superscalar with similar functional units, a simple loop that corresponds to a vector instruction, such as the pairwise additions of two vectors, can execute at the same throughput rate as a vector instruction. In addition, a superscalar can find parallelism in complex loops. Loops do not need to be decomposed into simple vector instructions which require partial expressions be buffered in vector registers. More importantly, a superscalar can find parallelism in loops with recurrences and conditional statements.

The ability of a superscalar machine to execute custom generated parallel code eliminates the need for buffering vectors of partial results. For example, a vectorizing compiler must decompose the loop in Figure 8-1(a) into two, each corresponding to a vector-add instruction (Figure 8-1 (b)). The partial sums must be buffered in a vector register. On a superscalar machine, the partial results can be operated on as soon as they are generated, as illustrated in Figure 8-1(c). This reduces the number of registers needed and possibly memory accesses.

(a)
```
FOR i := 0 TO n DO BEGIN
   c[i] := a[i]+b[i]+c[i];
END;
```

(b)
```
FOR i := 0 TO n DO BEGIN
   t[i] := a[i]+b[i];
END;
FOR i := 0 TO n DO BEGIN
   c[i] := t[i]+c[i];
END;
```

(c)
```
FOR i := 0 TO n DO BEGIN
   t := a[i]+b[i];
   c[i] := t+c[i];
END;
```

Figure 8-1: Reduced register requirement in superscalar machines
(a) source program, (b) vector code, (c) scalar code.

A recurrence does not necessarily mean serial execution for superscalar machines. As long as there are other operations that can execute in parallel with the recurrence computation, a high computation rate can still be obtained using software pipelining. The degree of parallelism in a vectorized loop is of the order of the number of iterations in the loop. A recurrence, however, limits the degree of parallelism by the ratio of independent operations to the length of the cyclic dependence. This limited form of parallelism can be exploited in superscalar processors because of their unique zero synchronization overhead. The compiler strategy for superscalar machines is different from that for vector machines. A vectorizing compiler tries to decompose a loop into smaller loops, separating recurrences from vectorizable loops. A superscalar compiler, on the other hand, tries to jam independent loops together. The vectorizable loop may be executed on the idle functional units while the program computes a recurrence!

In addition to recurrences, hierarchical reduction allows us to find parallelism even in loops with

conditional statements. Hierarchical reduction also reduces the penalty typically associated with short vectors. In a superscalar machine, the scalar operations can be overlapped with the prolog and epilog of a software pipelined loop. This easy integration of scalar and vector operations makes the performance of the system less sensitive to the size of the data. Moreover, software pipelining can be applied even to outer loops, making the advantages of software pipelining applicable even for programs containing short innermost loops.

The instruction scheduling and register assignment techniques have been implemented in the compilers for the Warp and iWarp machines, and have been extensively evaluated [18]. The Warp processor has a peak computation rate of 10 MFLOPS, an impressive performance for a machine built in 1986. This peak computation rate is achieved by a high degree of parallelism and specialization. In a single instruction, a Warp processor can perform one 7-staged floating-point addition, one 7-staged floating-point multiplication, one memory read, one memory write, two integer operations, and one branch operation.

We have analyzed the performance of a set of seventy-two programs and the Livermore kernels. The performance of most of the programs fall between the 1 to 4 MFLOPS range, with a 2.8 MFLOPS average. This utilization of resources is higher than that typically observed in supercomputers. Performance analysis of the software pipeliner shows that the scheduler is successful in exploiting parallelism once the parallelism is detected. About three-quarters of over one hundred loops pipelined are provably optimal. When compared with code generated by a compiler that finds parallelism only within a basic block, most of the loops achieve a speed up of between two and six.

9. Conclusions

This paper presents an overview of compiler optimizations that exploit parallelism in a superscalar machine. High-level loop transformations improve data locality and place parallelism in the innermost loops, in preparation for instruction level optimizations. Software pipelining is the basic technique that finds parallelism across iterations in inner loops. Hierarchical reduction helps deliver a high level of performance for a broader range of applications, for example, by permitting software pipelining to be used even for loops with conditional statements. And lastly, modulo variable expansion eliminates dependence constraints due to reuse of registers between iterations.

The superscalar architecture is a promising alternative to vector machines. We now have compiler techniques that can generate highly efficient parallel code directly from user programs. Given the same hardware functional units, a superscalar machine delivers the same performance of a vector machine if the program is vectorizable. And the superscalar machine is decidably superior to vector machines when the computation contains recurrences and conditional statements. A superscalar does not exhibit a dichotomy in performance depending on whether the code is vectorizable or not.

Compiler optimizations require programs to be analyzable statically. A superscalar architecture has an organization that is more easily enhanced to handle programs that are not amenable to static analysis. By a judicious use of hardware to provide dynamic information to cooperating software, processors that deliver a consistent high-performance through instruction level parallelism are possible.

Acknowledgments

The research on instruction scheduling and register assignment was performed as part of the Warp/iWarp systolic array projects at Carnegie Mellon University. I'd like to thank H. T. Kung, my thesis advisor, for the guidance in this research, and Thomas Gross for his indispensable contribution to the compiler design and construction, and the rest of the Warp/iWarp team for their contribution to the projects. The work on data locality was performed here at Stanford with Michael Wolf. The research is supported in part by DARPA contract N00014-87-K-0828.

References

[1] Allen, R. and Kennedy, K.
Automatic Translation of FORTRAN Programs to Vector Form.
ACM Transactions on Programming Languages and Systems 9(4):491-542, October, 1987.

[2] Anderson, D. W., Sparacio, F. J., and Tomasulo, R. M.
The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling.
IBM Journal of Research and Development :8-24, Jan., 1967.

[3] Annaratone, M., Arnould, E., Gross, T., Kung, H. T., Lam, M., Menzilcioglu, O. and Webb, J. A.
The Warp Computer: Architecture, Implementation and Performance.
IEEE Trans. on Computers C-36(12):1523-1538, Dec., 1987.

[4] Banerjee, U.
Dependence Analysis for Supercomputing.
Kluwer Academic Publishers, 1988.

[5] Bloch, E.
The Engineering Design of the Stretch Computer.
In *Proc. Eastern Joint Computer Conference*, pages 48-58. 1959.

[6] Borkar, S., Cohn, R., Cox, G., Gleason, S., Gross, T., Kung, H. T., Lam, M., Moore, B., Peterson, C., Pieper, J., Rankin, L., Tseng, P. S., Sutton, J., Urbanski, J. and Webb, J.
iWarp: An Integrated Solution to High-Speed Parallel Computing.
In *Proc. Supercomputing '88*, pages 330-339. Nov., 1988.

[7] Cohn, R., Gross, T., Lam, M. and Tseng, P. S.
Architecture and Compiler Tradeoffs for a Long Instruction Word Microprocessor.
In *Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III)*, pages 2-14. April, 1989.

[8] Colwell, R. P., Nix, R. P., O'Donnell, J. J., Papworth, D. B., and Rodman, P. K.
A VLIW Architecture for a Trace Scheduling Compiler.
IEEE Trans. on Computers C-37(8):967-979, Aug., 1988.

[9] Dehnert, J. C., Hsu, P. Y.-T. and Bratt, J. P.
Overlapped Loop Support in the Cydra 5.
In *Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III)*, pages 26-38. April, 1989.

[10] Eisenbeis, C., Jalby, W. and Lichnewsky, A.
Squeezing More CPU Performance out of a Cray-2 by Vector Block Scheduling.
In *Proc. Supercomputing 88.* 1988.

[11] Ferrante, J., Ottenstein, K. and Warren, J.
The Program Dependence Graph and its Use in Optimization.
ACM Trans. on Programming Languages and Systems , July, 1987.

[12] Fisher, J. A.
Trace Scheduling: A Technique for Global Microcode Compaction.
IEEE Transactions on Computers C-30(7):478-490, July, 1981.

[13] Fisher, J. A.
The VLIW Machine: A Multiprocessor for Compiling Scientific Code.
Computer :45-53, July, 1984.

[14] Gannon, D., Jalby, W., and Gallivan, K.
Strategies for Cache and Local Memory Management by Global Program Transformation.
Journal of Parallel and Distributed Computing 5:587-616, 1988.

[15] Goodman, J. R. and Hsu, W. C.
Code Scheduling and Register Allocation in Large Basic Blocks.
In *Proc. 1988 International Conference on Supercomputing*, pages 442-452. July, 1988.

[16] Gross, T. and Lam, M.
Compilation for a High-Performance Systolic Array.
In *Proc. ACM SIGPLAN 86 Symposium on Compiler Construction*, pages 27-38. June, 1986.

[17] Kohn, L. and Fu, S.-W.
A 1,000,000 Transistor Microprocessor.
In *Proc. 1989 International Solid-State Circuits Conference Digest of Technical Papers*, pages 54-55. Feb., 1989.

[18] Lam, M.
A Systolic Array Optimizing Compiler.
PhD thesis, Carnegie Mellon University, May, 1987.

[19] Lam, M.
Software Pipelining: An Effective Scheduling Technique for VLIW Machines.
In *ACM Sigplan '88 Conference on Programming Language Design and Implementation*.. June, 1988.

[20] Patel, J. H. and Davidson, E. S.
Improving the Throughput of a Pipeline by Insertion of Delays.
In *Proc. 3rd Annual Symposium on Computer Architecture*, pages 159-164. Jan., 1976.

[21] Porterfield, A.
Compiler Management of Program Locality.
Technical Report, Rice University, Jan, 1988.

[22] Rau, B. R. and Glaeser, C. D.
Some Scheduling Techniques and an Easily Schedulable Horizontal Architecture for High Performance Scientific Computing.
In *Proc. 14th Annual Workshop on Microprogramming*, pages 183-198. Oct., 1981.

[23] Rau, B. R., Kuekes, P. J. and Glaeser, C. D.
A Statically Scheduled VLSI Interconnect for Parallel Processors.
In *VLSI Systems and Computations*, pages 389-395. October, 1981.

[24] Thornton, J. E.
Parallel Operation in the Control Data 6600.
In *AFIPS Conference Proceedings FJCC*, pages 33-40. 1964.

[25] Touzeau, R. F.
A Fortran Compiler for the FPS-164 Scientific Computer.
In *Proc. ACM SIGPLAN '84 Symp. on Compiler Construction*, pages 48-57. June, 1984.

[26] Wolf, M. E. and Lam, M. S.
An Algorithm to Generate Sequential and Parallel Code with Improved Data Locality.
1989.

[27] Wolfe, M. J.
Optimizing Supercompilers for Supercomputers.
PhD thesis, University of Illinois at Urbana-Champaign, October, 1982.

PROGRES DU LOGICIEL

ADVANCES IN SOFTWARE

PARALLEL PROGRAMMING TRENDS: SPECIFYING AND EXPLOITING PARALLELISM

Constantine D. Polychronopoulos

Center for Supercomputing Research and Development
and Department of Electrical and Computer Engineering
University of Illinois at Urbana–Champaign
Urbana, Illinois 61801

Abstract

Parallel computer architectures and hardware have evolved impressively in the last few years from the architecture and the hardware point of view. Progress on the software side can be best characterized as moderate. The lack of widely acceptable methodologies and software to support parallel programming is profound even on the most advanced parallel machines. Parallel programming is a complex task and the performance of a parallel program can be influenced by many different factors such as coding of parallel constructs and/or restructuring, scheduling schemes and scheduling overhead, synchronization and/or communication cost, program and data partitioning and memory allocation. In this paper we discuss the major aspects of parallel programming. Parallel programming environments are considered in three fundamental phases: *parallelism specification, parallelism exploitation*, and *supporting environments and tools*. A parallel programming environment built at the University of Illinois is discussed as a case study. Finally, we address the influence of parallel programming on multiprocessor operating systems, and discuss future research directions.

This work was supported in part by the National Science Foundation under Grant NSF MIP–8410110, the U.S. Department of Energy under Grant DE–FG02–85ER25001, a Grant from AT&T, and a 1989 NSF–PYI Grant.

PARALLEL PROGRAMMING TRENDS: SPECIFYING AND EXPLOITING PARALLELISM

Constantine D. Polychronopoulos

Center for Supercomputing Research and Development
and Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

1. Introduction

High-speed computers often referred to as supercomputers have invaded such fields as animation and advertising, graphics and industrial computer-aided design, and other business applications, let alone more traditional fields such as weather forecasting, seismic modeling and oil exploration, fluid dynamics, nuclear physics, and other numeric-intensive applications in science and engineering. With more computational power available, scientists can study problems which previously were impossible to model, or increase the size, and thus the accuracy of models for scientific and engineering problems which are already in use.

Traditionally, increased computing power has been achieved through more advanced technologies that allowed higher degrees of integration and switching speeds, more efficient packaging and cooling and so on. Even though there is still room for improvement in switching speeds and integration through the evolution of new technologies, the improvement strides will probably be less significant than in the past, and will soon be limited due to physical barriers. Nevertheless, the demand for more powerful computer systems keeps increasing and even today's most powerful supercomputers are unable to solve certain problems in a reasonable time period. This limitation prompted researchers to investigate architectural approaches to increased computing speeds through new designs and enhancements at the component level. Some of these architectural solutions were more or less radical to the conventional "von Neuman" architecture. An obvious approach and one which has gained much ground, is increasing performance through the replication of conventional processing elements, which work in a coordinated manner and communicate with each other through some type of an interconnection network.

These computer systems, well-known as parallel processors or multiprocessor machines, are the subject of this paper. We focus on their software aspects, and more specifically on the programming issues of parallel computers. A tremendous amount of research and hardware development of such machines has been done from the early days of computing. Many different architectures have been proposed and many of them have culminated with prototypes or commercial systems. Up until the last few years however, the software aspects of parallel machines had been overlooked. This lead to a point where very powerful parallel machines can be built but cannot be used to their fullest potential. Software support for programming these systems was minimal up until recently, and it is still inadequate.

Although this has become apparent and has attracted attention on the fundamental issues of parallel programming, we are still far from understanding the general and global nature of the problem called *parallel programming*. We have only begun to see parallel languages, let alone standards, portability, and programming tools. Of course parallel programming as a relatively new field has not matured yet. Many crucial problems remain unsolved or partially solved. It is true that in parallel programming one faces more complex problems that in serial programming. Another issue that adds more complexity to parallel programming is the variety of architectural models of parallel machines that have not yet been (or cannot be) abstracted from the programming level. For example, our knowledge as to whether our target model is a distributed or shared memory model has profound ramifications even at the algorithm design level (and of course at the program development level). After all, it may not be even possible to reach the same abstraction and portability levels for parallel programming, as the case has been for serial programming. In the former case the primary targets are efficiency and speed which bring parallel programming closer to

the machine level. Compromising speed and efficiency for portability and productivity may not be a desirable solution.

2. Organisation

In this paper we review and discuss many aspects of parallel programming and summarise recent approaches to some of the fundamental issues. Section 3 gives a summary of parallel computer architectures but the rest of the paper focuses on software aspects of parallel programming. Section 4 gives a high-level discussion of parallel programming where the most important issues are identified and discussed. The following sections focus on each particular aspect of parallel programming and present some of the most recent approaches to solving each of them. More attention is placed on parallel languages, parallelising compilers, and scheduling.

In particular, Section 5 highlights the main concurrency features of a number of parallel programming languages and systems. Section 6 discusses parallel programming through automatic program restructuring, and gives examples of the application of a few representative transformations. Section 7 goes into more detail with parallelism exploitation issues such as partitioning, synchronisation, and scheduling. Different approaches to the general scheduling problem are considered, and loop scheduling is examined in more detail.

Section 8 gives an overview of our approach to some of the above issues in the context of the Parafrase-2 project. A novel and comprehensive environment for automating the parallelism exploitation phase is discussed. Section 9 considers multiprocessor operating system issues, and provides a multiprocessor operating system taxonomy based on three important functionalities. Finally, some concluding remarks are given in Section 10.

3. Parallel Computer Architectures

In order to identify the class of computer architectures which are the subject of this discussion, we shall briefly review and classify computer architectures which support some sort of concurrent or parallel processing in the context mentioned above. A typical serial computer is based on the principle of *control flow* where instructions are executed one at a time in a predefined order; each program is viewed as a single instruction stream. A program counter points to the current instruction to be executed. Instruction execution can be accomplished in four stages: instruction decoding, fetching of operands, execution, and storing of results. Since these activities can be functionally independent, a new instruction could be initiated as soon as the current instruction exits the first phase. Thus four instructions can be active in their execution cycle, each passing through a different phase. This technique known as *pipelining* was realised in early computers and provided limited parallelism; asymptotically, four instructions could be executed in the same time it would have taken one instruction to execute without pipelining. The idea of pipelining was not used only in the context of the instruction execution cycle, but also in a number of different activities during program execution.

A straightforward (and oversimplified) extension to the single program counter computer is to extract several independent instruction streams from a single program, and process each of them separately on a different conventional computer. This is the main idea upon which most modern parallel computers are based. A number of stand-alone processing elements are interconnected together (for "occasional" communication) but each processing element remains (in its control structure) a conventional computer. A radically different model of computing machine is the *dataflow* model. The notion of a program counter does not exist in the dataflow model; instead, an instruction is executed as soon as its operands are available [ArNi87]. Although the dataflow principle is particularly attractive, its realisation is less attractive. On a finite size dataflow machine we still have to face problems which are present in the control flow model [GPKK82]. Since more instructions can be ready to fire than execution units available, one needs to decide on some order of execution. This order will have a profound effect on when other instructions become ready to execute. One also needs to decide on which unit a particular instruction should execute. These are only some of the problems which make the realisation of a dataflow system more complex and costly than an equally powerful control flow system.

Table 1. Parallel and Vector/Parallel Computers

Machine	Archit	Memory	Intercon.	No.proc	PeakPe	OS	Languages
Alliant FX series	SIMD+MIMD	shared	bus	8	188 mflops	concentrix*	F+/C+
BBN Butterfly	MIMD	shared	switch net	256	600 mips	crystalis*	F+/C/L+
Convex C-240	SIMD+MIMD	shared	bus	4	320+mflops	unix	F+/C+/A
ELXSI 6400	MIMD	shared	bus	12	156 mips	embos/unix	F/P/C
Encore Multimax	MIMD	shared	bus	20	40 mips	unix	F/C/P
FPS T-series	SIMD+MIMD	local	hypercube	16K	16 mflops/node	N/A	Occam
Intel iPSC/2-VX	SIMD+MIMD	local	hypercube	128	848 mflops	NX	F+/C/L
IP-1	MIMD	shared	crossbar	33	600 mflops	unix	F/C
Meiko M40	MIMD	local	reconfig.	1K	1 gflop	N/A	O/F/P
NCube-10	MIMD	local	hypercube	1024	500 mflops	Vertex/Unix	F/C
Sequent Balance 2100	MIMD	shared	bus	30	21 mips	Unix	F/C/A/L/P
Cray XM-P	SIMD+MIMD	shared	direct	4	1 gflops	cos	F+/C/P
Cray 2	SIMD+MIMD	shared	direct	4	2 gflops	cos	F+/C/P
Denelcor HEP	MIMD	shared	3-neigh. switch	16	160 mips	unix	F+/C/P
ETA-10	SIMD+MIMD	shared	direct	8	gflop-range	own/unix	F+/C/P
IBM 3090/600-VF	SIMD+MIMD	shared	direct	4	430 mflops	VM/XA	F+
NEC SX-3/44	SIMD+MIMD	shared	direct	4	22 gflops	SXOS/unix	F+
CEDAR	SIMD+MIMD	shared	omega	32	3+gflops	xylem	F+/C/L

A: Ada
C: C
F: Fortra..
L: Lisp
P: Pascal

A number of dataflow computers have been built so far with the most recent one being the Sigma–1 of ETL [HiSN84], [TYUY86]. Many of these borrowed principles from the conventional von Neumann architecture but none became a competitive and cost–effective product. The idea of dataflow can be applied more successfully at different levels of program execution if combined with the control flow model. For the rest of this presentation we focus on the more traditional control flow model upon which is based the great majority of existing parallel machines.

There are three basic approaches to parallel processing. In the first scenario, parallelism is exploited by executing many instances of the same instruction(s) on different sets of data. For example, for the multiplication of two vectors of size n we can execute concurrently n multiplication instructions, each operating on a different pair of vector elements. Thus in principle one would multiply two vectors in the same amount of time required to multiply two scalars. This model of parallel computation is often called *Single–Instruction–Multiple–Data* or SIMD and architectures that support this model are termed SIMD architectures. The pipeline model is clearly SIMD. From the control structure point of view, SIMD machines have a single control unit which is responsible for instruction issuing and execution. Common realisations of SIMD computers are vector, pipelined, and array machines. Such computer systems include the STARAN, the Illiac IV, the ICL–DAP, the Goodyear MPP, the Borroughs BSP, the CDC Cyber 205, the Cray–1, the Fujitsu VP–100/200, Hitachi S–810, NEC SX–2, Convex–1, and the Connection machine, just to mention a few.

The second class of parallel machines are those that can process independent instruction streams (each operating on its own data set) simultaneously and are therefore termed *Multiple–Instruction–Multiple–Data* or MIMD systems. MIMD systems are usually composed of a number of independent stand–alone processing elements. It is very rear however to extract multiple instruction streams which are completely independent from real applications. Some control and data information exchange between different streams is usually necessary. To realise this communication, processing elements in a MIMD system need to be able to communicate. For applications where this information exchange is very infrequent, the best processor interconnection scheme is probably a point–to–point connection. Under this scheme each processor operates ordinarily out of its private memory. Processors which need to communicate assemble packets of information called *messages* and they forward them to the requesting processor. This mode of communication is known as *message–passing* and computer systems supporting message–passing are usually designed as *distributed memory* machines. For applications where this information exchange between different processor is very frequent, a better way of communication is through a shared memory where different processors can read and write data to a common memory location. Thus communication is achieved through sharing of memory; a processor writes into a particular memory location and another processor reads from the same location. Parallel machines based on this architecture are known as *shared memory* multiprocessors or parallel computers. Shared memory is the predominant architecture in modern parallel computers and is the underlying architecture model for most of the discussion that follows. Table 1 gives a summary (by no means exhaustive) of shared and distributed memory parallel computers [GeAG88], [Hwan87]. Although some of the software aspects discussed in this paper are particularly suited for shared memory multiprocessors, most of the ideas are applicable to both machine architectures.

4. Parallel Programming Issues

In this section and for the remaining of this paper we will concentrate on the programming aspects of parallel computers ranging from supercomputers to minisupercomputers. As mentioned earlier, parallel computers have evolved impressively in the last few years from the architecture and the hardware point of view. Progress on the software side can be best characterised as moderate.

There are many reasons for the slow spread of parallel programming. Parallel programming is still an art at its infancy, and as such, it lacks standards and software tools for parallel program development. Most programmers are accustom to traditional serial programming. They have in their disposal a plethora of programming languages, editors, compilers, libraries, debuggers and numerous other tools that make programming in any preferred style an easy task. More importantly, many of these tools and environments are standards which make codes portable between a large variety of sequential computers. When it comes to parallel programming, none of the above holds true.

Table 2. PARALLEL PROGRAMMING ISSUES		
SPECIFICATION OF PARALLELISM	**EXPLOITATION OF PARALLELISM**	**SUPPORT ENVIRONMENTS & TOOLS**
• Language constructs for expressing and packaging data/functional parallelism at all granularity levels. • Constructs for synchronisation and communication. • Means for expressing arbitrary nesting, repetitive structures and networks. • Task/process abstraction from architecture details. • Means for avoiding and/or resolving nondeterminancy. • Distributed data structures.	• Load balancing and low run-time overhead simultaneously. • Fast synchronisation/communicat. • Dynamic selection of granularity of parallel tasks within well-defined bounds. • Low overhead process creation and management. • Minimal OS involvement. • Intra/interprocedural dependence analysis and powerful parallelising compilers. • Adaptive operating systems. • Hardware support for synchronisation, context switching, proc. allocation.	• Tools for debugging and tracing nondeterminancy. • Program profiling and static performance analysis. • Graphical interactive user interfaces. • Optimisation and performance data bases. • Numerical stability analysers. • Expert systems.

In an attempt to identify, examine, and propose alternative solutions to the crucial issues of parallel programming, we start by dividing the aspects of parallel programming into three well–defined phases. The first phase is that of *Parallelism Specification*, and includes issues involved in, and means for expressing parallelism in algorithms and programs. The second phase groups together issues involved in the *Exploitation of Parallelism*, with *performance* being the major underlying factor. Finally, the third phase involves *Support Environments and Tools* which provide the means for achieving the goals targeted by the first two phases, in a "user–friendly" and globally efficient manner. Table 2 summarises the goals and issues involved in each case. In the rest of this section we address each of the three phases separately and for each, we examine the underlying goals and the major realisation and performance issues.

SPECIFICATION OF PARALLELISM: The first phase in the development of a "parallel" program involves the selection or design of a suitable parallel algorithm for a particular application and parallel architecture. The problem of parallelism specification starts at this point. An ideal parallel language would provide syntactic constructs which make parallelism specification possible and easy at all granularity levels. Put in other words, the language should provide the means for expressing *maximal* parallelism. By maximal, we mean explicit parallel coding of *all* parallel operations in a given algorithm.

There are two reasons for this: the *complexity* of parallelism qualification, and *portability*. Having the programmer decide whether parallelism at the operation level is more appropriate than at the loop or subroutine level, is not desirable due to the enormous potential complexity of this task (assuming that we refer to a case where parallelism is abundant at all levels). On the other hand, qualifying parallelism during the writing of a program inevitably implies tuning the program for a particular machine. Thus portability (with respect to performance) between different parallel architectures (e.g., a VLIW and a parallel scalar architecture) is difficult to achieve.

A parallel language which supports manual programming for maximal parallelism alleviates both of these problems and shifts the responsibility of *parallelism qualification* to the compiler. There is little doubt that this can be best done by an optimising compiler. In fact, maximally parallel programs would make dependence analysis a much easier task for parallelising compilers. We return to this issue later in this section as well as in the second part of the paper.

Besides syntactic structures for coding parallel constructs, parallel languages must provide means for packaging basic structures into hierarchical parallel structures, for synchronising accesses to shared data or means for communicating data between different parallel tasks. Memory hierarchies in parallel machines give rise to the need for language attributes for classifying data as *private, task–shared*, or *global*, and for dynamic memory allocation of temporary structures.

Parallel languages cannot "mature" before the parallel programming community understands deeper the fundamental principles of parallel processing and builts more experience in programming parallel machines. The present lack of flexible parallel languages can be partly overcome by interactive tools, which compensate for the lack of expressiveness of parallel languages through program annotations, optimising compilers, debuggers, performance and program profilers, etc. As these integrated interactive environments become more powerful we shift to fully automated parallelism specification methods; even the parallelism specification task is taken away from the programmer.

As indicated in Table 2, parallelism specification is only one of the many and complex aspects of parallel programming. This complexity leaves little doubt about the necessity of interactive parallel programming environments, which encompass all of the above mentioned tools and more. The central tool in such an environment would necessarily be a parallelising compiler. Such a compiler could automatically parallelise programs written in a serial language, as well as *verify*, *enhance*, and *qualify* the parallelism of already *parallel* programs. One would then be attempted to ask: why then the need for parallel languages? This is a question without many obvious answers which has drawn considerable debate. Given the present limitations of parallelising compilers, one could argue that manual parallel programming would produce better quality parallel code than such a compiler. An important issue for parallel languages is that of parallel algorithm design. Manual parallel programming will force programmers to "think in parallel" and will expedite research on parallel algorithm design.

In summary, parallelism specification can be done manually by means of a parallel programming language, automatically through a restructuring (parallelising) compiler, or through a combination of the above. Section 5 discusses parallelism specification from the programming languages perspective. Having parallelism specified externally (at the source level) or internally (at the intermediate representation level through a compiler), the next step is the exploitation or mapping of parallelism onto a parallel computer.

EXPLOITATION OF PARALLELISM: The main aspects of parallelism exploitation include qualification of parallelism, packaging or partitioning, and scheduling or resource (processor/memory) allocation. These are only the "abstract" aspects of the parallelism exploitation phase. When parallelism is to be fully exploited (i.e., minimize run-time overheads and balance loads across processors), then one needs to address compiler, operating system, as well as hardware issues as discussed below. A partial set of practical approaches to the parallelism exploitation problem includes [Beck89], [Bokh88], [CGMW88], [Coff76], [Fox87], [Gokh87], [Gupt89], [Jaya88], [KaNa84], [Mann84], [PoKu87].

We assume that parallelism specification is done by means of a programming language which supports maximal expression of parallelism, or by means of automatic program restructuring. Below we attempt to justify why parallelism exploitation can be best done through the compiler and/or the run-time system (although there are hardly any voices against this approach).

Qualification of parallelism (Compiler): Consider a maximally parallel program which is to be compiled for a specific parallel machine. In general, parallelism in such a program can be present at several different levels of granularity. For example, we may have parallel operations within single statements and vector statements, several such statements composing independent basic blocks, independent basic blocks nested inside parallel loops, independent parallel and/or serial loops, independent subroutine calls, and other higher level objects which may potentially execute in parallel.

Likewise, when it comes to real parallel computers, we face three (not necessarily orthogonal) questions:

> 1) Since parallel machines have a limited number of computational elements, and assuming we have "parallelism explosion" in a given program, one needs to decide which parallel constructs will eventually execute in parallel, and which will be ignored (serialised).
>
> 2) Since few machines support parallelism in the hardware for all above cases, some parallel constructs need to be serialised or restructured to other constructs supported by a specific machine (put in other words, parallelism may need to be repackaged). As an example consider a loop which has been coded as a parallel/vector loop in a source language, and is to be executed on a vector computer supporting multidimensional vector statements.
>
> 3) Overhead: Exploiting parallelism in the hardware does not come for free. The familiar pipeline set-up and/or start-up time is overhead paid for vector instructions. Compensation code is the overhead paid for supporting VLIW instructions in a viable manner [Nico84]. Process creation time, queue access and scheduling, and synchronisation-communication time are overheads associated with MIMD parallelism [Poly88]. Due to these overheads parallel execution of certain constructs may result in execution times longer than the corresponding serial ones.

The parallelism qualification phase is responsible for optimizing the code with respect to each of these scenarios.

Packaging or partitioning (Compiler): Partitioning refers to the process of merging or splitting well-defined units of computation into larger or smaller ones respectively. In order to maintain consistency in our terms

we define a *process* to be a unit of computation which is scheduled as a whole and is executed under the control of a single program counter (PC). Hence a process may be serial code, or SIMD code (vector, VLIW instructions etc.). A *task* is composed of one or more processes but it is treated as a unit by the compiler. *In simple terms, partitioning decides whether sets of tasks may potentially execute in parallel.* Merging dependent tasks is appropriate if their parallel execution demands excessive synchronisation or communication between those tasks during execution. Splitting a large task into smaller ones is a way of reducing the granularity and hence increase the exploitable parallelism. Partitioning is discussed in further detail in Section 7.1.

Scheduling (Compiler/Run-time system/OS): Scheduling of the independent processes or tasks of a program on different processors is an activity which may affect execution performance dramatically. Depending on whether scheduling is performed statically (at compile/load-time) or dynamically (at run-time) it may or may not be separated from the partitioning phase. Typically, there are two level of scheduling in a multiprocessor system. Scheduling at the job level where processor resources ed to be allocated fairly among several users to accomplish multiprogramming, and scheduling withing a job in order to distribute independent tasks to different processors to accomplish multiprocessing [Poly89]. Scheduling is the subject of Section 7.3.

Hardware support: The overhead involved in parallel program execution can be reduced through several software and hardware approaches. Hardware support for synchronisation, context switching, queueing, interprocessor communication, and processor allocation is necessary for very high performance parallel computers. As parallel processing overhead keeps decreasing, parallelism exploitation at lower granularity levels becomes effective. Most of the modern parallel computers incorporate specialised hardware for supporting these operations. For instance, Alliant FX/8 uses a control bus for realising fast microtasking; a set of synchronisation registers in the Cray X-MP and Y-MP is used in a similar fashion; a large register file in the Convex C-240 is used for fast context switching and micro/macrotasking [CGMW88]. Hardware support for such special operations as barrier synchronisation may improve the performance of parallel tasks significantly [Alli85], [Beck89], [Gupt89].

SUPPORT ENVIRONMENTS & TOOLS: The issues involved in the parallelism specification and exploitation phases are many and often overwhelmingly complex for any programmer. On the other end it is often impossible or difficult for the compiler to gather all the information necessary to carry out these optimisations. Thus, an interaction between the user and the compiler is mandatory for achieving the best result in general.

User intervention may happen at several different levels. A parallel programming environment must provide the appropriate user interface to allow for easy and effective interaction with the user (Figure 1). Such an interface should provide the tools for representing a program at many different levels, starting from the source level to the dependence graph, and up to the task or subroutine call graph level. User feedback can be in the form of predicates which specify the relationship between atomic structures, or value-ranges for variables, in the form of task qualifiers etc.

In addition, a programming environment must support parallel program debugging and tuning. Nondeterminism and other race conditions in a multiprocessor make parallel program debugging a particularly critical issue. Post-mordem analysis and performance profilers are also needed to allow further tuning for enhancing performance. Numerical stability analysers can be useful in determining the effect of program restructuring on the stability of computations [BrGa89]. Static timing analysis of a parallel program is necessary to guide many phases of a parallelising compiler [Cytr84], [Poly88], [SBDN87]. For example, the parallelism qualification phase is based on compile-time estimates of code and/or data structure size, execution time estimates, etc.

Programming environments will allow a user to display segments of a program in different representations ranging from the code itself to dependence and control flow graphs to task graphs. In addition, a profiler can give information as to where a program spends most of its time, or detailed timing profile for an entire application. A library of different synchronisation schemes provides synchronisation alternatives

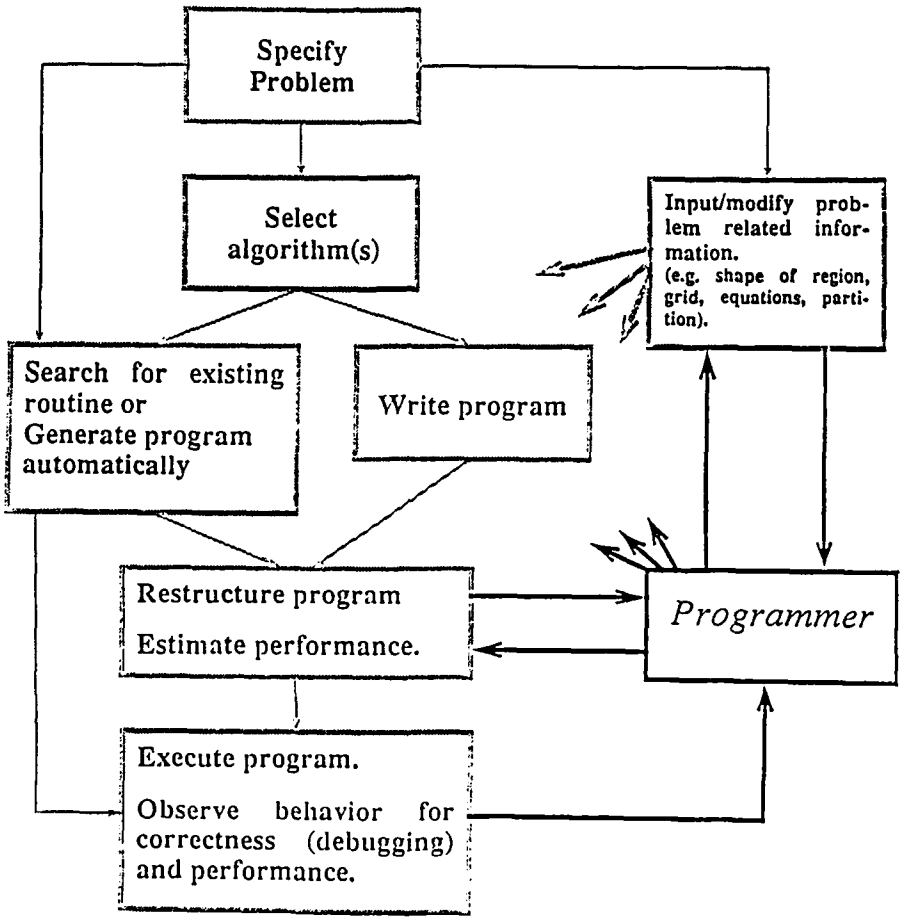


Figure 1. A parallel programming environment.

for different circumstances. The most appropriate type of synchronisation for each circumstance can then be chosen by the user of automatically by the compiler. A number of other libraries, data bases, and other program manipulation tools complete the picture of a parallel programming environment which lies on top of a powerful parallelising compiler. It will not be long before such programming environments become common.

Figure 1 shows the different phases of a complete programming environment. The same diagram can be viewed as a parallel program development and execution cycle. Each of the boxes in the figure correspond to a number of tools that perform a specific function. For example, the "program restructuring" box includes tools such as a parallelising compiler, static performance analyser, program data-base for interprocedural analysis, code generation, and possibly profile history, graphical interfaces for displaying program representations, etc. All these tools must be integrated and interfaced in a convenient manner to provide for maximum user productivity and program performance, which is the end-goal. An expert system may be built on top of such an environment to provide user support as well as to guide several program optimisations such as multi-version code generation [WaGa89], [Wolf89]. It is worth noting that at present, there exists no programming environment that supports automation of even a small subset of the functions discussed above.

5. Parallel Languages and Systems

Few programming languages provide adequate support for parallel programming. Parallel language design is by and large at an experimental stage. One of the earliest and most influential formalisms toward a parallel programming language was the CSP proposed by Hoare [Hoar78]. CSP is the foundation upon which Occam was built for transputer based networks, and it influenced the design of other languages and systems as well (e.g., Linda, [ShCG86]).

At present, manual parallel programming is accomplished in three different ways: through the use of "parallel" languages, through program annotations, or through a combination of language constructs, program annotations, and tools that work between the compiler and the run-time or the operating system, and provide virtual processors that can be user controlled. Early multiprocessors provided parallel programming support via annotations in the form of comment cards or function calls. These annotations tap to the run-time library and eventually to the operating system to provide the means for creating multiple tasks, allocating memory for them, and scheduling them on the computing elements of a multiprocessor. The first version of the Cray multitasking library worked in a similar way [Cray85].

Following similar approaches most multiprocessor vendors provided extended run-time systems with parallel processing support, typically referred to as multi/macro/micro-tasking libraries. Run-time systems, however, do not provide a portable environment for parallel programming – they are tailored to a particular machine. Portability was the major force behind the development of the Schedule package at Argonne, which allows the user to explicitly specify data dependences, partitioning, and allocation of tasks to virtual processors supported by that environment [DoSo87].

A combination of language extensions and run-time systems is being currently used by most super and superminicomputer vendors, and many more have been developed at universities and other research labs. An example of such a system is the Linda environment [AhCG86]. At different times Linda has been suggested as a language, as a run-time support system, and even as an environment for parallel programming. Based on CSP principles, Linda provides convenient (but not necessarily efficient) constructs for sharing and communicating data between different processors. Like Linda the great majority of these systems provide a convenient and portable environment for parallel program development, but they are, nevertheless, quite restrictive when it comes to types of parallel constructs supported, scheduling, synchronisation, and above all efficiency and performance. Furthermore, none of these systems provides *automatic* solution to the crucial issues of parallel programming mentioned in the previous section. They do however, represent the state-of-the-art in parallel programming support environments.

In the rest of this section, we shall review the most interesting extensions to widely used languages for supporting parallelism. Language extensions have been proposed for a variety of existing serial languages, both functional and imperative. User enthusiasm has varied and as it would have been expected

Table 3. Parallel Lisp Systems.

Institution	Computer System	Type of System	Language Specific Features	Source Language	Type of Parallelism	References
BBN Inc.	Butterfly	Shared Memory Parallel Processor	No	Butterfly Lisp Multischeme	Explicit	[AlSr89]
Electrotechnical Laboratory	EM-3	Dataflow	Yes	EMLISP	Implicit	[TYUY86]
Gold Hill Computers, Inc.	Intel iPSC	Non-shared Memory Parallel Processor	No	CCLISP	Explicit	
Electrical Communications Labs., NTT. Japan	DFM	Dataflow	Yes	Valid	Implicit	[ATHM86]
M.I.T.	Concert	Shared Memory	No	Multiplisp	Explicit	[Hals86]
Thinking Machines Corp.	Connection Machine	SIMD	No	CmLISP	Explicit	[Hill85]
University of California Berkeley	SPUR	Shared Memory Parallel Processor	Yes	Extended LISP	Explicit/Implicit	[Patt87]
University of Illinois at Urbana-Champaign	Alliant FX/8	Shared Memory Parallel Processor	No	Scheme	Implicit	[Harr86]
University of Kyoto	M68000-based Parallel Processor	Shared Memory Parallel Processor	Yes	C-LISP	Explicit	[SATO83]
University of Mexico	AHR	Dataflow	Yes	Pure LISP	Implicit	[Guzm88]

Table 4. Parallel Prolog Systems.

Institution	Computer System	Type of System	Language Specific Features	Source Language	Type of Parallelism	References
Gigalips Project (Argonne Natl. Lab., Manchester U., and Swedish Institute of C.S.)	Denelcor HEP Sequent Balance Encore Multimax	Shared Memory Parallel Processor	Yes	Standard Prolog	OR	[Warr87]
Fujitsu Labs.	KABU-WAKE	Non-shared memory Parallel Processor	Yes		OR	[SSKM86]
IBM Laboratory Boeblingen, Germany	AIK-0	Shared Memory Parallel Processor	Yes		AND/OR	[Diel86]
ICOT Research Center	PIM-D	Dataflow Machine	Yes	KL1/GHC	AND/OR	[IKKR86]
Imperial College	Alice	Reduction Machine	No	Parlog	AND/OR	[ClGr88]
University of California at Berkeley	Aquarius	Shared Memory Parallel Processor	Yes	Standard Prolog	AND	[DePa85]
University of California at Berkeley	Sun 3/50 workstations	Distributed System	Yes	Extended Prolog	AND	[CaRo88]
University of Illinois at Urbana-Champaign	Alliant FX/8	Shared Memory Parallel Processor	No	Standard Prolog	OR	[KaPS88]
University of Tokyo	PIE	Cluster-Shared Memory	Yes		OR	[Tana86]
Weismann Institute of Science	Intel iPSC	Non-shared Memory Parallel Processor	No	Flat Concurrent Prolog	AND	[Shap86]

it depends on the popularity of the base serial language in the first place. Table 3 gives a summary of existing parallel Lisp implementations for a variety of parallel architectures, including dataflow machines [PaHK88]. Perhaps the most well-known Lisp dialect is Multilisp (based on Scheme) which supports parallelism through the *future* construct [Hals86].

At the language level, futures provide the means for decoupling references to a variable from the evaluation of that variable or structure. A future associated with a variable can be referenced before that variable has been evaluated; in that case the execution of the object making the reference is blocked until the evaluation is complete. Thus futures provide a vehicle for synchronising accesses to atomic and compound structures. Although futures provide a flexible and powerful abstraction, when it comes to performance, the real issue lies in the implementation and efficiency of this abstraction.

Similar language extensions have been proposed for Prolog, with Concurrent Prolog being the most well-known parallel dialect [Shap86]. Table 4 gives a summary of parallel prolog systems currently in use [PaHK88]. In the next few sections we look in more detail at parallel Fortran languages, Fortran being by and large the predominant language for programming high performance multiprocessor systems. Occam is reviewed next as the representative of languages which were developed originally as parallel programming languages.

5.1. Parallel Fortran Dialects

It is not random that most of the "parallel" languages in use today are extensions and enhancements to Fortran. Almost all supercomputer vendors supply an enhanced (vector and/or parallel) Fortran version with their machines. In this section we overview the most important features of various Fortran dialects.

Fortran 8X: The only notable Fortran 8X features for "parallel" programming are the array operations and statements [Lawr75], [Paul82]. Nevertheless, Fortran 8X is evolving in the right direction as a more flexible and general purpose language by including facilities for complex user-defined data types, recursion, more control constructs etc [ANSI86]. The new standard is a superset of Fortran 77 even though several old and redundant features have become candidates for elimination in the next standard. In addition to five types of intrinsic literal constants and scalar variables, 8X provides facilities for user-defined data types. General type declarations have the following format:

```
[access] TYPE type-name [(type-param-name-list)]
                        type-specification
END TYPE [type-name]
```

where *access* can be PUBLIC or PRIVATE and *type-specification* is a sequence of intrinsic and/or other type declarations. An example of a type declaration is shown below.

```
TYPE date
     CHARACTER(LEN=7) day
     CHARACTER(LEN=10) month
     INTEGER year
END TYPE date
```

The declaration and use of complex data objects in Fortran 8X resembles that of records in Pascal. The language provides for array declarations of fixed and variable sizes. For instance the statement

```
REAL, ARRAY(-2:5, 10) :: A
```

declares A to be a two dimensional array (rank two) of reals with eight rows and 10 columns. Vector constants can be specified as lists of elements enclosed in square brackets. The sequence [5,5,5,7,8,8] is a vector constant of size seven and rank one.

In terms of parallel constructs there is not much more than array expressions and assignment statements. Array expressions and assignments are allowed between conformable arrays of compatible types. If A, B, C, and V are declared as follows,

```
REAL, ARRAY(10,10) :: A,B,C
REAL, ARRAY(5) :: V
```

then C = A+B defines the elements of C to be the sum of the corresponding elements of A and B. The correspondence is by position in the extend (dimension). Thus, the same array assignment statement can be rewritten as C(1:10) = A(1:10)+B(1:10). Similarly, A = A+5 increments all elements of A by five. The statement A(1,1:5) = V results in overwriting the first five elements of the first row of A by the corresponding elements of vector V.

Finally, modules in 8X along with the PUBLIC and PRIVATE attributes on type declarations allow for packaging and restricted use of data objects at different scopes of a program. Although Fortran 8X has not been frozen at present, it is fair to say that the language is evolving as a more general purpose language by providing the programmer with facilities for data abstraction, more efficient memory allocation, structured programming and more powerful control constructs. Nevertheless, it provides minimal support for concurrent programming.

CEDAR and Cray Fortran: The Cedar Fortran was designed based on the 8X, Alliant, and Cray Fortran extensions, and includes a number of additional extensions for multiprocessing support on the Cedar machine [GPHL88], [KDLS86]. Given the clustered organisation of the Cedar multiprocessor, and the desire to provide more control to the user, Cedar Fortran supports two types of concurrent loops: CDOALL for intra-cluster concurrency, and SDOALL for inter-cluster concurrency. Consider the following doubly nested parallel loop.

```
GLOBAL A(10,20), I
SDOALL I = 1, 10
      INTEGER J
LOOP
      CDOALL J = 1, 20
           A(I,J) = I+J
      ENDCDOALL
ENDSDOALL
```

Each iteration of the outer loop is assigned to a different processor cluster of the machine. Within each iteration of the SDOALL we have another concurrent loop which is executed within the cluster owning its current I value. Since A is written by different clusters, it is allocated in global memory. In addition to GLOBAL Cedar Fortran provides the CLUSTER memory attribute, for structures allocatable to cluster memory. In the above example, integer J is cluster-private. Notice that since vector instructions are supported by the Cedar (Alliant) processors, the machine supports up to three levels of parallelism.

Cedar locks and events are identical to those of Cray Fortran. The same is true for macrotasking primitives. A new task can be spawn by executing the following statement: `int = ctskstart(num_proc, sub [,arg]...)`, where sub is the subroutine name containing the task with an optional list of arguments, and num_proc is the number of processors requested for that task. `logical = ctskdone (int)` returns true/false depending on whether task `int` has completed or not. A join or barrier synchronisation can be accomplished through `call ctskwait (int)` which suspends execution of the calling routine until task `int` completes execution.

Both the Cray and Cedar dialects include extensive support for macro and microtasking, synchronisation, and dynamic memory allocation. Most of the Cedar Fortran facilities resemble closely those of Cray's. Both languages support multitasking through the run-time system as opposed to language extensions. Since the multitasking mechanisms are essentially the same as those of the more recent IBM Fortran, and since the latter provides true language extensions for tasking, we have chosen not to discuss tasking in the Cedar and Cray Fortran, in favor of more detailed review of the IBM Fortran.

IBM Fortran: Recently, IBM released the first version of its Parallel Fortran for the MVS/XA and the VM/XA SP operating systems. The IBM Parallel Fortran is probably the most "loaded" Fortran dialect for multiprocessing so far [IBM88]. It provides both direct language extensions for parallel processing as

well as a set of useful library routines and compiler directives. The combination of powerful language extensions along with (limited) automatic parallelisation in the compiler gives the IBM Fortran a distinct advantage over many other Fortran dialects. For parallel task execution the IBM run-time library follows closely Cray's macro and microtasking, but more facilities for tasking are provided at the language level in this case.

In the IBM Fortran, a *task* is a complete environment with its own local space and code. Tasks can also share space with other tasks. Tasks can be explicitly created and manipulated by the programmer. Tasking in the IBM Fortran is implemented in two stages. In the first stage, tasks are explicitly created but they are not executable. We often refer to originated (but not allocated) tasks as *virtual* or simply tasks. In the next phase these virtual tasks are allocated work (parts of user code), again explicitly through facilities provided by the language. We call this the *binding* phase and we refer to work allocated to virtual tasks as *user* or *real tasks* or simply tasks whenever the context is clear.

In addition to explicit tasking, *implicit tasks* created automatically by the run-time library are used to execute explicitly coded parallel loops. In the rest of this section we concentrate on language rather than library and compiler features. As mentioned above, the user creates a number of tasks at the beginning of the program and performs the binding of real to virtual tasks. In addition to binding, synchronization, load balancing, and in general the entire tasking process is under the direct control of the programmer. But let us review the most interesting features of the language.

First, we consider parallel loops. Vector processing was supported by the previous IBM Fortran version for the 3090 series, and the syntax of vector statements is identical to that of Fortran 8X. The new Fortran extensions include a parallel loop construct whose syntax is as follows:

```
PARALLEL LOOP label [,] index=l, u [, str]
       [PRIVATE (var [, var] ...)]
       [DOFIRST [LOCK]
              dofirst-block]
       [DOEVERY
              doevery-block]
       [DOFINAL [LOCK]
              dofinal-block]
label CONTINUE
```

The PARALLEL LOOP header initiates a loop which is to be executed by many implicit tasks. Depending on the loop type and size the implicit tasks may be as many as the number of iterations of the loop (thus different iterations may be executing simultaneously and independently), or many iterations may be assigned to a single implicit task. Variables which are private with respect to each loop iteration may be declared as such within the loop, following the PRIVATE clause.

The body of a parallel loop consists of a prologue which starts after the DOFIRST clause (*dofirst-block*), the main loop body which starts after the DOEVERY statement (*doevery-block*), and an epilogue (*dofinal-block*) which starts after the DOFINAL statement. The DOFIRST and DOFINAL blocks are executed once by each implicit task participating in the execution of the loop. The DOEVERY block contains the code which is ' be executed by each iteration of the loop. The DOFIRST and DOFINAL blocks can be executed exc' ely (by the implicit tasks assigned to the loop) if the LOCK attribute is specified. Each task obtains the lock before entering the corresponding block, and releases the lock upon exit from that block. An example of a parallel loop performing a reduction follows.

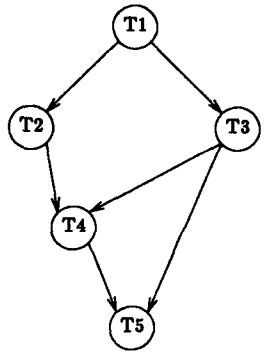
```
sum = 0
PARALLEL LOOP 10 i = 1,N
       PRIVATE(s)
       DOFIRST
           s = 0
       DOEVERY
           DO 20 j =1,M
             s = s+a(i,j)
```

```
20              CONTINUE
              DOFINAL LOCK
                sum = sum+s
10    CONTINUE
```

Another important feature of the IBM Fortran is the parallel CASE statement which can be used in much the same way as cobegin–coend to achieve parallel execution of independent computations. The syntax of the PARALLEL CASES statement is shown below.

```
PARALLEL CASES
      [PRIVATE(var [,var]...)]
      {CASE [m [, WAITING FOR {CASE n | CASES(n1 [,n2]...)}]]
         case-block}...
END CASES
```

All different *case-blocks* are evaluated simultaneously by different implicit tasks. If the WAITING FOR attribute is specified for a CASE, that *case-block* will not execute before all cases referenced in that WAITING FOR have completed execution. Therefore, one may use the cases statement to schedule acyclic task graphs for parallel execution. Consider for example the following task graph



This graph may be coded in Fortran using the parallel case statement as follows:

```
PARALLEL CASES
      CASE 1
         T1;
      CASE 2, WAITING FOR CASE 1
         T2;
      CASE 3, WAITING FOR CASE 1
         T3;
      CASE 4, WAITING FOR CASES (2,3)
         T4;
      CASE 5, WAITING FOR CASES (3,4)
         T5;
END CASES
```

Explicit task definition and manipulation is supported by direct language extensions. Any number of tasks may be started anywhere in the program by using the ORIGINATE statement,

```
ORIGINATE ANY TASK takid
```

where *takid* is the identifier of the task initiated by the statement. A task created by the ORIGINATE statement is owned by the task from which the statement was executed. The following loop,

```
      DO 10 i = 1,10
         ORIGINATE ANY TASK tskid(i)
10    CONTINUE
```

initiates 10 tasks which can be referenced inside the same scope by their identifier *tskid(*)*. Upon initiation, tasks are not assigned specific work unless an explicit SCHEDULE statement is executed. The SCHEDULE statement binds a real task (user–defined) to an already originated task.

There are two versions of the SCHEDULE command. The first is the SCHEDULE TASK command with the following syntax:

```
SCHEDULE TASK tskid,
    [TAGGING(tag1 [,tag2]...),]
    [SHARING(shrcom [,shrcom]...),]
    [COPYING(cpcom [,cpcom]...),]
    [COPYINGI(cpicom [,cpicom]...),]
    [COPYINGO(cpocom [,cpocom]...),]
    CALLING subx[([arg1 [,arg2]...])]
```

The second is the SCHEDULE ANY TASK whose syntax (besides the clause ANY) is identical to that of the SCHEDULE TASK. Let us use the abbreviations ST and SAT for the two statements. In ST, *tskid* specifies the identifier of a currently unused task which will execute the scheduled subroutine (or user task). In contrast, SAT *returns* this identifier in *tskid*.

The argument(s) in the TAGGING attribute is a scalar value which is used to tag the piece of work which is being scheduled. These tags can be used to identify tasks and determine their status (i.e, completed, executing). The arguments in the SHARING clause are the names of COMMON blocks which are shared with the scheduled task. The COPYING part of the command takes as argument the name of a COMMON block in the environment of the scheduling task. The contents of that block are copied into a COMMON block of the same name but which is created in the scheduled task's environment. Upon completion of the scheduled task, the contents of the latter block are copied back into the former. COPYINGI is as above but *no copying back is performed upon completion* of the task. The argument(s) in COPYINGO is the name of a common block in the environment of the *scheduled* task. The contents of this block are copied back to a COMMON block of the same name in the environment of the *scheduling* task after the completion of the scheduled task.

Finally, the CALLING part of the statement specifies the name of the subroutine to be scheduled for execution (*subx*). Hence, unlike implicit tasks in the CASE statement, tasks which are explicitly scheduled with the SCHEDULE command must be organised as subroutines. Another interesting feature of the SCHEDULE command is that it allows the programmer to perform manual binding of program tasks to virtual tasks (through the ST version of the command), but it also allows more dynamic binding through the SAT version. In both cases, however, virtual tasks must be explicitly originated before the binding.

Synchronisation of SCHEDULEd tasks can be achieved through the use of the WAIT FOR statement. Every SCHEDULE statement in a program must have a corresponding WAIT FOR statement. The latter forces the issuing task to wait until the corresponding scheduled task completes. There are three versions of the WAIT FOR statement. The first,

```
WAIT FOR TASK tskid [,TAGGIGN(tag1 [,tag2]...)]
```

blocks execution of the issuing task until the task with identifier *tskid* completes. The second version of the command, WAIT FOR ANY TASK, with identical syntax as the first, blocks execution until *any* task issued by the scheduling task completes execution. Hence, if only four tasks with identifiers TSKID(1:4) are SCHEDULEd from a given subprogram, then the sequence

```
WAIT FOR TASK TSKID(1)
WAIT FOR TASK TSKID(2)
WAIT FOR TASK TSKID(3)
WAIT FOR TASK TSKID(4)
```

System	Microtasking				Data	Macrotasking			
	doall	doacross	high-level spreading	automatic parallelization	shared/private common	spawn	wait	locks	events
Alliant	no	yes	no	yes	no	no	no	no	no
BBN Butterfly – future plans	Doparallel,unlocked Do ... Continue	Doparallel,locked Do ... Begin Ordered End Ordered	Begin Parallel End Section	no	Shared Common Common	no	no	yes	no
Cray	yes	no	yes	no	Common Task Common	yes	yes	yes	yes
Encore Parallel Fortran	yes	doall send/wait end doall	Parallel End Parallel	yes	Shared Private Volitile	no	wait and barrier	yes	yes
IBM Parallel Fortran PRPQ – future plans	Parallel loop Continue	no	Parallel Case End Case	yes	yes	yes	yes	unknown	unknown
Sequent	no	yes	no	yes	shared by compiler option	yes	yes	yes	no

Table 5. Parallel Fortran Dialects and Systems

will block further execution until all four tasks complete and so does the sequence

```
WAIT FOR ANY TASK tskid
WAIT FOR ANY TASK tskid
WAIT FOR ANY TASK tskid
WAIT FOR ANY TASK tskid
```

The third version is the WAIT FOR ALL TASKS statement. If we substitute the SCHEDULE clause in the above statement with the DISPATCH clause we get another statement with identical syntax and similar semantics. The only difference is that a task which has been assinged work through a DISPATCH can be reassigned other work by another DISPATCH without using a corresponding WAIT FOR statement. However, that task cannot be reassinged work through a SCHEDULE statement unless a WAIT FOR ALL TASKS has been issued. The latter is the only means of synchronising dispatched work. Continuing the previous example, a single WAIT FOR ALL TASKS statement would also be equivalent to the two WAIT FOR sequences above.

In addition to these language extensions, the IBM Parallel Fortran provides also a rich repertoire or intrinsics and library routines for parallel locks and events as well as a set of compiler directives which facilitate manual and semi-automatic parallel programming. With the exception of parallel loop handling, the IBM Fortran provides the most complete extensions for tasking compared to other parallel Fortran dialects. A summary of other vector and parallel Fortran dialects appears on Table 5 [GPHL88].

Occam: As a representative of a language originally designed as parallel we selected Occam, a language which has been specifically designed (based on CSP) for distributed memory computers. The fact that Occam is one of the first attempts to design a genuine message-based language makes it quite interesting. Despite the almost enthusiastic acceptance of Occam by the transputer community, the language suffers (from) many drawbacks. Before we comment on the advantages and the peculiarities of Occam let us examine the basic facilities.

Unlike the languages that have been examined so far, the notion of global or shared variables and structures in Occam does not exist. Rather, data exchange between different parts of a program occurs through explicitly defined channels and is completely asynchronous. Program modules are organized in processes which communicate through user-defined channels. Communication is achieved with messages which are assembled by a transmitting process and are forwarded through channels to a receiving process. Thus, variables are always local to each process.

Occam programs can be written using three primitive processes and a number of constructs which provide the means for grouping primitives into more complex program units. The three primitives are assignment, input, and output statements. An assignment statement has the general form `v := expr` and assigns the value of expression `expr` to variable `v`. I/O is always accomplished in Occam through channels. Channels can be declared in a program just like any other variable using the attribute CHAN. As discussed later (virtual or program) channels can be mapped directly to hardwired channels of a particular transputer configuration. If `chan1` and `chan2` are variables of type CHAN, then the primitive process `chan1 ! expr` outputs the value of expression `expr` to channel `chan1`. An input process of the form `chan2 ? v` inputs the value of variable `v` from channel `chan2`.

Complex nested processes can be formed by using a number of constructs provided by the language. Any well defined program unit (starting from the primitives) is a process with is own instruction stream and data. It is convenient to represent Occam programs using directed graphs with nodes representing processes and arcs corresponding to program defined channels. Since exchange of data and other communication can occur only through channels, two or more Occam processes may communicate only if they are simultaneously active (executing). If a process tries to communicate with another process whose execution has not started or which is blocked, then the first process also becomes blocked. Thus, it is not hard to specify a set of processes which can lead to a deadlock. Another peculiar feature of Occam is that primitive processes must be written in a single line. For example, breaking a lengthy assignment process into two lines my produce incorrect results. Some ad hoc rules can be used to avoid such cases but these rules are not enforced by most Occam implementations.

In Occam indentation defines the scope of objects and constructs. Variables are local to the process immediately following their declaration and at the same indentation level. The scope of constructs includes all processes which are indented two spaces to the right of the construct specifier. Unlike most other languages, lexical ordering does not relate to flow of control order. Occam provides a number of flow of control constructs which are discussed below. The construct SEQ can be used to specify sequential execution order for all processes following at the same indentation level. For example,

```
SEQ
  Q
  S
  W
```

amounts to executing process Q before S which is executed before W. The inverse result is obtained by using the construct PAR in place of SEQ. All processes within the scope of a PAR and at the same indentation level can execute in parallel. Two other control constructs include the IF and ALT processes. An IF process comprises a number of processes each preceded by a boolean expression. In this case evaluation of the boolean expressions is performed in lexicographic order; the process following the first boolean expression to evaluate to true is the one which is executed. The ALT construct can be thought of as an IF process whose branches are evaluated simultaneously, and the first branch to satisfy the test condition is the one taken. However, in ALT each component process is preceded by an input process, or by a boolean expression followed by an input process. The first input process that completes successfully is the one whose branch is taken. These input processes act like *guard* inputs. Consider the following process.

```
CHAN chan1, chan2:
INT v:
ALT
  chan1 ? v
    IF
      v>0
        v:=0
      TRUE
        SKIP
  chan2 ? v
    v:=0
```

Variables chan1 and chan2 are declared to be of type channel and v of type integer (all three are local to the process following at the same indentation level, i.e., ALT). In the input guards in the above fragment input a value from chan1 or chan2 to variable v. If the value comes from chan1 then the process following that guard is executed. In that case, if v is positive it becomes 0 otherwise it remains unaltered. SKIP is a special empty process. If input is performed through channel chan2 then v becomes also 0. If neither of the input guards completes then the process is deadlocked. If both guards complete simultaneously one path is chosen arbitrarily. To avoid this type of nondeterminism a prioritized alternative process can be specified as PRI ALT. In that case priority is given to the process which is lexically first.

Occam allows to specify replicated constructs using the above processes. The general syntax of a replicated process is

```
<header> index=base FOR count
    process-body
```

where <header> can be any of the SEQ, PAR, IF, or ALT constructs, *process-body* is a sequence of processes for SEQ and PAR, a sequence of processes each preceded by a boolean for IF, or a sequence of processes each preceded by an input guard or by a boolean and an input guard for ALT. The effect of a replicator is to spawn a number of <header> processes indicated by *count*. Each process can be referenced by its *index* value. Consider the following example.

```
[10][10] INT a, b, c :
```

```
INT i :
PAR i=1 FOR 10
  INT j :
  SEQ j=1 FOR 10
    INT s, k :
    s:=0
    SEQ k=1 FOR 10
      s := s + a[i][k] * b[k][j]
    c[i][j] := s
```

The above Occam program performs the multiplication of two matrices, a and b and stores the result in c. The replicated PAR spawns 10 independent processes, each computing a vector times matrix product. Thus the replicated PAR construct serves as a parallel loop in this case. Similarly, the two inner replicated SEQs function as serial loops (even though the middle SEQ could be replaced by a replicated PAR).

This example brings up many "loose ends" of Occam. The semantics of the language is not clear in many cases and it is probably left up to each implementation to decide the exact semantics of the language. First, let us consider the replicated PAR. According to the definition parallel Occam processes can execute simultaneously on different processors, say transputer nodes. On the other hand, processes executing in parallel can communicate only through channels defined by the programmer. If in the above example we assume that all three matrices are allocated to the local memory of one processor, then each of the ten processes should send the locally computed row of c to the appropriate processor through a user-defined channel. In the example above this does not happen. By using an assignment statement to store the computed elements of c, we imply that the ten parallel processes created by the replicated PAR are to execute on the same node, and thus are effectively serialised. On the other hand, a smart compiler could detect this case and translate the assignment into a distributed assignment statement which completes from two ends of a common channel. This could be done for example as

```
channel ? s
channel ! c[i][j]
```

where channel is one of ten user-defined channels. Thus, even the low level details of process formation and processor allocation is left to the discretion of the user. This puts an unreasonable demand on the average user and it is in complete disharmony with one of the main goals of parallel programming, i.e., to obscure low level details that demand high level of expertise from the user. To the best of our knowledge this problem is not eased by existing Occam compilers; more needs to be done for developing optimising Occam compilers and restructurers.

6. Parallelising Compilers

Automatic program restructuring and optimisation for SIMD parallelism is an extensively researched subject which was pioneered by Kuck and his colleagues at the University of Illinois, Kennedy and his associates at Rice University, and Allen and her colleagues at IBM [ABCC87], [AlKe82], [AlKe87], [Bane88], [KKLW80], [KKPL81], [PaWo86]. Many other researchers at several institutions have made very significant contributions on the subject as well. Many of these ideas are directly applicable to restructuring for MIMD parallelism although this later subject is far from being well-established. Restructuring for SIMD parallelism is more commonly referred to as *vectorization*. Restructuring for MIMD parallelism is known as *parallelization* or *concurrentization* with the former being the term of our choice.

Parallelising compilers transform not only serial programs, but they can further parallelise programs written in a parallel language. Moreover, such compilers can be used to partially validate the user-specified parallelism in an already parallel program. This section addresses the most important issues involved in automatic parallelisation, and discusses viable solutions. Since it is not possible to give a thorough introduction to this subject in this paper, we have chosen to discuss representative transformations as examples [ChCi87], [Cytr84], [GiPo88], [Nico84], [Kuck78], [PaWo86], [Wolf82]. A more complete discussion on program transformations can be found in [AlKe87], [PaWo86], [Poly88], [Wolf82].

6.1. Computing Data Dependences

Automatic parallelisation is based on data and control flow information that the compiler gathers by analysing the program. Obviously, the more accurate the information gathered by the compiler, the more aggressive the parallelisation. During dependence analysis, the compiler examines the flow of data through a program to determine execution orderings which need to be obeyed for not violating the original semantics of the program. Before we make a brief introduction to data dependences we need to establish the necessary notation.

A *program* is a list of $k \in N$ statements. S_i denotes the i–th statement in a program (counting in lexicographic order). I_j denotes a DO loop index, and i_j a particular value of I_j. N_j is the upper bound of a loop index I_j, and all loops are assumed to be normalised, i.e., the values of an index I_j range between 1 and N_j. We have two types of statements, *scalar* and *indexed* statements. An indexed statement is one that appears inside a loop, or whose execution is explicitly or implicitly controlled by an index (e.g., vector statements). All other statements are scalar. The *degree* of a program statement is the number of distinct loops surrounding it, or the number of dimensions of its operands. $S_i(I_1,I_2,...,I_k)$ denotes a statement of degree k, where I_j is the index of the j–th loop or dimension. An indexed statement $S_i(I_1,...,I_k)$ has $\prod_{j=1}^{k} N_j$ different *instances*, one for each value of each of I_j,(j=1,...,k). S_i will be used in place of $S_i(I_1,...,I_k)$ whenever the set of indices is obvious. We say that statement S_i precedes S_j in the order of execution, and denote it by $S_i < S_j$, if under the serial control flow S_i is executed before S_j.

Two statements S_i and S_j are said to be involved in a *flow* dependence $S_i \delta S_j$ if and only if S_i precedes S_j in the serial execution order, and a variable written by S_i is read by S_j. An *antidependence* between S_i and S_j is defined as the flow dependence above, except that in this case, a variable read by S_i is written by S_j; an antidependence is denoted by $S_i \bar{\delta} S_j$. An *output* dependence is again defined as above but with S_i and S_j writing to the same variable, and is denoted by $S_i \delta^o S_j$. In all three cases S_i is called the dependence *source* and S_j is the dependence *sink*. For each data dependence involving statements $S_i(i_1,...,i_k)$ and $S_j(j_1,...,j_k)$ of degree k we define the r–th distance ϕ_r, or $\phi_r(\delta)$, to be $\phi_r = j_r - i_r$, $(1 \le r \le k)$. The k–tuple $<\phi_1,\phi_2,...,\phi_k>$ is called the dependence distance vector. As an example of dependence calculation, consider the following loop

```
      DO I = 1, N
S1:     A(I+K) = ...
S2:     ... = A(I)
      ENDO
```

where K is a nonnegative integer constant. Here $s_1 \le s_2$ and $IN(s_1) \cap OUT(s_2) \neq \emptyset$, but we cannot determine yet whether a flow dependence from s_1 to s_2 exists. Several factors must be considered in this case. For a dependence to exist we must have two values of the index I, I_1 and I_2, such that $1 \le I_1 \le I_2 \le N$ and $I_1 + K = I_2$. To test this we must know the values of K and N. In most programs the value of K is known at compile–time but this is not always true for loop bounds like N. If $K \le N$ then a dependence may exist. However if $K > N$ no dependence between the two statements can exist. Frequently loop bounds are not known at compile–time but to be on the safe side they are assumed to be "large". In general, dependences can be computed by solving a Diophantine equation similar to the above. Algorithms for computing data dependences are given in [Bane88], [WoBa87].

The program *data dependence graph* or DDG, is a directed graph $G(V,E)$ with a set of nodes V corresponding to statements in a program, and a set of arcs E representing data dependences between statements. A DO loop denotes an fixed–iterative loop, which is serial. A loop whose iterations can execute in parallel and in any order is called DOALL. The dependences in certain loops may allow only partially overlapped execution of successive iterations. These loops are called DOACROSS and are mentioned in only a few cases in this paper [Cytr84]. Of course, a loop is marked as being DO, DOALL, or DOACROSS after the necessary dependence analysis for that loop has been carried out.

6.2. Automatic Program Parallelisation

After determining dependences and building the DDG, the compiler starts the process of optimising and restructuring the program from a serial into a parallel form. During the optimisation phase several architecture independent optimisations are applied to the source code to make it more efficient, more suitable for restructuring, or both. The restructuring phase which actually transforms the program into a vector and/or parallel form is also organised into architecture independent and architecture dependent subphases.

Loop Vectorisation and Loop Distribution: When compiling for a vector machine, a vectorising compiler attempts to generate vector instructions out of innermost loops [Kenn80], [PaWo86], [Wolf82]. To do this the compiler must check all dependences inside the loop. In the most simple case where dependences do not exist the compiler can *distribute* the loop around each statement and create a vector statement for each case. Vectorising the following loop

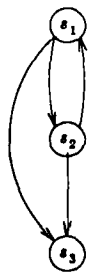
```
      DO I = 1, N
S1:     A(I) = B(I) + C(I)
S2:     D(I) = B(I) * K
      ENDO
```

would yield the following vector statements

```
S1:   A(1:N) = B(1:N) + C(1:N)
S2:   D(1:N) = K * B(1:N)
```

In the original loop one element of A and one element of D were computed at each iteration. In the latter case however all elements of A are computed before computation of D starts. This is the result of distributing the loop around each statement. In general, loop distribution around two statements s_i and s_j (or around two blocks B_i and B_j) is legal if there is no dependence between s_i and s_j (B_i and B_j), or if there are dependences in only one direction. By definition, vectorisation is only possible on a statement–by–statement basis. Therefore in a multistatement loop, loop distribution must be applied before vector code can be generated. As an example, consider the following serial loop.

```
  DO I = 1, N
S1:   A(I+1) = B(I-1) + C(I)
S2:   B(I) = A(I) * K
S3:   C(I) = B(I) - 1
  ENDO
```



The data dependence graph is shown on the right. A simple traversal of the dependence graph would reveal its strongly connected components (SCC). Loop distribution takes place around each SCC. Those SCCs with single statements (that do not have self–dependences) can be vectorised. The result of vectorising the above loop would be:

```
DO I = 1, N
   A(I+1) = B(I-1) + C(I)
   B(I) = A(I) * K
ENDO
C(1:N) = B(1:N) - 1
```

Due to the dependence cycle the loop cannot be distributed around s_1 and s_2.

Loop Interchange: Loop interchange can used to interchange the nest depth of a pair of loops in the same nest, and can be applied repeatedly to interchange more than two loops in a given nest of loops [PaWo86], [Wolf82]. As mentioned above, when loops are nested vectorization is possible only for the innermost loop. Loop interchange can be used in such cases to bring the vectorizable loop to the innermost position. For example, the following loop

```
DO I = 2, N
   DO J = 2, M
      A(I,J) = A(I, J-1) + 1
   ENDO
ENDO
```

is not vectorizable because the innermost loop (a recurrence) must be executed serially. But the outermost loop is parallel. By interchanging the two loops and vectorizing the innermost we get

```
DO J = 1, M
   A(1:N,J) = A(1:N, J-1) + 1
ENDO
```

Loop interchange is not always possible. In general a DOALL loop can be interchanged with any loop nested inside it. The inverse is not always true. Serial loops for example cannot always be interchanged with loops surrounded by them. Interchange is illegal, for example, in the following loop.

```
DO I = 2, N
   DO J = 1, M
      A(I,J) = A(I-1,J+1) + 1
   ENDO
ENDO
```

In general loop interchange is impossible when there are dependences between any two statements of the loop with "<" and ">" directions [Wolf82]. In vectorization, interchange should be done so that the loop with the largest number of iterations is brought to the innermost position. This would create vector statements that will operate on long vector operands. For memory–to–memory systems (e.g., CDC Cyber 205) long vectors are particularly important. If on the other hand we compile for a scalar multiprocessor, bringing the largest loop (in terms of number of iterations) in the outermost position is more desirable, since that would allow the parallel loop to use more processors.

Node Splitting and Statement Reordering: Loop vectorization and parallelization is impossible when the statements in the body of the loop are involved in a dependence cycle. Dependence cycles that involve only flow dependences are hard to break. There are cases however where dependence cycles can be broken resulting in total or partial parallelization of the corresponding loops. One case where cycle breaking is possible is with dependence cycles that involve flow and antidependences. Consider for example the following loop whose statements are involved in a dependence cycle with a flow and an antidependence.

```
 DO I = 1, N
s1:  A(I) = B(I) + C(I)
s2:  D(I) = A(I-1) * A(I+1)
 ENDO
```

Due to the dependence cycle this loop cannot be vectorized. Node splitting can be employed here to eliminate the cycle by splitting the node (statement) which causes the anti–dependence. This is done by renaming variable A(I+1) as TEMP(I) and using its new definition in statement s_2. After the cycle is broken loop distribution can be used to distribute the loop around each statement. The loop can now be distributed around s_2 but not around s_1 and s_3, since there are dependences in both directions. Statement reordering can be used here to reorder the statements of the loop (reordering is not always legal). The loop satisfies now the "one direction dependences" rule, and thus it can be distributed around each statement

resulting in the three vector statements shown below.

```
TEMP(1:N) = A(2:N+1)
A(1:N) = B(1:N) + C(1:N)
D(1:N) = A(0:N-1) * TEMP(1:N)
```

Loop Blocking: *Loop blocking* or *strip mining* is a transformation that creates doubly nested loops out of single loops, by organising the computation in the original loop into chunks of equal size [PaWo86]. Loop blocking can be useful in many cases. It is often used to manage vector registers, caches, or local memories with small sizes. Many vector computers for example have special vector registers that are used to store vector operands. Loop blocking can be used to partition vector statements into chunks of size K, where K is the length of the vector registers. The following loop

```
DO I = 1, N
   A(I) = B(I) + C(I)
ENDO
```

will become (after blocking)

```
DO J = 1, N, K
   DO I = J, MIN(J+K, N)
      A(I) = B(I) + C(I)
   ENDO
ENDO
```

In the same way blocking can be used to overcome size limitations of caches and local memories. In parallel–vector machines operations with long vector operands can be partitioned into shorter vectors, assigning each of the short vectors to a different processor. In this case loop blocking will introduce a parallel loop as in the following example. Consider the vector statement.

```
A(1:N) = B(1:N) * C(1:N)
```

In a *system with P–processors* (and if $N>>P$), this vector operation can be speeded up further by blocking it as follows.

```
K = TRUNC(N/P)
DOALL I = 1, P
    A((I-1)K+1:IK) = B((I-1)K+1:IK) *
                                C((I-1)K+1:IK)
ENDO
A(PK+1:N) = B(PK+1:N) * C(PK+1:N)
```

Notice that iteration pealing was implicitly used to eliminate the use of the intrinsic function MIN in the DOALL statement.

Cycle Shrinking: *Cycle shrinking* transforms a serial DO loop into two perfectly nested loops; an outer serial and a parallel inner loop [Poly88]. It is based on the observation that although there is a static flow dependence $S_1 \delta S_2$ between two statements S_1 and S_2 of a loop, there may be instances of S_1 and S_2 that are not involved in a dependence (if the dependence distance is greater than one). Cycle shrinking extracts these dependence–free instances of the statements inside a loop, and creates an inner parallel loop. Consider for example the following loop.

```
DO I = 1, N
   X(I) = Y(I) + Z(I)
   Y(I+3) = X(I-4) * W(I)
ENDO
```

Such a loop would be treated as serial by the existing compilers. However, if cycle shrinking is applied the same loop will be transformed to the following one.

```
DO J = 1, N, 3
```

```
      DOALL I = J, J+2
          X(I) = Y(I) + Z(I)
          Y(I+3) = X(I-4) * W(I)
      ENDOALL
ENDO
```

The transformed loop can now be executed λ=3 times faster, where λ is the cycle *reduction factor*. The larger the distance λ, the greater the speedup. Consider a DO loop with k statements which are involved in a dependence cycle. If the reduction factor for that cycle is λ, cycle shrinking results in an improvement factor of $\lambda * k$. This is true since not only the iterations of the *DOALL* loop created by cycle shrinking are independent, but the statements inside each iteration are also independent. Thus parallel loop execution can be combined with parallel execution of the statements of each iteration.

Loop Spreading: Few transformations exist for interloop optimisations and all of them regard memory related optimisations (e.g., loop fusion [Wolf82]). *Loop spreading* extracts parallelism from loops in cascade [GiPo88b]. In particular, loop spreading is most useful for chains of serial loops with interloop dependences. The transformation works by pairing off iterations from different adjacent loops and executing those iterations in parallel. Thus the expected speedup improvement is bounded by the number of serial loops in a chain. In this section we give an example of a simple case of two serial loops with interloop dependences. A complete version of loop spreading appears in [GiPo88b].

Let B_1 and B_2 be two serial loops in sequence and let $\beta_i(j)$ denote the j-th iteration of B_i. The transformation will produce a new serial loop such that each iteration of the new loop will execute one iteration of B_1 (say $\beta_1(i)$) and one iteration of B_2 (say $\beta_2(i-k)$) in parallel. The problem is to determine which iterations should be combined in order to maximise the parallelism yield by loop spreading. This is equivalent to computing the best value of k in $\beta_2(i-k)$. Consider for example, the following serial loops with interloop dependences.

```
DO I = 1, 10
   X(3I+4)= A(I-1)+1
   A(I) = Y(-2I+25)
ENDO

DO I = 1, 10
   D(I) = X(4I+2)
   X(I+1) = D(I)**2 + D(I-1)
   E(I) = Y(-2I+23)
ENDO
```

Notice that loop spreading does not alter the order of execution of the iterations of each loop. Thus, loop contained dependences are satisfied in all cases. The transformation must assure that interloop dependences are also satisfied, by choosing a value of k such that the pair-wise parallel execution of iterations of the two loops does not violate any dependence. Or equivalently, if iterations $\beta_1(i)$ and $\beta_2(i-k)$ are to be executed in parallel we must verify that no flow, anti, or output dependences exist for all i such that $1 \leq i \leq N$. It is clear that for maximum parallelism we must choose the minimum value of k, which in the case of the above example is k=3. Hence after loop spreading the above loop becomes

```
DO I =1, 10
   COBEGIN
    β1(I);
   IF (I > K) THEN β2(I-3);
   COEND
ENDO

DO I = 8, 10
  β2(I)
```

```
ENDO
```

Assuming $\beta_1(I)$ and $\beta_2(I)$ take the same time, τ, to execute, the total execution time of the original loops is 20τ, while that of the transformed loop is 13τ. This is the best possible overlap for the above example.

Run–Time Dependence Testing Most techniques that have been developed to analyse array subscripts and determine loop dependences, solve this problem at compile–time. This of course is desirable because there is no run–time overhead. Another alternative would be to determine data dependences dynamically at run–time. The next transformation (*run–time dependence checking* or RDC), does precisely this. When the dependence distances vary between different iterations, cycle shrinking is rather conservative. RDC is a more suitable technique since it sequentialises only those iterations that are involved in a true dependence. All remaining iterations can execute in parallel. Consider for example the following loop

```
DO I = 1, N
   A(2I-1) = B(I-1) + 1
   B(2I+1) = A(I+1) * C(I)
ENDO
```

The distance of the flow dependence from the first to the second statement can take the values 1, 2, 3,.... RDC has two phases. An *implicit* and an *explicit* phase. The implicit part involves computations performed by the compiler which are transparent to the user. The explicit phase transforms the loop itself. The basic idea is to be able to determine at run–time whether a particular iteration of a loop depends on one or more previous iterations. This requires some recording of relevant information from previous iterations. For a loop `DO I = 1, N` and for a dependence $S_j\ \delta_j\ S_{j+1}$ in that loop, we define the *dependence source vector* (or DSV) R_j to be a vector with N elements, where non–zero elements indicate the values of I for which S_j is a dependence source, and zero elements in R_j correspond to values of I for which S_j is not involved in a dependence. The elements of DSV are initialised to zero by the compiler. A single bit–vector V with subscripts in the range [1...N] is also created and is initialised to zero. Vector V is called the *synchronization vector.*

Following initialisation, the compiler inserts in the transformed loop code which records dependence sources and synchronises dependence sinks on their corresponding *sources. Even though the compiler* inserts dependence–testing code in a loop, the actual dependence resolution occurs during the execution of the loop. The transformed version of the previous loop is shown below.

```
   DOALL I = 1, N
      COBEGIN
IF (1 ≤ R1(I+1) < I) WAIT ON V(I+1);
IF (1 ≤ R2(I-1) < I) WAIT ON V(I-1);
      COEND
      A(2I-1) = B(I-1) + 1
      B(2I+1) = A(I+1) * C(I)
      CLEAR V(I)
   ENDO
```

Run–time dependence checking belongs to a family of transformations called hybrid (static and dynamic) transformations. In all cases the principle is the same: if the compiler cannot resolve a particular problem but it can precisely identify the problem, it can generate code which solves that problem during execution, when run–time information becomes available.

7. Partitioning, Synchronisation, and Scheduling

In Sections 5 and 6 we reviewed manual parallel programming through the use of parallel languages, and automatic program restructuring via parallelising compilers. However, the major advantage of parallelising compilers is not only automatic parallelisation of serial programs, but their potential to automate the process of program partitioning, synchronisation, and scheduling (which are by far more complex than parallelism detection and often ad hoc).

7.1. Partitioning and Task Formation

Program partitioning is one of the most important phases of parallel programming. It is closely linked to scheduling and even though it is discussed separately here, partitioning should be considered in conjunction with scheduling. Informally, the partitioning phase decides which modules of a program can execute in parallel and how. A parallel program can be represented by a directed graph called the *task graph*, where nodes represent program modules and arcs represent ordering and dependence relations between nodes. The nodes of a program graph are called *tasks*. A task can consist of a single or multiple *user processes* or *u-processes*. U-processes are units of code with their own instruction stream and private memory space. Thus tasks are static entities which simply correspond to specific modules of a program. On the other hand u-processes are generated from tasks upon dispatching, and in addition to their code they own part of the virtual address space.

So far partitioning has been treated mostly from a theoretical point of view as a problem which is often decoupled from scheduling [Bokh88]. Most of the existing partitioning algorithms assume an idealized representation of a program which usually takes the form of a directed graph as the initial representation. This graph can represent a user-defined partition or a compiler representation of the program graph (e.g., control flow or data dependence graph, or a higher-level program representation). Nodes represent tasks and arcs represent communication channels between tasks. Such graphs are evaluated under the assumption that each node may execute on a different processor at any time. Assumptions such as known execution times of nodes and communication weights among nodes, are then used to derive a more efficient task graph. This can be done by merging nodes together to eliminate heavy communication links, or by splitting large nodes into smaller ones to increase the degree of parallelism in the graph. The end-result is a more efficient representation of the program graph or even an explicit assignment of groups of nodes to specific processors. Even though the above models are far from being representative of real programs, the algorithms which have been developed can be used for approximate solutions and have contributed to the analysis and understanding of many important aspects of the problem.

On a more practical basis the partitioning problem can be considered from two different angles: the data and the instruction stream viewpoint. In the first case, partitioning is based on the decomposition of data objects upon which computation is performed. Each processor is assigned the work corresponding to a specific data domain. This form of partitioning is often called *data partitioning* or *horizontal partitioning*. Data partitioning is feasible when the same type of computation is performed on all data domains. Typical computations of this type include loops and other repetitive computations, i.e., the same u-process is executed for each data domain.

The second type of partitioning called *functional* or *vertical partitioning* results in the formation of tasks from syntactically identifiable pieces of code. Thus different partitions operate on different data objects or on the same data object but in some specific order. For example, forming two tasks out of two disjoint outer loops or two different subroutine calls is a case of functional partitioning. Another common term for functional partitioning is *high-level spreading*. Partitioning must be done such that the following goals are met.

- The tasks formed by partitioning a program should be as independent as possible, i.e., sharability of data objects between tasks should be minimal. This implies that data objects should be decomposed such that different components correspond to different tasks. Notice that both data and functional partitioning conform to this requirement.

- Tasks should be of approximately equal size. As it will be shown later, this helps in balancing the load across processors by using simple and fast scheduling heuristics. Data partitioning tends to satisfy this requirement while functional partitioning does not.

- The size of the tasks formed should be a function of the overhead incurred during task scheduling and the synchronisation overhead. Put in other words, tasks should be large enough compared to the overhead involved inorder to achieve any speedup. Roughly speaking, this means that the total overhead associated with the parallel execution of a

task should be less than the its serial execution time.

- A balance must be achieved between communication and scheduling overhead, and degree of parallelism in a task graph. These two objectives are inconsistent since minimising overhead tends to merge all tasks into fewer large tasks, while increasing the degree of parallelism (i.e., the number of independent tasks in the graph) tends to decompose large tasks into their smallest constituents. The degree of parallelism should be considered in conjunction with the number of available processors.

- Finally partitioning should be based on realistic assumptions about the program. For example, the type of a task is readily available to the compiler but the execution time of a task is not. If exact algorithms are used for partitioning, e.g. critical path, they should be adapted to compensate for inaccuracies.

Even though we will return to the partitioning issue later in this paper, for now we assume that this phase provides the following phases of parallel programming with a well-defined decomposition of a program into a set of tasks.

7.2. Synchronisation and Communication

Depending on the architecture of the target machine, explicit synchronisation or communication instructions must be inserted in a parallel program to ensure correct parallel execution. Again we look at the problem from the compiler point of view. Synchronisation may or may not be needed depending on the order of task execution. For example, if different tasks involved in a data or control dependence are allowed to execute in parallel, then explicit synchronisation instructions must be inserted. Another type of synchronisation is needed if tasks are restricted to execute in parallel only if they are independent; in that case a single (barrier) synchronisation point between two tasks will suffice. Similarly, in the distributed memory case, appropriate messages should be explicitly routed between different tasks. However, communication within a single processor (task) can take the form of synchronisation through the local memory (a much less expensive operation).

Synchronisation and/or communication introduce overhead during parallel execution. As mentioned earlier, overhead estimates are used in deciding how a program can be partitioned into tasks. Thus, prior to partitioning, it must be known where synchronisation and communication is needed. This information need not be in the form of the corresponding instructions; cost and some qualitative information is sufficient for determining overheads. After partitioning has been specified, only those synchronisation instructions needed to enforce the dependences expressed in the task graph need to be inserted in the program.

Alternatively, synchronisation and communication instructions can be inserted in a program wherever applicable. After partitioning and possibly scheduling, an optimisation phase must follow to eliminate redundant synchronisation and communication instructions. Approaches to the later optimisation problem are reported in [MiPa86]. It is also clear that the static or dynamic nature of partitioning and scheduling affects directly the generation and optimisation of synchronisation instructions. For example, if a task is allowed to disintegrate dynamically during execution, static synchronisation (which must consider the worst case) may result in superfluous synchronisation at run-time.

7.3. Scheduling and Overhead Analysis

Scheduling is one of the most performance-sensitive phases of parallel programming. Deciding what task or process executes on what processor and in what order is a nontrivial problem. Again, scheduling should be done such that program finish time is minimised. Partitioning is some type of incomplete scheduling, since it specifies an "abstract" processor allocation.

Although the end-result of scheduling (which is minimum execution time through load balancing and low overhead) is the same for both shared and distributed memory systems, the approaches differ for each

model. In shared memory parallel processors all tasks and data objects of a program are equally accessible to all processors. This simplicity allows more flexibility in scheduling tasks. In distributed memory systems one of the early differences is the need to down–load different tasks on different processors. Adjacency of the interconnection structure has a large implication on how efficiently scheduling can be done. For example, if a complete interconnection is used for a distributed memory system, a task graph can be scheduled in exactly the same way as if it was to be executed on a shared memory system, by simply translating synchronisation instructions to equivalent message passing instructions. However, if the interconnection is more sparse (e.g., hypercubes), different issues arise. Besides the extra overhead of down–loading, one needs to face other problems such as process migration and nonuniform communication costs which further complicate the overall optimisation problem.

Unless a precise static allocation is achieved process migration should be allowed to balance the load across the entire system. Because exact program information is unavailable at compile–time, a large number of scheduling algorithms for distributed systems is based on the process migration principle; thus allowing quick initial allocations which balance themselves by transmitting and receiving other tasks from neighboring processors. This activity introduces significant space and time costs since its implementation requires for each processor the maintenance and update of load–tables regarding adjacent processors. If system–wide migration is allowed, then in a p–processor machine each processor needs to maintain load information for the other $p-1$ processors. Moreover, since migration outdates this information, periodic system–wide updates must take place by having processors exchange load information among themselves. Even though the above is the best approach for achieving the most perfectly balanced load, it incurs a tremendous overhead. A middle–ground solution is to define neighborhoods of processors and allow process migration only within each neighborhood. Thus load tables and updates occurs independently within each group by compromising load balancing. How these neighborhoods are defined depends heavily on the interconnection structure and a number of other parameters. Down–loading, communication, and process migration have a profound affect on task granularity. Because all these extra scheduling activities are orders of magnitude more expensive that simple synchronisation, tasks need to be much larger inorder to effectively amortise the overhead and achieve speedups over serial execution. Increasing substantially the granularity of tasks results in decreasing the degree of parallelism of a program, Therefore, it also restricts the number of applications which can benefit from parallel execution on such systems.

In contrast, none of the above activities, besides synchronisation, occur during scheduling in shared memory parallel machines. Migration is not necessary since due to sharing of common memory and in terms of cost, such a system can be thought of as completely connected — the communication cost between any pair of processors is constant. Moreover, down–loading is not necessary (except is the case of fully static scheduling). Tasks can remain in shared memory and be dispatched by processors as needed. This greatly simplifies the scheduling problem for such architectures, but even in this simpler form, scheduling remains a crucial albeit all but trivial problem to solve.

Thus, based on the above issues, if one was to judge conformable (comparable) shared versus distributed memory parallel architectures based on the scheduling issue alone, the superiority of the former architecture model is clear. Of course, if very "dense" interconnection structures are employed for distributed systems this advantage drifts away. Nevertheless, one can argue that in such a case the cost of building multiport nodes may become prohibitive and we soon converge to an architecture which can be more effectively realised as a shared memory.

The remaining discussion on scheduling focuses on shared memory architectures even though many of the ideas discussed can also be applied to distributed memory systems. There are three fundamental approaches to scheduling: *static, dynamic,* and *hybrid.* Each approach depends on how much information about a program is available to the scheduler, as well as on the phase this information becomes available. We consider below each approach separately.

• **Static scheduling:** As implied by the term, static scheduling can be performed either by the programmer or by the compiler before program execution. Parallelism is exploited by spreading different computations over different processors statically. Thus, the programmer knows exactly what parts of a program will execute on each processor. For static scheduling to be effective one needs to have detailed knowledge about a

Type of Scheduling	Best OS mode	Average balancing	Average synch/comm overhead	Other run-time overhead	Average complexity	Detail of program info needed
Static	monoprog.	low	low	low	high	high
Dynamic	multi/mono	high	high	high	low	low
Hybrid	multi/mono	moderate	moderate	moderate	moderate	moderate

Table 2. Characteristics of scheduling strategies.

program (such as the outcome of conditionals, the size of loops, etc) available in advance. Knowledge about the architecture of the target machine (e.g., number of physical processors, timing of I/O and functional units) is also necessary. Such details are not usually available but coarse estimates can be supplied by the programmer or the compiler.

The main advantage of static scheduling is its low run-time overhead; no scheduling activities occur during program execution. It is also easier to trace the execution of a statically scheduled parallel program and thus, debugging becomes less of a problem. Overhead however is paid during compilation or in terms of programming time. Static scheduling algorithms involve usually polynomial or even exponential complexity. Therefore, more time is spend at compilation. The major drawback is the unrealistic, in general, assumptions upon which static scheduling is based. The result is typically an unbalanced load. Since parallel execution time is defined by the last processor to finish, unbalanced load results in longer parallel execution times and low machine utilisation. In general, pure static scheduling is *not* a suitable approach for general-purpose parallel machines. It can be effectively used for specialised architectures such as systolic arrays or VLIW-based machines.

• **Dynamic scheduling:** Dynamic scheduling is complementary to static in both its advantages and disadvantages. Dynamic scheduling is implemented at run-time through the operating system, the compiler, the hardware, or a combination thereof. Scheduling is based on simple, typically constant-time heuristics which work satisfactorily in most cases. Knowledge about program characteristics is not needed; at best some qualitative knowledge can be useful. The major drawback of dynamic scheduling is the run-time overhead that it incurs. Since decisions are taken during program execution, scheduling activities waste processor cycles. The more sophisticated the scheduling heuristics the higher their complexity and thus the higher the overhead. On the other hand, dynamic scheduling achieves the highest degree of load balancing under the same initial conditions.

In principle, dynamic scheduling can be used at all levels or task granularity starting from the instruction level up to the program level. However, the suitability of a particular scheme depends on many factors such as the complexity and the overhead in both time and hardware. For general purpose parallel processors dynamic scheduling is most appropriate for large-to-medium granularity tasks.

• **Hybrid scheduling:** Since static and dynamic are complementary approaches with respect to balancing and overhead, a combination of the two may result in more efficient implementations of scheduling on the average. Few hybrid schemes have been designed or implemented on real machines. Even though they involve more complexity from the design and implementation perspectives, they do offer the most attractive alternative.

During hybrid scheduling, some of the scheduling takes place at compile-time and some at run-time. Qualitative information about programs is usually enough for hybrid schemes to work efficiently. We

overview such an approach in Section 8. Static partitioning with dynamic task scheduling is a type of hybrid scheduling, since partitioning directly affects scheduling as indicated earlier. Table 6 summarizes the various approaches to parallel program scheduling and their characteristics with respect to complexity, overhead, load balancing, and input information needed about the subject program.

7.3.1. Loop Scheduling, Microtasking, and Macrotasking

Cray Research was the first supercomputer vendor to introduce software which supported the specification and scheduling of parallel constructs on the Cray X-MP series. This software known as *multitasking* provided only the environment for creating and executing tasks of the same program on different processors, but it left the responsibility for defining, scheduling, and synchronising tasks to the programmer. The multitasking environment was based on a collection of macros which allowed the user to create processes (much like Unix processes) through calls to the operating system. These invocations to the operating system made multitasking a very expensive means for exploiting program parallelism. In addition, tasks needed to be organised as subroutines with the extra overhead of subroutine call paid for every instantiation of a new task. The overhead associated with multitasking made it useful only in cases where large subroutines could execute simultaneously. Many other parallel computer vendors followed with different implementations of multitasking.

Later versions of multitasking software avoided excessive overhead and made parallelism exploitation at the loop level possible. Multitasking at the loop level is more commonly known as *microtasking*. Microtasking works much the same way as multitasking with the main difference being that operating system involvement is kept minimal. Instead of invoking the operating system to create a process and allocate space for it every time a new task needs to be initiated, microtasking creates a number of processes at the beginning of program execution. These processes which we call here *system processes* or *s-processes* remain live throughout the execution of a program. When a u-process (task) needs to be scheduled for execution, an s-process is fetched from a queue of s-processes and it is bound to that particular u-process. This is when an s-process receives *context* and it is called an s-process with context or *s-c-process*. Upon completion of the execution of an s-c-process the s-process is not destroyed (as it was the case with early versions of multitasking), rather it returns as an empty s-process to the above queue, and it can be used later to execute another u-process form the same program. Thus s-processes function as vehicles which carry u-processes through execution in a physical processor.

Micortasking avoids unnecessary overhead by reusing s-processes within each program. In more efficient implementation of microtasking s-processes are created not just once per program, but once when the system is cold-started and they are bound to u-processes from possibly different user programs at the same time. Thus the overhead is further reduced. The number of s-processes so created is a system parameter and it depends among others on the number of physical processors and memory space available. Micortasking is commonly used for the execution of parallel loops by many processors. In most cases each different loop iteration is bound to an s-process.

Microtasking as described above is a suitable solution for loop scheduling (data partitioning) but not necessarily for high-level spreading (functional partitioning) where the definition of a task is arbitrary. During the execution of parallel loops all s-processes inherit the same calling environment, but this is not always the case with tasks generated from high-level spreading. To support parallelism at this level multitasking is still used under the distinguishing term of *macrotasking*. Macrotasking is a more tunned implementation of multitasking but it is based on the same framework.

It is important however, to realize that microtasking and macrotasking provide the environment which supports parallel programming at the user-level but they do not offer solutions as to how tasks should be organized (partitioning), synchronized, and scheduled. This is still the programmer's or the compiler's responsibility. In Section 8 we discuss a more general environment which in addition to providing capabilities equivalent to micro and macrotasking, it automates the process of partitioning and scheduling in a unique way.

Loop Scheduling (Process Level)

Loops in numerical programs can be fairly complex with conditional statements and subroutine calls. A general solution should distribute iterations to processors at run-time based on the availability of processors and other factors. However, the overhead associated with run-time distribution must be kept very low for dynamic scheduling to be practical. We consider here three possible schemes for loop scheduling. The first two are well-known and one or the other is used by most modern parallel machines; the third is a more efficient approach. The three schemes differ in the number of loop iterations that they assign to each idle processor, and thus in the balancing of load and the total execution time.

One Iteration at a Time (Self-Scheduling): This scheduling scheme is commonly referred to as *self-scheduling*. An idle processor picks a single iteration of a parallel loop by exclusively incrementing the loop indices [Smit81], [TaYe86]. Thus if N is the total number of iterations of a loop, self-scheduling involves N dispatch operations. Roughly speaking, if p processors are involved in the execution of the loop, then each processor gets N/p iterations. Let B be the average iteration execution time and σ the overhead involved with each dispatch. Then self-scheduling is appropriate if $B \gg \sigma$ and there is a large variation of the execution time of different iterations. Because self-scheduling assigns one iteration at a time, it is the best dynamic scheme as far as load balancing is concerned. However, a perfectly balanced load is meaningless if the overhead used to achieve it exceeds a certain threshold. Overall, self-scheduling may be appropriate for loops with relatively small number of iterations which have variable execution times, and only if σ is small compared to B.

Chunk-Scheduling: *chunk-scheduling* is in principle the same as self-scheduling. But in this case, a fixed number of iterations (chunk) is allocated to each idle processor (as opposed to a single iteration at a time). By doing so, one can reduce the overhead by compromising load balancing. This is clear since the unit of allocation is of higher granularity now, and thus the potential variation of finish time among the processors is also higher. There is a clear tradeoff between load balancing and overhead. At one extreme, the chunk size is roughly N/p and each processor performs only one dispatch per loop. The variation of finish time is also the highest in this case. At the other extreme, the chunk size is one and we have self-scheduling with perfect load balancing and maximum overhead. Intermediate values of the chunk size in the range $[1 \ldots \lceil N/p \rceil]$ will produce results that are better or worse than either of the extreme cases. The main drawback of chunk-scheduling is the dependence of chunk size on the characteristics of each loop which are unknown even at run-time. Worse yet, even for the same loop, the execution time is not monotonous with monotonically increasing or decreasing chunk size. This makes the derivation of an optimal chunk size practically impossible even on a loop-by-loop case.

Guided Self-Scheduling: Self-scheduling achieves a perfect load balancing but it also incurs maximum overhead. On the other hand chunk-scheduling is an (unsuccessful) attempt to reach a compromise between load balancing and overhead, and the result maybe quite unexpectable. The third scheme, *guided self-scheduling* (or GSS) [PoKu87], is in general, a much better and more stable approach to reach this compromise. The idea is to start the execution of a loop by allocating chunks of iterations whose size starts from $\lceil N/p \rceil$ and keeps decreasing until all the iterations are exhausted. The last $p-1$ chunks of iterations are of size one. Thus, chunk sizes vary between the two extremes. Figure 2 gives the GSS algorithm.

The advantages of GSS are many. First, the property of decreasing chunk size is built-in and no extra computation is required to enforce this policy. This simplicity allows for easy and efficient implementation. Secondly, the two main objectives of perfectly balanced load and small overhead are achieved simultaneously. By allocating large chunks at the beginning of the loop we keep the frequent dispatching and thus the overhead low. At the same time, the small chunks at the end of the loop serve to "patch holes" and balance the load across all processors. For some ideal cases, GSS is provably optimal. This cannot be said for either self or chunk-scheduling. GSS has been implemented in the Cray's autotasking library as well as in DEC's Fortran 5.0 compiler. For a parallel loop with N iterations the average number of dispatch operations per processor for self, chunk, and guided self-scheduling is N/p, N/kp, and $\log_e(N/p)$ respectively.

Guided Self-Scheduling

Input A parallel loop L with N iterations, and p processors.
Output The optimal dynamic schedule of L on the p processors. The schedule is reproducible if the execution time of the loop bodies and the initial processor configuration (of the p processors) are known.

- If R_i is the number of remaining iterations at step i, then set $R_1 = N$, i=1, and for each idle processor do.

REPEAT

- Each idle processor (scheduled at step i) receives

$$x_i = \left\lceil \frac{R_i}{p} \right\rceil$$

  iterations.

- $R_{i+1} = R_i - x_i$
- The range of the loop index is $I \in [N-R_i+1, \ldots, N-R_i+x_i]$
- $i = i + 1$

UNTIL $(R_i = 0)$

Figure 2. The GSS algorithm.

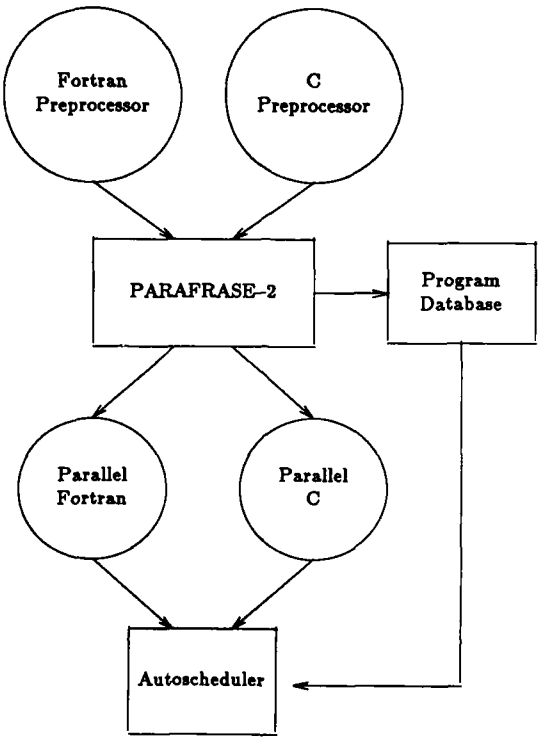


Figure 3. The structure of the Parafrase–2 multilingual compiler.

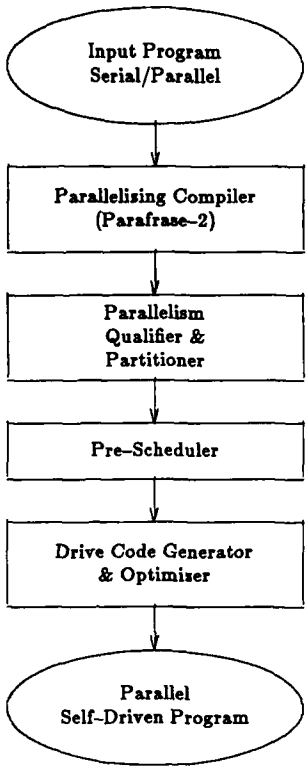


Figure 4. The major modules of an auto–scheduling compiler.

8. A Performance–Oriented Environment for Parallel Programming

In this section we discuss our approach to a fully automated environment for automatic, efficient, and effective parallelism specification and exploitation. Having considered partitioning, synchronization, and scheduling in a general context, it should be easy to justify our approach to an integrated programming environment in the Parafrase–2 project.

The organization of Parafrase–2, the first *multilingual* parallelising compiler to be built, is shown in Figure 3. This ongoing project at the University of Illinois' CSRD, aims to develop a single parallelizing compiler for many popular programming languages [PGHL89]. Even though at present the system compiles C and several Fortran dialects, provisions have been made to add Pascal and other languages in the future. Adding a new language involves a preprocessor for that language, which translates a source program *into a common intermediate representation*. The major emphasis of the Parafrase–2 project is the second phase of parallel programming, namely that of parallelism exploitation. In the rest of this section we describe our approach to this latter part of the compiler.

There are advantages and disadvantages to each scheduling approach. With the growing variety and complexity of parallel architectures monolithic approaches to program scheduling become obsolete. Both compilers and run-time systems need to cooperate in order to achieve desirable results. Pure static schemes are too unrealistic to be practical. Similarly, pure dynamic schemes that ignore useful program information are bound to fail badly in certain cases [Poly88]. An ideal scheduler should use to its advantage information about a program, but it should also operate in the "obvious and least expensive" mode whenever information is inadequate. Our approach to the parallelism packaging and scheduling problem is a blend of compiler and run-time schemes. Figure 4 shows the different components of our framework as parts of what we call an *auto-scheduling compiler* [Poly88].

Partitioning and qualification of parallelism: This phase is responsible for partitioning the code (and/or data structures) of a program into identifiable modules which are treated as units when it comes to firing. For example, compiling vector or VLIW instructions, or grouping a set of instructions into a unit is part of partitioning. Program partitioning can be done statically by the compiler. In a fully dynamic environment (e.g., dataflow) partitioning is implicit and depending on our execution model, an allocatable unit can range from a single instruction to a set of instructions. Static (explicit) partitioning is desirable because it exploits readily available information about program parallelism and can consider overhead and other performance factors. In our case we use a semi-static partitioner: the formation of tasks is based on the syntax of the language and other program information, but a task is allowed to be decomposed into subtasks dynamically during program execution. The result of the partitioner is a program task graph with nodes

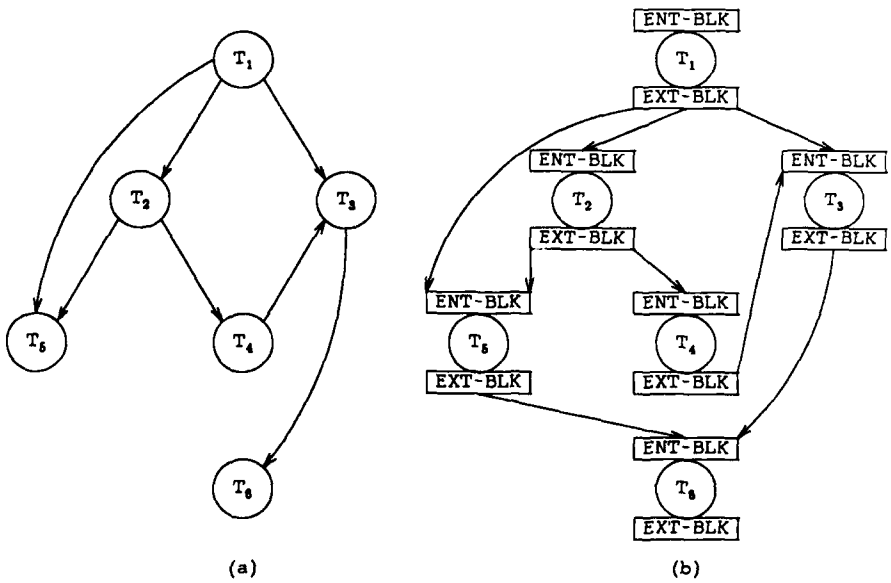


Figure 5. (a) A task graph. (b) The graph with entry and exit blocks.

corresponding to tasks (instruction modules) and arcs representing data and control dependences. Another activity of the partitioner is to perform overhead analysis and determine where tasks need to be split or merged in order to effectively amortise the overhead that arises from dispatching and synchronising the components of such tasks.

Pre–scheduling: After a program is partitioned into a set of identifiable tasks, the compiler can enforce certain scheduling restrictions based on its knowledge about the program. In the presence of dynamic scheduling, pre–scheduling is necessary to eliminate race conditions which may result in performance degradation. Such conditions arise due to implementation of task queues and their access protocols. In the case of a single task queue and under the assumption that parallel tasks remain queued until all processes are spawn (which is often the case in real implementations), race conditions arising during queueing operations may create "artificial" performance bottlenecks. Consider for example the case where P idle processors start executing a program consisting of a large serial $task(s)$ and a parallel task $task(p)$. If $task(p)$ is queued in front of $task(s)$, and assuming that $task(p)$ is large enough to spread across all P processors, then $task(s)$ will execute only after the parallel task has completed, and thus $P-1$ processors will remain idle during the durations of $task(s)$'s execution time. Such scenarios can arise in more complex task graphs as well. By giving priority to serial tasks and delaying parallel tasks for as long as all processors are busy, such performance hasards can be eliminated.

Dynamic task scheduling (nested parallel tasks): Scheduling at the task level is then performed dynamically during execution. Traditionally, dynamic task scheduling has been supported through the operating system directly, or indirectly through the run–time system (e.g., earlier versions of Cray multitasking). The major drawback to this approach is the enormous overhead involved in OS invocations.

The auto–scheduling approach proposed in [Poly88] is based on the idea that the program itself is responsible for packaging and managing its parallelism, and thus the processors allocated to it. Under an ASC environment, the compiler which has access to vital program information, generates *drive–code* for each task in the task graph representation of a program. Each task is instrumented with an *entry–block* (ENT–BLK) and an *exit–block* (EXT–BLK). An example of a program task graph is shown in Figure 5a. After drive–code generation the same graph is instrumented as in Figure 5b. This drive–code is responsible for recording precedence relations, enforcing synchronisation, queueing and dequeueing tasks, and decomposing parallel tasks into their constituent processes.

Tasks are treated as units of execution. Tasks which are ready to execute are queued in a ready–task queue. The ready–queue is also a data structure created, owned, and manipulated by each user–program. Each idle processor tries to dispatch the next available task from the queue (if any). Also, tasks are queued and thus are qualified for execution as soon as they become "ready". It is important to note that parallel tasks are dynamically decomposed into a number of smaller processes whose size and requirements depends on the scheduling scheme used. For example parallel loops can be decomposed under the GSS algorithm to achieve load balancing while keeping run–time overhead low. In a simplified scenario each physical processor executes the following loop.

```
LOOP FOREVER
     - pick front queue image;
     - load program counter;
     - execute;
END LOOP
```

Part of task execution involves the execution of the drive–code in the task exit–block which updates other tasks and queues their corresponding images. A typical task exit–block which is shown below

EXIT–BLOCK:

Task–dependent module:

```
-Barrier synchronization;
-Select processor to dequeue current task;
```

```
-Select processor to execute the following;
```

Task-independent module:

```
FOR (all successors of current task) DO
      -Update (dependences of) successors;
      -Queue freed successors;
ENDFOR
```

consists of two modules, the task-dependent and task-independent modules. The former is code that the compiler generates to perform barrier synchronisation (in the case of parallel tasks), select the processor to dequeue the corresponding task from the queue (e.g., the processor to dispatch the last iteration of a loop), and select the processor to execute the second module of the exit-block (in a parallel task that processor will be the one to clear the barrier). The latter module is independent of the type of the task. Only a specific processor (selected by the first module) executes this part. The code in this module updates the precedence relations and queues those successors that have no pending predecessors. Similarly, entry-blocks are organised in two modules, a task-independent and a task-dependent module as shown below.

ENTRY-BLOCK:

Task-independent module:

```
-Allocate private variables and/or stack space;
-Copy parent stack (optional);
```

Task-dependent module:

```
-Execute initialization code (if any);
-Compute number of iterations for this processor;
-Update loop indices;
```

Loop (or parallel task) scheduling: Upon queueing, a serial task is dispatched at once as soon as a processor becomes idle. However, a parallel task can draw several processors to execute it and thus it remains queued until exhausted. The most frequent and important type of parallel tasks are parallel loops. Section 7.3.1 discussed and compared existing and new approaches to dynamic scheduling of parallel loops. These loop scheduling schemes can be viewed as dynamic partitioning of a loop into a set of allocatable units.

Within a processor: We finally face the problem of scheduling within a processor at the fine granularity level. Packaging of parallelism and scheduling at this level is more architecture dependent than any of the earlier phases of program scheduling [Nico84].

9. Multiprocessor Operating Systems

Traditionally, the terms "parallel processing" or "multiprocessing" refer to the parallel execution of different components of a single application. If carried to the extreme, parallel processing is best realized in a batch environment where applications programs execute one after another, and at any given moment, only one program executes on a parallel machine. At the other extreme, a parallel processor machine can be used as a purely multiprogramming box to boost throughput rather than individual turnaround time. Unfortunately, many of today's parallel computers are used in the latter mode of operation. This is mainly due to our little experience and understanding about the new operating system issues brought forward by parallel computers. Classical OS problems such as processor and memory allocation need to be reconsidered for parallel machines [Rash86], [Poly89].

From the machine utilisation point of view the best mode of operation would be somewhere between the two extremes: multiprocessing and multiprogramming at the same time. We call this *polymorphous* processing or *polyprocessing*. Few systems support some primitive flavor of polyprocessing. Table 7 gives an operating system taxonomy based on three fundamental properties. According to this taxonomy an operating system is classified based on whether it is a Single-user (batch) or Multi-user system, whether it

allows Preemption or Non-preemption, and finally on whether it supports Serial or Parallel program execution.

In Table 7 a three-letter abbreviation is used to denote each class. Clearly, the case of SPS is not desirable. All other classes are viable cases including the single-user/preemptive/parallel or SPP. The multi-user/preemptive/serial or MPS class characterises the majority of the operating systems of modern parallel computers. Even though several machines are promoted as MPP systems, only in rare cases one can operate effectively under this mode.

More research needs to be done in evaluating the merrits and drawbacks of each of these classes. Although MPP is intuitively the most desirable one, it may not necessarily be the most performance-oriented one. Idealy, a multiprocessor OS should be *adaptive*, i.e., able to switch between several of the modes in Table 7 automatically, based on the size and the characteristics of the workload. This appears to be the most important feature of future multiprocessor operating systems.

In order to illustrate the significance (with respect to performance) of adaptive OS, consider a large parallel application written e.g., in Cray or Cedar Fortran. Such a program would be heavily instrumented with calls to the macro/microtasking library in order to receive parallel execution. Recall that each call to the run-time system incurs a significant overhead. If that program is executed during a high workload period, it may (based on its priority, demands, etc), end up executing serially (since a finite number of physical processors need to be shared between a large number of jobs). Thus, all the overhead paid for creating parallel processes, scheduling, and performing meaningless synchronisation may be pointless. On the other hand, the same program may execute overnight on an idle system, in which case, it will execute in parallel justifying the overhead of e.g., macro/microtasking. Of course, it is impossible and impractical for the user to edit the code each time the program runs. This should be the responsibility of the OS: based on its knowledge of the workload, it can selectively *deactivate* (some) parallel processing directives in certain programs, which may end up executing serially any way.

In order to built such sophistication into an OS, compiling and operating system issues need to be considered simultaneously. For instance, in an auto-scheduling environment, the responsibility of the OS is to direct idle processors to user program queues, in order to accomplish multiprogramming. It is then the user's program sole responsibility to manage the allocated processor(s) on its own tasks. Some processor allocation issues for adaptive OS are discussed in [Poly89].

10. Conclusions

The lack of methodologies and software to support parallel programming is profound even on the most advanced parallel machines. Parallel programming is a complex task and the performance of a parallel program can be influenced by many different factors such as coding of parallel constructs and/or restructuring, scheduling schemes and scheduling overhead, synchronisation and/or communication cost,

	Single-user		Multi-user	
	Serial	Parallel	Serial	Parallel
Preemptive	SPS	SPP	MPS	MPP
Non-preemptive	SNS	SNP	MNS	MNP

Table 7. A multiprocessor operating system taxonomy.

program and data partitioning and memory allocation. Since there is no general quantitative or even qualitative measure for these parameters, the process of optimising one or more of them is highly empirical, and definitely, application dependent. Presently the state of the art necessitates parallel programming on a case–by–case (i.e. application) basis. This is probably the most serious barrier in the wide–spread use of parallel machines and programming.

The restructuring compilers of the future will undoubtly need to possess many properties that presently, one finds only in a handful of sophisticated (but experimental) parallelising compilers. So far, the attention has only focused on optimisation and restructuring techniques. However, the complexity of the new parallel machines requires the compiler to perform many additional functions than just restructuring. Scheduling is a candidate for the compilers of the near future. Memory management, minimisation of interprocessor communication, synchronisation and various other types of overhead are important issues that could be tackled by the compiler. Another important aspect of the near future compilers for parallel machines is ease of use and interaction with the user. There are many cases where the user's assistance (in the form of assertions for example) is necessary for parallelising a program and exploiting the resulting parallelism.

It is very likely that in the next few years we will see a transfer of many run–time activities (that are now considered the operating system's responsibility), to the compiler. This will become necessary as performance becomes more of a critical factor. Any activity involving the operating system is known to involve a large overhead. This overhead cannot be tolerated above a certain point. Also, in time–sharing, systems knowledge of specific program characteristics is not necessary to achieve high throughput. In parallel processor environments however, knowledge of program characteristics is necessary for minimising program turnaround time. Thus the shift of operating system functions to the compiler will be a logical consequence. Compilers will become highly interactive and far more complex than modern restructurers, while the software layer between the user and the hardware called the operating system will become thinner, at least in high performance computer systems.

Parallelism in algorithms and programs may be implicit, or may be explicitly specified at several different levels. When parallelism exists in fixed–size "quantums", it is rather easy to understand and exploit. The unstructured nature of parallelism makes its efficient exploitation and programming to be complex tasks. Devicing methods and tools that automatically perform these tasks is thus a very important research subject.

Acknowledgements: The author would like to thank David Padua for providing me with Tables 3,4, and 5, and to David Kuck for his comments on early drafts of this paper.

REFERENCES

[AhSU86] A.V. Aho, R. Sethi, and J.D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, Reading, Massachusetts, 1986.

[AhCG86] S. Ahuja, N. Carriero, and D. Gelernter, "Linda and Friends," *IEEE Computer*, Vol. 19, No. 8, August 1986.

[ABCC87] F. Allen, M. Burke, P. Charles, R. Cytron, and J. Ferrante, "An overview of the PTRAN analysis system for multiprocessing," Report RC 13115 (#56866), IBM T.J. Watson Research Center, Yorktown Heights, New York (1987).

[AlKe82] J.R. Allen and K. Kennedy, "PFC: A Program to Convert Fortran to Parallel Form," Techn. Rept. MASC-TR82-6, Rice University, Houston, Texas, March 1982.

[AlKe87] R. Allen and K. Kennedy, "Automatic Translation of FORTRAN Programs to Vector Form," *ACM Transactions on Programming Languages and Systems*, Vol. 9, No. 4, October 1987.

[AlSr88] D. C. Allen and N. S. Sridharan. "Application of the Butterfly parallel processor in artificial intelligence," pp. 153–164 in *Parallel Computation and Computers for Artificial Intelligence*, J. S Kowalik Ed., Kluwer Academic Publishers, Boston, Mass. (1988).

[Alli85] Alliant Computer Systems Corp., "FX/Series Architecture Manual," Acton, Massachusetts, 1985

[ATHM86] M. Amamiya, M, Takesue, R. Hasegawa, and H. Mikami. "Implementation and evaluation of a list-processing-oriented data flow machine," pp. 10–19 in *13th Annual International Symposium on Computer Architecture*,

[ANSI86] American National Standards Institute, *American National Standard for Information Systems. Programming Language Fortran S8 (X3.9-198x)*. Revision of X3.9–1978, Draft S8, Version 99, ANSI, New York, April 1986.

[ArNi87] Arvind and R. S. Nikhil, "Executing a Program on the MIT Tagged-token Dataflow Architecture," *Proceedings of PARLE conference*, Eindhoven, The Netherlands, Springer-Verlag LNCS 259, June 1987.

[Bane88] U. Banerjee, *Data Dependence Analysis for Supercomputers*, Kluwer Academic Publishers, Norwell, MA, 1988.

[Beck89] C. Beckmann, "The Effect of Scheduling and Barrier Synchronisation on Loop Performance," M.S. Thesis in progress, CSRD, Univ. of Illinois, June 1989.

[Bokh88] S. Bokhari, "Partitioning Problems in Parallel, Pipelined and Distributed Computing," *IEEE Transactions on Computers*, Vol. 37, No. 1, January 1988.

[BKPR87] K.C. Bowler, R.D. Kenway, G.S. Pawley, and D. Roweth, "An Introduction to Occam 2 Programming," Chartwell-Bratt Publishing Ltd, Sweden, 1987.

[BrGa89] M.C. Brunet and S. Gallopoulos, "Statistical and Deterministic Estimates of Stability," CSRD Tech. Rept., June, 1989.

[CGMW88] M. Chastain, G. Gostin, J. Mankovich, and S. Wallach, "The Convex C240 Architecture," *Proceedings of Supercomputing'88*, Orlando, Florida, November 14–18, 1988.

[ChCi87] H.–B. Chen and Y.–G. Ci, "Parallel Execution of Non–DO loops", Proceedings of the *1987 International Conference on Parallel Processing*, August 1987, pp. 512–516.

[Chen83] S. Chen, "Large–scale and High–speed Multiprocessor System for Scientific Applications – Cray–X–MP–2 Series," *Proc. of NATO Advanced Research Workshop on High Speed Computing, Kawalik(Editor)*, pp. 59–67, June 1983.

[ClGr88] K. Clark and S. Gregory. "Parlog: Parallel programming in logic," pp. 109–130 in *Parallel Computation and Computers for Artificial Intelligence*, J. S Kowalik Ed., Kluwer Academic Publishers, Boston, Mass. (1988).

[Coff76] E.G. Coffman, Jr., ed., *Computer and Job–shop Scheduling Theory*, John Wiley and Sons, New York, 1976.

[Cray85] "Multitasking User Guide," Cray Computer Systems Technical Note, SN–0222, January, 1985.

[Cytr84] R.G. Cytron, "Compile–Time Scheduling and Optimisations for Multiprocessor Systems," Ph.D. Thesis, Dept. of Computer Science, University of Illinois, Sept., 1984.

[DHML86] J. Davies, C. Huson, T. Macke, B. Leasure, and M. Wolfe, "The KAP/S–1: An advanced source–to–source vectoriser for the S–1 Mark IIa supercomputer," pp. 833–835 in *Proceedings of the 1986 International Conference on Parallel Processing*, IEEE Press, New York (1986).

". ti –225u ">=225 [DePa85]"
A.M. Despain and Y.N. Patt, "Aquarius–A High Performance Computing System for Symbolic/Numeric Applications," in COMPCON'85, IEEE Computer Society Press, Los Alamitos, CA, 1985.

[Diel86] H. Diel. "Parallel logic programming based on an extended machine architecture," pp. 15–30 in *Fifth Generation Computer Architectures*, J. V. Woods Ed., North–Holland, Amsterdam (1986).

[DoDu87] J.J.Dongarra and I.S. Duff, "Advanced Architecture Computers," Technical Report AERE R–12415, Harwell Laboratory, Oxon OX11 0RA, January 1987.

[DoSo87] J.J. Dongarra and D. C. Sorensen, "Schedule: Tools for Developing Parallel Fortran Programs," in *The Characteristics of Parallel Algorithms,* edited by L.H. Jamieson, D.B. Gannon, and R.J. Douglass, MIT Press, 1987.

[Fox87] G. C. Fox, "Domain Decomposition in Distributed and Shared Memory Environments. A Uniform Decomposition and Performance Analysis for the NCUBE and JPL Mark IIIfp Hypercubes," Proceedings of the *1987 International Conference on Supercomputing*, Springer–Verlag LNCS Vol. 297, February 1987.

[GaJG88] D. Gannon, W. Jalby, and K. Gallivan "On the Problem of Optimizing Parallel Programs for Hierarchical Memory Systems", Proceedings of the *1988 ACM International Conference on Supercomputing*, St. Malo, France, July 1988.

[GaJo79] M.R. Garey and D.S. Johnson, "Computers and Intractability, A Guide to the Theory of NP–Completeness," W.H. Freeman and Company, San Francisco, California, 1979.

[GaMc88] R. P. Gabriel and J. McCarthy. "Qlisp," pp. 63–90 in *Parallel Computation and Computers for Artificial Intelligence*, J. S Kowalik Ed., Kluwer Academic Publishers, Boston, Mass. (1988).

[GeAG88] E.F. Gehringer, J. Abullarade, M.H. Gulyn, "A Survey of Commercial Parallel Processors," ACM SIGARCH Computer Architecture Newsletter, Vol. 16, No. 4, Sept. 1988.

[GGKM83] A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer — Designing an MIMD Shared–Memory Parallel Machine," *IEEE Trans. on Computers,* Vol. C–32, No. 2, February 1983.

[GiPo88a] M. Girkar and C. D. Polychronopoulos, "Partitioning Programs for Parallel Execution," Proceedings of the *1988 ACM International Conference on Supercomputing*, St. Malo, France, July 4–8, 1988.

[GiPo88b] M. Girkar and C. D. Polychronopoulos, "Compiler Issues for Supercomputers," Technical Report CSRD No. 676, Center for Supercomputing Research and Development, University of Illinois, March 1988.

[Gokh87] M. B. Gokhale, "Exploiting Loop Level Parallelism in Nonprocedural Dataflow Programs", Proceedings of the *1987 International Conference on Parallel Processing*, August 1987, pp. 305–311.

[GPKK82] D. Gajski, D. Padua, D. Kuck and R. Kuhn, "A Second Opinion on Dataflow Machines and Languages," *IEEE Computer,* February 1982.

[Gupt89] R. Gupta, "The Fuzzy Barrier: A Mechanism for High Speed Synchronisation " of Processors," ASPLOS–III Proceedings, Boston, MA, April 3–6, 1989.

[Gusm88] A. Gusman. "AHR: A parallel computer for pure LISP," pp. 201–221 in *Parallel Computation and Computers for Artificial Intelligence*, J. S Kowalik Ed., Kluwer Academic Publishers, Boston, Mass. (1988).

[GPHL88] M. Gussi, D. Padua, J. Hoeflinger, and D. H. Lawrie, "Cedar Fortran and Other Vector and Parallel Fortran Dialects," *Proceedings of Supercomputing'88,* Orlando, FL, November 14–18, 1988.

[Hals86] R. H. Halstead, "Parallel Symbolic Computing," *IEEE Computer,* Vol. 19, No. 8, August 1986.

[Hals85] R. H. Halstead, Jr. "Multilisp: A language for concurrent symbolic computation," *ACM Trans. on Programming Languages and Systems,* Oct. 1985.

[Harr86] W. L. Harrison. *Compiling LISP for evaluation on a tightly coupled multiprocessor,* Rep. 565, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign (Mar. 1986).

[Hill85] W. D. Hillis. *The Connection Machine,* MIT Press, Cambridge, Mass. (1985).

[HiSN84] K. Hiraki, T. Shimada, and K. Nishida, "A Hardware Design for the Sigma-1, A Dataflow Computer for Scientific Computations," in *Proceedings of the 1984 International Conference on Parallel Processing,* August, 1984.

[Hoar78] C.A.R. Hoare, "Communicating Sequential Processes," *CACM,* Vol. 21, No. 11, 1978.

[Hwan87] K. Hwang, "Advanced Parallel Processing with Supercomputer Architectures," *Proceedings of the IEEE,* October, 1987.

[IBM88] IBM Parallel FORTRAN, Language and Library Reference Manual, IBM Corporation, March 1988.

[IKKR86] N. Ito, M. Kishi, E. Kuno and K. Rokusawa. "The dataflow-based parallel inference machine to support two basic languages in KL1," pp. 123–145 in *Fifth Generation Computer Architectures,* J. V. Woods Ed., North-Holland, Amsterdam (1986).

[Jaya88] D. N-M. Jayasimha, "Communication and Synchronisation in Parallel Computation," Ph.D. Thesis, Center for Supercomputing R & D, University of Illinois, August 1988.

[KaNa84] H. Kasahara and N. Seinosuke, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing," *IEEE Trans. Compt.,* Vol. C-33, No. 11, Nov., 1984.

[KaPS88] L. V. Kale, D. C. Sehr, and D. A. Padua. "OR-parallelism in standard sequential Prolog," forthcoming.

[Kenn80] K. Kennedy, "Automatic Vectorisation of Fortran Programs to Vector Form," Technical Report, Rice University, Houston, TX, October, 1980.

[KoCh87] J. S. Kowalik and K. M. Chalfan. "High-speed computing and artificial intelligence connection," *The International Journal of Supercomputer Applications,* 1(1) pp. 106–110 (Spring 1987).

[KKLW80] D.J. Kuck, R.H. Kuhn, B. Leasure, and M. Wolfe, "The Structure of an Advanced Vectorizer for Pipelined Processors," *Fourth International Computer Software and Applications Conference,* October, 1980.

[KKPL81] D.J. Kuck, R. Kuhn, D. Padua, B. Leasure, and M. Wolfe, "Dependence Graphs and Compiler Optimizations," *Proceedings of the 8-th ACM Symposium on Principles of Programming Languages,* pp. 207–218, January 1981.

[KDLS86] D. J. Kuck, E. S. Davidson, D. H. Lawrie, and A.H. Sameh, "Parallel Supercomputing Today and the Cedar Approach," *Science 231,* 4740 February 28, 1986, pp. 967–974.

[KrWe85] C. Kruskal and A. Weiss, "Allocating Independent Subtasks on Parallel Processors," *IEEE Trans. on Software Engineering,* Vol. SE–11, No. 10, October, 1985.

[Kuck78] D. J. Kuck, *The Structure of Computers and Computations,* Volume 1, John Wiley and Sons, New York, 1978.

[Kuck84] D.J. Kuck et. al., "The Effects of Program Restructuring, Algorithm Change and Architecture Choice on Program Performance," *International Conference on Parallel Processing,* August, 1984.

[Lawr75] D.H. Lawrie et. al., "Glypnir — A Programming Language for Illiac IV," *Communications of the ACM,* Vol. 18, No. 3, March 1975.

[Mann84] R. Manner, "Hardware Task/Processor Scheduling in a Polyprocessor Environment," *IEEE Trans. Compt.,* Vol. C–33, No. 7, July, 1984.

[MiPa87] S. P. Midkiff and D. A. Padua, "Compiler Algorithms for Synchronization," *IEEE Transactions on Computers,* Vol. 36, No. 12, December 1987.

[Nico84] A. Nicolau, "Parallelism, Memory Anti-Aliasing and Correctness for Trace Scheduling Compilers," Ph.D. Thesis, Yale University, June 1984.

[Paul82] G. Paul, "VECTRAN and the Proposed Vector/Array Extensions to ANSI FORTRAN for Scientific and Engineering Computation," Research Report RC 9223, IBM T.J. Watson Research Center, January 1982.

[PaWo86] D.A. Padua, and M. Wolfe, "Advanced Compiler Optimizations for Supercomputers," *Communications of the ACM,* Vol. 29, No. 12, pp. 1184–1201, December 1986.

[PaHK88] D. A. Padua, W. L. Harrison III, and D. J. Kuck, "Machines, Languages, and Compilers for Parallel Symbolic Computing," in *Biological and Artificial Intelligence Systems,* E. Clementi and S. Chin, Editors, ESCOM Science Publishers, 1988.

[Patt87] D. Patterson, "A Progress Report on SPUR," Computer Architecture News, Vol. 15, No. 15, February 1987.

[PfNo85] G. F. Pfister, V. A. Norton, "'Hot Spot' Contention and Combining in Multistage Interconnection Networks," *Proceedings of the 1985 International Conference on Parallel Processing*, St. Charles, IL, August, 1985.

[PoKu87] C. D. Polychronopoulos and D. J. Kuck, "Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers," *IEEE Transactions on Computers*, Vol. C-36, No. 12, December, 1987.

[Poly88] C. D. Polychronopoulos, *Parallel Programming and Compilers*. Kluwer Academic Publishers, Norwell, MA, August 1988.

[Poly89] C. D. Polychronopoulos, "Multiprocessing vs. Multiprogramming," *Proceedings of the 1989 International Conference on Parallel Processing*, St. Charles, IL, August 1989.

[PGHL89] C.D. Polychronopoulos, M. Girkar, M. Haghighat, C.-L. Lee, B. Leung, D. Schouten, "Parafrase-2: A New Generation Parallelising Compiler," *Proceedings of the 1989 International Conference on Parallel Processing*, St. Charles, IL, August 1989.

[Rash86] R. F. Rashid, "From RIG to Accent to Mach: The Evolution of a Network Operating System," Technical Report, Dept. of Computer Science, Carnegie-Mellon University, May 1986.

[Shap86] E. Shapiro, "Concurrent Prolog: A Progress Report," *IEEE Computer*, Vol. 19, No. 8, August 1986.

[ScKo86] R. G. Scarborough and H. G. Kolsky, "A Vectorising Fortran Compiler," *IBM Journal of Research and Development* **30**(2) pp. 163-171 (Mar. 1986).

[SATO83] S. Sugimoto, K. Agusa, K. Tabata, and Y. Ohno. "A Multi-Microprocessors for concurrent Lisp," pp. 703-710 in *Proc. of the 1983 International Conference on Parallel Processing*, IEEE Press, New York (1983).

[SSKM86] Y. Sohma, K. Satoh, K. Kumon, H. Masuzawa and A. Itashiki. "A new parallel inference mechanism based on sequential processing," pp. 3-15 in *Fifth Generation Computer Architectures*, J. V. Woods Ed., North-Holland, Amsterdam (1986).

[SBDN87] K. So, A. S. Bolmarcich, F. Darema, and V. A. Norton, "A Speedup Analyzer for Parallel Programs", Proceedings of the *1987 International Conference on Parallel Processing*, August 1987, pp. 653-662.

[ScGa85] K. Schwan and C. Gaimon, "Automatic Resource Allocation for the Cm* Multiprocessor," *Proc. of the 1985 International Conference on Distributed Computing Systems*, 1985.

[ShTs85] C. C. Shen and W. H. Tsai, "A graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using Minimax Criterion," *IEEE Trans. Compt.*, Vol. C-34, No. 3, March, 1985.

[Smit81] B. Smith, "Architecture and Applications of the HEP Multiprocessor Computer System," *Real Time Processing IV, Proc. of SPIE*, pp. 241-248, 1981.

[Ston77] H. S. Stone, "Multiprocessor Scheduling With the Aid of Network Flow Algorithms," *IEEE Trans. on Software Engineering,* Vol. SE–3, No. 1, Jan., 1977.

[TaYe86] P. Tang and P. C. Yew, "Processor Self–Scheduling for Multiple–Nested Parallel Loops," *Proceedings of the 1986 International Conference on Parallel Processing,* August, 1986.

[TYUY86] K. Toda, Y. Yamaguchi, Y. Uchibori, and T. Yuba. "Preliminary measurements of the ETL LISP–based data–driven machine," pp. 235–253 in *Fifth Generation Computer Architectures,* J. V. Woods Ed., North–Holland, Amsterdam (1986).

[ThCr88] R. H. Thomas and W. Crowther, "The Uniform System: An Approach to Runtime Support for " Large Scale Shared Memory Parallel Systems", Proceedings of the *1988 International Conference on Parallel Processing,* August 1988, St. Charles, IL.

[WaGa89] K.–Y. Wang and D. Gannon, "Applying AI Techniques to Program Optimization for Parallel Computers," in *Parallel Processing for Supercomputers & AI,* K. Hwang and D. DeGroot, Editors, McGraw–Hill series in supercomputing and parallel processing, McGraw–Hill, 1989.

[Warr87] D. H. D. Warren. "OR–parallel execution models of Prolog," pp. 243–259 in *Proc. of the 1987 International Joint Conference on Theory and Practice of Software Development,* Springer–Verlag, New York (1987).

[Wolf82] M. J. Wolfe, "Optimising Supercompilers for Supercomputers," Ph.D. Thesis, University of Illinois at Urbana–Champaign, DCS Report No. UIUCCDCS–R–82–1105, 1982.

[ZhYe84] C. Q. Zhu and P. C. Yew, "A Synchronization Scheme and Its Applications for Large Multiprocessor Systems," *Proc. of the 1984 International Conference on Distributed Computing Systems,* pp. 486–493, May 1984.

Parallel Execution of Fortran Programs on the EWS Workstation

M.C. Giboulot, E.R. Lebon, M. Loyer, H. Shafie

GIPSI S.A.
2, Bd Vauban B. P. 268
78053 St Quentin en Yvelines Cedex
FRANCE

&

F. Thomasset

INRIA
Domaine de Voluceau-Rocquencourt B. P. 105
78150 Le Chesnay Cedex
FRANCE

1 Presentation of EWS

EWS (EuroWorkStation) is an ESPRIT II project. Its objective is the creation of a High Performance Technical Workstation.
The members of the project are Siemens (D), BULL (F), CHORUS (F), GIPSI s.a (F), Grupo APD s.a (E), Rutherford Appleton Laboratories (GB), INRIA (F), INESC (P), FhG-AGD of Darmstadt University (D) and Brunel University (GB).

For our part, we realize a Multi-SPARC Workstation (with 4 processors) that can be considered as

- a Basic Part with one SPARC CPU, which will run standard ABI applications,
- a Computational Extension (with 3 other SPARC processors). Extra computational power will be obtained by generating a code concurrently run by the 4 processors that will synchronize and share special data (contained in so-called tagged cells) via a dedicated bus which bypasses memory accesses. To generate code dedicated to that architecture, we will customize an existing SPARC Compiler.

The MultiSPARC architecture will allow EWS to benefit from evolutions in the SPARC family.

An aspect of our participation is to exploit parallelism on this architecture ; this article focuses on this aspect and more precisely on the parallel execution of Fortran programs.

2 Parallelization of Fortran Programs

2.1 Tools for Automatic Parallelization

Many Fortran dialects and many Fortran program restructuring tools flourished these last years along with the wave of parallel processors.

Although one may claim that making sequential Fortran programs run efficiently on a parallel architecture is easy, a simple test can outline the difficulty of the job on actual architectures [3].

On the other hand, designing a Parallel Fortran common to the various parallel architectures is still a "work in progress" [5].

Our objectives were not to design another parallel Fortran dialect but design and implement tools for transforming sequential Fortran applications into parallel Fortran applications and allow them to run on a Multi-SPARC workstation.

In the EWS Project, we intend to realize a Parallelizer and a Parallel Fortran Compiler.

- The Parallelizer analyses the DO loops of a Fortran program to detect parallelism *independently* on the architecture of the machine.

  The output of the Parallelizer will be a Fortran Program where sequential and parallel regions are distinguished and where synchronization barriers are minimized by using techniques similar to the ones presented in [1]. BULL is in charge of the realization of the interactive parallelizer.
- The Compiler uses the results of the Parallelizer to implement parallel execution on a Multi-SPARC architecture. GIPSI is in charge of the realization of Parallel Fortran compiler

This work is done in cooperation with the ESPRIT project GIPE which builds an interactive programming environment for Fortran.

2.2 Output of the Parallelizer

Currently, as a parallelizer, we use a prototype issued from VATIL which is an automatic vectorizer created by INRIA [4]. This vectorizer was written in LeLisp [2] and used for various targets including the vector extension of DPX-1000 ; it uses a symbolic analysis of dependences which was partly described in [4], allowing full use of information given by the programmer. With respect with [4], the method for computing dependences has been changed, so that integer programming problems are solved with the help of a simplex method ; the resulting heuristics yields fairly reasonnable times for dependence computation.

The output of the Parallelizer is a Fortran source where parallelizable Fortran DO loops are transformed into a list of parallel and sequential DO loops. There are two possibles levels of output :

- one called *user-level output* for interaction with the user, displaying the distribution of the original loop into parallel and sequential loops, as well as the constraints imposed on execution by dependences ; we plan to modify the syntax of this output towards PCF Fortran [5].
- an other called *compiler-level output* to be read by the compiler ; this compiler-level ouput contains the same informations as the user-level output but with mechanisms dedicated to the execution on EWS.

For example, with the input :

```
      dimension x(100), y(100), z(100)
      dimension t(100), w(100)
      dimension u(100), s(100)

      do 1 i = 1,n
         x(i) = y(i) + z(i)
         w(i) = x(i+1) * w(i)
         y(i) = w(i)+y(i+1)
         u(i) =  w(i)+u(i+1)
         z(i) = 3 * y(i)
         t(i) = t(i)+w(i)
         s(i)= s(i-1)+u(i)*y(i)
 1       continue
       end
```

The parallelizer produces the following user-level output (the compiler-level output will be shown in the forthcoming paragraphs) :

```
C$PARALLEL REGION

C$PARALLEL DO 1
C$DIRECTIVE LOCAL={i}
       do 1 i=1,n
       w(i) = w(i)*x(1+i)
       t(i) = t(i)+w(i)
 1     continue

C$PARALLEL DO 2 AFTER 1
C$DIRECTIVE LOCAL={i}
       do 2 i=1,n
       x(i) = y(i)+z(i)
 2     continue

C$SEQUENTIAL DO 3 AFTER 2
C$DIRECTIVE LOCAL={i}
       do 3 i=1,n
       y(i) = w(i)+y(1+i)
       u(i) = u(1+i)+w(i)
       s(i)= s(i-1)+u(i)*y(i)
 3     continue

C$PARALLEL DO 4 AFTER 3
C$DIRECTIVE LOCAL={i}
       do 4 i=1,n
       z(i) = 3*y(i)
4      continue

C$END PARALLEL REGION
```

In a PARALLEL DO loop, all the iterations of the loop are independent and can be executed in any order without any synchronization.

In a SEQUENTIAL DO loop, there are dependences between the iterations and the iterations must be done sequentially.

2.3 Extensions to Fortran 77

The compiler-level ouput of the preprocessor is expressed in an extended Fortan 77 where the main extensions are :

- **local variables** : these are variables local to a section of code, and instantiations of that section of code running concurrently have their own instantiation of these variables,
- **reduction variables** : these variables must be acceded by sections running concurrently with mutual exclusion,
- **multithreading operations** : these operations allow several processors to cooperate in the execution of a parallel region ; they are done via subroutine calls which will be inlined by the compiler into SPARC instructions.

2.4 Impact on the Fortran Compiler

We decided to adapt the Sun 4 Fortran Compiler and to minimize our interventions into the compiler by using dedicated pre- and postprocessors ;

We made that choice because many SPARC compilers are in development and we hope that this strategy will allow us to adapt our works on any of these compilers at a low cost.

The two main interventions done in the compiler are :

- protect the code associated with parallel regions from optimizations, meaningful for a mono-SPARC but erroneous for a Multi-SPARC,
- generate specific code which controls access to tagged cells.

2.5 Parallel Execution of Fortran Programs

2.5.1 Mechanisms for Communication and Synchronization

- A **task** is an execution environment and the basic unit of resource allocation. A task includes a virtual address space for code and data. A Fortran program is a task.
- A **thread** is the basic unit of execution. It consists of all processor states necessary for independent execution (e.g. hardware registers). A thread executes on the virtual address space of a task. A SEQUENTIAL DO loop will be executed on one thread and a PARALLEL DO loop will be split into several threads.
- A **cell** is the basic mechanism for communication and synchronization between threads. A cell contains a data plus a tag which indicates whether the data is available[1] or not. The tag of a cell has the same value, at any time, for all the processors. The cells are implemented in a special memory space called the synchronization space.

There are three atomic operations on a cell :

- **LOCK**
  syntax : `d=LOCK(cell)`
  semantics :

```
wait until tag(cell) == available
```

[1] "available" means "readable at that moment".

```
d=value(cell)
tag(cell)=unavailable
```

- **UNLOCK**
  syntax : `call UNLOCK(cell,d)`
  semantics :

```
value(cell)=d
tag(cell)=available
```

- **GET**
  syntax : `d = GET(cell)`
  semantics :

```
wait until tag(cell) == available
d=value(cell)
```

Remarks :

1. Each atomic operation is inlined into one RISC instruction.
2. The operations **LOCK** ... **UNLOCK** permit an access with mutual exclusion to data and sections of code.
3. An **UNLOCK** not preceeded by a **LOCK** means an initialization of a cell (this usually occurs in a monoprocessor section just before entering a multiprocessor section).
4. An access via **GET** to a cell means an access to a volatile value of the cell.

2.5.2 Execution of a Parallel Region

The output of the Parallelizer is organized in regions ; some of them are parallel regions which can be concurrently executed by severals processors ; the other regions will be executed by only one processor which is called the main processor.

While the main processor is running a non parallel region, the other processors are idle waiting for the address of the next parallel region to be run concurrently. That address will be posted in a special cell called `multi` by the main processor.

The run-time context of a processor is divided in two parts :

- a general context which is shared by all the processors and composed of the executable code and the data in memory and cells,
- a private context which is specific to each processor and composed of its registers and its stack.

The distribution of the execution on the several processors brings about data transfers between the private context of processors. One of the tasks of the code generation is to minimize these tranfers and to correctly initialize the private region of each processor.

A parallel region is a sequence of PARALLEL and SEQUENTIAL DO loops to which are associated threads that constitute the minimal execution unit that processors concurrently try to execute.

To a SEQUENTIAL DO loop is associated one thread ; to a PARALLEL DO loop are associated several threads.

To each loop of a parallel region is associated a cell, the value of which indicates whether the loop

is ready to be executed or not, and, in the former case, which threads are to be executed if it is a PARALLEL loop.

All the processors concurrently run the code of a parallel region and test the cells to determine the threads to be executed. The compiler uses autoscheduling techniques inspired from [6] and generates code that provides autoscheduling of the processors without any system call.

The outlines of the main pieces of code are :

code run by the main processor :

```
      call INITMULTI(nb_cell,nb_proc)
C Initialization of the cells associated to the loops of the region
C with the UNLOCK operation
        ...
C End of initialization
      call MULTIREGION(region,..)
```

where

```
     subroutine INITMULTI(nb_cell,nb_proc)
C nb_proc is the number of processor offered by the machine
 99   if (GET(nextmulti .neq. 0) goto 99
      UNLOCK(nextmulti,nb_proc-1)
C nextmulti : when the value of this cell is 0, its means that all the
C             auxilliary processors are ready for the execution of the
C             next multiregion.
      UNLOCK(endmulti,nb_proc)
C endmulti : when the value of this cell is 1, its means that all the
C            auxilliary processors have finished the execution of the
C            current multiregion.
     end INITMULTI

     subroutine MULTIREGION(region,..)
C storage of the parameters in exchange zone (tagged cells)
        ...
C End of the storage
      UNLOCK(multi,@region)
      call @multi
 99   if (GET(endmulti) .neq. 1) goto 99
      LOCK(multi)
      n = LOCK(endmulti)
      UNLOCK(endmulti,n-1)
     end MULTIREGION
```

code concurrently run by all the processors :

```
     subroutine region(...)
C    stop is a cell, the value of which  becomes equal to 1
C    when all the threads have been selected

9999
        ...         --- PARALLEL and SEQUENTIAL loops
     if(GET(stop).le.0) goto 9999
     end region
```

code run by the auxilliary processors :
This code is written in SPARC assembly language ; a Fortran-like version is :

```
999     addr=GET(multi)
           ...      --- initialization of the private context
                    --- from dedicated cells
        call @addr
        n=LOCK(endmulti)
        call UNLOCK(endmulti,n-1)
 99     if (GET(endmulti) .neq. 0) goto 99
        n=LOCK(nextmulti)
        UNLOCK(nextmulti,n-1)
        goto 999
```

2.5.3 Algorithms to generate synchronization between the loops of a region

One must keep in mind that :

- the code between C$PARALLEL REGION and C$END PARALLEL REGION will be run concurrently by the different processors allocated to the Fortran program,
- a SEQUENTIAL DO loop is associated one thread,
- a PARALLEL DO loop is split into several threads ; this number of threads is not bounded by the number of processors allocated to the Fortran program ; the threads of a PARALLEL DO loop are independent of each other,
- the whole synchronization is realized via access to cells and without any system call.

Region with only SEQUENTIAL DO loops

Let us consider the previous example where the PARALLEL DO loops are considered as SEQUENTIAL DO loops :

```
C$PARALLEL REGION

C$SEQUENTIAL DO 1
    ...

C$SEQUENTIAL DO 2  AFTER DO 1
    ...

C$SEQUENTIAL DO 3  AFTER DO 2
    ...

C$SEQUENTIAL DO 4  AFTER DO 4
    ...

C$END PARALLEL REGION
```

The compiler transforms the region following the hereafter algorithm :

- A cell is associated with each DO loop[2].
- If a loop is independent, its cell is initialized to 1.
- If the loop `DO n` depends on p other DO loops, the cell associated to `DO n` is initialized to $1 - p$.
- There is an extra cell called `stop` which is initialized to a value equal to minus the number of leaf-loops plus one. This cell is decremented every time a leaf-loop completes.
- When a loop completes, the cells of the loops depending on that loop are incremented by 1 via a multithreading operation `SIGCHORE(cell)`.
- When the value of a cell becomes equal to 1, the associated loop becomes eligible.
- When a loop is selected, its cell is reset to 0.
- The test of the eligibility and the loop selection is done via a multithreading operation `ISREADY(cell)`.

All the processors allocated to the task (i.e. the Fortran program) try concurrently to execute the threads (i.e the eligible DO loops). When `stop` becomes equal to 1, all the processors but one become idle waiting for the next region to be concurrently executed.

For example, the loop `DO 3` will be changed into

```
C$SEQUENTIAL DO 32
      if (ISREADY(3) .gt. 0) then
           do 32 i=1,n
      ...
32         continue
    call SIGCHORE(4)
endif
```

The multithreading operations are written in SPARC assembly language and inlined. Their meaning – in Fortran – is :

```
function ISREADY(cell)
  integer isready
  isready= LOCK(cell)
  if (isready .gt. 0) then
       call UNLOCK(cell,isready-1)
    else
       call UNLOCK(cell,isready)
  endif
  return isready
end
```

and

```
subroutine SIGCHORE(cell)
  integer status
  status=LOCK(cell)
```

[2]In all the examples, the cell p is associated to the "do p" loop.

```
      call UNLOCK(cell,status+1)
    end
```

Region with only a PARALLEL DO loop

A PARALLEL DO loop is split into several threads to be concurrently executed by processors allocated to the task. There are different strategies to fix the number of threads and the scheduling of the iterations between the threads.
If the loop is a genuine parallel loop, there are two strategies to fix the number of threads :

- If one knows that the computing time is approximately the same for each iteration of the loop (e.g. a loop with no conditional instruction), then the number of threads p will be set to the number of processors allocated to the task and each thread will execute n/p iterations.
- If one knows that the computing time varies significantly between the n iterations, a better balance of the work between the processors executing the threads will be obtained if there are more threads executing less iterations.

In both cases, each thread executes a chunk of consecutive iterations (the size of a chunk depends on the number of iterations and on the number of threads executing concurrently the loop).
The genuine parallel loop becomes :

```
C$PARALLEL DO 1
      status=READY(1)
      if (status .gt. 0) then
            myfirst=1+chunk*(status-1)
            mylast= min(myfirst+chunk-1,n)
            do 1 i=myfirst,mylast
              ...
1           continue
      endif
```

Remark :
In some cases, the number of threads is imposed by dependences between iterations. For example, if we change the instruction of the loop DO 2 into :

```
x(i)= y(i)+z(i)+x(i-3)
```

then this loop is parallelizable if and only if 3 threads execute one iteration every 3 ones.
So the Parallelizer will indicates :

```
C$ PARALLEL DO 2 AFTER DO 1 - THREAD=3
```

This number of threads imposed by dependences between iterations is not limited by the number of available processors.
We will call this kind of loop pseudo parallel loop because the iterations are not independent and the parallelism is obtained via an artefact.

A pseudo parallel loop must be split into p threads and each thread executes one iteration every p iterations.
The pseudo parallel loop becomes :

```
C$PARALLEL DO 2 -THREAD=p
        status=ISREADY(2)
        if (status .gt. 0) then
              do 1 i=status,n,p
                ...
 1            continue
        endif
```

Region with SEQUENTIAL and PARALLEL DO loops

The algorithm for transforming such a region is a generalization of the former algorithm used for regions with only SEQUENTIAL DO loops :

- A cell is associated with each DO loop.
- If a loop is independent, its cell is initialized to the number of its associated threads.
- If the loop `DO n` depends on p other threads, the cell associated to `DO n` is initialized to $1 - p$.
- There is an extra cell called `stop` which is initialized to a value equal to minus the number of leaf-threads plus one. That cell is decremented every time a leaf-thread completes.
- When a thread completes, the cells of the loops depending on that thread are incremented by 1.
- When a loop becomes eligible, if that loop is a PARALLEL loop then its cell is set to the number of threads into which the loop is split and if it is a SEQUENTIAL loop it is set to 1.
- Each time a loop is selected, its cell is decremented by 1 and a thread is executed.

This implies the introduction of an other multithreading operation : to this effect we add an optional parameter to `SIGCHORE(cell)` with default value 1 ; this represents the number of threads associated to the loop when it becomes eligible. Its semantics are :

```
subroutine SIGCHORE(cell,value)
  integer status
  status=LOCK(cell)
  if (status .lt. 0) then
        call UNLOCK(cell,status+1)
    else
        call UNLOCK(cell,value)
  endif
end
```

Note that when value = 1, `SIGCHORE(cell,1)` has the same meaning as the previously given semantics for `SIGCHORE(cell)`, because `istatus` can never become strictly positive.

A Comprehensive Example

Let the result of the Parallelizer be :

```
C$PARALLEL REGION

C$PARALLEL DO 1
    ...

C$PARALLEL DO 2  AFTER DO 1 ~ THREADS = 3
    ...

C$SEQUENTIAL   DO 31 AFTER DO 2
    ...

C$SEQUENTIAL   DO 32 AFTER DO 1
    ...

C$SEQUENTIAL   DO 33 AFTER DO 31, DO 32
    ...

C$PARALLEL DO 4  AFTER DO 31
    ...

C$END PARALLEL REGION
```

Each SEQUENTIAL DO loop will have one thread.
The loop DO 2 will be split into 3 threads.
The loops DO 1 and DO 4 will be split into 4 threads.
The whole region will become :

```
C$PARALLEL REGION
      call UNLOCK(1,4)
      call UNLOCK(2,-3)
      call UNLOCK(31,-2)
      call UNLOCK(32,-3)
      call UNLOCK(33,-1)
      call UNLOCK(4,0)
      call UNLOCK(stop,-4)
      call INITMULTI(region,...)
C$END PARALLEL REGION

      subroutine region(....)
C$LOCAL   i, status, myfirst, mylast, chunk
        ...
      chunk=(n-1)/4 + 1
9999  continue
C$PARALLEL DO 1
      status=ISREADY(1)
      if (status .gt. 0) then
          myfirst=1+chunk*(status-1)
          mylast=min(n,myfirst+chunk-1)
          do 1 i= myfirst,mylast
           ...
    1     continue
          call SIGCHORE(2,3)
```

```
        call SIGCHORE(32)
      endif
C$PARALLEL DO 2 AFTER DO 1 - THREAD=3
      status=ISREADY(2)
      if (status .gt. 0) then
        do 2 i=status,n,3
          ...
    2   continue
        call SICHORE(31)
      endif
C$SEQUENTIAL DO 31 AFTER DO 2
      if (ISREADY(31) .gt. 0) then
        do 31 i=1,n
          ...
   31   continue
        call SIGCHORE(4,4)
        call SIGCHORE(33)
      endif
C$SEQUENTIAL DO 32 AFTER DO 1
      if (ISREADY(32) .gt. 0) then
        do 32 i=1,n
          ...
   32   continue
        call SIGCHORE(33)
      endif
C$SEQUENTIAL DO 33  AFTER DO 31, DO 32
      if (ISREADY(33) .gt. 0) then
        do 33 i=1,n
          ...
   33   continue
        call SIGCHORE(stop)
      endif
C$PARALLEL DO 4  AFTER DO 31
      status=ISREADY(4)
      if (status .gt. 0) then
        myfirst=1+chunk*(status-1)
        mylast=min(n,myfirst+chunk-1)
        do 4 i= myfirst,mylast
          ...
    4   continue
        call SIGCHORE(stop)
      endif
      if (GET(stop) .le. 0) goto 9999
      end
```

3 Conclusion

We have completed a prototype version of the parallel compiler where the multithreading is implemented via lightweight processes on a SUN 4. This version allows us to check the results of the parallelizer and of the compiler.

During the second quarter of 1990, a MultiSparc EWS prototype will be available to evaluate our multithreading mechanisms on actual hardware.

We also plan to extend our multithreading mechanisms to the parallelism defined in PCF Fortran

and to design a parallelizer that takes as input Fortran 77 programs and produces PCF Fortran programs.

References

[1] R. Allen, D. Callahan, K. Kennedy *"Automatic Decomposition of Scientific Programs for Parallel Execution"* ACM, Principles of Programming Languages (1987).

[2] J. Chailloux, M. Devin, F. Dupont, J. M. Hullot, B. Serpette, J. Vuillemin *"LeLisp: Version 15.2, Manuel de Référence"* INRIA, Nov. 1988 ().

[3] A. H. Karp, R. G. Babb II *"A Comparaison of 12 Parallel Fortran Dialects"* IEEE Software (Sept. 1988).

[4] A. Lichnewsky, F. Thomasset *"Introducing Symbolic Problem solving Techniques in the dependence testing Phases of a Vectorizer"* International Conference on Supercomputing, ACM (June 1988).

[5] The Parallel Computing Forum *"PCF Fortran: Language Definition - Version 1"* (Aug. 1988).

[6] C. D. Polychronopoulos *"Toward Auto-scheduling Compilers"* The Journal of Supercomputing, 2, 297–330 (1988).

Sparse Matrix Computations on Supercomputers*

Harry A.G. Wijshoff
Center for Supercomputing Research and Development
University of Illinois at Urbana Champaign
Urbana, Illinois 61801
U.S.A.

Abstract

In this paper some of the key issues are described for the implementation of efficient sparse computational codes on supercomputers. Whereas dense computations run mostly near the peak performance of new architectures, for sparse computations this is certainly not true. Sparse matrix computations are characterised by the relative small number of operations per data element and the irregularity of the computation. Both facts may significantly increase the overhead time due to memory traffic. Further the development of sparse code is far from trivial. The use of sophisticated data structures together with complex control flow makes designing sparse codes an almost unmanageable task.

1 Introduction

A major difficulty associated with sparse matrix computations is that the relation hardware-algorithm is complex because of the random nature of the computations. Sparse matrix computations are characterized by several features: (i) complex data handling, (ii) irregular data streams, (iii) indirect addressing, and (iv) a low ratio of arithmetic operations to data element references. The first feature is caused by the fact that sparse matrices are stored in a condensed format in order to minimize the storage requirements,

*This work was supported in part by the National Science Foundation under Grant No. US NSF CCR-8717942, and the US Department of Energy under Grant No. US DOE DE-FG02-85ER25001.

which leads to substantial overhead. The second leads to lower effective memory bandwidth, the third complicates vectorization and parallelization of the sparse matrix codes, and the fourth can lead to excessive data movement and loss of data locality.

As a result the performance of sparse matrix computations is dictated by the ability of a specific architecture to support high and irregular data traffic between computational elements and memory. In fact the common assumption that long-running numerical application software is CPU-limited [11] is questionable. Super/Parallel Computers today are very subtly tuned for matching the computational cycle with memory access cycle. This results in very good performance of regular computations in which not too many data streams from and to memory are involved. It seems likely that this trend will continue with future systems becoming even more dependent on regular data transfers. It appears that computational kernels in which there are at most three regular data streams involved, can be exploited fairly well by most of the currently marketed supercomputers. However, in some cases, one can detect a significant decrease in performance when there are three or more data streams instead of just two data streams per operation [14].

Not only is the performance of architectures stressed by sparse matrix computations; they also constitute a major problem for restructuring compilers and program environments. As a major part of loop structures in sparse codes employ indirect addressing (subscripted subscripts), data dependencies can mostly not be determined at compile time. The only known restructuring techniques are based on an at runtime evaluation of the indirection arrays [13,16]. The evaluation of these indirection arrays is too costly, however, if the number of operations performed on the array elements is small. For a detailed account for the performance tradeoff of these techniques see [13]. For some instances of sparse codes the overhead due to runtime evaluation of the indirection arrays can be nullified. This is in particular true for iterative methods for solving linear systems of equations. As these methods involve a series of matrix-vector multiplies (and possibly triangular solves) where the sparsity structure of the matrix does not change from one iteration to the other, the evaluation has only to be performed once.

The development of scientific libraries for efficient sparse matrix computations codes is a major effort. This is not only due to the fact that the various different architectures require very different techniques in order to be utilised efficiently, but also most existing sparse matrix computation codes cannot be viewed as consisting out of a number of higher level computational

primitives. After all the components of a sparse matrix computation code interact strongly with each other. For instance, in a direct solver for systems of linear equations, the handling of fill-in determines the storage format of the matrix which directly influences the reordering used in the code, and the choice of a pivoting strategy influences both the fill-in handling as the reordering. This directly implies that, if higher level primitives can be identified, these primitives need to be implemented in various ways in order to be utilized in different code instances.

Another major problem formed by sparse computations is the complexity of code development. This is mainly caused by the fact that the use of complicated condensed formats for storing the sparse matrices forces the programmer to implement explicitly garbage collection routines to free space, and routines to reformat data structures. Secondly, the exploitation of parallelization and vectorization involves often sophisticated preprocessing code. To illustrate this, in the following table we counted for two sparse Fortran codes the total number of lines of code and the number of lines in the code in which floating point operations are performed.

	# of actual lines	# lines with fl. point operations
Ma28	1826	28
McSparse	4947	30

Ma28 is a commonly used package for solving general sparse systems of linear equations developed at Harwell [2]. McSparse is a package for solving these systems which exploits different levels of parallelism and hierarchical memory systems [7,19]. As can been from this table the number of lines which contain floating point instructions is negligible. This does not directly imply that the amount of time spend performing these floating points instructions is negligible. However, it shows that most of the effort of developing these code is spend in the non-computational part of the code.

The above described difficulties are less of an issue for structured sparse computations. Structured sparse matrices arise mainly from the numerical handling of partial differential equations by either finite element or finite differences techniques. These matrices differ from general sparse matrices in the sense that mostly a diagonal storage scheme can be used, which can be exploited to obtain longer vector operations. Also irregular data streams do not occur as frequently as for general sparse matrices which alleviates the constraints on the effective memory bandwidth. Further, complex data manipulations are not needed. Because of this, the reminder of the paper deals

mainly with unstructured sparse computations. In the next two sections some specific solutions for the implementation of direct methods and iterative methods are described for solving general sparse systems of equations on parallel/super computers.

2 Direct Solvers for Sparse Linear Systems

Implementations of direct solvers for general sparse matrices on vector and parallel architectures mostly perform very poorly. This is mainly caused by the fact that the exploitation of parallelism and vectorization in such codes is strongly dependent on the sparsity structure of the matrix A. In the case that the sparsity pattern of the matrix A is arbitrary, it is almost impossible to exploit any parallelism/vectorization a priori. There are essentially three different techniques for generating parallelism in direct solvers for general linear systems of equations. The first technique exploits parallelism at the level of the rank-1 update. Secondly multiple rank-1 updates can be performed in parallel. This mostly requires a search for a set of diagonal or triangular pivots. Thirdly a global ordering (tearing technique) can be used to decompose the sparse matrix into blocks which can be factored in parallel.

Another technique, which is recently being used to obtain efficient implementations of these codes, is based on identifying sub-systems which are sufficiently small and have a reasonable amount of fill-in. These sub-systems can be treated as dense matrices and dense factorizations can be used which are mostly much more efficient than sparse factorizations. A good example of this development is given by the *multifrontal* code [4]. In this code frontal matrices are assembled by examining the elimination tree. However, the efficiency of the multifrontal approach is degraded if the pattern of the sparse matrix is far from symmetric.

Although tearing techniques are mostly considered for introducing large grain parallelism in a sparse solver, they also identify subsystems in which the amount of fill-in is reasonable large. Because tearing techniques bring a sparse systems into bordered upper triangular block form, fill-in will be concentrated to the diagonal blocks and the border. This is in particular true for the "coupling block", which in most cases will get near dense during the factorization. In the remainder of this section a direct solver, McSparse, is being discussed which is currently being developed at CSRD. The underlying idea for developing this solver was to obtain a sparse code which would exploit different levels of parallelism. As new architectures are getting more

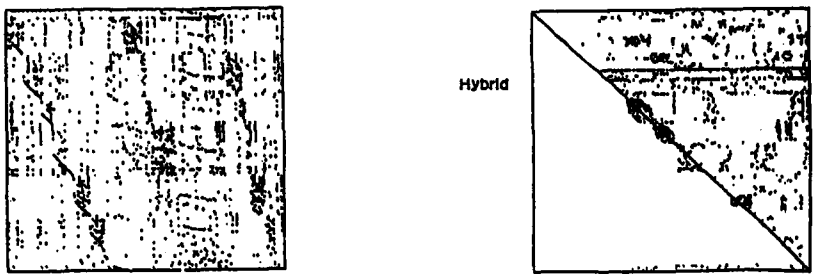


Figure 1: The ordering H*

and more hierarchically structured it will become very important in the near future to device codes which allow more than one form of parallelism to be utilized. The Cedar architecture [10] was mainly used to develop a first version of McSparse. Currently the code is being ported onto a Cray 2 and a Cray YMP.

McSparse is based on finding large grain parallelism such that the factorisation of a general sparse matrix A can be divided up into partitions which allow the clusters or processors to work in parallel on the problem. (On Cedar, finer granularity parallelism can be exploited within a cluster.) Large grain parallelism is obtained by ordering the matrix by a hybrid ordering H* which is related to tearing techniques for nonsymmetric matrices [5]. Tearing techniques, however, are based mostly on nonsymmetric orderings (column and row interchanges). The H* algorithm differs in that it uses an initial nonsymmetric ordering (H0) and subsequent symmetric orderings to achieve the desired form. The H* ordering produces a bordered block upper-triangular form. The ordering is a hybrid form combining *Tarjan's algorithm* [17], the H1 ordering which is derived from *Tarjan's algorithm*, and the H2 ordering which is based on a modified form of *Nested Dissection* [8] that exploits the nonsymmetric structure of the matrix. Additionally, H* tries to enhance the stability of the pivoting strategies used in the factorization algorithm. This is accomplished via three techniques. First, the H0 phase generates a transversal such that the element on the diagonal for each column is within a bound of the largest element within the column. In the H1 and H2 phases, two techniques are used which monitor the size of the elements placed in the border. In figure 1 a sparse matrix (bp_800 from the Harwell/Boeing collection [3]) is depicted before and after being reordered by H*.

The factorization and solution of the partitioned system can exploit the

above ordering in several ways. These include modifications to: a standard sparse LU decomposition, a block LU decomposition, a decomposition combined with a low rank update such as the Woodbury formula [9], and iterative methods which may use the decomposition as a preconditioner. For the moment only attention is paid to the modifications of the standard sparse LU decomposition to exploit the form effectively and generate an accurate solution.

The implementation of the solver first performs the H* ordering on a single cluster and then distributes the matrix to the other clusters and places certain portions of the border in the global memory to be shared by the clusters. The clusters then compute the LU decomposition of each diagonal block. This factorization can be done using dense techniques suitable for the cluster's architecture (BLAS3-based partial pivoting) or a parallel sparse factorization routine which exploits finer grain parallelism. Of course, the diagonal blocks which result from H* may not be nonsingular or well-conditioned. This is handled by either artificially forcing nonsingularity or by dynamically redefining the border during the factorization. In the latter technique the unknowns which cause difficulty are cast into the border and eliminated at a later stage. After these factorizations are completed the off-diagonal entries in the upper triangular part of the matrix are updated. The entries in the border are then updated. This update may exploit the hierarchical structure of the border depending upon its size. Finally, the diagonal block coupling the border elements is factored. This is typically dense but may be performed on a single cluster or multiple clusters depending on its size. The forward solve is partitioned naturally as a result of the factorization. However, during the factorization phase the U matrix is redistributed to allow an efficient backward solve (which may be done repeatedly if coupled with an iterative method).

3 Sparse Basic Linear Algebra Subroutines for Iterative Methods

As was already mentioned in the introduction, identifying computational primitives for sparse codes might not always be possible, however, for iterative methods these primitives are easy to identify. Recently many papers appeared which describe efficient implementations for triangular solves and matrix-vector multiplies [1,6,15]. In this section a systematic way is described how to obtain efficient implementations for these computational

kernels on a vector/concurrent architecture.

Sparse computations are characterized by the intrinsic complexity of the data handling. In order to cope with the complexity of data handling, the design of sparse BLAS primitives should encompass both the data manipulation capabilities of the architecture as well as the requirements imposed by the sparse computation. This is achieved by essentially taking the following four steps:

- Defining the suitable Data Access Types
- Handling Data Locality
- Handling the Irregularity of the Computation
- Handling Parallelism.

We will demonstrate this for the Sparse Matrix A times Dense Matrix(or Vector) B primitive (SpMxM(V)). The primitive SpMxM derives, for instance, from an iterative method to obtain the eigenvectors of a sparse matrix. At each iteration the iteration matrix is multiplied with the approximates of the eigenvectors. The primitive SpMxV, which is a special case of the former one, is the crucial component with respect to performance in most iterative solvers.

In vector/concurrent architectures, e.g. CEDAR, Alliant FX series, Cray series, there are essentially two different types of data access: vector access and scalar access. For the primitive SpMxM we can think of two possible ways of realising these vector accesses. One realization is based upon the row/columns of matrix A and/or B, and the other realization is obtained by extending each row (column) of A to a full row (column) by shifting all the non-zero entries of A to the top (to the right). The latter extended rows (columns) are also called jagged diagonals, generalised columns and stripes, see [6,12,14]. A represents the sparse matrix and B the dense matrix throughout this section. The following table depicts which combinations of these accesses makes sense for the implementation of SpMxM:

A / B	scalar	row	column	ext. row	ext. column
scalar	X		X		
row	X				
column		X			X

The scalar-A/scalar-B version is certainly implementable but does not exploit the vector capabilities of the architecture under investigation. This leaves us with four different types of implementation for SpMxM.

For multiprocessors with an hierarchical structured memory system care has to be taken with respect to data locality. By this we the ability to keep data in the highest level of a system, e.g., vector registers or cache, for as long as possible during a computation. The data locality of a computation is largely determined by the time delay between re-usage of data. The following strategy is used to optimize data handling in these architectures.

1. **Vector Register** utilization: reduction of the number of data streams to be accessed by each CE.
2. **Cache** utilization: reduction of the length of the data-streams.

The reduction of the number of data streams is obtained by keeping each operand as long as possible in a vector register during the computations. For each of the above mentioned four versions this amounts into two variants. For instance, for the scalar-A/row-B version, either, each row of A can be kept in a vector register for as long as possible, or each row of the result matrix. The number of data streams can be even further decreased by applying a blocking technique. By blocking we mean that the innermost loop of a nested loop is not iterated for a maximal number of times, but only in chunks of a certain length. The following table shows the reduction of the number of datastreams for each of the eight versions.

Version # Datastreams	1A	1B	2A	2B	3A	3B	4A	4B
Originally	3	3	3	3	4	4	5	5
Before Blocking	2	2	1	3*	2	4*	3	3*
After Blocking	1*	1*	1*	1*	2*	2*	1*	1*

The entries in this table which are suffixed by a * indicate a non-optimal reduction of the data streams caused by the occurrence of indirect addressing.

The length of the data streams can be reduced by again applying a blocking technique. This blocking technique is applied in the same way as the above mentioned, but its functionality is quite different. Whereas both blocking techniques try to decompose a DO-loop in chunks of a certain length, the first mentioned blocking is constrained to the vector processing

capabilities of an arch'tecture. In order to distinguish the two forms of blocking we call the latter one vertical blocking.

The use of condensed storage formats for the sparse matrix can have different impacts on the 8 implementations. First indirect addressing can be caused in the innermost loop body. Secondly the loop boundaries can become dependent on the structure of the matrix, and, thirdly, the length of the vector operations may be affected. If we summarize all the effects of using a condensed storage format for the sparse matrix, we obtain the following table:

Version / Effect	1A	1B	2A	2B	3A	3B	4A	4B
Ind. Addr.			X	X	X	X	X	X
Inner Lp. Bnd.	X	X						X
Outer Lp. Bnd.							X	
Vector Length			X	X	X	X		

For a more detailed account of these issues, and, specifically, the handling of parallelisation and experimental data on the implementation of the primitives SpMxM(V) the reader is referred to [18].

Concluding we can say that sparse BLAS implementations can speed up the performance of sparse computations considerably. So, in the case of the Alliant FX/8 (FX/80), on which the performance of unstructured sparse computations lies within the range of 0.1–10 Mflops, specialized higher order BLAS routines can speedup this performance to 5–30 (8–50) Mflops.

References

[1] E. Anderson and Y. Saad, *Solving Sparse Triangular Linear Systems on Parallel Computers*, International Journal of High Speed Computing, vol. 1, no. 1, pp. 73–96 (1989).

[2] I. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

[3] I.S. Duff, R.G. Grimes, and J.G. Lewis, *Sparse matrix test problems*, ACM Transactions on Mathematical Software, vol. 15, no. 1, pp. 1–14 (1989).

[4] I.S. Duff and J.K. Reid, *The multifrontal solution of unsymmetric sets of linear equations*, SIAM J. Sci. Stat. Comput., vol. 5, no. 3, 1984.

[5] A.M. Erisman, R.G. Grimes, J.G. Lewis, W.G.Jr. Poole, and H.D. Simon, *Evaluation of orderings for unsymmetric sparse matrices*, SIAM J. Sci. Stat. Comput., 8:600–624, 1987.

[6] J. Erhel and B. Philippe, *Multiplication of a vector by a sparse matrix on supercomputers*, INRIA/IRISA Report, 1988.

[7] K. Gallivan, B. Marsolf, and H.A.G. Wijshoff, *A Large-Grain Parallel System Solver*, Technical Report in Preparation (submitted to SIAM Conf. on Par. Proc. for Scient. Comput., 1989).

[8] A. George, *Nested dissection of a regular finite-element mesh.*, SIAM J. Numer. Anal., 10:345–363, 1973.

[9] G. Golub and C. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1983.

[10] D. Kuck, E. Davidson, D. Lawrie, and A. Sameh, *Parallel supercomputing today and the Cedar approach* in Experimental Parallel Computing Architectures, J. Dongarra ed., pp. 1-20, North-Holland, New York, 1987.

[11] F.H. McMahon, *The Livermore Fortran Kernels: A computer Test of the Numerical Performance Range*, Tech. Rep. UCRL-53745, Lawrence Livermore National Laboratory, Livermore CA, 1986.

[12] R. Melhem, *Parallel Solution of Linear Systems with Striped Sparse Matrices*, Parallel Computing, vol. 6, no. 3, pp. 165–184 (1988).

[13] C.D. Polychronopoulos, *Parallel Programming and Compilers*, Kluwer Academic Publishers, Boston, 1988.

[14] Y. Saad and H.A.G. Wijshoff, *A Benchmark Package for Sparse Computations*, proceedings of the 1988 International Conference on Supercomputing, pages 500–509, ACM Press, New York, 1988.

[15] J.H. Saltz, *Automated Problem Scheduling and Reduction of Synchronization Delay Effects*, ICASE Report no. 87–22, July 1987.

[16] J.H.Saltz, R. Mirchandaney, and K. Crowley, *The DoConsider Loop*, proceedings of the 1989 International Conference on Supercomputing, pages 29–40, ACM Press, New York, 1989.

[17] R.E. Tarjan, *Depth-first search and linear graph algorithms.* SIAM J. Computing, vol. 1, 1972, pp. 146-160.

[18] H.A.G. Wijshoff, *Implementing Sparse BLAS Primitives on Concurrent/Vector Processors: a Case Study*, CSRD Report 843, University of Illinois, January 1989.

[19] H.A.G. Wijshoff, *Symmetric Orderings for Unsymmetric Sparse Matrices*, CSRD Report 901, University of Illinois, August 1989.

Parallelism on CRAY Multiprocessor Systems: Concepts of Multitasking

W.E. Nagel

Zentralinstitut für Angewandte Mathematik
Forschungszentrum Jülich GmbH (KFA)
Postfach 1913, 5170 Jülich, Fed. Rep. Germany

Abstract

Modern supercomputers like CRAY X-MP and CRAY Y-MP achieve their high computing speed by using both vector and parallel hardware. This paper gives a short introduction into the CRAY Y-MP system architecture and describes the multitasking concepts which can be used on this machine. There are three different concepts which support parallelism on the programming language level: macrotasking, microtasking, and autotasking. Macrotasking supports coarse-grain parallelism on the level of subroutines. With microtasking, fine-grain parallelism can be used even on the level of DO loops. In contrast to these concepts where the multitasking primitives have to be introduced by hand, the new autotasking concept offers an automatic way for finding parallelism in existing FORTRAN programs. The concepts and implementations are discussed, and measurements of the overhead as well as performance results for kernels and an application program are presented.

Moreover, the overall system performance is of interest when multitasking concepts are used. Therefore, a programming system is developed, generating synthetic user programs which simulate a given work load in a flexible way. The resulting benchmark environment is used to insert additional sequential as well as parallel programs. This technique guarantees constant system load and enables reasonable comparisons. First results obtained from these investigations have proved the efficiency especially for the fine-grain concepts which provide good performance in dedicated as well as batch oriented multiprogramming environments; for selected production codes on the CRAY Y-MP these concepts are now used to evaluate their effects on a loaded system.

Keywords. CRAY Y-MP, multitasking, macrotasking, microtasking, autotasking, parallel programming, linear algebra kernels, benchmark environment, hardware performance monitor.

1. Introduction

The CRAY X-MP and CRAY Y-MP are multiprocessor systems with a shared memory. A short introduction into the system architecture is given in section 2. On these vector-supercomputers, parallelism on the programming language level is handled by three modes of multitasking.
Macrotasking supports parallelism on the subroutine level ([7,13,18]). Task creation, synchronization, and communication are specified explicitly by the programmer using subroutine calls. Macrotasking exploits the intrinsic parallelism of a problem by partitioning the computations into N tasks which are simultaneously executed on N processors. Inefficiencies arise when the tasks are not well balanced or synchronization is needed to often ([4]).
The second strategy is called microtasking and works on the statement level ([7,12]). It makes use of compiler directives inserted by the programmer. These directives are passed to a preprocessor which generates subroutine calls for the creation of parallel tasks and their synchronization. In contrast to macrotasking, the program parallelism is dynamically mapped to the number of CPUs available at run time.
The third strategy, which has been implemented recently, is called autotasking ([1,10,17,23]). It is based on improved dependency analysis techniques which provide an automatic mechanism for detecting parallel regions of code (normally DO loops) without user intervention. To improve the performance, this process may be supported by additional informational preprocessor directives specified by the programmer. In comparison with microtasking functionality is improved. The parallel primitives are slightly changed, whereas the synchronization techniques used have been proved to be efficient also in the microtasking implementation. In contrast to microtasking, autotasking supports the flexible definition of parallel regions at any place in a subroutine, for each of the parallel regions the data scope is analyzed and can be defined ex-

plicitly. These multitasking concepts available on CRAY machines are described in detail in section 3.

An evaluation of multitasking concepts should also take into account runtime measurements. There are several ways to assess the efficiency of multitasking implementations (see also Fig. 1):

1) Overhead measurements of multitasking primitives:
   The execution of multitasking primitives causes overhead due to the runtime used by the primitives itself. Section 3 provides a set of overhead timings for the main multitasking primitives.

2) Performance measurements of program kernels:
   In order to explore the strength of the multitasking implementations for kernels often used in scientific programs, parallel algorithms for linear algebra problems ([12,21]) have been implemented and extensively studied on the CRAY multiprocessor systems. A short summary of the results can be found in section 4.

3) Performance measurements of application programs:
   Section 5 presents results for a large application program implemented on the CRAY systems using multiple CPUs. Moreover, this section describes program modifications which are useful to improve load balancing and to exploit the total parallelism.

4) Measurements of the overall system performance:
   To analyze efficiency of multitasking concepts with respect to the total computer system, benchmark programs are used to generate a well-defined work load. The generation and selection of such benchmark programs is fundamental for the performance evaluation. Section 6 describes a program system developed by the author which generates benchmark programs capable to simulate any given user load in a flexible way. This system is used on a CRAY multiprocessor system to assess the efficiency of the multitasking concepts with respect to both user programs and operating system activities.

2. *The CRAY Y-MP Multiprocessor System*

The CRAY Y-MP is a shared memory multiprocessor system with up to 8 processors, and it is the successor system of the CRAY X-MP. A description of the CRAY X-MP system can be found in [12]; as far as the logical structure is concerned both CPU types are identical. Each CRAY Y-MP CPU is a high-speed vector processor with specialized pipelined functional units which can be utilized in parallel to perform high-speed (floating point) compu-

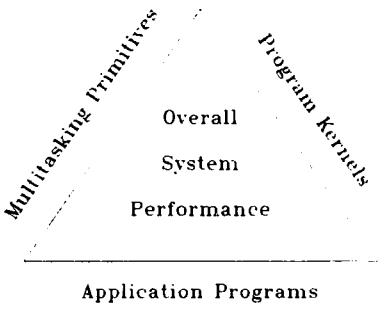


Fig. 1. Evaluation of multitasking concepts with respect to four areas

tations. The processors have a cycle time of 6 nsec. The main memory consists of up to 64 megawords and is realized in ECL technology providing an access time of 30 nsec. Compared with CRAY X-MP, the memory organization has been significantly improved; a description of the differences can be found in ([8]). The detailed system characteristics of the two CRAY multiprocessor systems installed at KFA are presented in Tab. 1.

	CRAY X-MP/416	CRAY Y-MP8/832
number of CPUs	4	8
CPU cycle time	8.5 nsec	6 nsec
number of functional units per CPU	13	13
main memory	16 MW (organized in 64 banks)	32 MW (organized in 256 banks)
memory access time	34 nsec	30 nsec
Solid-state Storage Device (SSD)	32 MW	128 MW

Tab. 1. System characteristics of the CRAY multiprocessors installed at KFA

The CPUs of one CRAY Y-MP system are tightly coupled via the shared main memory and 9 identical groups of registers, called clusters, containing 8 shared address registers (SB), 8 shared scalar registers (ST), and 32 binary semaphore registers (SM) each. These registers can be accessed by all processors; depending on program requirements either one cluster or multiple clusters, or eventually no cluster will be attached to a CPU. These hardware

features provide the basis for high-speed communication between parallel tasks when multitasking is used.

3. *Multitasking Strategies*

The multitasking concepts provided by CRAY Research Inc. support the exploitation of parallelism on different programming levels. They are realized by means of subprogram libraries, which are linked together into the library UTLIB, and they allow the execution of one program in parallel using *tasks*. In multitasking terminology, a task is a synonym for a *user library task*. User library tasks are generated by calls to the multitasking library subroutines. Each multitasking program is handled by the library scheduler which belongs to the multitasking library. The library scheduler is the interface between the multitasking program and the operating system. It handles the user library tasks and connects them to *exchange packages*. Afterwards, the job scheduler will find all necessary information in the exchange package in order to attach a physical CPU to the task.

For a program which does not use multitasking at all, the master task is executed. Within a program which uses multitasking, the master task will create further tasks by calling library subprograms. These new tasks are executed in parallel with the master task.

A subprogram may be called by more than one task simultaneously. Thus, it is necessary to guarantee that each task can access local variables without conflicts. Therefore, all local variables of the subprogram must be stored within the local memory area of each task (stack); the *cft77* compiler enables this feature. Subprograms compiled with this option are called *reentrant*.
If a non-reentrant subprogram is to be executed by multiple tasks in parallel, for each task a copy of this subprogram with a different name must be used, or the subprogram called must be imbedded into a critical region. In order to guarantee correct results when manipulating variables and data elements within a critical region, this region must not be entered by more than one task at a time.

3.1. The Macrotasking Concept

Macrotasking is the kind of multitasking where parallelism is realized at subprogram level. Within a program subprograms may be executed as different tasks. In order to use macrotasking, the application has to be partitioned into a fixed number of tasks; the user has to create the tasks explicitly by subprogram calls. Task management is done by the library scheduler. The library scheduler gets information about synchronization and communication requirements via the subprogram calls. It controls special queues and initiates the necessary activities. The job scheduler attaches the tasks to physically available CPUs ([7,13]). These tasks can be executed correctly in parallel, if there are no prohibitive mutual dependencies concerning synchronization and sequencing.

The macrotasking library consists of four different parts, which can be characterized by different functions of the imbedded routines:

1) Routines to manipulate tasks:

   - TSKSTART(*tskarray*,*name*[,*list*])
     generates a new task with identification *tskarray* for subroutine *name* and passes the parameters of *list* to the subroutine.
   - TSKWAIT(*tskarray*)
     waits for the end of the task identified by *tskarray*.

2) Routines to control events[1]:

   - EVASGN(*name*[,*value*])
     declares the INTEGER variable *name* as event variable.
   - EVWAIT(*name*)
     waits for the event *name*.
   - EVPOST(*name*)
     signals the event *name* to the scheduler.

3) Routines to control barriers[2]:

   - BARASGN(*name*[,*value*])
     declares the INTEGER variable *name* as barrier variable, *value* specifies the number of tasks which have to synchronize.
   - BARSYNC(*name*)
     signals to the scheduler that this task has arrived at the barrier specified with *name*.

4) Routines to control critical regions:

   - LOCKASGN(*name*[,*value*])
     declares the INTEGER variable *name* as lock variable.

1 Events can be used to force a certain order of execution between tasks.
2 Barriers are used to assure that all tasks have reached a particular program location.

- LOCKON(*name*)
  signals to the scheduler that this task will enter the critical region connected to the variable *name*.
- LOCKOFF(*name*)
  signals to the scheduler that this task leaves the critical region connected to the variable *name*.

Multitasking primitive timings as measured on CRAY Y-MP can be found in Tab. 2 at the end of section 3. For the generation and termination of tasks some additional work is needed to allocate storage locations. The basic overhead for generating a new task via a TSKSTART call is about 62 microseconds. The synchronization of the data access to global variables (i.e. for reading or updating) can be done using events, locks, and barriers. The overhead for an *event post*[3] - *event wait* is about 16.1 microseconds, the overhead for a *lock set* - *lock free* is about 5.5 microseconds. The generation of lock and event variables as well as the release of such variables takes from 2 to 3 microseconds. For a *barrier* synchronization of two tasks about 13.6 microseconds have to be paid, the assignment of the barrier variable costs about 6.5 microseconds, the release of this variable about 3.6 microseconds.

For certain kinds of programs, macrotasking leads to efficient use of the multiprocessor machine. But often the user has to deal with three types of problems which may reduce the speedup achievable by this multitasking strategy:

1) For small granularity, the system overhead is too large due to the high synchronization frequency involved.
2) Some of the tasks cannot be executed in a balanced way on the multiple CPUs.
3) The number of processors does not match the fixed number of tasks specified within the macrotasking program.

Several manufacturers of multiple-CPU systems like CRAY and IBM are promoting developments in the field of parallel programming, in particular to reduce synchronization overhead within multitasking. Microtasking is an approach that allows the efficient use of multiple processors even for small granularity ([12,18]).

3.2. The Microtasking Concept

Microtasking provides parallelization on the statement level, thus providing a different interface to multitasking. Statements, sequences of statements, and complete subprograms may be executed in parallel on multiple processors of a CRAY Y-MP machine, for instance, when microtasking is used.
Instead of using a scheduler all communication and synchronization is done by accessing shared registers directly ([7,14]). The overhead of the communication via registers is much smaller than the overhead to access corresponding variables in main memory as realized within the macrotasking library routines.

The microtasking features are provided by preprocessor directives ([7,14]). These directives are coded into the user program; this leads to multitasking programs which are easier to understand than in the macrotasking version. The usage of directives does not affect the portability of programs. Because the directives must be coded with a 'C' in the first column, all FORTRAN compilers will interpret such statements as comment statements.
The microtasking strategy is based on the preprocessor PREMULT ([7,14]), which takes the user program as input and generates three separate subroutines for each subroutine which is to employ microtasking:

1) Master routine:
   The master routine is a program coded in assembler language, which is called by the same name as the original program. This routine decides whether the parallel version of a subprogram will be used or not.
2) SNGL version:
   The SNGL version contains FORTRAN program code for the single-task execution of the program.
3) MULT version:
   The MULT version is a modified FORTRAN version of the subprogram to be executed in parallel.

Within the MULT version PREMULT translates directives into corresponding library calls. There are several directives available to the user; they can be classified according to their usage:

1) Demand and return of physical CPUs:
   - CMIC$ GETCPUS n
     declares the maximum number of CPUs the program can use.
   - CMIC$ RELCPUS
     returns the required CPUs back to the operating system.
2) Definition of control structures:
   - CMIC$ DO GLOBAL
     marks a DO loop to be executed in parallel if more than one CPU is available.

[3] One task is still waiting on the event which is posted.

- CMIC$ PROCESS
  marks the beginning of a program part which must be executed by one processor.
- CMIC$ ALSO PROCESS
  marks the beginning of a further program part which could be executed in parallel to other processes. This directive may be used several times.
- CMIC$ END PROCESS
  marks the end of a PROCESS part.
- CMIC$ STOP ALL PROCESS
  provides a way to stop parallel execution before all computations are finished.

3) Safeguarding of critical regions:

- CMIC$ GUARD n
  marks the beginning of the critical region named n.
- CMIC$ END GUARD n
  marks the end of the critical region named n.

The safeguard directives are also translated within the SNGL version because exclusive access to variables also must be guaranteed for sequential subroutines which are called by microtasking tasks.

Contrary to macrotasking, microtasking parallelism is specified by the definition of control structures. A microtasking control structure declares a part of a program which must be finished before a code part outside of this region may be executed. Within such a control structure *processes* are defined. A microtasking process is a sequence of instructions which always is executed as a single task. Microtasking distributes such processes dynamically to the actually available CPUs. Program segments outside of microtasking control structures are executed by all tasks in an unpredictable sequence.

The dispatch and wait overhead for scheduling a microtasking subroutine is low and costs about 2.8 microseconds on a CRAY Y-MP. The overhead of executing a parallel loop instead of a normal DO loop is $0.450 * ch + 2.6$ microseconds (*ch* is the number of chunks, each chunk contains a definite number of loop iterations) which corresponds to $75 * ch + 433$ CPU cycles. To execute several PROCESS directives about $1.0 * pd + 2.3$ microseconds (*pd* is the number of PROCESS directives) must be paid, and locking a critical section using the GUARD directives causes an overhead of 0.8 microseconds.

Microtasking has proved high efficiency of the synchronization primitives, but still the user has to introduce parallelism manually by inserting preprocessor directives.

3.3. The Autotasking Concept

The main goal of the CRAY autotasking concept is to provide an efficient and automatic mechanism to parallelize programs on CRAY multiprocessors (see [1,23]). Autotasking integrates the microtasking concept: most of the communication and synchronization is done also by accessing shared semaphore registers directly. Additional tasks execute a *Wait-on-Semaphore* operation on a shared register. If there is any parallel work to do, the semaphore is reset, and all tasks are able to execute parts of the work. CRAY autotasking will support fine-grain parallelism on all CRAY machines running UNICOS[4].

In contrast to microtasking, fine-grain parallelization with autotasking is done by the automatic parallelization of DO loops. The features are provided by the compiling system *cf77* which can be partitioned into three parts: *fpp*, *fmp*, and *cft77*. The preprocessor *fpp* analyzes FORTRAN programs; it is able to detect DO loops executable in parallel, and it marks these loops with preprocessor directives. These directives are translated by the *fmp* preprocessor which is the second part of the *cf77* compiling system, and the *cft77* generates machine code for the modified FORTRAN program.

The autotasking primitives represent a superset of the microtasking primitives with several extensions improving functionality. With microtasking, a parallel region has to begin at the top of a subroutine. This has been changed significantly: autotasking supports flexible definition of parallel regions at any place of a subroutine. The preprocessor *fmp* generates a set of subroutines for each of the parallel regions, and all available CPUs will enter at the beginning of these subroutines. Implicit synchronization takes place at the bottom of the code marked by the PARALLEL DO, the END CASE, or the DO ALL directives. For each of the parallel regions the data scope of the variables (SHARED or PRIVATE) is analyzed and can be defined explicitly.

In comparison with microtasking, only the directives which define multitasking control structures are changed, the directives for CPU handling and safeguarding remain unchanged. The following list gives a short description of the new autotasking primitives

[4] An autotasking version for the COS operating system is being released just now.

- CMIC$ PARALLEL
  marks the beginning of a parallel region where multiple CPUs may enter. The parallel processes within these parallel regions have to be marked with PARALLEL DO or CASE directives.
- CMIC$ END PARALLEL
  marks the end of a parallel region.
- CMIC$ DO PARALLEL
  marks a DO loop to be executed in parallel if more than one CPU is available within a parallel region.
- CMIC$ DO ALL
  defines a separate parallel region for the following DO loop only. In addition, this DO loop is marked to be executable in parallel by more than one CPU.
- CMIC$ CASE
  marks the beginning of a program part which could be executed in parallel to other processes. The directive may be used several times.
- CMIC$ END CASE
  marks the end of a CASE part.
- CMIC$ SOFT EXIT
  provides a way to stop parallel execution before all computations are finished.

Several directive options like threshold tests, flexible definition of chunk sizes (including the guided self scheduling approach [22]), automatic partitioning of long vector loops etc. have been introduced to optimize execution of parallel regions. Furthermore, there are some minor syntax changes. Compared with microtasking under the COS operating system, the synchronization speed is drastically increased. Instead of calling a function as within microtasking, autotasking uses inline code for synchronization. The overhead of executing a parallel loop instead of a normal DO loop (see also [23]) is about $0.150 * ch + 1.1$ microseconds on a CRAY Y-MP (*ch* is the number of chunks) which corresponds to $25 * ch + 183$ CPU cycles. A critical section can be locked using the GUARD directives leading to 0.95 microseconds overhead.

Autotasking can coexist with macrotasking and microtasking within one program system, but autotasking and microtasking are not allowed to be used together within one subroutine. At the moment, nested parallelism (i.e. parallel loops inside a parallel loop) is not supported by autotasking.

The basic overhead of all multitasking concepts is summarized in Tab. 2. Owing to the actual implementation, the macrotasking and microtasking measurements are done under the COS Rel. 1.17 operating system whereas autotasking is measured under UNICOS 5.0.

Multitasking *primitive*	CRAY Multitasking Concepts		
	Macro-tasking (COS)	**Micro-tasking (COS)**	**Auto-tasking (UNICOS)**
Dispatch-Wait	62	2.8	2.5
Parallel Loop	-	*0.450*ch + 2.6*	*0.150*ch + 1.1*
Parallel Case	-	*1.0*pd + 2.3*	*0.2*pd + 1.9*
Event Post-Wait	16.1	-	-
Lock Set-Free	5.5	0.8	0.95
Barrier Synchronization	13.6	-	-

Tab. 2. Overhead caused by multitasking primitives: All times are measured in microseconds on a CRAY Y-MP8/832. *ch* is the number of chunks, and *pd* stands for the number of process resp. case directives.

4. Linear Algebra Kernels

Using multitasking to take advantage of parallelism within a program gives the chance to speedup special programs appreciably. This is especially true for linear algebra kernels which are heavily used in large application packages. Moreover, multitasking strategies and corresponding problems can be studied in detail executing such programs.

As matrix multiplication has a simple structure, this algorithm is often used to evaluate the different multitasking strategies and to document the potential benefit for such easy-to-use algorithms. There exist several algorithms and implementations for the matrix multiplication; the subroutine MXV from SCILIB (CRAY subroutine library) can be used, for example, to perform the matrix-vector operations (BLAS 2). The usage of microtasking is introduced by marking the DO loop with a DO GLOBAL directive (see [12]). This program is often used within literature as an example to document the effectiveness of microtasking; there, microtasking is used in a natural and easy way. Fig. 2 shows the speedup[5] obtained for the library routine MXV using microtasking and autotasking as well as the macrotasking version of a parallelized MXM subroutine from SCILIB (BLAS 3). It can be seen that for a vector length larger than 128 neither overhead nor memory organization problems are of significant influence, and the total performance is quite satisfying.

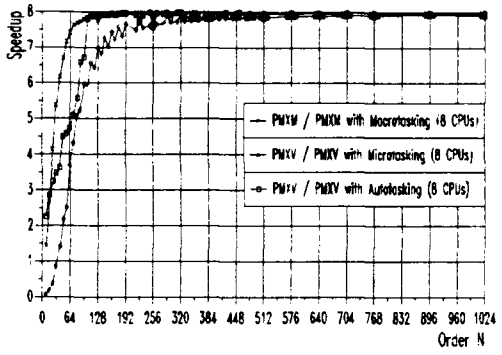


Fig. 2. Matrix multiplication using BLAS 2 and BLAS 3 library routines with multitasking

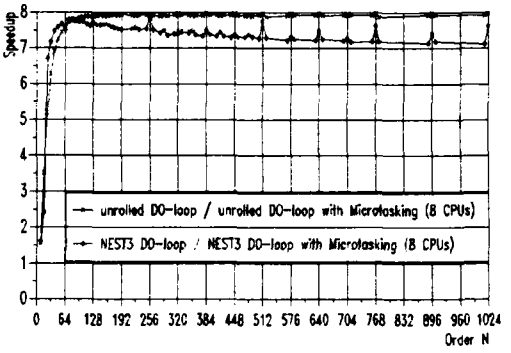


Fig. 3. Matrix multiplication using DO loops and microtasking

Within user programs, three nested DO loops are normally used as an algorithm for the matrix multiplication instead of library routines. This implementation makes it possible to work with a portable code running on most of the available machines, and microtasking can be specified in an easy way by marking the outer loop with a DO GLOBAL directive. J.J. Dongarra has introduced the unrolling technique ([9]). for single-task vectorizing programs, in particular to remove memory conflicts. Using 8-way unrolling the memory-section and bank conflicts ([12,20]) are significantly reduced.

Fig. 3 shows the speedup obtained for both versions of the DO loop algorithm using microtasking which documents the benefit from unrolling in combination with microtasking (see also [12]).

In addition to matrix multiplication, some linear algebra algorithms like LU and Cholesky decomposition heavily used in application programs within scientific and technical computing environments are studied. Results for these kernels can be found in ([12,13]).

5. Parallel Application Program

To get deeper insight into the multitasking implementations, the concepts were used to parallelize a numerical simulation program which seems to be typical for a wide range of applications running at our site. Numerical simulation is used more and more, for example, to get detailed information about crystal growth processes. In the Czochralski crystal growth from electrically conductive melts, the application of an external magnetic field has become a very useful technique for improving crystal quality. The simulation leads to a system of coupled partial differential equations, i.e. incompressible Navier-Stokes and convective heat equations, which have to be solved. It provides information about the qualitative and quantitative difference between the influence of a stationary transverse magnetic field and a vertical magnetic field on the flow and temperature distribution in the Silicon melt, for instance, by means of numerical simulations in a three-dimensional mathematical model of the Czochralski crystal growth. A more detailed description of the simulation program can be found in [12,15,19].

This program is used to carry out parameter studies to get detailed information about the Czochralski bulk flow. The sequential version of the simulation program is highly optimized for vectorization, running after some code modifications with a speed of about 195 MFLOPS on one processor of a CRAY Y-MP. The CPU time needed to solve one of these problems varies from 10 to 30 CPU hours on one processor of a CRAY Y-MP, depending on material and geometrical parameters and the considered time interval. The simulation program has a simple structure, and using FLOWTRACE ([6]) it was obvious that after a short time of initialization more than 99% of the

5 Time measurements are done by calls to IRTC on a dedicated CRAY Y-MP8/832. The operating system level was COS 1.17 BF 1 resp. UNICOS 5.0. The speedup is calculated as the ratio of corresponding times. For kernel measurements always the minimum of three executions is taken as the time result to remove the additional work for the first TSKSTART calls.

total CPU time is spent in three subroutines. Based on this information further examinations could focus on these subroutines.

Exploiting parallelism with the macrotasking concept on the CRAY X-MP, for each of these subroutines four independent tasks are generated with the TSKSTART primitive. These tasks are synchronized with the new barrier synchronization primitive introduced with the COS 1.17 operating system. Most of the computational work is done in three-fold nested loops: outer I-loop (running from 2 to 26), then the K-loop (running from 2 to 34), and the inner J-loop (running from 2 to 92). An example of a typical loop nest can be found in Fig. 4.

```
      DO 20008 I=2,LM1
        RFTP=RFI(I)*DPHII*SI
        DO 151 K=2,NM1
          DO 151 J=2,M
            CR(I,J,K)=CR(I,J,K)+SI*DRI*
     >        (DFF(I,J,K)-DFF(I+1,J,K))
            CF(I,J,K)=CF(I,J,K)+RFTP*
     >        (DFF(I,J,K)-DFF(I,J+1,K))
            CZ(I,J,K)=CZ(I,J,K)+SI*DZI*
     >        (DFF(I,J,K)-DFF(I,J,K+1))
  151   CONTINUE
20008 CONTINUE
```

Fig. 4. Typical loop nest

The *cft77* compiler vectorizes only on the innermost DO loop. Therefore, efficient vectorization is achieved by running the longest loop in vector mode. Using macrotasking, the outer I-loops are partitioned into as many parts as CPUs are available, and each task takes its own work based on a special identifier associated with this task (see Fig. 5).

The microtasking parallelism is introduced by several DO GLOBAL directives which specify independent I-loop iterations. Because of the microtasking concept which guarantees explicit synchronization of all active tasks at the bottom of a microtasking control structure (MCS) no additional synchronization is needed.

```
      SUBROUTINE VELO(ID)
C     declarations
      COMMON/NOTASKS/NTSKS
      COMMON/EVENTS /EVENT1
C     partition DO loop, starting at 2,
C                        ending at LM1
      I1=2
      I2=LM1
      ITER=(I2-I1+1)/NTSKS
      ILAST=(I2-I1+1)-ITER*NTSKS
      II1=I1+(ID-1)*ITER+MIN(ID-1,ILAST)
      II2=I1+(ID  )*ITER+MIN(ID,ILAST)-1
C     . . .
      DO 111 I=II1,II2
        DO 111 K=2,NM1
          DO 111 J=2,M
C           . . .
111   CONTINUE
C     . . .
      CALL SYNCH(ID,EVENT1,NTSKS)
C     . . .
      END
```

Fig. 5. Macrotasking version of subroutine VELO running with identifier ID

The time[6] is measured to check the results for a time interval of the simulation process which represents a typical situation where the flow is stabilized. This time interval covers a few seconds of the real crystal growth experiment and corresponds to one CPU hour of one processor of a CRAY X-MP. The results obtained for the macrotasking version show that the overhead introduced by the additional library calls is about 4%. With microtasking the overhead is reduced to 1%. The time measurement for both versions are presented in Tab. 3.

The sequential (i.e. single-task) reference represents the wall clock time[7] needed by the best single task program version simulating 3.4 seconds of the simulation process. The next row provides the time measurement for the multitasking versions when four CPUs are used.

---

6 Time measurements were done by calls to IRTC on a dedicated CRAY X-MP/416. The operating system level was COS 1.16 BF 3.

7 Time the user has to wait for the result on a dedicated machine.

		macro-tasking	micro-tasking
sequential reference (one program version running in dedicated mode)		2747 s	
4 CPUs	wall clock time (in seconds)	864 s	822 s
	Speedup obtained	3.18	3.34

Tab. 3. Speedup results on CRAY X-MP/416 with 64 memory banks (COS 1.16 BF 3)

In these experimental results the dominant factor which limits the speedup achieved in this application is memory contention which introduces an additional overhead. Using two CPUs, up to 98 per cent of the theoretical speedup is gained, with four CPUs only 90 per cent of the theoretical speedup is obtained by microtasking.

With both concepts, macrotasking and microtasking, it was not possible to get a reliable timing profile for each of the loop nests in an automatic way. This has changed with the autotasking concept. There, a parallel region can be identified with one loop nest, and timings can be inserted to measure this nest. The preprocessor PARANAL ([12]) developed by the author was adapted to insert automatically timing routines just around the loop nests. Each loop nest is numbered, initialization and accumulation of the real time spent in these nests is done in FORTRAN arrays without further intervention.

For the application program, about 40 loop nests are encountered. From the timing information, about 15 loop nests are considered to be important (more than 0.1% of the total time spent in the loop). Fig. 4 shows a typical loop nest (nest 21) representing about 28% of the total CPU time even for a short simulation interval. Autotasking parallelism is introduced by DO ALL directives which specify independent DO loop iterations. These directives are inserted automatically by *fpp* which is part of the *cf77* compiling system. The autotasking concept guarantees explicit synchronization of all active tasks at the bottom of the DO ALL, and no additional synchronization is needed. To compare the results with the previous timings shown in Tab. 3, Fig. 6 shows speedups[8] using up to four CPUs of the CRAY Y-MP system, executing an autotasking version of the simulation program with some minor code changes and additional *fpp* information directives. Each of the bars represents one loop nest, on the x-axis the total time (in seconds) spent within these loops and the loop nest number is given, the y-axis gives the speedup results obtained for each of the loop nests as well as for the whole program. Using 4 CPUs a speedup of about 3.5 is obtained; this improved result is based on the new memory conflict resolution strategies (see [8]). The overhead of about 1.5% for the parallel version using one CPU, compared with the optimal sequential version, documents the efficiency of the implementation as far as autotasking primitives are involved.

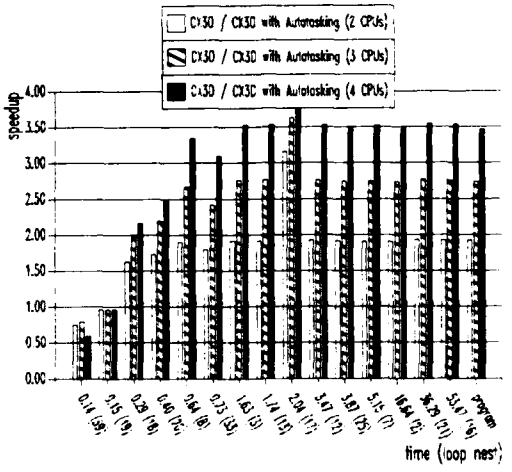


Fig. 6. Original Czochralski bulk flow simulation program using up to four CPUs

As can be seen in Fig. 7, increasing the number of CPUs does not lead to a linear increase of the speedup. This is especially true for code regions with small granularity (i.e. two-fold nested loops No. 20 and No. 39), where the communication overhead exceeds the computational work. Using 5 CPUs, the whole program can be executed very efficiently achieving a speedup of about 4.4, but for higher number of CPUs the total number of iterations of the I-loop (25) which have to be scheduled cannot be distributed in a load balanced way to 7 or even 8 processors. The granularity of the iterations which have to be scheduled afterwards is rather large, using 8 CPUs a quarter of the real time used within this loop is spent in the last iteration. Moreover, loop nest No. 17 is a special case of a search

[8] Time measurements were done by calls to the timing routine IRTC on a dedicated CRAY Y-MP8/832 running native UNICOS 5.0.

loop where the outer loop is parallelized. In this case, the search order is slightly changed which leads to a superlinear speedup for the two- and the three-processor version. On the other hand, more than four CPUs do not increase the speedup.

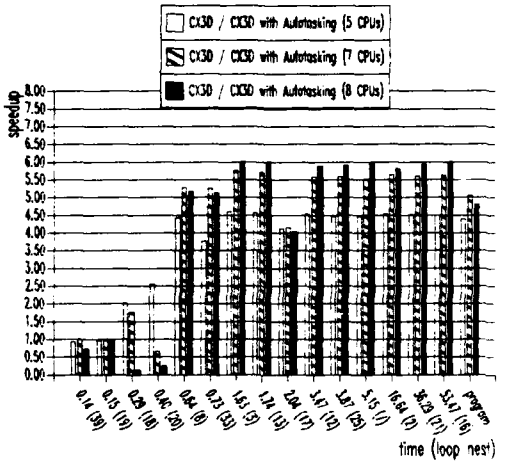


Fig. 7. Original Czochralski bulk flow simulation program using up to 8 CPUs

number of CPUs	number of iterations for each CPU	remaining iterations (running in parallel)	speedup limited by Amdahl's law	speedup achieved for loop 21
1	25	0	1	1
2	12	1	1.92	1.92
3	8	1	2.77	2.76
4	6	1	3.57	3.54
5	5	0	5	4.51
6	4	1	5	4.88
7	3	1	6.25	5.60
8	3	1	6.25	5.93

Tab. 4. Load balance analysis for original loop 21

Tab. 4 gives a summary about the speedup limits based on load balance reasons. The number of iterations which have to be distributed to the processors is much too small to lead to satisfying results for larger number of CPUs.

Nested parallelism is not supported by the autotasking implementation, therefore other possibilities have to be considered. As a consequence, the program was analyzed once again to look for further possible program transformations. After checking data dependencies, the program was restructured by hand in the following way:

1) The indices for the outer two loops are stored in the FORTRAN arrays IJINDEX and IKINDEX once at the beginning of the program (see Fig. 8).

```
      DO 100 I=2,LM1
        DO 100 J=2,M
          IJLENGTH=IJLENGTH+1
          IJINDEX(IJLENGTH,1)=I
          IJINDEX(IJLENGTH,2)=J
100   CONTINUE
      DO 200 I=2,LM1
        DO 200 K=2,NM1
          IKLENGTH= IKLENGTH+1
          IKINDEX(IKLENGTH,1)=I
          IKINDEX(IKLENGTH,2)=K
200   CONTINUE
```

Fig. 8. Additional FORTRAN code to store loop indices in a shared array (executed once)

2) All three-fold nested loops similarly to loop 21 are translated to two dimensional loops with higher number of outer loop iterations (see Fig. 9).

```
CMIC$ DO ALL SHARED(...) PRIVATE(...)
      DO 20008 IK=1,IKLENGTH
        I=IKINDEX(IK,1)
        K=IKINDEX(IK,2)
        RFTP=RFI(I)*DPHII*SI
        DO 151 J=2,M
          CR(I,J,K)=CR(I,J,K)+SI*DRI*
     >         (DFF(I,J,K)-DFF(I+1,J,K))
          CF(I,J,K)=CF(I,J,K)+RFTP*
     >         (DFF(I,J,K)-DFF(I,J+1,K))
          CZ(I,J,K)=CZ(I,J,K)+SI*DZI*
     >         (DFF(I,J,K)-DFF(I,J,K+1))
  151   CONTINUE
20008 CONTINUE
```

Fig. 9. FORTRAN code for the modified loop nest No. 21

3) The search loop (loop 17, see Fig. 10) is treated similar to loop 21, and in this case the collected indices of the J- and the K-loops were used. In addition, the compiler directive *CDIR$ SUPPRESS FOUND* is used to assure that the shared variable *FOUND* is not held in a register but instead immediately stored into the memory.

```
CMIC$ PARALLEL SHARED(...) PRIVATE(...)
CMIC$ DO PARALLEL CHUNKSIZE(NPROC)
      DO 200 IJ=1,IJLENGTH
        I=IJINDEX(IJ,1)
        J=IJINDEX(IJ,2)
CDIR$ SUPPRESS FOUND
        IF(FOUND)GOTO 200
        DO 95 K=2,NM1
95      IF(ABS(DFF(I,J,K)).GE.EPS)GOTO 99
        GOTO 200
99      CONTINUE
        FOUND =.TRUE.
CDIR$ SUPPRESS FOUND
CMIC$ SOFT EXIT
        GOTO 210
200   CONTINUE
210   CONTINUE
CMIC$ END PARALLEL
```

Fig. 10. Modified search loop, implemented with autotasking

With these code modifications there is a much higher level of parallelism (825 resp. 3003 loop iterations instead of 25). Each of these iterations can be executed in parallel without changing any data dependencies.
The modified sequential program is running a little bit slower (about 185 MFLOPS), therefore in Fig. 11 the time results of the modified program are compared to the best sequential version running on one processor. Using 8 CPUs a speedup of about 6.1 is achievable leading to a total speed of about 1,200 MFLOPS on a CRAY Y-MP multiprocessor system.

6. *Multiprogramming and Parallelism*

Today most of the CRAY multiprocessor systems are still used within a multiprogramming environment, where the individual processors execute different jobs which are totally independent of each other. All programs compete for the available resources. With multitasking, they may

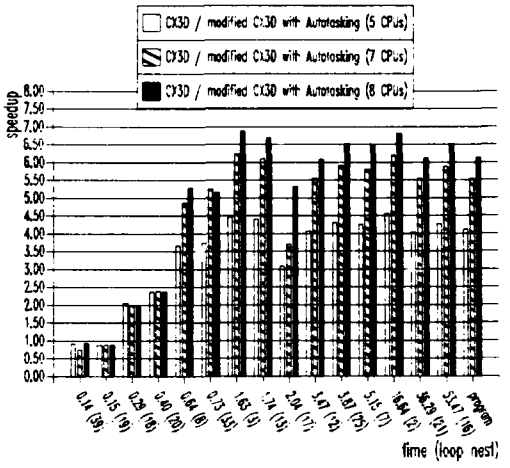


Fig. 11. Modified Czochralski bulk flow compared to the best sequential version

also execute different parts of one program in parallel. The behavior of programs using multitasking within a multiprogramming environment and the influence of these programs to the overall work load is of great interest for the rating of multitasking concepts with respect to their efficiency in practice.

The operating systems COS and UNICOS allow the execution of a parallel program on a dedicated machine or in a multiprogramming batch environment. A dedicated machine is a computing environment which makes all its resources (for example all CPUs) available to one program without any restrictions. All of the results presented above are obtained in such an environment.

Using one of the multitasking concepts within a multiprogramming batch environment, each task has to compete for the resources with other tasks of the same program and with tasks of other programs executed at the same time. Tasks of a program with higher priority may force resources to be withdrawn from tasks generated by programs with lower priority. Usually, the turnaround time of a program is higher if it is executed within a multiprogramming batch environment than the time it needs on a dedicated machine.

To evaluate the impact of multitasking onto the multiprogramming environment and to ensure reproducible results, there is a need to have a well-defined and constant system load. This can be done by choosing a benchmark to generate a multiprogramming environment. After fixing the

work load, some test jobs are inserted into the job mix to study various effects.

6.1. Synthetic Benchmark Programs

Benchmark programs should reflect real program behavior. The selection of such benchmark programs is an important basis for the performance evaluation which may also have a strong influence on the results obtained. For example, in ([2,3]) a benchmark is described which is memory bound and therefore leads to a decreased turnaround time when microtasking is used. Usually, there is no way of studying the same benchmark with different memory requirements. In general, the flexible adaptation of such real-life application programs to required benchmark parameters (i.e. modified run-time, different memory requirements, changed I/O behavior etc.) is limited because of the strong correlation between the parameters. This is especially true in a multiprocessor system where the load of one processor may have influence on the performance of the programs running on the other processors (i.e. bank conflicts, I/O-blocking, etc.). On the other hand, there is a strong requirement for such flexible modifications of the work load where each single benchmark program should reflect a certain behavior. Therefore, a system is developed supplying synthetic benchmark programs in order to simulate any given work load.

On shared-memory multiprocessor systems like CRAY X-MP and Y-MP, there are only a few program characteristics which influence the activities of the other processors: For example, if the memory requirement of one job is the total memory installed at a site, then this may cause the other CPUs to execute idle loops. In this category of interprocessor dependent values the required memory size, the memory activities, and the I/O traffic can be categorized. Other values like the requested CPU time, the job priority, and the MFLOPS rate are benchmark program parameters which have no direct influence on jobs running on other processors. Based on these considerations the synthetic benchmark programs are generated as follows:

> Some of the values mentioned above, i.e. the memory size (MSIZ), the job priority (P), and the job time (T) are values which can be assumed to be static. For each set of such parameters different versions of the JCL are used.
>
> Values like the number of million memory references per second (MMREFS) or the number of million array elements transferred per second to an I/O device (MIOS) are highly system context dependent. For those values dynamic decisions have to be made to guarantee a certain program behavior for this program.

These dynamic decisions are based on information from the hardware performance monitor (HPM) which is available on CRAY X-MP and CRAY Y-MP systems (see [6]). The HPM supplies a set of eight counters which track certain hardware related events, i.e. the number of executions of specific instructions, memory activities, MFLOPS etc. With this information the complete program characteristics can be discovered at each point during execution. A decision can be made if the characteristics match the desired program properties. A set of kernels exists (containing presently about 100 items) with different execution properties. In the synthetic benchmark program, each call to the HPM leads to a decision which kernel is the next one to be executed to approximate the described properties. At the end, the benchmark program behavior is completely composed of the weighted mean of the properties of all kernels selected during runtime.

The selection of the kernels is done in the following way: each program kernel is represented by a point $\mathbf{p}$ = (MMREFS,MIOS,MFLOPS) in a three-dimensional space. The actual program performance of a single benchmark program $\mathbf{r}$ is a weighted linear combination of the performance of program kernels K_i, referencing a point $\mathbf{p}_i$, using t_i CPU time. At decision time kernel K_j is chosen which minimizes the Euclidean distance[9] $\|...\|$ from the given reference point $\mathbf{r}$, which means:

select K_j so that $\Delta(i_j) = \min_i \Delta(i)$ with

$$\Delta(l) := \left\| \mathbf{r} - \frac{(\sum_{k=1}^{j-1} t_{i_k})\,\mathbf{r}^{(j-1)} + t_l\,\mathbf{p}_l}{(\sum_{k=1}^{j-1} t_{i_k}) + t_l} \right\| .$$

This selection procedure enables the simulation of any three-dimensional point of the convex hull covering all the program kernels K_i. Any point $\mathbf{r}$ = (MMREFS,MIOS,MFLOPS) outside of this hull will be approximated by the point of the convex hull which minimizes the Euclidian distance to this point.

The sequence of kernel selections made at runtime is written into a file. Further examinations of the same benchmark may be controlled by this protocol file. This

[9] All values are normalized to the interval (0,1).

leads to a system load which is exactly the same because the same sequence of kernels K_i is executed without any respect to the actual performance.

Besides the protocol file, the synthetic benchmark program provides run-time output reflecting the program behavior:

- Start time and end time of the job (first and last statement of the benchmark program are timing routines);
- Required program characteristics (specified by the input parameters);
- Obtained program characteristics (based on the information supplied by the HPM).

In general, the difference between the given program properties and the benchmark program characteristics obtained are rather small. Already after a few seconds (2 to 5 sec.), the approximated values are very close to the given target (if the target lies within the convex hull of the kernels). As an example, Tab. 5 describes a subset of parameters of the execution history of one benchmark program, named BE001. The table shows that the values specified for the job and the values really obtained by kernel selection are quite the same. In this example, within 60 CPU seconds about 10,000 kernel selection decisions are made. If the kernel selection is done by using the protocol file, the obtained program performance is about 2 to 7 per cent higher because of the omitted calls to the HPM.

program characteristics for BE001	CPU time (T)	Memory Size (MSIZ)	MMREFS	MFLOPS
Specified by the input	60.00	2MW	110.00	80.00
Obtained by kernel selection	60.28	2MW	110.33	80.19

Tab. 5. Program characteristics of benchmark program BE001

Using synthetic benchmark programs in this way, the only information needed for simulating a real workload are the program characteristics of the important jobs running at a site. All these data are available through the accounting data and the HPM. With this information a copy of the real workload can be executed in an artificial benchmark environment, and the real time can be scaled on a much lower level.

6.2. The Benchmark Generation

The simulation of a normal work load implies that a mix of such benchmark programs must be executed. To built such a job mix a benchmark generator is convenient. The benchmark system generator described here takes the required program characteristics for all benchmark programs as an input and generates the set of synthetic programs which dynamically approximate the characteristics for each program by using the HPM.

Here, a special benchmark is used which consists of 14 different synthetic programs simulating a given user load. This benchmark was executed on the CRAY X-MP/416 under the COS operating system. For simplicity, all work load jobs are running with priority 6, the memory requirement for each of the benchmark programs is 2 MW (Mega Words). The other job characteristics are slightly changed from job to job, for example the memory activities vary from 50 to 110 MMREFS.

Based on this work load the influence of additional test jobs on the total system is examined. As test jobs, a sequential as well as a parallel version of the numerical simulation program described in section 5 is inserted into the job mix. Each of the test jobs has done the same amount of work. About 4MW main memory are required to execute the jobs, and they ran with job priority 7.

6.3. The Benchmark Analysis Package

The total benchmark supplies information about different influences sequential and parallel programs have on the system behavior. With respect to the amount of data describing the system behavior, an analysis without automatic tools is not practicable. For that reason, a benchmark analysis package was developed by the author to provide collected information for all benchmark programs. It provides tables and graphical charts, because visualization may be helpful in finding the typical benchmark behavior as well as hot spots.

Fig. 12 shows one of the visualizations created by the benchmark analysis package. It is a benchmark execution profile where a sequential job is inserted as a test job (left hand side). For each job the chart shows the start and end time (in seconds) relative to the start of the first job (BE001). It also contains information how long the jobs stayed in the machine and how much CPU time was consumed (black color). The wait time is computed as the difference between turnaround time and CPU time. In the right corner, the sum of the CPU and the wait time for all jobs and for the test job is printed. The total time indicates

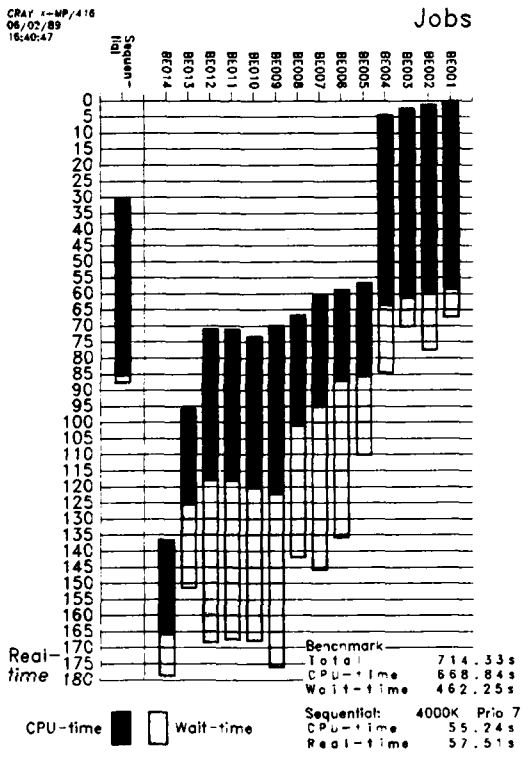


Fig. 12. Benchmark environment, sequential test job inserted

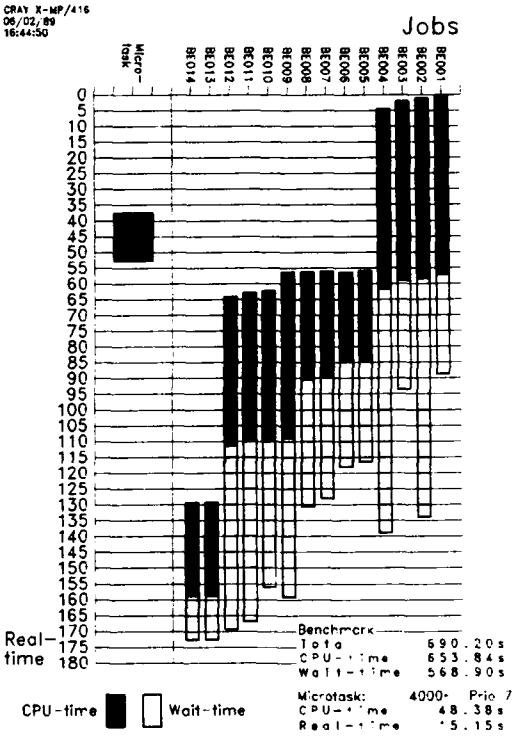


Fig. 13. Benchmark environment, microtasking test job inserted

the real time for the whole benchmark, multiplied with the number of CPUs. Fig. 13 shows the same benchmark, but executing the microtasking version of the simulation program. Comparing these two charts it can be realized that inserting microtasking can lead to both, reduction of turnaround time and slightly decreased CPU time for the whole benchmark. On the other hand, the accumulated wait time is increased which means that at least some jobs have to stay longer in the machine.

All the information used above is provided by the HPM and timing routines for each program. But there is also a need to look at the system activities to get a knowledge about the rate of multitasking concepts with respect to the interests of a computer center. To measure system activities, the system performance monitor (SPM) can be used (see [5]). It provides several blocks of information concerned with all important system activities. Fig. 14 shows a sketch of the information structure available by the SPM.

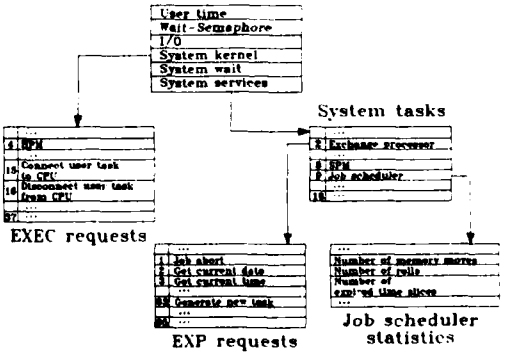


Fig. 14. Sketch of the information provided by the benchmark analysis package, based on system performance monitor (SPM) data

On the first level, information about system-wide accounting data like user time and wait-on-semaphore time, but also, for example, the time required by operating system services are collected. Each of these services is provided by a system task; for each of the system tasks the total execution time and the number of executions can be displayed on the next level. For special system tasks like the exchange processor or the job scheduler further information and statistics are supplied on the next level. By the example of the exchange package (EXP) requests, the presentation capabilities of the benchmark analysis package are shown. Either the total information for all 96 EXP requests is given in one bar chart, or, as in Fig. 15, selected values are collected in a table.

EXP requests	number of calls
Get current date (2)	18
Enter message in logfile (4)	1,132
Open dataset (8)	261
Close dataset (11)	55
Create/modify local dataset (12)	440
Get next control statement (14)	295
Permanent dataset management request (17)	138
Dispose dataset (21)	16
Return accumulated CPU-time (23)	154
Delay job (29)	15
Roll job (44)	0
Request memory (59)	77
Generate new task (63)	9
Manipulate HPM (71)	90

Fig. 15. Tool output for EXP requests

With this tool, the impact of inserted test jobs can be easily visualized. For the benchmark described in section 6.2, the total system activities measured by the SPM are listed in Tab. 6.

Time (seconds)	Inserted Test Jobs		
	Sequential (priority 7)	**Micro-tasking (priority 7)**	**Micro-tasking (priority 6)**
User	670.582	663.073	667.498
Wait-on-Semaphore	0.000	6.414	0.358
I/O	98.561	101.944	99.292
Operating system kernel	2.150	2.175	4.077
System wait	0.122	0.136	0.127
Operating system services	6.932	7.117	7.645

Tab. 6. Time distribution from the system performance monitor: Benchmark programs are running with priority 6, system load was kept fixed

The amount of user time is nearly the same as the time accumulated by the single jobs. Parameters which cannot be measured on a job basis like operating system activities can be stated by means of this tool.

As a first result of the investigations, the folklore of strongly increasing operating system overhead as a consequence of the usage of multitasking seems to be disproved. The additional system activities are only slightly raised, the wait-on-semaphore cycles can be decreased to nearly zero if the priority of the multitasking program is on the same level of priority or lower than with the other benchmark jobs. These results seem to be typical for the fine-grain multitasking concepts; for macrotasking jobs at least slightly increased CPU times are observed. In future, this benchmark environment can be used for further intensive examinations, and it will also be implemented under the UNICOS operating system on the CRAY Y-MP to study the efficiency of multitasking concepts in a UNIX multiprogramming job mix.

7. *Conclusion*

The functionality of the multitasking concepts varies widely from the parallel execution of subroutines with macrotasking, the introduction of parallelism with directives in microtasking, to the automatic parallelization of autotasking. For macrotasking, the availability of better tools for the detection of parallelism and the debugging of programs, working on an interprocedural basis, is urgent. The new autotasking concept is a superset of the earlier

microtasking implementation: all microtasking primitives are translatable to new autotasking directives, although both concepts can coexist within one program system. Therefore, autotasking will become the more important fine-grain concept. The compiling system *cf77* which implements the autotasking concept provides remarkable capabilities in the field of automatic parallelization. The analysis features still should be enhanced, but user intervention yet remains necessary in complex cases.

In future, the influence of multitasking concepts in a multiprogramming environment should be studied more intensively because the efficient support of such concepts by the operating system is one requirement for the acceptance of multitasking for production codes in computer center environments.

Acknowledgement

I want to thank F. Hossfeld and P. Weidner for their excellent support and continuous encouragement. I am grateful to K. Wingerath from the Department of Solid State Research (IFF) of KFA Jülich for permission to use the crystal growth simulation program, and to H. Reger who has provided the autotasking timings. Helpful discussions with U. Detert, W. Gürich, and J.-Fr. Hake are gratefully acknowledged.

References

[1] Autotasking User's Guide, CRAY-Research Inc, SN-2088 (1988).

[2] M. Bieterman, The impact of microtasking in a multiprogramming environment, *Proc. of CRAY User Group Meeting (Fall)* (1987) 144-148.

[3] M. Bieterman, Microtasking general purpose partial differential equation software on the CRAY X-MP, *The Journal of Supercomputing* **2** (4) (1988) 381-414.

[4] D.A. Calahan, Task granularity studies on a many-processor CRAY X-MP, *Parallel Computing* **2** (2) (1985) 109-118.

[5] COS Internal Reference Manual Vol.2: STP, CRAY-Research Inc, SN-0141A (1988).

[6] COS Performance Utilities Reference Manual, CRAY-Research Inc, SN-0146 (1987).

[7] Multitasking User Guide, Revision F, CRAY-Research Inc, SN-0222 (1989).

[8] U. Detert, Memory Performance of CRAY X-MP and CRAY Y-MP, to appear in *Proc. of CRAY User Group Meeting (Fall)* (1989).

[9] J.J. Dongarra and S.C. Eisenstat, Squeezing the Most out of an Algorithm in CRAY FORTRAN, *ACM Transactions on Mathematical Software* **10** (3) (1984) 219-230.

[10] M. Furtney, Parallel Processing at CRAY Research, *Proc. of CRAY User Group Meeting (Fall)* (1988).

[11] F. Hoßfeld, Vector-supercomputers, *Parallel Computing* **7** (1988), 373-385.

[12] F. Hossfeld, R. Knecht, and W.E. Nagel, Multitasking: Experiences with Applications on a CRAY X-MP, *Parallel Computing* **12** (3) (1989).

[13] S. Knecht, Möglichkeiten des Multitasking zur Beschleunigung von Standardalgorithmen, Jül-Spez-361, Kernforschungsanlage Jülich (1986).

[14] A. Liegmann, Die Strategie des Microtasking als Mittel zur Beschleunigung von Programmen auf dem Vektorrechner CRAY X-MP, Jül-Spez-435, Kernforschungsanlage Jülich (1988).

[15] M. Mihelcic and K. Wingerath, Numerical Simulations of the Czochralski Bulk Flow in an Axial Magnetic Field: Effects on the Flow and Temperature Oscillations in the Melt, *Journal of Crystal Growth* **71** (1985) 163-168.

[16] W.E. Nagel, Using Multiple CPUs for Problem Solving - Experiences in Multitasking on CRAY X-MP/48, *Parallel Computing* **8** (1988) 223-230.

[17] W.E. Nagel, Exploiting Autotasking on a CRAY Y-MP: An Improved Software Interface to Multitasking, to appear in *Parallel Computing*.

[18] W.E. Nagel and F. Szelényi, Multitasking on Supercomputers: Concepts and Experiences, IBM Tech. Rep. ICE-VS05, IBM ECSEC (1989).

[19] W.E. Nagel and K. Wingerath, Three-Dimensional Numerical Simulations of the Czochralski Bulk Flow on a CRAY X-MP Multiprocessor Architecture, *Proc. 1988 International Conference on Supercomputing* (1988) 266-272.

[20] W. Oed and O. Lange, On the effective bandwidth of interleaved memories in vector processor systems, *IEEE Trans. Computers C-34* (1985) 949-957.

[21] J.M. Ortega, Introduction to parallel and vector solution of linear systems, Plenum Press, New York, 1988; also: The *ijk* forms of factorization methods I. Vector computers, *Parallel Computing* **7** (2) (1988) 135-147.

[22] C.D. Polychronopoulos and D.J. Kuck, Guided Self-Scheduling: A practical scheduling scheme for parallel supercomputers, *IEEE Trans. Comput. C-36* (1987) 1425-1439.

[23] H. Reger, Ein Vergleich der Multitasking-Implementierungen auf CRAY X-MP und IBM 3090, Diplomarbeit RWTH Aachen (1989).

MODELISATION NUMERIQUE

NUMERICAL MODELS

Wavelet Solution of Linear and Nonlinear Elliptic, Parabolic and Hyperbolic Problems in One Space Dimension

Roland Glowinski*, Wayne Lawton**, Michel Ravachol***, Eric Tenenbaum**

Abstract

In this exploratory paper we discuss the Daubechies wavelet solution of boundary value problems and initial boundary value problems for ordinary and partial differential equations in one space dimension. The theoretical and numerical results suggest that for the above class of problems wavelets provide a robust and accurate alternative to more traditional methods such as finite differences and finite elements. The one dimensional analysis done in this paper can also be seen as a necessary step to the solution of mulidimensional problems where various technical issues remain to be resolved.

* University of Houston, AWARE, Inc., and INRIA
** AWARE, Inc., 124 Mount Auburn Street, Cambridge, MA, 02138, USA
***University of Houston, AWARE, Inc., and Dassault Industries

Table of Contents

Introduction

This paper describes selected developments in wavelet-based numerical methods for solving partial differential equations (PDE's). The scope of this paper is limited to a small but representative set of equations in one space variable. Methods for solving problems in two and three dimensions are currently being developed.

The wavelet-based methods adapt a Ritz-Galerkin technique that uses functions associated with the orthonormal bases of compactly supported wavelets constructed in [Daubechies, 1988]. As in the case of the earlier orthonormal wavelet basis constructed in [Meyer, 1985], Daubechies' bases are unconditional bases for the Sobolev spaces and therefore provide accurate approximations of PDE's solutions. Furthermore, the 'multiresolution analysis' properties of these bases, described in [Mallat, September 1987] and [Meyer, 1986], are ideally suited for multigrid methods and adaptive grid refinement methods. Finally, the compact support of the basis functions makes the 'wavelet transform' algorithm, described in [Mallat, May 1987], extremely efficient for computing numerical solutions of PDE problems.

Numerical solution of a PDE problem requires a discretization method that reduces the problem to finding the solution of an algebraic equation (AE) in a finite number of unknowns. The latter problem is amenable to digital computation. PDE's involve functions that model spatially distributed, and possibly time varying, physical quantities such as temperature, velocity, or displacement, and differential operators that model the physical processes which determine the static or dynamic behavior of these quantities. Discretization methods, such as finite differences, finite elements, and spectral methods, represent the solution function u by an approximation v defined by N discrete parameters. Then the differential operator and the constraints such as initial values and boundary conditions are approximated by algebraic operations involving these parameters. This results in an AE whose exact solution determines v. A discretization method is effective if the truncation error u-v tends to 0 'rapidly' as N increases.

Numerical solution of a PDE problem also requires a method for solving the resulting AE that results from discretization. It is only required to obtain an approximate solution w of the AE such that the algebraic error v-w is comparable to the truncation error u-v. Since the total error e = u-w satisfies e = truncation error + algebraic error, if v-w ≈ u-v then additional computation is more effectively utilized by increasing the number of discretization parameters to decrease the truncation error. Algebraic solution methods consist of direct methods, such as LU factorization, which yield a solution accurate to within finite word length limitations, and iterative methods, such as conjugate gradients and multigrid , which yield a solution whose accuracy increases with the number of iterations.

The analytical and computational properties of scaling functions and wavelet functions provide powerful numerical methods for discretizing PDE's and for solving the resulting AE's. Our paper is organized as follows.

Section 1 reviews the construction of the scaling functions and wavelet functions described in [Daubechies, 1988] and derives the analytical and computational properties that address the requirements for solving PDE's. Sobolev approximation properties of the scaling functions in $H^m(R)$ and of their restrictions to (0,1) in $H^m(0,1)$ are derived as a consequence of the vanishing moments properties of the associated wavelets. These properties provide effective wavelet-based discretization methods. The computational properties of wavelets are derived from the hierarchical structure of the scaling functions. These properties include algorithms for expanding functions in wavelet bases from thier sampled values and for representing differential operators with respect to wavelet bases.

Section 2 addresses general results concerning the existence, uniqueness, and approximation of the solutions of linear variational problems in Hilbert spaces. Examples concerning elliptic boundary value problems in one space dimension are used to illustrate some of the above notions.

Section 3 discusses wavelet-based solutions of second order linear elliptic PDE's with Neumann and Dirichlet boundary conditions. A variational formulation of the Dirichlet problem is used to reduce it to a small number of Neumann problems involving Lagrange multipliers. A Galerkin basis consisting of translates of a dilated scaling function, truncated so as to have support in the interval (0,1), is used to discretize the Neumann problem. The resulting AE's are solved using LU and Choleski factorization methods. Numerical results are presented to illustrate the effectiveness of the new methodology.

Section 4 discusses the wavelet-based solution of singularly perturbed second order linear elliptic boundary value problems. Solutions of these problems may exhibit boundary layers requiring higher resolutions where the strong gradients exist. A domain decomposition method is used to compute and match wavelet based solutions in the low and high gradient regions.

Section 5 discusses multigrid methods for solving the second order linear elliptic problems with periodic boundary conditions.

Section 6 discusses wavelet-based solutions of linear and nonlinear parabolic equations in one space dimension. Here space approximations using wavelets are combined with time stepping methods to solve the one dimensional heat equation and the regularized Burgers equation. Furthermore, in the case of the Burgers equation, we discuss the use of basis consisting of wavelets together with scaling functions in order to filter spurious oscillations developoed in the shock region.

Section 7 discusses the wavelet-based solution of the linear advection equation $\partial u/\partial t + \partial u/\partial x = 0$. As in Section 6 a wavelet approximation for the space variable is combined with time stepping methods.

Finally, Section 8 provides further comments together with our conclusions on these preliminary experiments with wavelets as tools for solving differential equations.

1 Description and Basic Properties of the Daubechies Scaling Functions and Wavelet Functions

1.1 Description of Daubechies' Functions.

The Daubechies scaling functions and wavelet functions constructed in [Daubechies, 1988] are considerably more complex and elusive than the more familiar elementary functions. Therefore, it is convenient to consider these functions as generalizations of a simpler set of functions described below.

Define functions ϕ and ψ by :

(1.1) $\phi(x) = 1$ for $0 \le x < 1$, else $\phi(x) = 0$,

(1.2) $\psi(x) = 1$ for $0 \le x < 1/2$, $\psi(x) = -1$ for $1/2 \le x < 1$, else $\phi(x) = 0$.

and define $\phi_{j,k}(x) = 2^{j/2}\phi(2^j x-k)$ and $\psi_{j,k}(x) = 2^{j/2}\psi(2^j x-k)$ for any pair of integers (j,k). The functions $\{\psi_{j,k} : j \ge 0, k = 0,...,2^j-1\}$ are the classical Haar functions, first introduced in [Haar, 1910] to provide an orthonormal basis for $L^2(0,1)$. The Daubechies scaling functions and wavelet functions are generalizations of the Haar scaling functions $\{\phi_{j,k}\}$ the Haar wavelet functions $\{\psi_{j,k}\}$. We will describe the properties of the Haar functions using subspaces V_n and W_n of $L^2(R)$, n any integer, defined by :

(1.3) V_n = closure of linear space spanned by $\{ \phi_{n,k} : k$ an integer$\}$,

(1.4) W_n = closure of linear space spanned by $\{ \psi_{n,k} : k$ an integer$\}$.

Then the following properties hold:

(1.5) $V_{n+1} \supset V_n$ for every integer n,

(1.6) Closure $(\bigcup_n V_n) = L^2(R)$,

(1.7) $\{\phi_{n,k} : k$ an integer$\}$ is an orthonormal basis for V_n for every integer n,

(1.8) W_n is the orthogonal complement of V_n in V_{n+1},

(1.9) $\{\psi_{n,k} : k \text{ an integer}\}$ is an orthonormal basis for W_n for every n,

(1.10) $\phi_{j,k}$ and $\psi_{j,k}$ have compact support for all integers j, k,

(1.11) $\int \phi_{j,k}(x)dx = 2^{-j/2}$ and $\int \psi_{j,k}(x)dx = 0$ for all integers j and k.

A further consequence of properties (1.5),(1.6), and (1.8) is

$$(1.12) \quad L^2(R) = V_n \oplus \sum_{j \geq n} \oplus W_j = \sum_{j} \oplus W_j, \quad \text{for every integer n,}$$

Now let P_n denote the orthogonal projection of $L^2(R)$ onto V_n and let Q_n denote the orthogonal projection of $L^2(R)$ onto W_n, therefore $P_{n+1} = P_n + Q_n$. Then for any $f \in L^2(R)$, $P_n(f)$ yields a 2^{-n} 'low resolution' approximation to f and a 'higher resolution' approximation $P_{n+1}(f)$ may be obtained by adding a 'high frequency' component $Q_{n+1}(f)$.

For every integer $N \geq 1$ Daubechies constructs a pair of functions ϕ and ψ that are the functions in (1.1),(1.2) for N = 1 and that generalize these functions for N > 1. Her construction involves the following steps:

Step 1 Construct a finite sequence h(0),...,h(2N-1) satisfying :

$$(1.13) \quad \sum_k h(k)h(k+2m) = \delta_{0m}, \text{ for every integer m,}$$

$$(1.14) \quad \sum_k h(k) = \sqrt{2},$$

$$(1.15) \quad \sum_k g(k)k^m = 0, \text{ whenever } 0 \leq m \leq N-1, \text{ where } g(k) = (-1)^k h(-k+1).$$

(Note that for m = 0, equation (1.15) is implied by equation (1.13) and equation (1.14))

Step 2 Construct the trigonometric polynomial $m_0(y)$ by

$$(1.16) \quad m_0(y) = \sqrt{2}\sum_k h(k)\exp(iky),$$

Step 3 Construct the scaling function ϕ so its Fourier transform $\phi\hat{}$ satisfies :

(1.17) $\phi\hat{}(y) = (1/\sqrt{2\pi}) \prod m_0(2^{-k}y)$,

Step 4 Construct the wavelet function ψ by

(1.18) $\psi(x) = \sum_k g(k)\phi(2x-k)$.

For $N > 1$ these functions define sets $\{\phi_{n,k}\}$, $\{\psi_{n,k}\}$ and subspaces V_n and W_n of $L^2(R)$ satisfying properties (1.3)-(1.11) and, in addition, the following properties:

(1.19) $\phi_{n,k} = \sum_j h(j-2k)\phi_{n+1,j}$ and $\psi_{n,k} = \sum_j g(j-2k)\phi_{n+1,j}$ for any integer n,

(1.20) support$(\phi_{n,k}) = [2^{-n}k, 2^{-n}(k+2N-1)]$, support$(\psi_{n,k}) = [2^{-n}(k+1-N), 2^{-n}(k+N)]$,

(1.21) $\int \psi_{j,k}(x)x^m dx = 0$ for all integers j and k and any integer $0 \le m < N-1$,

(1.22) $\phi_{j,k}$ and $\psi_{j,k} \in C^{\lambda(N)}$ = space of Holder continuous functions with exponent $\lambda(N)$, where $\lambda(2) = 2-\log_2(1+\sqrt{3}) \approx .5500$, $\lambda(3) \approx 1.087833$, $\lambda(4) \approx 1.617926$, and $\lambda(N) \approx 0.3485N$ for large N.

Remark 1.1 Equations (1.19)-(1.21) are derived from the properties of $\phi_{n,k}$ and $\psi_{n,k}$ in [Daubechies, 1988]. Property (1.22) is derived in [Daubechies and Lagarias]. Graphs of the Daubechies scaling functions and wavelet functions for $2 \le N \le 4$ are illustrated in Figure 1.1.

1.2 Approximation Properties of Daubechies' Functions.

For $m \ge 1$ and Ω an open interval of R (e.g. $\Omega = (0,1)$ or $\Omega = R$) define spaces:

(1.23) $H^0(\Omega) = L^2(\Omega)$ with the standard Hilbert Space inner product $\langle .,. \rangle$,

(1.24) $H^m(\Omega) = \{f \in H^{m-1}(\Omega) : f' \in H^{m-1}(\Omega)\}$ with Hilbert Space inner product $(.,.)_m$ defined inductively by $(.,.)_0 = \langle .,. \rangle$ and $(f,g)_m = \langle f,g \rangle + (f',g')_{m-1}$, and associated norm $\|.\|_m$,

(1.25) $D(\Omega)$ = space of infinitely differentiable functions with compact support contained in Ω (i.e. if $\Omega = (0,1)$ these functions 'vanish' at 0 and 1)

(1.26) $H_0^m(\Omega)$ = closure of $D(\Omega)$ in $H^m(\Omega)$ with respect to the norm $\|.\|_m$.

(1.27) for any fixed integer $N \geq 1$ and any integer n, $V_n(W)$ = restriction to Ω of functions in V_n where V_n is defined as in (1.3) by the scaling function corresponding to N, i.e. $V_n(R) = V_n$.

For any integer $m \geq 0$, closed interval I, and function $f : I \rightarrow R$ we define

$$(1.28)\quad E_m(f,I) = \min_{p(x)} \{ \max_{x \in I} |f(x) - p(x)| \},$$

where p(x) ranges over all polynomials of degree $\leq m$. Then

Lemma 1 If $N \geq 1$, $f \in D(R)$ with $B = \max\{|f^{(N)}(x)|\}$, then for any $I = [a,b]$,

$$(1.29)\quad E_{N-1}(f,I) \leq 2B(b-a)^N/(4^N N!).$$

Proof This is one form of Jackson's theorem, ref. [Dahlquist and Bjorck, 1974].

Lemma 2 If f and B are as above and ψ is a wavelet with $N \geq 1$, then

$$(1.30)\quad |\langle f,\psi_{j,k}\rangle| \leq C2^{-j(N+1/2)} \text{ where } C = 2B(2N-1)^{N+1/2}/(4^N N!), \text{ for all integers j, k.}$$

Proof Follows from equation (1.21), lemma 1.1, and Schwarz's inequality.

Lemma 1.3 If f and C are as above, let $m \geq 0$, and choose $N > m$ such that the associated scaling function ϕ and wavelet function ψ belong to $H^m(R)$ (the existence of such an integer is implied by property (1.22)), let n be any integer, let V_n be as in (1.3) and let P_n denote orthogonal projection onto V_n, then

$$(1.31)\quad \|f - P_n(f)\|_m \leq \sum_{j \geq n} \sum_{k} |\langle f,\psi_{j,k}\rangle| \, \|\psi_{j,k}\|_m \leq D2^{-n(N-m)}.$$

where $D = CE\|\psi\|_m/(1-2^{-(N-m)})$ and $E^2 = (2N-1)$x(length of smallest interval containing support (f)).
Proof Follows from (1.30) and the fact that $\|\psi_{j,k}\|_m \le 2^{jm}\|\psi\|_m$.

Lemma 1.4 The set of restrictions of functions in D(R) to Ω is dense in $H^m(W)$ for every $m \ge 0$.

Proof This is the classical density theorem which follows from the classical prolongation theorem for Sobolev spaces, [Adams, 1975], [Brezis, 1983]

We are now prepared to derive the main result of this section that, together with the general approximation results in section 2, provides the mathematical justification for wavelet-based solution methods for boundary value problems discussed in later sections.

Theorem 1.1 Let $m \ge 0$ and choose N, ϕ, and ψ as in lemma 1.3, let $g \in H^m(\Omega)$, then for any $\varepsilon > 0$ there exists an integer n such that

(1.32) $\|g-h\|_m < \varepsilon$,

where h is the restriction to Ω of a function in V_n.

Proof Follows from lemma 1.3 and lemma 1.4.

Remark 1.2 It follows from property (1.22) that for m = 1, N = 3 satisfies the hypothesis of theorem 1.1. This choice is adequate to treat second order elliptic boundary value problems in one space dimension. For m = 2, N = 7 satisfies the hypothesis of theorem 1.1 and hence is adequate to treat fourth order elliptic boundary value problems.

1.3 Computational Properties of Daubechies' Functions.

Computational properties are mathematical properties that are related to algorithm requirements and algorithm performance. This section will discuss computational properties related to both direct algorithms and iterative algorithms.

1.3.1 The Mallat Transform.

Fix an integer $N \ge 1$ and let ϕ and ψ be the associated scaling function and wavelet function. For any integer n let V_n and W_n be defined as in (1.3),(1.4). Then from properties (1.7)-(1.9) it follows that every $f \in V_{n+1}$ admits the representation

$$(1.33)\quad f(x) = \sum_k c(k)\phi_{n+1,k} = \sum_k a(k)\phi_{n,k} + \sum_k b(k)\psi_{n,k}$$

where a,b, and c are square summable sequences. Then by property (1.19) the sequences a and b are determined from the sequence c by a unitary transformation T: $L^2(Z)xL^2(Z) \rightarrow L^2(Z)$, given by

$$(1.34)\quad T(a,b) = c \text{ where } c(k) = \sum_j h(k-2j)a(j) + \sum_j g(k-2j)b(j).$$

The inverse $T^{-1} : L^2(Z) \rightarrow L^2(Z)xL^2(Z)$ is given by

$$(1.35)\quad T^{-1}(c) = (a,b) \text{ where } a(k) = \sum_j h(j-2k)c(j) \text{ and } b(k) = \sum_j g(j-2k)c(j).$$

Remark 1.3 The transformation T is called the Mallat transformation and is described in detail in ref. [Mallat, May 1987]. For any integer n, integer $J \geq 1$, and $f \in L^2(R)$, the functions $Q_{n-1}(f), Q_{n-2}(f), \ldots, Q_{n-J}(f), P_{n-J}(f)$ can be computed by first computing $P_n(f)$ and then applying the Mallat transform J times. This yields a 'multiresolution' analysis of a function f. The computation of T or T^{-1} requires *only* N multiplies and N-1 additions for each 'output value'.

1.3.2 Computing the Expansion of Functions.

We will now discuss how to compute the expansion of functions with respect to a 'wavelet' basis from sampled values of the function. This 'expansion' problem is formalized as follows. Fix an integer $N \geq 1$ and let ϕ and ψ be the associated scaling function and wavelet function as in Section 1.3.1. Let n be any integer, let V_n be defined by (1.3), and let P_n denote the orthogonal projection of $L^2(R)$ onto V_n. The 'expansion' problem consists of computing an approximation to $P_n(f)$ where $f \in L^2(R)$. This is equivalent to the problem of evaluating an approximation to the integrals

$$(1.36)\quad c(k) = \int_{2^{-n}k}^{2^{-n}(k+2N-1)} \phi_{n,k}(x)f(x)dx \qquad \text{for every integer k.}$$

This requires an approximate knowledge of f(x) over each dyadic interval $I_{n,k}$ having the form $I_{n,k} = [2^{-n}k, 2^{-n}(k+2N-1)]$.

Our general approach to this approximation problem is described as follows. For each integer k let $g_k(x)$ be a function (perhaps a distribution) defined on $I_{n,k}$ that approximates (perhaps in the weak sense) the restriction of f to $I_{n,k}$, then define

$$(1.37)\quad d(k) = \int_{2^{-n}k}^{2^{-n}(k+2N-1)} \phi_{n,k}(x) g_k(x)\,dx \qquad \text{for every integer } k,$$

to approximate c(k) for every integer k, then define an approximation $P_n\tilde{}(f)$ to $P_n(f)$ by

$$(1.38)\quad P_n\tilde{}(f) = \sum_k d(k)\phi_{n,k}$$

The resulting error function can be related to the errors |d(k)-c(k)| by applying Holder type inequalities to the equation

$$(1.39)\quad P_n\tilde{}(f)(x) - P_n(f)(x) = \sum_{k \ge 2^n x - 2N + 1}^{k \le 2^n x} (d(k)-c(k))\phi_{n,k}(x)$$

We describe two specific techniques for computing approximations d(k) to the expansion coefficients c(k) of $P_n(f)$ from sampled values of f at dyadic rationals $\{f(2^{-n-s}j) : j \in Z\}$ where $s \ge 0$. Their performance, measured by the computational complexity and by the asymptotic bounds for the error |d(k) - c(k)| as n and s are large, is discussed under the assumption that f is infinitely differentiable and that $\phi(x)$ has Holder exponent $\lambda(N) > 1$, (i.e. $N \ge 3$ and $\phi(x)$ is continuously differentiable). These assumptions are made in Section 5 to discuss the computational complexity of wavelet-based multigrid methods for certain classes of problems.

Approximation Technique 1 Choose an integer $s \ge 0$ and let $g_k(x)$ be the measure

$$(1.40)\quad g_k(x) = 2^{-n-s} \sum_{j = 2^s k}^{j = 2^s(k + 2N - 1)} f(2^{-n-s} j)\, \delta(x - 2^{-n-s} j)$$

to obtain

$$(1.41)\quad d(k) = 2^{-s-n/2} \sum_{j=0}^{2^s(2N - 1)} \phi(2^{-s} j)\, f(2^{-n} k + 2^{-n-s} j)$$

This requires $\approx 2^s(2N - 1)$ operations to compute each d(k) and therefore it requires $\approx 2^{n+s}(2N - 1)$ operations to compute the expansion of $P_n(f)$ per unit interval. Under the assumptions on f and ϕ above, it can be proved that there exists constants C_1 and C_2 such that

$$(1.42) \quad |d(k) - c(k)| < C_1 2^{-3n/2-s} + C_2 2^{-3n/2-\lambda s} \quad \text{where } \lambda = \min\{1, \lambda(N) - 1\}$$

Remark 1.4 In [Resnikoff, March 1988] explicit expressions for the values of ϕ at dyadic rationals as rational functions of the parameters $\{h(i) : i = 0,...,2N-1\}$ that define ϕ in equations (1.13)-(1.18) are derived and the following equation is derived

$$(1.43) \quad 2^{-s} \sum_{j=0}^{2^s(2N-1)} \phi(2^{-s}j) = 1$$

This important result is required to derive relation (1.42) as well as to compute d(k) in equation (1.41).

Remark 1.5 Relation (1.42) implies that if $\lambda < 1$, a more efficient algorithm for computing the expansion $P_m(f)$ is obtained by combining the approximation technique above with the inverse Mallat transform as follows. First choose an integer $n > m$ and an integer $s \geq 0$ and calculating $\{d(k)\}$ for the expansion $P_n(f)$. Then compute n - m stages of the inverse Mallat transform as described in Remark 1.3 to obtain $P_m(f)$. Optimizing the choice of n and s requires specification of the required accuracy and the value of the constants C_1 and C_2 in relation (1.42).

Approximation Technique 2 Choose integers $s \geq 0$, $p \geq 1$ and q and choose $g_k(x)$ to be the Lagrange interpolating polynomial to f(x) at the set of points $\{2^{-n-s}(2^s k + q + j) : 1 \leq j \leq p\}$. This polynomial is defined by

$$(1.44) \quad g_k(x) = \sum_{j=1}^{j=p} f(2^{-n-s}(2^s k + q + j))L_j(2^{n+s}x - 2^s k - q)$$

where

$$(1.45) \quad L_j(y) = \prod_{\substack{i=1 \\ i \neq j}}^{i=p} [(y - i)/(j - i)], \qquad \text{for } j = 1,...,p$$

to obtain

$$(1.46)\quad d(k) = \sum_{j=1}^{j=p} M(j)\, f(2^{-n-s}(j + 2^s k + q))$$

where

$$(1.47)\quad M(j) = 2^{-n/2} \int_0^{2N-1} L_j(2^s x - q)\phi(x)dx \qquad \text{for } j = 1,...,p$$

This requires p operations to compute each d(k) and thus $\approx 2^n p$ operations to compute the expansion of $P_n(f)$ per unit interval. Under the assumptions on f and ϕ above, Newton's general interpolation formula [Dahlquist and Bjorck,1974] implies there exists a constant C_3 such that

$$(1.48)\quad |d(k) - c(k)| \le C_3 2^{-(p+1/2)n}$$

Clearly, for $p \ge 2$, this technique is asyptotically more efficient than approximation technique 2. Furthermore, the M(j) are linear combinations of the 0,...,p - moments of ϕ. The latter can be expressed as rational functions of the $\{h(i) : i = 0,...,2N-1\}$ using the results in [Resnikoff, March 1988].

1.3.3 Representing Differential Operators in Wavelet Bases

Let $m \ge 0$ and let $N > m$ be such that the associated scaling function $\phi \in H^m(R)$. Let V_n be defined as in equation (1.27) and let P_n denote the projection of V onto V_n. Let $H^m(R_p)$ denote the set of functions in $H^m(R)$ that are periodic of period 1 (intuitively, R_p represents a circle) together with the inner product

$$(1.49)\quad (f,g)_m = \sum_{j=0}^{m} \int_{x=0}^{x=1} f^{(j)}(x) g^{(j)}(x)dx,$$

and associated norm $\|.\|_m$. Furthermore, let $V_n(R_p)$ denote $V_n \cap H^m(R_p)$. Then clearly

$$(1.50)\quad V_n(R_p) = \{f \in H^m(R) : f = \sum_k c(k)\phi_{n,k} \text{ and } c(k+2^n) = c(k) \text{ for all } k\}$$

Let m, N, and ϕ be as above and let V denote either $H^m(\Omega)$, for some interval Ω of R, or $H^m(R_p)$.

For any integer n let V_n denote either $V_n(\Omega)$, defined as in equation (1.27), or $V_n(R_p)$, defined as in equation (1.50). For any operator A : V --> V^* (the dual space of V) and for any integer n, let A_n : V --> V^* denote the operator defined by

(1.51) $\langle A_n(f),g\rangle = \langle P_n^*(A(P_n(f))),g\rangle = \langle A(P_n(f)),P_n(g)\rangle$, for every f, g $\in$ V,

where $\langle .,. \rangle : V^* \times V \to R$ denotes the canonical pairing and P_n^* denotes the adjoint of P_n. Clearly, the sequence $\{A_n\}$ provides a sequence of approximations to A as n increases. Furthermore, the operator A_n is defined by the set

(1.52) $\{a(n,i,j) = \langle A_n(b_i),b_j\rangle : \{b_i\}$ form a basis for $V_n\}$.

The techniques in [Resnikoff, March 1988] provide explicit formuli for calculating $\{a(n,i,j)\}$ where A is any differential operator whose coefficients are piecewise polynomial functions whose associated intervals have dyadic rational endpoints. In this case, each a(n,i,j) is a rational function of the scaling parameters {h(k)} that define ϕ in equations (1.13)-(1.17).Furthermore, a(n,i,j) = 0 whenever $|i-j| \geq 2N-1$.

For the remainder of this section we will assume that either V = $H^m(R)$ or V = $H^m(R_p)$ and that A is a differential operator having constant coefficients. Then clearly for any integer n, a(n,i,j) is a function of n and of j-i which we will denote by a(n,j-i) (arithmetic on indices will be considered modulo 2^n if V = $H^m(R_p)$). Identify V_n with its dual in $L^2(R)$. Then the operator A_n, with respect to either the basis $\{\phi_{n,k} : k$ an integer$\}$ of $H^m(R)$ or the basis $\{\phi_{n,k} : 0 \leq k < 2^n\}$ of $H^m(R_p)$, is represented as convolution with the function a(n,.). Therefore, the eigenfunctions of A_n have the form

$$(1.53)\quad F_\omega = \sum_k \exp(i\omega k)\phi_{n,k}$$

and have corresponding eigenvalues that are expressed in terms of the Fourier series of a(n,.) by

$$(1.54)\quad A_n(F_\omega) = \hat{a}(n,\omega)F_\omega, \text{ where } \hat{a}(n,\omega) = \sum_k a(n,k)\exp(i\omega)$$

where for V = $H^m(R)$, $0 \leq \omega < 2\pi$ and k is summed over all integers, and for V = $H^m(R_p)$, ω assumes the discrete values $\{2\pi k/2^n : 0 \leq k < 2^n\}$ and k is summed over $0 \leq k < 2^n$. The function $\hat{a}(n,\omega)$ will be called the spectrum of the operator A_n. Clearly, if V = $H^m(R)$ and if A = d^r/dx^r (r-th derivative operator), then $\hat{a}(n,\omega) = 2^{nr}\hat{a}(0,\omega)$. Figure 1.2 illustrates the spectrum of the operator A_0

for $A = d^2/dx^2$ where ϕ corresponds to $N = 3$ and $V = H^1(R)$. In this case $a(0,0)=-5.2679$, $a(0,k)=0$ for $k \geq 5$, and $a(0,.)$ is symmetric (therefore $\hat{a}(0,w)$ is real). The spectrum of differential operators will be used extensively in Section 5 and in Section 6.1.

2 Classical Results Concerning the Approximation of Linear Variational Problems

2.1 Linear Variational Problems in Hilbert Spaces.

All the linear elliptic boundary value problems to be discussed in this paper can be reduced to the following (variational) formulation

$$\begin{aligned} &\text{Find } u \in V \text{ such that} \\ &a(u,v) = L(v), \text{ for every } v \in V. \end{aligned} \tag{2.1}$$

In (2.1) V, a(.,.), L are as follows:

(i) V is a Hilbert Space (real for simplicity) with scalar product (.,.) and associated norm ||.||.

(ii) a: VxV --> R is a bilinear form (possibly non symmetric), continuous, and V-elliptic over VxV; the last property means that there exists $\alpha > 0$ such that

$$a(v,v) \geq \alpha \|v\|^2, \text{ for every } v \in V. \tag{2.2}$$

(iii) L: V --> R is linear and continuous.

If properties (i)-(iii) hold it follows then from the Lax-Milgram Theorem that problem (1.1) has a unique solution.

Remark 2.1 If the bilinear form a(.,.) is symmetric (i.e. a(v,w) = a(w,v) for every v, w in V) then problem (1.1) is equivalent to the following minimization problem:

$$\begin{aligned} &\text{Find } u \in V \text{ such that} \\ &J(u) \leq J(v), \text{ for every } v \in V, \end{aligned} \tag{2.3}$$

where J is defined by

(2.4) $J(v) = (1/2)a(v,v) - L(v)$.

Indeed, equation (2.1) can be seen as the Euler-Lagrange equation associated to problem (2.3).

Remark 2.2 Let V^* denote the dual space of V and let $<.,.> : V^* \times V \to R$ denote the canonical duality pairing. Then it follows from the Riesz Representation Theorem that there exists $A \in Isom(V,V^*)$, uniquely defined, such that

(2.5) $a(v,w) = <Av,w>$, for every $v, w \in V$.

Then, for notational convenience, introduce $l \in V^*$ such that

(2.6) $L(v) = <l,v>$, for every $v \in V$.

Problem (2.1) is therefore equivalent to the following 'linear equation'

(2.7) $Au = l$.

Remark 2.3 It follows from (2.5) and (2.6) that

(2.8) $|a(v,w)| \leq \|A\| \|v\| \|w\|$, for every $v, w \in V$,

(2.9) $a(v,v) \geq \|A^{-1}\|^{-1} \|v\|^2$, for every $v \in V$,

where $\|A\|$ and $\|A^{-1}\|$ are the standard operator norms. Indeed, the largest constant α in (2.2) is precisely $\|A^{-1}\|^{-1}$.

2.2 Examples

We illustrate the above generalities by discussing below the variational formulation of some boundary value problems for second order differential equations.

Example 2.1 To illustrate the above generalities, lets consider the following homogeneous Dirichlet problem on the interval [0,1] :

(2.10) $-u'' + \sigma u = f$ in (0,1),

(2.11) $u(0) = u(1) = 0$,

where in (2.10) σ is a positive constant and $f \in L^2(0,1)$ (actually σ can even be 'slightly' negative and f can be less regular than an L^2 function). It can be shown that solving problem (2.10),(2.11) in $H_0^1(0,1) = \{u \in H^1(0,1): u(0)=u(1)=0\}$ is equivalent to solving a linear variational problem of the form (2.1) with:

(i) $V = H_0^1(0,1)$; $(v,w) = \int_0^1 (v'w' + vw)dx$, $\|v\| = (v,v)^{1/2}$.

(ii) $a(v,w) = \int_0^1 (v'w' + \sigma vw)dx$.

(iii) $L(v) = \int_0^1 fvdx$.

It can be shown that the hypothesis of the Lax-Milgram Theorem are satisfied here implying that the corresponding problem (2.1) has a unique solution in $H_0^1(0,1)$ which is also the unique solution of (2.10),(2.11) in the above space. In this case, V^* can be identified with the dual space $H^{-1}(0,1)$ of $H_0^1(0,1)$ and A with the operator $A: H_0^1(0,1) \dashrightarrow H^{-1}(0,1)$ defined by $A(v) = -v'' + \sigma v$. This concludes (momentarily) the Dirichlet problem.

Example 2.2 Now consider the following inhomogeneous Neuman problem on the interval [0,1] :

(2.12) $-u'' + \sigma u = f$ in $(0,1)$,

(2.13) $-u'(0) = c$, $u'(1) = d$,

where in (2.12) σ is a positive constant and $f \in L^2(0,1)$. It can be shown that solving problem (2.12),(2.13) in $H^1(0,1)$ is equivalent to solving a linear variational problem of the form (2.1) with

$V = H^1(0,1)$, $a(.,.)$ the same as in example 2.1, and L given by:

$$\text{(i)}\quad L(v) = \int_0^1 fv\,dx + cv(0) + dv(1).$$

As in example 2.1, it can be shown that the hypothesis of the Lax-Milgram Theorem are satisfied here implying that the corresponding problem (2.1) has a unique solution in $H^1(0,1)$ which is also the unique solution of (2.12),(2.13) in the above space. In this case it is very convenient to identify the space $(H^1(0,1))^*$ with $H^{-1}(0,1) \times R^2$; then the operator A is defined by

$$(2.14)\quad Av = \{-v'' + sv,\ \{(v-v_0)'(1),\ -(v-v_0)'(0)\}\ \},$$

where v_0 is the unique solution in $H_0^1(0,1)$ of the Dirichlet problem

$$(2.15)\quad -v_0'' + v_0 = -v'' + \sigma v,$$

$$(2.16)\quad v_0(0) = v_0(1) = 0.$$

Equation (2.15) holds in $H^{-1}(0,1)$ and it is quite obvious that v_0 depends linearly and continuously on v.

2.3. General Approximation Results.

Concerning now the approximation of problem (2.1),(2.7) we consider a family $\{V_n\}_n$ of closed subspaces of V; the V_n are finite dimensional in practice. On V_n it is then quite natural to 'approximate' problem (2.1) by

(2.17)
Find $u_n \in V_n$ such that

$a(u_n,v) = L(v)$ for every $v \in V_n$.

Problem (1.17) obviously has a unique solution from the Lax-Milgram Theorem. It is then a simple exercise to prove the following approximation property:

$$(2.18)\quad \|u_n - u\| \le \|A\|\,\|A^{-1}\|\,\|v - u\|, \text{ for every } v \in V_n.$$

If the bilinear form a(.,.) is symmetric then the above inequality can be refined to yield

(2.19) $\|u_n - u\| \leq (\|A\| \|A^{-1}\|)^{1/2} \|v - u\|$, for every $v \in V_n$.

Remark 2.4 Relation (2.18) clearly implies

(2.20) $\|u_n - u\| \leq \|A\| \|A^{-1}\| \text{ Inf } \{ \|v - u\| : v \in V_n \}$;

and an analogous relation follows from (2.19).

From the results above we have

(2.21) $\lim_{n \to +\infty} \|u_n - u\| = 0$ if $\lim_{n \to +\infty} (\text{Inf } \{ \|v - u\| : v \in V_n \}) = 0$.

In the case where $V_1 \subset V_2 \subset \ldots V_{n-1} \subset V_n$ ... then (2.21) is automatically satisfied if the closure of $\bigcup_n V_n$ is equal to V.

In the following sections we shall use the properties of the Daubechies scaling functions and wavelet functions discussed in Section 1 to construct subspaces V_n of $H^1(\Omega)$ (where Ω is an open interval of R) satisfying relation (2.21).

3 Wavelet Solution of Linear Elliptic Problems

In this section we shall discuss the numerical solution of linear elliptic problems in one space dimension. We shall consider the Neumann problem first since the solution methodology for the Dirichlet problem that we discuss in this section is essentially based on the solution of a very small number of Neumann problems with Lagrange multipliers on the right hand side. Using numerical experiments we shall evaluate the quality of the wavelet solution; indeed the results to be described in the following sections seem to indicate that wavelets provide accurate approximate solutions, for linear elliptic boundary value problems in one space dimension.

3.1 Solution of the Neumann Problem

3.1.1 Formulation of the Neumann Problem

The Neumann problem to be discussed in this section can be formulated as follows:

(3.1) $-(\alpha u')' + \beta u + \gamma u' = f$ in $(0,1)$,

(3.2) $-(\alpha u')(0) = c$, $(\alpha u')(1) = d$.

Using the approach described in Section 2.2, Example 2.2, it can be shown that problem (3.1),(3.2) has the following variational formulation

$$\begin{aligned} &u \in V, \\ &a(u,v) = L(v), \text{ for every } v \in V, \end{aligned} \tag{3.3}$$

with

(3.4) $V = H^1(0,1)$,

$$(3.5)\quad a(v,w) = \int_0^1 \alpha(x)v'w'dx + \int_0^1 \beta(x)vwdx + \int_0^1 \gamma(x)v'wdx, \text{ for every } v, w \in V,$$

$$(3.6)\quad L(v) = \int_0^1 fvdx + dv(1) + cv(0), \text{ for every } v \in V;$$

we assume here that $f \in L^1(0,1)$, but indeed L(.) can be any linear continuous functional over $H^1(0,1)$. Sufficient conditions to apply the Lax-Milgram Theorem to the variational problem (3.3) consist of:

(3.7) $0 < \alpha_0 \le \alpha(x) \le \alpha_M$ a.e. on $(0,1)$,

(3.8) $0 < \beta_0 \le \beta(x) \le \beta_M$ a.e. on $(0,1)$,

(3.9) $\gamma \in L^2(0,1)$, $\| \gamma \|_{L^2(0,1)} < \min(\alpha_0, b_0)$.

3.1.2 Wavelet-Galerkin Approximation of the Neumann Problem

Let N = 3 and let ϕ be the corresponding scaling function as defined in Section 1. Let n be any integer and let V_n be defined as in Section 1. Define $V_n(0,1)$ to be the restriction to of all functions in

V_n. By Theorem 1.1, every function in $H^1(0,1)$ can be approximated arbitrarily closely by an element in $V_n(0,1)$ for sufficiently large n, hence

$$(3.10)\quad \text{Closure}\Big(\bigcup_n V_n(0,1)\Big) = H^1(0,1).$$

Therefore, the family of subspaces $V_n(0,1)$ of $H^1(0,1)$ is well suited to the Galerkin solution of the Neumann problem (3.1),(3.2),(3.3). The Galerkin formulation of the above Neumann problem for every integer n is given by

$$(3.11)\quad \begin{aligned} &u_n \in V_n(0,1), \\ &a(u_n,v) = L(v), \text{ for all } v \in V_n(0,1). \end{aligned}$$

Since (0,1) is bounded, the space $V_n(0,1)$ is finite dimensional and is a closed subspace of $H^1(0,1)$. This implies problem (3.11) has a unique solution. It follows from (2.21) and (3.10) that

$$(3.12)\quad \lim_{n \to} (u_n - u) \to 0 \text{ in } H^1(0,1).$$

3.1.3 Solution of the Approximate Problem

Fix any integer n, the solution u_n of the approximate problem (3.11) can be represented as

$$(3.13)\quad u_n = \sum_{k=1}^{p} u_{n,k}\phi_{n,k-2N+1}, \qquad \text{where } p = 2^n + 2N - 2 \text{ and } u_1,\ldots,u_p \in R,$$

where the functions are considered to be restricted to (0,1). This yields the following system of linear equations in p unknowns

$$(3.14)\quad \sum_{k=1}^{p} a(\phi_{n,k-2N+1},\phi_{n,j-2N+1})u_{n,k} = L(\phi_{n,j-2N+1}), \qquad \text{for every } j = 1,\ldots,p.$$

This can be written in matrix notation as

$$(3.15)\quad AU = F$$

with

(3.16) $A = (a_{ij}), \quad a_{ij} = a(\phi_{n,j-2N+1}, \phi_{n,i-2N+1})$

and

(3.17) $F = (f_i), \quad f_i = L(\phi_{n,i-2N+1})$

and

(3.18) $U = (U_{n,i})$.

In the case where the bilinear form a(.,.) defined by (3.5) is $I_*^1(0,1)$ - elliptic, the above matrix A is positive definite implying that problem (3.15) has a unique solution. We shall also observe that if the function g vanishes throughout (0,1) then the bilinear form a(.,.) is symmetric implying that problem (3.15) can be solved by standard conjugate gradient algorithms or by Cholesky factorization.

Remark 3.1 The evaluation of the a_{ij} and f_i in (3.16),(3.17) required to compute the solution of the approximate problem (3.11) can be performed using standard numerical quadrature methods. In the special case that the α, β, and γ are piecewise polynomial then a_{ij} and f_i can be exactly calculated as rational functions of the scaling parameters h(k), k = 0,...,2N-1 using the techniques described in Section 1.3.

Remark 3.2 Figure 3.1 illustrates the first derivative ϕ' of ϕ corresponding to N = 3 evaluated at 321 uniformly distributed points over the interval [0,5] using the standard first difference approximation $\phi'(x) \approx [\phi(x+h) - \phi(x)]/h$. Since $\phi' \in C^{.087833}$ it is 'barely' continuous. Therefore, considerable care must be taken to assure accurate evaluation of the a_{ij} and f_i.

Remark 3.3 It follows from the Fredholm Alternative that problem (3.1),(3.2) is well posed if u = 0 is the only solution of problem (3.1),(3.2) when f = 0 throughout (0,1) and c = d = 0.

3.1.4 Numerical Experiments

We computed wavelet solutions for three instances of problem (3.1),(3.2):

Test Problem 3.1
For this problem

(3.19) $c = d = 0,$

and the functions α, β, and γ are described by

$$\alpha(x) = 1 + x \qquad \text{for } 0 < x < .5,$$
(3.20)
$$\alpha(x) = 10 - x \qquad \text{for } .5 < x < 1,$$

$$\beta(x) = 20 - 10x \qquad \text{for } 0 < x < .5,$$
(3.21)
$$\beta(x) = 1 + x \qquad \text{for } .5 < x < 1,$$

(3.22) $\gamma(x) = 0 \qquad \text{for } 0 < x < 1.$

The right hand side f was chosen such that the solution u of problem (3.1),(3.2) is

(3.23) $u(x) = x^3/3 - x^2/2 + 1;$

indeed $f \in H^{-1}(0,1)$ and over each interval (0,.5) and (.5,1) it can be represented by a polynomial, however, at x = .5, it has both a 'jump' discontinuity and a Dirac measure component.

To solve this problem we chose N = 3 and n = 3 in (3.13) to obtain an approximate problem (3.11),(3.15) involving p = 12 unknowns. This linear problem was solved using a Cholesky factorization technique (permitted here since $\gamma = 0$ over (0,1) implies a(.,.) is symmetric). Figures 3.2(a) and 3.2(b) illustrate the coefficients $\alpha(x)$ and $\beta(x)$ over (0,1); Figure 3.3(a) illustrates the comparison between the exact solution (dotted graph) and the computed solution (solid graph). Figure 3.3(b) illustrates the variation of the error $e_n = u - u_n$ over (0,1). We observe from Figures 3.3(a) and (b) that e_n is small over (0,1) and does not exhibit any special behavior at x = .5 (where α, β and f are discontinuous and/or singular).

Test Problem 3.2
For this problem

(3.24) $c = 1$ and $d = 0$,

and the functions α, β, and γ are described by

$$\alpha(x) = 1 + x \quad \text{for } 0 < x < .5, \qquad \alpha(x) = 2 + x \quad \text{for } .5 < x < 1, \tag{3.25}$$

$$\beta(x) = 1 + x^2 \quad \text{for } 0 < x < .5, \qquad \beta(x) = 2 + x \quad \text{for } .5 < x < 1, \tag{3.26}$$

$$\gamma(x) = 1 + x \quad \text{for } 0 < x < .5 \qquad \gamma(x) = 2 - x \quad \text{for } 0 < x < 1. \tag{3.27}$$

The right hand side f was chosen such that the solution u of problem (3.1),(3.2) is

(3.28) $u(x) = x - x^2/2$;

again $f \in H^{-1}(0,1)$ and exhibits the same qualitative behavior at $x = .5$ as in the previous test problem.

To solve this problem we chose $N = 3$ and $n = 4$ in (3.13) to obtain an approximate problem (3.11),(3.15) involving $p = 20$ unknowns. Since $\gamma \neq 0$ over (0,1) the matrix A in (3.15) is non symmetric. Therefore, this linear problem was solved using an LU factorization technique. Figures 3.4(a),(b),(c) illustrate the coefficients $\alpha(x)$, $\beta(x)$, and $\gamma(x)$ over (0,1); Figure 3.5(a) illustrates the comparison between the exact solution (dotted graph) and the computed solution (solid graph). Figure 3.5(b) illustrates the variation of the error $e_n = u - u_n$ over (0,1). We observe from Figures 3.5(a) and (b) that, as in the previous example, e_n is small over (0,1) and does not exhibit any special behavior at $x = .5$.

Test Problem 3.3
This test problem concerns the solution of the following one dimensional Neumann problem

(3.29) $-u'' + u = (1 + \pi^2)\sin\pi x + 1$ on (0,1),

(3.30) $-u'(0) = u'(1) = -\pi$,

whose exact solution is given by

(3.31) $u(x) = \sin\pi x + 1$.

We have solved problem (3.29),(3.30) using the wavelet based method described in Sections 3.1,3.2, taking N = 3 and m = 3,4,5,6,7, the corresponding values of p being then 12,20,36,68,132. We took advantage of these various calculations to study the influence of m (and p) on the approximation error; therefore in Figures 3.6(a) to 3.6(e) we have plotted the variation of $e_n = u - u_n$ on (0,1). In Figure 3.7 we have shown on a log-log scale the variation of $\|e_n\|$ as a function of p (=p(n)) showing that the above error varies approximately like $p^{-2.5}$. This behaivor suggest that the above appproximation is between piecewise linear and piecewise quadratic approximation.

3.2 Solution of the Dirichlet Problem

3.2.1 Formulation of the Dirichlet Problem

The Dirichlet problem to be discussed in this section can be formulated as follows:

(3.32) $-(\alpha u')' + \beta u + \gamma u' = f$,

(3.33) $u(0) = c$, $u(1) = d$;

we suppose here that $f \in H^{-1}(0,1)$.

Consider $v \in H_0^1(0,1)$ ($= \{v : v \in H^1(0,1), v(0) = v(1) = 0\}$); then multiplying (in the sense of the duality pairing) both sides of equation (3.32) by v we obtain

$$(3.34)\quad \int_0^1 (\alpha u'v' + \beta uv + \gamma u'v)dx = \langle f,v\rangle, \quad \text{for every } v \in H_0^1(0,1).$$

Suppose that problem (3.32),(3.33) has a solution u in $H^1(0,1)$; then u necessarily satisfies the following variational condition

$$(3.35)\quad \begin{aligned} &u \in H^1(0,1),\ u(0) = c,\ u(1) = d,\\ &a(u,v) = \langle f,v\rangle \text{ for every } v \in H_0^1(0,1). \end{aligned}$$

In (3.35) the bilinear form a(.,.) is defined by (3.5). Actually, (3.35) implies (3.32),(3.33). Suppose that conditions (3.7)-(3.9) on the functions a, b, g are satisfied, then we can easily prove that the bilinear form a(.,.) is continuous and $H_0^1(0,1)$ - elliptic over $H_0^1(0,1)xH_0^1(0,1)$. This implies that any solution of (3.32),(3.33),(3.35) is necessarily unique. To prove the existence of a solution we shall use again the Lax-Milgram Theorem. To overcome the (small) difficulty associated to the fact that unless $c = d = 0$, $u \notin H_0^1(0,1)$, we introduce the functions u_0 and w defined by

$$u_0(x) = c + (d-c)x, \tag{3.36}$$

and

$$w = u - u_0. \tag{3.37}$$

The function w is clearly a solution of the following variational problem

$$\begin{aligned} &w \in H_0^1(0,1), \\ &a(w,v) = \langle f,v\rangle - a(u_0,v) \quad \text{for every } v \in H_0^1(0,1). \end{aligned} \tag{3.38}$$

From the properties of a(.,.) the right hand side of equation (3.38) depends linearly and continuously on $v \in H_0^1(0,1)$. We can therefore apply the Lax-Milgram Theorem to problem (3.38) to obtain a unique solution w. Combining (3.37),(3.38) we have thus proved the existence of a solution u to problem (3.32),(3.33),(3.35) (uniqueness was already established).

Remark 3.4 Using the fact that the seminorm defined by $\|v'\|_{L^2(0,1)}$ is a norm over $H_0^1(0,1)$, equivalent to the standard $H^1(0,1)$ - norm, problem (3.32),(3.33),(3.35) is still well posed if we assume for example that $\beta = 0$ over (0,1) and γ = constant over (0,1), relation (3.7) being still satisfied.

3.2.2 Reduction of the Dirichlet Problem to the Neumann Problem

There are various ways of solving Dirichlet problems using variational methods. Two fairly classical methods are the following:

(i) Approximate $H^1(\Omega)$ by a finite dimensional subspace V_h, then decompose V_h as follows:

(3.39) $V_h = V_{0h} \oplus M_h$

where V_{0h} approximates $H_0^1(\Omega)$, then construct a function $u_{\gamma h} \in M_h$ that satisfies (approximately) the Dirichlet boundary condition. Finally, introducing $w_h = u_h - u_{\gamma h}$ reduce the approximate Dirichlet problem to a variational problem in V_{0h}. This follows the approach discussed in Section 3.2.1 to solve via (3.36)-(3.38) the Dirichlet problem (3.32),(3.33),(3.35). The construction of $u_{\gamma h}$ in dimension ≥ 2 may be a complicated problem in itself for some types of approximations.

(ii) Reduce the solution of the Dirichlet problem to the solution of Neumann problems using boundary multipliers and/or penalization of the Dirichlet condition. This approach is used in some sense in production finite element codes for solving elliptic problems.

In this paper we will focus on the second approach since it provides a possible methodology for solving multidimensional elliptic problems. However we will also briefly comment on the first approach which seems to be more technically involved and will be discussed in more detail in a forthcoming paper.

In order to describe approach (ii) we consider the following abstract Dirichlet problem in $H^1(0,1)$:

(3.40)
$$u \in H^1(0,1),\ u(0) = c,\ u(1) = d,$$
$$a(u,v) = \langle f,v \rangle \text{ for every } v \in H_0^1(0,1),$$

where $f \in H^{-1}(0,1)$ and where the bilinear form $a(.,.)$ is continuous and elliptic over $H^1(0,1)$ (the case where $a(.,.)$ is elliptic over $H_0^1(0,1)$ but not over $H^1(0,1)$ will be addressed later), then by the Lax-Milgram Theorem, problem (3.40) is well posed. Denote by $L: H^1(0,1) \rightarrow R$ a linear continuous functional satisfying

(3.41) $L(v) = \langle f,v \rangle$ for every $v \in H_0^1(0,1)$;

such a functional always exists by the Riesz Representation Theorem. To the Dirichlet problem (3.40) we associate the following problem:

(3.42) $\{u, \lambda\} \in H^1(0,1) \times R^2; \quad \lambda = \{\lambda_1, \lambda_2\}$,

(3.43) $a(u,v) = L(v) + \lambda_1 v(0) + \lambda_2 v(1)$, for every $v \in H^1(0,1)$,

(3.44) $u(0) = c,\ u(1) = d.$

Suppose that problem (3.42)-(3.44) has a solution $\{u,\lambda\}$; taking $v \in H_0^1(0,1)$ in (3.43) it follows

from (3.41) that u is also the solution of (3.40). We shall now show problem (3.42),(3.44) can be reduced to the solution of 3 Neumann problems and to that of a 2x2 well posed linear system. Define u_0,u_1,u_2 as the solutions of the following variational problems in $H^1(0,1)$:

$$\begin{aligned} &u_0 \in H^1(0,1), \\ &a(u_0,v) = L(v), \text{ for all } v \in H^1(0,1), \end{aligned} \tag{3.45}$$

$$\begin{aligned} &u_1 \in H^1(0,1), \\ &a(u_1,v) = v(0), \text{ for all } v \in H^1(0,1), \end{aligned} \tag{3.46}$$

$$\begin{aligned} &u_2 \in H^1(0,1) \\ &a(u_2,v) = v(1), \text{ for all } v \in H^1(0,1). \end{aligned} \tag{3.47}$$

The function u in (3.42)-(3.44) necessarily satisfies

$$u = u_0 + \lambda_1 u_1 + \lambda_2 u_2 \tag{3.48}$$

implying that λ satisfies

$$\begin{aligned} &u_1(0)\lambda_1 + u_2(0)\lambda_2 = c - u_0(0), \\ &u_1(1)\lambda_1 + u_2(1)\lambda_2 = d - u_0(1). \end{aligned} \tag{3.49}$$

If (3.49) has a solution $\{\lambda_1,\lambda_2\} = \lambda$ then the pair $\{u_0 + \lambda_1 u_1 + \lambda_2 u_2, \lambda\}$ is a solution of (3.41)-(3.44). To show that (3.49) has a solution we shall verify that the matrix

$$\Lambda = \begin{vmatrix} u_1(0) & u_2(0) \\ u_1(1) & u_2(1) \end{vmatrix} \tag{3.50}$$

is positive definite; to show this property take $\mu = \{\mu_1,\mu_2\} \in R^2$ and associate to μ the function u_μ defined by

(3.51) $u_\mu = \mu_1 u_1 + \mu_2 u_2$.

The function u_μ is clearly the unique solution of

$$\begin{aligned} &u_\mu \in H^1(0,1), \\ &a(u_\mu, v) = \mu_1 v(0) + \mu_2 v(1), \text{ for all } v \in H^1(0,1). \end{aligned} \tag{3.52}$$

We also have, from (3.50)-(3.52) and from the ellipticity of a(.,.)

(3.50) $\Lambda\mu.\mu = \mu_1 u_\mu(0) + \mu_2 u_\mu(1) = a(u_\mu, u_\mu) \geq 0$ for all $\mu \in R^2$.

Suppose now that $\Lambda\mu.\mu = 0$, then (3.50) and the H^1 - ellipticity of a(.,.) implies that $u_\mu = 0$ over (0,1), this fact and (3.52) implies $\mu_1 = \mu_2 = 0$. Matrix Λ being positive definite is regular and therefore $\lambda = \{\lambda_1, \lambda_2\}$ is uniquely determined from (3.49). In fact we have proved that problem (3.40) and (3.41)-(3.44) are equivalent.

Remark 3.5 If the bilinear form a(.,.) is symmetric then problem (3.40) is equivalent to the minimization problem

$$\begin{aligned} &u \in V_{c,d}, \\ &J(u) \leq J(v), \text{ for every } v \in V_{c,d}. \end{aligned} \tag{3.54}$$

where $V_{c,d} = \{v: v \in H^1(0,1), v(0) = c, v(1) = d\}$ and $J(v) = (1/2)a(v,v) - L(v)$.
Furthermore, the pair $\{u,\lambda\}$, which is a solution of (3.41)-(3.44), is a stationary point over $H^1(0,1)xR^2$ of the following Lagrangian functional

$$L(v,\mu) = J(v) + \mu_1(v(0)-c) + \mu_2(v(1)-d).$$

The vector λ is therefore the Lagrange multiplier associated to the two linear constraints $v(0) - c = 0$, $v(1) - d = 0$. In the nonsymmetric case we shall still call λ a multiplier. The (important) case where the bilinear form a(.,.) is $H^1(0,1)$ - elliptic can be treated by a similar approach. We shall suppose for simplicity that

(3.55) $a(v,v) \geq \gamma \|v'\|^2_{L^2(0,1)}$ for all $v \in H^1(0,1)$ with $\gamma > 0$.

In that case, instead of (3.41)-(3.44), we associate to (3.40) the following problem (with $r > 0$):

(3.56) $\{u,\lambda\} \in H^1(0,1)xR^2$; $\lambda = \{\lambda_1,\lambda_2\}$,

(3.57) $a(u,v) + r[u(0)v(0) + u(1)v(1)] = L(v) + \lambda_1 v(0) + \lambda_2 v(1)$, for all $v \in H^1(0,1)$,

(3.58) $u(0) = c$, $u(1) = d$.

Define $a_r(.,.)$ by

(3.59) $a_r(v,w) = a(v,w) + r[v(0)w(0) + v(1)w(1)]$;

it follows then by (3.55) that

(3.60) $a_r(v,v) \geq \gamma(\|v'\|_{L^2(0,1)})^2 + r(|v(0)|^2 + |v(1)|^2)$;

since it can be (easily) shown that $v \rightarrow [(\|v'\|_{L^2(0,1)})^2 + |v(0)|^2 + |v(1)|^2]^{1/2}$ defines over $H^1(0,1)$ a norm equivalent to the usual $H^1(0,1)$ - norm. It follows from (3.60) that if $r > 0$ the bilinear form $a_r(.,.)$ is $H^1(0,1)$ - elliptic. From this property we can easily prove that problems (3.40) and (3.56)-(3.58) are equivalent; also, the pair $\{u,\lambda\}$ can be obtained through the solution of 3 'Neumann' problems associated to the bilinear form $a_r(.,.)$ *followed by the solution of a 2x2 linear system* associated to a positive definite matrix. The wavelet implementation of this technique is discussed in Section 3.2.3.

3.2.3 Numerical Experiments

We computed wavelet solutions of three instances of problem (3.32),(3.33).

Test Problem 3.4 For this problem $u(0) = 0$ and $u(1) = 5/6$; on the other hand α and β are given by (3.20) and (3.21), respectively, and $\gamma = 0$. The right hand side f has been chosen such that the solution u of problem (3.32),(3.33) is

(3.61) $u(x) = x^3/3 - x^2/2 + 1$.

To solve this problem we chose $N = 3$ and $n = 3$ in (3.13) to obtain an approximate problem involving 12 unknowns. The maximum norm error between the exact and computed solutions is 1.2×10^{-3}, while the boundary conditions are exactly satisfied as we can see in Figure 3.8 where both exact and

computed solutions are represented. As one can see our procedure for treating the Dirichlet condition via the solution of Neumann problems (see Section 3.2.2) is fairly accurate, particularly in the neighborhood of the boundary points 0 and 1.

Test Problem 3.5 For this problem we have u(0) = 0, u(1) = .5 and a, b, and g defined by (3.25),(3.26) and (3.27), respectively. The right hand side f has been chosen such that the solution u of problem (3.32),(3.33) is

(3.62) $u(x) = x - x^2/2$.

To solve this problem we chose N = 3 and n = 4 to obtain an approximate problem involving 20 unknowns. The maximum norm error between the exact and computed solutions is 6.0×10^{-4}, while the boundary conditions are exactly satisfied as we can see in Figure 3.9, where we compare the exact and computed solutions, and in Figure 3.10, where we have shown the variatiojn of the error on [0,1].

Test Problem 3.6 This test problem concerns the solution of the following one dimensional Dirichlet problem

(3.63) $-u'' + u = (1 + \pi^2)\sin\pi x + 1$ on (0,1),
(3.64) $u(0) = u(1) = 1$,

whose exact solution is given by

(3.65) $u(x) = \sin\pi x + 1$.

We have solved problem (3.63),(3.64) taking N = 3 and m = 3,5,6, the corresponding values of p being then 12,36,68. We took advantage of these various calculations to study the influence of m (and p) on the approximation error. Figures 3.11(a) to 3.11(c) shows the variation of $e_n = u - u_n$ on (0,1) for m = 3,5,6 respectively.

4 Solution of Singularly Perturbed Linear Elliptic Problems

We consider in this section the wavelet solution of a particular one dimensional Dirichlet problem, namely

(4.1) $-\varepsilon u'' - u' = 1$ on (0,1)

(4.2) $u(0) = u(1) = 0$,

with $\varepsilon > 0$. We shall focus our attention to the cases where ε is 'small'. The exact solution of problem (4.1),(4.2) is given by

(4.3) $u_\varepsilon(x) = (1-x) - [\exp((1-x)/\varepsilon)-1] / [\exp(1/\varepsilon)-1];$

for small values of ε it exhibits a boundary layer of thickness of order ε in the neighborhood of $x = 0$. The solution $u_\varepsilon(x)$ has been shown in Figure 4.1 for $\varepsilon = 1/710$.

Our motivation with this example is to study the ability of wavelet approximations to represent stiff gradient phonomena such as those occuring in fluid mechanics and semiconductors for example.

In order to solve problem (4.1),(4.2) using the wavelet techniques discussed in Section 3 we split the interval (0,1) into $(0,x_c)$ and $(x_c,1)$ with $0 < x_c < 1$. The idea here is to take x_c small and to solve problem (4.1),(4.2) using a domain decomposition method; the subproblem associated to $(0,x_c)$ will treat the boundary layer behavior of the solution near $x = 0$ while the subproblem associated to $(x_c,1)$ will describe the smoother component of $u_\varepsilon(x)$. The local solutions are matched at x_c using a multiplier method which will force the continuity of local solutions and of their first order derivatives at $x = x_c$ (see [Bourgat, Glowinski, LeTallec, Vidrascu, 1989] for further details concerning domain decomposition techniques in multidimensions containing the one used here as a special case).

Problem (4.1),(4.2) has been solved for $\varepsilon = 1/710$ with $x_c = 1/32$ using first a combination of 20 basis functions in $(0,x_c)$ and 35 in $(x_c,1)$ and then using a combination of 36 and 35 basis functions in $(0,x_c)$ and $(x_c,1)$ respectively. The corresponding results are shown in Figures (4.2)-(4.4). Observe that the wavelet solutions accurately approximate the behavior in the boundary layer despite the fact that in our calculation x_c was indeed of the order of $\sqrt{\varepsilon}$ instead of ε, reflecting the super-linear approximation properties of the Daubechies wavelets.

5 Multigrid Solutions of Linear Elliptic Problems with Periodic Boundary Conditions

In previous sections we discussed wavelet solutions of problems obtained by directly solving the system of algebraic equations resulting from discretization. Frequently in practice, these algebraic equations are so large as to require the use of iterative methods that apply a sequence of relaxation operations to improve the current estimate v. Typical relaxation operations, including Gauss Seidel, Jacobi, Successive Over Relaxation, and Preconditioned Conjugate Gradients, have a tendency to dampen high frequency components of the error at a faster rate than the low frequency components. This phenomena may result in slow convergence. Multigrid methods, described in [Brandt,1977], [Briggs,1987], [Fedorenko,1961,1987], and [McCormick,1987], address this problem by utilizing relaxation methods at multiple scales of resolution to balance the dampening across all frequency components of the error. Multigrid methods can utilize diverse relaxation techniques and various spatial discretization methods, including finite differences and finite elements.

The multiscale properties of general wavelets, as described in [Mallat,1987] and [Meyer,1985,1986,1988], together with the superior approximation and computational properties of the specific class of Daubechies wavelet bases described in Section 1, suggest that the Daubechies wavelet bases may provide an effective tool for developing improved multilevel methods. This section describes preliminary theoretical and numerical results for wavelet based multilevel methods applied to a simple class of model problems.

5.1 Multilevel Solutions

The linear elliptic equation to be discussed is given by

(5.1) $A u = f,$

where A is a linear strongly elliptic differential operator of order 2 and where A, u and f are periodic with period 1. Let $N \geq 3$, therefore $\phi \in H^1(R)$ and for all $n \geq 0$ let $V = H^1(R_p)$ and $V \supset V_n = V_n(R_p)$ be defined as in Section 1.3.3. Since the operator A is strongly elliptic, it is an isomorphism from V onto $V^* = H^{-1}(R_p)$ and therefore there exists a unique solution to problem (5.1) if $f \in V^*$.

Let P_n denote the projection of V onto V_n. Since $V^* \supset V \supset V_n$, P_n^* projects V^* into V_n^*. For every $n \geq 0$ the approximation of problem (5.1) corresponding to the subspace V_n is

(5.2) $A_n u_n = P_n^*(f).$

where $A_n = P_n^* A P_n$ and $u_n = P_n(u)$. Clearly, by the Lax-Milgram theorem problem (5.2) has a unique solution u_n which, by property (2.18), satisfies $\| u_n - u \| \leq \|A\| \|A^{-1}\| \| v - u \|$ for every $v \in V_n$, where all the norms are with respect to $V = H^1(R_p)$. By inequality (1.31) it follows that $\| u_n - u \|$ is on the order of h^{N-1} where $h = 2^{-n}$ corresponds to the "step size" of the approximation. Thus the H^1 norm of the error has order N-1. If the coefficients of the operator A are "smooth" and if $f \in L^2$ then it may be shown that $u \in H^2$ and furthermore that the L^2 norm of the error $u_n - u$ may be of order as high as N. This conclusion does not hold in general.

Throughout the remainder of this section we assume that A has constant coefficients and that $f \in L^2$, therefore by the preceding remarks the L^2 norm of the truncation error has order N. We will proceed to describe the full multigrid V-cycle method (here we use the terminology in [Briggs,1987]). First, we choose integers $n_f > n_0 \geq 1$ which correspond to the finest and the coarsest levels of approximations to problem (5.1). The solution of the approximate problem (5.2) for $n = n_0$ will always be solved using a direct method. The first step of the algorithm consist of computing the solution v_0 of problem (5.2) for $n = n_0$. Next, we increment $n \leftarrow n+1$ and choose v_{n-1} ($\in V_n$) as the initial guess for the solution of problem (5.2) at level n (this requires calculating the expansion of v_n in a new basis

of V_n, using the Mallat transform T defined by equation (1.34)). Next, apply a V-cycle relaxation operation to improve the estimated solution v_n of problem (5.2). In this paper we discuss the V-cycle procedures based on Jacobi relaxation techniques. The Jacobi relaxation operators are defined by the following affine mappings J_i: V_n --> V_n, for i = 1, 2

(5.3) Linear Jacobi Relaxation v_n <-- $J_1(v_n) = v_n + c\, r_n$,

(5.4) Quadratic Jacobi Relaxation v_n <-- $J_2(v_n) = v_n + c\, r_n + dA_n r_n$.

where in (5.3),(5.4) the residual r_n is defined by

(5.5) $r_n = P_n^*(f) - A_n v_n$;

here V_n^* is identified with V_n so that $r_n \in V_n$. In (5.3),(5.4) the constants c and d are optimally chosen as follows:

First define (in the sense of [Briggs,1987]) the set of the oscillatory eigenvalues of the operator A_n (where as above V_n^* is identified with V_n so that A_n maps V_n to itself) by

(5.6) $\Lambda = \{a\hat{}(n,\omega) : \pi/2 \le \omega \le 3\pi/2\}$;

as in equation (1.54), and then define λ_1 and λ_2 by

(5.7) $\lambda_1 = \min\{\lambda : \lambda \in \Lambda\}$, and $\lambda_2 = \max\{\lambda : \lambda \in \Lambda\}$.

The constants c and d and their corresponding dampening ratios D (over the set Λ) for the above Jacobi relaxation procedures are given by

(5.8) Linear Jacobi : $c = 2/(\lambda_1+\lambda_2)$, $D = (\lambda_2-\lambda_1)/(\lambda_1+\lambda_2)$,

(5.9) Quadratic Jacobi : $d = -2/[\lambda_1\lambda_2+(\lambda_1+\lambda_2)^2/4]$, $c = -d(\lambda_1+\lambda_2)$, $D = (\lambda_2-\lambda_1)^2/[(\lambda_1+\lambda_2)^2+4\lambda_1\lambda_2]$.

If $A = -d^2/dx^2$ then D = 1/3 for the linear Jacobi relaxation procedure (5.3) applied to the standard finite difference approximation of problem (5.1); on the other hand we have D = 2/3 if (5.3) is applied to the (N=3) wavelet approximation of problem (5.1). Concerning the quadratic Jacobi relaxation procedure (5.4), it yields D =.315 for the (N=3) wavelet approximation of problem (5.1); indeed we obtain an even smaller dampening ratio D = 1/17 for the finite difference approximation of problem

(5.1), but the increased complexity of the corresponding algorithm is such that only the linear procedure (5.3) is used in practice for finite difference approximations (on the other hand procedure (5.4) is computationally advantages for (N=3) wavelet approximations of (5.1)).

The first step of the V-cycle Jacobi relaxation consists of applying either (5.3) or (5.4) to the initial guess v_n so as to dampen the oscillatory components of the error $e_n = u_n - v_n$ to within truncation error. Under the assumptions on A and f above, the truncation error for wavelet approximations of problem (5.1) satisfies $\| e_{n+1} \| \approx 2^{-N} \| e_n \|$ (to be compared to $\| e_{n+1} \| \approx 2^{-2} \| e_n \|$ for finite difference approximations), therefore it is necessary to apply approximately $-N / \log_2(D)$ Jacobi iterations to dampen all of the oscillatory eigenfunction components of the error by a factor of 2^{-N}. The second step of the V-cycle Jacobi relaxation procedure consists dampening the non-oscillatory components of the error as follows:

(i) Calculate the projection $P_{n-1}(r_n)$ of the current residual onto V_{n-1} (this requires calculating the inverse Mallat transform T^{-1} defined by equation (1.35)).

(ii) Solve the residual equation

(5.10) $A_{n-1} e_{n-1} = P_{n-1}(r_n)$.

(iii) Update the estimated solution v_n by

(5.11) $v_n \leftarrow v_n + e_{n-1}$:

observe that this requires calculating the expansion of e_{n-1} in the basis for V_n using the Mallat transform T defined by equation (1.34). In practice step (ii) will consist of ν_n iterations of the linear or quadratic Jacobi relaxation procedure; also in step (iii) the update (5.11) will be followed by μ_n iterations of our chosen relaxation procedure. Furthermore, in step (ii) the procedure described by step (i) is applied recursively to the estimate obtained after Jacobi relaxation in order to dampen the remaining non-oscilatory error components.This results in a complete V-cycle in the traditional sense (cf., e.g., [Briggs,1987]) that starts at level $n > n_0$, then descends to level n_0, and finally proceeds back to level n+1. This V-cycle is then repeated until the finest level n_f is reached.

Comparing the total number of arithmetic operations required to achieve an L^2 error equal to ε using various full multigrid V-cycles for solving problem

(5.12) $-u'' + u = f$ on (0,1),

(5.13) $u(0) = u(1), u'(0) = u'(1)$,

we obtain (with $P_1 \approx \varepsilon^{-1/2}$ and $P_2 \approx \varepsilon^{-1/N}$)

Finite Difference Linear Jacobi	#operations $\approx$ 45 P_1
Wavelet Linear Jacobi	#operations $\approx$ 12N(4N–2) P_2
Wavelet Quadratic Jacobi	#operations $\approx$ .7x12N(4N–2) P_2

which shows that for wavelet approximations the quadratic Jacobi relaxation procedure is more advantageous than the linear one, both being superior - for "small" ε - to the linear Jacobi relaxation procedure applied to the finite difference approximation of problem (5.1) (at least if $N \geq 3$, which is always the case in practice).

5.2 Numerical Experiments

In this section the multilevel methods discussed in Section 5.1 have been applied to the practical solution of problem (5.12), (5.13) for f given by

(5.14) $f(x) = \sin 2\pi x + \sin 10\pi x.$

For the right hand side (5.14) the exact solution of problem (5.12), (5.13) is given by

(5.15) $u(x) = \sin 2\pi x / (1 + 4\pi^2) + \sin 10\pi x / (1 + 100\pi^2).$

The variations of f and u are shown in Figure 5.1.

First, with h = 1/I we have used the following finite difference scheme to approximate problem (5.12)-(5.14) by

(5.16) $-(u_{i+1} + u_{i-1} - 2u_i) / h^2 + u_i = f(x_i), \quad 1 \leq i \leq I;$

we force the periodicity of the solution by requiring in (5.16)

(5.17) $u_0 = u_I$ if i = 1, and $u_{I+1} = u_1$ if i = I.

The approximate problem (5.16),(5.17) has been solved using for h = 1/256 using a six level realization of the general multilevel method described above. For this test problem we employed the linear Jacobi relaxation procedure (5.3) with $\nu_n = \mu_n = 2$. Figure 5.2 shows the computed solution u_h

and the variation of the error $u - u_h$ over [0,1]; the max norm of the error is 3.3×10^{-6}.

Next we have considered the numerical solution of problem (5.12)-(5.14) by a combination of (N=3) wavelet approximation and the multilevel methods of Section 5.1. We have compared the performance of the linear and quadratic Jacobi relaxation procedures using in both cases $n_f = 6$ and $n_0 = 3$ resulting in 4 levels; this corresponds to dim $V_n = 2^n$, $3 \le n \le 6$. In both cases we have used $\nu_n = \mu_n = 4$. Figures 5.3 (linear Jacobi) and 5.4 (quadratic Jacobi) illustrate the computed solution u_6 and the corresponding error $u - u_6$. The respective maximum norms of the errors are 9×10^{-6} and 3.6×10^{-6}. These results indicates the superiority of the methodology combining wavelet approximation and quadratic Jacobi relaxation procedures.

Experiments concerning the multilevel wavelet solution of multidimensional boundary value problems are in progress and will be reported in a forthcoming article.

6 Parabolic Problems

This section discusses initial value problems of the following form:

(6.1) $\partial u/\partial t + Au = f$ for $t > 0$,

and

(6.2) $u(x,0) = u_0(x)$.

where A is a (possibly nonlinear) elliptic operator in the space variable x. For simplicity we will assume that A, u and f are periodic in x with period 1. In contrast to the ordinary differential equations treated in previous sections, a temporal as well as a spatial discretization is required. As prototype problems of this class we shall focus on the linear heat equation (where $A = -\partial^2/\partial x^2$) and the regularized Burgers equation (where $Au = -\nu\, \partial^2 u/\partial x^2 + u\partial u/\partial x$). Various finite difference time discretization schemes will be combined with wavelet space approximations to solve the above two problems.

6.1 Solution of Heat Equation

6.1.1 Formulation of the Problem

We consider in this section the numerical solution of the following heat equation

(6.3) $\partial u/\partial t - \partial^2 u/\partial x^2 = f$ for $t > 0$ and $x \in (0,1)$,

(6.4) $u(0,t) = u(1,t), \quad \partial u(0,t) / \partial x = \partial u(1,t) / \partial x,$

(6.5) $u(x,0) = u_0(x).$

Denote the functions $x \to f(x,t)$ and $x \to u(x,t)$ by $f(t)$ and $u(t)$, respectively. It is assumed that, for almost every $t > 0$, $f(t) \in V^* = H^{-1}(R_p)$ and $u(t) \in V = H^1(R_p)$ (where V and its dual space V^* are defined as in Section 1.3.3). A variational formulation is obtained by multiplying equation (6.3) by $v \in V$ and integrating by parts with respect to x. This yields

(6.6) $$<\partial u/\partial t, v> + \int_0^1 \partial x \, dv/dx \, dx = < f, v >, \quad \text{for every } v \in V,$$

where, in (6.6), $< . , . >$ denotes the duality pairing between V^* and V which reduces to the L^2 scalar product when both arguments are in L^2.

6.1.2 Wavelet Approximation and Time Discretization of Problem (6.3)-(6.5)

Let $N \geq 3$, let $n \geq 1$ and let $V_n = V_n(R_p)$ be the suspace of $V = H^1(R_p)$ as defined in Section 1.3.3. Using relation (6.6) as a guideline, we approximate problem (6.3)-(6.5) by the following problem: Find a function $u_n(t)$ satisfying for almost every $t > 0$

(6.7) $$\int_0^1 \partial u_n(t)/\partial t \, v \, dx + \int_0^1 \partial u_n(t)/\partial x \, dv/dx \, dx = \int_0^1 f_n(t) \, v \, dx, \text{for every } v \in V_n,$$

(6.8) $u_n(0) = u_{0n},$

where u_{0n} is the L^2 projection $P_n(u_0)$ of the initial data u_0 on V_n and where $f_n(t) = P_n^*(f(t))$ is the projection of $f(t)$ on V_n^* (identified with V_n ; indeed, $f_n(t)$ is the unique element of V_n satisfying

$$\int_0^1 f_n(t) \, v \, dx = <f(t), v>, \quad \text{for all } v \in V_n).$$

Problem (6.7)-(6.8) is equivalent to a system of first order ordinary differential equations obtained by substituting v in (6.7) with the elements of a basis of V_n. This system is equivalent to the following initial value problem in V_n

(6.9) $\partial u_n/\partial t + A_n u_n = f_n \text{ for } t > 0,$

(6.10) $u_n(0) = u_{0n}.$

where $A_n = P_n^* A P_n$ is the V_n approximation of $A = -\partial^2/\partial x^2$. Problem (6.9),(6.10) has the following closed form solution

$$(6.11) \quad u_n(t) = \exp(-tA_n)\, u_{0n} + \int_0^t \exp(-A_n(t-s))\, f_n(s)\, ds.$$

For simplicity assume that f(t) and therefore $f_n(t)$ are differentiable. We use a time discretization of problem (6.9),(6.10) that results from approximating the Taylor series for $u_n(t+\Delta t)$ by the following quadratic polynomial in Δt

$$(6.12) \quad u_n(t+\Delta t) \approx u_n(t) + \Delta t\,[f_n(t) - A_n u_n(t)] + .5\,\Delta t^2[\partial f_n(t)/\partial t - A_n(f_n(t) - A_n u_n(t))]$$

This yields the following explicit time stepping scheme :

$$(6.13) \quad u_n^0 = u_{0n},$$

$$(6.14) \quad u_n^{k+1} = u_n^k + \Delta t\,[f_n^k - A_n u_n^k] + .5\,\Delta t^2[(\partial f_n/\partial t)^k - A_n(f_n^k - A_n u_n^k)],$$

which is of Lax-Wendroff type. Comparison with the exact solution (6.11) shows this scheme is *second order accurate with respect to* Δt. Furthermore, it can be shown that this scheme is stable if and only if

$$(6.15) \quad \Delta t \le 2/\lambda_{n,max}$$

where $\lambda_{n,max}$ denotes the largest eigenvalue of A_n. It can be shown using the same type of analysis done in Section 1.3.3 that for the (N=3) wavelet and for large n , $\lambda_{n,max} \approx (1/14)2^{2n}$. Therefore, stability of the wavelet solution of the heat equation obtained by the explicit time stepping scheme above requires

$$(6.16) \quad \Delta t \le .143\, h^2$$

where $h = 2^{-n}$ corresponds to the space step size. The stability bound for the corresponding finite difference (in space) scheme is $.5h^2$.

Remark 6.1 Scheme (6.13),(6.14) has the inconvenience of using the time derivative of f; also, in its

present form, it is not well suited to the solution of nonlinear problems. Therefore in practice we shall use the following Runge-Kutta form of the Lax-Wendroff scheme:

(6.17) $u_n^0 = u_{0n}$,

then for $k \geq 0$, u_n^k being known we compute u_n^{k+1} via

(6.18) $w_n^{k+1/2} = u_n^k + .5\,\Delta t\,(f_n^k - A_n u_n^k)$,

(6.19) $w_n^{k+1} = u_n^k + \Delta t\,(f_n^k - A_n u_n^k)$,

(6.20) $u_n^{k+1} = w_n^{k+1/2} + .5\,\Delta t\,(f_n^{k+1} - A_n w_n^{k+1})$.

Schemes (6.13),(6.14) and (6.17)-(6.20) coincide if $f = 0$ and their stability and accuracy properties are the same, the last scheme being more practical for obvious reasons.

Remark 6.2 For many applications the stability condition (6.16) imposes a prohibitively large number of time steps. For such cases we should choose an implicit time discetization scheme such as the one used for the Burgers equation in the following Section 6.2. In the case of the heat equation (6.3)-(6.35) this will lead to the solution at each time step of an elliptic problem for which the methods described in Sections 3 and 5 still apply.

For more complete description and analysis of numerical methods for initial value problems for differential operators one may consult, e.g., [Raviart, Thomas, 1983] and [Strikwerda, 1989] (see also the references therein).

6.1.3 Numerical Experiments

The methodology described in Section 6.1.2 has been applied to the solution of the heat equation (6.3)-(6.5) for $f = 0$ and $u_0(x) = 1 / (1 - .5 \sin 2\pi x)$. Figure 6.1 shows the variation, over (0,1), of u_0 and of the solution u at time $t = .01$.

Figure 6.2 compares, for $t = .01$, the exact solution of the heat equation (6.3)-(6.5) with the exact solution $u_h(t)$ of the ordinary differential system obtained from the finite difference space discretization of (6.3)-(6.5), using $h = 1/256$. This error here is due solely to finite difference space discretization.

Figure 6.3 compares, for $t = .01$, the above semidiscrete solution $u_h(t)$ with a fully discretized finite difference solution $v_h(t)$ of heat equation (6.3)-(6.5) using the above Lax-Wendroff scheme, with $\Delta t = 5 \times 10^{-6}$ (approximately the maximum stable step size). This error here is due to time discretization since the space discretizations are identical).

Figure 6.4 compares the variation of the functions u_0, $P_n(u_0)$ (for the N=4 wavelets), and $u_n(.01)$,

where here $u_n(t)$ is the exact solution of the semidiscrete heat equation (6.9),(6.10); this figure has to be compared to Figure 6.1.

Figure 6.5 compares, for t = .01, the exact solution u of the heat equation (6.3)-(6.5) with the exact solution u_n of the semidiscrete heat equation (6.9),(6.10), (obtained from the N=4 wavelet spatial discretization with n = 5, i.e. h = 1/32).

Figure 6.6 compares, for t = .01, u_n above with the fully discretized N=4 wavelet solution $v_n(t)$ of heat equation (6.3)-(6.5) using the above Lax-Wendroff scheme, with $\Delta t = 10^{-4}$ (approximately the maximum stable step size). Due to the fact that N=4 wavelets provide approximation having comparable accuracy to finite differences, using fewer degrees of freedom (e.g. 32 < 256), the maximum stable step size is significantly increased. This drastically decreases the required computation. However, since the time steps are larger, a more accurate (than, e.g., first order forward Euler) time stepping scheme is required to balance the high spatial accuracy provided by the wavelets. Comparison of Figures 6.5 and 6.6 indicates that the Lax-Wendroff scheme provides this balance.

6.2 Solution of the Regularized Burgers Equation

6.2.1 Formulation of the Problem

In this section we will discuss the numerical solution of the Regularized Burgers Equation defined with $\nu > 0$ by

(6.21) $\partial u(x,t)/\partial t - u(x,t)\partial u(x,t)/\partial x = \nu\, \partial^2 u(x,t)/\partial x^2$ for $t > 0$ and $0 < x < 1$,

(6.22) $u(x,0) = u_0(x)$,

completed by appropriate boundary conditions (Neumann, Dirichlet, periodic, ...) at x = 0 and x = 1. Related problems arise in many branches of science and engineering particularly fluid mechanics and petroleum resevoir simulation. Indeed this problem is idealy suited for formulating and evaluating new numerical methods for eventually solving the Navier Stokes Equations.

6.2.2 Time Discretization of Problem (6.21),(6.22)

Let $\Delta t > 0$ be a time discretization step. We can discretize problem (6.21),(6.22) using, for example, the following semi-implicit scheme:

(6.23) $u^0 = u_0$,

then for $k \geq 0$ we compute u^{k+1} from u^k via

(6.24) $(u^{k+1} - u^k)/\Delta t - u^k \partial u^{k+1}/\partial x \simeq \nu \partial^2 u^k/\partial x^2$ on (0,1),

completed by boundary conditions at x = 0 and x = 1. Scheme (6.23),(6.24) is clearly semi-implicit and at each time step it provides an elliptic problem similar to those discussed in Sections 3 and 5, implying therefore, that it can be easily coupled to the wavelet based space approximation described there. Numerical experiments indicates that Scheme (6.23),(6.24) has good stability properties (there exist more sophisticated and accurate schemes; in this paper we limit ourselves to the above scheme since it is well suited for feasibility studies).

6.2.3 A Space-Time Adaptive Wavelet Method for the Regularized Burgers Equation

Equation (6.21) is a regularized version of the inviscid Burgers equation obtained by taking $\nu = 0$. It is a well known fact that the solutions of this equation may develop discontinuities (shocks) even if the initial data u_0 is very smooth. Indeed for small values of ν the solution may develop very strong gradients making the numerical solution of (6.21),(6.22) a nontrivial problem. Various methods can be developed to reproduce accurately the fast variation of the solution near the shock. Such methods include adaptive mesh refinement, entropy control methods (see, e.g., [Tadmor, 1989] and references therein for shock capturing techniques). In this section we describe a space-time adaptive wavelet method to solve the regularized Burgers equation; this method uses, at each time step k, a discretization of problem (6.23),(6.24) based on a suitably chosen subset S^k of scaling functions and wavelets. These scaling functions and wavelets will all belong to some subspace V_n (defined in previous sections) for sufficiently large n. The basic steps of this adaptive method are:

Adaptive Method

(i) Discretize in space the elliptic problem (6.24) using a ($N \geq 3$) wavelet subspace V_n where n is chosen to be sufficiently large so as to represent accurately $\partial u/\partial x$.

(ii) At each time $t = k\Delta t$, compute an initial guess wavelet solution $w_n^{k+1} \in V_n$ of problem (6.21),(6.22) discretized in space using V_n and discretized in time by any **explicit** method (e.g. Forward Euler or Lax-Wendroff) from the known solution u_n^k.

(iii) Use the inverse Mallat transform (recursively using equation (1.35)) to calculate coefficients of the expansion of w_n^{k+1} in terms of a combination of scaling functions and wavelets.

(iv) Choose a subset S^k of the scaling functions and wavelets appearing in the expansion of w_n^{k+1} in (iii) so that the corresponding truncated expansion is sufficiently accurate.

(v) Calculate the solution u_n^{k+1} of Problem (6.23),(6.24) using the approximating subspace spanned by S^k.

This method provides a dynamical in time sequence of Galerkin approximations of the elliptic problems elliptic problem (6.24) for $k \geq 0$.

6.2.4 Numerical Experiments

The first test problem is defined by taking $\nu = 2x10^{-3}$ in (6.21), $u_0(x) = \exp(-8(1-x))$ and

(6.25) $\partial u(0,t)/\partial x = 0$, $u(1,t) = 1$,

as boundary conditions. To solve the above problem we have been combining the semi-implicit Scheme (6.23),(6.24) with a wavelet dicretization based on Vn with N = 3 and n = 6, implying 68 basis functions; the time discretization step $\Delta t = 10^{-3}$. The discrete Neumann-Dirichlet problems occuring at each time step have been solved using the Lagrange multiplier technique described Section 3.2 to treat the Dirichlet condition at x = 0. Figure 6.7 shows the solution at times 0, .03, and .18 ; illustrating the development of a quasi shock starting at t = .03 and fully developed at t = .18.

Lets describe now the second test problem:

For computational convenience we have taken $0 < x < 64$ and assumed periodic boundary conditions for u and $\partial u/\partial x$ at $x = 0$ and $x = 64$. The viscosity parameter ν has been chosen equal to .5 and the initial value u_0 is defined by the piecewise polynomial C^1 function shown in Figure 6.8. This figure also shows the solutions at 5,10,15,20,25, and 30 time steps computed using a finite difference in space and in time discretization with 128 grid points (h = 1/2) and a time step $\Delta t = .5$. These accurate solutions are used as a reference to which we compare our wavelet based solutions.

Figure 6.9 shows (upper left) the finite difference solutions at 5,10,15,20,25, and 30 time steps of this test problem computed using 64 grid points ($h \approx 1$) and time step $\Delta t = .5$; and (upper right) the corresponding errors (compared against the reference solutions). Also shown (lower left) are the (N=3) wavelet solutions of this problem computed using n = 6 (64 basis functions - all scaling functions) and a time step $\Delta t = .5$; and (lower right) the corresponding errors.

Figure 6.10 shows two sets of wavelet solutions at 5,10,15,20,25, and 30 time steps of this test problem computed using (N=3) wavelet solutions of this problem using the space-time adaptive wavelet method described in Section 6.2.3 with n = 6 and a time step $\Delta t = .5$. The top right shows the solutions obtained using, for all k, a subset S^k of V_n consisting of 32 scaling functions and wavelets; and the top right shows the corresponding errors. The bottom right shows the solutions obtained using, for all k, a subset S^k of V_n consisting of 16 scaling functions and wavelets; and the bottom right

shows the corresponding errors.

Comparing the errors shown in Figure 6.9 indicates that the wavelet discretization provides accuracy comparable to finite differences with the same number of degrees of freedom if approximation by scaling functions in V_n is used. The errors shown in the top right of Figure 6.10 indicates that approximation by a combination of scaling functions and wavelets can achieve the same accuracy using significantly fewer degrees of freedom. However, the errors in the bottom right of Figure 6.10 indicate that a significant reduction of accuracy can result if too few degrees of freedom are utilized. We are currently investigating improved space-time adaptive wavelet methods.

7 Linear Advection Problem

This section discusses the following initial value problem:

(7.1) $\partial u/\partial t = \partial u/\partial x \qquad$ for $t > 0$ and $x \in (0,1)$,

(7.2) $u(1,t) = 0 \qquad$ for $t > 0$,

(7.3) $u(x,0) = u_0(x)$.

7.1 Solution of Linear Advection Equation

We solved problem (7.1) using first an explicit Lax-Wendroff time discretization to obtain the following semi discrete problem:

(7.4) $u^0 = u_0$,

(7.5) $u^{k+1} = u^k + \Delta t\, \partial u_n^k/\partial x + .5\, \Delta t^2 \partial^2 u_n^k/\partial x^2$,

followed by the wavelet space discretization to obtain the following scheme

(7.6) $u_n^0 = u_{0n}$,

(7.7) $u_n^{k+1} = u_n^k + \Delta t\, A_n u_n^k + .5\, \Delta t^2 B_n^2 u_n^k$,

where A_n , B_n represent the wavelet approximations of the operators $A = \partial/\partial x$ and $B = \partial^2/\partial x^2$ by the subspace V_n. Scheme (7.6),(7.7) is stable for sufficiently small $\Delta t > 0$ (of the order of h).

7.2 Numerical Experiment

Scheme (7.6),(7.7) was used to compute the wavelet solution of problem (7.1),(7.3) using N=3, n = 6 (68 basis functions), and $\Delta t = .001$. Figure 7.1 shows the initial data (top) and the solution at times t = .410 (middle) and t = .820 (bottom).

8 Further Comments and Conclusion

In this paper we have been exploring the potential of wavelet based approximations for the numerical solution of boundary and initial value problems in one space dimension. For this class of problems, wavelets compare favorably with finite element and finite difference approximation. Indeed it is our opinion that wavelets share some of the computational properties of finite element and spectral methods and that they are well suited for multilevel solution methods. The generalization to multidimensional problems is nontrivial, particularly for curved boundaries; we think however that combining fictitious domain methods methods with wavelets may lead to powerful algorithms for fairly general two and three dimensional domains.

Aknowledgements:

The authors would like to thank L. Auslander, S. Burrus, I. Daubechies, R. Gopinath, J. Huffman, A. Latto, C. H. Li, H. Resnikoff, and R. O. Wells for their help, comments, and suggestions. They acknowledge also the support of DARPA under GRANT # F49620 / 89 / C / 0125

References

[1] J. O. Adams: *Sobolev Spaces*, Academic Press, New York, 1975.

[2] J.F. Bourgat, R. Glowinski, P. LeTallec, and M. Vidrascu, "Variational formulation and algorithm for trace operator in domain decomposition calculations", in *Domain Decomposition Methods*, T. F. Chan, R. Glowinski, J. Periaux, and O. B. Widlund, eds., SIAM, Philadelphia, (1989), pp. 3-16.

[3] A. Brandt: "Multi-level adaptive solutions to boundary value problems", *Mathematics of Computation* 31(138), (1977), pp. 333-390.

[4] H. Brezis: *Analyse fonctionnelle*, Theorie et applications, Masson, Paris, 1983.

[5] W. L. Briggs: *A Multigrid Tutorial*, SIAM, Philadelphia, 1987.

[6] E. Cortina and S. M. Gomes, "A wavelet based numerical method applied to free boundary problems", IPE Technical Report, Sao Jose dos Campos, SP, Brasil, 1989.

[7] G. Dahlquist and A. Bjorck: *Numerical Methods*, Prentice-Hall, New Jersey, 1974.

[8] I.Daubechies: "Orthonormal bases of compact supported wavelets", *Communications on Pure and Applied Mathematics* 51, (1988), pp. 909-996.

[9] I.Daubechies and J. Lagarias: "Two-scale difference equations. I. Global regularity of solutions, and --. II. Infinite matrix products, local regularity and fractals", Preprints AT&T Bell Laboratories; to be published.

[10] R. P. Fedorenko: "A relaxation method for solving elliptic difference equations", *USSR Comput. Math. and Math. Phys.* 1, (1961), p. 1092.

[11] R. P. Fedorenko: "On the speed of convergence of an iterative process", *USSR Comput. Math. and Math. Phys.*4, (1964), p. 227.

[12] R. Glowinski: *Numerical Methods for Nonlinear Variational Problems*, Springer-Verlag, New-York, 1984.

[13] A. Haar: *Math. Ann.* 69, (1910), p. 336.

[14] S. Mallat: "A theory for multiresolution signal decomposition: the wavelet representation", Preprint GRASP Lab., Dept. of Computer and Information Science, Univ. of Pennsylvania, May 1987.

[15] S. Mallat: "Multiresolution approximation and wavelets." Preprint GRASP Lab., Dept. of Computer and Information Science, Univ. of Pennsylvania, September 1987.

[16] McCormick S. F., editor, *Multigrid Methods*, SIAM, Philadelphia, 1987.

[17] Y.Meyer: "Principe d'incertitude, bases hilbertiennes et algebre d'operateurs," Seminaire Bourbaki 38, 662 (1985-86).

[18] Y. Meyer: "Ondelettes et functions splines", Seminaire Equations aux Derivees Partielles, Ecole Polytechnique, Paris, France, December 1986.

[19] Y. Meyer: *Ondelettes et Operateurs*, Hermann, Paris, 1988.

[20] P. A. Raviart and J. M. Thomas, *Introduction a L'Analyse Numerique des Equations aux Derivees Partielles*, Masson, Paris, 1983

[21] H. L. Resnikoff : "Foundations of arithmeticum analysis. II. On compactly supported wavelets with applications to differential equations", Preprint, AWARE, Inc., Cambridge, Massachusetts, March 1988.

[22] T. J. Rivlin: *An Introduction to the Approximation of Functions*, Dover, New York, 1969.

[23] J. C. Strikwerda, *Finite Difference Schemes and Partial Differential Equations*, Wadsworth & Brooks/Cole, Pacific Grove, California, 1989.

[24] E. Tadmor, "Shock capturing by the spectral viscosity method", ICASE Report No. 89-67, September 1989.

DOMAIN DECOMPOSITION METHODS FOR UNSTEADY CONVECTION-DIFFUSION PROBLEMS

Yu. A. Kuznetsov

Department of Numerical Mathematics, USSR Academy of Sciences,

Moscow, USSR

Abstract - This paper deals with systems of linear algebraic equations arising in application of the finite element method to elliptic boundary value problems with singularly perturbed operators. These problems appear in utilization of implicit difference methods for solving parabolic equations including unsteady convection-diffusion problems. To solve FEM-systems, the paper suggests both iterative methods with multilevel domain decomposition preconditioners (DD-preconditioners) and non-iterative DD-methods with overlapping subdomains. The latter methods exploit the property of fast exponential decay of grid Green's functions of singularly perturbed elliptic operators. The justification and practical implementation of the DD-methods suggested are discussed.

1. *INTRODUCTION*

In recent years, the construction, justification and practical implementation of domain decomposition methods (DD-methods) for solving partial differential equations have been causing an ever-increasing interest among specialists in numerical mathematics and mathematical modelling. The review of the up-to-date results obtained in this field can be found in the Proceedings of the 1st and 2nd International Symposiums on Domain Decomposition Methods [5,6]. A great progress was made in constructing and justifying methods to solve elliptic problems [2,3,7,8,11,13,14].

This paper suggests two types of algorithms of the DD-method for approximate realization of elliptic difference schemes for unsteady convection-diffusion problems.

The first group of algorithms is based on the standard idea of exploiting a positive definite preconditioner in the iterative procedure. For such preconditioner the paper suggests to use the multilevel DD-preconditioner for a symmetric positive definite elliptic operator [10], which is chosen spectrally equivalent to the non-symmetric coercitive operator of the original problem.

The second group of DD-methods is based on a new idea of replacing the grid problem in the original domain by a series of grid problems for subdomains of a smaller size [9,12]. These subproblems being solved, a grid function is constructed which approximates the grid function, solution of the original problem, with a prescribed accuracy. Section 4 discussed mathematical faundations for such replacement while Sections 5-7 consider specific versions of the DD-method with overlapping subdomains and particular features of their implementation.

2. PROBLEM FORMULATION

Let Ω be a two-dimensional polygonal domain with the boundary $\partial\Omega$ and Γ_1 be a closed subset of $\partial\Omega$, consisiting of a finite number of segments of stright lines.

Let us consider the unsteady convection diffusion problem

$$\begin{aligned} &\frac{\partial u}{\partial t} - \nabla(a\nabla u) + \vec{b}\cdot\nabla u + cu = f \quad \text{in} \quad \Omega \times (0;T] , \\ &u = 0 \qquad\qquad \text{on} \quad \Gamma_1 \times (0;T] , \\ &a\frac{\partial u}{\partial \nu} + \sigma u = 0 \qquad \text{on} \quad \Gamma_2 \times (0;T] , \\ &u(0) = u^0 \qquad\qquad \text{in} \quad \Omega . \end{aligned} \tag{2.1}$$

Here, $T = \text{const} > 0$; Γ_2 is a subset of $\partial\Omega$, consisting of a finite number of open segments ($\Gamma_1 \cap \Gamma_2 = \emptyset$, $\Gamma_1 \cup \bar{\Gamma}_2 = \partial\Omega$); ν is the vector of the external normal to Γ_2; $\vec{b} = (b_1, b_2)$; a, b_1, b_2, c, f, σ and u^0 are given bounded piecewise-smooth functions. It is assumed that $a \geq a_0 = \text{const} > 0$ in Ω and $\sigma \geq 0$ on Γ_2.

Define $V = \{v:\ v \in H^1,\ v = 0 \text{ on } \Gamma_1\}$ as a subspace of the Sobolev space

H^1 with the same norm $\|\cdot\|_V = \|\cdot\|_{H^1}$ and the bilinear form

$$a(u,v) = \int_\Omega [a\nabla u\cdot\nabla v + (\vec{b}\cdot\nabla u)v + cuv]d\Omega + \int_{\Gamma_2} \sigma uv\, d\Gamma . \qquad (2.2)$$

We assume the form $a(u,v)$ to be positive semi-definite on V.

Under the assumptions made, problem (2.1) can be formulated as follows: find $u = u(t) \in V$ such that $u(0) = u^0$ and for each $t \in (0;T]$ the following identity is valid:

$$(\frac{du}{dt}, v) + a(u,v) = (f,v) \quad \forall v \in V , \qquad (2.3)$$

where $(\ ,\ )$ is an ordinary scalar product in the space $L_2(\Omega)$.

Let Ω_h be a triangulation of Ω, such that $\Gamma_1 \cap \bar{\Gamma}_2$ belongs to the set of vertices of triangles from Ω_h and V_h be a standard piecewise-linear finite element subspace of V [4]. Apply to solving problem (2.3) the simplest implicit scheme of the first order accuracy in time with the FEM-approximation in spatial variables. The discrete problem can be formulated as follows: for $k = 1,\ldots,m$ find functions $u_h^k \in V_h$ such that

$$\left(\frac{u_h^k - u_h^{k-1}}{\Delta t}\ ,\ v_h \right) + a(u_h^k, v_h) = (f, v_h) \quad \forall v_h \in V_h . \qquad (2.4)$$

Here, $\Delta t = T/m$, m is a positive integer and u_h^0 denotes an approximation of the function u^0.

For the sake of simplicity, assume the funciton u_h^0 to belong to V_h. Then (2.4) can be replaced with another formulation of the discrete problem, which is more convenient for what follows: for $k = 1,\ldots,m$ find $w_h = u_h^k - u_h^{k-1} \in V_h$ such that

$$(w_h^k, v_h) + \Delta t\cdot a(w_h^k, v_h) = -\Delta t\cdot \xi^k(v_h) \quad \forall v_h \in V_h, \qquad (2.5)$$

where

$$\xi^k(v) = a(u_h^{k-1}, v) - (f,v) . \qquad (2.6)$$

Problem (2.5)-(2.6) for each $k \geq 1$ leads to the system of linear algebraic equations

$$Aw = g \tag{2.7}$$

with a positive definite (in the Euclidean space R^N) $N \times N$ matrix

$$A = M + \Delta t \cdot K \tag{2.8}$$

and a vector $g \in R^N$. Here, M is the mass matrix and K is a stiffness matrix generated by the form $a(u, v)$.

This paper is aimed at constructing methods of approximate solution of system (2.7)-(2.8) which are based on the domain decomposition ideas. To this end, we need auxiliary matrices.

Let us prescribed a symmetric elliptic form

$$\tilde{a}(u, v) = \int_{\Omega} [\tilde{a} \nabla u \cdot \nabla v + \vec{\tilde{b}} \cdot \nabla (uv) + \tilde{c} uv] d\Omega + \int_{\Gamma_2} \tilde{\sigma} uv \, d\Gamma , \tag{2.9}$$

whose coefficients $\tilde{a}$, $\tilde{b}_1$, $\tilde{b}_2$, $\tilde{c}$ and $\tilde{\sigma}$ possess the properties similar to those of the coefficients of the form $a(u, v)$. We assume the form $\tilde{a}(u, v)$ to be equivalent to the form $a(u, v)$ in the sense that

$$c_1 \tilde{a}(v, v) \leq a(v, v) \leq c_2 \tilde{a}(v, v) \quad \forall v \in V , \tag{2.10}$$

where c_1 and c_2 are positive constants.

Using the relations

$$(\tilde{K} u, v) = \tilde{a}(u^h, v^h) \quad \forall u^h, v^h \in V_h \ (u, v \in R^N) \tag{2.11}$$

determine a symmetric $N \times N$ matrix $\tilde{K}$ and prescribe the $N \times N$ matrix

$$\tilde{A} = M + \Delta t \cdot \tilde{K} . \tag{2.12}$$

It is obvious that

$$c_1 (\tilde{K} v, v) \leq (Kv, v) \leq c_2 (\tilde{K} v, v) \quad \forall v \in R^N, \tag{2.13}$$

where c_1 and c_2 are taken from (2.10). This implies in particular that if the matrix K is positive definite (semi-definite), then the matrix $\tilde{K}$ will be also

positive definite (semi-definite). In any case, the matrices A and $\tilde{A}$ are simultaneously positive definite.

Let us formulate some statements whose proofs are directly implied by the above assumptions.

Statement 2.1 The following inequality is valid:

$$\tilde{c}_1(\tilde{A}v, v) \leq (Av, v) \leq \tilde{c}_2(\tilde{A}v, v) \quad \forall v \in R^N, \tag{2.14}$$

where $\tilde{c}_1$ and $\tilde{c}_2$ are positive constants.

Consider the eigenvalue problem

$$\mu\tilde{A}v = Av \ . \tag{2.15}$$

Statement 2.2. The eigenvalues of problem (2.15) belong to the rectangle $[\tilde{c}_1; \tilde{c}_2]\times[-d; d]$ of complex plane, where $\tilde{c}_1$ and $\tilde{c}_2$ are constants from inequalities (2.14) and d is a positive constant.

Apply to solving system (2.7) the iterative method

$$\tilde{A}(w^j - w^{j-1}) = -\alpha(Aw^{j-1} - g) \ , \quad j = 1,2,... \tag{2.16}$$

with a constant parameter $\alpha > 0$. Then Statement 2.2 implies the following

Statement 2.3. There exists $\hat{\alpha} = \text{const} > 0$ such that for any $\alpha \in (0; \hat{\alpha}]$ iterative method (2.16) converges at the rate of geometric progression with the factor $q = q(\alpha) = \text{const} < 1$.

Remark 2.1. Here and henceforth, we mean constants independent of the grid Ω_h and of the quantity Δt. Obviously these constants can depend on the coefficients of the bilinear forms, the geometry of the domain Ω and the structure of the set Γ_1.

Let us prescribe a symmetric positive definite $N\times N$ matrix B such that

$$\hat{c}_1(Bv, v) \leq (\tilde{A}v, v) \leq \hat{c}_2(Bv, v) \quad \forall v \in R^N, \tag{2.17}$$

where $\hat{c}_1$ and $\hat{c}_2$ are positive constants, and apply to solving system (2.7) the

iterative method

$$B(w^j - w^{j-1}) = -\alpha(Aw^{j-1} - g) , \quad j = 1,2,\dots , \tag{2.18}$$

with a constant parameter $\alpha > 0$.

Under the assumptions made, the following statement is valid.

Statement 2.4. There exists $\hat{\alpha} = \text{const} > 0$ such that for any $\alpha \in (0;\hat{\alpha}]$ iterative method (2.18) converges at the rate of geometric progression with the factor $q \equiv q(\alpha) = \text{const} < 1$.

3. TWO-LEVEL DOMAIN DECOMPOSITION METHOD

In this section, we will construct a two-level DD-preconditioner for the matrix A of system (2.7), which will satisfy conditions (2.17) of Statement 2.4.

To this end, we partition the grid domain Ω_h into grid subdomains $G_{1,h}$, $G_{2,h}$ and $G_{3,h}$ as shown in Fig.1. In other words, we assume that $\text{mes}(\partial G_{1,h} \cap \partial G_{3,h}) = 0$ and each subdomain $G_{i,h}$ is partitioned into non-overlapping subdomains $G_{i,h}^{(j)}$, $j = 1,\dots,m_i$, where m_i, $i = 1,2\ 3$, are positive integers. Obviously each of $G_{i,h}$, $i = 1,2,3$, is a union of triangles from Ω_h.

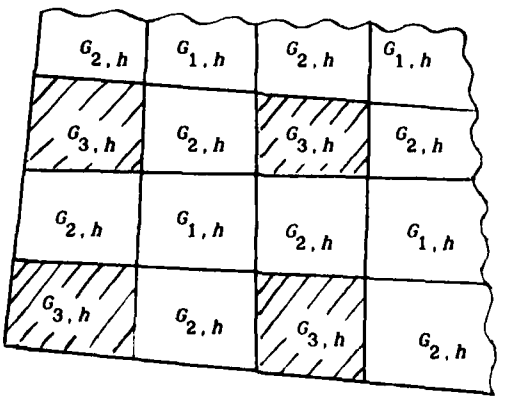


Figure 1. Partitioning Ω_h into subdomains $G_{1,h}$, $G_{2,h}$ and $G_{3,h}$.

Let us partition the grid domain Ω_h into grid subdomains $\Omega_{1,h} = G_{1,h} \cup G_{2,h}$ and $\Omega_{2,h} = G_{3,h}$ (see Fig.2) and denote by γ_h the boundary between the subdomains $\Omega_{1,h}$ and $\Omega_{2,h}$, i.e. $\gamma_h = \partial\Omega_{1,h} \cap \partial\Omega_{2,h}$. Then define the stiffness matrices

$$\begin{bmatrix} \tilde{A}_1 & \tilde{A}_{1\gamma} \\ \tilde{A}_{\gamma 1} & \tilde{A}_{\gamma}^{(1)} \end{bmatrix}, \quad \begin{bmatrix} \tilde{A}_{\gamma}^{(2)} & \tilde{A}_{\gamma 2} \\ \tilde{A}_{2\gamma} & \tilde{A}_2 \end{bmatrix} \tag{3.1}$$

generated by the bilinear form $(u,v) + \Delta t \cdot \tilde{a}(u,v)$ for the subdomains $\Omega_{1,h}$ and $\Omega_{2,h}$ which are considered as superelements of the grid domain Ω_h. It is obvious that

$$\tilde{A} = \left[\begin{array}{cc|c} \tilde{A}_1 & \tilde{A}_{1\gamma} & 0 \\ \tilde{A}_{\gamma 1} & \tilde{A}_{\gamma}^{(1)} + \tilde{A}_{\gamma}^{(2)} & \tilde{A}_{\gamma 2} \\ \hline 0 & \tilde{A}_{2\gamma} & \tilde{A}_2 \end{array}\right] \equiv \begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} \\ \tilde{A}_{21} & \tilde{A}_{22} \end{bmatrix}. \tag{3.2}$$

Finally, following the standard technique of domain decomposition methods with alternating Neumann-Dirichlet boundary conditions [2,7,11,14] we define the $N \times N$ matrix [3]

$$\tilde{B} = \left[\begin{array}{cc|c} \tilde{A}_1 & \tilde{A}_{1\gamma} & 0 \\ \tilde{A}_{\gamma 1} & \tilde{A}_{\gamma}^{(1)} + \tilde{A}_{\gamma 2}\tilde{A}_2^{-1}\tilde{A}_{2\gamma} & \tilde{A}_{\gamma 2} \\ \hline 0 & \tilde{A}_{2\gamma} & \tilde{A}_2 \end{array}\right] \equiv \begin{bmatrix} \tilde{B}_{11} + \tilde{A}_{12}\tilde{A}_{22}^{-1}\tilde{A}_{21} & \tilde{A}_{12} \\ \tilde{A}_{21} & \tilde{A}_{22} \end{bmatrix}, \tag{3.3}$$

where

$$\tilde{B}_{11} = \begin{bmatrix} \tilde{A}_1 & \tilde{A}_{1\gamma} \\ \tilde{A}_{\gamma 1} & \tilde{A}_{\gamma}^{(1)} \end{bmatrix} \tag{3.4}$$

is the stiffness matrix for the subdomain $\Omega_{1,h}$. We call the matrix $\tilde{B}$ from (3.3) the one-level DD-preconditioner for the matrix $\tilde{A}$ from (3.2).

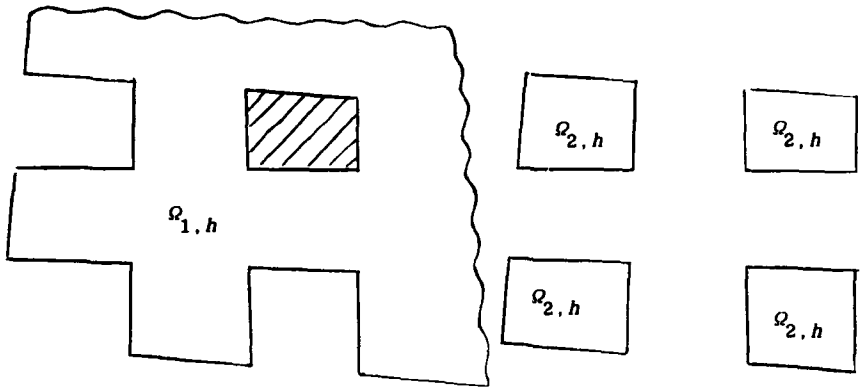


Figure 2. Partitioning Ω_h into subdomains $\Omega_{1,h} = G_{1,h} \cup G_{2,h}$ and $\Omega_{2,h} = G_{3,h}$.

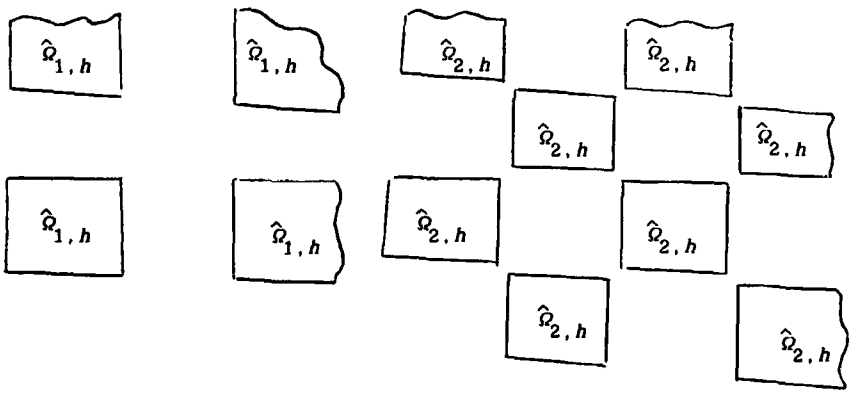


Figure 3. Partitioning $\hat{\Omega}_h = \Omega_{1,h}$ into subdomains $\hat{\Omega}_{1,h} = G_{1,h}$ and $\hat{\Omega}_{2,h} = G_{2,h}$.

Continue the procedure of construction of the two-level DD-preconditioner. To this end, set $\hat{\Omega}_h = \Omega_{1,h}$ and $\hat{A} = \tilde{B}_{11}$. Then partition $\hat{\Omega}_h$ into two grid subdomains $\hat{\Omega}_{1,h} = G_{1,h}$ and $\hat{\Omega}_{2,h} = G_{2,h}$ (see Fig.3) with the common boundary $\hat{\gamma}_h = \partial G_{1,h} \cap \partial G_{2,h}$ and define the stiffness matrices

$$\hat{B}_{11} = \begin{bmatrix} \hat{A}_1 & \hat{A}_{1\hat{\gamma}} \\ \hat{A}_{\hat{\gamma}1} & \hat{A}_{\hat{\gamma}}^{(1)} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \hat{A}_{\hat{\gamma}}^{(2)} & \hat{A}_{\hat{\gamma}2} \\ \hat{A}_{2\hat{\gamma}} & \hat{A}_2 \end{bmatrix} \tag{3.5}$$

generated by the bilinear form $(u,v) + \Delta t \cdot \tilde{a}(u,v)$ for the subdomains $\hat{\Omega}_{1,h}$ and $\hat{\Omega}_{2,h}$

which are considered as superelements of the grid domain $\hat{\Omega}_h$. It is obvious that

$$\hat{A} = \left[\begin{array}{c|c|c} \hat{A}_1 & \hat{A}_{1\hat{\gamma}} & 0 \\ \hline \hat{A}_{\hat{\gamma}1} & \hat{A}_{\hat{\gamma}}^{(1)}+\hat{A}_{\hat{\gamma}}^{(2)} & \hat{A}_{\hat{\gamma}2} \\ \hline 0 & \hat{A}_{2\hat{\gamma}} & \hat{A}_2 \end{array}\right] \equiv \left[\begin{array}{cc} \hat{A}_{11} & \hat{A}_{12} \\ \hat{A}_{21} & \hat{A}_{22} \end{array}\right]. \tag{3.6}$$

Finally, we define the matrix

$$\hat{B} = \left[\begin{array}{cc} \hat{B}_{11}+\hat{A}_{12}\hat{A}_{22}^{-1}\hat{A}_{21} & \hat{A}_{12} \\ \hat{A}_{21} & \hat{A}_{22} \end{array}\right] \tag{3.7}$$

as the one-level DD-preconditioner for the matrix $\hat{A}$ from (3.6).

The resulting two-level DD-preconditioner [10] for the matrix $\tilde{A}$ from (3.2) will be defined by the formula

$$B = \left[\begin{array}{cc} \hat{B}+\tilde{A}_{12}\tilde{A}_{22}^{-1}\tilde{A}_{21} & \tilde{A}_{12} \\ \tilde{A}_{21} & \tilde{A}_{22} \end{array}\right]. \tag{3.8}$$

The eigenvalues of the matrices $\tilde{B}^{-1}\tilde{A}$ and $\hat{B}^{-1}\hat{A}$ are known to belong to the segments $[1;\tilde{d}]$ and $[1;\hat{d}]$, respectively, where $\tilde{d}$ and $\hat{d}$ are positive integers greater than unity. It is easy to show [10] that in this case the eigenvalues of the matrix $B^{-1}\tilde{A}$ belong to the segment $[1;d]$ where $d = \tilde{d}\hat{d}$. Under certain conditions in which the so-called extension theorem [1,15,16] is valid, the numbers $\tilde{d}$, $\hat{d}$ and, hence, d are independent of the grid Ω_h and the grid step size Δt. Now we formulate some requirements imposed on the sizes of subdomains $G_{l,h}^{(j)}$, under which the quantities $\tilde{d}$ and $\hat{d}$ will be independent of the grids.

For each of the simply connected subdomains $G_{l,h}^{(j)}$ we define inscribed and circumscribed circles with the radii $r_{l,h}^{(j)}$ and $R_{l,h}^{(j)}$, respectively. Assume that positive constants α_1, α_2 and $\varkappa$ exist such that

$$\alpha_1(\Delta t)^{\varkappa} \leq r_{l,h}^{(j)} < R_{l,h}^{(j)} \leq \alpha_2(\Delta t)^{\varkappa}$$

and for each subdomain $G_{i,h}^{(j)}$ there exists its mapping onto the square with the side length $(\Delta t)^{\varpi}$, which is given by a function from H_h and has an inverse mapping bounded by a constant S.

Under the assumptions made, the following statement is valid.

Statement 3.1. For any $\varpi \leq 0.5$ the eigenvalues of the matrix $B^{-1}\tilde{A}$ belong to the segment $[1;d]$, where d is a positive constant.

This statement and Statement 2.4 directly imply

Statement 3.2. If for the preconditioner B for the matrix A of system (2.7) we choose the matrix B from (3.8) provided $\varpi \leq 0.5$, then there exists $\tilde{\alpha} = \text{const} > 0$ such that for any $\alpha \in (0;\tilde{\alpha}]$ iterative method (2.18) converges at the rate of geometric progression with the factor $q \equiv q(\alpha) = \text{const} < 1$.

4. GRID GREEN FUNCTION ESTIMATE

Statement 3.2 implies that for approximate solution of system (2.7) by method (2.18) with accuracy $\varepsilon \in (0;1)$ in the sense of the inequality

$$\|w_h^{j_\varepsilon} - w_h\|_{L_2(\Omega)} \leq \varepsilon \|g_h\|_{H^1(\Omega)} \tag{4.1}$$

being valid, it is sufficient to choose $j_\varepsilon = c_0 \ln \varepsilon^{-1}$, where c_0 is a positive constant.

Let $\hat{\Omega}_h$ be a subdomain of the mesh domain Ω_h and $\operatorname{supp} g_h \subset \hat{\Omega}_h$. Embed $\hat{\Omega}_h$ into a subdomain $\hat{\Omega}_{h,\varepsilon} \subset \Omega_h$ (see Fig.4) such that any $x = (x_1, x_2) \in Q_{h,\varepsilon} = \Omega_h \backslash \hat{\Omega}_{h,\varepsilon}$ satisfies the inequality

$$\rho(x;\hat{\Omega}_h) = \min_{y \in \hat{\Omega}_h} |x - y| \geq \hat{c}(\Delta t)^{\varpi} \ln \varepsilon^{-1}, \tag{4.2}$$

where $\hat{c}$ is a positive constant and $\varpi = 0.5$.

Statement 4.1. Under the assumptions made, a positive constant $\hat{c}$ exists such that $w_h^{j_\varepsilon} = 0$ for any $x \in Q_{h,\varepsilon}$ (see Fig.4).

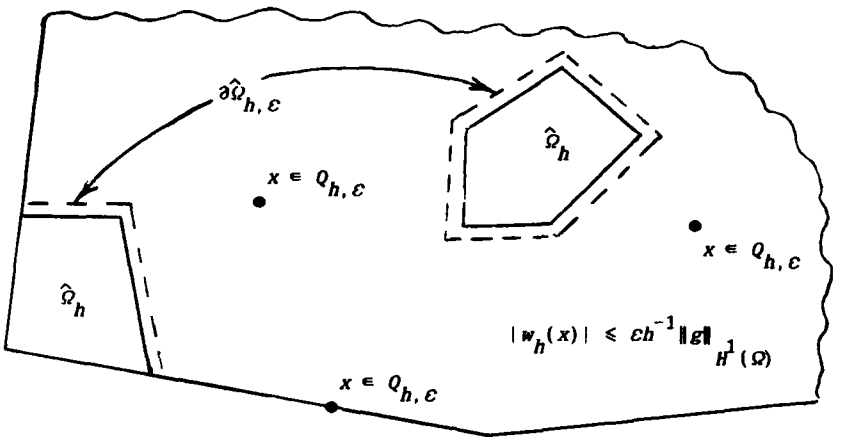


Figure 4. Interpretation of the behaviour of the grid Green function.

This statement and inequality (4.1) directly imply validity of the inequality

$$\|w^h\|_{L_2(Q_{h,\varepsilon})} \leq c\|g_h\|_{H^1(\Omega)}, \tag{4.3}$$

from which in case of regular meshes ($h \sim N^{1/2}$) we have

$$\|w^h\|_{C(Q_{h,\varepsilon})} \leq ch^{-1}\|g_h\|_{H^1(\Omega)}. \tag{4.4}$$

We have thus proved that when the distance from the point of location of the source function increases the grid Green's function of the operator $M + \Delta t \cdot K$ decreases as $\exp(-\text{const}\,|y - x|/\sqrt{\Delta t})$. This fact is in complete harmony with the results obtained before for the case of symmetric bilinear forms [9,12], specifically for diffusion problems.

In practice, as a rule, one chooses $\varepsilon \sim (\Delta t)^s$, where $s \in [1;2]$. In this case, the boundary of the domain $\hat{\Omega}_{h,\varepsilon}$ lies away from the boundary of the domain $\hat{\Omega}_h$ at the distance of order $(\Delta t)^s \ln(\Delta t)^{-1}$, which is for small values of Δt considerably less than the diameter of the domain Ω.

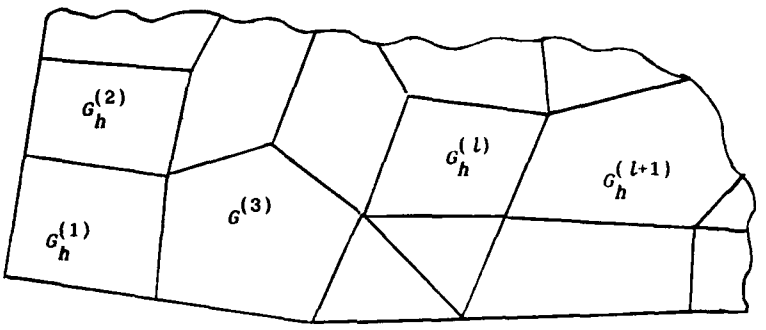


Figure 5. Partitioning Ω_h into subdomains $G_h^{(l)}$, $l = 1,\ldots,p$.

Note that from the standpoint of the arithmetic cost of realization of one step of method (2.18) it is preferable to choose $\varpi = 1/2$, i.e. to choose the greatest value of ϖ for which this method has the convergence rate with the *factor* q *independent of the grid* Ω_h. There is no reason for supposing that if the value of ϖ is decreased, the convergence rate of method (2.18) can be considerably improved.

5. NON-ITERATIVE DD-METHOD WITH OVERLAPPING SUBDOMAINS

The previous section leads us to a new approach to constructing DD-methods for approximate solution of system (2.8). This approach was anounced and justified for diffusion problems in [9,12]. Here, we will consider two versions of this approach under the assumption that the grid Ω_h is regular.

Let us partition the grid domain Ω_h into $p > 1$ grid subdomains $G_h^{(1)},\ldots,G_h^{(p)}$ as shown, for example, in Fig.5 and determine vectors $g_h^{(l)} \in R^N$ by restricting linear form (2.6) onto subdomains $G_h^{(l)}$, $l = 1,\ldots,p$. In other words, we determine functions $g_h^{(l)} \in V^N$ such that $\mathrm{supp}\, g_h^{(l)} \in G_h^{(l)}$, $l = 1,\ldots,p$, and $\sum_{l=1}^{p} g_h^{(l)} = g_h$.

The linearity of system (2.8) implies that

$$w = \sum_{l=1}^{p} w^{(l)} , \tag{5.1}$$

where $w^{(l)}$ are solutions to the systems

$$Aw^{(l)} = g^{(l)} , \quad l = 1, \ldots, p . \tag{5.2}$$

Our aim is to compute the vector w with accuracy ε in the uniform norm, i.e. to find the vector $\hat{w} \in R^N$ satisfying the inequality

$$\|\hat{w} - w\|_\infty \leq \varepsilon . \tag{5.3}$$

Fix the value of $l \geq 1$ and embed the subdomain $G_h^{(l)}$ into the grid subdomain $G_{h,\varepsilon}^{(l)}$ such that all points $x \in Q_{h,\varepsilon}^{(l)} = \Omega_h \backslash G_{h,\varepsilon}^{(l)}$ satisfy the inequality

$$\rho(x; G_h^{(l)}) \geq \hat{c}(\Delta t)^{1/2} \ln\left(\frac{\varepsilon h}{p} \right)^{1/2} \tag{5.4}$$

with a constant $\hat{c}$ (see Fig.6). Denote by $A^{(l)}$ the stiffness matrix for the subdomain $G_{h,\varepsilon}^{(l)}$ considered as a superelement and by $g_\varepsilon^{(h)}$ the restriction of the vector $g^{(l)}$ onto the same subdomain, and consider the system

$$A^{(l)} \hat{w}_\varepsilon^{(l)} = g_\varepsilon^{(l)} . \tag{5.5}$$

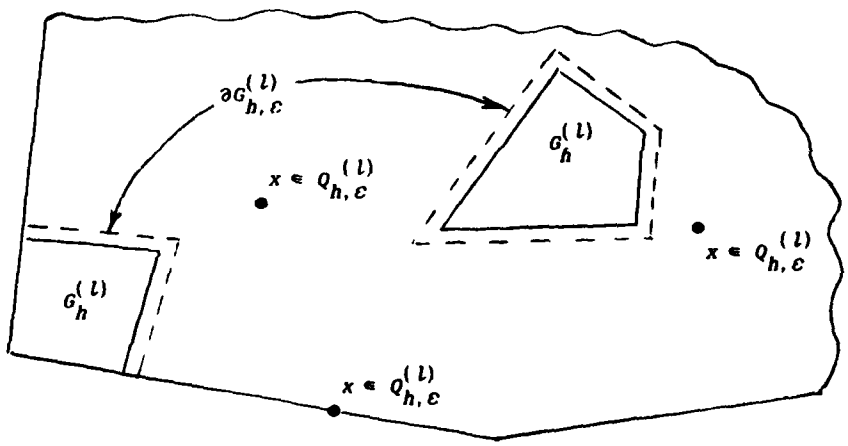


Figure 6. Embedding $G_h^{(l)}$ into $G_{h,\varepsilon}^{(l)}$.

Finally, denote by $\hat{w}^{(l)}$ the vector from R^N whose restriction onto $G^{(l)}_{h,\varepsilon}$ is equal to the vector $\hat{w}^{(l)}_{\varepsilon}$ and onto $Q^{(l)}_{h,\varepsilon}$ is the zero vector.

Statement 5.1. There exists a value of the constant $\hat{c}$ from inequality (5.3) such that the vector $\hat{w}^{(l)}$ constructed above satisfies the inequality

$$\|\hat{w}^{(l)} - w^{(l)}\|_{\infty} \leq \frac{\varepsilon}{p} . \tag{5.6}$$

This implies that the vector

$$\hat{w} = \sum_{l=1}^{p} \hat{w}^{(l)} \tag{5.7}$$

satisfies inequality (5.3), i.e. approximates the solution to system (2.8) with accuracy ε in the uniform norm.

Let us briefly discuss the other version of the approach to constructing DD-methods with overlapping subdomains. Choose a subdomain G_h of the grid domain Ω_h and formulate the problem: find the components of the vector, the solution to system (2.7), corresponding to this subdomain with accuracy ε. In other words, if we denote by w_G the restriction of the vector w onto the subdomain G, then it is necessary to find a vector $w_{G,\varepsilon}$ such that the following inequality is valid:

$$\|w_G - w_{G,\varepsilon}\|_{\infty} \leq \varepsilon . \tag{5.8}$$

To solve the problem formulated, embed the subdomain G_h into the subdomain $G_{h,\varepsilon}$ such that the inequality

$$\rho(x; G_h) \geq \hat{c}(\Delta t)^{1/2} \ln(\varepsilon h^{-1}) \tag{5.9}$$

is valid for any point $x \in Q_{h,\varepsilon} = \Omega_h \setminus G_{h,\varepsilon}$. Then denote by $\hat{\lambda}_{\varepsilon}$ the stiffness matrix for the domain $G_{h,\varepsilon}$ and by $\hat{g}_{\varepsilon}$ the restriction of the vector on the right-hand side of system (2.7) onto the same subdomain, and consider the system

$$\hat{\lambda}_{\varepsilon}\hat{w}_{\varepsilon} = \hat{g}_{\varepsilon} . \tag{5.10}$$

Finally, denote by $w_{G,\varepsilon}$ the restriction of the vector $\hat{w}_{\varepsilon}$ onto the subdomain G_h.

Statement 5.2. There exists a value of the constant $\hat{c}$ from inequality (5.9), such that the constructed vector $w_{G,\varepsilon}$ satisfies inequality (5.8).

It is obvious that choosing various subdomains G_h of the domain Ω_h in a successive or parallel way we can compute all the components of the vector, the solution to the system (2.7), with a given accuracy. When solving problems for subdomains we can use different direct and iterative methods. The estimates of the total computational cost in case of the diffusion problem and rectangular piecewise-uniform meshes were given in [9].

6. CONVECTION-DIFFUSION PROBLEMS FOR SUBDOMAINS

In DD-methods with overlapping subdomains considered in the previous section, it is necessary to solve repeatedly convection-diffusion problems in subdomains of a small size. Below, we will note a characteristic property of such problems which enables us to construct very efficient computational algorithms to solve them.

For the sake of simplicity, we confine ourselves to considering the differential problem in the square G with the side length $(\Delta t)^{\varepsilon}$, where $\varepsilon \in (0;1/2]$. Thus, let us consider the equation

$$\begin{aligned} w + \Delta t\cdot[-\nabla(a\nabla w) + \vec{b}\cdot\nabla w] &= g \quad \text{in} \quad G\,, \\ w &= 0 \quad \text{on} \quad \partial G \end{aligned} \tag{6.1}$$

provided $\operatorname{div}\vec{b} = 0$.

Map the square G onto the unit square G_0, i.e. introduce new variables $\tilde{x} = (\Delta t)^{-\varepsilon}x$. As a result, we arrive at the differential problem

$$\begin{aligned} \tilde{w} + (\Delta t)^{1-2\varepsilon}[-\tilde{\nabla}(\tilde{a}\tilde{\nabla})\tilde{w} + (\Delta t)^{\varepsilon}\cdot\vec{\tilde{b}}\cdot\tilde{\nabla}\tilde{w}] &= \tilde{g} \quad \text{in} \quad G_0 \\ \tilde{w} &= 0 \quad \text{on} \quad \partial G_0 \end{aligned} \tag{6.2}$$

This implies that with respect to small subdomains the role of the convection term in equation (6.1) becomes considerably less as compared with the diffusion term than with respect to the original domain.

Apply to solving the system

$$Aw = g \tag{6.3}$$

arising as a result of the FEM-approximation of the problem (6.1) the iterative method

$$B(w^j - w^{j-1}) = -\alpha(Aw^{j-1} - g) \ , \quad j = 1,2,\dots \ , \tag{6.4}$$

with a constant parameter $\alpha > 0$ and the matrix $B = \frac{1}{2}(A + A^T)$.

Statement 6.1. Under the assumption made, there exists $\hat{\alpha} = \text{const} > 0$ such that for any $\alpha \in (0;\hat{\alpha}]$ method (6.4) converges at the rate of geometric progression with the factor $q = c(\Delta t)^{\frac{1}{2}} < 1$, where c is a positive constant.

To solve system (6.3), it is thus possible to suggest different iterative methods with the preconditioners B spectrally equivalent to the matrix $A + A^T$. In this case, the smaller the diameters of the subdomains G_h, the faster the convergence rate of the method.

7. CONCLUSION

Since this paper does not contain proofs in detail and considers only two-dimensional problems, we would like to make some comments.

The fact of exponential decay of grid Green's function of the matrix $M + \Delta t \cdot K$ is known to many scientists, and in number of cases it can be established analytically or by using obvious arguments [9]. However, insufficient attention has been paid so far to the application of this prperty. In our view, with the requirement for high precision of numerial computations rising and, hence, with finer spatial grids and smaller values of Δt used, the importance of domain decomposition methods will be increasing. As seen from Section 5, these

methods are very convenient for implementation on parallel computers.

The results of this paper can be considerably improved if we impose some additional constraints on the forms $a(u,v)$ and the grids. These results can be extended in an obvious way to the three-dimensional case. In Section 3, for example, the two-level DD-preconditioner can be replaced by the three-level one with the domain Ω_h partitioned into cubes. The implicit scheme of the first order accuracy in time can be replaced by the Crank-Nicholson scheme and all the main results remain valid. All these problems still need be theoretically and practically investigated in details.

REFERENCES

1. G. P. Astrakhantsev, Iterative methods for solving veriational difference schemes for two dimensional second order elliptic equations. Doctoral thesis, *LOMI Akad. Nauk SSSR, Leningrad*, 1972 (in Russian).

2. P. E. Bjorstad and O. B. Widlund, Iterative methods for the solution of elliptic problems on regions for the solution of elliptic problems on regions partitioned into substructures, *SIAM J. Numer. Anal.* (1986) 23, 1097-1120.

3. J. H. Bramble, J. E. Pasciak and A. H. Schatz, An iterative method for elliptic problems on regions partitioned into substructures. *Math. Comp.*, Vol.46, 1986, p.361-389.

4. P. G. Ciarlet, The Finite Element Methods for Elliptic Problems, North-Holland, Amsterdam-New York, 1978.

5. Domain Decomposition Methods for Partial Differential Equations, Proceedings of the 1st Int. Symp., Eds. R. Glowinski, G. H. Golub, G. A. Meurant, J. Periaux, SIAM, Philadelphia, 1988.

6. Domain Decomposition Methods, Proceedings of the 2nd Int. Symp., Eds. T. F. Chan, R. Glowinski, J. Periaux, O. B. Widlund, SIAM, Philadelphia, 1989.

7. M. Dryja, A decomposition method for solving finite element equations, Inst. Inform., Report No.122, UW, Warsaw, 1983, p.18-47.

8. R. Glowinski, J. Periaux and Q. V. Dihn, Domain decomposition methods for

nonlinear problems in fluid dynamics, INRIA Rapports de Recherche, No.147, 1982.

9. Yu. A. Kuznetsov, New algorithms for approximate realization of implicit difference schemes, *Soviet J. Numer. Anal. and Math. Modelling*, Vol.3, No.2, 1988, p.99-114.

10. Yu. A. Kuznetsov, Multilevel domain decomposition methods, In: *Numerical analysis and mathematical modelling*, Dept. Numer. Math. USSR Acad. Sci., Moscow, 1989, p.109-126 (in Russian, submitted to the special issue of Applied Numerical Mathematics on domain decomposiiton methods, Ed. W. Proskurowski, 1989).

11. V. I. Lebedev and V. I. Agoshkov, Poincare-Steklov operators and their applications in analyses, Dept. of Numerical Math. USSR Academy Sci., Moscow, 1983 (in Russian).

12. G. I. Marchuk and Yu. A. Kuznetsov, Approximate algorithms for implicit difference schemes, In: *Analyse Mathematique et Applications*, Gauthier-Villars, Paris, 1988, p.357-371.

13. G. I. Marchuk, Yu. A. Kuznetsov and A. M. Matsokin, Fictitious domain and domain decomposition methods. *Soviet J. Numer. Anal. and Math. Modelling*, Vol.1, No.1, 1986, p.3-35.

14. A. M. Matsokin, Fictitious components method and the modified difference analogue of the Schwartz method. In: *Vychislitel'nye Metody Lineinoi Algebry*, ed. G. I. Marchuk, Vychisl. Tsentr Sib. Otdel. Acad. Nauk SSSR, Novosibirsk, 1980, p.66-77 (in Russian).

15. A. M. Matsokin and S. V. Nepomnyaschikh, On the convergence of the alternating subdomain Schwartz method without intersections. In: *Metody Interpolyatsii i Approksimatsii*, ed. Yu. A. Kuznetsov, Vychisl. Tsentr Sib. Otdel. Acad. Nauk SSSR, Novosibirsk, 1981, p.85-97 (in Russian).

16. O. Widlund, An extension theorem for finite element spaces with there applications. Technical Report 233, Dept. of Computer Science, Courant Inst., New York University, 1986.

SIMULATION NUMERIQUE EN PHYSIQUE ET CHIMIE

COMPUTATIONAL PHYSICS AND CHEMISTRY

Localized Basis Functions and Other Computational Improvements in Variational Nonorthogonal Spectral Methods for Quantum Mechanical Scattering Problems Involving Chemical Reactions†

David W. Schwenke
Mail Stop 230-3
NASA Ames Research Center, Moffett Field, CA 94035, USA

Donald G. Truhlar
Department of Chemistry and Supercomputer Institute
University of Minnesota, Minneapolis, MN 55455, USA

Donald J. Kouri
Department of Chemistry and Department of Physics
University of Houston, Houston, TX 77204, USA

Abstract. In this paper we very briefly review the Generalized Newton Variational Principle for 3-dimensional quantum mechanical reactive scattering. Then three techniques are described which improve the efficiency of the computations. First we use the fact that the Hamiltonian is Hermitian to reduce the number of integrals computed, and then we exploit the properties of localized basis functions in order to eliminate redundant work in the integral evaluation. In addition we suggest a new type of localized basis function with desirable properties. Finally we show how partitioned matrices can be used with localized basis functions to reduce the amount of work required to handle the complex boundary conditions. The new techniques do not introduce any approximations into the calculations, so they may be used to obtain converged solutions of the Schrödinger equation.

†*preprint for Ninth International Conference on Computing Methods in Applied Sciences and Engineering, Paris (organized by INRIA), 29 Jan.-2 Feb. 1990. This work was supported in part by NASA, the National Science Foundation, and the R. A. Welch Foundation.*

1. Introduction

There are several approaches to solve multidimensional quantum mechanical scattering problems. The most widely studied practical methods in physical chemistry are based on writing the solution of the time-independent Schrödinger equation with nonhomogeneous boundary conditions as an expansion over products of unknown non-square-integrable radial relative translational functions and known square-integrable internal-orbital functions [1,2]. This leads to coupled ordinary differential equations for the radial functions, and these are usually solved by propagation along the radial scattering coordiante [3]. In atomic and chemical physics this is usually called the close coupling method. Each internal-orbital function is called a channel, and the number of coupled equations equals the number of coupled channels. In 1979 a workshop [4] was held comparing most of the available specialized techniques for solving these equations to each other. In addition they were compared to a widely used, state-of-the-art, general-purpose variable-order, variable-stepsize predictor-corrector (PC) algorithm [5]. Interestingly, adding the computer times to solve four test cases, the PC algorithm rated 15^{th} out of 15 schemes tested, with a computer time 19 times greater than the best specialized scheme. This shows the great utility of special methods and the great progress achieved in developing highly accurate specialized techniques for atomic and molecular collisions.

The four trial problems used to compare algorithms in the 1979 workshop involved 15–22 coupled channels [4]. With supercomputers and efficient vectorization [6] and storage management [7] strategies, much larger single-arrangement problems have been treated successfully, *e.g.*, a four-body problem with a realistic potential function and 1358 coupled channels has been solved [8] by the technique discussed above of propagation along a scattering coordinate, and a model problem involving the scattering of an atom by a corrugated crystal surface with 18711 channels has been treated successfully [9] by a time propagation algorithm [10].

Rearrangement collisions, however, pose special difficulties. For example, we will consider a rearrangement collision consisting of the reaction of an atom A with a diatom BC: A scatters onto BC, a rearrangement (chemical reaction) occurs, and AB scatters off C, or AC scatters off B; the solution also contains terms and boundary conditions corresponding to A scattering nonreactively off BC. To define a single propagation coordinate, we must either introduce special coordinates that complicate the differential operators and the boundary conditions [11–19], or one must introduce nonlocal potential operators that convert the differential equations into integrodifferential equations [20–22]. Non-propagative methods for rearrangement scattering have been developed to avoid these problems, especially for electron

scattering (*e.g.*, electron-helium scattering in which the incident electron may exchange with either bound electron of helium). Because of the essentially infinite ratio of nuclear and electronic masses and the simplicity of the coulomb potential, the coordinates and nonlocal potential operators both become particularly simple for electron scattering. A variety of basis set techniques—*nonorthogonal* spectral methods—have been developed and successfully applied to such problems [23–29].

Chemical reactions are harder to treat than electron scattering not only because the coordinate transformations and potential functions are more complicated (potential functions for chemical reactions typically involve hundreds of lines of code as contrasted to the simple coulomb potential that completely suffices for electron-atom scattering in nonrelativistic problems), but also because the de Broglie wavelengths are smaller, *i.e.*, there is more stucture in the solutions. In the last few years we have developed a variational nonorthogonal spectral method for chemical reactions [30,31], and we have applied it to obtain converged solutions to the time-independent Schrödinger equation with rearrangement scattering boundary conditions and up to 844 coupled channels [32]. Our method is based on a generalization of a variational principle due originally to Newton [33,34], and it is based on expanding the reactive amplitude density in a nonorthogonal basis set. Other basis set variational methods [35–41] based on the Kohn variational principle [42] and sharing many attractive features in common with our approach have also been proposed recently. All these approaches, as well as new nonvariational basis set techniques [43,44] are very encouraging for improving the computational efficiency of nonorthogonal spectral methods for the quantum dynamics of reactive scattering. The present paper is concerned only with the approach based on the generalized Newton variational principle (GNVP). The theoretical formulation of the method [31] and some computational improvements [42] are presented elsewhere. The present paper, after a brief overview of the working equations, describes three additional computational improvements, including improved use of inherent symmetries and a new more localized basis set.

The final variational equations in the method described here may be obtained in several ways. They were originally derived by applying the GNVP to the problem posed as a set of coupled Fredholm integral equations of the second kind [10,31,45]. (They can also be obtained from the formulation of the problem as coupled integrodifferential equations [20,21,46] or with a scattered wave or outgoing wave variational principle [47–50] based directly on the Schroödinger equation with nonhomogeneous boundary conditions.) There are many general techniques for solving Fredholm integral equations of the second kind in a single variable [51]. Just as the specialized techniques developed in physical chemistry for single-arrangement scattering are much more efficient than general-purpose predictor-corrector algo-

rithms for coupled ordinary differential equations, we believe that the specialized techniques developed for solving the coupled equations describing reactive scattering are also more efficient than general techniques developed previously for solving coupled integral equations.

Spectral techniques are now widely employed for solutions of problems in fluid dynamics [52–55]. The choice of basis functions in these applications is very critical, and the same is true for treating reactive scattering. One possibility is to choose basis functions to minimize the number of nonzero weights in the quadratures leading to a given matrix element; this kind of consideration leads to using, *e.g.*, Lobatto functions [40]. Another possibility is to choose the basis functions to allow more efficient quadratures by using fast Fourier transforms or fast cosine transforms; the latter can be accomplished, *e.g.*, by using Chebyshev basis functions [55]. A third possibility is to use basis functions that reduce the number of integrals and the time to solve the final coupled linear equations; this has been our philosophy so far. In the present work we discuss a new consideration, namely a choice of basis function that minimizes the calculational effort in obtaining the half-integrated Green's functions that enter the GNVP and in the integrations over these functions.

2. Theory

First we outline the formalism of the basic calculations, and then the improvements are described in detail.

2.1. General Equations

We consider atom-diatom reactive scattering with three arrangements: $\alpha = 1$ denoting $A + BC$, $\alpha = 2$ denoting $B + AC$, and $\alpha = 3$ denoting $C + AB$. The formalism and notation are the same as used previously [31,42]. The Schrödinger equation is

$$(H - E)\Psi^{n_o} = 0 \tag{1}$$

where H is the Hamiltonian, E is the total energy, and Ψ^{n_o} is the wave function with complex nonhomogeneous boundary conditions corresponding to an incoming wave in channel n_o and outgoing waves in the other channels. The wave function determines the scattering matrix, a complex matrix of scattering amplitudes from which all physical observables of the scattering process may be calculated. Although the boundary conditions on the final solution are complex, to correspond to the physical conditions, we form the solution to the problem in such a way that most of the computations involve real quantities. The channel label n is a collective index denoting arrangement α and internal quantum numbers specifying a channel in that arrangement. The initial channel and initial arrangement (or—sometimes

below—any special channel and its corresponding arrangement) are denoted n_o and α_o, respectively. We use conservation of total angular momentum J, parity, and arrangement symmetry, if present, to block diagonalize the problem, and all further considerations refer to a single block of N coupled channels.

Equation (1) is rewritten in three different ways ($\alpha = 1, 2, \ldots, 3$) as

$$(H_\alpha^D + V_\alpha^C - E)\Psi^{n_o} = 0 \tag{2}$$

where H_α^D is a called the distortion Hamiltonian. It contains the kinetic energy and a part of the potential that only couples channels in the same arrangement (with subblocks called distortion blocks), and V_α^C contains the rest of the potential. First we define the regular solutions $\psi^{\alpha_o n_o}$ of

$$(H_\alpha^D - E)\psi^{n_o} = 0 \tag{3}$$

for various possible initial channels n_o and the principal value Green's functions defined by

$$G_\alpha^D = \mathcal{P}(E - H_\alpha^D) \tag{4}$$

for all three arrangements. These are called the distorted waves and the distorted wave Green's functions. Then we apply the GNVP to solve for the remaining coupling due to the V_α^C potentials. This is accomplished by expanding the reactive amplitude density [31,45] in a square-integrable ($\mathcal{L}^2$) basis of functions Φ_β, with $\beta = 1, 2, ..., M$. Each basis function is a product of a radial translational function $t_{m_\beta n_\beta}^{\alpha_\beta}(R_{\alpha_\beta})$, where R_{α_β} is the radial translational coordinate in arrangement α_β, and an internal-orbital function $\phi_{n_\beta}^{\alpha_\beta}$ corresponding to channel n_β in this arrangement. The GNVP tnen leads to a matrix equation for the scattering matrix in which the matrix elements are integrals over V_α^C, G_α^D, and $G_\alpha^D V_{\alpha'}^C G_{\alpha'}^D$ sandwiched between the various ψ^n and Φ_β.

An important computational aspect of the resulting equations is that every Green's function always appears in an integrand multiplied by a basis function in the same arrangement. Thus we never compute the Green's functions themselves, but only a set of integrals over these functions. These integrals are called half-integrated Green's functions (HIGFs). The radial distorted waves and radial HIGFs are computed by a finite difference boundary value method (FDBVM) [45,56], using an irregular mesh that contains Gauss-Legendre quadrature nodes for all further quadratures of these quantities as a sub-mesh [31]. This avoids the difficulties [57,58] of integrating over functions with discontinuous slopes.

Then the scattering matrix is given by [42]

$$S_{nn_o} = \delta_{\alpha_n \alpha_o}\, {}^oS_{nn_o} + \mathcal{S}_{nn_o}, \tag{5}$$

where $^{o}\mathbf{S}$ is the scattering matrix for the distortion potential, $\underline{\underline{\mathcal{S}}}$ is the correction produced by the remainder of the potential, and α_n is the arrangement associated with channel n. (We ignore the notational complications due to the presence of closed channels, for these details play no important role in the subsequent discussion.) The correction to the scattering matrix due to the coupling potential is given by

$$\underline{\underline{\mathcal{S}}} = \underline{\underline{\mathcal{S}}}^{B} + \tilde{\mathbf{B}}^{T}\tilde{\mathbf{C}}^{-1}\tilde{\mathbf{B}}, \tag{6}$$

where the matrices in Eq. (6) are given by

$$\underline{\underline{\mathcal{S}}}^{B} = \mathbf{A}^{T}\underline{\underline{\mathcal{K}}}^{B}\mathbf{A}, \tag{7}$$

$$\tilde{\mathbf{B}} = (\mathbf{B} + \mathbf{X}^{T}\underline{\underline{\mathcal{K}}}^{B})\mathbf{A}, \tag{8}$$

and

$$\tilde{\mathbf{C}} = \mathbf{C} - \mathbf{B}\mathbf{X} - \mathbf{X}^{T}\underline{\underline{\tilde{\mathcal{B}}}} + \mathbf{X}^{T}\underline{\underline{\mathcal{D}}} - \mathbf{X}^{T}\underline{\underline{\mathcal{K}}}^{B}\mathbf{X}. \tag{9}$$

$\mathbf{A}$ is the transformation which takes the regular distorted waves from real standing wave boundary conditions to complex scattering matrix boundary conditions, $\mathbf{X}$ is the transformation required to form the outgoing wave HIGF given the real function, and

$$\mathcal{K}^{B}_{nn_o} = \int dR_{\alpha_o} \sum_{n'} \Delta^{\alpha_o}_{n'n_o} \mathcal{F}_{nn'}(R_{\alpha_o})\, {}^{(r)}f^{\alpha_o}_{n'n_o}(R_{\alpha_o}), \tag{10}$$

$$B_{\beta n_o} = \int dR_{\alpha_o} \sum_{n'} \Delta^{\alpha_o}_{n'n_o} \mathcal{G}_{\beta n'}(R_{\alpha_o})\, {}^{(r)}f^{\alpha_o}_{n'n_o}(R_{\alpha_o}), \tag{11}$$

$$\mathcal{D}_{n\beta_o} = \int dR_{\alpha_o} \tilde{\mathcal{T}}_{nn\beta_o}(R_{\alpha_o}) t^{\alpha_o}_{m\beta_o n\beta_o}(R_{\alpha_o}), \tag{12}$$

$$\tilde{\mathcal{B}}_{n\beta_o} = \int dR_{\alpha_o} \sum_{n'} \Delta^{\alpha_o}_{n'n} \mathcal{F}_{nn'}(R_{\alpha_o})\, \dot{g}^{N}_{n'\beta_o}(R_{\alpha_o}), \tag{13}$$

and

$$\begin{aligned} C_{\beta\beta_o} = &\int dR_{\alpha_o} \mathcal{T}_{\beta n\beta_o}(R_{\alpha_o}) t^{\alpha_o}_{m\beta_o n\beta_o}(R_{\alpha_o}) \\ &- \int dR_{\alpha_o} \sum_{n'} \Delta^{\alpha_o}_{n'n\beta_o} \mathcal{G}_{\beta n'}(R_{\alpha_o}) \dot{g}^{N}_{n'\beta_o}(R_{\alpha_o}). \end{aligned} \tag{14}$$

A three-body system involves three internal coordinates, and these equations involve the final (third) integration of the quadrature over these coordinates. The matrix element $\Delta^{\alpha_o}_{nn_o}$ is one if channel n and channel n_o are both members of the same distortion block of arrangement α_o and are zero otherwise, ${}^{(r)}f^{\alpha_o}_{nn_o}$ is the regular radial function for the distortion potential defined by n and n_o and satisfying real

boundary conditions (*i.e.*, it is one of the radial relative translational functions obtained by solving a set of close coupling equations for ψ^{n_o}), β_o is a specific basis function associated with arrangement α_o, and $\grave{g}^N_{n\beta}$ is the radial part of the real HIGF associated with the distortion block containing n and basis function β. The HIGF can be expressed as [31]

$$\grave{g}^N_{n\beta}(R_\alpha) = \int dR'_\alpha \; g_{nn_\beta}(R_\alpha, R'_\alpha)\; t^\alpha_{m_\beta n_\beta}(R'_\alpha), \tag{15}$$

where n and β are both associated with the same arrangement α, and the radial Green's function is defined as

$$g_{nn'}(R_\alpha, R'_\alpha) = \sum_{n''} \Delta^\alpha_{nn''}\Delta^\alpha_{n'n''} \begin{cases} {}^{(r)}f^\alpha_{nn''}(R_\alpha)\,{}^{(i)}f^\alpha_{n'n''}(R'_\alpha) & R_\alpha < R'_\alpha \\ {}^{(i)}f^\alpha_{nn''}(R_\alpha)\,{}^{(r)}f^\alpha_{n'n''}(R'_\alpha) & R_\alpha > R'_\alpha \end{cases}, \tag{16}$$

where ${}^{(i)}f^\alpha_{nn''}$ is the irregular analog of ${}^{(r)}f^\alpha_{nn''}$. In addition the integrals $\mathcal{F}_{nn_o}$, $\mathcal{G}_{\beta n_o}$, $\tilde{\mathcal{T}}_{nn_o}$, and $\mathcal{T}_{\beta n_o}$ are given by the expressions

$$\mathcal{F}_{nn_o} = \begin{cases} \sum_{n'} \Delta^{\alpha_o}_{nn'}\,{}^{(r)}f^{\alpha_o}_{n'n}(R_{\alpha_o})e_{n'n_o}(R_{\alpha_o}), & \alpha = \alpha_o; \\ \int dR_\alpha \sum_{n'} \Delta^\alpha_{nn'}\,{}^{(r)}f^\alpha_{n'n}(R_\alpha)W^{\alpha\alpha_o}_{n'n_o}(R_\alpha, R_{\alpha_o}), & \text{otherwise}, \end{cases} \tag{17}$$

$$\mathcal{G}_{\beta n_o} = \begin{cases} \sum_{n'} \Delta^{\alpha_o}_{n_\beta n'}\grave{g}^N_{n'\beta}(R_{\alpha_o})e_{n'n_o}(R_{\alpha_o}), & \alpha = \alpha_o; \\ \int dR_\alpha \sum_{n'} \Delta^\alpha_{n_\beta n'}\grave{g}^N_{n'\beta}(R_\alpha)W^{\alpha\alpha_o}_{n'n_o}(R_\alpha, R_{\alpha_o}), & \text{otherwise}, \end{cases} \tag{18}$$

$$\tilde{\mathcal{T}}_{nn_o} = \begin{cases} \Delta^{\alpha_o}_{nn_o}\,{}^{(r)}f^{\alpha_o}_{n_o n}(R_{\alpha_o}), & \alpha = \alpha_o; \\ \int dR_\alpha \sum_{n'} \Delta^\alpha_{nn'}\,{}^{(r)}f^\alpha_{n'n}(R_\alpha)\mathcal{B}^{\alpha\alpha_o}_{n'n_o}(R_\alpha, R_{\alpha_o}), & \text{otherwise}, \end{cases} \tag{19}$$

and

$$\mathcal{T}_{\beta n_o} = \begin{cases} \Delta^{\alpha_o}_{n_\beta n_o}\grave{g}^N_{n_o\beta}(R_{\alpha_o}), & \alpha = \alpha_o; \\ \int dR_\alpha \sum_{n'} \Delta^\alpha_{n_\beta n'}\grave{g}^N_{n'\beta}(R_\alpha)\mathcal{B}^{\alpha\alpha_o}_{n'n_o}(R_\alpha, R_{\alpha_o}), & \text{otherwise}. \end{cases} \tag{20}$$

The integrals (17–20) contain the inner two quadratures of the three-dimensional integration mentioned above. For integrations over functions defined in two different arrangements, say α and α_o, the quadratures are carried out in a coordinate system consisting of R_α, R_{α_o}, and the angle $\Delta_{\alpha\alpha_o}$ between the vector from the atom to the diatom in arrangement α and the analogous vector in α_o. The inner loop involves integration over $\Delta_{\alpha\alpha_o}$ and yields $\mathcal{B}^{\alpha\alpha_o}_{n'n_o}$ or $\mathcal{C}^{\alpha\alpha_o}_{n'n_o}$, from which we calculate $W^{\alpha\alpha_o}_{n'n_o}$ [31]. The various middle loops are given in Eqs. (17–20), and the various outer loops are indicated in Eqs. (10–14). Similarly $e_{n'n_o}(R_{\alpha_o})$ is defined as a two-dimensional integral over internal coordinates orthogonal to R_{α_o} when all functions in the integrand are associated with the same arrangement α_o. Further details of

the quantities in Eqs. (1–20) are not necessary for the discussion in this paper—see Refs. [31,42] for full details.

2.2. Hermitian Properties of the Hamiltonian

Let us consider the integrals given by Eqs. (17–20) for a triatomic system where all arrangements are different. Now the matrices $\underline{\underline{\mathcal{K}}}^B$ and $\mathbf{C}$ are symmetric while $\mathbf{B}$, $\underline{\underline{\tilde{\mathcal{B}}}}$, and $\underline{\underline{\mathcal{D}}}$ are rectangular. For the symmetric matrices it is necessary to just compute the lower triangle, *i.e.*, we can restrict $\alpha \leq \alpha_o$; thus only 3 of the 6 possible reactive arrangement pairs are required. However at first glance it may seem that for the rectangular matrices it will be necessary to use all 6 of the reactive arrangement pairs, but we now show that this is not so. That is, the rectangular matrices can also be assembled from only the 3 reactive arrangement pairs required for the symmetric matrices.

To derive the relation we require, it is useful to express the matrix elements of the rectangular matrices in Dirac notation [31]:

$$B_{\beta n_o} = < \Phi_\beta | G_\alpha^D \mathcal{U}_{\alpha\alpha_o} | \psi^{\alpha_o n_o} > = < \Phi_\beta | G_\alpha^D \mathcal{U}^{\alpha_o} | \psi^{\alpha_o n_o} >, \tag{21}$$

$$\tilde{\mathcal{B}}_{n\beta_o} = < \psi^{\alpha n} | \mathcal{U}^{\alpha_o} G_{\alpha_o}^D | \Phi_{\beta_o} >, \tag{22}$$

and

$$\mathcal{D}_{n\beta_o} = < \psi^{\alpha n} | \mathcal{U}^{\alpha_o} | \Phi_{\beta_o} >, \tag{23}$$

where

$$\mathcal{U}_{\alpha\alpha_o} = -(2\mu/\hbar^2)[H - E + \delta_{\alpha\alpha_o}(E - H_\alpha^D)], \tag{24}$$

$$\mathcal{U}^{\alpha_o} = -(2\mu/\hbar^2)[H - H_{\alpha_o}^D], \tag{25}$$

and μ is the reduced mass. Now for inter-arrangement integrals, $\mathcal{U}_{\alpha\alpha_o}$ is just $-(2\mu/\hbar^2)(H - E)$, which is Hermitian; thus since $B_{\beta n_o}$ is real it can be written as

$$B_{\beta n_o} = < \psi^{\alpha_o n_o} | \mathcal{U}_{\alpha_o\alpha} G_\alpha^D | \Phi_\beta > . \tag{26}$$

Now we also know that [31]

$$\mathcal{U}_{\alpha_o\alpha} G_\alpha^D = \mathcal{U}^\alpha G_\alpha^D - 1 + \delta_{\alpha_o\alpha}, \tag{27}$$

thus for inter-arrangement matrix elements,

$$B_{\beta n_o} = < \psi^{\alpha_o n_o} | \mathcal{U}^\alpha G_\alpha^D | \Phi_\beta > - < \psi^{\alpha_o n_o} | \mathcal{U}^\alpha | \Phi_\beta >, \tag{28}$$

or

$$B_{\beta n_o} = \tilde{\mathcal{B}}_{n_o\beta} - \mathcal{D}_{n_o\beta}. \tag{29}$$

This has two consequences. First of all Eq. (9) can be rewritten as

$$\tilde{\mathbf{C}} = \mathbf{C} - \mathbf{B}\mathbf{X} - \mathbf{X}^T\mathbf{B}^T + \mathbf{X}^T\underline{\underline{\tilde{\mathcal{D}}}} - \mathbf{X}^T\underline{\underline{\mathcal{K}}}^B\mathbf{X}, \tag{30}$$

where $\underline{\underline{\tilde{\mathcal{D}}}}$ is zero unless an intra-arrangement integral is involved, in which case it is equal to $\underline{\underline{\mathcal{D}}}$ (see also Ref. [59]). Thus there is no need to compute and store the additional matrices $\underline{\underline{\tilde{\mathcal{B}}}}$ and $\underline{\underline{\mathcal{D}}}$. This means that there are no integrals required for the complex scattering matrix boundary conditions which are not also needed for calculations which employ real reactance matrix boundary conditions. However this is not the only advantage. If in calculating the integrals given by Eqs. (17–20), the restriction $\alpha < \alpha_o$ is made, then there is enough information to calculate $B_{\beta n_o}$ for $\alpha < \alpha_o$ and the matrix elements $\tilde{\mathcal{B}}_{n\beta_o}$ and $\mathcal{D}_{n\beta_o}$ for $\alpha < \alpha_o$. Then by using Eq. (29), one can obtain $B_{\beta n_o}$ for $\alpha > \alpha_o$.

Thus it is not necessary to evaluate the integrals (17–20) for $\alpha > \alpha_o$. This is an important simplification because the calculation of these integrals of Eqs. (11–14) usually requires a substantial fraction of the whole time taken up by the integral calculation. Thus for systems having no symmetry, this amounts to almost a factor of 2 decrease in the work involved, since only 3 out of the 6 possible reactive arrangement pairs are required. For systems having identical atoms this factor is smaller: For systems of the type $A + B_2$, *i.e.*, where B is the same kind of atom as C, it is necessary only to perform 2 of the possible 3 unique reactive arrangement pairs, while for $A + A_2$ collisions, no savings above those already achieved by using arrangement symmetry [42] are possible from this technique.

2.3. Localized Basis Functions

In this section we show how the calculations simplify if $t^{\alpha}_{mn}(R_\alpha)$ is a localized function. Although it is convenient numerically to bypass the calculation of the irregular function in Eq. (16) and solve for the HIGFs directly, it is necessary to consider the expressions (15–16) for the HIGF in terms of the regular and irregular distorted waves and the basis functions in order to see the effect of localization.

In particular suppose that the basis function $t^{\alpha}_{m_\beta n_\beta}$ is zero for $R'_\alpha < R^S_\beta$ and $R'_\alpha > R^L_\beta$. Then the limits on the integral in Eq. (15) reduce from the original 0 to ∞ to the computationally more attractive R^S_β to R^L_β. Thus if $R_\alpha < R^S_\beta$, then in the integral of Eq. (15) we will have $R_\alpha < R'_\alpha$ for all R'_α, and the HIGF will be equal to a linear combination of the columns of the regular function. Similarly if $R_\alpha > R^L_\beta$, the HIGF will be equal to a linear combination of the columns of the irregular function. Now in practice we wish to avoid calculating the irregular function; however this is not difficult since we only need to know this function at large distances, and for large distances one of the HIGFs will do just as well. We will order the basis functions so that $\beta = 1, ..., N$ correspond to the basis functions

which have the smallest values of R_β^L for each n_β. Note: N, the number of channels, is less than M, the number of basis functions. Thus we can write

$$\dot{y}^N_{n\beta_o}(R_\alpha) = \begin{cases} \sum_{n'} {}^{(r)}f^\alpha_{nn'}(R_\alpha) d^{\alpha S}_{n'\beta_o} \Delta^\alpha_{n'n_{\beta_o}} & R_\alpha < R^S_{\beta_o} \\ \sum_{n'} \dot{y}^N_{nn'}(R_\alpha) d^{\alpha L}_{n'\beta_o} \Delta^\alpha_{n'n_{\beta_o}} & R_\alpha > R^L_{\beta_o} \\ \dot{y}^N_{n\beta_o}(R_\alpha) & \text{otherwise,} \end{cases} \tag{31}$$

where $d^{\alpha S}_{n'\beta_o}$ and $d^{\alpha L}_{n'\beta_o}$ are proportionality constants, $\dot{y}^N_{nn'}$ is the HIGF having the smallest value for R^L_β, and $\dot{y}^N_{n\beta_o}$ is the numerically generated function. The proportionality constants for small R_α can be evaluated via

$$d^{\alpha S}_{n\beta} = \Delta^\alpha_{n_\beta n} \int dR_\alpha \, {}^{(i)}f^\alpha_{n_\beta n}(R_\alpha) t^\alpha_{m_\beta n_\beta}(R_\alpha) \tag{32}$$

if the irregular function is known, or more practically by forming the ratio of the numerically determined HIGF and the regular distorted wave. We form the average of the radial functions over several distances immediately prior to R^S_β and then form the ratio to obtain $d^{\alpha S}_{n\beta}$. One difficulty with this procedure arises when distortion potential blocks contain both open and closed channels. In this case, some of the columns of ${}^{(r)}f^\alpha$ are so small near the origin that it is not possible to obtain an accurate inverse; then the procedure fails. This problem can be avoided by decoupling the open and closed channels.

The proportionality constant for large R_α can be determined by a numerical ratio in a similar manner to the procedure for $d^{\alpha S}_{n\beta}$ or by solving

$$d^\alpha_{\beta n} = \sum_{n'} d^\alpha_{n'n} d^{\alpha L}_{n\beta}, \tag{33}$$

where

$$d^\alpha_{\beta n} = \Delta^\alpha_{n_\beta n} \int dR_\alpha \, {}^{(r)}f^\alpha_{n_\beta n}(R_\alpha) t^\alpha_{m_\beta n_\beta}(R_\alpha), \tag{34}$$

and $d^\alpha_{n'n}$ corresponds to the integral for the basis function having the smallest value for R^L_β. The integrals in Eq. (34) are also required for the large-R_α boundary conditions for the HIGFs [31].

Equation (31) has several consequences. First of all, it is clear that once ${}^{(r)}f^\alpha_{n_\beta n}$ and $\dot{y}^N_{nn_o}$ are known, it is only necessary to calculate and store the remaining HIGFs in the ranges R^S_β to R^L_β. In practice since we determine the $\dot{y}^N_{n\beta_o}$ by solving an inhomogeneous form of the finite difference boundary value method [31,42] (FDBVM), we can reduce the finite difference grid used for the regular function and the first HIGF, which goes from $R^F_{\alpha,1}$ to $R^F_{\alpha,N(F)}$, to a grid which goes from $R^F_{\alpha,1}$ to just beyond the maximum value of R^L_β. Because $R^F_{\alpha,N(F)}$ is determined by the distance

where the potential becomes negligible, whereas the maximum value of R_β^L is determined by the distance where the difference between the potential and the distortion potential becomes negligible, this can result in a considerable savings. The most important consequence of Eq. (31) however, arises from applying it to the integrals in Sect. 2.1.

There are two classes of integrals to consider. First of all there are the radial integrals with one HIGF and no basis functions, which we will approximate by a quadrature sum:

$$I_{\beta n}^1 = \sum_{i=1}^{N^{RQ}} M_{\beta n}^g(i), \tag{35}$$

where

$$M_{\beta n}^g = w_i^\alpha \sum_{n'} \Delta_{n' n_\beta}^\alpha \dot{g}_{n'\beta}^N(R_{\alpha i}) M_{n' n}(R_{\alpha i}), \tag{36}$$

w_i^α being a quadrature weight and $M_{nn'}$ some matrix function. The integrals falling in this case are $\mathcal{G}_{\beta n_o}$, $\mathcal{T}_{\beta n_o}$, $\tilde{\mathcal{B}}_{n_o\beta}$ (for which the transpose of Eq. (35) and following are used), the intra-arrangement parts of $B_{\beta n_o}$, and the inter-arrangement parts of $C_{\beta\beta_o}$. It is useful to define a quantity analogous to $M_{\beta n'}^g$ called $M_{nn'}^f$ which differs by replacing the HIGF with the regular function. Then applying Eq. (31), the integral becomes

$$\begin{aligned} I_{\beta n}^1 = &\sum_{\hat{n}} \Delta_{\hat{n} n_\beta}^\alpha d_{\hat{n}\beta}^{\alpha S} [\sum_{i=1}^{i_\beta^S} M_{\hat{n} n}^f(i)] + \sum_{i=i_\beta^S+1}^{i_\beta^L} M_{\beta n}^g(i) \\ &+ \sum_{\hat{n}} \Delta_{\hat{n} n_\beta}^\alpha d_{\hat{n}\beta}^{\alpha L} [\sum_{i=i_\beta^L+1}^{N^{RQ}} M_{\hat{n} n}^g(i)] \end{aligned}, \tag{37}$$

where $R_{\alpha i_\beta^S}$ is the largest quadrature point less than R_β^S and $R_{\alpha i_\beta^L}$ is the largest quadrature point less than R_β^L. (For the HIGF with the smallest R_β^L, we set i_β^L equal to N^{RQ}.) It should be noted that the quantities in the first sum must be calculated anyway—for the list of integrals mentioned after Eq. (36), these correspond to $\mathcal{F}_{nn_o}$, $\tilde{\mathcal{T}}_{nn_o}$, $\mathcal{K}_{n_o n}^B$, $\mathcal{K}_{nn_o}^B$, and $B_{\beta n_o}$, respectively. Thus we see that the radial quadrature points fall into three regions: those less than all i_β^S where only quantities involving the regular function need be accumulated, those greater than all R_β^L where only quantities involving the regular function and the HIGFs with the smallest value of i_β^L need to be accumulated, and points in the intermediate region where the sums will include some number of HIGFs less than the full set.

Now consider the case when there are two HIGFs. Here the integral can be written

$$I^2_{\beta\beta_o} = \sum_{i=1}^{N^{RQ}} M^{gg}_{\beta\beta_o}(i), \tag{38}$$

where

$$M^{gg}_{\beta\beta_o} = w_i^\alpha \sum_{n'n''} \Delta^\alpha_{n'n_\beta} \Delta^\alpha_{n''n_{\beta_o}} \dot{g}^N_{n'\beta}(R_{\alpha i}) M_{n'n''}(R_{\alpha i}) \dot{g}^N_{n''\beta_o}(R_{\alpha i}). \tag{39}$$

The integrals falling in this case are the intra-arrangement parts of **C**. The presence of two HIGFs greatly complicates this case compared to Eq. (37). There are now six cases to consider: nonoverlapping basis functions, one basis function contained within the other, and other overlapping basis functions, with the bra or ket starting first. However it can be shown that the result of using Eq. (31) is

$$\begin{aligned} I^2_{\beta\beta_o} = &\begin{cases} \sum_{\dot{n}} \Delta^\alpha_{\dot{n}n_{\beta_o}} I^1_{\beta\dot{n}}(i^S_{\beta_o}) d^{\alpha S}_{\dot{n}\beta_o} & i^S_{\beta_o} > i^S_\beta \\ \sum_{\dot{n}} \Delta^\alpha_{\dot{n}n_\beta} d^{\alpha S}_{\dot{n}\beta} I^1_{\beta_o\dot{n}}(i^S_\beta) & i^S_\beta > i^S_{\beta_o} \end{cases} + \sum_{i=max(i^S_\beta, i^S_{\beta_o})}^{min(i^S_\beta, i^S_{\beta_o})} M^{gg}_{\beta\beta_o}(i) \\ &+ \begin{cases} \sum_{\dot{n}} \Delta^\alpha_{\dot{n}n_{\beta_o}} [I^2_{\beta\dot{n}}(N^{RQ}) - I^2_{\beta\dot{n}}(max(i^L_{\beta_o}, i^S_\beta, i^S_{\beta_o})] d^{\alpha L}_{\dot{n}\beta_o} & i^L_\beta > i^L_{\beta_o} \\ \sum_{\dot{n}} \Delta^\alpha_{\dot{n}n_\beta} d^{\alpha L}_{\dot{n}\beta} [I^2_{\dot{n}\beta}(N^{RQ}) - I^2_{\dot{n}\beta}(max(i^L_\beta, i^S_\beta, i^S_{\beta_o}))] & i^L_{\beta_o} > i^L_\beta \end{cases} \end{aligned} \tag{40}$$

In this equation, $I^1_{\beta n}$ corresponds to B, and an argument of i is interpreted as the sum up to the i^{th} quadrature point. It should be noted that for non-overlapping basis functions, the middle sum will have no contribution. In addition, this formula applies as well to matrix elements consisting of one HIGF and one translational basis function. In this case, $d^{\alpha S}_{n\beta}$ and $d^{\alpha L}_{n\beta}$ for the translational basis function are zero and $I^1_{\beta n}$ corresponds to $\underline{\underline{\tilde{D}}}$. Since the number of terms in the middle sum is much less than the number of radial quadrature points, N^{RQ}, using Eq. (40) greatly reduces the work required for evaluating the integrals. It should be noted that care needs be taken to avoid excessive round off error when evaluating the last subtraction in Eq. (40). Optimally one would separately store the contributions from the various intervals between $min(i^S_\beta, i^S_{\beta_o})$ and $max(i^L_{\beta_o}, i^S_\beta, i^S_{\beta_o})$ or $max(i^L_\beta, i^S_\beta, i^S_{\beta_o})$, since in this way no explicit subtraction is required.

Another way to exploit Eq. (31) is to take linear combinations of the HIGFs and use these as basis functions. Consider the HIGF labeled by β. We can form the combination

$$\dot{g}^{\mathcal{L}}_{n\beta}(R_\alpha) = g^N_{n\beta}(R_\alpha) - \sum_{n'} \dot{g}^N_{nn'}(R_\alpha) d^{\alpha L}_{n'\beta}, \tag{41}$$

in which case the analog of Eq. (31) for $\grave{g}^{\mathcal{L}}_{n\beta}$ becomes

$$\grave{g}^{\mathcal{L}}_{n\beta}(R_\alpha) = \begin{cases} \sum_{n'} {}^{(r)}f^{\alpha}_{nn'}(R_\alpha) d^{\alpha \mathcal{S}\mathcal{L}}_{n'\beta} \Delta^{\alpha}_{n'n_\beta} & R_\alpha < R^{\mathcal{S}} \\ 0 & R_\alpha > R^{L}_{\beta} \\ \grave{g}^{\mathcal{L}}_{n\beta}(R_\alpha) & \text{otherwise,} \end{cases} \tag{42}$$

where $R^{\mathcal{S}}$ is the smallest value of $R^{S}_{n'}$, and $d^{\alpha \mathcal{S}\mathcal{L}}_{n'\beta_o}$ is the new small R_α proportionality constant. There are two advantages of this formulation. First of all, when using the $\grave{g}^{\mathcal{L}}_{n\beta}$, Eqs. (37) and (40) simplify, because now only the first two terms are present. This is especially important for Eq. (40), because one now avoids the final subtraction, which is complicated to implement in a manner which avoids roundoff error. The second advantage arises when transforming to complex boundary conditions. This will be discussed in the next section.

One disadvantage of using the $\grave{g}^{\mathcal{L}}_{n\beta}$ is that while previously the first sum in Eqs. (37) and (40) went from 1 to i^{S}_{β} or $min(i^{S}_{\beta}, i^{S}_{\beta_o})$, it now goes from 1 to $i^{\mathcal{S}}$, where $R_{\alpha i^{\mathcal{S}}}$ is the largest quadrature point less than $R^{\mathcal{S}}$. Since $i^{\mathcal{S}}$ will be smaller than i^{S}_{β} or $min(i^{S}_{\beta}, i^{S}_{\beta_o})$, the overall efficiency is not as great. However, one can diminish this effect by not using in Eq. (41) the $\grave{g}^{N}_{nn'}$ which are the HIGFs with the smallest values for large-R_α zero limit, but rather using HIGFs which have as large small-R_α zero limits as possible subject to the constraint their large-R_α zero limit is not larger than R^{L}_{β}. This will maximize $i^{\mathcal{S}}$.

We now consider the effect on the matrix elements when the $\grave{g}^{\mathcal{L}}_{n\beta}$ are substituted for the $\grave{g}^{N}_{n\beta}$ in Eqs. (10–14) and (17–20). Since Eq. (41) can be written as

$$\grave{\mathbf{g}}^{\mathcal{L}} = \grave{\mathbf{g}}^{N}\mathbf{L}, \tag{43}$$

where the matrix elements of $\mathbf{L}$ are the constant factors in Eq. (41), we see that provided we combine the $t^{\alpha}_{m_\beta n_\beta}$ in Eq. (14) using the same rule as was used to produce the $\grave{g}^{\mathcal{L}}_{n\beta}$, then we obtain

$$\mathbf{B}^{\mathcal{L}} = \mathbf{L}^{T}\mathbf{B} \tag{44}$$

and

$$\mathbf{C}^{\mathcal{L}} = \mathbf{L}^{T}\mathbf{C}\mathbf{L} \tag{45}$$

when using the $\grave{g}^{\mathcal{L}}_{n\beta}$. Provided that $\mathbf{L}^{-1}$ exists, it is easy to see that

$$\mathbf{B}^{\mathcal{L}^T}\mathbf{C}^{\mathcal{L}^{-1}}\mathbf{B}^{\mathcal{L}} = \mathbf{B}^{T}\mathbf{C}^{-1}\mathbf{B}, \tag{46}$$

thus it is not necessary to transform back to the $\grave{g}^{N}_{n\beta}$. In order to ensure that $\mathbf{L}^{-1}$ exists, it is necessary to retain at least one $\grave{g}^{N}_{n\beta}$ per channel.

Now consider the transformation from real to complex boundary conditions, Eqs. (8), (9), and (30). Since the $\grave{y}^{\mathcal{L}}_{n\beta}$ are strictly zero beyond the distance where the associated localized basis functions are zero, they will be independent of the boundary conditions. That is, the subblock of $\mathbf{X}$ associated with $\grave{y}^{\mathcal{L}}_{n\beta}$ will be the unit matrix, the subblock of $\underline{\underline{\tilde{D}}}$ will be the zero matrix, and the portions of $\tilde{\mathbf{C}}^{\mathcal{L}}$ which are associated with two of the $\grave{y}^{\mathcal{L}}_{n\beta}$ will be real and the same as $\mathbf{C}^{\mathcal{L}}$. In the next section, we will show how this can be exploited to save work in evaluating Eq. (6).

Thus we see that substantial work can be saved in evaluating the integrals when localized basis functions are used. We now consider the choice of such functions. In our previous work using variational methods, we have used distributed gaussian functions [30–32,42,59–74] or sine-type functions [31] as a basis. Strictly speaking, the gaussian functions are not local, since they are zero only asymptotically; however for practical purposes they differ significantly from zero only in a narrow region. Thus one procedure to use would be to set the gaussian to zero whenever it fell below some fraction of its maximum, say 10^{-14}. However this procedure introduces discontinuities into the integrands which can cause slow convergence of the numerical integrals. Thus we seek a localized function which is continuous and has continuous derivatives. The function we will use is inspired by the cutoff function of Ref. [75], and is given by

$$t = \begin{cases} \exp[-\frac{a}{1-(x/b)^2}] & |x| < b \\ 0 & |x| \geq b \end{cases} \tag{47}$$

We call this function a cut off gaussian" (COG). It has the property that for small x/b, it behaves like $\exp[-a(x/b)^2]$, so can be made similar to our previous basis functions, yet it is localized within b of its center. It should be noted that as $a \rightarrow \infty$ with fixed a/b^2, the COG becomes a gaussian.

2.4. Partitioned Matrices

We will partition the matrices into blocks consisting either of functions which are localized (the $\grave{y}^{\mathcal{L}}_{n\beta}$) or those which are delocalized. Thus we write

$$\mathbf{B}^{\mathcal{L}} = \begin{pmatrix} \mathbf{B}_c \\ \mathbf{B}_{\mathcal{L}} \end{pmatrix}, \tag{48}$$

and

$$\mathbf{C}^{\mathcal{L}} = \begin{pmatrix} \mathbf{C}_{cc}\mathbf{C}^T_{\mathcal{L}c} \\ \mathbf{C}_{\mathcal{L}c}\mathbf{C}_{\mathcal{L}\mathcal{L}} \end{pmatrix}, \tag{49}$$

where the subscript $\mathcal{L}$ means localized and c delocalized. If we solve the matrix equation

$$\underline{\underline{K}} = \underline{\underline{K}}^B + \mathbf{B}^{\mathcal{L}T}\mathbf{C}^{\mathcal{L}^{-1}}\mathbf{B}^{\mathcal{L}}, \tag{50}$$

by blocks, we obtain

$$\underline{\underline{K}} = \underline{\underline{K}}^{Bf} + \mathbf{B}^{f^T}\mathbf{C}^{f^{-1}}\mathbf{B}^f, \tag{51}$$

where the folded matrices are given by

$$\underline{\underline{K}}^{Bf} = \underline{\underline{K}}^{B} - \mathbf{B}_{\mathcal{L}}^T\mathbf{C}_{\mathcal{L}\mathcal{L}}^{-1}\mathbf{B}_{\mathcal{L}}, \tag{52}$$

$$\mathbf{B}^f = \mathbf{B}_c - \mathbf{C}_{\mathcal{L}c}^T\mathbf{C}_{\mathcal{L}\mathcal{L}}^{-1}\mathbf{B}_{\mathcal{L}}, \tag{53}$$

and

$$\mathbf{C}^f = \mathbf{C}_{cc} - \mathbf{C}_{\mathcal{L}c}^T\mathbf{C}_{\mathcal{L}\mathcal{L}}^{-1}\mathbf{C}_{\mathcal{L}c}. \tag{54}$$

Now consider solving Eq. (6) by the same procedure. The result is

$$\underline{\underline{S}} = \underline{\underline{S}}^{Bf} + \tilde{\mathbf{B}}^{f^T}\tilde{\mathbf{C}}^{f^{-1}}\tilde{\mathbf{B}}^f, \tag{55}$$

where the complex folded matrices are obtain from Eqs. (7), (8), and (30) using the real folded matrices of Eqs. (52–54). A similar procedure was employed in the context of the Kohn variational principle in Ref. [36].

Several things should be noted concerning the above procedure as it affects the GNVP calculations. First of all, if the number of localized functions is considerably larger than the number of delocalized functions; as may often be the case, the work to produce the folded matrices will be greater than the work to evaluate Eq. (55). This means that the computational effort involved in a calculation with complex boundary conditions will be very similar to what would have been required if real boundary conditions had been used. Also the memory requirements will be similar, because it will not be necessary to store the imaginary part of $\tilde{\mathbf{C}}$.

3. Applications and Directions for Future Work

The techniques presented here and previously provide an efficient method for large-scale quantum mechanical calculations of chemical reaction dynamics based on the generalized Newton variational principle. Some applications that have been made include the calculation of converged cross sections for the H + H_2 [63] and D+H_2 [32,71] reactions and converged state-selected reactive transition probabilities for these reactions [30,31,42,59,61,62,65,66.69] and for the O+H_2 [59,72] and O+HD [59,67] reactions. We have also presented converged reactive transition probabilities for the F + H_2 reaction with total angular momentum J = 0–2 [64,68,70,73,74]. We have calculated converged collisional delay times, which require very stable (numerically differentiable) solutions as a function of energy, for H+H_2 with $J = 0$, 1, and 4 [69], for D+H_2 with $J = 0$ [65], and for F+H_2 with $J = 0$ and 1 [68,70]. An

earlier nonvariational version of the method was used to obtain converged transition probabilities for the $H + H_2$ [45], $D + H_2$ [45,76], $O + H_2$ [46,77], $O + OH$ [46], and $H + HBr$ [78] reactions.

For $O + H_2$ we have obtained very well converged results with an average of as few as three gaussians per channel [72]. A recent study of basis set requirements for $F + H_2$ showed that excellent convergence can be achieved with 10 gaussians per channel in the $F + H_2$ arrangement and 18 gaussians per channel in the $H + HF$ arrangement [73]. Further efficiencies can be achieved by using better basis sets, *e.g.*, by basis set contraction [31,74,79,80] or the use of localized basis sets as described in the present paper. Another promising approach is based on the reinterpretation of the GNVP using a scattered wave variational principle [47–50]. This allows for hybrid basis sets which effectively convert some of the integrals over $G_\alpha^D U^{\alpha_o} G_{\alpha_o}^D$, as in $C_{\beta\beta_o}$, into simpler energy-independent integrals. All these approaches are being explored for further applications.

4. Summary

We have introduced new techniques to reduce the work required in applying the generalized Newton variational principle to three-dimensional reactive scattering calculations. The underlying idea behind these developments is to minimize redundant work as much as possible. This is accomplished in two ways. First of all the fact that the Hamiltonian is Hermitian is used to decrease the number of inter-arrangement integrals which must be calculated. Even for a system with no symmetry, *e.g.*, $O + HD$, this reduces by half the number of two dimensional integrals which are performed before the final integration of the three-dimensional exchange integrals. Secondly we introduce a localized translational basis set which need not differ significantly from our previous basis functions and then exploit the effect this has on the half-integrated Green's functions to reduce the amount of work required to calculate these functions, the amount of storage required to save these functions, the amount of work required for the integrals over these functions, and the work required for the final linear-equations step when complex boundary conditions are used.

References

1. A. M. Arthurs and A. Dalgarno, Proc. Roy. Soc. London A256 (1960) 540.
2. W. A. Lester, Adv. Quantum Chem. 9 (1975) 199.
3. D. Secrest, in *Atom-Molecule Collision Theory*, R. B. Bernstein, ed., Plenum,

New York, 1979, p. 265.

4. *Algorithms and Computer Codes for Molecular Quantum Scattering Theory* (2 vols.), L. Thomas, ed., Lawrence Berkeley Laboratory, Berkeley, 1979.

5. L. F. Shampine and M. K. Gordon, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, W. H. Freeman, San Francisco, 1975.

6. D. W. Schwenke and D. G. Truhlar, in *Supercomputer Applications*, R. W. Numrich, ed., Plenum, New York, 1985, p. 215.

7. D. W. Schwenke, K. Haug, D. G. Truhlar, R. Schweitzer, J. Z. H. Zhang, Y. Sun, and D. J. Kouri, Theoret. Chim. Acta 72 (1987) 237.

8. D. W. Schwenke and D. G. Truhlar, J. Chem. Phys. 88 (1988) 4800.

9. R. C. Mowrey, H. F. Bowen, and D. J. Kouri, J. Chem. Phys. 86 (1987) 2441.

10. D. J. Kouri, Y. Sun, R. C.Mowrey, J. Z. H. Zhang, D. G. Truhlar, K. Haug, and D. W. Schwenke, in *Mathematical Foundations of Computational Chemical Physics*, D. G. Truhlar, ed., Springer-Verlag, New York, 1988, p. 237.

11. R. A. Marcus, J. Chem. Phys. 49 (1968) 2610.

12. R. E. Wyatt, ACS Symp. Ser. 56 (1977) 185.

13. L. M. Delves, Nucl. Phys. 9 (1959) 391.

14. D. M. Hood and A. Kuppermann, in *The Theory of Chemical Reaction Dynamics*, D. C. Clary, ed., Reidel, Dordrecht, 1986, p. 193.

15. S. A. Cuccaro, P. G. Hipes, and A. Kuppermann, Chem. Phys. Lett. 155 (1989) 154.

16. D. W. Schwenke, D. G. Truhlar, and D. J. Kouri, J. Chem. Phys. 86 (1987) 2772.

17. G. A. Parker, R. T Pack, A. Lagana, B. J. Archer, and Z. Bacic, in *Supercomputer Algorithms for Reactivity, Dynamics, and Kinetics of Small Molecules*, A. Lagana, ed., Kluwer, Dordrecht, 1989, p. 105.

18. J. M. Launay and B. Lepetit, Chem. Phys. Lett. 144 (1988) 346.

19. G. C. Schatz, Chem. Phys. Lett. 92 (1988) 150.

20. D. A. Micha, Ark. Fys. 30 (1965) 407.

21. W. H. Miller, J. Chem. Phys. 50 (1969) 407.

22. Y. C. Tang, AIP Conf. Proc. 162 (1987) 174.

23. F. E. Harris and H. H. Michels, Meth. Comp. Phys. 10 (1971) 143.

24. D. G. Truhlar, J. Abdallah, Jr., and R. L. Smith, Adv. Chem. Phys. 25 (1974) 211.

25. P. G. Burke and W. D. Robb, Adv. At. Mol. Phys. 11 (1975) 143.

26. J. Callaway, Phys. Rep. 45 (1978) 89.

27. R. K. Nesbet, *Variational Methods in Electron-Atom Scattering Theory*, Plenum, New York, 1980.

28. D. L. Lynch, V. McKoy, and R. R. Lucchese, ACS Symp. Ser. 263 (1984) 89.

29. L. A. Collins and B. I. Schneider, in *Electron-Molecule Scattering and Photoionization*, P. G. Burke and J. B. West, eds., Plenum, New York, 1988, p. 147.

30. D. W. Schwenke, K. Haug, D. G. Truhlar, Y. Sun, J. Z. H. Zhang, and D. J. Kouri, J. Phys. Chem. 91 (1987) 6080.

31. D. W. Schwenke, K. Haug, M. Zhao, D. G. Truhlar, Y. Sun, J. Z. H. Zhang, and D. J. Kouri, J. Phys. Chem. 92 (1988) 3202.

32. N. C. Blais, M. Zhao, D. G. Truhlar, D. W. Schwenke, and D. J. Kouri, Chem. Phys. Lett., submitted.

33. R. G. Newton, *Scattering Theory of Particles and Waves*, McGraw-Hill, New York, 1986, Sect. 11.3.

34. G. Staszewska and D. G. Truhlar, J. Chem. Phys. 86 (1987) 2793.

35. M. R. Hermann and W. H. Miller, Chem. Phys. 109 (1986) 163.

36. W. H. Miller and B. M. D. D. Jansen op de Haar, J. Phys. Chem. 86 (1987) 6213.

37. J. Z. H. Zhang and W. H. Miller, Chem. Phys. Lett. 140 (1987) 329.

38. J. Z. H. Zhang, S. Chu, and W. H. Miller, J. Chem. Phys. 88 (1988) 6233.

39. A. C. Peet and W. H. Miller, Chem. Phys. Lett. 149 (1988) 257.

40. D. E. Manolopoulos and R. E. Wyatt, Chem. Phys. Lett. 152 (1988) 23.

41. D. E. Manolopoulos and R. E. Wyatt, Chem. Phys. Lett. 159 (1989) 123.

42. D. W. Schwenke, M. Mladenovic, M. Zhao, D. G. Truhlar, Y. Sun, and D. J. Kouri, in *Supercomputer Algorithms for Reactivity, Dynamics, and Kinetics of Small Molecules*, A. Lagana, ed., Kluwer, Dordrecht, 1989, p. 131.

43. M. Baer, J. Chem. Phys. 90 (1989) 3043.

44. F. Webster and J. C. Light, J. Chem. Phys. 90 (1989) 265.

45. J. Z. H. Zhang, D. J. Kouri, K. Haug, D. W. Schwenke , and D. G. Truhlar, J. Chem. Phys. 88 (1988) 2492.

46. J. Z. H. Zhang, Y. C. Zhang, D. J. Kouri, B. C. Garrett, K. Haug, D. W. Schwenke, and D. G. Truhlar, Faraday Discuss. Chem. Soc. 84 (1987) 371.

47. L. Schlessinger, Phys. Rev. 167 (1968) 1411.

48. L. Schlessinger, Phys. Rev. 171 (1968) 1523.

49. Y. Sun, D. J. Kouri, D. G. Truhlar, and D. W. Schwenke, Phys. Rev. A, submitted.

50. Y. Sun, D. J. Kouri, and D. G. Truhlar, Nucl. Phys., in press.

51. L. Delves and J. L. Mohamed, *Computational Methods for Integral Equations*, Cambridge University Press, Cambridge, 1985.

52. R. Peyret and T. D. Taylor, *Computational Methods for Fluid Flow*, Springer-Verlag, New York, 1983.

53. C. A. J. Fletcher, *Computational Galerkin Methods*, Springer-Verlag, New York, 1984.

54. *Spectral Methods for Partial Differential Equations*, G. Voight, D. Gottlieb, and M. Hussaini, eds., SIAM, Philadelphia, 1984.

55. C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang, *Spectral Methods for Fluid Dynamics*, Springer-Verlag, New York, 1988.

56. D. G. Truhlar and A. Kuppermann, J. Amer. Chem. Soc. 93 (1971) 1840.

57. D. H. Oza and J. Callaway, J. Comp. Phys. 68 (1987) 89.

58. B. Basden and R. R. Lucchese, J. Comp. Phys. 77 (1988) 524.

59. Y. Sun, C. Yu, D. J. Kouri, D. W. Schwenke, P. Halvick, M. Mladenovic, and D. G. Truhlar, J. Chem. Phys. 91 (1989) 1643.

60. I. P. Hamilton and J. C. Light, J. Chem. Phys. 84 (1986) 306.

61. M. Mladenovic, M. Zhao, D. G. Truhlar, D. W. Schwenke, Y. Sun, and D. J. Kouri, Chem. Phys. Lett. 358 (1988) 146.

62. M. Zhao, M. Mladenovic, D. G. Truhlar, D. W. Schwenke, Y. Sun, D. J. Kouri, and N. C. Blais, J. Amer. Chem. Soc. 111 (1989) 852.

63. M. Mladenovic, M. Zhao, D. G. Truhlar, D. W. Schwenke, Y. Sun, and D. J. Kouri, J. Phys. Chem. 92 (1988) 7035.

64. C. Yu, Y. Sun, D. J. Kouri, P. Halvick, D. G. Truhlar, and D. W. Schwenke, J. Chem. Phys. 90 (1989) 7608.

65. M. Zhao, D. G. Truhlar, D. J. Kouri, Y. Sun, and D. W. Schwenke, Chem. Phys. Lett. 156 (1989) 281.

66. N. C. Blais, M. Zhao, M. Mladenovic, D. G. Truhlar, D. W. Schwenke, Y. Sun, and D. J. Kouri, J. Chem. Phys. 91 (1989) 1038.

67. G. C. Lynch, P. Halvick, D. G. Truhlar, B C. Garrett, D. W. Schwenke, and D. J. Kouri, Z. Naturforsch. 44a (1989) 427.

68. C. Yu, D. J. Kouri, M. Zhao, D. G. Truhlar, and D. W. Schwenke, Chem. Phys. Lett. 157 (1989) 491.

69. M. Zhao, M. Mladenovic, D. G. Truhlar, D. W. Schwenke, O. Sharafeddin, Y. Sun, and D. J. Kouri, J. Chem. Phys. 91 (1989) 5302.

70. C. Yu, D. J. Kouri, M. Zhao, D. G. Truhlar, and D. W. Schwenke, Int. J. Quantum Chem. Symp., in press.

71. M. Zhao, D. G. Truhlar, D. W. Schwenke, and D. J. Kouri, J. Phys. Chem., submitted.

72. P. Halvick, D. G. Truhlar, D. W. Schwenke, and D. J. Kouri, J. Phys. Chem., submitted.

73. C. Yu, D. J. Kouri, M. Zhao, D. G. Truhlar, and D. W. Schwenke, unpublished.

74. M. Zhao, D. G. Truhlar, D. W. Schwenke, C. Yu, and D. J. Kouri, J. Phys. Chem., submitted.

75. F. H. Stillinger and T. A. Weber, Phys. Rev. A 28 (1983) 2408.

76. K. Haug, D. W. Schwenke, Y. Sun, D. G. Truhlar, J. Z. H. Zhang, and D. J. Kouri, J. Phys. Chem. 90 (1986) 6757.

Software Environments for the Parallel Solution of Partial Differential Equations

L. Ridgway Scott
the University of Houston

Most commercial supercomputers are now sold as multi–processor systems, and recent performance gains have come primarily from the additional processors rather than improvement in the single–processor performance. New systems based on large numbers of low–cost VLSI processors have been introduced that offer supercomputer performance at a fraction of the cost of conventional architectures. However, a major obstacle to achieving either the overall performance of the multi–processor supercomputers or the cost–effectiveness of the newer "parallel VLSI supercomputers" is the difficulty of programming them. Performance gains are also being achieved by the development of complex algorithms such as multi–grid, however such algorithms are often avoided even on conventional architectures due to the difficulty of programming them. We are developing techniques for implementing highly efficient (and theoretically justified) algorithms for scientific computation on parallel architectures by combining expertise in both computer science and mathematics. The objective is to do so in a way that the resulting codes are both efficient on, and portable among, a wide range of parallel computer architectures.

Most research on parallel programming languages in the past has been devoted to programming multiple tasks on a single processor, e.g., as must be done in an operating system for a conventional computer. However, new performance gains are now expected from parallel–cpu computers that co–operate on a single task. Programming language techniques for the latter environment are currently being studied by a number of researchers. We have proposed a language construct that allows code on one process(or) to access variables explicitly (by name only) that are "stored" in another process(or). Thus it allows one to program a distributed–memory machine as if it has a common address or name space. This technique led to significantly shorter development time for parallel codes, as well as improved portability (implementations were carried out on both the NCUBE and iPSC) and reliability. We have found it possible to program a large number of diverse algorithms quickly and to obtain excellent performance. We are extending these techniques to include the use of advanced programming languages which allow the implementation of abstract data types. This makes coding and debugging finite element and finite difference applications much faster and more reliable.

COMPUTATIONAL ASPECTS OF "FAST" PARTICLE SIMULATIONS

CHRISTOPHER R. ANDERSON*

Abstract. In many particle simulations the calculation of the potential (velocity field, force, etc.) requires $O(N^2)$ operations where N is the number of particles. In this paper we describe the basic ideas behind three methods which are employed to reduce this operation count to approximately O(N). We discuss the issue of parameter selection for these methods and present some computational evidence which demonstrate the importance of making good choices for the method parameters. We conclude with some opinions about the relative merits of the methods.

1. Introduction. The purpose of this talk is to discuss several methods for reducing the computational time required to carry out particle simulations. The simulations I have in mind are those which occur in a wide variety of physical problems - plasma physics, astronomy, incompressible fluid flows, etc. In rather general terms, in order to compute the evolution of the particles in these simulations, one is is required to compute a potential, force, or velocity which every particle induces on every other particle. If one computes this interaction explicitly (using a formula I will give below) then the computational time is proportional to $O(N^2)$ where N is the number of particles. I shall refer to this explicit calculation as a "direct" method. There are so called "fast" methods for which the computational work is O(N) (or $O(N \log N)$, etc.).

Now the $O(N^2)$, or direct, method is very easy to implement, it takes about fifteen lines of Fortran code. It is also relatively simple to write the code so that advanced computational hardware (vector or multiprocessor units) can be used to significantly reduce the computational time. On the other hand the O(N), or fast, methods are rather difficult to implement (often several hundred lines of code) and organizing the computation so that vector or multiprocessors are utilized efficiently is a challenging problem. These fast methods typically have parameters which must be specified in advance of the computation. A wrong choice of these parameters can lead to a "fast" method which takes more CPU time than the direct method. Moreover, there are several fast methods, each with somewhat different properties, so that the choice of which method to use can be difficult. In spite of these drawbacks there is still great interest in the O(N) type methods - primarily because

* Department of Mathematics, UCLA, Los Angeles, California, 90024
Research Supported by ONR Contract #N00014-86-K-0691, NSF Grant DM586-57663
and IBM Fellowship D880908

the benefit of using such a scheme (when implemented correctly) can be enormous. As an example, in a two dimensional vortex calculation the running time for a direct evaluation of the velocity field induced by N=31,000 particles was 360 seconds, while a fast method took only 4 seconds [3].

This reduction in running time is sufficient to stimulate a bit of interest in these fast methods, and so I intend to discuss three different fast methods. While these do not exhaust the set of methods one can choose from, these are methods I am familiar with, and I believe they form a representative sample of the whole class. While the methods may appear to be very different, each of them utilizes the same basic principle, and I will "derive" the methods by discussing the basic principle and indicating how this is then developed into the methods completed form. This discussion will be rather general, and one should consult the references for the precise details. The general information is not without value, for it is through this general understanding that one can appreciate the issues related to parameter selection and other implementation details. I will show some computational evidence which shows why one should be concerned with parameter selection and indicate how one might choose the relavent parameters optimally. Lastly I will discuss some aspects of method selection. Each of these methods take considerable time to implement (or to just learn how to run efficiently) and so the issue of appropriate method selection is important.

2. Fast Methods. The computational task in the particle simulation one performs depends on the type of simulation – for vortex calculations one computes the velocity field induced by the particles, for galaxy simulations one computes the force induced by gravitational attraction etc. Rather than deal with each, I will discuss fast methods for the following model problem: Given N charged particles at locations x_i with strengths κ_i then the goal is to calculate the potential $\phi(x_i)$, where ϕ is a solution of

$$\Delta\phi = \sum_{i=1}^{N} \delta(x - x_i)\kappa_i \tag{2.1}$$

and $\delta(x)$ is Dirac's delta function and Δ is the Laplacian. (If one wants forces then one computes the gradient of this solution.) I will work with the two-dimensional case since the key ideas behind the methods don't differ much when one goes from two to three dimensions and the two dimensional case is easier to describe. There are certain efficiency issues which change with dimension and I will address these specifically. I am also not discussing computations in bounded domains. This feature introduces some important complications, but it does not influence the basic structure of the fast methods, and so I am assuming the computation is carried out in all of $\mathbf{R}^2$.

Often in simulations one uses smoothed delta functions (or "blobs") and the model problem in this case is identical to (2.1) but we solve

$$\Delta\phi = \sum_{i=1}^{N} \Psi_\epsilon(x - x_i)\kappa_i \tag{2.2}$$

where Ψ_ϵ is a smoothed delta function whose support is contained within a disk of ϵ. The choice of Ψ and ϵ is important for a simulations accuracy, but not particularly important for the methods used to accelerate the computation of (2.2). I make the assumption that the blobs have support contained within disks or spheres of radius ϵ, and so the blob functions can not be completely general.

The solution ϕ of (2.1) is given by

$$\phi(x) = \sum_{i=1}^{N} \frac{\kappa_i}{2\pi} \log(|x - x_i|) \tag{2.3}$$

or for (2.2)

$$\phi(x) = \sum_{i=1}^{N} \frac{\kappa_i}{2\pi} \Phi_\epsilon(x - x_i) \tag{2.4}$$

where Φ_ϵ is the potential induced by a single blob. (This can often be calculated explicitly by solving $\Delta\Phi = \Psi_\epsilon(x)$ using the method of separation of variables.)

From (2.3) or (2.4) it is clear that if we evaluate the solution ϕ at each point $x_i, i = 1, ...n$, then this computation requires $O(N^2)$ operations. For particle simulations, the larger the value of N the better, and so there is great interest in reducing this operation count.

A common ingredient to all of the fast methods which I am going to describe consists of representing the potential induced by a cluster of particles by a single computational element. The savings in computation time comes about because this new computational element typically takes less work to evaluate than evaluating the potential directly for the individual particles. The "cost" of this procedure is the loss of accuracy which occurs when one represents this potential by the new computational element. The implementation of this idea is perhaps most clearly seen in the multipole method [8]. Consider a cluster of M particles contained within a disk of radius R. (See figure 1). Outside the disk, the potential is the real part of an analytic function and therefore can be represented as the real part of a Laurent expansion. (Here we are identifying the x-y plane with the complex z plane.) Thus, we have

$$\phi = \mathrm{Re}\,(\sum_{i=1}^{M} \frac{1}{2\pi} \log(z - z_i)) = \mathrm{Re}\,(\kappa \log(z - z_0) + \sum_{k=1}^{\infty} \frac{a_k}{(z - z_k)^k})$$

where z_0 is the center of the disk containing the particles and κ and a_k are coefficients calculated from the strengths and locations of the particles in the disk. If an evaluation point is outside a disk of radius 2R then the error in using p-terms of this Laurent expansion is $O((\frac{1}{2})^p)$. Thus, given some desired accuracy, we select a value of p which assures this accuracy and use the p-term truncated expansion to represent the potential induced by M particles, i.e. we use the approximation

$$\phi = \mathrm{Re}\,(\sum_{i=1}^{M} \frac{1}{2\pi} \log(z - z_i)) \approx \mathrm{Re}\,(\kappa \log(z - z_0) + \sum_{k=1}^{p} \frac{a_k}{(z - z_k)^k})$$

This finite Laurent series is the new computational element, and the cost of evaluating this element is $O(p)$ compared to $O(M)$ which would be required to evaluate the potential directly. Since the number of particles M does not factor into the error estimate, one obtains computational speedup if M is much greater than p. There is of course work involved in constructing the finite expansion - but this only requires O(M) computations and need be done only once. This work is therfore "shared" when there are multiple evaluations of the potential.

This idea of collapsing a cluster of particles to another computational element which is easier to evaluate can take other forms. In particle in cell methods a finite difference grid of mesh width h covers the the cluster of computational particles and any desired evaluation points. (See figure 2.) One then assigns values of charge density to the grid nodes to approximate the charge distribution of the computational particles. Typically a given particles charge is assigned to a small number of nearby grid nodes. To evaluate the potential which is induced by these particles we solve a discrete Laplace equation with the forcing function given by this grid charge density. The result is an approximation to the potential at all of the grid nodes. To evaluate the potential at points between the grid nodes

an interpolation formula is used. Again, the work to evaluate the potential once the grid charge approximation has been made is independent of the number of particles M. In this particle in cell approach, the new computational element is the combination of a finite set of grid values (used represent the charge density on the grid) and an inversion of a discrete Laplacian followed by an appropriate interpolation. (This may seem peculiar to lump the inversion of the discrete Laplacian into the element, but this is appropriate since each term in the multipole expansion really corresponds to inverting a Laplacian.)

There is the question of accuracy, and this is what distinguishes the two particle in cell techniques I will discuss. In the first type of method one selects an assignment scheme, a discrete Laplace approximation, and an interpolation scheme and this determines the method. Charge densities are transferred to a grid, a discrete Laplace equation is solved, and then the potential is interpolated at the desired evaluation points. There is no restriction as to where the evaluation point is with regard to the particles which give rise to the potential. There are numerous choices of each of the components, essentially leading to a hierarchy of methods with increasing complexity and providing increasing accuracy. I am not going to go into detail about the different possibilities, one can find many of them described in [10], but merely note that all of these procedures have an accuracy which is closely related to the size of the finite difference grid which is used to cover the computational domain. The finer the computational mesh used, the more accurate the results. I shall refer this type of method as a PIC (particle in cell) method.

Another type of method [1] is similar to the type describe above, but with two distinct differences. The first difference is that attention is paid to the location of the evaluation point with regard to the particles which give rise to the potential. If the evaluation point is too close to the source particles, then the potential from the grid is not used and a direct evaluation is performed. The motivation for this comes from the observation that the error in the potential of the particle in cell procedure is greatest near the source particles and diminishes rapidly away from the source particles. The second aspect is the use of a method for assigning charges to the grid so that the potential which is obtained after the inversion of the discrete Laplacian is not coupled to the order of accuracy of the discrete Laplacian. (Essentially the assignment scheme consists of figuring out what should be placed on the grid so that after the inversion of the discrete Laplacian, the potential values have an accuracy which is independent of the mesh size). These changes lead to a method for which the accuracy is relatively insensitive to the grid size and is determined by other considerations - for example the size of the region in which the direct evaluation is used as well as parameters associated with the charge assignment procedure. Since the method involves correcting the potential in regions localized about the source points, I will refer to this technique as the method of local corrections (MLC). (One should note that the evaluation of the potential from grid values is only done at points away from the source points, a region in which the potential is analytic, so that high order interpolation formulas can be used which take advantage of this fact. [1], [12])

I have described the basic idea behind three fast methods - the multipole method uses truncated Laurent expansions and the PIC and MLC methods use potentials induced by grid based values, to approximate the potential due to a cluster of particles. Of course, there is a bit of work in creating a complete method from this basic idea. For the particle in cell approaches it is not difficult to see how a complete method is formulated. Since the problem of finding the potential is linear, one considers all particles together, assigns their charge (mass, vorticity, etc.) to the grid, solves one discrete Laplacian and then interpolates the resulting potential at all of the required evaluation points. If there are local corrections to be done (MLC method), then for each evaluation point the value of the potential induced by nearby particles is computed by formula of type (2.3) or (2.4).

This value is added to the potential at the evaluation point and a value corresponding to the nearby particles contribution to the interpolated grid potential is subtracted from the potential at the evaluation point.

Efficiently implementing the concept behind the multipole method is a little more difficult. There are several ways one can exploit the finite Laurent expansions, and I will just discuss one of them, that due to Greengard and Rokhlin [8] To use the idea effectively a box is chosen which covers the computational domain. By recursively dividing the box by factors of two, a nested set of grids is constructed. One selects a finest level of refinement m so that there are 2^m boxes in each direction at that level. For a given set of evaluation points in one of these finest level boxes, the potential at these points is formed from a direct interaction with particles in the nearest boxes added to the potential due to multipole expansions associated with the particles in a hierarchy of surrounding boxes. The further away one is from the evaluation points, the larger the box used to construct the multipole expansion. One wants to use multipole expansions with the largest box possible, but there is the constraint that the disk which covers the source particles be a sufficient distance from the evaluation point box to retain accuracy. As an example of the multipole structure which is used, see figure 3. In this figure the hatched circle indicates the box in which the evaluation points are associated with and the other circles represent multipole expansions. The center of the circle is the center of the multipole, and all the particles in the box in which the circle sits are used to form the particular multipole expansion. It is clear that as the evaluation points change the set of multipoles which are used to approximate the potential changes. However, what doesn't change is the fact that some subset of multipoles for small boxes, the next larger boxes, etc. is always used. Thus, in the implementation one first constructs all of the multipoles associated with all the different size boxes. The next step is to be clever in the evaluation of the appropriate members of this hierarchy. An idea introduced by Greengard and Rohklin is to have an equivalent hierarchy of power series expansions to accomplish this. Rather than go into details about how this is done (see [2] or [8]), it is instructive to consider how the potential of one particle is "communicated" to an evaluation point via this network of multipoles and power series expansions.

The source particle in figure 4 is denoted by "x" and the evaluation point is denoted by an "o". The existence of the source particle causes multipole expansions to be formed for every box which contains that particle at all the successively coarser levels. Each of these multipole expansions is indicated by a disk. These expansions are formed recursively, and so the potential that the source particle induces is first represented by the multipole expansion on the finest level. The potential induced by this expansion is then in turn represented by the multipole expansion on the next coarser level, etc.. Next the potential induced by the coarse level multipole expansion is represented by a coarse level power series expansion which is associated with the box containing the evaluation point. (The largest shaded disk in figure 4.) The potential induced by this power series expansion is then in turn represented by a a power series expansion at the next finer level, etc. until the potential is finally represented by a power series expansion associated with the finest level. This finest level expansion is then evaluated to obtain the potential induced by the source particle at the evaluation point. Every source particle "talks" to the evaluation points through this hierarchy of multipoles and power series expansions. The number of levels which are used in this communication process depends on the distance between the particles - the further away the source and evaluation point are, the more levels which are used. An important point to note is that this computational structure utilizes elements which are localized about the computational particles themselves. In this communication process, multipoles and power series expansions associated with boxes not directly above any source particles or evaluation points are not used. This is a bit different from the particle in cell methods

in which the communication network, a finite difference grid, covers the whole domain and is not localized about the particles.

3. Parameter Selection. The operation count of each of the methods described above is O(N) for the multipole method and $O(N) + O(M \log M)$ for grid based methods in which an M×M grid is used. Thus, in terms of this type of asymptotic operation count, as N is changed the distinction between the methods is covered up by the "O" symbol. A comparison in efficiency must be based on the size of the constant in the asymptotic work estimate, and this constant depends on a whole host of factors. The size of this constant is directly influenced by several of the parameters which must be specified before the fast methods can be used. I would now like to discuss some of these parameters, and indicate how they might be optimally chosen. It is only after such choices are made that meaningful information about the relative efficiencies of the methods might be determined.

In the standard PIC method, one must select a charge assignment scheme,a discrete Laplace approximation, an interpolation scheme, and a grid size. The finer the grid, the more work involved in inverting the Laplacian. The more accurate the interpolation and assignment schemes, the more work is required to implement them. In most calculations the method of assignment, Laplace inversion, and interpolation is not changed, and so one has only to specify the grid size to be used for any particular calculation. Since the accuracy is tied to the size of the grid, accuracy considerations should dictate it's choice. Certainly the coarsest grid should be chosen for which the calculation retains sufficient accuracy. What sufficient accuracy is, is difficult to determine. This is really a question about the sensitivity of the particle simulation to inaccuracies in the potential (or velocity, or force etc.), i.e. it depends on the particular problem under consideration. With PIC methods it is often said that you cannot believe any result about particle structure which occurs on a sub-grid scale - you *should not* use such schemes if the phenomenon depends critically on particle motion on such scales. This is of coarse rather conservative advice, since it fails to distinguish between the higher order and lower order PIC methods.

We are fortunate that for some particle based methods, there is a convergence proof which explicitly account for the fact that a PIC method is used [7]. The central idea is to show that using a PIC method introduces an implicit smoothing. One then shows that the particle simulation with this smoothing converges to the solution of the equations as both the number of particles increases and the grid of the PIC method (and hence the smoothing) tend to zero. The error estimates worked out in this proof may be of use in determining the correct grid size one needs to obtain accurate solutions. (These estimates can also assist in developing systematic methods for analyzing charge assignment and interpolation schemes.) The fact that PIC methods are in some sense equivalent to smoothed particle schemes is interesting, for this indicates that if one is using the smoothed particle approach, then one might seriously consider a standard PIC method. One can conceivably obtain similar answers and at the same time have a computationally efficient method. Of course, with a PIC method it is difficult to control the type of smoothing introduced, but recent work by Merriman [11] should be of assistance here. (The approach of Merriman is also useful in that it indicates how one can accurately implement smoothing in the presence of boundaries.)

The selection of parameters is rather different for the other two methods. In the multipole method, one chooses the number of terms p in the finite Laurent and power series expansions. This choice is determined by the desired accuracy. (Again, what this should be depends on the problem under consideration.) Next, one must select the level of refinement used when the computational domain is decomposed into boxes. Unlike the standard particle in cell method, the size of the grid does not influence accuracy. However, the size of the grid does influence the computational efficiency of the method. In figure 5 we show the CPU

time required by an implementation of the multipole method [2] on a serial machine (SUN Sparcstation) for the evaluation of the potential of N randomly distributed particles. Each of the curves represents the computation performed with different levels of refinement. The results show that there is a marked difference in the efficiency of the method depending on the level of refinement used. For small numbers of particles, too much refinement is clearly undesirable. This fact is expected, since if there are too few particles per box (say less than p), then we are certainly doing more work by using a p term expansion to represent their potential.

Choosing the right level of refinement thus presents something of a problem. What makes the problem more complicated is that the amount of work depends on the distribution of particles. In figure 6 we show the CPU time for various numbers of particles when they are distributed in a rectangle with a ten to one ratio. One sees that it is indeed distinct from the case of particles uniformly distributed in the square. This dependence on the particle distribution is rather discouraging, since it therfore seems difficult to design an automatic procedure which would minimize the computation time. Fortunately, as discussed in [2], it is observed that the work for each level can easily be computed in advance and the minimum level selected. The key idea is to "dry run" the code, and, rather than carry out all of the required operations, increment counters based on particle density and the number of terms in the expansions. Using the value of these counters, the running time can be well estimated. Due to the recursive nature of the method, this is very easy to implement, and one can obtain quickly very good estimates of the running times of the multipole code for several refinement levels. With estimates for the work required by the various levels of refinement, one just selects the level with the least amount of work and carries out the computation. This strategy works well and the actual CPU time taken by the program forms a lower envelope for the timing curves in both figures 5 and 6.

The situation for the particle in cell method with local corrections is analogous to the multipole code. A certain amount of accuracy is specified, and this dictates the choice of parameters associated with the charge assignment and interpolation scheme. Once the accuracy is specified one must select the size of the grid. In figure 7 we show the results of timings for different size grids and given level of accuracy. (This was a velocity calculation computed on a Cray XMP [3] - timings for a potential calculation would be analogous.) Again, one should determine the optimal grid size to minimize the computational effort. As before, this can be done by "dry running" the code and computing timing estimates based on particle densities. Rather than compute these estimates for all possible grid sizes, it is computed for three grid sizes and the minimum of the quadratic interpolant is used to determine the optimal size.

If the asymptotic estimates are scaled to represent true CPU time, then the constants appearing in them will depend on on the particular hardware at hand. Certainly, there is much to be gained if advanced computational hardware can be utilized effectively. I have not looked into this aspect in great detail (other than implement the codes I work with on a vector/multiprocessor machine) but I am aware that it is not a simple matter to optimize any of these algorithms to take advantage of special hardware. For information about implementing PIC type methods on a parallel and vector processors, you might have a look at [4] and [5], while for a discussion of some of the difficulties involved in implementing multipole type methods, one might consult [9] or [6]. One of the problems which arises in taking advantage of vector/multiprocessor hardware is that the parallel or vectorization technique must be dynamically determined to reflect the changing nature of the interaction between the particles.

4. Conclusions. As I said earlier, the three methods I have spoken of by no means exhaust the set of possible methods, but I think they form a representative sample. One is

of course interested in specific recommendations for method choice. If I were just starting out, I would use the direct method. It is very easy to program and you can be assured of the accuracy of the computation. After a true need for a faster method has developed, the next choice is between a grid based method and a multipole based scheme. For two dimensions, it is not clear to me that one of the methods is substantially superior over the other. It is true that controlling the accuracy is easier when using a multipole method - i.e. it essentially depends on one parameter, p, the number of terms used in the expansions. For grid based schemes, accuracy is determined by several factors and it takes more work to find the correct combination to achieve some level of accuracy. Aside from this accuracy consideration any specific choice should probably be determined by other factors - such as the difficulty of implementation, the ability to exploit specific computational hardware, the ease of incorporating boundary conditions, etc. For three dimensions, there may be a reason to choose one method over the other, but the choice depends on particle distribution. If the particles are uniformly distributed then either a multipole method or a grid based scheme would be acceptable. If the particles are not uniformly distributed, then there may be good reason to select a multipole method. As mentioned earlier, the network of computational elements in a multipole method are localized about the particle locations, and so, if the particles are distributed in a lower dimensional fashion (i.e. along a line or surface) then the computational network is also of lower dimension. This can mean great savings, both in computation time and in storage. A grid based method takes no advantage of the lower dimensionality of the particle distribution - the computational structure fills out the complete box in which the particles reside. This introduces a significant computational overhead into the grid based method. Of course if one uses an adaptive grid method, then this observation must be modified.

REFERENCES

[1] C. R. Anderson, "A Method of Local Corrections for Computing the Velocity Field Due to a Distribution of Vortex Blobs", J. Comp. Phys., **62**, 1988, pp. 111-123.

[2] C.R. Anderson, "An Implementation of the Fast Multipole Method Without Multipoles", *in preparation.*

[3] S.B. Baden, *privite communication.*

[4] S.B. Baden, "Run-Time Partitioning of Scientific Continuum Calculations Running on Multiprocessors", Lawrence Berkeley Laboratory Report, LBL-23625, Berkeley, Ca., 1987.

[5] S.B. Baden,"Very Large Vortex Calculations in Two Dimensions", in *Vortex Methods*, C. Anderson and C. Greengard, (Eds.), Lecture Notes in Mathematics, 1360, Springer-Verlag, 1988.

[6] J.E. Barnes, "A Modified Tree Code: Don't Laugh, It Runs", Astrophysics Preprint Series, IASSNS-AST 89/13, Institute for Advanced Study, Princeton, NJ., 1989.

[7] G.H. Cottet, "Convergence of Vortex-In-Cell Methods for the Two Dimensional Euler Equations", Math. Comp., **49**, 1987, pp. 407-425.

[8] L. Greengard and V. Rokhlin, "A Fast Algorithm for Particle Simulations", J. Comp. Phys., **73**, pp. 325, 1987.

[9] L. Hernquist, "Vectorization of Tree Traversals", Astrophysics Preprint Series, IASSNS-AST 88/64, Institute for Advanced Study, Princeton, NJ., 1988.

[10] R. Hockney and J. Eastwood, *Computer Simulations Using Particles*, McGraw Hill, New York, 1979.

[11] B. Merriman, "Smooth Particle Methods on Bounded Domains", Tech. Rept. 89-06, Dept. of Comp. Sci., Univ. of Chicago, 1989.

[12] A. Mayo, "On the Accuracy of a Class of Momentum Conserving Particle Mesh Methods, and the Interpolation of Elliptic Functions on Uniform Meshes", IBM Research Report, RC 14306 (#64088), T.J. Watson Research Center, Yorktown Heights, NY., 1989.

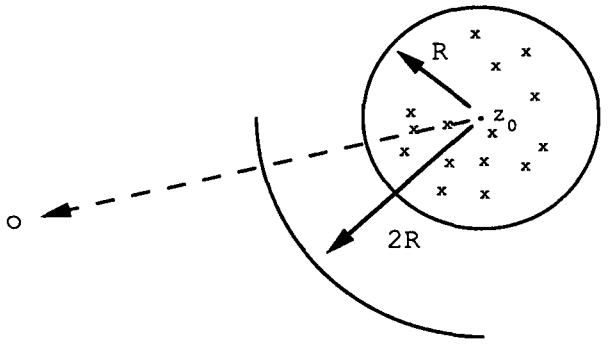


Figure 1

The potential induced by the particles in the disk of radius R is represented by a finite term Laurent expansion centered at z_0 .

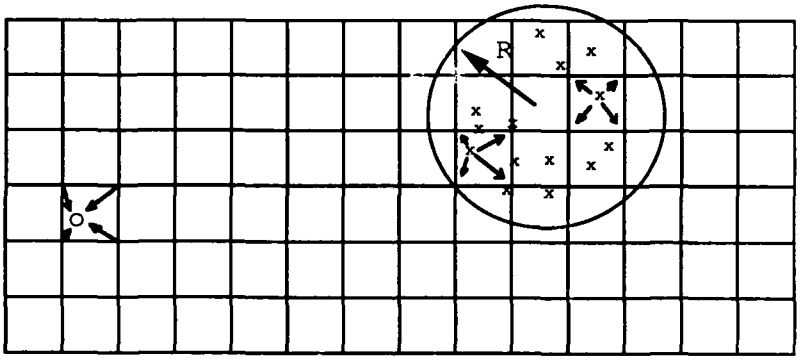


Figure 2

The potential induced by particles in the disk of radius R is evaluated by interpolating a solution of the discrete Laplacian. The right hand side for the discrete Laplacian is constructed by assigning the charge of the particles to the finite difference mesh.

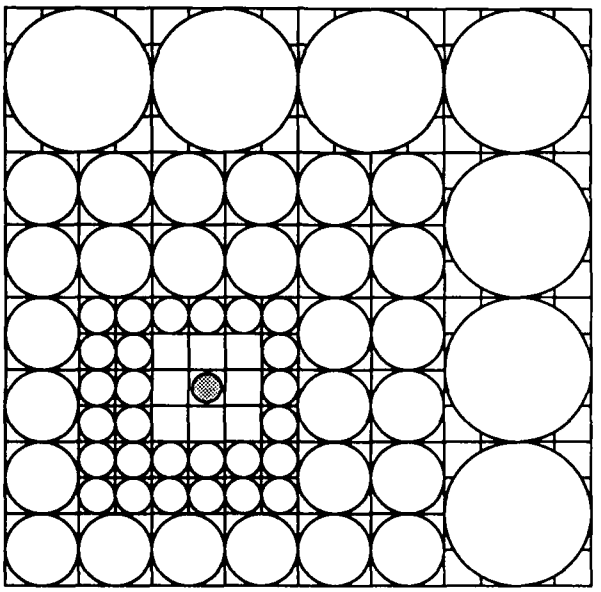


Figure 3

Circles indicate multipole expansions used to compute the potential at the evaluation points in the shaded circle

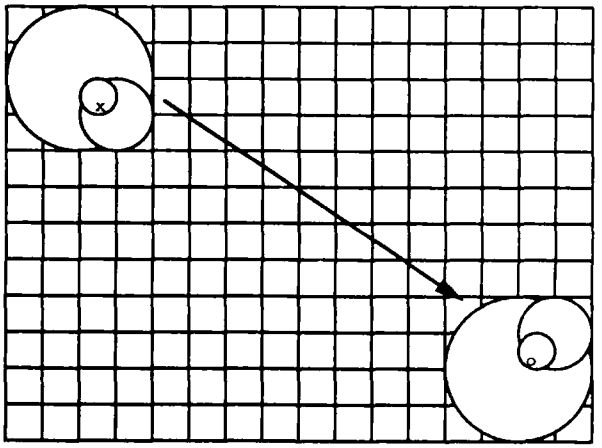


Figure 4

The multipole (plain disks) and power series expansions (dotted disks) used to communicate the potential induced by the particle at x to the point at o.

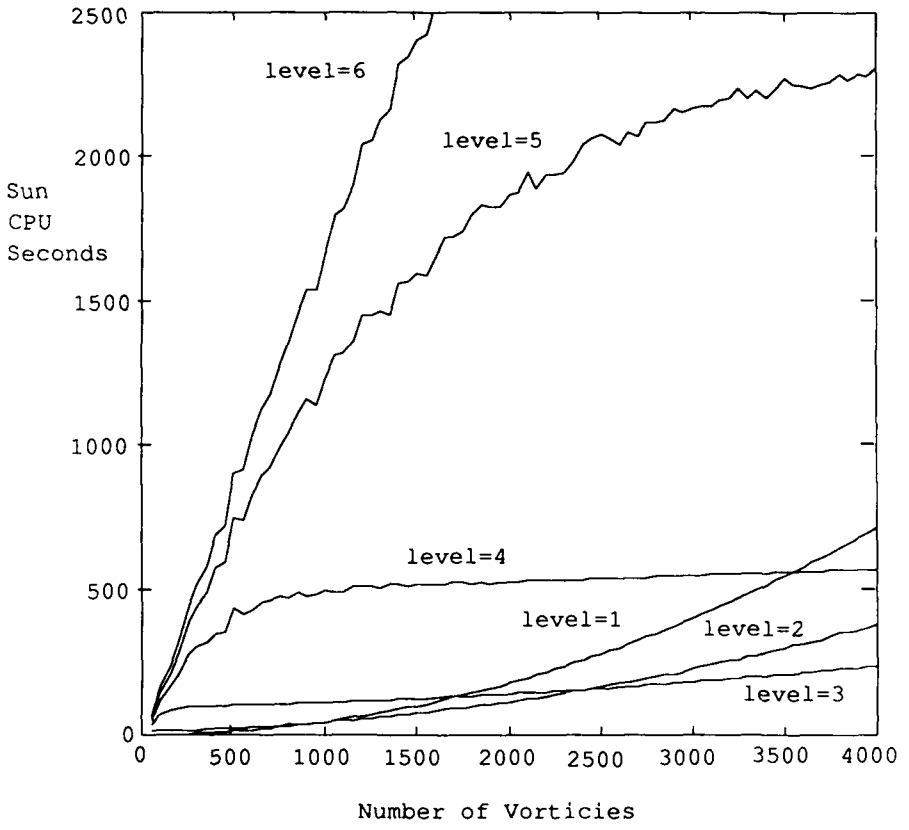


Figure 5

CPU time (in seconds) vs. number of vorticies for particles uniformly distributed in a unit box. Each curve represents computation time for a refinement of space into 4^level boxes.

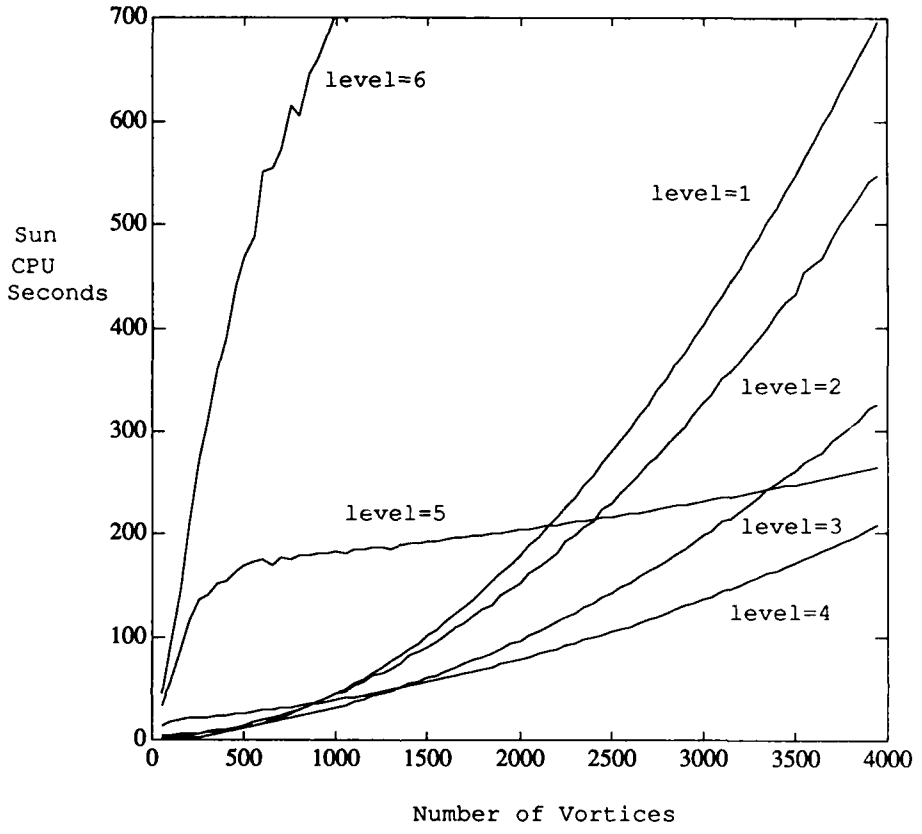


Figure 6

CPU time (in seconds) vs. number of vorticies for particles uniformly distributed in a rectangle with an aspect ratio 10 to 1. Each curve represents the refinement of the surrounding square into 4^level boxes.

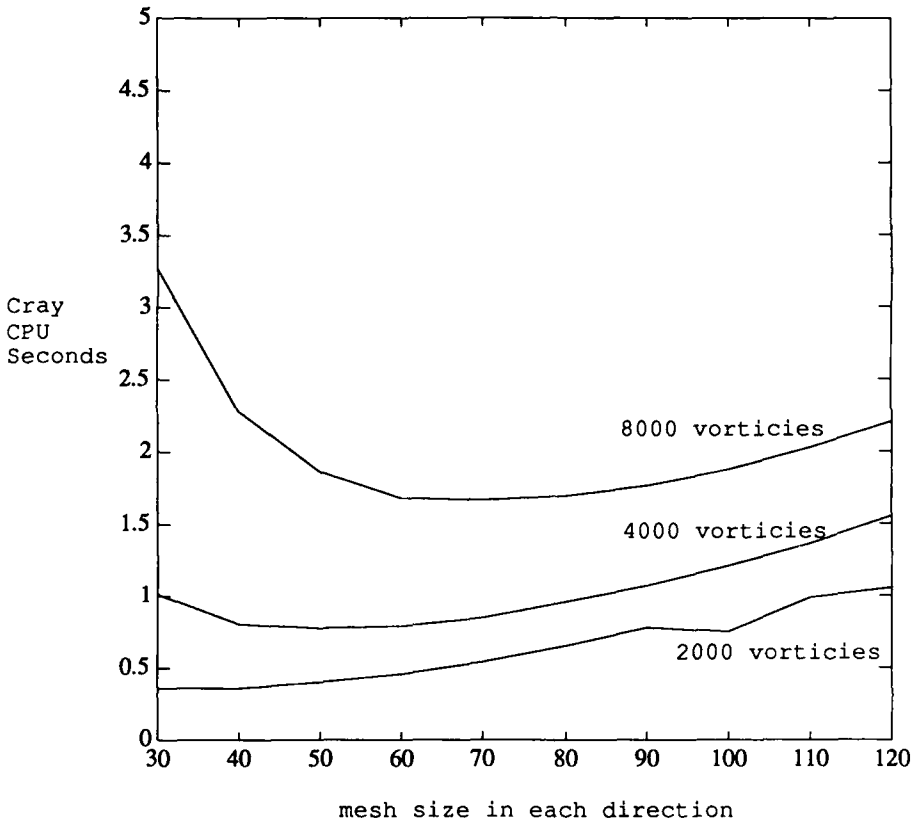


Figure 7

CPU time (in seconds) vs. size of mesh in each direction.
Different curves correspond to differing number of vorticies.

SEMICONDUCTOR MODELLING VIA THE BOLTZMANN EQUATION[1]

P. DEGOND[2]
F. DELAURENS[2]
F.J. MUSTIELES[2]

"9-th International Conference on Computing Methods in Applied Sciences and Engineering"
Paris (France), January 29 to February 2, 1990

ABSTRACT : This paper is devoted to the presentation of a new numerical method for the simulation of the Boltzmann Transport Equation of semiconductors, the weighted particle method. A detailed presentation of the method can be found in [1,2] and its mathematical analysis has been performed in [3]. In this paper, we will describe the kinetic model of the Boltzmann Equation and present the numerical method that we propose. We deal with two different cases : first, an homogeneous case, and then an inhomogeneous one, where one has to solve a coupled Boltzmann-Poisson system. The numerical study of this latter case has been done in [4], and is detailed in [5].

1. INTRODUCTION

Most of the numerical simulations of semiconductor devices use the drift-diffusion model [6,7]. This model is based upon Ohm's law (for drift) and Fick's law (for diffusion) and states that the average velocity of the carriers is proportional to the electric field ; the proportionality coefficient is a field dependent mobility. This relation is obtained at equilibrium, as a consequence of the balance between the free acceleration of the carriers and their diffusion by the defects of the crystal lattice. The time needed for this equilibrium to be reached is the momentum relaxation time (mean time between collisions), so that Ohm's law is valid as long as this relaxation time is shorter than the time needed for the carriers to cross the device. But, in submicron devices, some carriers have almost collisionless (or ballistic) flights, and thus the average velocity can be higher than Ohm's law predicted value

1 This work has been partially supported by the "Centre National d'Etudes des Télécommunications", under grant n° 878B087 LAB/ICM/TOH, and by the "Direction des Etudes et Recherches Techniques", under grant n° 87/283.

2 Centre de Mathématiques Appliquées ; URAD CNRS n° 756
Ecole Polytechnique - 91128 Palaiseau Cedex - FRANCE

[6,8]. In fact, the drift-diffusion model does not take into account the main features of transport in submicronic devices [9] : the presence of ballistic carriers, the large proportion of high velocity ("hot") carriers, and the large gradients of carrier density and temperature.

For the simulation of hot electron effects, a more involved hydrodynamic model has been proposed by many authors [8,10,11]. It consists of conservation equations for the mass, momentum and energy, and is deduced from the Boltzmann Transport Equation by the moment method, under the assumption that the distribution function of the carriers is a drifted Maxwellian. Scattering processes are accounted for by empirically defined relaxation times for momentum and energy at the right hand side of these conservation equations. Some other modifications [12,13] incorporate thermal effects. Nevertheless, this model hardly describes the ballistic and hot electron effets, and no completely satisfactory hydrodynamic model seems to be available yet.

The kinetic model (the Boltzmann Equation) then seems to give the most accurate description of the physics attainable by numerical computations. In this paper, we will recall the main features of the semiconductor Boltzmann equation, and describe the weighted particle method. We will provide the results of numerical simulations in two different cases. First, the homogeneous field model provides a nice framework for the validation of our method particularly concerning the deterministic treatment of the collision term [1,2]. The physical situation is that of an infinite sample of semiconductor imbedded in a uniform electric field. This model also provides the stationary velocity and energy as a function of the applied electric field, as well as the relaxation times for energy and momentum [14] , which are useful in a hydrodynamical model. Second, we turn to the simulation of a one-dimensional inhomogeneous structure, which requires to solve a coupled Boltzmann-Poisson system [4,5]. For more details about the model, the numerical method in both the homogeneous and inhomogeneous cases, we refer the reader to [1,2,4,5] . The numerical analysis of the method in the homogeneous case has been performed in [3] . For a more detailed physical description of the kinetic model, we refer the reader to [6,15,16].

2. THE SEMICONDUCTOR BOLTZMANN EQUATION

We will suppose that the electrons are the only charge carriers in the device. This is a reasonable assumption for many N-doped unipolar devices. The Boltzmann equation for the electron distribution function $f(x,k,t)$ (where x is the position, k the wave-vector and t the time) is written in the following way :

$$\partial_t f + v(k).\nabla_x f - (q/\hbar) E(x,t).\nabla_k f = Q(f)(x,k,t)$$
$$x \in \Omega \subset \mathbf{R}^3 ; k \in B \subset \mathbf{R}^3 , t \geq 0 . \qquad (1)$$

In the equation (1), Ω stands for the device geometry and B for the first Brillouin zone. q is the positive elementary charge and $\hbar$ the reduced Planck constant. The field $v(k)$ is given and provides the velocity versus wave-vector relationship for an electron in the semiconductor material. $Q(f)$ is the scattering operator describing the interactions of the electrons with the lattice defects. The electric field $E(x,t)$ is related to the electron density $n(x,t)$ via Poisson's equation :

$$E(x,t) = -\nabla\phi(x,t) \tag{2}$$

$$-\Delta\phi(x,t) = \frac{q}{\varepsilon}\,(n_D(x) - n(x,t)) \tag{3}$$

$$n(x,t) = \int f(x,k,t)\,\rho_s\,dk \tag{4}$$

ε and ρ_s are respectively the permittivity of the material and the density of states in the k-space, and have known values ; $n_D(x)$ is a given doping profile resulting from the fabrication technology.

For quantum mechanical reasons, it may happen that one needs to distinguish between several species of electrons which are characterized by different (effective) masses and thus, different functions $v(k)$. These different species are called valleys. For instance, in Gallium Arsenide, three valleys have to be considered, the Γ, L and X valleys. In this case, the model consists of as many distribution functions as valleys, each of them solving its own Boltzmann equation. These different equations are coupled in two ways : first, via an "intervalley" collision operator, of a similar form as the "intravalley" one (see below), second, via Poisson equation. We refer to [1,2,5,6,9] for more details. For the sake of simplicity, we will describe the kinetic model and the weighted particle method for a single valley model, and thus, for a single distribution function.

The equations (1)-(4) must be supplemented by initial and boundary conditions :

$$\begin{aligned} f(x,k,0) &= f_0(x,k) \\ f(0,k,t) &= g_0(k,t) \quad \text{for } v(k) \geq 0 \\ f(L,k,t) &= g_L(k,t) \quad \text{for } v(k) \leq 0 \\ \phi(0,t) &= \phi_0(t) \quad , \quad \phi(L,t) = \phi_L(t) \end{aligned}$$

with f_0, g_0, g_L, ϕ_0 and ϕ_L suitably given.

The integral scattering operator $Q(f)$ is written :

$$Q(f)(x,k,t) = \int_B [S(x,k',k)\, f(x,k',t)(1-f(x,k,t)) - S(x,k,k')\, f(x,k,t)(1-f(x,k',t))]\, dk' \qquad (7)$$

$S(x,k,k')$ are known transition rates depending upon the physical nature of the involved scattering processes. The (1-f) factors originate from Pauli's exclusion principle and make Q a non linear operator. For some examples of transition rates, we refer the reader to [2,5,6,17]. The overall transition rate may be written in the form :

$$S(x,k,k') = \sum \phi(x,k,k')\, \delta(\varepsilon(k') - \varepsilon(k) \pm \hbar\omega_p) \qquad (8)$$

The sum is to be taken over + and -, respectively standing for the emission and the absorption of a phonon of energy $\hbar\omega_p$ by an electron, and over all the possible scattering mechanisms. The Delta function accounts for the conservation of the energy of the electron/phonon system during the collision. The function $\phi(x,k,k')$ depends upon the nature of the scattering mechanism.

The coupled system of Boltzmann equation (1) and Poisson's equation (2,3,4) is non linear and induces collision damped plasma oscillations of high frequencies. In the practical situations, the doping profile function $n_D(x)$ has very steep gradients. The overall problem is a high dimension stiff problem.

3. THE NUMERICAL METHOD : GENERAL PRESENTATION

The most widely spread numerical method for solving the semiconductor Boltzmann equation is the Monte-Carlo method (cf. [6] and references therein), although other methods have been used in particular geometries (cf. Reed's method [18]) or for particular collision operators (see the recent method developped by Baranger [9] or Kuivalainen and Lindberg [19]). The Monte-Carlo method is quite noisy and thus, the affordable number of particles is generally not sufficient to get a sharp resolution of the distribution function, by statistical average. The moments of the distribution function such as the current and energy densities can be recovered with a sharp resolution, but only through time averages which make the description of the transient regimes uneasy. The new method and the new algorithms which we will describe in this paper are somehow derived from the Monte-Carlo method, but are intended to provide a more accurate numerical approximation.

The weighted particle method was first introduced by G.H. Cottet, S. Mas-Gallic and P.A. Raviart [20,21], for viscous perturbations of the incompressible Euler equation. Then the method was

adapted to the treatment of collision terms in kinetic equations [22]. Its first application to the semiconductor Boltzmann equation has been done in [1,2] and an error analysis relevant to this particular physical context has been performed in [3]. In the deterministic particle method, the particles move along the characteristics of the convective (first order differential) part of the equation, while the collision term is taken into account through the variation of the weights of the particles. The collision integral is evaluated by a discrete quadrature where the particles themselves play the role of quadrature points.

The weighted particle method is based upon the approximation of the distribution function by a sum of Delta measures :

$$f(x,k,t) \simeq f_h(x,k,t) = \sum_{i=1}^{N} \omega_i f_i(t) \delta(x - x_i(t)) \otimes \delta(k - k_i(t)) \qquad (9)$$

$x_i(t)$, $k_i(t)$, $f_i(t)$ and $\omega_i(t)$ are respectively the position, the wave-vector, the weight and the control volume of the particle i ; they evolve in time according to :

$$\frac{dx_i}{dt} = v(k_i) \qquad x_i(0) = x_i^0 \qquad (10\,a)$$

$$\frac{dk_i}{dt} = -\frac{q}{\hbar} E_i(t) \qquad k_i(0) = k_i^0 \qquad (10\,b)$$

$$\frac{df_i}{dt} = Q_i(t) \qquad f_i(0) = f_i^0 \qquad (11)$$

$$\omega_i(t) = \omega_i^0 \qquad (12)$$

where $E_i(t)$ and $Q_i(t)$ are respectively the approximations of the electric field and of the collision operator acting on the i-th particle. The initial x_i^0, k_i^0, f_i^0 and ω_i^0 are chosen so that :

$$f_0(x,k) \simeq \sum_{i=1}^{N} \omega_i^0 f_i^0 \delta(x - x_i^0) \otimes \delta(k - k_i^0) \qquad (13)$$

The time differential system (10,11) can be solved by any classical scheme. In our computations, we used either the order 2 Adams-Bashforth scheme or a mixed Adams-Bashforth and backwards Euler scheme.

To define $Q_i(t)$, we introduce a cut-off function $\zeta_\alpha(x)$ such that :

$$\zeta_\alpha(x) = \frac{1}{\alpha^3}\,\zeta\left(\frac{x}{\alpha}\right) \quad ; \quad \zeta(x) = \zeta(-x) \quad ; \quad \int \zeta(x)\,dx = 1 \qquad (14)$$

where ζ is a compactly supported function. We delocalize the integral operator Q(f) in position using this ζ_α function, and then we perform a numerical quadrature using the particles as quadrature points. Therefore we let, omitting the t-dependence of x_i, k_i and f_i (see [4] for details) :

$$Q_i(t) = \sum_{j=1}^{N} \left[S_\beta(x_j,k_j,k_i)\,f_j(1-f_i) - S_\beta(x_i,k_i,k_j)\,f_i(1-f_j)\right] \zeta_\alpha(x_j - x_i)\,\omega_j \qquad (15)$$

The transition rates S_β are smooth regularizations of the transition rates S given by (8) ; they are written :

$$S_\beta(x,k,k') = \sum \phi(x,k,k')\,\xi_\beta(\varepsilon(k') - \varepsilon(k) \pm \hbar\,\omega_p) \qquad (16)$$

where ξ_β is a compactly supported function defined in a similar way as ζ_α . The computation of $E_i(t)$ will be detailed in section 5.

4. THE HOMOGENEOUS CASE

Throughout this paragraph, we will suppose that the electric field is an external electric field denoted by E which is uniform (in space) and constant (in time). We assume that the problem is homogeneous in space and that $n_D(x) = n_D$ is independent of x. Thus, the dependence upon x vanishes as well as the coupling with the Poisson equation. We assume the axisymmetry of the wave-vectors with respect to the field axis, and describe a wave-vector k by its two components k_1 and k_2, respectively parallel and perpendicular to the field. Equations (9,10 b,11,12,13,15) describe the weighted particle method in this particular case.

Figure 1 shows a comparison between our method and the Monte-Carlo method [17], for a homogeneous sample of Gallium Arsenide doped at $n_D = 5.10^{15}$ impurities per cm^3 , at temperature T = 77 K, imbedded in a constant electric field E = 10 kV/cm. The band diagram of Gallium Arsenide was described by a standard three valley model, and the integral operator was of the form (7,8), with about 40 different scattering mechanisms [2]. Figure 1 displays the mean velocity, mean energy and density versus time and the results show a very good agreement between our results and the Monte-Carlo method. Figure 2 displays the three dimensional views of the distribution function, during its time evolution. With the homogeneous model we can compute the stationary characteristics

of Gallium Arsenide [14]. Figure 3 shows the stationary mean velocity and mean energy versus electric field and figure 4, the momentum and energy relaxation times versus energy. These informations are useful for hydrodynamic simulations. Finally, we mention that the homogeneous field model can also be used to describe the bidimensional transport of electrons parallel to a heterojunction interface [23].

5. AN INHOMOGENEOUS CASE

In this paragraph, we concentrate on a one-dimensional inhomogeneous case, for which the solution of the coupled Boltzmann-Poisson system is necessary. More details can be found in [4,5]. In this inhomogeneous case, the model consists of the Boltzmann equation (1) and of the Poisson equation (2,3,4). The mutual Coulomb interaction between charged particles is fully taken into account. The electric field has a constant direction, and we use an axisymmetric geometry relative to its axis, like in the homogeneous case. We used the following doping profile, already used by Baranger in [9] :

$$\begin{aligned} n_D(x) &= N^+ \quad \text{for } 0 \le x \le x_1 \\ &= N^- \quad \text{for } x_1 < x < x_2 \\ &= N^+ \quad \text{for } x_2 \le x \le L \end{aligned}$$

with $N^- = 2.10^{15}$ cm^{-3}, $N^+ = 10^{18}$ cm^{-3}, $L = 1{,}2\ \mu$m, $x_1 = 0{,}4\ \mu$m and $x_2 = 0{,}8\ \mu$m. The behaviour of this N^+-N-N^+ structure is dominated by the dynamics of the carriers in the N^- region. However, a sharp numerical description of this region is not easy because, due to the large inhomogeneities ($N^+/N^- = 500$), the numerical errors on the electron density in the N^+ region are of the same order of magnitude as the density itself in the N^- region. Moreover, if the trajectories of the particles are not accurately solved, the fast particles may jump over the peaks of the electric field which stand near the N^+-N and N-N^+ junctions, instead of "seeing" them.

Again, equations (9) to (15) describe the weighted particle method. We only need to detail the computation of $E_i(t)$ (10 b) using the Poisson equation. We considered two methods. First the classical "Particle In Cell" (PIC) method [24,25], in which one introduces a mesh of equally spaced points, and an interpolation function. An assignment procedure using this function gives the approximation of the electronic density at the grid points. The Poisson equation is then solved with a finite difference scheme, to get the approximation of the electric field on the grid poi. . he field is finally interpolated at the location of the i-th particle, to obtain the value of $E_i(t)$.

The second method uses the Green's functions of Poisson's equation and relies on an exact

representation of the mutual Coulomb interaction between the particles. Indeed, an integration of (9) with respect to the wave-vector gives a particle representation of the electronic density :

$$n_h(x,t) = \sum_{j=1}^{N} \omega_j f_j(t) \delta(x - x_j(t))$$

The "exact" electric field, solution of Poisson's equation (2,3) for this approximate density can be computed at the location of the i-th particle by means of the Green's kernel K(x,y) of Poisson's equation with periodic boundary conditions :

$$\frac{\partial K}{\partial x}(x,y) = \delta(x-y) - \frac{1}{L} \;\; ; \;\; K(0,y) = K(L,y) \;\; ; \;\; \int_0^L K(x,y)\, dy = 0 \;\; .$$

Therefore we let [5] :

$$E_i(t) = \frac{q}{\varepsilon_0 \varepsilon_r} \left[\int_0^L K(x_i(t),y)\, n_D(y)\, dy - \sum_{j=1}^{N} \omega_j f_j(t)\, K(x_i(t),x_j(t)) \right] - \frac{\phi_L - \phi_0}{L}$$

where ϕ_0 and ϕ_L are the prescribed boundary conditions on the potential at x=0 and x=L.

We present simulations of a GaAs $N^+N\text{-}N^+$ structure at T = 300 K, and we compare our results with those obtained by Baranger [9]. In our simulations, we used a two-valley model for GaAs, and the physical description of the interactions as given by formula (8) while Baranger's model involves a simplified 2-valley relaxation time model for the description of the collisions.

Figure 5 shows the electronic density profiles at t = 1 ps : the qualitative agreement with Baranger's results is satisfactory, although our results are more noisy. Indeed, the effect of including a second valley is the same for both results : the total electronic density increases in the $N\text{-}N^+$ junction area, where the Γ-L transfer occurs (i.e. where an important fraction of electrons belonging to the Γ valley reach high enough energies to transfer into the L valley by means of the collision processes). The potential slightly decreases in the N region (Figure 6), and thus the field curvature is changed near the $N\text{-}N^+$ junction (Figure 7). The mean total current (which is a function of time only) has an oscillatory transient regime, and then converges towards the value found by Baranger in [9] (Figure 8). The total mean velocity is lower in a two-valley model than in a single-valley model, and it reaches its maximum further from the $N\text{-}N^+$ junction (Figure 9). Figures 10 presents a snapshot of the distribution function in the L-valley at t = 1 ps as a function of position and energy, showing that the Γ-L transfer clearly occurs the $N\text{-}N^+$ junction. We refer the reader to [9] for a detailed

discussion of the involved physical phenomena.

6. CONCLUSION

We have presented a new numerical method for the Boltzmann equation of semiconductors, based on a deterministic treatment of the collision opertor which may provide an accurate description of the physics of electron transport. It proved to be very satisfactory in the homogeneous model and to provide useful informations for more macroscopic models. In the inhomogeneous case, the comparisons show that the method is reliable and is able to give an accurate picture of the physical phenomena occuring in submicron structures.

REFERENCES

[1] NICLOT B., DEGOND P., POUPAUD F., J. Comput. Phys. 78 (1988), p. 313.

[2] DEGOND P., MUSTIELES F.J., *"Le logiciel SPADES ; documentation scientifique"*, Rapport interne n° 196, CMAP, Ecole Polytechnique (1989).

[3] DEGOND P., NICLOT B., Numer. Math. 55 (1989), p.599.

[4] DEGOND P., GUYOT-DELAURENS F., *"Particle simulations of the semiconductor Boltzmann Equation for one-dimensional inhomogeneous structures"* ; to appear in J. Comput. Phys.

[5] DELAURENS F., MUSTIELES F.J., *"Le logiciel SPADES-2 ; Documentation Scientifique"*, contrat CNET n° 878B087 LAB/ICM/TOH.

[6] REGGIANI L. (ed.), *"Hot electron transport in semiconductors"*, Topics in Applied Physics, Springer, Berlin (1985).

[7] SELBERHERR S., *"Analysis and Simulation of Semiconductor Devices"*, Springer Verlag, Wien, New-York (1984).
MARKOWICH P.A., *"The Stationary Semiconductor Device Equations"*, Springer Verlag, Wien, New-York (1986).

[8] SHUR M.S., EASTMAN L.F., IEEE Trans. Electron Device ED 26 (1979) 1677.

[9] BARANGER H.U., WILKINS J.W., Phys. Rev. B 30 (1984), p. 7349 ; Phys. Rev. B 36 (1987), p.1487.
BARANGER H.U., *"Ballistic Transport in a Submicron Semiconducting Structure : a Boltzmann Equation Approach"*, PhD Thesis, Cornel University (1986).

[10] RUDAN M., ODEH F., Compel 5 (1986), p. 149.

[11] COOK R.K., FREY J., Compel 1 (1982), p. 65.

[12] COOK R.K., FREY J., IEEE Trans. Electron Device ED 29 (1982), p. 1970.

[13] BLOETEKJAER K., IEEE Trans. Electron Device ED 17 (1970), p. 38.

[14] MUSTIELES F.J., *"Le logiciel SPADESNEW"*, Contrat CNET n° 878B087, LAB/ICM/TOH; Manuscript, CMAP, Ecole Polytechnique (1989).

[15] RODE D.L. in *"Semiconductors and Semimetals"* 10, Academic Press, New-York (1975).

[16] CONWELL, E.M., in Solid State Physics 9, Academic Press, New-York (1967).

[17] HESTO P., *"Simulation Monte-Carlo du transport non stationnaire dans les dispositifs submicroniques ; influence du phénomène balistique dans GaAs à 77 K"*, Thèse, Université d'Orsay (1984).

[18] REES H.D., J. Phys. Chem. Solids 30 (1969), p. 643.

[19] KUIVALAINEN P., LINDBERG K., in *"High Speed Electronics"*, edited by B. Küllback and H. Beneking, Springer Verlag, New-York (1986) p. 40.

[20] COTTET G.H., MAS-GALLIC S., *"A particle method to solve transport-diffusion equations. part 1 : the linear case"*, Numer. Math. to appear.
MAS-GALLIC S. and RAVIART P.A., Numer. Math. 51 321 (1987).

[21] DEGOND P. and MAS-GALLIC S., Math. Comput. 53 (1989), p. 485 and 53 (1989), p. 509.

[22] MAS-GALLIC S., Transp. Theory Stat. Phys. 16 (1987), p. 855.
MAS-GALLIC S., POUPAUD F., Transp. Theory Stat. Phys. 17 (1988) p. 311.

[23] DEGOND P., GUYOT-DELAURENS F., MUSTIELES F.J., NIER F., *"Particle simulation of bidimensional electron transport parallel to a heterojunction interface"*, to appear in Compel.
DEGOND P., GUYOT-DELAURENS F., MUSTIELES F.J., NIER F., *"Simulation particulaire du transport bidimensionnel d'électrons parallèle à l'interface d'une hétérojonction"*, Rapport interne n° 189, CMAP, Ecole Polytechnique (1989).

[24] HOCKNEY R.W., EASTWOOD J.W., *"Computer Simulations using Particles"*, Mc Graw Hill, New-York (1981).

[25] BIRDSALL C.K., LANGDON, *"Plasma physics via Computer Simulations"*, Mc Graw Hill, New-York (1985).

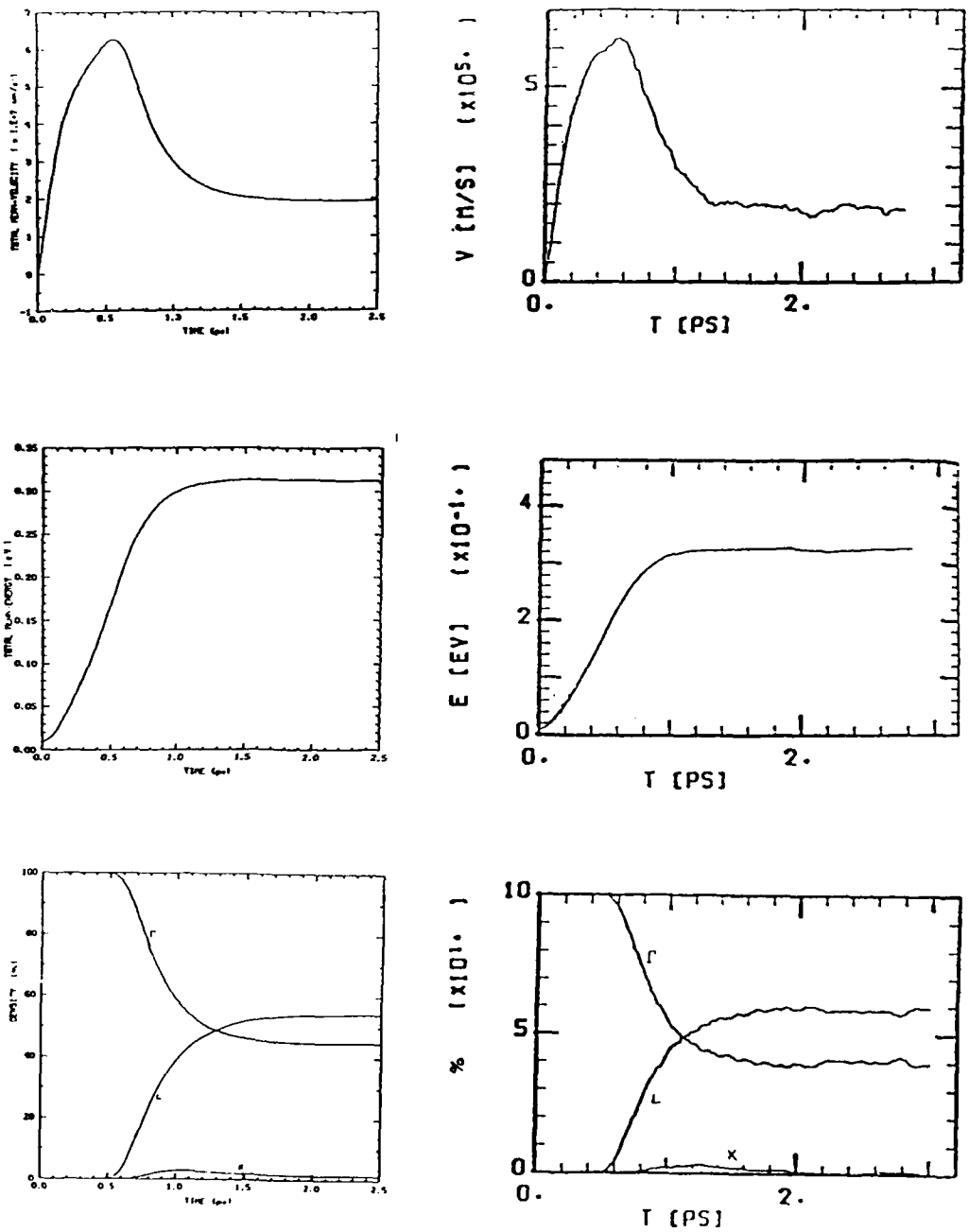


Figure 1 : Top : total mean velocity ($\times 10^7$ cm/s)
Middle : total mean energy (eV)
Bottom : Electron population in each valley (%)
as a function of the time (ps). The left curves are obtained by the deterministic method ; the right ones by a Monte-Carlo method [17]

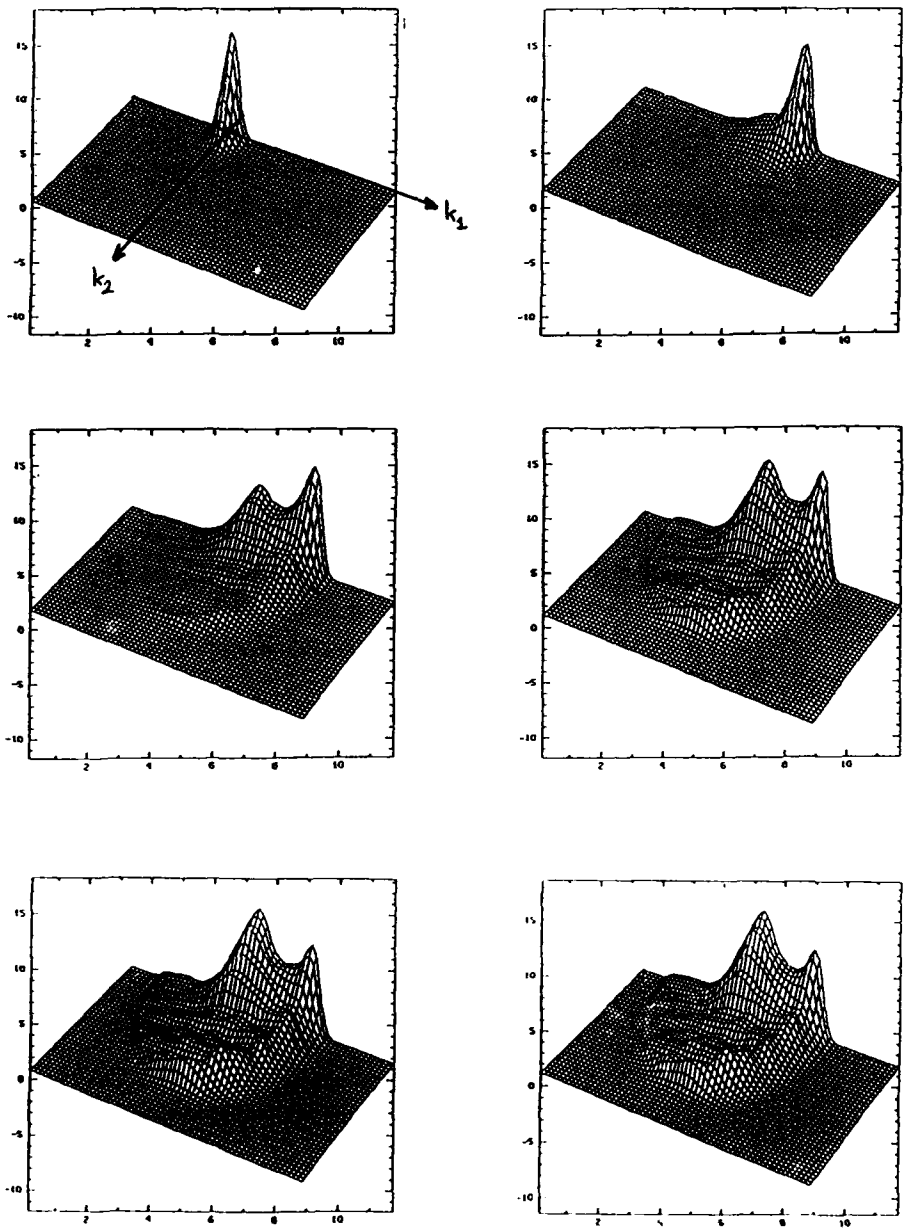


Figure 2 : Distribution function in the Γ valley as a function of the parallel (k_1) and perpendicular (k_2) components of the wave-vector k. The six graphs correpond to the times (from left to right and from top to bottom) : t = 0 , 0.5 , 1 , 1.5 , 2 , 2.5 ps .

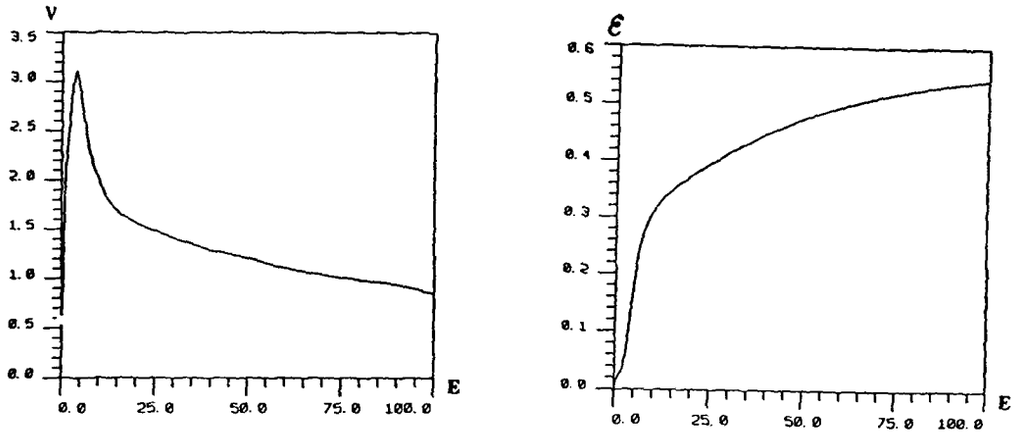


Figure 3 : Left : stationary mean velocity ($\times 10^7$ cm/s)
Right : stationary mean energy (eV)
as a function of the electric field (kV/cm)

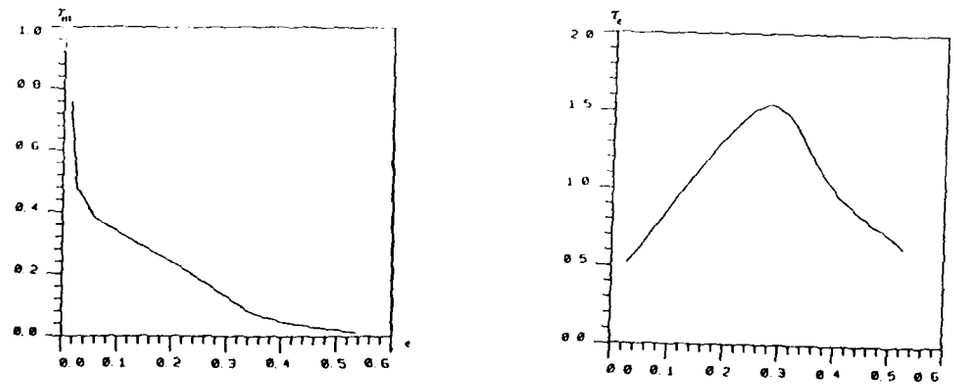


Figure 4 : Left : momentum relaxation time (ps)
Right : energy relaxation time (ps)
as a function of the energy (eV)

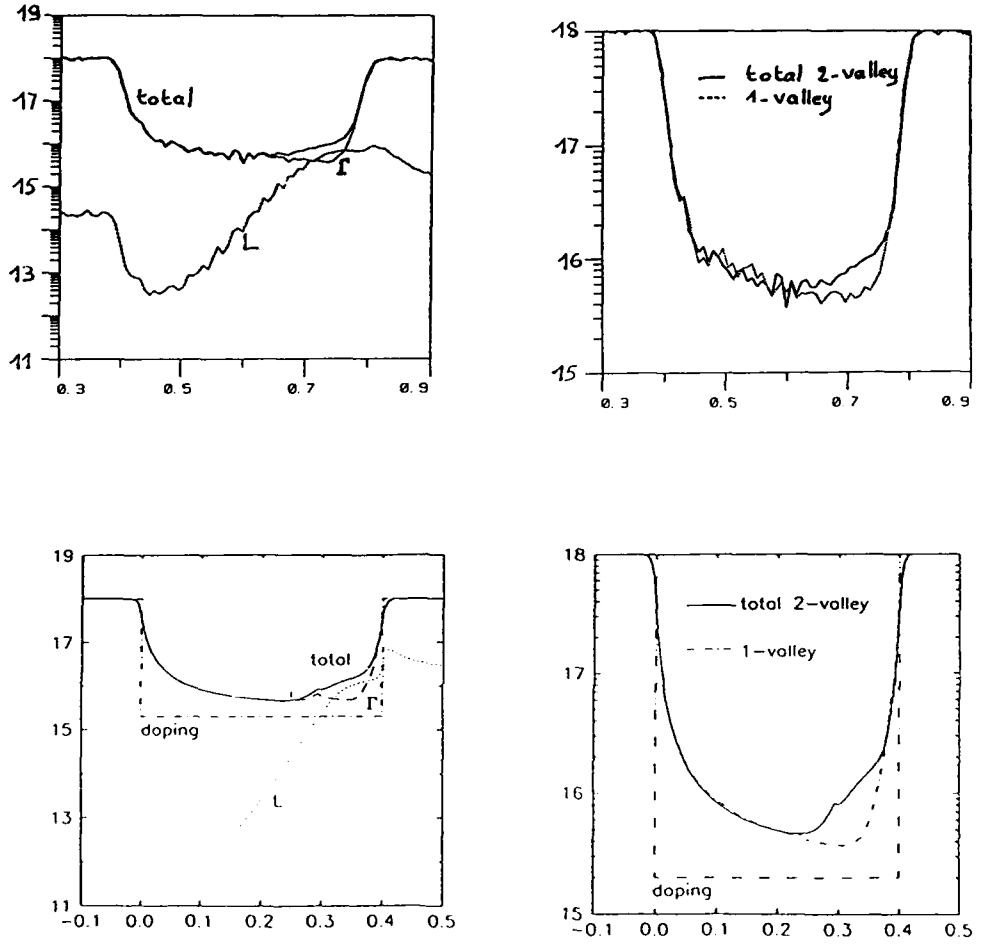


Figure 5 : Left : Γ valley, L valley and total electron densites (cm^{-3} , log scale) at the stationary state

Right : total electron densities (cm^{-3} , log scale) at stationary state for a 2 valley model compared with a 1 valley model

as a function of the distance in the structure (μm). The top curves are obtained by the deterministic method ; the bottom ones by Baranger's model [9].

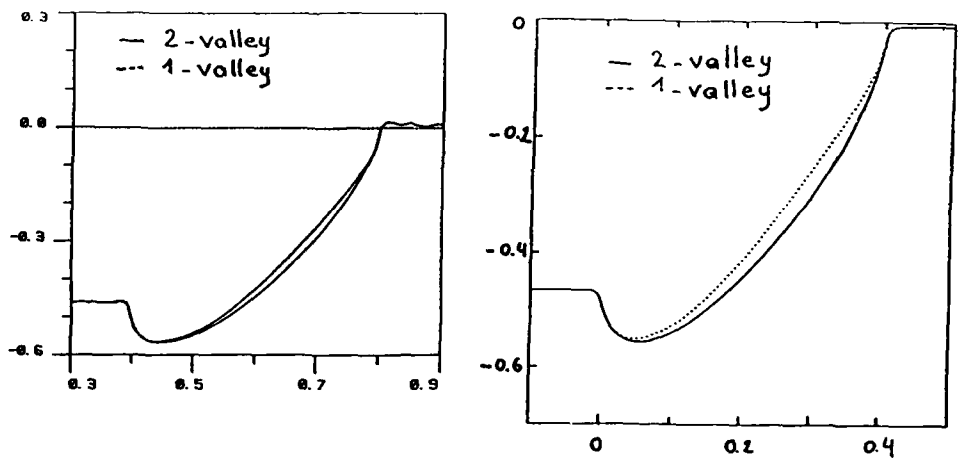


Figure 6 : Electric potential (V) at stationary state, as a function of the distance in the structure (µm). Comparison between a 2 valley and a 1 valley model. The left curves are obtained by the deterministic method ; the right ones by Baranger's model [9].

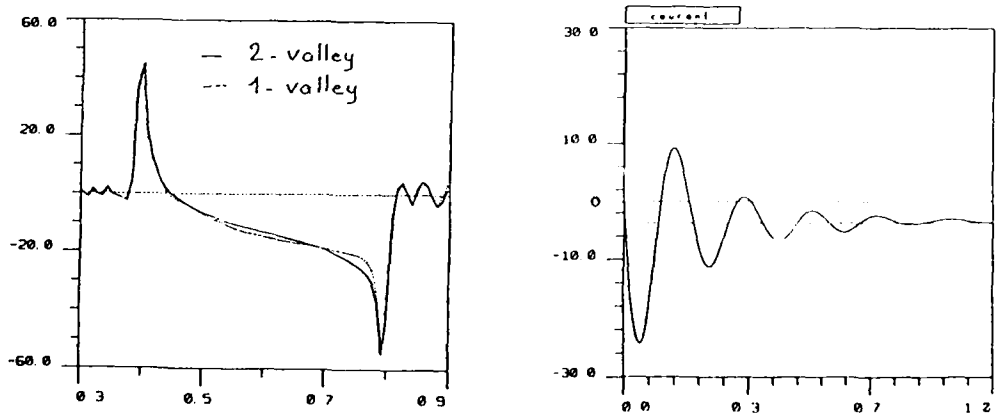


Figure 7 : Electric field (kV/cm) at the stationary state, as a function of the distance in the structure (µm). Comparison between a 2 valley and a 1 valley model.

Figure 8 : Total current (× 10^4 A/cm^2) as a function of the time (ps).
solid line : results by the determinstic method
dotted line : Baranger's value of the stationary current.[9]

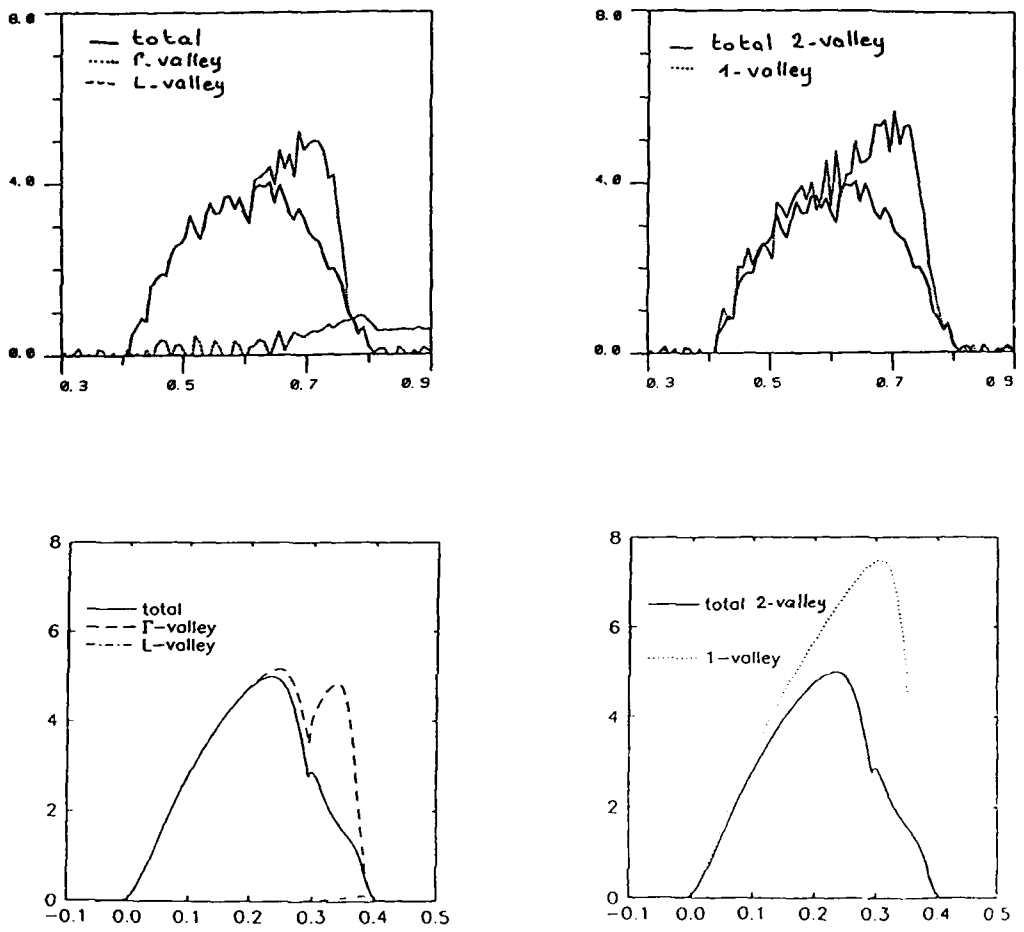


Figure 9 : Left : Γ valley, L valley and total mean velocities ($\times 10^7$ cm/s) at the stationary state

Right : total mean velocities ($\times 10^7$ cm/s) at the stationary state for a 2 valley model compared with a 1 valley model

as a function of the distance in the structure (μm). The top curves are obtained by the deterministic method ; the bottom ones by Baranger's model [9].

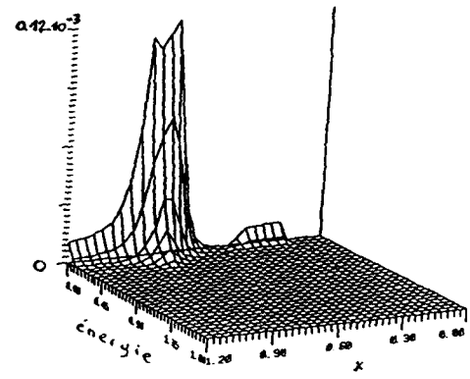


Figure 10 : Distribution function in the L valley as a function of the distance in the structure (μm ; increasing from right to left) and of the energy (eV).

NUMERICAL SIMULATION OF RAREFRIED GAS FLOWS

Hans Babovsky
AGTM, Universität Kaiserslautern

This talk is concerned with the progress of the Arbeitsgruppe Technomathematik (AGTM) in developing a simulation code for rarefied gas flows and in performing 2d and 3d calculations for realistic situations. This work has been done in connection with a project in the R & D program for the European space shuttle Hermes.

1. Theoretical aspects

The reference point in the development of our simulation code is an appropriate Boltzmann equation for the description of rarefied gas flows. Our intention is to yield approximate solutions to this equation.

The Boltzmann equation for a density function $f = f(t,x,v)$ in phase space has the form

$$Df = J(f,f)$$

with the free streaming operator

$$Df = \left(\frac{\partial}{\partial t} + v\cdot\nabla_x\right)f$$

and the collision integral $J(f,f)$. The idea underlying our scheme is to approximate f by a system of N moving particles $(x_i(t),v_i(t))_{i\leq N}$. Precisely, the problem is to invent an (artificial) "dynamics" for the N-point system in such a way that the simulated solution keeps close to the exact solution of the Boltzmann equation.
This dynamics is constructed by decoupling free flow and collisions. While the first step is performed easily by a translation of the x-coordinates:

$$(x_i,v_i) \longrightarrow (x_i+\Delta t\cdot v_i,v_i)\ ,$$

the second step imitating collisions through discontinuous changes of velocities is crucial for an efficient simulation scheme. In our code, this is done by choosing triples (v_i, w_i, b_i) consisting of velocity pairs (v_i, w_i) and of (collision) parameters b_i and by applying an appropriate transformation ψ to calculate the new velocities (v_i^{new}, w_i^{new}) (compare [1]):

$$v_i^{new} = \psi(v_i, w_i, b_i) ,$$

$$w_i^{new} = \psi(w_i, v_i, b_i) .$$

In order to be consistent with the Boltzmann equation, choosing the triples requires certain conditions to be satisfied (see [2]). There are several possibilities: On one hand the purely random game called "Monte Carlo version", on the other hand so-called "Low Discrepancy versions" reducing fluctuations. (Our code presently applied uses random numbers as well as Low Discrepancy sequences. A detailed description of this code will be provided in [4].) The use of these schemes is justified by a Law of Large Numbers stating that the simulated solutions are good approximations to the exact solutions if the particle number N is large enough. (A proof of this Law of Large Numbers goes along the lines of the proof in [2], [3].)

However, in practical calculations one is forced to use particle systems much too small to be in the (theoretical) domain of validity of the Law of Large Numbers. This was motivation for us to study the behaviour of small particle systems. Here, ergodic theory yiel strong mathematical results [5]. We could

- show that time averages of small particle systems do not represent solutions of the Boltzmann equation but are affected with a systematic error which is expected to be the smaller, the bigger the particle system is;
- for simple cases provide an exact formula for the systematic error which allows to perform corrections to the simulated solutions;
- prove that boundary conditions - for example at artificial boundaries limiting the region of calculation - can have a strong influence on the systematic error; thus, well perform-

ing boundary conditions at non-physical boundaries may have a strong impact on the validity of the scheme.

2. Aspects of modelling

Aspects of modelling play an important role when simulating gas flows in realistic situations. Aspects of particular interest are

- boundary conditions
- interior energies
- gas mixtures
- chemical reactions.

Boundary conditions

They have to be modelled in such a way that coefficients of interest are obtained correctly. For flows around bodies, of particular interest are for example drag, shear stress and lift coefficients. As experiments show, several of these depend in a very sensitive way on the type of boundary conditions used. As a consequence, a simulation code has to be flexible enough to include various classes of boundary conditions in order to properly model the situation of interest.
We have compared different classes of boundary conditions including

- specular reflection
- diffuse reflection with accomodation coefficient
- the Cercignani-Lampis model.

To this end we have performed 3d calculations of flows around flat discs with different angles of attack α, and have compared these results with experimental results obtained by Legge [6]. Some data for Argon are presented in figures 1 and 2 where we compare the pressure drag coefficient versus the accomodation coefficient and versus the Knudsen number. In figure 2, the experimental data are represented by the points, while the lines show the simulated results. Similar calculations have been done for the gas Nitrogen.

Interior energies

Here, we are mostly interested in imitating rotational energies which becomes necessary when simulating polyatomic gases. The

most frequently used model on the market is the Larsen-Borgnakke model. We have analyzed its mathematical structure and extended it to a larger class of models. Furthermore, we have implemented Kuscer's VHS model which from a physical point of view seems much more reliable than the models described above. We have considered also certain geometrical models like loaded sphere models.

Investigations of shock structures have shown significant dependence of the results on the model. (Results are described in papers to be published soon.) However, in realistic situations the choice of the most appropriate model may fail due to the lack of sufficient experimental data.

Gas mixtures

Binary gas mixtures with approximately equal densities are readily included in our code. Big differences of densities, however, required the introduction of weighted particles which forced us to slightly change the collision model.
This has now also been successfully implemented. As an example, figure 3 shows shock profiles in a two component gas.

Chemical reactions

For the modelling of chemical reactions much theoretical work is still necessary. There are several models on the market but none of them is very satisfactory from a theoretical point of view. From a practical point of view, further numerical investiations and experiments are necessary to test the relevance of these models.

3. Computational aspects

Presently, we are mainly concerned with the simulation of 2d and 3d calculations of flows around bodies of different shapes. For example, we have performed 2d calculations around double ellipses and 3d calculations around flat discs with different angles of attack at Mach 15.6 (mainly in order to compare boundary models with experimental data) and around a delta wing with Knudsen numbers down to 0.01.

For our calculations, we used a rectangular grid structure (which allows to easily refind particles after free flow) with an adaptive grid refinement. This cell system turned out to be efficient and to reduce the computational effort quite well.

The complete code (including all the features described above) and has been proven to be approximately five times faster than other simulation methods. For further details, see [4].

All calculations have been performed on the VP 100 in Kaiserslautern and the VP 400 in Karlsruhe.

References

[1] H. Babovsky, F. Gropengießer, H. Neunzert, J. Struckmeier, B. Wiesen: "Low Discrepancy Methods for the Boltzmann Equation", Proc. 16th Int. Symp. on Rarefied Gas Dynamics, Vol 118, 85-99, Washington D.C. (1989)

[2] H. Babovsky: "A Convergence Proof for Nanbu's Boltzmann Simulation Scheme", Eur. J. Mech. B/Fluids, 8, 44-51 (1989)

[3] H. Babovsky, R. Illner: "A Convergence Proof for Nanbu's Simulation Method for the Full Boltzmann Equation", SIAM J. Numer. Anal. 26, 45-65 (1989)

[4] F. Gropengießer, H. Neunzert, J. Struckmeier, B. Wiesen, to be published

[5] H. Babovsky, to be published

[6] H. Legge: "Force and Heat Transfer Measurements on a Disc at 45°-90° Angle of Attack in Free Jet Flow Using AR, HE, N_2, H_2 as Test Gases", IB 222-89 A 07, research report, DLR Göttingen (1989)

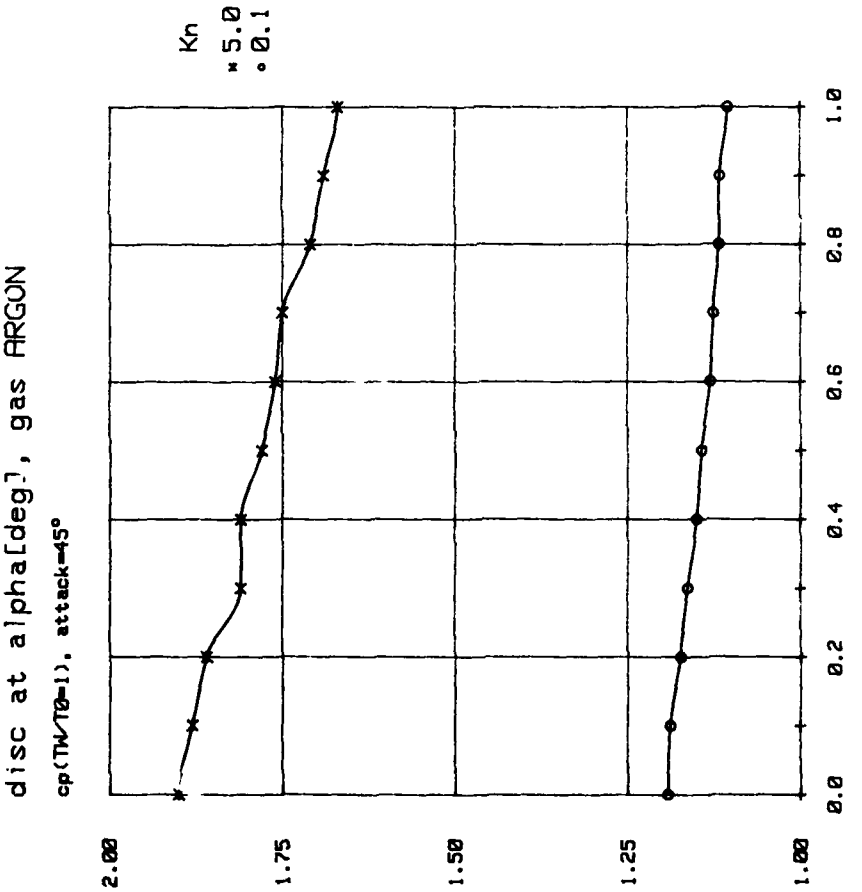


figure 1 : Pressure drag coefficient vs. accomodation coefficient

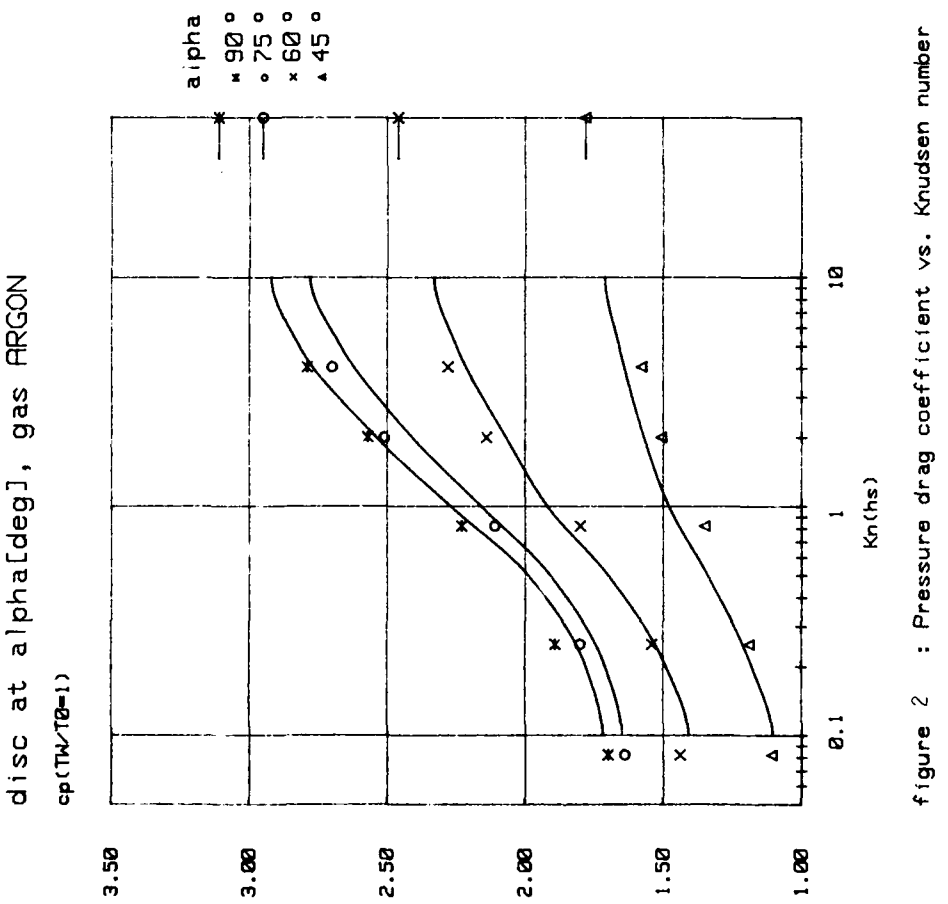


figure 2 : Pressure drag coefficient vs. Knudsen number

a) n1/n2=.8/.2

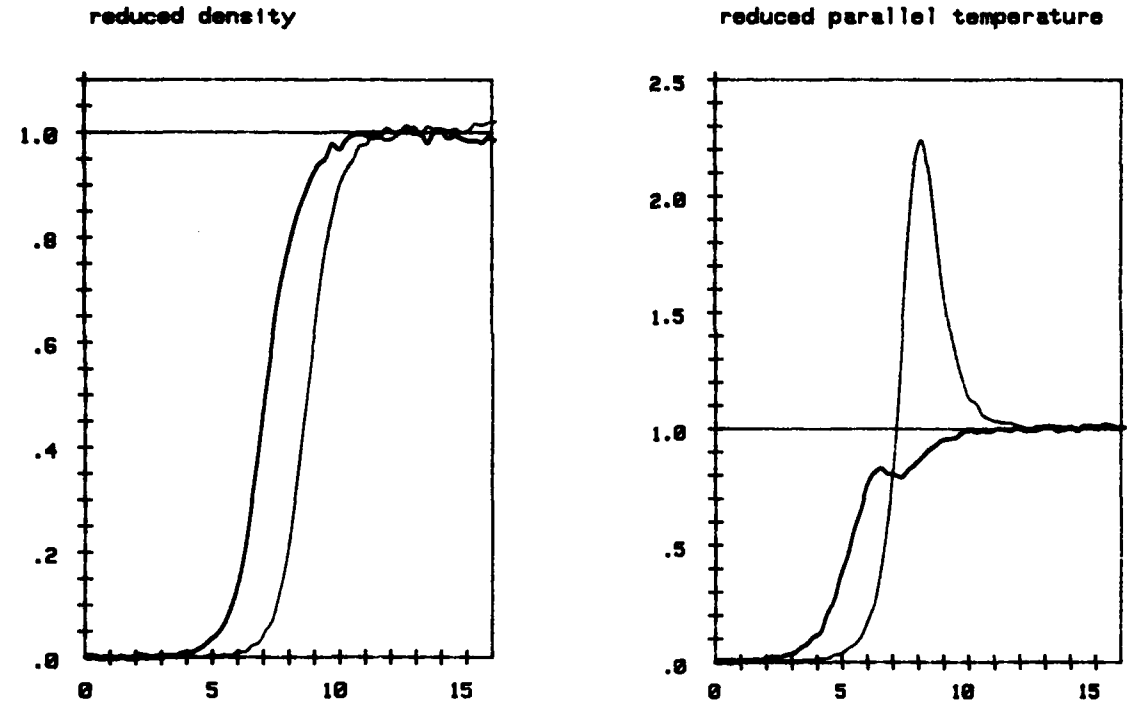


figure 3a : Profiles of the reduced densities and parallel kinetic temperatures for a two component gas with mass ratio 1/10

b) n1/n2=.95/.05

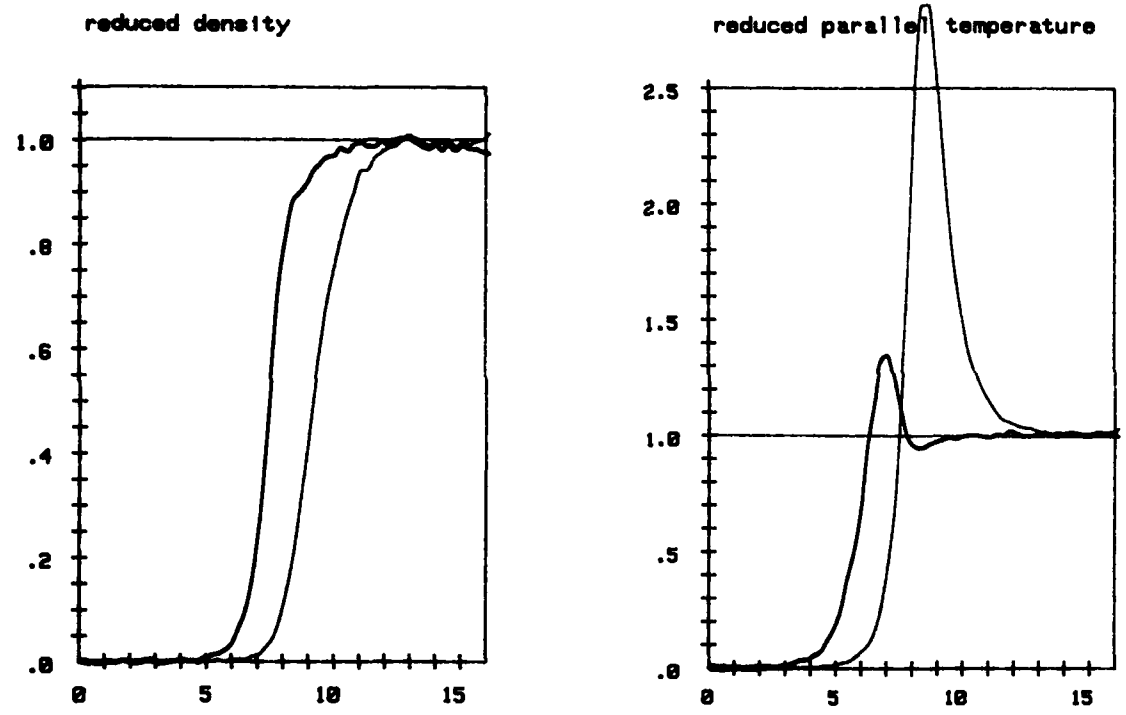


figure 3b : Profiles of the reduced densities and parallel kinetic temperatures for a two component gas with mass ratio 1/10

THE PERIODIC BOLTZMANN SEMICONDUCTOR EQUATION.

J.F. Bourgat [1], R.Glowinski [2], P. Le Tallec [3], J.F. Palmier [4]

========

Extended abstract

This paper is concerned with the numerical solution of the Boltzmann semiconductor equation. In the case of a problem homogeneous in space, the electronic function is solution of the integro-differential equation

$$\frac{\partial f}{\partial t} - \frac{q}{\hbar} E \cdot \nabla_k f = Q(f) \quad \text{in} \quad \mathbb{R}^3$$
$$\int f(k)dk = 1.$$

Here, Q represents the linear scattering operator given by

$$Q(f) = \int_{\mathbb{R}^3} (s(k';k)f(k') - s(k;k')f(k))dk'.$$

Our main objective is the development of a fast deterministic solution procedure for the numerical computation of the steady state solutions of the above problem.

In this framework, our discretization strategy is based on an upwind finite difference approximation of the gradient terms and on a deterministic conservative calculation of the integral operators. The resulting algebraic system is then solved by a least squares methodology which reduces the system to a quadratic minimization problem to be solved by a standard conjugate gradient algorithm.

In the axisymmetric steady case problem with periodic boundary conditions, which models superlattices, this methodology gives nice results for all values of the electric fields. These results are confirmed by an unsteady analysis which, although more expensive numerically, gives additional information on the relaxation behavior of the distribution function towards its steady state.

[1] INRIA, Domaine de Voluceau, 78153 Le Chesnay Cedex, France

[2] Dept. of Mathematics, University of Houston, 4800 Calhoun Road, Houston, Texas 77004 USA

[3] Université Paris Dauphine, Place du Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, France

[4] CNET, Laboratoire de Bagneux, 196 av. Ravera, 92220 Bagneux, France

APPLICATIONS DES SUPERCALCULATEURS

SUPERCOMPUTER APPLICATIONS

INCOMPRESSIBLE FLOW COMPUTATIONS USING VARIOUS VELOCITY-PRESSURE ELEMENTS+

T.E. Tezduyar, D.K. Ganjoo, and R. Shih
Department of Aerospace Engineering and Mechanics
and Minnesota Supercomputer Institute
University of Minnesota
Minneapolis, MN 55455

Abstract

A comparative investigation, based on a series of numerical tests, of various velocity-pressure elements used for incompressible flow computations is presented. These elements are implemented in conjunction with the one-step and multi-step temporal integration of unsteady Navier-Stokes equations. The group of elements studied includes the element with a Petrov-Galerkin stabilization that allows equal-order (bilinear) interpolation functions for velocity and pressure. The test cases chosen are the standing vortex problem, the lid-driven cavity flow, and flow past a circular cylinder.

1. Introduction

In this paper we conduct a comparative study of various finite elements used for incompressible flow computations based on the velocity-pressure formulation of unsteady Navier-Stokes equations. The elements covered by this study are **Q1P0** (bilinear velocity/ discontinuous piecewise constant pressure), **Q2Q1** (biquadratic velocity/ bilinear pressure), **pQ2Q1** ("pseudo" biquadratic velocity/ bilinear pressure), **Q2P1#** (biquadratic velocity/ discontinuous piecewise bilinear pressure), **Q2P1Δ** (biquadratic velocity/ discontinuous piecewise linear pressure), **pQ2P1Δ** ("pseudo" biquadratic velocity/ discontinuous piecewise linear pressure), and **Q1Q1/ε** (bilinear velocity/ bilinear pressure with Petrov-Galerkin stabilizer). We implemented these elements by generalizing the one-step formulation developed by Brooks and Hughes [1] for the **Q1P0** element, and also by using the multi-step formulations presented in Tezduyar, Liou, and Ganjoo [2]. In all these formulations we use the streamline-upwind/Petrov-Galerkin (SUPG) method [1,2] to prevent the spurious oscillations that might appear in the presence of dominant advective terms.

In the one-step formulation the SUPG supplement to the weighting function is applied to all the terms in the momentum equation. For the element with equal-order (bilinear) interpolation functions for velocity and pressure, in addition to the SUPG supplement, another Petrov-Galerkin supplement is added to the weighting function to stabilize the element. We will refer to this supplement as the PSPG ("pressure-stabilizing" Petrov-Galerkin) supplement. The PSPG supplement is defined by utilizing the ideas introduced by Hughes and Franca [3]. The Petrov-Galerkin formulation introduced in [3] is capable of accommodating arbitrary orders of interpolation for the (steady-state) solution of Stokes problem.

+ This research was sponsored by NSF under grant MSM-8796352.

The T6 formulation [2] is an extension of the T3 formulation [2]. The T3 formulation is a three-step method and starts out with a splitting scheme in which the pressure and the viscous terms are treated implicitly in the first and third steps while the advective terms are treated implicitly in the second step. This type of splitting is a special case of the kind found in the θ-scheme [4]. In the T6 formulation, each step of the T3 formulation is subdivided into two sub-steps to isolate the advective terms, and the SUPG supplement is applied only to the sub-steps involving the advective terms. A PSPG supplement is added to the weighting function in the "Stokes sub-steps" when using equal-order (bilinear) functions for velocity and pressure.

We consider three numerical tests: the standing vortex problem [5], the lid-driven cavity flow at Re=400, and flow past a circular cylinder at Re=100. The purpose of the standing vortex problem is to determine the level of numerical dissipation involved in a numerical solution technique. The lid-driven cavity problem involves singularities in the pressure field and, therefore, is regarded as a stringent test case. The cylinder problem has been studied by several researchers in the past (see for example [6]) and has become a benchmark problem [2].

2. The Governing Equations

Let Ω and (0,T) denote the spatial and temporal domains with x and t representing the coordinates associated with Ω and (0,T). We consider the following velocity-pressure formulation of the incompressible Navier-Stokes equations:

$$\rho\,(\partial \mathbf{u}/\partial t + \mathbf{u}\cdot\nabla\,\mathbf{u}) - \nabla\cdot\sigma = \mathbf{0} \qquad \text{on } \Omega\times(0,T)\,, \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0 \qquad \text{on } \Omega\times(0,T)\,, \tag{2}$$

where ρ and $\mathbf{u}$ are the density and velocity and σ is the stress tensor given as

$$\sigma = -p\,\mathbf{I} + 2\mu\,\varepsilon(\mathbf{u}) \tag{3}$$

with

$$\varepsilon(\mathbf{u}) = (\nabla\mathbf{u} + (\nabla\mathbf{u})^{T})/2\,. \tag{4}$$

Here p and μ represent the pressure and viscosity while $\mathbf{I}$ denotes the identity tensor. Both the Dirichlet and Neumann type boundary conditions are taken into account as shown below:

$$\mathbf{u} = \boldsymbol{g} \qquad \text{on } \Gamma_{\boldsymbol{g}}\,, \tag{5}$$

$$\mathbf{n}\cdot\sigma = \boldsymbol{h} \qquad \text{on } \Gamma_{\boldsymbol{h}}\,, \tag{6}$$

where $\Gamma_{\boldsymbol{g}}$ and $\Gamma_{\boldsymbol{h}}$ are complementary subsets of the boundary Γ.

3. Spatial and Temporal Discretizations

Let $\mathcal{E}$ denote the set of elements resulting from the finite element discretization of the computational domain Ω into subdomains Ω^e, e=1,2,...,n_{el}, where n_{el} is the number of elements. We associate to $\mathcal{E}$ the finite dimensional spaces H^{kh} and H^{mh}, where k and m represent the orders of the interpolation functions used. The trial and test function spaces are given as

$$S_{\mathbf{u}}^{h} = \{ \mathbf{u}^h \mid \mathbf{u}^h \in (H^{kh})^{n_{sd}} , \mathbf{u}^h = \boldsymbol{g}^h \quad \text{on } \Gamma_{\boldsymbol{g}} \} , \tag{7}$$

$$V_{\mathbf{u}}^{h} = \{ \mathbf{w}^h \mid \mathbf{w}^h \in (H^{kh})^{n_{sd}} , \mathbf{w}^h \doteq \mathbf{0} \quad \text{on } \Gamma_{\boldsymbol{g}} \} , \tag{8}$$

$$S_{p}^{h} = V_{p}^{h} = \{ q \mid q \in H^{mh} \} , \tag{9}$$

where n_{sd} is the number of space dimensions.

The one-step formulation employed in this work is essentially the same as the one used in [1] : find $\mathbf{u}^h \in S_{\mathbf{u}}^{h}$ and $p^h \in S_{p}^{h}$ such that

$$\begin{aligned}
&\int_{\Omega} \mathbf{w}^h \cdot \rho \, (\partial \mathbf{u}^h / \partial t + \mathbf{u}^h \cdot \nabla \, \mathbf{u}^h) \, d\Omega + \int_{\Omega} \varepsilon (\mathbf{w}^h) : \sigma^h \, d\Omega \\
&+ \sum_{e=1}^{n_{el}} \int_{\Omega^e} \delta^h \cdot [\rho \, (\partial \mathbf{u}^h / \partial t + \mathbf{u}^h \cdot \nabla \mathbf{u}^h) - \nabla \cdot \sigma^h] \, d\Omega \\
&+ \sum_{e=1}^{n_{el}} \int_{\Omega^e} \varepsilon^h \cdot [\rho \, (\partial \mathbf{u}^h / \partial t + \mathbf{u}^h \cdot \nabla \mathbf{u}^h) - \nabla \cdot \sigma^h] \, d\Omega \\
&+ \int_{\Omega} q^h \nabla \cdot \mathbf{u}^h \, d\Omega = \int_{\Gamma_{\boldsymbol{h}}} \mathbf{w}^h \cdot \boldsymbol{h}^h \, d\Gamma , \quad \forall \mathbf{w}^h \in V_{\mathbf{u}}^{h} , \quad \forall q^h \in V_{p}^{h} .
\end{aligned} \tag{10}$$

Here δ^h is the streamline-upwind / Petrov-Galerkin (SUPG) supplement to the weighting function $\mathbf{w}^h$ [7], and ε^h is another Petrov-Galerkin (PSPG) supplement to stabilize the element against pressure oscillations [3, 7]. We define ε^h as follows:

$$\varepsilon^h = E \, (1/\rho) \nabla q^h \tag{11}$$

with

$$E = c \, z_e \, h / (2 \, \| \mathbf{u}^* \|) , \tag{12}$$

where c is a free parameter (which we normally set to 1), h is the element length, $\mathbf{u}^*$ is a global velocity, and

$$z_\varepsilon = \begin{cases} \mathrm{Re}^*/3 & 0 \le \mathrm{Re}^* \le 3 \\ 1 & \mathrm{Re}^* \ge 3 \end{cases} . \tag{13}$$

The "Reynolds number" is defined as

$$\mathrm{Re}^* = \|\mathbf{u}^*\| h / (2\nu) \ , \tag{14}$$

where ν is the kinematic viscosity. For the limiting values of Re^* the expression for E takes the following forms:

$$E \to e\, h / (2\, \|\mathbf{u}^*\|) \quad \text{for } \mathrm{Re}^* \to \infty \ , \tag{15a}$$

$$E \to e\, h^2 / (12\nu) \quad \text{for } \mathrm{Re}^* \to 0 \ . \tag{15b}$$

The semi-discrete equations corresponding to equation (10) can be written as follows:

$$\tilde{\mathbf{M}}\, \mathbf{a} + \tilde{\mathbf{N}}(\mathbf{v}) + \tilde{\mathbf{K}}\, \mathbf{v} - \tilde{\mathbf{G}}\, \mathbf{p} = \tilde{\mathbf{F}} \ , \tag{16}$$

$$\mathbf{G}^T \mathbf{v} + \mathbf{M}_\varepsilon\, \mathbf{a} + \mathbf{N}_\varepsilon(\mathbf{v}) + \mathbf{K}_\varepsilon\, \mathbf{v} - \mathbf{G}_\varepsilon\, \mathbf{p} = \mathbf{E} + \mathbf{F}_\varepsilon \ , \tag{17}$$

where $\mathbf{v}$ is the vector of unknown nodal values of $\mathbf{u}^h$, $\mathbf{a}$ is the time derivative of $\mathbf{v}$, and $\mathbf{p}$ is the vector of nodal values of p^h. The matrices $\tilde{\mathbf{M}}$, $\tilde{\mathbf{N}}$, $\tilde{\mathbf{K}}$, and $\tilde{\mathbf{G}}$ are derived, respectively, from the time-dependent, advective, viscous, and pressure terms. The vector $\tilde{\mathbf{F}}$ is due to the Dirichlet and Neumann type boundary conditions (i.e. the g and h terms in equations (5) and (6)), whereas the vector $\mathbf{E}$ is due to the Dirichlet type boundary condition. All the arrays with a superposed tilde can be decomposed into their Galerkin and SUPG parts:

$$\tilde{\mathbf{M}} = \mathbf{M} + \mathbf{M}_\delta \ , \tag{18}$$

$$\tilde{\mathbf{N}} = \mathbf{N} + \mathbf{N}_\delta \ , \tag{19}$$

$$\tilde{\mathbf{K}} = \mathbf{K} + \mathbf{K}_\delta \ , \tag{20}$$

$$\tilde{\mathbf{G}} = \mathbf{G} + \mathbf{G}_\delta \ , \tag{21}$$

$$\tilde{\mathbf{F}} = \mathbf{F} + \mathbf{F}_\delta \ , \tag{22}$$

where the subscript δ identifies the SUPG contribution. Similarly the subscript ε identifies the PSPG contribution.

Remarks

1. The PSPG perturbation is employed only when using equal orders of interpolation for velocity and pressure.

2. The equation systems (16) and (17) can be solved implicitly. When using equal-orders of interpolation the unknowns are ordered node by node leading to a reasonable bandwidth; this is defined as the consistent (C) system. However when using unequal orders of interpolation for velocity and pressure, we reorder the unknowns in such a way that all unknown velocities appear before unknown pressures; we define this as the consistent-reordered (CR) system.

3. The equation systems (16) and (17) can also be solved by treating the velocity explicitly in the momentum equation. The way the **Q1P0** element is used in [2] leads to a symmetric coefficient matrix for pressure. The **Q1Q1/ε** element leads to a nonsymmetric coefficient matrix for pressure due to the presence of PSPG perturbation terms. However, if the PSPG terms which cause such nonsymmetry are neglected, then the coefficient matrix for the pressure can be "symmetrized". All explicit one-step computations presented in this paper are based on such a symmetrization, and the results are obtained with 2 passes.

The T6 formulation [2] is described as follows:

find $\tilde{\mathbf{u}}^h_{n+\theta} \in (S^h_{\mathbf{u}})_{n+\theta}$ such that

$$\int_\Omega \mathbf{w}^h \cdot \rho \left((\tilde{\mathbf{u}}^h_{n+\theta} - \mathbf{u}^h_n) / (\theta \Delta t) + \mathbf{u}^h_n \cdot \nabla \mathbf{u}^h_n \right) d\Omega$$
$$+ \sum_{e=1}^{n_{el}} \int_{\Omega^e} \delta^h \cdot [\rho \left((\tilde{\mathbf{u}}^h_{n+\theta} - \mathbf{u}^h_n) / (\theta \Delta t) \right.$$
$$\left. + \mathbf{u}^h_n \cdot \nabla \mathbf{u}^h_n \right)] d\Omega = 0 , \qquad \forall \mathbf{w}^h \in V^h_{\mathbf{u}} ; \tag{23}$$

find $\mathbf{u}^h_{n+\theta} \in (S^h_{\mathbf{u}})_{n+\theta}$ and $p^h_{n+\theta} \in S^h_p$ such that

$$\int_\Omega \mathbf{w}^h \cdot \rho \left(\mathbf{u}^h_{n+\theta} - \tilde{\mathbf{u}}^h_{n+\theta} \right) / (\theta \Delta t) \, d\Omega + \int_\Omega \varepsilon (\mathbf{w}^h) : \sigma^h_{n+\theta} \, d\Omega$$
$$+ \sum_{e=1}^{n_{el}} \int_{\Omega^e} \varepsilon^h \cdot [\rho \left(\mathbf{u}^h_{n+\theta} - \tilde{\mathbf{u}}^h_{n+\theta} \right) / (\theta \Delta t) - \nabla \cdot \sigma^h_{n+\theta}] \, d\Omega$$
$$+ \int_\Omega q^h \nabla \cdot \mathbf{u}^h_{n+\theta} \, d\Omega = \int_{\Gamma_h} \mathbf{w}^h \cdot \boldsymbol{h}^h_{n+\theta} , \quad \forall \mathbf{w}^h \in V^h_{\mathbf{u}} , \forall q^h \in V^h_p ; \tag{24}$$

find $\tilde{u}^h_{n+1-\theta} \in (S^h_u)_{n+1-\theta}$ such that

$$\int_\Omega w^h \cdot \rho \left((\tilde{u}^h_{n+1-\theta} - u^h_{n+\theta}) / ((1-2\theta)\Delta t) \right) d\Omega$$
$$+ \int_\Omega \varepsilon(w^h) : \sigma^h_{n+\theta}\, d\Omega = \int_{\Gamma_h} w^h \cdot h^h_{n+\theta}\, d\Gamma \quad , \qquad \forall\, w^h \in V^h_u ; \tag{25}$$

find $u^h_{n+1-\theta} \in (S^h_u)_{n+1-\theta}$ such that

$$\int_\Omega w^h \cdot \rho \left((u^h_{n+1-\theta} - \tilde{u}^h_{n+1-\theta}) / ((1-2\theta)\Delta t) + u^h_{n+1-\theta} \cdot \nabla u^h_{n+1-\theta} \right) d\Omega$$
$$+ \sum_{e=1}^{n_{el}} \int_{\Omega^e} \delta^h \cdot [\rho \left((u^h_{n+1-\theta} - \tilde{u}^h_{n+1-\theta}) / ((1-2\theta)\Delta t) \right)$$
$$+ u^h_{n+1-\theta} \cdot \nabla u^h_{n+1-\theta})]\, d\Omega = 0 , \qquad \forall\, w^h \in V^h_u ; \tag{26}$$

find $\tilde{u}^h_{n+1} \in (S^h_u)_{n+1}$ such that

$$\int_\Omega w^h \cdot \rho \left((\tilde{u}^h_{n+1} - u^h_{n+1-\theta}) / (\theta\Delta t) + u^h_{n+1-\theta} \cdot \nabla u^h_{n+1-\theta} \right) d\Omega = 0$$
$$+ \sum_{e=1}^{n_{el}} \int_{\Omega^e} \delta^h \cdot [\rho \left((\tilde{u}^h_{n+1} - u^h_{n+1-\theta}) / (\theta\Delta t) \right)$$
$$+ u^h_{n+1-\theta} \cdot \nabla u^h_{n+1-\theta})]\, d\Omega = 0 , \qquad \forall\, w^h \in V^h_u ; \tag{27}$$

find $u^h_{n+1} \in (S^h_u)_{n+1}$ and $p^h_{n+1} \in S^h_p$ such that

$$\int_\Omega w^h \cdot \rho \, (u^h_{n+1} - \tilde{u}^h_{n+1}) / (\theta\Delta t)\, d\Omega + \int_\Omega \varepsilon(w^h) : \sigma^h_{n+1}\, d\Omega$$
$$+ \sum_{e=1}^{n_{el}} \int_{\Omega^e} \varepsilon^h \cdot [\, \rho \, (u^h_{n+1} - \tilde{u}^h_{n+1}) / (\theta\Delta t) - \nabla \cdot \sigma^h_{n+1}\,]\, d\Omega$$
$$+ \int_\Omega q^h \nabla \cdot u^h_{n+1}\, d\Omega = \int_{\Gamma_h} w^h \cdot h^h_{n+1} \ , \ \forall\, w^h \in V^h_u \ , \ \forall\, q^h \in V^h_p \ . \tag{28}$$

Remarks

4. The parameter θ is the one used in the θ-scheme [4]; we set it to 1/3.

5. Unlike the T6 algorithm described in [2], we add the SUPG supplement in all stages involving the advection term, i.e. in equations (23), (26), and (27). The PSPG supplement added in equations (24) and (28) is for the **Q1Q1/ε** element only.

6. The matrix forms corresponding to equations (23), (25), (26), and (27) can be solved implicitly or explicitly as described in [2]. The matrix form of the two "Stokes sub-steps", i.e. equations (23) and (28), are quite similar to the matrix form of the one-step formulation; they can be solved implicitly or by treating the velocity explicitly. Except for the **Q1Q1/ε** element, the results presented in this paper are based on the explicit treatment of all sub-steps, and are obtained with 5-3-3-3-5-3 passes. For the **Q1Q1/ε** element the "Stokes sub-steps" are treated implicitly, and the results are obtained with 5-1-3-3-5-1 passes.

4. The Velocity-Pressure Elements Used

The velocity-pressure elements used in this paper are shown in Figure 1.

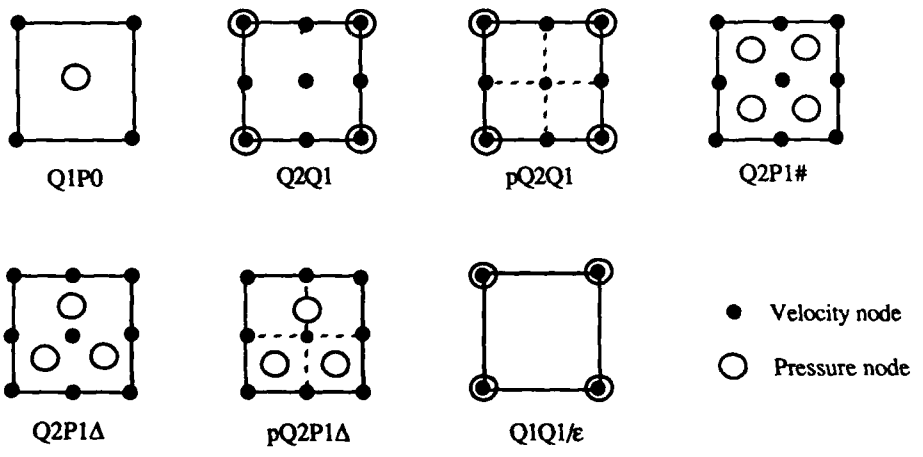


Figure 1. The velocity-pressure elements used.

We now describe each element briefly.

Q1P0 This element employs bilinear interpolation for velocity and discontinuous piecewise constant interpolation for pressure. It does not satisfy the Babuska-Brezzi condition and is known to suffer from spurious pressure modes. Nevertheless it is a popular element.

Q2Q1 This is another popular element; it employs biquadratic interpolation for velocity and bilinear interpolation for pressure.

pQ2Q1 This is the "pseudo" version of the **Q2Q1** element in which the velocity is piecewise bilinear over each sub-element. In Figure 1 these sub-elements are denoted by dashed lines.

Q2P1# This element uses biquadratic interpolation for velocity and discontinuous piecewise bilinear interpolation for pressure. It does not satisfy the Babuska-Brezzi condition and is known to produce spurious pressure modes.

Q2P1Δ This element employes biquadratic interpolation for velocity and piecewise discontinuous linear interpolation for pressure.

pQ2P1Δ This is the "pseudo" version of the **Q2P1Δ** element in which the velocity is piecewise bilinear over each sub-element. In Figure 1 these sub-elements are denoted by dashed lines.

Q1Q1/ε This element employes bilinear interpolations for both velocity and pressure. Although this element does not satisfy the Babuska-Brezzi condition, it can be stabilized by adding a PSPG supplement to the regular weighting function.

5. Numerical Tests and Observations

Although for all elements computations were performed with both one-step and T6 formulations, we only show the selected results for the T6 formulation. For the purpose of comparison, in each problem the meshes generated by using different elements have the same distribution of velocity nodes. The nodal values of the pressure, stream function, and vorticity are obtained by least-squares interpolation.

The standing vortex problem

This test problem was suggested to us by Gresho (see [5]). The purpose of the test is to get an indication of how much numerical dissipation a formulation introduces. The flow is inviscid and is contained in a 1×1 box. The initial condition consists of an axisymmetric velocity profile with zero radial velocity and with the circumferential velocity given as $u_\theta = \{\ 5r \text{ for } r < .2,\ 2\text{-}5r \text{ for } .2 < r < .4,\ 0 \text{ for } r > .4\}$. Since this initial condition is also the exact steady-state solution, the numerical formulation should preserve this "standing" vortex as accurately as possible. The finite element mesh is uniform and contains 20×20 elements for **Q1P0** and **Q1Q1/ε** elements. For the higher-order elements we use 10×10 elements. The time step is 0.05; based on a constant "element length" of 0.05 this results in a peak local Courant number of 1.0.

Some of the solutions obtained at t = 3 (i.e. after 60 time steps) are shown in Figures 2-4. The table below shows, for various elements, the percentage of the vortex kinetic energy retained after 60 time steps.

Element	One-step implicit	One-step explicit	T6 explicit
Q1P0	22.6%	22.6%	94.7%
Q2Q1	Unstable	Unstable	Unstable
pQ2Q1	Unstable	Unstable	Unstable
Q2P1#	Unstable	Unstable	Unstable
Q2P1Δ	84.5%	84.5%	92.7%
pQ2P1Δ	85.5%	85.4%	91.7%
Q1Q1/ε	93.7%	86.9%	88.2%

Clearly the T6 formulation is less dissipative than the one-step explicit formulation. Although with the T6 formulation all elements seem to yield comparable levels of dissipation, with the one-step formulation the **Q1P0** element shows significantly higher dissipation. Moreover we observe that for higher-order elements the difference in energy dissipation between the one-step and T6 formulations is not so large as it is for the **Q1P0** element. We also note that the solution obtained with the **pQ2P1Δ** element is very close to the solution obtained with the **Q2P1Δ** element. We found that the Galerkin one-step formulation is unstable for all elements except for the **Q1P0** element which is only marginally stable even with an implicit (consistent-reordered, expensive) formulation.

The lid-driven cavity flow

In this problem the lid of the cavity has unit velocity; based on this velocity and the dimension of the cavity the Reynolds number is 400. We choose a uniform grid of 32×32 elements for **Q1P0** and **Q1Q1/ε** elements. A uniform grid of 16×16 elements is used for the higher-order elements.

Some of the steady-state results are shown in Figures 5-7. The results obtained with the higher-order elements (i.e. **Q2Q1**, **pQ2Q1**, **Q2P1#**, **Q2P1Δ**, and **pQ2P1Δ**) are all in close agreement. Also, the results obtained with the **Q1P0** element is close to those obtained with the **Q1Q1/ε** element. The differences in the solutions obtained with the higher- and lower-order elements can be attributed to the size of the leakage area near the moving lid.

Flow past a circular cylinder

In this problem we have a uniform upstream flow; the Reynolds number based on the cylinder diameter is 100. The different meshes employed are shown in Figure 8.

Some of the steady-state results are shown in Figures 9-11. Except for the **Q2P1#** element, which produces oscillations in the velocity field all around the cylinder, the results obtained with the higher-order elements (i.e. **Q2Q1**, **pQ2Q1**, **Q2P1Δ**, and **pQ2P1Δ**) are all in close agreement. For the **Q1P0** and **Q1Q1/ε** elements we observe some small differences in the pressure field; these differences become slightly more noticeable in the upstream region. Unlike it was in the driven cavity problem, both higher- and lower-order

elements result in solutions which are in good agreement when one inspects the velocity and the variables derived from the velocity. Pressure fields, on the other hand, exhibit some very small differences. The drag coefficients obtained with these elements are 1.162 (**Q1P0**), 1.149 (**Q2P1**Δ), and 1.154 (**Q1Q1**/ϵ).

References

1. A.N. Brooks and T.J.R. Hughes, "Streamline-Upwind/ Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on Incompressible Navier-Stokes Equation", *Computer Methods in Applied Mechanics and Engineering*, 32 (1982), pp. 199-259.

2. T.E. Tezduyar, J. Liou and D.K. Ganjoo, "Incompressible Flow Computations Based on the Vorticity-Stream Function and Velocity-Pressure Formulations", University of Minnesota Supercomputer Institute Report UMSI 89/96, May 1989.

3. T.J.R. Hughes and L.P. Franca, "A New Formulation for Computational Fluid Dynamics: VII. The Stokes Problem with Various Well-posed Boundary Conditions: Symmetric Formulations that Converge for All Velocity/Pressure Spaces", *Computer Methods in Applied Mechanics and Engineering*, 65 (1987), pp. 85-96.

4. M.O. Bristeau, R. Glowinski, and J. Periaux, "Numerical Methods for the Navier-Stokes Equations: Applications to the Simulation of Compressible and Incompressible Viscous Flows", *Computer Physics Report*, 6 (1987), pp. 73-187.

5. P.M. Gresho and S.T. Chan, "Semi-consistent Mass Matrix Techniques for Solving the Incompressible Navier-Stokes Equations", Lawrence Livermore National Laboratory, Preprint UCRL-99503, 1988.

6. M. Braza, P. Chassaing and H.H. Minh, "Numerical Study and Physical Analysis of the Pressure and Velocity Fields in the Near Wake of a Circular Cylinder", *Journal of Fluid Mechanics*, 165 (1986), pp. 79-130.

7. T.E. Tezduyar, J. Liou, D.K. Ganjoo, and M. Behr, "Incompressible Flow Computations with the Finite Element Method", to appear, Hemisphere Publishing.

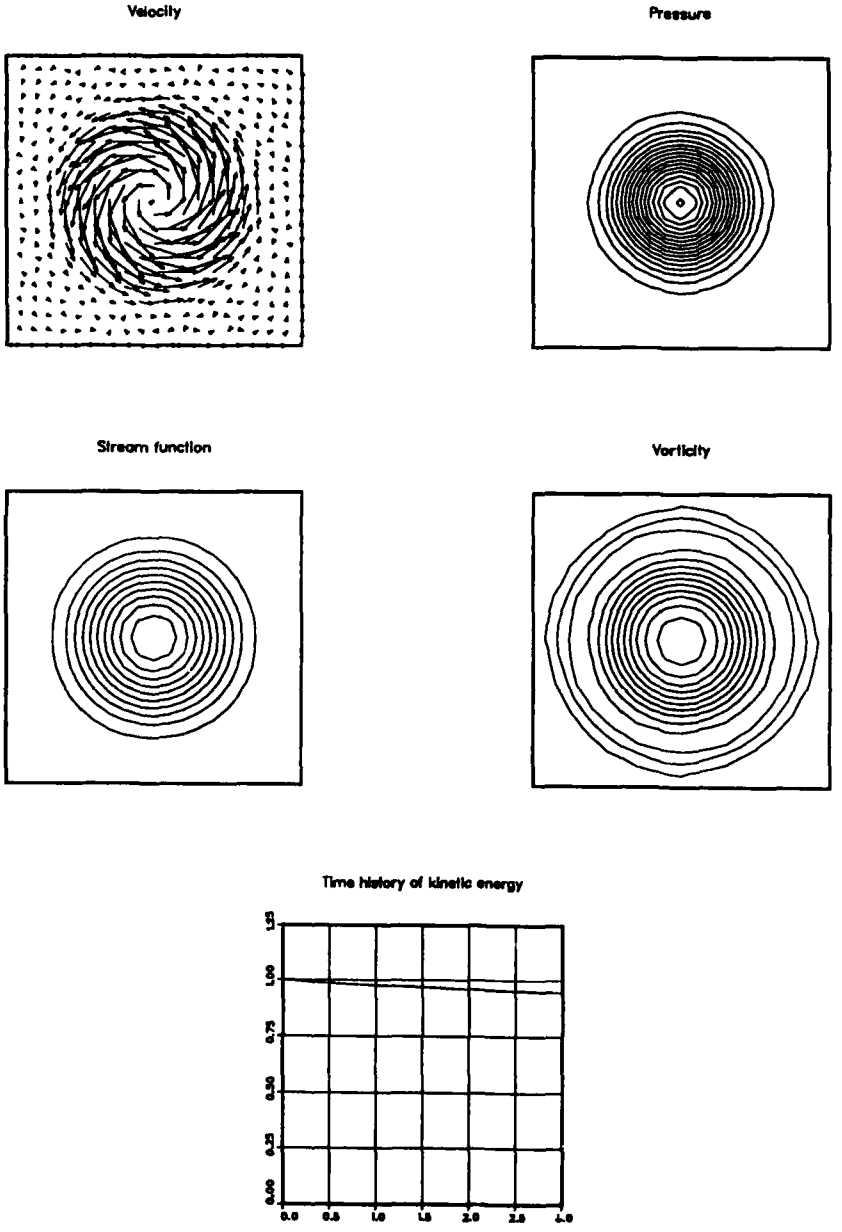


Figure 2. Solution of the standing vortex problem at t=3 with Q1P0/T6.

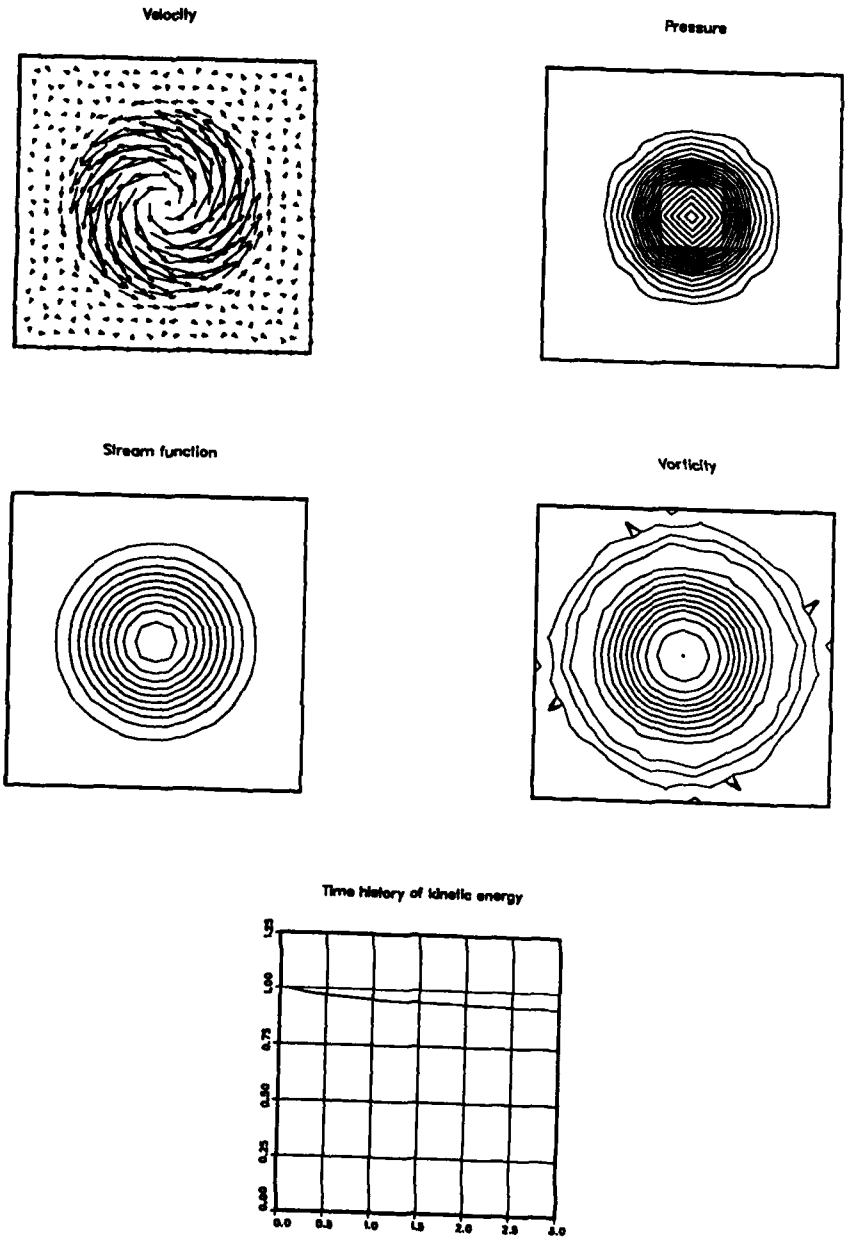


Figure 3. Solution of the standing vortex problem at t=3 with Q2P1Δ/T6.

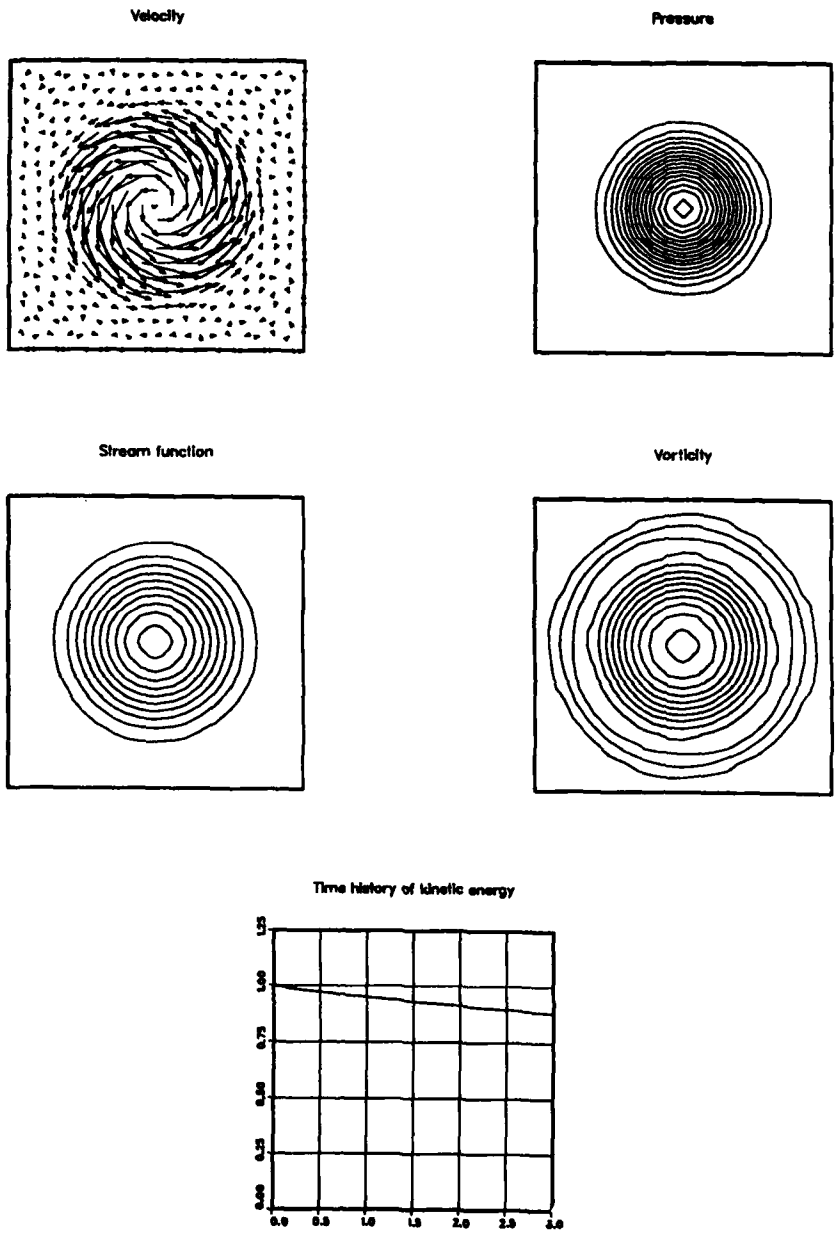


Figure 4. Solution of the standing vortex problem at t=3 with Q1Q1/E/T6.

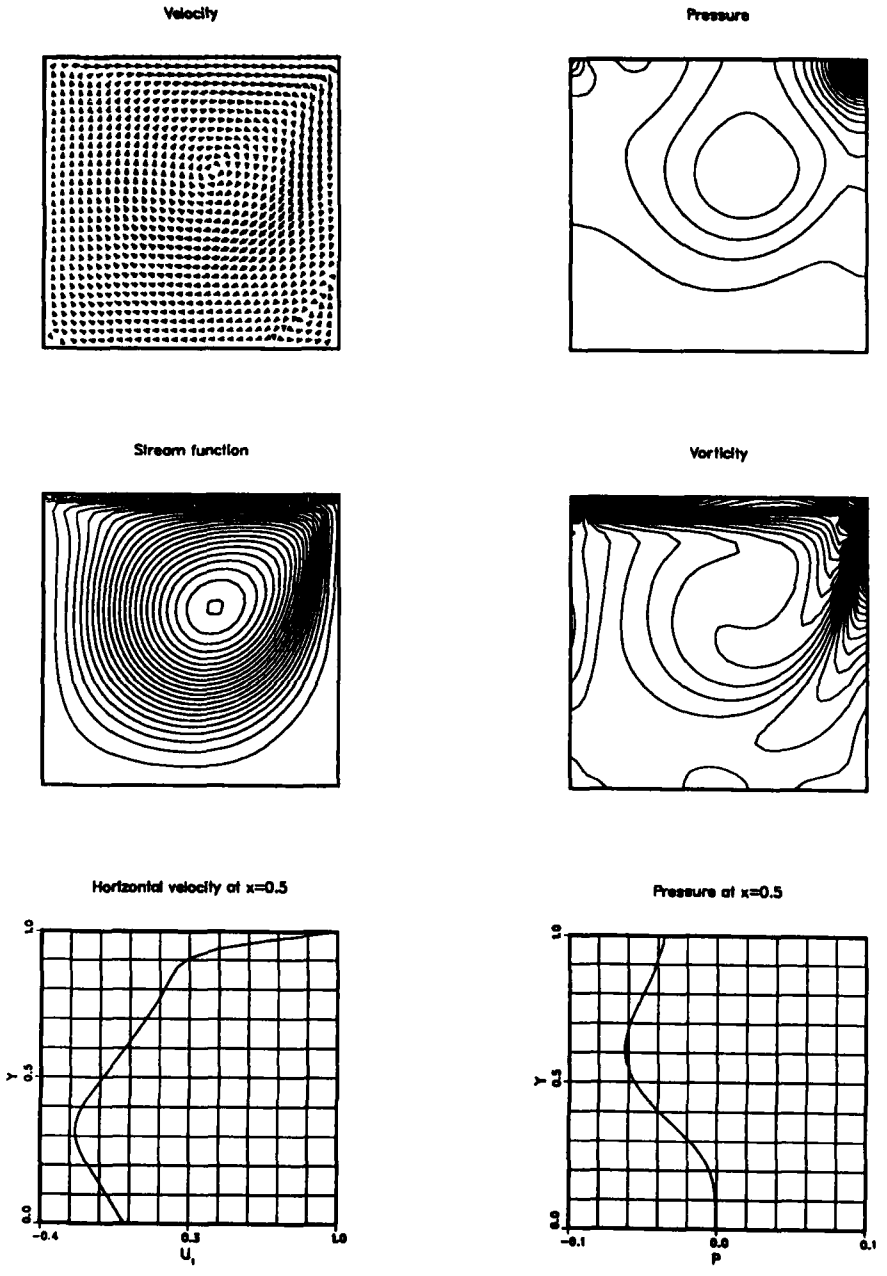


Figure 5. Driven cavity flow at Reynolds number 400: steady-state solution obtained with Q1P0/T6.

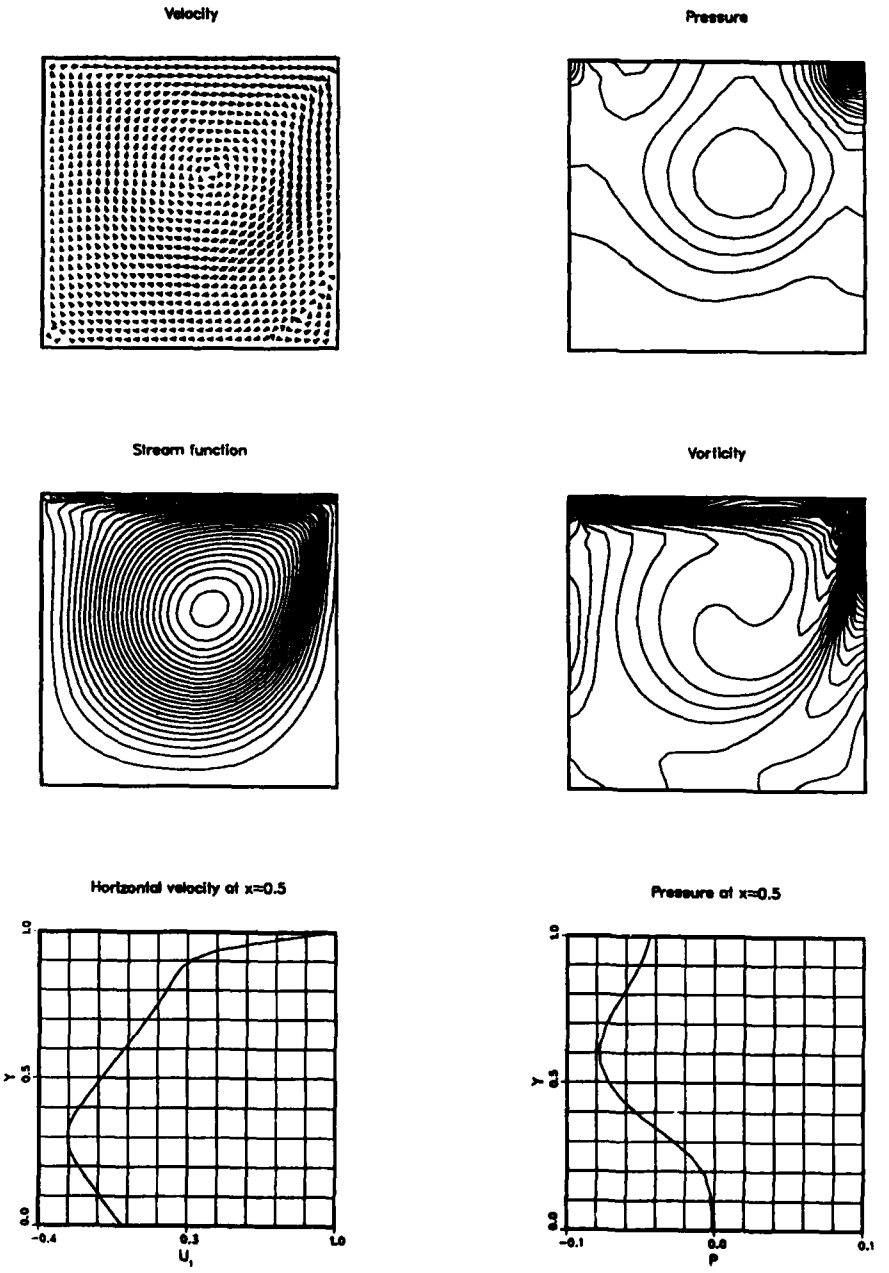


Figure 6. Driven cavity flow at Reynolds number 400: steady-state solution obtained with Q2P1Δ/T6.

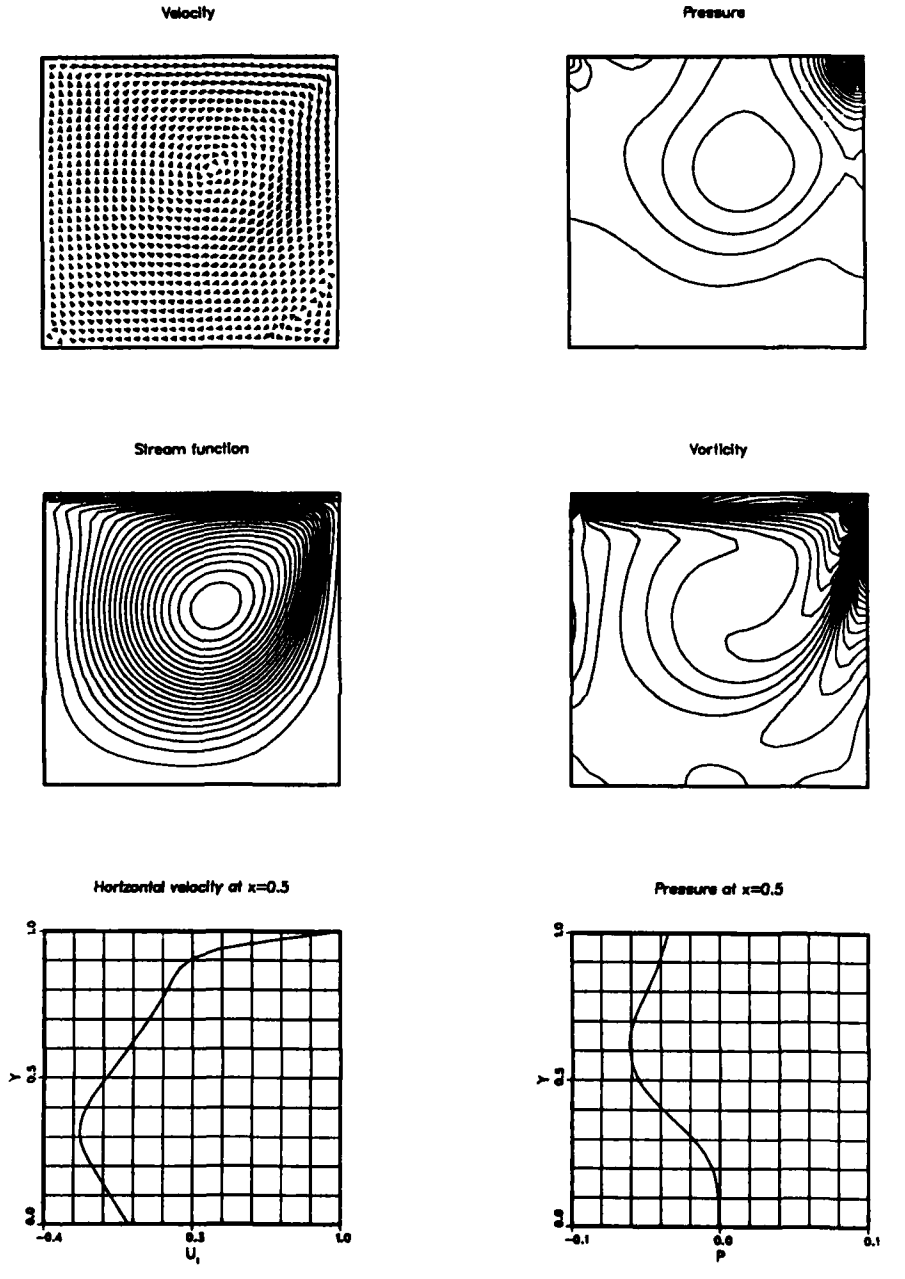


Figure 7. Driven cavity flow at Reynolds number 400: steady-state solution obtained with Q1Q1/ε/T6.

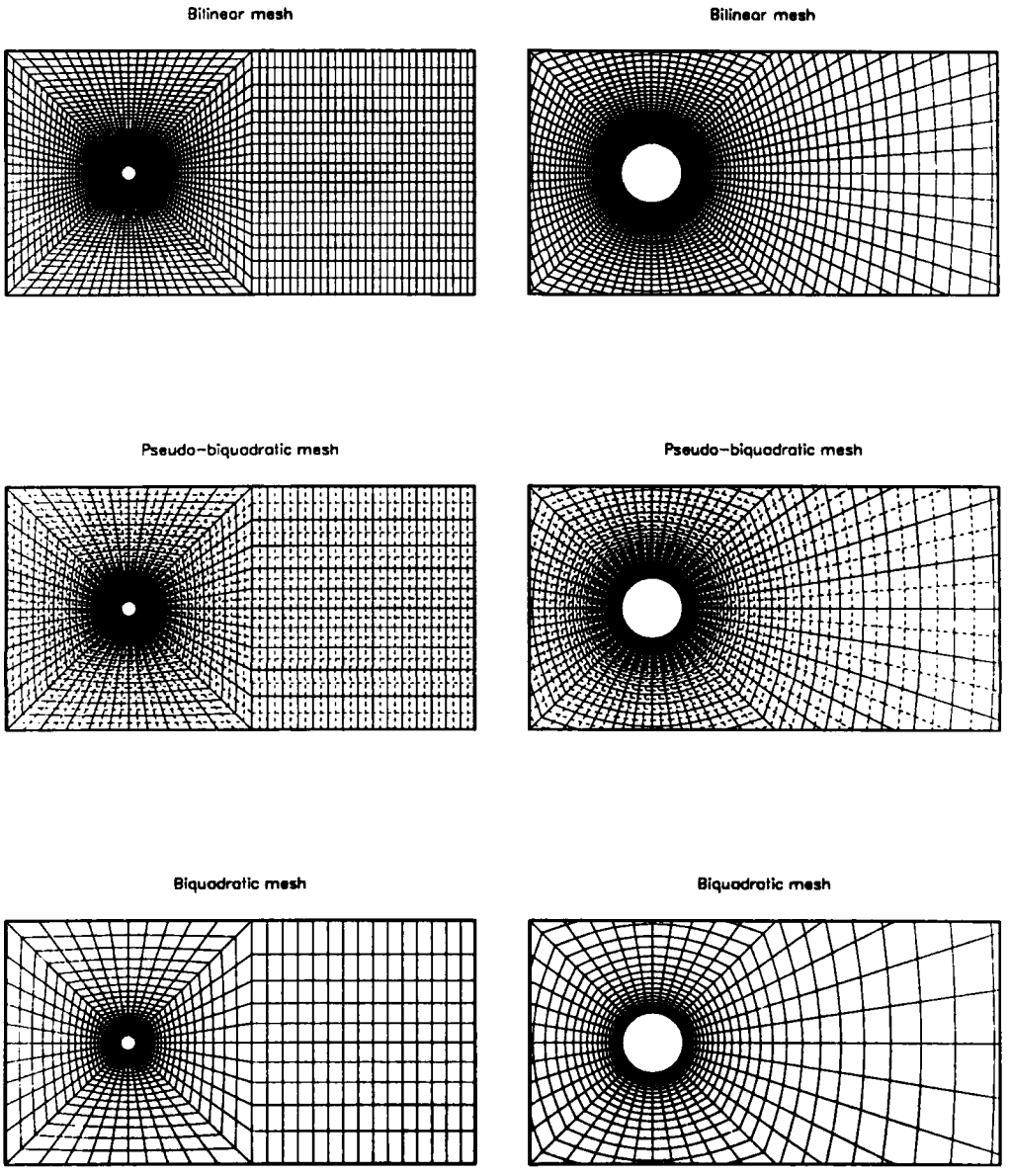


Figure 8. Meshes for the problem of flow past a circular cylinder:
(5240 elements, 5350 velocity nodes, and 5240 pressure nodes for Q1P0 element,
1310 elements, 5350 velocity nodes, and 3930 pressure nodes for pQ2P1Δ element,
1310 elements, 5350 velocity nodes, and 3930 pressure nodes for Q2P1Δ element).

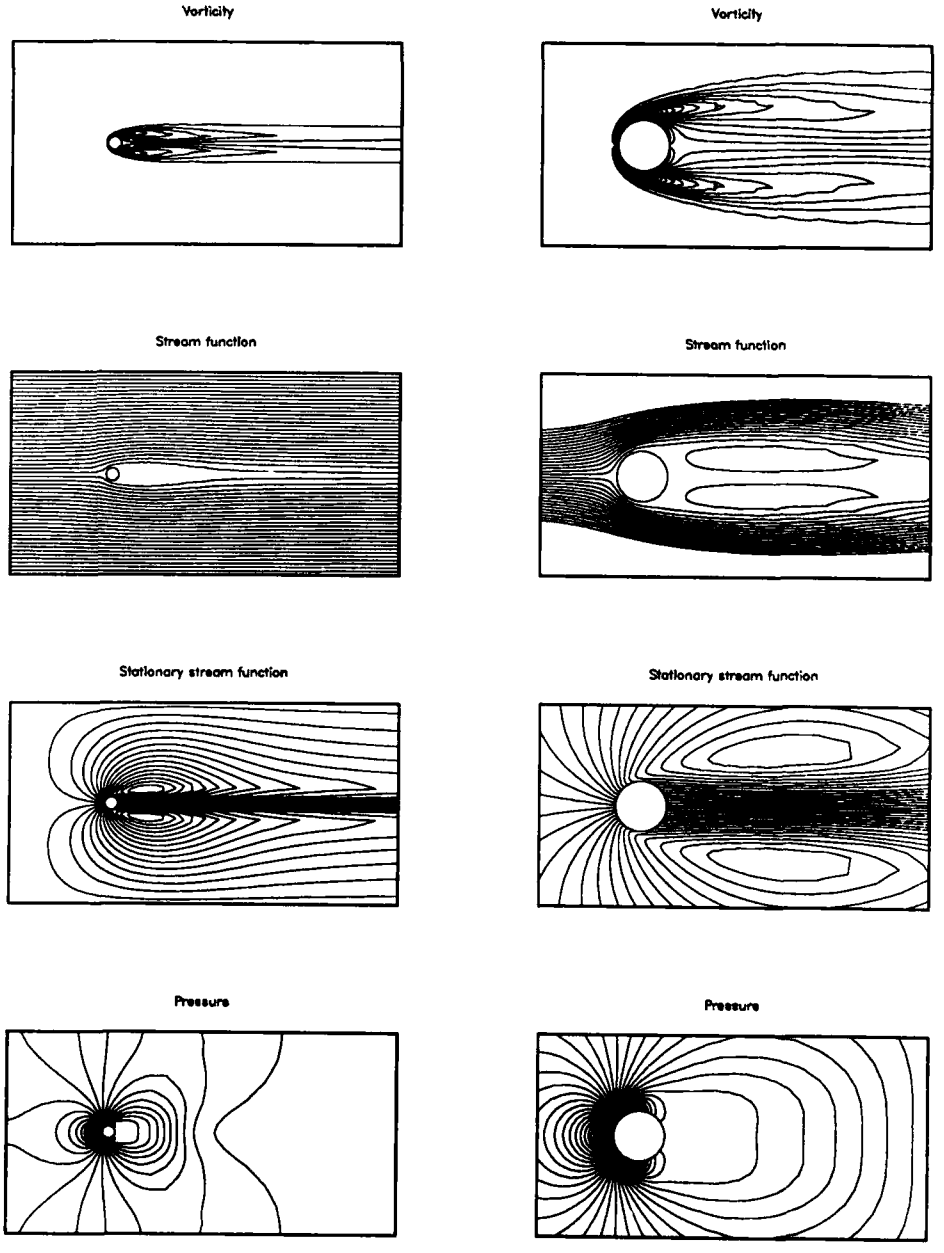


Figure 9. Flow past a circular cylinder at Reynolds number 100: steady-state solution obtained with Q1P0/T6.

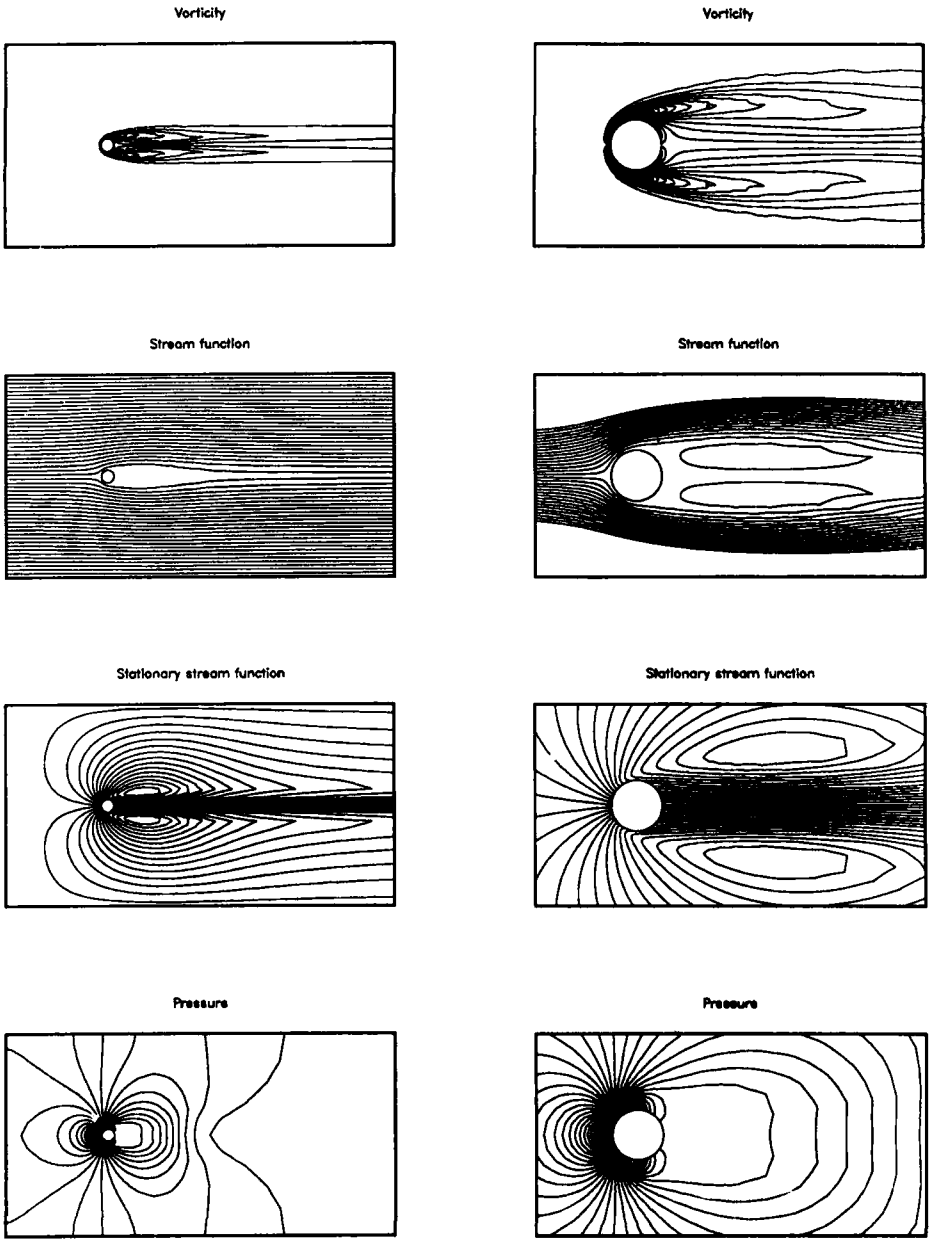


Figure 10. Flow past a circular cylinder at Reynolds number 100: steady-state solution obtained with Q2P1Δ/T6.

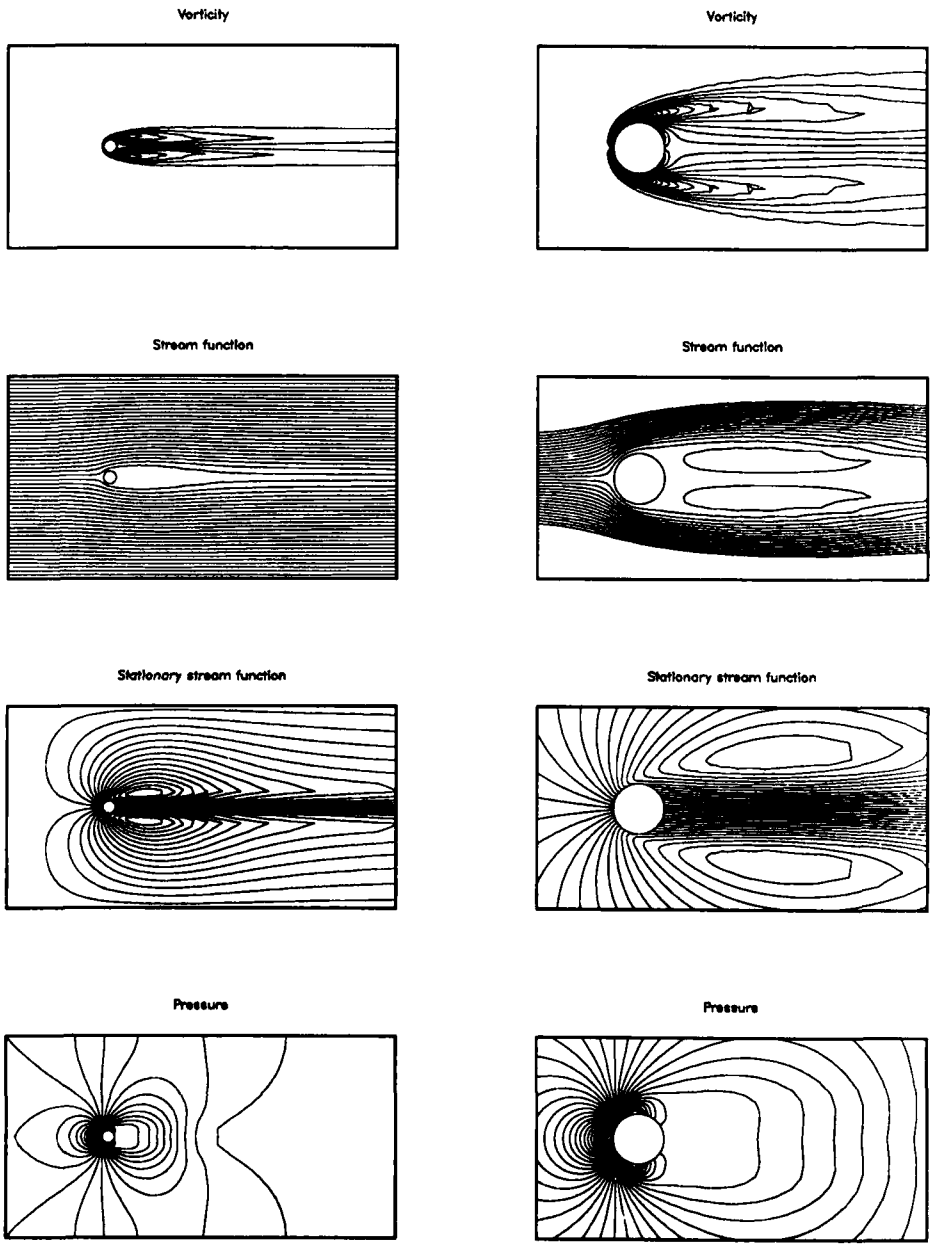


Figure 11. Flow past a circular cylinder at Reynolds number 100: steady-state solution obtained with Q1Q1/ε/T6.

Improving a Parallel Ray Tracing Algorithm on an iPSC/2 by Emulating a Read-only Shared Memory

Didier Badouel - Thierry Priol
IRISA/INRIA - Rennes

Abstract

The production of realistic image generated by computer requires a huge amount of computation and a large memory capacity. The use of highly parallel machines allows this process to be performed faster. Distributed memory parallel computers, like hypercubes or *transputer*-based machines, offer an interesting ratio performance/cost assuming that a load balancing and a partition of the data domain is found. This paper deals with the demonstration that emulating a shared memory on these computers seems to be the best way to parallelize algorithms like ray tracing which use large read-only databases with no obvious distribution. Results are given which allow to compare with a previous parallel ray tracing algorithm that we have implemented on an iPSC/2.

1 Introduction

The ray tracing algorithm, based on simple optics' laws, provides the simulation of illumination effects such as the shading, the reflection, and the refraction. In order to evaluate the color of each pixel of an image, a geometric model is used to describe the objects in a scene and a photometric model is used to define the behaviour of objects with respect to light sources. Each light intensity contribution for a pixel is evaluated with two kinds of computation.

the *geometric calculations* evaluate the closest intersection between a ray and the objects in the scene. Their number increases with the *photometric* complexity of the scene, i.e. with the number of rays, and with the *geometric* complexity of the scene, i.e with the number and the shape of the objects. Several attempts have been proposed to minimize the amount of ray/object intersection. These solutions are based on what we call an *object access structure* which allows a fast search of objects along a ray path. They can be grouped in two approaches:

- creation of a tree of bounding volumes [24, 18],
- subdivision of the scene extents in an adaptative way [2, 12, 17] or a regular way [1, 6, 10, 11].

the *photometric calculations*, once the impact point determined, are used to evaluate the light intensity contribution of a ray. Their number is proportional to the number of intersection points. According to the photometric properties of the objects, new rays are shot from the intersection point in order to take into account the contribution to the pixel intensity of the neighboring objects [9, 14, 28]. In fact, if the object is transparent (respectively reflective) then a new ray is shot in the refracted direction (respectively in the reflective direction).

The purpose of this paper is not to compare the different models used for the geometric and the photometric calculations, a detailed discussion can be found in [23] or in [10]. The aim of the paper is to focus on the problem of the amounts of computation and memory requirement whatever algorithmical choices which are made. For our parallel ray tracing algorithm, we have opted for :

- a polyhedral description for the objects.
- a regular grid as *object access structure*.
- the Whitted model [28] improved by the Phong model [21] for the photometric evaluation.

Computing realistic images require several millions rays, and several hundreds of thousands objects. For each ray, the closest intersection point with the scene must be computed. Thus, it is the great number of ray/object intersections which makes the ray tracing so expensive. Despite the improvements for ray/scene intersections, the ray tracing algorithm still is too slow on sequential computer. Moreover, latest researches such as stochastic sampling [8, 19] and sophisticated light models [7, 16, 25] require more and more computations. New algorithmical improvements could not decrease substantially the synthesis time. Therefore, since few years, we are studying the use of Distributed Memory Parallel Computers (DMPC) which are low cost supercomputers. A DMPC is a MIMD computer where each processor has a local memory used to store its own code and data. All the processors of a DMPC are connected through a network.

For our experimentations, we have used an iPSC/2. The iPSC/2 system consist of a *cube* connected to a *host* processor. The *cube* houses all the nodes connected through a hypercube network topology. Each node consists of the Intel 80386 microprocessor supplied with a 80387 floating point co-processor and 4 Mbytes of local memory. It is equipped with the Direct Connect Module (DCM) for high speed routing message between nodes. The performance of a 64 nodes are approximatively 256 MIPS and 20 MFLOPS. The node support a vector extension board with peak performance 20 MFLOPS per node. The system available at IRISA is configured in 64 nodes with no vector extension. The *host* processor contains the software development tools. It is connected via a special link to node cube 0. It performs compilation, program loading and I/O operation with the hypercube. The iPSC/2 can be programmed in C or FORTRAN. A communication library has been added to these languages to allow sending of receiving messages between nodes.

The standard programming methodology consists in subdividing the problem to be solved in a set of communicating tasks [15] and map them on processors. Each node contains in its local memory the code and the data for its processes, and all the processes on the *cube* and on the *host* can communicate via the exchange of messages. The conception of a parallel algorithm requires special attentions for correctness and efficiency, avoiding deadlocks and ensuring a load balancing. The computation load balancing for data driven problems (like ray tracing), where the dynamic behaviour is quite difficult to be modelized, is experimentally measured. The performances of a parallel algorithm is commonly given in terms of speed-up and efficiency. The speed-up is the ratio of the running time of a processor to the running time obtained with p processors. This quantity represents, in fact, the number of processors effectively used during the parallel execution of the algorithm. As for the efficiency, it is equal to the ratio of the speed-up to the number of processors. It represents the average utilization of the processors. These two quantities are enough to measure the computation load balancing of a parallel algorithm but are not the only performance criteria. In particular, for problems using large databases, the partition of the data domain must be considered.

Before describing our parallel implementation of the ray tracing, we first discuss on which are the computations and the databases required in this algorithm, and what are their amounts.

2 Computations involved in the ray tracing algorithm

The purpose of this section is to present the basic algorithms, for the geometric computations, considered in our ray tracing. We only present the algorithms necessary to solve ray/scene intersections as they represent more than 80% of the synthesis time for ray tracing complex scenes. In our implementation, we use at one and the same time a *regular grid* and object extends called *slabs* defined by Kay and Kajiya [18]. For each ray, the grid provides a list of polygons which localization is near the ray direction. Then, before computing the intersection ray/polygon, a test is made using the *slabs* to avoid this computation when the ray pass outside the polygon extends. The use of these two filters at the same time is justified by the fact that the grid which minimize the synthesis time is not the one which provides the smallest number of polygons.

Ray and polygon representations

- The parametric representation of a ray is :

$$R(t) = O + D.t \tag{1}$$

  where, O is the origin of the ray, D the direction of the ray, and t the parameter of the representation.

- A polygon is described by its vertices V_i ($i \in \{0, \cdots, N-1\}, N \geq 2$). Let x_i, y_i and z_i the coordinates of the vertex V_i. The normal of the plane containing the polygon, N, is computed with the cross product :

$$N = (V_1 - V_0) \times (V_2 - V_0)$$

  For any point P of the plane we have $P.N = cst$. This constant value is computed by the dot product $d = -V_0.N$. The arithmetic representation of the plane, is calculated once for all, and stored in the polygon description.

$$N.P + d = 0 \tag{2}$$

Ray/polygon intersection

Using a ray tracing method with polygonal databases, we must define a fast algorithm to compute ray/polygon intersection. A barycentric approach has been described in [27], the following algorithm is quite similar but faster. The goal of the algorithm is not only to determine if a Ray goes through the polygon, but must then determine the coordinates of the intersection point and parameters to localize this point with respect to the polygon's vertices. These parameters are used to compute the interpolated normal at this point, and can be used also to compute the entry of a texture map.

The evaluation of the parameter t corresponding to the intersection point between the ray and the embedding plane of the polygon, can be obtained using the equations (1) and (2) :

$$t = \frac{d - N.O}{N.D} \tag{3}$$

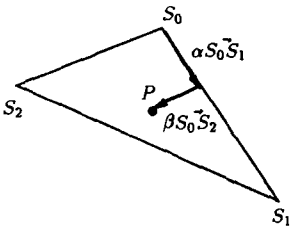


Figure 1: Parametric representation of the point P.

If polygon and ray are parallel ($N.D = 0$), or if the intersection is ahead the origin of the ray ($t \leq 0$), or if a closer intersection has been already found ($t > min_t$), the intersection will be rejected. If not, we must determine if the intersection point is inside the polygon. This is done using a parametric resolution. This solution is based on triangles. If a polygon has n vertices ($n > 3$), it will be view as a set of $n - 2$ triangles. The only constraint is to use convex polygon. The point P (cf Fig. 1) is given by:

$$\vec{V_0P} = \alpha.\vec{V_0V_1} + \beta.\vec{V_0V_2} \tag{4}$$

The point P will be inside the triangle $(V_0V_1V_2)$ if :

$$\alpha \geq 0, \beta \geq 0 \text{ and } \alpha + \beta \leq 1$$

The computation of α, β requires to resolve a system of three equations and with two unknows which can be reduce in a system of two equations with two unknows when working in one of the plane perpendicular to the axis. In order to discard the degenerated polygons where the projection would be a segment, we choose the plane of 'biggest' projection (as in [27]) computing the value i_0 representing the direction of the projection plane.

$$i_0 = \begin{cases} 0 & si\ |N_x| = Max(|N_x|, |N_y|, |N_z|). \\ 1 & si\ |N_y| = Max(|N_x|, |N_y|, |N_z|). \\ 2 & si\ |N_z| = Max(|N_x|, |N_y|, |N_z|). \end{cases}$$

Consider i_1 and i_2 (i_1 and $i_2 \in \{0, 1, 2\}$), the indices different from i_0 representing the two other directions, and (u, v) the two-dimentional coordinates of the vectors $\vec{V_0P}$, $\vec{V_0V_1}$ and $\vec{V_0V_2}$.

$$u_0 = P_{i_1} - V_{0_{i_1}} \qquad u_1 = V_{1_{i_1}} - V_{0_{i_1}} \qquad u_2 = V_{2_{i_1}} - V_{0_{i_1}}$$

$$v_0 = P_{i_2} - V_{0_{i_2}} \qquad v_1 = V_{1_{i_2}} - V_{0_{i_2}} \qquad v_2 = V_{2_{i_2}} - V_{0_{i_2}}$$

Then, the solutions are :

$$\alpha = \frac{det\begin{pmatrix} u_0 & u_2 \\ v_0 & v_2 \end{pmatrix}}{det\begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix}} \quad \text{and} \quad \beta = \frac{det\begin{pmatrix} u_1 & u_0 \\ v_1 & v_0 \end{pmatrix}}{det\begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix}}$$

The interpolated normal from the point P is obtained by :

$$N_P = (1 - (\alpha + \beta)).N_0 + \alpha.N_1 + \beta.N_2$$

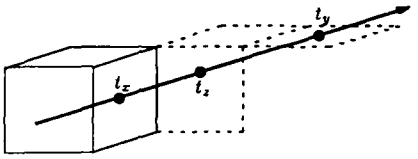


Figure 2: Running through a 3D grid.

Using a regular 3D grid

A 3D grid is a discrete representation of the scene space, it subdivises this space in equal parallelepipedic sub-volumes called *voxels*. Thus, the *discrete* run of a ray through the grid is a set of ordered voxels. Each voxel of the grid contains a list of identifiers for the polygons passing through its volume. During the synthesis task, the polygons kept for a given ray are those member of the voxels encountered during the run of the ray. In order to avoid several intersections of the same polygon with a given ray, an identifier (Ray_{id}) is stored in the polygon description and represents the last ray compared with this polygon.

The method chosen to run through the grid is the same as the one described in [1] ; Beforehand, the first voxel encountered by the ray is computed. This voxel is either the voxel containing the origin of the ray (O) or the entry voxel when the ray comes from outside the grid. For each ray, the following values are initialized :

- the constants δt_x, δt_y and δt_z represent the increment of t in each direction x, y, z.
- the variables t_x, t_y and t_z represent the values of t for the next boundary voxel when crossing in the direction x, y or z (cf Fig. 2).

The incremental run of the grid is then computed in a easy way ; For each step, the comparison between t_x, t_y and t_z gives the direction where the next voxel is located. When a step is made in the i direction, the variable t_i is incremented with the constant value δt_i.

Using Slabs

The slabs (cf [18]) are convex extents delimited by pairs of parallel planes (see in figure 3 a 2D example). One slab is characterized by a normal direction N_i and two values d_i^{min} and d_i^{max}, such as the equation of the planes bordering a polygon in the direction Ni are :

$$N_i \cdot P + d_i^{min} = 0 \quad \text{and} \quad N_i \cdot P + d_i^{max} = 0 \tag{5}$$

The values d_i^{min} and d_i^{max} are evaluated with the projection of each vertex V_j onto the line of direction N_i.

$$d_{ij} = N_i \cdot V_j \qquad d_i^{min} = \min_j(d_{ij}) \qquad d_i^{max} = \max_j(d_{ij})$$

The values d_i^{min} and d_i^{max} are stored in the polygon description. During the synthesis task, the intersection ray/slab gives a segment of the ray such as $t_i^{min} \leq t \geq t_i^{max}$. These values are computed using the ray representation (Equ. 1) and the slabs representation (Equ. 5).

$$t_i^{min} = \frac{d_i^{min} - N_i.O}{N_i.D} \qquad t_i^{max} = \frac{d_i^{max} - N_i.O}{N_i.D}$$

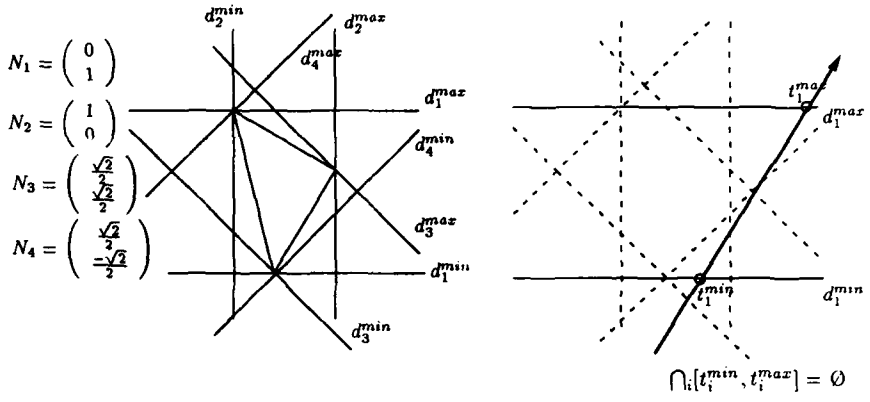


Figure 3: Description and intersection of the *slabs* extends.

Choosing different slab directions for each polygon would be possible, however, when the slab directions are the same for all the polygons, some improvements can be made. The idea (proposed in [18]), is to pre-compute for each ray, the following values :

$$S_i = N_i \cdot O \quad \text{and} \quad T_i = \frac{1}{N_i \cdot D}$$

And then, the intersection ray/slab only requires the following computations :

$$t_i^{min} = (d_i^{min} - S_i)T_i \quad \text{and} \quad t_i^{max} = (d_i^{max} - S_i)T_i$$

Since $\bigcap_i [t_i^{min}, t_i^{max}]$ is an empty segment, we can conclude that the calculation of the intersection between the ray and the object may be avoided.

Partitioning the ray tracing algorithm

The geometrical databases involved in these algorithms are the polygons description, and the grid description with its associated voxels. The amount of these databases rapidly reaches several tens millions of bytes (Mbytes). For example, in our results, we present a database which has required 140 Mbytes of memory. The problem of memory amount becomes more crucial when using texture databases. Thus, our study of parallelisation did not hold the algorithms based on processing without dataflow as they do not achieve a data distribution which allow to render complex scenes.

Since the computation of each pixel is independent from the others, the computation can be easily distributed among the processors. As there are much more pixels than processors, load balancing can be achieve by using a server/client programming model. A server process assigns the computation of a pixel to a client process running on a non-busy processor.

Thus, the problem of parallelisation for such an algorithm is to insure both a good database and computation partitions. We have got a first experience with a parallel ray tracing (see [22, 23]) based on processing with ray dataflow. This algorithm took up the Cleary's idea [5] and subdivises the ray tracing problem into sub-regions distributed among the differents

Cache Device	Memory Device	Item of transfer
Registers	Main Memory	Word
Hardware Cache	Main Memory	Block
Main Memory	Secondary Storage	Page
Node Memory	Distributed Memory	Object

Figure 4: Several uses of the memory cache mechanism.

processor elements (PEs). The rays are exchanged between processors when they left one region for the next one. This experience had given several interesting solutions to insure a static load balancing, but the efficiency of this algorithm decreases when the number of processors increases. The main reason of this behaviour is due to messages with an increasing number when using more PEs, and with not a uniform distribution among the PEs.

In conclusion, to find a good computation load balancing for the ray tracing algorithm, while respecting the constraints such as the size of each local memory and the communication rate between PEs, is very complex when using a message passing model of programming. Thus, we have opted from now on a shared memory model of programming to solve our problem.

3 Emulating a read-only shared memory for ray tracing

Why such a model for distributed memory parallel computers ?

Resulting from the difficulty of the message passing model of programming, severals studies have been done to define mechanisms that implements a shared data model in distributed systems [3, 4, 20]. The goals of this works is to provide a better abstraction of data mapping over a set of distributed memories. In order to offer a general tool, while not degrading performances, in [20] and [3] strategies are studied to maintain data consistency between copies of modified variables . Our study is in the reverse order, trying to optimize a specific parallel application (ray tracing), we came up with emulating a shared memory. The data management following this abstraction is quite attractive but our first objective is always the efficiency of data accesses. The aim of our study is to show that an emulation of a shared memory on a DMPC is the best way to parallelize algorithms such as ray tracing which use large read-only databases with no obvious domain decomposition. With a DMPC, a portion of each node's memory can be used to store a part of the shared database and the remaining portion as a cache to speed up low global accesses. The notion of cache, managed by software in our case, is the core of an efficient shared memory emulation.

Caches were introduced to palliate the gap between fast processor cycle times and slow large memory access times. Generally speaking, a memory cache is any hardware or software device storing in a relatively small but fast access area a selected part of a database stored in a larger but slower access memory. A general presentation of cache memories can be found in [26]. For example, in the concept of virtual paging memory, the primary memory can be viewed as a cache for the secondary memory. The use of a cache device improves the bandwidth between the processor and its memory, in our case it increases the bandwidth between a node and a virtual global memory. Several devices use the concept of memory cache as chown in the figure 4.

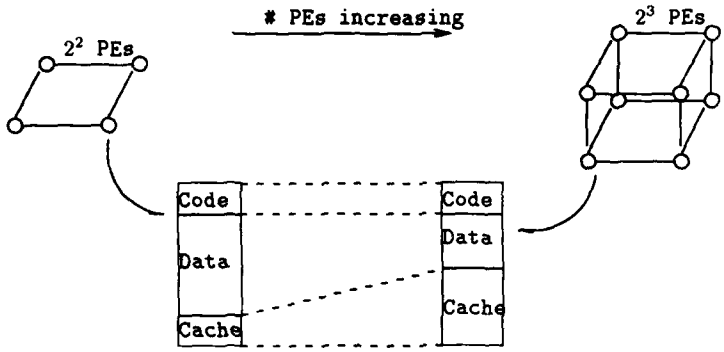


Figure 5: A user node memory description.

Why such a model may be efficient for ray tracing ?

Various characteristics of the ray tracing algorithm led us to design a software tool to emulate a global memory access in the context of distributed memories. These characteristics are as follows:

- the huge amount of memory necessary for this algorithm makes the database load balancing as important as the computation load balancing. Increasing size problems are a challenge for DMPC ;
- due to the coherence property and topological property of 3D objects, only a small part of the whole database is required at a given time. Thus a caching mechanism can be efficient for our problem ;
- due to illumination effects (shading, reflection, refraction), the small part of database necessary to evaluate one pixel is nearly impossible to be statically determined. Thus the database memory management must be dynamic ;
- the calculation of an image uses the database in a read-only way, there is no problem of data coherency management.

How we distribute the shared memory ?

In the ray tracing algorithm, the shared memory is constituted by the database and the bitmap (pixel map). The sharing of pixels will be discussed in the next section. The database contains the photometric and geometric parameters of the objects constituting the scene, and last but not least, the objects access structure. The mechanism used to manage the global memory is called *Object Paging*. This designation includes two aspects, first the virtual memory emulation is done only for data memory, and second the indivisible item of storage is an *object*.

We have seen that in our implementation, objects are polygons and their access structure is a regular grid. Later on, we will call *object* an item of a page of the global database which can be transferred between local memories (a polygon, a voxel of the grid ... etc). An object belongs to one and only one page. One constraint is that the size of any object must be lower than the

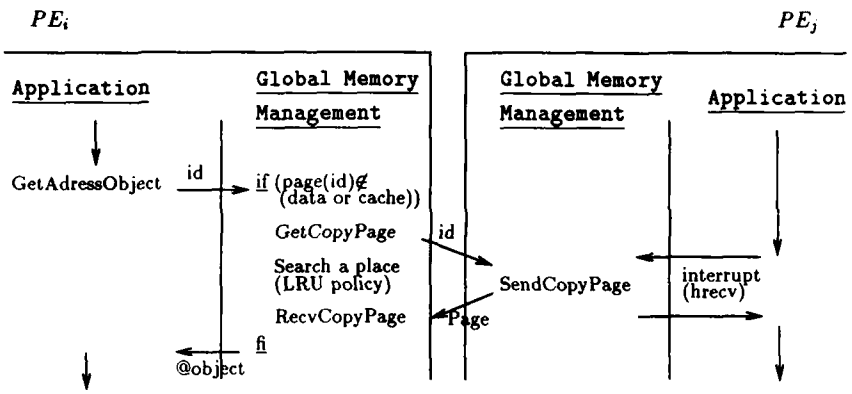


Figure 6: Accessing a global object.

size of a page, and if the size of a page is not a multiple of the size of an object, the difference represents lost memory cells. However, the main advantage is that the memory location of one object is contiguous.

In our algorithm, the whole database is first equally distributed over the set of nodes without any particular mapping. Therefore each PE's memory almost contains the same number of pages. A local memory of a PE is organized as shown in figure 5. Each local memory is divided in three parts: one containing the code to execute the application, an other to store a portion of the database, and the free space is used as a cache memory to optimize global accesses to the distributed global memory. The two last parts (database memory) are divided into pages to allow a memory management.

How we access an object of the shared memory ?

During the synthesis task, the application (ray tracing) can potentially access the whole database through a software memory management. For each node, when a cache default is detected, i.e. the page is neither in its local database nor in the cache memory, then a request is sent to the node responsible for this page. In this case, when a node receive the page, it stores it in the cache memory according to a LRU (Last Recently Used) policy. This search is done during the communication of the new page, and thus does not cause extra cost (cf. Fig 6).

In our implementation, an object is characterized by two numbers : the first one (id_1) is the identifier of the class where the object belongs, and the second one (id_2) is the member identifier inside this class. The numbers (id_1,id_2) represent one unique location in the global memory. A class is a set of objects having the same type. All the objects of one class are stored in contiguous pages to make the global memory management easier. The informations relative to one class are :

- $first_{page}(id)$, the first page where the objects of class id are located.
- $size_{object}(id)$, the size of an object of class id.

- $nb_{object_{per_{page}}}(id)$, the number of objects of class id in one page.

Thus, in order to access a global object, we must determine :

- in what page the object is located ?

$$num_{page} = first_{page}(id_1) + id_2 \ / \ nb_{object-per-page}$$

- on what node is located this page ?

$$num_{node} = num_{page} \ \% \ nb_{node}$$

- where is this page wrt this node ?

$$dep_{node} = num_{page} \ / \ nb_{node}$$

- where is the object wrt this page ?

$$dep_{page} = id_2 \ \% \ nb_{object-per-page}$$

When we have determined the address of the page($@page$), after getting this page from an other node if necessary, we can evaluate the address of the object as follows :

$$@object = @page + dep_{page} \times size_{object}(id_1)$$

For better performances, we have chosen the values as power of two. Thus, all the operations necessary to calculate the global address of an object only require logical operations.

Work distribution and bitmap distribution

Once a shared database has been emulated, the work distribution ensuring a load balancing is quite simple. Each PE is owner of a part of the bitmap. For example, if we use 32 PEs to compute an image with a $512X512$ resolution, each PE manages a $32X32$ sub-bitmap. We use square (or nearly square) sub-bitmap in order to exploit as much as possible the ray coherence property. If the PEs could directly address the frame buffer, a centralized control would not be necessary. As we do not have this facility on the iPSC/2, a copy of the bitmap is managed by the host computer of the hypercube in order to access the frame buffer. The synthesis of each sub-bitmap requires global data accesses at the beginning of the task, and progressively the number of external requests decreases as the memory cache keeps the pertinent items of the global database.

When a PE completes the computation of its sub-bitmap, he sends a request to get an *item of work* (i.e a set of pixels) from a PE still working on its own sub-bitmap. This request moves along a ring topology. If this request goes back without satisfaction, the PE knows that the image is achieved. This local termination detection is sufficient for our application.

In order to insure a good work balancing, the only parameter to be determined is the size of this *item of work*. If its size is minimal (i.e. item of work = one pixel), then we have the best work balancing we can obtained, assuming that the computation of one pixel is indivisible over the set of nodes, but the cost in communication is then higher. Therefore, to take benefit of a good work balancing, we must not generate more work in communication than work in computation. experimental results (see Fig. 7) show that a size of about 3 x 3 pixels offers a good compromise.

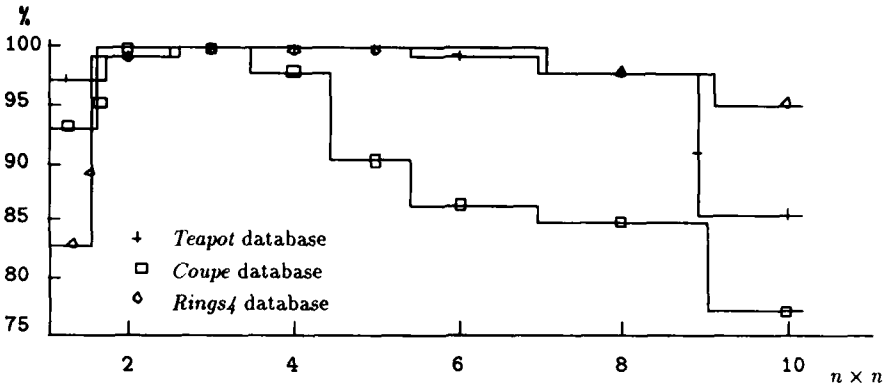


Figure 7: Relative efficiency using different size for an *item of work*.

Results

Tests of our parallel ray tracing has been performed on a set of scenes call *Standard Procedural Databases* (SPD) provide by Eric Haines (see [13]) and other scenes (including the famous *Teapot* from the university of Utah) described with the Neutral File Format (NFF) of Eric Haines. Several synthesis times are given by the table in figure 10.

First results of our algorithm on the iPSC/2 are promising. If we compare the results obtained by this method (cf. Fig. 8 and 9) with our previous work [22, 23], we can emphasize on the improvements brought by the shared database model of programming. The behaviour of this algorithm is what a user of parallel machines expecting. Indeed, the use of more PEs allows to solve problems faster, and to consider larger problems. This is due to the characteristics of the the software global memory management ;

- for a sufficient size of memory cache, the PEs can work rapidly since the number of requests to others is small ;
- the size of the memory cache is flexible. Indeed, with a memory fixed-sized problem, i.e. a fixed-size database, using more PEs increases the computation power of course, but also provides a better memory management as local cache memory increases (see Fig. 5).

One of our goals is to render a database the largest as possible. At present, we have rendered the *tetra10* database which contains more than one million (1 048 576) polygons. The size of this scene with its *object access structure* requires the use of 109 452 pages (× 1 280 Bytes), which represents a shared memory of about 140 MBytes. The synthesis time with 64 nodes is 8 mn 46 sec. We can noticed that this database can not be rendered with 32 nodes (with 4 MBytes of memory per node).

4 Conclusion

The aim of our study to parallelize the ray tracing method is to bring out a model of parallel programming well suited for this kind of algorithm. Due to the difficulty to appreciate the

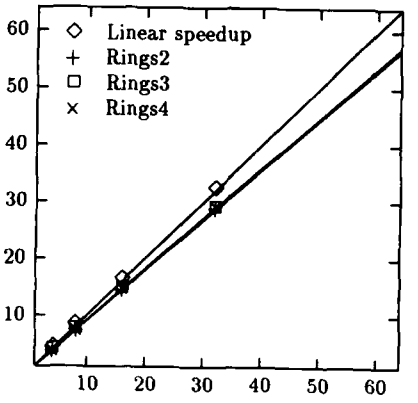


Figure 8: Speedup for the *Rings* images.

# Proc. :	1	2	4	8	16	32	64
Rings2	1.00	0.95	0.93	0.91	0.91	0.90	0.89
Rings3	1.00	0.95	0.92	0.91	0.90	0.89	0.88
Rings4	1.00	0.95	0.93	0.92	0.91	0.90	0.88

Figure 9: Efficiency for the *Rings* images.

Database	# Polygons	Synthesis Time
Teapot	3754	1 mn 59 sec
Coupe	15408	5 mn 44 sec
Rings4	18002	8 mn 48 sec
Tetra9	262144	2 mn 21 sec
Tetra10	1048576	8 mn 46 sec

Figure 10: Examples of synthesis times with 64 nodes.

performance of the various parallel ray tracing algorithms, we have done and keep on doing experiences on an iPSC/2 hypercube. Comparing the behaviour of our first algorithm (see [22, 23]) using a message passing model of programming with the behaviour of the last one, described in this paper, which uses a shared database model of programming, we advocate the shared model approach when using large read-only database with no obvious distribution.

References

[1] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *EUROGRAPHICS'87*, pages 3-9, Amsterdam, 1987.

[2] B. Arnaldi, T. Priol, and K. Bouatouch. A new space subdivision for ray tracing CSG modelled scenes. *Visual Computer*, 3(2):98-108, August 1987.

[3] R. Bisiani, A. Nowatzyk, and M. Ravishankar. *Coherent Shared Memory on a Message Passing Machine*. Technical Report CMU-CS-88-204, Carnegie Mellon, December 1988.

[4] N. Carreiro and D. Gelernter. The s/net's linda kernel. *ACM Transactions Computer Systems*, May 1986.

[5] J.G Cleary, B. Wyvill, G.M. Birtwistle, and R. Vatti. *Multiprocessor Ray Tracing*. Research Report 83/128/17, University of Calgary, October 1983.

[6] J.G. Cleary and G. Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *Visual Computer*, 4(2), July 1988.

[7] M. Cohen and D. Greenberg. The hemi-cube, a radiosity solution for complex environments. *ACM Computer Graphics*, 19(3), 1985.

[8] R.L. Cook. Stochastic sampling in computer graphics. In *Siggraph'86 Tutorial*, SIGGRAPH, 1986.

[9] R.L. Cook and K.E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1):7-24, January 1982.

[10] O. Devillers. *Méthodes d'optimisation du tracé de rayons*. PhD thesis, Laboratoire informatique de l'ENS, rue d'Ulm, Paris, 1988.

[11] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, April 1986.

[12] A.S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10), 1984.

[13] E. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 7(11), November 1987.

[14] A. Roy Hall and Donald P. Greenberg. A testbed for realistic image synthesis. *IEEE Computer Graphics and Applications*, 3(8):10-20, November 1983.

[15] C. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666-677, 1978.

[16] J.T. Kajiya. The rendering equation. In *SIGGRAPH'86*, pages 143–150, SIGGRAPH, August 1987.

[17] M.R. Kaplan. Space-tracing, a constant time ray tracer. In *SIGGRAPH'85 tutorial on the uses of spatial coherence in ray tracing*, 1985.

[18] T.L. Kay and J.T. Kajiya. Ray tracing complex scenes. *ACM Computer Graphics*, 20(4), August 1986.

[19] M.E. Lee, R.A Redner, and S.P. Uselton. Statiscally optimized sampling for distributed ray tracing. *ACM Computer Graphics*, 19(3), 1985.

[20] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. In *Symposium on Principles of Distributed Computing*, pages 229–239, ACM SIGACT-SIGOPS, Calgary, Canada, 1986.

[21] B.T. Phong. Illumination for computer generated pictures. *Communication of the ACM*, 18(6):311–317, June 1975.

[22] T. Priol and K. Bouatouch. Static load balancing for a parallel ray tracing on a MIMD hypercube. *Visual Computer*, 5:109–119, March 1989.

[23] Thierry Priol. *Lancer de rayon sur des architectures parallèles: Etude et mise en œuvre.* PhD thesis, Institut de Formation Supérieure en Informatique et Communication, Rennes, juin 1989.

[24] S.D. Roth. Ray casting for modeling solids. *Computer Graphics and Image Processing*, 18(2), February 1982.

[25] F. Sillion and C. Puech. A general two-pass method integrating specular and diffuse reflection. In *SIGGRAPH'89*, pages 335–344, SIGGRAPH, July 1989.

[26] A.J. Smith. Cache memories. *Computing Surveys*, 14(3), September 1982.

[27] J.M. Snyder and A.H. Barr. Ray tracing complex models containing surface tessellations. *ACM Computer Graphics*, 21(4), July 1987.

[28] T. Whitted. An improved illmination model for shaded display. *Computer Graphics and Image Processing*, 23(6), June 1980.

Domain decomposition methods with non-overlapping subregions and parallel computing .

François-Xavier Roux
O.N.E.R.A.
29 Av de la Division Leclerc
BP 72 92322 CHATILLON Cedex
FRANCE
(1) 46 57 11 60 ext. 22 42

Abstract .

We present the two most classical domain decomposition methods with non overlapping subdomains : a conforming one, the Schur complement method, and a non conforming one, based on introducing a Lagrange multiplier in order to enforce the continuity requirement at the interface of the subdomains .

We show that these methods represent two dual formulations of a condensed problem on the interface .

The problem of the parallel implementation of these methods is adressed, and the results of some numerical experiments for ill conditioned three dimensional structural analysis problems are given .

1. Introduction .

The simplest parallel algorithm for solving elliptic partial differential equations is based on solving the complete problem through the conjugate gradient method with parallelisation of the matrix-vector product. This can be done by performing in parallel the computation for the lines or the rows of the matrix associated with different substructures. But, with sparse matrices arrising from finite element methods, the amount of computation of a matrix-vector product depends in a linear way of the number of variables. As the data transfers depend in a linear way of the number of variables too, the parallelisation of these products typically leads to fine grain parallelism .

Moreover, for large structural analysis problems, using a global conjugate gradient method is really problematic, because of the ill conditioning and the large numbers of degrees of freedom .

It is possible to decrease the dimension of the problem and to get parallel algorithms with large granularity by using domain decomposition methods .

Some of these methods involve overlapping subregions and are derived from the Schwarz alternative principle, see [1] and [2]. It has been proved in [3] that the Schwarz alternative procedure consists in solving the condensed problem associated with the Schur complement operator by the mean of a block Gauss-Seidel algorithm .

Other methods involve non-overlapping subregions. These methods consist in solving a condensed problem on the interface between the sudomains. The condensed operator is defined with the help of the inverses of local matrices associated with independant local problems .

These methods appear to be better suited to finite element or finite difference methods, first, because they lead to solve the condensed problem on the interface through the preconditioned conjugate gradient method, and secondly, because it is generally more difficult to split an unstructured mesh in overlapping than in non-overlapping subregions. And, at last, the Schwarz alternative method is less intrinsically parallel .

In this paper, we present, first, the most standard domain decomposition method with non-overlapping subregions : the Schur complement method .

Secondly, we present a non conforming method, based on introducing a Lagrange multiplier to enforce the continuity requirement at the interface between the sub-domains, that we call the hybrid method, because it is similar to the well known hybrid finite element method for the elasticity equations .

We show that the two methods are associated with dual formulations of the condensed problem on the interface .

Then, we adress some practical problems for implementing these methods, concerning the topology of the interface, and the choice of the local solver .

At last, we give some results with the implementation of the hybrid method for solving an ill conditioned three dimensional structural analysis problem .

In the sequel of this paper we shall use the vocabulary of the linear elasticity. But the methods presented here can, of course, apply to any second order elliptic partial differential equations .

2. The Schur complement method .

The most classical domain decomposition method with non-overlapping subdomains, the so-called Schur complement method, is based on the Gaussian elimination of degrees of freedom inside the substructures . Consider the linear elasticity equations on a domain Ω, and $\mathbf{K}$ the matrix associated with a Lagrangian finite element approximation of the displacement fields . Split the domain Ω into two open subsets Ω_1 and Ω_2 with Γ_3 the inner intersection of the boundaries Γ_1 and Γ_2 of Ω_1 and Ω_2 .

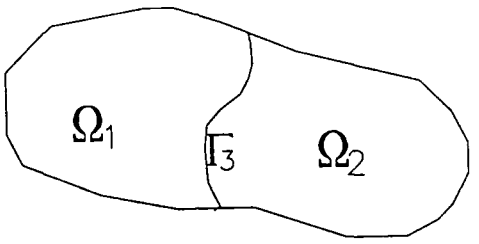


Figure 1 : non-overlapping domain decomposition .

The stiffness matrix associated with the renumbering of the degres of freedom according to the splitting of the domain into these three subsets can be written in the block form below :

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{11} & 0 & \mathbf{K}_{13} \\ 0 & \mathbf{K}_{22} & \mathbf{K}_{23} \\ \mathbf{K}_{13}^t & \mathbf{K}_{23}^t & \mathbf{K}_{33} \end{bmatrix}$$

The stiffness matrices associated with the linear elasticity equations on Ω_1 and Ω_2 with Neumann boundary condition on Γ_3 are:

$$\mathbf{K}^{(1)} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{13} \\ \mathbf{K}_{13}^t & \mathbf{K}_{33}^{(1)} \end{bmatrix} \text{ and } \mathbf{K}^{(2)} = \begin{bmatrix} \mathbf{K}_{22} & \mathbf{K}_{23} \\ \mathbf{K}_{23}^t & \mathbf{K}_{33}^{(2)} \end{bmatrix}$$

The coefficients of the matrices $\mathbf{K}_{33}^{(1)}$ and $\mathbf{K}_{33}^{(2)}$ are the contributions from the integrals over Ω_1 and Ω_2 of the basis functions associated to the nodes of Γ_3, and so $\mathbf{K}_{33} = \mathbf{K}_{33}^{(1)} + \mathbf{K}_{33}^{(2)}$.

To solve the global problem ,

$$\mathbf{K}\,\mathbf{x} = \mathbf{b} \quad \textit{with} \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix} \quad \textit{and} \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix},$$

one can perform a Gaussian elimination of the degrees of freedom inside the open subsets Ω_1 and Ω_2 and get the following condensed problem, involving only the degrees of freedom on Γ_3 :

$$\left[\mathbf{K}_{33} - \mathbf{K}_{13}^t \mathbf{K}_{11}^{-1} \mathbf{K}_{13} - \mathbf{K}_{23}^t \mathbf{K}_{22}^{-1} \mathbf{K}_{23} \right] \mathbf{x}_3 = \mathbf{b}_3 - \mathbf{K}_{13}^t \mathbf{K}_{11}^{-1} \mathbf{b}_1 - \mathbf{K}_{23}^t \mathbf{K}_{22}^{-1} \mathbf{b}_2 \qquad (1)$$

The associated matrix, the Schur complement matrix S, is symmetric and positive definite (see [4]). This is proved by the following equality that consists in a change of basis for the dot product associated with the K matrix :

$$\begin{bmatrix} \mathbf{I} & 0 & 0 \\ 0 & \mathbf{I} & 0 \\ -\mathbf{K}_{31}\mathbf{K}_{11}^{-1} & -\mathbf{K}_{32}\mathbf{K}_{22}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{K}_{11} & 0 & \mathbf{K}_{13} \\ 0 & \mathbf{K}_{22} & \mathbf{K}_{23} \\ \mathbf{K}_{31} & \mathbf{K}_{32} & \mathbf{K}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{I} & 0 & -\mathbf{K}_{11}^{-1}\mathbf{K}_{13} \\ 0 & \mathbf{I} & -\mathbf{K}_{22}^{-1}\mathbf{K}_{23} \\ 0 & 0 & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{11} & 0 & 0 \\ 0 & \mathbf{K}_{22} & 0 \\ 0 & 0 & \mathbf{S} \end{bmatrix} .$$

So, the problem (1) can be solved through the conjugate gradient method without actually computing the coefficients of the matrix S .

Let x_3 be a given displacements vector on Γ_3. The product Sx_3 is given by :

$$S\,x_3 = S^{(1)}\,x_3 + S^{(2)}\,x_3$$

$$\textit{with} \quad S^{(1)} = K_{33}^{(1)} - K_{13}^{t}\,K_{11}^{-1}\,K_{13} \quad \textit{and} \quad S^{(2)} = K_{33}^{(2)} - K_{23}^{t}\,K_{22}^{-1}\,K_{23}$$

Computing the product $S^{(1)}\,x_3$ involves two steps .

First step, the solution of the Dirichlet problem on Ω_1 with the boundary conditions given by x_3 on Γ_3 :

$$K_{11}\,x_1 = -\,K_{13}\,x_3$$

Second step, the computation of the product of the $K^{(1)}$ matrix by the vector $(x_1,x_3)^t$ that is easily shown to be equal to $(0 , S^{(1)}u_3)^t$.

So, solving the equation (1) through the conjugate gradient method gives a parallel algorithm with a very good granularity because the main part of the work consists in the computation of local independant contributions to the product by the Schur complement matrix, that involves mainly the solution of independant local problems .

3. A preconditioner for the Schur complement method .

In the previous section, we have seen that the product by the local Schur complement matrix can be computed by solving a problem with Dirichlet boundary conditions on the interface and then computing the trace of the corresponding internal forces .

That leads to the following equation :

$$\begin{bmatrix} 0 \\ S^{(1)}\,x_3 \end{bmatrix} = \begin{bmatrix} K_{11} & K_{13} \\ K_{31} & K_{33}^{(1)} \end{bmatrix} \begin{bmatrix} K_{11} & K_{13} \\ 0 & I \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ x_3 \end{bmatrix} . \qquad (2)$$

The Schur complement matrix is the discrete operator associated with the mapping of the trace of displacements field on the interface onto the trace of the internal forces. This mapping is a so called Steklov-Poincaré's operator (see [5]) .

So, the inverse of the local Schur complement matrix can be computed by mapping the trace of the internal forces field onto the trace of displacements field on the interface. That leads to solve a local problem with Neumann boundary conditions .

Let us note :

$$\begin{bmatrix} x_1 \\ x_3 \end{bmatrix} = \begin{bmatrix} K_{11} & K_{13} \\ 0 & I \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ x_3 \end{bmatrix} ,$$

equation (2) shows that x_3 is the restriction on Γ_3 of the solution of the Neumann problem (7) :

$$\begin{bmatrix} K_{11} & K_{13} \\ K_{31} & K_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ S^{(1)}\,x_3 \end{bmatrix} .$$

An efficient preconditioner for the condensed problem associated with the Schur complement matrix can be build with the following shape :

$$M = D_1\,[S^{(1)}]^{-1}\,D_1^{t} + D_2\,[S^{(2)}]^{-1}\,D_2^{t} ,$$

where the D_i matrices are weighting matrices such that $D_1 + D_2 = I_{|\Gamma_3}$ (see, for instance, [6] and [7] .

As the Schur complement matrix is a mapping of the trace of displacements field on the interface onto the trace of the internal forces, the residual of the conjugate gradient algorithm is homogeneous to a forces field, whenever the problem is related to the displacements field. The preconditioner presented here consists in mapping the gradient vector back in the primal space associated to the displacements .

Computing this preconditioner leads to the same degree of parallelism as the plain algorithm, because it consists mainly in solving independant local Neumann problems, and then assembling the local contributions on the interface .

4. The hybrid finite element method .

Another domain decomposition method is based on a mechanical approach and involves the introduction of a Lagrange multiplier on the interfaces to remove the continuity constraint .

The equations of the linear elasticity equations with homogeneous boundary conditions are :

$$\begin{cases} \mathbf{Au} = \mathbf{f} & \text{in } \Omega \\ \mathbf{u} = 0 & \text{on } \Gamma_0 \end{cases} \qquad \text{with } (\mathbf{Au})_i = -\frac{\partial(a_{ijkh}\varepsilon_{kh}(\mathbf{u}))}{\partial x_j} \tag{3}$$

The usual variational form of this problem consists in finding $\mathbf{u}$ in $(H^1(\Omega))^3$ satisfying the boundary condition $\mathbf{u} = 0$ on Γ_0 wich minimizes the energy functional :

$$\mathbf{I}(\mathbf{v}) = \frac{1}{2}\mathbf{a}(\mathbf{v},\mathbf{v}) - (\mathbf{f},\mathbf{v}) \qquad \text{with} \quad \mathbf{a}(\mathbf{u},\mathbf{v}) = \int_\Omega a_{ijkh}\varepsilon_{kh}(\mathbf{u})\varepsilon_{ij}(\mathbf{v})\,dx \quad .$$

Let us consider the same splitting of the domain as in the previous section. For a sake of simplicity, let us assume that the boundary of the interface Γ_3 is embedded in Γ_0 the part of the boundary of Ω with homogeneous Dirichlet conditions. Then, the traces on Γ_3 of the displacements fields $\mathbf{u}$ satisfying the boundary condition $\mathbf{u} = 0$ on Γ_0 belong to the space $(H_{00}^{1/2}(\Gamma_3))^3$.

Solving the linear elasticity equation consists in finding two functions $\mathbf{u}_1$ and $\mathbf{u}_2$, in the functional spaces V1 and V2 of the fields belonging to $(H^1(\Omega_1))^3$ and $(H^1(\Omega_2))^3$ that satisfy the boundary conditions on Γ_1 and Γ_2, which minimize the sum of the energies : $\mathbf{I}(\mathbf{v}) = \mathbf{I}_1(\mathbf{v}_1) + \mathbf{I}_2(\mathbf{v}_2)$, with the continuity constraint : $\mathbf{v}_1 = \mathbf{v}_2$ on Γ_3 .

The dual form of the continuity condition is :

$$(\mathbf{v}_1 - \mathbf{v}_2 , \mu)_{\Gamma_3} = 0 \quad \textit{for each } \mu \textit{ in } [(H_{00}^{1/2}(\Gamma_3))^3]' \quad .$$

The primal hybrid variational principle is based on removing the intersubdomain continuity constraint by introducing a Lagrange multiplier (see for instance [8] or [9]). Under the assumption that the so-called Ladyzenskaia-Babuska-Brezzi condition is satisfied :

$$Sup(\mathbf{v}_1 - \mathbf{v}_2 , \mu) \qquad \geq \quad C\ |\mu|_{[(H_{00}^{1/2}(\Gamma_3))^3]'} \qquad ,$$

$$|(\mathbf{v}_1 , \mathbf{v}_2)|_{\mathbf{v}_1 \times \mathbf{v}_2 = 1}$$

one can show (see [10]) that the problem of minimization with constraint above is equivalent to finding the saddle-point of the Lagrangian :

$$\mathbf{L}(\mathbf{v},\mu) = \mathbf{I}_1(\mathbf{v}_1) + \mathbf{I}_2(\mathbf{v}_2) + (\mathbf{v}_1 - \mathbf{v}_2 , \mu)_{\Gamma_3} \quad . \tag{4}$$

This means finding the fields $(\mathbf{u}_1,\mathbf{u}_2)$ *in* $\mathbf{V1}\times\mathbf{V2}$ and the Lagrange multiplier λ in $[(H_{00}^{1/2}(\Gamma_3))^3]'$ which verify :

$$\mathbf{L}(\mathbf{u},\mu) \leq \mathbf{L}(\mathbf{u},\lambda) \leq \mathbf{L}(\mathbf{v},\lambda) \ ,$$

for each field $\mathbf{v} = (\mathbf{v}_1,\mathbf{v}_2)$ in $\mathbf{V1}\times\mathbf{V2}$, and each μ in $[(H_{00}^{1/2}(\Gamma_3))^3]'$.

Clearly, the left inequality imposes $(\mathbf{u}_1 - \mathbf{u}_2 , \mu)_{\Gamma_3} \leq (\mathbf{u}_1 - \mathbf{u}_2 , \lambda)_{\Gamma_3}$ and so $(\mathbf{u}_1 - \mathbf{u}_2 , \mu)_{\Gamma_3} = 0$ for each μ in $[(H_{00}^{1/2}(\Gamma_3))^3]'$, thus the continuity constraint is satisfied by the solution of the saddle-point problem .

The right inequality implies : $\mathbf{I}_1(\mathbf{u}_1) + \mathbf{I}_2(\mathbf{u}_2) \leq \mathbf{I}_1(\mathbf{v}_1) + \mathbf{I}_2(\mathbf{v}_2)$ for each $(\mathbf{v}_1,\mathbf{v}_2)$ in $(H^1(\Omega))^3$, that means that $\mathbf{u}_1$ and $\mathbf{u}_2$ minimize the sum of the energies on Ω_1 and Ω_2 among the fields satisfying the continuity requirement, and so $\mathbf{u}_1$ and $\mathbf{u}_2$ are the restrictions to Ω_1 and Ω_2 of the solution of the primal problem (3) .

The classical variational interpretation of the saddle-point problem (4) leads to the equations :

$$\begin{cases} \mathbf{A}_1\mathbf{u}_1 + \mathbf{B}_1^*\lambda = \mathbf{f}_1 & \text{in } \Omega_1 \\ \mathbf{u}_1 = 0 & \text{on } \Gamma_0 \cap \Gamma_1 \\ \mathbf{A}_2\mathbf{u}_2 - \mathbf{B}_2^*\lambda = \mathbf{f}_2 & \text{in } \Omega_2 \\ \mathbf{u}_2 = 0 & \text{on } \Gamma_0 \cap \Gamma_2 \\ \mathbf{B}_1\mathbf{u}_1 - \mathbf{B}_2\mathbf{u}_2 = 0 & \text{on } \Gamma_3 \end{cases} \tag{5}$$

$\mathbf{A}_1$ and $\mathbf{A}_2$ are the differential operators of the linear elasticity equations on Ω_1 and Ω_2 with Neumann boundary conditions on Γ_3, and $\mathbf{B}_1$ and $\mathbf{B}_2$ the trace operators over Γ_3 of functions belonging to V1 and V2 .

The analysis of these equations shows that the Lagrange multiplier λ is in fact equal to the interaction forces between the substructures along their common boundary. Clearly, to get independant local displacement

problems, it is necessary to introduce the forces on the interfaces. On a structurai analysis point of view it is hardly a surprise. Nevertheless, the precise functional analysis above is usefull because it allows classical results about finite element approximations for hybrid or mixed differential equations to be used .

5. Discretization of the hybrid formulation .

A discretisation with finite elements of the hybrid formulation (5) leads to the following set of linear equations in which the notations of variables associated to discrete problems are the same than the ones formerly used for the continuous formulation .

$$\begin{cases} \mathbf{K}^{(1)}\mathbf{u}_1 + \mathbf{B}_1^t\boldsymbol{\lambda} = \mathbf{f}_1 \\ \mathbf{K}^{(2)}\mathbf{u}_2 - \mathbf{B}_2^t\boldsymbol{\lambda} = \mathbf{f}_2 \\ \mathbf{B}_1\mathbf{u}_1 - \mathbf{B}_2\mathbf{u}_2 = 0 \end{cases} \tag{6}$$

By elimination of the displacements in the equation (6), the problem can be written with respect with $\boldsymbol{\lambda}$ only :

$$\left[\mathbf{B}_1 \mathbf{K}^{(1)^{-1}} \mathbf{B}_1^t + \mathbf{B}_2 \mathbf{K}^{(2)^{-1}} \mathbf{B}_2^t \right] \boldsymbol{\lambda} = \mathbf{B}_1 \mathbf{K}^{(1)^{-1}} \mathbf{f}_1 - \mathbf{B}_2 \mathbf{K}^{(2)^{-1}} \mathbf{f}_2$$

So $\boldsymbol{\lambda}$ satisfies the following equation :

$$\mathbf{D}\,\boldsymbol{\lambda} = \mathbf{b} \tag{7}$$

$$\textit{with } \mathbf{D} = \begin{bmatrix} \mathbf{B}_1 & -\mathbf{B}_2 \end{bmatrix} \begin{bmatrix} \mathbf{K}^{(1)^{-1}} & 0 \\ 0 & \mathbf{K}^{(2)^{-1}} \end{bmatrix} \begin{bmatrix} \mathbf{B}_1^t \\ -\mathbf{B}_2^t \end{bmatrix} \quad \textit{and } \mathbf{b} = \begin{bmatrix} \mathbf{B}_1 & -\mathbf{B}_2 \end{bmatrix} \begin{bmatrix} \mathbf{K}^{(1)^{-1}} & 0 \\ 0 & \mathbf{K}^{(2)^{-1}} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix}$$

Obviously, the **D** matrix is symmetric positive. It is definite if the interpolation spaces chosen for $\mathbf{u}_1$ and $\mathbf{u}_2$ and $\boldsymbol{\lambda}$ satisfy the discrete Ladyzenskaia-Babuska-Brezzi condition .

To be able to use the standard approximation results for the mixed or hybrid formulations, it is necessary to find such finite elements spaces that the discrete Ladyzenskaia-Babuska-Brezzi condition is uniformly satisfied according to h, the mesh size parameter .

But, generally, checking the uniform Ladyzenskaia-Babuska-Brezzi condition for the discrete problem may be tough (see [11] and [12]). The finite elements used for the Lagrange multiplier must be associated with polynomials of one degree less than the ones used for the primal unknowns, as the Lagrange multiplier is homogeneous to some partial derivatives of the solution of the primal problem. When using the hybrid formulation as described above to get a domain decomposition method, the Lagrange multiplier is introduced just to enforce the continuity condition. The values of the discrete multiplier do not need to be a good approximation of the continuous interaction forces between the substructures .

So, satisfying the uniform Ladyzenskaia-Babuska-Brezzi condition is not necessary in this case. The discrete form of the continuity constraint : $\mathbf{v}_1 = \mathbf{v}_2$ on Γ_3, can be written simply : $\mathbf{v}_1 = \mathbf{v}_2$ for each degree of freedom located on Γ_3 .

With such a condition, the discrete $\mathbf{B}_i$ matrices are just boolean restriction matrices .

Taking this approximation is equivalent to have finite elements associated with the same polynomials for the Lagrange multiplier $\boldsymbol{\lambda}$ and for the dispacements fields, and to take a collocation approximation for the integral :

$$\int_{\Gamma_3} (\mathbf{v}_1 - \mathbf{v}_2)\, \boldsymbol{\mu}\, d\gamma \ .$$

The uniform Ladyzenskaia-Babuska-Brezzi condition for the discrete problem is not satisfied. But the displacements fields $\mathbf{u}_1$ and $\mathbf{u}_2$, solution of the discrete hybrid problem, are conforming, due to the condition : $\mathbf{u}_1 = \mathbf{u}_2$ for each degree of freedom located on Γ_3. So these fields are, in fact, the restriction over the two subdomains of the solution of the discrete primal global problem, for which the standard approximation results apply .

As a consequence of this form of discretization, one can see that the method can be applied even though the boundary of the interface Γ_3 is not embedded in Γ_0 the part of the boundary of Ω with homogeneous Dirichlet conditions .

6. Solution of the discrete hybrid problem .

The problem (7) can be solved through the conjugate gradient method for it is possible to compute the product of the **D** matrix by a vector, although the matrix itself is never computed .

Let $\boldsymbol{\mu}$ be a vector, computing the product $\mathbf{v} = \mathbf{D}\,\boldsymbol{\mu}$ involves the following three steps .

Step one :

$$\text{computation of the matrix-vector , } \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} B_1^t \\ -B_2^t \end{bmatrix} \begin{bmatrix} \mu \\ \mu \end{bmatrix} ,$$

that is just a reordering operation because the B_i matrices are boolean .

Step two :

$$\text{computation of the product , } \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} [K^{(1)}]^{-1} \\ [K^{(2)}]^{-1} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} ,$$

that means computing the solution of two independant local sets of linear equations associated with the local linear elasticity problems with Neumann boundary conditions on the interface Γ_3 .

Step three : computation of the variation on the interface of the displacements fields w_1 and w_2 ,

$$v = \left[B_1 , - B_2 \right] \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = B_1 w_1 - B_2 w_2 .$$

Obviously, the main step is the second one and can be performed in parallel, whereas only the step three involves interprocessor data transfers. So, this method leads to a parallel algorithm with the same kind of granularity as the Schur complement method .

The B_i matrices obtained with the discrete hybrid method presented in the previous section are boolean matrices. So, the contribution $D^{(1)}$ of the subdomain number 1 to the dual interface matrix is equal to :

$$D^{(1)} = B_1 [K^{(1)}]^{-1} B_1^t = \begin{bmatrix} 0 & I \end{bmatrix} \begin{bmatrix} K_{11} & K_{13} \\ K_{13}^t & K_{33}^{(1)} \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ I \end{bmatrix} .$$

Let us note :

$$\begin{bmatrix} C_1 \\ C_3^{(1)} \end{bmatrix} = \begin{bmatrix} K_{11} & K_{13} \\ K_{13}^t & K_{33}^{(1)} \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ I \end{bmatrix} .$$

Then $D^{(1)}$ is equal to the matrix $C_3^{(1)}$.

From the previous relation, one can see that the C_1 and $C_3^{(1)}$ matrices satisfy the following equations :

$$K_{11} C_1 + K_{13} C_3^{(1)} = 0$$
$$K_{31} C_1 + K_{33}^{(1)} C_3^{(1)} = I$$

Hence, by elimination of the C_1 matrix in the previous equations, we can derive the folowing equality :

$$- K_{31} K_{11}^{-1} K_{13} C_3^{(1)} + K_{33}^{(1)} C_3^{(1)} = \left(K_{33}^{(1)} - K_{31} K_{11}^{-1} K_{13} \right) C_3^{(1)} = I .$$

So, the $D^{(1)}$ matrix is, in fact, the inverse of the Schur complement matrix $S^{(1)}$.

On a functional analysis viewpoint, the hybrid method is the dual method of the Schur complement method, because the condensed problem with the hybrid method is related to the forces on the interface, when the Schur complement operator is related to the displacements field on the interface .

On the linear algebra viewpoint, with the discretization presented here, the duality of the two methods is simply represented by the following relation between the two interface operators :

$$D^{(i)} = \left[S^{(i)} \right]^{-1} .$$

7. Topology of the interface for conforming and non conforming domain decomposition methods .

There are some features of the interface topology which could make the domain decomposition method with Lagrange multiplier more suitable for a parallel implementation than the conforming Schur complement method .

When a point belongs to several subdomains, the coefficients of the condensed matrix for the degrees of freedom associated with this point will be the sum of the contributions of the various subdomains to which the point belongs. As regards data dependency within the context of the implementation of the method on multi-processor systems, each processor performing the computation associated with one subdomain, it means that the result of the product by the Schur complement matrix for such nodes will depend on more than two local contributions .

In a distributed memory context it means that subdomains are neighboring from the moment that they have just one common node. In the case of a chessboard decomposition of a two dimension domain, each subdomain has eight neighbours. For real three-dimensional topology, the number of neighbours can be very large. For a decomposition in cubes, for instance, there would be as many neighbours as the sum of the numbers of faces, edges and vertices of a cube that is 26 .

In a shared memory context, there is no problem with data transfers, but the assembly of the result of the product by the Schur complement matrix is still complex, due to the fact that the number of local contributions depends on the location of the point .

Let us now consider the domain decomposition method with Lagrange multiplier. The only sequential part of the computation of the product of the dual matrix **D** by a vector lies in the third step consisting in the computation on each interface of the variation of the displacements fields :

$$\begin{bmatrix} \mathbf{B}_1 & -\mathbf{B}_2 \end{bmatrix} \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix} = \mathbf{B}_1 \mathbf{w}_1 - \mathbf{B}_2 \mathbf{w}_2 \ .$$

The $\mathbf{B}_1$ and $\mathbf{B}_2$ matrices are the discrete operators associated with the weak formulation of the continuity of the displacement fields on the interface :

$$(v_1 - v_2 , \mu)_{\Gamma_3} = 0 \quad \textit{for each } \mu \textit{ in } [(\mathbf{H}_{00}^{1/2}(\Gamma_3))^3]' \ .$$

If the interface Γ_3 between two subdomains has a zero integral, this equation vanishes. In fact $\mathbf{B}_1$ and $\mathbf{B}_2$ are discrete trace operators, and the continuous trace operators are defined only on subsets of the boundary with non zero integrals. This is still true even though taking the same degrees of freedom for the Lagrange multiplier than for the restrictions over the interface of the displacements fields, as it was presented in the previous section .

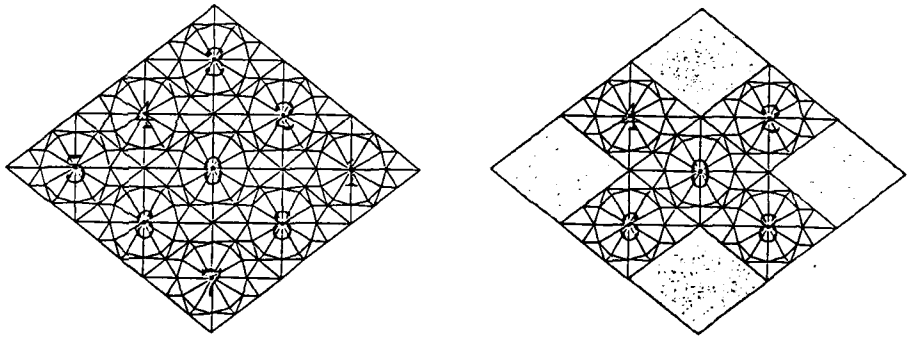


Figure 2 : neighboring domains with the Schur complement method and the hybrid method .

For each degree of freedom located on a point belonging to more than two subdomains, there are then as many degrees of freedom for the Lagrange multiplier as pairs of subdomains whose interface has a non zero integral .

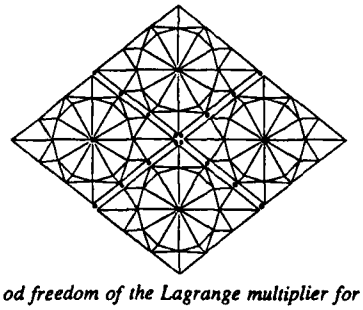


Figure 3 : degrees od freedom of the Lagrange multiplier for intersecting edges .

In the case of a chessboard decomposition of a two-dimensional domain, each subdomain has only four neighbours, one for each edge. Each degree of freedom for the displacements fields located at a vertex is associated with four degrees of freedom for the Lagrange multiplier, one for each edge intersecting at the vertex. For a decomposition in cubes of a three-dimensional domain, there are as many neighbours as faces, i.e. only six .

For implementation on a parallel system with local memory, it means that the number of processors connected to each node of the system needs to be equal to four for a two-dimensional splitting, with the same topology as for a finite difference grid, and equal to six for three-dimensional decomposition, still with the topology of a three-dimensional regular grid. In both cases the number of neighbors is obviously minimum .

In both shared memory or distributed memory contexts, the assembly of the product of a vector by the dual interface matrix D is simpler because all the points have the same status, and there are exactly two local contributions to the computation of the product for all the interface nodes .

8. Presentation of a structural analysis problem for a composite beam .

The test problem we consider consists in solving the linear elasticity equations for a composite beam made of a little more than one hundred stiff carbon fibers bound by an uncompressible elastomer matrix .

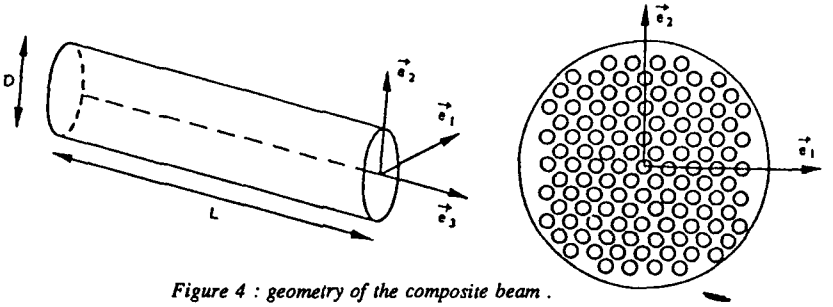


Figure 4 : geometry of the composite beam .

Homogeneization methods do not work for such a device with macroscopic-scale discontinuity and very different materials. For instance the Young modulus in the direction of the axis of the beam is 53000 MPa for the fibers and 7.8 MPa for the elastomer. But, due to the composite feature, the finite element mesh for solving the problem with discontinuous coefficients must be very refined, for it must discern the material discontinuity. That leads to a very large matrix, so the problem can be solved only through iterative methods like the conjugate gradient method .

Figure 5 : a composite "pencil" .

However, substructuring is very easy in the present case for the beam is made of similar jointed composite "pencils" consisting in one of the fibers with its elastomer matrix.

Furthermore, it must be noticed that the problem we tackle is very ill conditioned .

First, for geometrical reason when trying to solve the pure bending problem, i.e. the case of a fixed bottom of the beam and transverse stresses on the top. The condition number of the matrix of the problem increases with the ratio of the length of the beam upon the width. This is exactly the case we are the most interested in solving .

Secondly, for material reasons, because of the composite feature, and because of the uncompressibility of the elastomer. To enforce this constraint, we introduce a penalty parameter and the condition number increases when this parameter tends to 0 .

So, this problem gives a very good example of a stiff mechanical engineering problem with natural substructuring. For both the ill conditioning and the high dimension, only supercomputers allow to tackle it. For the same reason and because substructuring is straightforward, it is a very interesting problem for testing domain decomposition methods on parallel supercomputers.

Moreover, it is simple to build smaller test problems with the same features in solving the elasticity equations in domains made of only a few pencils, with the same global ratio of the length upon the width than for the complete beam .

9. Choice of the local solver .

The first tests of the hybrid finite element method have been performed with solution of the independant local subproblems through the conjugate gradient algorithm .

According to expectations, the hybrid method gave good results under the parallelism point of view. But comparison with the conforming global conjugate gradient method with a parallel matrix-vector product, yielded to the conclusion that the non-conforming hybrid method was generally much more expensive, although the conforming conjugate gradient is less efficient for parallel processing .

The reason is clear : the ratio of the length upon the width is higher for one pencil than for a beam made of several pencils. Thus, substructuring leads to local problems with condition number greater than the one of the conforming primal problem. Then the choice of the conjugate gradient method for solving the local equations yields to a more expensive algorithm .

Clearly, the solution consists in using a direct method for solving the local problems. It is possible because the number of degrees of freedom in the substructures can be much smaller than the one of the complete domain. Furthermore, as each local set of equations needs to be solved several times, the time for the LU decomposition is not predominant as it is generally the case for direct solution methods .

This problem with the condition number of the local matrices seems to be linked with special features of the particular problem we try to solve. But the crucial point with domain decomposition methods is the convergence speed of the outer iterative process, because each iteration requires the solution of all local problems and, thus, is very expensive .

Thus, it is better to locate the interfaces in regions where the solutions are smooth. And as a consequence the ill conditioning due to geometry may well be worse in the subdomains .

Moereover, iterative methods are sometimes faster than direct methods because of the cost of the LU factorization of the matrices. But when there are many right hand sides, and it is the case with domain decomposition methods because of the outer iterative procedure, direct methods are generally more effective .

Furthermore, the domain splitting may be performed in such a way that the substructures have a slender shape, in order to get small bandwidths for the local matrices. So, the cost of the computation of the Choleski factorization of all the local matrices is much lower, in both CPU time and memory requirements, than for the matrix of the complete problem .

At last, it is clear that the use of iterative methods for solving the local problems prevents optimal load balancing because the number of operations cannot be forecasted .

The hybrid method with solution of the local problems through the Choleski method has been implemented on CRAY-2 and CRAY-YMP832 machines for the three dimensional problem presented in the previous section of the paper .

Tests have been performed with subdomains consisting in one or a few pencils and numbers of subdomains between four and sixteen .

With highly optimized backward and forward substitutions for the local problems, (see [13]), speed-ups have been more than 3 on CRAY2 and more than 7 on CRAY-YMP. These performances are nearly the best possible, due to the problems with memory contention on these machines. The global computation speeds have been 700 Mega-flops on CRAY2 and a little bit less than 2 Giga-flops on CRAY-YMP .

These results prove the ability of algorithms based on domain decomposition methods with solution of the local problems through direct solvers to yield maximum performances with vector and parallel supercomputers .

10. Some comparisons of the performances of the hybrid domain decomposition method and the global Choleski facturization .

The table above presents some results obtained with a direct global solver, the Choleski factorization, and the hybrid domain decomposition method, for three different test problems .

In all the cases, we computed the results of the pure bending problem for three dimensional beams, consisting in four, nine, or sixteen composite pencils .

We indicate the number of subdomains, each subdomain made of one pencil, the global number of degrees of freedoms, the number of degrees of freedom on the interface, and the CPU times and memory requirements .

The Poisson ration for the elastomere is 0.49 .

The CPU times include the time for the Choleski factorization of all the local matrices for the hybrid domain decomposition method .

The stopping criterion for the outer conjugate gradient algorithm is :

$$| \mathbf{u}_n - \mathbf{u} | / | \mathbf{u} | \leq 10^{-8} \text{ , and } | \mathbf{K}\mathbf{u}_n - \mathbf{b} | / | \mathbf{b} | \leq 10^{-4} \text{ ,}$$

where $\mathbf{u}$ is the result obtained with the global Choleski factorization, and $\mathbf{K}$ the global stiffness matrix .

number of subdomains	4	9	16
global number of d.o.f	13000	28000	48000
	hybrid domain decomposition method		
number of interface d.of.	2450	7350	14700
number of iterations	130	210	300
cpu time(s)	20	73	193
memory size(mw)	1.6	4.5	9.5
	global Choleski factorization		
cpu time(s)	15	130	650
memory size(mw)	3.6	16	50

These tests show that the domain decomposition method may be, with a well suited splitting of the domain, more efficient than the direct solution, on both memory requirements and CPU time viewpoints, even for very ill conditioned three-dimensional problems .

Furthermore, the domain decomposition method is much better suited for parallel processing, and it can be efficiently implemented on distributed memory machines, because the main part of the data lies in the LU decomposition of the matrices of the local problem that can be located in local memories. The data transfers involve only the traces of the fields on the interface, and, so, are several orders of magnitude smaller than the number of operations to be performed in parallel for solving the local sub-problems (see [14] for a parallel implementation on an Intel hypercube machine of a preconditioner based on the Schur complement method) .

11. Conclusions .

The Schur complement or the hybrid domain decompositions methods appear, in practice, to have mixed characteristics of direct and iterative solution methods .

They are iterative methods, because they consist in solving an interface problem through the preconditioned conjugate gradient method. Like other iterative methods, they entail lower memory filling than direct methods, because only the LU factorization of small local matrices are to be stored .

But with domain decomposition methods, the dimension of the problem to be solved through an iterative method is much smaller than the dimension of the complete problem. And the matrix of the condensed problem on the interface is much denser than the usually sparse matrix of the complete problem. And its condition number is lower, because the elimination of the variables associated with the internal nodes represents some kind of block Jacobi preconditioner .

These characteristics make these domain decomposition methods, when using direct local solvers, much more robust than standard iterative methods. They represent a good way to use direct solvers for problems with such large numbers of degrees of freedom that the solution of the complete problems through a LU factorization would not be affordable .

The tests presented in the previous section show that domain decomposition methods can, with a well suited splitting of the domain, be less expensive than direct solvers in both CPU time and memory requirements, even fore very ill conditioned three dimensional structural analysis problems .

Furthermore, these methods lead to a very high degree of parallelism, and they are very well suited for being implemented on parallel systems with local memories, like distributed memory or hierarchical memory machines.

An open question that needs to be solved to make such algorithms general purpose solvers lies in the problem of mesh splitting and interface localization .

Finding an optimal substructuring requires to take into consideration different problems .

The subdomains must have such a shape that the local matrices have a low bandwidth in order to make the use of direct local solver efficient. That may lead to large interfaces. But, the less points there are on the interface, the smaller the dimension of the dual problem is, that should be better to ensure fast convergence of the outer conjugate gradient .

Furthermore, the condition numbers of the local problems and of the dual problem depend upon the aspect ratio of the substructure and of the interface .

To get round the local ill conditioning, the use of direct local solvers and a reorthogonalization process for the outer conjugate gradient (see [15]) seem to be effective .

But the repercussions for the condition number of the dual interface operator of the geometry of the decomposition are difficult to anticipate, because they depend not only on the aspect ratio of the interface but also on the mechanical features of the global problem .

References .

[1] P.L. lions , "On the Schwarz Alternative Principle : I", in *Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations, Paris, 1987*, R. Glowinski, G.H. Golub, G. Meurant, and J. Periaux, eds., SIAM, Philadelphia, 1988 .

[2] P.L. lions , "On the Schwarz Alternative Principle : II", in *Proceedings of the Second International Conference on Domain Decomposition Methods for Partial Differential Equations, Los Angeles, 1988*, T.F. Chan, et al, eds., SIAM, Philadelphia, 1989 .

[3] T. F. Chan and D. Goovaerts , " Schwarz = Schur : overlapping versus non overlapping domain decomposition ," to appear in *SIAM J. Sci. Stat. Comp.* .

[4] P.E. Bjordstad and O.B. Wildlund , "Iterative methods for solving elliptic problems on regions partitioned into substructures" , *SIAM J. Numer. Anal.* Vol. 23, No. 6, December 1986 .

[5] V. I. Agoshkov , " Poincaré's-Steklov operators and domain decomposition methods ," in *Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations, Paris, 1987*, R. Glowinski, G.H. Golub, G. Meurant, and J. Periaux, eds., SIAM, Philadelphia, 1988 .

[6] P. Le Tallec , J.F. Bourgat , R. Glowinski and M. Vidrascu , " Variational Formulation and Conjugate Gradient Algorithm for Trace Operator in Domain Decomposition Method ," in *Proceedings of the Second International Symposium on Domain Decomposition Methods for Partial Differential Equations, Los Angeles, 1988*, T.F. Chan, et al, eds., SIAM, Philadelphia, 1989 .

[7] Y.-H. De Roeck , " A local preconditioner in a domain decomposed method ," CERFACS Report TR89/10, March 1989, 31057 Toulouse Cedex, France .

[8] T.H.H. Pian and P. Tong , "Basis of finite element methods for solid continua" , in *Internat. J. Numer. Methods. Engrg.* , vol. 1 ,1969, pp. 3-28 .

[9] T.H.H. Pian , "Finite element formulation by variational principles with relaxed continiuty requirements" , in *The Mathematical Foundation of the Finite Element Method with Applications to Partial Differential Equations* , Part II , A.K. Aziz, ed., Academic Press, New-York, 1972, pp. 671-687 .

[10] F. Brezzi, "On the existence, uniqueness and approximation of saddle-point problems arising from Lagrangian multipliers", *R.A.I.R.O., Analyse Numerique*, 8, R2, 1974 .

[11] P.A. Raviart and J.M. Thomas , "Primal hybrid finite element methods for 2nd order elliptic equations" , *Math. of Comp.* , vol. 31,number 138,pp. 391-413 .

[12] V. Girault and P.A. Raviart , *Finite Element Methods for Navier-Stokes Equations* , Springer-Verlag, Berlin, 1986 .

[13] F.-X. Roux , "Tests on parallel machines of a domain decomposition method for a structural analysis problem", *Proceedings of 1988 International Conference on Supercomputing, 1988, St. Malo, France* .

[14] D. Goovaerts, R. Piessens, "Implementation of a domain decomposed preconditioner on an iPSC/2", *Parallel Computing 1989*, D.J. Evans and F.J. Peters eds., Elsevier, Amsterdam, 1989 .

[15] F.-X. Roux, "Acceleration of the outer conjugate gradient by reorthogonalization for a domain decomposition method with Lagrange multiplier", to appear in *Proceedings of SIAM Fourth International Conference on Domain Decomposition Methods, March 20-22, 1989, Houston, Texas* .

DATA MANAGEMENT FOR 3-D ADI ALGORITHM ON HYPERCUBE APPLICATION TO THE DESIGN OF A PARALLEL NAVIER-STOKES SOLVER

P. LECA and L MANE
ONERA, Parallel Computing Subdivision
B.P. 72, 92322 Chatillon Cedex, France

ABSTRACT

The success of highly parallel distributed memory multiprocessors will depend mainly on their efficiency when running realistic application codes. This paper concerns the adaptation to hypercube multiprocessors of a 3-D Navier-Stokes solver based on ADI algorithm. Two solutions for the management of data transfer through the communication network are discussed. Performance results of the implementation on a 32 nodes iPSC2-SX are also presented.

Keywords: Navier-Stokes equations. ADI algorithms. Parallel computers. Hypercube. Distributed memory multiprocessors.

1. Introduction. Numerical simulation of three-dimensional time- dependant flows is a field of CFD which needs a large computational power so as to yield realistics results. In this paper we study a parallel solver for 3-D unsteady Navier-Stokes partial differential equations adequated to distributed memory multiprocessors. Section 2 is devoted to the numerical method and the algorithm used to solve the N-S equations. The main part of the solver consists in the solution of Poisson equation by a modified ADI algorithm. Section 3 discusses some mapping strategies on parallel processors and give performances obtained on the iPSC2-SX using a ring ADI algorithm's. Section 4 describes a so-called hypercube ADI algorithm's which used i-cycle index-digit permutations to move data all over the processors. Finally Section 5 compares the two approaches for different architectural parameters of the communication network.

2. The 3-D Navier-Stokes solver. For several years ONERA [1] and LIMSI [2] collaborates on the development of 2-D and 3-D parallel Navier-Stokes solvers for shared and distributed memory multiprocessors. A 2-D solver for the numerical simulation of unsteady separated flows around an airfoil at high Reynolds numbers has been implemented on a shared memory multiprocessor [4]. A 3-D parallel solver is now under development [5], which is based on the ADI method as exposed in the following sections.

2.1. Navier-Stokes equations and numerical method. The unsteady 3-D Navier-Stokes equations for incompressible flows are written following a velocity-vorticity ($\vec{V} - \vec{\omega}$) formulation [8]:

$$\Delta\vec{V} + \vec{\nabla} \times \vec{\omega} = \vec{0} \tag{2.1}$$

$$\frac{\partial\vec{\omega}}{\partial t} + (\vec{V} \cdot \vec{\nabla})\vec{\omega} = (\vec{\omega} \cdot \vec{\nabla})\vec{V} + \nu\Delta\vec{\omega} \tag{2.2}$$

The equations 2.1 and 2.2 are approximated by using a centred finite difference method. Then the alterning direction method of [3] is used for both the Poisson and the vorticity transport equations so as to obtain a numerical solution with a precision of order 2 in space and 1 in time.

2.2. ADI algorithm for 3-D Poisson equation. The iterative algorithm which has been chosen for the solution of the equation 2.1 is a fractional step method with stabilization corrections [6], which is a generalization of the Douglas ADI method [7].
The equation :

$$(L_x + L_y + L_z)\mathbf{U} = \Phi \tag{2.3}$$

with $L_x = \frac{\partial^2}{\partial x^2}$, $L_y = \frac{\partial^2}{\partial y^2}$, $L_z = \frac{\partial^2}{\partial z^2}$

is then solved by iterating the three steps of 2.4 :

$$\begin{array}{l} \textit{Iteration } n : \\ (L_x - 2\omega_{n,x}\mathbf{I})\mathbf{U}^{n+\frac{1}{3}} = -(L_x + 2L_y + 2L_z + 2\omega_{n,x}\mathbf{I})\mathbf{U}^n + 2\Phi \\ (L_y - 2\omega_{n,y}\mathbf{I})\mathbf{U}^{n+\frac{2}{3}} = L_y\mathbf{U}^n - 2\omega_{n,y}\mathbf{U}^{n+\frac{1}{3}} \\ (L_z - 2\omega_{n,z}\mathbf{I})\mathbf{U}^{n+1} = L_z\mathbf{U}^n - 2\omega_{n,z}\mathbf{U}^{n+\frac{2}{3}} \end{array} \tag{2.4}$$

where $\omega_{n,x}, \omega_{n,y}, \omega_{n,z}$ are acceleration parameters.
Considering data dependencies, each one of these three steps is related to a linear recurrence equation in one of the space directions [10].

[1] Office National d'Etudes et de Recherches Aerospatiales
[2] Laboratoire d'Informatique et de Mathématiques pour les Sciences de l'Ingénieur BP 30, 91406 Orsay Cedex, France

3. Distributed memory multiprocessors and mapping strategies. Distributed memory architectures appear as a solution for the design of highly parallel multiprocessors. In such systems a set of processors (nodes) is connected in some fixed topology through a communication network. As the opposite of shared memory system, data allocation onto processors local memory is one of the key to efficiently exploit the potential parallelism of an algorithm. A determinate solution for data allocation may inhibit the activity of a part of the processors. Hence the systematic analysis of data dependencies is an obligatory step in the design of well adapted codes for such highly parallel machines. Moreover most of these machines do not presently hide the parallelism to the user. Data flow between processors must be expressed explicitly by means of message passing primitives.
Efficient algorithms for these multiprocessors are often the result of a trade-off between the reduction of the time due to data communication in the network and the reduction of the computation time. Moreover sending or receiving a message is an operation which has an incompressible cost. Then some gain may be obtained by reducing the number of messages flowing through the network. The next sections discuss the influence of the mapping strategies on the data structure of the computational domain when considering ADI algorithm.

3.1. Data structures for ADI algorithm. Considering the computational domain as a cube the first question concerns its partitioning into equal substets and the assignment of these subsets onto the processors. Several solutions have been already studied. In [11,12] Saad exposes solutions for the implementation of 2-D ADI algorithms on ring and grid networks. Saad also discusses alternatives to the standard Gaussian elimination for the tridiagonal systems in ADI such as substructuring and cyclic reduction. Results of implementation of ring and substructuring solutions for 2-D algorithms are presented in [10] and estimated performances for the version of the 3-D Navier-Stokes solver using substructuring may be found in [4].
For the 3-D case the splitting of the cube into pencils is suited to the ring algorithm's, see Figure 1, while a domain decomposition into sub-cubes is adapted to the substructuring algorithm on a 3-D processors grid, see Figure 2. The estimation of the performances of the substructuring algorithm will be presented in an extended version of this paper. Staying with the standard Gausssian algorithm for the solution of the three steps of 2.4, we discuss here an alternative to the ring algorithm's which reduces the number of messages travelling in the network at a cost of an increase in the amount of data transfered.

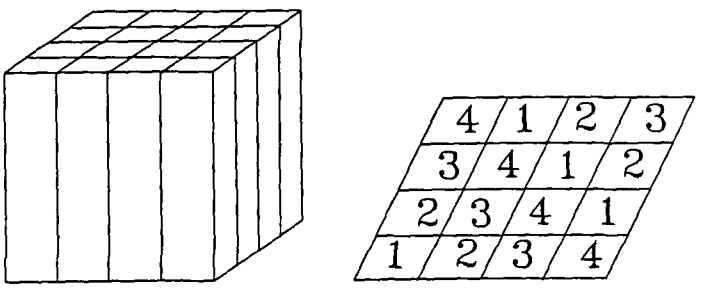


FIG. 1. *Data structure and assignment for the ring algorithm's*

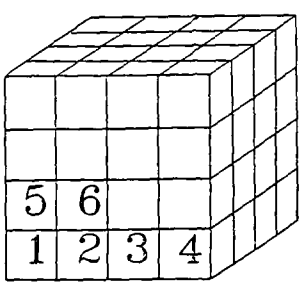


FIG. 2. *Data structure and assignment for a 3-D processors grid*

TABLE 1
Ring ADI on iPSC2-SX

one time step of the 3-D N-S code (64^3)		
number of proc.	16	32
time in sec.	102	63

3.2. Implementation of ring ADI algorithm's on the iPSC2-SX. Considering the data structure of Figure 1 the ring algorithm's has been implemented on the 32 nodes iPSC2-SX installed at ONERA. The iNTEL iPSC2-SX is a distributed memory multiprocessor which interconnects processors (i80386 microprocessor coupled to a Weitek 1167 coprocessor) in a hypercube topology network. In a P processors hypercube network, each processor with number $i = 0...P-1$ is directly connected to $log_2(P)$ neighboring processors according to the binary representation of i [9].
The table 1 presents the ellapsed times for one time step of the 3-D Navier-Stokes parallel solver which has been obtained with 16 and 32 processors of the iPSC2-SX. For a 64^3 domain the corresponding ellapsed time per mesh point and time step is 2.4×10^{-4} sec.

4. Hypercube ADI algorithm's. The 3-D ADI algorithm solves tridiagonal linear systems alternatively in each of the 3 spatial direction, say $1, 2, 3$. Noticing that these systems can be solved independently, then it is possible to transpose the data structure before each step of one ADI iteration so as to obtain local data storage in each processor adequated to computation in direction $1, 2$ or, 3. This approach induces a clear separation between the communication phase and the computation phase. This data permutation, which can be done in $log_2(P)$ steps using nearest neighbors communications, may be considered as a generalisation of the matrix transpose algorithm exposed in [13].
Let us consider an hypercube with $P = P_1 \times P_2 = 2^{q_1} \times 2^{q_2}$ processors ($q_1 \geq q_2$). We assume now, without any loss of generality, that the computational domain is a cube of dimension N, with $N = 2^l$ and $l \geq q_1$. Considering a splitting of the domain into $P_2 \times P_1 \times P_1$ blocks of size $\frac{N}{P_2} \times \frac{N}{P_1} \times \frac{N}{P_1}$, a block can be identified by the tuple $(i,j,k)_{i=0...P_2-1, j=0...P_1-1, k=0...P_1-1}$, while a processor is identified by $(l,m)_{l=0...P_2-1, m=0...P_1-1}$.
Starting from the initial mapping suitable for parallel computation of the ADI step in direction 3 which is ,
MAP-3 : $\forall k$, *store block* (i,j,k) *on processor* (i,j) *at adress* k
we detail in the following how to modify this mapping in order to obtain a

storage of the 3-D data structure adequated to ADI steps in direction 2 or 1.

4.1. Index-digit permutations for ADI algorithm. We use here a modified notation of the one used in [2,3] but the techniques are similar. Explanation of the intermediarry steps of the data movement in the hypercube needs the use of the binary representation for the processors and blocks identifiers :
$B(i_{q_2}...i_1 \mid j_{q_1}...j_1 \mid k_{q_1}...k_1)_2$ represents the block $B(i,j,k)_{10}$. Following the initial mapping, this block is stored at adress $(k_{q_1}...k_1)_2$ on the processor $(i_{q_2}...i_1, j_{q_1}...j_1)_2$. The adresses of the differents blocks of the data struture are made of two fields, one for the processor number, and the other one for the adress in local memory. The permutation suitable for ADI algorithm is an *index-digit* permutation. As demonstrates in [2] this kind of permutation can be implemented as a sequence of *i-cycles* [3] . It is interesting to look at what are the permutations or inter-processor communications that must be implemented on one specific processor $(i,j)_{10}$ in order to transpose the data structure.
Starting from the MAP-3 mapping we want to obtain the MAP-2 mapping (suitable for implicit computation into direction 2) by implementing *i-cycles*. Where, MAP-2 is :
$\forall k$, *store block* (i,k,j) *on processor* (i,j) *at adress* k
The first operations which are necessary for this operation are given in Table 2. Three different kinds of data movement are involved in this transposition. The permutation (labelled by $\xrightarrow{(G)}$) concerns a modification of the blocks storage in the local memory of the processor and gathers the blocks that will be exchanged. The second one (labelled by $\xrightarrow{(E)}$) corresponds to a communication between two neighboring processors. The third one (labelled by $\xrightarrow{(S)}$) scatters the blocks, obtained from a neighbor processor, at non consecutive adress in local memory.
Though more complicated [4], the transposition of the data structure so as to obtain a mapping for computation into direction 1 can be implemented using the same three *i-cycles*.
We must notice that each *i-cycle* operation involves $\frac{P_1}{2}$ blocks and that the initial mapping MAP-3 is obtained after any even number of ADI iterations.

[3] an *i-cycle* is an *index-digit* permutation in which the most significant digit of the adress is exchanged with any other digit, either in the adress or the processor number.
[4] from $B(i_{q_2}...i_1 \mid k_{q_1}...k_1 \mid j_{q_1}...j_1)_2$ to $B(k_{q_1}...k_{q_1-q_2+1} \mid i_{q_2}...i_1 k_{q_1-q_2}...k_1 \mid j_{q_1}...j_1)_2$

TABLE 2
From MAP-3 to MAP-2

$B(i_{q_2}...i_1 \mid j_{q_1}...j_1 \mid k_{q_1}...k_1)_2 \xrightarrow{(E)}$
$B(i_{q_2}...i_1 \mid k_{q_1} j_{q_1-1}..j_1 \mid j_{q_1} k_{q_1-1}...k_1)_2$
$B(i_{q_2}...i_1 \mid k_{q_1} j_{q_1-1}..j_1 \mid j_{q_1} k_{q_1-1}...k_1)_2 \xrightarrow{(G)}$
$B(i_{q_2}...i_1 \mid k_{q_1-1} j_{q_1-1}..j_1 \mid j_{q_1} k_{q_1}...k_1)_2$
$B(i_{q_2}...i_1 \mid k_{q_1-1} j_{q_1-1}..j_1 \mid j_{q_1} k_{q_1}...k_1)_2 \xrightarrow{(E)}$
$B(i_{q_2}...i_1 \mid k_{q_1-1} k_{q_1}..j_1 \mid j_{q_1} j_{q_1-1}...k_1)_2$
$B(i_{q_2}...i_1 \mid k_{q_1-1} k_{q_1}..j_1 \mid j_{q_1} j_{q_1-1}...k_1)_2 \xrightarrow{(S)}$
$B(i_{q_2}...i_1 \mid k_{q_1} k_{q_1-1}..j_1 \mid j_{q_1} j_{q_1-1}...k_1)_2$
...
...
$B(i_{q_2}...i_1 \mid k_{q_1}...k_1 \mid j_{q_1}...j_1)_2$

4.2. Implementation on iPSC2-SX. These ideas have been used for implementing a Poisson equation solver on the 5-cube iPSC2-SX. The Table 3 presents the iPSC2 FORTRAN version of the three *i-cycles* G, E, S. From the informations obtained when running this Poisson solver, performance of the 3-D Navier-Stokes solver can be precisely evaluated. Moreover, operations in each of the space direction of the computational domain being independant, the 32 nodes of the iPSC2 can be used to simulate a 1024 nodes system running the hypercube ADI algorithm. The Table 4 gives the evolution of the ellapsed time for computing one time step of the 3-D Navier-Stokes solver versus the number of processors (recall that the ring algorithm's takes 63 sec. on a 32 processors iPSC2-SX).

TABLE 3
FORTRAN implementation of i-cycles for ADI on the iPSC2

```
C     perform transposition
      DO 1 M=1,PAS(GWDIR)
      TYPE=M+100*GWDIR+1000*COUNT
      IDEB=(1-MYBIT(TRANSDIR,M))*SMBLK+1
C     gather blocks
      CALL PACK(NMBLK,SMBLK,PENCIL(IDEB),BUF)
C     exchange blocks
      CALL CSEND(TYPE,BUF,SMES,NEXTPASS(TRANSDIR,M),1)
      CALL CRECV(TYPE,BUF(IWORK),SMES)
C     scatter blocks
      CALL UNPACK(NMBLK,SMBLK,PENCIL(IDEB),BUF(IWORK))
      SMBLK=SMBLK*2
      NMBLK=NMBLK/2
1     CONTINUE
```

TABLE 4
Hypercube ADI on iPSC2-SX (simulation results)

one time step of the 3-D N-S code (64^3)								
number of proc.	8	16	32	64	128	256	512	1024
time in sec.	325.9	171	93.9	49	27.6	15.33	9.2	5.8

5. Influence of the communication network. The ring and hypercube algorithms differs only in the way the data strucure is organized and in the management of the inter-processors communications. A simple model of the communication network will be used to evaluate the relative performance of these two algorithms.
To send or receive a message of length N from one processor to a neighbor takes a time of :

$$\tau + N \times \frac{1}{V} \tag{5.1}$$

Where τ is the *start-up* time (in sec.) and V the communication bandwidth (in MB/sec.). Using this model the complexity of the communications involved in one time step of the two versions of the 3-D Navier-Stokes solver can be evaluated :

- *ring algorithm* : $4 \times (P-1) \times (\tau + (\frac{N^2}{P} \times \frac{1}{V}))$
- hypercube algorithm : $2 \times (q_1 + q_2) \times (\tau + (\frac{N^3}{2 \times P} \times \frac{1}{V}))$

The Figures 3, 4, 5, 6 show some comparisons of the communication costs of the two algorithms for different values of the parameters τ and V and assuming that the computational domain has a dimension $N = 64$. Let us notice that in this case the ring algorithm's cannot use more than a 6-cube. For Figure 3 we take parameters representative of an iPSC2 system (assuming the possibility to interconnect 4K processors). Figure 4 (resp. 5) show estimation relative to a NCUBE system [4] (resp. an AMETEK-2010 system [5])). In Figure 6 we consider a "modified" version of the T800 Transputer [6] with up to twelve links. [5]

6. Conclusion. We have exposed a solution for the management of data communications in a distributed memory multiprocessor which extends the use of Gaussian elimination for ADI algorithm when the number of processors is greater than the dimension of the computational domain. The performances obtained on the iPSC2-SX and the estimations done for different parameters featuring the communication network show us that the efficiency of this solution depends on the start-up time and communication time ratio. A comparison of this approach with the one based on substructuring techniques will be presented in a forthcoming paper.

Acknowledgements. This research was supported by grants from DRET.

[5] It is not our purpose here to evaluate the communication networks of these multiprocessors, and this all the more since Transputer system and AMETEK-2010 used a grid network, but rather to compare the performance of the two algorithms on a realistic basis.

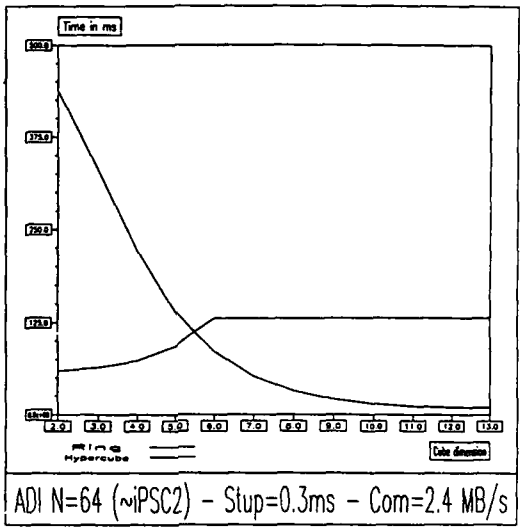


FIG. 3. *Communication costs of the two algorithms*

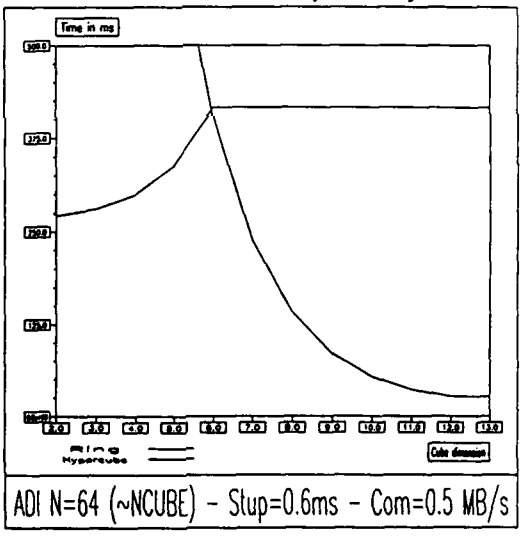


FIG. 4. *Communication costs of the two algorithms*

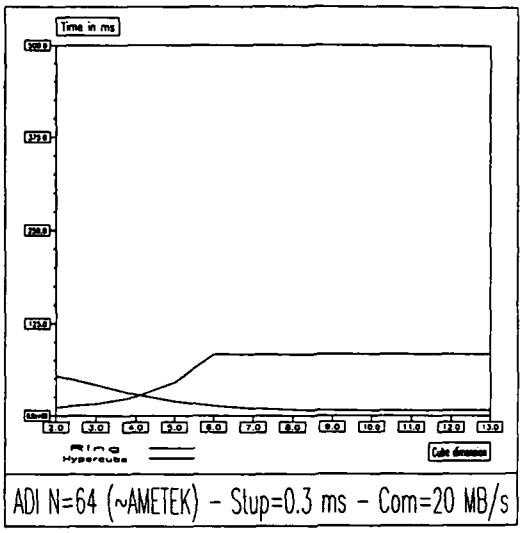


FIG. 5. *Communication costs of the two algorithms*

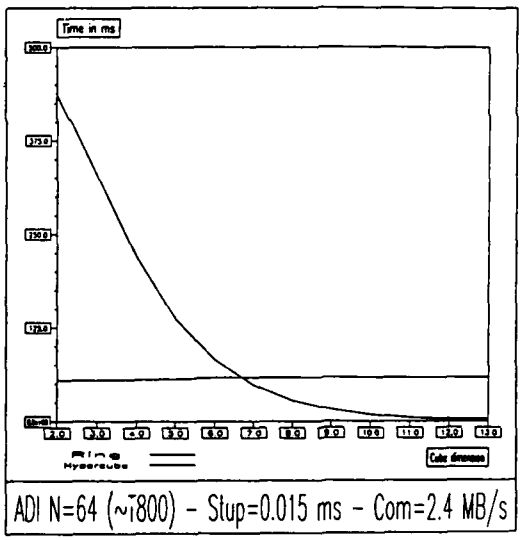


FIG. 6. *Communication costs of the two algorithms*

REFERENCES

[1] P.N. SWARZTRAUBER, *Multiprocessor FFTs*, Parallel Computing Vol. 5, No. 1 and 2, July 1987.

[2] P.N. SWARZTRAUBER AND R.A. SWEET, *Vector and parallel methods for the direct solution of Poisson's equation*, J. of Computational and Applied Math. Vol. 27, No. 1 and 2, Sept. 1989.

[3] K. AZIZ AND J. HELLUMS, *Numerical solution of the three dimensional equations of motion for laminar convection* Physics of Fluids, Vol 10 No. 2, Feb. 1967.

[4] P. LECA, L. MANE AND L. TA PHUOC, *Parallel algorithms for 2D and 3D Navier-Stokes equations on shared and distributed memory multiprocessors* 7th Int. Conf. on Finite Element Methods in Flow Problems, U. of Alabama, Apr. 3-7 1989.

[5] L. MANE AND L. TA PHUOC, *Simulation of unsteady flows at high Reynolds numbers on an experimental multiprocessor system* Recherche Aerospatiale No. 1987-2

[6] NN. YANENKO, *Méthode à pas fractionnaires,résolution des problèmes polydimensionnels de physique mathématique* Armand Colin Ed. 1968.

[7] J. DOUGLAS JR, *Alternating direction methods for three space variables* Numerische Mathematik 4 p. 41-63, 1962.

[8] W. LABIDI AND L. TA PHUOC, *Numerical resolution of 3D Navier-Stokes equations in vorticity-velocity formulation* 11th Int. Conf. on Numerical Methods in Fluids Dynamics, Williamsburg (USA) June 1988.

[9] Y. SAAD, M.H. SCHULTZ, *Data communication in Hypercubes*, Research Report YALEU/DCS/RR-DRAFT, July 1985.

[10] P. LECA, *Conception d'algorithmes parallèles pour des systèmes multiprocesseurs à mémoires distribuées : application à la programmation du système multi-FPS164 LCAP1* Int. Sem. on Scientific Supercomputers, Paris, France Feb. 1987.

[11] Y. SAAD, *On the design of parallel numerical methods in message passing and shared memory environments* Int. Sem. on Scientific Supercomputers, Paris, France Feb. 1987.

[12] E. GALLOPOULOS AND Y. SAAD, *Some Fast Elliptic Solvers on Parallel Architectures and Their Complexities* Int. Journal of High Speed Computing. Vol. 1, No. 1, May 1989.

[13] G. FOX AND ALL., *Solving problems on concurrent processors* Prentice-Hall Int. Ed. 1988.

METHODES NUMERIQUES

NUMERICAL METHODS

Rayleigh Quotient Iteration as Newton's Method

J. E. Dennis and R. A. Tapia

Rice University
Houston, Texas
U. S. A.

Abstract

The inverse, shifted inverse and Rayleigh quotient iterations are wellknown algorithms for computing an eigenpair of a symmetric matrix. In this talk we established that each one of these three algorithms can be viewed as a standard form of Newton's method from the constrained optimization literature. Our equivalence leads naturally to a new proof of the cubic convergence of Rayleigh Quotient Iteration.

Krylov subspace methods: theory, algorithms, and applications

Youcef Saad
Research Institute for Advanced Computer Science

Abstract

This paper gives an overview of projection methods based on Krylov subspaces for solving various types of scientific problems. The main idea of this class of methods when applied to a linear system $Ax = b$, is to generate in some manner an approximate solution to the original problem from the so-called Krylov subspace Span$\{b, Ab, \ldots, A^{m-1}b\}$. Thus, the original problem of size N is approximated by one of dimension m, typically much smaller than N. Krylov subspace methods have been very successful in solving linear systems (Conjugate Gradients, GMRES,..) and eigenvalue problems (Lanczos, Arnoldi,..) and are now becoming popular for solving nonlinear equations. We will show some of the main ideas in Krylov subpace methods and discuss their use in solving linear systems, eigenvalue problems, parabolic partial differential equations, Lyapunov matrix equations, and nonlinear system of equations. Some numerical experiments are presented to illustrate the concepts.

Key words: Krylov subspace methods, Conjugate Gradients, Parabolic equations, nonlinear Partial Differential Equations.
AMS(MOS) Classification: 65F.
Acknowledgements: This work was supported by the NAS Systems Division and/or DARPA via Cooperative Agreement NCC 2-387 between NASA and the University Space Research Association (USRA). Work was performed at the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035.

1 Introduction

In recent years Krylov subspace methods have become a useful and popular tool for solving large sets of linear and nonlinear equations, as well as large eigenvalue problems. One of the main reasons for their popularity is their simplicity and generality. When dealing with large systems of equations, projection methods based on Krylov subspaces are often found to be very efficient alternatives to the traditional approaches that are based on direct methods. This trend is likely to accelerate as models are becoming more complex and give rise to larger and larger matrices for which direct methods become prohibitively expensive.

Because of the success of these methods in solving large linear systems of equations, much recent work has been devoted to extending their applicability to solving other types of problems in Scientific Computing. For example, there has been substantial progress made in using these methods for solving the nonlinear equations in computational fluid dynamics [37, 22]. In addition, recent work has shown how these methods can be used to solve equations in control such as Lyapunov equations [32] and there is current interest in solving time dependent partial differential equations by the method of lines [17].

The purpose of this paper is to describe the general concepts used in Krylov subspace methods and to give an overview of the different ways in which they are used. As will be seen the method is fairly universal in that it can be used to provide approximate solutions to virtually any linear problem and nonlinear. However, it is clear that the actual success of the method will depend critically on the nature of the matrices at hand. Thus, conjugate gradient type methods are very successful for symmetric positive definite linear systems but have been rather unsuccessful with highly indefinite problems.

The next section is a brief introduction to Krylov subspaces. Section 3 discusses the application of the method to linear systems, and Section 4 is on eigenvalue problems. Section 5 will be on evaluating the product of the exponential of a matrix A times a vector with some applications. Finally, Section 6 will discuss the use of Krylov subspace methods for solving nonlinear problems.

2 Krylov subspaces

Given a square matrix A and a nonzero vector v, the subspace defined by

$$K_m \equiv span\ \{v, Av, A^2v, ...A^{m-1}v\} \tag{1}$$

is referred to as a the m-th Krylov subspace associated with the pair (A, v) and is denoted by $K_m(A, v)$ or simply by K_m if there is no ambiguity. We start by stating a few elementary properties of Krylov subspaces. Recall that the minimal polynomial of a vector v is the nonzero monic polynomial p of lowest degree such that $p(A)v = 0$. Clearly, the Krylov subspace K_m is the subspace of all vectors in $\mathbb{C}^N$ which can be written as $x = p(A)v$, where p is a polynomial of degree not exceeding $m - 1$.

Proposition 2.1 *The Krylov subspace K_m is of dimension m if and only if the degree of the minimal polynomial of v with respect to A is not less than m.*

In practice it is rather uncommon that the degree of the minimal polynomial is less than N, even in exact arithmetic. If this were to happen then it is usually helpful rather than harmful because of the following proposition.

Proposition 2.2 *Let μ be the degree of the minimal polynomial of v. Then K_μ is invariant under A and $K_m = K_\mu$ for all $m \geq \mu$.*

Thus, in case μ is small we can work work in subspace of dimension μ and be able to solve the problem exactly in this small subspace.

Working directly with the basis $\{A^j v\}_{j=0,\ldots,m-1}$ is likely to lead to serious numerical difficulties. Most Krylov subspace methods utilize either orthogonal or bi-orthogonal bases of K_m. Thus, the procedure introduced by Arnoldi [1] builds an orthogonal basis of the Krylov subspace K_m by the following algorithm.

Arnoldi's algorithm:

1. *Start:* Choose a vector v_1 of norm 1.
2. *Iterate:* for $j = 1, 2, \ldots, m$ compute,

$$h_{ij} = (Av_i, v_j) \quad i = 1, 2, \ldots, j \tag{2}$$

$$w = Av_j - \sum_{i=1}^{j} h_{ij} v_i \tag{3}$$

$$h_{j+1,j} = \|w\|_2 \tag{4}$$

$$v_{j+1} = w/h_{j+1,j} \tag{5}$$

This algorithm is mathematically equivalent to a Gram-Schmidt process applied to the power sequence $v, Av, \ldots, A^{m-1}v$, in that it would deliver the same sequence of v_i's *in exact arithmetic.* The algorithm will stop if the vector w computed in (4) vanishes which happens if the degree of the minimal polynomial for v is j. This is referred to a 'lucky' breakdown since as was seen above it means that the original problem (linear system, eigenvalue problem) can be solved exactly in a j-th dimensional subspace.

The following are a few simple but important properties satisfied by the algorithm.

Proposition 2.3 *The vectors $v_1, v_2, \ldots, v_m$ form an orthonormal basis of the subspace $K_m = span\{v_1, Av_1, \ldots, A^{m-1}v_1\}$.*

Proposition 2.4 *Denote by V_m the $N \times m$ matrix with column vectors $v_1, \ldots, v_m$ and by H_m the $m \times m$ Hessenberg matrix whose nonzero entries are defined by the algorithm. Then the following relations hold:*

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T \tag{6}$$

$$V_m^T A V_m = H_m \tag{7}$$

Note that when A is symmetric then (7) implies that the matrix H_m is tridiagonal symmetric and as a result Arnoldi's algorithm simplifies into an algorithm which involves only three consecutive vectors at each step. The corresponding algorithm is the well-known Lanczos algorithm.

The second of the relations in the proposition indicates that the Hessenberg matrix H_m is nothing but the matrix representation of the projection of A onto K_m, with respect to the orthogonal basis V_m. Analysis of various projection methods based on Krylov subspaces, indicate that, loosely speaking, K_m contains the most significant information of A, in that the outermost eigenvalues of A are well represented by those of its projection onto K_m, for large enough m. The main idea of Krylov subspace methods is to project the original problem into K_m. In the next sections we will see how this is done via simple Galerkin type procedures, for standard linear algebra problems. Then in the following sections we will address other types of problems.

The relation (6) has been exploited in [13] for solving special Sylvester's equations that arise in the design of reduced-dimensional state estimator. The Arnoldi and block-Arnoldi algorithms have been used in [5] to compute numerically the controllability of a linear system.

3 Krylov subspace methods for solving linear systems

Given an initial guess x_0 to the linear system

$$Ax = b, \tag{8}$$

a general *projection method* seeks an approximate solution x_m from an affine subspace $x_0 + K_m$ of dimension m by imposing the Petrov-Galerkin condition

$$b - Ax_m \perp L_m \tag{9}$$

where L_m is another subspace of dimension m. A Krylov subspace method is a method for which the subspace K_m is the Krylov subspace

$$K_m(A, r_0) = span\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}, \tag{10}$$

in which $r_0 = b - Ax_0$. The different versions of Krylov subspace methods arise from different choices of the subspaces K_m and L_m and from the ways in which the system is preconditioned. The most common choices of K_m and L_m are the following.

1. $L_m = K_m = K_m(A, r_0)$. The conjugate gradient method is a particular instance of this case when the matrix is symmetric positive definite. Another method in this class is the Full Orthogonalization Method (FOM) [29] which is closely related to Arnoldi's method for solving eigenvalue problems [1]. Also in this class is ORTHORES [19], a method that is mathematically equivalent to FOM. Axelsson [2] also derived a similar algorithm for general nonsymmetric matrices.

As an example we outline here the FOM method for solving linear systems. Assume that we take $v_1 = r_0/\|r_0\|_2$ and run m steps of Arnoldi's method described in the previous section.

Then, the approximate solution is of the form $x_0 + V_m y_m$ where y_m is some m-vector. The Galerkin condition (9) with $L_m = K_m$ gives immediately that $y_m = H_m^{-1}\|r_0\|_2 e_1$.

2. $L_m = AK_m; K_m = K_m(A, r_0)$. With this choice of L_m, it can be shown, see e.g., [33] that the approximate solution x_m minimizes the residual norm $\|b - Ax\|_2$ over all candidate vectors in $x_0 + K_m$. In contrast, there is no similar optimality property known for methods of the first class when A in nonsymmetric. Because of this, many methods of this type have been derived for the nonsymmetric case [3, 19, 15, 34]. The Conjugate Residual method [10] is the analogue of conjugate gradient method that is in this class. The GMRES algorithm [34] is an extension of the Conjugate Residual method to nonsymmetric problems.

3. $L_m = K_m(A^T, r_0); K_m = K_m(A, r_0)$. Clearly, in the symmetric case this class of methods reduces to the first one. In the nonsymmetric case, the biconjugate gradient method (BCG) due to Lanczos [21] and Fletcher [16] is a good representative of this class. There are various mathematically equivalent formulations of the biconjugate gradient method [30], some of which are more numerically viable than others. An efficient variation on this method, called CGS (Conjugate gradient squared) was proposed by Sonneveld [35].

Apart from the above three basic methods there are a number of techniques for nonsymmetric problems that are mathematically equivalent to solving the normal equations $A^T Ax = A^T b$ or $AA^T y = b$ by the conjugate gradient method. We will comment that these methods are often too quickly dismissed as inferior because of the fact that the condition number of the original problem is squared. For problems that are strongly indefinite they do represent, however, the only viable alternative, since none of the above three types of methods would work in this situation.

An important factor in the success of conjugate gradient-like methods is the preconditioning technique. This typically consists of replacing the original linear system (8) by, for example, the equivalent system

$$M^{-1}Ax = M^{-1}b \tag{11}$$

In the classical case of the incomplete LU preconditionings, the matrix M is of the form $M = LU$ where L is a lower triangular matrix and U is an upper triangular matrix such that L and U have the same structure as the lower and upper triangular parts of A respectively. In the general sparse case, the incomplete factorization is obtained by performing the standard LU factorization of A and dropping all fill-in elements that are generated during the process. This is referred to as ILU(0), or IC(0) in the symmetric case.

4 Krylov subspace methods for eigenvalue problems

An idea that is basic to sparse eigenvalue calculations is that of projection processes [31]. Given a subspace K spanned by a system of m orthonormal vectors $V \equiv [v_1, \ldots, v_m]$ a projection process onto $K \equiv span\ \{V\}$ computes an approximate eigenpair $\tilde{\lambda} \in \mathbb{C}, \tilde{u} \in K$ that satisfy the Galerkin condition,

$$(A - \tilde{\lambda}I)\tilde{u} \perp K \tag{12}$$

The approximate eigenvalues $\tilde{\lambda}$ are the eigenvalues of the $m \times m$ matrix $C = V^T AV$. The corresponding approximate eigenvectors are the vectors $\tilde{u}_i = Vy_i$ where y_i are the

eigenvectors of C. Similarly, the approximate Schur vectors are the vector columns of VU, where $U = [u_1, u_2, \ldots, u_m]$ are the Schur vectors of C, i.e., $U^T C U$ is quasi-upper triangular. Thus, one possible method for computing eigenvalues/ eigenvectors of large sparse matrices is to use the Arnoldi process [1, 28] which is a projection process onto $K_m = span\{v_1, Av_1, \ldots, A^{m-1}v_1\}$. Once the Arnoldi vectors $v_1, \ldots, v_m$ have been generated we can use V_m for a projection process onto K_m. The matrix $V_m^T A V_m$ which is needed for this purpose is nothing but the upper Hessenberg matrix H_m generated by the algorithm.

Note that the Arnoldi algorithm utilizes the matrix A only to compute successive matrix by vector products $w = Av$, so sparsity can be exploited. As m increases, the eigenvalues of H_m that are located in the outermost part of the spectrum start converging towards corresponding eigenvalues of A. However, the difficulty with the above algorithm is that as m increases cost and storage increase rapidly. One solution is to use the method iteratively: m is fixed and the initial vector v_1 is taken at each new iteration as a linear combination of some of the approximate eigenvectors. Moreover, there are several ways of accelerating convergence by preprocessing v_1 by a Chebyshev iteration before restarting, i.e., by taking $v_1 = t_k(A)z$ where z is again a linear combination of eigenvectors.

A technique related to Arnoldi's method is the nonsymmetric Lanczos algorithm [24, 12] which produces a nonsymmetric tridiagonal matrix instead of a Hessenberg matrix. Unlike Arnoldi's process, this method requires multiplications by both A and A^T at every step. On the other hand it has the big advantage of requiring little storage (5 vectors). Although no comparisons of the performances of the Lanczos and the Arnoldi type algorithms have been made, the Lanczos methods are usually recommended whenever the number of eigenvalues to be computed is large.

Finally, if the matrix is banded an efficient solution is the shift and invert strategy which consists of using one of the above iterative methods (subspace iteration, Arnoldi, or Lanczos) for the matrix $(A-\sigma I)^{-1}$, where σ is some shift chosen say at the center of some small region of the complex plane where eigenvalues are sought. The matrix $(A - \sigma I)^{-1}$, need not be explicitly computed: all we need is to factor $(A - \sigma I)$ into LU and subsequently at each step of the iterative method solve two triangular systems one with L and the other with U. Thus band structure can be fully exploited. In [25] several implementations of the shift and invert strategy are considered and the problem of avoiding complex arithmetic when A is real is addressed.

5 Approximation to $e^A v$ and applications

Computing approximations to the exponential of a matrix is usually not too hard a problem for small dense matrices. For large matrices, this can become a rather challenging task because of the fact that e^A is, in general a dense matrix even when A is very sparse. Fortunately, in realistic applications it is often not the exponential of the matrix that is sought but rather the product of this exponential with some vector v. The question of approximating $e^A v$ for any given vector v was considered in [17] where polynomial and rational approximations to the exponential were used. Here we summarize only the method proposed in [17] that is

based on polynomial approximation to $e^A v$. The desired approximation to $e^A v$ is expressed in the form,

$$e^A v \approx p_{m-1}(A)v \tag{13}$$

where p_{m-1} is a polynomial of degree $m-1$. Thus, the vector on the right-hand-side of (13) is an element of the Krylov subspace (1) and it is convenient to express it in the orthonormal basis $V_m = [v_1, v_2, v_3, \ldots, v_m]$ generated by Arnoldi's algorithm seen earlier. Therefore we will write the desired approximation $x_m = p_{m-1}(A)v$ as $x_m = V_m y$ where y is an m-vector. There remains to choose the unknown y. In [17], the choice $y = \beta e^{H_m} e_1$ with $\beta = \|v\|_2$ was suggested, leading to the following formula,

$$e^A v \approx \beta V_m e^{H_m} e_1 \tag{14}$$

The quality of this approximation was also analyzed in [17] and the following result was shown.

Theorem 5.1 *Let A be any square matrix and let $\rho = \|A\|_2$. Then the error of the approximation (14) is such that*

$$\|e^A v - \beta V_m e^{H_m} e_1\|_2 \leq 2\beta \frac{\rho^m e^\rho}{m!}. \tag{15}$$

Experiments reported in [17], reveal that the approximation (14) can be quite accurate even for moderate values of the degree m. The theorem shows convergence of this approximation as m increases to infinity, but the bound (15) is not sharp in general. Note also that the above approximation is exact when $m = N$, see [17].

To illustrate the concepts described in this sections we now describe two applications. The first is in solving parabolic equations and the second in handling large Lyapunov matrix equations.

5.1 Application 1: parabolic equations

One application of the above formulas is that one can approximate $e^{tA}v$ for all t as

$$e^{tA} v \approx \beta V_m e^{tH_m} e_1. \tag{16}$$

This provides a way of solving the model homogeneous ordinary differential equations $\dot{w} = -Aw$, whose solution is

$$w(t) = e^{-tA} w_0 \tag{17}$$

in which w_0 is the initial condition.

We now consider the following linear parabolic partial differential equation:

$$\frac{\partial u(x,t)}{\partial t} = Lu(x,t) + s(x), \quad x \in \Omega \tag{18}$$

$$\begin{aligned} u(0,x) &= u_0, \ \forall x \in \Omega \\ u(t,x) &= \sigma(x), \ x \in \partial\Omega, t \geq 0. \end{aligned} \tag{19}$$

where L is a second order partial differential operator of elliptic type, acting on functions defined on the open bounded set Ω. If the standard semi-discretization method (method of lines) is used to solve (18), then this partial differential equation is discretized with respect to space variables, resulting in a system of ordinary differential equations of the form

$$\begin{aligned} \frac{dw(t)}{dt} &= -Aw(t) + f \\ w(0) &= w_0 \end{aligned}$$

whose solution is explicitly given by

$$w(t) = A^{-1}f + e^{-tA}(w_0 - A^{-1}f) \tag{20}$$

which simplifies to (17) in the case of a homogeneous system ($f = 0$). Note that if we denote by $\hat{w}(t) \equiv w(t) - A^{-1}f$ and accordingly, $\hat{w}_0 \equiv w_0 - A^{-1}f$, then $\hat{w}(t)$ satisfies a homogeneous system and we have

$$\hat{w}(t) = e^{-tA}\hat{w}_0 \tag{21}$$

It is therefore possible to obtain the solution at time t in one single step, if desired, by applying the matrices A^{-1} and e^{-tA} to certain vectors.

If, instead of attempting to compute the solution at time t in one single step, we use a time-stepping procedure, then we will write at a given step t,

$$\hat{w}(t + \delta) = e^{-\delta A}\hat{w}(t). \tag{22}$$

To be able to use (22) in a numerical procedure, we need to be able to compute a vector of the form $\exp(-\delta A)v$ at every step of the procedure. The techniques outlined above can be used for this purpose. An important point to make here is that the corresponding procedure derived from (14) is essentially an explicit scheme, because it only requires evaluating matrix by vector products.

We would like now to discuss some of the aspects of the method based on the above approach. First, as was just pointed out the method is explicit in nature, since it does not require any solution of linear systems with the matrix A. The question that may be raised here concerns its stability. In fact, there is no stability difficulty in the usual sense of ODE methods because of the very nature of the scheme. Indeed, assuming that at every step the approximation to the exact solution $\hat{w}(t)$ at equidistant intevals is defined as $w_{k+1} = e^{-A\delta}w_k + \epsilon_k$, where ϵ_k is the error introduced in the approximation of the exponential term (including arithmetic rounding), we see immediately by comparing with the formula (21) for the exact solution that the error e_k at each step satisfies:

$$e_{k+1} = e^{-A\delta}e_k + \epsilon_k \tag{23}$$

In other words the scheme is unconditionally stable, in presence of a positive definite operator A.

The above stability property has been extended to a similar scheme proposed in [18] for the case where f may depend on time and A is independent of time. Note, however, that for this more realistic case one must be careful concerning the scheme used, as there are schemes whose stability do depend on the stepsize.

This may seem to be in contradiction with the usual conventional wisdom that explicit schemes require very small time steps for stability to be guaranteed. The first reason why this does not apply here is that the argument given above is limited to very specific problems, namely problems with constant coefficients. In fact the exact solution can be computed in one step provided high enough order approximation to the exponential is used!

The key point here is the possibility of using high order approximation. The importance of using high order schemes both in explicit and implicit methods has been emphasized in a few recent papers, see e.g., [36], [27]. We report here an experiment from [18] to further illustrate this point. The experiment was performed on a Cray Y-MP/832.

We consider a three-dimensional problem of the form

$$\begin{aligned} u_t &= u_{xx} + u_{yy} + u_{zz},\ x, y, z \in (0,1) \\ u &= 0 \text{ on the boundary} \end{aligned}$$

which is discretized using 17 grid points in each direction. This yields a matrix of size $N = 15^3 = 3375$. The initial conditions are chosen once the matrix is discretized, in such a way that the solution is known for all t. We take

$$u(0, x_i, y_j, z_k) = \sum_{i',j',k'=1}^{n} \frac{1}{i'+j'+k'} \sin\frac{ii'\pi}{n+1} \sin\frac{jj'\pi}{n+1} \sin\frac{kk'\pi}{n+1}$$

The above expression is simply an explicit linear combination of the eigenvectors of the discretized operator.

The goal is to integrate this partial differential equation between t=0 and t=0.1, and achieve an error-norm at $t = 0.1$ which is less than $\epsilon = 10^{-10}$. Here by error-norm we mean the 2-norm of the absolute error. Both the dimension m of the Krylov subspace and the time-step Δt can be varied. Normally, we would first choose a degree m and then try to determine the maximum Δt allowed to achieve the desirable error level. However, for convenience we have proceeded in the opposite manner: we first select a step-size Δt and then determine the minimum m that is needed to achieve the desirable error level.

What is shown in Table 1 is the various time steps chosen (column 1) and the minimum values of m (column 2) to achieve an absolute error less than $\epsilon = 10^{-10}$ at t=0.1. We show in the third column the total number of matrix-by-vector multiplications required to complete the integration. The times required to complete the integration on a Cray Y-MP are shown in the fourth column and the final 2-norm of the error achieved is shown in the 5-th column. The vector $e^{-H_m}e_1$ was computed using Padé or Chebyshev rational approximation to the exponential. The exact type of approximation used in each case is indicated in the last column of the table, with $P(k,k)$ meaning Padé of type (k,k) and $C(k,k)$ meaning Chebyshev of the type (k,k). Here the Chebyshev approximation corresponds to the best uniform approximation to e^{-x} on the positive real axis as described in [11].

Δt	m	M-vec's	Time (sec)	$\|Error\|_2$	Method
0.5000E-04	6	12006	0.8173E+01	0.1957E-11	P(2,2)
0.1000E-03	7	7007	0.4793E+01	0.3308E-10	P(2,2)
0.5000E-03	10	2010	0.1342E+01	0.1800E-10	P(4,4)
0.1000E-02	12	1200	0.7983E+00	0.2260E-10	P(4,4)
0.5000E-02	20	400	0.2672E+00	0.5271E-10	P(8,8)
0.1000E-01	26	260	0.1740E+00	0.7247E-10	P(8,8)
0.2000E-01	34	170	0.1080E+00	0.3236E-10	C(14,14)
0.3000E-01	39	156	0.9876E-01	0.6362E-10	C(14,14)
0.4000E-01	44	132	0.8030E-01	0.4122E-10	C(14,14)
0.5000E-01	49	98	0.5932E-01	0.5791E-10	C(14,14)
0.1000E+00	71	71	0.4186E-01	0.9993E-10	C(14,14)

Table 1: Performance of the polynomial scheme with varying accuracy on the Cray YMP.

Since the matrix is symmetric, we have used a Lanczos algorithm to generate the $v_i's$ instead of the full Arnoldi algorithm. No reorthogonalization of any sort was performed. The matrix consists of 7 diagonals, so the matrix by vector products are performed by diagonals resulting in a very effective use of the vector capabilities of the YMP. It was estimated that the average Mflops rate reached (excluding the calculation of $\exp\{-H_m\}e_1$) was around 220. This is achieved with virtually no code optimization.

Observe the tremendous gains in computational time and in the number of calls to the matrix by vector multiplication routine, as the order increases. The gain in time is nearly 200 between the lowest degree used (6) and the highest degree used (71).

One the main attractions of a scheme based on this approach is the high degree of parallelism that it offers. There are opportunities to exploit parallelism in virtually every part of the algorithm. However, it is often argued that the the loss of efficiency, incurred by mandatory smaller step-sizes, exhibited by explicit schemes as compared with implicit schemes outweighs the benefits of the high degree of parallelism permitted by the explicit scheme. For simple problems such as the one tested above, the argument is certainly not true because of the possibility of using high order schemes as described in this section. It remains to be seen whether the argument is still valid for the more complex case where A and/or f depend on time.

The usability of this approach has recently been extended to the case of a non-constant forcing term f with very encouraging results. Extensions to the more general case where A is also time dependent, or a nonlinear function in w, are currently under investigation. Note that this would provide alternative ways of solving time dependent partial differential equations by the method of lines.

5.2 Application 2: the Lyapunov matrix equation

Another direct application described in [32], is for solving the matrix Lyapunov equation,

$$AX + XA^T + bb^T = 0, \tag{24}$$

which is a common problem in the study of dynamical systems with single input,

$$\dot{u} = Au + bg. \tag{25}$$

The exact expression for the equation (24) in the case where the corresponding system (25) is controllable is given by the expression

$$X = \int_0^\infty e^{\tau A} bb^T e^{\tau A^T} d\tau. \tag{26}$$

known as the controllability Grammian of the dynamical system. In [32] the solution X as provided by (26) was approximated by replacing the function $e^{\tau A}b$ by its approximation (16). Interestingly, the result of substituting the approximation (16) in (26) leads to an approximate solution of the form $X_m = V_m G_m V_m^T$, where V_m is the matrix of the Arnoldi vectors, and G_m is an $m \times m$ matrix which, incidentally, is the solution of a Lyapunov matrix equation involving $m \times m$ matrices. In fact, a rather unexpected result shown in [32] is that this approximation provided by the above integration process is mathematically equivalent to a Galerkin method applied to (24) over the subspace of matrices of the form $V_m G V_m^T$, where V_m is fixed and G runs over the set of $m \times m$ matrices. The inner product used for this Galerkin process is defined by $< X, Y >= tr(Y^T X)$.

Large Lyapunov equations that cannot be handled otherwise have been solved in this manner. As an illustration, we now consider a test example derived from the discretization of a partial differential equation of the form:

$$\frac{\partial u}{\partial t} = \Delta u + F(x,y)g(t) \tag{27}$$

in a rectangular domain, with Dirichlet boundary conditions. Here $\Delta = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ is the Laplacean operator. If we discretize the rectangle using $n_x + 2$ points in the x direction and $n_y + 2$ points in the y direction, the above equations lead to a matrix problem of the form:

$$\dot{u} = Au + bg \tag{28}$$

where A is square of dimension $N = n_x n_y$. In this experiment we took $n_x = 20$ and $n_y = 40$ leading to a matrix of size 800. We have taken b to be simply e_1 the first column of the identity. Tests with other choices for b showed similar results. First we would like to show the behavior of the residual achieved by the Krylov subspace method outlined above, as the degree m varies. Table 2 shows the scaled Frobenius norm of the residual, i.e., the quantity $\|AX_m + X_m A^T + bb^T\|_F$, where $\|Z\|_F = (tr[Z^T Z]/N)^{1/2}$. This is done for $m = 5, 10, 15, 20$.

We have used the Arnoldi process instead of the Lanczos algorithm on purpose, despite the fact that the matrix is symmetric, in order to give an idea of the behavior in the more

m	$\|Res\|_F$	Time (sec)
5	1.10 E-04	0.18
10	5.40 E-06	0.23
15	7.92 E-07	0.35
20	1.92 E-07	0.45

Table 2: Performance of the Krylov subspace method for the Lyapunov matrix equation.

general situation. The table indicates that the accuracy of the Krylov subspace approximation to the Lyapunov equation is good for very small m and then improves slowly. The times reported in this table are in seconds on the Ardent Titan with two processors and have been obtained using the -O3 compiling option. For details and comparisons with other techniques the reader is referred to [32].

6 Nonlinear Krylov subspace methods

This section gives an overview of some basic techniques based on Krylov subspaces for solving systems of nonlinear equations. We start by discussing the various ways in which general nonlinear projection methods can be defined.

6.1 Nonlinear projection methods

There are several ways of generalizing the standard Galerkin or Petrov Galerkin methods to nonlinear equations. For example, Marion and Temam [23] define nonlinear Galerkin methods by projecting the original equations onto nonlinear manifolds instead of linear subspaces. Our approach is more conventional in that we still use linear subspaces but may impose nonlinear Galerking conditions.

Consider the nonlinear system

$$F(u) = 0, \tag{29}$$

where F is a nonlinear function from $\mathbb{R}^N$ to $\mathbb{R}^N$. At each iteration of a general nonlinear projection method we select a (linear) subspace K and we seek an approximate solution to (29) of the form $u + \delta$ where δ belongs to the subspace K and u is the current iterate. Note that the subspace K changes at every step of the nonlinear iteration. The standard case examined in [8] is when K is a Krylov subspace associated with the Jacobian of F at the current iterate. The various nonlinear projection methods we consider differ in the way the vector δ is chosen in the subspace.

A natural choice for the next iterate is to select a vector δ in K such that

$$f(u + \delta) \equiv \frac{1}{2}\|F(u + \delta)\|_2^2 \tag{30}$$

is minimized. Although this is a nonlinear least squares problem, from a practical point of view it is much easier to solve than the original problem when the dimension m of K is much smaller than N. The motivation for this approach is that one can exploit a number of highly efficient packages, such as MINPACK, or NL2SOL, for solving least squares problems of small dimension, such as (30).

Let $V = [v_1, v_2, ..., v_m]$ be an $N \times m$ matrix whose column vectors represent an orthonormal basis of the subspace K and write δ as

$$\delta = Vy, \tag{31}$$

where y is an m-vector. The function (30) to be minimized becomes a function of y defined by

$$g(y) = \frac{1}{2}\|F(u + Vy)\|_2^2 \tag{32}$$

The gradient of this function at y is given by

$$\nabla g(y) = V^T J(u + Vy)^T F(u + Vy) \tag{33}$$

where $J(x)$ is the Jacobian of F at the point $x \in \mathbb{R}^N$. Notice that the gradient of f is $\nabla f(u) = J(u)^T F(u)$ and so we have the simple relation $\nabla g(y) = V^T \nabla f(u + Vy)$.

A necessary, but not always sufficient, condition for y^* to be a minimum of (32) is that the gradient of g at y^* vanishes, i.e., we must have

$$V^T J(u + Vy^*)^T F(u + Vy^*) = 0 \tag{34}$$

This suggests simply solving the equations,

$$(J(u + Vy)V)^T F(u + Vy) = 0 \tag{35}$$

as a means for finding a minimizer of (32), although we know that the set of solutions of (35) is larger than the set of minimizers of (32). We refer to the above system of nonlinear equations as the set of *normal equations* for minimizing (32).

When solving the above normal equations the Jacobian must be reevaluated at each new iterate and this may be uneconomical. An alternative is to freeze $J(u + Vy)V$ to be the system of vectors computed at, say, $y = 0$ and solve the set of modified equations:

$$(J(u)V)^T F(u + Vy) = 0 \tag{36}$$

This is a particular case of the *Petrov-Galerkin condition*

$$W^T F(u + Vy) = 0, \tag{37}$$

where W is an $N \times m$ matrix. Two particular cases are noteworthy:

1. $W = V$ which corresponds to the Galerkin case.
2. $W = JV$ which was naturally derived above;

When F is linear the first case corresponds to the conjugate gradient method if the coefficient matrix is symmetric and Arnoldi's method otherwise. The second case corresponds to the class of methods based on minimizing the residual norm, a few representatives of which are ORTHOMIN, GCR, GMRES, see [34] for details.

Finally, one may linearize $F(u + Vy)$ in (37) around u and derive fully linearized techniques which correspond to solving the linear system,

$$W^T[F(u) + J(u)Vy] = 0 \tag{38}$$

where $J(u)$ is the Jacobian of F at the current iterate u. The above linear system is m-dimensional and will admit a unique solution if the section $W^T J(u)V$, which is an $m \times m$ matrix, is nonsingular. In the particular case where $W = JV$ this means that the columns of JV must be linearly independent. Note that (38) represents one way of approximately solving the Newton system

$$F(u) + J(u)\delta = 0 \tag{39}$$

at every step of Newton's method. Thus, the fully linearized techniques are a particular case of a class of methods that are commonly referred to as *inexact Newton methods* and have been studied in the literature, (see, e.g., [14, 6, 26, 8, 7]).

6.2 Globally convergent nonlinear Krylov methods

In this section we only consider the fully linearized methods in the sense defined above. To guarantee global convergence, the usual inexact Newton methods must be modified in several ways. A few such modifications have been proposed in [8]. Moreover, in [7] a number of convergence results for these techniques have been established. We would like to summarize some of these results here.

The simplest modification involves a backtracking procedure. In this technique, an iterate u_n is given and we define the next iterate in the form $u_n + \lambda p_n$, where p_n is any descent direction and λ is selected by a procedure which ensures that the function f decreases sufficiently at each iteration and that the iterate makes sufficient progress towards the solution. One such procedure based on linesearch backtracking is described below. The search direction p_n is provided by an approximate solution to the Newton system $J(u_n)p = -F(u_n)$, e.g., via FOM or GMRES. It is easy to show that p_n is a descent direction at u_n whenever we have

$$\|F(u_n) + J(u_n)p_n\|_2 < \|F(u_n)\|_2,$$

which means that the residual norm for the Newton system $J(u_n)p = -F(u_n)$ must be strictly reduced from that associated with $p = 0$. In particular, it is common to require that a condition of the form

$$\|F(u_n) + J(u_n)p_n\|_2 \leq \eta_n \|F(u_n)\|_2,$$

where $\eta_n \leq \eta < 1$, in the context of iterative methods.

In the procedure described below the two parameters $\theta_{\text{r in}}, \theta_{\max}$ are such that $0 < \theta_{\min} \leq \theta_{\max} < 1$, the simplest choice being $\theta_{\min} = \theta_{\max} = 1/2$. The procedure requires another

parameter $\epsilon^* > 0$ which is used to essentially rescale the *starting step* in the process in order to prevent it from from being too small.

Algorithm 3.1: General Backtracking Procedure

1. Set $\lambda = \max\{1, \epsilon^* \frac{|\nabla f(u_n)^T p_n|}{\|p_n\|_2^2}\}$.
2. If $f(u_n + \lambda p_n) \leq f(u_n) + \alpha\lambda\nabla f(u_n)^T p_n$, then set $\lambda_n = \lambda$, and exit. Else:
3. Choose $\hat{\lambda} \in [\theta_{\min}\lambda, \theta_{\max}\lambda]$; set $\lambda \leftarrow \hat{\lambda}$. Go to (2).

The following theorem [7] is a general convergence result for sequences generated the above algorithm.

Theorem 6.1 *Let $f \equiv \frac{1}{2}\|F\|_2^2$ be differentiable and assume that its gradient is such that*

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq \gamma\|x - y\|_2, \text{ for all } x, y \in \mathbb{R}^N. \tag{40}$$

Let p_n be such that $\|F_n + J_n p_n\|_2 \leq \eta\|F_n\|_2$ for all n, with $\eta < 1$. Further, let each iterate be chosen by the General Backtracking Algorithm. Then, either

$$\lim_{n\to\infty} f(u_n) = 0 \tag{41}$$

or

$$\lim_{n\to\infty} \|p_n\|_2 = \infty. \tag{42}$$

Moreover, superlinear convergence will essentially take place at the additional condition that $\eta_n \to 0$. Global convergence of the method using the trust region model approach has also been analyzed in [7].

One of the most successful ways of using nonlinear Krylov subspace methods is for solving nonlinear equations in which the Jacobian of F is not available or is too expensive to compute. Note that the cost of producing the Jacobian may well take into account the initial programming effort. The reason why we can still use the methods outlined above is that Krylov subspace methods do not require the Jacobian matrix J explicitly, but only its action on a vector v. This action can be well approximated by a difference quotient of the form

$$J(u)v \approx \frac{F(u + \sigma v) - F(u)}{\sigma},$$

where u is an approximation to a solution of (29), and σ is some small scalar. The above observation has been exploited in several papers [37, 22, 20, 9] to accelerate fixed-point iterations of the form

$$u_{n+1} = M(u_n)$$

by applying the above techniques to the system $F(u) \equiv u - M(u) = 0$. Typically, the Jacobian of the mapping F is a dense matrix and it may be impractical to compute it for large problems.

6.3 Application : equations of semi-conductor device simulation

There are several ways of writing the equations of semi-conductor device simulation. One form of the equations uses the "quasi-Fermi levels" v and w that are related to the electron density n and hole density p by $n = e^{u-v}$ and $p = e^{w-u}$. The dimensionless form of steady state equations are as follows,

$$\nabla\cdot(e^{u-v}\nabla v) = 0, \tag{43}$$

$$\nabla\cdot(e^{-u+w}\nabla w) = 0, \tag{44}$$

$$-\nabla^2 u + e^{u-v} - e^{w-u} - k_1 = 0, \tag{45}$$

subject to appropriate (possibly mixed) boundary conditions. The first two equations represent the continuity equations for electrons and holes while the third is Poisson's equation for the (normalized) potential u. The term k_1 is the doping profile (in units of the intrinsic density of the semiconductor), which is essentially a source term. For more details about the above system of equations and the assumptions made see, e.g., [4, 20].

The above system represents a coupled nonlinear system of three partial differential equations. The so-called decoupling algorithm used in this context to solve this system consists of a Block Gauss-Seidel iteration on the discrete version of the above equations. It can be briefly described as follows. Given the dimensionless potential u from the previous iteration, one obtains the intermediate variables v and then w by solving (43) and then (44). Then the potential equation (45) is solved to obtain a new potential $\tilde{u}$. The whole mapping from the old potential u to the new potential $\tilde{u}$ will be denoted T:

$$\tilde{u} = T(u). \tag{46}$$

Whereas the above algorithm is very robust in practice, it is found that there are situations where it becomes very slow. The method has been in general abandoned in favor of the Full Newton schemes to solve the coupled system (43)-(44)-(45), see reference [4]. The Newton equations are typically solved by direct methods. As simulators are now starting to cover three-dimensional models, there is a regain of interest in iterative methods for solving the Newton equations.

Another approach proposed in [20] is to apply an acceleration procedure to the fixed point iteration $u_{k+1} = T(u_k)$. In other words we would like to solve the system

$$u - T(u) = 0.$$

Note that the Jacobian of T is generally a dense matrix here and would be rather difficult to compute. However, as was stressed before there is no difficulty using a procedure such as GMRES to compute the zero of $u - T(u)$ even when the Jacobian is not explicitly available. This was implemented and tested in [20]. A comparison with the non-accelerated version of the decoupling algorithm and the accelerated version showed considerable gains in speed, a factor of about 8 in the example reported. Moreover, GMRES acceleration was also vastly superior to two variants of Chebyshev acceleration schemes. This substantial superiority can be attributed to the capacity of GMRES to take advantage of the clustering of the spectrum

of the Jacobian of T around the origin. This property of $\sigma[T_u]$ can be related to the compact differentiability of the continuous mapping T. Like other conjugate gradient type methods, GMRES is able to take advantage of a spectrum in which the rate of convergence is slowed down by a few isolated eigenvalues only. Chebyshev type schemes do not take advantage of any clustering of the spectrum, but only the overall size and shape of its convex hull.

7 Conclusion

We have shown several ways in which Krylov subspaces can be used to solve various types of scientific problems. The scope of application areas where these methods can be used has been steadily widening in recent years. One might say that the method constitutes in effect a universal way of reducing the dimensionality of the original problem. Perhaps one of the most challenging areas where the method can be used is in solving inverse problems. Some of the ideas presented in Section 6 may possibly be exploited for this purpose but there is still much to be done in this direction.

Acknowledgements

The author would like to acknowledge that part of the work presented herein is work published or in progress with coauthors. Specifically, the work in Section 5 is joint work with E. Gallopoulos [17, 18]. The work in Section 6 is joint work with Peter Brown, see references [7, 8].

References

[1] W. E. Arnoldi. The principle of minimized iteration in the solution of the matrix eigenvalue problem. *Quart. Appl. Math.*, 9:17–29, 1951.

[2] O. Axelsson. Conjugate gradient type-methods for unsymmetric and inconsistent systems of linear equations. *Linear Algebra Appl.*, 29:1–16, 1980.

[3] O. Axelsson. A generalized conjugate gradient, least squares method. *Num. Math.*, 51:209–227, 1987.

[4] R. E. Bank, W. M. Coughram Jr., W. Fichtner, E. H. Grosse, D. J. Rose, and R. K. Smith. Transient simulation of silicon devices and circuits. *IEEE Trans. Computer Aided Design*, CAD-4:436–451, 1985.

[5] D. Boley and G. H. Golub. The Lanczos-Arnoldi algorithm and controllability. *Systems and Control Letters*, 4:317–324, 1987.

[6] P. N. Brown. A local convergence theory for combined inexact-Newton/ finite difference projection methods. *SIAM J. Num. Anal.*, 24:407–434, 1987.

[7] P. N. Brown and Y. Saad. Globally convergent techniques in nonlinear Newton-Krylov algorithms. Technical Report 89–, Research Institute for Advanced Computer Science, 1989.

[8] P. N. Brown and Y. Saad. Hybrid Krylov methods for nonlinear systems of equations. *SIAM J. Sci. Stat. Comp.*, 1990. To appear.

[9] T. F. Chan and K. R. Jackson. Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms. *SIAM J. Stat. Scien. Comput.*, 7:533–542, 1984.

[10] R. Chandra. *Conjugate Gradient Methods for Partial Differential Equations.* PhD thesis, Yale University, Computer Science Dept., New Haven, CT. 06520, 1978.

[11] W. J. Cody, G. Meinardus, and R. S. Varga. Chebyshev rational approximations to e^{-x} in $(0, +\infty)$ and applications to heat-conduction problems. *J. Approx. Theory*, 2:50–65, 1969.

[12] J. Cullum and R. Willoughby. A Lanczos procedure for the modal analysis of very large nonsymmetric matrices. In *Proceedings of the 23rd Conference on Decision and Control, Las Vegas*, 1984.

[13] B. N. Datta and Y. Saad. Arnoldi methods for large Sylvester-like observer matrix equations. in preparation.

[14] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 18(2):400–408, 1982.

[15] H. C. Elman. *Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations.* PhD thesis, Yale University, Computer Science Dept., New Haven, CT., 1982.

[16] R. Fletcher. Conjugate gradient methods for indefinite systems. In G.A. Watson, editor, *Proceedings of the Dundee Biennal Conference on Numerical Analysis 1974*, pages 73–89, New York, 1975. University of Dundee,Scotland, Springer Verlag.

[17] E. Gallopoulos and Y. Saad. On the parallel solution of parabolic equations. In R. De Groot, editor, *Proceedings of the International Conference on Supercomputing 1989, Heraklion, Crete, June 5-9, 1989.* ACM press, 1989.

[18] E. Gallopoulos and Y. Saad. Parallel solution of parabolic equations by the polynomial approximation approach. Technical report, RIACS, NASA Ames research center, Moffett Field, CA, 1990. In preparation.

[19] K. C. Jea and D. M. Young. Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods. *Linear Algebra Appl.*, 34:159–194, 1980.

[20] T. Kerkhoven and Y. Saad. Acceleration techniques for decoupling algorithms in semiconductor simulation. Technical Report 684, University of Illinois, CSRD, Urbana, IL., 1987.

[21] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bur. Standards*, 49:33–53, 1952.

[22] M. Mallet, J. Periaux, and B. Stoufflet. Convergence acceleration of finite element methods for the solution the euler and navier stokes equations of compressible flow. In *Proceedings of the 7-th GAMM Conference on Numerical methods in Fluid Dynamics.* INRIA, North-Holland, 1987.

[23] M. Marion and R. Temam. Nonlinear Galerkin methods. *SIAM J. Numer. Anal.*, 26:1139–1157, 1989.

[24] B. N. Parlett and D. R. Taylor andf Z. S. Liu. A look-ahead Lanczos algorithm for nonsymmetric matrices. *Mathematics of Computation*, 44:105–124, 1985.

[25] B.N. Parlett and Y. Saad. Complex shift and invert strategies for real matrices. Technical Report YALEU/ DCS-RR-424, Yale University, Computer Science Dept., New-Haven, Connecticut, 1985.

[26] A.C. Hindmarsh P.N. Brown. Matrix-free methods for stiff systems of odes. *SIAM J. Num. Anal.*, 23:610–638, 1986.

[27] M. F. Reusch, L. Ratzan, N. Pomphrey, and W. Park. Diagonal Padé approximations for initial value problems. *SIAM J. Sci. Statist. Comput.*, 9:829–838, 1988.

[28] Y. Saad. Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices. *Linear Algebra Appl.*, 34:269–295, 1980.

[29] Y. Saad. Krylov subspace methods for solving large unsymmetric linear systems. *Mathematics of Computation*, 37:105–126, 1981.

[30] Y. Saad. The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems. *SIAM J. Numer. Anal.*, 19:470–484, 1982.

[31] Y. Saad. Projection methods for solving large sparse eigenvalue problems. In B. Kagstrom and A. Ruhe, editors, *Matrix Pencils, proceedings, Pitea Havsbad*, pages 121–144, Berlin, 1982. University of Umea, Sweden, Springer Verlag. Lecture notes in Math. Series, Number= 973.

[32] Y. Saad. Numerical solution of large Lyapunov equations. Technical Report 89-20, RIACS, Ms 230-5, NASA Ames, Moffett Field, CA 94035, 1989. Also in these proceedings.

[33] Y. Saad and M. H. Schultz. Conjugate gradient-like algorithms for solving nonsymmetric linear systems. *Mathematics of Computation*, 44(170):417–424, 1985.

[34] 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.

[35] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Scient. Statist. Comput.*, 10(1):36–52, 1989.

[36] H. Tal-Ezer. Spectral methods in time for parabolic problems. *SIAM J. Numer. Anal.*, 26:1–11, 1989.

[37] L.B. Wigton, D.P. Yu, and N.J. Young. GMRES acceleration of computational fluid dynamics codes. In *Proceedings of the 1985 AIAA conference, Denver 1985*, Denver, 1985. AIAA.

AN INTRODUCTION TO THE STRUCTURE OF LARGE SCALE NONLINEAR OPTIMIZATION PROBLEMS AND THE LANCELOT PROJECT

by A.R. Conn[1], N.I.M. Gould[2] and Ph.L. Toint[3]

December 8, 1989

Abstract. This paper presents the authors' personnal views on two fundamental aspects amongst the recent developments in the growing field of large scale nonlinear mathematical programming. Important concepts for the description of problem structure are discussed in detail. A systematic approach to software development for this class of problems is also presented. The approach incorporates both suitable numerical algorithms and user oriented standard format for problem specification in a modular and coherent system.

[1] **Department of Combinatorics and Optimization, University of Waterloo, Ontario, Canada**

[2] **Computer Science and Systems Division, Harwell Laboratory, Oxfordshire, England**

[3] **Department of Mathematics, Facultés Universitaires ND de la Paix, Namur, Belgium**

Invited talk at the *Ninth international conference on computing methods in applied sciences and engineering*, Paris, January 1990.

Keywords : Large scale problems, nonlinear programming, problem structure, software development.

1 Introduction

The field of numerical optimization, one of the most challenging areas of numerical analysis, has recorded an impressive growth in the past few years. Apart from widely known developments in linear programming (see [23], for instance), this growth has also been fueled by a steadily increasing interest in nonlinear problems involving a large number of variables. However, the importance of the new results and the power of the new algorithms developed are probably not fully appreciated by the larger community of numerical analysts and, even more importantly, by the community of potential users of large scale nonlinear optimization methods. Recent publications devoted specifically to large scale problems include [18], [9], [10], and [2].

It is a widely held view that only rather small problems can be handled by the techniques available. Specifying nonlinear optimization problems in more than 20 variables, say, is still considered by many as a risky modelling approach, mostly because the problem's solution is likely to be impossible with the existing algorithms. It is true that this view was justified ten years ago. The present situation is however quite different and it is the purpose of this paper to stress this change and to present some of the concepts that resulted in this progress.

These concepts will be presented here from the authors' very personnal (and maybe biased) point of view. In particular, no attempt is made to discuss every concept and significant recent development in the area, rather the exposition will focus on two topics that are considered to be fundamental by the authors. We will also outline our present and forthcoming research in this area, both from the algorithmic and software perspective.

The first part of this paper is devoted to an introduction to the classes of partially separable and group partially separable functions. Understanding and exploiting these concepts is, in our opinion, central to the development of efficient and reliable methods for large scale problems, much in the same way that sparsity is a key to large scale numerical linear algebra. This introduction is contained in Section 2.

The second part of the paper discusses the LANCELOT software project, whose purpose is to produce a self contained system for large scale nonlinear optimization. The discussion emphasizes the main objectives of the system, with a brief discussion of some data input and implementation issues.

2 The structure of nonlinear problems

2.1 A tutorial on partial separability

Very few numerical analysts would contest today the crucial role played by the various techniques for exploiting the structure of a problem in the development of practical computational methods for large scale problems. Sparsity in large systems of linear equations, domain decomposition in the numerical solution of partial differential equations and structure in the interpolation equations for function approximation are probably the three examples that come first to mind. The situation is entirely similar in large scale nonlinear optimization, where exploiting the structure of large problems is the only reasonable way to tackle their solution.

Introduced in 1981, by Andreas Griewank and the third author in [15], the notion of a partially separable function can be viewed as a way to describe the structure of a nonlinear function in terms of an underlying geometry (i.e. subspaces and their relations).

It is the authors' belief that this notion can be very helpful, not only to algorithm designers, but also to potential users of these algorithms. Indeed, for a method to exploit structure, it is very beneficial that the user describes this structure in a way coherent with the implementation. This in turn supposes that the user should be at least moderately familiar with the principles of the description used.

The concept of partial separability will be introduced and analysed by a simple yet meaningful example: the discretised minimum surface problem over the unit square. We will not be interested in this problem as such, but rather in showing that one of its discretised formulations exhibits the type of structure that we want to exploit.

The minimum surface variational problem is well known, and consists in finding the surface of minimum area that interpolates a given continuous function on the boundary of the unit square. The unit square itself is discretised uniformly into m^2 smaller squares, as shown in Figure 1.

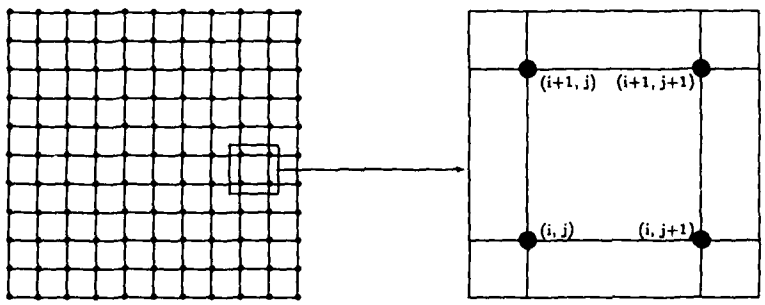


Figure 1: Discretisation of the unit square

The variables of the problem will be chosen as the "height" of the unknown surface above the $(m+1)^2$ vertices of the m^2 smaller squares. The area of the surface above the (i,j)-th discretised square is then approximated by the formula

$$s_{i,j}(x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1}) = \frac{1}{m^2}\sqrt{1 + \frac{m^2}{2}[(x_{i,j} - x_{i+1,j+1})^2 + (x_{i,j+1} - x_{i+1,j})^2]}, \qquad (1)$$

where the squares and variables are indexed as shown in Figure 1. The precise justification of this formula does not matter here, but we concentrate instead on its form.

The variables in the sets

$$\{x_{1,j}\}_{j=1}^{m+1}, \quad \{x_{i,m+1}\}_{i=1}^{m+1}, \quad \{x_{m+1,j}\}_{j=1}^{m+1}, \text{ and } \{x_{i,1}\}_{i=1}^{m+1}, \qquad (2)$$

correspond to the boundary conditions and are assigned given values, while the others are left free. The complete problem is then to minimise the objective function

$$f(x) = \sum_{i,j=1}^{m} s_{i,j}(x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1}) \tag{3}$$

over all the free variables[1].

A first analysis quickly shows that, using a row-wise ordering of the variables, the Hessian $\nabla^2 f(x)$ has a block-tridiagonal sparsity pattern with tridiagonal blocks. We note that storing this matrix, which is typically required in one form or the other in most efficient algorithms, therefore requires storing $O[5(m+1)^2]$ real numbers. More importantly, this sparsity structure may be discovered by considering the Hessian of $s_{i,j}$ as a function of $x_{i,j}$, $x_{i+1,j}$, $x_{i,j+1}$ and $x_{i+1,j+1}$, and then by assigning each of the rows and columns of this dense 4×4 matrix to the relevant row and column of the larger $\nabla^2 f(x)$. If we consider each $s_{i,j}$ as a function of the complete set of $(m+1)^2$ variables, denoted by $s_{i,j}(x)$, we see that it satisfies the important property that

$$s_{i,j}(x) = s_{i,j}(x + w) \tag{4}$$

for all vectors w in the *invariant subspace*

$$N^s_{i,j} = \{x \in \mathbb{R}^{(m+1)^2} | w_{i,j} = w_{i,j+1} = w_{i+1,j} = w_{i+1,j+1} = 0\}. \tag{5}$$

Each $s_{i,j}$ is therefore invariant with respect to all translations corresponding to vectors of $N^s_{i,j}$. From this invariance, it is again easy to deduce that $\nabla^2 s_{i,j}(x)$ is a (very) sparse matrix, but we stress the point that (4)–(5) is in fact the most important observation when analysing the structure of the second derivatives of our model problem. *We may then interpret the sparsity pattern of this last matrix as a* consequence *of the problem structure: sparsity is easily derived from the structure, but the reverse is not true.*

If we now examine the function $s_{i,j}$ in more detail, we easily see that, instead of being a function of the four variables $x_{i,j}$, $x_{i+1,j}$, $x_{i,j+1}$ and $x_{i+1,j+1}$, it is, in fact, a function of the two *internal variables*

$$u_{i,j} \stackrel{\text{def}}{=} x_{i,j} - x_{i+1,j+1} \quad \text{and} \quad v_{i,j} = x_{i,j+1} - x_{i+1,j}. \tag{6}$$

If we write the simple linear transformation

$$\begin{pmatrix} u_{i,j} \\ v_{i,j} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_{i,j} \\ x_{i,j+1} \\ x_{i+1,j} \\ x_{i+1,j+1} \end{pmatrix}, \tag{7}$$

we can then reformulate $s_{i,j}$ in terms of these internal variables as

$$s_{i,j}(x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1}) \stackrel{\text{def}}{=} \hat{s}_{i,j}(u_{i,j}, v_{i,j}), \tag{8}$$

where the new function $\hat{s}_{i,j}(u_{i,j}, v_{i,j})$ is easily derived from (1) and is given by

$$\hat{s}_{i,j}(u_{i,j}, v_{i,j}) = \frac{1}{m^2}\sqrt{1 + \frac{m^2}{2}(u^2_{i,j} + v^2_{i,j})}. \tag{9}$$

[1] Various obstacle problems can also be obtained by specifying suitable bounds on the variables.

Computing now the gradient and Hessian of $\hat{s}_{i,j}$ with respect to its two arguments, we obtain that

$$W^T \nabla \hat{s}_{i,j}(u_{i,j}, v_{i,j}) = \nabla s_{i,j}(x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1}), \tag{10}$$

where the matrix

$$W = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix}, \tag{11}$$

and also that

$$W^T \nabla^2 \hat{s}_{i,j}(u_{i,j}, v_{i,j}) W = \nabla^2 s_{i,j}(x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1}), \tag{12}$$

where $s_{i,j}$ is again considered as a function of four variables. But the Hessian of $\hat{s}_{i,j}$ is now a 2×2 symmetric matrix, while that of $s_{i,j}$ is 4×4! Furthermore, we did not index the matrix W, because the same linear transformation holds for all values of i and j. Storing the second derivatives of our problem therefore requires storing W once, plus storing the m^2 Hessians of the functions $\hat{s}_{i,j}$, which amounts to $O[3(m+1)^2]$ real numbers. This is a substantial reduction compared to the $O[5(m+1)^2]$ required by the more classical "sparsity oriented" approach.

In order to abstract from these observations, we simply note that (4) holds not only for the vectors w in $N^s_{i,j}$, but for all w in

$$N_{i,j} = N^s_{i,j} + \{w \in \mathbf{R}^{(m+1)^2} | w_{i,j} = w_{i+1,j+1} \text{ and } w_{i,j+1} = w_{i+1,j}\}. \tag{13}$$

We can then use all the above remarks to define *partially separable functions* as follows.

We say that $f(x)$ is partially separable if and only if

1. it can be written as a sum of *element functions*, that is

$$f(x) = \sum_{i=1}^{p} f_i(x), \tag{14}$$

2. each of these element functions has a nontrivial *invariant subspace*, that is if, for each $i \in \{1, \ldots p\}$, there exists a subspace $N_i \neq \{0\}$ such that, for every $x \in \mathbf{R}^n$ and $w \in N_i$, we have that

$$f_i(x) = f_i(x + w). \tag{15}$$

This definition is not the most general one (see, for example [15]), but has the advantage of being rather intuitive. As in the minimum surface example above, we will be mostly interested in the case where *the dimension of the invariant subspaces N_i is large compared to the total dimension of the problem.* We may then set up a linear transformation for each element, that transforms the *problem variables* $\{x_i\}_{i=1}^n$ into *internal variables* for the element, $\{u_i\}_{i=1}^{n_i}$ say, as expressed by the relation

$$u = W_i x. \tag{16}$$

In our example, this transformation matrix is obtained by composing the four variables at the corners of the considered discretised square with (11).

Once the transformation from problem variables to internal ones is defined, it is only necessary to store, compute and/or update the derivatives

$$\nabla f_i(u) \text{ and } \nabla^2 f_i(u), \tag{17}$$

whose dimensions are small.

Moreover, the situation arising in our model example is very common. The transformation can be separated into the product of two distinct operators: the first selects a (usually small) number of problem variables that explicitly appear in a given element, and the second defines a further linear transformation of these selected variables yielding the internal ones. We note that the first of these operators need not to be expressed in terms of a matrix, but merely in terms of a list of problem variables associated with the considered element. This saves a substantial amount of matrix manipulation. As in our example, it also frequently happens that the second of these transformations, that is from the selected problem variables to the internal ones, is independent of the element under consideration. Otherwise, frequently there are only a few different such transformations[2], which again saves storage and computation.

Since the invariant subspaces are not necessarily spanned by vectors of the canonical basis (as in our example), we see that the notion of partial separability extends that of sparsity.

More formally, we may state the following result.

Theorem 1 *Every twice continuously differentiable function from* $\mathbf{R}^n$ *into* $\mathbf{R}$ *having a sparse Hessian matrix is partially separable.*

The reader is referred to [15, section 2] for a more detailed discussion of this basic property.

Finally, it may be worthwhile to note that separable functions, that is functions of the form

$$f(x) = \sum_{i=1}^{p} f_i(x_i), \tag{18}$$

are clearly a very restricted case of partially separable functions. Because the variables x_i must all be different in (18), we prefer to call these functions *totally separable*.

2.2 Group partial separability

A significant proportion of practical large optimization problems exhibit another very important structure: the assignment of sets of element functions to *groups*. The most typical and pervasive example is probably that of the least-squares problem, where element functions are gathered into groups which are then squared.

Grouping nonlinear elements into sets is also desirable if we consider solving constrained problems: it is indeed necessary to distinguish the element functions associated with the objective from those associated with the constraints.

In order to achieve this grouping, we have to extend the notion of partial separability and define a slightly more general class. We will say that the real function $f(x)$ is a *group partially separable function* if and only if it is of the form

$$f(x) = \sum_{j=1}^{l} g_j(h_j(x)), \tag{19}$$

[2] In a finite element application, for instance, there are as many transformations as distinct element types in the problem.

where the *group functions* g_j are twice continuously differentiable functions from $\mathbf{R}$ into itself, and where their arguments $h_j(x)$ are partially separable functions from $\mathbf{R}^n$ into $\mathbf{R}$. Expanding the h_j, we obtain the expression

$$f(x) = \sum_{j=1}^{\ell} g_j \left(l_j(x) + \sum_{i \in J_j} f_i(x) \right), \tag{20}$$

where $l_j(x)$ is the linear part of $h_j(x)$, if any: the element functions $f_i(x)$ $(i \in J_j)$ now contain the purely nonlinear part of the j-th group.

Our model problem of the previous subsection also exhibits this structure. Indeed, we can choose the group functions as

$$g_{i,j}(y) = \frac{1}{m^2}\sqrt{y}, \tag{21}$$

(where y is called the *group variable*), the linear part of the groups as

$$l_{i,j}(x) = 1 \tag{22}$$

and the two nonlinear element functions of the (i,j)-th group as

$$f_1 = \frac{m^2}{2}(x_{i,j} - x_{i+1,j+1})^2 \text{ and } f_2 = \frac{m^2}{2}(x_{i,j+1} - x_{i+1,j})^2, \tag{23}$$

where we used again our double indexing convention for the group indices. In the same spirit as above, each of these element functions clearly has two elemental variables and only one internal.

An algorithm capable of minimising group partially separable functions is a very powerful tool, as it can be applied, without any modification, to nonlinear least-squares problems, constrained problems (in particular, to their (augmented) Lagrangian formulations) and many other cases. Such an algorithm, called SBMIN, has been developed by the authors in the context of the LANCELOT project that is presented below.

Clearly, we can interpret the use of group partial separability as an additional step (compared to partial separability alone) in the exploitation of the computational tree associated with a given real function of several variables, and then wonder if one more step could not bring further advantages. The resulting procedure would then become closer and closer to the exploitation of the complete tree, as advocated by McCormick and co-workers in the concept of *factorable functions* (see [19], instance). A complete discussion of the relative merits of partial vs complete exploitation of the computational tree associated with a given problem is outside the scope of the present paper, but should, at least, take the following arguments into account.

- There is a difference between using the complete computational graph [14] for efficient calculation of various quantities (for example, derivatives) that are requested by a given algorithm, and using the complete tree in the algorithm itself.

  The first approach is used by the new promising automatic differentiation algorithms, as discussed in [14], while the second raises the question of the possible use of derivatives of an order higher than two. This very interesting approach has been taken by R. Schnabel and co-authors (see [22] for an introduction to the subject), but the applications have been restricted to small problems and specific subsets of the higher derivatives. Whether or not

this type of techniques can be extended to large problems and complete higher derivatives (and whether this is desirable) remains an open question: if a Hessian matrix is large, a third order derivative tensor is huge... but again structure might play an important role here.

- Storage being a factor of importance for large problems, one has to reach a compromise between storage needs and efficiency for a given algorithm.

  The storage required for an algorithm using group partial separability is of the same order as that required for a specialised nonlinear least squares solver, and using specialised software for this last application is widely recommended. This indicates that the balance between storage and efficiency is reasonably achieved in our context.

- A number of problems involve "black boxes", that is user supplied routines for computing some problem dependent numerical functions, for which the structure is unknown. Although it may be possible in the future to process these routines using a specially designed "precompiler" that would extract information about their underlying structure, it is still necessary for today's algorithms to use these black boxes as they stand. Furthermore, if these black boxes link some specific subsets of variables together, this property should be expressed in the structural description of the complete problem. This can be handled very naturally in the (group) partially separable framework by identifying such black boxes with element functions.

2.3 Structure and new computer architectures

An important issue in the design of algorithms for large scale problems is their potential use of advanced computer architecture. The question is already important for small problems (see [22]), but is even more so for large ones, because the amount of calculation that is purely internal to the algorithm (therefore excluding problem functions evaluation) rises significantly and may well become the dominant computational cost.

The exploitation of partial separability and group partial separability on parallel computers is quite straightforward and efficient (see [17] and [16]). The overall idea is very simple. We note that the computational tasks in an algorithm using partial separability are of three types:

functions and derivatives evaluations : Because of the partial separable structure, all the element functions are independent, and their evaluation can be spread over the available processors in a purely asynchronous manner, an ideal situation in parallel computing.

internal linear algebra : This is one of the areas where the use of parallel computers have been most studied and for which efficient algorithms are now available. At the k-th iteration of a typical Newton-type method, partially separable problems give rise to linear systems of the form

$$\nabla^2 f(x_k)s_k = -\nabla f(x_k),. \tag{24}$$

From the linear algebra point of view, the structure of such systems is extremely similar to that of finite element systems, as the coefficient matrix is the sum of element Hessians

which must ultimately be assembled. Efficient algorithms for this class of problems are well studied and developed (see [1] and [12] for instance). Both iterative methods (eg. conjugate gradients variants) or direct algorithms (eg. multifrontal techniques) have been applied to large scale partially separable optimization problems in this context (see [17] and [6]).

internal element handling and updating : These are the algorithm's inner calculations that handle the element functions (including the Hessian approximation update for quasi-Newton methods). Again, these can be shared by the available processors because of the independence of these functions.

Of course, if one restricts one's attention to sparsity of the Hessian matrix, important speedups can still be achieved in internal linear algebra, as discussed above, but parallelisation of the two other types of computational tasks is then more difficult.

As a conclusion, we may say that the use of partial and group partial separability facilitates the algorithms to use the potentialities of parallel computers, providing a natural problem decomposition, which then results in an efficient partitioning of the computational work amongst the processors.

2.4 The sources of (group) partially separable problems

One of the major sources of (group) partially separable is the discretisation of continuous problems. Both finite differences and finite elements approximations result in problems of this type, mostly because of the "locality" of the involved operators, that only relate variables corresponding to "neighbouring" discretisation points. An interesting collection of such problems has been recently gathered by J. Moré in [20]. This collection features, amongst others, chemical engineering applications, variational inequalities, biomedical modelling, boundary value problems and elasticity analysis.

Nonlinear network problems form another important source of group partially separable problems, ranging from urban traffic equilibria (see [13] for a excellent survey of this type of applications) to water and gas resource management [21]. The structure again results from the same "locality" property that we mentioned for discretised problems: nonlinear variables explicitly interact when they are close to each other in the considered network (they are typically associated to arcs incident to a given node, or to nodes at the extremities of a given arcs).

Other classes of problems that often exhibiting partially and/or group partially separable structure include

- multiperiod planning models,
- input-output macro-economic models,
- multiobjective optimization,
- nonlinear matrix equations.

It seems therefore fair to say that (group) partially separable problems actually occur in most fields where large scale nonlinear optimization is itself relevant. This is not surprising if one

recalls Theorem 1, but it is worthwhile to note that the decomposition (14) or (19) often arises very naturally in the problem formulation, its description therefore requiring a minimal amount of additional effort.

3 The LANCELOT software project

The LANCELOT software project was started by the authors more than two years ago. The meaning of the LANCELOT acronym is explained by the banner displayed in Figure 2.

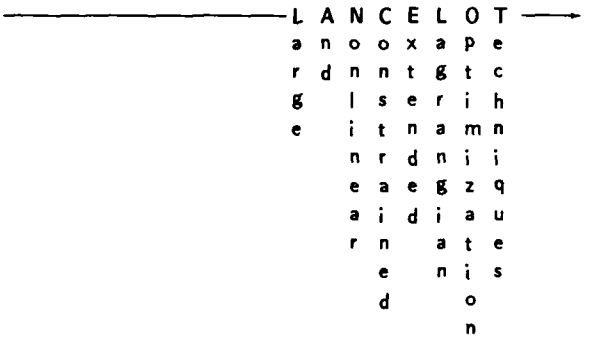


Figure 2: The LANCELOT banner

According to this banner, LANCELOT 's purpose is to attack large problems involving non-linear objectives and/or constraints by using techniques based on the Lagrangian function[3].

The main characteristics of the LANCELOT software can be described as follows.

Use of problem structure: As discussed in the first part of the paper, the use of problem structure is the only reasonable way to tackle large problems. For the reasons explained above, (group) partial separability seems the right concept to invoke for this purpose: LANCELOT will therefore explicitly handle these types of structure.

On the other hand, this capacity to exploit structure will inevitably result is some inefficiency when handling unstructured problem. Care will be taken to ensure that this inefficiency is not too severe.

Efficiency and reliability: LANCELOT will be efficient and reliable. At the beginning of a software project, such a statement is of course a little preposterous. What is meant is that the algorithm design and implementation will systematically use efficient and reliable structures and methods.

[3]The "s" at the end of "techniques" is intentional: the present pilot version of the software already uses two different exytensions or augmentations of the Lagrangian.

An important point is that strong emphasis is put on the theoretical justification of the algorithms and structures used by the software. Suitable convergence theory should be available (and, in part, already is: see [3], [5], [7], [11]). Furthermore, the final implementations and the studied algorithms should differ as little as possible. This requirement of a well established supporting theory is not considered as a sufficient condition guaranteeing software reliability, but as truly necessary.

This theoretical support will be completed by intensive testing on model problems, both practical and more academic. The first ones are crucial because they reflect best the situation in which the software will be applied. The second ones are important too because they introduce sometimes extreme numerical difficulties: the performance of the system when faced with these difficulties is easier to isolate and to improve on such idealised problems (see [4] and [6] for preliminary tests).

Of course, the final efficiency and reliability achieved is best judged by the end-users!

Scope: The domain of application of the LANCELOT system will include a large part of smooth nonlinear optimization problems. Although primarily focussed on large scale problems, LANCELOT will also cover small and medium size ones. It will exploit specific types of constraints, including

- none (unconstrained problems),
- simple bounds on the problem's variables,
- linear network-type constraints,
- general linear constraints,
- convex constraints, where the feasible domain is such that a (possibly approximate) projection can be efficiently computed,
- general nonlinear nonconvex constraints.

Extension to nonsmooth problems is of interest, but is not planned at this stage. The emphasis will be on nonlinear problems: LANCELOT is not presently intended to compete with large linear programming packages.

Ease of use: Inputting problems to LANCELOT will be reasonably easy. A standard input format for nonlinear problems (SDIF) has been proposed by the authors to achieve this objective. It features a number of facilities to describe problem structure, along the lines analysed in Section 3 of this paper. For instancee, it automatically handles multi-indexed variables as they naturally arise from discretizations of multi-dimensional problems (as the minimum surface example presented above). This format has been formalised in [8] and has already been used for the input of a fairly sizeable set of large problems. Although not as complete as a true modelling language, it nevertheless provides an important practical help in specifying structured problems, as well as invaluable internal data consistency checks.

It is the authors' experience that large structured nonlinear problems arising from applications[4] have been fully specified using the SDIF, validated and solved by the pilot version of LANCELOT , the whole process taking less than one hour (which we consider quite reasonable).

Adaptability: The LANCELOT system is also designed in a very modular and hierarchical way which, in turn, provides a good adaptability of the system to extensions, both algorithmic and implementation oriented.

This organisation is also made necessary by the need to provide more than one methodology for some of the algorithmic parts of the system: preconditioning the large linear systems arising from Newton's equation requires, for example, that several strategies (simple diagonal scaling, incomplete factorisation, modified band techniques, ...) be available to the user.

The adaptability of the LANCELOT software is also enhanced by the choice of a reverse communication interface for the system.

Portability: Because the LANCELOT system is designed to be easily portable, the programming language most commonly used for scientific applications, Fortran 77, has been chosen for its development. Strict conformity with the standard of the language is enforced at all levels of the system. Transfer between different machines (CRAY, IBM, DEC and SUN mainframes and workstations, ...) and operating systems (VM/CMS, VMS, UNIX, ...) also takes place during the development phases, in order to ensure maximal portability, not only of the end-product, but also of the successive pilot codes.

Following the arguments of Section 2.3, the code is also designed in a way that has the potential to make it efficient on parallel and/or vector computers.

4 Conclusions

We have shown how the structure of large complex nonlinear problems can be analysed using the concept of (group) partial separability. We have also discussed some aspects of the LANCELOT project, whose purpose is the implementation of this approach in a practical software tool.

Development of the LANCELOT system is ongoing, both from the theoretical and software viewpoints. As alluded to above, some layers and functionalities of the system are already operational and being tested. Detailed numerical experiments with these modules will shortly be reported elsewhere.

The coherence of theoretical concepts with their applications to "real world" problems and the coherence of the theoretical concepts between themselves are considered by the authors to be of central importance. An approach to large scale nonlinear programming that has this type of coherency has been outlined in this paper, ranging from abstract convergence theory and

[4]The examples we have in mind here were proposed by practitioners in the fields of energy modelling and finite element applications. They involve more that 1000 nonlinear variables and some of them have nonlinear constraints.

structure analysis to practical software implementations. In a domain just reaching maturity, this unified perspective is desirable.

References

[1] P. Amestoy and I.S. Duff, "Vectorization of a multiprocessor multifrontal code", International Journal of Supercomputers Applications, vol. 3, pp. 41-59, 1989.

[2] T.F. Coleman and Y. Li (eds.), "Large Scale Numerical Optimization", SIAM Publications (to appear), 1990.

[3] A.R. Conn, N.I.M. Gould and Ph.L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds", SIAM Journal on Numerical Analysis, vol. 25, pp. 433–460, 1988. Correction, same journal, vol. 26, pp. 764–767, 1989

[4] A.R. Conn, N.I.M. Gould and Ph.L. Toint, "Testing a class of methods for solving minimization problems with simple bounds on the variables", Mathematics of Computation, vol. 50(182), pp. 399–430, 1988.

[5] A.R. Conn, N.I.M. Gould and Ph.L. Toint, "A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds", Report 88/23, Dept of Mathematics, FUNDP Namur (B), 1988.

[6] A.R. Conn, N.I.M. Gould, M. Lescrenier and Ph.L. Toint, "Performance of a multifrontal scheme for partially separable optimization", Report 88/4, Dept of Mathematics, FUNDP Namur (B), 1988.

[7] A.R. Conn, N.I.M. Gould and Ph.L. Toint, "Convergence of quasi-Newton matrices generated by the Symmetric Rank One update", Mathematical Programming (to appear), 1989.

[8] A.R. Conn, N.I.M. Gould and Ph.L. Toint, "A proposal for a Standard Data Input Format for large-scale nonlinear programming problems", Report 89/18, Dept of Mathematics, FUNDP Namur (B), 1989.

[9] A.R. Conn, N.I.M. Gould and Ph.L. Toint (eds.), "Large Scale Optimization", Mathematical Programming, vol. 45(3), 1989.

[10] A.R. Conn, N.I.M. Gould and Ph.L. Toint (eds.), "Large Scale Optimization — Applications", Mathematical Programming, vol. 48(1), 1990.

[11] A.R. Conn, N.I.M. Gould, A. Sartenaer and Ph.L. Toint, "Global convergence of a class of trust region algorithms for optimization using inexact projections on convex constraints", Report 89/17, Dept of Mathematics, FUNDP Namur (B), 1989.

[12] I.S. Duff, N.I.M. Gould, M. Lescrenier and J.K. Reid, "The multifrontal method in a parallel environment", in "Reliable Scientific Computation", M.G. Cox and S.J. Hammarling (eds.), Oxford University Press, 1988.

[13] M. Florian, "Mathematical programming applications in national, regional and urban planning", in "Mathematical Programming: recent developments and applications", M. Iri and K. Tanabe (eds.), Kluwer Academic Publishers, Dordrecht, pp. 57–82, 1989.

[14] A. Griewank, "On automatic differentiation", in "Mathematical Programming: recent developments and applications", M. Iri and K. Tanabe (eds.), Kluwer Academic Publishers, Dordrecht, pp. 83–108, 1989.

[15] A. Griewank and Ph.L. Toint, "On the unconstrained optimization of partially separable functions", in "Nonlinear Optimization 1981" (M.J.D. Powell, ed.), Academic Press, London, 1982.

[16] M. Lescrenier, "Partially separable optimization and parallel computing", in "Parallel Optimization on Novel Computer Architectures", R.R. Meyer and S. Zenios (eds.), A.C. Baltzer Scientific Publishing Co, Switzerland, 1989.

[17] M. Lescrenier and Ph.L. Toint, "Large scale nonlinear optimization on the FPS164 and CRAY X-MP vector processors", International Journal of Supercomputer Applications, vol. 2(1), pp. 66–81, 1988.

[18] O.L. Mangasarian and R.R. Meyer (eds.), "Parallel Methods in Mathematical Programming", Mathematical Programming Vol. 42(2), 1988.

[19] G.P. McCormick, "Nonlinear programming: theory, algorithms and applications", Academic Press, New York, 1983.

[20] J.J. Moré, "A collection of nonlinear model problems", Report ANL/MCS-P60-0289, Argonne National Laboratory, Argonne (USA), 1989.

[21] A.J. Osiadacz and D.J. Bell, "Optimization techniques for large networks: gas and water", in "Simulation and optimization of large systems", A.J. Osiadacz (ed.), Clarendon Press, Oxford, pp. 175–192, 1988.

[22] R.B. Schnabel, "Sequential and parallel methods for unconstrained optimization", in "Mathematical Programming: recent developments and applications", M. Iri and K. Tanabe (eds.), Kluwer Academic Publishers, Dordrecht, pp. 227–262, 1989.

[23] M.J. Todd, "Recent developments and new directions in linear programming", in "Mathematical Programming: recent developments and applications", M. Iri and K. Tanabe (eds.), Kluwer Academic Publishers, Dordrecht, pp. 109–158, 1989.

Adaptive Polynomial Preconditioning for HPD Linear Systems

Steven F. Ashby[†]
Lawrence Livermore National Laboratory

Thomas A. Manteuffel[‡]
University of Colorado at Denver

James S. Otto[‡]
University of Colorado at Denver

December 11, 1989

Abstract

This paper explores the use of adaptive polynomial preconditioning for hermitian positive definite linear systems, $Ax = b$. Such preconditioners are easy to employ and well-suited to vector and/or parallel machines. After examining the role of polynomial preconditioning in conjugate gradient methods, we discuss the least squares and Chebyshev preconditioning polynomials. We determine those eigenvalue distributions for which each is well-suited. We also describe an *adaptive procedure* for dynamically computing the optimum Chebyshev polynomial preconditioner. Finally, in a variety of numerical experiments on a Cray X-MP/48 and Alliant FX/8, we demonstrate the effectiveness of adaptive polynomial preconditioning. Our results suggest that relatively low degree (2–16) polynomials are usually best.

Prepared for the proceedings of the Ninth International Conference on Computing Methods in Applied Sciences and Engineering, Paris, January 29–February 2, 1990.

[†] This work was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, Department of Energy, by Lawrence Livermore National Laboratory under contract W-7405-ENG-48.
[‡] This research was supported in part by the National Science Foundation under grant DMS 8704109.

1. Introduction.

This paper examines polynomial preconditioning for hermitian positive definite (hpd) linear systems of equations, $Ax = b$. Such systems arise in many scientific applications. For example, the matrix resulting from the 7-point finite difference approximation to a three-dimensional self-adjoint elliptic PDE is large, sparse, and hpd. The conjugate gradient (CG) method of Hestenes and Stiefel [19] is a popular and effective solution technique for these linear systems, especially when combined with a preconditioner. The incomplete Cholesky (IC) factorization of Meijerink and van der Vorst [23] is often an effective preconditioner for CG, but other choices include Jacobi and SSOR. In this paper, we will consider *polynomial preconditioning* for conjugate gradient methods. That is, we will solve

$$C(A)Ax = C(A)b \tag{1.1}$$

where $C(\lambda)$ is a *preconditioning polynomial* and $C(A)$ is the associated polynomial preconditioner. We will assume that $C(\lambda)$ has real coefficients, in which case both $C(A)$ and $C(A)A$ are hermitian.

Polynomial preconditioning has several advantages. First, it is simple: there are only two intrinsic operations, matrix-vector multiplication (*matvec*) and vector addition (*saxpy*). The user need only specify the polynomial degree and initialize a few parameters; the preconditioning may be implemented automatically. Since polynomial preconditioning requires only matrix-vector multiplication, it is ideally suited to "matrix-free" computations [6].

Polynomial preconditioning is also versatile. As discussed in [3], polynomial preconditioners may be used in variety of CG methods. The best-known of these is the PCG method of Concus, Golub, and O'Leary [11]. However, one may exploit the special properties of polynomial preconditioners to devise new CG methods [4]. The key to this versatility is commutativity: a polynomial in A commutes with A. In other words, the preconditioner C commutes with the matrix A, a property generally not shared by other preconditioners.

The main advantage of polynomial preconditioning is its suitability for vector and/or parallel architectures. If the matvec is vectorizable, as when A has a regular sparsity structure, polynomial preconditioning is effective on vector machines; see [3, 12, 21, 22]. In contrast, incomplete factorizations are difficult to vectorize, especially for the nonexpert. It is also possible to chain the matvecs (implicit in the preconditioning), thereby enhancing data locality and reducing memory traffic; see [9, 10, 27, 28]. Polynomial preconditioning is also effective on parallel machines, especially those on which inner products are a bottleneck. This is so because polynomial preconditioned CG methods converge in fewer steps than unpreconditioned CG, and thus compute fewer inner products, albeit at the cost of several matvecs per step instead of one. However, in many applications the matvec is parallelizable, and so we can expect an overall reduction in CPU time on some architectures by substituting matvecs for inner products. The effectiveness of polynomial preconditioning has been demonstrated on an Alliant FX/8 [24] and on a Connection Machine [7].

A common complaint about polynomial preconditioning is that, unlike incomplete Cholesky, it is only marginally better than unpreconditioned CG, which we will call CGHS. This criticism is misguided because it is based on the number of iterations required for convergence, rather than the CPU time. Although ICCG may take fewer iterations than PPCG, the latter often takes less time [12, 22]. Moreover, even when incomplete Cholesky is more effective, it can be

further accelerated by using a polynomial preconditioner. Specifically, one applies CG to

$$C(M^{-1}A)M^{-1}Ax = C(M^{-1}A)M^{-1}b \tag{1.2}$$

where M is the matrix representation of the incomplete factorization. Notice that if M and A are hermitian, then so is the preconditioner $C(M^{-1}A)M^{-1}$. Several CG methods are applicable under these conditions [1, 2, 4].

We emphasize that polynomial preconditioning is most effective on parallel machines, where it typically outperforms incomplete Cholesky. Moreover, since polynomial preconditioning can be implemented automatically, it is as easy to use as CGHS. Thus, any improvement over CGHS is obtained essentially for free.

1.1. Outline of Paper. In the next section we review preconditioned CG methods. After presenting two implementations of a CG method, we discuss the various ways in which a polynomial preconditioner can be used. In § 3 we examine polynomial preconditioning. In particular, we discuss the least squares and Chebyshev preconditioning polynomials, study them in the context of CG methods, and show that the latter minimizes a bound on the condition number of the preconditioned matrix. We compare the two polynomials in § 4. In a variety of numerical experiments we determine those eigenvalue distributions for which each is well-suited. In § 5 we describe an *adaptive procedure* for dynamically computing λ_c and λ_d, the smallest and largest eigenvalues of our hpd matrix A. These extreme eigenvalues are needed to determine the best Chebyshev polynomial preconditioner for many eigenvalue distributions. We also present numerical results demonstrating the accuracy and efficiency of the adaptive procedure. Finally, in § 6, we summarize some numerical experiments which demonstrate the effectiveness of polynomial preconditioning on a variety of test problems. Our results suggest that relatively low degree (2–16) polynomials are usually best.

2. Preconditioned CG Methods.

In this section we examine the use of polynomial preconditioners in CG methods. To do this it is useful to first characterize CG methods. The discussion below is culled from [3] and [4].

In [4] it is shown that any CG method is characterized by three matrices: an hpd inner product matrix B, a left preconditioning matrix C, and the original system matrix A. The resulting CG method, $CG(B,C,A)$, minimizes $||e_i||_B = \langle Be_i, e_i\rangle^{\frac{1}{2}}$ over $V_i(CA, Cr_0)$, where

$$V_i(CA,Cr_0) = \text{span}\{Cr_0,(CA)Cr_0,(CA)^2Cr_0,...,(CA)^{i-1}Cr_0\} \tag{2.1}$$

is a Krylov subspace of dimension at most i, e_i is the error in the current iterate, r_0 is the initial residual, and $\langle\cdot,\cdot\rangle$ denotes the usual Euclidean inner product. By specifying the inner product matrix B, we obtain a particular CG method. For example, when A is hpd, one may take $B = A$ and $C = I$, which yields CGHS, the original method of Hestenes and Stiefel. If one takes $B = A^2$ and $C = I$, the conjugate residual (CR) method results.

The most robust implementation of a CG method is the so-called Odir algorithm [30]:

$$p_0 = Cr_0 \tag{2.2}$$

$$\alpha_i = \frac{\langle Be_i, p_i \rangle}{\langle Bp_i, p_i \rangle} \tag{2.3}$$

$$x_{i+1} = x_i + \alpha_i p_i \tag{2.4}$$

$$r_{i+1} = r_i - \alpha_i A p_i \tag{2.5}$$

$$\gamma_i = \frac{\langle BCAp_i, p_i \rangle}{\langle Bp_i, p_i \rangle} \tag{2.6}$$

$$\sigma_i = \frac{\langle Bp_i, p_i \rangle}{\langle Bp_{i-1}, p_{i-1} \rangle} \tag{2.7}$$

$$p_{i+1} = CAp_i - \gamma_i p_i - \sigma_i p_{i-1} \tag{2.8}$$

where x_i is the current iterate, $r_i = b - Ax_i$ is its residual, and p_i is the current direction vector. This algorithm converges to the solution of $Ax = b$ whenever BCA is hermitian. (For necessary and sufficient conditions, see [4, 13].) Since the error e_i is unknown, B must be chosen so that α_i is *computable*. For example, $B = A$ and $B = A^2$ yield computable CG methods. One can also express α_i in terms of C, which allows greater flexibility in designing computable CG methods. Specifically, a CG method is computable whenever C^*Be_i is computable [4].

When BCA is hpd, the cheaper and more familiar Omin algorithm [30] will converge:

$$s_0 = Cr_0 \tag{2.9}$$

$$p_0 = s_0 \tag{2.10}$$

$$\alpha_i = \frac{\langle Be_i, s_i \rangle}{\langle Bp_i, p_i \rangle} \tag{2.11}$$

$$x_{i+1} = x_i + \alpha_i p_i \tag{2.12}$$

$$r_{i+1} = r_i - \alpha_i A p_i \tag{2.13}$$

$$\beta_i = \frac{\langle Be_{i+1}, s_{i+1} \rangle}{\langle Be_i, s_i \rangle} \tag{2.14}$$

$$s_{i+1} = Cr_{i+1} \tag{2.15}$$

$$p_{i+1} = s_{i+1} + \beta_i p_i. \tag{2.16}$$

Whereas Odir uses a 3-term recursion involving CAp_i to generate the new direction vector p_{i+1}, Omin uses a 2-term recursion involving the preconditioned residual, s_{i+1}. Unfortunately, Omin may "stall" when BCA is indefinite, in which case the more expensive Odir, or an Odir/Omin hybrid, algorithm should be used [4, 8].

When A is hpd and $C = C(A)$, there are several choices for the inner product matrix B. A few of the resulting CG methods are listed in Table 2.1. Notice that BCA is hermitian in all cases, and so Odir converges. The Odir restrictions in Table 2.1 are sufficient to insure that B is hpd. The Omin restrictions are sufficient to guarantee that both B and BCA are hpd. The first method is PCG. Like CGHS, it minimizes the A-norm of the error, but does so over a preconditioned Krylov subspace. Although the matrix A must be hpd to define a norm, the preconditioner $C(A)$ only needs to be hermitian for Odir. If $C(A)$ is hpd, one may use the more efficient Omin algorithm. The method GCGHS, which is CGHS on CA, minimizes in the $B = C(A)A$ norm, and so $C(\lambda)$ must be chosen so that the preconditioned matrix is hpd. As we will see, this is possible. The advantage of GCGHS is this: if $C(A)$ is a good preconditioner, then $C(A)A \approx I$, and so the method more nearly minimizes the Euclidean norm of the error. The

Method	B	CA	Odir Restrictions	Omin Restrictions
PCG	A	$C(A)A$	A hpd	A hpd, $C(A)$ hpd
GCGHS	$C(A)A$	$C(A)A$	$C(A)A$ hpd	C(A)A hpd
PCR	$AC(A)A$	$C(A)A$	$C(A)$ hpd	A hpd, $C(A)$ hpd
PPCR	A^2	$C(A)A$	none	C(A)A hpd
GCR	$(C(A)A)^2$	$C(A)A$	none	C(A)A hpd

Table 2.1: Polynomial Preconditioned CG Methods for HPD A

next method, PCR, requires that $C(A)$ be hpd, in which case $C(A)A$ is hpd because A is hpd. The last two methods, PPCR and GCR, employ $B = A^2$ and $B = (C(A)A)^2$, respectively. The Odir algorithm will converge for either method; Omin is applicable if $C(A)A$ is hpd. Note that PPCR is possible because $CA = AC$ (which implies that BCA is hermitian), an advantage of C being a polynomial in A. The last method, GCR, is simply CR applied to the preconditioned matrix, CA. We remark that each method except PCG is applicable to hermitian indefinite A; see [5]

Finally, we note that the spectral and B condition numbers of CA are identical for each of the methods in Table 2.1. That is, $\kappa_I(CA) = \kappa_B(CA)$, where $\kappa_B(CA) = \|CA\|_B\|(CA)^{-1}\|_B$. Thus, estimates for the extreme eigenvalues of CA yield a bound on $\kappa_I(CA)$, which may be used to implement a stopping criterion based on the true error, rather than the more usual residual error. Eigenvalue estimates for CA are easily obtained from the CG iteration parameters [4, 11]. This is also the basis for the adaptive procedure discussed in § 5.

3. Polynomial Preconditioning.

In this section we examine several choices for $C(\lambda)$. We wish to choose C to accelerate convergence of the CG iteration. One usually chooses C to approximate A^{-1} in some sense, for example, by choosing $C(\lambda) \approx \lambda^{-1}$. Of course, there are several ways of doing this. As we will see, there is no single "best" polynomial; the proper choice of $C(\lambda)$ depends on the eigenvalue distribution of A, which is seldom known a *priori*.

A simple choice for $C(\lambda)$ is based on the Neumann series. Let $A = M - N$ and consider

$$A^{-1} = (M - N)^{-1} = (I + G + G^2 + G^3 + \cdots)M^{-1} \tag{3.1}$$

where $G = M^{-1}N$. If the spectral radius of G is less than one, the series converges. We obtain our polynomial approximation to A^{-1} by truncating the Neumann series [2, 7, 12, 22]. The advantage of this polynomial is its simplicity: there are no parameters to estimate. Unfortunately, it may yield a poor preconditioner. If one desires a polynomial preconditioner of degree $m - 1$, one can do much better than the Neumann series polynomial. For example, Jordan [22] has shown that the Chebyshev polynomial (§ 3.2) is superior. Experiments also suggest that the optimum degree for the Neumann series polynomial is two [2, 12, 22], whereas the optimum Chebyshev or least squares polynomial degree is often higher [3, 24, 27].

To obtain a better preconditioner, recall that $C(A)$ should approximate A^{-1} in some sense. That is, $C(\lambda)$ should be the "best" polynomial approximation to λ^{-1} on some set S containing the spectrum of A, $\sigma(A)$. Since A is hpd, we will take $S = [c, d]$, where $0 \leq c \leq d$. Ideally, $c = \lambda_c$ and $d = \lambda_d$, the smallest and largest eigenvalues of A. We next define the "best"

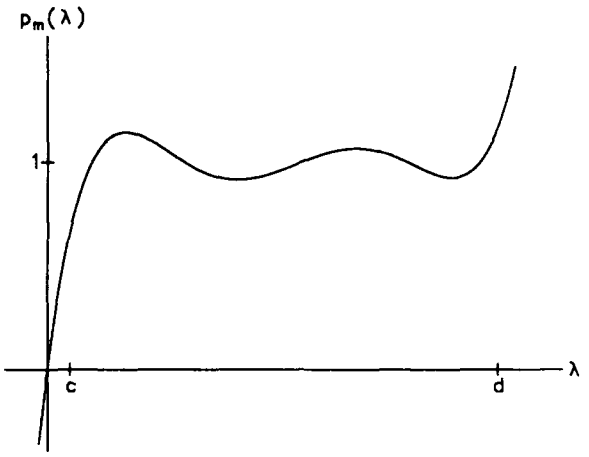


Figure 3.1: Least squares preconditioned polynomial ($m = 5$) for $S = [0, 20]$

polynomial to be that one which solves the following approximation problem:

$$\min_{C \in \pi_{m-1}} \|1 - C(\lambda)\lambda\| \tag{3.2}$$

where π_{m-1} is the set of polynomials of degree at most $m - 1$. All that remains is to specify the norm.

3.1. The Least Squares Polynomial Preconditioner. Let us define the inner product

$$\langle f, g \rangle = \int_c^d f(\lambda)\overline{g(\lambda)}\omega(\lambda)d\lambda \tag{3.3}$$

where $\omega(\lambda)$ is a positive weight function on $S = [c, d]$. It induces the following norm:

$$\|f\|_\omega^2 = \int_c^d |f(\lambda)|^2\omega(\lambda)d\lambda. \tag{3.4}$$

The solution to (3.2) in this norm is called the weighted *least squares* polynomial. The associated *preconditioned* polynomial, $p_m(\lambda) = C(\lambda)\lambda$, is illustrated in Figure 3.1. (We call $p_m(\lambda)$ the preconditioned polynomial because $p_m(A)$ is the preconditioned matrix.) Since the related residual polynomials, $r_m = 1 - C(\lambda)\lambda$, are orthogonal with respect to the weight function $\lambda\omega(\lambda)$, the least squares polynomial may be computed via a three-term recursion, which is computationally stable and efficient. See also [21, 27].

Unlike the Chebyshev polynomial described below, the least squares polynomial is biased in its suppression of the eigenvalues of A. For example, when $\omega \equiv 1$, the eigenvalues of larger modulus are mapped closer to 1 than those of smaller modulus. If the eigenvalue distribution

of A were known, one could choose ω to exploit this bias. In particular, one might consider a *Jacobi* weight function,

$$\omega(\lambda) = (d-\lambda)^{\alpha}(\lambda - c)^{\beta}, \quad \alpha, \beta > -1. \tag{3.5}$$

By appropriately choosing α and β, one portion of $\sigma(A)$ could be emphasized over another.

One should choose the weight function ω so that $p_m(\lambda) = C(\lambda)\lambda$ is positive on $[\lambda_c, \lambda_d]$. This guarantees that $p_m(A)$ is hpd, which makes practicable the Omin implementation of each method in Table 2.1. (Note that PCG and PCR are applicable because $C(A)$ is hpd whenever $p_m(A) = C(A)A$ is hpd.) If one employs a Jacobi weight function on $[c,d]$, one may show [29, page 166] that $p_m(\lambda) > 0$ on $[c,d]$ if $(\alpha,\beta) \in W_1 = \{(\alpha,\beta) : \alpha \geq -1/2,\ \beta \geq -1/2\}$, which includes the Legendre weight function $\omega \equiv 1$ $(\alpha = \beta = 0)$.

Saad [27] has noted that the least squares polynomial is relatively insensitive to c, and so one may take $c = 0$. Then, if a Jacobi weight function is used, the least squares polynomial is given by a scaled and translated Jacobi polynomial corresponding to α and $\beta + 1$. Moreover, if $(\alpha,\beta) \in W_2 = \{(\alpha,\beta) : -1 < \alpha < -1/2,\ \beta > -1\}$, the relative extrema of the least squares polynomial decrease in magnitude on S [29]. This property, which is in stark contrast to the equioscillation property of the Chebyshev preconditioned polynomial (see below), may be used to bias the preconditioner toward the large eigenvalues of A. This property also insures that $p_m(\lambda)$ is positive on $(0,d]$, which is important in many of the CG methods discussed in § 2.

Despite its bias, the least squares polynomial yields an effective preconditioner in many cases [21, 27]. Since one may take $c = 0$, there is no need to estimate the smallest eigenvalue of A. The right endpoint is usually taken to be the Gershgorin estimate for λ_d. We discuss a more sophisticated *adaptive procedure* for dynamically estimating λ_c and λ_d in § 5.

3.2. The Chebyshev Polynomial Preconditioner. Another interesting norm is the uniform norm:

$$\|f\|_{\infty} = \max_{\lambda \in S} |f(\lambda)|. \tag{3.6}$$

The solution to (3.2) in this norm is obtained from a shifted and scaled Chebyshev polynomial:

$$C(\lambda)\lambda = 1 - \frac{T_m(\frac{d+c-2\lambda}{d-c})}{T_m(\frac{d+c}{d-c})} \tag{3.7}$$

where $T_m(x)$ is the m^{th} Chebyshev polynomial of the first kind [25]. Notice that $C(\lambda)$ is indeed a polynomial in λ. It is attractive for several reasons. First, like the least squares polynomial, it may be computed from a three-term recursion, which is computationally convenient. Second, since this polynomial is explicitly known, it is much easier to devise an adaptive procedure for dynamically computing the optimal endpoints c and d. Finally, this polynomial is unbiased in its suppression of those eigenvectors constituting the error. In other words, the Chebyshev preconditioning polynomial is well-suited to those matrices whose eigenvalues are densely and nearly uniformly distributed throughout the interval $S = [c,d]$. See § 4.

This last fact follows from the Chebyshev minimax property, which states that the preconditioned polynomial, $p_m(\lambda) = C(\lambda)\lambda$, equioscillates about 1; see Figure 3.2. This equioscillation property has several other implications. For example, if $\sigma(A) \subset [c,d]$, then $\sigma(p_m(A)) \subset [1-\epsilon_m, 1+\epsilon_m]$, where $\epsilon_m = \|1 - p_m\|_{\infty} = |T_m^{-1}(\frac{d+c}{d-c})|$. Since $\epsilon_m < 1$, the preconditioned matrix,

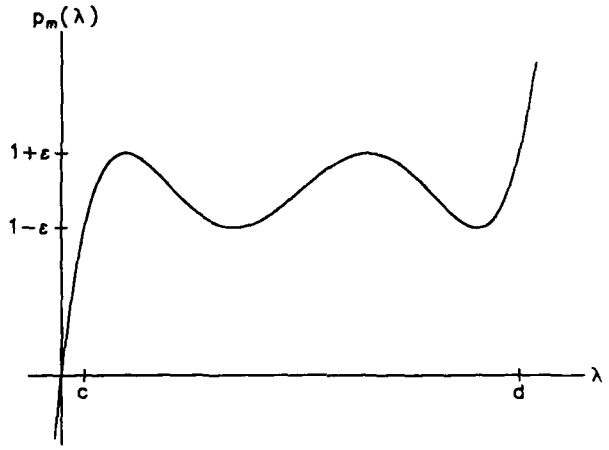


Figure 3.2: Chebyshev preconditioned polynomial ($m = 5$) for $S = [1, 20]$

$p_m(A)$, is hpd. One may therefore apply CGHS to $p_m(A)$, yielding the method we call GCGHS. PCG is also applicable because $C(A)$ is hpd. Note that the spectral condition number of $p_m(A)$, $\kappa(p_m(A))$, satisfies

$$\kappa(p_m(A)) \leq \frac{1+\epsilon_m}{1-\epsilon_m} \tag{3.8}$$

when $\sigma(A) \subset [c, d]$. Since ϵ_m is a monotonically decreasing function of m, this bound may be made as small as desired by taking m large enough. Specifically, if

$$m \geq \frac{\cosh^{-1}(\frac{\delta+1}{\delta-1})}{\cosh^{-1}(\frac{\kappa(A)+1}{\kappa(A)-1})}, \tag{3.9}$$

then $\kappa(p_m(A)) < \delta$ for any $\delta > 1$. This follows from the definition of $T_m(\lambda)$ for $\lambda > 1$.

The bound (3.8) yields an estimate of the number of CG steps required for convergence. One needs approximately

$$\frac{\ln(\delta/2)}{\ln(cf)} \tag{3.10}$$

steps to reduce the error by an amount δ [18], where

$$cf = cf(p_m(A)) = \frac{\sqrt{\kappa(p_m(A))} - 1}{\sqrt{\kappa(p_m(A))} + 1} \tag{3.11}$$

is the *CG convergence factor* for the hpd matrix $p_m(A)$. If the eigenvalues of $p_m(A)$ are uniformly distributed throughout $[1-\epsilon_m, 1+\epsilon_m]$, then (3.10) is fairly accurate. Since $\kappa(p_m(A)) \leq \kappa(A)$ for $m > 1$, a Chebyshev polynomial preconditioned CG method will usually converge in fewer iterations than the unpreconditioned CGHS method. Of course, each iteration is more

expensive, requiring m matvecs instead of one. We remark that $\kappa(p_m(A))$ is minimized when $c = \lambda_c$ and $d = \lambda_d$.

The Chebyshev polynomial preconditioner is also optimum in that it minimizes a bound on $\kappa(C(A)A)$. This is a consequence of the following

Theorem 3.1. *A solution to*

$$\min_{C \in \pi_{m-1}} \frac{\max_{\lambda \in S} |C(\lambda)\lambda|}{\min_{\lambda \in S} |C(\lambda)\lambda|} \tag{3.12}$$

is given by the Chebyshev preconditioning polynomial.

Proof: First observe that (3.12) does not possess a unique solution. In particular, if Q solves (3.12), then so does γQ, where γ is any nonzero constant. We may assume $C(\lambda)\lambda > 0$ for $\lambda \in S$ without loss of generality. (If $C(\lambda) \leq 0$ for some $\lambda \in S$, then (3.12) is unbounded.) Thus, we may restrict ourselves to those polynomials $C(\lambda)$ for which

$$1 - \min_{\lambda \in S} (C(\lambda)\lambda) = \max_{\lambda \in S} (C(\lambda)\lambda) - 1 = \epsilon(C) = \epsilon. \tag{3.13}$$

The problem (3.12) is now equivalent to minimizing $\frac{1+\epsilon}{1-\epsilon}$. This is, in turn, equivalent to solving (3.2) in the uniform norm. ■

Remark: If p_m is the Chebyshev preconditioned polynomial for S and $\sigma(A) \subset S$, the ratio in (3.12) gives a bound on the condition number of $p_m(A)$. Moreover, this bound is minimized with respect to S when $S = [\lambda_c, \lambda_d]$.

This theorem is similar to Theorem 3 in [21], but our proof is different. It shows the equivalence of the minimax approximation problem (3.2) and the minimization problem (3.12). We also remark that Rutishauser [26] was the first to propose Chebyshev polynomial preconditioning for CGHS; his motive was to mitigate its rounding errors. We advocate polynomial preconditioning because it is well-suited to vector and/or parallel architectures.

3.3. Implementation. To implement least squares or Chebyshev polynomial preconditioning, one neither explicitly forms the powers of A nor determines the coefficients of $C(\lambda)$. Instead, one executes m steps of a nonstationary 2-step iteration. (In the case of Chebyshev polynomial preconditioning, one uses the Chebyshev iteration [18].) Specifically, one applies the 2-step iteration to the linear system $Aw = v$ with $w_0 = 0$, where v is the vector to be preconditioned, usually the residual. One may show that $w_m = C(A)v$. Note that we need only $m - 1$ matrix-vector multiplications because the final residual need not be computed. We also remark that the three-term recursion underlying the least squares and Chebyshev polynomials insures the stable evaluation of the preconditioning polynomial $C(\lambda)$. See also [21, 27].

4. Chebyshev versus Least Squares.

In this section we compare the least squares and Chebyshev preconditioning polynomials in a variety of numerical experiments. We will qualitatively describe those matrices for which the least squares polynomial yields a better preconditioner than the optimal Chebyshev preconditioner. Moreover, we will explain why this is so. The importance of the stopping criterion will also be discussed.

Let us begin by dispelling a common misconception: the least squares polynomial is *not* universally superior to the optimal Chebyshev polynomial. (The optimal Chebyshev polynomial

is the one based on $[\lambda_c, \lambda_d]$. Recall that it yields an optimum preconditioner in the sense of Theorem 3.1.) This follows from a result of Greenbaum [17], who established a partial ordering on preconditioners. In brief, her result implies that the least squares preconditioner cannot be best for every initial guess, x_0. However, it is better in certain cases. For although the optimal Chebyshev polynomial minimizes the condition number of $p_m(A)$, this does not alone determine the rate of convergence of the preconditioned CG method. The eigenvalue distribution of $p_m(A)$ is also important. Because of its equioscillation property, the Chebyshev polynomial tends to map $[c, d]$ uniformly into $[1-\epsilon_m, 1+\epsilon_m]$, obliterating any favorable clustering of the eigenvalues of A. The unweighted least squares polynomial ($\omega \equiv 1$) tends to map the larger eigenvalues of A most closely about 1, giving less weight to the smaller eigenvalues. This is due to the tendency of its relative extrema to decrease in magnitude on S. (This property can be made strict by an appropriate choice of weight function; see § 3.1.) Thus, if there are relatively few eigenvalues of A near c, these will become isolated eigenvalues of the least squares preconditioned matrix, the majority of whose eigenvalues will be clustered about 1. On the other hand, if the eigenvalues of A are dense near c, there will be no such clustering of eigenvalues. The proper choice of polynomial therefore depends on the spectrum of A. We will now explore this question numerically.

In the experiments below, the test matrices are diagonal with $N = 100{,}000$ eigenvalues between $\lambda_c = \delta$ and $\lambda_d = 1+\delta$. The true solution is the vector having 1 in each of its components and $x_0 = 0$. Three eigenvalue distributions are considered. In the first, $\lambda_k = \delta + 1 - 1/k$, $k = 2, \ldots, N-1$, and so the eigenvalues are dense near the right endpoint d. In the second, $\lambda_k = \delta + 1/(N-k+1)$, and so the eigenvalues are dense near the left endpoint c. In the third, the eigenvalues are uniformly distributed. In Figures 4.1–4.4, we plot the PCG relative error, $\log_{10}(\|e_i\|_2/\|e_0\|_2)$, against i for three polynomials of degree $m = 9$. The first is a least squares polynomial (with Legendre weight $\omega \equiv 1$) based on $[0, 1+\delta]$; the second is the optimal Chebyshev polynomial based on $[\delta, 1+\delta]$; and the third is a Chebyshev polynomial based on $[\zeta, 1+\delta]$, where ζ is chosen so the related least squares and Chebyshev residual polynomials have the same first root. By choosing ζ in such a manner, we force this LS-Chebyshev polynomial to mimic the behavior of the least squares polynomial in $(0, \zeta)$, and so the two polynomials behave alike.

In Figure 4.1, we have the dense-right eigenvalue distribution with $\delta = 10^{-3}$. Since the least squares polynomial is small on the large eigenvalues of A, the least squares PCG method converges much more rapidly than the optimal Chebyshev PCG method. In Figure 4.2, the eigenvalues are dense near the left endpoint, and the optimal Chebyshev PCG method converges faster. Similar results were observed for other values of δ. In Figures 4.3–4.4, we have the uniform eigenvalue distribution. When $\delta = 10^{-3}$, the gap between successive eigenvalues is $1/N$, which is smaller than λ_c, and the optimal Chebyshev PCG method converges fastest. However, if $\delta = 10^{-5}$, the gap between successive eigenvalues is larger than λ_c, and the least squares PCG method converges faster. We have seen similar behavior in several other experiments. In short, the optimal Chebyshev polynomial appears to be superior to the least squares polynomial when the gap between successive eigenvalues is small relative to the size of λ_c. The optimal Chebyshev polynomial is also superior to the least squares polynomial when the eigenvalues of A are dense near both endpoints of S or throughout S. In the latter case, for instance, if we

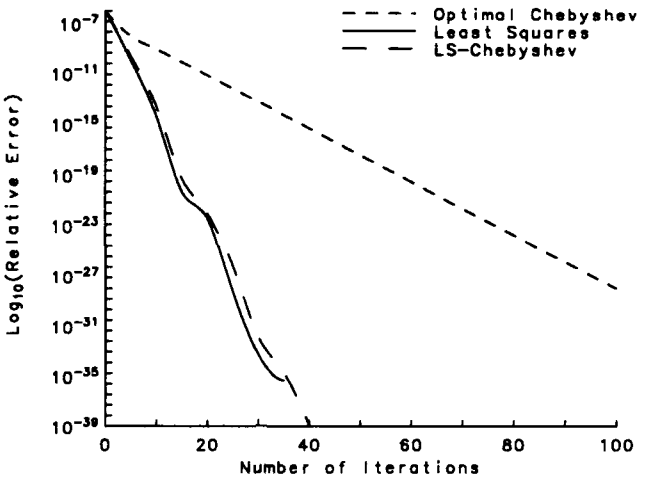


Figure 4.1: Dense-Right Eigenvalue Distribution; $\delta = 10^{-3}$

fix $\delta = 10^{-3}$ and increase N, the optimal Chebyshev polynomial performs best for large N.

As claimed, the LS-Chebyshev polynomial behaves like the least squares polynomial and, depending on the eigenvalue distribution of A, may be superior to the optimal Chebyshev polynomial. For example, if the eigenvalues of A are *sparse near* c and dense near d (recall Figure 4.1), the LS-Chebyshev PCG method will usually converge in fewer iterations than the optimal Chebyshev PCG method. The explanation is similar to that for the superiority of the least squares polynomial: The LS-Chebyshev polynomial maps those eigenvalues in $[\zeta, d]$ more tightly about 1 than does the optimal Chebyshev polynomial. Of course, those eigenvalues in $[c, \zeta)$ are mapped further away from one. However, there are relatively few eigenvalues in $[c, \zeta)$; moreover, they become isolated eigenvalues of $C(A)A$. It is well-known that CG rapidly damps the error in the direction of the corresponding eigenvectors. After doing this, it is able to focus its effort on the dense part of the spectrum where the LS-Chebyshev polynomial does a better job of clustering the eigenvalues about 1. Since one seldom knows how the eigenvalues of A are distributed, one must rely on an adaptive procedure to find the optimum S. If one knew the eigenvalue distribution of A, an appropriately weighted Chebyshev or least squares polynomial could be used to achieve faster convergence. Freund [14] has recently proposed using the Lanczos eigenvalue estimates to obtain such a weight function, and his results are promising. In particular, he has shown that the resulting preconditioned CG method often converges faster than the method based on the optimal Chebyshev polynomial. Unfortunately, there is no guarantee that the preconditioned matrix will be hpd for all S, and this can make difficult or impossible the robust implementation of some adaptive CG algorithms.

Finally, we note that the choice of stopping criterion can also affect the choice of polynomial. Since the least squares polynomial is small on the large eigenvalues of A, it is biased toward

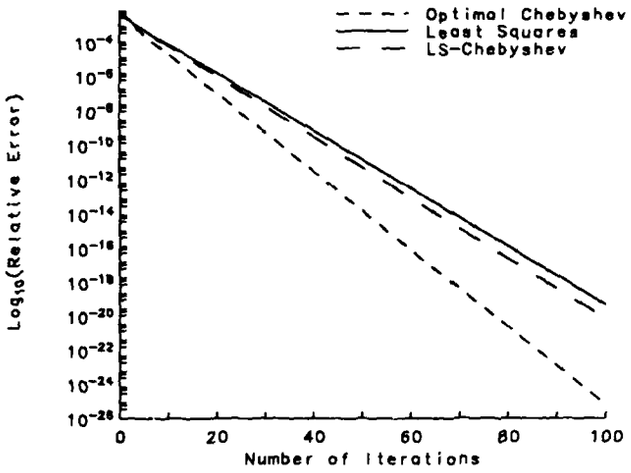


Figure 4.2: Dense-Left Eigenvalue Distribution; $\delta = 10^{-3}$

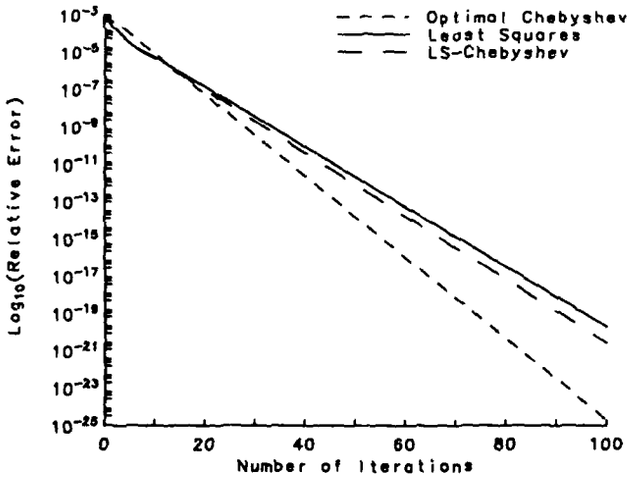


Figure 4.3: Uniform Eigenvalue Distribution; $\delta = 10^{-3}$

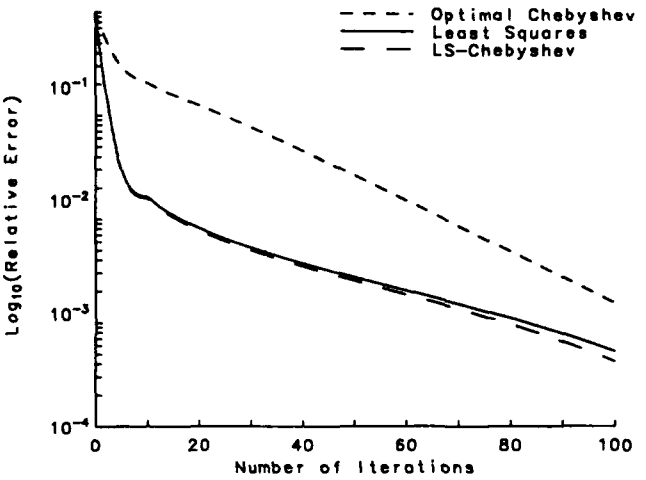


Figure 4.4: Uniform Eigenvalue Distribution; $\delta = 10^{-5}$

this part of the spectrum. Those eigenvectors associated with the large eigenvalues of A are consequently damped the most. If one bases the stopping criterion on the relative residual, the eigenvectors corresponding to these large eigenvalues are given greater weight. Thus, this stopping criterion is ideally suited to the least squares polynomial. The optimal Chebyshev polynomial, on the other hand, is well-suited for use in stopping criteria based on the true error. (Although the true error is unknown, it can be bounded [4].) The difference between these stopping criteria can be as large as $\kappa_B(A)$.

4.1. The Need for an Adaptive Procedure. Recall that the weighted least squares and uniform norms are defined with respect to the positive interval S, which we have assumed contains the spectrum of A. That is, we have assumed that the smallest and largest eigenvalues of A, λ_c and λ_d, are given. Unfortunately, this is seldom true. In the case of the least squares polynomial, one may avoid this difficulty by choosing c and d to be the Gershgorin estimates for λ_c and λ_d. In particular, one may take $c = 0$. The resulting preconditioner is often effective, but there are eigenvalue distributions for which the optimal Chebyshev polynomial is better. Here one needs accurate estimates for the extreme eigenvalues of A. Although this might be viewed as a reason for using the Neumann series or least squares polynomial, it is not. As we will see, one may dynamically estimate λ_c and λ_d from the CG iteration parameters. This is equivalent to dynamically determining the optimum polynomial preconditioner (in the sense of Theorem 3.1). The resulting *adaptive* polynomial preconditioned CG algorithm works remarkably well in practice: it quickly and accurately determines λ_c and λ_d. We describe this idea in the next section.

5. Adaptive CG Algorithms.

In this section we discuss *adaptive* CG algorithms. In such an algorithm we apply a given CG method to the preconditioned linear system $C(A)Ax = C(A)b$, where $C(\lambda)$ is the current preconditioning polynomial. Information about the spectrum of A is extracted from the CG iteration parameters and used to obtain a better preconditioner, $\tilde{C}(A)$. If this new preconditioner is sufficiently better than $C(A)$, the CG method is restarted with $\tilde{C}$; otherwise, the current iteration resumes with C. In this way the adaptive CG algorithm *dynamically* determines the optimum polynomial preconditioner for A.

Determining this optimum preconditioner is equivalent to determining the smallest set $S = [c, d]$ that contains $\sigma(A)$, the spectrum of A. Ideally, $S = \Sigma(A) = [\lambda_c, \lambda_d]$, the convex hull of $\sigma(A)$. This yields the optimum Chebyshev preconditioned polynomial, p_m, which minimizes the bound on $\kappa(p_m(A))$ obtained from (3.12). However, the extreme eigenvalues of A are seldom known a priori, and so S is only an approximation to $\Sigma(A)$. The development of an *adaptive procedure* for dynamically improving this approximation is the subject of this section, which is taken from [3].

Although we will consider only the Chebyshev polynomial, a similar procedure could be used with the least squares polynomial to extract eigenvalue estimates from the CG iteration. However, this is unnecessary since the least squares polynomial is insensitive to its inner endpoint. One may take $c = 0$ and choose d to be the Gershgorin estimate for λ_d. See [27].

5.1. Description of the Adaptive Procedure.

Given an interval S that approximates $\Sigma(A)$, and a Chebyshev preconditioning polynomial $C(\lambda)$ based on S, a CG method is applied to $C(A)Ax = C(A)b$. After a prescribed number of steps, say ℓ, the adaptive procedure is called:

(1) Compute eigenvalue estimates for $p_m(A) = C(A)A$.

(2) Extract eigenvalue estimates for A and update S.

(3) Determine the new preconditioning polynomial, $\tilde{C}(\lambda)$.

(4) Resume or restart the CG iteration, whichever is appropriate.

After another ℓ CG steps, the adaptive procedure is called again, and so on until convergence.

Eigenvalue estimates for $p_m(A)$ are easily obtained from the CG iteration parameters by exploiting the equivalence of the CG and Lanczos algorithms [4, 11, 16]. (See [15] for an alternative.) As we will see, it is then easy to recover eigenvalue estimates for A when the degree m of the polynomial is odd. Once we have these estimates, we can expand S and determine the new Chebyshev preconditioning polynomial. If this polynomial is "much" better than the current polynomial (§ 5.2), the CG iteration is *restarted*. By this we mean that the current iteration is abandoned and the CG method is applied to $\tilde{C}(A)Ax = \tilde{C}(A)b$. The new initial guess is the last iterate of the previous iteration or some linear combination of past iterates.

To elucidate, suppose we are executing a CG iteration. Let S be the current approximation to $\Sigma(A)$ and let

$$p_m(\lambda) = C(\lambda)\lambda = 1 - \frac{T_m(\frac{d+c-2\lambda}{d-c})}{T_m(\frac{d+c}{d-c})} \tag{5.1}$$

be the current Chebyshev preconditioned polynomial. Note that the image of S under p_m is $J_\epsilon = [1-\epsilon, 1+\epsilon]$, where $\epsilon = T_m^{-1}(\frac{d+c}{d-c})$; recall § 3.2. Next assume the adaptive procedure has

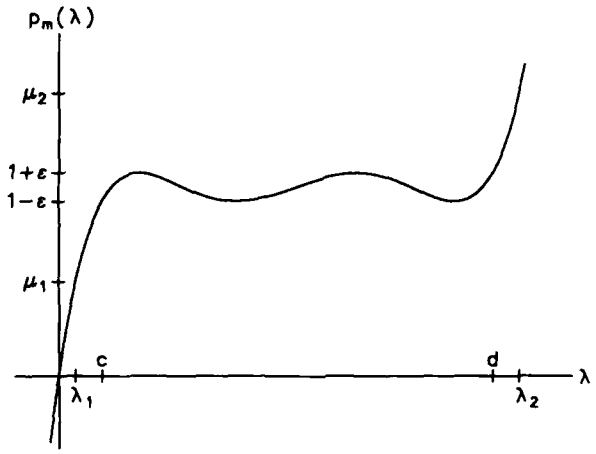


Figure 5.1: Chebyshev preconditioned polynomial ($m = 5$) for $S = [1, 10]$

been called, and let μ be an eigenvalue estimate for $p_m(A)$ such that $\mu \in \Sigma(p_m(A))$, the convex hull of $\sigma(p_m(A))$. (This is true of the estimates we will obtain.) The desired eigenvalue estimate for A is one of the inverse images of μ; the task is to determine which one. It is important that the inverse image chosen lie in $\Sigma(A)$. Otherwise S is improperly and irrevocably expanded, which slows the convergence of subsequent CG iterations.

Suppose first that $\mu \in J_\epsilon$. Then there exists an inverse image of μ inside S. Since there is no justification for expanding S, this estimate may be discarded. If every eigenvalue estimate for $p_m(A)$ is in J_ϵ, there is no need to update S, and the CG iteration resumes. The adaptive procedure has yielded no new information.

Now suppose $\mu \notin J_\epsilon$. Since $\mu \in \Sigma(p_m(A))$, μ must have an inverse image in $\Sigma(A)\backslash S$, which means there is an eigenvalue of A outside S. If an estimate, λ, of this eigenvalue can be recovered, S can be expanded, and a new, better preconditioner computed. When m is odd, it easy to extract λ from μ, as may be seen in Figure 5.1. For example, let $\mu = \mu_1 < 1 - \epsilon$. Since $p_m(\lambda)$ is monotonically increasing for $\lambda \in (0, c)$, there is a unique $\lambda_1 \in (0, c)$ such that $\mu_1 = p_m(\lambda_1)$. Moreover, since $\mu_1 \in H(p_m(A))$, λ_1 must lie in $\Sigma(A)$. Therefore, c should be decreased to λ_1. Similarly, if $\mu = \mu_2 > 1 + \epsilon$, d should be increased to λ_2, the unique inverse image of μ_2. Note that estimates for only the smallest and largest eigenvalues of $p_m(A)$ are needed, for these yield estimates for the extreme eigenvalues of A.

To compute the inverse images of μ_1 and μ_2, a rootfinder could be used, but this is unnecessary because $p_m(\lambda)$ is known explicitly. For m odd and $d \neq c$, one may show

$$\lambda_1 = \frac{1}{2}\left((d+c) - (d-c)\cosh\frac{1}{m}\cosh^{-1}\left(\frac{1-\mu_1}{\epsilon}\right)\right) \tag{5.2}$$

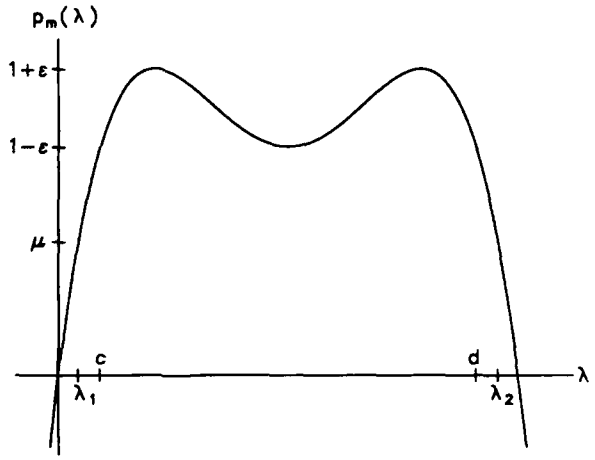


Figure 5.2: Chebyshev preconditioned polynomial ($m = 4$) for $S = [1, 10]$

and

$$\lambda_2 = \frac{1}{2}\left((d+c) + (d-c)\cosh\frac{1}{m}\cosh^{-1}\left(\frac{\mu_2 - 1}{\epsilon}\right)\right). \tag{5.3}$$

If $d = c$ (a common choice for the initial S), then

$$\lambda_1 = d\left(1 - (1-\mu_1)^{1/m}\right) \quad \text{and} \quad \lambda_2 = d\left(1 + (\mu_2 - 1)^{1/m}\right). \tag{5.4}$$

So far we have assumed that m is odd, which is important for two reasons. To see why, consider Figure 5.2, in which m is even. As before, any eigenvalue estimates for $p_m(A)$ in J_ϵ are discarded. Since both tails of $p_m(\lambda)$ are negative, there can be no estimate $\mu > 1 + \epsilon$, so suppose $\mu < 1 - \epsilon$. There are now two inverse images, λ_1 and λ_2, at least one of which lies in $\Sigma(A)$. The question is this: which one is it? If the wrong one is chosen, the set S may be incorrectly enlarged, and the CG method will converge more slowly than necessary. To avoid this *ambiguity*, we shall always choose m odd.

Another advantage of choosing m odd is that it yields *robust* CG methods. By this we mean that $p_m(A)$ is hpd for any hpd A and for any set S. If this were not true, the CG method might not be defined in the early iterations. For example, if m were even and there were an eigenvalue of A greater than the largest root of $C(\lambda)\lambda$, $p_m(A)$ would be indefinite, in which case GCGHS and PCR are inappropriate, as are the Omin implementations of PCG, PPCR and GCR. One would have to use the Odir or Odir/Omin implementation of PCG, PPCR, or GCR. When m is odd, on the other hand, $p_m(A)$ is hpd for any set S and the Omin implementation of each method in Table 2.1 is applicable.

5.2. Resume versus Restart. After eigenvalue estimates for A are computed, the set S is expanded, and a new preconditioning polynomial, $\tilde{C}(\lambda)$, is determined. The adaptive

procedure must now decide whether the CG iteration should be resumed using the current preconditioner or restarted using the new preconditioner. This choice is discussed next.

Let $p_m(\lambda)$ be the preconditioned polynomial for $S = [c,d]$, the current approximation to $\Sigma(A)$. Also let $\mu_1 < 1-\epsilon$ and $\mu_2 > 1+\epsilon$ be the new eigenvalue estimates for $p_m(A)$, and let λ_1 and λ_2 be their inverse images. If the iteration is resumed using the the current preconditioner, the revised convergence factor is

$$cf_{rev} = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \tag{5.5}$$

where $\kappa = \mu_2/\mu_1$ is the revised estimate for $\kappa(p_m(A))$.

Next let $\tilde{p}_m(\lambda)$ be the preconditioned polynomial for $\tilde{S} = [\lambda_1, \lambda_2]$, the new approximation to $\Sigma(A)$. If the iteration is restarted using the new preconditioner, the convergence factor is

$$cf_{new} = cf(\tilde{p}_m(A)) = \frac{\sqrt{\tilde{\kappa}}-1}{\sqrt{\tilde{\kappa}}+1} \tag{5.6}$$

where

$$\tilde{\kappa} = \kappa(\tilde{p}_m(A)) \approx \frac{1+\tilde{\epsilon}}{1-\tilde{\epsilon}}, \qquad \tilde{\epsilon} = T_m^{-1}\left(\frac{\lambda_2+\lambda_1}{\lambda_2-\lambda_1}\right) \tag{5.7}$$

is the condition number estimate for $\tilde{p}_m(A)$.

Although $cf_{new} < cf_{rev}$, the difference may be too small to warrant restarting the CG iteration. Using equation (3.10), it is possible to predict the number of CG steps the current iteration will need to converge to within some tolerance. It is also possible to predict how many steps the restarted iteration will need. If these two numbers differ by one, for example, the iteration should be resumed. Otherwise, the CG iteration should be restarted.

Since S is either expanded or unchanged with each call to the adaptive procedure, it is important that the initial set, S_0, be such that $S_0 \subset \Sigma(A)$. If the matrix A is scaled to have unit diagonal, one may take $S_0 = [1,1]$. A more general choice is $S_0 = [\tau,\tau]$, where $\tau = trace(A)/N$ and N is the order of A. One might also consider initially using the LS-Chebyshev polynomial.

It is important to note that $S = \Sigma(A)$ need *not* give the optimum rate of convergence. In practice, convergence may be more rapid when S is a proper subset of $\Sigma(A)$. To understand why, recall that the true rate of convergence of a CG method depends on the distribution of the eigenvalues of A within $\Sigma(A)$. For example, suppose $\lambda_1 \ll \lambda_2$. Then $S = [\lambda_2, \lambda_N]$ is a better choice than $[\lambda_1, \lambda_N]$. The reason is that CG methods pick out isolated eigenvalues and rapidly suppress the error in the direction of corresponding eigenvectors. Unfortunately, one seldom knows such detailed information about $\sigma(A)$. Although the recent idea of Freund [14] to use the Lanczos eigenvalue estimates to approximate the eigenvalue distribution of A is appealing, it is unclear whether this can be done dynamically.

5.3. Performance. Having introduced the theory behind the adaptive procedure, we now consider its performance in practice. As we will see, the adaptive procedure works remarkably well in that it quickly and accurately determines λ_c and λ_d. Since this section is concerned only with the performance of the adaptive *procedure*, the test problems are small. Results for much larger problems are given in § 6, where we examine the performance of adaptive polynomial preconditioned CG algorithms.

$N = 2500$			$m = 7$
k	λ_c	λ_d	action
0	0.10000e+01	0.10000e+01	initial
5	0.24762e–01	0.19870e+01	restart
10	**0.27832e–02**	**0.19962e+01**	restart
15	0.27577e–02	0.19972e+01	resume
20	0.19262e–02	0.19981e+01	resume
25	0.18981e–02	0.19981e+01	resume
30	0.18968e–02	0.19981e+01	resume
true	0.18967e–02	0.19981e+01	

Table 5.1: PCG Adaptive Procedure for 5-Point Laplacian

The tables below summarize the behavior of the adaptive procedure for two simple test problems. The matrices have order 2500 and result from a 5-point and 9-point finite difference approximation to the 2-dimensional Laplacian. Although PCG results are given only for a polynomial of degree 7, these results are typical. In each table we list the estimates for λ_c and λ_d computed by the adaptive procedure, which is called every five steps. The adaptive algorithm is initially given $c = d = 1$. In the last column we report the action taken by the adaptive procedure.

Consider Table 5.1. After five steps, the adaptive procedure found new estimates for λ_c and λ_d and decided to restart the iteration using a new preconditioning polynomial based on these estimates. After another five steps, the adaptive procedure refined its estimates for λ_c and λ_d and again restarted. From here on it continues to improve its estimates for λ_c and λ_d, but opts to resume the iteration using the polynomial determined at step 10. Similar behavior is seen in Table 5.2. We remark that λ_c and λ_d are found more quickly with higher degree polynomials.

The performance described here is typical. Although the estimates for λ_c and λ_d eventually converge to their true values, the adaptive procedure often finds satisfactory estimates early on in the iteration. In other words, the adaptive procedure is able to find a nearly optimum polynomial preconditioner within a few calls. This means that there is little overhead associated with the adaptive procedure. Finally, we remark that the resume versus restart decision is an important one: it can make a dramatic difference in the number of steps required for convergence to the solution of the linear system.

$N = 2500$			$m = 7$
k	λ_c	λ_d	action
0	0.10000e+01	0.10000e+01	initial
5	0.17330e–01	0.14347e+01	restart
10	0.15041e–01	0.15959e+01	restart
15	**0.22899e–02**	**0.15968e+01**	restart
20	0.22899e–02	0.15990e+01	resume
25	0.22899e–02	0.15992e+01	resume
true	0.22753e–02	0.15992e+01	

Table 5.2: PCG Adaptive Procedure for 9-Point Laplacian

6. Numerical Experiments.

In this section we demonstrate the effectiveness of adaptive polynomial preconditioning on a Cray X-MP/48 and Alliant FX/8 for some large matrices arising in hydrology. In particular, we show that polynomial preconditioned PCG (PPCG) can converge in less CPU time than the unpreconditioned CGHS method. Although we do not compare it with other preconditionings, we emphasize that polynomial preconditioning can be used to further accelerate any other preconditioning, for example, an incomplete factorization.

6.1. Description of Experiments.

Our test matrices, which arise in the modeling of groundwater flow in a heterogeneous aquifer, result from the 7-point finite difference approximation to a three-dimensional elliptic PDE with variable coefficients. Although several parameters determine the difficulty of the problem, we isolate just two. In the first set of experiments, run on a Cray X-MP/48, the hydraulic conductivity field K is uncorrelated, which makes the problem difficult. In the second set of experiments, run on an Alliant FX/8, the field is correlated. For each machine we vary γ, the standard deviation of the $\ln K$ field. As γ increases, so does $\kappa(A)$, the condition number of A. See [24] for details.

In the tables below, m is the degree of the preconditioned polynomial, $p_m(\lambda)$. The first row of each table, $m = 1$, corresponds to the unpreconditioned CGHS method. We next give the number of CPU seconds required for convergence of the CG iteration, which includes the adaptive procedure. The iteration was halted once the relative error was brought below 10^{-8} on the Cray and 10^{-6} on the Alliant. In the last column of each table we list the ratio of CGHS time to PPCG time. If this ratio is greater than 1, we say that polynomial preconditioning is *effective*. In all the experiments, the right-hand-side vector b was chosen so that the true solution vector has 1 in each component, and the initial guess was the zero vector. Since the matrix was symmetrically scaled to have unit diagonal, we set $c_0 = d_0 = 1$ in the adaptive procedure. New eigenvalue estimates were computed every ten steps with a maximum of ten calls to the adaptive procedure. Finally, we note that the results below were taken from [3] and [24].

6.2. Discussion of Results.

In Tables 6.1–6.3 we report results for a single vector processor of a Cray X-MP/48. In the first table, the condition number of A is about 60,000, as estimated by the adaptive procedure. Here we obtain a 15% improvement over CGHS with a polynomial of degree 5. In the next two tables, $\kappa(A)$ is 160,000 and 360,000, respectively, corresponding to $\gamma = 1.5$ and $\gamma = 1.75$. Notice that polynomial preconditioning is more effective here: it reduces the CPU time required to solve the problem by about 25%. Also observe that the optimum m is increasing with $\kappa(A)$.

In Tables 6.4–6.6 we see similar qualitative results for the Alliant FX/8, which is an 8-vector-processor machine. Although the problems are much larger, they are not nearly as ill-conditioned. (We estimate the condition numbers to be 8,400, 14,000, and 26,000.) We once again see the best performance on the hardest problem. Moreover, notice the much larger CGHS/PPCG ratios: the time required to solve the problem has been nearly cut in half. The computer architecture does indeed make a difference.

$N = 103,823$ $\gamma = 1.0$

m	Iterations	Seconds	CGHS/PPCG
1	1112	18.00	1.00
3	386	15.77	1.14
5	242	15.65	1.15
7	229	20.59	0.87
9	215	24.82	0.73
11	152	21.35	0.84

Table 6.1: Cray X-MP/48 CPU Times

$N = 103,823$ $\gamma = 1.5$

m	Iterations	Seconds	CGHS/PPCG
1	2315	37.04	1.00
3	780	31.81	1.16
5	473	30.56	1.21
7	341	30.22	1.23
9	268	30.16	1.23
11	227	31.28	1.18
13	213	34.84	1.06

Table 6.2: Cray X-MP/48 CPU Times

6.3. Conclusions. In this section we have the demonstrated the effectiveness of polynomial preconditioning on a Cray X-MP/48 and an Alliant FX/8. We have seen that polynomial preconditioning is most effective when the matrix A is ill-conditioned. Moreover, as $\kappa(A)$ increases, so does the optimum degree m. In general, however, low degree (2–16) preconditioning polynomials are usually best. In contrast, high degree (20–50) polynomia's are usually best for hermitian indefinite matrices [3, 5]. Although we have presented results for only the hydrology problem, our conclusions are supported by a variety of other numerical experiments, including those in [3, 7, 12, 22, 24].

We emphasize that our adaptive CG algorithms are as easy to use as CGHS, yet can reduce

$N = 103,823$ $\gamma = 1.75$

m	Iterations	Seconds	CGHS/PPCG
1	4126	66.76	1.00
3	1383	57.30	1.17
5	833	54.67	1.22
7	600	53.72	1.24
9	469	53.82	1.24
11	386	53.60	1.25
13	328	53.55	1.25
15	287	53.55	1.25
17	255	53.36	1.25
19	235	55.10	1.21

Table 6.3: Cray X-MP/48 CPU Times

$N = 410,625$ $\gamma = 1.0$

m	Iterations	Seconds	CGHS/PPCG
1	468	887.61	1.00
3	183	673.77	1.32
5	103	541.13	1.64
7	82	564.95	1.57
9	70	598.83	1.48
11	57	586.48	1.51

Table 6.4: Alliant FX/8 CPU Times

$N = 410,625$ $\gamma = 1.7$

m	Iterations	Seconds	CGHS/PPCG
1	607	1141.40	1.00
3	249	910.98	1.25
5	155	801.22	1.42
7	95	664.83	1.76
9	79	684.41	1.70
11	69	714.39	1.62

Table 6.5: Alliant FX/8 CPU Times

the CPU time required to solve the linear system. The amount of reduction depends on the computer architecture. We note that Holst [20] has obtained results similar to those for the Alliant on a Cray 2. In particular, he has reported CGHS/PPCG ratios of nearly 2 to 1, which is far better than those achieved on the X-MP. Chan et al. [7] have shown that polynomial preconditioning is competitive with other preconditioners on the massively parallel CM-2.

7. Summary. In this paper we have explored the use of adaptive polynomial preconditioning for hermitian positive definite linear systems. Such preconditioners are easy to employ and well-suited to vector and/or parallel computer architectures. After reviewing preconditioned CG methods, we showed how one could use a polynomial preconditioner in a variety of different ways. We then discussed the least squares and Chebyshev preconditioning polynomials, studied them in the context of CG methods, and showed that the latter minimizes a bound on the condition number of the preconditioned matrix. We next compared the two

$N = 410,625$ $\gamma = 2.3$

m	Iterations	Seconds	CGHS/PPCG
1	839	1584.50	1.00
3	308	1108.60	1.43
5	205	1057.20	1.50
7	134	913.34	1.73
9	103	869.94	1.82
11	88	889.15	1.78

Table 6.6: Alliant FX/8 CPU Times

polynomials in a variety of numerical experiments. In particular, we sought to determine those eigenvalue distributions for which each is well-suited. The least squares polynomial is superior for those matrices whose eigenvalues are dense near the largest eigenvalue, λ_d. In contrast, the Chebyshev preconditioner is superior when the eigenvalues are dense throughout the interval or when the gap between successive eigenvalues is smaller than the smallest eigenvalue, λ_c. We next described an *adaptive procedure* for dynamically computing λ_c and λ_d, which are needed to determine the optimal Chebyshev polynomial preconditioner. The accuracy and efficiency of this adaptive procedure was also demonstrated. Finally, in the previous section, we presented some numerical results that demonstrate the effectiveness of adaptive polynomial preconditioning for some large matrices arising in hydrology. Our results suggest that relatively low degree (2–16) polynomials are usually best. Moreover, the optimum degree m of the polynomial tends to increase with the condition number of A, as does the effectiveness of polynomial preconditioning.

Acknowledgements. We wish to thank Professor Paul Saylor for his many helpful comments and thoughtful questions. We also thank Philip Meyer for providing the hydrology test problems and associated software. The first author wishes to thank Professor Roland Glowinski for inviting him to present these results at the Ninth International Conference on Computing Methods in Applied Sciences and Engineering.

References

[1] L. M. Adams. *Iterative Algorithms for Large, Sparse Linear Systems on Parallel Computers.* PhD thesis, Univ. of Virginia, Applied Mathematics, 1982.

[2] L. M. Adams. *M-step Preconditioned Conjugate Gradient Methods.* **SIAM J. Sci. Stat. Comput.**, 6(2):452–463, 1985.

[3] S. F. Ashby. *Polynomial Preconditioning for Conjugate Gradient Methods.* PhD thesis, Univ. of Illinois at Urbana-Champaign, Dept. of Computer Science, December 1987. Available as Technical Report 1355.

[4] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor. *A Taxonomy for Conjugate Gradient Methods.* Technical Report UCRL-98508, Lawrence Livermore National Laboratory, 1988.

[5] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor. *Adaptive Polynomial Preconditioning for Hermitian Indefinite Linear Systems.* Technical Report UCRL-100970, Lawrence Livermore National Laboratory, 1989.

[6] P. N. Brown and A. C. Hindmarsh. *Matrix-free Methods for Stiff Systems of ODE's.* **SIAM J. Numer. Anal.**, 23(3):610–638, 1986.

[7] T. F. Chan, C. J. Kuo, and C. Tong. *Parallel Elliptic Preconditioners: Fourier Analysis and Performance on the Connection Machine.* Technical Report CAM 88-22, Univ. of California, Los Angeles, Dept. of Mathematics, 1988.

[8] R. Chandra, S. C. Eisenstat, and M. H. Schultz. *The Modified Conjugate Residual Method for Partial Differential Equations.* In R. Vichnevetsky, editor, **Advances in Computer Methods for Partial Differential Equations II**, pages 13–19, 1977.

[9] A. Chronopoulos. *A Class of Parallel Iterative Methods Implemented on Multiprocessors.* PhD thesis, Univ. of Illinois at Urbana-Champaign, Dept. of Computer Science, November 1986. Available as Technical Report 1267.

[10] A. T. Chronopoulos and C. W. Gear. *Implementation of s-Step Methods on Parallel Vector Architectures.* Technical Report 1346, Univ. of Illinois at Urbana-Champaign, Dept. of Computer Science, June 1987.

[11] 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**, pages 309–332. Academic Press, New York, 1976.

[12] P. F. Dubois, A. Greenbaum, and G. H. Rodrigue. *Approximating the Inverse of a Matrix for Use on Iterative Algorithms on Vector Processors.* **Computing**, 22:257–268, 1979.

[13] V. Faber and T. A. Manteuffel. *Necessary and Sufficient Conditions for the Existence of a Conjugate Gradient Method.* **SIAM J. Numer. Anal.**, 21(2):352–362, 1984.

[14] R. Freund. *Polynomial Preconditioners for Hermitian and Certain Nonhermitian Matrices.* SIAM Annual Meeting, San Diego, CA, 1989.

[15] G. H. Golub and M. D. Kent. *Estimates of Eigenvalues for Iterative Methods.* **Math. Comp.**, 53(188), 1989.

[16] G. H. Golub and C. F. Van Loan. **Matrix Computations**. The Johns Hopkins University Press, Baltimore, MD, 1983.

[17] A. Greenbaum. *Comparison of Splittings Used with the Conjugate Gradient Algorithm.* **Numer. Math.**, 33:181–194, 1979.

[18] L. A. Hageman and D. M. Young. **Applied Iterative Methods**. Academic Press, New York, 1981.

[19] M. R. Hestenes and E. Stiefel. *Methods of Conjugate Gradients for Solving Linear Systems.* **J. Research of NBS**, 49:409–435, 1952.

[20] M. J. Holst. *Private communication*, 1989.

[21] O. G. Johnson, C. A. Micchelli, and G. Paul. *Polynomial Preconditioning for Conjugate Gradient Calculations.* **SIAM J. Numer. Anal.**, 20(2):362–376, 1983.

[22] T. L. Jordan. *Conjugate Gradient Preconditioners for Vector and Parallel Processors.* In G. Birkhoff and A. Schoenstadt, editors, **Proceedings of the Conference on Elliptic Problem Solvers**, New York, 1984. Academic Press.

[23] 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(137):148–162, 1977.

[24] P. D. Meyer, A. J. Valocchi, S. F. Ashby, and P. E. Saylor. *A Numerical Investigation of the Conjugate Gradient Method as Applied to Three-Dimensional Groundwater Flow Problems in Randomly Heterogeneous Porous Media.* **Water Resources Research**, 25(6):1440–1446, 1989.

[25] T. J. Rivlin. **The Chebyshev Polynomials**. John Wiley and Sons, New York, 1974.

[26] H. Rutishauser. *Theory of Gradient Methods.* In **Mitteilungen aus dem Institut fur Angewandte Mathematik Nr. 8**, pages 24–49, 1959.

[27] Y. Saad. *Practical Use of Polynomial Preconditionings for the Conjugate Gradient Method.* **SIAM J. Sci. Stat. Comput.**, 6(4):865–881, 1985.

[28] P. E. Saylor. *Leapfrog Variants of Iterative Methods for Linear Algebraic Equations.* **J. Comp. & Applied Math.**, 24:169–193, 1988.

[29] G. Szego. **Orthogonal Polynomials**. Colloquium Publications 23, Revised Edition, American Mathematical Society, Providence, RI, 1959.

[30] D. M. Young and K. C. Jea. *Generalized Conjugate Gradient Acceleration of Nonsymmetrizable Iterative Methods.* **Linear Algebra Appl.**, 34:159–194, 1980.