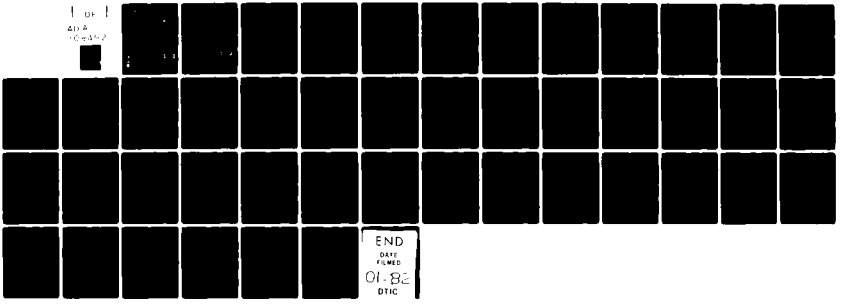
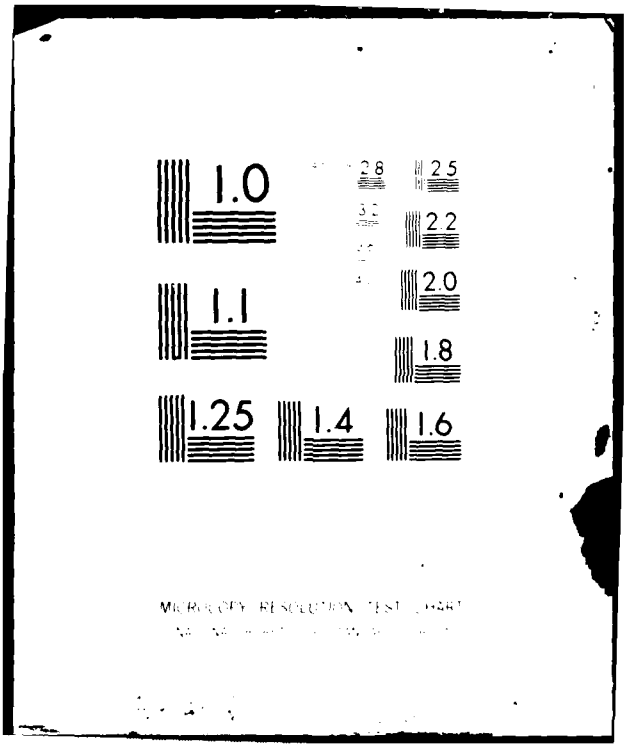
AD-A108 492 MARYLAND UNIV COLLEGE PARK INST FOR PHYSICAL SCIENCE--ETC F/G 12/1
A QUANTITATIVE EVALUATION OF THE FEASIBILITY OF, AND SUITABLE H--ETC(U)
SEP 81 P ZAVE, G E COLE N00014-77-C-0623
UNCLASSIFIED BN-971 NL




1.0
1.1
1.25
1.4
1.6
2.0
2.2
2.5

INSTITUTE FOR PHYSICAL SCIENCE
AND TECHNOLOGY

12

LEVEL II

Technical Note BN-971

A QUANTITATIVE EVALUATION OF THE FEASIBILITY OF, AND SUITABLE HARDWARE ARCHITECTURES FOR, AN ADAPTIVE, PARALLEL FINITE-ELEMENT SYSTEM

by

Pamela Zave

and

George E. Cole, Jr.

September 1981

UNIVERSITY OF MARYLAND

A QUANTITATIVE EVALUATION OF THE FEASIBILITY OF, AND SUITABLE HARDWARE ARCHITECTURES FOR, AN ADAPTIVE, PARALLEL FINITE-ELEMENT SYSTEM

Pamela Zave and George E. Cole, Jr.

This research was supported in part by the Office of Naval Research, Mathematics Branch, under Contract N00014-77-C-0623 at the University of Maryland.

Authors' present addresses: P. Zave, Bell Laboratories, Room 7C-207, Murray Hill, NJ 07974; G.E. Cole, Johns Hopkins Hospital, Laboratory Information Systems, Baltimore, MD 21205.

Abstract This paper describes an experimental implementation of a design for an adaptive, parallel finite-element system. The implementation was used to simulate the performance of this design on several microprocessor-based multiprocessor architectures. We conclude that it will be possible to build hardware/software finite-element systems which exploit data segmentation to achieve flexibility, modularity, and the ability to process very large problems.

A QUANTITATIVE EVALUATION OF THE FEASIBILITY OF, AND SUITABLE HARDWARE ARCHITECTURES FOR, AN ADAPTIVE, PARALLEL FINITE-ELEMENT SYSTEM

1. INTRODUCTION

1.1. Toward a New Generation of Finite-Element Systems

The finite-element method is an important and widely applicable engineering tool, but current finite-element systems (as characterized by NASTRAN) present certain practical difficulties to the user. They have relatively inflexible structures, and demand numerous critical and difficult decisions--which must sometimes be made in an arbitrary fashion. Computations are expensive and time-consuming.

The goal of the FEARS (Finite-Element, Adaptive Research Solver) project is to lay the foundations for a new generation of finite-element systems which are both superior engineering tools, and capable of solving problems of dramatically increased size. This project has been underway at the University of Maryland for several years.

Our approach to improving the method as an engineering tool entails: (a) interchangeable user-oriented interfaces based on a generalized mathematical formulation of the problem, (b) reliable *a posteriori* error estimation in various norms, (c) adaptive features to help the user achieve acceptable accuracy at

acceptable cost, and (d) support of team cooperation. These points are elaborated below.

(a) The mathematical formulation presented in [Zave & Rheinboldt 79] is based on the concept of substructure analysis used widely in engineering (the domain is defined as a composition of subdomains of a special type, such as curvilinear rectangles). The formulation is application-independent, and constitutes a general "finite-element solver" for a class of boundary value problems for partial differential equations defined by a weak mathematical formulation. In other words, the problem is given in terms of an appropriate bilinear form on certain Hilbert spaces. A library of such forms could potentially be accessed by commands in various application-dependent interface languages. This would minimize input data, and also make the latest progress in finite elements available to users who are not specialists in this area.

(b) The system provides inexpensive, but reliable, estimation of the error in a desired norm. The error estimation is computed as a sum of error indicators (dependent on the desired norm), which are associated with single elements and computable locally from the finite-element solutions only.

(c) The adaptive features relate especially to selection of the mesh (in the so-called "h-version" of finite elements), or in general to the degree of the elements and the mesh when the "p" or combined "p-h version" is used. Here we will concentrate on the h-version used in FEARS (for more on the p-version see [Babuska et al. 81], [Babuska & Szabo 81], [Babuska & Dorr 81]).

It can be shown that optimal meshes equilibrate the indicators (make them approximately equal). This leads to the principle for adaptive generation of the mesh, namely equilibration of the error indicators.

The adaptive features include prediction of the computational costs, from any given mesh and solution, of achieving further accuracy. This capability is used as the basis for a command language which allows the user to specify the meaning of the error, acceptable levels of the given error, and acceptable computational costs. The system obeys these commands automatically, generating low-level control decisions which meet the goals and constraints if possible.

(d) In many engineering situations several people want to work cooperatively on the same project, sharing relevant data but not interfering unduly with each other's efforts. Modern perspectives on multiprogramming make such controlled interaction possible.

The above principles (a), (b), and (c) are explained further in [Babuska & Miller 81], [Babuska & Rheinboldt 78a], [Babuska & Rheinboldt 78b], [Babuska & Rheinboldt 78c], [Babuska & Rheinboldt 79a], [Babuska & Rheinboldt 79b], [Babuska & Rheinboldt 79c], [Babuska & Rheinboldt 80], [Rheinboldt 81], and [Rheinboldt & Mesztenyi 80]. They have been developed experimentally in the implementation of the FMP (FEARS Mathematical Program) system, which runs on Univac and CDC machines, and will be generally available at the end of 1981.

1.2. An Experiment in Parallelism and Modularity

Our approach to increasing the size of systems that can be solved entails a system architecture based on parallel processing and data segmentation, so that technical and economic advances in computer hardware can be exploited. The FEARS design presented in [Zave & Rheinboldt 79] accomodates the engineering features described above, and has such a modular, parallel structure. This structure can be used to exploit the power of new and existing "distributed" hardware architectures; a monolithic system, on the other hand, cannot reasonably be implemented on anything but a centralized computer facility.

This paper presents the results of experimentation with PF (Parallel FEARS), an implementation of the FEARS design specialized for studying the effects of parallelism and modularity on system performance. Section 3 describes how we implemented the design so as to get data on how it would perform if it were running on several particular hardware architectures, why we chose those architectures, and what experimental cases were run. Section 4 summarizes the results relating to data segmentation, real-time speedup, and hardware architectures.

2. THE FEARS DESIGN

2.1. Processes

The FEARS design is specified in terms of "processes". A process represents independent computation on local data. The

various processes of a system are encapsulated, and thus communicate only by sending messages. Since processes are understood to perform their computations asynchronously in parallel with one another, organizing a system into processes provides explicit parallelism. The "process" abstraction is well-known and heavily used in the literature of computer systems and parallel processing.

Figure 1 sketches the process and communication structure of FEARS. Subsequent sections will explain the data contained in each of these processes, the computations performed within them, and their characteristic interactions.

Process-level parallelism is completely independent of instruction-level parallelism, which is used by vector and pipeline processors to speed up the execution of numerical algorithms. The FEARS design neither includes nor precludes the use of instruction-level parallelism in the execution of individual process computations. There is a certain divergence of approaches on a practical basis, however, because the computer architectures which are best suited to process-level parallelism (see 3.2) feature large numbers of processors. Vector/pipeline processors tend to be too expensive for such casual replication.

Our reasons for designing FEARS as a set of interacting processes were twofold. One was the hope that process-level parallelism would provide a more powerful, cost-effective source of real-time speedup in finite-element computations than instruction-level parallelism. (This hope is in part justified by the speedup observed in the non-automated use of substructure

analysis in many engineering projects.) The other was the notion that a finite-element system organized into encapsulated processes would have a distributed, modular character that would be of potential use in many computing environments.

Consider, for instance, the (traditional) environment of the large, centralized, multiprogrammed computer. The biggest problem encountered by the implementor of a finite-element system in such an environment will be storage management: since the necessary data cannot possibly all fit in the portion of main memory allocated to one user, carefully planned overlays must swap the appropriate portion in for each phase of the computation. The rest of the data will be disposed among a set of mass-storage files, access to which must be minimized.

The process organization of FEARS provides exactly the kind of structure that is necessary to do this. It identifies local computations with the data they need, in such a way that references to non-local data are minimized. Many operating systems make explicit processes available to users. These operating systems would perform storage management for FEARS automatically--because it is their normal mode of operation to swap in the data of a process when it is about to be scheduled to run.

2.2. Subdomains

FEARS solves a system of two-elliptic partial differential equations in two dimensions. The domain is defined as a union of two- , one- , and zero-dimensional subdomains. Each domain has

some homogeneity characteristic with respect to the mathematical formulation; for example, either the form of the differential equation, the boundary condition, or the error norm is constant on the entire subdomain.

The user of FEARS describes the domain of his particular boundary-value problem in terms of two-dimensional four-sided figures (Figure 2(a)), each of which is automatically mapped (one-to-one and smoothly) onto the unit square (Figure 2(b)). Adaptive refinement proceeds independently within each unit square (Figure 2(c)).

A process is assigned to each two- , one- , and zero-dimensional open subdomain in the user's domain description. Figure 2(d) represents the open subdomains immediately adjacent to a particular two-dimensional subdomain (2DS), in terms of the processes which would be assigned to them. A subdomain's process contains all the system's data relevant to that subdomain, namely the bilinear form, mappings onto the curvilinear domain, adjacency relations to other subdomains, boundary conditions, loads and load points, mesh definition, and current solutions. The process is responsible for all transformations on these data, and for answering the questions of other processes about the data (since they cannot access them directly). Open subdomains must be used, of course, to avoid the expense of redundant data and the problems of keeping multiple copies consistent with one another. Thus in Figure 2(d), the one- and zero-dimensional subdomains which are interior to the original problem domain are also adjacent to (part of the border of) some other two-

dimensional subdomain.

During refinement the operations of deciding which elements to refine are completely internal to each 2DS process. If an element on the border is refined, however, this has the effect of creating a new mesh point in the adjacent one-dimensional subdomain. The 2DS process sends a message to the 1DS process requesting it to form the additional point.

Most other phases of the computation have the same character: 2DS processes do almost everything locally, needing only to communicate with the 1DS and ØDS processes on their boundary in situations involving the closed subdomain. A 1DS process acts as the interface between the 2DS processes on either side of it, answering questions and honoring update requests from both. These local operations are finally connected, of course, in the global solution of the macro-stiffness equations. Elimination goes on at the level of the two-dimensional subdomains separately, until finally the only unknowns left are located in one- and zero-dimensional subdomains. These form a fairly dense submatrix (in the case of only one two-dimensional subdomain, this submatrix is full), which is solved in its own dedicated process.

Since most of the parallelism in FEARS comes from the subdomain processes, the user sets the degree of parallelism in his initial domain description. He is also responsible for making a reasonably intelligent division of the domain, one which balances the amount of computation to be done by each process. This seems to us a legitimate way to use the engineer's skills

and afford him some control, without requiring too many hours of drudgery or decisions that can adversely affect the validity of the numerical results.

2.3. Projects and Problems

The FEARS design is intended to support the situation in which a number of people are working on a common project. This is done by separating the data which are common to all work on the project from those data belonging to individual efforts. The "project" information consists of the subdomain adjacency structure, the bilinear forms, and the mappings onto the curvilinear domain space. The "problem" information consists of the loads and load points, boundary conditions, mesh geometry, and current solutions. Thus an individual can explore the effects of particular right-hand sides on the mesh refinement and solution values, while other individuals are doing their own.

The FEARS system is initialized by distributing the pieces of the project data to their respective subdomain processes (thus the system's data is partitioned along two dimensions: the subdomain division and project vs. problem).

Each active problem is represented by a "user" process. When the user is not performing finite-element computations, his problem information resides within the user process. The user process can perform post-processing, graphic display, and other non-FEARS operations, and it can query subdomain processes (so as to get project data) even while the subdomain processes are engaged in finite-element computations.

One user at a time can be doing finite-element computations. Once that user gets control of the system, his "turn" is initialized by the distribution of his problem data to the subdomain processes. The system performs various "passes" (2.4), as the user directs. The effect of a pass on the problem data is to make the mesh finer and the solutions more accurate. When the user has no more directives to the system his turn is over, and the updated problem data is sent back from all the subdomain processes to the user process which was its source.

2.4. Passes

The natural unit of computation within FEARS is the "pass", each of which has the potential to refine the mesh, compute new solution values, and estimate the error of the solution under a prescribed norm. Passes are divided into phases, and are controlled centrally from a "control" process which sends out an order to each subdomain process at the beginning of each phase telling it what to do during that phase. The phase ends when the control process has received answers or reports from all relevant subdomain processes signifying that they have finished their work. Thus the control process synchronizes the subdomain processes in accordance with the phases.

There are two kinds of passes: long and short. A long pass has refinement, solution, and error estimation phases, while a short pass is one phase all by itself. Each phase will be discussed in turn.

2.4.1. The Refinement Phase of a Long Pass

At the beginning of the refinement phase, the control process sends to each 2DS process an error indicator value called the "high cutoff". This value is obtained by analyzing previous indicators, so that the equilibration will be achieved as well as possible. The 2DS process proceeds to refine each element in its current mesh whose error indicator is above the high cutoff; refinement means that the square corresponding to the particular element is subdivided into four squares. New points are added to one-dimensional subdomains, when necessary, as described above.

When finished with the refinement, each 2DS process reports back to the control process with information needed to compute computational costs and control further functions. When all the 2DS processes have finished, the control process requests accounting information from all 1DS processes. The phase ends when all 1DS processes have reported.

2.4.2. The Solution Phase of a Long Pass

During the solution phase of a long pass, new solution values are calculated on all "active" two-dimensional subdomains. An "active" two-dimensional subdomain is one containing at least one element whose error indicator is higher than a "low cutoff" value. Any subdomain which has just been refined is active, because the low cutoff is always set lower than the high cutoff.

At the borders of active regions, the current solution values are taken as boundary conditions for the solution within

the region. The solution of the resulting linear system requires the participation of a "solver" process, and different solver processes can handle noncontiguous active regions in parallel (Figure 3).

The solution phase begins when the control process sends orders to active 2DS processes and active solver processes. An order to a solver process contains a list of all points in its active region. From this information the solver sets up a template for the block macrostiffness matrix corresponding to these points.

Meanwhile the active 2DS processes have begun to work on their orders, which contain the identification of which of the 1DS and ØDS processes on their boundary are external (constant) or internal (variable) to the active region. The 2DS processes use a wave-front technique for computing microstiffness matrices (querying their borders of 1DS and ØDS processes as needed) and decomposing the block macrostiffness matrix corresponding to their internal points (see [Rheinboldt & Mesztenyi 80], especially Figure 9). During this procedure they generate a set of updates for the block matrices representing the one- and zero-subdomains in their regions.

When a 2DS process has completed its decomposition, it sends the updates to its solver process. When a solver process has received updates from all 2DS processes in its active region, it completes the decomposition of the part of the macro-stiffness matrix corresponding to its one- and zero-subdomain points augmented by the right-hand side. This allows the solver to

complete the solution for its block, which it then sends back to the 1DS and 0DS processes, where they are incorporated into the current mesh. A solver process completes its work by sending a restart signal to each 2DS process in its region.

Upon receiving the restart signal, an active 2DS process completes the solution for the block corresponding to its local points; to do this, it must ask its neighboring 1DS and 0DS processes for their new solution values. It also supplies its neighboring 1DS processes with information they will need to compute new directional derivatives (see 2.4.3). A 2DS process completes its work in this phase by signalling the control process that it has finished. When the control process has heard from all active 2DS processes, the phase ends.

2.4.3. The Error Estimation Phase of a Long Pass

"Irregular nodes" are nodes formed by adaptive refinement which do not participate in solutions because they are not vertices of four elements (Figure 4). The directional derivatives which are needed to compute error indicators are obtained by interpolation. In the case of irregular nodes found in 1DS processes, that interpolation must be performed after the 1DS process has received solution information from the 2DS processes on either side.

Thus the error estimation phase begins when the control process orders all 1DS processes to perform this interpolation. After all 1DS processes have reported completion, all 2DS processes are ordered to compute new error indicators. They do

this, and report back to the control process with information needed for cost accounting and further control decisions.

2.4.4. The Short Pass

A short pass is a shortcut which enables rapid, approximate adaptive refinement of the mesh. This is clearly desirable, since all activity before the solution on the final mesh serves only to generate that mesh. Since short passes produce considerable compromises in accuracy, however, they should be interrupted at regular intervals by long passes.

A short pass begins when the control process sends orders containing a high cutoff to all 2DS processes. Each 2DS process refines all its elements with error indicators above the high cutoff, computes a solution value for each new point by treating the corners of the refined elements as boundary conditions (Figure 5), and computes error indicators for the new elements. Thus solutions for new points and error indicators for new elements are computed using only information local to the elements from which they were refined. The phase ends when each 2DS process has reported back with control and accounting information.

2.5. Automatic Control

Each pass of the FEARS system requires the following control decisions: whether there is to be another pass, whether the pass is to be long or short, and what the high and low (long pass

only) cutoffs are to be. The FEARS design allows the user to take these decisions himself, or to have them made automatically.

In "automatic" mode, the user initiates a sequence of passes with a command that specifies the desired accuracy he hopes to achieve, a limit on the computational cost he is willing to incur, a limit on the number of passes that may be performed in response to this command, and the desired ratio of short to long passes.

Based on this command, the control process orders a sequence of passes which always ends with a long solution over the whole domain. It attempts to satisfy the user's goals for cost and accuracy, based on the information reported to it by the subdomain processes at the ends of phases. Costs are estimated as described in [Zave & Rheinboldt 79]. Whatever the outcome, the cost and accuracy of the result are reported to the user.

Details of the user interface and automatic control decisions can be found in [Zave 79].

3. THE FEARS EXPERIMENT

3.1. Specification and Implementation of the Design

The FEARS design is specified formally in the executable specification language described in [Zave 82]. The specification describes processes and interactions as well as functions and data structures; it is complete down to, but not including, numerical algorithms and numerical data structures. The data

structures used to support an adaptable mesh are presented in [Rheinboldt & Mesztenyi 80].

The FEARS design is implemented by the program PF in FORTRAN V and assembly language on a Univac 1108. Our implementation runs configurations with one user process, one solver process, and up to 81 subdomain processes (16 2DS, 40 1DS, 25 0DS). Each 2DS process is limited to approximately 256 elements.

PF is divided into three distinct parts: (a) There is code which is an image of the formal specification, and therefore defines the process, control, and communication structure down to the level that numerical functions are applied to numerical data structures. (b) There is support code on which the "specification" runs. It simulates parallel computation of the processes and carries out inter-process communication. (c) There is numerical code, which is always called as a subroutine from the "specification" code. Running under a small interactive driver of its own, the numerical code forms the program FMP (see 1.1). The sizes of these program parts are approximately 8000 (code only), 15000 (code and data), and 18000 (code only) 36-bit words, respectively.

The output of a run of PF is twofold. There is the numerical output of the finite-element computation, which is identical to that of FMP. The other is "tasking" information--a machine-independent trace of parallel computation and process interactions. As is described in Section 3.2, the tasking information was used to simulate how FEARS would have performed if implemented on any one of several new computer architectures.

This intermediate representation made it easy and inexpensive for us to perform one FEARS run and then evaluate its performance on several machines.

A "task" is computation performed completely within one process, and is defined as a longest sequence of computational steps uninterrupted by the need to communicate with another process. Each process goes through a sequence of tasks, each task beginning with the receipt of a message from some process and ending with the sending of a message to some other process (Figure 6). A process may have to wait between tasks, if its continued computation is dependent on progress in other parts of the system.

The tasking information consists of the length (computation time) of each task, the size of each message, and the identification information used in interpreting the messages so as to produce the intended cooperation and synchronization among processes. This is sufficient to reconstruct the communication under different assumptions about computation and communication times.

3.2. Hardware Architectures and Their Simulation

3.2.1. Trends in Parallel Architectures

Continued decreases in the cost of computer hardware have sparked continuous interest in the use of parallel processing to increase speed and throughput. There are two general approaches

to doing this: parallelism within a single processor, applied to the execution of a single instruction stream, or the use of multiple processors. The readers are undoubtedly familiar with the much-publicized research activity in building powerful computers with instruction-level parallelism and in utilizing those computers for numerical applications; we have eschewed this approach for the reasons given in 2.1.

Multiprocessor architectures are the subject of equally vigorous research activity, the more so as they are often based on relatively small and simple microprocessors. The cost of such processors can be astoundingly low. One computer we shall consider (the ZMOB) is based on 256 Z80A microprocessors; although these chips now cost less than $10 apiece, the effective throughput of the machine is estimated at 75 million instructions per second.

In a multiprocessor architecture, each processor executes its own program, in parallel with other processors and independent of them except for occasional explicit communication. There is a natural correspondence between this mode of operation and the process structure of FEARS. Thus our investigation of hardware architectures suitable to become FEARS machines was limited to multiprocessors.

The biggest question in multiprocessor design is how the various processors communicate with each other. One approach to doing this is to have the processors share a common memory, as in the C.mmp architecture, in which 16 PDP-11/40E minicomputers with standard peripherals are connected to a common primary memory via

a crossbar switch (see [Jones & Schwarz 80] for a convenient overview). This approach has been largely discredited due to the problems caused by contention for the shared memory. Contention can degrade performance substantially even for a modest number of processors, and makes expansion to a large number of processors out of the question.

The other alternative is for each processor to own a private memory in which its programs and data are stored, and to communicate with other processors by means of buses (communication links) designed for the purpose. In the next section we describe two promising multiprocessor architectures of this type.

3.2.2. Two Multiprocessors

ZMOB is a research computer now being constructed at the University of Maryland ([Rieger 79], [Rieger et al. 80], [Rieger et al. 81]). It is based on 256 Z80A 8-bit microprocessors, each with up to 64K bytes of memory and an average instruction time of 2.5 microseconds. There are plans to augment the Z80A's with 32-bit floating-point hardware so that the machine can be used for research in numerical analysis. Times for floating-point operations are in the range of 30-70 microseconds.

Each processor in ZMOB can send messages to any other processor, where a message can be a byte sequence of any length. A processor sends a message by placing it in addressed, two-byte packets on a "conveyor belt" which links all the processors. Since the conveyor belt is circular, each "bin" on it visits each

processor on each revolution. When a bin containing a message packet arrives at the "mail stop" of the addressed processor, the addressed processor removes the packet.

Since each processor owns the bin in which it sends message packets (i.e. there are 257 bins on the conveyor belt, 256 for the Z80A's and one for the mainframe interface), there is no interprocessor competition for the link and the speed of message transmission is not affected by loading. Furthermore, the speed of the conveyor belt is such that a processor can send a message as fast as it is capable of loading bytes from memory into its output register, which requires a small instruction loop. In other words, there is a "perfect" communication link connecting the (relatively slow) processors. To receive a message, however, a processor must stop what it is doing (it will be notified by an interrupt) to store the incoming bytes in its memory one by one.

ZMOB is connected to a DEC VAX 11/780 via the 257th mail stop. The VAX is used for program preparation, mass storage, etc.

Cm* is a research computer built at Carnegie-Mellon University. It is composed of 50 microcomputer modules, each comprised of a DEC LSI-11 processor, a standard LSI-11 bus, memory, and devices ([Jones & Gehringer 80], [Jones & Schwarz 80]).

Up to 14 modules (10 in the current configuration) are gathered into a cluster. Communication within a cluster takes place on a "map" bus, and is managed by a switch called a "Kmap".

Any number of clusters (5 in the current configuration) can

belong to a Cm*. The Kmap's for each cluster are connected via intercluster buses.

The style of communication among modules can be altered by microprogramming the Kmap's, which are fairly powerful processors in their own right. In the mode we have studied, the Kmap's simulate a single uniformly addressed shared memory, but one with non-uniform performance. A local reference (a processor referencing its own memory) takes approximately 3 microseconds, an intracluster reference (controlled by the cluster's Kmap using the map bus) takes approximately 9 microseconds, and an intercluster reference (involving the participation of the Kmap's of both clusters, communicating on an intercluster bus) takes approximately 26 *microseconds.* A processor is not usually slowed down by other modules' references to its memory, as these are handled independently by the "Slocal" switch, which connects each module to its map bus.

3.2.3. Performance Simulation of the Multiprocessors

We simulated the performance of FEARS, as if implemented on ZMOB and several variants of Cm*, by post-processing the tasking information obtained by running PF. In effect we "reran" each computation, but only in enough detail to determine how long it would take, not enough to produce numerical results. To do this it was necessary to determine, on each simulated machine, how long it would take to run tasks and how long it would take to send messages. The relevant parameters are listed in Table 1.

Each task time measured on the Univac was multiplied by a

constant factor to reflect the relative slowness of the microprocessors (the Z80A and LSI-11 are approximately equal in power). These constants were chosen with some care, by counting and grouping machine instructions, determining conversion factors for each group, and then computing a weighted average of the conversion factors. Even so, the constant is a gross approximation at best. Yet there is no better way to compare the execution time of the same program on two machines short of implementing it on both machines ([Agrawala 79]). In addition to the various differences in the instruction sets themselves, there is great variability in the code produced by different compilers. In our case we could not do direct comparison because no FORTRAN compilers for the Z80A or LSI-11 were available to us.

In some configurations, tasks in the solver were given a much lower conversion constant than other tasks. This is because a multiprocessor is often interfaced with a larger conventional computer (as ZMOB is to a VAX), and solver tasks (representing linear system solutions) should probably be assigned to the latter machine.

The time it takes to send a message is dependent on the communication architecture of the machine. For ZMOB there is a fixed delay of 40.96 microseconds per word (each 36-bit Univac word is assumed to convert to four bytes, and the conveyor belt moves two bytes every 20.48 microseconds), with the additional overhead that both the sender and receiver must spend full time attending to the message while it is being transmitted, and cannot do any other work.

For Cm* there is a per-word message transmission delay, but it depends on whether messages are inter- or intra-cluster. Thus configuration Cm*-1 differs from -2 and -3 in that in Cm*-1, the user, control, and solver processes are each in their own clusters (separate from the clusters in which the subdomain processes reside). In Cm*-2 and Cm*-3 only the solver is isolated in its own cluster. The transmission times are determined by doubling the memory-reference times, again because the data path is two bytes while finite-element words are assumed to be four. We attempted some variations in the assignment of subdomain processes to clusters, but this had a negligible effect on the measurements.

The final simulated machine, Cm**, is a hypothetical Cm* with a single large shared memory suffering no contention. The fact that "messages" are still "sent" from one task to another reflects the FEARS modularity, which assigns each task its own private areas of the shared memory. The behavior of Cm** is the best-case boundary of the behaviors of all configurations, cluster assignments, etc. of Cm*.

3.3. Experimental Domains and Raw Data

Experiments were run using five basic domains (project files), which are depicted in Figure 7(a)-(e). Table 2 describes and gives the raw Univac processing data for each of the 22 jobs (a job is a "turn" on a "problem") run on the FEARS system.

In Table 2, "memory usage" refers to the data for program parts (a) and (c) described in 3.1. It is computed by summing

1300 words for the user process, 400 for the control process, 26000 for the solver process, 14000 for each 2DS process, 500 for each 1DS process, and 50 for each ØDS process. The subdomain data structures are at their maximum size, i.e. are capable of accomodating 256 elements in each two-dimensional subdomain. These figures reflect total file storage rather than main memory usage, because only the solver data or the data for one "frame" (a 2DS process and its surrounding 1DS and ØDS processes, as in Figure 2(d)) is in main memory at any given time.

4. EXPERIMENTAL RESULTS

When FEARS tasking information is run through a simulator of one of the five machine configurations, the output is a number called the "speedup factor". The speedup factor is a measure of how much faster that FEARS run would have been executed by that parallel machine, as compared to an implementation on a sequential machine with a processor of the same power as those in the parallel machine. It is computed by dividing the simulated completion time of the job into the sum of all its converted task times.

Thus the speedup achieved reflects both the advantage of parallel execution and the disadvantage of communication delay. Other advantages (such as smaller memories) and disadvantages (such as the need for more processors) of the proposed implementations are static in nature, and can therefore be evaluated without simulation.

The results are organized into the answers to three

questions: What is the effect of data segmentation? How much real-time speedup is achieved? What can we conclude about the suitability of various hardware architectures?

4.1. Data Segmentation

The FEARS design provides a nontrivial segmentation of the finite-element data along subdomain boundaries. This segmentation is potentially useful for dynamic storage management and system modularity, as well as parallel processing. For instance, it solves many of the technical problems involved in maintaining a very large finite-element domain, but only working on selected parts of it at any given time.

The experiments that best indicate the efficiency of the data segmentation are those with a single two-dimensional subdomain (Domain A). This is because the major overhead incurred on account of data segmentation is that a 2DS process must communicate with its frame of 1DS and ØDS processes, and the experiments with Domain A show this effect and no other.

The results of these experiments are shown in Table 3. The speedup factors are clustered around 1.00; this shows that the time overhead incurred for communication is compensated for by the parallelism, so that FEARS data segmentation does not slow finite-element computations down. The amount of parallelism is small here, of course, because the 1DS and ØDS processes do little computation.

These results will not be very different for any size of domain, because even in the largest of domains, a 2DS process

still communicates only with the other subdomain processes in its immediate frame. The only potential difference is that 1DS and 2DS processes may be slightly busier handling requests from the several 2DS processes whose frame they belong to. This effect is finite and bounded, however, and demands on these processes are relatively light.

4.2. Real-Time Speedup

The FEARS design also offers the potential for faster finite-element computation by taking advantage of parallel execution. This was evaluated by means of experiments with Domains B, C, D, and E, the results of which are given in Table 4. In addition to the speedup factor, Table 4 also reports a "speedup efficiency", which is the speedup factor divided by the number of two-dimensional subdomains. This is based on the notion that the number of two-dimensional subdomains indicates fairly the greatest degree of parallelism we could hope to achieve with a given domain.

Modest speedups are achieved, but the speedup factor seems to have an upper bound at about 5. For 16 2DS processes, the speedup efficiency is around 25 per cent. We were unable to run tests with more than 16 two-dimensional subdomains due to limits on computational resources.

We conjecture that the speedup factor would hover around 5 even for experiments with more such subdomains. This is based on careful inspection of the tasking data as run on ZMOB, which suggests the following explanation for the degree of parallelism

observed: At the beginning of each phase the control process must send an order to each participating subdomain process. Typically, by the time the orders to the fourth or fifth 2DS process are going out, the first 2DS processes to be activated are completing their work. Because of the ratio of communication to processing times, rarely if ever are more than 5 2DS processes active in parallel.

Better performance with respect to parallelism would be expected if the task-processing times were larger relative to communication delays. Probably the best way to achieve this would be to make each two-dimensional subdomain larger, with a substantial number of elements to process. We were unable to experiment further in this direction due to limitations on our implementation.

4.3. Hardware Architectures

It can be seen from Tables 3 and 4 that speedups are not significantly affected by the communication structures of the machines. Cm** performs the best, as would be expected, but even on jobs with 16 2DS processes it was only slightly faster than ZMOB. Thus we conclude that FEARS performance is not sensitive to differences between members of our general class of microprocessor-based multiprocessors.

The one factor that did make a difference is using a faster processor for solver tasks. Cm*-1 and Cm*-2, which do not have this feature, are definitely the worst performers.

5. CONCLUSION

Before drawing final conclusions, we would like to make a few observations about the measurements and measurement techniques used here. Unfortunately, performance simulation studies of this type are intrinsically high in cost and low in accuracy--they are attempting to find a representative simplification of complex dynamic behavior no one really understands. (One characteristic error is pessimistic results because normal optimizations and efficiency "tricks" are not exploited.) The cost and resource demands of simulation imposed important limitations on the test cases we could attempt (Section 4). Nevertheless, the stability of the measurements we were able to make argued against a proliferation of test cases within the range where experimentation was feasible.

The most important thing to be said about the speedup factors measured is that they are realistic because they include the whole computation, including initialization, communication, and largely sequential stages of processing. In studies of parallelism there is a great temptation to concentrate on the highly parallel stages of algorithms, ignoring the overhead stages which fall between them, and which may easily be more significant in terms of time and resources. No such "editing" has been done here.

The results on real-time speedup are not particularly promising. For machines in the class studied here, communication delays clearly dominate processing, so that overall speeds are not greater than those of conventional finite-element

implementations. It is possible that much faster communication hardware or much larger subdomain processes would correct the imbalance and lead to impressive real-time speedups, but this has not been demonstrated.

Ideally the result of this study would be compared to similar work, but little similar work seems to exist. There have been many years of computer science research interest in parallel algorithms, but the focus has been on mathematical analysis of compact single-purpose algorithms (for examples, see [Baudet 78] and [Fishburn 81]), rather than on empirical study of whole systems encompassing several algorithms and the bookkeeping to make them work together at some useful task. Furthermore, most of these algorithms assume a very large number of processors, such as one for each point in the problem domain.

We believe that the emphasis of those interested in parallel systems will soon change. We conjecture that whenever real-time speedup has been attempted in the practical context of a whole system to do some useful task, the results have been disappointing (and thus not widely disseminated). This is because even the best partitions of computations into distributed parallel tasks suffer too much communication overhead. Such partitions offer other advantages, however, such as increased modularity, reliability, maintainability, and extensibility of systems. These are increasingly important characteristics, and we predict that interest in using distributed partitions of systems as a way to achieve them will grow steadily.

In this light the FEARS design does look promising, because

the data segmentation works well and is not sensitive to minor variations in hardware architectures. It is a reasonable segmentation strategy from an engineering standpoint, and ought to be exploited in the making of more manageable, adaptive finite-element systems. One possible application, for instance, would be a finite-element system able to run on a home computer. Such a system would never have more than one 2DS process in main memory at a time. Although a computation might take several hours, the convenience would be similar to that of a pocket calculator, a combination that might prove tantalizing to many engineers.

Our current recommendation for a full-scale "finite-element machine" would be a multiprocessor with somewhat fewer, larger, more powerful processors than offered by ZMOB or Cm*. Each processor should have the power and memory to perform reasonably fast computations for reasonably large two-dimensional subdomains. Having several of these running at once would reproduce the best speedups realized by our experiments. More importantly, the machine could be organized around highly interactive control and a mass-storage-management system based on segmentation by subdomains, so that the engineer could focus on interesting subdomains and develop trial meshes quickly and inexpensively.

ACKNOWLEDGMENT

We are grateful to the other members of the FEARS project, Werner C. Rheinboldt, Ivo Babuska, and Charles K. Mesztenyi, for

their continuing cooperation and support.

REFERENCES

[Agrawala 79]
Ashok K. Agrawala, Personal communication, 1979.

[Babuska et al. 81]
I. Babuska, B.A. Szabo, and I.N. Katz, "The p-version of the finite element method", SIAM Jour. Numer. Anal. 18, 1981, pp. 515-545.

[Babuska & Dorr 81]
I. Babuska and M.R. Dorr, "Error estimates for the combined h and p versions of the finite element method", Numerische Mathematik, 1981, to appear.

[Babuska & Miller 81]
I. Babuska and A. Miller, "A-posteriori error estimates and adaptive techniques for the finite element method", Institute for Physical Science and Technology Tech. Note BN-968, University of Maryland, College Park, Maryland, 1981.

[Babuska & Rheinboldt 78a]
Ivo Babuska and Werner C. Rheinboldt, "A-posteriori error estimates for the finite element method", Int. Jour. Numer. Methods in Engr. 12, 1978, pp. 1597-1615.

[Babuska & Rheinboldt 78b]
Ivo Babuska and Werner C. Rheinboldt, "Error Estimates for Adaptive Finite Element Computations, SIAM Jour. Numer. Anal. 15, August 1978, pp. 736-754.

[Babuska & Rheinboldt 78c]
Ivo Babuska and Werner C. Rheinboldt, "On a System for Adaptive, Parallel Finite Element Computations", Proc. ACM Annual Conf., New York, New York, 1978, pp. 480-489.

[Babuska & Rheinboldt 79a]
Ivo Babuska and Werner C. Rheinboldt, "Adaptive Approaches and Reliability Estimations in Finite Element Analysis", Comp. Methods in Applied Mech. and Engr. 17/18, 1979, pp. 519-540.

[Babuska & Rheinboldt 79b]
Ivo Babuska and Werner C. Rheinboldt, "Analysis of Optimal Finite-Element Meshes in R1", Math. of Comp. 33, April 1979, pp. 435-463.

[Babuska and Rheinboldt 79c]

Ivo Babuska and Werner C. Rheinboldt, "On the Reliability and Optimality of the Finite-Element Method", Computers and Structures 10, 1979, pp. 87-94.

[Babuska & Rheinboldt 80]
Ivo Babuska and Werner C. Rheinboldt, "Reliable Error Estimation and Mesh Adaptation for the Finite-Element Method", Computational Methods in Nonlinear Mechanics, J.J. Oden, ed., North-Holland Publishing Co., Amsterdam, 1980, pp. 67-108.

[Babuska & Szabo 81]
I. Babuska & B.A. Szabo, "On the rates of convergence of the finite element method", Int. Jour. Numer. Methods in Engr., 1981, to appear.

[Baudet 78]
Gerard M. Baudet, "The Design and Analysis of Algorithms for Asynchronous Multiprocessors", Computer Science CMU-CS-78-116, Carnegie-Mellon University, Pittsburgh, Penn., April 1978.

[Fishburn 81]
John P. Fishburn, "Analysis of Speedup in Distributed Algorithms", Computer Sciences TR-431, University of Wisconsin--Madison, Madison, Wisconsin, May 1981.

[Jones & Gehringer 1980]
Anita K. Jones and Edward F. Gehringer, eds., "The Cm* Multiprocessor Project: A Research Review", Computer Science CMU-CS-80-131, Carnegie-Mellon University, Pittsburgh, Pennsylvania, July 1980.

[Jones & Schwarz 80]
Anita K. Jones and Peter Schwarz, "Experience Using Multiprocessor Systems--A Status Report", ACM Computing Surveys 12, June 1980, pp. 121-165.

[Rheinboldt 81]
Werner C. Rheinboldt, "Adaptive Mesh Refinement Processes for Finite Element Solutions", Int. Jour. Numer. Methods in Engr. 17, 1981, pp. 649-662.

[Rheinboldt & Mesztenyi 80]
Werner C. Rheinboldt and Charles K. Mesztenyi, "On a Data Structure for Adaptive Finite Element Mesh Refinements", ACM Trans. Math. Software 6, June 1980, pp. 166-187.

[Rieger 79]
Chuck Rieger, "ZMOB: A Mob of 256 Cooperative Z80A-Based Microcomputers", Proc. ARPA Image Understanding Workshop, Univ. Southern Calif., November 1979.

[Rieger et al. 80]

Chuck Rieger et al., "ZMOB: A Highly Parallel Multiprocessor", Proc. IEEE Workshop on Picture Description and Management, Asilomar, Calif., August 1980.

[Rieger et al. 81]
Chuck Rieger et al., "ZMOB: A New Computing Engine for AI", Computer Science TR-1028, University of Maryland, College Park, Maryland, March 1981.

[Zave 79]
Pamela Zave, "User's Manual for the FEARS System", Computer Science Department, University of Maryland, College Park, Maryland, October 1979.

[Zave 82]
Pamela Zave, "An Operational Approach to Requirements Specification for Embedded Systems", IEEE Trans. Software Engineering, 1982, to appear.

[Zave & Rheinboldt 79]
Pamela Zave and Werner C. Rheinboldt, "Design of an Adaptive, Parallel Finite-Element System", ACM Trans. Math. Software 5, March 1979, pp. 1-17.

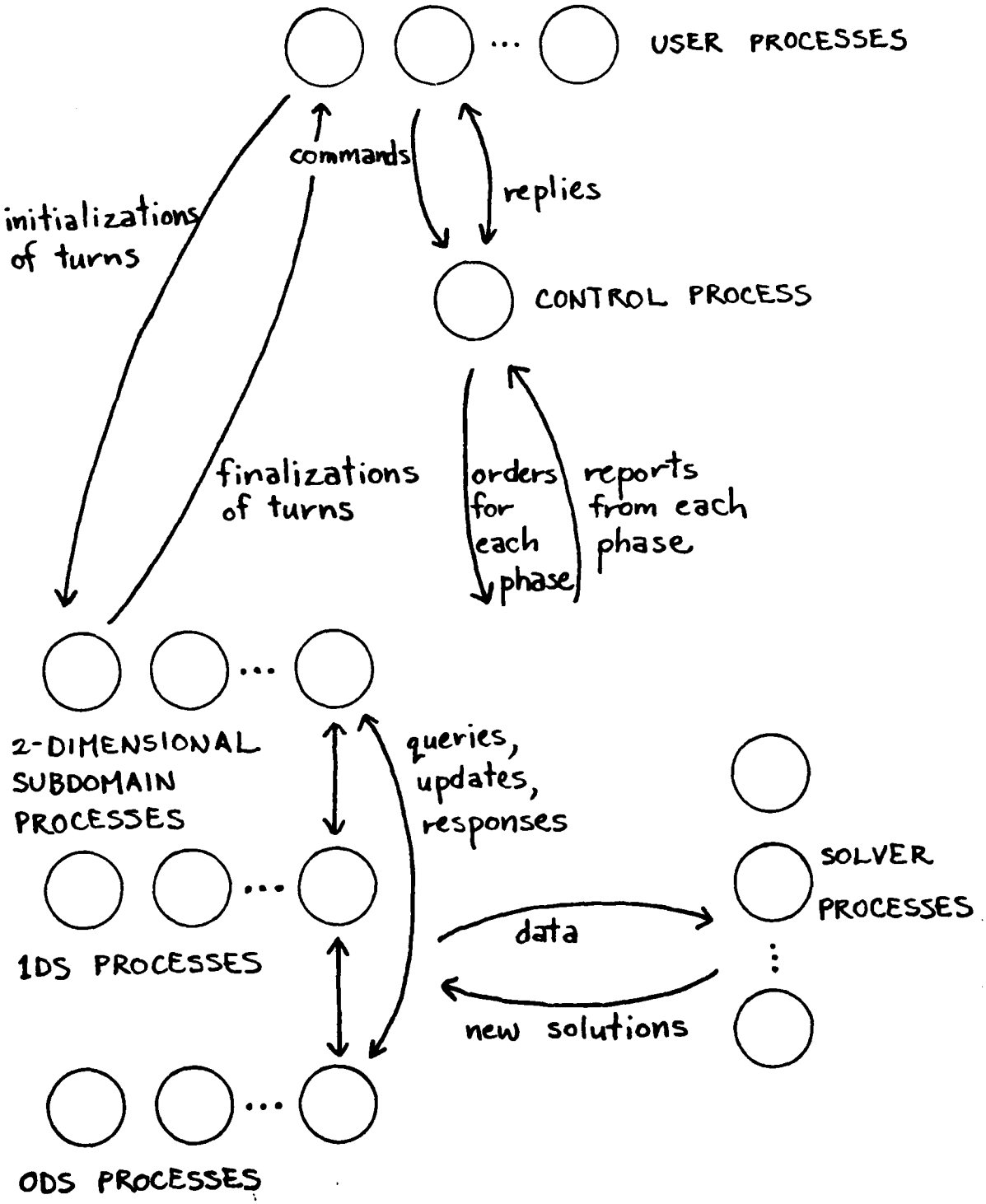
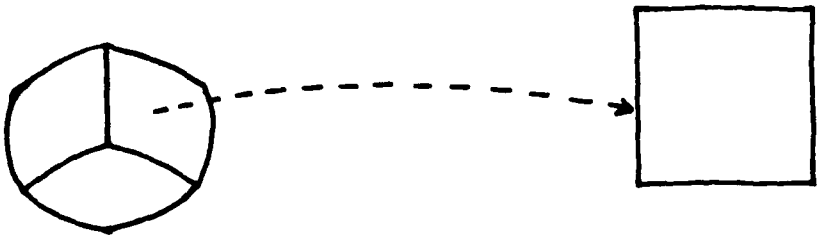
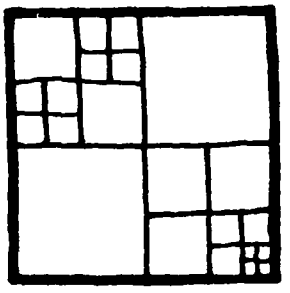


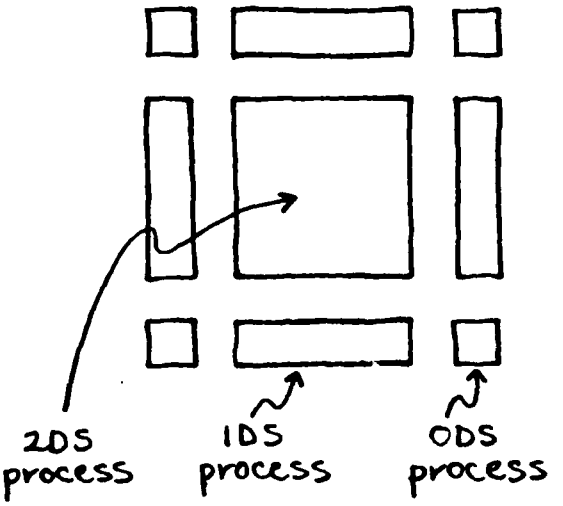
Figure 1. The process and communication structure of FEARS.



(a) figures of the problem domain

(b) figure of the problem domain mapped onto a unit square





(c) adaptive refinement within a square

(d) the process structure associated with a square

Figure 2. The partition into subdomains.

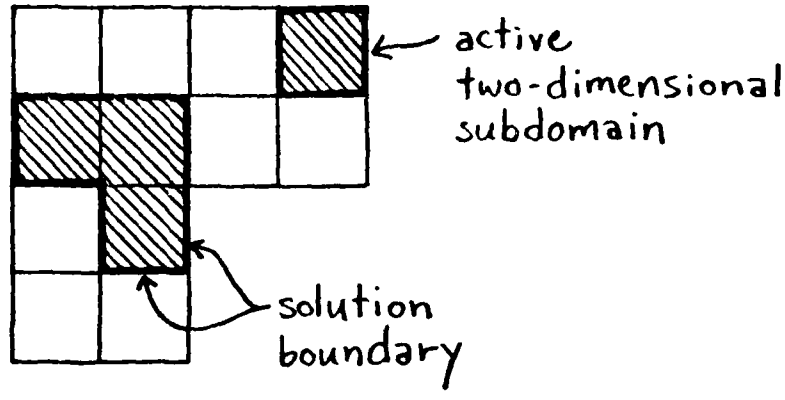


Figure 3. Parallel active regions.

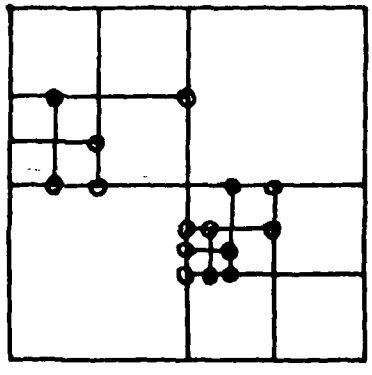


Figure 4. Irregular nodes (circled nodes are "irregular").

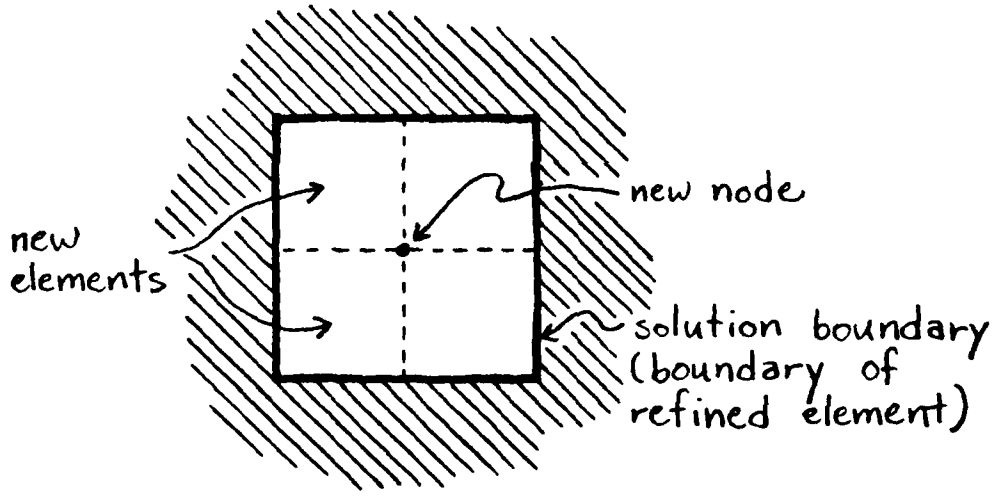


Figure 5. Refinement in a short pass.

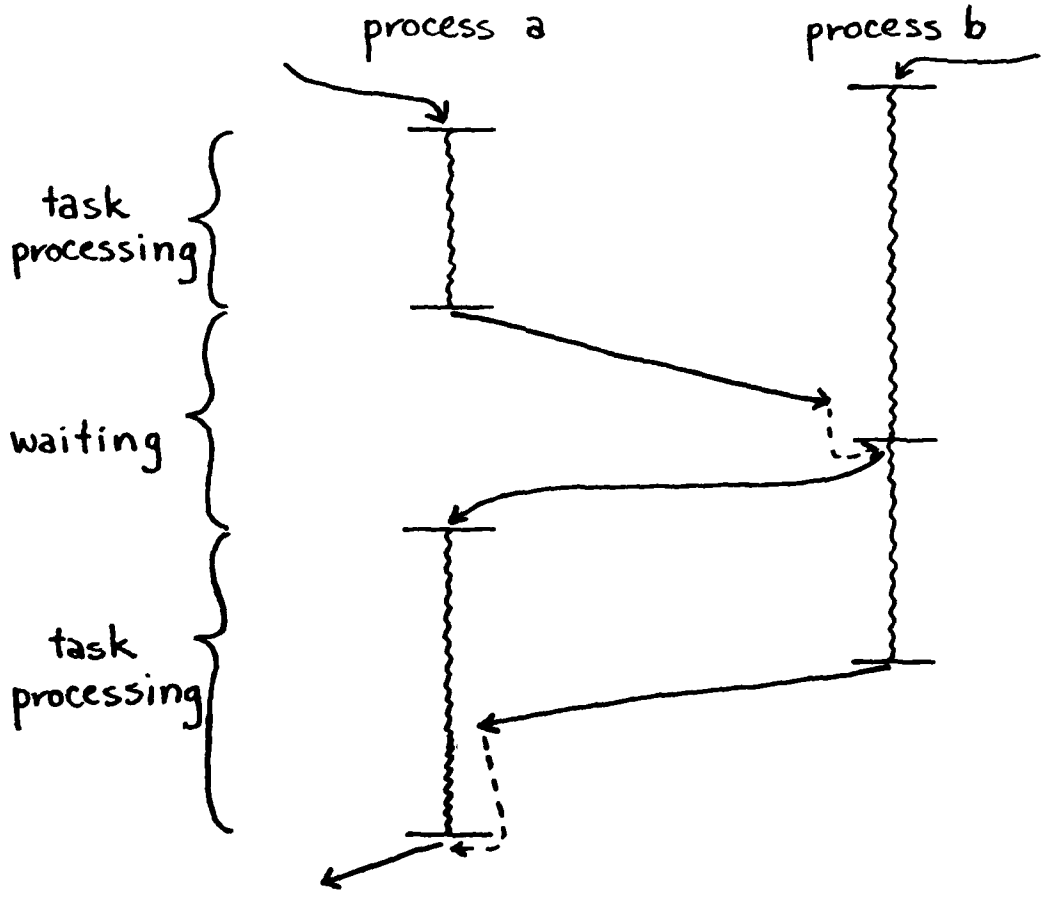


Figure 6. Tasks.

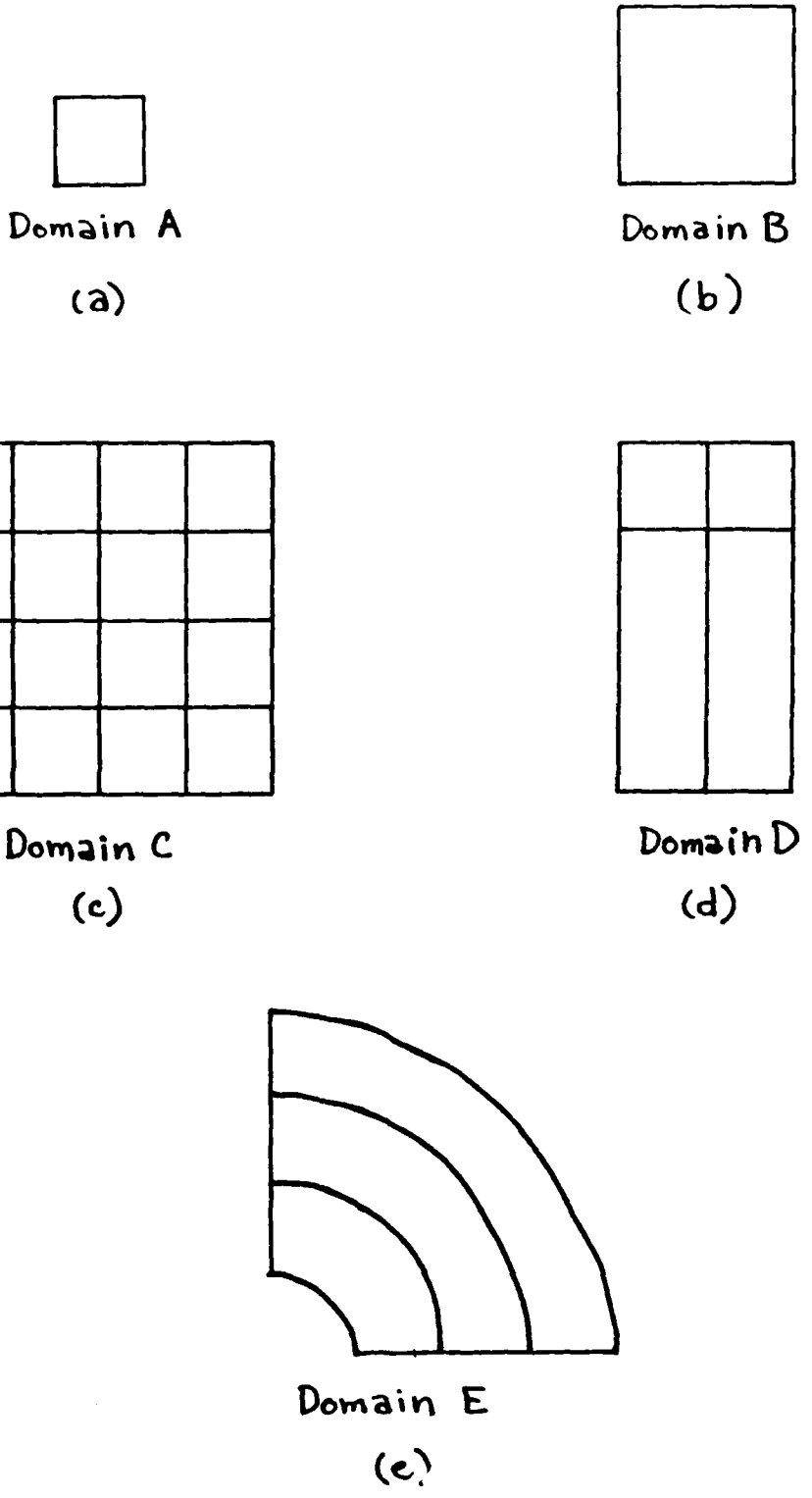


Figure 7. The five experimental domains.

SIMULATION PARAMETERS / SIMULATED MACHINES	task times		message transmission	
	conversion factor, non-solver tasks	conversion factor, solver tasks	transmission time (microseconds/ word)	additional overhead
ZMOB	5	1.2	40.96	sender and receiver detained during transmission
Cm*-1	5	5	18, 52	only sender detained during transmission
Cm*-2	5	5	18, 52	only sender detained during transmission
Cm*-3	5	1.2	18, 52	only sender detained during transmission
Cm**	5	1.2	6	only sender detained during transmission

Table 1. Simulated machines and simulation parameters.

Job Number	Domain	Description	Univac Sequential Processing Time (seconds)	Memory Usage (thousands of words)	Number of Tasks	Average Message Size (words)
1	A	long pass, 4 elements, no refinement	.7816	44	266	50.2
2	A	long pass refining 4 to 16 elements	3.404	44	338	49.8
3	A	long pass refining 16 to 64 elements	28.37	44	362	138
4	A	long pass, refinement at boundary to go from 64 to 112 elements	122.2	44	410	457
5	A	long pass refining 64 to 256 elements	244.8	44	410	857
6	B	long pass, 16 elements, no refinement	3.941	90	979	48.3
7	B	long pass refining 16 to 64 elements	28.64	90	1315	62.8
8	B	long pass, refinement at boundaries to go from 64 to 112 elements	67.21	90	1323	103
9	B	long pass refining 64 to 256 elements	262.4	90	1459	283
10	C	long pass, 64 elements, no refinement	20.21	273	3803	48.4
11	C	long pass, refinement at boundaries to go from 64 to 112 elements	41.51	273	4371	48.4
12	C	long pass refining 64 to 256 elements	180.8	273	5243	73.8
13	D	long pass, 16 elements, no refinement	4.778	90	1003	48.4
14	D	long pass, uneven refinement of 16 to 28 elements	9.025	90	1183	46.5
15	D	long pass, uneven refinement of 28 to 46 elements	15.53	90	1239	51.6
16	D	long pass, uneven refinement of 46 to 64 elements	25.33	90	1255	61.6
17	D	long pass, uneven refinement of 64 to 100 elements	55.42	90	1309	91.1
18	E	long pass, 12 elements, no refinement	3.531	75	782	47.6
19	E	long pass refining 12 to 48 elements	27.28	75	1062	67.2
20	A	3 short passes and 1 long, refining from 4 to 256 elements	267.3	44	614	574
21	B	2 short passes and 1 long, refining from 16 to 256 elements	283.1	90	1923	216
22	C	3 short passes with very uneven refinement followed by 1 long pass	295.9	273	5921	84.9

Table 2. Description of jobs and raw data.

Machine / Job Number	ZMOB	Cm*-1	Cm*-2	Cm*-3	Cm**
1	1.30	1.26	1.31	1.34	1.42
2	1.07	1.05	1.06	1.08	1.10
3	1.00	1.00	1.00	1.01	1.01
4	.994	--	--	--	1.00
5	.993	--	--	--	1.00
20	.994	--	--	--	1.00

Table 3. Speedup factors for jobs using Domain A.

Job	Number of 2DS	ZMOB		Cm**		Cm*-1		Cm*-2		Cm*-3	
		S.F.	S.E.	S.F.	S.E.	S.F.	S.E.	S.F.	S.E.	S.F.	S.E.
6	4	2.79	69.8	3.23	80.6	2.10	52.5	2.18	54.5	2.93	73.3
7	4	2.66	66.5	2.77	69.2	1.77	44.2	1.78	44.5	2.73	68.2
8	4	2.60	65.1	2.68	67.0	--	--	--	--	--	--
9	4	2.55	63.8	2.61	65.4	1.67	41.8	1.67	41.9	2.61	65.1
10	16	4.16	26.0	5.04	31.5	2.26	14.1	--	--	4.44	27.8
11	16	4.29	26.8	4.82	30.1	2.09	13.1	--	--	4.51	28.2
12	16	3.86	24.1	4.06	25.4	1.86	11.6	--	--	3.99	25.0
13	4	2.91	72.8	3.33	83.3	--	--	--	--	--	--
14	4	2.78	69.4	2.98	74.5	--	--	--	--	--	--
15	4	2.91	72.7	3.03	75.7	--	--	--	--	--	--
16	4	2.26	56.6	2.31	57.9	--	--	--	--	--	--
17	4	2.31	57.9	2.35	58.8	--	--	--	--	--	--
18	3	2.54	84.8	2.85	94.9	--	--	--	--	--	--
19	3	2.51	83.5	2.59	86.5	--	--	--	--	--	--
21	4	2.65	66.4	2.71	67.8	--	--	--	--	--	--
22	16	4.68	29.2	4.89	30.5	--	--	--	--	--	--

Table 4. Speedup factors and efficiencies for jobs using Domains B, C, D, and E. ("S.F." stands for "Speedup Factor"; "S.E." stands for "Speedup Efficiency", and is a percentage.

The Laboratory for Numerical Analysis is an integral part of the Institute for Physical Science and Technology of the University of Maryland, under the general administration of the Director, Institute for Physical Science and Technology. It has the following goals:

- To conduct research in the mathematical theory and computational implementation of numerical analysis and related topics, with emphasis on the numerical treatment of linear and nonlinear differential equations and problems in linear and nonlinear algebra.

- To help bridge gaps between computational directions in engineering, physics, etc. and those in the mathematical community.

- To provide a limited consulting service in all areas of numerical mathematics to the University as a whole, and also to government agencies and industries in the State of Maryland and the Washington Metropolitan area.

- To assist with the education of numerical analysts, especially at the postdoctoral level, in conjunction with the Interdisciplinary Applied Mathematics Program and the programs of the Mathematics and Computer Science Departments. This includes active collaboration with government agencies such as the National Bureau of Standards.

- To be an international center of study and research for foreign students in numerical mathematics who are supported by foreign governments or exchange agencies (Fulbright, etc.).

Further information may be obtained from Professor I. Babuška, Chairman, Laboratory for Numerical Analysis, Institute for Physical Science and Technology, University of Maryland, College Park, Maryland 20742.


ATE
LME