NASA-TM-84514 19820024127

NASA Technical Memorandum 84514

THE FINITE ELEMENT MACHINE: AN EXPERIMENT IN PARALLEL PROCESSING

O. O. Storaasli, S. W. Peebles, T. W. Crockett,
J. D. Knott, and L. Adams

JULY 1982

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

THE FINITE ELEMENT MACHINE: An Experiment in Parallel Processing

O. O. Storaasli and S. W. Peebles
NASA Langley Research Center
Hampton, Virginia

T. W. Crockett and J. D. Knott
Kentron Technical Center
Hampton, Virginia

L. Adams
University of Virginia
Charlottesville, Virginia

SUMMARY

The Finite Element Machine at the NASA Langley Research Center is a prototype computer designed to support parallel solutions to structural analysis problems. This paper describes the hardware architecture and support software for the machine, initial solution algorithms and test applications, preliminary results, and directions for future work.

INTRODUCTION

A large class of structural analysis problems is solved by computer using finite element and finite difference approximation techniques. Although these problems have traditionally been solved on conventional sequential computers, an analysis of these methods shows that they contain many calculations which could be performed simultaneously, thereby reducing the time required for a solution (ref. 1). To support this concurrency, special computers are needed which can perform many operations in parallel. One option is to construct vector computers which operate on large arrays of data, but this approach is only effective when the data can be structured appropriately. A different approach is to construct a machine which consists of a large number of general-purpose processing elements coupled together in a parallel architecture. Advances in microcomputer technology during the last decade have reduced the size and cost of computing elements, making construction of this type of parallel processor increasingly practical. Such processors are being actively investigated for their potential uses. Examples include CM* and C.mmp at Carnegie-Mellon University (ref. 2), ZMOB at the University of Maryland (ref. 3), and the New York University Ultracomputer (ref. 4).

Work is currently underway at NASA Langley Research Center to investigate solutions of structural analysis problems using parallel microprocessor

systems. Research topics include hardware configurations, software design, problem partitioning, and numerical algorithms. As part of this effort, a prototype parallel processor designated the Finite Element Machine (FEM) is being built and evaluated. This paper describes the Finite Element Machine, its support software, current applications and algorithms, and preliminary results.

FEM DESCRIPTION

To support parallel processing, an appropriate combination of hardware features and system software is needed. This section first outlines the FEM hardware organization, and then describes two packages of system software developed to provide control and run-time support.

Hardware Architecture

The architecture of the Finite Element Machine was specifically designed to support a parallel decomposition of structural problems by assigning nodes in the structural model to processors in the machine (refs. 1, 5, and 6). This approach is illustrated in figure 1 with an idealized wing model. The calculations to be performed at nodes in the model are mapped onto the array of microprocessors. The lines drawn between microprocessors indicate dependencies between nodes which lie on the same finite element, but have been mapped into different processors. Because of these dependencies, data transfer is required between these processors. The number of structural nodes is not limited to the number of processors, since multiple nodes may be assigned to a single processor (see, for example, ref. 7). The mapping of nodes onto processors is discussed in more detail in later sections.

The Finite Element Machine is a multiple-instruction multiple-data (MIMD) parallel processor consisting of an asynchronous array of interconnected microcomputers (the Array) linked to a minicomputer front end (the Controller). A block diagram of the architecture is shown in figure 2. Unlike many multiprocessor designs which use large shared memories, the FEM architecture provides each processor with its own local memory, and no sharing is possible. Instead, special communication hardware (described below) allows the processors to communicate with each other. The current prototype machine (figure 3) is being built in stages of 4, 16, and 36 processors. In principle, however, the architecture could be expanded to accommodate large numbers of processors (perhaps hundreds or thousands). At this writing, a four-processor Array is operational and the hardware for the 16- and 36-processor stages is nearly complete.

All processors in the Array are identical and consist of a 16-bit microprocessor, an attached floating-point unit, 32K bytes of random access memory (RAM), and 16K bytes of read-only memory (ROM). Serial I/O ports called "local links" provide data communication paths between a processor and up to twelve of its neighbors. The local links are reconfigurable and can support a variety of interconnection topologies. A time-multiplexed parallel "global

bus" connects all processors to each other and to the controller, and provides a general-purpose secondary communications path. A network of binary flags spans the Array and is used for processor synchronization and other signaling needs. A distributed "sum/maximum" network computes the sum and maximum of the inputs from all of the processors. This can be used for global calculations (ref. 8), cooperative sorting, and processor sequencing. For more details on the Array hardware, see reference 9.

The Controller is a small minicomputer which initiates and monitors activity on the Array and provides mass storage for programs and data on attached peripherals. It also hosts the user interface to the system, including interactive graphics.

Controller Support Software

The Controller provides the user with program development tools and the ability to define problems, activate and monitor the Array, and obtain and process results. The Controller runs a general-purpose disk-based operating system accessed by a menu-driven command interpreter. Commands on the Controller are implemented as control language procedures. All FEM commands are constructed in this manner. This approach presents the user with a consistent interface which is a natural extension of the Controller operating system. The system software provided on the Controller can be divided into four functional areas of support: program development, problem description, program execution on the Array, and post-processing or analysis.

Program development on the Controller is supported primarily by the vendor's standard software. A screen editor, an assembler, a reverse assembler, a Pascal compiler, and a link editor are available. Parallel application programs for the Array are written in Pascal; support for the FEM architecture is provided by a library of special routines. Users ordinarily select a program from a package of solution algorithms available for general use. Should a user prefer to write his own solution code or require special post-processing of data, he has access to all the necessary tools on the Controller.

Before executing the parallel solution program, the user must model his problem and provide data in accordance with the protocols of the intended solution algorithm. An interactive graphics interface allows the user to model structures and generate nodal coordinates, element connectivity, material properties, and constraints. As an alternative, the user may choose to create or modify data files using the text editor or his own utility program.

Normally, the program execution environment is established by entering a single Controller command. One command can be structured to call all necessary sub-commands via the command interpreter. Typically, the execution session involves Array initialization, selection of the Array configuration desired, downloading the selected algorithm and any necessary data, and entering an interactive execute mode. During program execution on the Array, all messages from a preselected reference processor and errors from all

processors are displayed on the user's CRT. Processors can send messages, make interactive queries, and report error conditions at any time during the execute sequence.

Three files are maintained for the user during execution on FEM. A "FEMDATA" file records all data transferred to the Controller, formatted in accordance with its type, and identified by source processor number. A "FEMLOG" file is used to record events over the course of an entire FEM session. This file is initialized when the Array is reset, and thereafter records each command invocation along with its associated data. For example, the command to download a program writes entry and exit messages, the name of the file downloaded, status information, and the load and entry addresses of the object code for each of the affected processors. The FEMLOG also records the source and error number for all errors as they occur. This process provides the user with a session record for later reference or analysis. A "FEMERROR" file is initialized at the beginning of each command. This file is used to record errors detected within the scope of a single command. It contains the error number, severity, source, processor status information, and an expanded error message for each error detected. If an error is detected during execution of any command, the FEMERROR file is displayed upon exit.

Additional user support is provided in the form of interactive debug commands. Debugging commands allow the user to dump memory and set breakpoints, to single step, halt, kill, and resume tasks, and to inspect and change status, registers, and memory. In addition, the Array keeps execution statistics and can be directed to trace execution and check in with the Controller at regular intervals to maintain confidence during long computations.

Software support on the Controller for post-processing and analysis of data consists of a set of utility programs. Commands are provided to sort the data file to provide a listing of communications by processor, analyze the trace information to determine where each processor spent its time during execution, and upload and format the results of computations on the Array. In addition, information such as nodal displacements can be displayed in graphic form (see figure 4).

System Software for the Array

System software for the array of microprocessors consists of an operating system, a subroutine library, and a set of diagnostics. The relationships of these components to each other and to the applications software are shown in figure 5. Although diagnostics are vitally important for the validation and maintenance of a computer system, they are beyond the scope of this paper.

A complete copy of the operating system, called Nodal Exec, is stored in ROM on each of the microcomputers in the Array. Nodal Exec is divided into two major sections, a nucleus and a package of command routines. The nucleus (the innermost portion of the operating system) provides such functions as interrupt handling, basic I/O, timing, memory allocation, task management, and a command monitor.

Command routines are used to implement all functions which the Controller may direct the microcomputers to perform. Such functions include downloading object code and data, establishing processor connectivity, executing programs, and uploading results. Several debug commands are also available to read and modify memory locations, inspect registers, set breakpoints, and step through instruction execution. The philosophy of Nodal Exec is to provide sufficient functionality with a set of relatively simple commands so that the Controller software can combine them into a sophisticated user interface.

A library of Pascal-callable subroutines, PASLIB, provides support facilities for application programs. The PASLIB routines are essentially an extension to Nodal Exec, serving as high-level supervisor calls or, in some cases, interfacing directly to hardware functions. They allow user programs to communicate with other processors and with the Controller, to use the flag and sum/max networks, to access data areas, and to perform arithmetic using the floating-point processor. Frequently used mathematical subroutines (e.g., vector dot product) are also available which use the stack architecture of the floating-point unit to optimize performance. The most commonly used PASLIB routines are stored in ROM on the processors. The remaining routines reside in a library file on the Controller where they can be linked to user programs and downloaded with the object code.

Nodal Exec and PASLIB support three major concepts which are important in understanding the flow of data on FEM: data areas, connectivity, and interprocessor communication. Explanations of each of these concepts are presented in the subsequent paragraphs.

Data areas are the primary mechanism for transferring data between the Controller and the application programs running on the processors in the Array. Data areas required by an application are allocated in each processor's memory prior to program execution. They contain space for a specified number of data items of a particular type. Allowable data types are integer, long integer, real, double precision, or user-defined records. Once allocated, a data area can be filled with data from the Controller, initialized to some value, or left empty. Application programs reference data areas via pointer variables. Data areas can provide input to the program, receive output, or both. Since data areas exist independently of the programs which access them, they can be used to pass information between separate programs which execute in a series. When a program (or series of programs) is finished, results stored in data areas are uploaded to the Controller for file storage or post-processing.

Connectivity is the concept of establishing communication paths between programs executing on different processors. Connectivity may be viewed at two levels, the logical problem level and the physical processor level. Logical connectivity refers to the interconnections between nodes in a structure by virtue of the fact that the nodes lie on the same finite element. Physical connectivity refers to the physical I/O interconnections between processors. For simple or regular structures, mapping the logical interconnection pattern onto the planar mesh of physical processors may be straightforward. In general, however, the mapping problem is difficult and the local neighbor

connections are insufficient; therefore, the global bus must be used. Bokhari has addressed this problem and developed an algorithm which attempts to maximize the use of the local links (ref. 10). This algorithm is implemented on the Controller by an auxiliary program whose output is the logical-to-physical mapping of node numbers to processor numbers.

Interprocessor communication takes place via the local and global I/O paths which were enabled during the connectivity process. Communication is based on the transmission of records which contain from one to 255 data words. An associated tag word in the record header is used to distinguish the information content when multiple records are sent to the same processor. Interprocessor communication is handled either synchronously or asynchronously by the system software. If synchronous mode is used, input from a neighboring processor is queued in the order in which it is received, and it must all be read and processed by the receiver. In the asynchronous case, only the most recently received record (for each different tag) is saved. An algorithm which uses this asynchronous or "chaotic" communication technique is discussed in the next section.

CURRENT ALGORITHMS AND APPLICATIONS

To solve structural problems in parallel requires the development of algorithms to support parallel computations and a scheme to partition the structural model for distribution among the processors. The following section discusses the assignment of problems to the Array, and gives results from several applications run on the four-processor version of FEM.

Problem Partitioning

Factors that influence the design of an appropriate algorithm for solving problems on FEM include the structural region discretization, the number of processors available, and the amount of communication required between processors. The following example illustrates these considerations.

Figure 6a shows a cantilevered rectangular plate in plane stress constrained on one edge and loaded on the opposite edge. If the plate is discretized by linear triangular finite elements, a structural node is common to at most six elements, and is connected to at most six other nodes (see figure 6b). This is significant because it implies that in the system of equations for the vector of displacements, $\mathbf{u}$, the stiffness matrix, K, is a sparse matrix containing at most 14 nonzero entries in each row:

$$K\ \mathbf{u} = \mathbf{f} \qquad (1)$$

Twelve of these entries represent contributions of the six neighboring nodes (two per node) to the solution at a given node while two additional entries are contributions from the given node itself.

The sparsity of the stiffness matrix, K, suggests that an iterative algorithm could be used to solve equation (1) by assigning one processor to calculate displacements at each node in the plate. For maximum efficiency, an algorithm should be developed such that each processor would only need to communicate information to its six neighbors via the dedicated local links, as shown in figure 6c. This scheme is feasible only if the number of processors is not less than the number of structural nodes and if a suitable iterative algorithm can be found to take advantage of the connectivity shown in figure 6c. Furthermore, such an algorithm is efficient only if the overhead due to communication between processors is not prohibitive.

In most instances, the number of structural nodes exceeds the number of available processors. For these cases, it is necessary to assign multiple nodes to a processor and to develop algorithms to solve for the displacements at these nodes. Figures 6d and 6e show, respectively, how nodes of the plate can be assigned to a 4- or 16-processor Array. The local links that are used by the processors in figures 6d and 6e are illustrated in figures 6f and 6g.

Even though FEM was designed with finite element discretizations in mind, the architecture also supports the solution of problems that are discretized by finite difference techniques. Two such problems and their associated discretizations are given in figure 7. The five star discretization of the membrane equation is shown in figure 7a and the discretization for the plate equation is given in figure 7b. For a one node per processor assignment, the iterative solution of the membrane equations using the discretization of figure 7a requires four local links of each processor while the iterative solution of the plate equation using the discretization of figure 7b requires all twelve of the local links. In the case of multiple nodes per processor, the solution algorithm determines the proper assignment of nodes to processors. In the following, algorithms that have been run on FEM for both finite element and finite difference discretizations are discussed.

FEM Applications

Solution algorithms for the first applications run on FEM used three standard iterative methods: Jacobi, conjugate gradient, and successive over-relaxation (SOR). These methods contain suitable parallelism and were used to solve sparse symmetric positive definite systems of linear structural equations resulting from finite element or finite difference discretizations.

Smith and Loendorf (ref. 11) solved a cantilevered wing box finite element model using the basic Jacobi iterative method. This small problem provided a useful benchmark for assessing a number of performance issues. Their results for one, two, and four processors show that the increased overhead for interprocessor communication largely offset the improvements gained by distributing the computation, thereby resulting in only modest reductions in the solution time. Their analysis suggests that there is a break-even point beyond which additional partitioning of a problem is ineffective. The results from this problem have also prompted a re-thinking of processor communication strategies and several modifications have been proposed to reduce overhead.

The same problem was also solved using an asynchronous Jacobi iterative method (see Baudet, ref. 12) in which each processor performs its calculations independently with no synchronization among processors. Intermediate results were passed between processors using the asynchronous communication mode discussed previously. The asynchronous Jacobi algorithm was run using two and four processors, and the results were inconclusive. In both cases, the asynchronous method required less time per iteration than the standard Jacobi technique, and the program converged to results which were similar to those of the standard Jacobi. However, the number of iterations required for convergence differed, with the two-processor case using about the same number as the standard Jacobi, and the four-processor case using more. The result was that for two processors, the asynchronous method slightly outperformed the standard Jacobi, but with four processors, the reverse was true. Further experimentation with modified communication procedures and other application problems is needed to better assess the asynchronous approach.

While the Jacobi iteration can be easily adapted as a parallel technique, it is not guaranteed to converge for general symmetric positive definite systems. The SOR method is guaranteed to converge for these systems, but is sequential in nature. To parallelize the successive overrelaxation method for FEM, the problem must be partitioned in such a way that the system is decoupled. A classical method of decoupling is the Red/Black ordering (ref. 13) for Laplace's equation. This procedure colors the discretization grid in a checkerboard fashion. Then an SOR sweep can be carried out by two Jacobi iterations, one on the equations corresponding to the red points, and one on the equations corresponding to the black points. This strategy does not work for higher order finite difference or finite element discretizations, however, because two colors are insufficient to decouple the system. Adams and Ortega (ref. 7) have developed a new iterative method that they call "Multi-Color" SOR which is a generalization of the Red/Black ordering. In Multi-Color SOR, an ordering is imposed on the sequence in which the displacements at the nodes are calculated, based on the number of colors required for decoupling. For example, if three colors (red, black, green) are used, the displacements for each color can be calculated by the processors in parallel. Each iteration of the algorithm first computes all red values, then all black values, and finally all green values. This scheme allows SOR to be implemented as a multiple sweep (one for each color) Jacobi-type method on FEM.

To test this method, Laplace's equation was solved on a square region discretized by quadratic triangular finite elements for which six colors are necessary and sufficient to color the discretization. This six-color SOR algorithm was programmed on a minicomputer to test its convergence properties and on the four-processor FEM to test its suitability for parallel implementation. A comparison showed that the problem converged with identical results on both machines.

A plane stress analysis of a plate was used to compare the Multi-Color SOR algorithm to the standard conjugate gradient method. The computer program for this procedure can be used to solve large plate problems by assigning three structural nodes to each processor or by assigning any multiple of three nodes to each processor if the number of processors is limited. The components of the program include the following:

1. Parallel assembly of stiffness matrix K

2. Three-color SOR solution of $K\,\mathbf{u} = \mathbf{f}$

   or alternatively,

   Conjugate gradient solution of $K\,\mathbf{u} = \mathbf{f}$

3. Parallel stress calculation

The Array can be used to assemble, in parallel, the stiffness matrix from the problem data without any communication between processors. Linear triangular finite elements are used to discretize the plate so that three colors are necessary and sufficient to implement SOR (see ref. 7). The calculation of the stresses can also be done in parallel without any processor communication. A more detailed description of the matrix assembly and the stress calculation processes is given in reference 14.

A comparison of the performance of four processors to one processor on a plane stress problem with 60 degrees of freedom is given in table 1. These speedups reflect the execution times of both the solution algorithms and the underlying system software. The maximum theoretical speedup for a four-processor system is 4.00. The processor efficiency values given are a measure of the overhead required for synchronization and communication in the multiprocessor case. Improved interprocessor communication times should increase the efficiency of these algorithms on FEM.

FUTURE DIRECTIONS

The solution of the plane stress analysis of a plate on FEM was felt to be a good starting place to address the issues of parallel matrix assembly, parallel displacement calculation, and parallel stress calculation. The experience gained by solving this problem provides a basis for the solution of more complex structural problems.

Although the initial applications of FEM have been based on iterative solution approaches, Gannon (ref. 15) has demonstrated that the architecture is sufficiently flexible to permit direct solution techniques. The study of such techniques on FEM is a major research area currently being investigated.

In conjunction with algorithm development, alternative processor interconnection strategies may also be investigated. To date, the local links have only been configured in an eight-nearest-neighbor planar mesh topology with toroidal wrap-around at the edges. This scheme leaves four of the links unused. Since the local links can be reconfigured by merely unplugging and rearranging the interconnecting cables, other topologies such as trees, rings, perfect shuffles (ref. 16), or cube-connected cycles (ref. 17) are possible. Because of the relatively large number of links per processor, it would even be possible to implement multiple interconnection patterns simultaneously (e.g., eight-neighbor mesh plus shuffle-exchange). The development of algorithms to make efficient use of alternate topologies is another topic for research.

The current FEM hardware is viewed as an experimental device rather than as a production machine. Data management considerations and an analysis of hardware and software performance are expected to point to changes in the architecture which could be incorporated in a second generation FEM. Such a machine would undoubtedly benefit from continuing advances in VLSI circuit technology which would improve performance and reduce the size and cost of components.

The potential range of FEM applications is not limited to structural analysis problems, although they provided the original motivation. By designing a parallel architecture thought to be suitable for finite element analysis, a flexible, reconfigurable machine has resulted which can emulate several distinct computer architectures. FEM could therefore be useful for parallel algorithms research in a number of disciplines.

CONCLUDING REMARKS

The Finite Element Machine is providing the impetus for development of new parallel techniques for the solution of structural analysis problems. Several of these techniques have been implemented and evaluated on the FEM hardware, thereby demonstrating that parallel solution of structural problems is a feasible approach. Initial results show that there can be a significant execution time advantage to be gained by using multiple cooperating processors. The degree of speedup is dependent on the number of processors, the algorithms used, and the extent to which the ratio of computation to interprocessor communication is maximized.

The work done so far has just begun to address the algorithms and applications suitable for investigation on FEM. Expansion of the machine to 16 and 36 processors will provide a research testbed for the exploration of many issues relating to parallel processing. It is hoped that the results of this research can be applied to future production computers which will benefit not only structural engineers, but other users as well.

REFERENCES

1. Loendorf, David D.: Advanced Computer Concepts for Engineering Analysis and Design, Ph.D. thesis (in progress), University of Michigan, Ann Arbor.

2. Jones, A. K.; and Schwarz, P.: Experience Using Multiprocessor Systems--A Status Report. Computing Surveys, Vol. 12, No. 2, June 1980, pp. 121-165.

3. Rieger, C.; Trigg, R.; and Bane, B.: ZMOB: A New Computing Engine for AI. University of Maryland, TR-1028, March 1981.

4. Gottlieb, Allan; et al.: The NYU Ultracomputer--Designing a MIMD, Shared-Memory Parallel Machine. Proceedings of the Ninth Annual Symposium on Computer Architecture, April 1982, pp. 27-42.

5. Jordan, Harry F.: A Special Purpose Architecture for Finite Element Analysis. Proceedings of the 1978 International Conference on Parallel Processing, August 1978, pp. 263-266.

6. Jordan, Harry F.; and Sawyer, Patricia L.: A Multi-Microprocessor System for Finite Element Structural Analysis. Trends in Computerized Structural Analysis and Synthesis, A. K. Noor and H. G. McComb, Jr., Editors, Pergamon Press, Oxford, 1978, pp. 21-29.

7. Adams, L.; and Ortega, J.: A Multi-Color SOR Method for Parallel Computation. Accepted for presentation at the 1982 International Conference on Parallel Processing, August 1982.

8. Jordan, Harry F.; Scalabrin, M.; and Calvert, W.: A Comparison of Three Types of Multiprocessor Algorithms. Proceedings of the 1979 International Conference on Parallel Processing, August 1979, pp. 231-238.

9. Jordan, Harry F., Ed.: The Finite Element Machine Programmer's Reference Manual. Computer Systems Design Group, University of Colorado, Boulder, 1979.

10. Bokhari, Shahid H.: On the Mapping Problem for the Finite Element Machine. Proceedings of the 1979 International Conference on Parallel Processing, August 1979, pp. 239-248.

11. Smith, Connie U.; and Loendorf, David D.: Performance Analysis of Software for an MIMD Computer. Report CS-1982-7, Department of Computer Science, Duke University, 1982.

12. Baudet, G.: Asynchronous Iterative Methods for Multiprocessors. Journal of the ACM, Vol. 25, April 1978.

13. Young, D.: Iterative Solution of Large Linear Systems, Academic Press, 1971.

14. Adams, L.: Iterative Algorithms for Parallel Computers. Ph.D. Dissertation (in progress), University of Virginia.

15. Gannon, Dennis: A Note on Pipelining a Mesh Connected Multiprocessor for Finite Element Problems by Nested Dissection. Proceedings of the 1980 International Conference on Parallel Processing, August 1980, pp. 197-204.

16. Stone, H. S.: Parallel Processing with the Perfect Shuffle. IEEE Transactions on Computers, Vol. C-20, No. 2, Feb. 1971, pp. 153-161.

17. Preparata, F. P.; and Vuillemin, J.: The Cube-Connected Cycles: A Versatile Network for Parallel Computation. Communications of the ACM, Vol. 4, No. 5, May 1981, pp. 300-309.

Table 1. Speedup Ratios for the Plane Stress Problem on Four Processors vs. One Processor

Algorithm	Speedup	Processor Efficiency
Stiffness Matrix Assembly	3.20	80%
3-Color SOR ($K\,u = f$)	2.84	71%
Conjugate Gradient ($K\,u = f$)	2.82	71%
Stress Calculation	4.00	100%

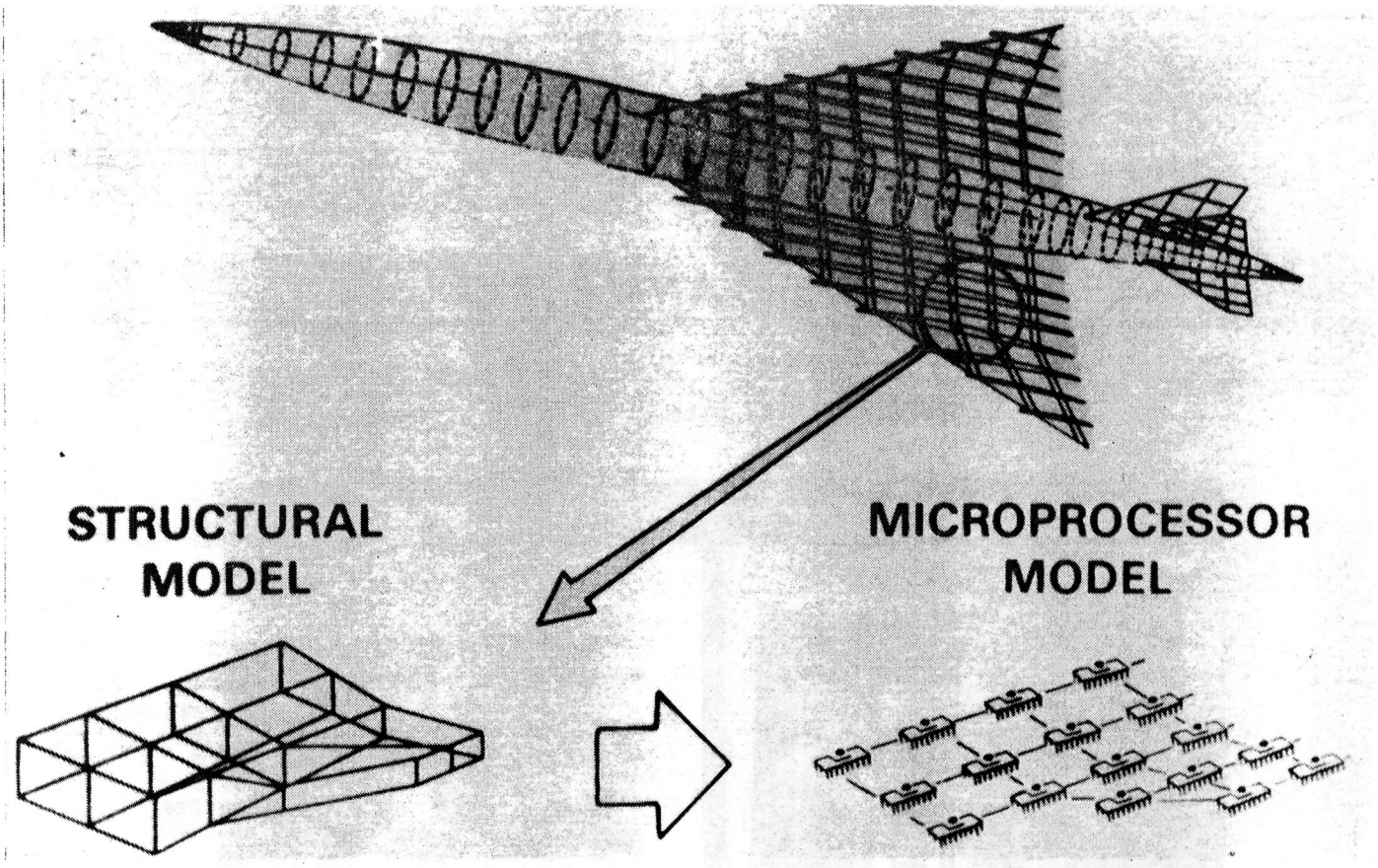


Figure 1.- A concept for parallel computation.

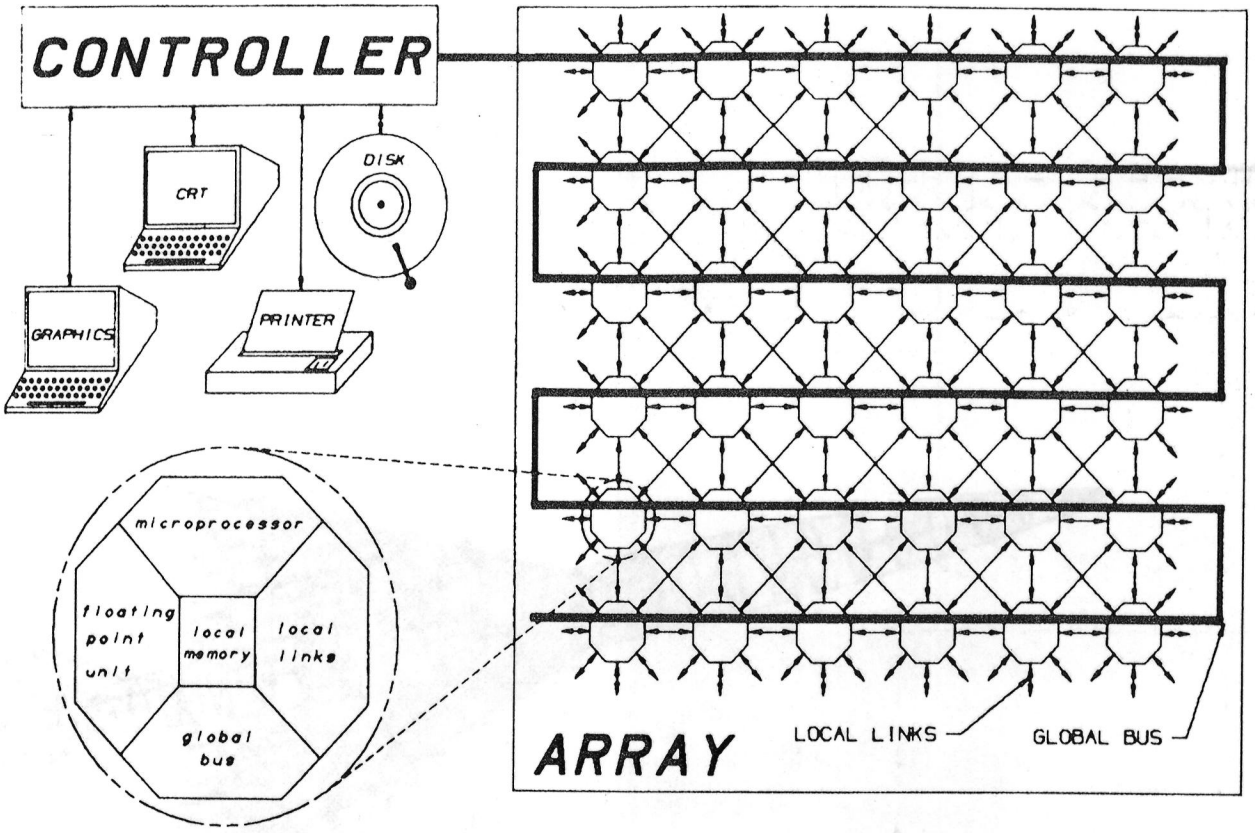


Figure 2.- Finite element machine block diagram.

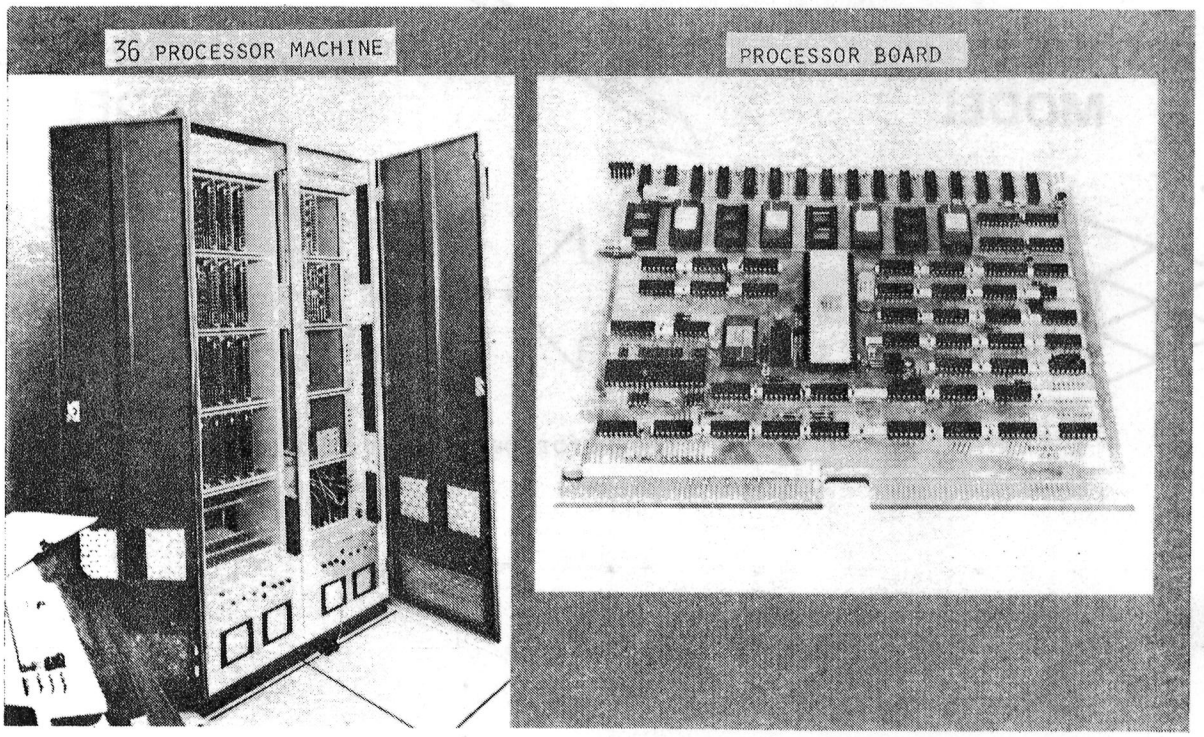


Figure 3.- Prototype finite element machine hardware.

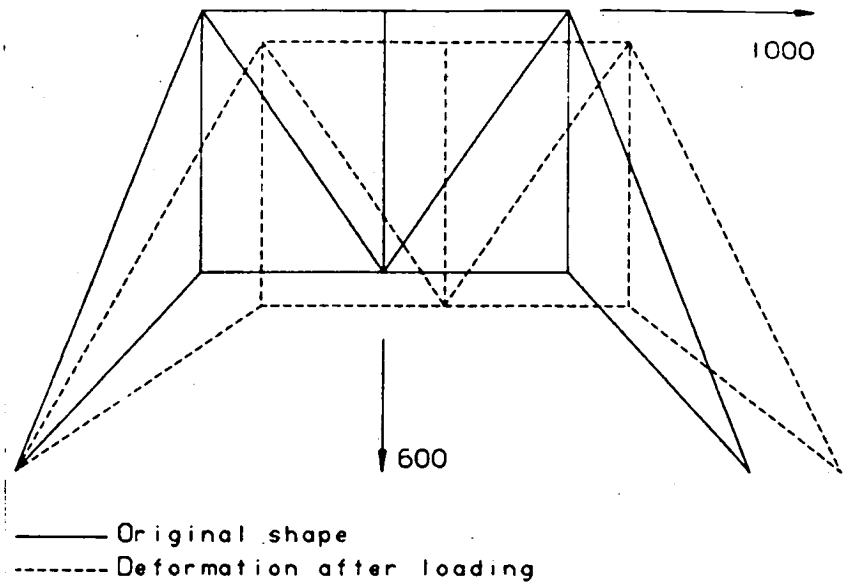


Figure 4.- Graphic display of nodal displacement.

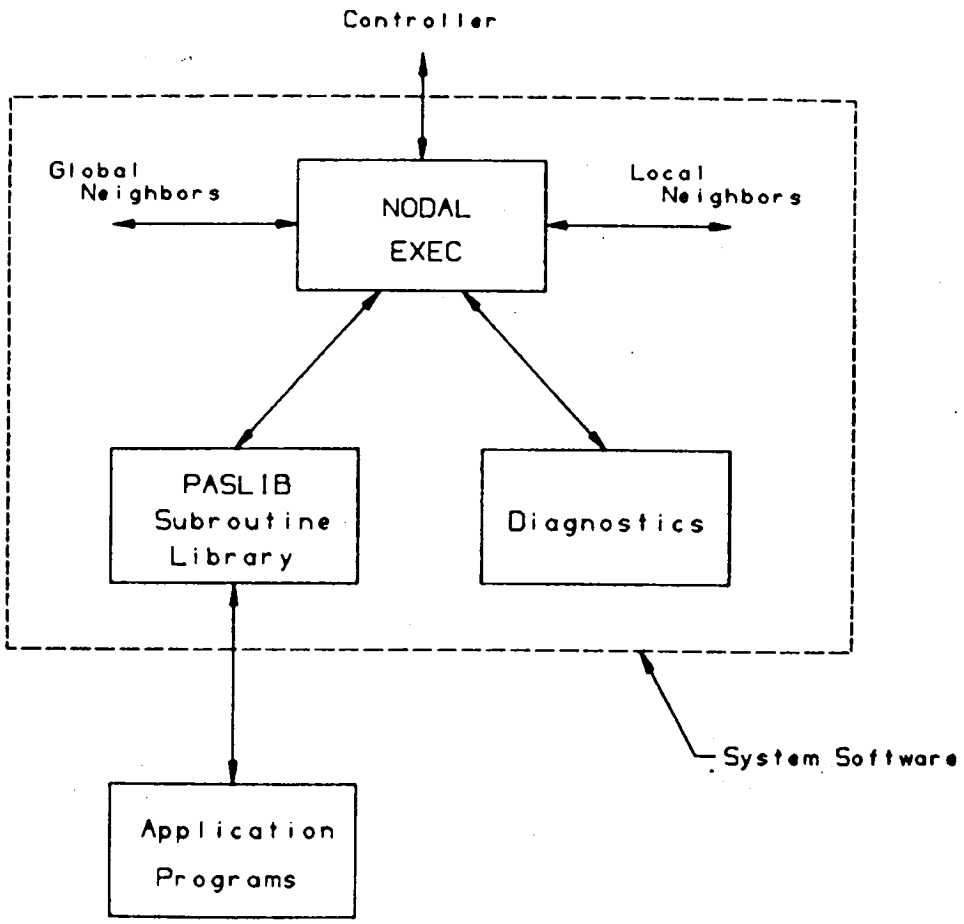
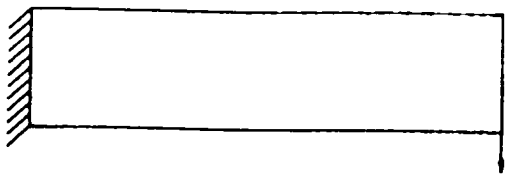
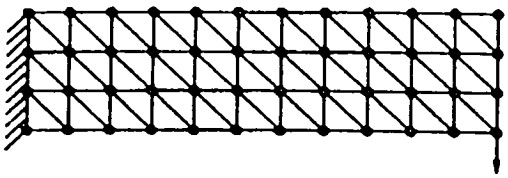


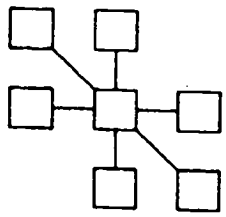
Figure 5.- Processor software configuration.



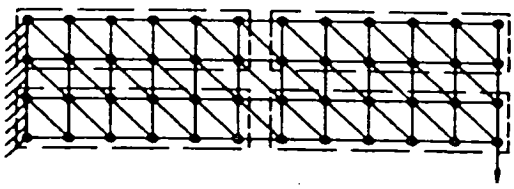
a). Plate under load.



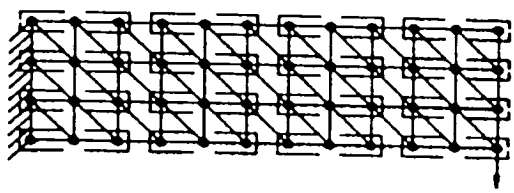
b). Finite element discretization.



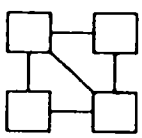
c). Six neighboring nodes.



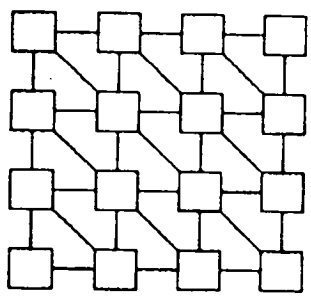
d). Four processor assignment.



e). Sixteen processor assignment.

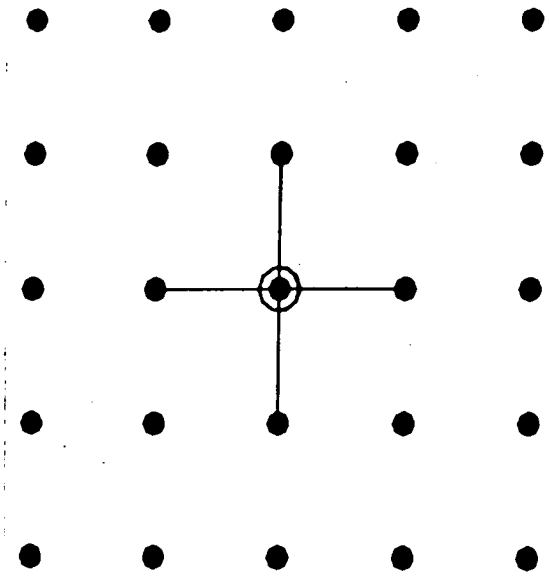


f). Connectivity for four processors.



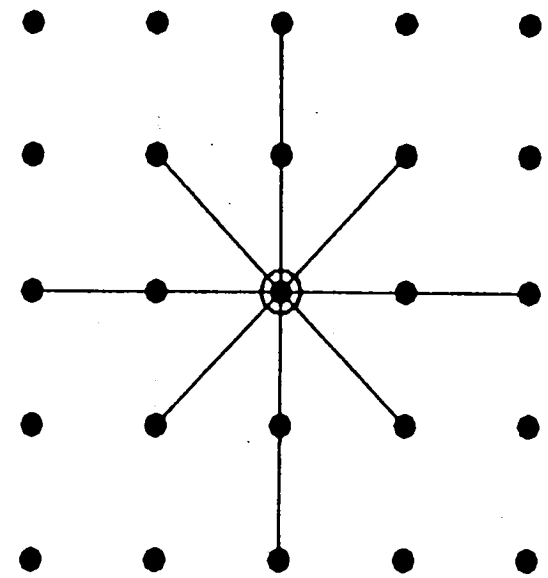
g). Connectivity for sixteen processors.

Figure 6.- Plane stress analysis of a plate.



a). Membrane Problem

$(\nabla^2 u = 0)$



b). Plate Bending Problem

$(\nabla^4 u = 0)$

Figure 7.- Finite difference discretizations.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM- 84514		

4. Title and Subtitle	5. Report Date
THE FINITE ELEMENT MACHINE: An Experiment in Parallel Processing	July 1982
	6. Performing Organization Code 505-33-63-01

7. Author(s)	8. Performing Organization Report No.
O. O. Storaasli and S. W. Peebles, NASA LaRC T. W. Crockett and J. D. Knott, Kentron L. Adams, University of Virginia	
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
NASA Langley Research Center Hampton, VA 23665	
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Memorandum
National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The Finite Element Machine at the NASA Langley Research Center is a prototype computer designed to support parallel solutions to structural analysis problems. This paper describes the hardware architecture and support software for the machine, initial solution algorithms and test applications, preliminary results, and directions for future work.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
parallel processing	Unclassified - Unlimited Subject Category 62

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	18	A02

For sale by the National Technical Information Service, Springfield, Virginia 22161