NASA TM-85710

NASA Technical Memorandum 85710

NASA-TM-85710 19840021474

SOLUTION OF A TRIDIAGONAL SYSTEM OF EQUATIONS ON THE FINITE ELEMENT MACHINE

Susan W. Bostic

MARCH 1984

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

CONTENTS

N84-29543#

SOLUTION OF A TRIDIAGONAL SYSTEM OF EQUATIONS ON THE FINITE ELEMENT MACHINE

INTRODUCTION

The solution of tridiagonal systems of equations on parallel computers has been the topic of many studies over the last ten years. Both direct and iterative solutions have been analyzed. In many cases, the analysis has been made on simulators or "theoretical" machines, in particular the paracomputer. The paracomputer is a theoretical machine which has an unlimited number of processors and no overhead for communication between processors. There has been limited experience in analyzing these solutions on actual parallel computing systems. An experimental hardware/software array processor system in operation at NASA's Langley Research Center, the Finite Element Machine, was used to study the issues involved in implementing these solution techniques.

The two algorithms which were run on the Finite Element Machine are the Accelerated Parallel Gauss method (ref. 1,2) an iterative method, and the Buneman algorithm (ref. 2), a direct method. Some of the issues involved when applying solution techniques in a parallel processing environment include: programming strategy, balancing the workload among the processors, and the amount of communication between the processors. This paper discusses these issues, analyzes the amount and type of overhead associated with the execution of the programs, and presents timing results obtained.

FINITE ELEMENT MACHINE

Hardware

The Finite Element Machine, being developed at NASA Langley Research Center, is an experimental MIMD (multiple instruction, multiple data) system. The architecture is that of an array of microprocessors with each processor capable of being connected to 12 other processors by a local link and each processor also capable of communicating to any other processor via a global bus. The FEM is connected to the outside world by a controller, on which programs are coded, compiled, debugged, and then downloaded to the array. Each processor has a copy of its own program, its own memory, and each can run asynchronously. A block diagram of the FEM is depicted in figure 1. At present eight microprocessors are installed in the array. Each microprocessor consists of three boards, one for processing (CPU) and two for communications (IO-1 and IO-2). The CPU board is a 16 bit microprocessor with 16K of ROM (read only memory), 32K of RAM (randomly accessed memory), and an AM9512 floating point chip (ref. 3).

The Finite Element Machine was initially designed for solution to structural analysis problems by the finite element and finite difference approximation methods. These types of problems have many calculations that can be solved simultaneously so they are ideally suited to parallel computing. Many methods used to solve such problems are time-consuming iterative techniques. The FEM architecture is being studied as to its capability of supporting the efficient solution of other types of problems as well.

Software

The development of new system software is essential to the effective use of the special capabilities of any parallel computer. The Finite Element Machine has custom designed software to provide control and run-time support for programs written for the FEM. An operating system, NODAL EXEC, resides on each processor. One function performed by this system is the definition of special areas called data areas, where data can be stored and saved during a session. Also residing on the processors are PASLIB routines, which are special function procedures which enable the processors to communicate, access data in data areas, and supply support for application programs (ref. 4). FEM ARRAY CONTROL SOFTWARE (FACS) supports program development, program execution, and analysis. FEM commands are implemented as control language procedures and are integrated into the host operating system. Commands are issued interactively at the terminal or submitted as batch stream commands (ref. 5).

By the use of FACS commands, the proper environment for executing a program on FEM is established. The hardware and software is initialized and the array configuration is selected. Data areas are defined and loaded and the program is downloaded.

Programs may be run in one of two modes: synchronous or asynchronous. Both programs discussed here were run in synchronous mode. In this mode, input to each processor is queued in order until it is received. In running in asynchronous mode, only the most recent data are used.

ALGORITHMS

Matching the parallelism of an algorithm to the parallelism of the computer is an essential ingredient in obtaining cost-efficient results. The number of arithmetic operations which can be performed simultaneously determines the parallelism of an algorithm. On a pipeline computer the parallelism is equal to the vector length. On an array processor the parallelism is measured by the number of operations done in parallel. This is not always constant throughout a solution.

The algorithms developed in the past to solve large scientific problems have been mainly designed to run on conventional sequential computers. The number of arithmetic operations involved determined the speed in which problems were solved. The idea of solving problems in parallel introduces new concepts in speed and efficiency. Particularly, the problems of synchronization and communication must be addressed. New algorithms must be developed and old ones re-examined. A brief discussion of tridiagonal systems and the algorithms investigated in the study is summarized below.

Tridiagonal Systems

Many problems being solved in the scientific community involve the solution of tridiagonal systems of equations such as the system solved in this study of the form (see fig. 2),

$$AX = Y$$

where X and Y are the column vectors of $\{x_i\}$ and $\{y_i\}$, $i = 1,2,\ldots,n$ with the $\{y_i\}$ given. The matrix A has diagonal elements $\{d_i\}$, $i = 1,2,\ldots,n$ (in this case $\{d_i\}$ is taken to be unity), subdiagonal elements $\{\ell_i\}$, $i = 2,3,\ldots,n$, and superdiagonal elements $\{u_i\}$, $i = 1,2,\ldots,n-1$. These systems often result from finite-difference approximations to second order differential equations as in Helmholtz, Laplace, Poisson, and diffusion equations (ref. 1). Both direct and iterative methods are used to solve such systems. While iterative methods seem to parallelize best, direct methods are often the best choice for efficiency in the overall problem solution. Thus investigations are needed of both methods. Although many techniques have been designed to solve a tridiagonal system of equations, most of these have been developed to run on sequential computers and are not able to be applied efficiently on a parallel computer. In many methods the values of the latest terms depend on previously computed terms which make them inherently sequential. Therefore, new approaches are being studied to introduce parallelism into tridiagonal solution methods. In practice, many such systems may be solved where A would stay the same and only the Y would vary.

Accelerated Parallel Gauss

The Accelerated Parallel Gauss Method was developed by Heller, Stevenson, and Traub (ref. 1). The method is an improved version of Traub's Parallel Gauss algorithm and is outlined in the appendix.

The classical Gaussian elimination algorithm for tridiagonal system may be stated as follows.

1. (Factor $A = (I + L)D(I + R)$.)

   Let $d_1 = 1$. For $i = 2,3,\ldots,n$, let $d_i = 1 - \ell_i u_{1i-1}/d_{i-1}$.

2. (Solve $(I + L)f = c$.)

   Set $\ell_i = a_i/d_{i-1}$ for all $i > 1$. Let $f_1 = c_1$. For $i = 2,3,\ldots,n$, let $f_i = c_i - \ell_i f_{i-1}$.

3. (Solve $(I + R)x = D^{(-1)}f$.)

   Set $r_i = u_i/d_i$ and $g_i = f_i/d_i$ for all i. Let $(x_n = g_n)$. For $i = n - 1, n - 2,\ldots,1$, let $x_i = g_i - r_i x_{i-1}$.

This classical algorithm is sequential because each calculation depends on previously calculated results. This method was converted into an iteration method by successively approximating the d, then the f, and finally the x. For example,

$$d_i^{(i)} = 1 - \ell_i b_{i-1}/d_{i-1}^{(i-1)} \quad \text{for all } i > 1,$$

where the superscripts denote the iteration step (appendix).

By using successive approximation we increase the amount of computation. Although the number of arithmetic operations increases, the fact that many computations take place in parallel can make the algorithm more time efficient on some parallel computers for a given number of equations. As implemented on FEM, all processors are busy all the time and all iterations are carried out in parallel. The APG method is efficient for certain types of problems; however, under certain conditions this algorithm is susceptible to round-off errors and the rate of convergence is sometimes slow.

The APG method consists of three steps, solving for d, solving for f, and then using back substitution to solve for x. The number of iterations for each step is determined by the convergence criteria (ref. 2).

Buneman Algorithm

The Buneman algorithm, developed by O. Buneman (ref. 2), is a variant of cyclic reduction and is outlined in the appendix. A direct solution is found by combining adjacent terms to obtain a relation between every other term, every fourth term, every eighth term, etc. For a system of N equations, $\log 2(N) - 1$ levels of reduction are carried out until the relation is between known values (the boundary conditions). All unknowns are then found by back substitution. The algorithm is numerically stable and the answer is correct to within the round-off error. The answer is obtained in a predetermined number of steps.

The application run on the FEM assigned several equations to each processor. The ratio of the amount of idle time and the communication time to the amount of computation time decreased as the size of the problem was increased.

PROGRAM DEVELOPMENT

Problem Mapping

One of the basic issues facing a programmer of a parallel computer is the mapping of the problem to the particular architecture of the computer. The workload should be equally distributed among the processors and the amount of communication should be minimized. In the APG case each processor only has to communicate with at most two of its nearest neighbors. In the Buneman case, as the number of processors increases, so does the number of processors one must communicate with increase. The test cases were run on up to eight processors and each processor was connected to every other processor by both a local link and the global bus.

The overhead of communication on the Finite Element Machine is high compared to the computation speed. This is a function of the software which provides input buffers and allows up to 255 words and several types of data to be exchanged.

Mapping of the Accelerated Parallel Gauss Method

The primary consideration in the program design was to balance the communication with the computation. For this reason, the equations were ordered in a way that minimizes the time a processor waits for a transfer.

For the APG method (see appendix), the initial values for the solutions of the odd equations of d and f are known; hence, these equations are solved first. They are solved in descending order so that the last value is the one that must be passed to its neighbor. The equations were arranged in such a way as to compute the last odd first and the first even last.

At every iteration, each processor has at most one receive and one send to its nearest neighbors. By increasing the number of equations to be solved in each processor, the computation time increases while the transfer time stays the same. For the algorithm to be most efficient, enough computation must be assigned to each processor so that time is not spent waiting for a transfer. Figure 3 shows a mapping of 20 equations on four processors.

For the iterations on d and f, the first processor only sends and the last processor only receives. This is reversed for the x iterations. This test case is extensible to a larger set of processors solving a larger system of equations. The order of solving equations in a system and the maximum number of equations per processor were issues addressed in the implementation of this method.

Mapping for Buneman Algorithm

The Buneman algorithm is a parallel direct method and in contrast to iterative methods, the solution is found in a predetermined number of steps. For a given number of equations, the same total number of calculation steps is independent of the number of processors. The number of transfers, however, varies with the number of processors. For an efficient approach, several equations should be assigned to each processor so that the amount of computation in each processor outweighs the communication tasks.

During the reduction stage, each processor sends results to its nearest neighbor (with the first only sending to the right and the last only sending to the left) until there is only one equation to solve in each processor. From this level, the number of working processors is divided in half until the reduction is down to one equation. During the back substitution stage, the communication is reversed. Each processor only has to receive one value. From that stage on, the processor has all the information it needs to solve the equations assigned to it.

The application program written for FEM assigned a middle processor to compute the last equation. This minimized the number and distance of the transfers. Figure 4 shows the mapping of 16 equations on four processors.

RESULTS

Figure 5 gives actual timing results for the two algorithms implemented on FEM for Nth order test problems. Tables 1 and 2 give timing results for the two algorithms where parallel efficiency and speedup are defined as:

$$\text{Parallel efficiency} = \frac{\text{Time to run on one processor}}{\text{NPE * time to run on NPE processors}}$$

$$\text{Speedup} = \frac{\text{Time to run on one processor}}{\text{Time to run on NPE processors}}$$

NPE = number of processors

Execution Statistics for APG

Where:

K = number of iterations for d

L = number of iterations for f

M = number of iterations for x

N = number of equations in system

K=50, L=100, M=100, N=100

Number of processors	Milliseconds	Efficiency	Speedup
1	33215		
2	17436	0.9525	1.9
4	9758	.851	3.4
8	6129	.6774	5.4

Table 1

Execution Statistics for Buneman

Where:

N=128

Number of processors	Milliseconds	Efficiency	Speedup
1	3982 (est)		
2	2238	0.89	1.78
4	1211	.82	3.29
8	672	.74	5.92

Table 2

ESTIMATED TIMING RESULTS

An estimated speedup was computed by analytically modeling the APG and Buneman algorithm programs as run on FEM. These formulas, summarized in the following section, are not expected to give exact results as each formula only takes into consideration the number of arithmetic operations and the number of transfers. Embedded in the program is also a decision-making code which includes tests to determine whether a value is to be sent or received. Table 3 compares the expected speedup of APG when run on 2, 4, and 8 processors as compared to the estimated and actual time to run on one processor. Table 4 gives similar results for the Buneman algorithm. Figure 6(a) shows plots of actual versus estimated times it would take to compute the equations on a sequential computer with the same speed as the FEM. Figure 6(b) shows the actual times compared to estimated times with the communication time for each processor included.

Accelerated Parallel Gauss

The total time for an APG solution is estimated to be

$$N[2*T_a(K + L + M) + 8*T_a] + 2T_t(K + L + M)$$

where

$T_a = 500$	Time for an arithmetic operation in microseconds
$T_t = 1870$	Time for a transfer (either a send or a receive) in microseconds
$N = 100$	Number of equations in system
$K = 50$	Number iterations to solve for d
$L = 100$	Number iterations to solve for f
$M = 100$	Number iterations to solve for x

(The 8 arithmetic operations include normalizing ℓ and u, initializing y and x, and computing ℓ*u, ℓ, g, and r. The two transfers are for a send and a receive for each processor, even though some only send or receive.)

Estimated Solution Time in milliseconds for the baseline APG problem (N=100) is

On one processor:	25400
Two processors:	13167.5
Four processors:	7285
Eight processors:	4531.5

	Actual speedup vs estimated speedup		
	Two processors	Four processors	Eight processors
Estimated speedup	$\frac{25400}{13167.5} = 1.929$	$\frac{25400}{7285} = 3.486$	$\frac{25400}{4531.5} = 5.618$
Actual speedup	$\frac{33215}{17436} = 1.905$	$\frac{33215}{9758} = 3.404$	$\frac{33215}{6129} = 5.419$

Table 3

Buneman Algorithm

The total time for the Buneman Solution is estimated from

$$N_e * (12621) + N_{s10} * T_{s10} + N_r * T_{r10} + N_e * T_{bs} + N_{s1} * T_{s1} + N_{r1} * T_{s1}$$

Where:	N_e =	Number of equations to solve in each processor (To solve a system of 128 equations, 255 equations are solved)
	N_{s10} =	Number of sends of ten words each
	N_{r10} =	Number of receives of ten words each
	N_{s1} =	Number of sends of one word each
	N_{r1} =	Number of receives of one word each
	T_{s10} = 3910	Time for a send of ten words
	T_{r10} = 9970	Time for a receive of ten words
	T_{s1} = 1870	Time for a send of one word
	T_{r1} = 6160	Time for a receive of one word

Estimated solution time in microseconds for the baseline Buneman problem (N=128)

One processor:	3981.8
Two processors:	2062.5
Four processors:	1132.4
Eight processors:	623.5

	Actual speedup vs estimated speedup		
	Two processors	Four processors	Eight processors
Estimated speedup	$\frac{3981.8}{2062.5} = 1.9305$	$\frac{3981.8}{11324} = 3.516$	$\frac{3981.8}{623.5} = 6.386$
Actual speedup	$\frac{3981.8}{2238} = 1.779$	$\frac{3981.8}{1211} = 3.288$	$\frac{3981.8}{672} = 5.925$

Table 4

CONCLUSIONS

Two algorithms for the solution of tridiagonal systems were implemented on the Finite Element Machine. Many variables influence the speedup that can be achieved for a given algorithm on the Finite Element Machine. Some of the issues addressed were: the size of the problem, the number of processors for a given case, the ratio of the computation time to the communication time, and the balancing of the workload among the processors. The amount and type of overhead was analyzed.

Execution statistics obtained by running these test cases on FEM were presented. By allowing computations to take place in parallel, the time to solve a large system of equations can be greatly reduced. Although the rate of speedup tends to decrease as more processors are added, the cost-effectiveness of the addition of processors needs to be taken into consideration. In some cases, additional processors may be necessary to achieve timely or cost effective results.

For problems where many tridiagonal systems need to be solved, the most efficient method may be for each processor to solve its own system with little or no communication with its neighbors. In a parallel processing environment there is still the problem of communicating with the controller to download data and upload results.

The problems summarized in this study should serve as benchmarks in evaluating other algorithms as well as making comparisons of timing, changes in the system, new software, different modes of operation, and various ratios of computation to communication.

APPENDIX

ACCELERATED PARALLEL GAUSS ALGORITHM

BUNEMAN ALGORITHM

ACCELERATED PARALLEL GAUSS* ALGORITHM

1. Let $d_o^{(k)} = 1$ for all k. Then for $k = 1, 2, \ldots \hat{K}$

$$d_i^{(k)} = 1 - \ell_i u_{i-1} \Big/ d_{i-1}^{(k-1)}; \quad i \text{ odd}$$

$$d_i^{(k)} = 1 - \ell_i u_{i-1} \Big/ d_{i-1}^{(k)}; \quad i \text{ even}$$

2. Define $\ell_i = u_i \Big/ d_{i-1}^{(k)}$ for $i > 0$. Let

$$f_i^o = y_i \quad \text{for all } i$$

$$f_o^{(k)} = y_o \quad \text{for all } k$$

Then for $k = 1, 2, \ldots \hat{L}$

$$f_i^{(k)} = y_i - \ell_i f_{i-1}^{(k-1)}; \quad i \text{ odd}$$

$$f_i^{(k)} = y_i - \ell_i f_{i-1}^{(k)}; \quad i \text{ even}$$

3. Define

$$q_i = f_i^{(\hat{L})} \Big/ d_i^{(\hat{K})} \quad \text{for all } i$$

$$r_i = u_i \Big/ d_i^{(\hat{K})} \quad \text{for all } i < N$$

4. Set

$$x_i^{(o)} = f_i^{(\hat{L})} \Big/ d_i^{(\hat{K})} \quad \text{for all } i$$

and

$$x_N^{(k)} = gN \quad \text{for all } k$$

*D. E. Heller, D. K. Stephenson, and J. F. Traub, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Then for $k = 1, 2, \ldots \hat{M}$

$$x_i^{(k)} = q_i - r_i \, x_{i+1}^{(k-1)}; \quad i \text{ even}$$

$$x_i^{(k)} = q_i - r_i \, x_{i+1}^{(k)}; \quad i \text{ odd}$$

$x_i^{(\hat{M})}$ are then an approximation to the true solution x.

BUNEMAN ALGORITHM*

1. Define for $i = 0, 1, \dots N$

$$\ell_i^{(o)} = l_i \qquad p_i^{(o)} = 0$$

$$d_i^{(o)} = d_i \qquad q_i^{(o)} = y_i$$

$$u_i^{(o)} = u_i$$

Then for $K = 0, 1, \dots n-1$

$$\ell_i^{(k+1)} = \ell_{i-2^k}^{(k)} \ell_i^{(k)} d_{i+2^k}^{(k)}$$

$$d_i^{(k+1)} = \ell_i^{(k)} d_{i+2^k}^{(k)} u_{i-2^k}^{(k)} + \ell_{i+2^k}^{(k)} d_{i-2^k}^{(k)} u_i^{(k)} - d_{i-2^k}^{(k)} d_i^{(k)} d_{i+2^k}^{(k)}$$

$$u_i^{(k+1)} = d_{i-2^k}^{(k)} u_i^{(k)} u_{i+2^k}^{(k)}$$

$$p_i^{(k+1)} = p_i^{(k)} + \left(q_i^{(k)} - \ell_i^{(k)} p_{i-2^k}^{(k)} - u_i^{(k)} p_{i+2^k}^{(k)}\right) \Big/ d_i^{(k)}$$

$$q_i^{(k+1)} = \ell_i^{(k)} d_{i+2^k}^{(k)} q_{i-2^k}^{(k)} + d_{i-2^k}^{(k)} u_i^{(k)} q_{i+2^k}^{(k)} - \left(\ell_i^{(k)} d_{i+2^k}^{(k)} u_{i-2^k}^{(k)} + \ell_{i+2^k}^{(k)} d_{i-2^k}^{(k)} u_i^{(k)}\right) p_i^{(k+1)}$$

with $i = j * 2^{k+1}$, $j = 0, 1, \dots 2^{n-k+1}$

*O. Buneman, Stanford University Institute for Plasma Research.

2. $d_{2^n}^{(n+1)} = \ell_{2^n}^{(n)} d_{2^{n+1}}^{(n)} u_o^{(n)} \ell_{2^{n+1}}^{(n)} d_o^{(n)} u_{2^n}^{(n)} - d_o^{(n)} d_{2^n}^{(n)} d_{n+1}^{(n)}$

$$p_{2^n}^{(n+1)} = p_{2^n}^{(n)} + \left(q_{2^n}^{(n)} - \ell_{2^n}^{(n)} p_o^{(n)} - u_{2^n}^{(n)} p_{2^{n+1}}^{(n)}\right) \Big/ d_{2^n}^{(n)}$$

$$q_{2^n}^{(n+1)} = \ell_{2^n}^{(n)} d_{2^{n+1}}^{(n)} q_o^{(n)} + d_o^{(n)} u_{2^n}^{(n)} q_{2^{n+1}}^{(n)} - \left(\ell_{2^n}^{(n)} d_{2^{n+1}}^{(n)} u_o^{(n)} + \ell_{2^{n+1}}^{(n)} d_o^{(n)} u_{2^n}^{(n)}\right) p_{2^n}^{(n+1)}$$

This completes the reduction stage.

3. In the back-substitution stage:

$$x_{2^n} = p_{2^n}^{(n+1)} + q_{2^n}^{(n+1)} \Big/ d_{2^n}^{(n+1)}$$

Then,

$$x_o = p_o^{(n)} + \left(q_o^{(n)} - u_o^{(n)} x_{2^n}\right) \Big/ d_o^{(n)}$$

$$x_{2^{n+1}} = p_{2^{n+1}}^{(n)} + \left(q_{2^{n+1}}^{(n)} - u_{2^{n+1}}^{(n)} x_{2^n}\right) \Big/ d_{2^{n+1}}^{(n)}$$

Finally, for $k = n-1, n-2, \ldots 1, 0,$

$$x_i = p_i^{(k)} + \left(q_i^{(k)} - \ell_i^{(k)} x_{i-2^k} - u_i^{(k)} x_{i+2^k}\right) \Big/ d_i^{(k)}$$

with $i = 2^k,\ 3 \cdot 2^k \ldots$

REFERENCES

1. Heller, D. E.; Stevenson, D. K.; and Traub, J. F.: Accelerated Iterative Methods for the Solution of Tridiagonal Systems on Parallel Computers. Journal of the Association for Computing Machinery, vol. 23, no. 4, October 1976, pp. 636-654.

2. Grosch, C. E.: Performance Analysis of Poisson Solvers on Array Computers. Supercomputers, Infotech International Limited, Maidenhead, Berkshire, England, vol. 2, 1979, pp. 147-181.

3. Storaasli, O. O.; Peebles, S. W.; Crockett, T. W.; Knott, J. D.; and Adams, L.: The Finite Element Machine: An Experiment in Parallel Processing. NASA TM-84514, July 1982.

4. Crockett, T. W.: PASLIB Programmer's Guide for the Finite Element Machine. NASA CR-172281, NASA Langley Research Center, Hampton, Virginia, December 1983.

5. Knott, J. D.: FACS-Fem Array Control Software Users Guide. NASA CR-172189, NASA Langley Research Center, Hampton, Virginia, August 1983.

6. Hockney, R. W.; and Jesshope, C. R.: Parallel Computers. Adam Hilger Ltd., Bristol, Great Britain, 1981.

NASA
L-83-10,247

FINITE ELEMENT MACHINE BLOCK DIAGRAM

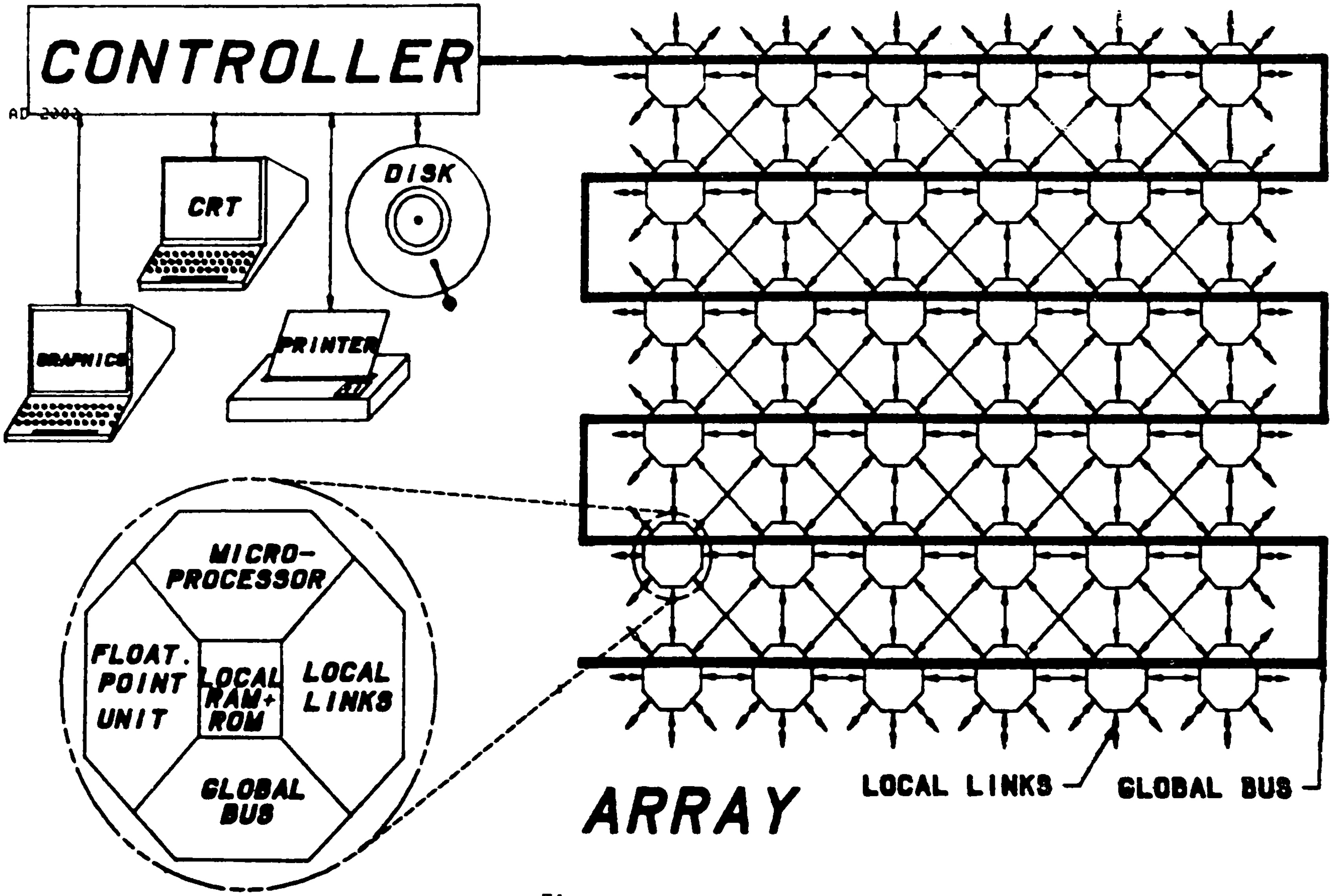


Figure 1. Finite Element Machine block diagram.

TRIDIAGONAL SYSTEM

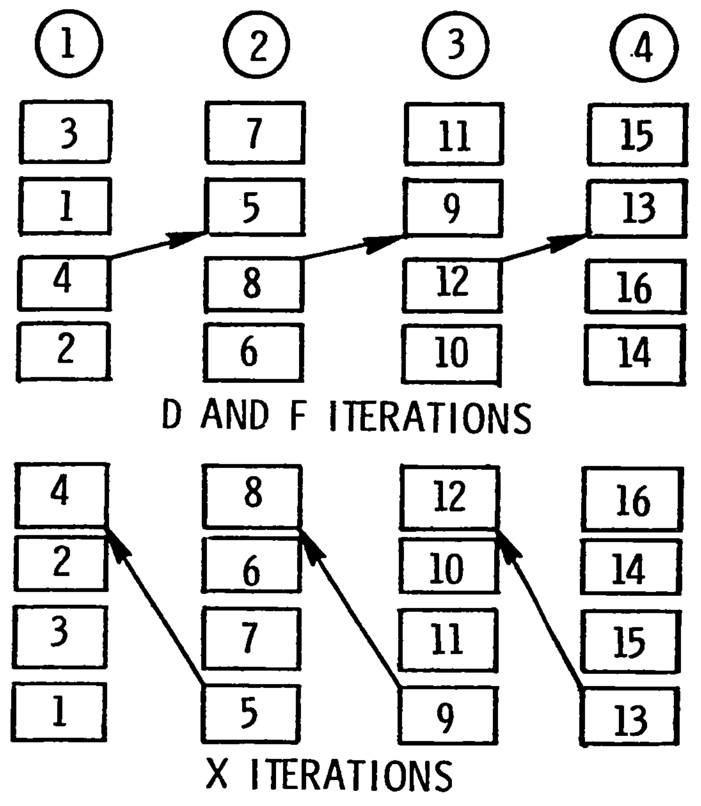
$$Ax \equiv \begin{bmatrix} 1 & u_1 & & \bigcirc \\ l_2 & 1 & u_2 & \\ \bullet & \bullet & \bullet & \\ \bullet & \bullet & \bullet & \\ \bigcirc & l_{n-1} & 1 & u_{n-1} \\ & & l_n & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \bullet \\ \bullet \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \bullet \\ \bullet \\ y_{n-1} \\ y_n \end{bmatrix}$$

DIAGONAL ELEMENTS = 1
li = LOWER DIAGONAL ELEMENTS
ui = UPPER DIAGONAL ELEMENTS

Figure 2. Tridiagonal system of equations.

ACCELERATED PARALLEL GAUSS *

ITERATIVE METHOD, ITERATIONS CARRIED OUT IN PARALLEL
ALL TRANSFERS ARE TO THE NEAREST NEIGHBOR
ALL PROCESSORS ARE BUSY ALL THE TIME



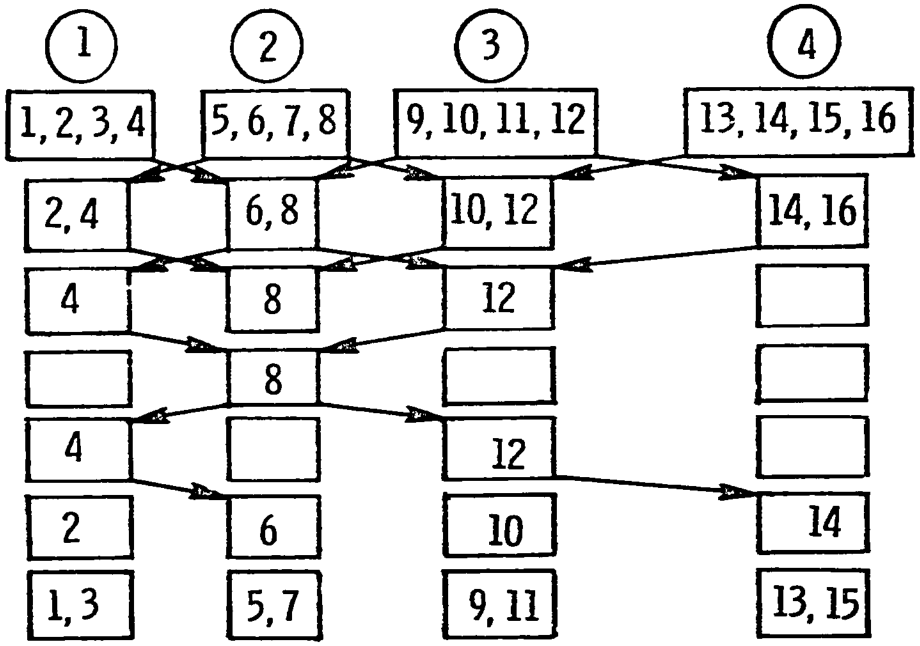
MAPPING OF 16 EQUATIONS ON FOUR PROCESSORS

*HELLER, STEVENSON AND TRAUB,
CARNEGIE - MELLON UNIVERSITY

Figure 3. Mapping of 16 equations on four processors (APG).

BUNEMAN*ALGORITHM

DIRECT METHOD, VARIANT OF CYCLIC REDUCTION
SOLUTION FOUND IN A FINITE NUMBER OF STEPS
SOLUTION IS EXACT (WITHIN ROUNDOFF ERROR)



MAPPING OF 16 EQUATIONS ON 4 PROCESSORS

*O. BUNEMAN, STANFORD UNIVERSITY
INSTITUTE FOR PLASMA RESEARCH

Figure 4. Mapping of 16 equations on four processors (Buneman).

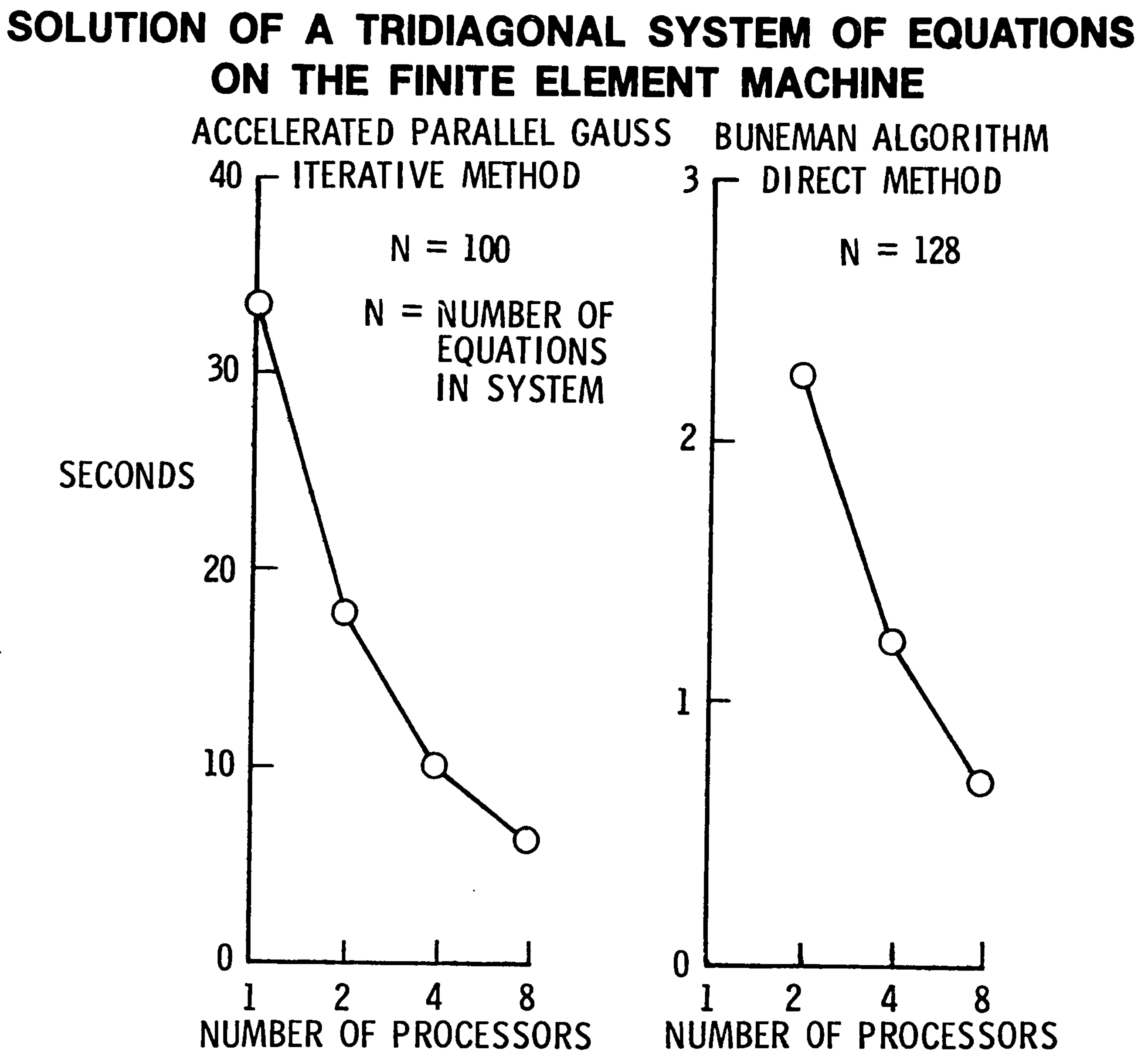
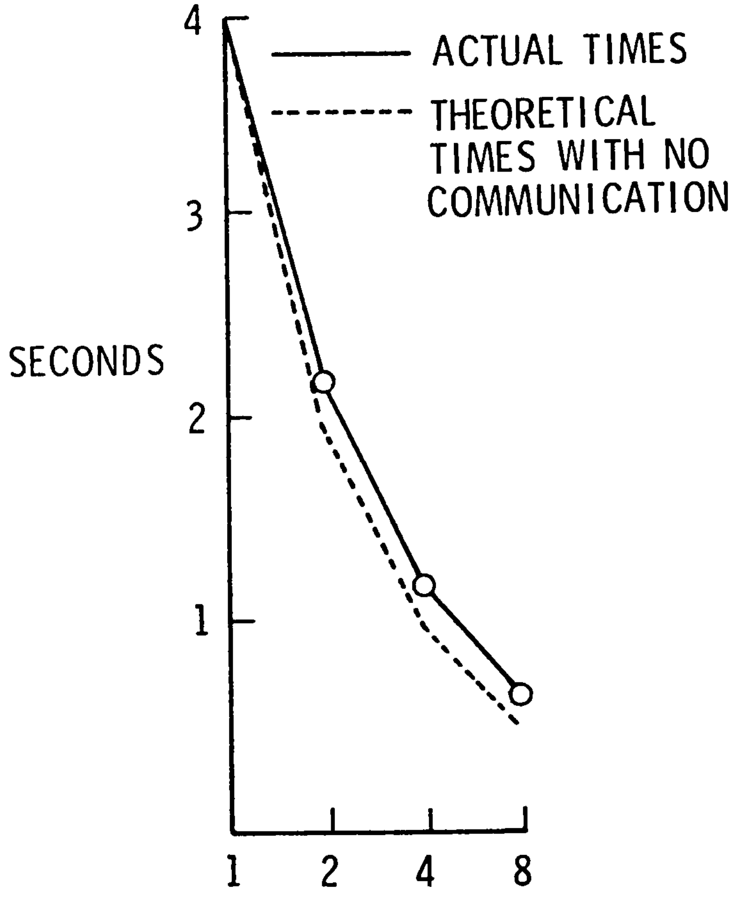
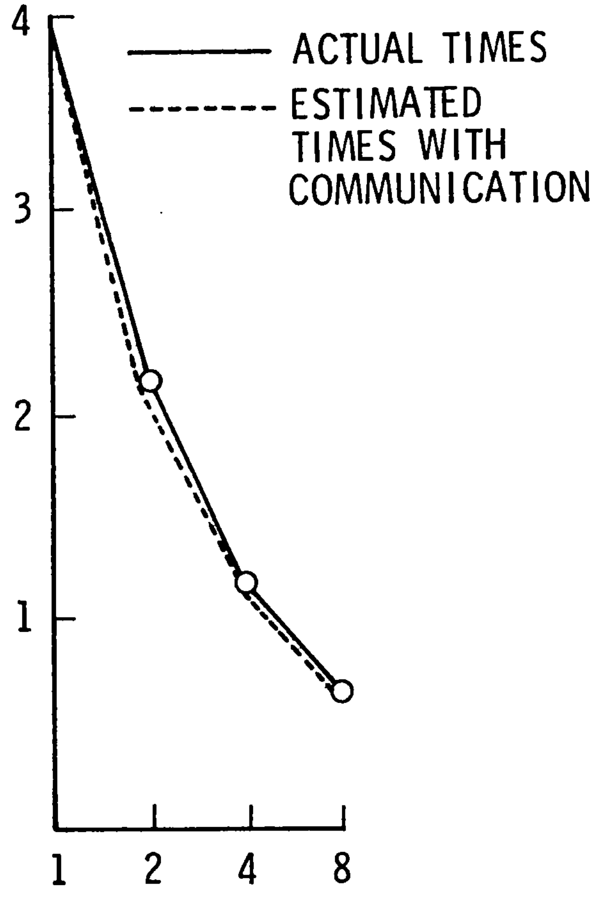


Figure 5. Timing results for the two algorithms implemented on FEM.

BUNEMAN ALGORITHM





NUMBER OF PROCESSORS

Figure 6(a). Actual times versus theoretical times with no communication.

Figure 6(b). Actual times versus estimated times with communication.

1. Report No. NASA TM-85710	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle SOLUTION OF A TRIDIAGONAL SYSTEM OF EQUATIONS ON THE FINITE ELEMENT MACHINE		5. Report Date March 1984
		6. Performing Organization Code 505-37-33-01
7. Author(s) Susan W. Bostic		8. Performing Organization Report No.
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665		10. Work Unit No.
		11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		13. Type of Report and Period Covered Technical Memorandum
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

Two parallel algorithms for the solution of tridiagonal systems of equations were implemented on the Finite Element Machine. The Accelerated Parallel Gauss method, an iterative method, and the Buneman algorithm, a direct method, are discussed and execution statistics are presented.

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

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 23	22. Price A02