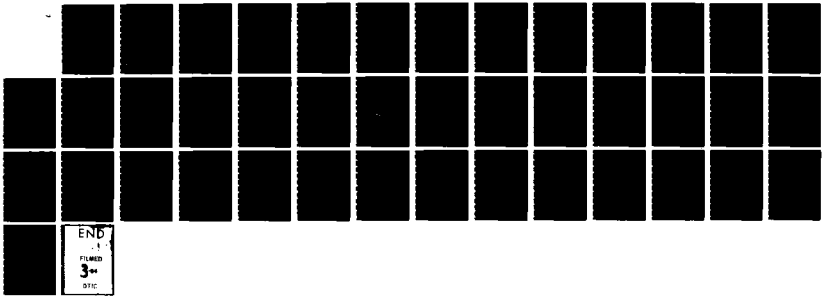
AD-A137 767 DESIGN OF ARRAY PROCESSOR SOFTWARE FOR NONLINEAR STRUCTURAL ANALYSIS(U) ARIZONA UNIV TUCSON DEPT OF AEROSPACE AND MECHANICAL ENGINEERING N SARIGUL ET AL. 1/1

UNCLASSIFIED DEC 83 TR-10 N00014-75-C-0837 F/G 9/2 NL

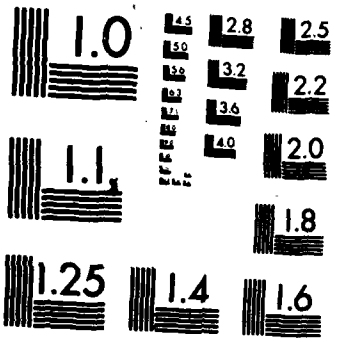


END

FILMED

3-84

DTIC



N00014-75-C-0837

DESIGN OF ARRAY PROCESSOR SOFTWARE FOR NONLINEAR STRUCTURAL ANALYSIS

N. Sarigul, M. Jin,
R. Kolar, and H. Kamel
UNIVERSITY OF ARIZONA
Aerospace & Mechanical Engineering Department
Tucson, Arizona 85721

~~November 17~~, 1983

Technical Report No. 10

Approved for public release, distribution unlimited

Department of the Navy
Office of Naval Research
Bandelier Hall West
University of New Mexico
Albuquerque, New Mexico 87131

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report No. 10	2. GOVT ACCESSION NO. AD A137767	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DESIGN OF ARRAY PROCESSOR SOFTWARE FOR NONLINEAR STRUCTURAL ANALYSIS		5. TYPE OF REPORT & PERIOD COVERED Technical 6/82-12/83 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Sarigul N., Jin M., Kolar R. and Kamel H.A.		8. CONTRACT OR GRANT NUMBER(s) N0001475C0837
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Arizona AME Dept. Bldg. 16, IGEL Room 204 Tucson, Arizona 85721		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 064-531/12-17-75
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Bandelier Hall West 87131 University of New Mexico, Albuquerque		12. REPORT DATE [redacted]/12/83 13. NUMBER OF PAGES 38
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same as above.		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Presented at the ASME Winter Annual Meeting, Boston, MA, November 13-18, 1983.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Array processors, finite element methods, nonlinear analysis, software engineering.

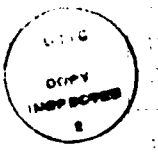
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report discusses ongoing research on the solution of large-scale nonlinear structural problems using a 32-bit minicomputer with an attached 64-bit array processor that communicate via a common memory interface. This configuration is typical of what is seen representative of future work stations with attached specialized processors. A user-oriented software package has been designed to allow the use of the given computer configuration by a typical en-

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

engineer or a scientific user without a detailed knowledge of the operation of the array processor or/and the complex data handling necessary to create and manipulate the data associated with the solution of large problems. The software was then used to implement typical building blocks of a nonlinear finite element code, and performance measurements were taken. Several test examples are considered using 3-D beam finite elements and the Newton Raphson solution scheme. The array processor could not be utilized as yet, due to the lack of the proper vendor software. Hence, a simulator was designed to predict the performance of the software. The simulator was based on reliable time measurements obtained from previous work with the same array processor, using a 16-bit host computer, as well as experiments with the current 32-bit host computer.



DESIGN OF ARRAY PROCESSOR SOFTWARE FOR NONLINEAR STRUCTURAL ANALYSIS

Sarigul N., Jin M., Kolar R. and Kamel H.A.

The University of Arizona
Aerospace and Mechanical Engineering Department

A-1

ABSTRACT

This paper presents ongoing research on the solution of large-scale nonlinear structural problems using a 32-bit minicomputer with an attached 64-bit array processor that communicate via a common memory interface. This configuration is typical of what we see as representative of future work stations with attached specialized processors. A user-oriented software package has been designed to allow the use of the given computer configuration by a typical engineer or a scientific user without a detailed knowledge of the operation of the array processor or/and the complex data handling necessary to create and manipulate the data associated with the solution of large problems. The software was then used to implement typical building blocks of a nonlinear finite element code, and performance measurements were taken. Several test examples are considered using 3-D beam finite elements and the Newton Raphson solution scheme. The array processor could not be utilized as yet, due to the lack of the proper vendor software. Hence, a simulator was designed to predict the performance of the software. The simulator was based on reliable time measurements obtained from previous work with the same array processor, using a 16-bit host computer, as well as experiments with the current 32-bit host computer.

1. INTRODUCTION

The numerical solution of nonlinear problems in structural mechanics requires the extensive use of modern digital computers (1). The demand it places on computer systems is such that the need for increased performance is constantly present. Today, state of the art computer systems include high performance vector machines as well as less expensive devices, such as array processors. Apart from the use of such array processors in conjunction with minicomputers, we foresee the increased marketing of powerful work stations with such attached devices. Sophisticated hardware of this kind usually requires complex programming techniques, if the full potential of the system is to be realized. In general applications, users object to the amount of effort needed to implement code in this fashion. One solution is to provide with the system a higher level language compiler (e.g. FORTRAN) which allows the development of code directly on the device (2,3). While this is a convenient solution, it does not provide optimum utilization of the hardware. Furthermore, it is often required to rewrite some key routines in order to approach the potential of the system. Whereas, the relative inefficiency of this approach is tolerable on a powerful machine, such as the CRAY2 or CYBER 205, it is our opinion that a more efficient approach is needed to use the less potent array processors. Refs. (11-12) report examples of the use of array processors in finite element analysis for restructured existing codes. Ref. (12) is particularly

interesting since it does demonstrate most impressive performance figures, using the specialized FORTRAN compiler. In the opinion of the authors, the machine used has both scalar and vector processing hardware, and cannot be directly compared to the attached processor examined in this paper, which has only vector functions, and is considerably less expensive.

In addition to the complexity of vector processing, another equally important issue is that of handling large data bases associated with large problems. Scientific and engineering users typically employ FORTRAN and would prefer to avoid creating complex data structures on disk. The most popular approach today appears to be the use of virtual memory systems which alleviate the need for out-of-core programming, but not necessarily system initiated disk I/O operations. Here again, convenience takes precedence over efficiency. And, although state of the art virtual systems perform relatively well, they are not, in our view, well suited for extensive computations. This is particularly true if vector devices are employed and where the flow of data from and to backing storage is an important issue (4-9).

In this paper, we propose the design of user-oriented software to support the solution of large problems by engineers and scientists using a 64-bit array processor which shares memory with a 32-bit minicomputer. The concept proposed would alleviate much of the

problems encountered by the applications programmer and the researcher. It provides the user with tools to help in the creation and manipulation of large matrices using the hypermatrix scheme. Reference (10) studies a similar scheme in the case of the CDC STAR-100 super computer.

Once the basic matrices for a computation have been established, a number of processors permit their use in complex computations. After experimentation with this approach, it appears that the availability of such processors also in subroutine form may be desirable for certain complex algorithms. Some results from the solution of 3-D frame structures using finite element analysis are included to validate the proposed software and measure its performance. The results presented here are of a preliminary nature. Further refinements of these concepts are anticipated.

NOMENCLATURE

U : Total number of unknowns in the problem.

S : Size of the submatrix.

NEL : Total number of elements.

NND : Total number of nodes.

b : Half bandwith.

B : Number of submatrices/half row.

P : Total number of partitions.

HC32 : 32-bit minicomputer.

AP64 : 64-bit array processor.

2. THE COMPUTER CONFIGURATION

The computer system used in the research consists of a low-end 32-bit minicomputer and a 64-bit Array Processor. The minicomputer may have an address space as large as 16MB and has a data bus with a maximum speed of (26.5 MBps). The host and the array processor are connected via a high speed common memory interface which minimizes the input/output transfer time between them (Fig. 1). The details of the computer configuration are given in Ref.(9).

3. SOFTWARE DESIGN

A simple approach to the design of general purpose structural analysis software to handle large problems using basic hypermatrix operations is proposed and developed. For the purpose of comparison and to assess the effectiveness of the array processor in speeding up the computation, the same algorithms are implemented on the minicomputer alone, as well as on the minicomputer/array processor

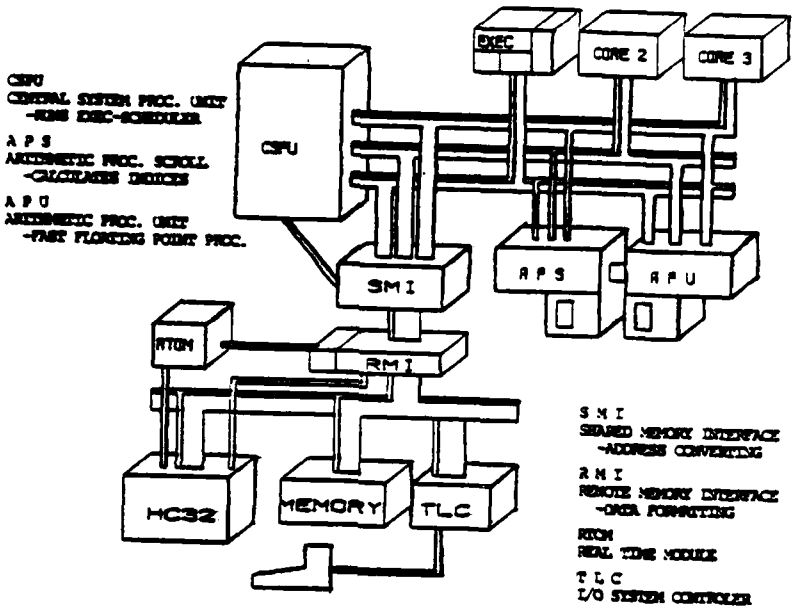


Figure 1 Basic Structure of the Computer Configuration. (AP64+HC32).

combination.

As mentioned before, two conflicting factors must be considered in software development; namely, convenience and efficiency. The first factor is important if the system is to be used by engineers for research and for applications software development. The second factor is equally important if the analysis is to be timely and economical. Both efficiency and convenience are considered, to a certain extent, in the design of the software discussed herein.

Attached (array) processors have an impressive cost-performance ratio (9). Although it is possible to program an array processor directly by using a special (machine) programming language, this

should be avoided as much as possible in applications. In the development of the proposed software, we have used the manufacturer supplied software; although, one might, in the future, enhance it by writing additional routines. From the point of view of the software user, the presence of the array processor is made as transparent as possible. According to this approach, the user exercises the system in two modes. During the first stage, the basic matrices required for the computation are established using a special subroutine library, which constitutes the first part of the software. In the second stage, these matrices are used to perform the required computations, using canned processors which form the second part of the proposed system. In an iterative or repetitive situation, such as that encountered in nonlinear computations, the two phases are entered alternately throughout the process. For time critical operations, it is possible, in principle, to merge several primary processors into a more complex one. The best way to do this is to provide the processors also in subroutine form, as part of a higher level library.

The software uses the hypermatrix representation, which is, in our view, an ideal candidate for array (vector) processing. A reasonably simple data structure is used, to avoid unnecessary complications at this stage. Nevertheless, the system is designed to handle 10000 degrees of freedom. For efficient operation, a half bandwith of 800

to 850 may be achieved, using double precision, with a 3 to 4 MB control memory.

Time consuming computational procedures, suited for vector operations are transferred to the array processor. In addition, to minimize the overhead time associated with minicomputer/array processor communication, calls to the array processor are performed whenever possible, as group calls, Vector Function Control Blocks vendor software terminology.

In summary, the software package consists of two main parts, a user library and the matrix processors. The first is loaded with the user program and aids in the creation and manipulation of the data base. The second part is a collection of independent programs to be invoked by the user, or by a suitable executive, in the correct order, to perform a specific computational sequence.

3.1 STORAGE OF MATRICES:

Each matrix is stored in hypermatrix form and identified by a unique name consisting of four characters. Each hypermatrix is stored in the form of two disk files: a sequential access file, in which directory information is stored; and a random access file, in which only the nonzero submatrices are stored. During the execution of a matrix processor, provisions are made for the storage of directory

information for three hypermatrices and three associated submatrices. In other words, each processor may have as many as three active operands at one time. Before an operation involving a matrix is performed, its directory is transferred into the core. After completion of the operation, the updated directory is written back on disk.

The directory information for a hypermatrix includes the name of the matrix, the number of row and column partitions and the total number of rows and columns. In addition, it contains a list of the number of columns per column partition, a list of the number of rows per row partition, a list of the number of nonzero submatrices per row partition, and another giving the number of the first row relative to the complete matrix within a row partition and a similar list for column partitions. Two auxiliary arrays are also present, giving the location of each nonzero submatrix within the matrix and the record number at which it is stored within the random access file. The maximum allowable submatrix size is currently 50 by 50. Due to the current core limitation on the system, the number of nonzero submatrices is limited to 3000. The system presently allows a maximum of 200 partitions, permitting hypermatrices of the order of 10000 by 10000, with an average (half) bandwidth of 750.

3.2 USER LIBRARY:

The user FORTRAN subroutine library is a tool for the creation and manipulation of hypermatrices. Via the library, the user may access matrix files, extract and deposit data, as well as manipulate them in a simple manner. Although the exact definition of the subroutines may be regarded as a matter of detail, an overview of the ones currently available would give some insight into the matter of the user library. The following subroutines have been included:

CREM
Creates a hypermatrix, by creating two files, one random accsess and one sequential. It requires the name of the hypermatrix, its size and partitioning scheme.

OPNM
Opens the files of an existing hypermatrix, given its name. Transfers directory information from disk to core in preparation for mathematical manipulations involving the hypermatrix.

CLSM
Stores directory information of a hypermatrix from core onto its sequential file and closes the matrix files.

DELM
Deletes the files of a hypermatrix and frees the corresponding core storage locations.

ADDB
Adds a block onto a hypermatrix given its name, block size, starting row and column addressed for the destination.

SUBB
Subtracts a block from a hypermatrix given its name, block size and starting row and column addresses of destination.

ADDMB
Adds a mapped block onto a hypermatrix given the block size, and lists of column and row addresses to be used in the mapping.

SUBMB
Subtracts a mapped block from a hypermatrix given the block size, and lists of column and row addresses.

INSB
Inserts a block into a hypermatrix given the block size and starting address.

EXTB
Extracts a block from a matrix given the block size and starting address.

INSMB
Inserts a mapped block into a matrix given the block size, and lists of row and column addresses.

EXTMB
Extracts a mapped block from a matrix given the block size, and lists of row and column addresses.

CREV
Creates two hypervector files; one random and one sequential, given vector name, size and partitioning scheme.

OPNV
Opens the files for an existing hypervector, given the vector name.

CLSV
Stores directory information of a hypervector from core onto the sequential file and closes the hypervector files.

DELV
Deletes hypervector files and frees the corresponding core storage location.

CREVTR
Creates a sequential (unpartitioned) vector file given the vector name and the size.

OPNVTR
Opens an existing (unpartitioned) vector file given the vector name.

RDVTR
Reads (unpartitioned) vector file from the disk.

WRVTR
Writes (unpartitioned) vector file to the disk.

3.3 MATRIX PROCESSORS:

The matrix processors are independent programs which perform specific hypermatrix operations and thereby allow the solution of linear and nonlinear structural problems efficiently and conveniently. The processors described in this section are considered primitives to be used in constructing nonlinear algorithms. They do not incorporate, as of yet, a control structure (executive). This may be done by using operating system commands, or by providing the processors in submatrix form and writing a main program to serve as the executive. The processors envisioned in the experimental software package are given below:

MMAD
Adds two hypermatrices and stores the result in the form of a third hypermatrix.

MMST
Subtracts two hypermatrices and stores the result into a third hypermatrix.

MSML
Multiplies a hypermatrix by a scalar value.

MVML
Multiplies a hypermatrix by a vector.

MMML
Performs the multiplication operation on two given hypermatrices. The result is stored into a third hypermatrix.

MTR
Takes the transpose of a given hypermatrix and stores it as a second hypermatrix.

TMML
Multiplies the transpose of a hypermatrix by another hypermatrix and stores the result as a third hypermatrix.

SYDEC
Decomposes a given symmetric hypermatrix into the L D U form using a generalized Cholesky scheme. In addition, finds the value of its determinant.

MEVV
Calculates a specified number of eigenvalues and eigenvectors of a given hypermatrix.

FRDBST
Using the upper triangular part, U, of a symmetric coefficient decomposed hypermatrix and a given right hand side vector, the processor calculates the unknown vector.

MDET
Finds the value of the determinant of a given hypermatrix.

4. TYPICAL PROCEDURE FOR NONLINEAR FINITE ELEMENT ANALYSIS

A typical procedure for the solution of a nonlinear structural problem, using the finite element method, is shown in Fig. 2. In general, a static or dynamic nonlinear finite element analysis, including geometric and material nonlinearities, is accomplished

using an incremental formulation. The relevant variables are updated incrementally at successive load or time steps (steps 3,4,5,6,7,8 in Fig. 2) in order to trace out the complete solution path.

In static analysis, for example, the governing incremental finite element equations of the discretized model can be expressed in the following form:

$$[K]\{\Delta D\} = \{R\} - \{S\}$$

where [K] is the tangent stiffness matrix corresponding to the configuration of the system at the current load level; $\{\Delta D\}$ is the vector of incremental nodal point displacements; {R} is the vector of externally applied nodal point loads; and {S} is the vector of nodal point forces equivalent to the internal stresses at the given load level. The resultant forces $\{\Delta R\}$ ($=\{R\}-\{S\}$) are called 'residual' or 'out of balance' forces and for nodal equilibrium, they should vanish. Since an approximate solution is used, the $\{\Delta R\}$ vector is nonzero. In most cases, to ensure sufficiently accurate and stable solutions, equilibrium iterations are performed for each load step to reduce ΔR to an acceptable magnitude.

By employing the well known Newton-Raphson method (13-15) as a representative solution scheme, the iterative solution procedure (steps 4,5,6,7 in Fig. 2) may be more accurately described by the equations:

$$[K]i\{\Delta D\}i+1 = \{R\} - \{S\}i$$

$$\{D\}i+1 = \{D\}i + \{\Delta D\}i+1$$

where [K]i and [S]i are based on the current displacements {D}i and are usually formed by assembling contributions from all elements. [K]i, [S]i and {D}i are updated after each cycle. Therefore, each iterative cycle involves assembly and factorization of the tangent stiffness matrix, computation of the residual forces, and the solution of a system of linear algebraic equations to find the displacements. After convergence at a given load level, external loads are incremented again to find the new equilibrium configuration.

The equilibrium iterations use matrix and vector operations repeatedly. The question here is whether the introduction of array processing would produce a significant speed up of the computation. This question can only be resolved by breaking down the overall process in its most elementary blocks, taking accurate measurements of the time required for their performance, constructing accurate parametric formulae reflecting these measurements and then imbedding them in a carefully designed simulator. Finally, representative problems must be selected and run, in conjunction with the simulator.

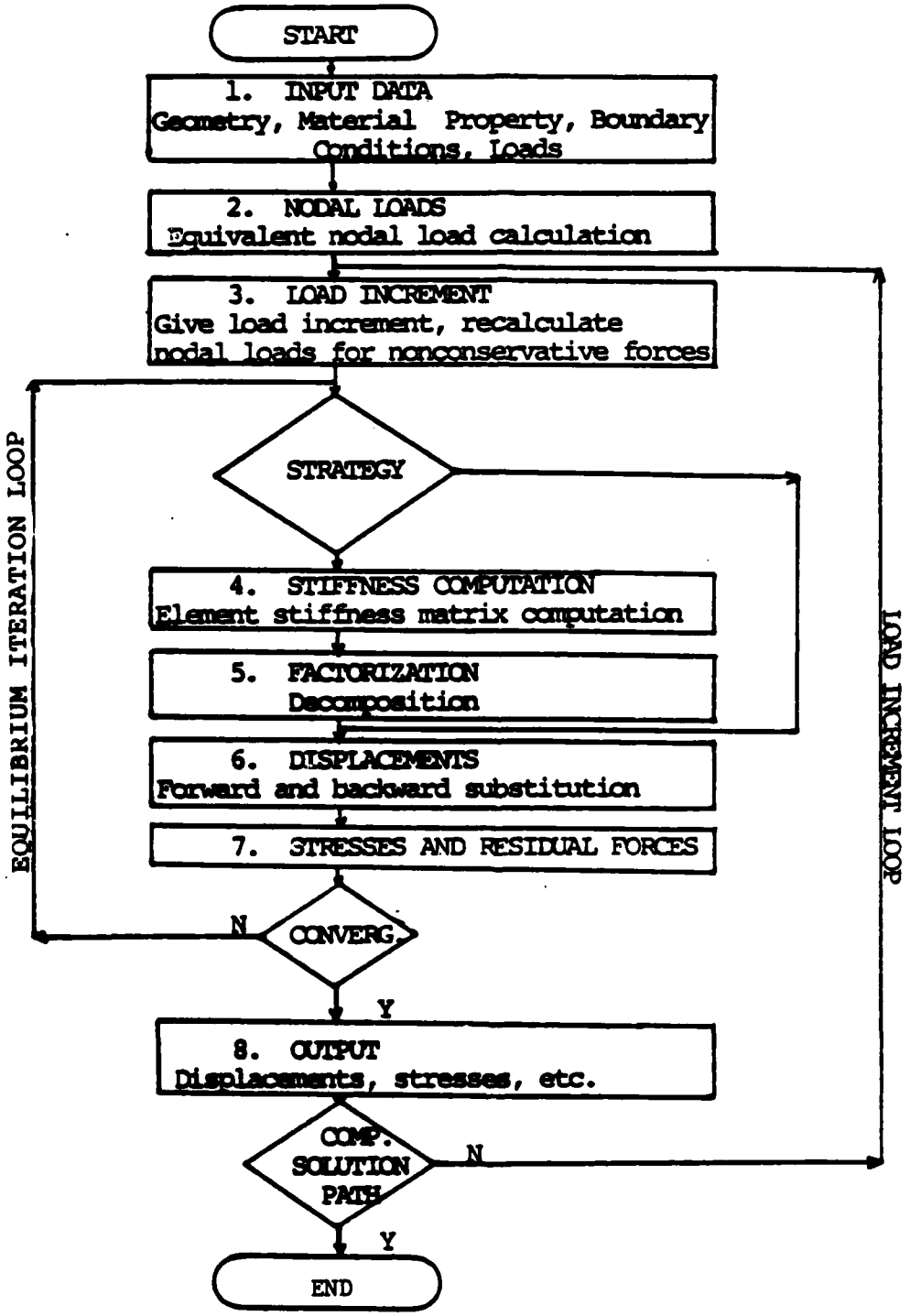


Figure 2. Typical Flow Chart For Nonlinear Finite Element Analysis

5. PERFORMANCE MEASUREMENTS

Time measurements of basic I/O and computational operations are the starting point for the design of the code modules. They are indispensible for the writing of efficient code. In addition, such measurements are invaluable for the design of simulators which help to optimize the processors without need for running large problems. Eventhough it is possible to estimate some of the measured values directly from vendor supplied hardware characteristics, we have found it more reliable to base such figures on actual time measurements.

5.1 TIME MEASUREMENTS OF VECTOR AND MATRIX OPERATIONS

Elapsed time measurements for certain basic vector and matrix operations have been taken for both the host computer, and the array processor. Table 1 summarizes the vector measurements. Table 2 summarizes time measurements for matrix operations. For operation on 500X1 vectors, the speed up factor ranges from 1.05 for vector clear to 3.5 for vector scalar multiplication. For 10000X1 vectors, the corresponding factors are 5.8 and 22.9, the later for the dot product. It is clear that, in order to obtain a significant speed advantage, unrealistically long vectors are required. The matrix operations, on the other hand, show substantial speed up ratios ranging from 11.5 for (10X10) matrices to 42.3 for (100X100) matrices in matrix multiplication and 24.6 to 83.0 for matrix inversion.

Matrix vector multiplication shows smaller speed-up factors of 1.35 to 28.8.

Vector Size	Dot Product		Vector Clear		Vector Add		Vector Subtract		Vector Sc. Mult.		Vector Division	
	HC32	AP64	HC32	AP64	HC32	AP64	HC32	AP64	HC32	AP64	HC32	AP64
100	3.3	4.4	0.8	3.5	1.8	4.9	1.8	4.9	3.0	3.9	5.3	5.3
625	20.6	5.0	5.1	4.1	10.5	5.7	10.5	5.7	18.8	4.5	33.1	9.0
2500	83.1	6.9	20.2	6.1	42.0	8.9	41.8	8.9	75.1	6.5	132.2	21.9
5625	186.7	10.1	45.4	9.2	94.5	14.1	93.9	14.1	168.9	9.7	297.3	43.5
10000	336.5	14.7	80.7	13.8	167.7	21.5	166.8	21.5	299.3	14.2	529.4	73.8

Table 1. Elapsed Time for Double Precision Vector Operations. (Times in msecs.)

MATRIX SIZE	Matrix Inverse		Matrix Multiplication		Matrix Vector Multiplication	
	HC32	AP64	HC32	AP64	HC32	AP64
10X10	80	3.25	47	4.1	4.2	3.12
25X25	1400	20.0	814	20.2	25.23	3.75
50X50	12100	160.0	5833	140.5	100.27	5.98
75X75	43900	527.0	20435	467.1	225.3	9.71
100X100	103700	1250.0	46679	1103	400.42	13.93

Table 2. Timing of Double Precision Matrix Operations. (Times in msecs.)

5.2 TIME MEASUREMENTS FOR INPUT/OUTPUT (I/C) OPERATIONS

Two different types of input/output (data transfer) operations are encountered; transfers between the host and array processor memories and I/O transfers between the host memory and the disk. Since the computer configuration used in the research has a common memory interface, the number of transfers of the first type is minimized. In a conventional host/array processor interface (8,9), the host transfer overhead was about 12 milliseconds. With the current common memory interface, the overhead is reduced to 3 milliseconds.

Measurements listed in Tables 3 and 4 show that I/O transfer times between the host and the disk are too high. As a matter of fact, it takes 3.3 times longer to transfer a (50X50) matrix than to multiply two such matrices together. Examining the time measurements carefully, it appears that there is a disk latency of 15 msecs. and a transfer rate of less then 0.04 MBps. This poor performance is clearly due to inefficient FORTRAN I/O routines. We believe that these transfer times can be reduced significantly, as was reported in Ref.(11) and we hope to report on this in the future. Meanwhile, we shall, later on in this paper, examine the effect of I/O transfer speed up.

Vector Size	Write or Read Elapsed	CPU
100	30	14.0
625	120	73.8
2500	470	291.3
5625	920	580.1
10000	1850	1159.0

Table 3 Double Precision Vector Read/Write From a Sequential Unformatted File. (Time in msecs.)

Matrix Size	Write or Read Elapsed	CPU
10X10	50	14
25X25	130	74
50X50	470	292
75X75	1030	653
100X100	1790	1161

Table 4 Double Precision Matrix Read/Write From a Random Unblocked File. (Time in msecs.)

6. SIMULATOR DESIGN

The simulator is a collection of subroutines. The primary group mimics the time consuming operations, such as matrix arithmetic and matrix I/O operations. The package collects operational statistics, amongst which are the number of operations and their type, along with the estimated CPU and Elapsed times for the complete process.

Although, in principle, no I/O or computational operations are executed, the computation can not be correctly simulated without the preservation of the hypermatrix data structure. This means that the hypermatrix directories must be generated and stored, but not the submatrices themselves.

In addition to that, the library and processors of the solution package use core buffering to eliminate unnecessary I/O transfers. The number of submatrices resident in the buffer is a function of available core space and greatly influences processor performance. This number is one of the parameters defined in the simulator.

The simulator is a valuable tool for developing and analyzing new algorithms in a complex hardware environment. Its value, in our case, is particularly great due to its low cost. In addition, its flexibility allows easy software modifications (4,16). The time measurements taken above represent an important and difficult part of the simulator design. Others include the careful exclusion of measurement noise, selection of the appropriate parameters and assuming a stable measurement environment.

7. APPLICATION TC REPRESENTATIVE PROBLEMS

7.1 SELECTION OF THE PROBLEM

In order to demonstrate the software performance, a 3-D frame structure model (Fig. 3) is selected. Three different cases are considered using the same general topology. The first example has 1890 degrees of freedom (d.o.f.) with a half bandwith (h.b.w.) of 240. The second has 3852 d.o.f, with h.b.w.= 336; and the last one has 8736 d.o.f., with h.b.w.= 432.

For all of the cases, the following assumptions are made: the submatrix size is 48 by 48, the core is large enough to store up to 91 such submatrices (maximum half bandwith for efficient operation=624 and core required = 2MB) and the load vector is assumed to reside fully in core.

7.2 RELATIVE PERFORMANCE

Of the numerous vector and matrix operations involved in the incremental Newton-Raphson iteration procedure, only two processors appear to benefit significantly from vectorization via a "subroutine box". These operations are characterized by extensive repetitive computational requirements compared to I/O operations. Furthermore, a minimum size of vectors and submatrices is needed before the operations may benefit from the use of an array processor (9). It is possible, in our view, to speed up some of the other processors as

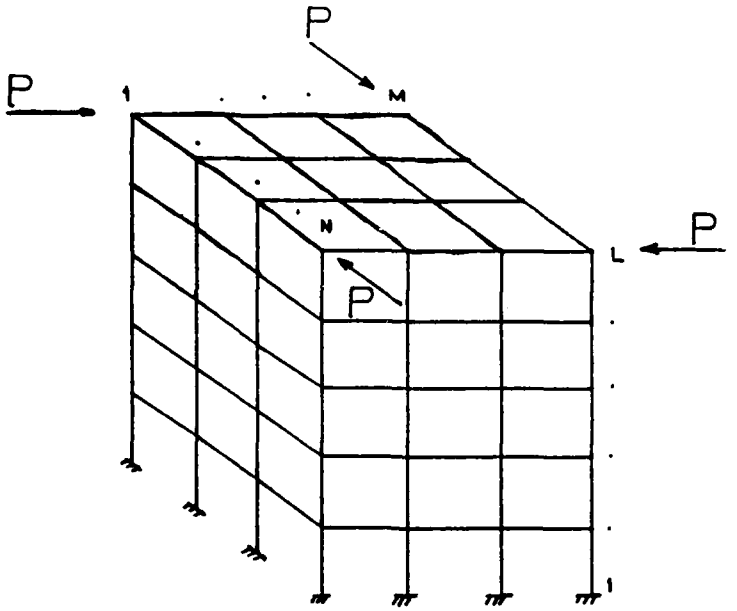


Figure 3. 3-D Frame Structure Model.

well, but it would require significantly more work and the results would not be as spectacular.

If we consider a large displacement - small strain problem, for example, the two prime candidates for vectorization are the matrix decomposition processor (SYDEC) and the forward and backward substitution processor (FRDBST).

A summary of the simulator results are given in Tables 5-7 for the cases described above. A speed up factor of 15.6 to 22.6 is obtained for the decomposition by using the attached array processor Figs. 4-5. The speed up factor increases with problem size. On the other hand, the speed up factor for the forward-backward substitution

decreases slightly with problem size, from 4.6 to 3.4, Fig. 5.

Tables (8.a-c) presents the detailed breakdown of the operations within the decomposition processor for all three examples. It also gives the total time taken for each class of operation. Upon careful examination of the data, it appears that a substantial speed up of the I/O operations, which is possible in our opinion, would cause a marked additional speed up by, perhaps, a factor of 2, as described later.

PROCESSOR	CPU Time		ELAPSED Time	
	HC32	HC32+AP64	HC32	HC32+AP64
DATA	158	158	224	224
LOAD	2	2	7	7
STIFF	283	283	357	357
SYDEC	3145	150	3202	205
FRDBST	556	81	606	131
STRESS	128	128	240	240
CONV	1	1	3	3
OUTPUT	1	1	6	6

Table 5 Relative Performance of Host vs. Host/AP for an 1890 d.o.f. Frame Structure. Time in secs. (NND=315, NEL=378, b=240, M=N=3, L=7).

PROCESSOR	CPU Time		ELAPSED Time	
	HC32	HC32+AP64	HC32	HC32+AP64
DATA	324	324	454	454
LOAD	3	3	9	9
STIFF	584	584	733	733
SYDEC	10041	418	10162	537
FRDBST	1182	207	1305	330
STRESS	265	265	493	493
CONV	1	1	3	3
CUTPUT	2	2	10	10

Table 6 Relative Performance of Host vs. Host/AP for a 3852 d.o.f. Frame Structure. (Time in secs.) (NND=642, NEL=783, b=336, M=4, N=3, L=10).

PROCESSOR	CPU Time		ELAPSED Time	
	HC32	HC32+AP64	HC32	HC32+AP64
DATA	741	741	1031	1031
LOAD	7	6	15	13
STIFF	1317	1317	1642	1642
SYDEC	34879	1273	35161	1558
FRDBST	2804	579	3139	913
STRESS	608	608	1129	1129
CONV	2	2	7	4
CUTPUT	5	5	20	20

Table 7 Relative Performance of host vs. host/AP for an 8736 d.o.f. Frame Structure. (Time in secs.) (NND=1436, NEL=1800, b=432, M=N=4, L=16).

Returning now to the Newton-Raphson method, it is possible to estimate the overall speed up factors for the basic load increment loop, as well as for the equilibrium iteration loop. Tables (9.a-c) shows that an increment of load, followed by a recomputation and decomposition of the stiffness matrix and a computation of the

displacement increment and residual forces, is speeded up by a factor of 4.70 (1890 d.o.f) to 7.82 (8736 d.o.f.). On the other hand, one equilibrium iteration was speeded up by 2.28 to 2.09. One might call this an appreciable improvement, although not spectacular. It is hoped that careful optimization of the I/O transfer speed will produce better ratios.

OPERATION Type	ELAPSED TIME per OPERATION		NUMBER of OPERATIONS	TOTAL TIME	
	HC32	HC32+AP64		HC32	HC32+AP64
Matrix Read	0.470	0.470	130	61	61
Matrix Write	0.470	0.470	143	67	67
Matrix Multiply	5.173	0.125	505	2612	63
Matrix Inversion	11.491	0.141	39	448	5
Others				14	9
Total				3202	205

Table 8.a Distribution of Elapsed Time in Decomposition. Time in Secs. (U=1890).

OPERATION Type	ELAPSED TIME per OPERATION		NUMBER of OPERATIONS	TOTAL TIME	
	HC32	HC32+AP64		HC32	HC32+AP64
Matrix Read	0.470	0.470	271	127	127
Matrix Write	0.470	0.470	364	171	171
Matrix Multiply	5.173	0.125	1725	8923	216
Matrix Inversion	11.491	0.141	80	919	11
Others				22	12
Total				10162	537

Table 8.b Distribution of Elapsed Time in Decomposition. Time in Secs. (U=3852).

OPERATION Type	ELAPSED TIME per OPERATION		NUMBER of OPERATIONS	TOTAL TIME	
	HC32	HC32+AP64		HC32	HC32+AP64
Matrix Read	0.470	0.470	551	259	259
Matrix Write	0.470	0.470	1008	474	474
Matrix Multiply	5.173	0.125	6246	32311	781
Matrix Inversion	11.491	0.141	181	2080	26
Others				37	18
Total				35161	1558

Table 8.c Distribution of Elapsed Time in Decomposition. Time in Secs. (U=8736).

	ELAPSED TIME		SPEED UP FACTOR
	HC32	HC32+AP64	
LOAD CYCLE			
Load Increment	6	6	
Stiffness Assembly	357	357	
Decomposition	3202	205	15.62
Displacement	606	131	4.63
Stresses	240	240	
TOTAL	4411	939	4.70
EQUILIBRIUM ITERATIONS			
Displacement	606	131	4.63
Stresses	240	240	
TOTAL	846	371	2.28

Table 9.a Speed up Factors For The Modified Newton-Raphson Procedure Using an Array Processor.(U=1890).

	ELAPSED TIME HC32	ELAPSED TIME HC32+AP64	SPEED UP FACTOR
LOAD CYCLE			
Load Increment	8	8	
Stiffness Assembly	733	733	
Decomposition	10162	537	18.92
Displacement	1305	330	3.95
Stresses	493	493	
TOTAL	12701	2101	6.05
EQUILIBRIUM ITERATIONS			
Displacement	1305	330	3.95
Stresses	493	493	
TOTAL	1798	823	2.18

Table 9.b Speed up Factors For The Modified Newton-Raphson Procedure Using an Array Processor.(U=3852).

	ELAPSED TIME HC32	ELAPSED TIME HC32+AP64	SPEED UP FACTOR
LOAD CYCLE			
Load Increment	15	13	
Stiffness Assembly	1642	1642	
Decomposition	35161	1558	22.57
Displacement	3139	913	3.44
Stresses	1129	1129	
TOTAL	41084	5253	7.82
EQUILIBRIUM ITERATIONS			
Displacement	3139	913	3.44
Stresses	1127	1127	
TOTAL	4266	2040	2.09

Table 9.c Speed up Factors For The Modified Newton-Raphson Procedure Using an Array Processor.(U=8736).

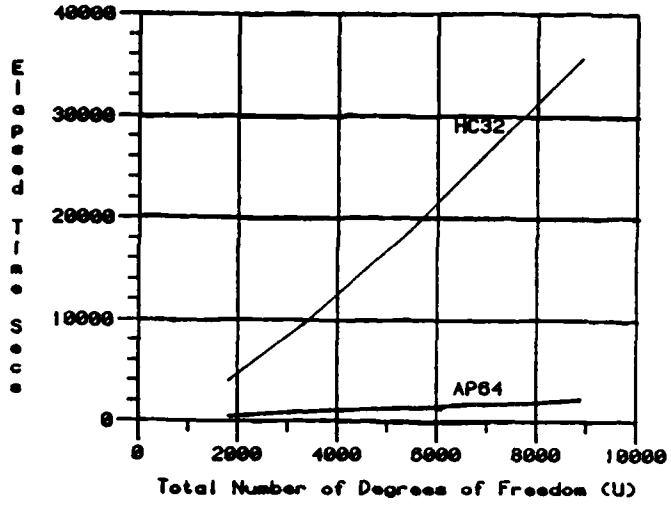


Figure 4. Arary Processor Performance in Decomposition.

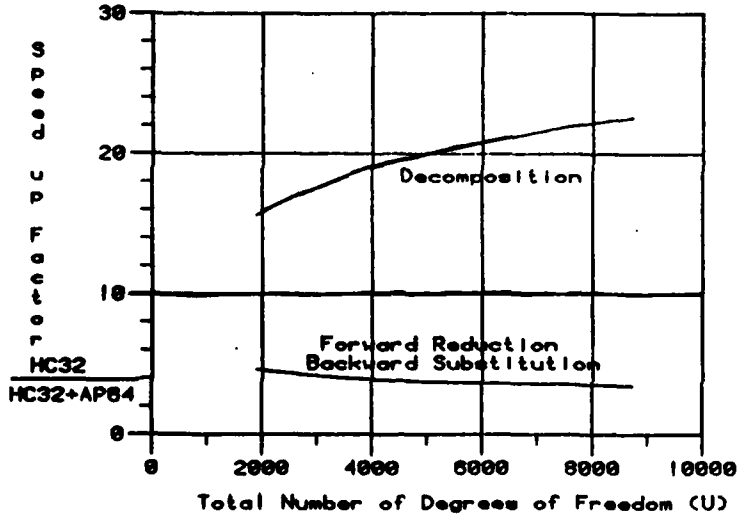


Figure 5. Variation of Speed up Factor With Problem Size.
(Buffer size = 100 submatrix).

7.3 EFFECT OF BUFFERING

In the decomposition processor, the I/O transfer time constitutes presently up to 60 per cent of the total execution elapsed time for the AP64+HC32 configuration, Tables (8.a-c). In order to improve the execution speed and increase the speed up factor, I/O transfers should be minimized. A standard technique in non-virtual memory systems is to use part of the core as a buffer. The goal is to read and write each stiffness submatrix only once. This can only be achieved if the number of submatrices in the buffer is larger than B(B+1)/2, where B is the integer part of ((b-1)/s+2). If the number is less than that, unnecessary I/O operations (thrashing) will result. In order to investigate the effect of buffer size in the decomposition process and illustrate the above point, time measurements were taken for different buffer sizes(Fig. 6). Larger buffers increase the speed up factor, until the maximum is reached at the critical buffer size (Fig. 7). It is clear that, while the buffering effect is minimal for the HC32 alone, it is crucial for the HC32+AP64 combination. More details are given in Tables (10.a,b).

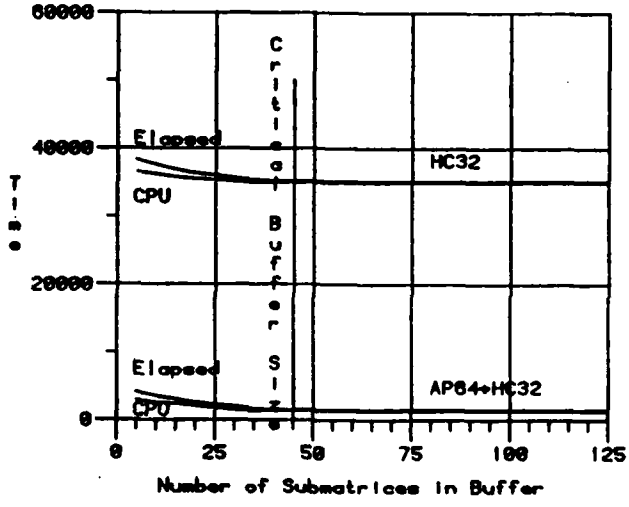


Figure 6 Effect of Buffer Size in Decomposition. Time in Secs. (U=6736, s=48, b=384, NEL=1800, NND=1436)

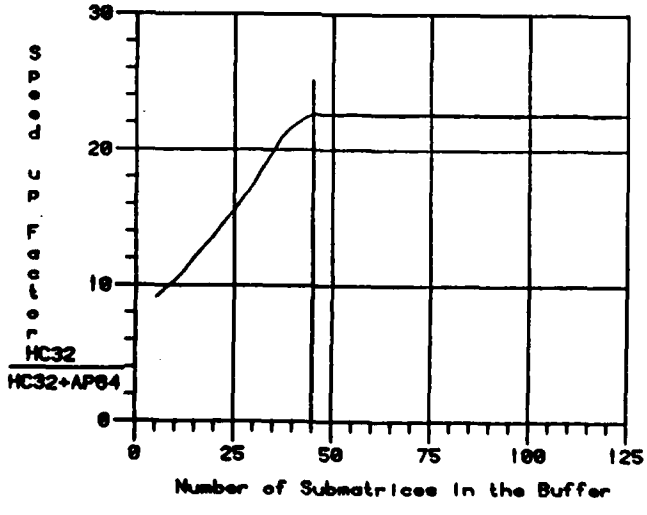


Figure 7 Variation of Speed up Factor With Buffer Size in Decomposition. (U=8736, s=48, b=384, NEL=1800, NND=1436)

Buffer Size (No. of Submat.)	Number of Submatrix Reads/Writes	CPU Time HC32	CPU Time HC32+AP64	Speed up Factor
5	6067	36494	2883	12.7
10	4837	36135	2525	14.3
25	2174	35356	1748	20.2
45	551	34879	1273	27.4
50	551	34879	1273	27.4
100	551	34879	1273	27.4

Table 10.a Effect of Buffer Size on SYDEC CPU Time.
(U=8736, NND=1436, NEL=1800, b=432, B=9).
Optimum Buffer size is 45.

Buffer Size (No. of Submat.)	Number of Submatrix Reads/Writes	Elapsed Time HC32	Elapsed Time HC32+AP64	Speed up Factor
5	6067	37747	4152	9.1
10	4837	37173	3573	10.4
25	2174	35924	2323	15.5
45	551	35161	1558	22.6
50	551	35161	1558	22.6
100	551	35161	1558	22.6

Table 10.b Effect of Buffer size on SYDEC Elapsed Time.
(U=8736, NND=1436, NEL=1800, b=432, B=9)
Optimum buffer size is 45.

8. EFFECT OF RELATIVE PERFORMANCE OF SYSTEM COMPONENTS

The results shown in this paper are based on experience with specific hardware. One of the aims of the research is to reach general conclusions regarding the profitability of the addition of an array processor of the type selected (subroutine box) to a typical minicomputer or personal work station. One might, at this stage,

extrapolate these results in order to reach more general conclusions. Due to the lack of time, however, the discussion is restricted to the decomposition processor performance. This is justified, since this processor represents the most crucial part in nonlinear analysis.

One of the first questions that comes to mind is the effect of more efficient I/O operations. It is clear that an increase in the I/O rate would produce an improved speed up ratio. On the other hand, the relationship between the I/O speed and the speed of the host computer and the array processor is an important factor which should be investigated.

Examining the operation counts for the large frame problem and extrapolating from them, assuming different host computer, array processor and I/O transfer speeds, interesting information is obtained and plotted in Figs.(8) and (9). The basic configuration is that of a host capable of 0.465 Mips (based on manufacturers specifications), an array processor capable of 1.82 MFLOPS in double precision (based on measurements of matrix multiplication) and an I/O transfer rate of 0.04 MBps (measured FORTRAN I/O). Since it is possible to increase the transfer rate by writing assemly language programs, plots are given to demonstrate the variation of the speed up factor as a function of the transfer rate. Transfer rates vary between 0.04 MBps ,which is the measured rate from the current system, and 1.2 MBps ,which is the maximum transfer rate of a typical

disk drive. In Fig. 8 the current array processor speed of 1.82 MFLOPS is used. Plots are included for different real and hypothetical computers with speeds varying between 0.36 Mips to 2.5 Mips. One concludes from this figure that an array processor increases the efficiency of a slower machine considerably for data transfer rates above 0.15 MBps.

On the other hand, the array processor is not as effective for faster minicomputers with speeds of 1 or 2 Mips; although, there still is an appreciable speed up. We conclude also that the array processor should be particularly effective with desk top computers, since the effective speeds of these computers will be somewhat low.

It would be interesting to examine the speed up factor for a hypothetical faster array processor. Assuming that, several years from now, an array processor with a speed ten times faster will be available. The speed up factor variations are shown in Fig. 9. Careful examination of this figure indicates that the faster array processor will greatly enhance host machines similar to those of today. However, higher disk transfer rates would be beneficial, up to 1.2 MBps and more.

We believe that Figs. 8 and 9 will be helpful to investigators in the selection of proper computer system components for a multiprocessor environment.

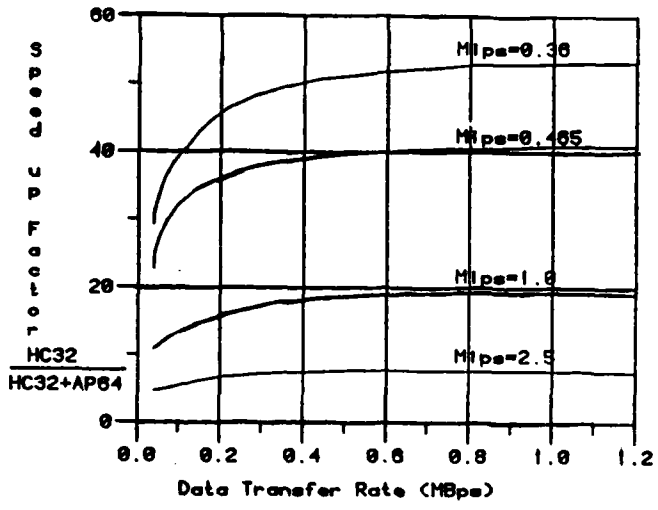


Figure 8 Effect of Data Transfer Rate on Speed up Factor for the Decomposition Processor.(U=8736). AP Speed = 1.82 MFLOPS.

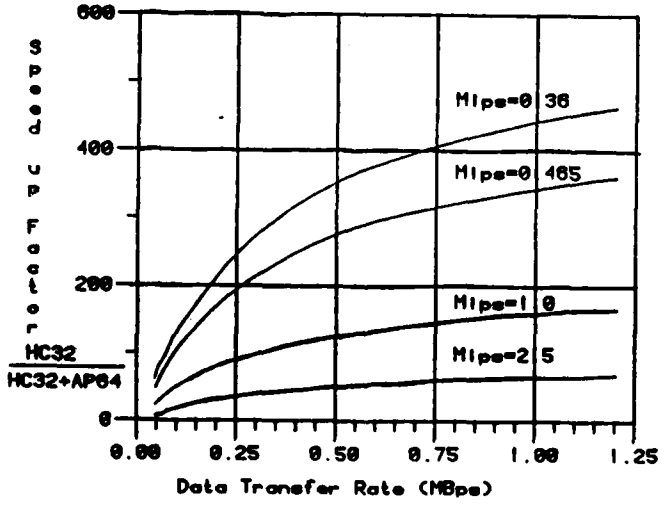


Figure 9 Effect of Data Transfer Rate on Speed up Factor for the Decomposition Processor.(U=8736). AP Speed = 18.2 MFLOPS.

9. CONCLUSIONS AND FUTURE WORK

Software was designed to support the development of large-scale nonlinear structural analysis programs in a multi-processor environment. Then, using the designed software, an application program was developed for the solution of large-scale nonlinear structural analysis problems via the finite element method. In order to predict the system performance under proper array processor software with a minimal cost, a simulator is designed.

Important issues in adapting nonlinear analysis procedures to array processors seem to be the proper task distribution between the host and the array processor, as well as efficient disk operations. The study gives preliminary results regarding the profitability of array processors. The cost effectiveness depends on the relative speed of the host and the array processor, as well as on the transfer rate of the disk.

10. ACKNOWLEDGMENTS

The support of the Office of Naval Research under contract NCOO14-75-C-0837 is gratefully acknowledged. Special thanks are due to Ms.Susan K. Lewis and Ms.Debbie Westerman for their careful assistance in the preparation of this manuscript.

REFERENCES

[1] A.K.Noor, 'Survey of Computer Programs for Solution of Nonlinear Structural and Solid Mechanics Problems',Computers&Structures, 13, pp.425-465,(1981).

[2] J.C.Knight, 'The Current Status of Super Computers', Computers & Structures, 10, pp.401-409, (1979).

[3] V.D.Poor,'The Concept of Attached Processing and Attached Recourse Computer Systems', in Gorsline G.W. ed.,Proc. of First Sigmini Symposium on Small Systems, (1978).

[4] G.A.Strohkorb and A.K.Noor,'Potential of Minicomputer/Array Processor System for Nonlinear Finite-Element Analysis',NASA T.M. 84566, (1983).

[5] H.A.Kamel and J.Maitan, 'Performance of Finite Element Algorithms on an Array Processor-minicomputer Based System', in: Wunderlich, Stein and Bathe eds.,Nonlinear Finite Element Analysis in Structural Mechanics, Springer Verlag, (1981).

[6] J.Maitan and H.A.Kamel, 'Performance of Minicomputers in Finite Element Analysis, Pre- and Postprocessing, ONR Tech. Rept. No.6, University of Arizona, (1980).

[7] J.Maitan, N.Sarigul, O.Paulisinski, and H.A.Kamel, 'Balanced Array Processor Configuration for Finite Element Analysis', presented at INRIA 5th Symposium on Computer Methods in Engineering and Applied Science, Le Chesnay, France,Dec. (1981).

[8] N.Sarigul, J.Maitan, and H.Kamel, 'Implementation of Some Finite Element Algorithms on a Minicomputer with an Attached Array Processor', presented at the CAFEM-6 Conference ,Paris, France, August (1981).

[9] N.Sarigul, J.Maitan, and H.Kamel, 'Solution of Nonlinear Structural Problems Using Array Processors', Computer Meths in Appl. Mechs. and Engng., 34, pp.939-954, (1982).

[10] A.K.Noor and S.J.Voigt,'Hypermatrix Scheme for Finite Element Systems on CDC STAR-100 Computer', Computers & Structures,15, pp.287-296,(1975).

[11] E.U.Cohler and J.A.Cohler,'Array Processors in Finite Element Modeling',Proc. Third World Congress and Exhibition on Finite Element Methods, Beverly Hills, California, Oct. 12-16, (1981).

[12] J.A.Swanson, G.R.Cameron and J.C.Hoberland,'Adapting the ANSYS Finite Element Analysis Program to an Attached Processor',Computer , pp.85-91, (June 1983).

[13] H.Matthies and G.Strang,'The Solution of Nonlinear Finite Element Equations',Int. Journal for Num. Methods in Engng.,14,pp.1613-1626,(1979)

[14] M.A.Crisfield, 'A faster Modified Newton-Raphson Iteration', Comp. Meth. in App Mech. and Engng., 20, pp.267-278, (1979).

[15] S.Utku, R.Melosh, M.Islam, and M.Salama, 'On Nonlinear Finite Element Analysis in Single-, Multi- and Parallel-Processors', Computers & Structures, 15,No.1,pp.39-47, (1982).

[16] D.A.Orbits and D.A.Calahan,'A Cray-1 Simulator and Its Application to Development of High Performance Codes', Proc. of the 1978 LASL Workshop on Vector and Parallel Processors, (1978).

END

FILMED

3-84

DTIC