NASA Technical Memorandum 84566

Potential of Minicomputer/Array-Processor System for Nonlinear Finite-Element Analysis

Gregg A. Strohkorb and Ahmed K. Noor

JUNE 1983

25th Anniversary
1958-1983

NASA

NASA Technical Memorandum 84566

Potential of Minicomputer/Array-Processor System for Nonlinear Finite-Element Analysis

Gregg A. Strohkorb and Ahmed K. Noor
The George Washington University
Joint Institute for Advancement of Flight Sciences
Langley Research Center
Hampton, Virginia

National Aeronautics
and Space Administration

**Scientific and Technical
Information Branch**

1983

CONTENTS

1 INTRODUCTION

The last two decades have witnessed an explosive growth in computer technology. Computer-hardware developments are most noticeable at the two extremes of the spectrum. At one end there are large expensive computer systems, usually referred to as supercomputers, such as the CRAY-1 S, CDC[1] CYBER 205, and Burroughs Scientific Processor (BSP), which have radically different architectures and very high performance (computational speed of the order of 100 million floating-point operations per second (100 MFLOPS) or more). At the other end of the spectrum are the various types of minicomputers, desktop computers, microprocessors, and programmable pocket calculators. These devices provide reasonable performance at less cost. However, their effectiveness is limited for solving large-scale structural problems such as those encountered in crash dynamics and large space structures.

Recently, array processors, which are low-cost, high-performance units used as attachments to minicomputers and mainframes, have gained popularity. The question arises as to whether the combination of a minicomputer and an array processor in a dual-processor system can extend the range of finite-element problems that can be solved effectively by the minicomputer. This capability is likely to be accomplished through efficient implementation of both the numerical algorithms used in the finite-element solution and the communication protocol between the host minicomputer and the array processor. The question of synchronization between the different parts of the system to ensure correct and efficient function of the minicomputer/array-processor system is of paramount importance. Studies have been made to assess the potential of minicomputer/array-processor systems for large structural calculations. Potential speedups of some of the matrix processors used in the SPAR structural-analysis program are studied in reference 1. Implementation of finite-element algorithms on a minicomputer with an attached array processor is discussed in references 2 and 3. In all the cited references, however, estimates are given for speedups of certain matrix operations within the finite-element program, but **no speedup estimates are given for the entire solution process.** Furthermore, no guidelines are presented for realizing the full potential of the minicomputer/array-processor system in large-scale structural calculations. The present paper focuses on these questions. Specifically, the objectives of the present paper are as follows:

(1) To explore the feasibility and potential of using a minicomputer/array-processor system for nonlinear finite-element analysis of large structural systems

(2) To identify the guidelines for computational strategy and programming techniques required for realizing this potential

In this study, a Prime 750 superminicomputer is the host computer, and a software simulator residing on the Prime is employed to assess and measure the performance of the Floating Point Systems (FPS) AP-120B array processor. The selection of the AP-120B was prompted by the fact that it is the most widely used array processor and the only commercially available one with a simulator on the Prime. In the present paper, emphasis is placed on understanding characteristics of the hardware and

---

[1]CDC: Registered trademark of Control Data Corporation.

interplay among the hardware components, selection of proper computational procedure, vectorization of numerical algorithms, minimization of input-output (I/O) operations, and overlapping host and array-processor operations. Two numerical examples are presented to demonstrate the gain in speed obtained by using the proposed algorithms on the minicomputer/array-processor system. Also, new advances in array-processor hardware by two commercial vendors are outlined, and possible improvements in computational algorithms are identified. The combination of hardware and algorithm improvements can significantly enhance the effectiveness of these systems for large-scale nonlinear analysis. The methodology developed in this paper is intended to guide further efforts on distributed finite-element computations.

2 SYMBOLS

$[B]$	linear-strain-displacement matrix
$\{\bar{B}(X)\}$	nonlinear-strain-displacement vector
$[C]$	material-stiffness matrix of structure
E	Young's modulus of the material
e	error norm, measuring error of reduced system
$\bar{e}$	error norm defined in equation (C7)
F_{IJK}, G_{IJKL}	nonlinear-stiffness coefficients of discretized structure
$\tilde{F}_{ijk}, \tilde{G}_{ijk\ell}$	nonlinear-stiffness coefficients of reduced system
H_ℓ	weighting coefficients for numerical quadrature, where $\ell = 1$ to n
h	shell thickness
$[J]$	Jacobian matrix of transformation from local to natural (dimensionless) element coordinates
$[K]$	linear-global-stiffness matrix
K_{IJ}	linear-stiffness coefficients of discretized structure
$\overset{*}{K}_{IJ}$	$\equiv K_{IJ} + F_{IJK}\, X_K + G_{IJKL}\, X_K\, X_L$
$\tilde{K}_{ij}$	linear-stiffness coefficients of reduced system
$[K^{(e)}]$	linear-element-stiffness matrix
L	length of shell
N	total number of displacement degrees of freedom
n	number of numerical quadrature points in element domain
Q_I	normalized external-load components

$\tilde{Q}_i$	normalized load components of reduced system
q	load parameter
R	radius of curvature of shell
r	number of unknowns of reduced equations
U	total strain energy of structure
$\bar{U}$	contribution of cubic and quartic terms in nodal displacement to strain energy
u_1, u_2, w	displacement components in coordinate directions
w_ℓ	$= H_\ell \, (\det[J])_\ell$ where $\ell = 1$ to n
X_I	nodal-displacement parameters
$\{X\}$	vector of nodal displacements
x_1, x_2, x_3	orthogonal coordinate system
β	condition number of Gram matrix of global-approximation functions
Γ_{Ji}	$= \dfrac{\partial^{i-1} X_J}{\partial \lambda^{i-1}}$ as defined in equation (3)
$\Delta(\ )$	change in a function
λ	path parameter in solution space
ν	Poisson's ratio of the material
$\{\sigma\}$	vector of nodal stress resultants
ϕ_1, ϕ_2	rotation components
ψ_i	unknowns of reduced equations

Subscripts:

I,J,K,L	indices ranging from 1 to N (number of degrees of freedom of full system)
i,j,k,ℓ	indices ranging from 1 to r (number of degrees of freedom of reduced system)
max	maximum

Superscripts:

e	elemental quantity

r	rth iteration cycle (see section 12.2)
T	transposition

Abbreviations:

AP	AP-120B array processor
CP	central-processing bound
CPU	central-processing unit
DCU	data-base complex utility
DOF	degrees of freedom
FPS	floating-point system
I/O	input-output operation
L.H.S.	left-hand side
MFLOPS	million floating-point operations per second
2-D,3-D,4-D	two-, three-, and four-dimensional, respectively

3 SUMMARY OF MAJOR-HARDWARE AND SYSTEM-SOFTWARE FEATURES OF MINICOMPUTER/ARRAY-PROCESSOR SYSTEM

The minicomputer/array-processor system has a number of features which distinguish it from current third-generation (mainframe) hardware. These features provide a high degree of interactive capability and attractive computational speeds. Although the speed of the system is below those of the large mainframes, its purchase cost is considerably less. A schematic drawing of the organization of an assumed minicomputer/array-processor system is shown in figure 1.

A detailed description of the hardware and software of both the host minicomputer and the array processor is given in the manufacturers' manuals (refs. 4 to 8), and their major features, which are exploited in the present study, are summarized herein.

3.1 Prime 750 Host Computer

The Prime 750 host computer belongs to the class of computers frequently referred to as superminis. (See ref. 9 for a classification of hardware systems.) It has a 32-bit-word (approximately 8 significant digits) central processor and a virtual-memory operating system. A basic feature of the Prime 750 is a three-level memory organization depicted in figure 2. The three levels are the cache memory, the central memory, and the disk system. The Prime 750 supports up to 2 million words (8M bytes) of central memory and 600 million words (2.4×10^9 bytes) of disk storage. The system can run in both interactive and batch modes, and up to 63 users can be supported simultaneously. The memory organization and the computational speed of the Prime 750 at the Langley Research Center are discussed in appendix A.

3.2 Floating Point Systems AP-120B Array Processor

An array processor (AP) is a high-speed special-purpose computational device which can perform repetitive computations on well-structured data sets at effective speeds far beyond those achieved by current minicomputers. The AP-120B array processor, also called an attached processor, has the following features (see ref. 10):

(1) It is designed and accessed as a peripheral (i.e., like a tape drive) for a conventional host minicomputer, and it is intended to enhance the performance of the host in specific numerical computing tasks.

(2) It achieves high performance through both parallelism and pipelining.

(3) It includes an arithmetic section containing one adder and one multiplier capable of operating in parallel.

(4) It is not hardwired. Rather, it can be programmed by the user in FORTRAN or assembler language to perform a variety of computational tasks.

Parallelism is a major feature of the AP-120B that allows high-speed vector and matrix processing. In addition, the parallel structure of the AP-120B allows the overhead of loop indexing, array indexing, and data fetching to be performed in parallel with floating-point computations. Parallel operations include integer indexing, branch instructions, memory fetches, memory storage, I/O with the host, and floating-point arithmetic. These features make the AP-120B a much faster processor than most general-purpose computers which perform operations sequentially. The AP-120B has a peak processing rate of 12 million floating-point operations per second (12 MFLOPS) and a concurrent 6 million integer and addressing operations per second.

The two major components of the AP-120B are the memory and the arithmetic unit. The organization of these two components is discussed in reference 11 and is summarized in appendix A. The memory includes main-data memory, table memory, program memory, data-pad registers, and address registers. (See fig. 3.) The function of each memory, including its size and word length, is given in appendix A.

The arithmetic unit of the AP-120B has a two-stage pipeline adder and a three-stage pipeline multiplier. Each can perform up to 6 million floating-point operations per second (6 MFLOPS). If both pipelines are kept constantly full, the speed can reach 12 MFLOPS, but it is often impossible to keep either or both pipelines full. Sustained speeds of up to 4 MFLOPS have been observed for some fluid-dynamics computations by W. T. Thompkins at the Massachusetts Institute of Technology. The interface between the host and the array processor is described in the succeeding subsection.

3.3 Interface Between Host Computer and Array Processor

There are two distinct operating systems which reside on the host: (1) the operating system of the host computer, and (2) a software driver for the array processor called "APEX" (array-processor executive). The host operating system controls the operation of the host, including the user program which, in turn, specifies the sequence of events necessary to perform the computation. The data on which the computations are performed reside on a disk controlled by the host. Some of these data are present in the memory of the host, where they may undergo simple reorganization on their way to and from the array processor.

The host interface to the array processor consists of an interface board which is controlled by APEX. This software is resident on the host and facilitates communication with the array processor by translating FORTRAN calls into small descriptive packages called "**function control blocks.**" These tasks are shipped to the array processor to initiate specific computations or data transfers on the latter device. The APEX is part of the host load module whose formation is depicted in figure 4. Additional details on the array-processor executive are given in appendix A.

3.4 Comments on Operation and Performance of Minicomputer/Array-Processor System

The following comments regarding the operation and performance of the minicomputer/array-processor system seem to be in order:

(1) The high speed of the AP-120B can be attributed to the following three factors: (a) speed of the components (167-ns cycle time compared with 600-ns cycle time on the Prime 750); (b) parallel architecture with complete parallelism in the sense that **all** memory and arithmetic units can perform operations in parallel; and (c) pipeline processing of arithmetic operations.

(2) The AP-120B performs vector operations with scalar hardware. Software loops constructed from scalar operations are used to program vector operations. By contrast, most vector computers, such as the CRAY-1 S and the CYBER 205, provide one set of hardware to perform scalar operations and another set to perform vector computations. The use of scalar hardware to perform both vector and scalar operations helped to keep the purchase cost of the AP-120B substantially below that of supercomputers. The penalty for this architecture is the increase in software complexity for vector operations. A detailed discussion of this subject is given in references 6 and 10. Sample timings for vector-software operations on the AP-120B are given in table I, and the effect of vector length on the number of results per second is depicted in figure 5.

(3) Parallelism on the AP-120B can be exploited only by the software. To synchronize and coordinate concurrent activities, the programmer or compiler must explicitly code all parallelism into the program. This situation requires use of the lowest level of programming language (called "microcode") and significantly complicates the software development. (See ref. 12.)

Extensive libraries of microcoded routines (some callable from the FORTRAN program) are available for use on the AP-120B. These routines include an assembler (APAL), a loader and a linker (APLINK), a software simulator (APSIM), an on-line software debugger (ADBUG), an executive driver (APEX), a software chaining utility (VFC), and an extensive mathematics library including vector operations, matrix operations, vector-to-scalar operations, and fast Fourier transform. (See ref. 7.)

3.5 Simulator

A software simulator (APSIM) has been developed by Floating Point Systems, Inc., to assess the performance of the array processor through simulating the operation of the AP system software APEX and the execution of the assembler programs on the array processor. The simulator can be used on the host minicomputer for performing actual computation and for providing run-time estimates on the array processor without actually using the array processor. The APSIM can run interactively, thereby providing the user with maximum control over the program execution. The simulation is limited to program execution on the AP; interaction with the host is not simulated. In the present study, however, the times expended in the communication between the Prime and the array processor are estimated to be only a small fraction of the total central-processing unit (CPU) time. Other difficulties observed in using the simulator are identified in appendix A.

4 SUMMARY OF COMPUTATIONAL PROCEDURE AND NONLINEAR FINITE-ELEMENT EQUATIONS

To simplify the presentation, discussion herein is limited to large-deflection static analysis of shells. The analytical formulation is based on a form of the geometrically nonlinear shallow-shell theory with the effects of transverse shear deformation included. A displacement formulation is used with the fundamental unknowns consisting of the three displacement components and the two rotations at each point of the middle surface of the shell structure. The shell is modeled by using 16-node quadrilateral elements with bicubic Lagrangian interpolation functions for each of the displacement and rotation components.

The computational procedure used herein is based on the combined use of finite elements and the classical Rayleigh-Ritz approximation. The procedure has been referred to as the reduced-basis technique and is described in detail in references 13 and 14. The discretization of the shell is done by using finite elements. The nonlinear system of finite-element equations describing the response of the discretized shell is then replaced by a reduced system of equations with considerably fewer unknowns through the application of the classical Rayleigh-Ritz technique. The reduced system of equations may be updated periodically as needed.

4.1 Mathematical Formulation

4.1.1 Governing finite-element equations.- A total Lagrangian formulation is used in describing the response of the shell, and the governing finite-element equation in index notation can be cast in the following form:

$$K_{IJ}\, X_J + \frac{1}{2} F_{IJK}\, X_J\, X_K + \frac{1}{3} G_{IJKL}\, X_J\, X_K\, X_L - qQ_I = 0 \qquad (1)$$

where K_{IJ} are the linear-stiffness coefficients; F_{IJK} and G_{IJKL} are nonlinear-stiffness coefficients; X_J are unknown nodal-displacement parameters; Q_I are normalized external-load components; and q is a load parameter. The range of the indices I,J,K,L is 1 to N where N is the total number of displacement degrees of freedom, and a repeated index denotes summation over its full range. The coefficients K_{IJ}, F_{IJK}, and G_{IJKL} are completely symmetric with respect to permutation of indices.

The loading is assumed to be conservative and proportional. As the load is increased, the value of the load parameter q changes, but the component Q_I remains constant; that is, the spatial distribution of the load does not change.

4.1.2 Basis reduction and reduced system of equations.- The nonlinear solution X_I is approximated, over a range of values of the parameter q, by a linear combination of a small number of global-approximation (or basis) vectors. The basis vectors consist of a nonlinear solution for a particular value of q and a number of its path derivatives (derivatives of X_I with respect to a path parameter λ which may be identified with a loading or displacement parameter) at the same value of q. The approximation can be expressed as

$$X_J = \Gamma_{Ji} \psi_i \tag{2}$$

where $J = 1$ to N, $i = 1$ to r, $r << N$, ψ_i are undetermined coefficients, and Γ_{Ji} is a transformation matrix whose columns represent global-approximation functions or basis vectors. Thus,

$$\Gamma_{Ji} = \left(X_J \; \frac{\partial X_J}{\partial \lambda} \; \frac{\partial^2 X_J}{\partial \lambda^2} \; \dots \; \frac{\partial^{r-1} X_J}{\partial \lambda^{r-1}} \right) \tag{3}$$

A Rayleigh-Ritz technique is then used to approximate equation (1) by a much smaller system of nonlinear algebraic equations in the new unknowns ψ_i. The reduced equations have the following form:

$$\tilde{K}_{ij} \; \psi_j + \frac{1}{2} \tilde{F}_{ijk} \; \psi_j \; \psi_k + \frac{1}{3} \tilde{G}_{ijk\ell} \; \psi_j \; \psi_k \; \psi_\ell - q \; \tilde{Q}_i = 0 \tag{4}$$

where

$$\tilde{K}_{ij} = K_{IJ} \; \Gamma_{Ii} \; \Gamma_{Jj} \tag{5}$$

$$\tilde{F}_{ijk} = F_{IJK} \; \Gamma_{Ii} \; \Gamma_{Jj} \; \Gamma_{Kk} \tag{6}$$

$$\tilde{G}_{ijk\ell} = G_{IJKL} \; \Gamma_{Ii} \; \Gamma_{Jj} \; \Gamma_{Kk} \; \Gamma_{L\ell} \tag{7}$$

$$\tilde{Q}_i = Q_I \; \Gamma_{Ii} \tag{8}$$

The subscripts i,j,k,ℓ are indices ranging from 1 to r. The subscripts I,J,K,L are indices ranging from 1 to N. A repeated index in the same term denotes summation over its full range.

The equations used in evaluating the basis vectors are obtained by successively differentiating the governing finite-element equation (eq. (1)), and solving the resulting system of linear simultaneous algebraic equations. The explicit form of these recursion relations is given in reference 15 and is not reproduced here.

The criterion for selecting the number of the basis vectors is based on monitoring the condition number of the Gram matrix β of the basis vectors and on terminating the generation of the basis vectors when β exceeds a prescribed value. (See ref. 13.) Also, upper and lower limits for the number of basis vectors were chosen to be 10 and 2, respectively.

4.2 Computational Procedure

The five key elements of the computational procedure which strongly affect the performance of the reduced-basis technique are: (1) efficient evaluation of the basis vectors and generation of the reduced system of equations; (2) characterization of the nonlinear response of the shell by means of a single scalar; (3) automatic selection of load step size and evaluation of the corresponding displacements and stress resultants; (4) sensing and controlling the error in the reduced system of equations; and (5) tracing postbuckling and post-limit-point paths. These elements were discussed in detail in references 13 and 14. The reduced-basis technique appears to be suitable for solving large-scale nonlinear problems on the minicomputer/array-processor system because of the following: (1) the operations involving the large arrays of the full system of equations are limited to generating basis vectors and reduced arrays and to sensing the error of the reduced equations (see ref. 13); and (2) many operations required in the solution process can be performed as vector operations.

4.3 Program Organization

The nonlinear analysis program used in the present study uses the data-base management system of the SPAR finite-element system. (See ref. 16.) The program is divided into small self-contained logical sections or processors. Each processor can run as a separate program and complete its execution in an interactive mode. Communication between different processors is made through the common SPAR data base. The processor organization and the control and data flow in the program are discussed in appendix B.

5 VECTORIZATION OF ARITHMETIC OPERATIONS

The array processor is specially designed for vector operations. The performance of a nonlinear finite-element program on the minicomputer/array processor system depends on the extent to which the pipeline and parallel-processing capabilities are used in the different modules of the system. Use of the scalar mode in all or most of the computation may provide moderate benefits (because of the small cycle time of the array processor and the complete parallelism in performing the operations). Full benefits of the array processor, however, are not obtained by use of the scalar mode. The process of designing a numerical algorithm to make effective use of the pipeline (or streaming) capability of the array processor is referred to as **vectorization.** The vectorization of any mathematical operation is dependent on the particular hardware. For a task to be vectorizable on the AP-120B, it should contain three characteristics: (1) repeated operations, (2) independence of each result from the others, and (3) the members of each operand are packed in either contiguous memory locations or at fixed intervals from each other (with some restrictions as seen in the discussion on interleave in appendix A).

In the present study, the following basic operations were vectorized: (1) evaluation of linear- and nonlinear-stiffness arrays $K_{IJ}^{(e)}$, $F_{IJK}^{(e)}$, and $G_{IJKL}^{(e)}$ for independent elements; (2) solution of linear algebraic equations using Cholesky's method;

(3) computation of the error norm and checking the convergence of the full-system solution in the Newton-Raphson iterative technique; (4) generation of basis vectors; and (5) computation of stress resultants and strain energy of the structure. These operations are listed in table II. Vectorization of each of the aforementioned operations included the design of the data structure of the operands (arrays or vectors) as well as the sequence of the computation. For operands with simple-data structure (e.g., full matrices and vectors), the math-library routines (ref. 8) of the array processor were used to perform the vector operations. On the other hand, for operands with complex-data structure (e.g., sparse matrices and completely symmetric three- or four-dimensional arrays), assembler codes were developed to exploit the special-data structure of these operands, thereby improving the efficiency of the vector operations. Details of the vectorization of the basic operations listed in table II are discussed in appendix C. Note that some of the vector operations listed in table II were overlapped with the host I/O operations, with a resulting improvement in efficiency.

6 PROGRAMMING CONSIDERATIONS TO EXPLOIT MINICOMPUTER/ARRAY PROCESSOR CAPABILITIES

After the proper computational procedure has been selected and the numerical algorithms have been vectorized, the realization of the full potential of the minicomputer/array-processor system requires the following: (1) the minimization of the I/O operations in order to make the program central-processing (CP) bound, and (2) organization of the computation and data in order to achieve maximum parallelism between the host and AP operations. The programming efforts to accomplish these tasks are outlined in this section. The major factors affecting the I/O operations are the host Prime 750 computer, the AP-120B, and the communication interface between the host and the AP as listed in table III. In the discussion of these factors, hardware, system software, and user software are considered with emphasis on the user-software techniques to minimize the I/O operations.

6.1 Minimization of I/O Operations

6.1.1 Minimizing the host system I/O.- The major hardware factors that affect I/O operations on the host include cache-memory size and speed, central-memory size and speed, and disk-access speed and transmission rate. The system-software factors affecting I/O include resource management and allocation and the paging algorithms between cache and central memory, as well as between central memory and disk.

On the user-software level, minimization of I/O can be achieved through the proper use of buffering techniques. Six buffering techniques were used in the present study. Three of these techniques were proposed by Mitch Modeleski of Systems Applications, Inc., for use in an atmospheric-simulation model. They were found to be effective on the Prime 650 computer. The other three techniques were developed by the first author of the present paper. In order to assess the effectiveness of the different techniques, a small FORTRAN program was developed to transfer 10 000 vectors of nodal displacements from central memory to disk. The CPU and I/O times for execution of this task were measured by using subroutines SNAP and SHOT. The details of the six buffering techniques and their performance are given in appendix D.

6.1.2 Minimizing the I/O within the array processor.- The hardware and system-software features that affect the I/O within the AP are listed in table III. On the user-program level, I/O operations within the AP can be minimized by minimizing the memory-fetch operations and data-path conflicts through proper selection of the buffering technique used between table memory and main-data memory. The details of these techniques are given in appendix D.

6.1.3 Minimizing I/O between host computer and array processor.- In a multiuser operating-system environment, the host overhead can be a significant portion of the total time needed to run a problem on the minicomputer/array-processor system. On the user-software level, merging data, chaining AP routines, and proper sequencing of AP calls can be used to reduce host-system overhead. Details are described in appendix D.

6.2 Parallel Host and Array-Processor Operations

The parallel architecture of the AP-120B allows overlapping of data transfer from the host computer with the execution of arithmetic operations on the AP. After data have been transferred, the host can perform data management and other tasks during the execution of the operations on the AP. Specifically, for the nonlinear finite-element program considered herein, the host can perform the following three data-management tasks while the AP is executing the arithmetic operations:

(1) Data storage and retrieval from disk

(2) Data reorganization required for host and AP programs

(3) Synchronization of data transmission to and from the AP

Overlapping the data transfer and execution of the AP requires bypassing the task-synchronization commands of the AP, which result in sequential operations, and designing the data structure in the program carefully, so that data sets are available whenever they are needed. Examples of the task-synchronization commands are APWR (wait on running), APWD (wait on data), and APWAIT (wait on data and running).

7 TEST PROBLEMS USED IN EVALUATION OF PROPOSED ALGORITHMS

To assess the performance of the vectorized algorithms and the programming techniques outlined in the previous sections, numerical studies were made on two problems of elastic collapse of cylindrical-shell structures. The problems were run on the Prime 750 minicomputer and subsequently on the minicomputer with an AP simulator. Results were compared to determine the potential gain in speed obtained by use of the AP in conjunction with the techniques discussed in the present paper.

The first test problem has a pear-shaped cross section (fig. 6), and the second structure has a circular cross section and a rectangular cutout (fig. 7). The two problems were used in reference 17 to assess the capability of programs to analyze shell structures. The load was applied to the cylinders by means of a uniform axial shortening which is increased incrementally until the cylinder collapses. Advantage was taken of the symmetry, and each structure was modeled by using 16-node, quadrilateral, shear-flexible, shallow-shell elements with bicubic Lagrangian interpolation functions (a total of 80 generalized-displacement degrees of freedom per element). The total degrees of freedom for the first structure is 1715, and for the second it is 3230. The finite-element models used for the two structures are shown in figures 6 and 7. Four distinct elements are used in modeling the first shell, and 52 elements are used in modeling the second shell. In both problems, the basis vectors were generated at zero loading and, therefore, their evaluation involved the decomposition of the linear-global-stiffness matrix [K]. Five basis vectors were used in the first problem, and six were used in the second.

The CP and I/O times were recorded which were required for evaluating the basis vectors (at zero loading), generating the reduced equations and marching three steps with these equations. Figures 8 and 9 show the CP times expended in 10 different processors for the 2 cylindrical-shell problems. The 10 processors are as follows: (1) evaluation of linear-stiffness arrays $[K^{(e)}]$; (2) evaluation of nonlinear-stiffness arrays $F^{(e)}$ and $G^{(e)}$; (3) assembly of elemental matrices; (4) incorporation of boundary conditions; (5) decomposition of the global-linear-stiffness matrix [K]; (6) forward reduction/back substitution (forward/back solve); (7) solution update; (8) postprocessing (stress resultant and strain-energy computation); (9) generation of basis vectors; and (10) solution of reduced equations. Three of these processors were not vectorized, namely, the assembly, boundary conditions, and solution of reduced equations. Also, since the basis vectors were generated at zero loading, no convergence check was needed for the full system. The percentage of time expended in each of the processors and the estimated gain in speed obtained by using the array processor are summarized in table IV.

As can be seen from table IV and figures 8 and 9, most of the solution time is spent in the processor used for the decomposition of the left-hand side and in the processor used for the evaluation of basis vectors and the generation of reduced equations. The use of the array processor is shown by the simulator to reduce the total times (CP + I/O) expended in these two processors by factors of 46.7 and 17.1 for the pear-shaped cylinder and by factors of 59.8 and 38.1 for the cylinder with a cutout. Estimates for total gains in speed for both sample problems are presented in figure 10. Although the total gains in the CP speed obtained by using the AP for the two problems are 9.2 and 18.1, respectively, the gains in the total (CP + I/O) times are reduced to 5.2 and 9.9, only. Note that the higher gain in speed for the larger problem was due to the higher percentage of time expended in the two processors used for the decomposition of the global-stiffness matrix [K] and generation of basis vectors. Both processors were effectively vectorized.

8 FURTHER IMPROVEMENTS TO MINICOMPUTER/ARRAY-PROCESSOR SYSTEMS

Since the development of the AP-120B, several new advances in AP hardware have taken place. Two of the advanced AP systems are described herein along with improvements in numerical algorithms and software design that are likely to enhance the effectiveness of the minicomputer/array-processor system for large-scale, nonlinear, finite-element analysis.

8.1 New Hardware

Among the new array processors with improved capacity and performance over those of the AP-120B, mention may be made of the FPS-164 (ref. 18) and the CSPI MAP-6400 (ref. 19).

The overall architecture of the first processor, FPS-164, is very similar to the AP-120B. The cycle time and machine timings are identical. However, the following major improvements can be identified:

(1) 64-bit floating-point arithmetic (compared with 38 bit on the AP-120B)

(2) 32-bit integer arithmetic (compared with 16 bit on the AP-120B)

(3) 24-bit addressing (compared with 20 bit addressing on the AP-120B)

(4) 1024 64-bit word-instruction cache-memory loading from main memory, which replaces the program memory of the AP-120B and allows larger programs to be accommodated in a more flexible way

(5) A clock for timing programs (which is lacking on the AP-120B)

(6) A 256-register subroutine call stack and 64 address registers

(7) Main memory expandable to 7.25 million 64-bit words (58M bytes)

A recent study has shown reductions in wall-clock time obtained by using the FPS-164 for a number of benchmark structural problems (900 to 3900 degrees of freedom) by factors of 7.5 to 10.6 over those required by the DEC VAX-11/780 supermini. The general-purpose computer program ANSYS[2] (ref. 20) was used in these studies.

The second group of array processors, the CSPI MAP series, have two major features which distinguish them from the FPS array processors. The first is the asynchronous multiple-bus architecture in contrast to the FPS synchronous architecture. The multiple asynchronous I/O buses on the CSPI MAP array processors significantly reduce data-path conflicts and provide fast concurrent I/O, thereby allowing the arithmetic units to be utilized more effectively for vector operations. However, difficulties can arise in coordinating several asynchronous processors for scalar operations. Processing speeds up to 4.3 MFLOPS have been achieved on the CSPI MAP-300 series. Furthermore, the asynchronous hardware architecture differs from the synchronous in that it allows individual components of the AP to be upgraded without redesigning the entire machine. The second major feature is the use of the shared-memory concept to eliminate the I/O overhead between the host and the array processor. This concept was implemented on the Gould Corporation's S.E.L. 32/77 superminicomputer. The CSPI MAP array processors were coupled with the S.E.L. 32/77 superminis through a common-memory interface to form VPS-3300 and VPS-6400 vector-processing systems. The former has a 32-bit word size, and the latter has a 64-bit word size.

The CSPI MAP series of peripheral array processors offer considerable versatility when connected with different host computers. Only a single board replacement is necessary to interface with new host computers. The use of a CSPI array processor for finite-element analysis is discussed in references 2 and 3.

[2]ANSYS: Registered trademark of Swanson Analysis Systems, Inc.

8.2 Improved Algorithms and Programming Language

To improve the effectiveness of the minicomputer/array-processor system for large-scale finite-element computations, improvements are needed in six areas:

(1) Parallel host and array-processor execution: A greater degree of parallel processing than that used in the present study can be achieved by careful design of the software.

(2) Parallel host and multiple array-processor execution: Many peripheral array processors can be added to the host computer. Additional levels of parallel-processing capability require effective algorithms on the host to control the parallel execution of operations on the host and on the peripheral devices.

(3) Vectorization of packed-data structures: Parallel processing within the array processor of packed-data structures can be accomplished by using table memory to store constant offsets into the packed-data structure.

(4) Computational algorithms for parallel processing: The most time-consuming algorithms in nonlinear finite-element analysis appear to be very adaptable to parallel processing. One criterion used in the selection of the computational algorithms should be to maximize parallel processing.

(5) Expert systems on host computer: Expert systems (ref. 21) are programs written to perform the decision-making processes of human experts. They are a result of research in an area of computer science called artificial intelligence. Among the possible applications for this capability are the following: (a) determination of the optimal number of nodes per hypermatrix block to be used in the Cholesky decomposition and solution of the full system; and (b) allocation of central-memory space for user I/O buffers.

(6) Host-computer programming language: The programming language of the Prime 750 host is FORTRAN, which is not designed for parallel processing. A parallel-processing language, such as ADA (ref. 22) or concurrent PASCAL (ref. 23), would be helpful for program development on a parallel-processing system.

8.3 User Interaction

Increased user interaction with the minicomputer/array-processor system can be achieved through the use of interactive graphics and relational data-base systems. (See ref. 24.) The latter is particularly useful to engineering users because of the natural form of representation of data, namely, two-dimensional tables or relations. A relational information manager (RIM) has recently been implemented on the Prime 750 and the DEC VAX-11/780 computers. (See ref. 24.)

9 CONCLUDING REMARKS

A study is made of the potential of using a minicomputer/array-processor system for large-scale, nonlinear, finite-element analysis. A Prime 750 superminicomputer is used as the host computer, and a software simulator residing on the Prime is employed to assess the performance of the Floating Point Systems AP-120B array processor. The major hardware characteristics of the system that significantly impact nonlinear finite-element computations are identified. Computational strategies and programming techniques are developed to exploit these characteristics. Two benchmark nonlinear problems with 1715 and 3230 degrees of freedom are selected to measure the anticipated gain in speed obtained by using the algorithms developed on the array processor. The estimated gains in central-processing-unit (CPU) speed for the two problems were 9.2 and 18.1; however, the reductions in the total solution times, central processing and input-output (CP + I/O), were only 5.2 and 9.9.

The following major hardware characteristics of the Prime 750/AP-120B system have been identified as having a significant impact on nonlinear finite-element computations:

(1) **Virtual memory** of the host Prime 750 supermini consisting of the three levels: cache memory, central memory, and disk, with a total storage capacity of over 600 million 32-bit words.

(2) **Pipeline processing** through the multiple pipelined floating-point arithmetic elements (adder and multiplier) of the array processor.

(3) **Parallel processing** achieved on three levels: (a) on the array-processor level, all memory and arithmetic units can perform operations in parallel; (b) on the host level through parallel CP and I/O operations; and (c) between the host and array processor (AP).

(4) **Distributed processing**: The computationally intensive portions of the program are allocated to the AP; and the other tasks, such as data management, control of the AP, and user interface, are allocated to the host supermini.

To exploit the aforementioned hardware features of the minicomputer/array-processor system in a large-scale, nonlinear, finite-element analysis, the following tasks were performed:

(1) **Proper selection of the computational algorithm**: Although a comprehensive study of the different computational algorithms was not made, the reduced-basis technique appears to be very suitable for use on minicomputer/array-processor systems.

(2) **Minimization of I/O operations** through the following: (a) the use of buffering techniques on the host computer; (b) treatment of the table memory and main-data memory of the array processor as a two-level virtual memory; (c) merging smaller arrays into larger ones; and (d) chaining several array-processor routines into a single routine in order to reduce the number of subroutine calls.

(3) **Vectorization of numerical algorithms**: Although most of the computationally intensive processors of the program have been vectorized in the present study, the efficiency of vector operations on arrays with a packed-data structure needs improvement.

(4) **Synchronization of parallel operations on the host and array processor.**

On the basis of the present study, the following two conclusions seem to be justified:

(1) The use of the aforementioned computational strategies and programming techniques developed in the present study make large-scale, nonlinear, finite-element computation on the minicomputer/array-processor system feasible. The computational speeds achieved are comparable to those of the large mainframes.

(2) The peak-performance rates of the AP-120B at 12 million floating-point operations per second (12 MFLOPS) cannot be achieved in practical finite-element computation. Computational speeds of 2 to 3 MFLOPS are possible for some of the vector operations. Further improvement in speed requires the development of low-level microcodes to take advantage of the parallelism in the system. This task can be fairly complicated.

Langley Research Center
National Aeronautics and Space Administration
Hampton, VA 23665
January 20, 1983

10 APPENDIX A

DISCUSSION OF MAJOR-HARDWARE AND SYSTEM-SOFTWARE FEATURES OF MINICOMPUTER/ARRAY-PROCESSOR SYSTEM

The memory organization and computational speed of the host Prime 750 superminicomputer are discussed in this appendix along with the memory organization, arithmetic units, executive software, and the simulator of the AP-120B array processor.

10.1 Prime 750

10.1.1 Memory organization.- A basic feature of the Prime 750 minicomputer at the Langley Research Center is its three-level memory organization depicted in figure 2. The first level of storage is the **cache memory** with a capacity of 4000 words (16K bytes) where each word is 32 bits (approximately 8 significant digits). The second level is the **central memory** whose capacity is 0.5 million words (2M bytes), and the lowest level of storage is a **moving-head-disk** system with a total capacity of 75 million words (300M bytes). The central processing unit **can reference only data and code stored in the cache memory.** Cache memory provides the CPU with high-speed access to central memory. It contains information that is a duplicate to that of central memory. The Prime 750 has an advertised cache-hit rate of 95 percent, meaning that 95 percent of the memory accesses requested by the CPU are found in cache memory. This reduces central-memory access time from a nominal of 600 ns to an effective 110 ns (cache-memory access time is 80 ns), thus allowing faster memory access for critical-performance portions of the code.

Transfer rates between the three memories are drastically different. The transfer rate from disk to central memory is 2 million words per second, not including setup time for each access. The setup time, which is the time needed to service the request and find the data on the disk, averages 600 ms. A maximum of one segment (32 000 words) can be transferred per disk call (8 ms). The average disk setup and transfer times range from 600 to 608 ms. Central-memory access times are 64.6 times faster than disk access time for an average segment transfer. Access times on cache memory are 7.4 times faster than those on central memory.

The memory system of the Prime 750 can be viewed from two different levels:

10.1.1.1 User level: The user views central memory as being divided into segments of 32 000 words each. The user program allocates central-memory space for the storage of data used by the program and can create user buffers for transferring data from central memory to disk. Special precautions have to be taken when user buffers cross segment boundaries.

10.1.1.2 System level: The system views and manages memory from a higher level than the user. If the user program and data cannot fit into central memory, the operating system decides which pages (groups of 512 words) will reside in central memory and which pages are transferred to disk. When the program references its address space for data located on disk, a page fault occurs and the desired page of data is brought into central memory while another page is moved out. This automatic overlay system is referred to as **virtual memory.**

The system also manages the use of cache memory through a microcoded program. When the CPU requests data from cache and the data are not there, the data are copied from central memory (which obtains pages from disk) in two-word blocks. The second word is automatically copied into cache to reduce the number of central-memory accesses in a prefetching procedure. The CPU can now access the data item. This creates an essentially two-level paging system. (See, for example, ref. 25.) The system memory-management procedures are invisible to the user. The system also creates I/O buffers that are invisible to the user. The size of some of these system I/O buffers can be controlled by the user as will be illustrated in a later section.

Note that if the Prime 750 system routines are used for I/O operations, then the data must be contained within one segment or a condition of "wrap around" occurs. As an example of "wrap around," consider an array {X} which has the dimension 10. If the first 5 elements of {X} are located at the end of 1 segment and the remaining 5 elements of {X} are located at the beginning of an adjacent segment, then a user request for the 10 items {X} from disk results in reading the first 5 items correctly. However, the second 5 are read from the data and/or the code at the beginning of the first segment instead of being read from the beginning of the second segment, that is, locations 6 to 10 of {X}. This condition necessitates the careful outlay of a program's central memory and I/O operations to the disk. As a result, 32 000 words (1 segment) is the maximum size of a user I/O buffer. This maximum is allowed only when the buffer starts at the beginning of a segment. Note that the wrap-around problem occurs only with disk I/O operations using Prime system routines. User buffers larger than one segment can be used, provided the Prime system I/O routine is not used.

10.1.2 Computational speed.- In table AI the computational speed of the Prime 750 is compared with that of other minicomputers and mainframes for a number of benchmark structure calculations. (See ref. 26.) The test was performed in a single-user environment. Table AI illustrates the performance measured in terms of CPU times available in the minicomputer cost range. The addition of an array processor to the Prime 750 system achieves the mainframe performance (i.e., CPU speeds) while remaining well below the mainframe cost range.

Timings of the nonlinear finite-element code described in subsequent sections were measured by using Prime system routine "TIMDAT." In order to verify the manufacturer's time estimates (ref. 4), a small test program was developed and used to measure the speed of various floating-point operations. Table AII shows that the measured timings fall within the manufacturer's range for each floating-point operation. The range of speed shown in the table occurs because in a multiuser environment the user has no control over whether cache memory or central memory is used for the double floating-point load operation. Therefore, all timings presented were measured in a single-user environment for consistency.

10.2 AP-120B Array Processor

The overall architecture of the Floating Point Systems AP-120B array processor is based on multiple special-purpose memories feeding two floating-point pipelined arithmetic units via multiple data paths. The details of the memory organization and arithmetic units are discussed subsequently.

10.2.1 Memory organization.- Memory of the AP-120B includes main-data memory, table memory, program memory, data-pad registers, and address registers. (See fig. 3.) A description of the function of each memory, including its size and word length, is given as follows:

10.2.1.1 Main-data memory: Main-data memory is the main storage file for the array processor and serves as the main interface (buffers) for communication with the host computer. It is composed of memory banks of 4000 or 16 000 38-bit words and is expandable to 320 000 38-bit floating-point words. Main-data memory is available in two speed ranges: 333 ns (slow) and 167 ns (fast). These times indicate the frequency with which a program can reference main-data memory. All AP-120B timings in this paper assume the use of fast main-data memory.

10.2.1.2 Table memory: Table memory is used to store floating-point constants and slowly changing data. This memory is available in ROM (read only memory) or RAM (random access memory) with a maximum size of 64 000 38-bit words.

10.2.1.3 Program memory: Program memory stores AP-120B programs in a maximum of 4000 64-bit words.

10.2.1.4 Data pads: Data-pad registers are referred to as X and Y. Each can store 32 intermediate floating-point numbers. Two sets of registers are needed since the floating-point multiplier and adder require different operands with parallel-access capability.

10.2.1.5 Address registers: The 16 address registers contain integer data items of size-16 bits. The inability of the AP-120B to achieve the maximum of 12 MFLOPS in most floating-point arithmetic operations is due to insufficient parallel data flow to keep the floating-point multiplier and adder fully occupied. Figure A1 details the data flow and interconnection network of the AP-120B.

The user has complete control over the flow of data in the AP memories. This is a complex task when timing characteristics of the different memories are considered. The following discussion will first address a timing characteristic called "**interleave**" which is peculiar to the main-data memory, and then the timing characteristics of other memories within the AP will be discussed.

Interleave is the sequential reference to different banks of main-data memory. Main-data memory is divided into banks to facilitate faster access than standard main-data memory, which takes three instruction cycles (500 ns). Each bank holds odd or even memory locations. Memory references to main-data memory are now allowed every 333 ns for slow memory and 167 ns for fast memory as long as interleave is not broken. If interleave is broken, the AP executes a "spin" operation which halts all processing until the memory-reference instruction can be satisfied. The AP assembler programmer must be careful that his data are accessed sequentially and that main-data-memory accesses have at least one instruction cycle between them for slow main-data memory. The instruction cycle on the array-processor hardware is 167 ns. All separate memories and arithmetic units in the AP are synchronized to the instruction cycle. For example, the contents of a main-data-memory location become available to the program three instruction cycles after the main-data-memory address register has been altered. Reference instructions to main-data memory are allowed only every 333 or 167 ns with the assumption that interleave is not broken. Data from table memory are available two instruction cycles after the table-memory access register is changed. Data from address registers and data-pad registers are available during the instruction cycle in which they are referenced. Data-pad X and data-pad Y cannot be referenced in the same instruction cycle. Further timing complications occur when considering the pipeline architecture of the floating-point adder and multiplier.

10.2.2 Arithmetic units.- The floating-point arithmetic units in the AP-120B have pipeline architecture which segments arithmetic computation into a sequence of basic operations (such as exponent comparison, coefficient alignment, addition, and normalize shift). The arithmetic unit can perform basic operations simultaneously on independent pairs of data elements, with each pair at a different stage in the computation.

There are two types of data elements on the array processor, scalar and vector. A **scalar** is a single-valued element of data usually contained in one storage word. A **vector** is an array of scalars usually contained in either contiguous locations of memory or at fixed intervals from each other. If the same operation is performed on successive data pairs (vector operation), results are available as frequently as 167 ns for some operations. Such operations are called vector operations and can lead to substantial gains in speed over scalar arithmetic operations where one operation is performed at a time.

Floating Point Systems, Inc., designed the arithmetic units of the AP-120B for a balance of vector and scalar capabilities. (See ref. 11.) This results in a vector processor that performs well on short vectors and sequential operations. Implementation of this idea involved adjusting the cycle time of the AP to shorten the arithmetic-unit pipelines to only two or three stages. Other vector processors, for example, the CDC STAR-100, have much longer pipelines and perform poorly on scalar operations.

Each vector operation on the array processor has associated with it a delay before the first result emerges from the pipeline. This time is called start-up time and is independent of the vector length. The start-up times on the AP-120B range from 333 to 500 ns for various floating-point operations.

The AP-120B has a two-stage pipeline adder and a three-stage pipeline multiplier. (See fig. A2.) Each can perform up to 6 million floating-point operations per second (6 MFLOPS).

10.3 AP Executive Software (APEX)

A table is maintained by APEX that describes the contents of the program-source memory in the AP. When the AP subroutine is called from the host program, APEX checks the subroutine table to see if the called routine is present in program-source memory. If the routine is not in memory and space is available in program-source memory, the program instructions are loaded normally. The routine's name and offset are added to APEX's table. However, if space is not available, APEX overwrites routines in program-source memory on a last-in, first-out basis. APEX loads parameters from the array-processor subroutine call statement into address registers and initiates execution of the program by the AP-120B. The time that elapses between the array-processor subroutine call in the FORTRAN program and the execution of the instructions in the AP-120B is "host overhead" for that call. This overhead is typically from 100 to 1000 μs per array-processor subroutine call. (See ref. 8.) Host overhead becomes very significant with a large number of calls to the array processor.

An I/O processor and programmable I/O processor are provided to allow direct-data transfer to a peripheral storage device (disk) by the array processor. Dependence on the host interface is thereby greatly reduced.

APPENDIX A

On the array-processor side the user can code operating-system functions to facilitate communication with the outside world, of which the host computer is a part. This is accomplished through direct communication with both the host operating system and APEX.

Communication and data-transfer operations between the AP and host computer are processed by a combination of hardware and software. There are three basic stages to consider:

(1) Before the array processor is used, its presence has to be acknowledged by the host computer through an initialization procedure.

(2) APEX is loaded as part of the host load module. (See fig. 4.)

(3) APEX is activated, thus allowing the array processor to run as an independent computer.

There are three ways to control the attached processor:

(1) Direct control: Function control blocks are generated, transferred, and executed separately.

(2) Function lists: A group of function control blocks are generated and transferred to the AP. These function lists can be executed many times without transfers or regeneration.

(3) User written-assembler routines: The most effective and most time consuming routine involves writing the program directly for the AP, thereby making the best use of its characteristics. This is required for algorithms involving sparse matrices and operations with overpacked data structures.

10.4 Comments on Use of AP-120B Simulator

Although the AP-120B simulator (APSIM) was very useful in assessing the performance of the array processor AP-120B, the following four difficulties were observed in using it:

(1) The simulation is limited to **program execution on the array processor**, and the interaction with the host computer is not simulated.

(2) The size of the main-data memory on the simulator is considerably smaller than that of the AP-120B (8192 words as compared with a maximum of 1-million words on the AP-120B). Therefore, smaller-size arrays and vectors are used.

(3) The single command that allows AP subroutines to be called from the AP assembler program (JSR command) is not operational on the simulator. This necessitated the use of two instructions (JUMP and BRANCH) to perform the same operation.

(4) The computational speed of the simulator is dependent on the speed of the host computer and is considerably slower than that of the array processor. Therefore, for effective use of the simulator in the present study, small sample problems were used to debug the code developed, and then the larger problems were run in a batch mode. Computation times were recorded to assess the gain in speed obtained by using the array processor.

APPENDIX A

TABLE AI.- CPU TIMES FOR BENCHMARK PROBLEMS OF STRUCTURES LISTED IN REFERENCE 26

Mainframe (cost range, \$2M to \$6M)	Time, sec	Minicomputer (cost range, \$50K to \$300K)	Time, sec
CDC CYBER 175	2	DEC VAX-11/780	23
IBM 360/95	3	Prime 750	32
IBM 370/168	4	DEC PDP-11/70	42
CDC 6600	8	Prime 500	47
IBM 360/75	10	S.E.L. 32/75	52
CDC CYBER 173	12	Prime 400	65
UNIVAC 1108	15	SIGMA V	71
		S.E.L. 32/55	72
		MODCOMP IV	85

TABLE AII.- ESTIMATED AND MEASURED TIMING FOR DOUBLE-PRECISION (64-BIT) FLOATING-POINT OPERATIONS ON THE PRIME 750 COMPUTER

Operation	Time, μs	
	Estimated (ref. 4)	Measured
Load	0.47	
Store	[a]0.93 to 2.61	
Add + Load + Store	[a]3.17 to 4.85	3.45
Multiply + Load + Store	[a]6.14 to 7.82	6.42
Divide + Load + Store	[a]9.39 to 11.07	9.89
Loop overhead	[a]3.02 to 3.14	

[a]Range shown is due to fast cache memory and slower central memory.

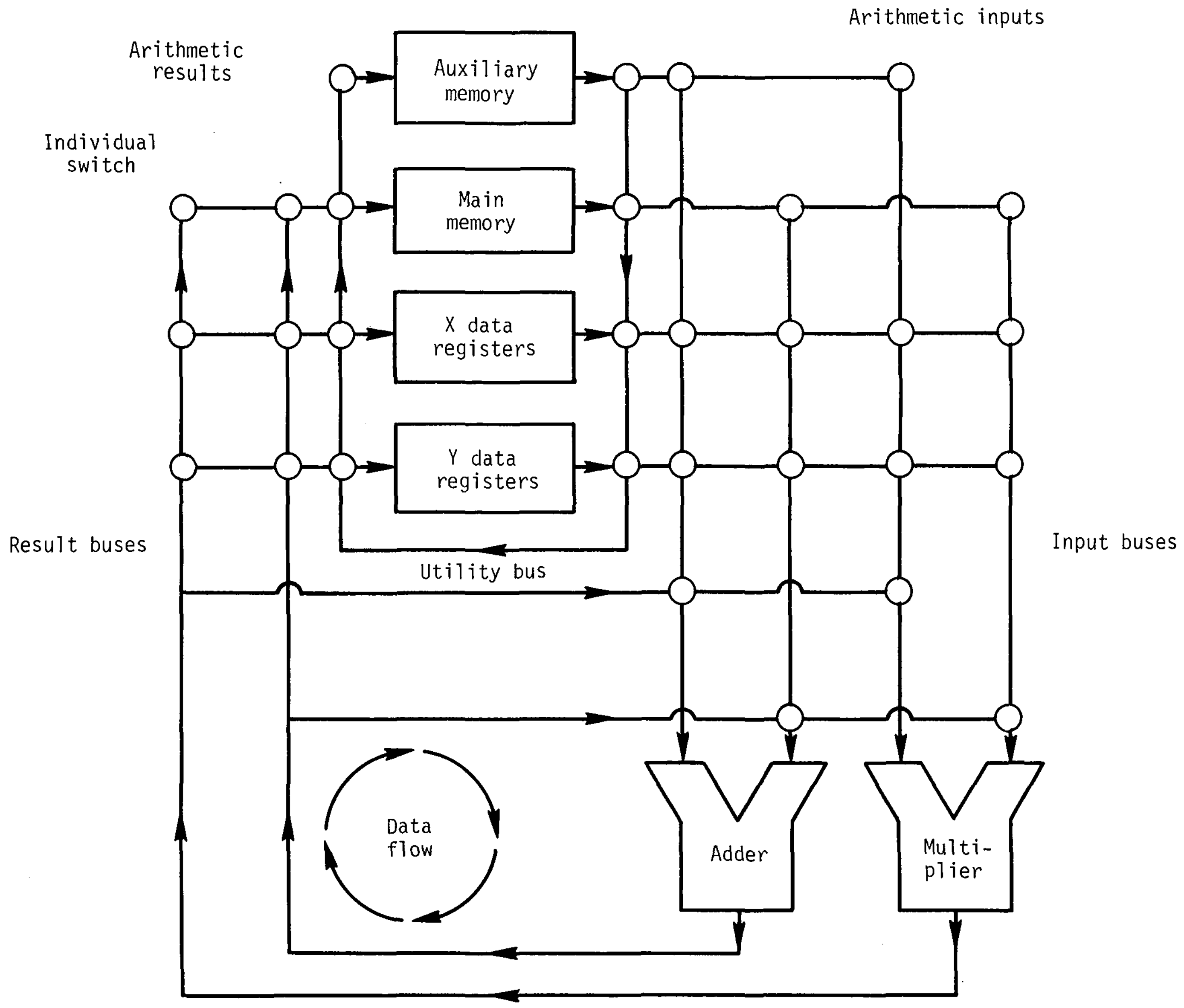


Figure A1.- The AP interconnection network and data flow.

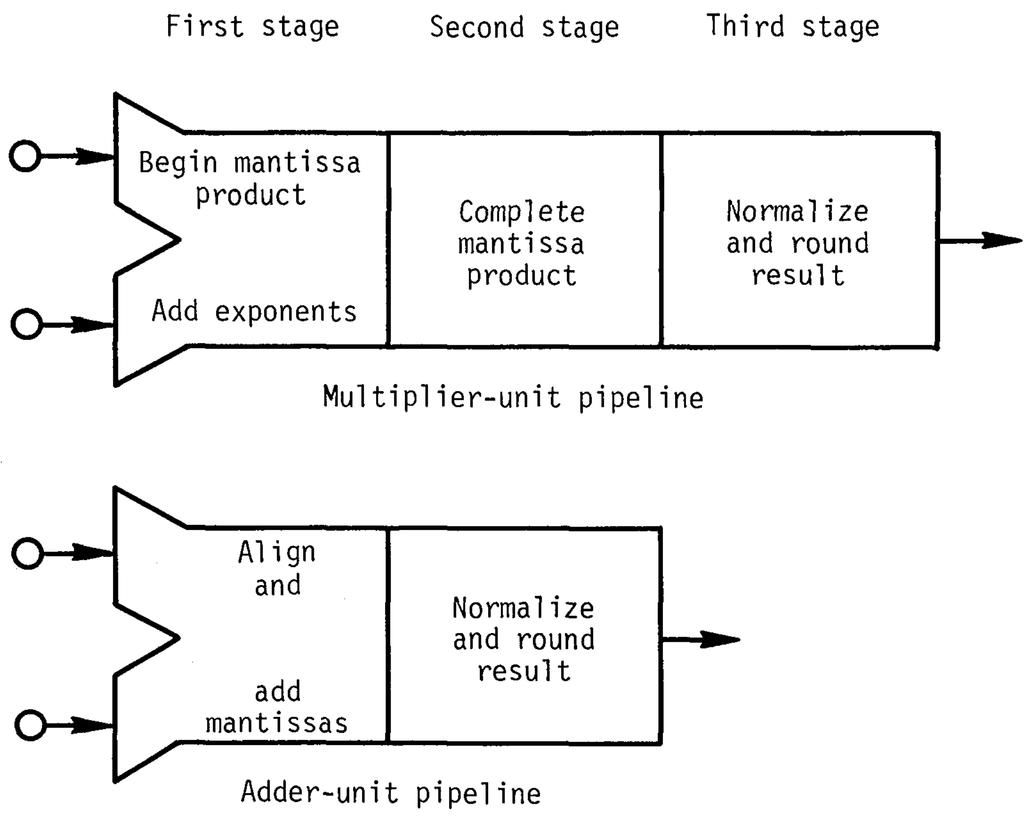


Figure A2.- The AP-120B arithmetic pipelines.

11 APPENDIX B

PROGRAM AND PROCESSOR ORGANIZATION

The program organization, control flow, and data flow for the computational procedure discussed in sections 4.1 and 4.2 are discussed in this appendix. The program is divided into self-contained units or processors whose organization is also discussed herein.

11.1 Program Organization

A computer program based on the reduced-basis technique was implemented on the CDC CYBER 175 computer under the NOS operating system at the Langley Research Center in a mostly batch-oriented environment. In the present study, the program was modified to make it more suitable for the interactive environment of the Prime 750 and to exploit the hardware characteristics of the minicomputer/array-processor system. The major features of the modified program are as follows:

(1) The program is divided into small self-contained logical sections or **processors.** Each processor can run as a separate program and complete its execution in an interactive time frame, thereby increasing the user control over the flow of computation.

(2) Communication between different processors is made through a common data base. The data-base complex utility (DCU) of the finite-element program system SPAR (see ref. 16) was selected for performing the data-manipulation and data-management tasks in the modified program. The DCU has been designed to minimize the disk I/O operations on the Prime computer.

(3) Most of the arithmetic operations were **vectorized** to exploit the pipeline-processing capability of the array processor, and an attempt was made to overlap the host and array-processor operations. The details of the vectorization of the numerical algorithms, programming techniques used for overlapping the host and array-processor operations, and minimizations of I/O operations are described in sections 5 and 6. A flow chart of the modified program is shown in figure B1. For the sake of clarity, a distinction is made between the full- and reduced-system processors. The former use full-system (global) arrays and vectors, and the latter use reduced-system arrays and vectors. The control flow, data flow, and processor organization in the modified program are discussed subsequently.

11.2 Control Flow

The control flow of the nonlinear finite-element program depicted in figure B1 represents the normal sequence of program execution. In addition to the normal sequence, the program design has the flexibility to allow interaction by display processors, alteration of executing data sets or program parameters, or elimination of the execution of a processor whenever needed. The following comments concerning the control flow described by figure B1 are in order:

(1) The individual blocks in the group of boxes marked "solve nonlinear reduced equations" are not numbered because they constitute a portion of the reduced-system solve processor.

(2) The reduced arrays referenced in box 6 are $\tilde{K}_{ij}$, $\tilde{F}_{ijk}$, $\tilde{G}_{ijk\ell}$ and $\tilde{Q}_i$.

(3) Box 9, which is designated "recover full solution," is considered as part of the reduced-system processor since reduced-system data are used to form the full-system vector $\{X\}$.

(4) The interface between the full and reduced systems is controlled by the decision box between the processors in boxes 17 and 18. This interface is important because full-system analysis is considerably more expensive than reduced-system solutions.

11.3 Data Flow

During execution of a processor a data-base file is created which is called a library. Each library is identified by a table of contents whose entries contain the following information (see refs. 27 and 28):

(1) A sequence number to determine the order of data-set creation

(2) A number indicating the relative location on disk where the data start

(3) The date and time of data-set creation

(4) An integer-error code

(5) The number of words in the data set

(6) The number of columns and number of columns times rows in the data set

(7) An integer-data-type code

(8) A four-part data-set name

The information in the table of contents completely describes the data being stored and allows the user to access these data, print it, copy it to another library or data set, write it to tape, or retrieve it from tape. The data sets can also be assigned symbolic names and be used in equations involving matrix and vector computations.

11.4 Processor Organization

The routines represented in the flow chart in figure B1 were grouped into separate processors or functional units and were connected through the common data-base DCU. For example, F and G arrays are formed by a single processor (fig. B1, box 1) and are stored on the common data base. These data are retrieved from the data base and are used to generate the basis vectors (fig. B1, box 5) in the processor called "generate basis vectors." The individual processors can be run in any sequence specified by the user, provided the data needed at any stage are available in the data base.

APPENDIX B

The following advantages for dividing the program into smaller processors connected through a common data base can be identified:

(1) Minimal central-memory requirements: Only COMMON BLOCKS and subroutines needed for the specific calculation are included in the processor.

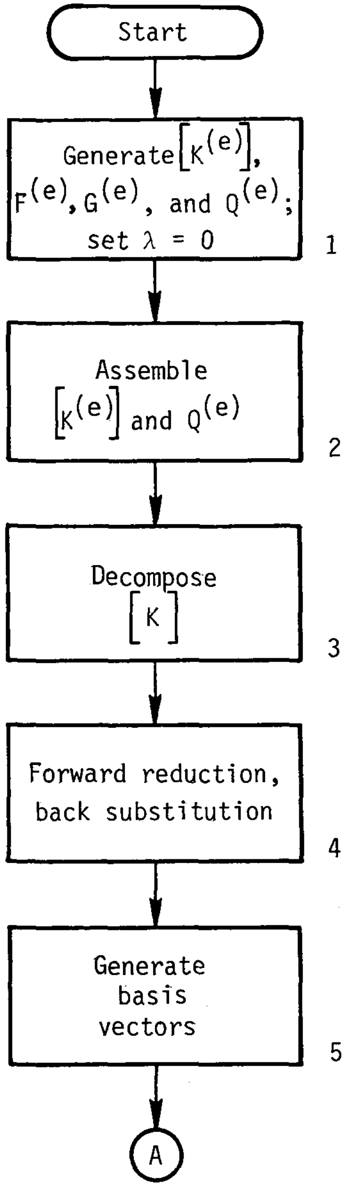
(2) User interaction and convenience: This includes interactive program development and debugging, easy numerical and/or graphical display of data, and easy manipulation and modification of data between processor executions.

(3) Faster run time: The wall-clock time can be reduced if the program is divided into processors. This is because the scheduling algorithms for most multi-user operating systems give higher priority to jobs that require small system resources, and the individual processors generally require smaller central-memory space during execution than that required by the entire program when it is run in a batch mode.

(4) Reduction of program-development time: Once a processor has been debugged, its data have been generated and sent to the data base. That processor need not be loaded or executed in subsequent debugging runs. Also, having checked the correctness of data on the data base, the debugging is confined to the processor being developed.

Adaptation of existing or new programs to the finite-element system is greatly simplified. Programs (e.g., graphical, data base, or computational) are easily connected to the data-base system through the SPAR FORTRAN interface. This is a significant saving in program conversion and development time.

(5) I/O optimization: The data-base system used in conjunction with the processor organization centralizes the places where I/O to the disk occurs. This allows easy optimization of I/O for the program. The Prime system routines were used in the SPAR program to minimize I/O operations.



Start
Generate $[K^{(e)}]$, $F^{(e)}$, $G^{(e)}$, and $Q^{(e)}$; set $\lambda = 0$
1
Assemble $[K^{(e)}]$ and $Q^{(e)}$
2
Decompose $[K]$
3
Forward reduction, back substitution
4
Generate basis vectors
5
A

FULL SYSTEM REDUCED SYSTEM

Figure B1.- Program organization. The paired blocks (3,12), (4,10), and (5,15) use the same subroutines.

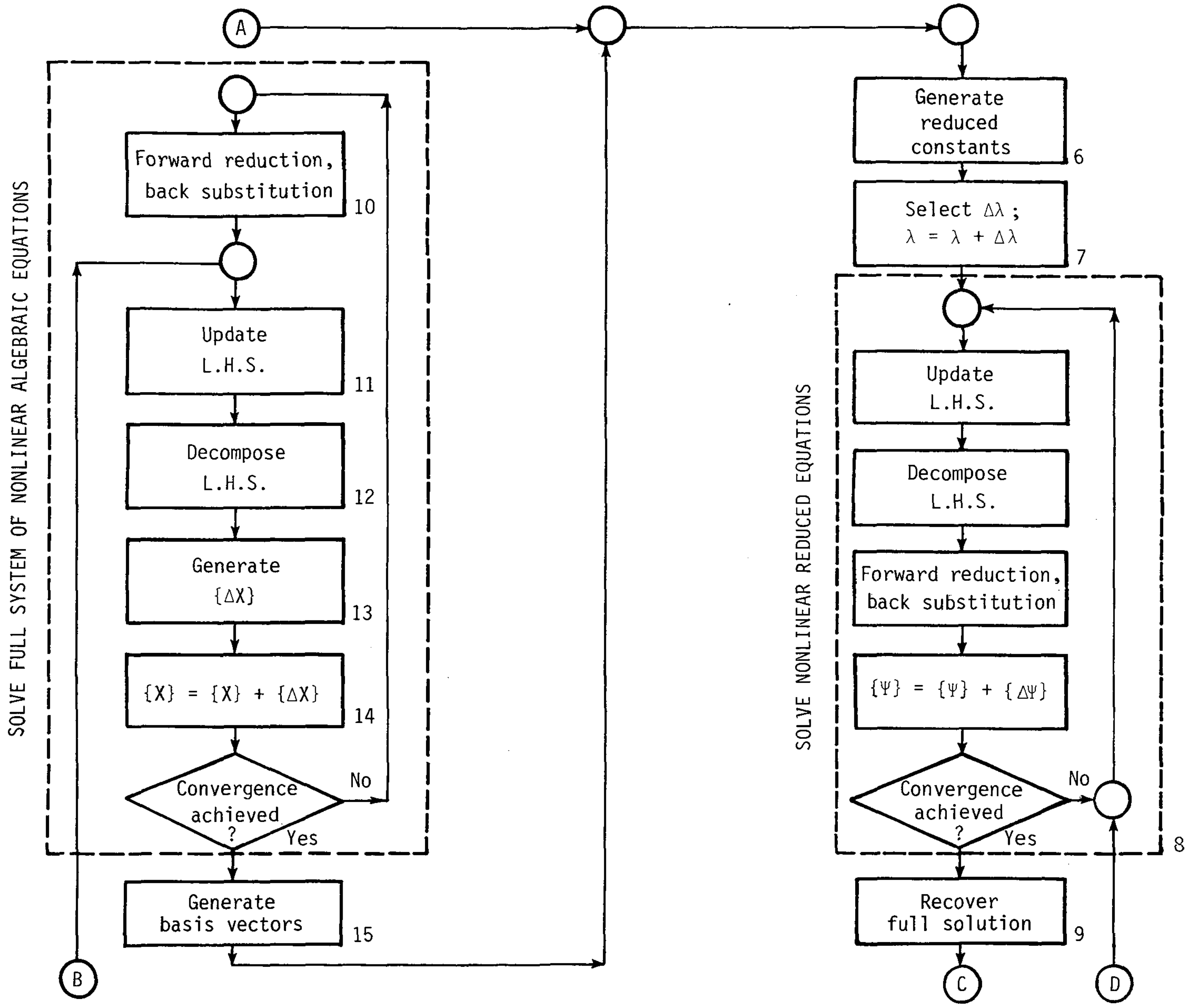


Figure B1.- Continued.

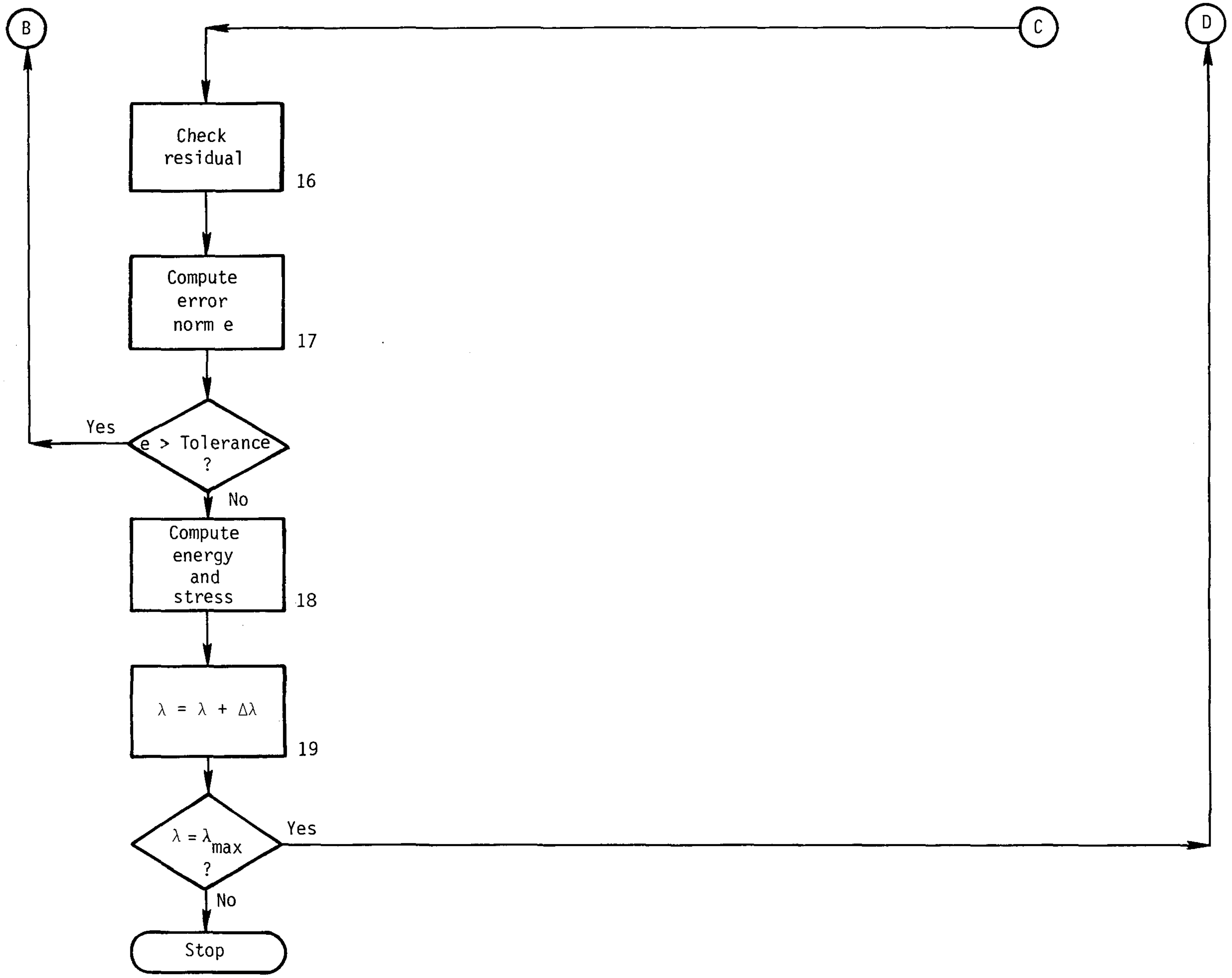


Figure B1.- Concluded.

12 APPENDIX C

DETAILS OF VECTORIZATION OF BASIC OPERATIONS

Some details of the vectorization of the basic operations listed in table II are discussed in this appendix. However, extensive work was done to improve the efficiency of the vector operations which is not detailed here. The data structures selected for the different operands are denoted by the symbols A, B, C, D, and E and are identified in table CI. A list of the array-processor routines used in the present study is given in table CII. Some of the vector operations were performed by using the math-library routines of the array processor; others required the development of special routines in assembler language.

12.1 Evaluation of Linear- and Nonlinear-Stiffness Arrays

12.1.1 Linear-element-stiffness matrix.- A linear-element-stiffness matrix $[K^{(e)}]$ can be conveniently expressed in the following form (see ref. 29):

$$[K^{(e)}] = \sum_{\ell=1}^{n} w_\ell \, [B]_\ell^T \, [C] \, [B]_\ell \qquad \text{(C1)}$$

$$w_\ell = H_\ell \left(\det[J]\right)_\ell \qquad \text{(C2)}$$

where $[K^{(e)}]$ is an 80 × 80 matrix (16-node quadrilateral element with five degrees of freedom per node), H_ℓ is a weighting coefficient, n is the number of quadrature points, [B] is an 8 × 80 linear-strain-displacement matrix, [C] is the material-stiffness matrix, and det[J] is the Jacobian matrix of the transformation from local to natural (dimensionless) coordinates in the element domain.

Two algorithms were tested for the vectorization of the operations involved in equation (C1). In the first algorithm, microcode was developed for use on the array processor to form the upper triangular portion of $[K^{(e)}]$. Vectors of variable length had to be used in order to take advantage of the symmetry of the matrix $[K^{(e)}]$. This resulted in increasing the overhead (associated with the extra data manipulation), with a consequent reduction in the speedup due to vectorization. The second algorithm used the library routines of the array processor (see table II) to form the full matrix $[K^{(e)}]$. No advantage was taken of the symmetry of this matrix. The formation of the full matrix $[K^{(e)}]$ using the second algorithm was found to be 5 percent slower than the generation of the upper triangular portion using the first algorithm. However, if the time required to generate the elements of the lower triangular portion of $[K^{(e)}]$ during the assembly process is taken into account, the second algorithm turns out to be faster and, therefore, it was used in the present study.

12.1.2 Generation of the nonlinear array $F^{(e)}_{IJK}$.- The nonlinear array $F^{(e)}_{IJK}$ is a sparse, completely symmetric, three-dimensional array with dimensions 80 × 80 × 80. For convenience, $F^{(e)}_{IJK}$ is partitioned into nonzero blocks of 16 × 16 × 16. (See fig. C1.) The blocks are rearranged so that the central block is in the upper left-hand corner as depicted in the right half of figure C1. Because of symmetry, only the corner block and four vertical blocks need to be generated. Moreover, the corner block is doubly symmetric (i.e., symmetric in two directions), and the four other blocks are symmetric in one direction.

The array $F^{(e)}_{IJK}$ was generated by using the array processor math-library routines listed in table II. Each block was formed as a full array. Then, it was reduced to symmetric form before being stored on the data base. The speedup ratio obtained by using the array processor was 4.3.

12.1.3 Generation of the nonlinear array $G^{(e)}_{IJKL}$.- The four-dimensional array $G^{(e)}_{IJKL}$ is completely symmetric. Generating $G^{(e)}_{IJKL}$ was attempted by using the array-processor math-library routines, but it failed to improve the performance because of the large amount of symmetry in the G array. Data preparation by the host computer for the array-processor library routines took longer than the time required by the original host routine to form G. Therefore, a microcoded routine had to be developed. The speedup ratio obtained by using this routine on the array processor was 14.4.

12.2 Vectorization of Newton-Raphson Iterative Technique

The recurrence relations for the Newton-Raphson iterative technique can be written in the following form:

$$[\overset{*}{K}^{(r)}]\{\Delta X^{(r)}\} = -\{f^{(r)}\} \tag{C3}$$

and

$$\{X^{(r+1)}\} = \{X^{(r)}\} + \{\Delta X^{(r)}\} \tag{C4}$$

where $\{\Delta X^{(r)}\}$ is the correction to the nodal-displacement vector $\{X^{(r)}\}$ obtained from the rth iteration cycle. The elements of the matrix $[\overset{*}{K}^{(r)}]$ and the vector $\{f^{(r)}\}$ are given by

$$\overset{*}{K}_{IJ} = K_{IJ} + F_{IJK}\, X_K + G_{IJKL}\, X_K\, X_L \tag{C5}$$

and

$$f_I = \left(K_{IJ} + \frac{1}{2} F_{IJK} X_K + \frac{1}{3} G_{IJKL} X_K X_L\right) X_J - qQ_I \tag{C6}$$

An error norm $\bar{e}$ is introduced as follows:

$$\bar{e} = \frac{1}{N}\sqrt{\frac{\{\Delta X^{(r)}\}^T\{\Delta X^{(r)}\}}{\{X^{(r+1)}\}^T\{X^{(r+1)}\}}} \tag{C7}$$

where N is the total number of degrees of freedom. The iteration is terminated when either $\bar{e}$ is less than a prescribed tolerance or the number of iterations reaches a preselected number. For the first load increment, the initial estimate of $\{X\}$ is chosen to be

$$\{X^{(0)}\} = \Delta q^{(1)}\left\{\frac{\partial X}{\partial q}\right\}_{q=0} \tag{C8}$$

The global-stiffness matrix $[\overset{*}{K}^{(r)}]$ in equation (C3) was partitioned by rows and columns into blocks which can be regarded as elements of a **hypermatrix**; this is analogous to the way that numbers are the elements of a matrix. An address or pointer matrix is used to identify the location of the nonzero submatrices in the hypermatrix. Zero submatrices are identified by a zero entry in the address matrix and are neither built nor stored. The vectors $\{\Delta X^{(r)}\}$ and $\{f^{(r)}\}$ are partitioned into subvectors which are consistent with the hypermatrix blocks.

The vectorization of the Newton-Raphson iterative technique can be conveniently divided into three phases:

(1) Vectorization of the evaluation of the right-hand side of equation (C6)

(2) Vectorization of the solution of equation (C3) by using the Cholesky method with hypermatrices

(3) Vectorization of the solution update and convergence check of equations (C4), (C7), and (C8)

The first phase can be accomplished by forming the full arrays $F^{(e)}$ and $G^{(e)}$ for the individual elements from their symmetric portions and contracting them with the nodal displacement $\{X^{(e)}\}$. The details of the latter two phases are discussed subsequently.

12.3 Vectorization of Cholesky's Method With Hypermatrices

12.3.1 <u>Brief summary of the basic method</u>.- The Cholesky method for solution of the global equations $[\overset{*}{K}]\{X\} = \{Q\}$ is expressed by the three hypermatrix equations:

Decomposition:

$$\left[\overset{*}{K}\right] = [U]^T[D][U] \tag{C9}$$

Forward reduction:

$$[U]^T\{Y\} = \{Q\} \tag{C10}$$

Back substitution:

$$[D][U]\{X\} = \{Y\} \tag{C11}$$

where $[U]$ is an upper triangular hypermatrix and $[D]$ is a diagonal hypermatrix (block diagonal matrix). The vector $\{Y\}$ is obtained by a forward reduction of the load vectors, and the displacement vector $\{X\}$ is then calculated by back substitution.

12.3.2 Organization of the hypermatrix computation.- In the Cholesky method, modified for the hypermatrix approach, the submatrices play the role of the elements in the ordinary (scalar) Cholesky method. This requires six basic operations on submatrices to be performed repeatedly in the solution process. (See eqs. (C9) to (C11).) These are as follows:

(1) Decomposition of a square positive-definite symmetric submatrix: $[A]$

(2) Product of an inverse of a square matrix and a rectangular matrix: $[A]^{-1}[E]$

(3) Difference between a matrix and a product of three matrices of the form $[H] - [E]^T[A][E]$

(4) Difference between a matrix and a product of two matrices of the form $[H] - [E]^T[S]$

(5) Product of an inverse of a square matrix and a vector: $[A]^{-1}\{Q\}$

(6) Difference between a vector and a product of a matrix and a vector of the form

(a) $\{Q\} - [E]\{R\}$

and

(b) $\{Q\} - [E]^T\{R\}$

where [A] represents any of the diagonal submatrices in the hypermatrix [K]; [H] is a symmetric submatrix; [E] and [S] represent any of the off-diagonal submatrices; and {Q} represents a vector. In operation (2) above the inverse is not formed explicitly. Rather, [A] is decomposed and the product $[A]^{-1}[E]$ is formed by forward reduction and back substitution. The six basic operations above, (1) to (6), will, henceforth, be referred to as **macrooperations.**

The first four macrooperations above, (1) to (4), are used in the decomposition phase, and the last two above, (5) and (6), are used in the forward-reduction/back-substitution phases (forward/back solve) of the solution process. The vectorization of the three macrooperations (3), (4), and (6) was accomplished by using the math-library routines of the array processor. On the other hand, the vectorization of the macrooperations (1), (2), and (6) required the development of a microcoded routine since no appropriate library routines are available to date for performing these tasks. The gain in speed obtained by using the array processor for each of the aforementioned macrooperations is given in table CIII. Note that the gain in speed in the operation $\{Q\} - [E]^T\{R\}$ is considerably less than that for $\{Q\} - [E]\{R\}$ because of the slow matrix transposition on the array processor.

12.4 Vectorization of Solution Update and Convergence Check

The operations involved in the solution update and convergence check of the Newton-Raphson technique are described by equations (C4), (C7), and (C8). The vectorization of these operations was accomplished by using the three math-library routines on the array processor: namely, vector addition (eq. (C4)), sum of the squares of the vector elements (eq. (C7)), and scalar times a vector (eq. (C8)).

In the vectorization process the full vector $\{\Delta X\}$ was constructed by using its subvector partitions obtained from the Cholesky hypermatrix-solution process in each iteration. In spite of the associated storage penalty with the formation of the full vector $\{\Delta X\}$, considerable improvement in the performance was obtained because of the reduction in the number of disk accesses and the number of calls to the array processor.

12.5 Vectorization of Evaluation of Basis Vectors and Generation of Reduced Arrays

The explicit form of the recurrence relations for the basis vectors is given in reference 15. As mentioned before, the same matrix $[\overset{*}{K}]$ appears on the left-hand side of each of the recurrence relations. Therefore, the evaluation of the basis vectors involves: (1) decomposition of the matrix $[\overset{*}{K}]$ once, (2) evaluation of the right-hand side of each of the recurrence relations, and (3) forward reduction/back substitution to generate each basis vector. Vectorization of the operations involved in (1) and (3) has already been discussed in the preceding sections. The evaluation of the right-hand side of each of the recurrence relations involves contraction of the arrays $F^{(e)}$ and $G^{(e)}$ with nodal-displacement vector and its various-order path derivatives. (See ref. 15.) This is accomplished by converting the arrays into full matrices and performing matrix-vector multiplication.

The procedure outlined in reference 15 was adopted in the present study for efficiently generating the reduced arrays $\tilde{K}_{ij}$, $\tilde{F}_{ijk}$, and $\tilde{G}_{ijk\ell}$ by taking advantage of the operations performed to evaluate the right-hand sides.

12.6 Vectorization of Strain-Energy and Stress-Resultant Computation (Post Processor)

The stress resultants and strain energy can be expressed in the following compact form:

$$\{\sigma\}_i = [C][B]_i\{X^{(e)}\} + [C]\{\bar{B}(X^{(e)})\} \tag{C12}$$

and

$$U = \sum_{\text{Elements}} \left(\frac{1}{2}\{X^{(e)}\}^T[K^{(e)}]\{X^{(e)}\} + \bar{U}(X^{(e)})\right) \tag{C13}$$

where $\{\sigma\}_i$ is the vector of stress resultants at node i; $[C]$ is the material-stiffness matrix; $\{X^{(e)}\}$ is the vector of nodal displacements for the element; $[B]_i$ is the linear-strain-displacement matrix evaluated at node i; $\{\bar{B}(X^{(e)})\}$ is the nonlinear-strain-displacement vector evaluated at node i; U is the total strain energy, and $\bar{U}(X^{(e)})$ represents the contribution of the cubic and quartic terms in the nodal displacements to the strain energy of an individual element.

The vectorization of the operations described in equations (C12) and (C13) was accomplished by using the math-library routines listed in table CII. To increase the gain in speed obtained by vectorization, the subvectors of nodal displacements and stress resultants were strung up into long vectors. The same was done for the matrix $[B]$ and the vector $\{\bar{B}(X^{(e)})\}$. The larger arrays could still fit into one segment of memory.

12.7 Comments on Vectorization

The following comments regarding vectorization of the numerical algorithms on the array processor seem to be in order:

(1) User-developed microcoded routines are more efficient to use than the array-processor math-library routines when (a) specialized algorithms are used, such as the Cholesky hypermatrix algorithm discussed in the preceding sections, and/or (b) vector operations involve sparse or symmetric arrays. On the other hand, the AP math-library routines are generally more efficient for vector operations involving full nonsymmetric matrices.

(2) Data reorganization (required prior to performing vector operations) was found to be slow on the array processor. An example of this is provided by the matrix transpose operation. (See table CIII.)

TABLE CI.- TYPES OF DATA STRUCTURES USED IN VECTORIZATION OF NONLINEAR FINITE-ELEMENT COMPUTATIONS

Type	Array	Description	Data structure used in vectorization
A	$[K^{(e)}]$	Full, symmetric, 2-D array (80 × 80)	Expand to full square matrix
B	$F^{(e)}$	Large, completely symmetric, 3-D array (80 × 80 × 80)	Array treated as full non-zero blocks of size 16 × 16 × 16
C	$G^{(e)}$	Large, completely symmetric 4-D array (16 × 16 × 16 × 16)	Expand array as if it were symmetric with reference to first three indices only
D	Global [K]	Large, sparse, symmetric, square matrix	Matrix partitioned into square submatrices (Cholesky hypermatrix blocks)
E	Partitions of $[K^{(e)}]$, stress resultants, strain, or displacement vectors	Small submatrix or vector	Combine a number of submatrices (or subvectors) into a larger matrix (or vector)

TABLE CII.- ARRAY-PROCESSOR MATH-LIBRARY ROUTINES USED IN PRESENT STUDY[a]

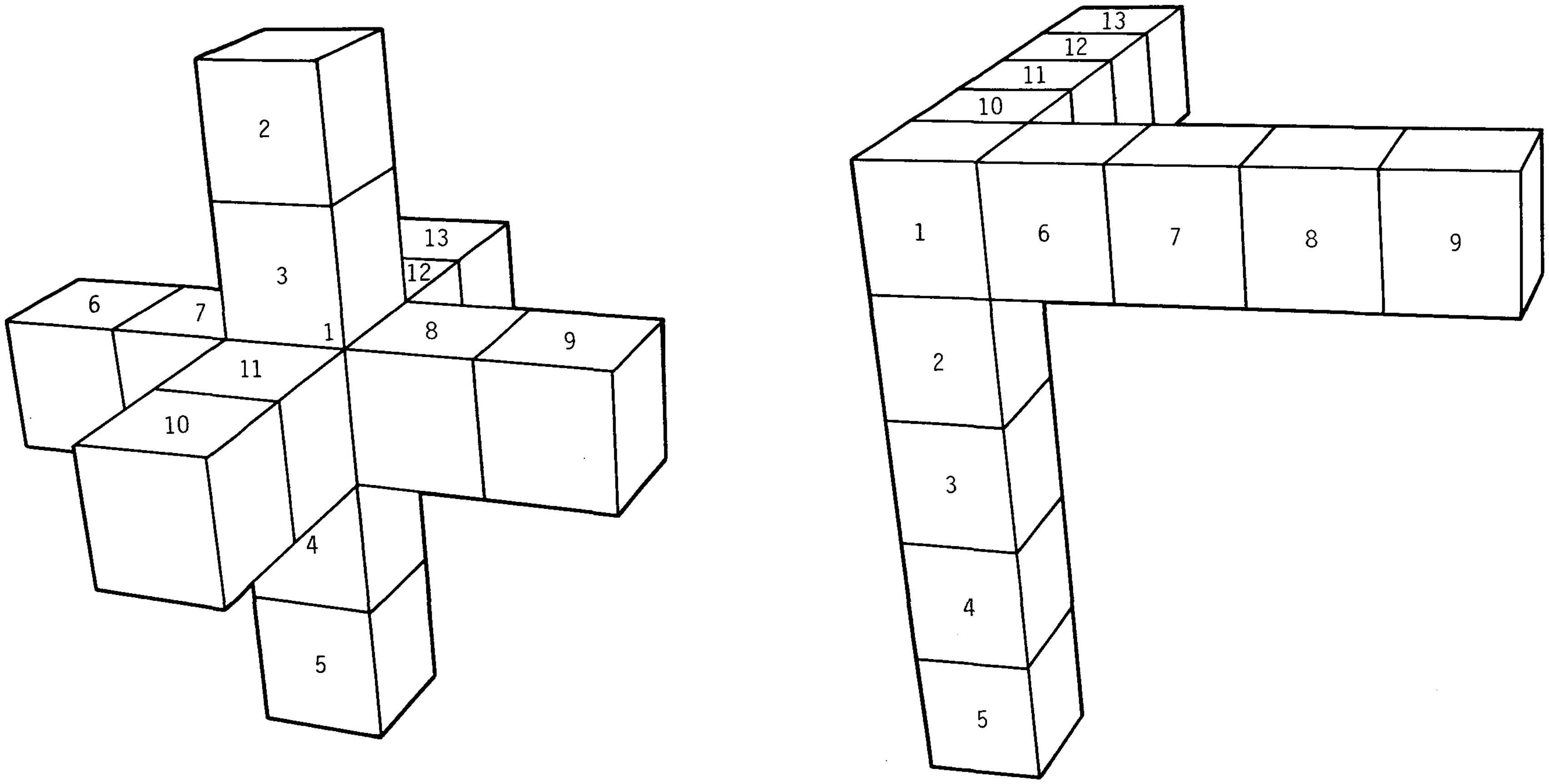
FPS library subroutine function	AP Assembler library code	FPS library subroutine name
Matrix multiply	1	MMUL
Matrix transpose	2	MTRANS
Vector addition	3	VADD
Vector subtraction	4	VSUB
Vector scalar multiply	5	VSMUL
Sum of vector elements squared	6	SVESQ

[a]Routines described in reference 8.

TABLE CIII.- SPEEDUP RATIOS OF PRIME 750/AP-120B FOR VECTOR OPERATIONS

Routine	Block no. in figure B1	Computation[a]	Assembler coded or AP math library	Estimated ratio of Prime 750/AP-120B
Linear-element-stiffness matrix	1	$\left[K^{(e)}\right] = \sum_{\ell=1}^{n} w_\ell [B]_\ell^T [C][B]_\ell$	Library	19.0
Generate $F_{IJK}^{(e)}$	1	See reference 14	Library	4.3
Generate $G_{IJKL}^{(e)}$	1	See reference 14	Assembler	14.4
Cholesky's method with hypermatrices	3,4	Macrooperation (1)	Assembler	22.9
		Macrooperation (2)	Assembler	22.3
		Macrooperation (3)	Library	29.6
		Macrooperation (4)	Library	72.2
		Macrooperation (5)	Assembler	22.4
		Macrooperation (6(a))	Library	84.5
		Macrooperation (6(b))	Library	33.8
Solution update and convergence check	14	$\{X\} * q$	Library	22.5
		$\sum_{i=1}^{N} X_i^2$	Library	50.6
		$\{X\} + \{\Delta X\}$	Library	23.3
Strain energy and stress resultant computation	18	Linear strain energy	Library	44.9
		Stress resultant	Library	72.3

[a]An asterisk * denotes multiplication.



(a) Original structure of blocks.

(b) Modified structure of blocks.

Figure C1.- Organization of nonzero blocks of three-dimensional array $F^{(e)}$.

13 APPENDIX D

BUFFERING TECHNIQUES USED FOR MINIMIZING I/O OPERATIONS ON HOST AND ARRAY PROCESSOR

The buffering techniques used for minimizing the I/O operations on the Prime 750 are described in this appendix. Extension of these techniques to the DEC VAX-11/780 are also given. Then, the minimization of I/O operations on the AP-120B, as well as between the host minicomputer and the array processor, is presented.

13.1 Buffering Techniques for Host Computers

13.1.1 Description of buffering techniques on Prime 750.-

13.1.1.1 Standard FORTRAN binary I/O: When standard FORTRAN binary I/O is used, the portion of the code employed to measure the output of 10 000 displacement vectors has the following form:

```
      REAL*4 DISP
      DIMENSION DISP(5)
      CALL SNAP
      DO 10 I=1,10 000
        WRITE(6) DISP
10    CONTINUE
      CALL SHOT ('FORTRAN BINARY,' 14)
```

Standard FORTRAN binary I/O uses a system I/O buffer of size-30 words plus a 2-byte pointer. The entire buffer is transmitted to disk at the end of each write statement. This produces a large output file of 1387 records. (See fig. D1.) Each Prime disk record equals 220 words (880 bytes). The total word output equals 305 140, compared with a user output of 50 000 words. Standard FORTRAN binary I/O proves inefficient because of the constant size of the system I/O buffer.

13.1.1.2 Set FORTRAN system buffer size: Prime system routine "ATTDEV" ("attach device") allows the user to set the size of the FORTRAN system I/O buffer. This routine is called once outside the "do loop" in the portion of code listed above, which sets the system I/O buffer to five words. The use of this technique considerably reduces the number of records in the output file compared with those used by the first technique (250 records as compared with 1387 records). (See fig. D1.)

13.1.1.3 System I/O routine: Prime system I/O routine PRWF$$ (position, read, and write file) permits direct transfer of data from memory to disk, thus completely bypassing the FORTRAN I/O buffer. Only user data are written to disk. No pointers or zero filling are used to enlarge the output file as with FORTRAN binary I/O. The write statement in the aforementioned FORTRAN is replaced with a call to PRWF$$. The use of this buffering technique reduces the CPU time by a factor of 4.5 and increases the data rate by a factor of 3.8 from the corresponding values when the second technique is used. (See fig. D1.)

Although PRWF$$ is a powerful I/O routine, it has the following two limitations:

(1) It can transmit data only to contiguous memory locations.

(2) It cannot transmit data across segment boundaries.

The first limitation is not a serious one since buffers are located in contiguous memory locations.

13.1.1.4 User buffering (32-bit words): This technique involves copying 32-bit data items into a user I/O buffer the size of one Prime segment (32 000 words) and outputting the buffer to disk when either it is full or the I/O operation is completed. Three routines are used for this method: INITBF (Initialize Buffer) initializes the output file and sets a pointer to zero; WRITBF (Write to Buffer) copies 32-bit data items into the buffer and outputs them to the specified logical unit when the buffer is full; and DONEBF (Done with Buffer) flushes the buffer and closes the output file. The use of this technique results in significant decreases in CPU and I/O times, as well as a dramatic increase in data rate. (See fig. D1.) The improvement in performance is due to the minimal number of disk accesses required by this method. The favorable CPU-I/O ratio of 2.27 enhances the prospects of CPU-I/O overlap within the Prime 750. PRIMOS (Prime Operating System) is capable of scheduling parallel processing between the CPU and disk I/O to allow higher system throughput.

13.1.1.5 User buffering (64-bit words): This technique is identical to the preceding one except that the data are copied in 64-bit words into the segment-size buffer. The data rate improves by a factor of 2 over the preceding technique, and the CPU-I/O ratio becomes most favorable for parallel CPU-I/O operation on the Prime 750.

13.1.1.6 User buffering all data - system pages buffer: This technique involves no user output to disk. The data are copied to a very large user buffer. If the buffer cannot fit in central memory, the system pages the data to and from disk as they are needed. This technique is used for large arrays produced as intermediate results.

Of the six techniques described, the last one produced the best results for total time and data rate. (See fig. D1.) This can be attributed to the absence of user-initiated I/O. The technique is restricted, however, for use with data that are not needed for permanent disk storage, and it should be used with care because of its large central-memory requirements.

13.1.2 Performance of I/O buffering techniques on the Prime 750.- To evaluate the effectiveness of the buffering techniques described in the preceding section in reducing the I/O operations, the techniques were applied to the cylinder problem with rectangular cutout (3230 degrees of freedom). Most of the I/O time in the program was expended in four processors, namely, generation of basis vectors, decomposition of the left-hand side, forward reduction/back substitution, and assembly of the elemental stiffness matrices and load vectors. The dramatic reductions in the I/O times obtained by using these techniques for the four processors are shown in figure D2. The use of these techniques transformed the I/O-bound program to one which is CP bound.

In figure D2, T_1 refers to the I/O times obtained by using the third buffering technique. The times T_2 and T_3 are associated with a buffering technique which is a combination of techniques 3, 5, and 6 outlined in appendix D. The number of nodes per hypermatrix block are 12 and 5, corresponding to array sizes of 60 × 60 and 25 × 25, respectively. Note that the improvement obtained by using the smaller-size arrays (5 nodes per hypermatrix block - 25 × 25 arrays) is due to the smaller number of zeros stored in the blocks.

The total (CP + I/O) times expended in 10 different processors for the 2 cylindrical-shell problems are shown in figures D3 and D4. The 10 processors are as follows: (1) evaluation of linear-stiffness arrays $[K^{(e)}]$; (2) evaluation of nonlinear-stiffness arrays $G^{(e)}$ and $F^{(e)}$; (3) assembly of elemental matrices; (4) incorporation of boundary conditions; (5) decomposition of the global-linear-stiffness matrix [K]; (6) forward reduction/back substitution; (7) solution update; (8) postprocessing (stress resultant and strain-energy computation); (9) generation of basis vectors; and (10) solution of reduced equations. Note that three of these processors were not vectorized; namely, the assembly, boundary conditions, and solution of reduced equations. Also, since the basis vectors were generated at zero loading, no convergence check was needed. The percentage of time expended in each of the processors and the estimated gain in speed obtained by using the array processor are summarized in table IV.

As can be seen from table IV and figures D3 and D4, most of the solution time is spent in the two processors used for the decomposition of the left-hand side and in the evaluation of basis vectors and the generation of reduced equations. The use of the AP (with minimal host I/O) reduces the total times (CP + I/O) expended in these two processors by factors of 46.7 and 17.1 for the pear-shaped cylinder and by factors of 59.8 and 38.1 for the cylinder with a cutout. The effect of I/O on total gains in speed for the two problems is a reduction in CP gains of 9.2 and 18.1, respectively, to total (CP + I/O) gains of 5.2 and 9.9. This difference of a factor of two between the CP gain and the I/O gain can be attributed to high I/O in the assembly of elemental matrices and generation of basis vectors.

13.1.3 Application of buffering techniques to DEC VAX-11/780.- Five of the six buffering techniques outlined above were implemented for the DEC VAX-11/780 minicomputer. (See fig. D5.) Although results are not as dramatic as on the Prime 750, steady increases in data rate result from use of these techniques. The authors believe that these techniques are not machine dependent and would be useful on most virtual-memory minicomputer systems.

Buffering techniques 3 to 6 of figure D5 used the VAX macro routine "#WRITE" to perform output to the disk. Subroutines INITBF, WRITBF, and DONEBF had to be written in VAX-11 macro language because a FORTRAN callable-system I/O routine was not available.

13.2 Buffering Technique for the AP-120B

The buffering technique presented herein involves table and main-data memory of the AP-120B. Data are buffered from main-data memory into table memory to avoid interleave problems with main-data memory. The following example was taken from the Cholesky decomposition of the global-stiffness matrix $[\overset{*}{K}]$. (See block 3 in fig. B1.) The FORTRAN code used in the decomposition of the diagonal blocks and the execution trace of this code are

FORTRAN Code:

```
      DO 40 I=2,N
      IM1 = I-1
      DO 20 L=1, IM1
        SUM = SUM + A(L,I) * A(L,I) * A(L,L)
   20 CONTINUE
   40 CONTINUE
```

As an example, consider the 5 × 5 symmetric matrix [A] whose elements occupy the following locations in memory:

$$[A] = \begin{bmatrix} 1 & 6 & 11 & 16 & 21 \\ 2 & 7 & 12 & 17 & 22 \\ 3 & 8 & 13 & 18 & 23 \\ 4 & 9 & 14 & 19 & 24 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix}$$

Execution trace =

I	IM1	L	A(L,I)	A(L,L)
2	1	1	6	1
3	2	1, 2	11, 12	1, 7
4	3	1, 2, 3	16, 17, 18	1, 7, 13
5	4	1, 2, 3, 4	21, 22, 23, 24	1, 7, 13, 19

To perform the multiplications indicated in the FORTRAN program as vector operations, the pairs of elements A(L,I) and A(L,L) need to be fetched from main-data memory. Consider, as an example, the case where I = 3 and L = 1. The element A(L,I) in location 11 is fetched from memory, and then the element A(L,L) in location 1 is fetched from memory. Since both elements are in odd memory locations, they reside in the same memory bank. Therefore, the access of the second element results in a spin operation. The execution trace shown above indicates that spin operations are certain to occur frequently. Each spin operation adds 167 ns to the loop times required to produce one result in a vector operation.

The spin operations are a direct consequence of interleaving in main-data memory. Since no interleaving is involved in accessing the table memory, the aforementioned problem can be overcome by using the table memory as a buffer for the main-data memory, that is, treating the two memories as a two-level virtual-memory system on the array processor. Only the three hypermatrix blocks needed by the Cholesky algorithm at any one time are kept in table memory. The main-data memory is then used as a mass-storage area wherein a large number of hypermatrix blocks reside. All computational modules are made to reference table memory only, thereby taking advantage of its faster access speed. The transfer of blocks between main-data memory and table memory is accomplished by using the math-library routines of the AP.

13.3 Minimizing I/O Between Host Computer and Array Processor

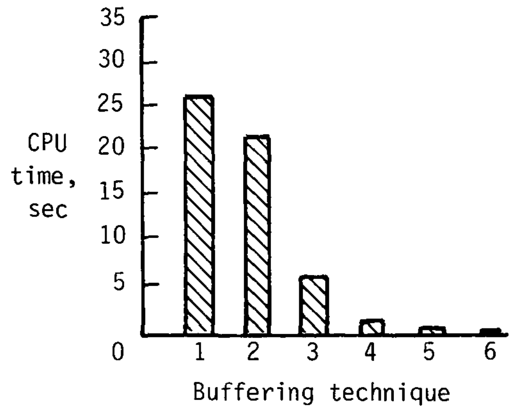
On the user-software level, the following two techniques can be used to reduce host-system overhead:

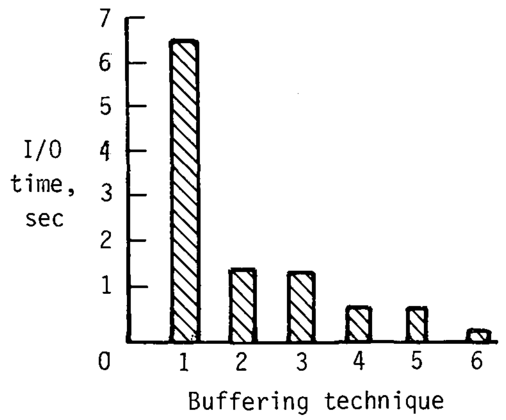
(1) Minimizing the number of calls to the array processor: This can be achieved by the following:

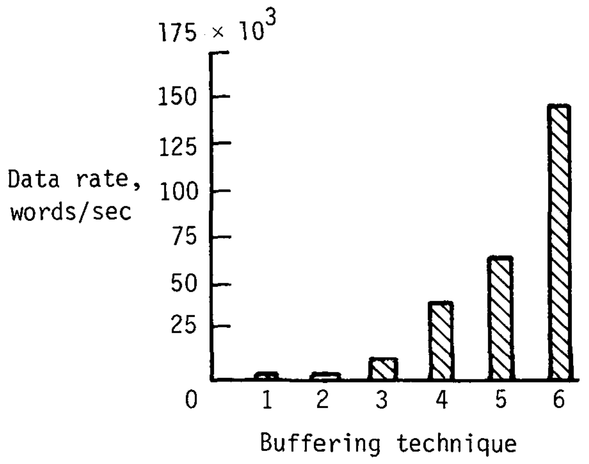
(a) Merging smaller arrays (e.g., vectors and matrices) into larger ones: If a number of short vectors are combined into a single long vector, only one data-transmission call to the AP will be needed. This may require a change in I/O block sizes and/or a change in the numerical-solution algorithm being used.

(b) Chaining several array-processor routines: When a number of array-processor subroutines are needed, it is more efficient to call these subroutines from a single main array-processor routine instead of calling them separately from the host FORTRAN program. Several routines can now be sent to the array processor in one call if the program-source memory is large enough. Program-source memory ranges in size from 512 to 3096 words with 256-word increments.

(2) Loading the most frequently used routines first: This is because the array-processor executive (APEX) uses a last-in, first-out technique in allocating memory space for the source program.



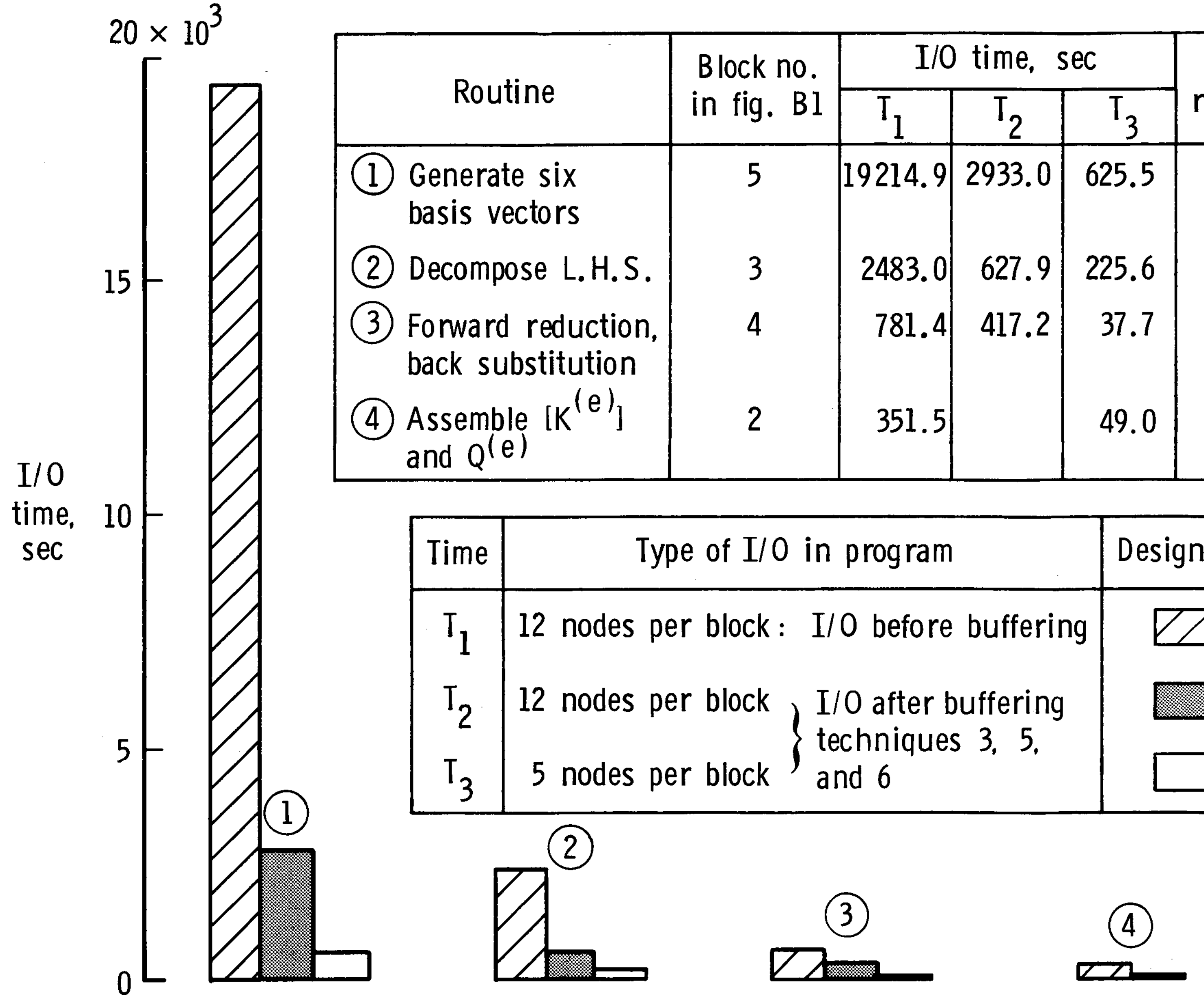




Buffering technique		CPU, sec	I/O, sec	Total, sec	Number of records	Data rate[a]	$\frac{\text{CPU}}{\text{I/O}}$
No.	Name						
1	FORTRAN binary	26.01	6.54	32.55	1 387	1 922	3.98
2	Set FORTRAN buffer size	21.86	1.34	23.20	250	2 287	16.31
3	SYSTEM I/O routine	5.78	1.26	7.04	228	8 651	4.59
4	Buffered (32 bit)	1.27	.56	1.83	228	39 370	2.27
5	Buffered (64 bit)	.78	.56	1.34	228	64 103	1.39
6	Buffered (paging)	.34	.16	.50	0	147 059	2.13

[a]Words per CPU second.

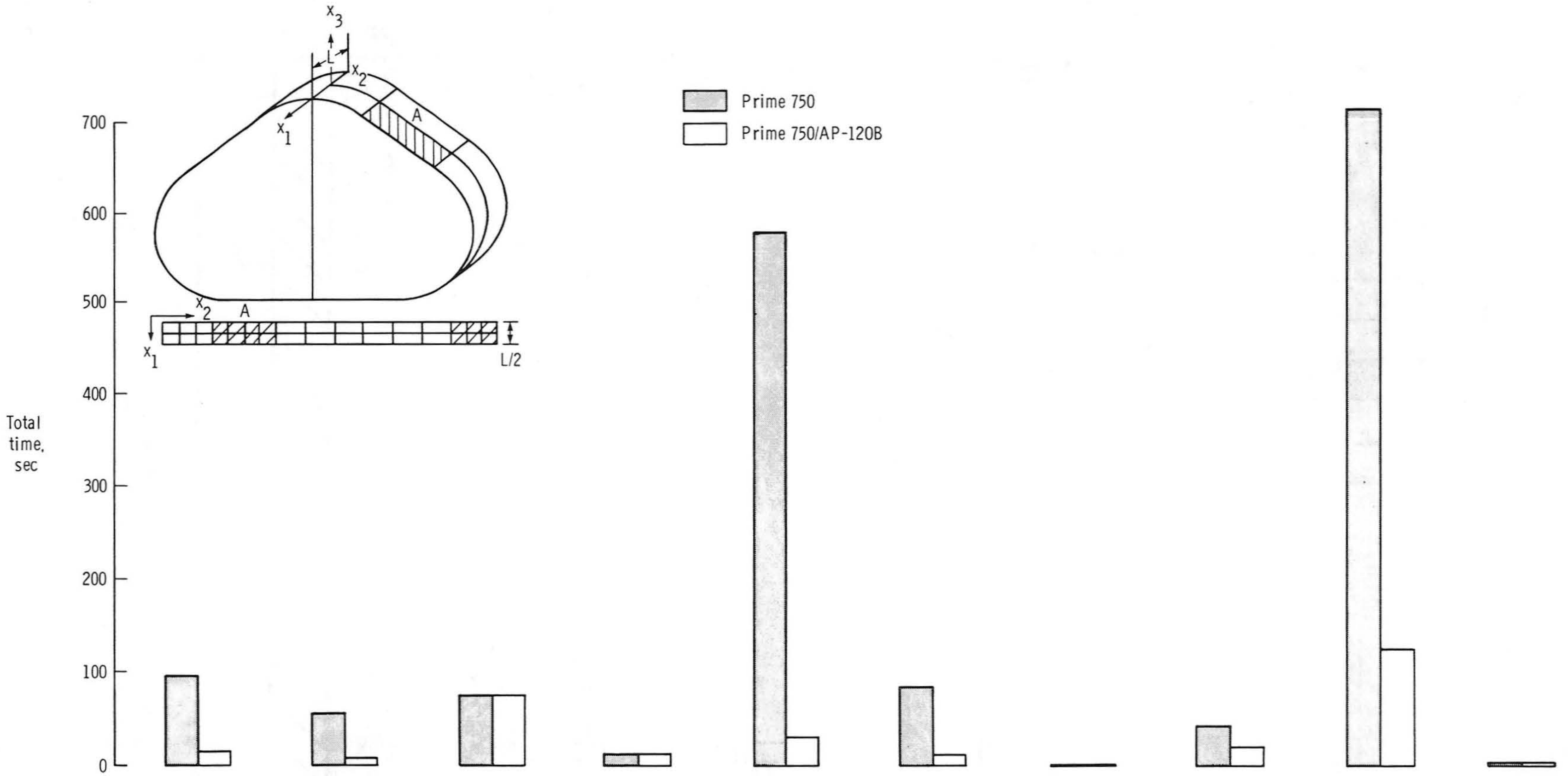
Figure D1.- Performance of buffering techniques on the Prime 750 to output 10 000 vectors to disk.



Routine	Block no. in fig. B1	I/O time, sec			Speedup ratio, T_1/T_3
		T_1	T_2	T_3	
(1) Generate six basis vectors	5	19214.9	2933.0	625.5	30.7
(2) Decompose L.H.S.	3	2483.0	627.9	225.6	11.0
(3) Forward reduction, back substitution	4	781.4	417.2	37.7	20.7
(4) Assemble $[K^{(e)}]$ and $Q^{(e)}$	2	351.5		49.0	7.2

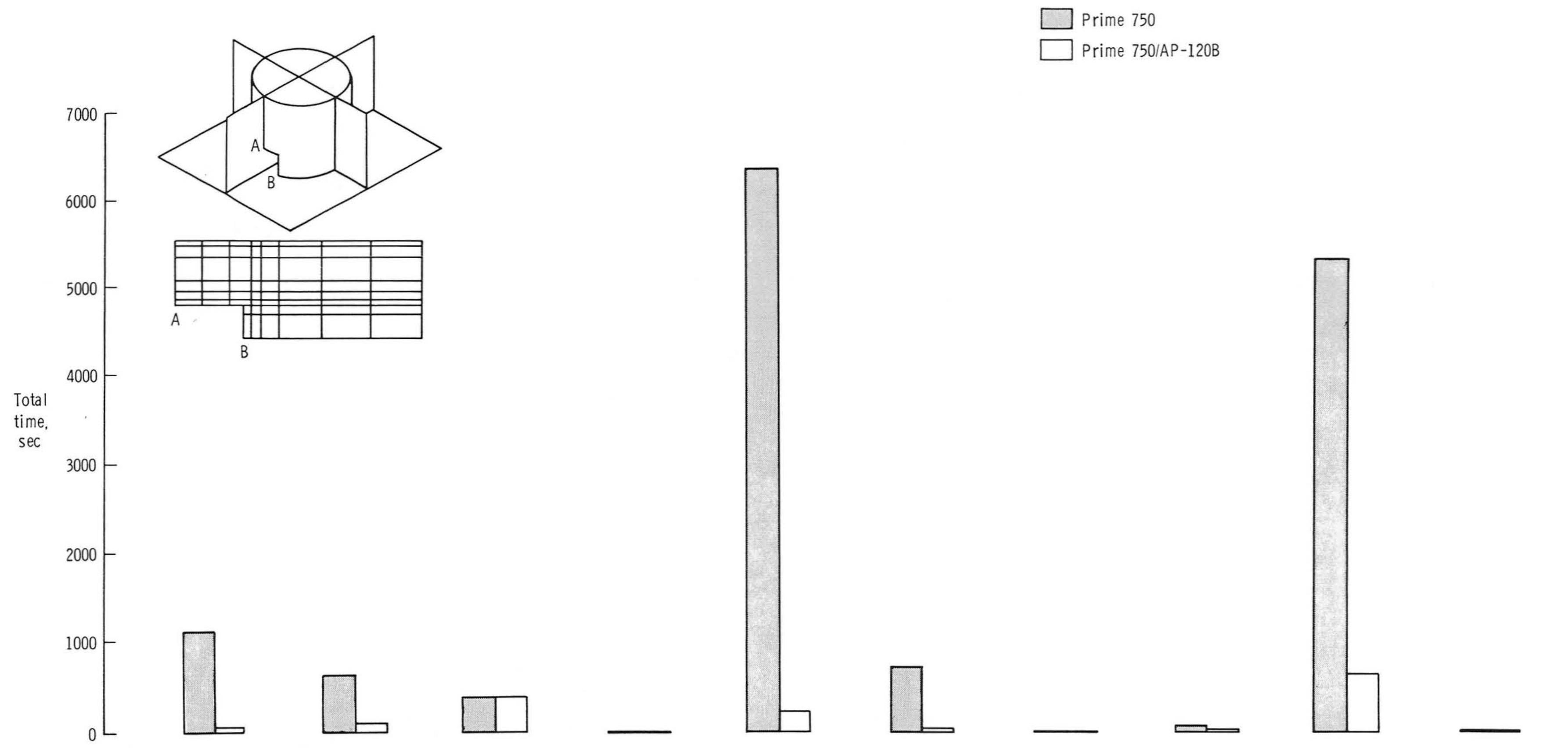
Time	Type of I/O in program		Designation
T_1	12 nodes per block:	I/O before buffering	▨
T_2	12 nodes per block	I/O after buffering techniques 3, 5, and 6	▩
T_3	5 nodes per block		□

Figure D2.- I/O reductions obtained by using buffering techniques on Prime 750. Cylindrical shell with rectangular cutout shown in figure 10.



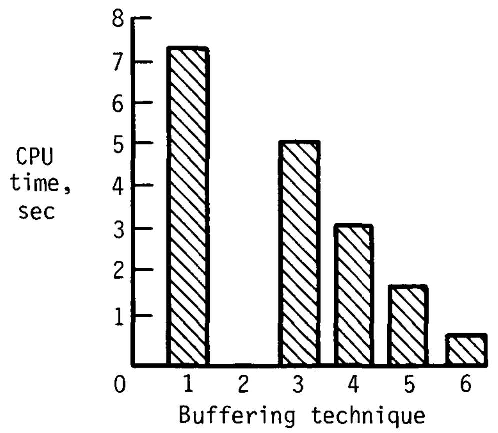
Routine	Generate $[K^{(e)}]$	Generate $F^{(e)}$ and $G^{(e)}$	Assembly	Boundary conditions	Decompose $[K]$	Forward reduction, back substitution	Solution update	Stress resultants and energy computation	Generate six basis vectors	Solution of reduced equations
Block no. in fig. B1:	1	1	2	2	3	4	14	18	15	8

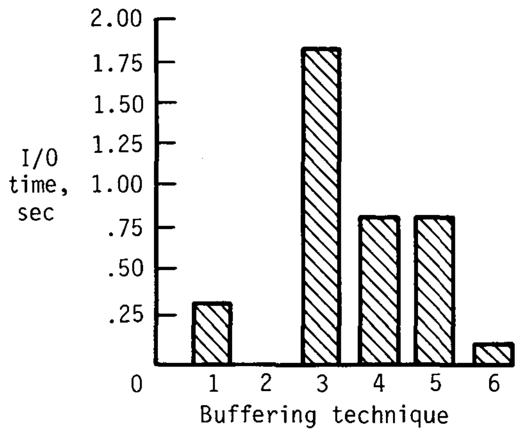
Figure D3.- Estimated total speedup ratios (CPU + I/O) obtained by using minicomputer/array-processor system. Pear-shaped cylindrical shell is shown in figure 6.

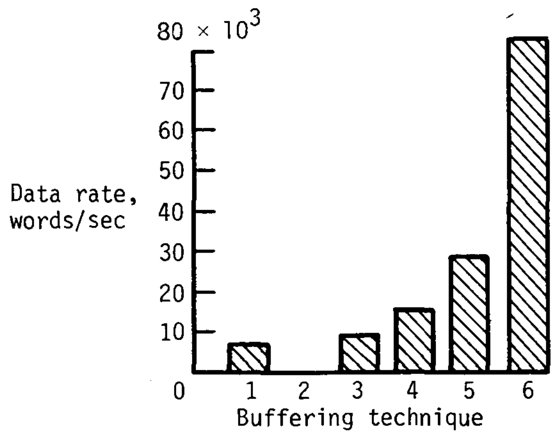


Routine	Generate $[K^{(e)}]$	Generate $F^{(e)}$ and $G^{(e)}$	Assembly	Boundary conditions	Decompose [K]	Forward reduction, back substitution	Solution update	Stress resultants and energy computation	Generate five basis vectors	Solution of reduced equations
Block no. in fig. B1:	1	1	2	2	3	4	14	18	15	8

Figure D4.- Estimated total speedup ratios (CPU + I/O) obtained by using minicomputer/array-processor system. Cylindrical shell with rectangular cutout is shown in figure 7.







Buffering technique		CPU, sec	I/O, sec	Total, sec	Number of records	Data rate[a]	CPU/I/O
No.	Name						
1	FORTRAN binary	7.3	.3	7.6	469	8 224	24.3
2	Set FORTRAN buffer size						
3	SYSTEM I/O routine	5.1	1.8	6.9	391	9 813	2.8
4	Buffered (32 bit)	3.1	.8	3.9	391	16 145	3.9
5	Buffered (64 bit)	1.7	.8	2.5	391	29 440	2.1
6	Buffered (paging)	.6	.1	.7	0	83 333	6.0

[a]Words per CPU second.

Figure D5.- Performance of buffering techniques on the DEC VAX-11/780 to output 10 000 vectors to disk.

14 REFERENCES

1. Ferson, Kent E.: Performance Evaluation of a Minicomputer/Array Processor System for Finite Element Applications. M.S. Thesis, Rensselaer Polytechnic Inst., Aug. 1979.

2. Sarigul, Nesrin; Maitan, Jacek; and Kamel, Hussein A.: Implementation of Some Finite Element Algorithms on a Minicomputer With an Attached Array Processor. University of Arizona paper presented at the CAFEM-6 (Paris, France), Aug. 24-25, 1981.

3. Sarigul, Nesrin; Maitan, Jacek; and Kamel, Hussein A.: Solution of Nonlinear Structural Problems Using Array Processors. University of Arizona paper presented at the FENOMECH 81 (ISD, Struttgart), Aug. 25-28, 1981.

4. System Architecture. Reference Guide PDR3060, Prime Computer, Inc., c.1981.

5. PRIMOS Subroutines PDR3621, Revision A. P/N MAN3251-001, Prime Computer, Inc., c.1980.

6. FPS Technical Publications Staff: Programmers Reference Manual - Part One, 1st ed. Publ. No. FPS-7319, Floating Point Systems, Inc., c.1978.

7. FPS Technical Publications Staff. Floating Point Systems, Inc.

   Vector Function Chainer Manual, First ed. Publ. No. FPS 7351, c.1977.

   Program Development Software Manual. Publ. No. 860-7292-002, c.1978.

8. FPS Technical Publications Staff: AP-120B Math Library - Parts One and Two, Third ed. Publ. No. FPS 7288-03, Floating Point Systems, Inc., c.1977.

9. Conaway, J. H.: Structural Engineering Software on Small Computers. Paper 80-C2/Aero-7, American Soc. Mech. Eng., Aug. 1980.

10. Karplus, Walter J.; and Cohen, Danny: Architectural and Software Issues in the Design and Application of Peripheral Array Processors. Computer, vol. 14, no. 9, Sept. 1981, pp. 11-17.

11. Charlesworth, Alan E.: An Approach to Scientific Array Processing: The Architectural Design of the AP-120B/FPS-164 Family. Computer, vol. 14, no. 9, Sept. 1981, pp. 18-27.

12. Cohen, Danny: A Methodology for Programming a Pipeline Array Processor. Proceedings - The 11th Annual Microprogramming Workshop, Inst. Electr. & Electron. Eng., Inc., c.1978, pp. 82-89.

13. Noor, A. K.; and Peters, J. M.: Reduced Basis Technique for Nonlinear Analysis of Structures. AIAA J., vol. 18, no. 4, Apr. 1980, pp. 455-462.

14. Noor, Ahmed K.: Recent Advances in Reduction Methods for Nonlinear Problems. Comput. & Struct., vol. 13, no. 1-3, June 1981, pp. 31-44.

15. Noor, Ahmed K.; Andersen, C. M.; and Peters, Jeanne M.: Global-Local Approach for Nonlinear Shell Analysis. Seventh Conference on Electronic Computers, American Soc. Civil Eng., 1979, pp. 634-657.

16. Whetstone, W. D.: SPAR Structural Analysis System Reference Manual - System Level 13A. Volume I: Program Execution. NASA CR-1589701, 1978.

17. Hartung, Richard F.; and Ball, Robert E.: A Comparison of Several Computer Solutions to Three Structural Shell Analysis Problems. AFFDL-TR-73-15, U.S. Air Force, Apr. 1973. (Available from DTIC as AD 762 946.)

18. Technology Review - Attached Processor Handles Large Scale Scientific Computing Applications. Comput. Des., vol. 19, no. 8, Aug. 1980, p. 44.

19. Cohler, Jonathan: Minicomputers - Array Processors Enhance Minicomputer Performance. Mini-MicroSystems, Apr. 1982, pp. 179-188.

20. DeSalvo, Gabriel J.; and Swanson, John A.: ANSYS® Engineeering Analysis System Examples Manual (for ANSYS Revision 4.0). Swanson Analysis Sys., Inc., May 1, 1982.

21. Webster, Robin; and Miner, Leslie: Expert Systems - Programming Problem-Solving. Technology, vol. 2, no. 1, Jan./Feb. 1982, pp. 62-73.

22. Brender, Ronald F.; and Massi, Isaac R.: What is Ada? Computer, vol. 14, no. 6, June 1981, pp. 17-22, 24.

23. Coleman, D.; Gallimore, R. M.; Hughes, J. W.; and Powell, M. S.: An Assessment of Concurrent Pascal. Software - Pract. & Exper., vol. 9, no. 10, Oct. 1979, pp. 827-837.

24. Comfort, Dennis L.; and Erickson, Wayne J.: RIM - A Prototype for a Relational Information Management System. Engineering and Scientific Data Management, NASA CP-2055, 1978, pp. 183-196.

25. Tanenbaum, Andrew S.: Structured Computer Organization. Prentice-Hall, Inc., c.1976.

26. Storaasli, Olaf O.; and Murphy, Ronald C.: Finite Element Analysis in a Minicomputer/Mainframe Environment. Research in Computerized Structural Analysis and Synthesis, NASA CP-2059, 1978, pp. 77-88.

27. Giles, Gary L.; and Haftka, Raphael T.: SPAR Data Handling Utilities. NASA TM-78701, 1978.

28. Cunningham, Sally W.: SPAR Data Set Contents. NASA TM-83181, 1981.

29. Zienkiewicz, O. C.: The Finite Element Method. McGraw-Hill Book Co., Ltd., c.1977.

TABLE I.- SAMPLE TIMINGS OF VECTOR-SOFTWARE OPERATIONS FOR LIBRARY ROUTINES DEVELOPED FOR THE AP-120B

[Library routines developed by Floating Point Systems, Inc.; data taken from ref. 8.]

Operation[a]		Time, µs (b)
		Startup + Loop time
Sum of vector elements	$\sum_{i=1}^{\bar{n}} A_i$	$0.833 + 0.333\bar{n}$
Sum of squares of vector elements	$\sum_{i=1}^{\bar{n}} A_i^2$	$1.333 + 0.333\bar{n}$
Vector add	$\{A\} + \{B\}$	$2.667 + 0.500\bar{n}$
Vector multiply (term by term)	$\{A\} * \{B\}$	$2.667 + 0.500\bar{n}$
Vector scalar multiply	$A * \{B\}$	$2.667 + 0.500\bar{n}$
Vector scalar multiply and constant vector add ...	$A * \{B\} + \{C\}$	$3.333 + 0.500\bar{n}$
Dot product	$\{A\}^T * \{B\}$	$3.333 + 0.500\bar{n}$
Vector multiply and add (term by term)	$\{A\} * \{B\} + \{C\}$	$3.333 + 0.833\bar{n}$
Two vector multiplies and one vector add	$(\{A\} * \{B\}) + (\{C\} * \{D\})$	$4.167 + 0.833\bar{n}$
Vector divide (term by term)	$\{A\}/\{B\}$	$4.167 + 1.667\bar{n}$

[a]An asterisk * denotes multiplication.
[b]The symbol $\bar{n}$ denotes the length of vector.

TABLE II.- VECTORIZATION OF ARITHMETIC OPERATIONS IN NONLINEAR-FINITE-ELEMENT PROGRAM

Function	Block no. in fig. B1	Data-structure type[a]					AP math library[b]	Assembler code developed	Overlap host I/O with AP
		A	B	C	D	E			
Generate $[K^{(e)}]$	1	√					1,2,3,5	√	√
Generate $F^{(e)}$	1		√				1,3,5		√
Generate $G^{(e)}$	1			√			1,3,5	√	√
Decompose L.H.S.	3				√		1,2,4	√	√
Forward reduction, back substitution	4				√		1,2,4	√	√
Convergence check	14					√	3,5,6		
Generate basis vectors	5	√	√	√	√		1,2,3,4	√	√
Compute energy and stress	18					√	1,2,3,5		

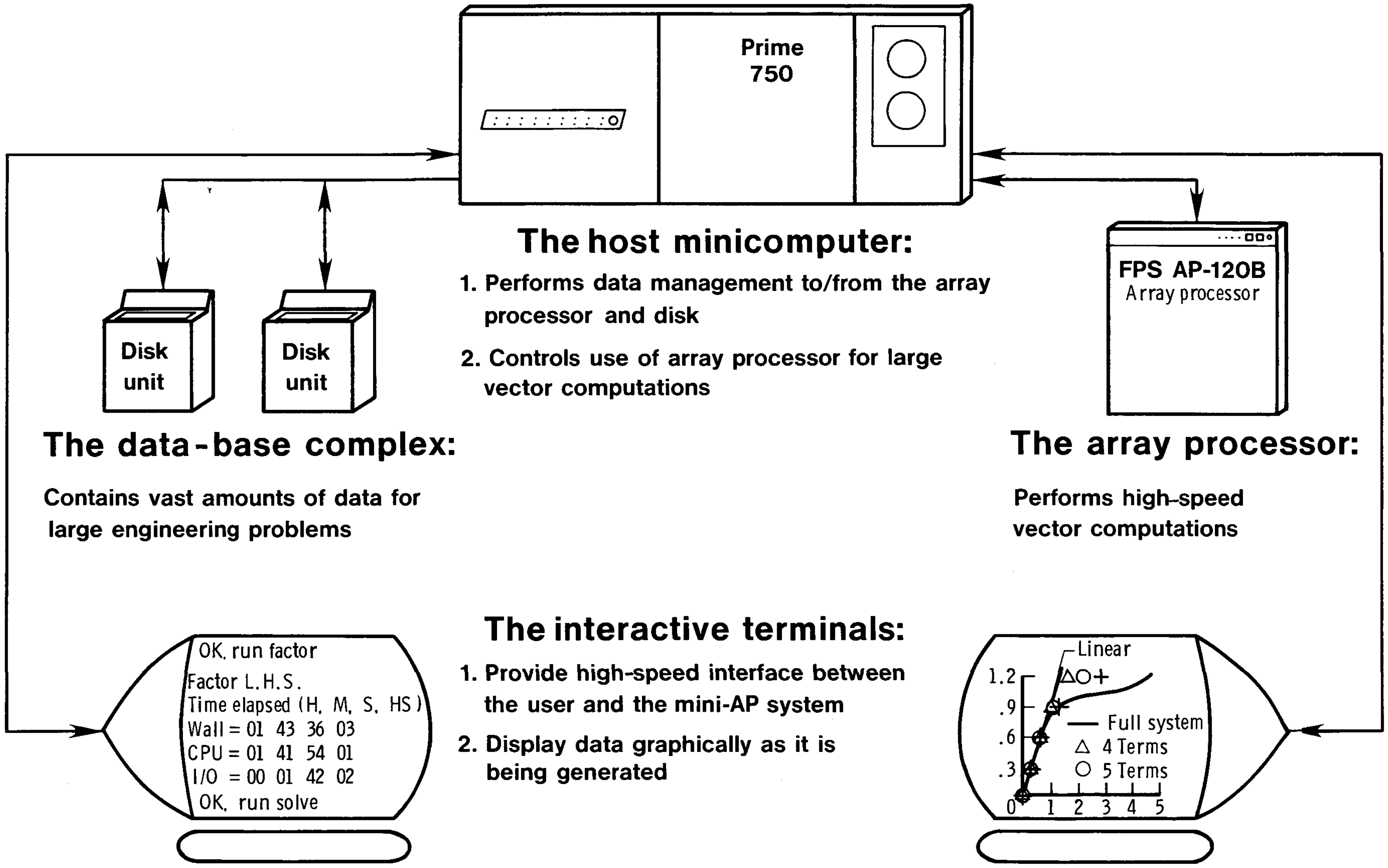
[a]See table CI.
[b]See table CII.

TABLE III.- HARDWARE AND SOFTWARE FACTORS AFFECTING I/O OPERATIONS IN MINICOMPUTER/ARRAY-PROCESSOR SYSTEM

Computer I/O level	Host computer (cache memory, central memory, and disk)	Array processor (main-data and table memories)	Host/AP interface
Hardware	● Cache-memory size ● Central-memory size ● Memory's access speed	● Main-data-memory size and speed ● Table-memory size ● Main-data-memory interleave ● Single global data base	● Use of peripheral storage device with AP
System software	● Resource management and allocation ● Paging algorithm between . Central memory and disk . Cache and central memories	● AP system I/O routines used in transferring data between table and main-data memories	● Host-operating-system response time to setup transfer from host to AP by APEX routines
User software	● Buffering techniques between central memory and disk	● Buffering techniques between table and main-data memories	● Minimize number of calls to AP . Merging small data sets . Chaining AP routines ● Load most frequently used routines first ● Overlap data transfer and AP execution

TABLE IV.- TIMING DISTRIBUTION FOR TWO BENCHMARK PROBLEMS USED IN PRESENT STUDY

Routine	Block no. in fig. B1	Time, percent				Speedup ratio			
		CPU		CPU + I/O		CPU		CPU + I/O	
		Problem		Problem		Problem		Problem	
		1	2	1	2	1	2	1	2
Generate form independent $[K^{(e)}]$	1	6.2	8.2	5.7	7.7	6.4	20.6	6.9	20.5
Generate form independent $F^{(e)}$ and $G^{(e)}$ arrays	1	3.6	4.5	3.4	4.4	7.5	6.5	7.3	6.3
Assembly	2	4.5	2.4	4.5	2.6	1.0	1.0	1.0	1.0
Incorporation of boundary conditions	2	<0.01	<0.001	0.7	<0.001	1.0	1.0	1.0	1.0
Decompose [K]	3	36.1	44.8	34.6	43.4	18.7	28.0	46.7	59.8
Forward reduction, back substitution	4	4.9	4.9	5.1	4.9	7.0	18.7	45.9	51.1
Convergence check and solution update ...	14	<0.001	<0.001	<0.001	<0.001	4.0	1.7	17.0	19.4
Post processor	18	2.9	0.5	2.6	0.5	2.2	1.9	2.2	2.1
Generate basis vectors and reduced equations	15	41.6	34.6	43.1	36.5	5.8	8.5	17.1	38.1
Solution of reduced equations	8	<0.01	<0.001	<0.01	<0.001	1.0	1.0	1.0	1.0
Total						9.2	18.1	5.2	9.9



The host minicomputer:

1. Performs data management to/from the array processor and disk
2. Controls use of array processor for large vector computations

The data-base complex:

Contains vast amounts of data for large engineering problems

The array processor:

Performs high-speed vector computations

The interactive terminals:

1. Provide high-speed interface between the user and the mini-AP system
2. Display data graphically as it is being generated

```
OK, run factor
Factor L.H.S.
Time elapsed (H, M, S, HS)
Wall = 01 43 36 03
CPU = 01 41 54 01
I/O  = 00 01 42 02
OK, run solve
```

Figure 1.- Schematic drawing of organization of minicomputer/array-processor system.

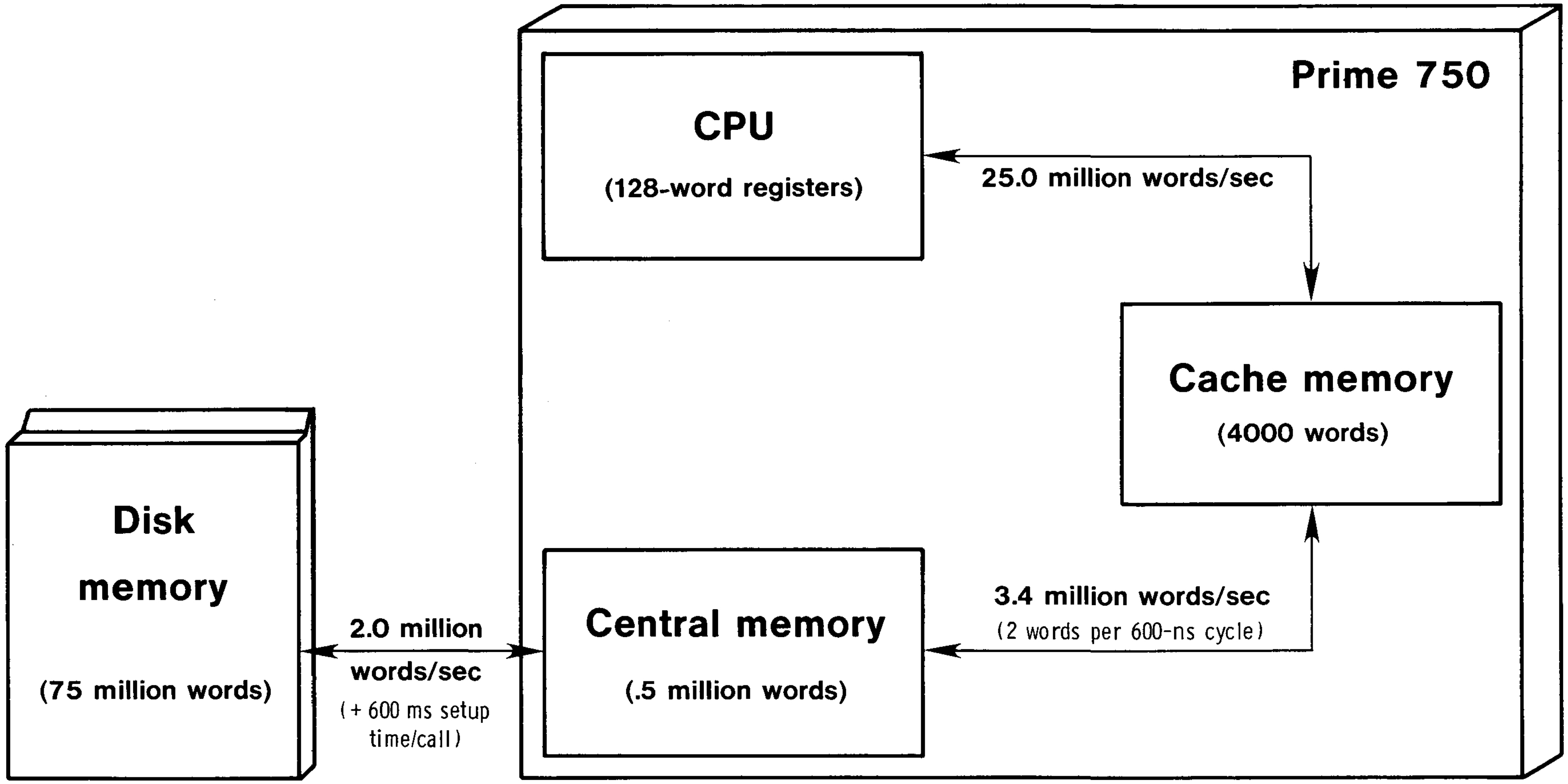
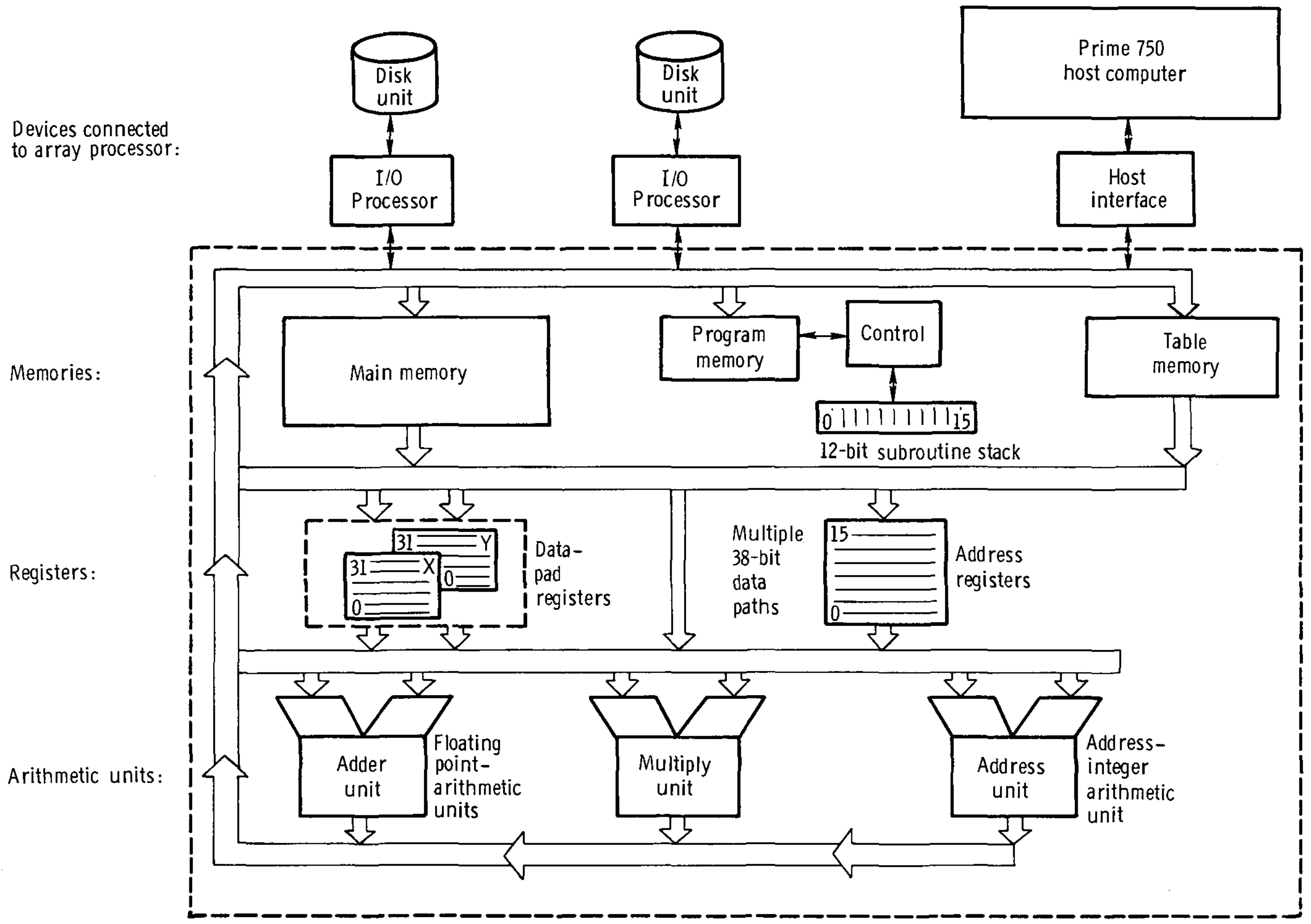


Figure 2.- Schematic drawing of three-level memory organization of Prime 750.
1 word, 32 bits (4 bytes).



Memory type	Word size (bits)	Maximum size (words)
Main	38	320K
Table	38	64K
Program	64	4K

Figure 3.- Floating Point Systems AP-120B array processor.

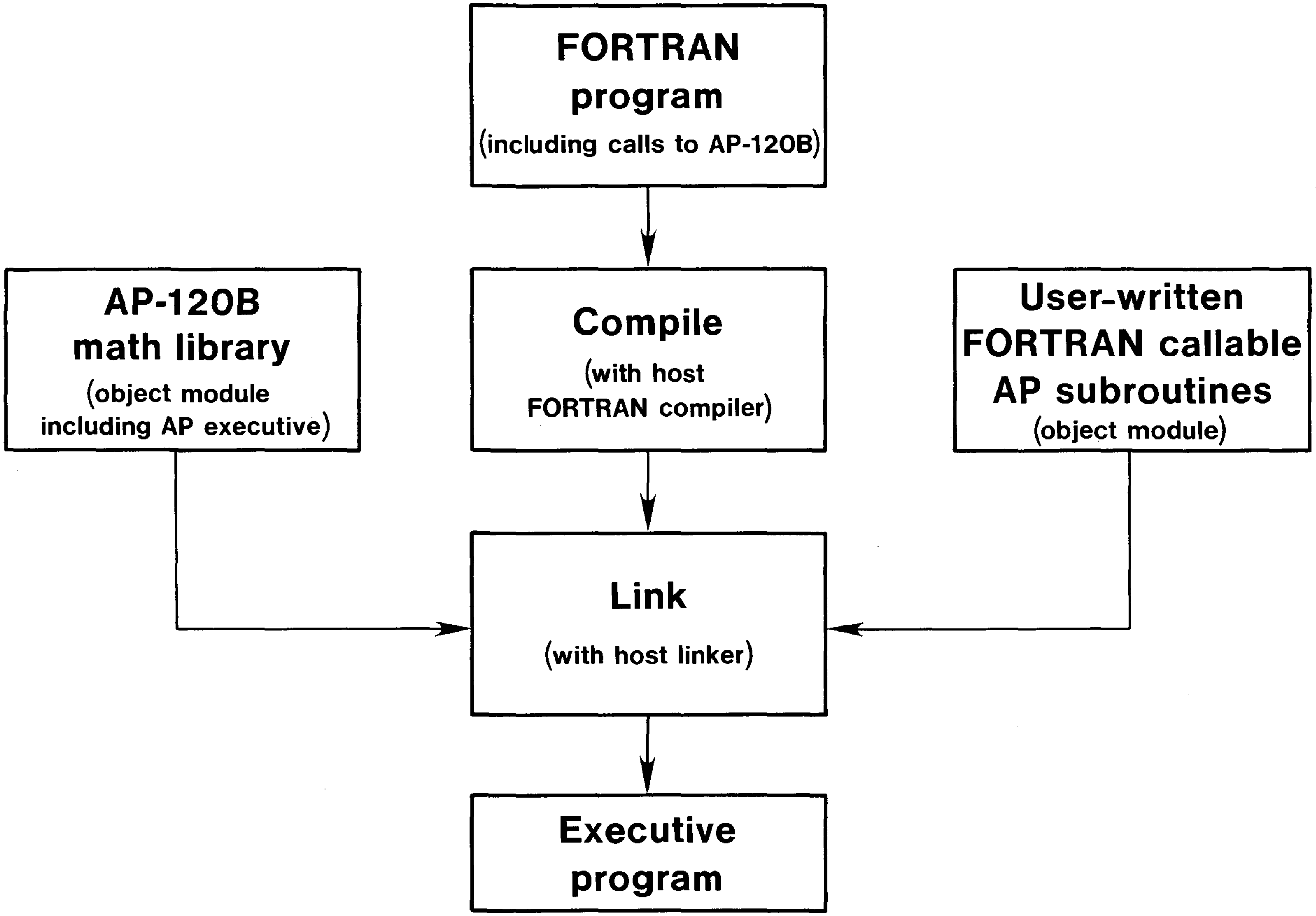
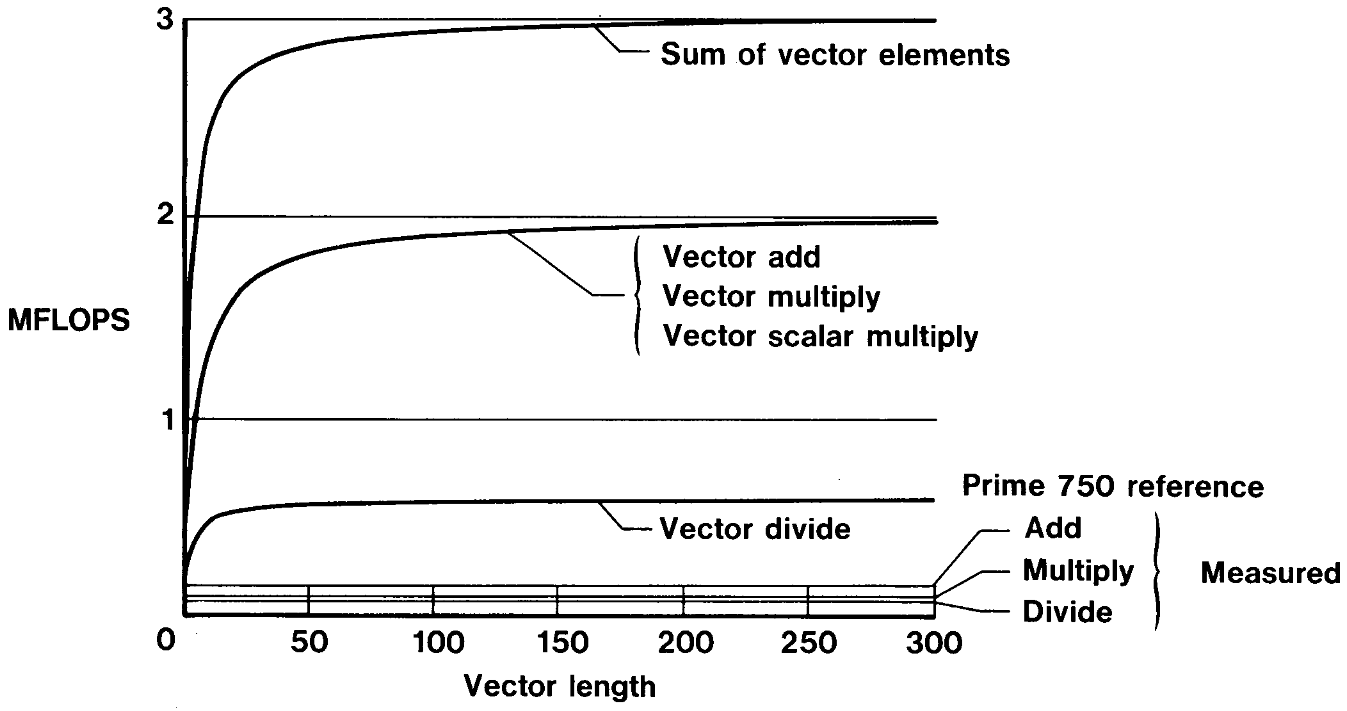
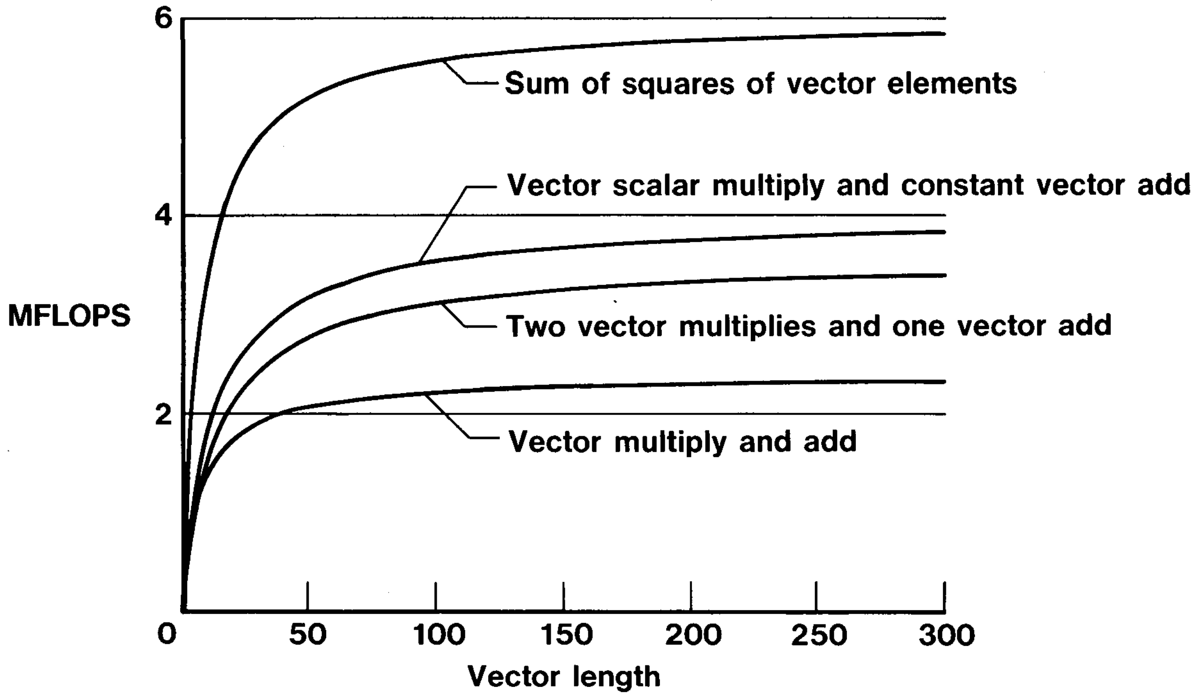

FORTRAN program (including calls to AP-120B)
AP-120B math library (object module including AP executive)
Compile (with host FORTRAN compiler)
User-written FORTRAN callable AP subroutines (object module)
Link (with host linker)
Executive program

Figure 4.- User interface with minicomputer/array-processor system.



(a) Speed of floating-point arithmetic operations.



(b) Combined vector operations performed in parallel.

Figure 5.- Effect of vector length on performance of AP-120B for data already available on the AP.

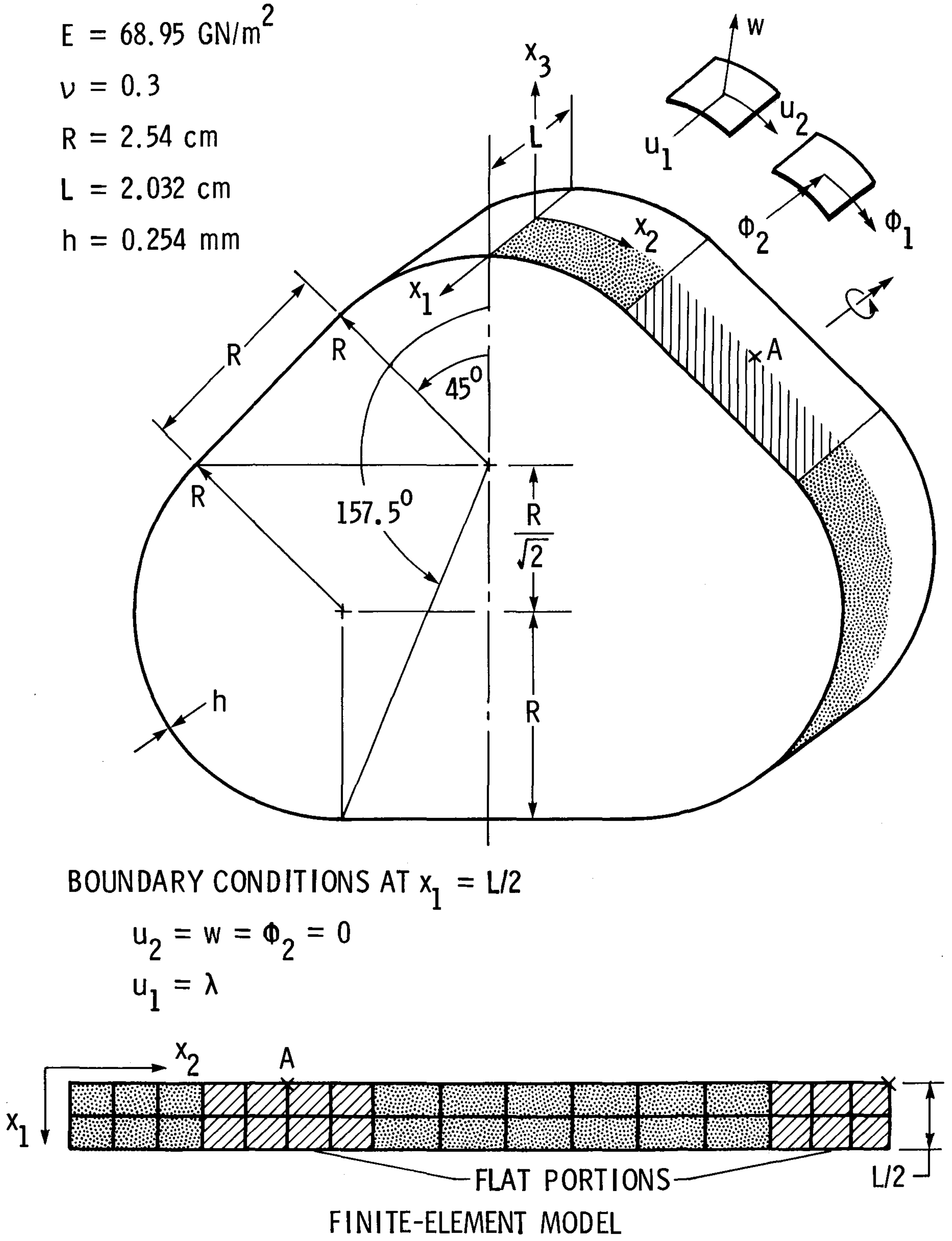


Figure 6.- Pear-shaped cylinder and finite-element model used in present study.

$E = 68.95\ \mathrm{GN/m^2}$

$\nu = 0.3$

$R = 0.1541\ \mathrm{m}$

$h = 0.3556\ \mathrm{mm}$

Boundary conditions at $x_1 = 0$

$u_1 = \lambda$

$u_2 = w = \phi_1 = \phi_2 = 0$

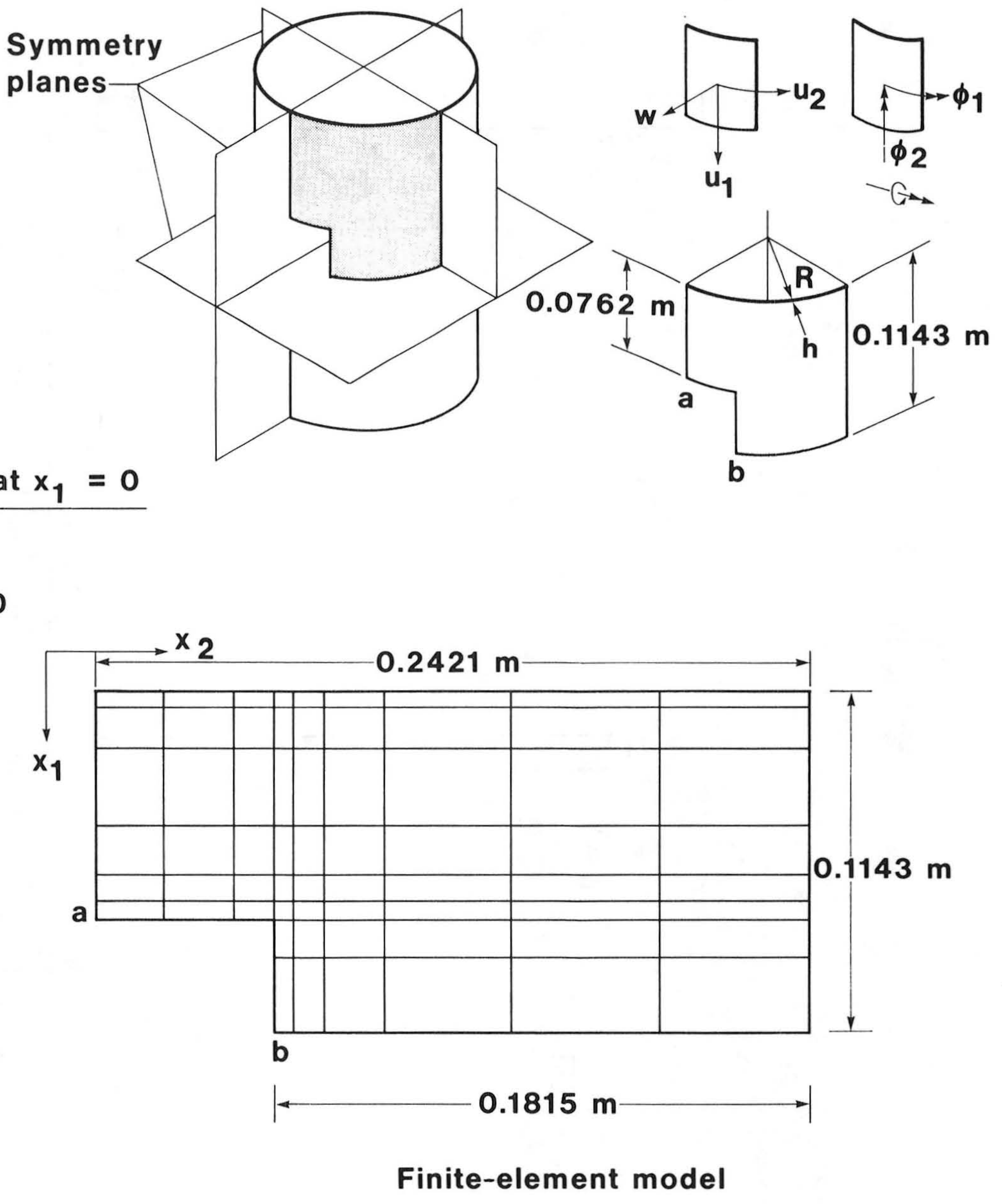
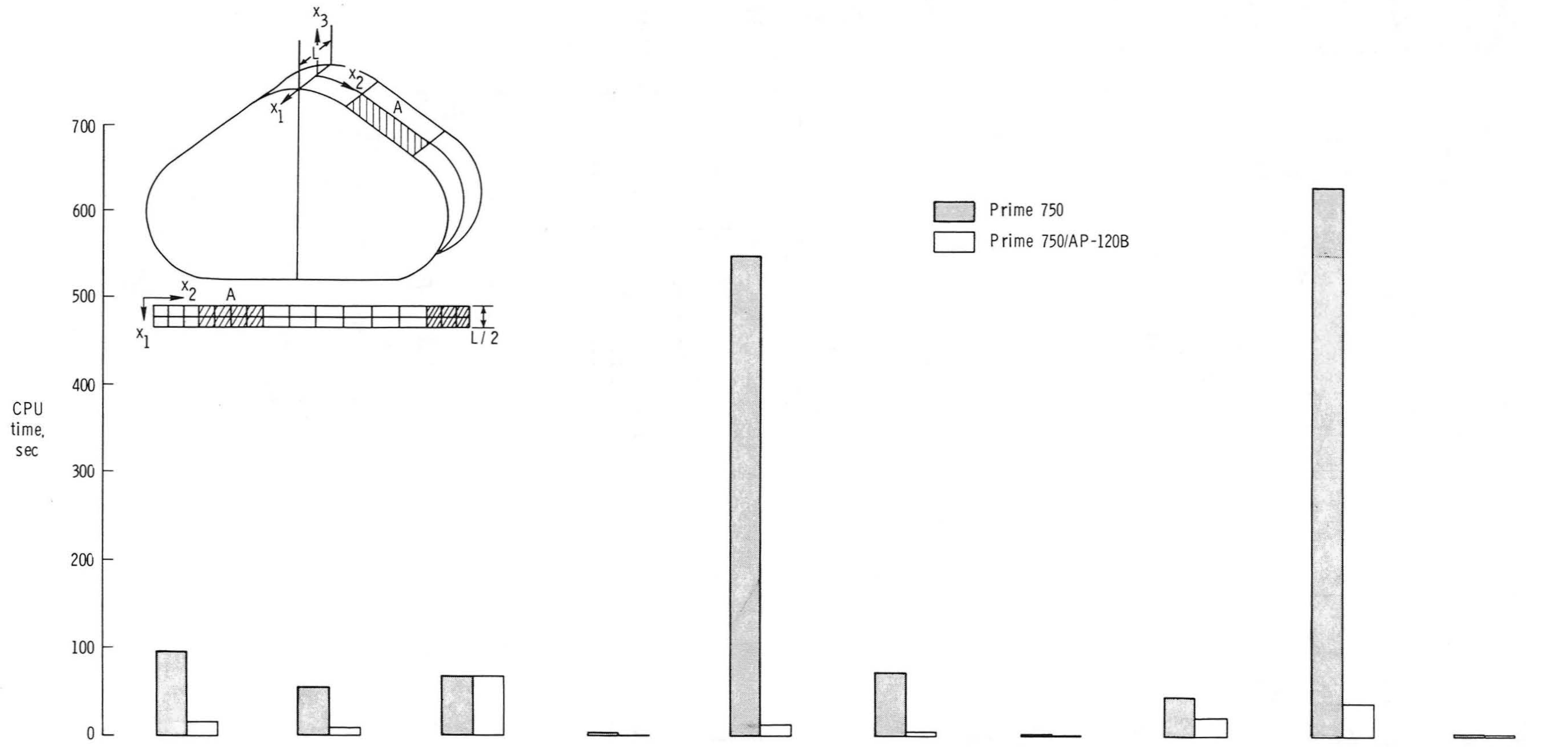
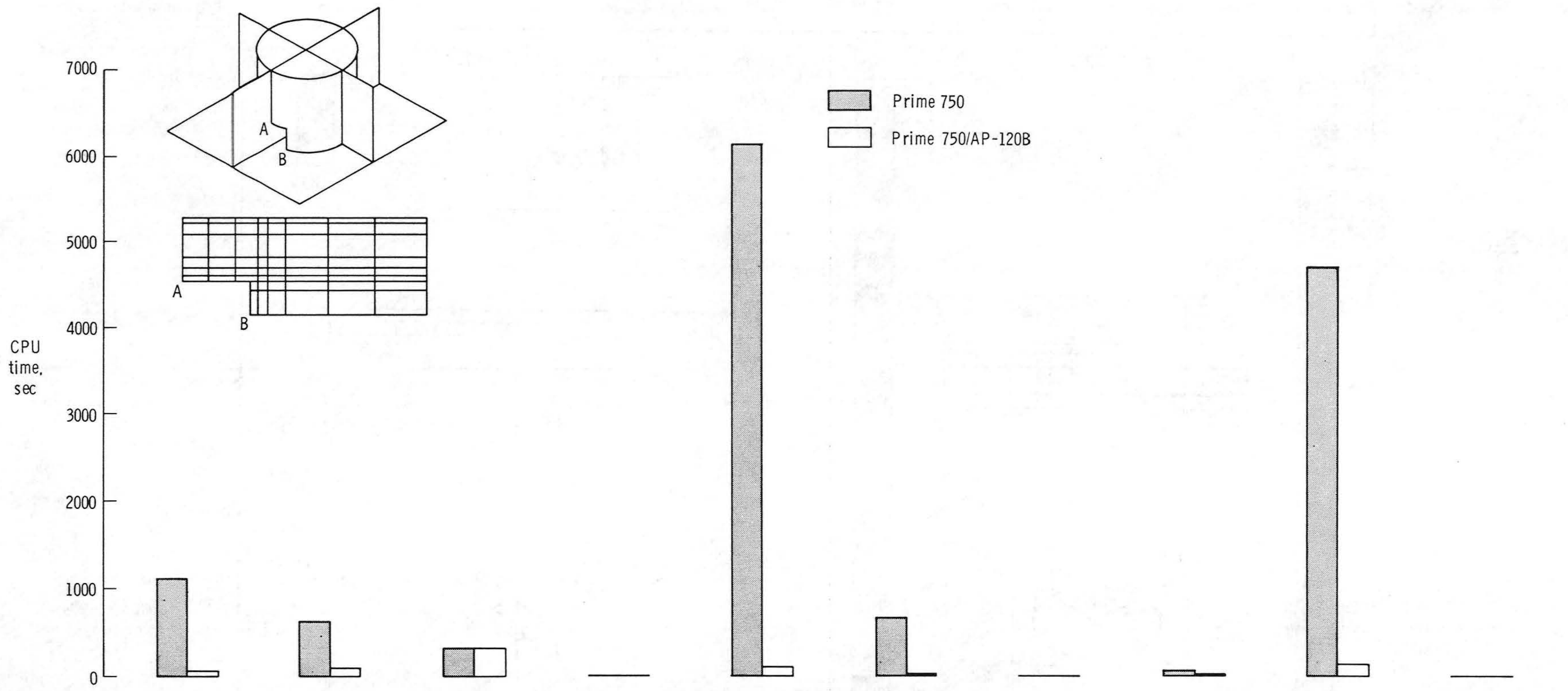


Figure 7.- Cylindrical shell with cutout and finite-element model used in present study.



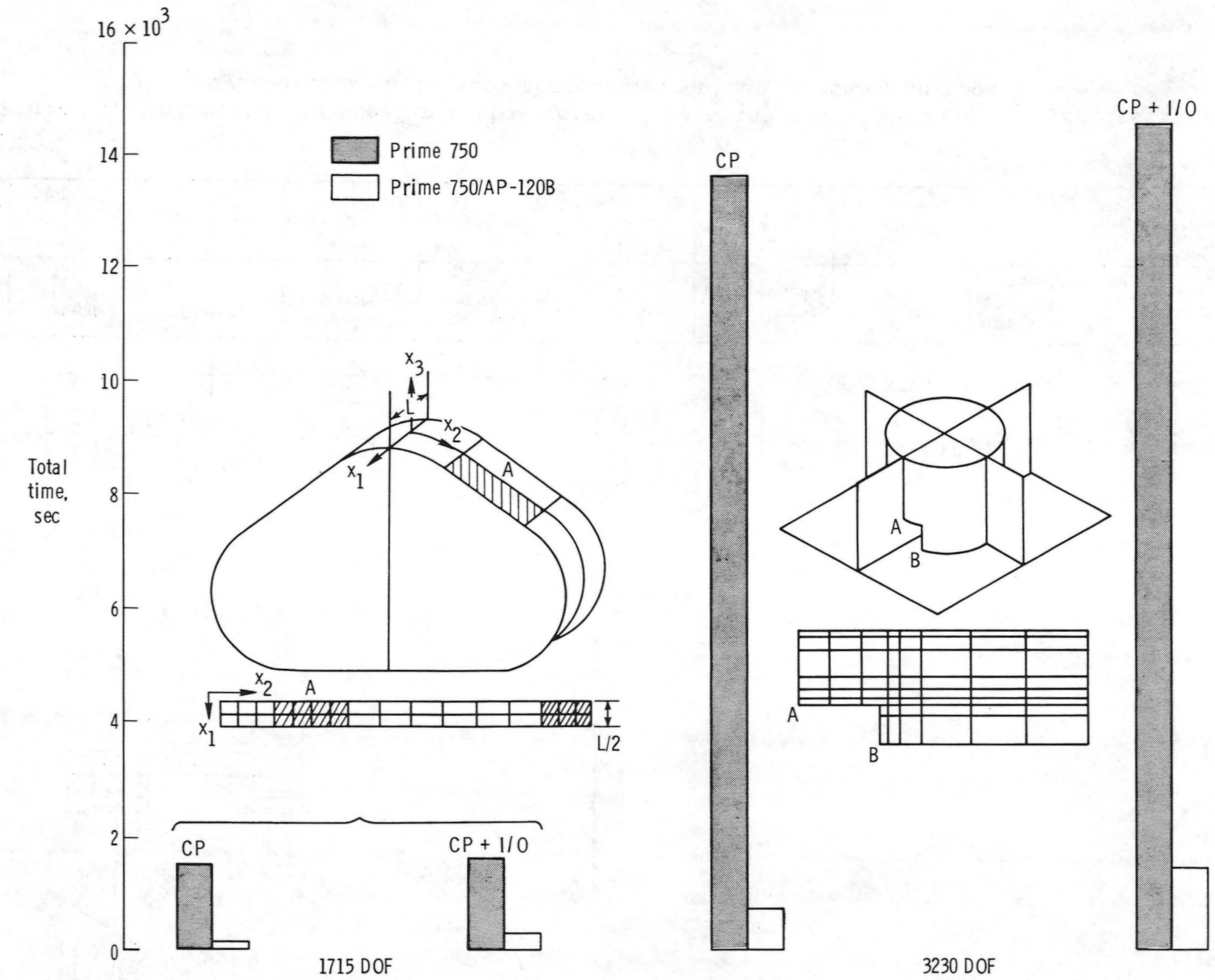
Routine	Generate $[K^{(e)}]$	Generate $F^{(e)}$ and $G^{(e)}$	Assembly	Incorporate boundary conditions	Decompose [K]	Forward reduction, back substitution	Solution update	Stress resultants and energy computation	Generate six basis vectors	Solution of reduced equations
Block no. in fig. B1:	1	1	2	2	3	4	14	18	15	8

Figure 8.- Estimated CPU speedup ratios obtained by using minicomputer/array-processor system. Pear-shaped cylindrical shell is shown in figure 6.



Routine	Generate $[K^{(e)}]$	Generate $F^{(e)}$ and $G^{(e)}$	Assembly	Incorporate boundary conditions	Decompose [K]	Forward reduction, back substitution	Solution update	Stress resultants and energy computation	Generate five basis vectors	Solution of reduced equations
Block no. in fig. B1:	1	1	2	2	3	4	14	18	15	8

Figure 9.- Estimated CPU speedup ratios obtained by using minicomputer/array-processor system. Cylindrical shell with rectangular cutout is shown in figure 7.



Speedup ratio	9.2	5.2	18.1	9.9

Figure 10.- Total speedup ratios for sample problems.

1. Report No. NASA TM-84566	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle POTENTIAL OF MINICOMPUTER/ARRAY-PROCESSOR SYSTEM FOR NONLINEAR FINITE-ELEMENT ANALYSIS		5. Report Date June 1983
		6. Performing Organization Code 505-33-63-01
7. Author(s) Gregg A. Strohkorb and Ahmed K. Noor		8. Performing Organization Report No. L-15532
		10. Work Unit No.
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		14. Sponsoring Agency Code

15. Supplementary Notes

Gregg A. Strohkorb and Ahmed K. Noor: The George Washington University Joint Institute for Advancement of Flight Sciences, Hampton, Virginia.

16. Abstract

A study is made of the potential of using a minicomputer/array-processor system for the efficient solution of large-scale, nonlinear, finite-element problems. A Prime 750 is used as the host computer, and a software simulator residing on the Prime is employed to assess the performance of the Floating Point Systems AP-120B array processor. Major hardware characteristics of the system such as virtual memory and parallel and pipeline processing are reviewed, and the interplay between various hardware components is examined. Effective use of the minicomputer/array-processor system for nonlinear analysis requires the following: (a) proper selection of the computational procedure and the capability to vectorize the numerical algorithms; (b) reduction of input-output (I/O) operations; and (c) overlapping host and array-processor operations. A detailed discussion is given of techniques to accomplish each of these tasks. Two benchmark problems with 1715 and 3230 degrees of freedom, respectively, are selected to measure the anticipated gain in speed obtained by using the proposed algorithms on the array processor. Results of the study of the two benchmarks indicate that these two problems would run faster on a Prime 750 coupled with the AP-120B than on the Prime 750 alone. The 1715 degree-of-freedom problem would run about 5 times faster, and the 3230 degree-of-freedom problem would run about 10 times faster. New advances in array-processor hardware are outlined, and possible improvements in the computational algorithms are discussed. The combination of the two can significantly enhance the effectiveness of the minicomputer/array-processor system for large-scale nonlinear analysis.

17. Key Words (Suggested by Author(s)) Minicomputers, Array processors, Finite elements, Nonlinear analysis, Vectorization, Parallel processing, Distributed processing	18. Distribution Statement Unclassified - Unlimited. Subject Category 39

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 70	22. Price A04

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

NASA-Langley, 1983