NASA TM-86387

NASA Technical Memorandum 86387

EXPLOITING PARALLEL COMPUTING WITH LIMITED PROGRAM CHANGES USING A NETWORK OF MICROCOMPUTERS

J. L. Rogers, Jr., and J. Sobieszczanski-Sobieski

FEBRUARY 1985

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

EXPLOITING PARALLEL COMPUTING WITH LIMITED PROGRAM CHANGES USING A NETWORK OF MICROCOMPUTERS

J. L. Rogers, Jr. and J. Sobieszczanski-Sobieski

NASA Langley Research Center

INTRODUCTION

As the speed of a single processor computer approaches a physical limit, computer technology is beginning to advance toward parallel processing to provide even faster speeds. Network computing and multiprocessor computers are two discernible trends in this advancement. Given the two extremes, a few powerful processors or many relatively simple processors, it is not yet clear how engineering applications can best take advantage of parallel architecture. Neither is it clear at this time the extent to which engineering analysis programs will have to be recoded to take advantage of this new hardware. Initial investigations of these questions can begin immediately by exploiting the physical parallelism of selected problems and the modular organization of existing programs to solve these problems.

To gain experience in exploiting parallel computer architecture without making major changes to the code, an existing program was adapted to perform finite element analysis by distributing substructures over a network of four Apple IIe microcomputers connected to a shared disk (Rogers and Sobieszczanski-Sobieski, 1983). This network of microcomputers is regarded merely as a simulator of a parallel computer because it should be obvious that substructure analysis of a practical problem of significant size should be performed on a computer with much more power than this particular microcomputer. In this network, one microcomputer controls the entire process while the others perform the analysis on each substructure in parallel.

After the substructure analysis was implemented in parallel, a new experiment was planned using this system. In this

experiment, the substructure analysis is used in an iterative, fully-stressed, structural resizing procedure to evaluate resizing in which the analyses of all substructures are not completed during a single iteration. Methods to handle the resulting mixture of old and new analysis data, referred to as asynchronous parallelism, need to be developed for parallel computing applications. Although the present work involves only structural analysis, this research gives some initial insight on how to configure multidisciplinary analysis and optimization procedures for decomposable engineering systems using either high-performance engineering workstations or a parallel processor supercomputer. In addition, the operational experience gained will facilitate the implementation of analysis programs on these new computers when they become available in an engineering environment.

BACKGROUND

In 1975 a feasibility study (Universal Analytics, 1975) was performed to determine the effort required to convert NASTRAN (NASTRAN User's Manual, 1983) to execute on the ILLIAC IV computer, and to assess the advantages that would be gained from such a conversion. The projected advantages in speed improvement were significant. For example, the decomposition of a 10,000 degree-of-freedom matrix on the ILLIAC IV was estimated to be 40-100% faster than on a CDC 6600 computer when the matrix could not be contained in central memory. The problem that the study pointed out was that the code conversion effort would require 110-140 man months over a period of 36-50 months. If funds had been supplied and the project begun in 1976, it would probably not have been completed until 1980. About two years later, in 1982, the ILLIAC IV was taken off-line. This historical example illustrates the difficulties that could be encountered in future wholesale conversions of engineering analysis codes to parallel processor computers which appear to be the wave of the future (Siewiorek, 1982; and Noor, Storaasli, and Fulton, 1983)

While such wholesale conversion efforts should ultimately provide the greatest increases in efficiency, it is also important in the interim to benefit from the speed improvements offered by parallel computing without the cost, manpower, and time involved in the conversion of major analysis codes. This research demonstrates that for structural analysis, the current investment in sequential, modular structural analysis programs can be used to exploit parallelism of a network of computers.

APPROACH

The approach taken for this project was to establish reference results using the Engineering Analysis Language (Whetstone, 1980), called EAL, to analyze a finite element model that was not substructured. An existing small finite element analysis code was then modified to handle substructures and applied to the same model on a CYBER mainframe computer. Next, this program was implemented on a microcomputer to test the substructure method sequentially. The program was then distributed over a network of these microcomputers with little change to the analysis code to test the substructure method executed in parallel. A Fully Stressed Design (FSD) capability was added to test the behavior of substructure analysis in an iterative process in which some of the analyses were completed before others.

The Model

The finite element model used for testing is shown in figure 1. This model contains 16 joints, 21 beam elements, and 42 degrees-of-freedom (the size of the model was limited by the memory of the microcomputer). The framework has three substructures with each substructure composed of seven beams. The cross-sections and material properties are identical for all beams. A load is applied at one of the boundary nodes as shown in figure 1.

The Small Finite Element Program

Input for the model was written for a small, undocumented finite element program developed in the past for a CYBER computer without any intent to ever use it for parallel processing. It did not even have an explicit substructuring capability. In this study, this program represented an "investment in existing software" that was to be salvaged. The results from the unchanged program were verified against the reference run. New code for substructuring based on equations from (Przemieniecki, 1968) was then added to the program. The model was divided into three substructures with the new code used to compute the boundary stiffness matrix for each substructure using equation 1:

$$K_b = K_{bb} - K_{bi} K_{ii}^{-1} K_b^T \qquad (1)$$

Each of the three 18x18 substructure stiffness matrices was reduced to 6x6 equivalent beam stiffness matrices (figure 2). These three stiffness matrices were input to the program, assembled to represent a stiffness-equivalent framework composed of three beams, each beam representing one substructure. The forces and displacements at the boundary

nodes were computed for each such beam. Modifications were made to the program for reading these forces from a file and applying them to the corresponding substructures. By applying support conditions to the substructures, solutions were obtained for the interior node displacements, internal forces, and elemental stresses. These results were also verified against the reference run. It should be noted that the substructure analysis was simplified because the external loads were applied only to the boundary nodes. Should any loads be applied to the interior substructure nodes, it would have been necessary to add code to transfer these loads to the boundary nodes.

Conversion to the Microcomputer

At this point, the program for sequentially performing substructure analysis existed on a CYBER mainframe computer. The next step was to convert the program to the microcomputer. Since the entire program was written in FORTRAN-77, the move was quite simple and the program was contained in the microcomputer's 64K byte memory without overlay. Although the program itself was entirely core resident, the test case shown in figure 1 was too large for analysis without substructuring. Therefore, the first step on the microcomputer was to run the substructuring sequentially. The problem took 57 minutes to execute. The results were verified against the reference run with little loss in precision (less than 1%).

Distributing the System

The approach selected for distributing the system was to use one microcomputer to execute a controller program and three microcomputers to analyze each of the substructures. All of the microcomputers were connected to a 20MB Corvus hard disk which was used for data communication among the computers. The operations assigned to each computer are shown in figure 3. The controller program started the system (operation 0), assembled the substructure stiffness matrices and solved for the forces on each substructure at the boundary nodes (operation 2), and output the data (operation 4). The substructure programs computed the substructure stiffness matrices (operation 1), and used the forces from the controller program to solve for internal forces, node displacements, and elemental stresses for each substructure (operation 3). Note that parallelism only exists in operations 1 and 3. The output of the data (operation 4) also could have been distributed, but it was found to be easier to keep it centrally located.

When distributing the system to four microcomputers, the purpose was to minimize changing the original analysis code. Only the procedures involved in operations 0, 2, and 4 were retained in the controller program while only those procedures

involved in operations 1 and 3 were retained in the substructure program.

A subroutine was added to both the controller program and the substructure program to schedule their execution. This scheduling was accomplished by using three files on the shared disk; one file for each substructure program. When it was time for the controller program to execute each of the three files contained a zero, and when it was time for a substructure program to execute, its respective file contained a nonzero number. Each program queried its file and if it was not its turn for execution it was put in a "holding pattern" by performing a simple multiplication loop before querying again. The system could have been implemented on only three processors with one processor doubling for executing the controller and substructure programs.

The ideal is to reduce the time required to solve the same problem sequentially on a single processor to (time/n) where n is the number of processors used to solve the problem. However, it is seen in figure 3 that not all of the calculations can be executed in parallel. In addition, some time was lost in an inevitable overhead such as checking and looping while waiting for a substructure or controller program to finish executing. Thus, the parallel system with substructures took about 27 minutes to complete execution, .47 of the time required for the reference run. This is short of the ideal, .25, but is still more than twice as fast as the sequential system.

The particular division of the structure from figure 1 into substructures is, of course, not the only one possible. If a larger number of smaller substructures was used, larger numbers of parallel computers could have been employed. However, the larger the number of substructures the larger the dimensionality of the assembled structure stiffness matrix (ultimately, if each substructure represents a single beam component, the assembled stiffness matrix would return to the size it would have had if no substructuring was used). Consequently, to minimize the overall computer time, an attempt should be made to balance the size of the assembled structure stiffness matrix against the size and number of the substructure stiffness matrices. The degree of the time reduction depends also on the number of substructuring levels (Sobieszczanski-Sobieski, James, and Dovi, 1983). Thus, tailoring the analysis process for a particular application to take advantage of multiprocessor efficiency is an important issue that faces an analyst using a multiprocessor system.

Resizing

An FSD algorithm was added to examine the behavior of parallel substructure analysis in an iterative process (figure 4). The FSD was performed by resizing all the beams in a given substructure according to the ratio of the maximum absolute normal stress occuring in the substructure to a specified allowable stress. The stress ratio was used as a scale factor to modify the beam cross-sectional moment of inertia. Consistently, the cross-sectional area was multiplied by the square root of the scale factor, and the cross-section linear dimensions were all multiplied by the scale factor to power 1/4. At this point, resizing was synchronized, which means the process would always wait until all data were updated before processing rather than mixing old and new data. An iteration history of the changes in the design variable (plotted as the factor on cross-section linear dimension) for each substructure is shown in figure 5.

Asynchronous Resizing

Since most of the engineering calculations performed in support of design are iterative in nature, the computational behavior of an iterative distributed process in which some subtasks are completed later than others because of unequal computational requirements for various subtasks is of significant interest (Baudet, 1978; and Sobieszczanski-Sobieski, 1982). If such an imbalance of computational requirements occurs, a choice can be made to let the iterative process continue, temporarily using old data for those processes which are late. The process then becomes asynchronous as it mixes new and old data. The effect of this mixing on the convergence and efficiency can easily be tested in a parallel system such as described above. The tests are conducted by bypassing analysis of selected substructures in some iterations. Obviously during the first loop through the system, all of the substructures will be analyzed to provide a starting point.

There is a large number of different ways in which an asynchronous iterative process can proceed. Using the framework structure from figure 1 as an example, it is conceivable to have at least the following variants.

1. Referring to figure 3, consider being at the outset of iteration "i". Operations 1.1, 1.2, and 1.3 are expected to yield the boundary stiffness matrices for substructures 1,2 and 3, all of which having been resized as a result of an FSD operation at the end of the previous iteration "i-1". Assume that operation 1.1 is late but the process moves on anyway using the old boundary stiffness matrix from iteration "i-1", that does not reflect the "i-1" resizing. That means that operation 2 combines the updated matrices for substructures 2

and 3 with an outdated matrix for substructure 1. In operations 3.1, 3.2, and 3.3, consistently, an old stiffness matrix that does not reflect the "i-1" resizing is used, while the updated stiffness matrices are used in operations 3.2, and 3.3. After this analysis based on the partially incorrect data, all substructures, including substructure 1, are subject to the FSD resizing.

2. Proceed as above, but do not resize that particular substructure for which the old stiffness matrix was used in the analysis (substructure 1 in this example).

3. Complicate variants 1 and 2 by changing: the number of substructures that are assumed to be "late", the number of iterations over which the old data are being used for each substructure, etc. Obviously, a very large number of possibilities can be considered.

It was expected that, for this particular model, the asynchronous processing would have little effect on the convergence other than slowing it down by going through more iterations. In fact, the asynchronous operation in this case may be regarded as a continuation of the FSD process from an artificially injected new starting point. In addition, an analogy can be made between this process and other iterative methods such as the Gauss-Seidel iterative algorithm for solving linear algebraic equations. These methods are error insensitive, i.e. if an error is entered into the process, the process recovers after several iterations and proceeds as if no error had been introduced. One may speculate that a different behavior will be observed in cases when nonlinear programing is used instead of FSD for nonconvex cases. Then, there will be a potential for such an asynchronous operation to trigger a switch to another path through the design space that could end up at a different local minimum.

To determine if the above expectations were correct, numerous test cases were executed. The existence of coupling among the substructures was demonstrated by holding substructure 1 constant. As seen in figure 6, this case converged to different results than is seen in the following figures thus showing that the substructures are coupled. Next, variants 1 and 2 were tested. The results shown in figures 7 and 8, respectively, indicate that the asynchronous operation, shown as connected lines, had only a slight influence on the convergence as manifested in small discrepancies that can be seen between the lines and symbols from the synchronous sizing. For instance, asynchronous results for substructures 2 and 3 (figure 7) are above the synchronous ones but both results converge after about eight iterations.

Variant 3, complicating variant 2, was tested using seven different cases. It was decided to complicate variant 2 rather than 1 because of the ease of implementing the variables. Table 1 lists the substructure(s) and iteration(s) that were delayed for each test case. In each of the test cases the results led to the expected behavior. Figures 9 and 10 (test cases 6 and 7 respectively) demonstrate typical results.

Table 1 Test Cases for Asynchronous Processing

CASE	SUBSTRUCTURE DELAYED	ITERATIONS DELAYED
1	2	2
2	1,2	2
3	1	Every other iteration
4	2	Every other iteration
5	1,2	Every other iteration
6	1	Even iterations beginning with 2
	and 2	Odd iterations beginning with 3
7	Random combinations	Random

CONCLUDING REMARKS

An experiment was conducted to determine if advantage can be taken of parallel processing without making major changes to an analysis code. This experiment used a network of four microcomputers to simulate a parallel processing computer. A small finite element analysis computer program with a substructuring capability was applied to a framework of beams. One microcomputer controlled the system while the other three analyzed the substructures. The results verified that the computer time was indeed reduced relative to the time required for solution on a single computer. The reduction was achieved with almost no change to the analysis portion of the code. The experiment also included resizing of the design variables using a Fully Stressed Design algorithm to simulate an iterative optimization to obtain an indication of the effect of asynchronous parallel computing on the convergence of an iterative process. Results from 10 test cases indicated that, for this model, asynchronous processing did not affect convergence other than possibly causing the process to go through more iterations.

These results are not completely general in that they only apply to an iterative process which is monotonically convergent. This process is typical of many processes in design. In general, if an iterative process is nonconvex (dependent on the starting point and the path taken) then this

conlusion would not apply and the asynchronous process may lead to different results.

REFERENCES

Baudet, G. (1978) The Design and Analysis of Algorithms for Asynchronous Multiprocessors. Ph.D Thesis, Department of Computer Science, Carnegie Mellon University.

Noor A.; Storaasli, O.; and Fulton, R. (1983) Impact of New Computing Systems on Finite Element Computations. ASME Publication H00275, pp. 1-32.

Przemieniecki, J. (1968) Theory of Matrix Structural Analysis. Ch. 9, McGraw-Hill Book Co.

Rogers, J., Jr. and Sobieszczanski-Sobieski, J. (1984) Initial Experiences with Distributing Structural Calculations Among Computers Operating in Parallel. NASA CP 2335, pp. 45-54.

Siewiorek, D. (1982) State of the Art in Parallel Computing. Abstracted from Computer Structures: Principles and Examples, McGraw-Hill Book Co.

Sobieszczanski-Sobieski, J. (1982) A Linear Decompostition Method for Large Optimization Problems. Blueprint for Development. NASA TM-83248.

Sobieszczanski-Sobieski, J.; James, B.; and Dovi, A. (1983) Structural Optimization by Multilevel Decomposition. AIAA Paper No. 83-0832.

The NASTRAN User's Manual (1983) NASA SP-222.

Universal Analytics Inc. (1975) Feasibility Study for the Implementation of NASTRAN on the ILLIAC IV Parallel Processor. NASA CR-132702.

Whetstone, D. (1980) EISI-EAL: Engineering Analysis Language. Proceedings of the Second Conference on Computing in Engineering, ASCE, pp. 276-285.

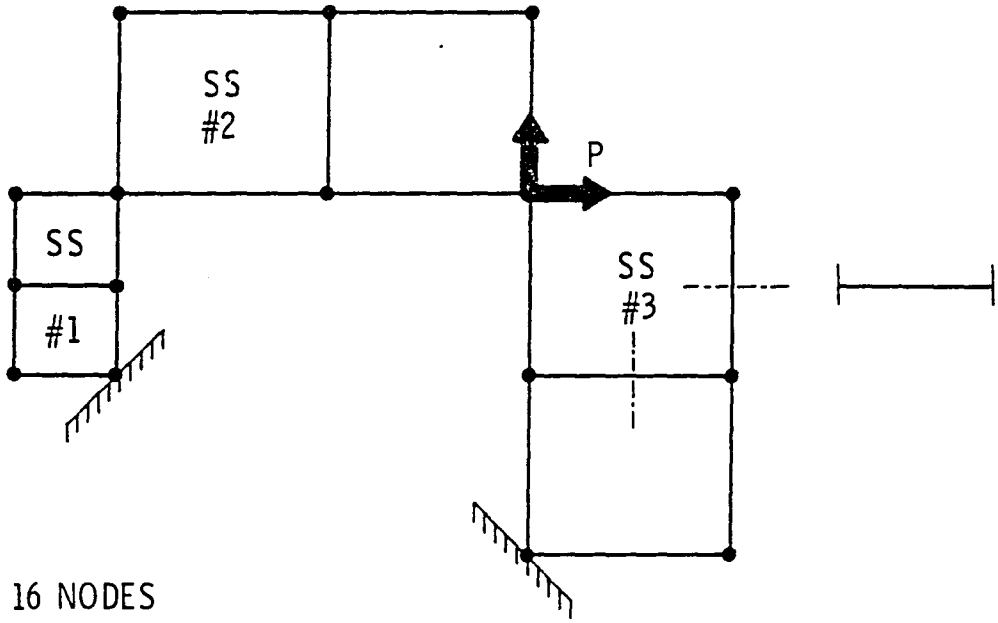

Figure 1 - Framework used for testing

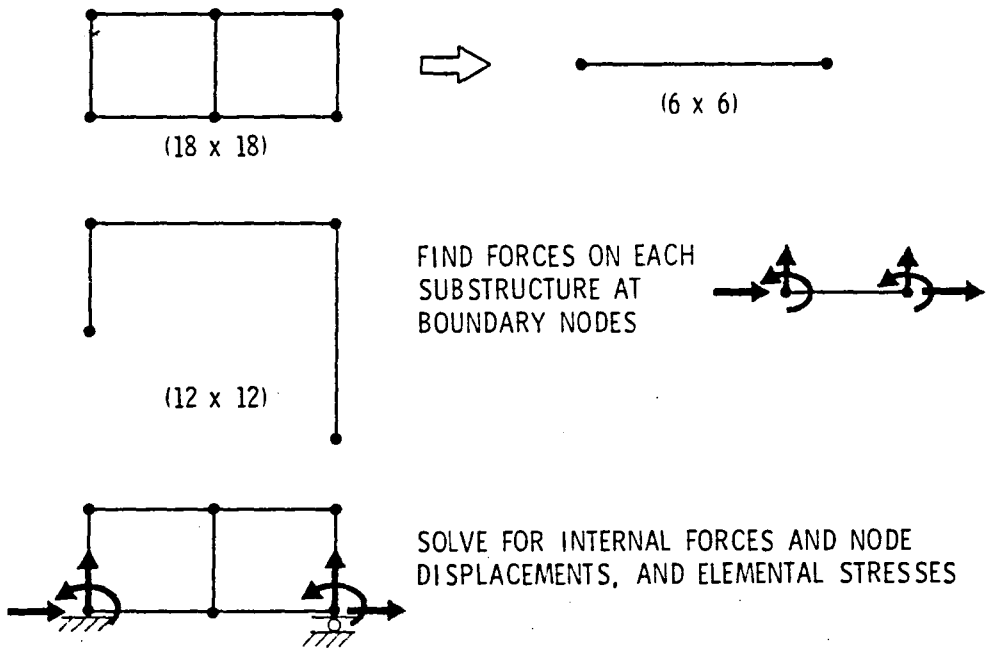

Figure 2 - Actions being taken at each step

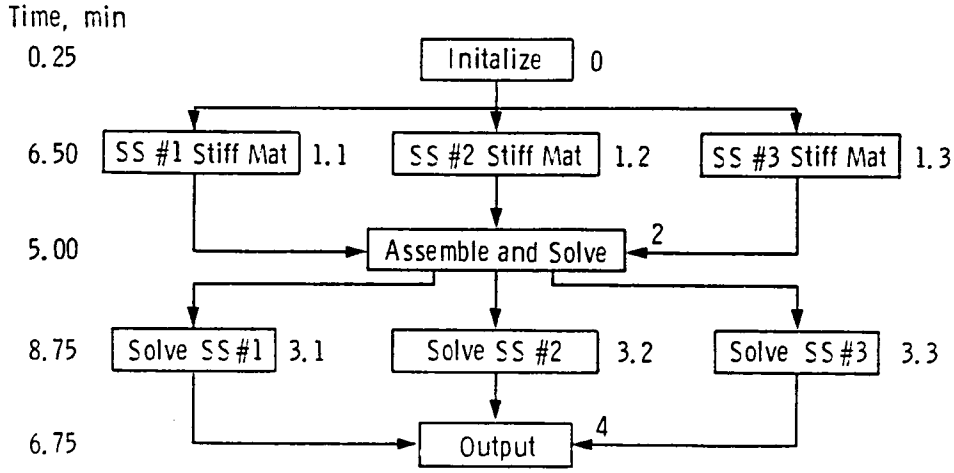

Figure 3 - Flowchart of substructure analysis

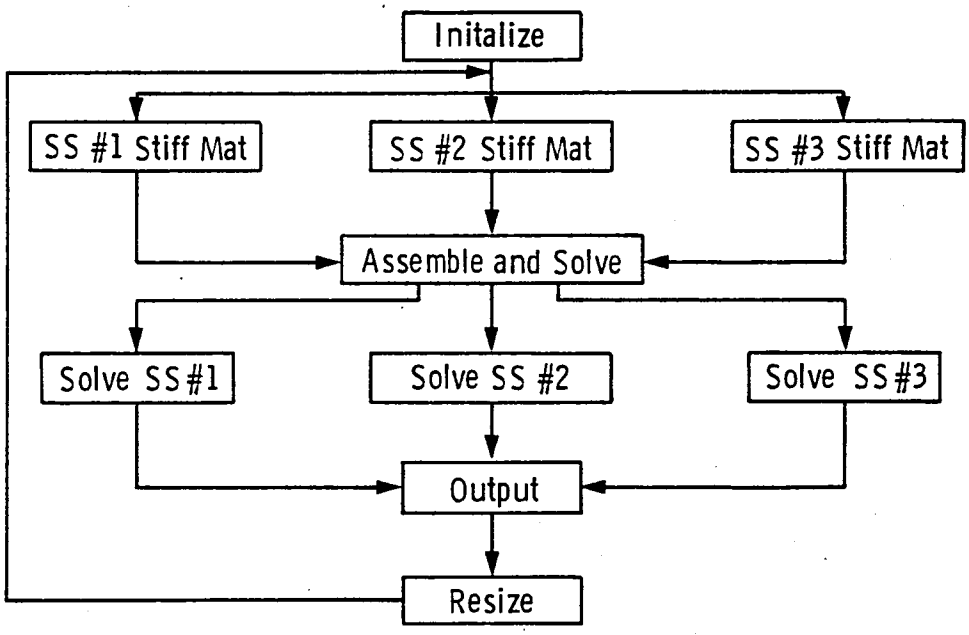

Figure 4 - Flowchart of substructure analysis with resizing

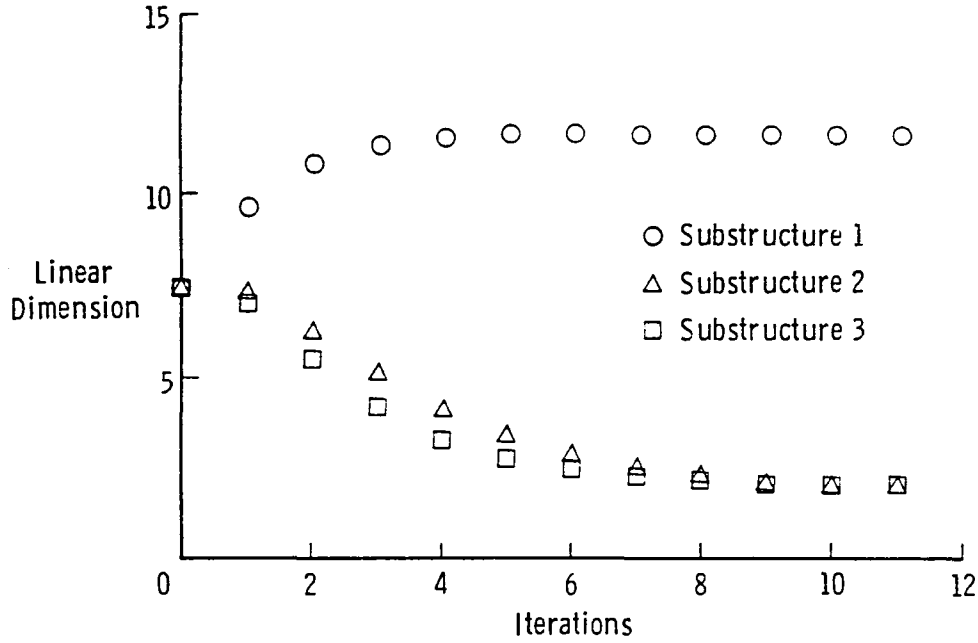

Figure 5 - Synchronous resizing

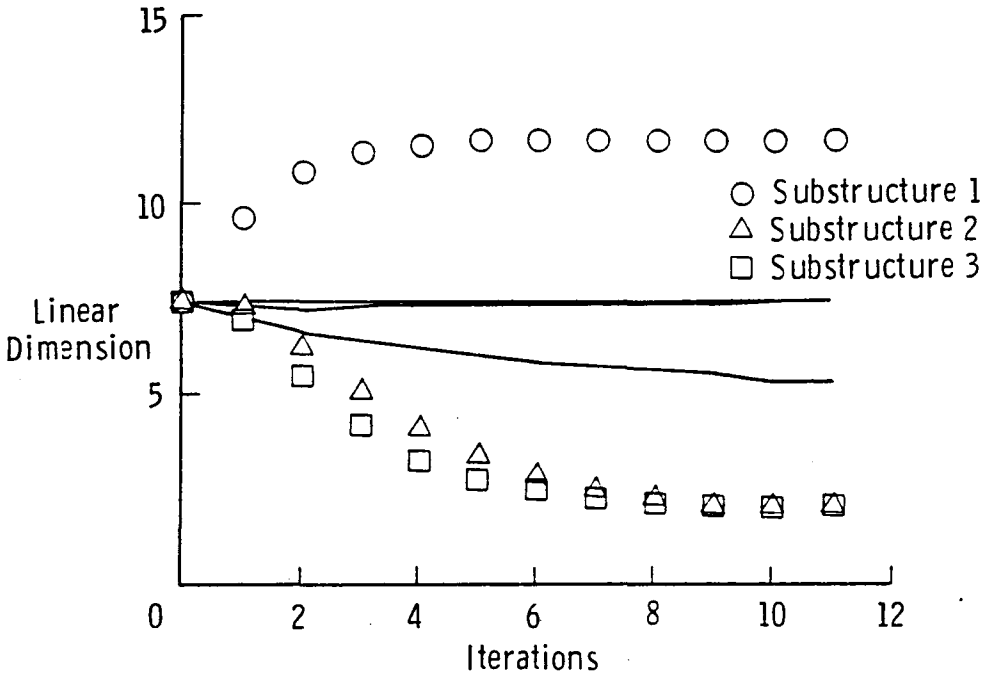

Figure 6 - Asynchronous resizing showing coupling

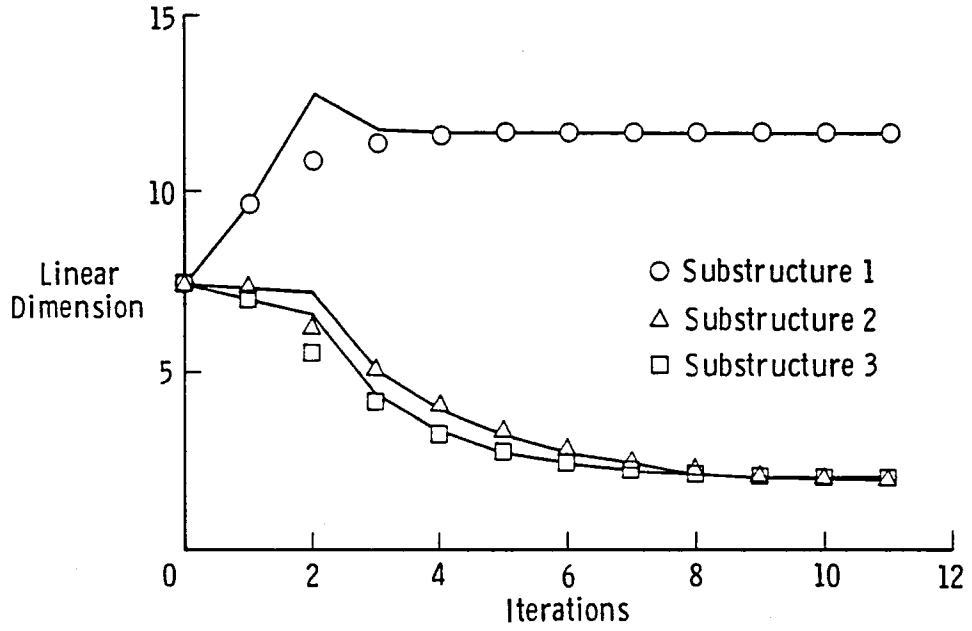

Figure 7 - Asynchronous resizing
(variant 1)

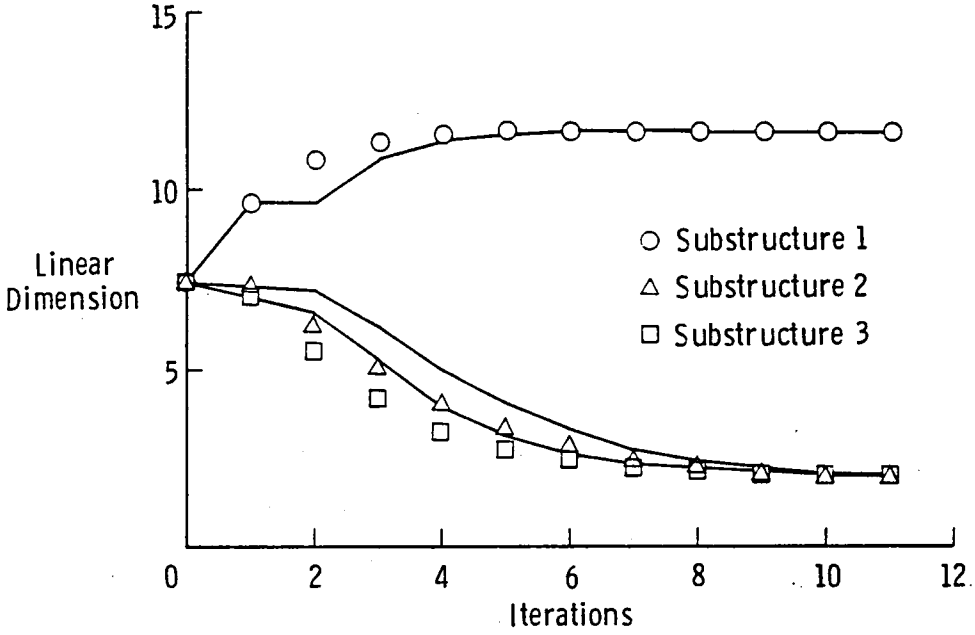

Figure 8 - Asynchronous resizing
(variant 2)

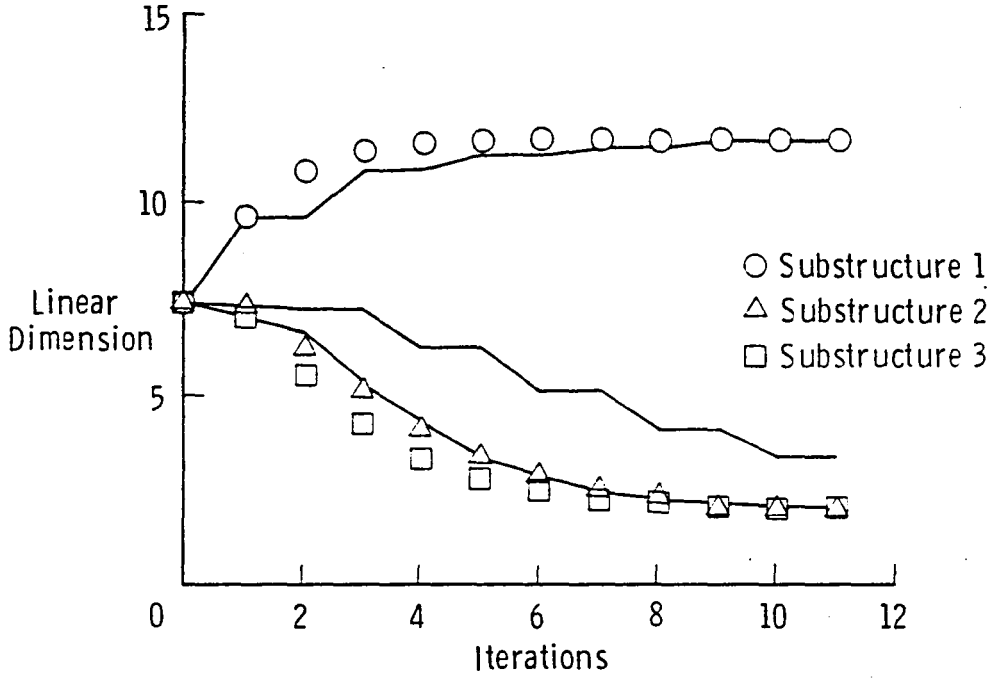

Figure 9 - Asynchronous resizing (delay substructure 1 on even iterations and substructure 2 on odd iterations)

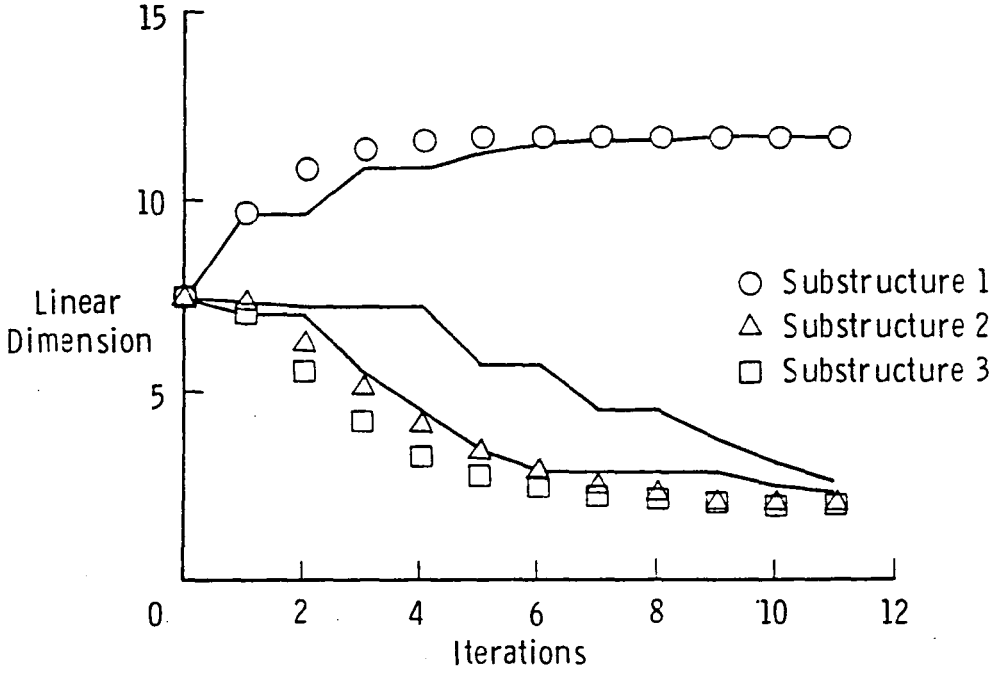

Figure 10 - Asynchronous resizing (random)

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-86387		

4. Title and Subtitle	5. Report Date
Exploiting Parallel Computing with Limited Program Changes Using a Network of Microcomputers	February 1985
	6. Performing Organization Code: 505-33-53-12
7. Author(s): J. L. Rogers, Jr., and J. Sobieszczanski-Sobieski	8. Performing Organization Report No.
	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

16. Abstract

As the speed of a single processor computer approaches a physical limit, computer technology is beginning to advance toward parallel processing. Network computing and multiprocessor computers are two discernible trends in this advancement. It is not clear at this time the extent to which engineering analysis programs will have to be recoded to take advantage of these new hardware opportunities.

Since most calculations supporting design are iterative, the computational behavior of an iterative distributed process in which some subtasks are completed later than others because of an imbalance in computational requirements is of significant interest. If such an imbalance occurs, the iterative process can continue choosing either to temporarily use old data for those processes which are late (asynchronous processing) or waiting for the completion of all processes (synchronous processing).

To study the effects of asynchronous processing, a small existing program was converted to perform finite element analysis by distributing substructure analysis over a network of four Apple IIe microcomputers connected to a shared disk, simulating a parallel computer. The substructure analysis uses an iterative, fully stressed, structural resizing procedure. A framework of beams divided into three substructures is used as the finite element model. The effects of asynchronous processing on the convergence of the design variables are determined by not resizing particular substructures on various iterations. Numerous test cases were executed.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Parallel processing, finite element analysis, substructuring, microcomputers, fully-stressed design	Unclassified - Unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	15	A02

End of Document