NASA-CR-172,466

NASA Contractor Report 172466

ICASE REPORT NO. 84-47

ICASE

PARALLEL TRIANGULARIZATION OF SUBSTRUCTURED FINITE ELEMENT PROBLEMS

Michael R. Leuze

Contract No. NAS1-17130
September 1984

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

NASA
National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23665

```
 14        1       1 RN/NASA-CR-172466
               DISPLAY 14/2/1
       85N11569*#   ISSUE 2   PAGE 241   CATEGORY 61   RPT#: NASA-CR-172466
       ICASE-84-47 NAS 1.26:172466   CNT#: NAS1-17130 NSF MCS-83-05693   84/09/00
       20 PAGES  UNCLASSIFIED DOCUMENT
UTTL: Parallel triangularization of substructured finite element problems
       TLSP: Final Report
AUTH: A/LEUZE, M. R.   PAA: A/(Vanderbilt Univ.)
CORP: National Aeronautics and Space Administration. Langley Research Center,
       Hampton, Va.    AVAIL.NTIS   SAP: HC A02/MF A01
MAJS: /*COMPUTATION/*FINITE ELEMENT METHOD/*PARALLEL PROCESSING (COMPUTERS)/*
       PARTITIONS (MATHEMATICS)/*STIFFNESS MATRIX
MINS: / DIFFERENTIAL EQUATIONS/ GRAPH THEORY/ HEURISTIC METHODS/ LINEAR
       EQUATIONS/ PROBLEM SOLVING
ABA:  Author
ABS:  Much of the computational effort of the finite element process involves
       the solution of a system of linear equations. The coefficient matrix of
       this system, known as the global stiffness matrix, is symmetric, positive
       definite, and generally sparse. An important technique for reducing the
       time required to solve this system is substructuring or matrix
       partitioning. Substructuring is based on the idea of dividing a structure
       into pieces, each of which can then be analyzed relatively indepenently.
       As a result of this division, each point in the finite element
       discretization is either interior to a substructure or on a boundary
ENTER:
```

PARALLEL TRIANGULARIZATION OF SUBSTRUCTURED FINITE ELEMENT PROBLEMS

Michael R. Leuze
Vanderbilt University

Abstract

Much of the computational effort of the finite element process involves the solution of a system of linear equations. The coefficient matrix of this system, known as the global stiffness matrix, is symmetric, positive definite, and generally sparse. An important technique for reducing the time required to solve this system is *substructuring* or *matrix partitioning.* Substructuring is based on the idea of dividing a structure into pieces, each of which can then be analyzed relatively independently. As a result of this division, each point in the finite element discretization is either interior to a substructure or on a boundary between substructures. Contributions to the global stiffness matrix from connections between boundary points form the K_{bb} matrix. This paper focuses on the triangularization of a general K_{bb} matrix on a parallel machine.

Support for this research was provided in part by the National Aeronautics and Space Administration under contract number NAS1-17130 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665, and in part by the National Science Foundation under Grant No. MCS-8305693.

N85-11569#

1. Introduction

The finite element method is an important tool for determining approximate solutions to systems of differential equations arising in such diverse physical problems as structural analysis, fluid flow, and heat transport. In the finite element approach, a region of interest (e.g., an airplane wing, a cross section of a pipe, or a nuclear reactor core) is discretized into individual elements. The solution then gives displacements, vorticities, or temperatures at those points where two or more elements are joined together. A complete description of the method has appeared numerous times in the literature, cf. [7]. Much of the computational effort of the finite element process involves the solution of a system of linear equations. The coefficient matrix of this system, known as the global stiffness matrix, is symmetric, positive definite, and generally sparse. To reduce the time required to solve this system, researchers have examined many techniques, among which is *substructuring* or *matrix partitioning*, cf. [6].

Substructuring is based on the idea of dividing a structure into pieces, which can then be analyzed relatively independently. As a result of this division, each point in the discretization is either interior to a substructure or on a boundary between substructures. The application of substructuring techniques has two obvious advantages. First, if the finite element method is applied to a structure, and a portion of that structure is then changed in some way, only interior and boundary points of the substructures which have changed need to be reexamined. This is an advantage, for example, in the case of a researcher considering aircraft structure who wishes to fit a new wing to an existing model. Second, with the increased availability of parallel processors, substructuring is a natural way to decompose a finite element problem into relatively independent subproblems. A separate processor can then be applied to the solution of each subproblem.

The discretization process in the finite element method gives rise to a graph in a very natural way. Points where two or more elements join together are the vertices or *nodes* of the graph; two nodes which border on a common element are joined by an edge. The structure of the global stiffness matrix is dependent upon the ordering of the nodes in this finite element graph and is, in fact, equivalent to the structure of the graph's adjacency matrix. If the nodes corresponding to interior points are numbered first, one substructure at a time, followed by the nodes corresponding to boundary points, the global stiffness matrix will have the form of the matrix in Figure 1, where $K_{ii}^{(j)}$ represents the contributions from connections between interior points of substructure j, $K_{bi}^{(j)}$ and $K_{ib}^{(j)}$ represent the connections between interior points of substructure j and the set of boundary points, and K_{bb} represents the contributions from boundary-to-boundary connections.

$$\begin{pmatrix} K_{ii}^{(1)} & & & & K_{ib}^{(1)} \\ & K_{ii}^{(2)} & & & K_{ib}^{(2)} \\ & & \ddots & & \vdots \\ & & & K_{ii}^{(n)} & K_{ib}^{(n)} \\ K_{bi}^{(1)} & K_{bi}^{(2)} & \dots & K_{bi}^{(n)} & K_{bb} \end{pmatrix}$$

Figure 1. Global Stiffness Matrix Structure

The question of the amount of parallelism inherent in the finite element process as a whole has been examined, cf. [1]. The focus of this paper is the triangularization of a general K_{bb} matrix on a parallel machine. This portion of the problem is of considerable importance for at least two reasons. First, if any part of the structure or region of interest is changed, at least a portion of the K_{bb} matrix must be re-triangularized. Second, as the number of processors in parallel machines increases, there will be a tendency to partition structures into more substructures. As the number of substructures increases, so does the relative number of boundary points and thus the relative size of the K_{bb} matrix.

It is assumed that during the solution of the system of linear equations required by the finite element process, the global stiffness matrix has been reduced by means of Gaussian elimination to the form of the matrix in Figure 2, so that only the K_{bb} matrix is yet to be triangularized.

2. Parallel Gaussian Elimination

The ordering of the rows and columns of a given matrix (or equivalently, the ordering of the nodes in the corresponding graph) required to minimize parallel Gaussian elimination times has been studied by Leuze and Saxton [5]. Following the development of their model, assume a completely connected parallel machine with an arbitrarily large number of processors. The triangularization of a symmetric system of linear equations could be programmed on such a machine as follows. Each row is stored in a separate processor as a list of nonzero coefficients with a stack of corresponding column indices. This stack indicates to a processor what communication with other processors is required. At each step of the elimination

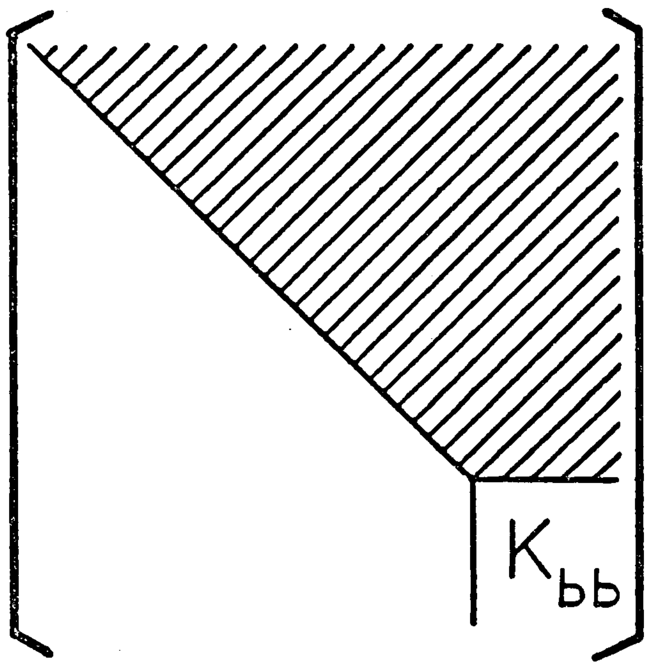


Figure 2. Partially Triangularized Matrix

process, each processor examines its stack. Suppose machine i (the processor containing row i) finds the index of machine j at the top of its stack ($i < j$). Machine i then sends its current row information to machine j and pops its stack. When machine j finds the index for machine i at the top of its stack, it uses the row information from machine i to eliminate row j's coefficient in column i and then updates its stack by merging in the stack of machine i. When all stacks are empty, the coefficient matrix is in upper triangular form.

Leuze and Saxton then developed a graph theoretic model for parallel Gaussian elimination. Their notation in [5] is followed here. Given an $n \times n$ symmetric matrix $A = \{a_{ij}\}$, define a graph $G = (V, E)$ where $V = \{r_1, r_2, \ldots, r_n\}$ (one vertex per row), and $E = \{(r_i, r_j) \,|\, a_{ij} \neq 0 \text{ and } i \neq j\}$. An *ordering* of V is a bijection $f\colon \{1, 2, \ldots, n\} \rightarrow V$. $G_f = (V, E, f)$ denotes an ordered graph. By application of a sequence of row and column interchanges, the matrix A can be made to correspond to G_f for any ordering f. For each vertex i, the *fill* of G_f for i is the set of edges $\{(j, k) \,|\, i < j < k, (i, j) \in E, (i, k) \in E, \text{ and } (j, k) \notin E\}$. The filled graph for ordered graph G_f is defined by adding the fill of G_f for each vertex i in order $i = 1, 2, \ldots, n$. The fill of G_f corresponds to additional nonzero coefficients of A introduced during the elimination process. A parallel time function $t\colon V \times V \rightarrow \mathbf{N}$ is defined for a filled ordered graph as follows:

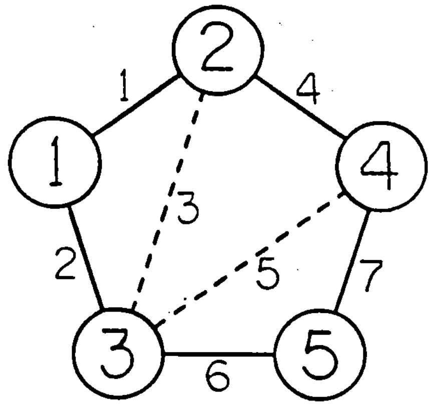
$t(1,1) = 0.$

For $i > 0$ and $j > i$,

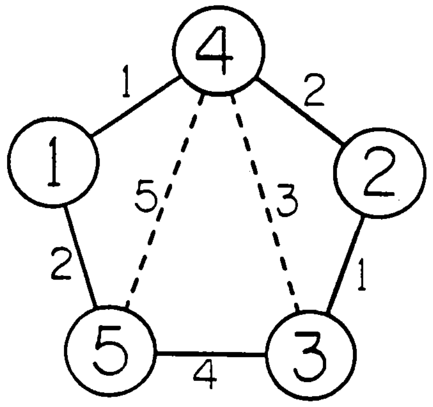
$$t(i,j) = \begin{cases} t(i,j-1), & \text{if } (i,j) \notin E; \\ \max[t(i,j-1)+1, \max\{t(k,j)+1 \mid k < i, (k,j) \in E\}], & \text{otherwise.} \end{cases}$$

For $1 < i \leq n$,

$$t(i,i) = \begin{cases} 0, & \text{if for all } k < i \quad (k,i) \notin E; \\ \max\{t(k,i) \mid k < i \text{ and} (k,i) \in E\}, & \text{otherwise.} \end{cases}$$



A



B

Figure 3. Ordered Graphs

In Figure 3, the arcs of the ordered graphs are labelled with the appropriate values from the timing function. Both orderings are optimum with respect to fill, but only ordering B is optimum with respect to the timing function.

Leuze and Saxton present no algorithms for optimum parallel orderings and, in fact, conjecture that the problem is *NP*-complete. They do, however, present two interesting results. First, it is demonstrated that the frequently used orderings which cluster nonzero elements near the diagonal are non-optimum with respect to the parallel time function. This phenomenon is easily understood through an examination of a matrix with nonzero elements tightly clustered near the diagonal, a tridiagonal matrix. In an $N \times N$ tridiagonal matrix, row i ($1 < i < N$) cannot be used in the elimination process until its first nonzero element is eliminated with information from row $i-1$. The process is forced to proceed sequentially from row 1 to row N. Second, a system of linear equations which can be solved without fill but for which any minimum parallel time ordering produces fill is presented. This system demonstrates that in the parallel model, there is not a perfect correlation between the amount of fill and the number of time steps required to solve a system, as is the case in the sequential model.

In this paper, orderings for the class of K_{bb} matrices will be examined. The special characteristics of the K_{bb} matrix will be examined in Section 3, heuristic orderings applied to K_{bb} matrices will be discussed in Section 4, and parallel Gaussian elimination times resulting from the various orderings will be presented in Section 5.

3. The Structure of the K_{bb} Matrix

When attempting to determine the K_{bb} matrix structure, the following observation about fill in a symmetric matrix is important. If, in the undirected graph corresponding to matrix A, there exists a path between node i and node j with intermediate nodes numbered less than $\min\{i,j\}$, fill will occur in matrix elements A_{ij} and A_{ji}, cf. [4], Ch. 5. Consequently, if the interior nodes of a substructure form a connected set, each boundary node of that substructure will be connected (either originally or by means of a *fill edge*) to every other boundary node of that substructure. It thus seems appropriate to divide the set of boundary nodes into subsets such that all nodes in a subset are boundary nodes to the same set of substructures.

For example, consider the division of a cube into eight substructures as indicated in Figure 4. The set of boundary nodes can be divided into subsets, such as the set of nodes on the face between substructures 1 and 2, the set of nodes on the edge between substructures 1, 2, 3, and 4, the single node which borders all eight substructures, etc. If each subset is designated by the substructures on which it borders, Table 1 contains a complete list of boundary node subsets for this example.

If it is assumed that a boundary node is originally connected only to interior nodes and other boundary nodes of the substructures it borders, then any two nodes in a given subset are indistinguishable with respect to connectivity, i.e., connected to precisely the same nodes. Based on this observation, it seems reasonable to assume that nodes within a given subset should be numbered consecutively. The question of how subsets should be ordered relative to each other then arises. This question will be examined in detail in Section 4.

The K_{bb} matrix can then be considered a partitioned matrix with partitions separating groups of rows or columns corresponding to boundary node subsets.

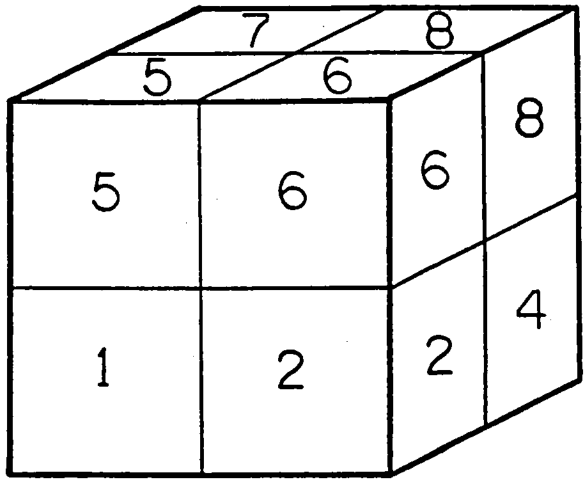


Figure 4. Substructured Cube

Table 1. Boundary Node Subsets

(1) 12	*(7)* 37	*(13)* 1234
(2) 13	*(8)* 48	*(14)* 1256
(3) 15	*(9)* 56	*(15)* 1357
(4) 24	*(10)* 57	*(16)* 2468
(5) 26	*(11)* 68	*(17)* 3478
(6) 34	*(12)* 78	*(18)* 5678
	(19) 12345678	

Since each boundary node of a substructure is connected to every other boundary node of that substructure, the blocks resulting from such a partitioning will either be composed entirely of nonzero elements or be composed entirely of zero elements. If the "row subset" and "column subset" of a block share a common substructure, that block will be nonzero; otherwise, that block will be zero. Furthermore, whenever fill occurs during the triangularization of the K_{bb} matrix, an entire block will be filled. Consequently, the structure of the K_{bb} matrix can be represented by a matrix with one row and one column per boundary node subset, together with information about the size of each subset. For example, the subdivided cube of Figure 4 with the boundary node subset ordering of Table 1 can be represented by the matrix of Figure 5.

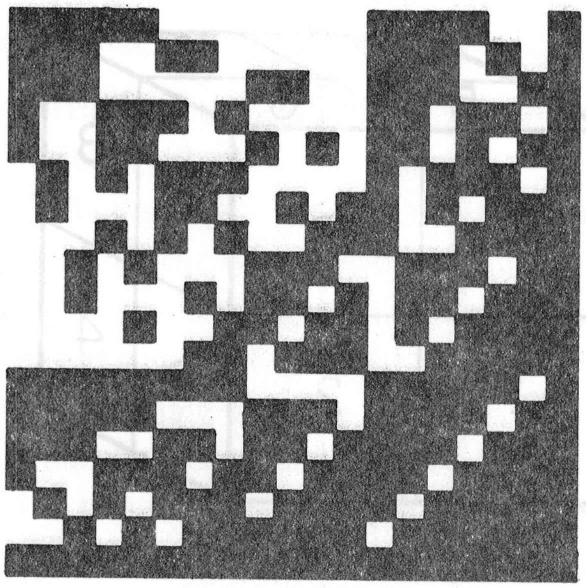


Figure 5. K_{bb} Matrix Structure

In Figure 5, a black square represents a nonzero block; a white square, a zero block. Block dimensions can be determined from boundary subset sizes. In this example, if each subcube (including boundary nodes) contains n^3 nodes, each "face" (bordering on only two subcubes) contains $(n-1)^2$ nodes, each "edge" (bordering on four subcubes) contains $n-1$ nodes, and the central "point" (bordering on all eight subcubes) consists of a single node.

4. Ordering Heuristics

Several heuristics for ordering boundary node subsets were applied to two different structures, a cube divided into 27 subcubes, each containing n^3 nodes, and an "airplane" (Figure 6) constructed from 92 square plates, each containing n^2 nodes. The substructure boundaries of the cube were divided into 98 subsets; there were 220 boundary node subsets for the airplane.

For each of the two structures, a rather arbitrary "natural" ordering (Ordering 0) of boundary node subsets was initially chosen. Subsets of the cube were divided into three categories; all "faces" were numbered first, followed by all "edges", and finally, all "points". All airplane subsets composed a single category. Within a

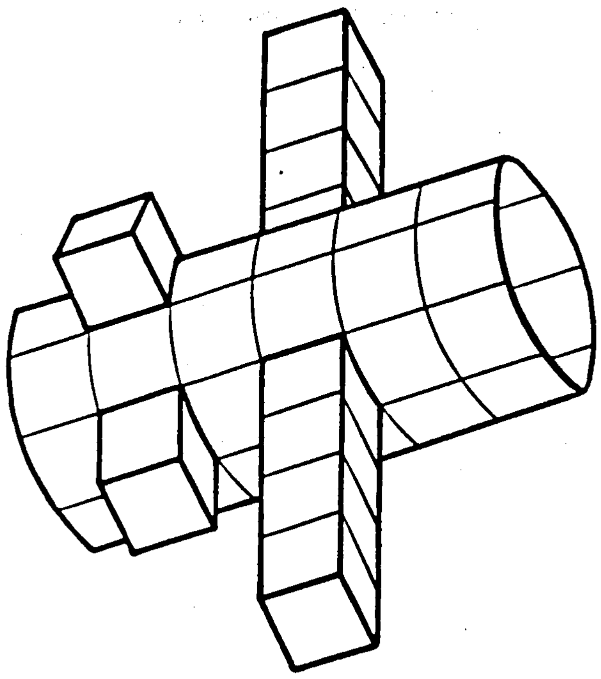


Figure 6. Structure of "Airplane"

category, boundary node subsets bordering on substructure 1 were numbered first, followed by unnumbered subsets bordering on substructure 2, etc.

The ordering heuristics described below are based on two observations of general graphs. (It should be noted, however, that the heuristics are applied not to general graphs, but rather to graphs with vertices consisting of sets of boundary nodes indistinguishable with respect to connectivity. *Block* reorderings are, therefore, performed on the corresponding matrices.) First, the total number of parallel time steps is smaller for those orderings which number first those nodes adjacent to relatively few other nodes. These graph orderings correspond to matrix orderings which number first those rows for which the *least* amount of work must be performed during the elimination process. Thus, intuitively, work can begin on intermediate rows more quickly. Second, the total number of parallel time steps is larger for those orderings which tend to number adjacent nodes consecutively. These are orderings which cluster nonzero elements of the matrix near the diagonal. Numbering of this type occurs in the Cuthill-McKee [2] and reverse Cuthill-McKee [3] orderings, which are included for comparison.

Ordering 1 (Cuthill-McKee): The first subset in the natural ordering is chosen arbitrarily as the starting subset and assigned the number 1. Then, for $i = 1, \ldots, N$ (where N is the total number of subsets), find all unnumbered subsets adjacent to subset i and number them in increasing order of degree.

Ordering 2 (Reverse Cuthill-McKee): This ordering is obtained by reversing the Cuthill-McKee ordering described above.

In the ordering descriptions that follow, two classes of variables, *DEG* and *ADJ*, are associated with each subset.

DEG variables hold the ***degree*** of a subset, the number of subsets to which that subset is adjacent. By selecting the subset with minimum *DEG* value to be ordered next, orderings which number subsets from lowest to highest degree are produced. *DEG1* contains the degree of a subset calculated ***a priori***. Its value does not change during the ordering process. Some ordering heuristics eliminate a subset when it has been numbered and pairwise connect all subsets adjacent to this subset. *DEG2* is dynamically updated to reflect the resulting changes in degree. The *DEG2* value of a subset is thus dependent upon the ordering selected and may increase as fill occurs or decrease as adjacent nodes are ordered and eliminated.

ADJ variables hold information about the set of ***numbered*** subsets to which an ***unnumbered*** subset is adjacent. By selecting the subset with minimum *ADJ* value to be ordered next, orderings which spread nonzero matrix elements, rather than cluster nonzero elements near the diagonal are produced. *ADJ1* contains the cardinality of this set of numbered subsets; *ADJ2* contains the maximum subset number from this set. *ADJ3* attempts to reflect both the set cardinality and the maximum value in the set. Whenever a subset i is numbered, for each unnumbered adjacent subset j, $ADJ3(j)$ (initially zero) gets the value one plus the maximum of $ADJ3(i)$ and $ADJ3(j)$. *ADJ4* is simply a flag which contains the value one if the set of adjacent numbered subsets is non-empty and the value zero otherwise. For the problems considered, all *ADJ4* flags were quickly set. Therefore, whenever it was detected that all flags were set, all were reset to zero.

For each of the orderings, the variables of primary and secondary importance are listed in Table 2. The unnumbered subset with minimum value for the variable of primary importance is numbered next. Ties are broken by minimizing the variable of secondary importance. Any remaining ties are broken by numbering first the subset appearing earliest in the "natural" ordering. Orderings 8, 9, and 10 are equivalent in a sense to the minimum degree algorithm [8], but correspond to matrix reordering by blocks rather than individual elements.

Table 2. Heuristic Orderings

Ordering	**Primary Variable**	**Secondary Variable**
3	*DEG1*	*ADJ1*
4	*ADJ1*	*DEG1*
5	*ADJ2*	*DEG1*
6	*ADJ3*	*DEG1*
7	*ADJ4*	*DEG1*
8	*DEG2*	—
9	*DEG2*	*ADJ1*
10	*DEG2*	*ADJ2*

5. Testing

Each of the heuristic orderings for boundary node subsets was applied both to the cube problem and to the airplane problem. For each ordering, parallel Gaussian elimination as described in [5] was applied to the resulting K_{bb} matrix. The total number of parallel time steps, the maximum number of processors used during any one time step, and the average number of processors used per time step were determined for various size problems. Values of n ranged from three to nine (each subcube contained n^3 nodes and each plate of the airplane contained n^2 nodes). It did not appear necessary to examine larger problems because of the regularity of the data. For every test case, it was possible to determine a complexity expression which exactly matched the parallel time step results for all values of n greater than four. Processor usage results were not quite as regular over the same range of values for n. Some complexity expressions appear to describe exactly the asymptotic behavior of the maximum processor values; in other cases (marked by the symbol "$\approx$"), asymptotic behavior was not reached within the test range. All maximum processor data was, however, of sufficient regularity to allow determination of the leading coefficient of the complexity expression with some degree of confidence. In addition, leading coefficients accurate to two decimal places were calculated for complexity expressions describing average processor usage. All complexity expressions are listed in Tables 3 and 4.

Actual values for parallel time steps and average number of processors for selected orderings are plotted in Figures 7, 8, 9, and 10. Data for several orderings are not plotted because the curves would lie extremely close to other curves which

Table 3. Complexity of Heuristic Orderings for Cube Problem

Ordering	Total steps	maximum processors	average processors
0	$108n^2 - 216n + 109$	$11.0n^2 + O(n)$	$7.16n^2 + O(n)$
1	$108n^2 - 216n + 109$	$18.0n^2 + O(n)$	$9.92n^2 + O(n)$
2	$108n^2 - 216n + 109$	$8.0n^2 + O(n)$	$5.52n^2 + O(n)$
3	$88n^2 - 176n + 89$	$14.5n^2 + O(n)$	$8.55n^2 + O(n)$
4	$79n^2 - 139n + 59$	$14.5n^2 + O(n)$	$8.58n^2 + O(n)$
5	$71n^2 - 120n + 46$	$12.5n^2 + O(n)$	$8.28n^2 + O(n)$
6	$71n^2 - 119n + 47$	$14.5n^2 + O(n)$	$9.20n^2 + O(n)$
7	$70n^2 - 117n + 44$	$12.5n^2 + O(n)$	$8.40n^2 + O(n)$
8	$63n^2 - 115n + 51$	$\approx 14.0n^2 + O(n)$	$7.83n^2 + O(n)$
9	$58n^2 - 102n + 43$	$\approx 14.0n^2 + O(n)$	$8.50n^2 + O(n)$
10	$63n^2 - 114n + 50$	$13.0n^2 + O(n)$	$7.83n^2 + O(n)$

Table 4. Complexity of Heuristic Orderings for Airplane Problem

Ordering	Total steps	maximum processors	average processors
0	$312n - 499$	$23.0n + O(1)$	$13.10n + O(1)$
1	$312n - 499$	$26.0n + O(1)$	$16.30n + O(1)$
2	$216n - 344$	$13.5n + O(1)$	$6.69n + O(1)$
3	$123n - 133$	$32.0n + O(1)$	$13.69n + O(1)$
4	$129n - 140$	$40.0n + O(1)$	$13.96n + O(1)$
5	$132n - 142$	$44.0n + O(1)$	$15.60n + O(1)$
6	$155n - 185$	$\approx 37.0n + O(1)$	$18.00n + O(1)$
7	$113n - 109$	$39.0n + O(1)$	$14.04n + O(1)$
8	$106n - 147$	$30.0n + O(1)$	$10.81n + O(1)$
9	$110n - 151$	$36.5n + O(1)$	$10.86n + O(1)$
10	$118n - 161$	$33.5n + O(1)$	$10.15n + O(1)$

are plotted. In all cases, the position of an unplotted curve relative to the positions of plotted curves may be determined by examination of the leading coefficient of the appropriate complexity expressions.

Experiments were conducted to compare the heuristic orderings with random

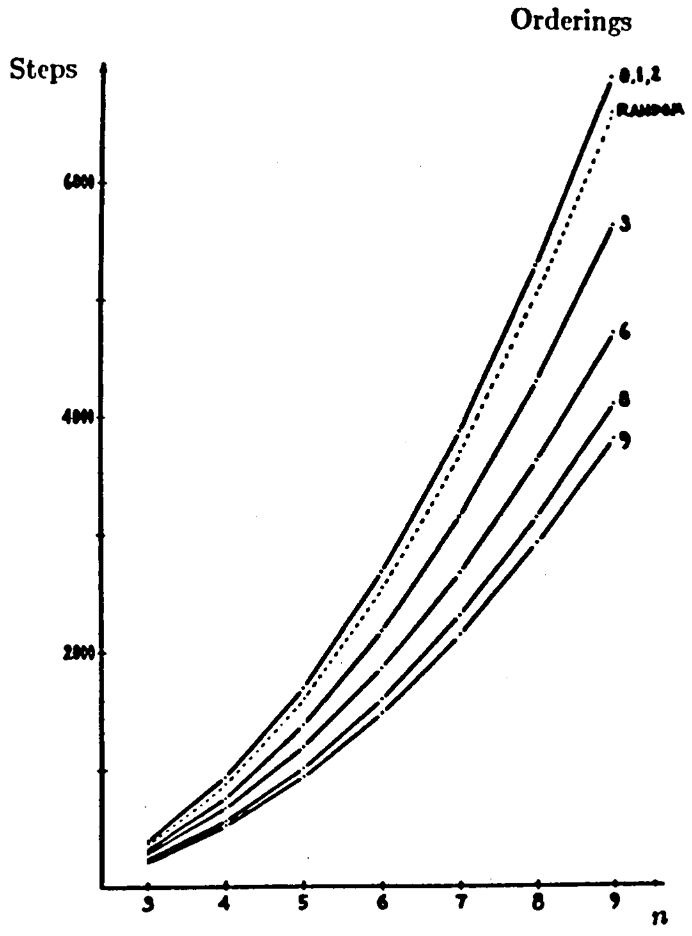


Figure 7. Parallel Time Steps for Various Orderings of the Cube Problem

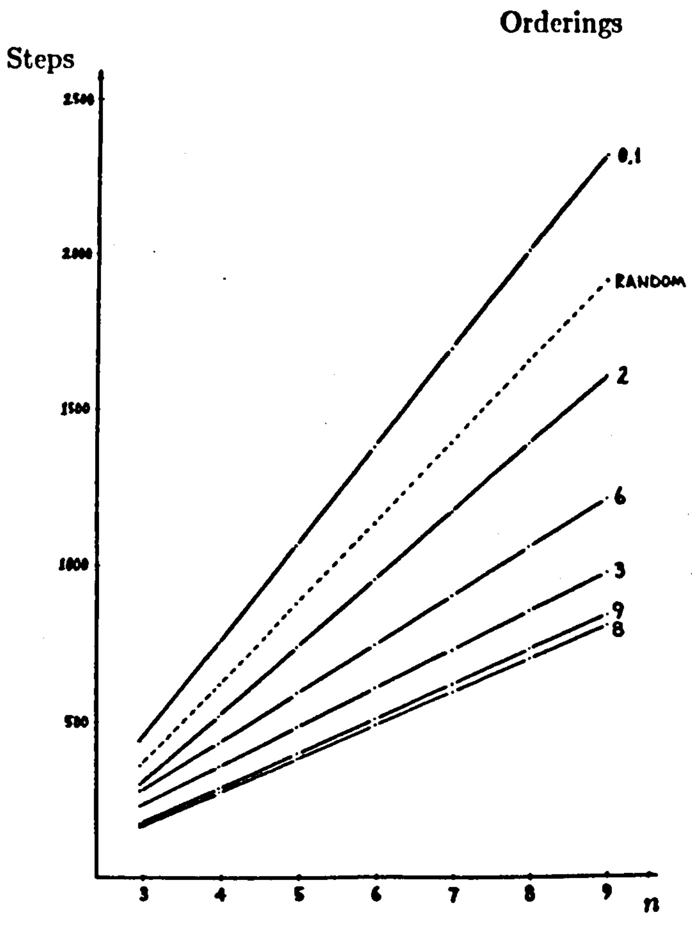


Figure 8. Parallel Time Steps for Various Orderings of the Airplane Problem

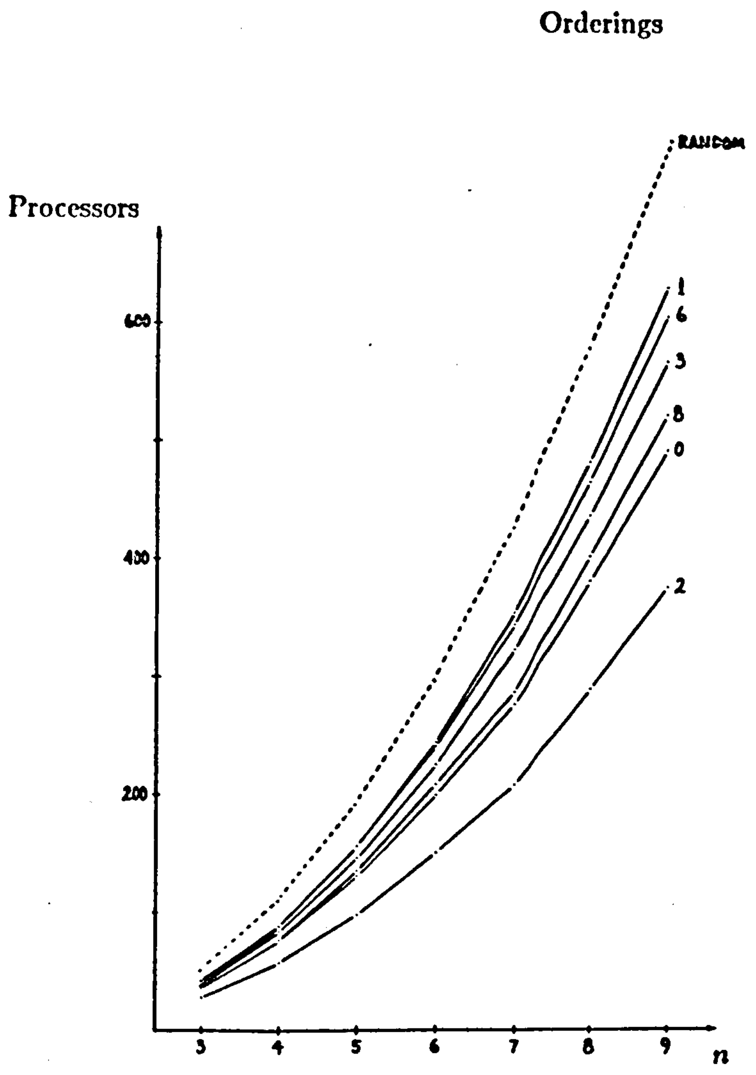


Figure 9. Average Processors for Various Orderings of the Cube Problem

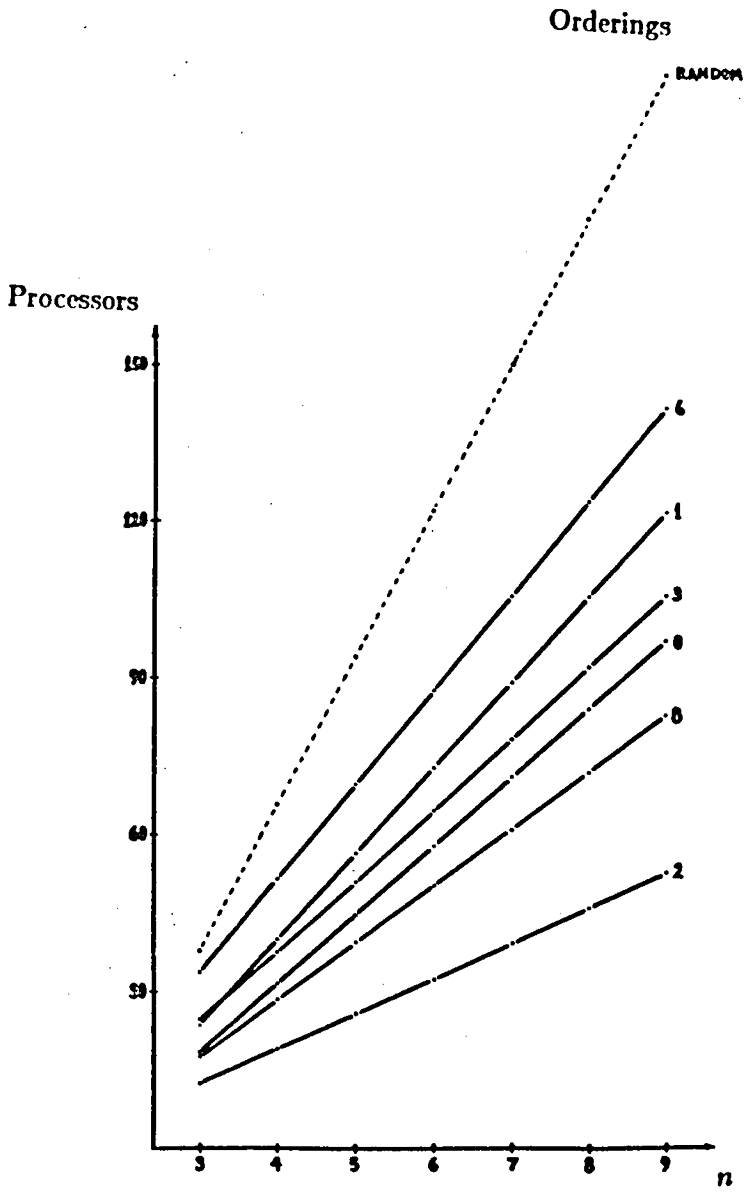


Figure 10. Average Processors for Various Orderings of the Airplane Problem

orderings. For the cube problem, 200 random orderings were examined; for the airplane problem, 100. Average results from this testing are listed in Table 5 and plotted in Figures 7, 8, 9, and 10.

Table 5. Average Complexities from Random Orderings

	Cube Problem	**Airplane Problem**
Total time steps:	$102.72n^2 + O(n)$	$256.04n + O(1)$
Maximum processors:	$21.96n^2 + O(n)$	$50.63n + O(1)$
Average processors:	$11.67n^2 + O(n)$	$27.58n + O(1)$

Figures 11 and 12 are histograms of the leading coefficients of the complexity expressions for total number of parallel time steps for the random orderings of the cube problem and airplane problem, respectively.

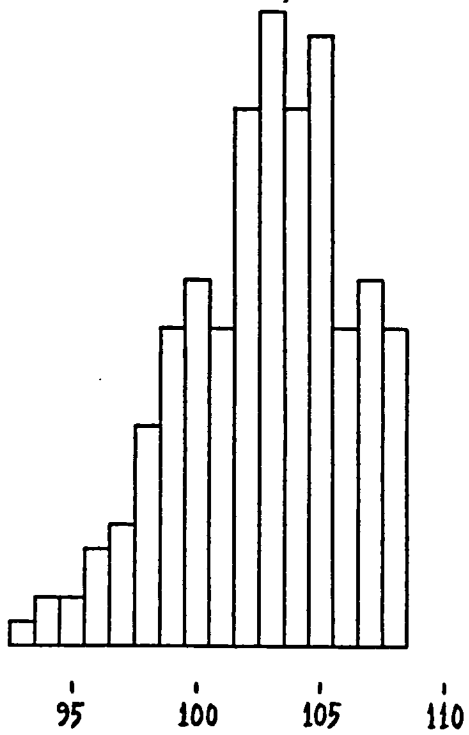


Figure 11. Leading Coefficient of the Parallel Time Complexity Expression for Random Orderings of the Cube Problem

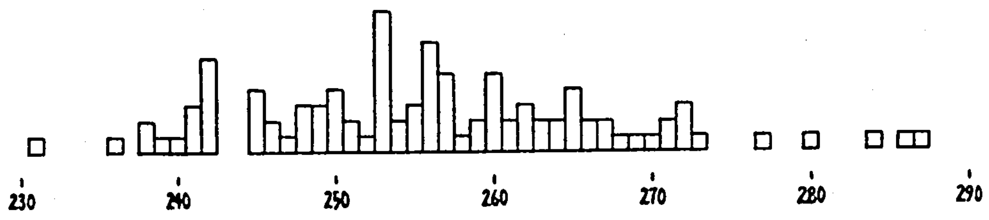


Figure 12. Leading Coefficient
of the Parallel Time Complexity Expression
for Random Orderings of the Airplane Problem

6. Conclusions

Methods to order matrices so that nonzero elements are clustered near the diagonal (such as Cuthill-McKee and reverse Cuthill-McKee) have been widely used in matrix algorithm implementations for sequential processors. There are three primary reasons for the popularity of these techniques: (a) they are easily applied to general matrices, (b) storage schemes for the reordered matrices are simple, and (c) fill is limited to the region near the diagonal. This work, however, demonstrates that these profile-reducing orderings are ill-suited for K_{bb} matrices to which parallel elimination is to be applied. The data of Figure 11 appear to fit a normal distribution truncated at the upper end. This truncation suggests that perhaps 108 is the maximum possible value for the leading coefficient of the parallel time complexity expression for the cube problem. If so, Cuthill-McKee and reverse Cuthill-McKee (which cluster nonzero elements near the diagonal) produce orderings which are among the worst with respect to total number of time steps required for parallel Gaussian elimination. For the airplane problem, the Cuthill-McKee ordering is worse than any of the 100 random orderings. In the other heuristic orderings, the function of the *ADJ* class of variables is to prevent clustering near the diagonal, thus increasing the possibility of parallel execution.

The attempt to develop heuristics for K_{bb} matrix orderings for parallel elimination appeared to be highly successful. Reordering the boundary node subsets significantly reduced the required number of parallel time steps. The best orderings for the cube required slightly more than one-half of the time steps required by the Cuthill-McKee ordering. For the airplane, time step results from the best orderings were approximately one-third of the Cuthill-McKee values. Similar results hold if comparison is made with average random orderings. Orderings which produced the best results with respect to total number of time steps were those which minimized *DEG2* as the variable of primary importance, i.e., variations of the minimum degree algorithm. Additional work is needed, however, to establish which tie-breaking rules are best. For example, the superior performance of ordering 8 over both orderings 9 and 10 in the airplane problem is unexpected, since ordering 8 simply uses ordering 0, rather than *ADJ* variables, to break ties.

Maximum processor data and average processor data appear to be closely (but not perfectly) correlated. The average number of processors used per time step is probably a more important measure of an ordering than the maximum number of processors used during any one time step. The reason for this is as follows. If the maximum required number of processors is not available, some work which could be performed at a particular time step will not be. However, that work which is performed will likely produce more work for the next time step. Consequently, the work to be performed at any given time could consist of both deferred work and newly available work. It appears, therefore, that if slightly more than the average processor requirement were available, total parallel time step values would not be adversely affected to any great extent. Further studies in which the number of processors is limited are needed to substantiate this conjecture.

Acknowledgement.

The author wishes to thank Loyce Adams and Merrell Patrick for helpful conversations which initiated this work, and Larry Dowdy and Steve Schach for their many useful comments after careful readings of preliminary drafts of this paper.

References

[1] L. M. Adams and R. Voigt, "A methodology for exploiting parallelism in the finite element process," *Proceedings of the NATO Workshop on High Speed Computations*, edited by J. Kowalik, *NATO ASI Series, Series F* **7**, Springer-Verlag, Berlin, 373–392.

[2] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," *Proc. 24th Nat. Conf. Assoc. Comput. Mach.*, ACM Publications (1969), 157–172.

[3] A. George, "Computer implementation of the finite element method," *Tech. Rept. STAN-CS-208*, Stanford University (1971).

[4] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ (1981).

[5] M. R. Leuze and L. V. Saxton, "On minimum parallel computing times for Gaussian elimination," *Congressus Numerantium* **40** (1983), 169–179.

[6] A. Noor, H. Kamel, and R. Fulton, "Substructuring techniques—status and projections," *Computers and Structures* **8** (1978), 621–632.

[7] G. Strang and G. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ (1973).

[8] W. F. Tinney, "Comments on using sparsity techniques for power system problems," *Sparse Matrix Proceedings*, IBM Research Rept. RAI 3–12–69 (1969).

1. Report No. NASA CR-172466 ICASE Report No. 84-47	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Parallel Triangularization of Substructured Finite Element Problems		5. Report Date September 1984
		6. Performing Organization Code
7. Author(s) Michael R. Leuze		8. Performing Organization Report No. 84-47
		10. Work Unit No.
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665		11. Contract or Grant No. NAS1-17130
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code 505-31-83-01

15. Supplementary Notes

Langley Technical Monitor: J. C. South, Jr.
Final Report

16. Abstract

Much of the computational effort of the finite element process involves the solution of a system of linear equations. The coefficient matrix of this system, known as the global stiffness matrix, is symmetric, positive definite, and generally sparse. An important technique for reducing the time required to solve this system is substructuring or matrix partitioning. Substructuring is based on the idea of dividing a structure into pieces, each of which can then be analyzed relatively independently. As a result of this division, each point in the finite element discretization is either interior to a substructure or on a boundary between substructures. Contributions to the global stiffness matrix from connections between boundary points form the K_{bb} matrix. This paper focuses on the triangularization of a general K_{bb} matrix on a parallel machine.

17. Key Words (Suggested by Author(s)) parallel computing finite element method substructuring	18. Distribution Statement 61 - Computer Programming & Software Unclassified - Unlimited

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