NASA Contractor Report 172219

ICASE

NASA-CR-172219
19840002763

A METHODOLOGY FOR EXPLOITING PARALLELISM
IN THE FINITE ELEMENT PROCESS

Loyce Adams

Robert G. Voigt

Contract No. NAS1-17070 and NAS1-17130
September 1983

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

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

A METHODOLOGY FOR EXPLOITING PARALLELISM IN THE FINITE ELEMENT PROCESS

Loyce Adams
Institute for Computer Applications in Science and Engineering

Robert Voigt
Institute for Computer Applications in Science and Engineering

Abstract

In most efforts to design parallel computing systems the hardware is fixed before consideration is given to the requirements of the applications that are to be executed on that hardware. This paper describes a methodology for developing a parallel system using a top down approach taking into account the requirements of the user. Substructuring, a popular technique in structural analysis, is used to illustrate this approach.

Support for this research was provided by the National Aeronautics and Space Administration under NASA Contracts No. NAS1-17070 and No. NAS1-17130 while the authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

INTRODUCTION

The finite element method is an important technique for constructing approximate solutions to boundary value problems. Very briefly,

(1) the region of interest is subdivided into elements,

(2) basis functions which span the subspace in which the approximate solution is assumed to lie are chosen,

(3) the contribution of the elements is determined by integrating the basis functions over each of the elements,

(4) the contributions of all of the elements are assembled into a single system,

$$Kx = f, \tag{1.1}$$

(5) the system is solved for the approximate solution.

A complete description of the process may be found in a number of references from the finite element literature such as Strang & Fix [1973].

Traditionally, the bulk of the computational work is contained in steps (3) and (5) and researchers have tried many techniques in order to reduce the required computational time. Some of these have involved improvements in numerical algorithms and the underlying software systems.

A particular example which will be discussed in the next section is that of substructuring or matrix partitioning, see for example Noor, Kamel and Fulton [1978]. Another example is the FEARS project begun at the University of Maryland, Zave and Rheinboldt [1979]. This latter effort attempts to improve the efficiency of the solution process for two dimensional problems by utilizing adaptive grids.

FEARS also uses another technique for improving performance that is becoming increasingly popular, namely, parallel computation. In its simplest

form, parallel computation means that relevant computations within the solution process of a single problem are performed simultaneously. In the FEARS project the parallelism is achieved by creating tasks for both the assembly and the solution process which may be executed on independent processors with a modest amount of communication, Zave and Cole [1983]. This suggests a computer organization of the multiple-instruction-multiple-data, MIMD, type in the classification of Flynn [1966].

Another MIMD computer concept under investigation for finite element analysis is the Finite Element Machine at the NASA Langley Research Center, Jordan [1978]. Ultimately, the system will contain 36 16-bit microprocessors with each processor connected to its eight nearest neighbors in a plane and connected to every other processor via a global communications bus. To date, this system has been used primarily to investigate the parallel assembly of (1.1) followed by its solution using iterative methods, Adams [1982].

Vector or pipeline computer organizations as manifested by the Control Data Corporation Cyber 200 series and the Cray Research Inc. Cray 1 series have also been used extensively for structural analysis. Much of this work has centered on just the solution of (1.1), see for example Noor and Lambiotte [1978]. In Section 3 we will point out some aspects of steps (3) and (4) that inhibit optimal utilization of the vector concept at least as it has been implemented to date.

Other architectural concepts have been proposed which may provide systems appropriate for finite element analysis of structures. In particular, Law [1982] discusses the use of systolic architectures, as introduced by Kung and Leiserson [1979], for steps (3) and (5) of the process. A more general partial differential equation solving system has been proposed by Gannon and Van Rosendale [1983], which is intended for three dimensional elliptic partial

differential equations with a solution technique based on the multigrid method.

The purpose of this paper is to provide a discussion of the parallelism available in the finite element process applied to three dimensional structural analysis problems. We use a top down methodology that allows the specification of the necessary environments required to exploit and analyze this parallelism. In particular, following a discussion of substructuring in Section 2, various types of parallelism are considered and the difficulties and opportunities of exploiting them are discussed in Section 3.

At this point many researchers would introduce a section on computer architecture and discuss the implementation details of a particular solution method. We will take a different tack.

We believe that it is essential to provide environments in which scientists can study the different issues that relate to the overall system. For example, calculating the stresses on the wing of an aircraft or investigating the inter-processor communication required by a numerical algorithm are both important in determining the eventual architecture. A methodology for developing the environments, the "virtual machine concept", is introduced in Pratt, et al., [1983] and we will rely heavily on concepts given there for the discussion in Section 4.

In Section 5, we develop a numerical analyst's virtual machine; in Section 6 we introduce an example 3-dimensional problem; and in Section 7 we discuss the example in light of the virtual machine. Finally, the benefits of this approach are discussed in Section 8.

2. SUBSTRUCTURING

The method of substructuring in structural analysis is based on dividing a large structure or region into pieces. In particular, if one is interested in displacements of a structure based on some external forces, the displacements for the interfaces of these pieces are determined first. This information may then be used to determine the values of the unknowns within each piece or substructure, see Noor, Kamel and Fulton [1978].

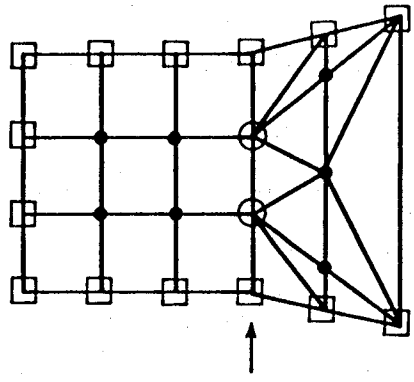


Figure 1. Example Structure

We will now make these ideas more precise by using Figure 1 as an example. The squares represent boundary points of the structure while the solid dots are interior points. The line segments indicate dependencies between points by defining the individual finite elements. Thus part 1 of the figure consists of rectangular elements that might represent plates and part 2 consists of triangular elements that might also be plates or some other element such as beams. There is a natural interface between the two parts of the structure indicated by the arrow. A structural engineer might use this interface to separate the structure into two "substructures". Note that these substructures are connected by only four points: two boundary points and two interface points indicated by circles. The circled points also become boundaries of the two substructures.

Now assuming that the elements are fixed and the basis functions are chosen, the individual element contributions to the overall problem are determined. These may be assembled into a global stiffness matrix once the ordering of the points is determined. The interior points of each substructure are numbered completely before considering the next substructure. Then the boundary and interface are numbered. A substructure, j, therefore yields a matrix of the form

$$\begin{bmatrix} K_{ii}^{(j)} & K_{ib}^{(j)} \\ K_{bi}^{(j)} & K_{bb}^{(j)} \end{bmatrix}$$

where $K_{ii}^{(j)}$ represents the contribution from the interior points, $K_{bb}^{(j)}$ represents the contribution from the boundary and interface points, which for simplicity we will refer to as boundary points, and $K_{ib}^{(j)}$ and $K_{bi}^{(j)}$ represents the dependencies or connections between interior points and boundary points.

For the entire structure one obtains the global stiffness matrix,

$$\left[\begin{array}{ccc|c} K_{ii}^{(1)} & & & K_{ib}^{(1)} \\ & K_{ii}^{(2)} & & K_{ib}^{(2)} \\ \hline K_{bi}^{(1)} & K_{bi}^{(2)} & & K_{bb} \end{array}\right] \equiv K$$

If the structure in Figure 1 is under some force and we are interested in the displacements we obtain a matrix equation (1.1) where

$$X = \left(x_i^{(1)}, x_i^{(2)}, x_b\right)^T \quad \text{and} \quad f = \left(f_i^{(1)}, f_i^{(2)}, f_b\right)^T$$

with x_i, f_i, and x_b, f_b representing the displacements and external forces at the internal and boundary points respectively.

For our purposes the most important feature of this matrix is that $K_{ii}^{(1)}$ and $K_{ii}^{(2)}$ are decoupled. This suggests a block elimination scheme for factoring the matrix K because $K_{ii}^{(1)}$ and $K_{ii}^{(2)}$ may themselves be factored in parallel.

Thus, substructuring can be viewed as a technique for decoupling the global stiffness matrix in order to introduce parallelism in the solution process. In general, we have system (1.1) with K of the form

$$\begin{bmatrix} K_{ii}^{(1)} & & & & K_{ib}^{(1)} \\ & K_{ii}^{(2)} & & & K_{ib}^{(2)} \\ & & \ddots & & \vdots \\ & & & K_{ii}^{(I)} & K_{ib}^{(I)} \\ K_{bi}^{(1)} & K_{bi}^{(2)} & \cdots & K_{bi}^{(I)} & K_{bb} \end{bmatrix} \tag{2.1}$$

where for most applications of interest K is symmetric and positive definite.

The solution process then requires the following steps:

1) For all $j = 1,\ldots,I$ form $K_{ii}^{(j)} = L_j U_j$;

2) For all $j = 1,\ldots,I$

Form $\left(K_{ii}^{(j)}\right)^{-1} K_{ib}^{(j)}$ and $\left(K_{ii}^{(j)}\right)^{-1} f_i^{(j)}$ by solving

$$L_j U_j (M_j) = K_{ib}^{(j)}$$

and

$$L_j U_j (q_j) = f_i^{(j)}$$

Form $\bar{K}_{bb}^{(j)} = K_{bb}^{(j)} - K_{bi}^{(j)} M_j$

$$\bar{f}_b^{(j)} = f_b^{(j)} - K_{bi}^{(j)} q_j$$

3) Form $\bar{K}_{bb} = \sum_j \bar{K}_{bb}^{(j)}$

$$\bar{F}_b = \sum_j F_b^{(j)}$$

where the $\sum_j$ denotes the addition of the substructure boundary contributions into the proper location in the global matrices.

4) Form $\bar{K}_{bb} = L_b U_b$ and solve $L_b U_b x_b = \bar{f}_b$

5) For all $j = 1, 2,\ldots,I$ solve $L_j U_j x_i^{(j)} = - K_{ib}^{(j)} x_b^{(j)} + f_i^{(j)}$

Algorithm 1. Solution via Substructuring

It is worth noting some distinguishing characteristics of the above process:

a) Steps 1), 2), and 5) exhibit natural parallelism requiring no cooperation among procesors.

b) The matrices $K_{ii}^{(j)}$ are banded but in general do not have the same size or structure.

c) Step 3) produces fill in the matrix $\bar{K}_{bb}$ and must be done with care in a parallel computing environment since it may change existing non zeroes in $\bar{K}_{bb}$ or introduce new non zeroes.

d) The matrix $\bar{K}_{bb}$ is banded.

Item b) points out the difficulty in using this solution technique on a vector computer if one were to try to vectorize across the substructures or $K_{ii}^{(j)}$. The diversity of the $K_{ii}^{(j)}$ suggests a collection of asynchronous processors.

The activity described in c) can require multiple modifications to the same location in K_{bb}. This would occur, for example, in the structure given in Figure 1 for all nodes on the line indicated by the arrow. In general, it occurs for all nodes on the common boundary of two or more substructures. This will be discussed further in the next section.

In Section 5 we will discuss a specific example which will provide insight into the size and form of the block matrices in (2.1). We will also compare the substructuring approach with a different ordering of the nodes which yields a single banded matrix K for which the bandwidth is small.

3. PARALLELISM IN SUBSTRUCTURING

In this section, we discuss the parallelism that is inherent in the substructuring procedure outlined in Section 2.

We begin by focusing on the creation of the matrices of (2.1). These matrices can be generated simultaneously for all substructures by independent tasks. The tasks must be of the MIMD type since the substructures normally will contain different types of elements as shown in Figure 1, and hence will require different operations during the elemental integrations. Only when each substructure is identical can we achieve parallelism by vectorizing across the substructures. Each substructure may be composed of many elements, and integrations over these elements may also be carried out asynchronously; in essence, an element may be considered as the smallest possible substructure.

It is the diversity of substructures and elements that make the assembly process unattractive for vector computers such as the Cyber 200 or Cray 1, or SIMD arrays. In both types of computer systems one needs to perform the same arithmatic operations on a group of operands or vectors. This is generally not possible either within a substructure or across substructures.

During the solution process, several operations may be performed asynchronously across the substructures. First, the matrix factorization in Step 1 of the algorithm of Section 2 will parallelize across all j. Then the formation of the new matrices indicated in Step 2 may also be done in parallel for all j:

$$K_{bi}^{(j)} \left(K_{ii}^{(j)}\right)^{-1} K_{ib}^{(j)}$$

$$K_{bi}^{(j)} \left(K_{ii}^{(j)}\right)^{-1} F_{b}^{(j)} \qquad (3.1)$$

In addition, within a given substructure, we have the flexibility to assume that (3.1) is performed in a sequential manner, if we have limited available parallelism or alternatively, we may choose to exploit the parallelism in any one or all of the following:

(1) the solution of the systems

$$K_{ii}^{(j)} M_i^{(j)} = K_{ib} \quad \text{and} \quad K_{ii}^{(j)} q_i^{(j)} = f_i^{(j)}$$

depending on the particular form of $K_{ii}^{(j)}$, followed by

(2) the parallel multiplication of the matrices

$$C = K_{bi}^{(j)} \; M_i^{(j)}, \; d = K_{bi}^{(j)} \; q_i^{(j)} \quad \text{and finally}$$

(3) the matrix and vector subtractions $K_{bb}^{(j)} - C$ and $f_b^{(j)} - d$.

Second, the formation of the $\bar{K}_{bb}$ in Step 3 of the algorithm can be done in parallel phases as follows: If all substructures are "colored" such that any two substructures that share a common interface node are different colors, it is clear that all substructures of the same color will not have contributions to the same location in the matrix K_{bb} and therefore may be assembled simultaneously without memory contention, (Berger, et. al., [1982]). The creation of K_{bb} may then be achieved in C parallel phases where C is the number of colors.

So far, the parallelism in the formation of the matrices $\bar{K}_{bb}$ and $\bar{f}_b$ for the solution for the boundary nodes has been described. We now turn to what appears to be the sequential part of the algorithm, namely the solution of $\bar{K}_{bb} \, x_b = \bar{f}_b$ in Step 4.

As with the solution of (3.1), the characteristics of the matrix $\bar{K}_{bb}$ must be known to extract all the parallelism. However, some observataions can be made now and will be made more precise for an example problem in Section 5. First, $\bar{K}_{bb}$ will be of size b × b where b is the total number of boundary nodes in the structure; furthermore, b will generally be an order of magnitude smaller than the number of interior nodes. The fill-in that occurs in $\bar{K}_{bb}$ will only be between the interface nodes of a given substructure. Hence for most structures, the matrix $\bar{K}_{bb}$ will be sparse and can be ordered as a banded matrix with as small a bandwidth, β, as possible. If $\bar{K}_{bb}$ is of size q × q, the system $\bar{K}_{bb}\ x_b = \bar{f}_b$ can be solved by factorization in q steps using β tasks. These tasks, however, do not have the completely asynchronous nature of the substructure tasks described earlier since they must cooperate during the decomposition of $\bar{K}_{bb}$. These techniques for solving this system will be described in detail in Section 5 for a particular example.

The operations in Step 5 are again completely asynchronous and may be done in parallel across the substructures. This provides the solution for the interior nodes once the solution for the boundary nodes for the substructure has been obtained. As was pointed out earlier, if we have limited parallelism, the operations may be done sequentially for a given substructure. To describe all the parallelism in the step, we must know the form of $K_{ii}^{(j)}$ and $K_{ib}^{(j)}$. This is problem dependent and also varies across the substructures.

At this point, the parallelism in the substructuring technique has been described, but many questions must be answered before an efficient overall environment for specifying and extracting this parallelism can be determined. In the next section, we give the methodology that will help us

begin to answer these questions and in Section 5 we demonstrate this methodology with an example problem.

4. THE VIRTUAL MACHINE CONCEPT

It is often tempting to take the description of the parallelism in a given solution process, like that described in Section 3, and propose a computer hardware organization to support its implementation. We believe that decisions about hardware should not be made until the user environment and the several levels of software required to effectively implement the finite element process have been carefully studied. This top-down approach to design, or virtual machine concept, is described in Pratt, et al. [1983] and will be briefly discussed below.

Each class of computer user would like to view the machine that runs his problems in different ways. To date we have considered the following four levels:

User's Virtual Machine. The perspective of a structural engineer may be that of a workstation that allows him to store the description of his structural models, to use applications packages to analyze the models, and finally to display the results.

Researcher's Virtual Machine. The numerical analyst or research user may view his machine in terms of a high-level language (like Fortran) that allows him to specify the data structures, operations and their sequences, and the parallelism in the linear algebra necessary to implement efficiently a structural engineer's application.

Systems Programmer's Virtual Machine. By specifying the tasks, their scheduling, communication between them, and the storage representation of the data, the system programmer's virtual machine that implements the high level language can be defined.

Hardware Virtual Machine. The last level of virtual machine which implements the system programmer's low level language may be the hardware itself. ("Virtual" because it may be implemented by micro programs on yet lower level hardware.)

By formally specifying the data objects, operations on these data objects, control mechanisms, and storage management techniques of each virtual machine, a detailed hardware/software design can be obtained that specifies the function of each level as well as its implementations on the next lower level. Our research uses the methods of H-graph semantics, Pratt [1981], for making this formal specification. Simulations can then be used to test the feasibility and efficiency of the overall system before commitment to hardware is made.

In the next section, we will give some insight into this approach by showing how the numerical analyst's virtual machine can be designed (not formally specified) using Algorithm 1 for the substructuring technique of the last section.

5. A NUMERICAL ANALYST'S VIRTUAL MACHINE

In this section we introduce the data objects, the operations on those data objects, and the sequence controls required to define the virtual machine

for Algorithm 1 of Section 3. The data objects are listed in Table 1. We have also included the storage required by these data objects for a particular example to be introduced in Section 7. The variables t, a, etc. in Table 1 are parameters of that example. The data object T_j is a table of integers needed for an operation explained later and the data object $\overline{K}$ is a matrix that will be discussed later.

Table 1. Data Object and Their Required Storage in Terms of Floating Point Numbers for the Example of Section 6.

DATA OBJECT	TYPE	SIZE	STORAGE
$K_{ii}^{(j)}$ $(L_j U_j)$	Banded Sym. Matrix Band: $6t^2 + t + 12$	$6t^3 \times 6t^3$	$72t^5$ (after fill)
$K_{ib}^{(j)}$	Sparse-Blocked Matrix	$6t^3 \times 6(a^3 - t^3)$	$1944t^2$
$\overline{K}_{bb}^{(j)}$	Dense Matrix	$6(a^3 - t^3) \times 6(a^3 - t^3)$	$36(a^3 - t^3)^2$
$\overline{K}_{bb}$	Banded Sym. Matrix Band: $6n^2 + 6s$	$q \times q$ $q = \Theta(n^2 d)$	$36n^4 d$ (after fill)
M_j	Dense Matrix	$6t^3 \times 6(a^3 - t^3)$	$36t^3 (a^3 - t^3)$
Q_j, $F_b^{(j)}$	Dense Vectors	$6t^3 \times 1$	$18t^3$
X_b, $\overline{F}_b$	Dense Vectors	$q \times 1$	$12(n^3 - d^3 t^3)$
T_j	Dense Integer Matrix	$6(a^3 - t^3) \times 2$	$12(a^3 - t^3)$ integers
$\overline{K}$	Banded Sym. Matrix Band: $6n^2 + 6n + 12$	$N \times N$	$36n^5$

The second step in describing the numerical analyst's virtual machine is to list the operations that must occur on these data objects. At this point, we make the assumption that the operations within a given substructure, that is a particular j in steps 1), 2), and 5), will be done sequentially. As mentioned in Section 3, more parallelism can be obtained within a given substructure by exploiting available matrix operations, but for simplicity we do not consider that here. On the other hand, we assume that factorization and solution with $\bar{K}_{bb}$ in Step 4) will be done in parallel. This will be described in detail later. The necessary operations are summarized below.

Table 2. Operations on Data Objects

OPERATION	on	DATA OBJECT	creates	DATA OBJECT
Sequential Decompose		Symmetric Banded Matrix		
Sequential Forward Solve		Lower Triangular Banded System		
Sequential Backward Solve		Upper Triangular Banded System		Dense Vector
Replace/Add (Matrix)		Dense Matrix/Banded Sym Matrix		
Replace/Add (Vector)		Dense Vector/Dense Vector		
Parallel Decompose		Symmetric Banded Matrix		
Parallel Forward Solve		Lower Triangular Banded System		Dense Vector
Parallel Backward Solve		Upper Triangular Banded System		Dense Vector
Select		Dense Vector		Dense Vector

Sequence of Operations

A control mechanism appropriate to specify the sequence of operations in Algorithm 1 is the FORALL statement which has the form

FORALL J in SET DO

BEGIN

STATEMENT 1

.

.

.

STATEMENT n

END

STATEMENT n + 1

STATEMENTS 1,...,n will be executed for each J simultaneously, to the extent possible, and STATEMENT n + 1 will not be executed until all instances of J are completed.

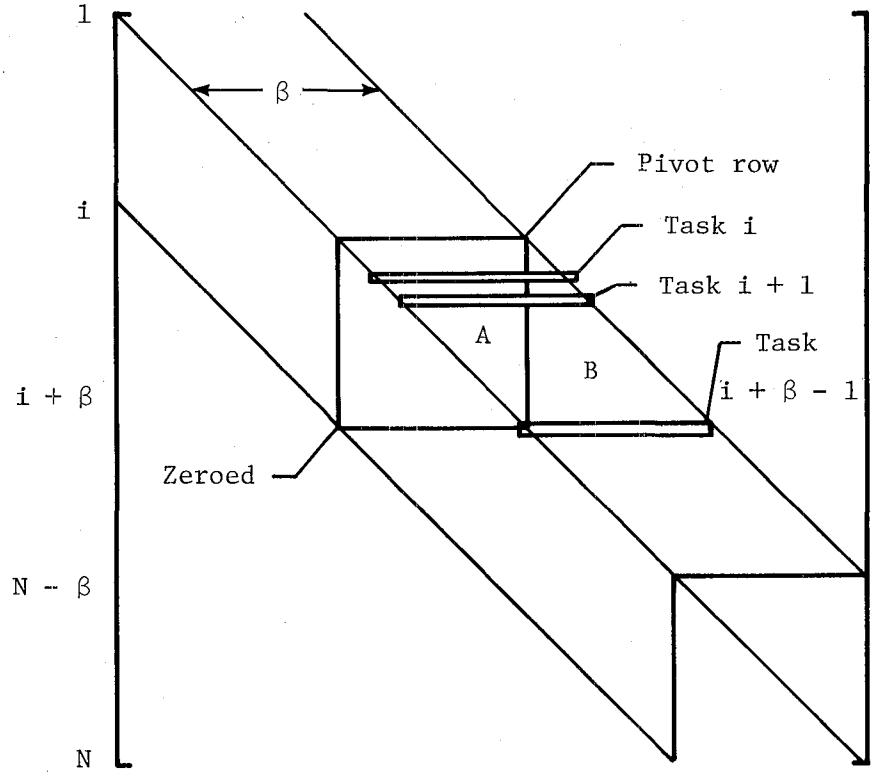
The meaning of the operations in Table 2 will now be explained. The first three are the standard operations associated with the Cholesky solution technique. The replace/add operation of any dense matrix A into a symmetric banded matrix B where both are visualized as being organized by rows and columns requires the following steps:

(1) A lookup in table T_j (of Table 1) to find the subscripts i' and j' corresponding to the subscripts i and j of a_{ij}.

(2) Adding a_{ij} to the value of b_{ij} and replacing this sum into b_{ij}. Note that if B were organized by bands, a similar lookup would be required where i' would be the band number and j' the element within the i'th band. The select operation must involve a similar table lookup, for instance, to extract a substructure's boundary values $X_b^{(j)}$ from X_b.

At this point a high level version of the numerical analyst's virtual machine has been sketched; however, it is likely that a researcher might want to study some or all of the operations in Table 2 in further detail. To demonstrate how this might be done we will focus on the "Parallel Decompose" operation. The approach we will discuss involves the generation of parallel tasks, and we will address the question of how the tasks must communicate and synchronize with each other. This begins to raise issues at the level of the system programmer's virtual machine and even at the level of the actual hardware. We will leave a detailed discussion of these levels along with a comparison of other techiques for implementing the "Parallel Decompose" for a future paper.

If the symmetric N × N matrix K has bandwidth β as shown below,



the basic algorithm consists of N-1 steps. At step i, β tasks numbered i..i + β - 1 simultaneously operate on the β rows, one task per row, directly below the pivot row i, updating the components of K in region A as shown. To perform step i, each of the β tasks must have access to the (β + 1) coefficients of the pivot row. In addition, task j must have access to row j + 1 of K and this row will be called task j's computation row. Furthermore, after step i is completed task i terminates, and task j continues to operate on the same computation row for j - i more steps at which time row j + 1 will become the pivot row and task j will terminate. This description is not correct from step N - β to step N, but this special case will be ignored here.

The following issues must be considered:

(1) How and when are the tasks created and destroyed. In particular, do the same β tasks move through the array K (or array K move through these tasks) or are new tasks created and old ones destroyed from step to step of the process?

(2) Does the creator (parent) task also perform any operations on the pivot row or is a (β+1)st task required to do this? Which approach leads to less communication?

(3) How do these tasks get access to the pivot row and their current computation row?

We will not attempt to answer all of these questions here, but rather we will discuss one approach in which a parent task is in control of the process. The parent task "owns" the array K, and initiates β subtasks with the data for their computation row and then executes the following repeatedly:

- o perform any necessary operations on the pivot row,
- o "broadcast" the pivot row to these tasks,
- o initiate a new task for the last row in the next step,
- o wait for the first subtask to complete and send back its completed computation row (the new pivot row),
- o broadcast this row to the β tasks.

The β subtasks on a given step are more passive. In general, they execute the following β times

- o receive a pivot row from the parent,
- o perform computation on their computation row using this pivot row
- o wait until the next pivot row is received.

After β pivot rows are used, the subtask sends the parent a terminate signal and also the values of its computation row which will become the new pivot row. Note that only one subtask can be sending the parent task the new pivot row at a given time. This approach requires addition of the operations given in Table 3 to our virtual machine:

Table 3. Operations for Parallel Decompose

OPERATION	EFFECT
INITIATE	Initiates a task with a Dense Vector as initial data.
SEND	Sends a Dense Vector to a particular task.
RECEIVE	Receives a Dense Vector from a particular task
BROADCAST	Sends a Dense Vector to a given set of tasks.

Now, the implementation of this approach in the virtual machine we have designed to this point is given below in Algorithm 2.

```
PROCEDURE PARENT:
     K: symmetric banded matrix (N, 6n^2 + 6n + 12)

    BEGIN
      FORALL i in 1.. β  DO
              BEGIN
                   INITIATE SUBTASK (i) WITH ROW (i + 1) OF K;
              END;
        FOR i in 1.. N - 1 DO
              BEGIN
              -- DO NECESSARY COMPUTATIONS ON ROW (i) OF K
              BROADCAST ROW (i) OF K TO ALL SUBTASKS;
              INITIATE SUBTASK (i + β) WITH ROW  (i + β + 1) of K;
              RECEIVE ROW (i + 1) OF K FROM SUBTASK (i);
              END;
END;


TASK SUBTASK (id, V);
   V:  dense vector  (β + 1);                          (*computation row*)
   PIVOT-ROW:  dense vector  (β + 1);

BEGIN
         FOR i in 1..  β DO
              BEGIN
              RECEIVE PIVOT-ROW from PARENT;
              --COMPUTE ON PIVOT-ROW AND V
              END;
         SEND V TO PARENT:
END;
```

Algroithm 2. Parallel Decompose

Algorithm 2 requires three communications per step; namely, the parent must receive the next pivot row, broadcast it to the β subtasks, and the subtasks must receive this pivot row before computation can begin. Hence, the required communication is $O(3N)$ where we are assuming these three types of communication cost the same amount and the unit of cost is that required to communicate $\beta + 1$ elements. Clearly, a system that requires time largely independent of β for this communication is desired.

This now completes the discussion of the numerical analyst's virtual machine. In the next section we introduce a three dimensional example, and in Section 7 we analyze the example in light of the virtual machine.

6. THREE DIMENSIONAL EXAMPLE

In this section we introduce a model structure with which to study the substructuring process as given in Algorithm 1. The model is an n-cube composed of d^3 individual a-cubes as shown in Figure 2.

We consider each a-cube to be a substructure composed of finite elements. The exact type of finite element(s) is not important to our analysis, nor is the exact form of the partial differential equation; instead, we focus on the connectivity of the nodes which determines the structure of the matrices and data objects of Sections 3 and 5. In particular, we assume that each node in an a-cube is connected to its eight nearest neighbor nodes in its x-y plane as well as its nine nearest neighbors in x-y planes directly above and below.

The authors are indebted to Piyush Mehrotra for developing Table 3 and Algorithm 2.

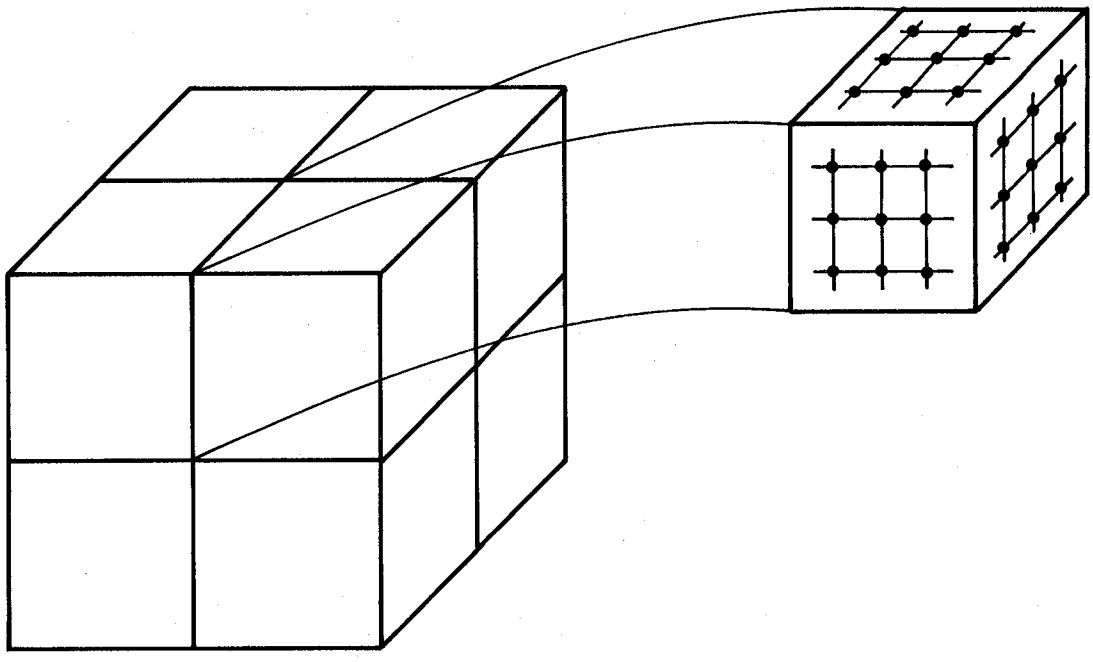


Figure 2. n-cube

To determine the structure and sizes of the data objects in Table 1, we assume there are 6 equations at each node (for example 3 displacements and 3 rotations) and set

$$N = 6n^3$$

$$t = a - 2$$

$$y = 6t^3$$

$$x = 6(a^3 - t^3)$$

$$q = 6n^3 - 6d^3t^3$$

$$s = (d+1)\left(2n - (d + 1)\right).$$

Furthermore, we assume that all interior nodes of a given a-cube are numbered left to right, front to back in a given horizontal plane with planes

considered bottom to top. After all a-cubes are numbered, the remaining boundary points are numbered the same way by considering them to be on horizontal planes. Then, each $K_{ii}^{(j)}$ is a $6t^3 \times 6t^3$ matrix with the following $t \times t$ block tridiagonal structure,

$$K_{ii}^{(j)} = \begin{array}{c} \\ 1 \\ 2 \\ \bullet \\ \bullet \\ t \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & \bullet & \bullet & t \end{array} \\ \begin{bmatrix} A & A & & & \\ A & \bullet & \bullet & & \\ & \bullet & \bullet & \bullet & \\ & & \bullet & \bullet & A \\ & & & A & A \end{bmatrix} \end{array}$$

with

$$A_{6t^2 \times 6t^2} = \begin{array}{c} \\ 1 \\ 2 \\ \bullet \\ \bullet \\ t \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & \bullet & \bullet & t \end{array} \\ \begin{bmatrix} B & B & & & \\ B & \bullet & \bullet & & \\ & \bullet & \bullet & \bullet & \\ & & \bullet & \bullet & B \\ & & & B & B \end{bmatrix} \end{array}$$

$$B_{6t \times 6t} = \begin{array}{c} \\ 1 \\ 2 \\ \bullet \\ \bullet \\ t \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & \bullet & \bullet & t \end{array} \\ \begin{bmatrix} C & C & & & \\ C & \bullet & \bullet & & \\ & \bullet & \bullet & \bullet & \\ & & \bullet & \bullet & C \\ & & & C & C \end{bmatrix} \end{array}$$

and C a 6×6 matrix.

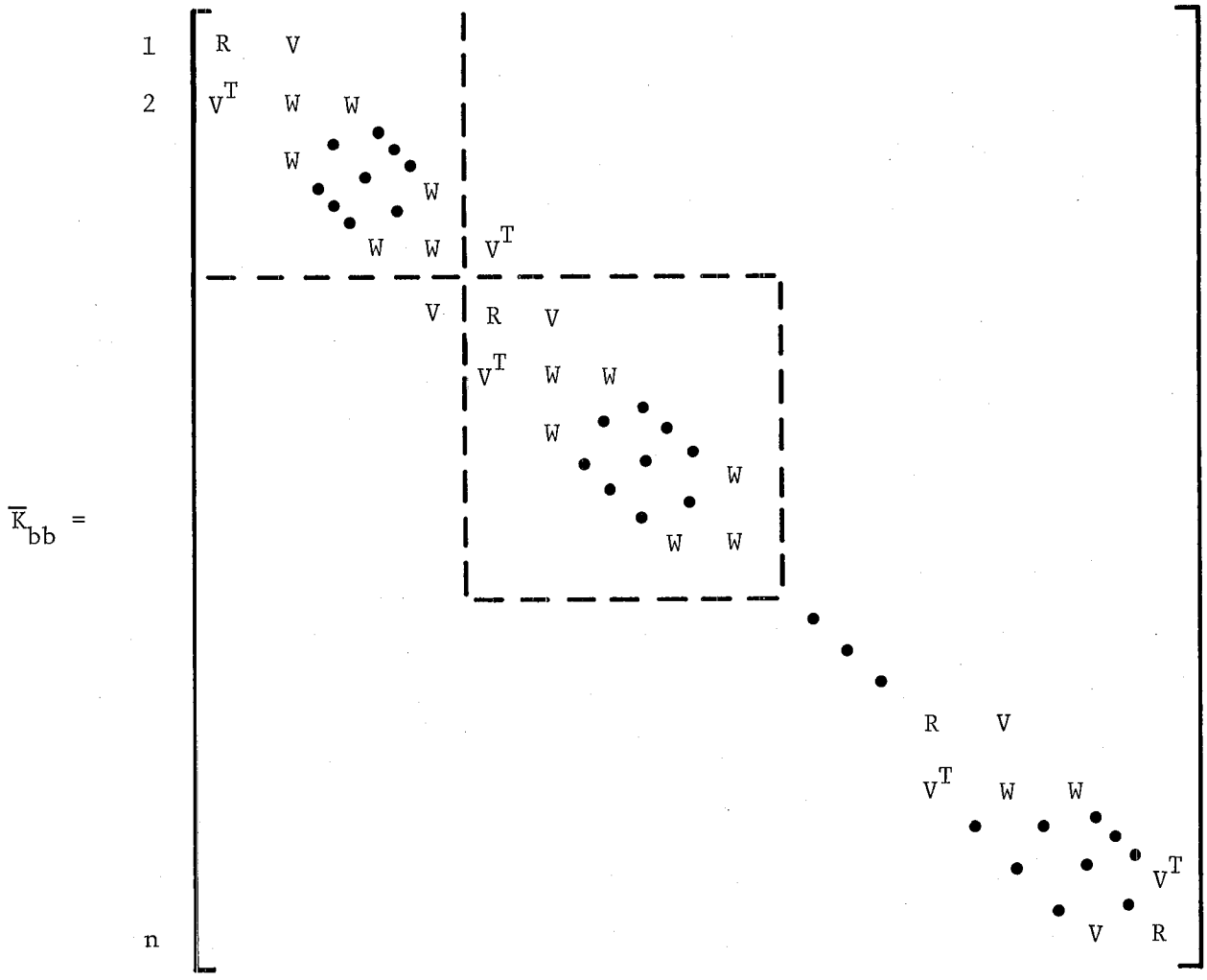
Each A matrix specifies the connectivity present in two adjacent x-y planes; each B matrix, the connectivity of two horizontal rows in a given plane; and each C matrix represents the connectivity of any two nodes in a row. During the factorization in Step 1), $K_{ii}^{(j)}$ doesn't fill outside the bandwidth of $6t^2 + 6t + 12$. Similarily, each $K_{ib}^{(j)}$ matrix is a $6t^3 \times 6(a^3 - t^3)$ matrix with the following structure,

$$K_{ib}^{(j)} = \begin{array}{c} \\ 1 \\ 2 \\ \\ \\ t \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & & & t+1 & a \end{array} \\ \begin{bmatrix} D & E & E & & & & \\ & E & E & E & & & \\ & & E & \bullet & \bullet & & \\ & & & \bullet & \bullet & \bullet & \\ & & & & \bullet & \bullet & \bullet \\ & & & & E & E & D \end{bmatrix} \end{array}$$

and represents the connectivity of the t interior planes to the a planes containing boundary points. The D matrices are of size $6t^2 \times 6a^2$ and the E matrices are of size $6t^2 \times 6(4a - 4)$. They can be broken down further to reflect the connectivity between rows in a given plane and finally between nodes. For simplicity we assume $K_{ib}^{(j)} = \left(K_{bi}^{(j)}\right)^T$. Note that in general because of varying elements and connectivity, the matrices denoted by the A's, B's, C's, D's and E's have neither the same numerical values nor the same zero, non-zero structure.

Each $K_{bb}^{(j)}$ is a $6(a^3 - t^3) \times 6(a^3 - t^3)$ block tridiagonal matrix representing the connectivity of the boundary nodes on the a planes to each other. However, during the process of eliminating $K_{bi}^{(j)}$ in Step 2 this matrix fills and for our purposes we assume that it becomes dense.

Lastly, of particular note, $\overline{K}_{bb}$ is a $q \times q$ block tridiagonal matrix of the form,



where the indicated blocks R, V, and W are of sizes $6n^2 \times 6n^2$, $6n^2 \times 6s$, and $6s \times 6s$, respectively. This matrix provides the connectivity of the boundary nodes to each other on all n planes after the fill-in from Step 3. The bandwidth of $\overline{K}_{bb}$ is $6n^2 + 6s$ and we assume storage is required within the entire band due to the fill-in caused in Step 3 and the subsequent factorization in Step 5. The storage requirements are summarized in Table 1 where for simplicity we include only the highest order term.

If we order the nodes of the original n-cube left to right, front to back, bottom to top with no substructuring the resulting matrix denoted by $\overline{K}$ in Table 1, is N x N and will be discussed later. Since $t = O(n/d)$, as n

becomes large, the maximum storage required for Algorithm 1 is either $O(72\ (\frac{n}{d})^5)$ or $O(36n^4d)$ depending on the value d. Nevertheless, this storage will be less than that for $\bar{K}$, $O(36n^5)$, whenever $d < O(n)$. Note that we do not specify how this data is to be stored or what kind of memory system is provided since these decisions should be made at a lower level of the virtual machine. Our purpose here is simply to provide the magnitude and form of the data.

7. ANALYSIS OF THE EXAMPLE

We now summarize the amount of parallel arithmetic and communication, and the required number of tasks for the substructuring technique of Algorithm 1 (using Algorithm 2 for the solution of $K_{bb}\ x_b = f_b$) and for the traditional band solver (also using Algorithm 2). We then give conditions for when one technique might be preferred over the other.

Table 4 summarizes this information where a denotes the amount of arithmetic in units of floating point multiplication/addition pairs, c denotes the number of times a bandwidth of numbers are communicated, and t represents the number of tasks.

The number of sequential operations for either method is $O(n^7)$.

Table 4. Operation Counts

METHOD	FACTOR $K_{ii}^{(j)}$	FILL-IN $\overline{K}_{bb}^{(j)}$	FACTOR $\overline{K}_{bb}$	TOTAL
Substr.	a: $108\ \frac{n^7}{d^7}$	a: $(36)^2\ \frac{n^7}{d^7}$	a: $108n^4d$	a: $(36)^2\frac{n^7}{d^7}$ if $d<n^{3/8}$
				$108\ n^4d$ otherwise
	c : 0	c : 0	c: $54n^2d$	c: $54n^2d$
	t: d^3	t: d^3	t: $6n^2+12nd$	t: $\Theta(6n^2+6s,d^3)$
Banded			a: $36n^5$	a: $36n^5$
			c: $18n^3$	c: $18n^3$
			t: $6n^2+6n+12$	t: $6n^2+6n+12$

For simplicity, we have omitted the time required for the forward and backward substitutions. If they are done in parallel, both the arithmetic and communication complexities are less than that of the factorizations. However, for a complete design these must be considered since the type of communication required is slightly different than that for factorization and would add more operations to the virtual machine.

Now, if we let

C_a = cost of one floating point multiply/add

C_c = cost of broadcasting or receiving $\beta + 1$ floating point numbers,

the total amount of work, W_s and W_b required by the substructuring and banded methods is

$$W_s = C_a(36)^2 \frac{n^7}{d^7} + C_c(54)n^2d \quad \text{if} \quad d < n^{3/8}$$

$$C_a(108)n^4d + C_c(54)n^2d \quad \text{otherwise} \tag{5.1}$$

$$W_b = C_a(36)n^5 + C_c(18)n^3$$

respectively.

First, observe from (5.1) that the substructuring technique requires less communication than the banded solver if $d < n/3$. Second, the arithmetic for substructuring is also less than that of the banded solver whenever d is in the following approximate range.

$$(36)^{1/7}n^{2/7} < d < n^{3/8} \tag{5.2}$$

or

$$n^{3/8} < d < n/3 \tag{5.3}$$

The inequality (5.2) can be satisfied only for $n > 280$, and if $n = 280$ the underlying problem contains 132 million equations. Since problems of such size are beyond serious consideration at this time, we will focus on the second inequality. Note that when d satisfies (5.3) the arithmetic and

communication work increases linearly with d; thus the optimal operation count occurs for the smallest d greater than $n^{3/8}$. Some representative values of important parameters for substructuring are compared with those for the simple banded matrix approach in Table 5.

Table 5. Size of Key Parameters for Typical Problems (Entries $\times 10^6$)

n	d	Arithmetic Parallel Floating Point Operations	Storage Floating Point Numbers	Communications (number of bandwidths)
12	Banded	9	9.0	.310
	3	7	6.2	.023
18	Banded	68	68.0	.10
	3	34	30.1	.05
24	Banded	287	287	.25
	4	143	109	.12

Inequality (5.3) can be interpreted as meaning that d must be at least $0\ (n^{2/7})$ so that the parallelism at the substructure level overcomes the sequential operations within a given substructure, but on the other hand, d can not be more than $0\ (n/3)$ so that the work of putting the substructures together (solving $\bar{K}_{bb}\ x_b = f_b$) is not too large. Note that in the limit as $d \to n$, the substructuring technique requires three times more arithmetic and three times more communication than does the band solver. This is due to the extra fill-in in the matrix $\bar{K}_{bb}$ that results from the substructure ordering.

Of course $\bar{K}_{bb}$ could be ordered in a variety of ways in order to decouple it and to introduce more parallelism. For example, the red/black ordering could be applied to the x-y boundary planes. However, an argument similar to the one above indicates that this approach will introduce more work as $d \to n$. A possible important advantage is that fewer tasks may be required. For example, the red/black ordering requires $O(24nd)$ tasks rather than the $O(6n^2 + 12nd)$ tasks reported in Table 4 for the banded ordering.

In summary, the substructuring and banded solver techniques have been programmed using the data objects, operations, and sequence control mechanisms of our virtual machine. If the implementation of this virtual machine on the other levels of virtual machine is "free" we can make the statements below:

(1) For the range of d in (5.3), the substructuring technique is more promising than the banded solver technique.

(2) For both techniques, as long as $C_a > \frac{C_c}{2n^2}$, the amount of time for arithmetic exceeds that for communication.

We realize that this analysis must be expanded to include costs of the lower levels of machine before the best method is really determined. Here, we have only attempted to give a flavor of the design process.

8. CONCLUSIONS

Many projects are under various stages of development to investigate parallel computer architectures. In almost all such efforts the basic hardware decisions are fixed at an early stage, long before the software organization and external environment have been considered and certainly before any application programs have been planned. This approach often leads to major difficulties at later stages when the software system must be made

operational on the fixed hardware. It is also difficult to measure and judge the useful computational power of such a design because of the masking effect of the layers of software that may be required to make the hardware accessible. Typically the realized computational effectiveness is far below what was expected based on hardware speeds and compromises must be made in the software development that inhibit performance on realistic problems. It is hoped that the use of the virtual machine concept will help identify the requirement of the users before the hardware is determined.

Within the framework of finite element analysis we have demonstrated how the numerical analyst can use the virtual machine to specify requirements for a solution technique based on substructuring. This led to the identification of a variety of data types and of operations on those data types. The virtual machine also provided a framework in which to study and compare parallel computation and communication. This study indicated when substructuring compared favorably to the more traditional method. In particular, inequality (5.3) showed that adding more substructures beyond a certain number resulted in more overall computation due to the added work in computing the boundary nodes. Also in the analysis of the solution for the boundary nodes, it became clear that it does not pay to introduce parallelism without understanding its ramifications. In the example studied such parallelism led to more fill-in and hence more work.

It should be noted that we are not suggesting that substructuring is the only alternative for this problem. We chose to analyze it as an example because it induces parallelism naturally and is a favorite among structural engineers. Other methods, particularly nested dissection, George and Liu [1981], with its minimum operation counts, and iterative methods with their natural parallelism should be investigated.

This paper has focused on the numerical analyst's virtual machine. We feel that the other virtual machines as discussed in Section 4 are equally important and we anticipate developing those machines in the near future.

Acknowledgement

This paper is a preliminary report on the activities of a group of researchers and would not have been possible without the contributions of all the group, which includes, in addition to the authors, Merrell Patrick of Duke University, Terrence Pratt of the University of Virginia and Piyush Mehrotra and John Van Rosendale of ICASE. The group has benefitted immeasurably from discussions with Robert Fulton and Olaf Storaasli of the Structures and Dynamics Division of the NASA Langley Research Center, Ahmed Noor of George Washington University and Tom Crockett and Judson Knott of the Finite Element Machine project at the NASA Langley Research Center.

REFERENCES

1. Adams, Loyce, [1982], "Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers," NASA Contractor Report No. 166027. Also published as a Ph.D. Thesis in the Department of Applied Mathematics and Computer Science at the University of Virginia, Charlottesville, VA in 1983.

2. Berger, Ph., P. Brouaye and J. C. Syre, [1982], "A Mesh Coloring Method for Efficient MIMD Processing in Finite Element Problems," Proceedings of the 1982 International Conference on Parallel Processing, IEEE Catalog No. 82CH 1794-7, pp. 41-46.

3. Flynn, Michael, [1966], "Very High-Speed Computing Systems," Proceedings IEEE, Vol. 54, pp. 1901-1909.

4. Gannon, Dennis and John Van Rosendale, [1983], "Parallel Architectures for Iterative Methods on Adaptive Block Structural Grids," Proceedings of the Monterey Elliptic Conference, G. Birkhoff, editor, Academic Press.

5. George, Alan and Joseph Liu, [1981], "Computer Solution of Large Sparse Positive Definite Systems," Prentice Hall, Englewood Cliffs, NJ.

6. Jordan, Harry, [1978], "A Special Purpose Architecture for Finite Element Analysis," Proceedings of the 1978 International Conference on Parallel Processing, IEEE Catalog No. 78CH1321-9C, pp. 263-266.

7. Kung, H. T. and Charles Leiserson, [1979], "Systolic Arrays (for VLSI)," Sparse Matrix proceedings 1978, I. S. Duff and G. W. Stewart, editors, Society for Industrial and Applied Mathematics, pp. 256-282.

8. Law, Kincho, [1982] "Systolic Schemes for Finite Element Methods," Department of Civil Engineering Report No. R-82-139, Carnegie-Mellon University, Pittsburg, PA.

9. Noor, Ahmed, Hussein Kamel and Robert Fulton, [1978], "Substructuring Techniques - Status and Projections," Computers and Structures, Vol. 8, pp. 621-632.

10. Noor, Ahmed and Jules Lambiotte, Jr., [1979], "Finite Element Dynamic Analysis on CDC STAR 100 Computer," Computers and Structures, Vol. 10, pp. 7-19.

11. Pratt, Terrence, [1981], "H-Graph Semantics," Department of Applied Mathematics and Computer Science Reports Nos. 81-15 and 81-16, University of Virginia, Charlottesville, VA.

12. Pratt, Terrence, Loyce Adams, Piyush Mehrotra, Merrell Patrick, John Van Rosendale and Robert Voigt, [1983], "The FEM-2 Design Method," Proceedings of the 1983 International Conference on Parallel Processing, IEEE Catalog No. 83CH1922-4, pp. 132-134.

13. Strang, G. and George Fix, [1973], An Analysis of the Finite Element Method, Prentice-Hall, Englewood Cliffs, NJ.

14. Zave, Pamela and George Cole, Jr., [1983] "A Quantitative Evaluation of the Feasibility of, and Suitable Hardware Architecture for, an Adaptive, Parallel Finite Element System," ACM Trans. Math Software, to appear.

15. Zave, Pamela and Werner Rheinboldt, [1979], "Design of an Adaptive, Parallel Finite-Element System," ACM Trans. on Math. Software vol. 5, pp. 1-17.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-172219		

4. Title and Subtitle	5. Report Date
A Methodology for Exploiting Parallelism in the Finite Element Process	September 1983
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Loyce Adams and Robert G. Voigt	83-33
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665	NAS1-17070 & NAS1-17130
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546	Contractor Report
	14. Sponsoring Agency Code

15. Supplementary Notes
Langley Technical Monitor: Robert H. Tolson
Final Report

16. Abstract

In most efforts to design parallel computing systems the hardware is fixed before consideration is given to the requirements of the applications that are to be executed on that hardware. This paper describes a methodology for developing a parallel system using a top down approach taking into account the requirements of the user. Substructuring, a popular technique in structural analysis, is used to illustrate this approach.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
parallel systems vertual machine linear algebra	62 Computer Systems 64 Numerical Analysis Unclassified-Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	37	A03

For sale by the National Technical Information Service, Springfield, Virginia 22161 NASA-Langley, 1983

End of Document