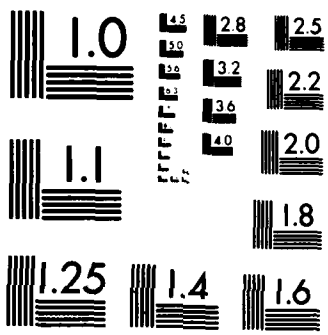
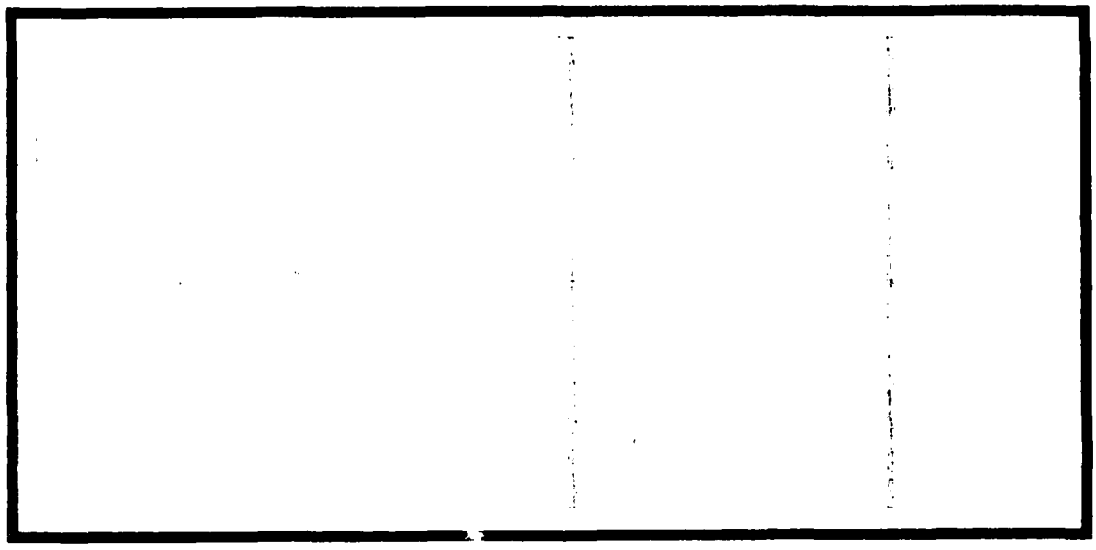
AD-A142 558 ON THE DESIGN OF A PIPELINED/SYSTOLIC FINITE ELEMENT SYSTEM(U) PITTSBURGH UNIV PA INST FOR COMPUTATIONAL MATHEMATICS AND APPLICATIONS R G MELHEM MAY 84 1/1

UNCLASSIFIED ICMA-84-71 N00014-80-C-0455 F/G 12/1 NL



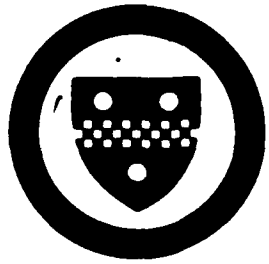
15

INSTITUTE FOR COMPUTATIONAL MATHEMATICS AND APPLICATIONS



Department of Mathematics and Statistics

University of Pittsburgh



84 05 29 046

Technical Report ICMA-84-71 May 1984

ON THE DESIGN OF A PIPELINED/SYSTOLIC FINITE ELEMENT SYSTEM*)

by

Rami G. Melhem

Department of Mathematics and Statistics
and
Department of Computer Science

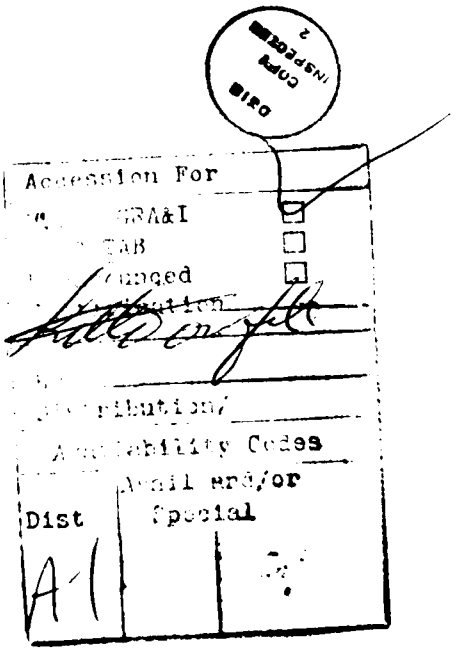
University of Pittsburgh

*)This work was in part supported by the Offic of Naval Research under Contract N0014-80-C-0455.

On the Design of a Pipelined/Systolic Finite Element System

ABSTRACT

A parallel finite element system is suggested based on the idea of pipelining the computations corresponding to the different finite elements. The systolic architecture is used extensively in the design to satisfy a regular and smooth flow of data in the pipe. Also a node numbering algorithm is developed in order to allow for the application of a frontal technique in the solution of the linear system of equations resulting from the analysis.



1. Introduction

In the past few years, many researchers have considered the use of some type of parallel processing in finite element analysis. For instance, Noor and al studied algorithms for performing the analysis on the CDC Star-100 (see e.g. [10]). Along the same line, Kamel and al [7] studied the usefulness of array processors, combined with mini computers, in finite element computations. Also, the use of multiprocessors in the solution of partial differential equations were studied (see e.g. [4]). However, different experiments showed that general multiprocessors are not expected to give satisfactory gain in the processing speed since the times for communication and data transfer dominate the running time (see e.g. [12]).

The most significant attempt in this area is the design of the finite element machine at ICASE [5]. In this project, a microprocessor is assigned to each node in the finite element grid. Each processor is connected to its eight immediate neighbors, and all the processors in the systems are connected through a global bus. This machine, however, has some limitations that result from the direct correspondence between processors and nodes. In general, it is most suitable if the interconnections between its processors follow the same pattern as the finite element grid.

A closer study of the different steps in linear finite element analysis shows that the computations may be divided into separate phases, where each phase depends only on the preceding phase. Hence the data can be transferred from phase to phase in a pipelined fashion. The computation within each phase is also well structured and mostly compute bound, which makes it a suitable candidate for systolic architectures.

In this paper, we suggest possible configurations for a finite element system that are based on the idea of pipelining the computations associated with the different elements in a finite element grid. Such system may be independent of the domain of any particular problem and of the number of elements in the grid that covers this domain. Our principal aim is to show that the pipeline/systolic idea may be a valid candidate for parallel finite element systems rather than to describe a 'ready to implement' or an optimal design for such a system, whatever may be meant by optimal.

The concept of systolic operations has been used extensively in the design to achieve a regular and smooth flow of data within each functional unit in the system. For the precise specification of the inputs and the outputs of the different units, we use the notation of the systolic model presented in [9]. The basic idea of the model is to associate with each communication link in a systolic network a data sequence comprising the data items that appeared on this link at consecutive time units. Accordingly, the computations performed by any cell in the network are expressed in terms of operators on sequences. The model is quite general and may be used for the specification and formal verification of systolic computations, as well as for their simulation. Here, we only introduce some basic definitions that will be used in the following sections.

Data sequences: A data sequence is a mapping from the set of positive integers to the set $R_\delta = R \cup \{\delta\}$, where R is the set of real number and δ is a special symbol called the "don't care" element. Each communication link in a systolic network is given a label of the form y_i and a data sequence η_i is associated with that link, that is the greek letter η corresponding to y is used. The interpretation of η_i is such that its t^{th} element, $\eta_i(t)$, is the data item that

appeared on the link y_i at time t, with $\eta_i(t) = \delta$ indicating that we do not care about the particular value at that time.

Sequence operators: We introduce informally the following sequence operators:

1) The shift operator Ω^r which inserts r δ-elements at the beginning of its operand. For example if $\eta = a_1, a_2, a_3, \ldots$, then $\Omega^2\eta = \delta, \delta, a_1, a_2, a_3, \ldots$.

2) The spread operator θ^r which inserts r δ-elements between every two successive elements of its operand. For example, $\theta\eta = a_1, \delta, a_2, \delta, a_3, \delta, \ldots$

3) The piping operator P_m^k has m operands and concatenates the first k elements of each of its operands to form one long sequence. For example, if $\xi = b_1, b_2, b_3, b_4, \ldots$, then $P_2^3(\xi, \eta) = b_1, b_2, b_3, a_1, a_2, a_3, \delta, \ldots$. The abbreviation $P_{e=1,n}^k(\eta^e)$ is used for $P_m^k(\eta^1, \ldots, \eta^m)$.

We next specify the class of problems that may be solved using our pipelined/systolic system.

2. Problem specification.

The class of boundary value problems considered here is specified by a variational formulation of the following generic form:

> Given a Hilbert space $\mathcal{W}$, a bilinear operator $\mathcal{B}$ and a corresponding functional $\mathcal{F}$ on $\mathcal{W}$, find the function $\phi \in \mathcal{W}$ such that

$$\mathcal{B}(v, \phi) = \mathcal{F}(v) \qquad \text{for all } v \in \mathcal{W} \tag{1}$$

We restrict ourselves to problems on a two dimensional domain Q, and we assume that $\mathcal{B}$ and $\mathcal{F}$ have the general forms:

$$\mathcal{B}(v,u) = \int_Q \sum_{r=0}^{2} \sum_{\ell=0}^{2} a_{r,\ell} D_r v D_\ell u \, dx \, dy + \mathcal{S}^0 \tag{2.a}$$

$$\mathcal{F}(v) = \int_Q f v \, dx \, dy + \mathcal{S}^1 \tag{2.b}$$

where $a_{r,\ell} = a_{\ell,r}$, $r,\ell = 0,1,2$ and f are problem dependent functions, D_1 and D_2 are the differential operators $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial y}$, respectively, D_0 is the identity operator and $\mathcal{S}^0$ and $\mathcal{S}^1$ are line integrals over the boundary ∂Q of Q. The forms of $\mathcal{S}^0$ and $\mathcal{S}^1$ depend on the boundary conditions and are not essential to the purpose of our discussion. We also assume that Q is covered by a finite element grid that contains n elements of the same type. Each element e, $1 \le e \le n$, is characterized by its geometric support Q^e, by k nodes on Q^e located at (x_i^e, y_i^e), $i = 1, \ldots, k$, and by some basis function associated with each node.

The nodes in the grid may be labeled by either a local or a global scheme. In a local scheme, each node in a certain element e is identified by a pair (e,i) for some i, $1 \le i \le k$. On the other hand, a global scheme assigns a unique integer j, $1 \le j \le n$, to each node, where n is the total number of nodes in the grid. The relation between the local label (e,i) of a node and its global label j is defined by some mapping $glob: [1,n] \times [1,k] \to [1,n]$, where $j = glob(e,i)$. Accordingly, we may define for each element e the boolean matrix M^e of order $k \times n$ such that $M^e(i,j) = 1$ if $glob(e,i) = j$, and $M^e(i,j) = 0$ otherwise.

Given a grid that covers Q, standard techniques may be applied to compute the finite element discretization of (1). Moreover, an isoparametric transformation may be used to map each element Q^e into a fixed element $\bar{Q}$ on some 2-dimensional space $(\bar{x}, \bar{y})$, and a numeric quadrature may be applied to evaluate the integral over $\bar{Q}$. Let q be the degree of the quadrature formula and denote

by $(\bar{x}_g, \bar{y}_g) \in \bar{Q}$ and w_g the quadrature points and weights, respectively.

If the coefficients $a_{r,\ell}$, $r,\ell=0,1,2$ and the load f are piece-wise constant functions equal to $a^e_{r,\ell}$ and f^e, respectively, on each element Q^e, it is well known that an approximate solution of (1) may be obtained as the solution of the linear system of equations

$$H u = b \tag{3}$$

where

1) u is the n-dimensional vector that contains the values of the approximate solution of (1) at the n nodes of the grid.

2) H is an $n \times n$ banded, symmetric, positive definite stiffness matrix. In order to generate H, we first compute a $k \times k$ elemental matrix H^e for each element e. The entries $H^e_{i,j}$, $i=1,\ldots,k$, $j=1,\ldots,i$ of H^e are given by

$$H^e_{i,j} = \sum_{r,\ell=0}^{2} a^e_{r,\ell} \sum_{g=1}^{q} w_g \det^e(\bar{x}_g, \bar{y}_g) \; D_r \bar{\psi}_i(\bar{x}_g, \bar{y}_g) \; D_\ell \bar{\psi}_j(\bar{x}_g, \bar{y}_g) \tag{4}$$

where $\bar{\psi}_i(\bar{x}, \bar{y})$, $i=1,\ldots,k$ are the basis functions associated with the standard element $\bar{Q}$, and $\det^e$ is the determinant of the mapping $Q^e \rightarrow \bar{Q}$. Each elemental matrix H^e that corresponds to a boundary element is then modified by the addition of a sparse matrix S^e that is computed from the term $\mathcal{G}^0$ in (2.a). The resulting elemental matrices, $\bar{H}^e = H^e + S^e$, $e=1,\ldots,m$ are finally assembled into the global matrix H according to $H = \sum_{e=1}^{m} M^{eT} \bar{H}^e M^e$

3) b is an n-dimensional global vector generated by first computing the elemental vectors b^e from

$$b_i^e = f^e \sum_{g=1}^{q} w_g \det^e(\bar{x}_g, \bar{y}_g) \bar{\psi}_i(\bar{x}_g, \bar{y}_g) \qquad i=1,\ldots,k \qquad (5)$$

Similar to the elemental matrices, each b^e corresponding to a boundary element is modified by the addition of a vector s^e and the resulting vectors, $\bar{b}^e$, $e=1,\ldots,m$ are then assembled into b according to $b = \sum_{e=1}^{m} M^{eT} \bar{b}^e$

In the previous discussion, it was assumed that ϕ is a real-valued function. It should be noted, however, that the same formulas are valid for functions with $d>1$ degrees of freedom. In this case $a_{r,1}$ and f are $d \times d$ matrices and d-dimensional vectors, respectively, and the entries of the M^e matrices are $d \times d$ unit matrices and zero matrices instead of ones and zeroes, respectively.

In the remainder of this paper, we will briefly describe the organization of a complete pipelined/systolic finite element system for the above class of problems. By its very nature, any systolic or pipelined system needs to be monitored by a host computer. In our system, the host is assumed to be a general purpose computer that contains the data base for the problem and constitutes the only means of communication between the user and the system. It is responsible for setting, initiating and feeding the systolic pipe with the appropriate data as well as collecting the outputs.

The configuration of the entire system depends on the method used for the solution of the linear system of equations (3), namely, a direct or an iterative method. We will consider systems with different types of solvers separately. However, we start by discussing a functional unit that should be included in any finite element system, namely a unit for the generation of elemental arrays.

3. The generation of elemental arrays.

The system proposed for the generation of the elemental arrays is composed of six functional units. These units, denoted in Figure 1 by N1,..., N6, are implemented using systolic components and are connected in a cascade such that the output of one unit is the input to the next unit. In order to compute the elemental arrays corresponding to a certain element e, the system should be supplied by 1) the values of the coefficients $a^e_{r,\ell}$, $r,\ell=0,1,2$ and the load f^e on element e, and 2) the coordinates of the k nodes (x^e_i, y^e_i), $i=1,\ldots,k$.

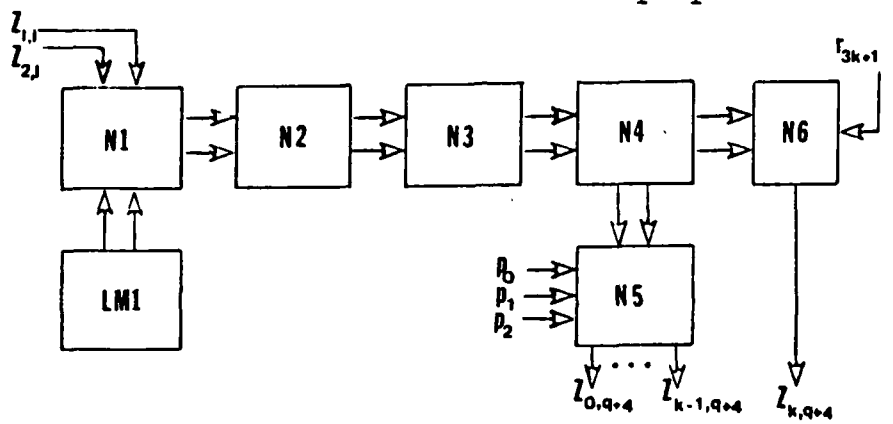


Figure 1 - Pipelined generation of the elemental arrays

In addition to the above data that are dependent on the specific element being processed, the system should also be supplied with the values of the basis functions $\bar{\psi}_i$, $i=1,\ldots,k$, and their derivatives $\partial\bar{\psi}_i/\partial\bar{x}$ and $\partial\bar{\psi}_i/\partial\bar{y}$ at the quadrature points $(\bar{x}_g, \bar{y}_g)$, $g=1,\ldots,q$. These values are independent of any particular element and hence may be preloaded into the local memory LM1 of the system and used repeatedly during the computation.

The precise specification of each computational unit and a formal verification of its operation is given in details in [3]. Here, we only present a

brief description of the function of each unit.

The unit N1 is composed of 2q 'multiply/add' systolic cells. Its function is to compute the elements of the Jacobian of the isoparametric transformation $Q^e \rightarrow \bar{Q}$. The coordinates of the k nodes of a certain element e are supplied to N1 through the input links $z_{1,1}$ and $z_{2,1}$ and the values of $\bar{\psi}_i(\bar{x}_g, \bar{y}_g)$, $\partial\bar{\psi}_i(\bar{x}_g, \bar{y}_g)/\partial\bar{x}$, and $\partial\bar{\psi}_i(\bar{x}_g, \bar{y}_g)/\partial\bar{y}$ are supplied from the local memory LM1. The exact specification of the inputs on $z_{1,1}$ and $z_{2,1}$ is

$$\zeta_{1,1} = \theta^2 \eta^e \qquad \text{and} \qquad \zeta_{2,1} = \Omega \theta^2 \xi^e \tag{5}$$

where for $t \leq k$, $\eta^e(t) = y^e_t$ and $\xi^e(t) = x^e_t$, and for $t > k$, $\eta^e(t) = \xi^e(t) = \delta$. That is, the x-coordinates of the k nodes are supplied to the system on $z_{1,1}$, starting at time one and separated from each other by two time units. Similarly, the y-coordinates are supplied on $z_{2,1}$ starting at time two and separated from each other by two time units.

The computed Jacobian and the basis functions are then passed to N2 which is also composed of 2q 'multiply/add' cells. Its task is to compute the values of $\nabla^r_i(g) = D_r\bar{\psi}_i(\bar{x}_g, \bar{y}_g)$, $r=0,1,2$, $i=1,\ldots,k$ and $g=1,\ldots,q$. These values are then passed to N3 which computes the determinant $\det^e$ of the transformation and the values of $\bar{\nabla}^r_i(g) = w_g \det^e(\bar{x}_g, \bar{y}_g) \nabla^r_i(g)$.

The unit N4 is composed of 3kq 'multiply/add' cells connected as a $q \times 3k$ rectangular array. It receives the values of $\bar{\nabla}^r_i(g)$ and $\nabla^\ell_j(g)$ from N3, and computes the integrals that appear in equation (4), namely

$$Y^{e,r,\ell}_{i,j} = \sum_{g=1}^{q} \bar{v}^{r}_{i}(g)\ v^{\ell}_{j}(g), \qquad i,j=1,\ldots,k,\ r,\ell=0,1,2, \tag{7}$$

The final step in the generation of H^e is the multiplication of each integral by the corresponding coefficient $a^e_{r,\ell}$ and the computation of the sum

$$H^e_{i,j} = \sum_{r,\ell=0}^{2} a^e_{r,\ell}\ Y^{e,r,\ell}_{i,j}$$

This step is performed by N5 which gets $Y^{e,r,\ell}_{i,j}$ from N4 and receives $a^e_{r,\ell}$ from the host on the input links p_s, $s=0,1,2$. In order to ensure that each coefficient meets the corresponding integral at the right time, the inputs should be supplied according to

$$\pi_s = \Omega^{q+3k+9}\ \alpha^e_s \tag{3}$$

where, for $t\le 3$, $\alpha^e_s(t) = a^e_{t\oplus 2s,t}$, with $\oplus$ denoting the modulo 3 addition.

The elements of the symmetric matrix H^e are produced on k output links, namely $z_{u,q+4}$, $u=0,\ldots,k-1$ (see Fig 1). More precisely, the elements of the u^{th} off diagonal of H^e appear on $z_{u,q+4}$ after $6u+q+3k+16$ time units from the initiation of the operation of the system, separated from each other by two time units. This is formally described by

$$\zeta_{u,q+4} = \Omega^{6u+q+3k+16}\ \theta^2\ \mu^e_u \qquad u=0,\ldots,k-1. \tag{9.a}$$

where

$$\mu^e_u(t) = \begin{cases} H^e_{t,t+u} & \text{if } t\le k-u \\ \delta & \text{if } t>k-u. \end{cases}$$

The function of N6 is the generation of the elemental load vectors. It

receives the values of $\overline{v}_i^r(g)$ from N4 and the load f^e from the host on the input link r_{3k+9}, and produces the elements of b^e on the output link $z_{k,q+4}$ according to

$$\zeta_{k,q+4} = \Omega^{q+9k+16}\ \theta^2\ \mu_k^e \qquad (9.b)$$

where, for $t \leq k$, $\mu_k^e(t) = b_t^e$, and for $t > k$, $\mu_k^e(t) = \delta$.

In summary, equations (9.a/b) indicate that the system completes the computation of one elemental matrix and vector in $12k+q+16$ time units, where a time unit is basically the time required to perform either a 'multiply/add' or a 'divide' operation, whichever is larger. Although this is a noticeable speed up over the serial execution of the operations involved in the computation, the system suggested here has two other important merits, namely

1) The smooth movement of data such that each data item arrives at the proper cell when it is needed. This eliminates any delay in execution due to complicated interprocess communication or slow memory fetch.

2) The ability to pipeline the computation corresponding to the different elements on the system. More specifically, if the input data for the different elements are pipelined at the rate of the data for one element every $3k$ time units, then the results will be produced at the same rate of one elemental matrix/vector every $3k$ time units. This pipelining of data may be described formally in terms of the piping operator of Section 1. Namely, if the inputs are described by

$$\zeta_{1,1} = P_{e=1,n}^{3k}(\theta^2\ \xi^e)$$

$$\zeta_{2,1} = \Omega\ P_{e=1,n}^{3k}(\theta^2\ \eta^e)$$

$$\pi_s = \Omega^{q+3k+9}\ P_{e=1,n}^{3k}(\theta^2\ \alpha_s^e) \qquad s=0,1,2$$

then it may be proved formally that the output of the system is described by

$$\zeta_{u,q+4} = \Omega^{6u+q+3k+16} \prod_{e=1,n}^{3k} (\theta^2 \mu_u^e) \qquad u=0,\ldots,k$$

where ξ^e, n^e, α_s^e and μ_u^e are as in (6), (8) and (9).

Although the term 'efficiency' is not precisely defined for systolic networks, we may roughly estimate the utilization of such networks by calculating the ratio U of the average number of cycles during which each cell in the system is doing useful work to the total number of cycles needed to complete the execution. For the six networks N1,...,N6, it may be shown that U is larger than 50%. This means that we are using, on the average, more than half of our resources, which is a relatively high utilization in parallel computation.

4. The effect of boundary conditions.

The next step in the analysis is the modification of the entries of the elemental arrays H^e and b^e, where necessary, to account for the boundary conditions. We consider here two types of modifications, namely

1) Modifications needed to force the solution to zero at some specified nodes of the grid. More specifically, in order to force the solution to zero at a certain node r, we may set $b_i^e=0$, $H_{i,j}^e=0$, $j=1,\ldots,k$, $j\neq i$, and $H_{i,i}^e=1$, for each e and i satisfying glob(e,i)=r. This is equivalent with the replacement the r^{th} equation in the linear system (3) by $u_r=0$ and thus guarantees that the solution is zero at node r.

2) Modifications that results from the so called "essential boundary conditions". As mentioned earlier, this type of conditions may be accounted for by the addition of a sparse matrix S^e and vector s^e to H^e and b^e, respectively.

Also, each entry in the modified arrays $\bar{H}^e$ and $\bar{b}^e$ should be associated with the corresponding global labels in preparation for the assembly stage. More specifically, each $\bar{H}^e_{i,j}$ should be associated with glob(e,i) and glob(e,j), and each $\bar{b}^e_i$ should be associated with glob(e,i).

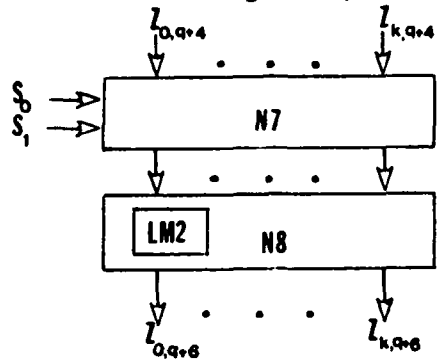


Figure 2 - Addition of the effect of boundary conditions.

The unit labeled N7 in figure 2 is responsible for the first modification. It is connected to N5 and N6 and receives from them the elements of H^e and b^e, $e=1,\ldots,n$, respectively. It also receives from the host 1) the k global labels glob(e,i), $i=1,\ldots,k$ of the nodes in each element e, and 2) for each node (e,i) a single bit that is set to one only if the solution at (e,i) is to be forced to zero. This information is supplied on the input links s_0 and s_1 according to

$$\sigma_i = \Omega^{q+3k+16} \; P^{3k}_{e=1,n}(\theta^2 \; \gamma^e_i) \qquad i=0,1$$

where the elements of γ^e_i are described, for $t \le k$, by

$$\gamma^e_1(t) = \text{glob}(e,t)$$

$$\gamma^e_0(t) = \begin{cases} 1 & \text{if the solution at node } (e,t) \text{ is forced to zero} \\ 0 & \text{otherwise} \end{cases}$$

The unit N8 is responsible for the addition of the correction arrays S^e and s^e to the elemental matrices H^e and b^e. However, because most of the

arrays S^e and s^e e=1,...,m are zero arrays, and if non zero, they contain only few non zero entries, the addition of special hardware for the computation of these non zero entries cannot be justified. More appropriately, these few entries may be computed by the host and preloaded into a local memory (LM2), and some logic may be built into N8 to retrieve these entries when the corresponding entries of the elemental arrays are received from N7. Detailed implementations for N7 and N8 are given in [3].

5. Systems that employ direct solvers

In Figure 3, we show a block diagram of a complete system that uses an LU decomposition for solving (3). It consists of the host and four functional units. The unit labeled GEN is the generator of the elemental arrays as described in some detail in sections 3 and 4. The output of GEN is then directed into the unit labeled ASSEMB. Its function is to assemble the global arrays H and b. The third unit, FACT, receives H and b from ASSEMB and simultaneously performs the LU factorization and produces the solution y of Ly=b. Finally, the unit BACK solves the triangular system Uu=y by back substitution.

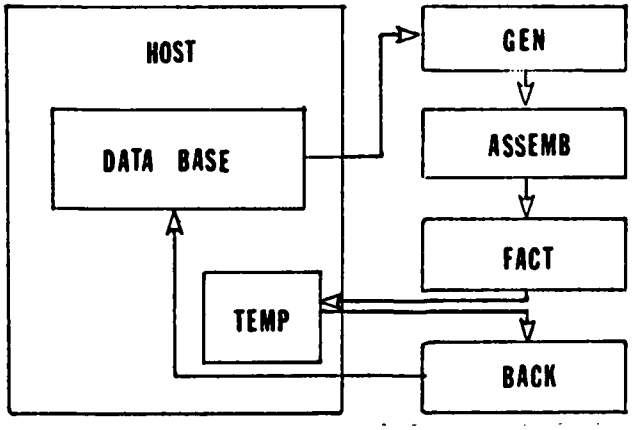


Figure 3 - A system that employs a direct solver.

In order to simplify the discussion, we consider only the assembly of H and we denote by $\bar{h}_i^e$ the i^{th} row of the symmetric matrix $\bar{H}^e$ and by h_i the i^{th}

row of the symmetric global matrix H. We also note that ASSEMB receives from GEN the elemental matrices $\bar{H}^1,\ldots,\bar{H}^m$, pipelined in the given order. The rows within each matrix $\bar{H}^e$ are pipelined in the order $\bar{h}^e_1,\ldots,\bar{h}^e_k$. This order is determined by the local labels given to the nodes of the grid.

The labels e=1,...,m given to the elements of the grid are of particular interest to us. If the element labels satisfy the property that any element e, $1\le e\le m$, contains at least one node that does not belong to any element 1,...,e-1, then we call such a labeling scheme a proper element labeling. The importance of proper element labeling will be apparent later.

Each element $\bar{H}^e_{i,j}$ is received by ASSEMB accompanied by the global labels glob(e,i) and glob(e,j) that specify the position at which it should be accumulated in H. More precisely, assuming that a band storage scheme is used for H, ASSEMB accumulates $\bar{H}^e_{i,j}$ in row glob(e,i) of H, and in the off-diagonal position |glob(e,i) - glob(e,j)|.

The assembled rows of H are then passed to FACT that proceeds with the LU factorization. Here, a frontal technique is naturally used to achieve two important goals: 1) To allow FACT and ASSEMB to execute in parallel, 2) To minimize the storage requirement of ASSEMB.

In order to be more specific, we introduce some terminology. During the assembly process, a row h_i is said to be active from the moment of the appearance of a row $\bar{h}^e_j$ with glob(e,j)=i, that is from the time when its assembly actually starts. On the other hand, h_i is called a ready row immediately after

the accumulation of the last row $\bar{h}^r_\ell$ with glob(r,ℓ)=i, that is after its assembly has been completed. In other words, a certain row of H is partially assembled when it is active but not yet ready. Once ready, a row may be passed from ASSEMB to FACT and its storage in ASSEMB may be released.

However, in all the known parallel schemes for the direct solution of Hu=b, the rows of H have to be processed in a sequential order, which means that the ready rows of H should be produced by ASSEMB in sequential order. Fortunately, we may satisfy this restriction by assigning appropriately global labels to the nodes. The following node numbering algorithm takes this restriction into consideration.

ALG1:

Given a proper element labeling for the finite elements and a corresponding local numbering of the nodes, obtain the global numbering by giving the nodes sequential numbers in the following order:

1) FOR j=1,...,k DO glob(1,j) = j

2) FOR e=2,...,m DO

  FOR i=1,...,k DO IF node (e,i) is already numbered THEN skip

  ELSE increase j by one and set glob(e,i)=j.

Now, assume that the nodes are numbered by ALG1 and that the bandwidth of the matrix resulting from this particular numbering is 2β+1. Then it may be proved [3] that, during the assembly process, the rows of H become active in a purely sequential order. Moreover, whenever a certain row i of H is active then the rows up to i-β-1 are ready and may be processed by the solver.

Definition: If, at a specific time during the assembly of H, a certain row i, $1 \le i \le n$ is active, then the rows 1,...,i-β-1 are called B_ready rows of H.

From the above discussion, it follows that B_ready rows are also ready rows, and that the rows of H become B_ready in a purely sequential order. However, a given row may be ready before it becomes B_ready. Being pessimistic, we will pass only B_ready rows from ASSEMB to FACT, except of course the last rows n-B,...,n that may be passed to FACT only after the assembly of H is completed.

In addition to satisfying the two goals stated earlier, the above rule for the interaction between ASSEMB and FACT allows ASSEMB to determine automatically the instant at which a row is ready to be passed to FACT. This eliminates the preprocessing step that is usually needed in frontal techniques to determine the instant at which a certain row is ready. More precisely, ASSEMB may keep a flag "BMAX" that indicates that any row h_i, $i \le BMAX$ is B_ready (and hence ready). This flag should be updated whenever a row $\bar{h}_j^e$ with $glob(e,j)-B-1>BMAX$ is received.

We now return to ALG1. Although this algorithm provides a good numbering scheme from the point of view of processing the assembly and the solution processes in parallel, we still have to ensure that it does not result in a large bandwidth B. For this we note that ALG1 is a two step algorithm; First, the elements are labeled, and then the nodes within the elements are numbered. To our knowledge, Fenves and Law [3] were the first to suggest a two step numbering scheme. They reported experimental results which show that if the Cuthill-Mckee [2] algorithm is used to number the elements in a two step algorithm, then the bandwidth of the resulting matrix is comparable with the best known algorithm for minimizing the bandwidth. Here, in the application of the Cuthill-Mckee algorithm, two elements are considered neighbors if they share a common boundary. We examined many strange shapes of meshes and in only rare cases did the application of the Cuthill-Mckee algorithm for numbering the

elements result in a non proper element labeling. Moreover, in all these rare cases, a proper labeling was easily obtained by changing the starting element. The existence and construction of a proper element labeling scheme for a given mesh is still a question that needs to be answered.

Possible implementations for ASSEMB and FACT.

It is clear that ASSEMB has to handle large amount of data at high rates, which necessitates the distribution of its task on a number of processors, each being responsible for the assembly of the elements in one or more diagonals of H. We consider here the extreme case where we have B+1 processor/memory units PM0,...,PMB with PMw responsible for the assembly of the w^{th} off-diagonal of H. A communication network, COMM, is needed to distribute the data received by ASSEMB to the appropriate processor. Namely, when an element $\bar{H}^e_{i,j}$ is received, accompanied with glob(e,i) and glob(e,j), COMM should direct these data to PMw, where w=|glob(e,i)-glob(e,j)|. It may be implemented as a binary tree network, where each node is a switch that uses the appropriate bit of w to decide whether to pass the information to its left or right successor.

In the implementation suggested in [8], the process executed by each PMw is driven by two interrupts, namely input and output interrupts. The input interrupt takes place when new data (typically $\bar{H}^e_{i,j}$ and glob(e,i)) are received from COMM. As a result, $\bar{H}^e_{i,j}$ is accumulated in the position v=glob(e,i) of the diagonal stored in PMw and the flag BMAX is set to max{BMAX,v-B-1}. Upon reception of all the elemental arrays, BMAX is set to n to indicate that any row in H may then be passed to FACT. On the other hand, the output interrupt is of a lower priority and takes place when the output port connected to FACT is ready

to receive new data. A counter "consumed_w" may be used to keep track of the next element that should be produced. However, if this element is in a row that is not yet B_ready then PMw would have to wait until consumed_w ≤ BMAX and then pass the element to FACT.

One difficulty arises from the distribution of the assembly process on the B+1 PM's, namely that upon receipt of a certain row $\bar{h}_i^e$, the entries of this row are distributed to the few PM's that are responsible for their accumulation. Hence, only these few PM's will detect the arrival of $\bar{h}_i^e$ and update accordingly their copy of the variable BMAX. The copies of BMAX in the other PM's will not be updated unless we provide for some sort of interprocess communication. In order to solve this problem, we may store BMAX in a global location shared by all PM's. However, since PM0 receives the diagonal element of every row arriving at ASSEMB, it seems natural to have only PM0 update BMAX. Another solution is to use a message passing technique where the updated value of BMAX is passed from PM0 to PM1 to PM2, ..., and so on.

With the above implementation of ASSEMB, FACT should have enough computing power to process the ready rows of H at the high rate at which they may be produced by ASSEMB. This power may be obtained from a very high speed array processor that may become available in the future as a result of advances in VLSI and optical communication technologies. However, with the current technology, the most suitable candidates for the implementation of FACT are systolic arrays.

Many systolic networks have been suggested in the literature for the LU decomposition of positive definite banded matrices (e.g. [5]), and specifically for symmetric matrices [1, 8]. They all require that the elements of the matrix be supplied diagonal-wise, which is compatible with the above structure of

ASSEMB. However, any implementation of FACT may not be synchronized by a global clock, namely because the rate at which ASSEMB produces the B_ready rows of H is not constant due to the nature of the assembly process. Hence, a self-timed technique [11] should be used for the interaction between ASSEMB and FACT and for the synchronization of the operation of FACT.

In order to study the utilization of FACT in this self-timed environment, we define r_c to be the rate at which FACT would consume the ready rows of H if they were always available when needed. This rate for the networks described in [1, 6, 8] is one row of H every 3 time units, that is $r_c = 1/3$. We also define $r_p(t)$ to be the rate at which ASSEMB produces the B_ready rows of H at any particular time t. For the implementations of GEN and ASSEMB discussed here, it may be shown that, at any time t, $r_p(t) \leq 1/3$ row/time unit. This means that the B_ready rows of H will be consumed by FACT as soon as they are produced by ASSEMB. This result has the following implications:

1) The storage in each PEw should be large enough to store only B+1 elements of the w^{th} off-diagonal of H at a time.

2) The generation and assembly of H form the bottle neck of the entire system. This is a clear indication that we should not concentrate our effort on finding faster parallel solvers for linear systems and neglect the problem of generating H and b fast enough to feed these fast solvers.

3) Although FACT is self-timed, we may accurately estimate the time at which it will complete its task. For example, if the systolic network in [3] is used for FACT, then it may be shown that the decomposition will be completed one time unit after FACT receives the last row of H. Moreover, the last B rows of H are made available to FACT just after ASSEMB receives its last input at time $3km+9k+q+16$. Since FACT is able to consume these B rows in 33 time units, we

may conclude that it will terminate its execution at time $3kn+9k+3B+q+17 \approx 3(kn+B)$.

Finally, we discuss the last unit in Figure 3. This unit, BACK, performs the back substitution step. Although its task is simple, BACK cannot start its computation before the last row of L and the last element of y are available. Hence, a temporary storage, TEMP, must be provided for storing the elements of L and y upon their generation until FACT terminates its execution. Note that the systolic network for back substitution described in [6] may be used for BACK. This needs 2n time units to execute, and hence the entire analysis will be completed in approximately $3(kn+B)+2n$ time units, which is a considerable speed up over the time for serially executing the $O(n^2)$ operations involved in the analysis.

Although the system described above profits from all apparent concurrencies in the analysis, it has a serious disadvantage, namely the dependency of its architecture on the bandwidth B of the matrix H. In order to be able to use a system designed for a certain bandwidth B for a problem with a larger bandwidth, we should be able to partition the computation appropriately to allow its execution on the existing hardware. This partitioning seems to be nontrivial for systolic LU decomposition networks due to their complex communication patterns. More research is needed on systolic or alternate architectures for the direct solutions of linear systems if we desire to have a system that is independent of the bandwidth B.

6. Systems that employ iterative solvers.

Direct solution schemes for the linear system (3) do not take advantage of the fact that H is highly sparse, thus missing a potential for savings in both

storage and execution time. For this reason, it is sometimes beneficial to use iterative schemes for the solution of (3) despite their obvious disadvantages, namely, the absence of a good criterion for chosing the initial point and the possible divergence or slow convergence of the iteration.

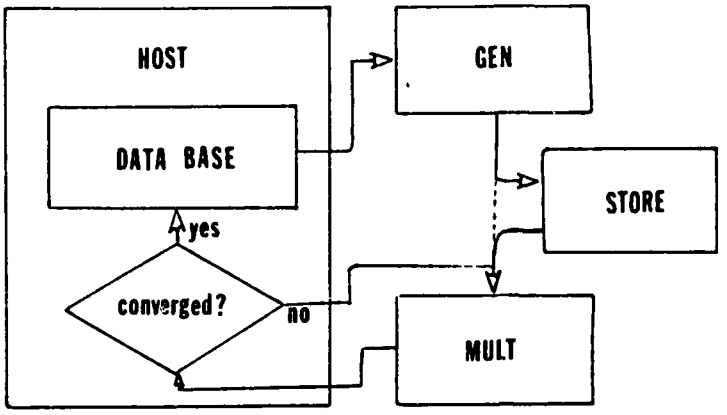


Figure 4 - A system that employs an iterative solver.

Many iterative schemes exist for the solution of Hu=b. Here, we restrict ourselves to those schemes that involves the matrix H only in the computation of its product with a certain vector. This product may be formed using the unassembled elemental arrays, thereby eliminating the need for the irregular assembly stage. More specifically, the product of H with any vector p may be computed from

$$Hp = \sum_{e=1}^{m} M^{eT} \bar{H}^{e} M^{e} p = \sum_{e=1}^{m} M^{eT} \bar{H}^{e} p^{e}.$$

This multiplication scheme is attractive in our case because the partial products $\bar{H}^{e} p^{e}$ for $e=1,\ldots,n$ may be pipelined at the same rate at which the arrays $\bar{H}^{e}$ are generated.

In Figure 4, we show a block diagram of a systolic system that employs iterative solvers. It is composed of the host and two systolic functional units; namely GEN for the generation of the elemental arrays and MULT for the matrix/vector multiplication. In this system, the host is more involved in the

computation than in the system that employs direct solvers. In fact, the host is a general purpose computer that executes a sequential finite element program and uses the systolic units GEN and MULT as high speed devices to perform some compute-bound operations in the program.

The block labeled STORE in Figure 4 is a storage device used to store the elemental arrays in order to use them in successive iteration steps. However, if the speed of STORE is such that it cannot provide MULT with the elemental arrays at the required high rate, then STORE may be eliminated from the system, and GEN may be used to regenerate the elemental arrays in each step of the iteration. This idea of regenerating the elemental arrays is more attractive if a multigrid technique is used for the solution of Hu=b. In that case, the regeneration of the elemental arrays becomes an essential operation. Note that the architectures of GEN and MULT neither depend on the specific grid that cover the domain of the problem nor on the bandwidth of the resulting matrix H. Hence, the matrices corresponding to the different grids may be generated from GEN without any system reconfiguration.

Finally, we note that only limited speed up may be obtained by using the pipelined idea with iterative solvers. This is namely due to the fact that successive steps in commonly used iterative solvers cannot be pipelined. For example, in the conjugate gradient method, a new step cannot be initiated before the termination of a dot product that depends on the previous iterate. It seems that the success of a pipelined/iterative finite element system is largely dependent on the availability of an iterative solution scheme in which successive steps may be pipelined.

7. Conclusion.

Pipelining is a straight forward approach for processing a certain computation in parallel on some hardware that is not dependent on the size of the problem to be solved. In this paper, we discussed pipelined solutions to linear finite element problems. In order to maximize the benefits from a pipelined system, the execution time of the different stages in the pipe should be approximately equal. This may be accomplished by including in each functional unit in the pipe an amount of hardware proportional to the computation performed by that unit. Unfortunately, in the case of complex computations as that involved in finite element analysis, this means that the architecture of each unit may depend on some parameters of the given problem. For instance, the LU factorization unit depends on the bandwidth B of the stiffness matrix and the array generation units depend on the number of nodes k in each finite element. Consequently, a system designed for a certain B and k cannot be used for problems with $B'>B$ or $k'>k$.

These restrictions on B and k result from the use of systolic networks with very simple types of cells for the majority of the functional units in the pipe. This has the advantage of achieving a smooth and regular flow of data in the system, thus increasing its execution speed. However, a system that is independent of B and k may be obtained if we allow for more flexible cells and, accordingly, partition the computation within each unit such that each cell is assigned to a larger share of the computation. The execution time of the different units in the modified system should be kept approximately equal. We did not discuss such a decoposition in this paper.

The common problem of communicating data at a high rate to and from systolic arrays are not present in the system suggested here. Namely, data are

supplied to the first unit in the pipe at a rate of 3k+10 items every 3k time units, and results are collected from the last unit at an average rate of B/3 item every 3k time units. The communication between the other pipe-units do not require any intervention from the host that controls the entire operation.

Finally, we hope that this work will lead to more research for the further exploration of the idea of pipelining finite element computations.

Acknowledgment

I would like to take the opportunity to thank my thesis advisor, professor Werner C. Rheinboldt, for his excellent guidance and advice during the course of this research. This work was supported in part by the Office of Naval Research under contract number N00014-80-C-0455.

References

1. R. R. Brent and F. T. Luk, "Computing the Cholesky Factorization using a Systolic Architecture," Technical Report 32-521 (Sept. 1982). Dept. of Computer Science, Cornell University

2. E. Cuthill and J. McKee, "Reducing the Bandwidth of Sparse Symmetric Matrices," Proc. of ACM national conference, New York, pp.157-172 (1969).

3. S. Fenves and K. Law, "A Two Step Approach to Finite Element Ordering," Report number R-81-130 (Aug. 1981). Department of Civil Engineering. Carnegie-Mellon University.

4. E. F. Gehringer, A. K. Jones, and Z. Z. Segall, "The Cm* Testbed," Computer Vol. 15(10), pp.40-53 (Oct. 1982).

5. H. F. Jordan, "A Multiprocessor System for Finite Element Structural Analysis," Computer and Structures Vol. 10, pp.21-29 (1979).

6. H. T. Kung and C. E. Leiserson, "Systolic Arrays for VLSI," in Introduction to VLSI Systems (1980). ed by Mead C. and Conway L., Addison_Wesley, Reading Mass.

7. J. Mai-tan, N. Sarigul, O. Palusinski, and H. Kanel, "Balanced Array Processor Configuration for Finite Element Analysis," N00014-75-C-0837 (Feb. 1982). University of Arizona

3. R. G. Melhem, An Abstract Systolic Model and its Application to the Design of Finite Element Systems., Ph.D. Thesis, Department of Computer Science, University of Pittsburgh, Dec. 1983.

9. R. G. Melhem and W. C. Rheinboldt, "A Mathematical Model for the Verification of Systolic Networks," SIAM J. on Computing. Vol. 13(3), (To appear) (Aug. 1984).

10. A. K. Noor and J. J. Lambiotte, "Finite Element Dynamic Analysis on CDC Star-100 Computer," Computer and Structures Vol. 10, pp.7-19 (1979).

11. C. L. Seitz, "System Timing," in Introduction to VLSI Systems. (1980). ed. C. Mead and L. Conway. Addison-Wesley, Reading, Mass.

12. P. Zave and G. Cole, "A Quantitative Evaluation of the Feasibility of a suitable Hardware Architecture for an Adaptive Finite Element System," ACM Trans. on Mathematical Software (Sept. 1983).

END

FILMED

8