Computers & Structures Vol. 20, No. 1–3, pp 55–65, 1985
Printed in the U.S A

0045–7949/85 $3.00 + .00
Pergamon Press Ltd.

TRENDS IN NUMERICAL ANALYSIS AND PARALLEL ALGORITHMS

SYSTOLIC ARRAYS FOR FINITE ELEMENT ANALYSIS

KINCHO H. LAW
Department of Civil Engineering, Rensselaer Polytechnic Institute, Troy, NY 12181, U.S.A.

Abstract—Systolic machines or arrays are devices attached to a conventional computer to perform special-purpose functions with extremely high speed. In this paper, systolic assembly-factorization networks for the stiffness method and the natural factor approach of finite element analysis are presented. The approach is to chain together two or more systolic arrays with compatible data flow patterns such that regularity of data flow can be maintained. By allowing concurrent operations, the time complexity required to execute the assembly-factorization networks is $O(N)$, where N is the number of nodal variables or degrees of freedom in the finite element mesh.

INTRODUCTION

New computer architectures have been, and continue to be, undergoing progressive development. One motivation for these architectures is to overcome the limitations of present single instruction–single data (SISD) computers by using devices suitable for parallelism and pipelining. These new parallel architectures have revolutionized the design of computer algorithms and promise to have significant influence on algorithms for finite element computations.

Systolic machines or systolic arrays are devices attached to a conventional computer to perform special-purpose functions with extremely high speed [1]. Such machines generally have regular structure in that the data are circulated in a network of simple and primitive processors and in a regular fashion. Each processor regularly pumps data in and out and performs some simple and short computations, so regular flow of data can be maintained in the network. The purpose of these high-performance, special-purpose devices is to meet specific application requirements or to off-load computations that are especially demanding for general-purpose computers.

The algorithms that match with systolic machines are termed systolic algorithms. Systolic algorithms have been successfully designed for many compute-bound problems, such as matrix multiplication, solution of system of equations and eigenvalue problems [2–5]. Although most algorithms designed for systolic architectures are generally simple single-purpose functions, the systolic concept can be integrated to more complex problems such as finite element methods.

The use of finite element methods involves very large amounts of computations. The algorithms based on these methods can be modularized into several phases, and they lend themselves naturally to parallel computation. For instance, individual element stiffness matrices can be generated in parallel independent of each other. The assembly, factorization and solution phases are often performed simultaneously, and they are implemented on most present computers as sequential operations. However, the high sparsity and regular pattern of the global stiffness matrix makes the assembly-factorization phase a potential candidate for high-level intelligent parallel computation.

Investigation on systolic algorithms for assembly and factorization has recently been initiated by the author [6]. The approach has been to properly chain together two or more compatible systolic arrays such that regularity of data flow can be maintained. This "chaining" approach is discussed in this paper. Assembly-factorization networks for the stiffness method and the natural factor approach of finite element analysis are presented. By allowing concurrent operations, the time complexity required to execute these networks is $O(N)$, where N is the number of nodal variables, or degrees of freedom, in the finite element mesh.

NOTATION FOR BANDED MATRICES

The systolic algorithms presented here are designed for banded matrix computations. In this section the definitions for the bandwidth of a matrix are described.

Let **A** be an n by m matrix. For the ith row of **A**, define

$$\sigma_{i*}(\mathbf{A}) = \max \{j \mid a_{ij} \neq 0\}$$

$$\beta_{i*}(\mathbf{A}) = \begin{cases} \sigma_{i*}(\mathbf{A}) - i + 1 & \text{if } \sigma_{i*}(\mathbf{A}) \geq i \\ 0 & \text{if } \sigma_{i*}(\mathbf{A}) < i \end{cases}$$

and

$$b_r(\mathbf{A}) = \max \{\beta_{i*}(\mathbf{A})\}.$$

The number $\sigma_{i*}(\mathbf{A})$ denotes the column subscript of the last nonzero entry in the ith row of matrix **A**. $\beta_{i*}(\mathbf{A})$ is the local row bandwidth of the ith row of **A** and is equal to the number of superdiagonal entries between the "diagonal" entry a_{ii} and the last nonzero entry of that row. $b_r(\mathbf{A})$ is called the row

bandwidth of **A**. Similarly, for the jth column of **A**, we define

$$\sigma_{*j}(\mathbf{A}) = \max \{i \mid a_{ij} \neq 0\}$$

$$\beta_{*j}(\mathbf{A}) = \begin{cases} \sigma_{*j}(\mathbf{A}) - j + 1 & \text{if } \sigma_{*j}(\mathbf{A}) \geq j \\ 0 & \text{if } \sigma_{*j}(\mathbf{A}) < j \end{cases}$$

and

$$b_c(\mathbf{A}) = \max \{\beta_{*j}(\mathbf{A})\}.$$

$\sigma_{*j}(\mathbf{A})$ denotes the row subscript of the last nonzero entry in the jth column of matrix **A**. $\beta_{*j}(\mathbf{A})$ is the local column bandwidth of the jth column of **A** and is equal to the number of subdiagonal entries between the diagonal entry a_{jj} and the last nonzero entry of that column. $b_c(\mathbf{A})$ is called the column bandwidth of **A**. Finally, the bandwidth $b(\mathbf{A})$ of matrix **A** is defined as

$$b(\mathbf{A}) = b_r(\mathbf{A}) + b_c(\mathbf{A}) - 1.$$

The definitions of σ_{i*}, β_{i*}, σ_{*j}, β_{*j}, $b_r(\mathbf{A})$, $b_c(\mathbf{A})$ and $b(\mathbf{A})$ are illustrated in Fig. 1a.

The bandwidth definitions also apply to partitioned matrices. Let an n by m matrix **A** be partitioned into N by M block entries, each of which is a p by q submatrix, i.e. $N = n/p$ and $M = m/q$. The block row bandwidth $B_r(\mathbf{A})$ denotes the maximum number of nonzero superdiagonal block entries in the rows. Similarly, the block column bandwidth $B_c(\mathbf{A})$ denotes the maximum number of nonzero subdiagonal block entries in the columns. The total block bandwidth $B(\mathbf{A})$ of matrix **A** is defined as

$$B(\mathbf{A}) = B_r(\mathbf{A}) + B_c(\mathbf{A}) - 1.$$

Matrix A

```
X X
X X X
X X X X
X X X X
  X X X
    X X
```

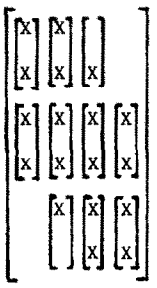
i	σ_{i*}	β_{i*}
1	2	2
2	3	2
3	4	2
4	4	1
5	4	0
6	4	0

j	σ_{*j}	β_{*j}
1	4	4
2	5	4
3	6	4
4	6	3

$b(\mathbf{A}) = 5$ $\quad b_r(\mathbf{A}) = 2$ $\quad b_c(\mathbf{A}) = 4$

(a)

Matrix A



i	σ_{i*}	β_{i*}
1	3	3
2	4	3
3	4	2

j	σ_{*j}	β_{*j}
1	2	2
2	3	2
3	3	1
4	3	0

$p = 2,\ q = 1$

$B(\mathbf{A}) = 4$ $\quad B_r(\mathbf{A}) = 3$ $\quad B_c(\mathbf{A}) = 2$

(b)

Fig. 1. Definitions of bandwidths for rectangular matrices: (a) bandwidths of a rectangular matrix; (b) block bandwidths of a rectangular matrix.

The definitions of $B_r(\mathbf{A})$, $B_c(\mathbf{A})$ and $B(\mathbf{A})$ are illustrated in Figure 1b.

SYSTOLIC VLSI ARCHITECTURES

The concept of systolic architectures is to map high-level computations into hardware structures. As noted by Kung [1], the basic principle of the systolic approach is to replace a single processing element, or processor, by an array of processors with built-in hardware instruction so that a high computational throughput can be achieved without having to increase the memory bandwidth. Systolic algorithms are designed to allow concurrent computations using the idea of pipelining and parallelism. Once a data item is brought out from memory, it is to be used effectively at each processor while passing through the array. Kung [1] summarized the four basic criteria for designing systolic algorithms:

1. The design should involve only a few types of simple processors.
2. Data and control flows should be simple and regular.
3. Algorithms should support high degrees of pipelining and parallelism.
4. Each input data item should be used in a multiplicative and effective manner.

Based on these criteria, many systolic matrix algorithms have been developed. The following sections examine a few selective systolic matrix algorithms, which are useful in finite element computations.

Banded matrix multiplication

The matrix product **C** of two matrices **A** and **B** can be computed using the following recurrence formula:

$$c_{ij}^{(1)} = 0 \tag{1a}$$

$$c_{ij}^{(k+1)} = c_{ij}^{(k)} + a_{ik}b_{kj} \tag{1b}$$

$$c_{ij} = c_{ij}^{(n+1)}. \tag{1c}$$

A systolic array for the multiplication of two square banded matrices has been developed by Kung and Leiserson [2]. The array consists of a network of hexconnected processors, each of which is an inner product step processor performing the operation given in eqn (1b).

The hex-connected network given by Kung and Leiserson can be extended immediately for multiplication of rectangular matrices. As shown in Fig. 2a, the coefficients of matrices **A**, **B** and **C** enter the network synchronously in the upper left, upper right and bottom of the array, respectively. Each coefficient c_{ij} is initially set to zero. As the coefficient passes through the network, it accumulates the inner product $a_{ik}b_{kj}$. Following the results from Kung and Leiserson [2], the complexity of the array for multiplying an n by m banded rectangular matrix **A** and an m by n banded rectangular matrix **B** can be summarized as

No. of processors: $b(\mathbf{A}) * b(\mathbf{B})$

Units of time: $3n + b$,

where $b = \min\{b(\mathbf{A}), b(\mathbf{B})\}$. Each time unit corresponds to the time required for the inner product operation and for passing the data to the neighboring processors. It should be noted that it is possible to use about one-third of the $b(\mathbf{A})b(\mathbf{B})$ processors in the network since only one out of every three consecutive processors is active at a given time.

An array for banded matrix multiplication was recently proposed by Huang and Abraham [4] and is shown in Fig. 2b. The array consists of a hex-connected network of inner product step processor. The coefficients of matrices **A**, **B** and **C** enter the array from the bottom left, bottom right and bottom, respectively. The complexity of this array for the multiplication of two banded matrices **A** of size n by m and **B** of size m by n is

No. of processors: $b(\mathbf{A}) * b(\mathbf{B})$

Units of time: $n + b$,

where $\mathbf{b} = \min\{b(\mathbf{A}), b(\mathbf{B})\}$.

LU *factorization*

Given a symmetric positive definite matrix **A**, the lower triangular matrix factor **L** and the upper triangular matrix factor **U** can be computed by the following recurrence formula:

$$a_{ij}^{(1)} = a_{ij} \tag{2a}$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} + \ell_{ik}(-u_{kj}) \tag{2b}$$

$$\ell_{ik} = \begin{cases} 0 & \text{if } i < k \\ 1 & \text{if } i = k \\ a_{ik}^{(k)} u_{kk}^{-1} & \text{if } i > k \end{cases} \tag{2c}$$

$$u_{kj} = \begin{cases} 0 & \text{if } k > j \\ a_{kj}^{(k)} & \text{if } k \le j, \end{cases} \tag{2d}$$

where a_{ij}, ℓ_{ik} and u_{kj} are coefficients of **A**, **L** and **U**, respectively. A hex-connected systolic array for **LU** factorization of a banded matrix has been developed by Kung and Leiserson [2]. This array is shown in Fig. 3.

The **LU** factorization array is made up of a large number of inner product step processors and a reciprocal processor [2]. The processors below the upper boundaries are inner product step processors and are hex-connected in the same manner as the matrix multiplication network. The processor at the top, denoted by a circle, is a reciprocal processor computing the reciprocal of its input and passing the result southwest and the input northward unchanged. The other processors on the upper bound-

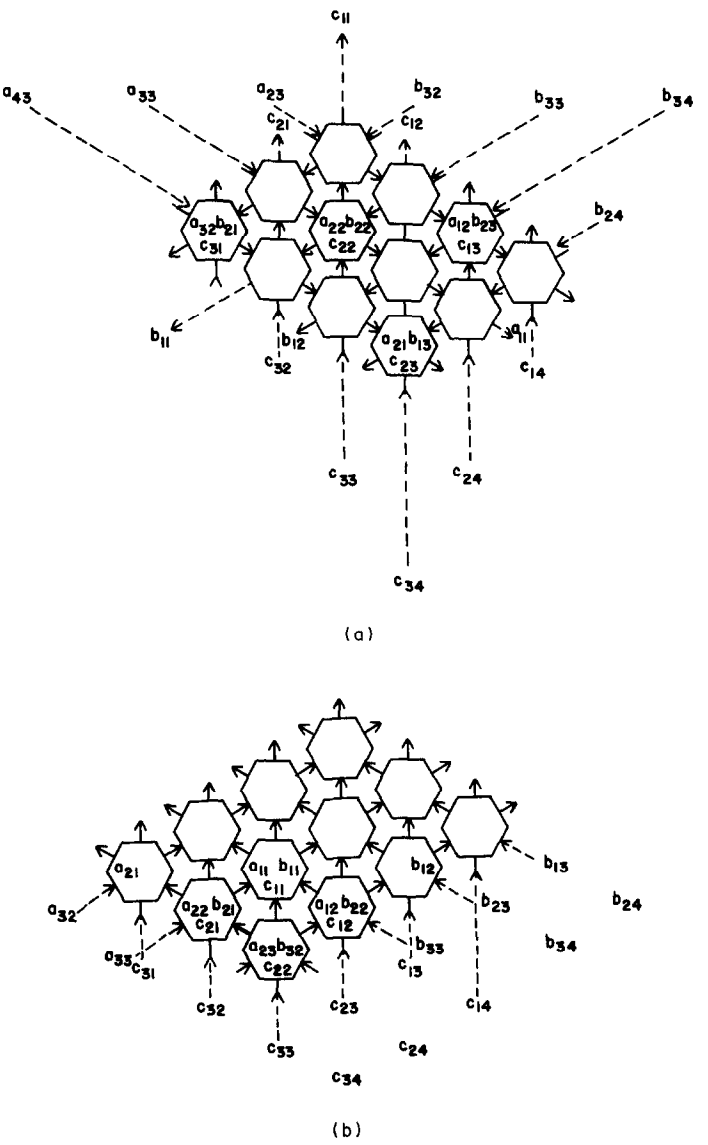


Fig. 2. Systolic arrays for banded matrix multiplication arrays [2, 4].

aries are also inner product step processors, but their orientations are changed: the ones on the upper left boundary are rotated 120° clockwise; the ones on the upper right boundary are rotated 120° counterclockwise.

For an n by n symmetric banded matrix **A**, the complexity of **LU** factorization using the hex-connected array can be summarized as [2]

$$\text{No. of processors:} \quad \frac{b(\mathbf{A})^2}{4}$$

$$\text{Units of time:} \quad 3n + \frac{b(\mathbf{A})}{2}.$$

Similar to the matrix multiplication array shown in Fig. 2a, each processor operates every third time step; the number of processors can, therefore, be reduced to $b(\mathbf{A})^2/12$.

QR *factorization of banded matrices*

For a rectangular matrix **A**, the Given's transformation decomposes the matrix into a matrix product **QR**, where **Q** and **R** are an orthogonal matrix and an upper triangular matrix, respectively. The following discussion is primarily concerned with computing the upper triangular matrix **R**.

Given two row vectors

$$\begin{aligned} &0, \ldots, 0, r_i, \ldots, r_{i+1}, \ldots, r_k, \ldots, r_m \\ &\text{and} \\ &0, \ldots, 0, a_i, \ldots, a_{i+1}, \ldots, a_k, \ldots, a_m, \end{aligned} \tag{3a}$$

the Given's transformation rotates the two vectors and replaces them by [10]

$$\begin{aligned} &0, \ldots, 0, r_i', r_{i+1}', \ldots, r_k', \ldots, r_m' \\ &0, \ldots, 0, 0, a_{i+1}', \ldots, a_k', \ldots, a_m', \end{aligned} \tag{3b}$$

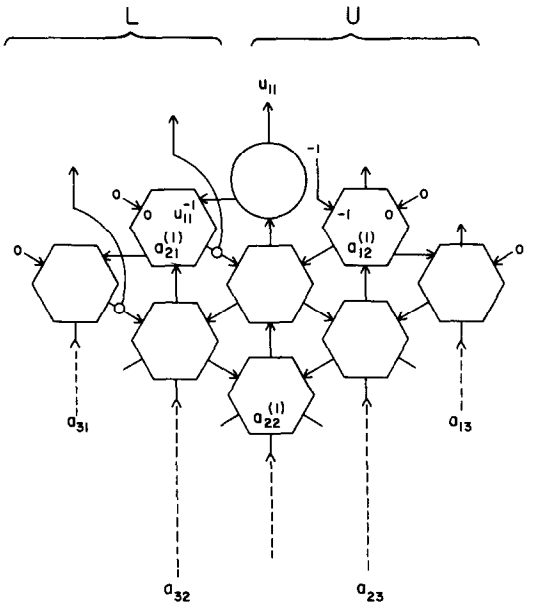


Fig. 3. Systolic array for LU factorization [2].

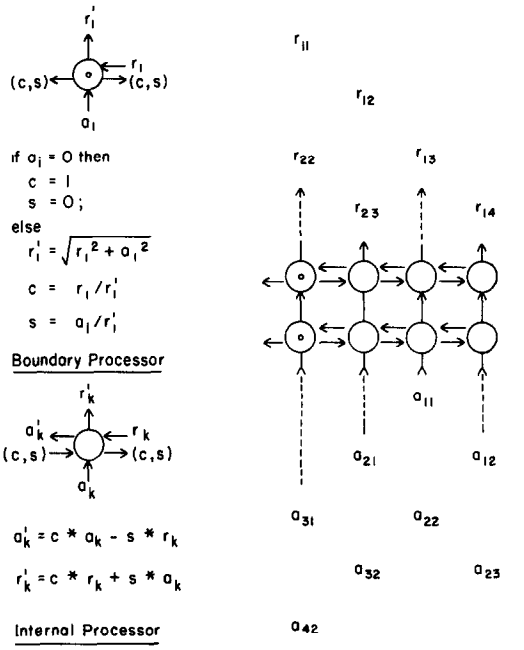


Fig. 4. Systolic array for **QR** decomposition [5].

where

$$\begin{Bmatrix} r_k' \\ a_k' \end{Bmatrix} = \begin{Bmatrix} c * r_k + s * a_k \\ -s * r_k + c * a_k \end{Bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{Bmatrix} r_k \\ a_k \end{Bmatrix} \quad (3c)$$

$$r_i' = \sqrt{r_i^2 + a_i^2}$$

$$c = \frac{r_i}{r_i'}$$

and

$$s = \frac{a_i}{r_i'}. \quad (3d)$$

A systolic array for the Given's transformation of a dense matrix has been suggested by Gentleman and Kung [3].

Recently, a systolic array for **QR** decomposition of a banded matrix has been proposed by Heller and Ipsen [5]. As shown in Fig. 4, the array consists of two basic types of processors: internal processors and boundary processors. The boundary processors are capable of performing the plane rotation operations given in eqn (3d), while the internal processors perform the transformation step given in eqn (3c).

For an m by n banded matrix **A** with column bandwidth $b_c(\mathbf{A})$ and total bandwidth $b(\mathbf{A})$, the complexity of the array for computing the upper triangular matrix **R** is

No. of processor: $b(\mathbf{A}) * b_c(\mathbf{A})$

Units of time: $2n + 2(b_c(\mathbf{A}) - 1)$.

Here, a time unit is the time required for one transformation step and for passing the data from one processor to its neighboring processor.

Solution of linear triangular systems

Given a nonsingular n by n lower triangular matrix **L** and an n-vector **y**, the solution **x**, where **Lx** = **y**, can be computed by the following recurrence formula:

$$z_i^{(1)} = 0 \quad (4a)$$

$$z_i^{(k+1)} = x_i^{(k)} + \ell_{ik} x_k \quad (4b)$$

$$x_i = \frac{y_i - z_i^{(i)}}{\ell_{ii}}. \quad (4c)$$

A linear systolic array for the solution of a linear triangular system has been proposed by Kung and Leiserson [2].

By interconnecting the linear arrays designed by Kung and Leiserson, an orthogonally connected network for solving linear triangular system with multiple right-hand vectors can be constructed as shown in Fig. 5. The systolic network consists of special processors (denoted by circles) performing the operation given in eqn (4c) and the inner step processors performing the operation given in eqn (4b). Given an n by n lower triangular matrix **L** with bandwidth b and an n by m matrix **Y**, the complexity required to compute the n by m matrix **X**, $\mathbf{X} = \mathbf{L}^{-1}\mathbf{Y}$, is

No. of processors: $m\,b$

Units of time: $2n + b + m$.

Since each processor operates every second time

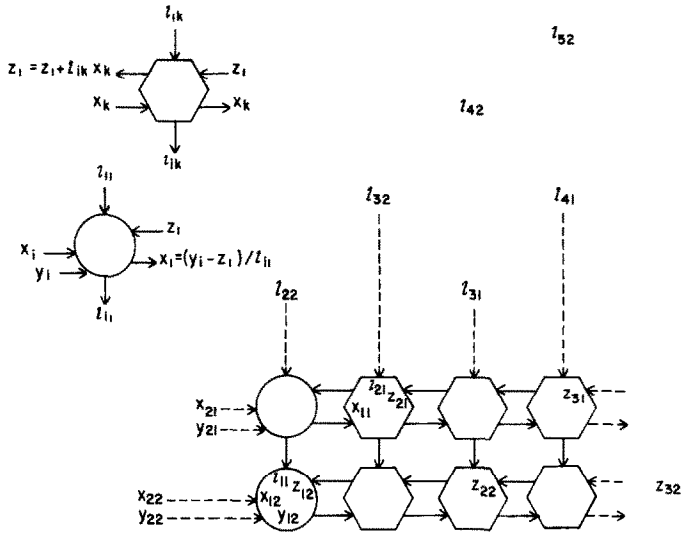


Fig. 5. Systolic array for linear triangular system with multiple right-hand vectors.

step, the number of processors can be reduced by about one-half.

SYSTOLIC ALGORITHMS AND PRIMITIVE FUNCTIONAL MODULES

A modular approach to develop VLSI devices has been proposed by Hwang and Cheng [7, 8]. The approach is to interconnect VLSI primitive modules, each of which is capable of performing a specific task of simple matrix operations, forming an integrated synchronized modular network for special-purpose computation. Data registers (or latches) are provided to allow the output from one module directly transferred as input to other modules. That is, intermediate results are passed from module to module without storing back to the memory.

Primitive arithmetic modules supporting submatrix computations have been used as building blocks to develop globally structured VLSI algorithms for large-scale matrix computations [7, 8]. Each module itself is an array of processors capable of handling small-scale matrix operations such as LU factorization, inversion of triangular matrices and matrix multiplication. This modular approach has been proposed to design synchronized VLSI systems for image processing and database management [8, 9]. This modular approach can also find its applications to finite element computations, where the algorithms are often expressed in terms of small (nodal or elemental) submatrix operations.

The basic structures of the globally structured networks and the systolic arrays are similar in that they allow concurrent operations using the idea of pipelining and parallelism. A globally structured network can be constructed by replacing the processing elements in the systolic array by the primitive arithmetic modules. The data are synchronized by interfacing successive modules with data latches, which hold the input and output of each module. Each latch releases and accepts information when triggered by its external clock signal.

Three primitive VLSI arithmetic chips are functionally described in Fig. 6. They are the M-type chip for accumulative multiplication of two small dense matrices, the D-type chip for LU factorization of a small dense matrix and the L-type chip for multiple solutions of a linear triangular linear system. Each primitive chip is itself an array of simple processors interfaced with data latches for pipelined operations. Examples of the M-type, D-type and L-type chips are the systolic arrays shown in Figs. 2, 3 and 5, respectively.

Globally structured VLSI networks for matrix multiplication and LU factorization can be constructed using the same structures as the systolic arrays by taking the primitive chips as building blocks. In the matrix multiplication array given in Fig. 2a, the inner product step processors can be substituted by the M-type chips for accumulative multiplication of two dense submatrices. For the multiplication of a partitioned matrix **A** with N by M block entries and a partitioned matrix **B** with M by N block entries, the complexity of the network

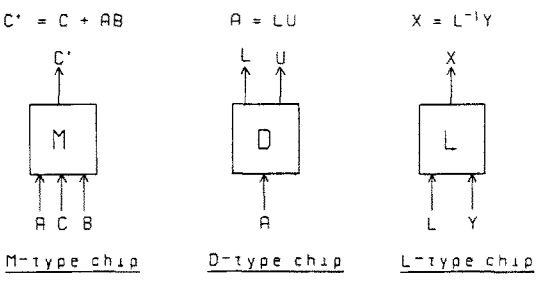


Fig. 6. Functional descriptions of primitive chips.

can be summarized as

No. of Primitive Processors: $B(\mathbf{A}) * B(\mathbf{B})$

Units of Time: $3n + B$,

where $B = \min\{B(\mathbf{A}), B(\mathbf{B})\}$. Each time unit is referred to the time required for an accumulative multiplication of two dense submatrices, and for passing the data from one primitive module to its neighboring modules.

The recurrence formula for the LU factorization given in eqn (2) can be rewitten as

$$a_{ij}^{(1)} = a_{ij} \tag{5a}$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} + \ell_{ik}(-u_{kj}) \tag{5b}$$

$$a_{kk}^{(k)} = \ell_{kk}\, u_{kk} \tag{5c}$$

$$\ell_{ik} = \begin{cases} 0 & \text{if } i < k \\ \ell_{ii} & \text{if } i = k \\ a_{ik}^{(k)}\, u_{kk}^{-1} & \text{if } i > k \end{cases} \tag{5d}$$

$$u_{kj} = \begin{cases} 0 & \text{if } k > j \\ u_{jj} & \text{if } k = j, \\ \ell_{kk}^{-1}\, a_{kj}^{(k)} & \text{if } k < j \end{cases} \tag{5e}$$

where a_{ij}, ℓ_{ik} and u_{kj} are block entries of **A**, **L** and **U**, respectively. The globally structured **LU** factorization network is hex-connected and is depicted in Fig. 7.

In the **LU** factorization network, the processors below the upper boundaries are the M-type chips for accumulative multiplication of two submatrices. The processor at the top is the D-type chip for **LU** factorization of a dense submatrix. The results from the top processor are passed northward. In addition, the top processor passes the upper triangular factor of the submatrix southwest and the lower triangular factor southeast. The processors on the upper boundaries are the L-type chips for the multiple solutions of a linear triangular system. It should be noted that the superdiagonal entries of **U** obtained from the upper right boundary carry a negative sign. For a partitioned banded matrix **A** with N by N block entries, the complexity of the **LU** factorization network can be stated as

No. of primitive processors: $B(\mathbf{A})\,\frac{2}{4}$

Units of time: $3N + \frac{B(\mathbf{A})}{2}$.

Each time unit is referred to the time required to complete a submatrix operation and for passing the data from one primitive processor to the neighboring processors.

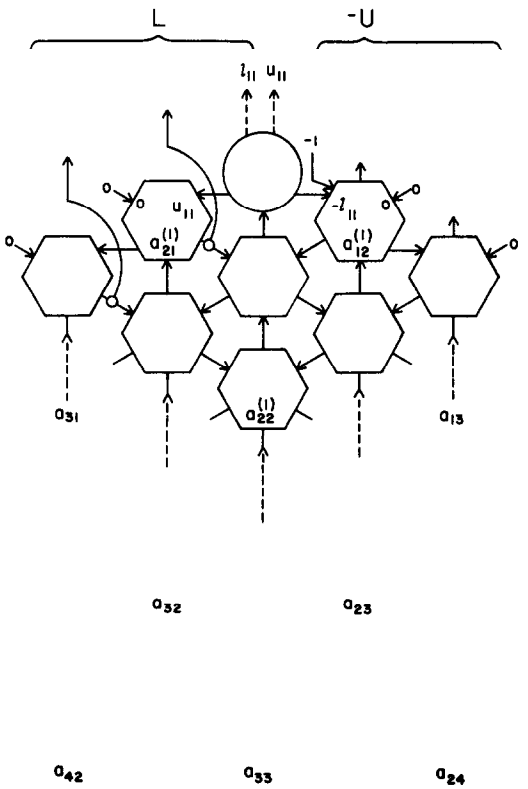


Fig. 7. Globally structured network for LU factorization.

CONFIGURATION OF A FINITE ELEMENT SYSTEM

The use of finite element methods involves very large amounts of computations. The algorithms based on these methods can be modularized into several processes. The computationally demanding processes include the generation of element stiffnesses, assembly, factorization and solution of the equilibrium equations. It may be cost effective to off-load these computationally demanding processes from general-purpose host computers to an integrated special-purpose VLSI system.

The integration of VLSI special-purpose devices into a complete system generally involves three components: the host processor, interface processors and the special-purpose devices [11]. A conceptual integrated finite element system is configured in Fig. 8.

In the integrated finite element system, the host processor is the central controller for the system. It runs the operating system and schedules and monitors activities of the components in the system. The host processor is responsible for performing necessary operations, preparing data stored in host memory and downloading the data to the respective special-purpose modules in the system.

Interface processors serve as communication links between the host computer and each individual special-purpose module. An interface processor has large memory for passing data in and out of each module. It can also serve to coordinate and interface the special-purpose modules through a microprogrammable processor.

The special-purpose modules in this integrated finite element system include the modules for gen-

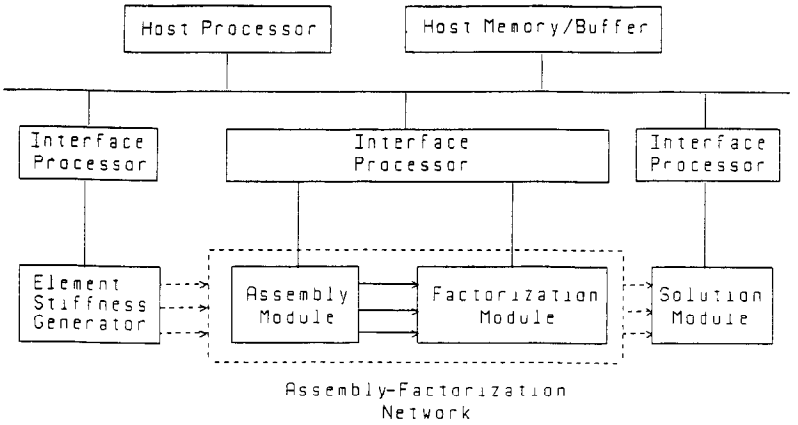


Fig. 8. Conceptual configuration of an integrated finite element system.

eration of finite element stiffness, assembly and factorization of global stiffness matrix and the solution of the equilibrium equation. Private memory to store necessary data is provided. The data are directly pipelined from module to module, without storing back to memory. Hence, it is important that the I/O behaviors of the special-purpose modules are uniform so that regularity of data flow through the integrated system can be maintained. For instance, the I/O from the assembly module to the factorization module should be uniform and compatible to allow concurrent computations in the two modules.

In the following sections, systolic networks for the assembly process are proposed for the stiffness method and the natural factor approach of finite element analysis. The results from the assembly modules are directly pipelined to the respective factorization modules. In fact, the assembly and factorization modules can be integrated into a single system.

SYSTOLIC NETWORKS FOR STIFFNESS METHOD OF FINITE ELEMENT ANALYSIS

In the congruent transformation approach of finite element analysis, the formation of the global stiffness matrix can be written as [12]

$$\mathbf{K} = \mathbf{A}^t \mathbf{K}^e \mathbf{A},$$

where $\mathbf{K}^e$ is an unassembled global stiffness matrix, where the block diagonal entries are the element stiffness matrices. $\mathbf{A}$ and $\mathbf{A}^t$ are, respectively, the global kinematics and global statics matrices. The congruent transformation approach involves the formation of $\mathbf{K}^e$ and $\mathbf{A}$, each of which in general is larger than $\mathbf{K}$. However, other advantages of this approach can be accounted [12]. First, the element stiffness matrices need not include the rigid body displacement modes; i.e. degrees of freedom corresponding to statically determinate stable support conditions may be excluded. Second, the matrix $\mathbf{A}$ can be used to establish the correspondence of element and global degrees of freedom accounting for the boundary conditions. Third, the generation of element stiffnesses does not require the information regarding to the geometric and physical boundary conditions, and thus the design of systolic array for generating element stiffnesses can be simplified. In general, the connectivity matrix $\mathbf{A}$ is a sparse matrix with unit values, direction cosines and lengths, and it is well banded if the finite elements and the nodal variables are properly ordered.

In sequential computation the matrix product given by eqn (6) is not performed explicitly but is done by direct summation of the contribution of the element stiffnesses at a degree of freedom to reduce the amounts of operations. However, in the assembly network proposed, the formation of the global stiffness matrix is carried out by the triple matrix product. As shall be seen, the matrix multiplication will not decrease the overall efficiency of the assembly-factorization process if the outputs from the assembly network are passed directly to the factorization network.

A globally structured modular network for the assembly process is given in Fig. 9. In this approach each element stiffness matrix is treated as a single entity. The matrix product $\mathbf{A}^t\mathbf{K}^e$ is computed using a linear network in which each processor is an M-type chip capable of performing an accumulative matrix multiplication. The results from the linear array are then pipelined directly to the hex-connected network, in which the processors are also M-type chips.

As shown in Fig. 9, the coefficients a^t_{ej} of matrix $\mathbf{A}^t$ and the element stiffnesses k^e are pipelined into the linear network synchronously. When the coefficient a^t_{ej} of $\mathbf{A}^t$ leaves the linear network, its transpose a_{ej} begins to enter the hex-connected network from the top boundary. Thus, a "wrap-around" scheme may be used to route the output coefficient a^t_{ej} from the linear network to the hex-connected network as input.

The time delay from the first element stiffness

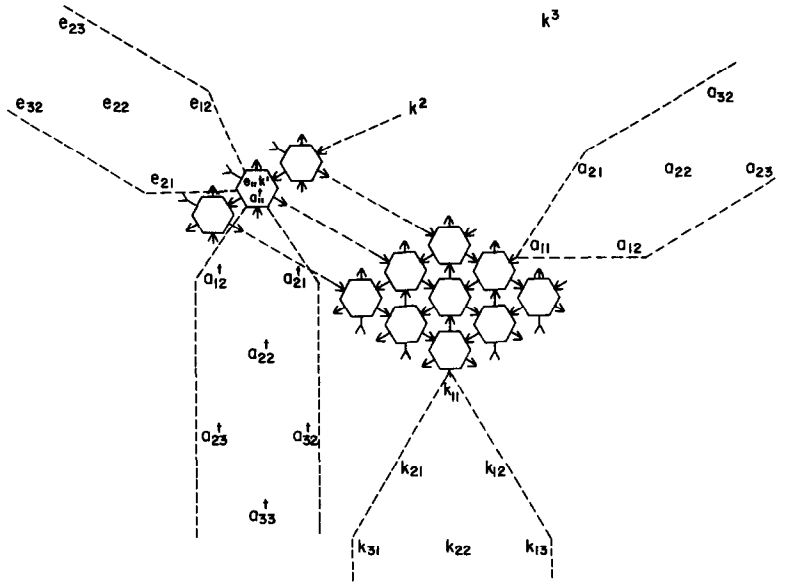


Fig. 9. Systolic network for assembling global stiffness matrix.

matrix $\mathbf{k}^l$ entering the linear network to the coefficient $e_{11} = \mathbf{k}^l a_{11}$ entering the hex-connected array is equal to the block column bandwidth $B_c(\mathbf{A})$ of $\mathbf{A}$. Therefore, the complexity of the assembly network can be stated as

No. of primitive processors: $B(\mathbf{A})^2 + B(\mathbf{A})$

Unit of time: $3N + B(\mathbf{A}) + B_c(\mathbf{A}) + 1$,

where N is the number of nodal variables in the finite element mesh, and a time unit is referred to the time required for an accumulative multiplication of two submatrices and for passing the data to the neighboring processors.

The fact that the data flows of the assembly network and of the LU factorization network are of a similar pattern suggests the possibility of chaining together the two "compatible" networks to form a single module. An assembly-factorization network is shown in Fig. 10. The arrows in the figure indicate the flow of the data. Two consecutive steps during the execution of the assembly-factorization module also are illustrated in Fig. 10.

The time required for a stiffness coefficient $\mathbf{k}_{ij}$ to pass through the LU factorization network equals $B(\mathbf{K})/2$, where $B(\mathbf{K})$ is the bandwidth of the global stiffness matrix and is at most equal to $2B(\mathbf{A}) - 1$. By taking advantage of concurrent execution of assembly and factorization, the complexity of the integrated assembly-factorization network can be stated as

No. of primitive processors: $2B(\mathbf{A})^2 + B(\mathbf{A})$

Units of time: $3N + 2B(\mathbf{A}) + B_c(\mathbf{A}) + 1$.

The above result concludes that by allowing concurrent execution the time complexity of assembly-factorization is only $B(\mathbf{A}) + B_c(\mathbf{A})$ more than that of the factorization alone. As in matrix multiplication and LU factorization, since each processor operates every third time step, it is possible to reduce the number of processors by about two-thirds.

SYSTOLIC NETWORK FOR NATURAL FACTOR APPROACH OF FINITE ELEMENT ANALYSIS

The natural factor formulation of the element stiffness matrices has proven to be a very successful method in finite element analysis, both from the physical and the mathematical point of view [13, 14]. The method exhibits some notable data flow properties suitable for systolic implementation.

Due to the quadratic nature of the energy functional, there often exists a "natural factor" $\mathbf{s}^e$ of the element stiffness matrix $\mathbf{k}^e$, where

$$\mathbf{k}^e = \mathbf{s}^{e^t}\,\mathbf{s}^e.$$

The global stiffness matrix $\mathbf{K}$ can be expressed as

$$\mathbf{K} = \mathbf{A}^t\mathbf{S}^t\mathbf{S}\,\mathbf{A} = \mathbf{E}^t\,\mathbf{E},$$

where $\mathbf{S}$ is an unassembled matrix containing the natural factors $\mathbf{s}^e$ of the element stiffness matrices, and $\mathbf{E} = \mathbf{SA}$. The matrix $\mathbf{E}$ is then reduced to an upper triangular form by QR decomposition, yielding the factorization.

$$\mathbf{K} = \mathbf{R}^t\mathbf{Q}^t\,\mathbf{Q}\,\mathbf{R} = \mathbf{R}^t\,\mathbf{R},$$

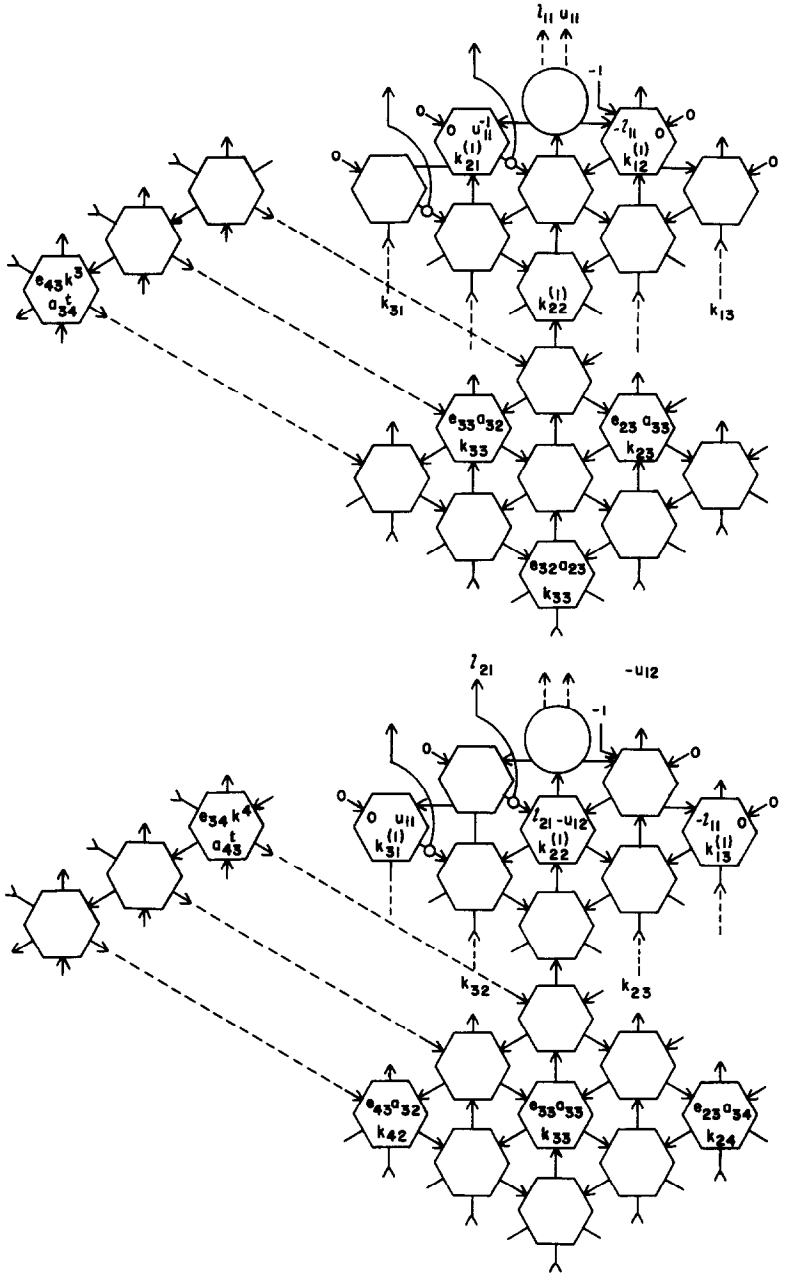


Fig. 10. Systolic assembly-factorization network for the stiffness method.

where **R** is the upper triangular factor of **K** and **Q** is an orthogonal matrix.

Similar to the systolic network for the stiffness analysis, the network for the natural factor formulation procedure can be designed by chaining together two systolic arrays: a matrix multiplication array to compute the product **E** = **SA**, and a **QR** decomposition array to compute the upper triangular matrix **R**. The systolic array of Heller and Ipsen [5], shown in Fig. 4, is a natural choice for the **QR** decomposition. The approach is to select an assembly array for computing the matrix product **E** = **SA**, whereof the results can be pipelined to the **QR** decomposition array.

In the **QR** decomposition array the coefficients enter the array in every two time units. In the matrix multiplication array designed by Huang and Abraham, the coefficients enter and leave the array in every time unit [4]. To ensure consistency and regularity in the data flow, delay processors can be added to the multiplication array. The systolic network for the natural factor approach of finite element analysis is shown in Fig. 11.

For the systolic network shown in Fig. 11, the input data are the natural factors of the element stiffnesses and the kinematics or connectivity matrix; the output is the upper triangular factor of the global stiffness matrix. Two consecutive steps dur-

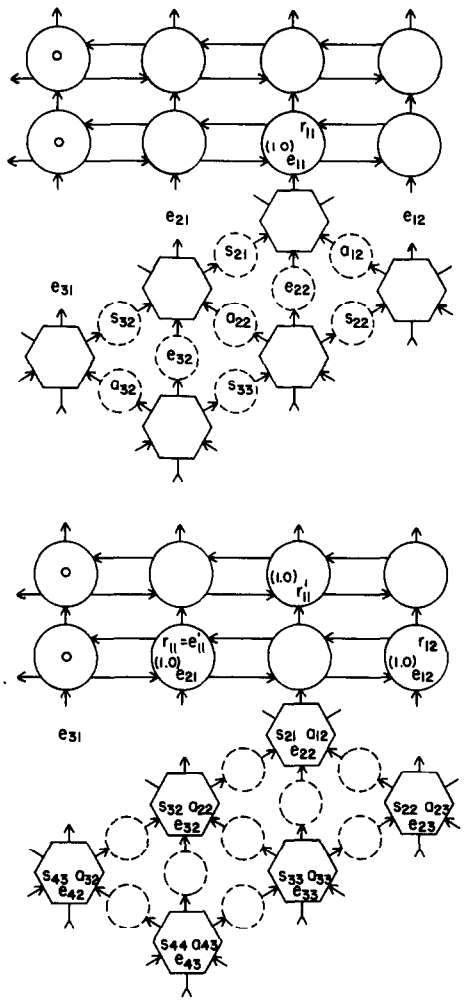


Fig. 11. Systolic assembly-factorization network for the natural method.

ing the execution of the assembly-factorization network are illustrated in Fig. 11.

The time required for a coefficient e_{ij} to pass through the matrix multiplication array, including the delay processors equals $2b - 1$, where $b = \min\{b(\mathbf{A}), b(\mathbf{S})\}$. The complexity of the network for the natural factor formulation can be stated as

No. of processors: $b(\mathbf{S}) * b(\mathbf{A}) + b(\mathbf{E}) + b_c(\mathbf{E})$

Units of time: $2n + 2b_c(\mathbf{E}) + 2b - 3$,

where n is the number of degrees of freedom in the finite element mesh. By allowing concurrent execution, the time complexity of the assembly-factorization procedure is only $2b$ more than that of **QR** decomposition alone.

CONCLUDING REMARKS

Systolic systems for the assembly-factorization procedures for the stiffness method and the natural factor approach of finite element analysis have been presented. The approach is to chain together two or more systolic modules with compatible data flow patterns. The outputs from the assembly module are directly pipelined to the factorization module without storing back to the memory of the host processor. Regularity of data flow is maintained in both assembly and factorization networks. By allowing concurrent computations, the time complexity of the assembly-factorization procedure is $O(N)$, where N is the number of nodal variables, or degrees of freedom, in the finite element mesh.

This paper has been devoted to the design of systolic networks for assembly and factorization. Efforts have recently been initiated for developing systolic systems for generating element stiffnesses [15]. Future research should include the integration of the systolic system for generating element stiffnesses into the finite element system.

REFERENCES

1. H. T. Kung, Why systolic architectures? *Computer* **15**, 37–46 (1982).
2. H. T. Kung and C. E. Leiserson, Systolic arrays for VLSI. in *Sparse Matrix Proceedings 1978*, pp. 256–282, Society for Industrial and Applied Mathematics (1979).
3. W. M. Gentleman and H. T. Kung, Matrix triangulation by systolic arrays. *Proceedings for Real Time Signal Processing IV*, SPIE, Vol. 298, pp. 19–26 (1981).
4. K. H. Huang and J. A. Abraham, Efficient parallel algorithms for processor arrays. *Proceedings of the 1982 International Conference on Parallel Processing*, IEEE, pp. 271–279 (1982).
5. D. E. Heller and I. C. F. Ipsen, Systolic networks for orthogonal decomposition. *SIAM J. Scient. Statist. Comput.* **4**, 261–268 (1983).
6. K. H. Law, *Systolic schemas for finite element methods*. No. R-82-139, Department of Civil Engineering, Carnegie–Mellon University (1982).
7. K. Hwang and Y. H. Cheng, Partitioned algorithms and VLSI structures for large scale matrix computations. *Fifth Symposium on Computer Arithmetic*, IEEE Computer Society, pp. 222–232 (1981).
8. K. Hwang and Y. H. Cheng, Partitioned matrix algorithms for VLSI arithmetic systems. *IEEE Trans. Comput.* **12**, (C-31) 1215–1224 (1982).
9. K. Hwang and K. S. Fu, Integrated computer architectures for image processing and database management. *Computer* **16**(1), 51–60 (1983).
10. W. M. Gentleman, Least square computations by Given's transformations without square roots. *J. Inst. Math. Appl.* **12**, 329–336 (1973).
11. H. T. Kung and S. W. Yu, Integrating high performance special purpose devices into a system. Tech. Rep., Department of Computer Science, Carnegie–Mellon University (1982).
12. R. H. Gallagher, *Finite Element Analysis—Fundamentals*. Prentice–Hall, Englewood Cliffs, New Jersey (1975).
13. J. H. Argyris and O. E. Bronlund, The natural factor formulation of the stiffness for the matrix displacement method. *Comp. Meth. Appl. Mech. Engng.* **5**, 97–119 (1975).
14. J. H. Argyris *et al.*, Finite element method—The natural approach. *Comp. Meth. Appl. Mech. Engng.* **17/18**, 1–106 (1979).
15. R. Melhelm, Formal verification of a systolic system for finite element stiffness matrices. Rep. No. ICMA-83-56, Department of Mathematics and Statistics, University of Pittsburgh (1983).