Formal Analysis of a Systolic System for Finite Element Stiffness Matrices*

RAMI MELHEM†

Department of Computer Science and Department of Mathematics, University of Pittsburgh, Pittsburgh, Pennsylvania 15261

An abstract systolic model, suggested in a previous paper, is extended to include computational cells with memories and multiplexing capabilities, and to specify formally the pipelining of computations through systolic networks. The basic idea is to represent the data items appearing on any communication link of a systolic network by a data sequence and to specify the operation on each cell by a set of equations using operators on sequences. The model is then applied successfully to a complex system that we suggest for pipelining the computation of the elemental stiffness matrices for finite element analysis.

1. INTRODUCTION

In [13] an abstract model was developed for the specification of systolic networks and the verification of the correctness of their operation. In this paper, we extend this model to allow for networks with slightly more complicated types of computational cells, namely cells that have periodic memory or multiplexing capabilities. The motivation for this extension is that we have to free ourselves from the simple inner product cell [10] if we want to use systolic networks in a wider range of applications. It should be noted, however, that the suggested extensions remain very simple in structure and should not result in a complicated design for the individual cells. Also it appears that the most desirable approach to the design of widely applicable systolic networks is to utilize a fairly general generic cell. If this generic cell were to be controlled by microcode (see, e.g., [6]), then it could be applied easily to the implementation of the suggested extended cells.

The model presented in [13] and extended in this paper is similar to another model developed independently by Chen and Mead [3]. Both separate the network function from the specific details of a certain computation and allow for a precise specification and a formal verification of systolic networks. However, the model in [3] is oriented toward a procedural specification, while we followed a more

* This work was in part supported under ONR contract N00014-80-C-0455.
† The current author address is: Department of Computer Science, Purdue University, West Lafayette, IN 47907.
algebraic approach. We should also mention previous approaches [4, 8, 16] for formalizing systolic networks by means of a delay operator and a notation that envisions the flow of data as a wave front propagating over the network. However, this notation does not seem to be powerful enough to describe the operation of general systolic networks, especially if more elaborate cells are to be used.

The extended model is applied to the description and verification of a pipelined systolic system designed for the computation of finite element stiffness matrices. This represents an important step in the finite element analysis extensively used by engineers and scientists for the solution of boundary value problems. Very briefly, finite element analysis [18] is a technique for solving partial differential equations on a certain domain \( Q \) with given conditions on the boundary of \( Q \). In the case of linear equations, it involves essentially the following four basis steps: (1) The generation of a finite element mesh that divides \( Q \) into \( m \) finite elements. (2) The generation of elemental stiffness matrices \( H^e \) and elemental load vectors \( b^e \) for each finite element \( e, e = 1, \ldots, m \). (3) The assembly of the global stiffness matrix \( H \) and of the load vector \( b \). (4) The solution of the linear system of equations \( Hx = b \).

In the past two decades, many finite element software systems have been developed and widely used. However, in practice, the time and storage required by these systems to complete an analysis may be extremely large. This usually imposes severe limitations on the size and type of the problem that can be handled and often leads engineers to use less accurate models or lower degrees of approximations. For this reason, many researchers have considered some form of parallel processing in the finite element analysis, as for instance, the use of array processors [15], general purpose multiprocessors [5], or adaptive, special purpose multiprocessor systems [17]. A common result in most of these experiments is that the time for data movement and interprocessor communication is very large and sometimes dominates the running time.

On the other hand, a machine was designed at the NASA–Langley Research Center [1, 9] specifically for finite element computations. In this machine, a rectangular array of processors is formed by connecting each processor to its eight nearest neighbors with a global bus connecting all the processors of the system. Each processor is assigned to the computations associated with one or more nodes in the finite element mesh.

Along the line of systolic architectures, Law [11] suggested a systolic network to assemble the global stiffness matrix, and Kung and Leiserson [10] and Brent and Luck [2] designed systolic networks that can be used for solving the resulting system of equations. However, no attempts have been made to use systolic networks for generating the elemental stiffness matrices, which is the subject of this paper.

The paper is arranged as follows: In Section 2, we review and extend the basic features of the systolic model presented in [13], and in Section 3, we give a general description of the system used to generate the elemental stiffness matrices. The different components of the system are then described and verified in Section 4. It is not surprising that the formal notation used in that section tends to be somewhat
tedious. However, it is unclear how the situation may be improved without sacrificing preciseness, especially in such a large system. In fact, we believe that the formal approach is most useful in large and complex systems, where informal arguments may lead to ambiguity and confusion.

2. Review and Extension of the Formal Systolic Model

In this section, we briefly review the main features of the abstract systolic model presented in [13]. Basically a systolic network is represented by a directed graph with two different types of nodes, namely interior nodes and I/O nodes corresponding to computational cells and I/O cells of the network, respectively. The edges of the graph model the communication links of the network. In order to identify the elements of the graph, every node is given a unique label and every edge is identified by a pair \((c, i)\), where \(c\) is a color assigned to the edge from a finite set of colors, and \(i\) is the label of the node at which the edge terminates. The only restriction placed upon the edge colors is that edges directed to the same node should have different colors and that the same holds for all edges directed out of a node.

In addition to the graph that reflects the topology of the network, the model associates with each edge an infinite data sequence which is the sequence of data items that appear on the corresponding communication link at consecutive time units. More precisely, let \(N\) and \(R\) be the sets of positive integers and real numbers, respectively, and set \(R_\delta = R \cup \{\delta\}\), where \(\delta\) is a special element called the "don't care" element. Then the data sequence \(\eta_i\) associated with the edge \((y, i)\) is a mapping \(\eta_i: N \rightarrow R_\delta\) such that \(\eta_i(t) \in R_\delta\) is the data item which appears on the link at time \(t\). If \(\eta_i(t) = \delta\) for some \(t\), this indicates that we do not care (or do not know) about the data on \((y, i)\) at the time \(t\). We use the convention of denoting the pair \((y, i)\) by \(y_i\) and the associated sequence by \(\eta_i\), where \(\eta\) is the greek letter corresponding to \(y\).

At this point, we note that we have chosen \(R\) to be the set of real numbers because of the nature of our problem. More generally, \(R\) could be any set of items that can be transmitted on the communication links of the network.

Let \(\bar{R}_\delta\) be the set of all sequences that contain at most a finite number of non-\(\delta\) elements. Then it is natural to define the termination function \(T: \bar{R}_\delta \rightarrow N_0 = N \cup \{0\}\) with the property that for any sequence \(\eta\), \(T(\eta)\) is the position of the last non-\(\delta\) element in \(\eta\). For the don't care sequence defined by \(\delta^*(t) = \delta\) for all \(t\), we then have \(T(\delta^*) = 0\). We also define the zero sequences \(\iota\) with \(\iota(t) = 0\) for \(1 \leq t \leq T(\iota)\) and any arbitrary large \(T(\iota)\).

The computation performed by a computational cell with \(m\) input links and \(n\) output links is now modeled by \(n\) causal sequence operators \(\Gamma_i: [\bar{R}_\delta]^m \rightarrow \bar{R}_\delta\). \(i = 1, \ldots, n,\) one for each output link. In essence, a causal operator is such that the \(r\)th element of the image sequence can depend only on any element \(j\) of its operands with \(j < r\). If the condition \(j < t\) is replaced by \(j \leq t\), the operator is called "weakly causal." For the exact definition of causal and weakly causal operators we refer to [13].
In order to model the computation of the entire network, we establish for each node of the network the sequence equations describing its operation, these are the equations relating the input sequences and the output sequences by means of causal operators. Then, if possible, we solve the resulting system of equations and obtain in this way an explicit relation between the network output sequences and the network input sequences. This relation is called the “network I/O description.” Finally, for a verification of the operation of the network for a specific form of input sequences, we substitute these particular sequences into the I/O description, which, possibly after some manipulation, yields an explicit form of the network output sequences.

As the above review already indicates, operators on sequences play a key role in our model. One way of defining sequence operators is to extend known operators on $\mathbb{R}$ to $\mathbb{R}_\delta$ by applying the operator element-wise to the elements of sequences. Examples are the sequence addition “$+$”, multiplication “$\cdot$”, and scalar multiplication “$\cdot$.” Element wise operators, in turn, can be classified in terms of the result of any operation involving the don’t care element $\delta$, namely: (1) $\delta$-regular operators for which the result of any operation involving $\delta$ is $\delta$. This class of operators treats $\delta$ as a “don’t know” quantity, and consequently the result cannot be known if any of the operands is not known. (2) Non $\delta$-regular operators, where $\delta$ is treated as a special symbol that affects the result of the operation. Example are the operators $\min_\delta$ and $\max_\delta$ defined in [13]. In practice, this class of operators can be used to model a network where the communication links are augmented by an additional wire to indicate whether the link carries valid data or not. The operation of each computational cell is then dependent on this additional piece of information.

A second type of operators consists of those defined directly on $\mathbb{R}_\delta$. In the remainder of this section we introduce several such operators that will be used in the specification and verification of our finite element system. For simplicity, given any operator $\Gamma: [\mathbb{R}_\delta]^n \to \mathbb{R}_\delta$, the notation $[\Gamma(\xi_1, \ldots, \xi_n)](t)$, will be employed to designate the $t$th element $\eta(t)$ of the image sequence $\eta = \Gamma(\xi_1, \ldots, \xi_n)$. This is consistent with the convention of using square brackets for grouping. We will also use the symbol “$\div$” for integer division and the Fortran function $\text{mod}(\ )$ that specifies the remainder of an integer division.

The shift operator. $\Omega': \mathbb{R}_\delta \to \mathbb{R}_\delta$ is defined by

$$[\Omega'\delta](t) = \delta \quad \text{if } r > 0 \text{ and } t \leq r$$

$$= \xi(t - r) \quad \text{otherwise}.$$ 

Hence, for $r > 0$, $\Omega'$ inserts $r \delta$-elements at the beginning of a sequence and therefore models the computation of a delay cell. On the other hand, for $r < 0$, $\Omega'$ trims the first $r$ elements of the sequence and thus is a noncausal operator which cannot be used to model computational cells. The role of the negative shift operator
is to provide in the proofs an inverse for the positive shift. More precisely, for any sequence $\xi$, we have $\Omega^{-r}\Omega^r\xi = \xi$. The converse is not always true, in the sense that $\Omega^r\Omega^{-r}\xi = \xi$ only if $\xi(t) = \delta$ for $t \leq r$.

The accumulator operator. $A^{r,k,s}: \mathbb{R}_\delta \to \mathbb{R}_\delta$ is defined to model a cyclic accumulator that starts operation at time $t = r$, accumulates a new element every $s$ time units and restarts a new cycle every $sk$ time units. The accumulator operator can be defined in terms of the following algorithm that computes $[A^{r,k,s}\xi](t)$ for any $t > 0$, given the sequence elements $\xi(j)$ for $j < t$.

IF $(t < r)$ THEN $[A^{r,k,s}\xi](t) = \delta$ /* accumulator is idle */
ELSE BEGIN

$t_r = t - \text{mod}((t - r) \div sk)$ /* time of last reset */

$na = ((t - t_r) \div s) + 1$ /* number of elements accumulated */

$[A^{r,k,s}\xi](t) = \sum_{j=0}^{na-1} \xi(t_r + sj)$ /* result of accumulating $na$ elements */

END

Evidently, this algorithm is equivalent to

$$[A^{r,k,s}\xi](t) = \delta$$ for $t < r$,

$$= \sum_{j=0}^{na-1} \xi(t_r + sj)$$ for $t \geq r$,

where $na$ and $t_r$ are as specified before. As an example, let

$$\xi = a_1, b_1, a_2, b_2, ..., a_7, b_7, \delta, \delta, ...,$$

then

$$A^{2,3,2}\xi = \delta, b_1, O, b_1 + b_2, O, b_1 + b_2 + b_3, O, b_4, O, b_4 + b_5, O, b_4 + b_5$$

$$+ b_6, O, b_7, O, \delta, \delta, ...,$$

where $O$ denotes an element that is equal to the preceding one.

The multiplexer operator. $M^{w_1,...,w_n}(\xi_1, ..., \xi_n): [\mathbb{R}_\delta]^n \to \mathbb{R}_\delta$ is defined to model a multiplexer that has $n$ inputs $\xi_1, ..., \xi_n$. It starts its operation at time $t = r$ and periodically multiplexes its inputs with a time ratio of $w_1: w_2: ...: w_n$. If the length of the multiplexer cycle is denoted by $k = \sum_{r=1}^{n} w_r$, then the following algorithm defines the multiplexer operator.
IF ($t < r$) THEN $\lfloor M_r^{w_1,\ldots,w_n}(\xi_1,\ldots,\xi_n) \rfloor(t) = \delta$  /* multiplexer idle */

ELSE BEGIN

$t_1 = t - \text{mod}((t - r) \div k)$  /* start of current cycle */

Find the largest integer $1 \leq e \leq n$

such that $(t - t_1) < \sum_{j=1}^{e} w_j$  /* determine interval within cycle */

$\lfloor M_r^{w_1,\ldots,w_n}(\xi_1,\ldots,\xi_n) \rfloor(t) = \xi_e(t)$  /* chose corresponding input */

END

As an example, let

$\zeta = a_1, a_2, \ldots, a_7, a_8, a_9, \delta, \delta, \ldots$  and $\eta = b_1, b_2, \ldots, b_7, \delta, \delta, \delta, \ldots$

then $M_2^{1,2}((\zeta, n)) = \delta, \delta, a_3, b_4, b_5, a_6, b_7, \delta, \delta, \delta, \ldots$

It is also interesting to note that the multiplexer operator can be used to model a demultiplexer cell. For example, if we want to sample the sequence $\xi$ at times $t = r, 2r, 3r, \ldots$, then we may express this operation as $M_r^{1-1}(\xi, \delta^*)$, where $\delta^*$ is the don’t care sequence introduced earlier.

The multiplexer operator can be used to define two further operators, namely, the expansion and the piping operators.

The expansion operator. $E_r^k : \mathbb{R} \rightarrow \mathbb{R}$ models a cyclic memory that is loaded at time $t = r$ and is overwritten every $k$ time units. It is formally defined by

$E_r^k \eta = M_r^{1-k-1}(\eta, \Omega^2 \eta, \ldots, \Omega^{k-1} \eta)$.

which, from the definition of the multiplexer operator, may be rewritten as

$E_r^k \eta = \delta$

$t < r$,

$= \eta(t - t_u)$  $t \geq r$,

where $t_u = \text{mod}((t - r) \div k)$. For example, with $\xi$ of (1) we have

$E_2^4 \xi = \delta, b_1, O, O, O, b_3, O, O, O, b_5, O, O, b_7, O, O, \delta, \delta, \ldots$.

It should be noted that the accumulator, multiplexer, and expansion operators are weakly causal operators, and that their definitions allow us to model cells with memory capabilities, despite the fact that our abstract model does not explicitly allow the nodes to have memories or internal states.

Besides the causal and weakly causal operators used in modeling computational cells, some sequence operators may be introduced for the sole purpose of allowing us to simplify the description of data sequences. Following are two such operators:
The piping operator. $P^k_m : [\overline{\mathcal{R}}_\delta]^m \rightarrow \overline{\mathcal{R}}_\delta$ defined by
\[ P^k_m(\eta^1, \ldots, \eta^m) = M^k_{i\ldots,j}(\eta^1, \ldots, \Omega^{i(1-k)}\eta^i, \ldots, \Omega^{(m-1)k}\eta^m) \]
and $T(P^k_m(\eta^1, \ldots, \eta^m)) = mk$. In other words, $P^k_m$ concatenates the first $k$ elements of each of the $m$ sequences $\eta^e$, $e = 1, \ldots, m$, and forms one long sequence. It is very useful for the verification of pipelined operations.

On the basis of the definition of the multiplexer operator it is easily shown that the following algorithm is equivalent with the above definition of the piping operator.

\[ \text{IF } (t > mk) \text{ THEN } [P^k_m(\eta^1, \ldots, \eta^m)](t) = \delta \]

\[ \text{ELSE BEGIN} \]
\[ \text{Find the largest integer } 1 \leq e \leq m \text{ such that } t \leq ek \]
\[ [P^k_m(\eta^1, \ldots, \eta^m)](t) = \eta^e(t - (e - 1)k) \]
\[ \text{END} \]

In the following sections, we will use the abbreviations $P^k_{s-1,m}(\eta^e)$ for $P^k_m(\eta^1, \ldots, \eta^m)$, and $P^k_m(\eta)$ for $P^k_m(\eta^1, \ldots, \eta)$.

The spread operator. $\theta^s : \overline{\mathcal{R}}_\delta \rightarrow \overline{\mathcal{R}}_\delta$ defined by
\[ [\theta^s] = \xi(t + s \overline{\eta^1}) \]
\[ = \delta \text{ otherwise.} \]

Hence $\theta^s$ inserts $s$ $\delta$-elements between every two elements of $\xi$. With the sequence $\xi$ of (1) we have, for example,
\[ \theta^2 = a_1, \delta, \delta, b_1, \delta, \delta, a_2, \delta, \delta, b_2, \ldots. \]

Controlling the Operation of Systolic Cells

As mentioned earlier, the operators $A^{r,k,s}$, $M^{n_1,\ldots,n_n}$, and $E^s$ can be used to model systolic cells, where the indices $r$, $k$, and $s$ control different timings as for instance, the reset times, the idle times, and the active times of the cell. One way of monitoring these different timings in physical cells is by providing each cell with a separate circuit that generates reset and idle signals. On the other hand, timings may be monitored also by signals external to the cell. This external control method treats data and control signals in a uniform manner [7], and is especially preferred if the timing signals can be propagated in the network systolically.

The external control approach is equivalent with a redefinition of the operators where the control indices $r$, $k$, and $s$ are replaced by an additional control argument. For example, the expression $E^s \xi$ used in modeling a periodic memory
RAMI MELHEM

cell may be replaced by $E(\xi, \gamma)$, where the nonperiodic expansion operator $E$ is defined by

$$[E(\xi, \gamma)](t) = [E(\xi, \gamma)](t-1) \quad \text{if } \gamma(t) = 0,$$

$$\xi(t) \quad \text{if } \gamma(t) = 1,$$

and the control sequence $\gamma$ controls the resetting of the memory element; that is,

$$\gamma(t) = 1 \quad t = r, r + k, r + 2k, \ldots,$$

$$= 0 \quad \text{otherwise.}$$

It should be easy to verify that in all the networks presented in Section 4, external control signals may be propagated in the network systolically.

3. PROBLEM DEFINITION AND GENERAL DESCRIPTION OF THE SYSTEM

The purpose of the systolic system presented in this paper is to generate the finite element stiffness matrices $H^e$, $e = 1, \ldots, m$, for a given finite element computation based on a given mesh on the domain $Q$ of the problem. In order to simplify the design and the description of the system, we assume that all elements are of the same type, and hence that the number $k$ of nodes per element is the same for all of them.

The class of problems to be considered is a fairly general class of 2-dimensional, stationary, elliptic boundary value problems [17]. The $(i,j)$th entry of the symmetric matrix $H^e$, corresponding to the element $e$, $1 \leq e \leq m$, is given by the general formula

$$H_{ij}^e = \sum_{r,l=0}^{2} a_{ij}^e \int_{Q^e} (D_r u_{e,i})(D_l u_{e,j}) \, dx \, dy, \quad i,j = 1, \ldots, k, \quad (2)$$

where $a_{ij}^e = a_{ij}^*, r, l = 0, 1, 2$, are constant coefficients specified on each element by the problem, and $D_1 u_{e,i} = \partial u_{e,i}/\partial x$, $D_2 u_{e,i} = \partial u_{e,i}/\partial y$, $D_0 u_{e,i} = u_{e,i}$, and $u_{e,i}(x, y)$ denote piece-wise smooth basis functions with the property that $u_{e,i}(x, y)$ is equal to 1 at the $i$th node of the finite element $e$ and to 0 at any other node in $Q$. The integration in (2) is performed over the area $Q^e$ of the finite element $e$.

In order to evaluate the integrals, an isoparametric transformation [18] is used to map the domain of each element $Q^e$ into a standard element $Q$ of the same type in another 2-dimensional space $(\hat{x}, \hat{y})$, namely

$$x = \sum_{i=1}^{k} \hat{u}_i(\hat{x}, \hat{y}) \, x_i^e \quad (3a)$$

$$y = \sum_{i=1}^{k} \hat{u}_i(\hat{x}, \hat{y}) \, y_i^e \quad (3b)$$
where $\tilde{u}_i(\tilde{x}, \tilde{y}) = u_e(x(\tilde{x}, \tilde{y}), y(\tilde{x}, \tilde{y})), \ i = 1, ..., k$, are the basis functions in the new space $(\tilde{x}, \tilde{y})$ and $(x_i^r, y_i^r), \ i = 1, ..., k$ are the coordinates of the $k$ nodes in the finite element $e$. The integrals are then evaluated numerically over $\tilde{Q}$ instead of $Q^e$.

Without entering into the mathematical details, we give only the final formula used to evaluate $H_{i,j}$:

$$
H_{i,j} = \sum_{r,l=0}^{z} a_{r,l}^e \sum_{q=1}^{q} w_q \det^e(\tilde{x}_q, \tilde{y}_q) \frac{\partial u_i}{\partial x_j} \frac{\partial u_j}{\partial x_i} 
$$

where $q$ is the order of the quadrature rule used in the numerical integration, $(\tilde{x}_q, \tilde{y}_q), g = 1, ..., q$ are the quadrature points with weights $w_q$ and $\det^e(\tilde{x}, \tilde{y})$ is the determinant of the Jacobian matrix $J^e$ of the transformation $Q^e \rightarrow \tilde{Q}$. From (3), this Jacobian is found to be

$$
\begin{pmatrix}
J_{1,1}^e & J_{1,2}^e \\
J_{2,1}^e & J_{2,2}^e
\end{pmatrix} = \begin{pmatrix}
\sum_{i=1}^{k} \frac{\partial u_i}{\partial x} x_i^r & \sum_{i=1}^{k} \frac{\partial u_i}{\partial y} x_i^r \\
\sum_{i=1}^{k} \frac{\partial u_i}{\partial x} y_i^r & \sum_{i=1}^{k} \frac{\partial u_i}{\partial y} y_i^r
\end{pmatrix}.
$$

Because of the regularity of the standard element $Q$, we can easily write the formulas for $\tilde{u}_i(\tilde{x}, \tilde{y})$ and its derivatives $\frac{\partial \tilde{u}_i}{\partial \tilde{x}}$ and $\frac{\partial \tilde{u}_i}{\partial \tilde{y}}$. Then the derivatives $\frac{\partial u_i}{\partial x}$, $r = 1, 2$ and $i = 1, ..., k$ used in (4) may be obtained from

$$
\begin{pmatrix}
\frac{\partial u_i}{\partial x} \\
\frac{\partial u_i}{\partial y}
\end{pmatrix} = \left[J^e\right]^{-T} \begin{pmatrix}
\frac{\partial \tilde{u}_i}{\partial \tilde{x}} \\
\frac{\partial \tilde{u}_i}{\partial \tilde{y}}
\end{pmatrix},
$$

where $\left[J^e\right]^{-T}$ is the inverse of the transposed Jacobian matrix $\left[J^e\right]^T$. This inverse is explicitly assumed to exist.

It should be noted that the quadrature points and weights as well as the basis functions $\tilde{u}_i$, and their derivatives $\frac{\partial \tilde{u}_i}{\partial \tilde{x}}$ and $\frac{\partial \tilde{u}_i}{\partial \tilde{y}}$, do not depend on the specific finite element that is to be processed. Hence, they may be computed at the quadrature points $(\tilde{x}_g, \tilde{y}_g)$, and pre-loaded into the system before it starts its operation which allows for their repeated use during the calculations of $H^e$ for $e = 1, ..., m$.

The following algorithm ALG1 computes the elemental stiffness matrices $H^e$ for $e = 1, ..., m$. (The steps N1 through N5 in the algorithm are partitioned in a manner needed for the description of our systolic system). In this algorithm, we denote by $\nabla^e_i(g)$ the value of the basis function $\tilde{u}_i(\tilde{x}_g, \tilde{y}_g)$ and by $\nabla_i^e(g)$ and $\nabla_i^e(g)$, $r = 1, 2$, the values of its derivatives $\frac{\partial \tilde{u}_i}{\partial \tilde{x}}$, $\tilde{x}_g, \tilde{y}_g$ and $\frac{\partial \tilde{u}_i}{\partial \tilde{y}}$, $\tilde{x}_g, \tilde{y}_g$, respectively.

**ALGORITHM ALG1.**

**INPUTS**

1. $(\nabla^e_i(g), \nabla_i^e(g), \nabla_i^e(g), g = 1, ..., q$ and $i = 1, ..., k$
2. For each finite element $e = 1, ..., m$
   2.1. $(x_i^r, y_i^r), i = 1, ..., k$
   2.2. $a_{r,l}^e, r = 0, 1, 2$ /* note that $a_{r,l}^e = a_{r,l}^e */
For each finite element \( e = 1, \ldots, m \) DO

(N1) Compute the Jacobian of the transformation at each quadrature point \( g = 1, \ldots, q \) from

\[
\begin{pmatrix}
J_{1,1}(g) & J_{1,2}(g) \\
J_{2,1}(g) & J_{2,2}(g)
\end{pmatrix} =
\begin{pmatrix}
\Delta_1(g) & \cdots & \Delta_k(g) \\
\cdots & \cdots & \cdots \\
\Delta_1(g) & \cdots & \Delta_k(g)
\end{pmatrix}
\begin{pmatrix}
x^e_l \\
y^e_l \\
x^e_k \\
y^e_k
\end{pmatrix}.
\]

(N2) For \( g = 1, \ldots, q \) compute the temporary quantities

\[
\begin{pmatrix}
T_{1,1}(g) & T_{1,2}(g) \\
T_{2,1}(g) & T_{2,2}(g)
\end{pmatrix} =
\begin{pmatrix}
J_{1,1}(g) & -J_{2,1}(g) \\
-J_{1,2}(g) & J_{2,2}(g)
\end{pmatrix}
\begin{pmatrix}
\Delta_1(g) & \cdots & \Delta_k(g) \\
\cdots & \cdots & \cdots \\
\Delta_1(g) & \cdots & \Delta_k(g)
\end{pmatrix}
\begin{pmatrix}
x^e_l \\
y^e_l \\
x^e_k \\
y^e_k
\end{pmatrix}.
\]

(N3) For \( g = 1, \ldots, q \) DO

(N3.1) \( \det^*(g) = J_{1,1}(g) J_{2,2}(g) - J_{1,2}(g) J_{2,1}(g) \)

(N3.2) \( \nabla_i^r(g) = (1/\det^*(g)) T_{i,1}(g), r = 1, 2, i = 1, \ldots, k \)

(N3.3) \( \tilde{\nabla}_i^r(g) = w_g \det^*(g) \nabla_i^r(g), r = 0, 1, 2, i = 1, \ldots, k. \)

(N4) For \( i = 1, \ldots, k \) compute the approximate integrals

(N4.1) For \( j = 1, \ldots, i - 1 \)

\[
Y_{r,l}^i = \sum_{g=1}^q \tilde{\nabla}_i^r(g) \nabla_j^l(g) \quad r, l = 0, 1, 2,
\]

(N4.2) \( Y_{r,l}^i = \sum_{g=1}^q \tilde{\nabla}_i^r(g) \nabla_j^l(g) \quad r = 0, 1, 2, l = 0, \ldots, r. \)

(N5) For \( i = 1, \ldots, k \) DO

(N5.1) For \( j = 1, \ldots, i - 1 \),

\[
H_{i,j}^r = \sum_{r = 0}^2 \sum_{l = 0}^2 a_{r,l} Y_{r,l}^i
\]

(N5.2) \( H_{i,i}^r = 2 \sum_{r = 0}^2 \sum_{l = 0}^2 (c_{r,l} a_{r,l}) Y_{r,l}^i \) where \( c_{r,l} \) equals to 1 if \( r \neq l \),

and to 0.5 if \( r = l. \)

Figure 1 shows a block diagram of the systolic system that executes this algorithm. It consists of a local memory LM to store the pre-loaded values of \( \nabla_i^r(g), \Delta_i^1(g), \) and \( \Delta_i^2(g), \) and five systolic subnetworks N1 \( \cdots \) N5 that are arranged in a cascade such that the output of a sub-network is an input for a following sub-network. Each sub-network is designed to perform the computation in the corresponding step in ALG1.
In order to compute the matrix $H^e$ for a certain element $e$, the coordinates of the nodes $(x_i^e, y_i^e)$, $i = 1, \ldots, k$, and the coefficients $a_{i, j}^e$, $r, l = 0, 1, 2$, for that element, are fed to the system via subnetworks N1 and N5, respectively. The entries $H_{i, j}^e$, $i = 1, \ldots, k$, $j = 1, \ldots, i$, of the symmetric matrix $H^e$ are then obtained from the subnetwork N5 after a delay period of $(q + 3k + 16)$ time units, where a time unit is the maximum time needed by any computational cell in the system to perform its operation. This is basically the time required to perform a multiply/add operation, or a division whichever is larger.

The system described in the next section provides a noticeable speedup of order $qk$ over the serial execution of ALG1. However, the real advantage of the system lies in the possibility of pipelining the computations of the stiffness matrices for $e = 1, \ldots, m$, and of obtaining one matrix every $3k$ time units. Of course, we also obtain the advantage of a non-conflicting and smooth data flow in the system which eliminates any delay in execution that might have been caused by complicated interprocess communications or slow memory fetch.

4. Formal Description of the System's Components

In this section, we describe the architecture of the five subnetworks N1,\ldots, N5, that execute the corresponding steps in algorithm ALG1. Our primary goal is to demonstrate the effectiveness of the formal model for a precise specification and verification of systolic networks with computational cells more complicated than those of the simple multiply/add type.

4.1. The Subnetwork N1

The graph of the systolic network N1 is composed of $2q$ interior nodes as shown in Figure 2a; each node is labeled by two integers $(i, g)$, $i = 1, 2$ and $g = 1, \ldots, q$, where $q$ is the number of points used in the numerical integration. The graph also shows the color assigned to each edge, namely $r$, $p$, or $z$. Each node $(i, g)$ represents a computational cell whose operation is described by the causal relations
\[ \begin{align*}
\zeta_{i,g+1} &= \Omega \zeta_{i,g} \\
\rho_{i+1,g} &= \Omega \rho_{i,g} \\
\pi_{i+1,g} &= \Omega^2 M^{3k-21,1}(\pi_{i,g}, \lambda_{i,g}, \xi_{i,g})
\end{align*} \] (5a), (5b), (5c)

where \( s = 1 \) for \( i = 2 \) and \( s = 3 \) for \( i = 1 \), and

\[ \begin{align*}
\lambda_{i,g} &= A^{g+i+k,3}[\rho_{i,g} * \zeta_{i,g}] \\
\xi_{i,g} &= A^{g+i+1+k,3}[\rho_{i,g} * \zeta_{i,g}].
\end{align*} \] (6a), (6b)

The graph in Fig. 2a and Eqs. (5), (6) specify N1 completely. In order to analyze the internal structure of each cell \((i, g)\) more closely, we first note that Eqs. (6) indicate that a cell should contain a multiplier and two accumulators (denoted \( A \) and \( \bar{A} \) in Fig. 2b). The accumulators start operating at times \( g+i \) and \( g+i+1 \), respectively, accumulate the output of the multiplier every third time unit and are reset to zero every 3\( k \) time units. The contents of these accumulators at consecutive time units are expressed by the sequences \( \lambda_{i,g} \) and \( \xi_{i,g} \), respectively. As is clear from Eq. (5c), each cell also contains a multiplexer that starts operating at time \( g+i-1 \) and multiplexes the input \( \pi_{i,g} \) and the contents of the accumulators with a time ratio of \( 3k-2:1:1 \). The delay element \( \Omega^{'s} \) is introduced in Fig. 2b under the assumption that the elements "*", \( A \), and \( M \) do not consume any time. In practical implementations however, these elements do consume some time and consequently the element labeled \( \Omega^{'s} \) has the function of a synchronizer rather than a delay.

In order to perform the calculation in step N1 of ALG1 for a certain finite element \( e, 1 \leq e \leq m \), the \( x \) and \( y \) coordinates of the \( k \) nodes in \( e \) should be supplied on the input links \( z_{1,1} \) and \( z_{2,1} \), respectively. Also, the values of the shape functions

![Fig. 2. (a) The graph for N1. (b) The structure of a cell \((i, g)\).](image-url)
and their derivatives at the quadrature points should be supplied from the local memory on the links \( r_{1,g}, g = 1, \ldots, q \). The precise timing of the inputs is specified by

\[
\begin{align*}
\pi_{1,g} &= \delta^* \\
\zeta_{i,1} &= \Omega^{-1} E_i^{1} \{ \theta^2 \zeta_{i}^* \} \\
\rho_{1,g} &= \Omega^{-1} P^3_k \{ M_{1,1,1}^{1,1,1}(\theta^2 \psi_{g,0}, \Omega \theta^2 \varphi_{g,1}, \Omega \theta^2 \varphi_{g,2}) \} \\
\end{align*}
\]

where

\[
T(\zeta_i^*) = T(\psi_{g,0}) = T(\varphi_{g,1}) = T(\varphi_{g,2}) = k
\]

with \( \zeta_i^*(t) = y_i^r, \zeta_2^*(t) = x_i^r, \psi_{g,0}(t) = \psi_0^r(g), \) and \( \varphi_{g,r}(t) = A_i^r(g), \) \( r = 1, 2. \) In other words, \( \zeta_1^r \) and \( \zeta_2^e \) contain the coordinates of the nodes in the finite element \( e, \) and \( \psi_{g,0}, \varphi_{g,1}, \) and \( \varphi_{g,2} \) contain the shape functions and their derivatives. A pictorial representation of these input sequences in the case \( k = 3 \) and \( q = 3 \) is provided in Fig. 8 using a time diagram in which the elements of the different sequences at consecutive time units are displayed.

We may explain informally the computation performed by the subnetwork by considering first a typical cell \((1, g), 1 \leq g \leq q, \) in its first column. Starting at time \( g, \) any cell \((1, g)\) receives the sequence \( y_1^r, y_2^r, y_1^e, y_2^e, \ldots, \) on the link \( z_{1,g}, \) and the sequence \( \psi_0^r(g), A_1^r(g), A_2^r(g), \psi_0^e(g), A_2^e(g), \psi_0^e(g), \ldots, \) on the link \( r_{1,g}. \) Corresponding elements on these two links are multiplied and the results are sent to the accumulators \( A \) and \( \hat{A} \) for the accumulation of the sums \( J_{1,1} = \sum_{i=1}^{r} A_1^r(g) y_i^r \) and \( J_{2,2} = \sum_{i=1}^{r} A_2^r(g) y_i^r, \) respectively. It is easy to see that these sums are completed at times \( g + 3k - 2 \) and \( g + 3k - 1, \) respectively. Similarly, each cell \((2, g)\) in the second column of \( N_1 \) completes the accumulation of \( J_{1,1} \) and \( J_{1,2} \) at times \( g + 3k - 1 \) and \( g + 3k, \) respectively. The multiplexers in the cells \((1, g)\) and \((2, g)\) then sample the accumulators such that the sequence \( J_{1,1}^r, J_{1,2}^r, J_{2,1}^e, J_{2,2}^e \) appears on \( r_{3,g} \) starting at time \( g + 3k. \) This result may be expressed formally by the following proposition that is proved in the appendix.

**Proposition 1.** If provided with the inputs described by (7), the subnetwork \( N_1 \) executes the step \( N_1 \) in \( ALG 1. \) More specifically, its outputs are described by

\[
\begin{align*}
\rho_{3,g} &= \Omega^g + P^3_k \{ M_{1,1,1}^{1,1,1}(\theta^2 \psi_{g,0}, \Omega \theta^2 \varphi_{g,1}, \Omega \theta^2 \varphi_{g,2}) \}, \quad g = 1, \ldots, q, \\
\pi_{3,g} &= \Omega^g + 3k - \beta_{g}^e, \quad g = 1, \ldots, q,
\end{align*}
\]

where \( T(\beta_g^e) = 4 \) and \( \beta_g^e(t) = J_{1,1}^r(g), J_{1,2}^r(g), J_{2,1}^e(g), \) and \( J_{2,2}^e(g) \) for \( t = 1, 2, 3, \) and \( 4, \) respectively.

**4.2. The Subnetwork \( N_2 \)**

The graph of the subnetwork \( N_2 \) is composed of \( q \) identical rows \( g = 1, \ldots, q \) (see Fig. 3a), where each row consists of three interior nodes \((i, g), i = 3, 4, 5. \) The edges are given the colors \( p, r, s, \) and \( s \) as shown in the figure. The input links of \( N_2 \) are
directly connected to the outputs of N1, and hence the input sequences \( \pi_{3,g} \) and \( \rho_{3,g}, g = 1, \ldots, q \) are described by the formulas (8).

For a given row \( g, 1 \leq g \leq q \), the computation of a cell may be described as follows:

For cells (3, g).

\[ \pi_{4,g} = g_3 \pi_{3,g}, \quad \rho_{4,g} = g_3 \rho_{3,g}, \quad \sigma_{5,g} = g_3 A_{5,1}^1 E_{3}^{3k} \begin{bmatrix} \pi_{3,g} & M_{g+3}^{1,1} E_{g+3, g}^{3k} \end{bmatrix}, \delta^* \]  

For cells (4, g).

\[ \pi_{6,g} = g_3 \pi_{4,g}, \quad \rho_{5,g} = g_3 \rho_{4,g}, \quad \sigma_{5,g} = g_3 A_{5}^1 E_{g+3}^{3k} \begin{bmatrix} -\pi_{3,g} \end{bmatrix}, E_{g+3, g}^{3k}, \delta^* \]  

For cells (5, g).

\[ \gamma_{7,g} = M_{g+1}^{1,1} (g_{5,g}, \sigma_{5,g}, \bar{\sigma}_{5,g}) \]

From the above specifications, it is clear that cells (3, g) and (4, g) have identical structure (see Fig. 3b) and differ only in the reset times of their accumulators, multiplexers, and memories. To reset these elements at the proper time, reset signals can be propagated in the network as explained in Section 2.

Consider a specific row \( g, 1 \leq g \leq q \) of N2. The two memories \( E \) and \( E \) of cell (3, g) store \( -J_{2,1}^g \) and \( J_{2,2}^g \) that appear on the link \( p_{3,g} \) at times \( g + 3k + 2 \) and \( g + 3k + 3 \), respectively. When the sequence \( V_{g,1}^{0}(g), A_{1}^1(g), A_{1}^2(g), V_{g,3}^{0}(g), A_{1}^3(g), A_{3,g}^3(g), \ldots \), appearing on \( r_{3,g} \) starting at time \( g + 3k + 2 \), each \( A_{i}^1(g), 1 \leq i \leq k \), is multiplied by \( J_{2,2}^g \) and each \( A_{3,g}^3(g) \) is multiplied by \( -J_{2,1}^g \). The outputs of the multiplexer are then accumulated in \( A \) which is reset every three time units, and hence contains the values of \( T_{i}^g(g) = J_{2,2}^g A_{i}^1(g) - J_{2,1}^g A_{i}^2(g), \quad i = 1, \ldots, k, \) at times \( g + 3k + 1 + 3i \). Similarly, the accumulator in cell (4, g) contains the values of \( T_{i}^g(g) = J_{1,1}^g A_{i}^1(g) - J_{1,2}^g A_{i}^2(g), \quad i = 1, \ldots, k, \) at times \( g + 3k + 2 + 3i \). Finally, cell (5, g) multiplexes \( V_{g,1}^{0}(g), T_{i}^1(g), \) and \( T_{i}^g(g), \) from the links \( p_{5,g}, s_{5,g}, \) and \( s_{5,g} \), respectively, into the link \( r_{7,g} \).
More specifically, the sequence \( \nabla^0_v(g), T^1_v(g), T^1_v(g), \nabla^0_v(g), T^1_v(g), T^1_v(g), \ldots \) is produced on \( r_{g,g} \) starting at time \( g + 3k + 4 \).

The following proposition describes precisely the output of N2. Its proof is similar to that of Proposition 1 and may be found in [12].

**Proposition 2.** If the input links of N2 are connected to the output links of N1, then N2 performs successfully step N2 in ALG1. More precisely, if the inputs to N2 are described by (8), then its outputs may be described by

\[
\begin{align*}
\pi_{6,g} &= \Omega^g + 3k + 3 \beta^g, \\
\rho_{7,g} &= \Omega^g + 3k + 4 M^{1,1,1}_1(\theta^2 \psi_{g,0}, \Omega \theta^2 \psi_{g,1}, \Omega^2 \theta^2 \psi_{g,2}),
\end{align*}
\]

where \( T(\psi_{g,i}) = k \) and \( \psi_{g,0}(t) = \nabla^0_v(g), \psi_{g,1}(t) = T^1_v(g), \psi_{g,2}(t) = T^2_v(g) \).

4.3. The Subnetwork N3

As in the case of N2, the subnetwork N3 is composed of \( q \) independent, identical rows. Each row performs the calculation corresponding to step N3 in ALG1 for a certain value of \( g, 1 \leq g \leq q \). Due to the variety of possible designs and to the simplicity of the network, we will not describe N3 in any detail. Instead, we will assume that, with the inputs described by (9), N3 takes five time units to complete its computation and to produce for any \( g, 1 \leq g \leq q \) the outputs

\[
\begin{align*}
\pi_{9,g} &= \Omega^g + 3k + 8 \text{M}^{1,1,1}_1(\theta^2 \tilde{v}_{g,0}, \Omega \theta^2 \tilde{v}_{g,1}, \Omega^2 \theta^2 \tilde{v}_{g,2}), \\
\rho_{9,g} &= \Omega^g + 3k + 8 \text{M}^{1,1,1}_1(\theta^2 v_{g,0}, \Omega \theta^2 v_{g,1}, \Omega^2 \theta^2 v_{g,2}),
\end{align*}
\]

where \( \tilde{v}_{g,i}(t) = \nabla^i_v(g), \tilde{v}_{g,i}(t) = \nabla^i_v(g), \) and the values of \( \nabla^i_v(g) \) and \( \tilde{\nabla}^i_v(g) \) are as given in step N3 of ALG1.

4.4. The subnetwork N4

In this subsection, we describe a network that completes the numerical integration by computing the quantities \( Y^i_{g,i} = \sum_{i=1}^{q} \nabla^i_v(g) \nabla^i_v(g) \) for the ranges of the indices in the corresponding step of ALG1. The subnetwork is described by the graph in Fig. 4 and the node I/O descriptions of a typical node \((i, g), 9 \leq i \leq 8 + 3k, 1 \leq g \leq q \), namely,

\[
\begin{align*}
\pi_{i+1,g} &= \Omega^2 \pi_{i,g}, \\
\rho_{i+1,g} &= \Omega \rho_{i,g}, \\
\zeta_{i+1,g} &= \Omega [\zeta_{i,g} + \pi_{i,g} * \rho_{i,g}].
\end{align*}
\]

As this description shows, each cell latches the \( p \) and \( r \) data streams by two and one time units, respectively. It also performs a multiply/add operation and puts the result on the \( z \) output link.
Fig. 4. The graph for N4.
The links $z_{i,1}, i = 9, ..., 3k + 8$ are set permanently to zero and $p_{g,g}$ and $r_{g,g}$ carry the outputs of N4, that is, $p_{g,g}$ and $r_{g,g}$ are described by (10). The data on the $p$ and $r$ colored links are appropriately delayed and multiplied such that the different terms of $Y_{i,i}$ are accumulated on the data stream moving down the $z$ colored links. Unfortunately, it is not at all simple to find an explicit description of the sequences on the output links $z_{i,q+1}, i = 9, ..., 3k + 8$. In order to simplify the equations, we will replace the index $i, 9 \leq i \leq 8 + 3k$ by $i = 9 + 3u + v$, where the indices $u$ and $v$ vary in time.

Fig. 5. The output of N4.
the ranges \( 0 \leq u \leq k - 1 \) and \( 0 \leq v \leq 2 \). More descriptively, we divide the \( 3k \) columns of \( N4 \) into \( k \) groups of 3 columns each. With this notation, we may prove the following proposition [12]:

**PROPOSITION 3.** With the inputs described by (10), the network \( N4 \) produces the results of step \( N4 \) in \( ALG4 \). More specifically, its outputs are described as follows: For \( 0 \leq u \leq k - 1 \) and \( 0 \leq v \leq 2 \),

\[
\eta_{i,j}^{r,l}(t) = Y_{i,j}^{r,l} + \eta_{i,j}^{r,l}(0) + \sum_{\eta_{i,j}^{r,l}(0) = 0}^{3k} M_{1,1,1}^{1,1,1}(\theta_{i,j}^{2}, \theta_{i,j}^{2}, \theta_{i,j}^{2}, \theta_{i,j}^{2}, \theta_{i,j}^{2}, \theta_{i,j}^{2})
\]

where \( \Box \) is a modulo 3 addition and for \( 0 \leq i \leq 2 \), we have

\[
T(\eta_{i,j}^{r,l}) = k - u \quad \text{if} \ r \leq l, \quad \text{and} \quad \eta_{i,j}^{r,l}(t) = Y_{i,j}^{r,l} + u \quad \text{if} \ r > l.
\]

Equation (11) shows that the output sequences \( \zeta_{u,v,q+1} \) contain the results of the numerical integration needed for the calculation of the stiffness matrices in the next subnetwork \( N5 \). It also specifies precisely the time of each output data item. In Fig. 5, this specification is translated into a time diagram, where we plot the elements of \( \zeta_{u,v,q+1} \) versus time for the special case \( k = q = 3 \).

4.5. The Subnetwork \( N5 \)

Figure 6 shows the graph for \( N5 \), which is composed of three different rows. The edges of the graph are given the colors \( p, r, s, b, z, z', z'', z''', z'''', z'''' \), where we used three different colors \( z', z'', z''' \) to satisfy the restriction that no two edges ending at a node have the same color.

We consider first row \( q+1 \) which contains \( 3k \) identical nodes. It receives the constants \( a_{i,j}^{r,l} \) on the links \( a_{i,j}^{r,l} + 1 \), \( r_{i,j}^{r,l} + 1 \), and \( s_{i,j}^{r,l} + 1 \) and distributes them appropriately on the \( b \) colored links such that each integral \( Y_{i,j}^{r,l} \) appearing on a \( z \)-colored link meets the corresponding constant \( a_{i,j}^{r,l} \) at the right time. More precisely, each cell \( (i, q+1) \) in that row latches the four data streams \( z, p, r, s \) by one time unit, and selects the output on the \( b \) link to be

\[
\beta_{i,q+2} = \Omega [h_i \cdot \pi_{i,q+1} + 1] \quad \text{if} \ i = 9 + 3u, u = 0, \ldots, k - 1,
\]

\[
= \Omega p_{i,q+1} \quad \text{if} \ i = 9 + 3u + 1, u = 0, \ldots, k - 1,
\]

\[
= \Omega q_{i,q+1} \quad \text{if} \ i = 9 + 3u + 2, u = 0, \ldots, k - 1,
\]

where \( h_i = 0.5 \) for \( i = 9 \) and \( h_i = 1.0 \) for \( i > 9 \). The factor 0.5 is needed to implement step \( N5.2 \) in \( ALG1 \), where only the \( Y_{i,j}^{r,l} \), \( l \leq r \) are explicitly available for the computation of \( H_{i,j}^{r,l} \), while we have \( Y_{i,j}^{r,l} = Y_{i,j}^{r,l} \) for \( l > r \).
ANALYSIS OF SYSTOLIC SYSTEMS

Fig. 6. The subnetwork NS.
For the proper operation of the system the input sequences should be described by
\begin{align}
\pi_{9,q+1} &= \Omega^{q+3k+9} P_k^1(\sigma_q^r) \quad (12a) \\
\rho_{9,q+1} &= \Omega^{q+3k+9} P_k^2(\sigma_q^r) \quad (12b) \\
\sigma_{9,q+1} &= \Omega^{q+3k+9} P_k^3(\sigma_q^r) \quad (12c)
\end{align}
where for \( j = 0, 1, 2 \), \( T(\sigma_q^r) = 3 \) and \( \sigma_q^r(t) = \sigma_q^{r-1} \square_{2/(r-1)} \), with \( \square \) denoting the modulo 3 addition operation. More descriptively, we input on each line three of the constants \( \sigma_q^r, r, l = 0, 1, 2 \), repeated \( k \) times as indicated by the piping operator \( P_k^1 \) (for more details see Fig. 8).

The \( 3k \) cells \((i, q + 2), 9 \leq i \leq 8 + 3k, \) in row \( q + 2 \) have basically the same structure: each is a multiplier/adder equipped with a demultiplexer that distributes the results to the output links \( \rho_{i+1, q+2} \) and \( \zeta_{u,q+3} \) (see Fig. 7 where \( u \) and \( v \) equal the quotient and the remainder of \((i-9)/3\), respectively). Formally, the operation of each cell \((i, q + 2)\) is described by
\begin{align}
\rho_{i+1, q+2} &= \Omega^2 M_{i+q+3k+1}(\rho_{i,q+2} + [\beta_{i,q+2} \star \zeta_{i,q+2}]) \\
\zeta_{u,q+3} &= \Omega M_{i+q+3k+1}(\rho_{i,q+2} + [\beta_{i,q+2} \star \zeta_{i,q+2}]),
\end{align}
where the input \( \rho_{9,q+1} \) is permanently set to the zero sequence 1. The function of this row is to compute the partial sums \( U_{i,j} = \sum_{i=0}^{2} \sigma_i Y_{i,j}^r \) and \( \sum_{i=0}^{2} (\sigma_i Y_{i,j}^r) Y_{i,j}^r \) for \( i \neq j \) and \( i = j \), respectively, where \( \sigma_i \) is as given in ALGl.

Finally, each group of three sequences \( \zeta_{u,q+3}^0, \zeta_{u,q+3}^1, \) and \( \zeta_{u,q+3}^2 \) is considered as input to a cell \((u, q + 3), 0 < u \leq k - 1, \) in row \( q + 3 \) of N5. The function of this cell is to accumulate the partial results appearing on its input links. That is, to compute the sums \( H_{i,j} = \sum_{r=0}^{2} U_{i,j}^r \). The operation of each cell in this row is described formally by
\begin{align}
\zeta_{u,q+4} &= \Omega [c_u \cdot A^{6u+3k+q+11,3,1} M_{6u+3k+q+11}(\zeta_{u,q+3}^0, \zeta_{u,q+3}^1, \zeta_{u,q+3}^2)],
\end{align}
where \( c_u \) equals 2 and 1 for \( u = 0 \) and \( u > 0 \), respectively.

---

**Fig. 7.** A typical cell in row \( q + 2 \) of N5.
Given these specifications, the following result may be proven [12]:

**Proposition 4.** If the inputs to the network \( N5 \) are given by (11) and (12), then the network's output sequences are given by

\[
\begin{align*}
\zeta_{w,q} + 4 &= \Omega^{6w + 3k + q + 16\theta^2\tilde{\mu}_w} \quad u = 0, \ldots, k - 1. \\
\end{align*}
\]  

(13)

where \( T(\tilde{\mu}_w) = k - u \) and \( \tilde{\mu}_w(t) = H_{t,t+u} \).

\[ \text{FIG. 8. The input and output sequences.} \]
Proposition 4 states that after an initial time period of $6u + 3k + q + 16$ units, each output link $z_{u,q+4}$ will carry the elements of the $u$th off-diagonal of the stiffness matrix $H^e$, separated from each other by 2 time units.

The behavior of the entire system is summarized in Fig. 8, where we show a time diagram of the data on all the input and output links of the global system. This represents a translation of the sequence equations (7), (12), and (13) for the special case $k = q = 3$. The data items in the input sequences $ζ_1, ζ_2, π_{g,q+1}, ρ_{g,q+1},$ and $σ_{g,q+1}$ depend on the finite element that is being processed and hence they must be provided from outside the system. On the other hand, the data in $ρ_{1,q}, g = 1,..., q$ do not depend on a particular finite element and thus, as mentioned in Section 3, they are provided from a memory local to the system.

4.6. Pipelined Operation

In the previous subsections, we considered only the computation of one elemental matrix and we showed that this computation may be completed in $9k + q + 10$ time units (see (13)). The issue of pipelining the computation of the different elemental matrices on the system was studied in [12], where it was shown that 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 every $3k$ time unit.” More specifically, if the data on the input links are described by

$$ζ_{i,1} = Ω^{i-1}P_{e=1,m}^{3k}(E^i_1θ^2ζ^e_i)$$
$$π_{g,q+1} = Ω^{q+3k+9}P_{e=1,m}^{3k}(P^1_k(ζ^e_0))$$
$$σ_{g,q+1} = Ω^{q+3k+9}P_{e=1,m}^{3k}(P^2_k(ζ^e_1))$$
$$ρ_{g,q+1} = Ω^{q+3k+9}P_{e=1,m}^{3k}(P^3_k(ζ^e_2))$$

then it may be proved formally that the sequences on the output links are

$$ζ_{u,q+4} = Ω^{6u + 6u + 3k + 16}P_{e=1,m}^{3k}(θ^2μ^e_u)$$

where $ζ^e_i, x^e_i,$ and $μ^e_u$ are as described in (7), (12), and (13). In other words, the computation of the $m$ elemental matrices will be completed after $(3m + 6)k + q + 10$ time units.

It is not surprising that using a fixed amount of hardware to pipeline the computation results in a speed up which is not dependent on the size of the problem. Another approach for speeding finite element analysis would be to use a hardware proportional to the size of the problem for processing the different elements in parallel. However, more research is needed in order to extract the parallelism across the elements without imposing restrictions on the domain of the problem or on the grid that covers this domain.
5. Concluding Remarks

In this paper, we demonstrate the flexibility and power of the systolic model presented in [13] by applying it to networks with memory and multiplexing capabilities. More specifically, new sequence operators were defined to model periodic memories, accumulators, and multiplexers. The extended model was then used for the specification and formal verification of a systolic system that can pipeline the computation of the elemental stiffness matrices. This system is a clear indication that the systolic architecture may be applied to complex computations, provided that we allow for computational cells slightly more complicated than the simple inner product type.

Although the abstract model has been used here to specify the architecture at the level of the computational cells, the same model can also be used for lower or higher levels of architectures provided we define appropriately the domain $R_S$ of the data items that are transmitted on the communication links of the network, and the corresponding operators.

It was proved in [12] that the solution of any system of causal equations does exist and may be obtained analytically. However, although there were no difficulties in establishing analytical proofs for the operation of the different components of our system, analytical verification of systolic networks may be sometimes complicated, especially in the presence of feedback loops. As a means for alleviating this problem, a computer program was developed [12] that solves iteratively any system of consistent causal equations for specific input sequences. The function of this solver is that of a simulator, and hence it may be used only in the verification of particular instances of computations rather than for the general verification of the network's operation.

Besides its value in demonstrating the power of the systolic model, the system that generates the elemental stiffness matrices appears to have merit of its own. Namely, it is a contribution to the design of an integrated finite element machine. Possible configurations for such a pipelined/systolic machine are discussed in [14], where a specific node numbering scheme and a corresponding frontal technique are suggested in order to allow the generation, assembly and solution phases of the analysis to execute in parallel. Here we note that the number of operations in the generation and the solution phases are of order $O(mk^2)$ and $O(nB^2)$, respectively, where $m$ is the number of elements, $k$ is the number of nodes within each element, $n$ is the total number of nodes and $B$ is the bandwidth of the stiffness matrix. Usually, $mk^2$ is only a fraction of $nB^2$. However, for realistic structure problems, the cost of the generation phase may be larger than the cost of the solution phase due to the irregularity of the computations in the former.

It should be clear that alternative designs do exist for the different units of the system. However, the ones presented in this paper were chosen for the following reasons: (1) The smooth flow of data between the units of the system, thus eliminating any external intervention that might be needed for buffering or rerouting the data. (2) The ability of pipelining the computations associated with
the different elements on the system without the need for any resetting or delay between successive computations. (3) The possibility of modifying the system to take advantage of the cases where the coefficients $a_{i,j}^r$ are equal to zero for $r = 0$ or $l = 0$ (e.g., flow problems). This requires only the change of some control parameters and results in a system that is 1.5 times faster than the current one. (4) The high utilization of the resources relative to other parallel systems. In fact, the average ratio of the number of cycles during which the cells in our system are doing useful work to the total number of cycles during execution is larger than 50%. (5) The simplicity of extending the system to cover problems with $d > 1$ degrees of freedom, where each $a_{i,j}^r$ is a $d \times d$ matrix and consequently each entry $H_{i,j}^r$ in $H^r$ is a $d \times d$ submatrix. In this case, we may replace the unit $N5$ with $d^2$ identical units, each of which generates the corresponding entry in $H^r$ when provided with the appropriate entry in $a_{i,j}^r$.

On the other hand, the use of simple cells in the design resulted in some restrictions on the maximum number of nodes $k$ allowed in each finite element and on the degree $q$ of the quadrature formula used in the numeric integration. Also, the extension of the system to cover nonlinear problems and problems with variable coefficients does not seem trivial, if possible at all.

Finally, it may seem that the correctness of systolic networks can be trivially established without any formal analysis. Although this may be true for simple networks, it is obviously not so for complex networks where many parameters have to be adjusted simultaneously, as, for example, the set times and the periods of the accumulators, memories, and multiplexers. In fact, the formal analysis was essential in the design of the system presented in this paper. Namely, it led to the rejection of many networks that seemed correct but were incorrect because of some wrong timing. Moreover, the construction of the correctness proofs helped in making the right choices of the different parameters of the system. We should also note that the formalization of the analysis of large computational networks is a first step toward the development of an automatic synthesizer for such networks.

**APPENDIX**

In this Appendix, we will prove Proposition 1 of Section 4.1. The proof of the other propositions included in this paper follow the same techniques and may be found in [12]. We start by listing few properties about combinations of the different sequence operators. These properties are directly verifiable from the definition of the operators and are very useful in simplifying the manipulation of the sequence expressions.

(P1) For any element-wise operator "op" with $\delta \ "op" \ \delta = \delta$ we have

\[
(1.1) \quad \Gamma(\xi) \ "op" \ \Gamma(\eta) = \Gamma(\xi \ "op" \ \eta)
\]
ANALYSIS OF SYSTOLIC SYSTEMS

\[(P2) \quad \Omega^r M_{r}^{i,...,m}(\xi_{1},...,\xi_{n}) = M_{r+1}^{i,...,m}((\Omega^{r} \xi_{1},...,\Omega^{r} \xi_{n})), \text{ for any } r \text{ and } s > -r\]

\[(P3) \quad A_{r,k,s}^{u} \xi = \Omega_{u}^{r} A_{r-u,k,s} \xi, \text{ for } u < r\]

\[(P4) \quad A_{r,k,s}^{1,1} = E_{1}^{r} A_{r+1,k,s}^{1,1}\]

\[(P5) \quad M_{r-k}^{k,n-m,1,...,1}(\xi_{1},...,\xi_{n}), \eta_{1},...,\eta_{m}) = M_{r+n-m,1,...,1}(\xi_{1},...,\xi_{n}), \eta_{1},...,\eta_{m},\]

\[(P6) \quad A_{r,k,s}^{i,1} M_{i+1}^{i,...,1}(\xi_{1},...,\xi_{s}) = A_{r,k,s}^{i,1} \eta_{1,1}, \text{ for } 1 < r < s.\]

**Proof of Proposition 1.** We start by solving the system of Eqs. (5) to obtain the network I/O description. That is, a relation between the network output sequences \(\rho_{3,g}, \pi_{3,g}, g = 1, \ldots, q\), and the network input sequences \(\zeta_{i,1}, \pi_{1,g}, \rho_{1,g}, i = 1, 2, g = 1, \ldots, q\). Obviously, the solutions of (5a), (5b), are

\[\zeta_{i,g} = \Omega^{i-1} \zeta_{i,1} \quad \text{and} \quad \rho_{i,g} = \Omega^{i-1} \rho_{1,g}.\] (14)

Then, from (5c) and (6), we obtain for \(g = 1, \ldots, q\) that

\[\pi_{3,g} = \Omega M_{g+1}^{2,1,1}(\pi_{2,g}, \lambda_{2,g}, \bar{\lambda}_{2,g}) = \Omega M_{g+1}^{2,1,1}(\Omega^{3} M_{g}^{2,1,1}(\pi_{1,g}, \lambda_{1,g}, \bar{\lambda}_{1,g}), \lambda_{2,g}, \bar{\lambda}_{2,g})\]

where \(\lambda_{i,g}\) and \(\bar{\lambda}_{i,g}\) are given by

\[\lambda_{i,g} = A_{r,i+1,k,3}[\Omega^{i-1} \rho_{1,g} \bullet \Omega^{i-1} \zeta_{i,1} ] \quad \text{(15a)}\]

\[\bar{\lambda}_{i,g} = A_{r,i+1,k,3}[\Omega^{i-1} \rho_{1,g} \bullet \Omega^{i-1} \zeta_{i,1} ] \quad \text{(15b)}\]

Using property (P2), we may rewrite

\[\pi_{3,g} = \Omega M_{g+1}^{2,1,1}(M_{g+2,1,1}(\Omega^{3} \pi_{1,g}, \Omega^{3} \lambda_{1,g}, \Omega^{3} \bar{\lambda}_{1,g}, \Omega^{3} \bar{\lambda}_{1,g}), \lambda_{2,g}, \bar{\lambda}_{2,g}).\]

Finally, by applying property (P5) we obtain for \(1 \leq g \leq q\).

\[\pi_{3,g} = \Omega M_{g+3}^{4,1,1,1}(\Omega^{3} \pi_{1,g}, \lambda_{2,g}, \bar{\lambda}_{2,g}, \Omega^{3} \lambda_{1,g}, \Omega^{3} \bar{\lambda}_{1,g}). \quad \text{(16a)}\]

This, together with

\[\rho_{3,g} = \Omega^{2} \rho_{1,g} \quad \text{(16b)}\]

form the network I/O description for \(N1\). Next, we substitute the specific inputs (7) into (16) in order to prove the results (8) of the proposition. The proof of (8a) follows directly from (16b) and (7c). To prove (8b), we first note that the operator \(P_{2,k}^{3}\) in (7c) indicates that the first \(3k\) elements of the argument are repeated twice in \(\rho_{1,g}\). This repetition is only necessary for the operation of the subnetwork \(N2\), and
will not be considered here. Hence, we will replace the last $3k$ elements of the repetition by don’t care elements, which reduces (7c) to

$$\rho_{1,g} = \Omega^{g-1}M^{1.1.1}_{1}(\theta^{2}\psi_{g,0}, \Omega\theta^{2}\phi_{g,1}, \Omega^{2}\theta^{2}\phi_{g,2}).$$  \tag{7e}

Now substitution of (7a), (7b), (7c) into (16a) results in

$$\pi_{3,g} = \Omega M^{3k-4.1.1.1}_{g+3}(\delta^{*}, \lambda_{2,g}, \pi_{2,g}, \Omega^{3}\lambda_{1,g}, \Omega^{3}\pi_{1,g}).$$  \tag{17a}

Here by (15a) and the definition of the $E$ operator and properties (P1) and (P3) we find that

$$\lambda_{i,g} = A^{g+i.3}[\Omega^{g+i-2}M^{1.1.1}_{1}(\theta^{2}\psi_{g,0}, \Omega\theta^{2}\phi_{g,1}, \Omega^{2}\theta^{2}\phi_{g,2}) * \Omega^{g+i-1}E^{2}
\theta^{2}\xi_{t}^{r}]
= \Omega^{g+i-2}A^{2.3.3}M^{1.1.1}_{1}(\theta^{2}[\psi_{g,0} * \xi_{t}^{r}], \Omega\theta^{2}[\phi_{g,1} * \xi_{t}^{r}], \Omega^{2}\theta^{2}[\phi_{g,2} * \xi_{t}^{r}])$$

and by (P6) and (P3) that

$$\pi_{i,g} = \Omega^{g+i-1}A^{1.k.3}\theta^{2}[\phi_{g,1} * \xi_{t}^{r}].$$  \tag{17b}

Similarly, we can show that

$$\pi_{i,g} = \Omega^{g+i}A^{1.k.3}\theta^{2}[\phi_{g,2} * \xi_{t}^{r}].$$  \tag{17c}

For a further simplification of Eqs. (17a), we consider the definition of the multiplexer operator with the restrictions (7d) on the involved sequences. This gives for $g = 1, \ldots, q$,

$$\pi_{3,g} = \Omega^{g+3k-1}\beta_{g}^{e},$$

where $T(\beta_{g}^{e}) = 4$ and

$$\beta_{g}^{e}(t) = \begin{cases} 2_2,g(g + 3k - 1) & \text{for } t = 1, \\ 2_2,g(g + 3k) & \text{for } t = 2, \\ 2_1,g(g + 3k - 2) & \text{for } t = 3, \\ 2_1,g(g + 3k - 1) & \text{for } t = 4. \end{cases}$$

Moreover, from (P4), the definitions of the shift, spread, and accumulator operators and from (7b), we obtain that

$$\lambda_{2,g}(g + 3k - 1) = \begin{cases} 1^{2,1}g(g + 3k - 1) & \text{for } t = 1, \\ 1^{2,1}g(g + 3k) & \text{for } t = 2, \\ 1^{1,1}g(g + 3k - 2) & \text{for } t = 3, \\ 1^{1,1}g(g + 3k - 1) & \text{for } t = 4. \end{cases}$$

By a similar argument, it can be shown that $\beta_{g}^{e}(2), \beta_{g}^{e}(3), \text{ and } \beta_{g}^{e}(4)$ are equal to $J_{1,2}^{e}(g), J_{2,1}^{e}(g), \text{ and } J_{2,2}^{e}(g)$, respectively, which proves the proposition. \qed
ACKNOWLEDGMENT

I would like to gratefully acknowledge the helpful discussions that I had with Werner Rheinboldt throughout the course of this research. I would like also to thank the referees for their suggestions that helped in improving the presentation of the paper.

REFERENCES