ON MAPPING CUBE GRAPHS ONTO LINEAR SYSTOLIC ARRAYS

I. V. Ramakrishnan

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-219 March 1983

ON MAPPING CUBE GRAPHS ONTO LINEAR SYSTOLIC ARRAYS[1]

I.V.RAMAKRISHNAN

DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN
AUSTIN, TEXAS 78712

ABSTRACT

Systolic algorithms for several important computational problems have been proposed for execution on two dimensional arrays (rectangular or hexagonal mesh). However, practical considerations render linear systolic arrays more useful than systolic arrays with higher connectivity in the context of existing computer systems. This paper identifies the structure of a large class of systolic algorithms for two-dimensional processor arrays and proposes a general methodology for mapping such algorithms onto linear arrays.

[1]This research was supported in part by the National Science Foundation under Grants MCS-8104017 and ONR Contract N00014-80-k-0987

1 Introduction

In [6, 7, 10] systolic arrays were proposed as a means of handling compute-bound problems in a cost-effective and efficient manner. Systolic architectures generally consist of a regular array of simple, identical processing elements which operate in synchrony. The processor array can be of many forms, for instance a linear array, a rectangular mesh, a hexagonal mesh, etc. These architectures are well suited for VLSI implementation.

An algorithm executing on a systolic array comprises of several data streams. Elements in distinct data streams move at different velocities (processors / cycle) while all elements in a given data stream move at the same velocity. Every processor in the array regularly receives data from each of the data streams, performs some short computation and pumps the data out. The array has several input/output ports through which external communication takes place, i.e., elements in the data streams are pumped in through these input ports and results of the computation are retrieved through the output ports.

A basic goal of systolic architectures is to achieve more computations per time unit from an existing system through the addition of an array of simple and identical processing elements. A host computer drives the array as a peripheral. No major changes to the existing system architecture should be required; thus the existing memory bandwidth should remain constant and the device should be interfaced to an existing system bus.

The constraints imposed on the structure of systolic arrays by the above goals make linear array processors the most useful class of systolic arrays. Algorithms for a number of important computational problems fit very naturally onto a linear processor array and many such algorithms have been designed [1, 3, 4, 5, 8, 11], and a few general methodologies have been proposed for designing them [2, 15]. However, algorithms for a number of other important computational problems fit naturally onto two dimensional array processors (rectangular or hexagonal mesh) rather than a linear array. These include algorithms for multiplication and LU-decomposition of matrices [6], relational database operations [8], signal processing operations [9], etc. Such algorithms are more complex in structure precluding a straightforward extension of existing methodologies for designing linear-array systolic algorithms.

This paper proposes a systematic methodology to map a class of algorithms that naturally fit on two-dimensional systolic arrays onto linear systolic arrays. The paper is organized as follows. In section 2 we briefly describe the linear systolic array model. In section 3 we provide a formal definition for programs that naturally fit onto two

dimensional systolic arrays. In section 4 we provide a methodology to map such programs onto linear systolic arrays, and in section 5 we illustrate the methodology by devising new algorithms for matrix multiplication on linear arrays.

2 Linear Systolic Array

A linear systolic array is a 4-tuple $Ar=<N,L_{Ar},\Psi_{Ar},S_{Ar}>$ as follows.

1. N is a sequence of identical processors with indices ranging from 0 to |N|-1.

2. $L_{Ar}=\{l1, l2, .., lk\}$ is a set of labels.

3. Every processor in the array has k input ports and k output ports, with each input port and output port assigned a unique label lj from L_{Ar}. Each processor in N is connected to its neighbors in the sequence through its I/O ports. In addition the first and last processors may have input and output ports connected to the host environment.

4. The array is driven by a single-phase global clock. In every clock cycle each processor computes a k-ary function Ψ_{Ar}.

5. The control unit of every processor is a finite state machine having a single state S_{Ar}. Every processor in the array is in S_{Ar} at the beginning of a clock cycle and returns to it at the end of the clock cycle. Consequently the processors in the array do not have any decison-making ability.

We will henceforth refer to a processor in the array by its index in the sequence N. Let s be the index of a processor. Let $si_t=<si_t^1,si_t^2,..,si_t^k>$ denote the k-tuple input to processor s at time t where si_t^j is the value at the input port labelled lj of processor s at time t. Let $so_t=<so_t^1,so_t^2,..,so_t^k>$ denote the k-tuple output computed by processor s at time t, i.e., $\Psi_{Ar}(si_t)=so_t$.

The linear systolic array has the following communication features.

1. A neighborhood constant n_{lj} is associated with every label lj in L_{Ar} such that for any processor s its output port labelled lj is connected to the input port of processor $s+n_{lj}$. A processor in a linear array can only communicate with its two neighbors and itself, and hence n_{lj} is one of $\{1,-1,0\}$.

2. The elements in a data stream move at constant velocity and hence a delay constant d_{lj} is associated with every label lj in L_{Ar} such that for any processor s, if so_t is the output computed by s at time t, then so_t^j appears at the input port labelled lj of processor $s+n_{lj}$ at $t+d_{lj}$.

3. External communication takes place through certain designated input/output ports namely,
   a. if $n_l=1$ then the input port labelled l of processor 0 and the output port labelled l of processor $|N|-1$ communicate with the host,

   b. if $n_l=-1$ then the input port labelled l of processor $|N|-1$ and the output port labelled l of processor 0 communicate with the host and

   c. if $n_l=0$ then a simple register in every processor serves as the input/output port labelled l. No input/ouput ports labelled l communicate with the host. A value is preloaded into this register before starting the computation and the result value (the preloaded value may be updated as computation progresses) is retrieved from this register after the computation terminates.

Lemma 2-1: Let k be some integer. Let n_l and d_l be the neighborhood and delay constants respectively of label l. If x is the value at the input port labelled l of processor s at time t then it will reach the input port labelled l of processor $s+k\times n_l$ ('+' and '$\times$' denote arithmetic plus and multiplication operators) at time $t+k\times d_l$.

Proof: Immediate consequence of the communication features of the linear array. □

[Note : Depending on the function computed by the processors in the array, the value x may change when it reaches the input port labelled l of processor $s+k\times n_l$]

3 Program Model

A program $G=<V,E,L_G>$ is a labelled DAG where

1. $V=V_G \cup SO_G \cup SI_G$, where V_G, SO_G and SI_G are three disjoint sets of vertices with SO_G the set of source vertices ,SI_G the set of sink vertices and V_G the set of remaining vertices, which we shall call computation vertices,

2. L_G is a set of labels. Let $|L_G|=k$, and

3. every vertex in V_G has k incident edges and k outgoing edges, where each incident and outgoing edge is assigned a unique label from L_G.

[Note: We will assume that G is connected]

Input edges and output edges in G are those edges that are directed out of and into source and sink vertices respectively.

In any execution of G on a linear systolic array, every computation vertex in G is a single instance of a function evaluation that is performed in a cycle by a processor in the array. Hence we can view the k input edges and the k output edges of a vertex v_x as representing the k-tuple input value and k-tuple output value computed by the processor when v_x is evaluated by the processor.

4 Mapping Programs on Linear Systolic Arrays

Intuitively, mapping is an assignment of computation vertices of G to processors in Ar at particular times. Let $T=\{0,c,2c,..\}$ be a sequence of time steps where c is the clock time of the global clock that drives the linear array. A mapping of G onto a linear array Ar is a 2-tuple $<PA,TA>$ where $PA:V_G->N$ and $TA:V_G->T$ are many-one functions mapping computation vertices onto processors and time steps respectively. Let Ψ be the function represented by any computation vertex in G.

A mapping is syntactically correct iff $\Psi=\Psi_{Ar}$, $L_G=L_{Ar}$ and the communication features of the linear array are preserved. Hence

1. for any label $l \in L_{Ar}$, if there is an edge labelled l directed from v_x to v_y then $PA(v_y)=PA(v_x)+n_l$ and $TA(v_y)=TA(v_x)+d_l$, and

2. since the processors do not have any decison-making ability, no two input/output values can appear at the same input port of a processor at the same time.

A program is correctly executed on a linear array iff

1. the mapping is syntactically correct, and

2. for any computation vertex v_x that is mapped on to a processor s, if the vertex v_x has an input edge(output edge), then the value denoted by the input edge (output edge) must be kept invariant until it reaches s from the

port of external communication (until it reaches the port of external communication from s).

5 Cube Graphs

In this section we provide a formal definition of program graphs that naturally fit onto two-dimensional systolic arrays.

Definition 5-1: For any label l in L_G, a major path labelled l is a directed path from a source vertex to a sink vertex such that the label of all the edges in the path is l.

We will refer to the value represented by a source (sink) vertex or input (output) edge as input (output) value.

Definition 5-2: Let $G=<V,E,L_G>$ be a program graph with its label set $L_G=\{l1,l2,l3\}$. Then, G is a Cube Graph iff there exists a one-one function $F:V_G \rightarrow I_1 \times I_2 \times I_3$ where:

1. V_G is the set of computation vertices in G and I_1, I_2 and I_3 are sequences of integers ranging from 0 to (h_1-1), 0 to (h_2-1) and 0 to (h_3-1) respectively,

2. F_{l1}, F_{l2} and F_{l3} are three projection functions of F, i.e., if $F(v_x)=<c_1,c_2,c_3>$ then $F_{l1}(v_x)=c_1$, $F_{l2}(v_x)=c_2$ and $F_{l3}(v_x)=c_3$,

3. for any label $l \in L_G$ and for any v_x and v_y in V_G, there exists a major path labelled l passing through v_x and v_y such that the distance from v_x to v_y is d iff $F_l(v_y)=F_l(v_x)+d$ and $\forall t \in L_G-\{l\}$, $F_t(v_y)=F_t(v_x)$.

A Cube Graph is an object in Euclidean 3-Space and we will refer to the 3 axes as $l1^{th}$, $l2^{nd}$ and $l3^{rd}$ axes. h_1, h_2 and h_3 are the maximum dimensions along $l1^{th}$, $l2^{nd}$ and $l3^{rd}$ axes respectively. If v_x is a computation vertex in a Cube Graph then we will refer to $F_{l1}(v_x)$, $F_{l2}(v_x)$ and $F_{l3}(v_x)$ as $l1^{th}$, $l2^{nd}$ and $l3^{rd}$ coordinate respectively and denote them by x_{l1}, x_{l2}, and x_{l3} respectively.

Let $H=\{1,-1\} \times \{1,-1\} \times \{1,-1\}$ be the cartesian product of the set $\{1,-1\}$. Let $w=<w_1, w_2, w_3> \in H$.

Definition 5-3: A Diagonalization of a Cube Graph is a pair $<D,w>$ with the following properties.

1. $D=\{D_1, D_2, .., D_k\}$ is a family of ordered sets of computation vertices and $D_1 \cup D_2 \cup .. \cup D_k = V_G$.

2. For any D_p in D, if v_x and v_y are in D_p then $w_1 \times x_{l1} + w_2 \times x_{l2} + w_3 \times x_{l3} = w_1 \times y_{l1} + w_2 \times y_{l2} + w_3 \times y_{l3}$.

3. Let T_D denote the indexing function associated with the ordered set D. For any pair of D_p and D_q in D, if v_x and v_y are in D_p and D_q respectively then $T_D(D_p) < T_D(D_q)$ iff $w_1 \times x_{l1} + w_2 \times x_{l2} + w_3 \times x_{l3} < w_1 \times y_{l1} + w_2 \times y_{l2} + w_3 \times y_{l3}$.

We will refer to w as the Diagonalization Factor of the Cube Graph. Let w_p denote the weight of the diagonal D_p in D, i.e., if v_x is a vertex in D_p then $w_1 \times x_{l1} + w_2 \times x_{l2} + w_3 \times x_{l3} = w_p$.

Definition 5-4: For any pair of diagonals D_p and D_q in D, D_q is the immediate successor of D_p iff there exists no diagonal D_r in D such that $w_p < w_r < w_q$.

Let *succ*(D_p) denote the immediate successor D_q of D_p. The assignment of indices to the diagonals in D is done as follows.

1. Assign index 1 to the diagonal with the least weight.

2. For any diagonal D_p, if D_p is assigned index i then aasign index i+1 to *succ*(D_p).

Henceforth we will be assuming the following:

1. G will refer to a Cube Graph and $l1$, $l2$ and $l3$ will refer to the three labels in its label set L_G.

2. The subscript of a diagonal will refer to its index, i.e., if D_p is a diagonal in D then its index is p.

Definition 5-5: A Mesh Graph is a Cube Graph with $|L_G|=2$, i.e., cardinality of the label set is 2.

Let $l \in L_G$. Let $MG=\{MG_1, MG_2, .., MG_h\}$ be the disconnected components formed by removing all the edges labelled l from G. Clearly, for any MG_i in MG, the label set MG_i is $L_G - \{l\}$.

Theorem 5-1: MG_i is a Mesh Graph.

Proof: Follows immediately from definitions of Mesh and Cube Graphs.

□

We next combine the disconnected components in MG into classes as follows.

Let $SG=\{SG_1, SG_2, .., SG_n\}$ be a family of sets of disconnected components such that $SG_i=\{MG_q \mid$ if v_x is a computation vertex in MG_q then $F_l(v_x)=i\}$ (i.e., components in the set SG_i have the property that the l^{th} coordinate of the computation vertices in these components is i).

[Note: F_l is F_{l1} if $l=l1$ or F_{l2} if $l=l2$ or F_{l3} if $l=l3$. Aslo the l^{th} coordinate is $l1^{th}$ coordinate if $l=l1$ or $l2^{nd}$ coordinate if $l=l2$ or $l3^{rd}$ coordinate if $l=l3$].

6 Mapping Algorithm

We now describe the algorithm to map a Cube Graph onto a linear array . Let $Ar=<N,L_{Ar},\Psi_{Ar},S_{Ar}>$ denote the linear array onto which G is mapped. Let $SG=\{SG_1, SG_2, .., SG_n\}$ be the family of sets of Mesh Graphs formed by removing all the edges that are labelled l. Without loss of generality, let $l=l3$. So the label set of any mesh graph within any set in SG is $\{l1, l2\}$. Let Ψ denote the function represented by a computation vertex in G.

Choose some Diagonalization Factor $w=<w_1, w_2, w_3>$ from H. Let D be the set of diagonals obtained for this w. Let $|D|=m$. Choose the number of processors in N to be m, i.e., let $|N|=|D|=m$. Let $\Psi_{Ar}=\Psi$ and $L_{Ar}=L_G$. Let $D=\{D_1, D_2, .., D_m\}$ denote the ordered set of diagonals in D and let $\{1,2,..,m\}$ denote the sequence of processor numbers in N.

We are now in a position to describe the algorithm that maps G onto Ar. The algorithm is explained in three phases. In the first phase we show how to choose the neighborhood constants n_{l1}, n_{l2} and n_{l3} for the labels $l1$, $l2$ and $l3$. We also show how to construct the function PA that maps computation vertices of G onto processors in Ar. In the second phase we show how to choose the delays d_{l1} and d_{l2} for the labels $l1$ and $l2$. We also show how to map Mesh Graphs in SG in this phase. In the third phase we show how to determine the delay d_{l3} for label $l3$. We also show how to construct the function TA that maps computation vertices onto time steps by composing the mappings of the Mesh Graphs constructed in phase two.

Phase One

1. if $w_1 \neq -1$ then choose $n_{l1}=w_1$, $n_{l2}=w_2$ and $n_{l3}=w_3$,

   if $w_1=-1$ then choose $n_{l1}=-w_1$, $n_{l2}=-w_2$ and $n_{l3}=-w_3$,

2. if $w_1 \neq -1$ then for every computation vertex v_x in diagonal D_i, let

$PA(v_x)=i$, i.e., map the computation vertices in the i^{th} diagonal onto processor i,

if $w_1=-1$ then for every computation vertex v_x in diagonal D_{m+1-i}, let $PA(v_x)=i$.

Phase Two

1. set $d_{l1}=1$. If $n_{l2}=1$ then set $d_{l2}=2$ else set $d_{l2}=1$,

2. for every SG_i do the following:

   a. let v_i denote the computation vertex whose coordinates are $<0,0,i>$. Let $TA(v_i)=t_i$ (we will show in phase three how to determine t_i),

   b. if v_x is a computation vertex in any mesh graph in SG_i, let $TA(v_x)=t_i+x_{l1}\times d_{l1}+x_{l2}\times d_{l2}$.

Phase Three

We first show how to determine d_{l3}.

1. if $n_{l1}=n_{l2}$ then

   a. if $h_1-h_2+n_{l3}\geq 0$ then choose $d_{l3}=h_1+2n_{l3}$,

   b. if $h_1-h_2+n_{l3}<0$ then choose $d_{l3}=h_2+n_{l3}$,

2. if $n_{l1}\neq n_{l2}$ then

   a. if $h_2-h_1+n_{l3}\geq 0$ then choose $d_{l3}=2h_2-1+n_{l3}$,

   b. if $h_2-h_1+n_{l3}<0$ then choose $d_{l3}=2h_1-1-n_{l3}$.

Once d_{l3} is determined, we compose the mapping of the mesh graphs in SG_i by letting $t_i=t_1+i\times d_{l3}$.

In the Appendix we have shown that this mapping is syntactically correct.

Phases one, two and three performs a syntactically correct mapping of a Cube Graph onto a linear array. However to demonstrate a correct execution of the program represented by the Cube Graph some semantic information about the function represented by the computation vertex in the graph needs to be used as we show in the following example.

Example 1: Consider multiplication of two matrices A and B as shown below:

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \begin{vmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{vmatrix} = \begin{vmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{vmatrix}$$

A program for computing this multiplication is given by the following recurrence:

$$c_{ij}^{(k+1)} = c_{ij}^{(k)} + a_{ik} \times b_{kj} \qquad 1 \leq i,k \leq , \quad 1 \leq j \leq 3.$$
$$c_{ij}^{(1)} = 0$$

This program is converted into the program graph shown in Figure 1. In Figure 1, p_{ij} and q_{ij} denote computation vertices. The horizontal, vertical and oblique incident edges of p_{ij} are labelled $l1$, $l2$ and $l3$ respectively. Similarly the horizontal, vertical and oblique outgoing edges of p_{ij} are labelled $l1$, $l2$ and $l3$ respectively. If the horizontal, vertical and oblique incident edges of p_{ij} or q_{ij} represent the values a, b and c respectively then the horizontal, vertical and oblique outgoing edges of p_{ij} or q_{ij} represent the values a, b and c+a×b respectively. In Figure 1, the oblique input edge incident on p_{ij} represents the value $c_{ij}^{(1)}$ which is 0. The oblique outgoing edge from q_{ij} reresents the final (output) value $c_{ij}^{(3)}$ of c_{ij}, i.e., $a_{i1} \times b_{1j} + a_{i2} \times b_{2j}$.

The program graph in Figure 1 is a Cube Graph as illustrated in Figure 2. The Cube Graph is shown without the source and sink vertices for purposes of clarity. The maximun dimensions of $l1^{th}$, $l2^{nd}$ and $l3^{rd}$ axes is 3, 2 and 2 respectively, i.e., $h_1=3$, $h_2=2$ and $h_3=2$. We next map this graph onto a linear array using the mapping algorithm of the previous section.

Let $w=<w_1, w_2, w_3>=<1, 1, 1>$. It can be verified that for this choice of w, the set D of diagonals is comprised of $\{D_1, D_2, D_3, D_4, D_5\}$ where $D_1=\{p_{11}\}$, $D_2=\{p_{12}, p_{21}, q_{11}\}$, $D_3=\{p_{13}, p_{22}, q_{12}, q_{21}\}$, $D_4=\{p_{23}, q_{13}, q_{22}\}$ and $D_5=\{q_{23}\}$. Since $|D|=5$, the linear array has 5 processors indexed from 1 to 5. Each processor is comprised of 3 pairs of input/output ports labelled $l1$, $l2$ and $l3$ respectively.

Let si_t^1, si_t^2 and si_t^3 denote the inputs at the input ports labelled $l1$, $l2$ and $l3$ respectively of processor indexed s at time t and let so_t^1, so_t^2 and so_t^3 denote the outputs computed by s at t. Then, $so_t^1=si_t^1$, $so_t^2=si_t^2$ and $so_t^3 = si_t^3 + si_t^1 \times si_t^2$.

From phase one, we obtain $n_{l1}=1$, $n_{l2}=1$ and $n_{l3}=1$. Also all the computation vertices in D_i are mapped onto processor i.

$n_{l2}=1$ and so from phase two, we obtain $d_{l1}=1$ and $d_{l2}=2$ as the delays for $l1$ and $l2$. The mapped Mesh Graphs SG_1 and SG_2 { SG_1 and SG_2 are obtained by removing all the edges labelled $l3$ from the Cube Graph } are shown in Figure 3.

Now $n_{l1}=n_{l2}$ and $h_1-h_2+n_{l3}>0$ and so from phase three, we obtain $d_{l3}=h_1+2n_{l3}=3+2=5$ and hence $t_2=t_1+5$. The composed mapping for the entire graph is shown in Figure 4. In Figure 4, I_1, I_2 and I_3 are the input ports labelled $l1$, $l2$ and $l3$ respectively of processor 1. O_1, O_2 and O_3 are the output ports labelled $l1$, $l2$ and $l3$ respectively of processor 5. These are the ports of the linear array through which external communication takes place. The elements of the matrices A, B and and C are pumped into the array through the ports I_1, I_2 and I_3 respectively. The computed values of matrix C emerge out of the port O_3.

Lastly, we must show that:

1. for any i and j, if $PA(p_{ij})=s$ (i.e., if s is the processor onto which p_{ij} is mapped) and $s > 1$ then the input value $c_{ij}^{(1)}$ does not change as it travels from I_1 to the input port labelled $l3$ of s,

2. for any i and j, if $PA(q_{ij})=s$ and $s < 5$ then the ouput value $c_{ij}^{(3)}$ does not change as it travels from the output port labelled $l3$ of s to O_3.

An element pumped into I_3 travels at a velocity of 0.2 processors / cycle $(1 / d_{l3})$. Hence if $PA(p_{ij})=s$ and $s > 1$ then by using lemma 2-1, we can compute the times at which the input value $c_{ij}^{(1)}$ appears at the input ports labelled $l3$ of processors indexed 1,2,..,S-1. Similarly if $PA(q_{ij})=s$ and $s < 5$ then we can compute the times at which the output value $c_{ij}^{(3)}$ appears at the input ports labelled $l3$ of s+1,s+2,..,5. This is shown in Table 1. Consider some row - say row 5 in Table 1. The entries t_1-11, t_1-6 and t_1-1 in columns 1, 2 and 3 denote the times at which the input value $c_{23}^{(1)}$ appears at the input port labelled $l3$ of processors indexed 1, 2 and 3 respectively.

Consider row 5 again. If the value 0 appears on any of the other two input ports of processors 1, 2 and 3 at times t_1-11, t_1-6 and t_1-1 then the value represented by $c_{23}^{(1)}$ is preserved. An element pumped into I_1 travels at the rate of 1 processor / cycle $(1 / d_{l1})$. Using lemma 2-1, it can be verified that if 0 is pumped into I_1 at times t_1-11, t_1-7 and t_1-3 then 0 will appear at the input ports labelled $l1$ of processors 1, 2 and 3 at times t_1-11, t_1-6 and t_1-1 respectively.

For every entry in Table 1, we compute the times at which 0 must be pumped into I_1 and this is tabulated in Table 2. Consider some row in Table 2, say row 6 The entries t_1-3 and t_1-4 in columns 1 and 2 indicate that for 0 to appear at the input port labelled $l1$ of processors 1 and 2 at time t_1-3, 0 must be pumped into I_1 at times t_1-3 and t_1-4.

From Table 2 we observe that it suffices to pump 0 into I_1 between t_1-11 and t_1-3 and also between t_1+8 to t_1+16.

EXAMPLE 2: Consider again, multiplication of matrices A and B of example 1 for a different choice of w. Let $w=<w_1, w_2, w_3>=<1,1,-1>$. For this choice of w, the set D of diagonals is comprised of $D_1=\{ q_{11} \}$, $D_2=\{ p_{11}, q_{12}, q_{21} \}$, $D_3=\{ p_{12}, p_{21}, q_{13}, q_{22} \}$, $D_4=\{p_{13}, p_{22}, q_{23} \}$, $D_5=\{ p_{23} \}$.

We use $|D|=5$ processors indexed from 1 to 5. The neighborhood constants for labels $l1$, $l2$ and $l3$ are $n_{l1}=1$, $n_{l2}=1$ and $n_{l3}=-1$. The vertices in D_i are mapped onto processor indexed i. The delays for the labels $l1$, $l2$ and $l3$ are $d_{l1}=1$, $d_{l2}=2$ and $d_{l3}=1$. The resulting mapping of the entire Cube Graph is shown in Figure 5. In Figure 5, I_1 and I_2 are the input ports labelled $l1$ and $l2$ respectively of processor 1 and O_3 is the output port labelled $l3$ of processor 1. Similarly O_1 and O_2 are the output ports labelled $l1$ and $l2$ respectively of processor 5 and I_3 is the input port labelled $l3$ of processor 5. These are the ports of external communication.

Constructions similar to those used for Table 1 and Table 2 are used to construct Table 3 and Table 4 respectively. From Table 4 we observe that it suffices to pump 0 into I_1 between t_1-7 and t_1-2 and also between t_1+3 and t_1+8.

7 Conclusions

We presented a methodology for mapping Cube Graphs onto linear systolic arrays. Cube Graphs are the syntactic structure of program graphs that are naturally executable on two-dimensional systolic arrays.

We illustrated the methodology by synthesizing linear array algorithms for matrix multiplication [13]. Another application of this methodology for synthesis of linear array algorithms for another important problem appears in [14].

The methodology can be generalized to map Hypercube Graphs (i.e., Cube Graphs in Euclidean K-Space where $K > 3$) onto linear systolic arrays. The details appear in [12].

8 APPENDIX

We show that the mapping is syntactically correct. We begin by first showing that the mapping preserves the neighborhood constant of the labels.

Theorem 8-1: Let $l \in L_G$ and let n_l and d_l be its neighborhood and delay constants respectively. Then, if $e=\langle v_x, v_y \rangle$ is the directed edge from v_x to v_y and its label is l then $PA(v_y)=PA(v_x)+n_l$.

Proof: Let v_x and v_y be the vertices in diagonals D_p and D_q respectively and w_p and w_q be the weights of D_p and D_q respectively. So,

$$w_1 \times x_{l1} + w_2 \times x_{l2} + w_3 \times x_{l3} = w_p$$
$$\text{and } w_1 \times y_{l1} + w_2 \times y_{l2} + w_3 \times y_{l3} = w_q$$

Let $l=l1$. Since $e=\langle v_x, v_y \rangle$ and label of e is $l1$ it follows from definition of cube graph that $y_{l1}=x_{l1}+1$, $y_{l2}=x_{l2}$ and $y_{l3}=x_{l3}$. Consequently, $w_q-w_p=w_{l1} \in \{1,-1\}$.

(1): Let $w_{l1}=1$. We show that D_q is $succ(D_p)$.Suppose D_q is not $succ(D_p)$. Let D_r be a diagonal distinct from D_p and D_q such that $w_p < w_r < w_q$. Since w_p, w_r and w_q are integers, it follows that $w_r-w_p \geq 1$ and $w_q-w_r \geq 1$ and hence $w_q-w_p \geq 2$. But $w_q-w_p=w_{l1}$ and $w_{l1}=1$ — a contradiction. So D_q is $succ(D_p)$ and index of D_q is $p+w_{l1}$ when $w_{l1}=1$.

(2): Let $w_{l1}=-1$. We show that D_p is $succ(D_q)$. Suppose D_p is not $succ(D_q)$. Let D_r be a diagonal distinct from D_p and D_q such that $w_q < w_r < w_p$. Since w_p, w_r and w_q are integers, it follows that $w_r-w_q \geq 1$, $w_p-w_r \geq 1$ and $w_q-w_p \geq -2$. But $w_{l1}=-1$ — a contradiction. So D_p is $succ(D_q)$ and index of D_q is $p+w_{l1}$ when $w_{l1}=-1$.

The mapping algorithm maps vertices in D_p onto processor p and those of D_q onto processor $p+w_{l1}$ and hence $PA(v_y)=PA(v_x)+w_{l1} \in \{1,-1\}$. Also from the mapping algorithm $n_{l1}=\pm w_{l1}$. So the theorem holds for $l=l1$. Similarly we can show that the theorem also holds when $l=l2$ and $l=l3$.

□

We next show that the mapping preserves the delay constant of every label l.

Theorem 8-2: Let $l \in L_G$ and let n_l and d_l be its neighborhood and delay constants respectively. Then if $e=\langle v_x, v_y \rangle$ is the directed edge from v_x to v_y and its label is l then $TA(v_y)=TA(v_x)+d_l$.

Proof: (A): Let $l \in \{l1,l2\}$. Clearly, v_x, v_y and e are all in the same mesh

graph within the same set in SG say SG_i. So y_{l3}-x_{l3}=0 and from the mapping algorithm, $TA(v_y)$-$TA(v_x)$=$(y_{l1}$-$x_{l1})d_{l1}$+$(y_{l2}$-$x_{l2})d_{l2}$

1. Let the label of e be $l1$ and so y_{l2}-x_{l2}=0 and y_{l1}-x_{l1}=1 and hence, $TA(v_y)$-$TA(v_x)$=d_{l1}

2. Let the label of e be $l2$ and so y_{l1}-x_{l1}=0 and y_{l2}-x_{l2}=1 and hence, $TA(v_y)$-$TA(v_x)$=d_{l2}

(B): Let the label of e be $l3$. So y_{l3}-x_{l3}=1, y_{l2}-x_{l2}=0 and y_{l1}-x_{l1}=0. Let v_x be a vertex in a mesh graph in SG_i. Clearly, v_y must be a vertex in some mesh graph in SG_{i+1}. From phase 3 of the mapping algorithm it can be shown that $TA(v_y)$-$TA(v_x)$=d_{l3}.

From (A) and (B) above the theorem follows.

□

Lemma 8-1: Let $l \in L_G$ and let P be a major path labelled l. Then the input value represented by the source vertex of P and the output value represented by the sink vertex of P can never appear simultaneously at the input port labelled l of any processor.

Proof: Let v_x be the first[2] computation vertex in P. Let v_s and v_f be the source and sink vertices respectively of P. Let k be the number of computaion vertices in P. In any major path $k \geq 1$. Let $PA(v_x)$=s and $TA(v_x)$=t. So the output value represented by v_f gets computed in processor s+k and this value emerges at the output port labelled l of s+k at time $t+(k+1)\times d_l$. The input value represented by v_s travels upto the input port labelled l of s and reaches it at time $t+k\times d_l$. Clearly, the lemma follows.

□

Lemma 8-2: Let $l \in L_G$ and $n_l \in \{1,-1\}$. Let P_1 and P_2 be two distinct major paths labelled l and let v_x and v_y be the first computation vertices in P_1 and P_2 respectively. Let $PA(v_x)=s_1$, $PA(v_y)=s_2$, $TA(v_y)=t_1$ and $TA(v_y)=t_2$. If the input/output values represented by source and sink vertices of P_1 and P_2 appear simultaneously at the input port of a processor and if $s_2 > s_1$ then $(t_2-t_1)n_l=(s_2-s_1)d_l$.

Proof: Assume without loss of generality tha the input values represented by the source vertices of P_1 and P_2 appear simultaneously at the input port of processor s. Also assume withot any loss of generality $s_2 > S_1$.

[2]the vertex that is the immediate successor of a source vertex in any major path

1.

Let n_l=1. The input port labelled l of processor 0 is the external input port through which the input value represented by source vertices labelled l are fed in. The input value represented by the sources of the major paths P_1 and P_2 pass through intermediate processors ranging from 0 to s_1 and 0 to s_2 respectively. s is one such intermediate processor. Let t be the time at which both the values appear at the input port labelled l of s. The time taken by the input value represented by source vertex of P_1 to reach the input port labelled l of s_1 is $(s_1-s)\times d_l+t$ which is $TA(v_x)$. Similarly the time taken by the input value represented by the source vertex of P_2 to reach the input port labelled l of s_2 is $(s_2-s)\times d_l+t$ which is $TA(v_y)$ and hence,

$t_2-t_1=(s_2-s_1)\times d_l$
$(t_2-t_1)\times n_l=(s_2-s_1)\times d_l$

2. Let n_l=-1. The input port labelled l of processor N-1 is the external input port. So the input value represented by source vertex of P_1 travels from |N|-1 to s_1 passing through the intermediate processor s and the input value represented by source vertex of P_2 travels from |N|-1 to s_2 passing through s. Let t be the time at which both these input values reach s. Time taken to reach s_1 by the input value represented by source vertex of P_1 is $t+(s-s_1)\times d_l$ and the time taken to reach s_2 by the input value represented by source vertex of P_2 is $t+(s-s_2)\times d_l$ and hence,

$t_2-t_2=(s_1-s_2)\times d_l$
$(t_2-t_1)\times n_l=(s_2-s_1)\times d_l$

From (1) and (2) the lemma follows.

□

We next show that the mapping ensures that no two input/output values appear simultaneously at the input port of any processor.

Theorem 8-3: Let $l\in\{l1,l2,l3\}$. Let P_1 and P_2 be two distinct major paths labelled l. The mapping ensures that the input/output value represented by the source/sink vertices of P_1 and P_2 never appear simultaneously at the input port labelled l of any processor.

Proof: Let $l=l1$ and v_x and v_y be the first computation vertices of P_1 and P_2 respectively. From the mapping algorithm we obtain,

$PA(v_y)-PA(v_x)=\Delta P=k_1\times n_{l1}+k_2\times n_{l2}+k_3\times n_{l3}$
$TA(v_y)-TA(v_x)=\Delta T=k_1\times d_{l1}+k_2\times d_{l2}+k_3\times d_{l3}$
where $k_1=(y_{l1}-x_{l1})$ and $-(h_1-1)\leq k_1\leq(h_1-1)$, $k_2=(y_{l2}-x_{l2})$ and $-(h_2-1)\leq k_2\leq(h_2-1)$, $k_3=(y_{l3}-x_{l3})$ and $-(h_3-1)\leq(h_3-1)$.

Withot loss of generality assume $\Delta P > 0$. Also assume that the input/output value represented by the source/sink vertices of P_1 and P_2 appear simultaneously at the input port labelled l of a processor. By lemma 8-2,

$d_{l1}\Delta P=n_{l1}\Delta T$ (*)

We next show that (*) cannot be satisfied.

1. Let $n_{l1}=n_{l2}=1$ and so by the mapping algorithm, $d_{l1}=1$ and $d_{l2}=2$.
   a. Let $h_1-h_2+n_{l3}\geq 0$. So $d_{l3}=h_1+2n_{l3}$ and (*) reduces to $k_3\times(h_1+n_{l3})+k_2=0$. Now $k_1=k_2\neq 0$ as P_1 and P_2 are distinct major paths labelled l. Also $h_2\leq h_1+n_{l3}$ and $-(h_2-1)\leq k_2\leq(h_2-1)$ and so (*) cannot be satisfied.

   b. Let $h_1-h_2+n_{l3}<0$ and so $d_{l3}=h_1+n_{l3}$ and (*) reduces to $k_3\times h_2+k_2=0$. Besides $k_2\leq h_2-1$ and so (*) cannot be satisfied.

2. Let $n_{l1}\neq n_{l2}$. without loss of generality, let $n_{l1}=1$ and $n_{l2}=-1$. So $d_{l1}=1$ and $d_{l2}=1$.
   a. Let $h_2-h_1+n_{l3}\geq 0$ and so $d_{l3}=2h_2-1-n_{l3}$. So (*) reduces to $2k_2+k_3\times(2h_2-1)=0$. But $k_2\leq h_2-1$ and so $2k_2<2h_2-1$ and so (*) cannot hold.

   b. Let $h_2-h_1+n_{l3}<0$ and so $d_{l3}=2h_1-1-n_{l3}$. So (*) reduces to $2k_2+k_3\times(2h_1-2h_3-1)=0$. Now $h_2<h_1-n_{l3}$ and so $2h_2<2h_1-2n_{l3}$. Also $k_2\leq h_2-1$ and so $2k_2\leq 2h_2-2$ and so $2k_2\leq 2h_2-2<2h_1-2n_{l3}-2<2h_1-2n_{l3}-1$ and consequently (*) cannot hold.

A similar proof can be used to show that the theorem holds for $l=l2$ and $l=l3$.

□

REFERENCES

[1] T.C. Chen, V.Y. Lum, and C. Tung.
The Rebound Sorter: An efficient Sort Engine for Large Files.
In *Proc. of the 4^{th} Int'l Conf. on Very Large Data Bases*, pages 312-318. , 1978.

[2] D. Cohen.
Mathematical Approach to Computational Networks.
Technical Report ISI/RR-78-73, ISI, Univ. of Southern California, 1978.

[3] M.J. Foster and H.T. Kung.
The Design of Special-Purpose VLSI Chips.
IEEE Computer 13(1), January, 1980.

[4] L.J. Guibas, H.T. Kung and C.D. Thompson.
Direct VLSI Implementation of Combinatorial Algorithms.
In *Proc. Conf. Very Large Scale Integration: Architecture, Design, Fabrication*, pages 509-525. California Institute of Technology, January, 1979.

[5] L.J. Guibas and F.M. Liang.
Systolic Stacks, Queues and Counters.
In *Proc. Conf. Advanced Research in VLSI*, pages 155-164. MIT, January, 1982.

[6] H.T. Kung and C.E. Leiserson.
Systolic Arrays (for VLSI).
In *Sparse Matrix Proc. 1978*, pages 256-282. SIAM, 1979.

[7] H.T. Kung.
Let's Design Algorithms for VLSI Systems.
In *Proc. Conf. Very Large Scale Integration: Architecture, Design, Fabrication*, pages 65-90. California Institue of Technology, January, 1979.

[8] H.T. Kung and P.L. Lehman.
Systolic (VLSI) Arrays for Relational Database Operations.
In *Proc. SIGMOD*, pages 105-116. , 1980.

[9] S.Y. Kung.
VLSI Array Processor for Signal Processing.
In *Proc. First MIT Conf. Advanced Research in Integrated Circuits.* , January, 1980.

[10] H.T. Kung.
Why Systolic Architectures.
IEEE Computer 15(1):37-46, January, 1982.

[11] C.E. Leiserson.
Systolic Priority Queues.
In *Proc. Conf. Very Large Scale Integration: Architecture, Design, Fabrication*, pages 199-214. California Institute of Technology, January, 1979.

[12] I.V. Ramakrishnan.
Characterization of Programs Correctly Executable on a Model of VLSI Array Processors (in preparation).
PhD thesis, Department of Computer Sciences, U.T. Austin, 1982.

[13] I.V. Ramakrishnan, D.S. Fussell and A. Silberschatz.
Systolic Matrix Multiplication on a Linear Array.
In *20th Annual Allerton Conf. on Computing, Control and Communication.* ,
October, 1982.

[14] P.J. Varman, I.V. Ramakrishnan, D.S. Fussell, and A. Silberschatz.
Robust Systolic Algorithms for Relational Database Operations.
Technical Report , University of Texas at Austin, 1982.

[15] U. Weiser, and A. Davis.
A Wavefront Notation Tool for VLSI Array Design.
In H.T. Kung, R.F. Sproull, and G.L. Steele, Jr. (editors), *VLSI Systems and Computations*, pages 509-525. Computer Science Press, 1981.

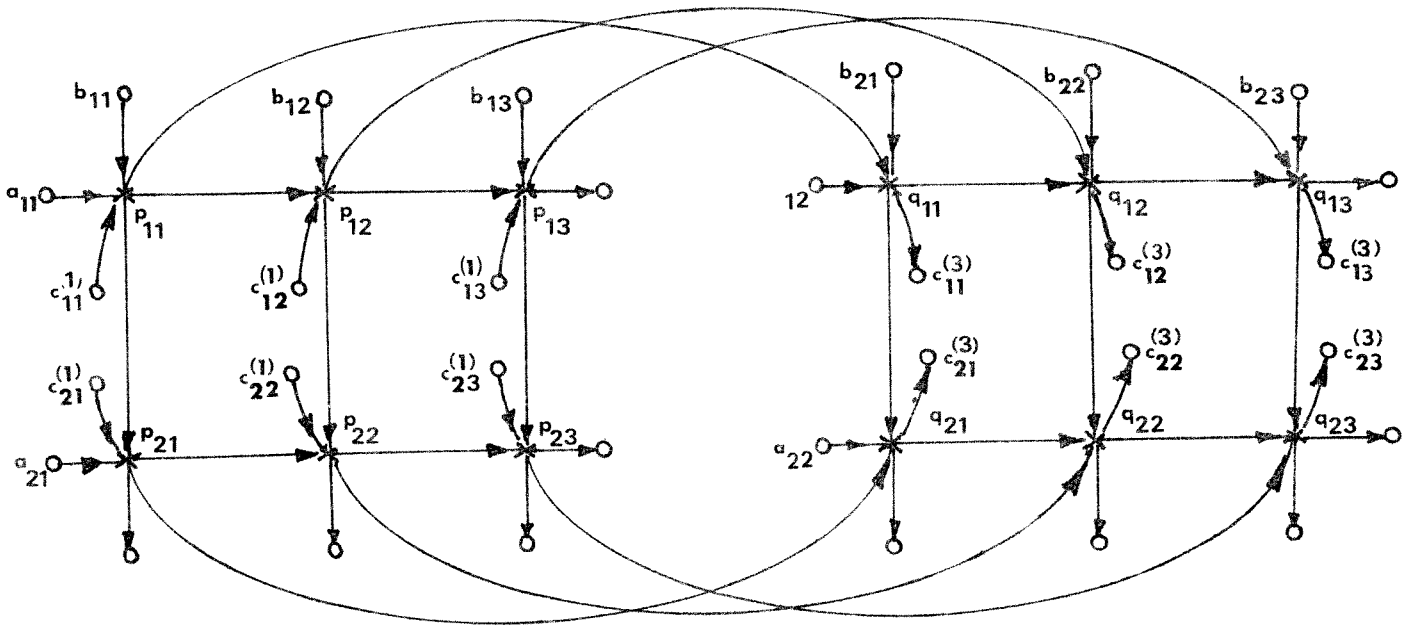
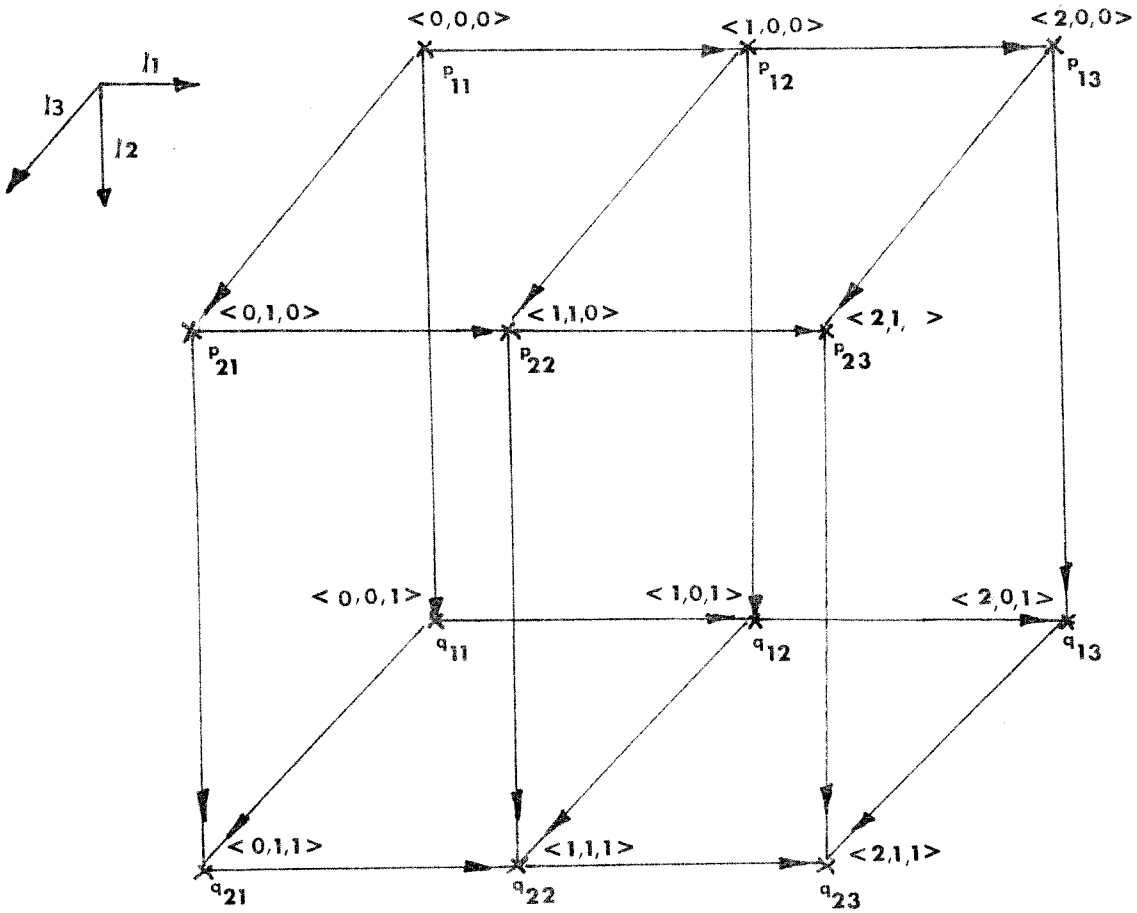


FIGURE –1



FIGURE– 2

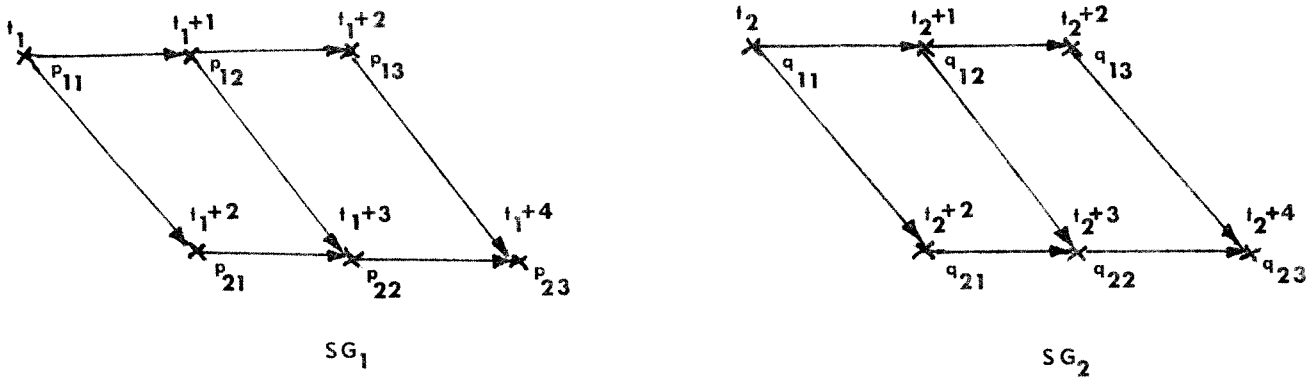


FIGURE-3

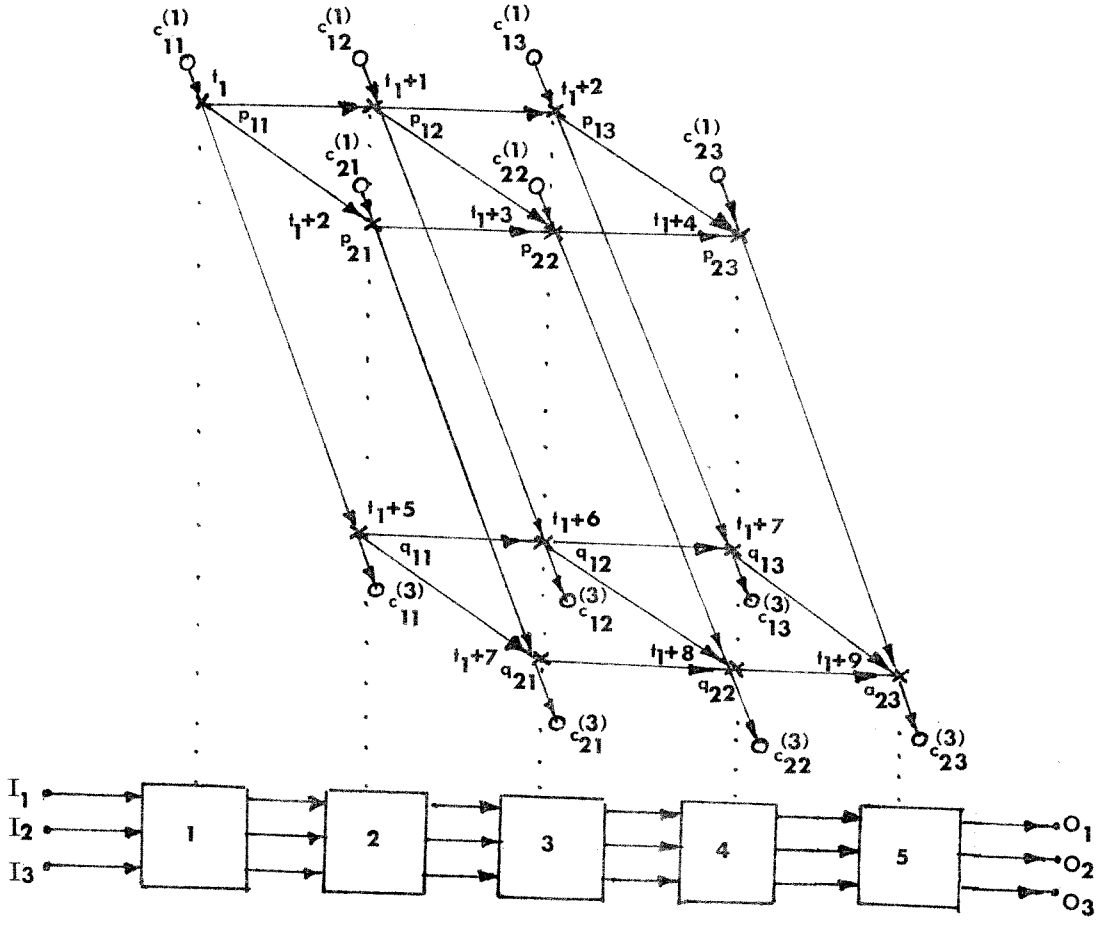


FIGURE-4

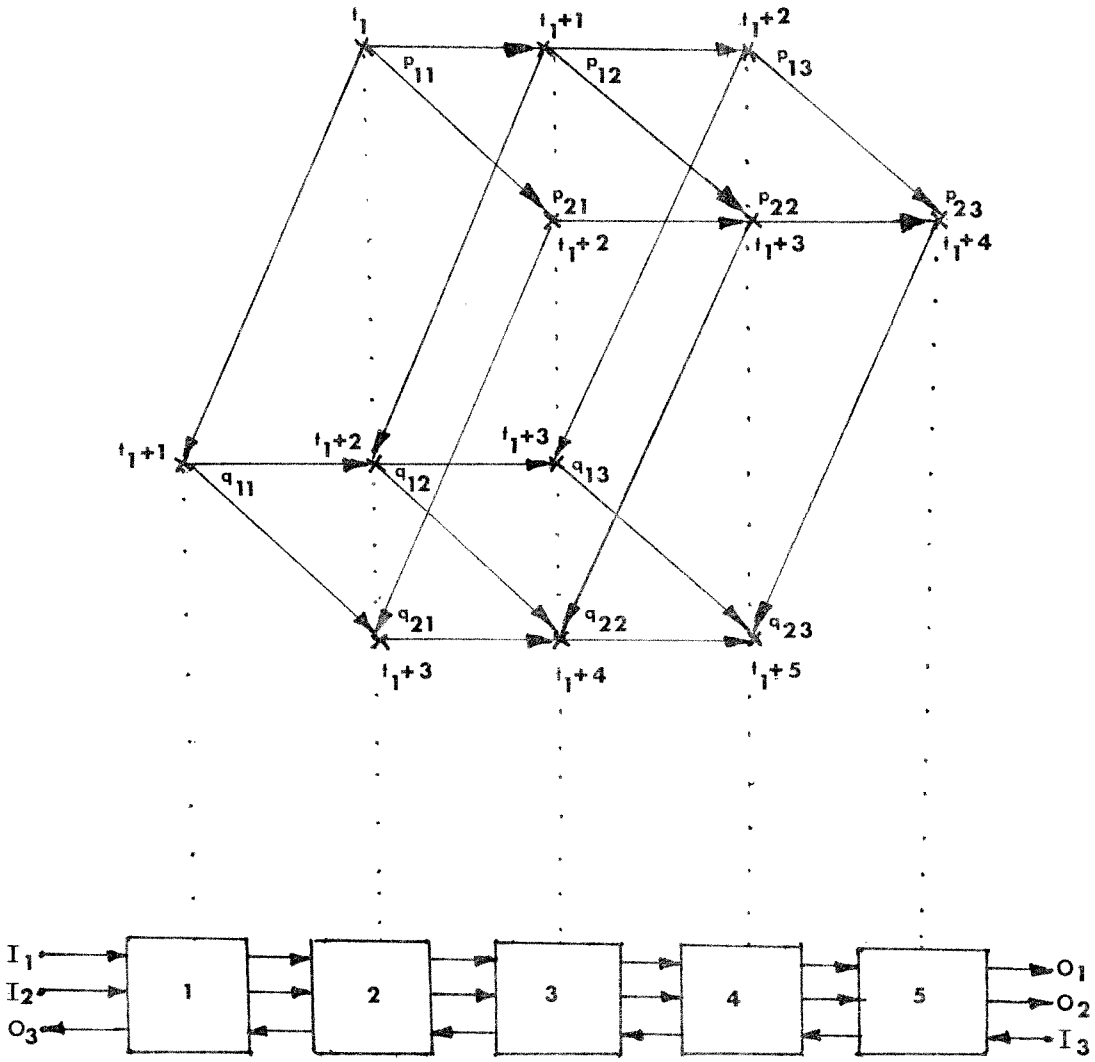

t_1
t_1+1
t_1+2
p_{11}
p_{12}
p_{13}
p_{21}
p_{22}
p_{23}
t_1+2
t_1+3
t_1+4
t_1+1
t_1+2
t_1+3
q_{11}
q_{12}
q_{13}
q_{21}
q_{22}
q_{23}
t_1+3
t_1+4
t_1+5
I_1
I_2
O_3
1
2
3
4
5
O_1
O_2
I_3

FIGURE – 5

INPUT-OUTPUT VALUE

PROCESSOR INDEX

	1	2	3	4	5
$c_{12}^{(1)}$	t_1-4				
$c_{13}^{(1)}$	t_1-8	t_1-3			
$c_{21}^{(1)}$	t_1-3				
$c_{22}^{(1)}$	t_1-7	t_1-2			
$c_{23}^{(1)}$	t_1-11	t_1-6	t_1-1		
$c_{11}^{(3)}$			t_1+10	t_1+15	t_1+20
$c_{12}^{(3)}$				t_1+11	t_1+16
$c_{13}^{(3)}$					t_1+12
$c_{21}^{(3)}$				t_1+12	t_1+17
$c_{22}^{(3)}$					t_1+13

TABLE 1

TIME STEPS

PROCESSOR INDEX

	1	2	3	4	5
t_1-11	t_1-11				
t_1-8	t_1-8				
t_1-7	t_1-7				
t_1-6		t_1-7			
t_1-4	t_1-4				
t_1-3	t_1-3	t_1-4			
t_1-2		t_1-3			
t_1-1			t_1-3		
t_1+10			t_1+8		
t_1+11				t_1+8	
t_1+12				t_1+7	t_1+8
t_1+13					t_1+9
t_1+15				t_1+12	
t_1+16					t_1+12
t_1+17					t_1+13
t_1+20					t_1+16

TABLE 2

INPUT–OUTPUT VALUE

PROCESSOR INDEX

	1	2	3	4	5
$c_{11}^{(1)}$			t_1-1	t_1-2	t_1-3
$c_{12}^{(1)}$				t_1	t_1-1
$c_{13}^{(1)}$					t_1+1
$c_{21}^{(1)}$				t_1+1	t_1
$c_{22}^{(1)}$					t_1+2
$c_{12}^{(3)}$	t_1+3				
$c_{13}^{(3)}$	t_1+5	t_1+4			
$c_{21}^{(3)}$	t_1+4				
$c_{22}^{(3)}$	t_1+6	t_1+5			
$c_{23}^{(3)}$	t_1+8	t_1+7	t_1+6		

TABLE 3

TIME STEPS

PROCESSOR INDEX

	1	2	3	4	5
t_1-3					t_1-7
t_1-2				t_1-5	
t_1-1			t_1-3		t_1-5
t_1				t_1-3	t_1-4
t_1+1				t_1-2	t_1-3
t_1+2					t_1-2
t_1+3	t_1+3				
t_1+4	t_1+4	t_1+3			
t_1+5	t_1+5	t_1+4			
t_1+6	t_1+6		t_1+4		
t_1+7		t_1+6			
t_1+8	t_1+8				

TABLE 4