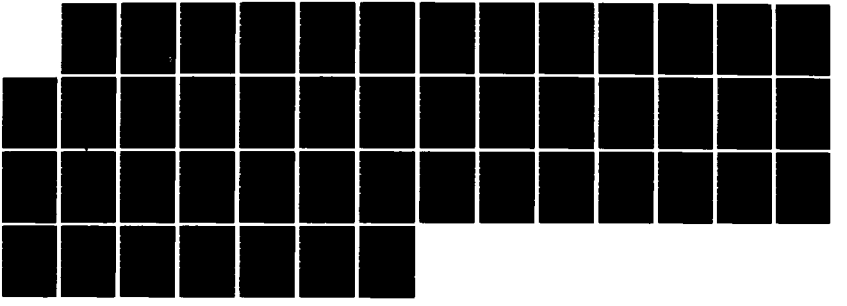
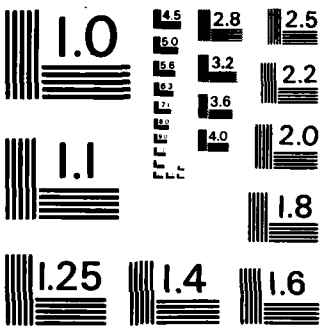
AD-A139 912 ON MAPPING HOMOGENEOUS GRAPHS ON A LINEAR ARRAY-PROCESSOR MODEL (U) MARYLAND UNIV COLLEGE PARK CENTER FOR AUTOMATION RESEARCH I V RAMAKRISHNAN ET AL. 1/1

UNCLASSIFIED OCT 83 CAR-TR-30 AFOSR-TR-84-0180 F/G 12/1 NL





CAR-TR-30
CS-TR-1339

October 1983

ON MAPPING HOMOGENEOUS GRAPHS ON A LINEAR ARRAY-PROCESSOR MODEL

I.V. Ramakrishnan*
D.S. Fussell
A. Silberschatz

Department of Computer Science
The University of Texas at Austin
Austin, TX 78712

CENTER FOR AUTOMATION RESEARCH

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

CAR-TR-30
CS-TR-1339

October 1983

ON MAPPING HOMOGENEOUS GRAPHS ON A LINEAR ARRAY-PROCESSOR MODEL

I.V. Ramakrishnan*
D.S. Fussell
A. Silberschatz

Department of Computer Science
The University of Texas at Austin
Austin, TX 78712

ABSTRACT

This paper presents a formal model of linear array processors suitable for VLSI implementation as well as graph representation of programs suitable for execution on such a model. A distinction is made between correct mapping and correct execution of such graphs on this model and the structure of correctly mappable graphs are examined. The formalism developed is used to synthesize algorithms for this model.

Index Terms-Array Processors, Graphs, Mapping, Parallel Algorithms, VLSI

This research was supported in part by the National Science Foundation and the Office of Naval Research under Grant MCS-8104017 and Contract N00014-80-0987.

* Current address: Department of Computer Science, University of Maryland, College Park, MD 20742. The preparation of this report was supported by the Air Force Office of Scientific Research under Contract ~~F-4960-83-C-0089~~ F49620-83-C-0082.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 84-0180
6a. NAME OF PERFORMING ORGANIZATION University of Maryland 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Research
6c. ADDRESS (City, State and ZIP Code) College Park, Md. 20742	7b. ADDRESS (City, State and ZIP Code) Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR 8b. OFFICE SYMBOL (If applicable) NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F49620-83-C-0082
8c. ADDRESS (City, State and ZIP Code) Bolling AFB DC 20332	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO. A7; WORK UNIT NO.
11. TITLE (Include Security Classification) ON MAPPING HOMOGENEOUS GRAPHS ON A LINEAR ARRAY-PROCESSOR MODEL	
12. PERSONAL AUTHOR(S) I. V. Ramakrishnan D. S. Fussell A. Silberschatz	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) October 1983 15. PAGE COUNT 44
16 SUPPLEMENTARY NOTATION	
17 COSATI CODES FIELD / GROUP / SUB GR.	18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper presents a formal model of linear array processors suitable for VLSI implementation as well as graph representation of programs suitable for execution on such a model. A distinction is made between correct mapping and correct execution of such graphs on this model and the structure of correctly mappable graphs are examined. The formalism developed is used to synthesize algorithms for this model.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☒ DTIC USERS ☐	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL LT COL ROBERT N. BUCHAL	22b TELEPHONE NUMBER (Include Area Code) (202) 767- 4939 22c OFFICE SYMBOL NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE. UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

1. Introduction

In [4, 7] specialized array processors were proposed as a means of handling compute-bound problems in a cost-effective and efficient manner. These array processors generally consist of a regular array of simple, identical processing elements which operate synchronously. A host computer drives the array as a peripheral. The array can be of many forms, for instance a linear array, a rectangular mesh, a hexagonal mesh, etc. Simplicity and regularity of these array processors render them suitable for VLSI implementation. High performance is achieved by extensive use of pipelining and multiprocessing.

A variety of algorithms have been designed for such arrays [1, 2, 5, 10]. An algorithm executing on such arrays is comprised of several data streams. A data stream is unidirectional, that is, it does not change its direction as it passes through processors in the array. 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 communicates with the host through certain input/output ports designated as external input/output ports and elements in distinct data streams are pumped in through distinct external input/output ports. We will henceforth refer to such algorithms as "array algorithms".

A few methodologies have been proposed for synthesizing array algorithms from program specifications [3, 6, 12]. However in all these methodologies the synthesis problem was not studied in a formal framework. In this paper we study the synthesis of array algorithms in a more rigorous framework using a more intuitive representation of programs, namely, data-flow descriptions of programs. In particular we will be studying the synthesis of algorithms for a linear array. The array is comprised of identical processors, that is, they all execute the same set of instructions in every instruction cycle, and they are all simple, that is, they do not have any addressable local memory and cannot perform branching. The linear array is driven either by a single-phase or two-phase global clock [8]. In a two-phase clocking scheme the two phases are nonoverlapping and adjacent processors are activated by the opposite phases of the clock. Two reasons motivate our study of such a model. Firstly, this model has been used for most of the published array algorithms. Secondly, and more importantly, linear arrays require a fixed I/O bandwidth. Hence they can be attached as a peripheral to the I/O bus of any existing host without requiring any change to the host's I/O bandwidth.

We formalize this linear-array model and then define the program graphs that are appropriate for execution on them. A program graph is a directed acyclic graph representing a computation. The edges represent values and the nodes represent computation of a function whose arguments are the values represented by the incoming edges. We distinguish between correct mapping and correct execution of such program graphs on the linear array model. The structure of correctly mappable graphs are then examined. We also briefly mention the importance of using some semantic knowledge (that is, some property of the function represented by the nodes in the graph) to correctly execute the graph.

The remainder of this paper is organized as follows. In section 2 we formalize the linear array and program graph models appropriate for execution on the linear array. We also provide precise definitions for correct mapping and correct execution of program graphs on the linear array. In section 3 we examine the structural properties of correctly mappable program graphs and support the formalisms by synthesizing a few published and some novel linear-array algorithms.

Since the proofs of the theorems are quite lengthy, and since the reader need not understand it in order to proceed, the details of the proofs are deferred to the Appendix.

2. Computational Models

We begin with a formal definition of the linear array that captures the intuitive linear array model described in the previous section.

2.1. Linear Array Model

A linear array is a 3-tuple $Ar=<N,L_{Ar},\Psi_{Ar}>$ where:

1. N is a sequence of identical processors with indices ranging from 1 to |N|.
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 either by a single-phase or a two-phase global clock. A phase can be viewed as the instruction cycle of a processor. In a single-phase clocking scheme all processors are activated in every phase and every processor computes a k-ary function Ψ_{Ar}. In a two-phase clocking scheme adjacent processors are activated during opposite phases of the clock and every processor computes Ψ_{Ar} in the phase it is active.

The function Ψ_{Ar} computed by a processor is a straight-line program. This restriction is imposed since we have assumed that a processor does not have any branching 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, that is, $\Psi_{Ar}(si_t)=so_t$.

For any label lj in L_{Ar}, let ρ_{lj} be the neighborhood relation imposed by label lj on processors in N. Let $<s,r>$ be any pair of processors in N.

Definition 2.1: We shall say that processor s is related to processor r by label lj denoted as $s\ \rho\ r$, iff the output port labelled lj of s is connected to the input port labelled lj of r. —

We will refer to a path of uniform labels through the array as a data stream. The linear array has the following communication features.

1. A processor in the linear array can only communicate with up to two neighbors. All data streams are unidirectional. Hence for any label lj in L_{Ar}, if ρ_{lj} is not an empty relation, then a neighborhood constant n_{lj} is associated with lj such that the output port labelled lj of any processor s is connected to the input port labelled lj of $s+n_{lj}$ where n_{lj} is one of $\{1, -1, 0\}$.
2. The elements in a data stream move at a constant velocity, and hence a non-zero positive 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 ρ_{lj} is empty then the input port and output port labelled lj of every processor communicate with the host,
   b. if $n_{lj}=1$ then the input port labelled lj of processor 1 and the output port labelled lj of processor |N| communicate with the host,
   c. if $n_{lj}=-1$ then the input port labelled lj of processor |N| and the output port labelled lj of processor 1 communicate with the host,
   d. if $n_{lj}=0$ then a register in every processor serves as the input/output port labelled lj. No

input/output port labelled l_j communicates 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.

We will call the input/output ports that communicate with the host external input/output ports.

The delay d_{l_j} can be implemented as a queue using a shift register of length d_{l_j}-1 if single-phase clocking is used and of length $(d_{l_j}-1)/2$ if two-phase clocking is used. At any time t, then, an activated processor s in the array performs the following sequence of operations:

1. Compute $\Psi_{Ar}(si_t)=so_t$ where $si_t=<si_t^1, si_t^2, ..,si_t^k>$ and $so_t=<so_t^1, so_t^2, ...so_t^k>$.
2. For every label l_j, dequeue the element at the head of the queue associated with l_j and place it at the output port labelled l_j of s.
3. For every label l_j, place so_t^j at the tail of the queue.

Figure 2.1 illustrates a linear array with $n_{l_1}=1$, $n_{l_2}=-1$, $n_{l_3}=0$. The neighborhood relation ρ_{l_4} imposed by label l_4 is empty.

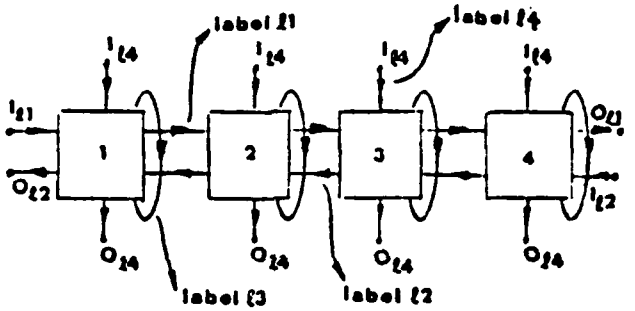


Figure 2.1

Henceforth, "linear array (arrays)" used in the rest of this paper will refer to the model defined above.

2.2. Homogeneous Graphs

The linear array is comprised of identical processors all of which compute the same function (or execute the same instruction) in every cycle. All the processors in the array cooperate in executing a single program. As all the processors in the array are identical, the straight-line programs they execute must also be identical. This motivates the following formalization of programs appropriate for execution on linear arrays.

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

1. $V=V_G \cup SO_G \cup SI_G$, and 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.

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 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 the function represented by v_x then, must be a straight-line program and we can view the k incoming edges and the k outgoing edges of a

vertex v_x as representing the k-tuple input value and k-tuple output value computed by the processor that evaluates v_x. A source vertex then, represents an input value and a sink vertex represents an output value. As every computation vertex represents the same function, we refer to these program graphs as Homogeneous Graphs.

Figure 2.2 illustrates a homogeneous graph. The solid and dashed horizontal edges are labelled $l1$ and $l2$ respectively. The vertical and oblique edges are labelled $l3$ and $l4$ respectively.

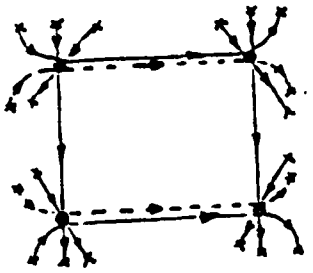


Figure 2.2

In Figure 2.2 and in all the other graphs illustrated in this paper we will be using "o" to represent computation vertices and "x" to denote source and sink vertices.

Although homogeneous graphs are a more limited class of program graphs than, for instance, general dataflow graphs, it does allow the representation of quite a number of interesting programs which are potentially suitable for execution on the linear array model. As we shall see, not even all homogeneous graph programs can be executed on the simple computing engines we have defined.

Henceforth we will assume the following:

1. G is a homogeneous graph.
2. The label of a source (sink) vertex is the same as that of the input (output) edge directed out of the source (directed into the sink) vertex.
3. Input (output) value will always refer to the value represented by a source (sink) vertex.

2.3. Mapping Homogeneous Graphs

We now give a precise formulation of correct mapping and correct execution of homogeneous graphs on linear arrays. Intuitively, mapping of G onto a linear array Ar assigns each computation vertex of G to a processor in Ar at a particular time step and also fixes the delay and neighborhood constant for every label in L_G. Assuming discrete time steps, let $T=\{0,1,2,..\}$ be the sequence of natural numbers representing the progress of computation from its start at time 0.

Definition 2.2: A mapping of G onto a linear array Ar is a 4-tuple <PA,TA,NA,DA> where:

1. $PA:V_G \rightarrow N$ and $TA:V_G \rightarrow T$ are many-one functions mapping computation vertices onto processors and time steps respectively.
2. Let I^+ be a set of positive non-zero integers. $NA:L_G \rightarrow \{1,-1,0\}$ and $DA:L_G \rightarrow I^+$ are many-one functions assigning neighborhood constants and delays to labels respectively.

[Note: $NA(lj)=n_{lj}$ and $DA(lj)=d_{lj}$]

We next formalize a correct mapping.

Definition 2.3: A mapping is syntactically correct iff

1. $\forall lj \in L_{Ar}$ and for any pair of computation vertices, v_x and v_y, if there is an edge labelled lj directed from v_x to v_y, then $PA(v_y)=PA(v_x)+n_{lj}$ and $TA(v_y)=TA(v_x)+d_{lj}$, and

2. no two input/output values can appear simultaneously at the same input port of a processor.

Let i be the input value represented by the source vertex of a computation vertex, say, v_x. Similarly, let o be the output value represented by the sink vertex of another computation vertex, say, v_y. Without loss of generality, let the labels of the source and sink vertices be lj. Now i is fed into the array and o is retrieved from the array through the external input port and external output port respectively associated with label lj. Let $TA(v_x)=t_1$ and $TA(v_y)=t_2$.

Definition 2.4: Entry Time for i and Exit Time for o is the time at which i is fed into and o is retrieved from the array respectively. Consumption Time of i and Production Time of o is t_1 and t_2+d_{lj} respectively.

We are now in a position introduce the notion of correct execution of homogeneous graphs.

Definition 2.5: G is correctly executed on a linear array if the following two conditions hold:

1. the mapping is syntactically correct, and
2. for every input value its value at entry and consumption times must be the same and for every output value its value at production and exit times must be the same.

Intuitively condition (2) means that we may be required to maintain a value input (outputted) to (by) the array constant as it passes through some number of processors inorder that it arrive unchanged at a processor (external output port) that will use it (from which it will be retrieved).

3. Syntactic Characterization

In this section we identify the structure of homogeneous graphs for which there exist syntactically correct mappings. For notational simplicity we will be using the following conventions.

1. Computation vertices will sometimes be referred to simply as "vertices".
2. A pair of vertices will always refer to a distinct pair of computation vertices unless specified otherwise.
3. A path will always refer to a undirected path between any pair of computation vertices. A path will always comprise of a sequence of distinct vertices unless it is a cycle in which case the first and last vertices are the same.
4. In any connected subgraph there exists a path between every pair of vertices in the subgraph through edges in the subgraph.
5. A maximally connected subgraph (that is, if there exists a path between any pair of vertices such that one of them is in the subgraph then the other must also be in the same subgraph) will be referred to as a connected component.
6. A syntactically correct mapping will sometimes be referred to simply as "correct mapping".

We now identify the relevant structural elements of G.

Definition 3.1: For any label lj in G, a major path labelled lj is a directed path from a source vertex v_x to a sink vertex v_y such that the label of v_x, v_y and all the edges in the path is lj.

The path label of a major path is the label of the edges in the path.

Definition 3.2: Two major paths are identical iff, ignoring the source and sink vertices in them, the two directed paths are the same.

For any label lj, let $E_{lj}=\{$major paths having the same path label $lj\}$. Not every E_{lj} is relevant for a syntactic characterization of homogeneous graphs. Consequently, we divide the labels of G into three classes:

1. $L_1=\{lj \mid$ there exists a pair of computation vertices v_x and v_y and a directed edge $e=<v_x,v_y>$ whose label is lj. Besides for any li and lj in L_1 there exists a major path in E_{lj}

that is not identical to any major path in E_{li}.} The major paths with these labels are relevant for structural characterization of correctly mappable graphs.

2. Let L_2={ lj | there exists a pair of computation vertices and a directed edge e=<v_x, v_y> whose label is lj. Besides, if lj is in L_2 then there exists an li in L_1 such that for every major path in E_{lj} there is an identical major path in E_{li}.} Given the major paths associated with the labels in L_1, the major paths associated with those in this class are redundant for structural characterization.
3. L_3={lj | there exists no pair of computation vertices v_x and v_y such that there is a directed edge e=<v_x,v_y> whose label is lj }.

Consider the graph in Figure 2.2 again. The solid and dashed horizontal edges are labelled $l1$ and $l2$ respectively. The vertical and oblique edges are labelled $l3$ and $l4$ respectively. L_1={$l1$, $l3$}, L_2={$l2$} and L_3={$l4$}.

Henceforth, throughout the rest of this paper, labels will be assumed to be in L_1 unless explicitly mentioned otherwise.

Definition 3.3: A minimally labelled connected component SG of G is a 3-tuple <V_{SG}, E_{SG}, L_{SG}> where $V_{SG}\subseteq V$, $E_{SG}\subseteq E$, $L_{SG}\subseteq L_1$ and $V_G\subset V_{SG}$ (that is, all the computation vertices in G are contained in V_{SG}). Besides, for any $lj\in L_{SG}$ if all the edges labelled lj in E_{SG} are removed then SG is disconnected.

[Note: Unlike a minimally labelled connected component, a connected component need not include all the computation vertices.]

We will henceforth refer to L_{SG} as the minimal label set of a graph G.

We have now developed the appropriate formal machinery to undertake a systematic analysis of the structure of program graphs and we begin by examining graphs that have exactly one label in their minimal label set. In particular let L_{SG}={$l\mu$}. This means that there exists a path between every pair of computation vertices through edges labelled $l\mu$. G is a homogeneous graph and hence there is only one such pair of incident and outgoing edge labelled $l\mu$ in any computation vertex. Consequently, there exists only one major path labelled $l\mu$ in G and the path labels of all other major paths are either in L_2 or in L_3. Figure 3.1 illustrates such a graph.

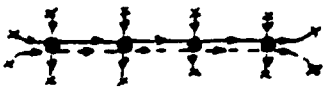


Figure 3.1

In Figure 3.1 the solid and and dotted horizontal edges are labelled $l1$ and $l2$ respectively. The vertical edges are labelled $l3$. L_1={$l1$}, L_2={$l2$} and L_3={$l3$}. Mapping such a graph is straightforward.

3.1. Θ Graphs

We next examine graphs which are comprised of two labels in their minimal label set. We denote the class of such graphs as Θ graphs. Θ is a large class that includes homogeneous program graphs for important computational problems like sorting, convolution, vector multiplication of band matrices, pattern matching, priority queue, linear recurrence, filtering, etc.

In particular, let L_{SG}={$l\mu$, $l\nu$}. G$\in\Theta$ signifies that there is a path between any pair of computation vertices in G through edges that are labelled μ or ν. The structure imposed on SG by any correct mapping is elegantly formalized below.

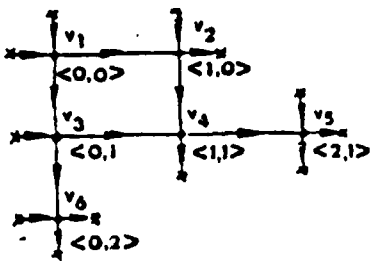
Definition 3.4: Let I_1 and I_2 be two sequences of integers such that the sequences in I_1 and I_2 range from 0 to h_1 and 0 to h_2 respectively and let $B\subseteq I \times I$. Then, SG is a Mesh Graph iff there exists a one-one function F:V_G->B such that the following property holds. Let $F_{l\mu}$ and $F_{l\nu}$ be the projection

functions of F, that is, for any v_x in V_G, if $F(v_x)=<m,n>$ then $F_{l\mu}(v_x)=m$ and $F_{l\nu}(v_x)=n$. For any v_x and v_y in V_G, there exists a directed path from v_x to v_y in a major path whose path label is $l\mu$ such that the distance from v_x to v_y in this directed path is d iff $F_{l\mu}(v_y)=F_{l\mu}(v_x)+d$ and $F_{l\nu}(v_y)=F_{l\nu}(v_x)$. A similar condition holds for a major path whose path label is $l\nu$.

Henceforth we will denote $F_{l\mu}(v_x)$ and $F_{l\nu}(v_x)$ as $x_{l\mu}$ and $x_{l\nu}$ respectively.

Figure 3.2 is an example of a Mesh Graph wherein the horizontal and vertical major paths are labelled $l\mu$ and $l\nu$ respectively.

Figure 3.2



We relate the structure of SG to the existence of a syntactically correct mapping in the following Theorem.

Theorem 3.1: If there exists a syntactically correct mapping for G then SG must be a Mesh Graph.

Proof: See Appendix

□

When G is finally mapped onto a linear array the computation vertices in G may be partitioned into sets that comprise vertices which are mapped onto the same physical processor. As we will see later on this is useful in expressing the structure of correctly mappable graphs in a simple way. To formalize this partitioning it is useful to define a Diagonalization of the Mesh Graph SG as follows.

Definition 3.5: Let $w=<w_1,w_2>\in\{<1,1>, <1,-1>, <1,0>, <0,1>\}$. A Diagonalization of SG 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 $w_1x_{l\mu}+w_2x_{l\nu}=w_1y_{l\mu}+w_2y_{l\nu}$.
3. Let Υ_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 $\Upsilon_D(D_p) < \Upsilon_D(D_q)$ iff $w_1x_{l\mu}+w_2x_{l\nu} < w_1y_{l\mu}+w_2y_{l\nu}$.

Henceforth, we will refer to D as the set of Main Diagonals and to w as the Main Diagonalization Factor. We will assume that the indices assigned to the diagonals in D range from 1 to |D| and if D_p is a diagonal in D then $\Upsilon_D(D_p)=p$, that is, the index of D_p in the ordering is p. We use the ordering of the diagonals in D to define an adjacency relation imposed on them by labelled edges.

Definition 3.6: Let D_p and D_q be in D. D_p a_{lj} D_q (read 'D_p is related by a_{lj} to D_q') iff there exists a computation vertex v_x in D_p and another computation vertex v_y in D_q and a directed edge $e=<v_x,v_y>$ whose label is lj.

Definition 3.7: a_{lj} is consistent with respect to Υ_D iff $\exists$ a constant m_{lj} such that $\forall D_p\in D$ and $\forall D_q\in D$, if D_p a_{lj} D_q then $\Upsilon_D(D_q)=\Upsilon_D(D_p)+m_{lj}$.

We will call m_{lj} the consistency constant of a_{lj}. Let $S_D=\{a_{lj} \mid lj\in L_1$ and a_{lj} is the adjacency relation on D imposed by edges labelled $lj\}$.

It is useful to define the set Dc of Complementary Diagonals that is obtained by diagonalizing SG by its Complementary Diagonalization Factor w_c where $w_c=<0,1>$ when $w\in\{<1,1>, <1,-1>, <1,0>\}$ and $w_c=<1,0>$ when $w=<0,1>$.

Let Υ_{Dc} denote the indexing function associated with Dc and $S_{Dc}=\{b_{ij} \mid ij \in L_1$ and b_{ij} is the adjacency relation on Dc imposed by edges labelled ij $\}$. Herein also we will assume that the index of the complementary diagonals in Dc ranges from 1 to |Dc| and if Dc_p is a complementary diagonal in Dc then its index is p. Consistency of b_{ij} with respect to Υ_{Dc} is defined similar to a_{ij}. Let c_{ij} denote the consistency constant of b_{ij}.

Consider Figure 3.2 again. Let $w=<1,-1>$ and so $w_c=<0,1>$. Then the set of main digonals $D=\{D_1, D_2, D_3, D_4\}$ is comprised of four diagonals where $D_1=\{v_6\}$, $D_2=\{v_3\}$, $D_3=\{v_1, v_4\}$ and $D_4=\{v_2, v_5\}$. The set of complementary diagonals $Dc=\{Dc_1, Dc_2, Dc_3\}$ is comprised of three diagonals where $Dc_1=\{v_1, v_2\}$, $Dc_2=\{v_3, v_4, v_5\}$ and $Dc_3=\{v_6\}$.

Let v_x and v_y be two vertices in the main diagonals D_p and D_q respectively and complementary diagonals Dc_s and Dc_r respectively. Then we will denote the difference in indices of D_q and D_p which is q-p as $\Delta_D(v_x,v_y)$. We will also denote the difference in indices of Dc_r and Dc_s which is r-s as $\Delta_{Dc}(v_x,v_y)$.

We next define two classes of graphs $\Theta_1 \subset \Theta$ and $\Theta_2 \subset \Theta$ where:

$\Theta_1=\{G \in \Theta \mid$ SG is a Mesh Graph and the main diagonalization factor w of SG is one of $\{<1,-1>, <0,1>, <1,0>\}\}$ and

$\Theta_2=\{G \in \Theta \mid$ SG is a Mesh Graph and the main diagonalization factor w of SG is $<1,1>\}$.

We provide a complete syntactic characterization of program graphs in Θ_1 which have syntactically correct mappings in the following Theorem. Before doing so we introduce the notion of transitive edges which is needed in the proof sketch of the Theorem.

Definition 3.8: Let $e=<v_x,v_y>$ be a directed edge from vertex v_x to vertex v_y. Then e is a transitive edge iff there exists a vertex v_z and edges $e_m=<v_x,v_z>$ and $e_n=<v_z,v_y>$.

Theorem 3.2: Let $G \in \Theta_1$. There exists a syntactically correct mapping for G if and only if there exists a pair <D,Dc> such that each of the following conditions is satisfied:

1. Every relation $a_{ij} \in S_D$ must be consistent with respect to Υ_D and its consistency constant m_{ij} is one of $\{1,-1,0\}$.
2. Every relation $b_{ij} \in S_{Dc}$ must be consistent with respect to Υ_{Dc}.
3. Let v_x and v_y be any two computation vertices. For any label ij if $c_{ij}\Delta_D(v_x,v_y)=m_{ij}\Delta_{Dc}(v_x,v_y)$ then there must be a major path labelled ij passing through v_x and v_y.

Intuitively, condition (1) ensures that a data stream is unidirectional and communication takes place only between adjacent processors while condition (2) ensures that a data stream moves at constant velocity and condition (3) ensures that no two values appear simultaneously at the input port of any processor.

We sketch the construction used in the sufficiency proof as this construction is used to illustrate synthesis of linear-array algorithms later on.

Proof: (Only If): See Appendix for details.

(If Part): Let $D=\{D_1, D_2, \ldots, D_n\}$ be the set of main diagonals where i denotes the index of any $D_i \in D$. Construct a linear array L_{Ar} with $|N|=n$. Now construct a mapping through the following steps.

1. Choose two-phase clocking if there exists a transitive edge labelled ij such that $m_{ij}=0$ or else choose a single-phase clocking scheme.
2. Let D_q be any diagonal in D and let v_x be any computation vertex in D_q. Then, let $PA(v_x)=q$. This assigns computation vertices to processors.
3. Next fix the neighborhood constant n_{ij} and delay constant d_{ij} for every label ij in L_1. Let

$n_{lij}=m_{lij}$. Let d_a and d_b be two constants which we will be using in the construction of the delays for the labels in L_1. If the main diagonalization factor w is $<1,-1>$ or there exists a transitive edge labelled lj such that $m_{lij}=0$ then let $d_a=2$ else let $d_a=1$. Let c_{min} be the minimum of all consistency constants among all the relations in S_{Dc}. If $c_{min}>0$ then set $d_b=1$ else set $d_b=1+|c_{min}|d_a$. Let $d_{lij}=m_{lij}d_b+c_{lij}d_a$.

4. Next construct the neighborhood and delay constant for the labels in L_2. By definition of L_2, if there exists a label lj in L_2 then there must exist some label li in L_1 such that for every major path in E_{lij} there is an identical major path in E_{li}. Hence let $n_{lij}=n_{li}$ and $d_{lij}=d_{li}$.
5. For every lj in L_3, let the neighborhood relation imposed by label lj on processors in N be empty and hence no processor's output port labelled lj is connected to the input port labelled lj of any processor.
6. Construct the function TA which assigns computation vertices to time steps. Let v_s be the computation vertex which is in $D_1 \in D$ and $Dc_1 \in Dc$. Let $TA(v_s)=t_0$. Let v_x be any computation vertex in $D_p \in D$ and $Dc_q \in Dc$. Then, let $TA(v_x)=t_0+(q-1)d_a+(p-1)d_b$.

Step 1 to step 6 described above completes the construction of a correct mapping. Refer Appendix to verify that the mapping is correct.

□

The three conditions of Theorem 3.2 are necessary but not sufficient for the existence of syntactically correct mappings for graphs in Θ_2. However in the next corollary we show that in certain cases it is both necessary and sufficient. Let $G \in \Theta_2$ and let $C=\{c_{lij}\}-\{c_{l\mu}, c_{l\nu}\}$.

Corollary 3.1: $\forall c_{lij} \in C$, if $c_{lij}>0$ or $\forall c_{lij} \in C$, if $c_{lij}<0$ then there exists a syntactically correct mapping for G if and only if the three conditions in Theorem 3.2 are satisfied.

Proof: Similar to Theorem 3.2 except in the construction of the expressions for the delays. If $c_{lij}>0$ then set $d_a=2$, $d_b=1$, $d_{l\mu}=1$ and $d_{l\nu}=3$. If $c_{lij}<0$ then set $d_a=-2$, $d_b=3$, $d_{l\mu}=3$ and $d_{l\nu}=1$. In the Appendix it is shown that this construction yields $d_{lij}>0$.

□

The sufficiency proof of Theorem 3.2 provides a methodology to synthesize linear-array algorithms for graphs in Θ. The construction used in the Theorem maps a program graph correctly. However, very often, to ensure its correct execution we need to use some property of the function represented by the computation vertices in the graph. The structure of graphs that can be executed without using such knowledge is characterized in [9].

We now apply the results described above to synthesize linear-array algorithms for computing the vector multiplication of band matrices, sorting and convolution.

Example 3.1: Consider multiplication of a Band Matrix M by a Vector X as shown below.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ a_{31} & a_{32} & a_{33} & a_{34} & & \\ & a_{42} & a_{43} & a_{44} & a_{45} & \\ & & a_{53} & a_{54} & a_{55} & \\ & & & a_{64} & a_{65} & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

Figure 3.3 is a program graph representing this computation.

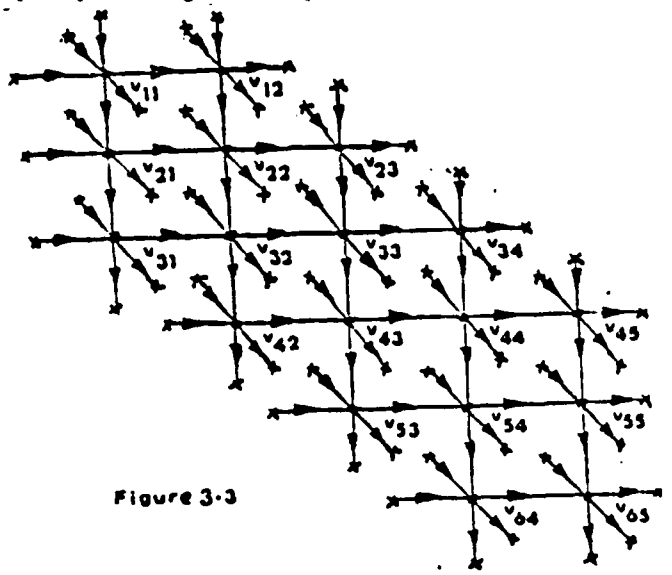


Figure 3.3

In Figure 3.3 v_{ij} denotes a computation vertex. The horizontal, vertical and oblique edges are labelled $l1$, $l2$ and $l3$ respectively. Let Ψ denote the function represented by any computation vertex in the graph. Ψ is a 3-ary function such that for any a, b and c, $\Psi<a,b,c>=<a+bc,b,c>$. Let Ψ_1, Ψ_2, Ψ_3 be the three projections of Ψ, that is, $\Psi_1<a,b,c>=a+bc$, $\Psi_2<a,b,c>=b$ and $\Psi_3<a,b,c>=c$. If a, b and c are the input values represented by the horizontal, vertical and oblique input edges of v_{ij} then the output values represented by the outgoing horozontal, vertical and oblique edges of v_{ij} are $\Psi_1<a,b,c>$, $\Psi_2<a,b,c>$ and $\Psi_3<a,b,c>$ respectively. The input value represented by every horizontal source vertex is initialized to 0. Let E_{l1}={horizontal major paths}, E_{l2}={vertical major paths} and E_{l3}={oblique major paths}. It can be seen that $L_1=\{l1,l2\}$, $L_2=\{\phi\}$ and $L_3=\{l3\}$.

Let SG be a connected component shown in Figure 3.4 that is obtained by removing all the edges labelled $l3$ and source and sink vertices labelled $l3$.

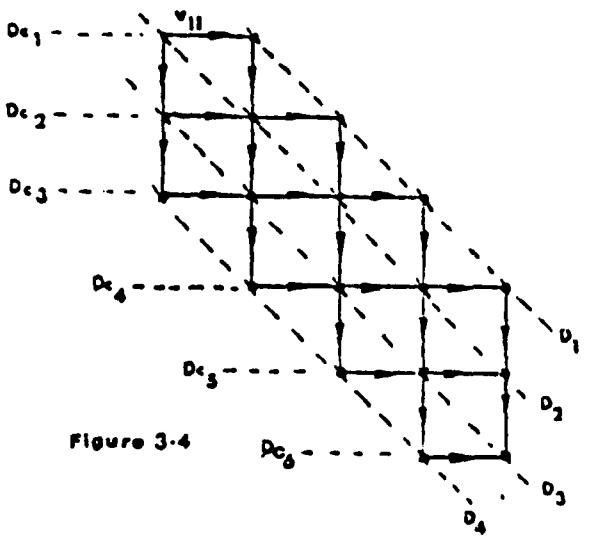


Figure 3.4

For porposes of clarity SG has been drawn without the source and sink vertices. It can be easily verified that the program graph in Figure 3.3 is in Θ as SG is a minimally labelled connected component comprised of $L_{SG}=\{l1, l2\}$. Now diagonalize SG with $w=<1,-1>$ to form the set of main diagonals D. It can be verified that $D=\{D_1,D_2,D_3,D_4\}$ is comprised of four diagonals where $D_1=\{v_{31},v_{42},v_{53},v_{64}\}$, $D_2=\{v_{21},v_{32},v_{43},v_{54},v_{65}\}$, $D_3=\{v_{11},v_{22},v_{33},v_{44},v_{55}\}$ and $D_4=\{v_{12},v_{23},v_{34},v_{45}\}$.

Next diagonalize SG with $w_c=<0,1>$ to form the set Dc of complementary diagonals. It can be verified that $Dc=\{Dc_1,Dc_2,Dc_3,Dc_4,Dc_5,Dc_6\}$ is comprised of six diagonals where $Dc_1=\{v_{11},v_{12}\}$, $Dc_2=\{v_{21},v_{22},v_{23}\}$, $Dc_3=\{v_{31},v_{32},v_{33},v_{34}\}$, $Dc_4=\{v_{42},v_{43},v_{44},v_{45}\}$, $Dc_5=\{v_{53},v_{54},v_{55}\}$ and $Dc_6=\{v_{64},v_{65}\}$.

In Figure 3.4 all the computation vertices belonging to the same diagonal in D lie on the same dashed line. Similarly all the computation vertices belonging to the same diagonal in Dc lie on one horizontal major path.

Now $S_D=\{a_{l1},a_{l2}\}$, $S_{Dc}=\{b_{l1},b_{l2}\}$ and $m_{l1}=1$, $m_{l2}=-1$, $c_{l1}=0$ and $c_{l2}=1$. It can be seen that this graph satisfies Theorem 3.2.

Next, using the construction in Theorem 3.2 we synthesize the linear-array algorithm in [5]. $|D|=4$ and hence the linear array has 4 processors indexed from 1 to 4. $m_{l1}\neq 0$ and $m_{l2}\neq 0$ and hence use single-phase clocking. Each processor is comprised of 3 pairs of input/output ports labelled $l1$, $l2$ and $l3$ respectively. The neighborhood relation ρ_{l3} is empty.

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 s at time t and let so_t^1, so_t^2 and so_t^3 denote the outputs computed by s at time t. Then $so_t^1=si_t^1+si_t^2si_t^3$, $so_t^2=si_t^2$ and $so_t^3=si_t^3$.

The computation vertices in D_1,D_2,D_3 and D_4 are mapped onto processors 1,2,3 and 4 respectively. From the construction of Theorem 3.2, we obtain $n_{l1}=1,n_{l2}=-1,d_{l1}=1$ and $d_{l2}=1$. The resulting mapped graph is shown in Figure 3.5.

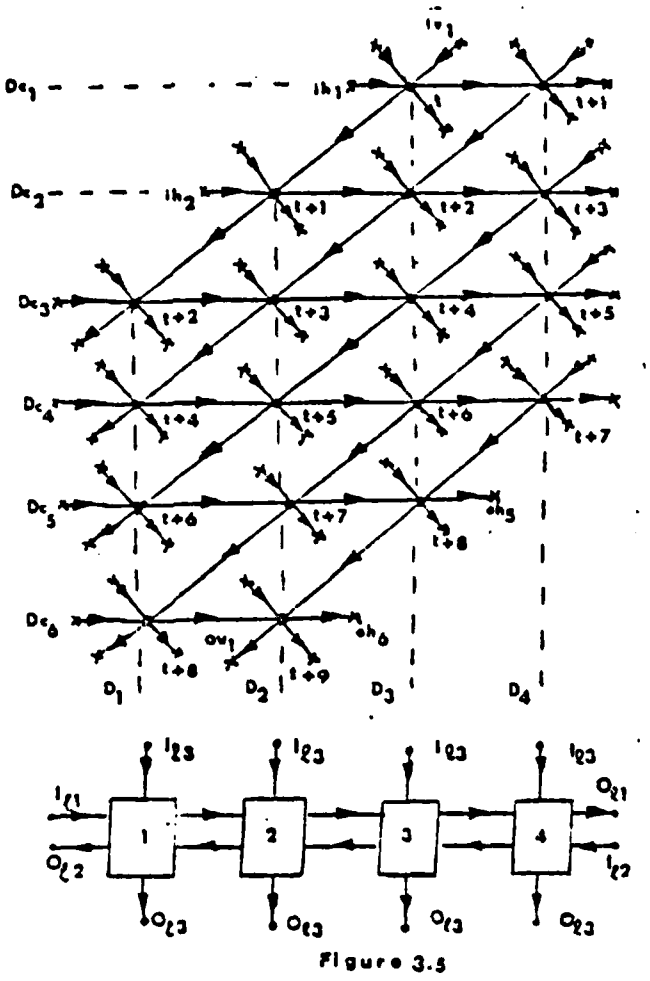


Figure 3.5

The time at which a computation vertex is mapped is indicated by the side of the vertex in Figure 3.5. For instance, the computation vertex on D_3 and Dc_2 is mapped at time t+2. For correctness of execution we must ensure the invariance of the two input values ih_1 and ih_2 at their consumption and entry times and the invariance of the two output values oh_5 and oh_6 at their exit and production times. The consumption times for ih_1 and ih_2 are t and t+1 respectively. Table 3.1 gives the times at which ih_1 appears at the input port labelled $l1$ of processors 1 and 2 and ih_2 appears at the input port labelled $l1$ of processor 1.

Table 3.1

	1	2	3	4
ih_1	t-2	t-1		
ih_2	t			

Any element pumped into I_{l1} or I_{l2} travels at the rate of 1 processor/cycle as $1/d_{l1}=1/d_{l2}=1$. Consider some row of Table 3.1, say 2. The entry in column 1 indicates that ih_2 appears at the input port labelled $l1$ of processor 1 at time t. Now Ψ_1 is such that for any b, $\Psi_1<a,b,0>=a+b0=a$ and hence by pumping 0 into the input port labelled $l3$ of processor 1 at t invariance of ih_2 at its entry and consumption time can be maintained. Similarly by pumping 0 into the input ports labelled $l3$ of processor 1 at t-2 and processor 2 at t-1 invariance of ih_1 at its entry and consumption times can be maintained.

The production times for oh_5 and oh_6 are t+9 and t+10 respectively. Table 3.2 gives the times at which oh_5 appears at the input port labelled $l1$ of processor 4 and oh_4 appears at the input ports labelled $l1$ of processors 3 and 4.

Table 3.2

	1	2	3	4
oh_5				t+9
oh_6			t+10	t+11

The entries in Table 3.2 are interpreted in the same way as the entries in Table 3.1. From Table 3.2 it is seen that by pumping 0 into the input port labelled $l3$ of processor 3 at t+10 and processor 4 at t+9 and t+11 invariance of oh_5 and oh_6 at their production and exit times can be maintained.

Lastly, as $\Psi_2<a,b,c>=b$ for any a and any c, the input value iv_1 and output value ov_1 do not change as they travel through processors in the array.

Example 3.2: We wish to sort the set of elements {2, 10, 5, 6}. A program graph that performs sorting is shown in Figure 3.6 below.

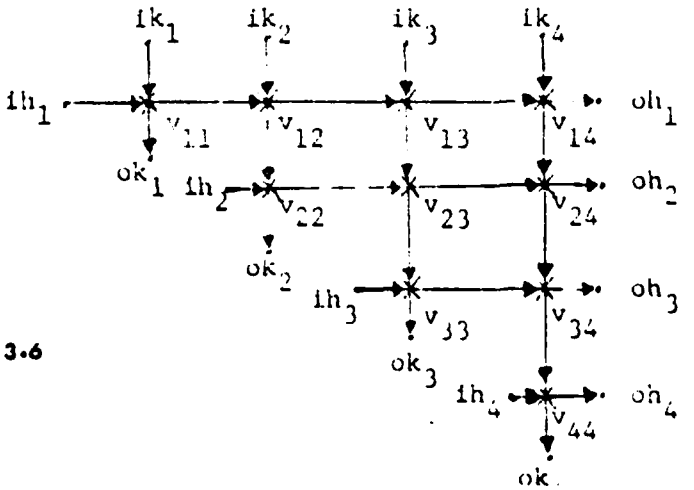


Figure 3.6

In Figure 3.6 v_{ij} denotes a computation vertex. Each computation vertex represents the computation of the minimum and maximum of the two input elements denoted by the incoming horizontal and vertical edges. The outgoing horizontal and vertical edges denote the minimum and maximum respectively of the two input elements computed by the computation vertex. The horizontal edges are labelled $l1$ and the source and sink vertices connected to horizontal edges are all labelled $l1$. The vertical edges are labelled $l2$ and the source and sink vertices connected to vertical edges are all labelled $l2$. The set of horizontal source vertices is $\{ih_1, ih_2, ih_3, ih_4\}$ and the set of horizontal sink vertices is $\{oh_1, oh_2, oh_3, oh_4\}$. Similarly the set of vertical source vertices is $\{ik_1, ik_2, ik_3, ik_4\}$ and the set of vertical sink vertices is $\{ok_1, ok_2, ok_3, ok_4\}$. The initial values represented by the source vertices ik_1, ik_2, ik_3 and ik_4 are 2, 10, 6 and 5 respectively. The initial values represented by the source vertices ih_1, ih_2, ih_3 and ih_4 are all ∞. It can be verified that the final values represented by the sink vertices oh_1, oh_2, oh_3 and oh_4 are 2, 5, 6 and 10 respectively. We synthesize the algorithm known in literature as the "rebound sorter" [1].

Let E_{l1}={horizontal paths} and E_{l2}={vertical paths}. Hence $L_1=\{l1, l2\}$, $L_2=\{\phi\}$ and $L_3=\{\phi\}$. It can be verified that this graph belongs to the class Θ as the minimally labelled connected component comprised of the two labels from L_1 is G itself.

Form the set of main diagonals D by choosing the diagonalization factor w to be <1,-1>. It can be verified that $D=\{\{v_{11},v_{22},v_{33},v_{44}\}, \{v_{12},v_{23},v_{34}\}, \{v_{13},v_{24}\}, \{v_{14}\}\}$.

Let $D=\{D_1,D_2,D_3,D_4\}$ where $D_1=\{v_{11},v_{22},v_{33},v_{44}\}$, $D_2=\{v_{12},v_{23},v_{34}\}$, $D_3=\{v_{13},v_{24}\}$ and $D_4=\{v_{14}\}$. It can be verified that the indices of D_1, D_2, D_3 and D_4 are 1, 2, 3 and 4 respectively in the orderiing of D.

D is obtained by diagonalizing with <1,-1> and hence form Dc by choosing its diagonalization factor w_c to be <0,1>. It can be verified that $Dc=\{\{v_{11},v_{12},v_{13},v_{14}\}, \{v_{22},v_{23},v_{24}\}, \{v_{33},v_{34}\}, \{v_{44}\}\}$. Let $Dc=\{Dc_1,Dc_2,Dc_3,Dc_4\}$ where $Dc_1=\{v_{11},v_{12},v_{13},v_{14}\}$, $Dc_2=\{v_{22},v_{23},v_{24}\}$, $Dc_3=\{v_{33},v_{34}\}$ and $Dc_4=\{v_{44}\}$. It can be verified that the indices of Dc_1, Dc_2, Dc_3 and Dc_4 are 1, 2, 3 and 4 respectively in the ordering of Dc.

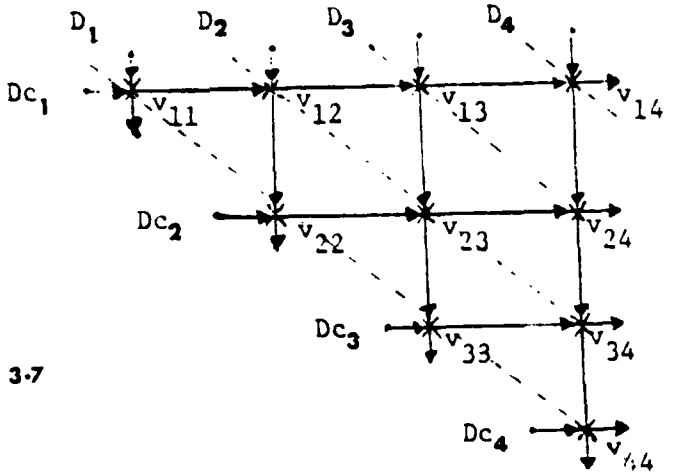


Figure 3.7

In Figure 3.7 above all the computation vertices belonging to a single diagonal in D lie on the same dashed line. Similarly, all the computation vertices belonging to a single diagonal in Dc lie on one horizontal major path.

Now, $S_D=\{a_{l1},a_{l2}\}$ and $S_{Dc}=\{b_{l1},b_{l2}\}$. a_{l1} and a_{l2} are consistent with respect to T_D and b_{l1} and b_{l2} are consistent with respect to T_{Dc}. Hence conditions 1 and 2 of Theorem 3.2 are satisfied. It can be seen that $m_{l1}=1$, $m_{l2}=-1$, $c_{l1}=0$ and $c_{l2}=1$. It can be also verified that condition 3 of Theorem 3.2 is satisfied by the sorting graph.

Using the construction described in Theorem 3.2 we map the sorting graph. |D|=4 and hence the linear

array has 4 processors indexed from 1 to 4. $m_{l1} \neq 0$ and $m_{l2} \neq 0$ and hence use single-phase clocking. Each processor is comprised of 2 pairs of input/output ports labelled $l1$ and $l2$ respectively

Let si_t^1 and si_t^2 denote the inputs at the input ports labelled $l1$ and $l2$ respectively of processor s at time t and let so_t^1 and so_t^2 denote the outputs computed by s at time t. Then, $so_t^1 = \text{Min}(si_t^1, si_t^2)$ and $so_t^2 = \text{Max}(si_t^1, si_t^2)$.

The computation vertices in D_1, D_2, D_3 and D_4 are mapped onto processors 1, 2, 3 and 4 respectively Using the construction in Theorem 3.2 we obtain $n_{l1}=1$, $n_{l2}=-1$, $d_{la}=2$, $d_{lb}=1$, $d_{l1}=1$ and $d_{l2}=1$. The resulting mapped graph is shown in Figure 3.8.

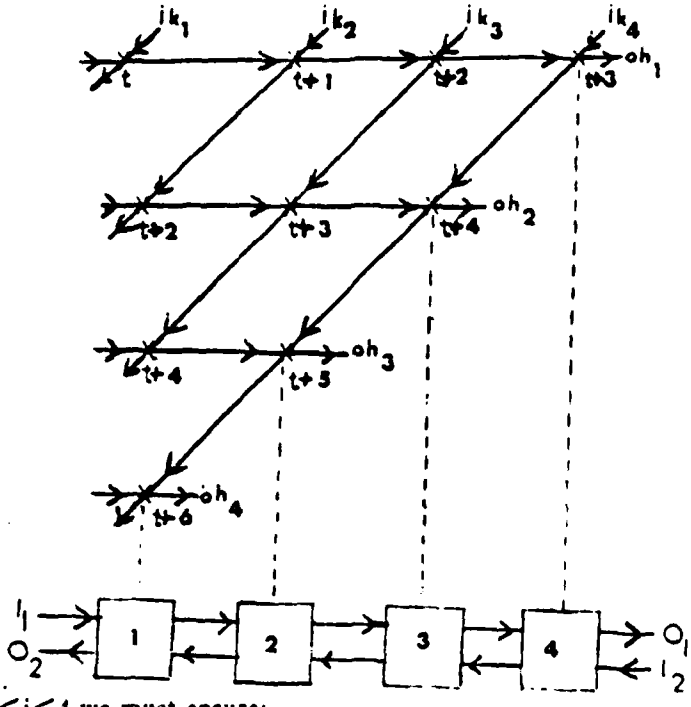


Figure 3.8

Lastly, for any j, $1 \leq j \leq 4$ we must ensure:

1. the invariance of input values, that is,
   a. if $PA(v_{jj})=s$ and $s>1$ then the value represented by ih_j must not change as it travels from I_1 to the input port labelled $l1$ of s,
   b. if $PA(v_{1j})=s$ and $s<4$ then the value represented by ik_j must not change as it travels from I_2 to the input port labelled $l2$ of s.
2. the invariance of output values, that is,
   a. if $PA(v_{jj})=s$ and $s>1$ then we must ensure that the value represented by ok_j remains invariant as it travels from the output port labelled $l2$ of s to O_2,
   b. if $PA(v_{j4})=s$ and $s<4$ then we must ensure that the value represented by oh_j remains invariant as it travels from the output port labelled $l1$ of s to O_1.

We need to use some semantic information of the minimum (Min) and maximum (Max) functions computed by a processor in the array in every cycle. We will use the property that $\text{Min}(x,-\infty)=-\infty$ and $\text{Max}(x,\infty)=\infty$ in our synthesis.

In the mapping we observe that for any j, $1 \leq j \leq 4$, $PA(v_{jj})=1$ and hence we need not consider (1a) and (2a).

An element pumped into I_2 travels at the rate of 1 proceesor / cycle (1 / d_{l2}). Hence, if $PA(v_{1j})$=s then we can compute the times at which the input value represented by ik_j appears at the input ports labelled $l2$ of processors 4,3,..,s-1. This is tabulated in Table 3.3. Similarly, if $PA(v_{j4})$=s then we can compute the times at which the output value represented by oh_j appears at the input ports labelled $l1$ of processors s+1,s+2,..,4. This is tabulated in Table 3.4.

	1	2	3	4
ik_1		t-1	t-2	t-3
ik_2			t	t-1
ik_3				t+1

TABLE 3.3

	1	2	3	4
oh_2				t+5
oh_3			t+6	t+7
oh_4		t+7	t+8	t+9

TABLE 3.4

Consider some row, say row 2, in Table 3.3 and Table 3.4. The entries t,t-1 in columns 3 and 4 of Table 3.3 denote the times at which the input value represented by ik_2 appears at the input port labelled $l2$ of processors 3 and 4 respectively. Similarly, the entries t+6 and t+7 in columns 3 and 4 of Table 3.4 denote the times at which the output value represented by oh_3 appears at the input port labelled $l1$ of processors 3 and 4 respectively.

Now consider row 2 of Table 3.3 and Table 3.4 again. If $-\infty$ appears at the input port labelled $l1$ of processors 3 and 4 at times t and t-1 respectively then the input value represented by ik_2 is preserved. Similarly, if ∞ appears at the input port labelled $l2$ of processors 3 and 4 at times t+6 and t+7 respectively then the output values represented by oh_3 is preserved.

For every entry in Table 3.3 we compute the times at which $-\infty$ must be pumped into I_1 and this is tabulated in Table 3.5. Similarly, for every entry in Table 3.4 we compute the times at which ∞ must be pumped into I_2 and this is tabulated in Table 3.6.

TABLE 3.5

	1	2	3	4
t-3				t-6
t-2			t-4	
t-1		t-2		t-4
t			t-2	
t+1				t-2

TABLE 3.6

	1	2	3	4
t+5				t+5
t+6			t+5	
t+7		t+5		t+7
t+8			t+7	
t+9				t+9

Consider some row, say row 2, in Table 3.5 and Table 3.6. The entry t-4 in column 3 of Table 3.5 indicates that for $-\infty$ to appear at the input port labelled $l1$ of processor 3 at time t-2, it must be pumped into I_1 at time t-4. Similarly, the entry t+5 in column 3 of Table 4 indicates that for ∞ to appear at the input port labelled $l2$ of processor 3 at time t+6, it must be pumped into I_2 at time t+5.

From Table 3.5 we observe that it suffices to pump $-\infty$ into I_1 at times t-6, t-4 and t-2. Similarly, from Table 3.6 we observe that it suffices to pump ∞ into I_2 at times t+5, t+7 and t+9.

Example 3.3: Consider the convolution problem defined as follows.

Given the sequence of weights $\{w_1, w_2, .., w_k\}$ and the input sequence $\{x_1, x_2, .., x_n\}$ compute the output sequence $\{y_1, y_2, .., y_{n+1-k}\}$ defined by $y_i = \sum_{j=1}^{k} w_j x_{i+j-1}$.

We illustrate the convolution problem on n=5 and k=3. The computation of the convolution problem for n=5 and k=3 is represented by the program graph of Figure 3.9.

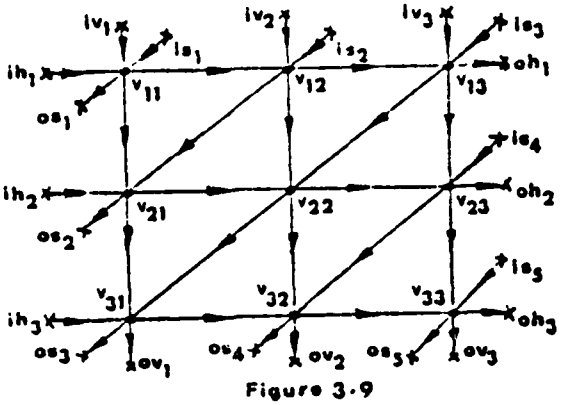


Figure 3.9

In Figure 3.9, $\forall i$ and $\forall j$ $|1 \leq i,j \leq 3$, v_{ij} represents a computation vertex. The horizontal, vertical and oblique edges are labelled $l1$, $l2$ and $l3$ respectively.

Let Ψ denote the function represented by any computation vertex in Figure 3.9. Ψ is a 3-ary function such that for any a, b and c, $\Psi<a,b,c> = <a+bc,b,c>$. Let Ψ_1, Ψ_2, Ψ_3 be the three projections of Ψ, that is, $\Psi_1<a,b,c> = a+bc$, $\Psi_2<a,b,c> = b$ and $\Psi_3<a,b,c> = c$. If a, b and c are the input values represented by the horizontal, vertical and oblique input edges of v_{ij} then the output values represented by the outgoing horozontal, vertical and oblique edges of v_{ij} are $\Psi_1<a,b,c>$, $\Psi_2<a,b,c>$ and $\Psi_3<a,b,c>$ respectively. $\forall p \mid 1 \leq p \leq 5$, $\forall q \mid 1 \leq q \leq 3$ and $\forall r \mid 1 \leq r \leq 3$, let the input values represented by is_p, iv_q and ih_r be x_p, w_q and 0 respectively. It can then be verified that the output values represented by oh_r is $\sum_{q=1}^{3} w_q x_{r+q-1}$.

Let E_{l1}={horizontal major paths}, E_{l2}={vertical major paths} and E_{l3}={oblique major paths}. It can be seen that $L_1=\{l1,l2,l3\}$, $L_2=\{\phi\}$ and $L_3=\{\phi\}$.

Let SG be the connected component shown in Figure 3.10 that is obtained by removing all the edges labelled $l3$ and source and sink vertices labelled $l3$.

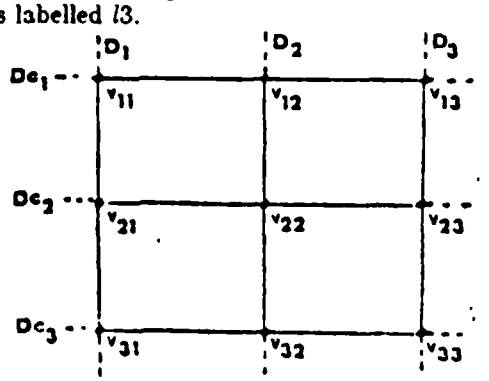


Figure 3.10

For purposes of clarity again SG has been drawn without the source and sink vertices. It can be seen that the program graph in Figure 3.9 is in Θ as SG is a minimally labelled connected component comprised of two labels $l1$ and $l2$.

Now diagonalize SG with $w=<1,0>$ to form the set D of main diagonals. It can be verified that $D=\{D_1,D_2,D_3\}$ is comprised of three diagonals where $D_1=\{v_{11},v_{21},v_{31}\}$, $D_2=\{v_{12},v_{22},v_{32}\}$ and $D_3=\{v_{13},v_{23},v_{33}\}$.

Next diagonalize SG with $w_c=<0,1>$ to form the set Dc of complementary diagonals. It can be verified that $Dc=\{Dc_1,Dc_2,Dc_3\}$ is also comprised of three diagonals where $Dc_1=\{v_{11},v_{12},v_{13}\}$, $Dc_2=\{v_{21},v_{22},v_{23}\}$, $Dc_3=\{v_{31},v_{32},v_{33}\}$.

In Figure 3.10 all the computation vertices belonging to a single diagonal in D lie on the same vertical major path. Similarly, all the vertices belonging to a single diagonal in Dc lie on the same horizontal major path.

Now $S_D=\{a_{l1},a_{l2},a_{l3}\}$, $S_{Dc}=\{b_{l1},b_{l2},b_{l3}\}$ and $m_{l1}=1$, $m_{l2}=0$, $m_{l3}=-1$, $c_{l1}=0$, $c_{l2}=1$ and $c_{l3}=1$. It can be verified that Theorem 3.2 is satisfied.

We next synthesize the linear-array algorithm in [7]. $|D|=3$ and hence the linear array has 3 processors indexed from 1 to 3. $m_{l2}=0$ and there exist transitive edges labelled $l2$. Hence use two-phase clocking. 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 s at time t and let so_t^1, so_t^2 and so_t^3 denote the outputs computed by s at time t. Then, $so_t^1=si_t^1+si_t^2\times si_t^3$, $so_t^2=si_t^2$ and $so_t^3=si_t^3$.

Using the construction in Theorem 3.2, we obtain $n_{l1}=-1$, $n_{l2}=0$ and $n_{l3}=1$. We also obtain $d_{l1}=1$, $d_{l2}=2$ and $d_{l3}=1$. The computation vertices in D_1, D_2 and D_3 are mapped onto processors 1,2 and 3 respectively. The resulting mapped graph is shown in Figure 3.11.

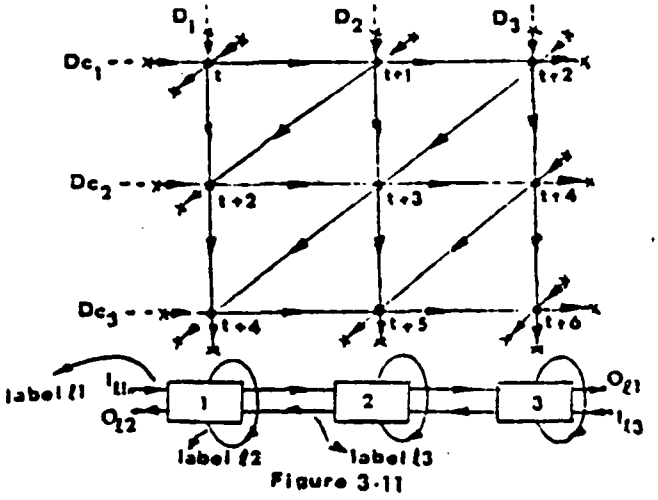


Figure 3.11

Lastly, we must some semantic properties of Ψ for correctness of execution. Ψ_2 and Ψ_3 are such that for any a,b and c, $\Psi_2<a,b,c>=b$ and $\Psi_3<a,b,c>=c$. Hence, the input/output value represented by the source/sink vertices of any vertical or oblique major paths does not change as it travels through processors in the linear array. In Figure 3.11 it is seen that the entry and consumption (production and exit) times for every input (output) value represented by every horizontal source (sink) vertex are the same.

Let t_s be the time when the computation begins. Clearly $t_s \leq t$. Since $n_{l2}=0$ a register in each processor serves as the input/output port labelled $l2$. Let r_1, r_2 and r_3 denote such a register in processors 1, 2 and 3 respectively. Then the input values of iv_1, iv_2 and iv_3 which are w_1, w_2 and w_3 respectively are preloaded into r_1, r_2 and r_3 respectively before t_s.

3.2. Cube Graphs

A natural generalization of the program graphs in Θ are graphs whose minimal label set is

comprised of more than two labels. Program graphs for important problems like matrix multiplication, lu-decomposition of matrices, operations on relations in relational databases are examples of such graphs. A complete characterization of such graphs seems difficult and in this section we examine an important subset of such program graphs and provide a technique for correctly mapping such graphs.

Let $G=<V,E,L_G>$ be a program graph with its label set $L_G=\{l1,l2,l3\}$. Let $I=\{I_1, I_2, I_3\}$ be a family of sets of sequences of integers ranging from 0 to h_1, 0 to h_2 and 0 to h_3 respectively. Let $B \subseteq I_1 \times I_2 \times I_3$.

Definition 3.9: G is a Cube Graph iff there exists a one-one function $F:V_G \rightarrow B$ where:

1. V_G is the set of computation vertices in G.
2. Let F_{l1}, F_{l2} and F_{l3} be three projection functions of F, that is, if $F(v_x)=<c_1,c_2,c_3>$ then $F_{l1}(v_x)=c_1$, $F_{l2}(v_x)=c_2$ and $F_{l3}(v_x)=c_3$. Let v_x and v_y be any two computation vertices in V_G. Then, for any label $l \in L_G$, 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 and $h_1 \geq 1$, $h_2 \geq 1$ and $h_3 \geq 1$. 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 $H=\{1\} \times \{1,-1\} \times \{1,-1\}$ be the cartesian product of the set $\{1,-1\}$. Let $w=<w_1, w_2, w_3> \in H$.

Definition 3.10: 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 $w_1x_{l1}+w_2x_{l2}+w_3x_{l3} = w_1y_{l1}+w_2y_{l2}+w_3y_{l3}$.
3. Let Υ_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 $\Upsilon_D(D_p)<\Upsilon_D(D_q)$ iff $w_1x_{l1}+w_2x_{l2}+w_3x_{l3}<w_1y_{l1}+w_2y_{l2}+w_3y_{l3}$.

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, that is, if v_x is a vertex in D_p then $w_1x_{l1}+w_2x_{l2}+w_3x_{l3}=w_p$.

Consecutive indices are assigned to the diagonals in D with the diagonal having the least weight assigned index 1.

Throughout the rest of this section G will refer to a Cube Graph. $l1$, $l2$ and $l3$ will refer to the three labels in its label set L_G and the subscript of a diagonal will refer to its index, that is, if D_p is a diagonal in D then its index is p.

[Remark 1: A Mesh Graph is a Cube Graph with $|L_G|=2$, that is, cardinality of the label set is 2 and Diagonalization of a Cube Graph is a generalization of Diagonalization of a Mesh Graph.

Remark 2: A minimally labelled connected component SG of a Cube Graph with $L_{SG}=\{l1, l2, l3\}$ is G itself.]

Let $l \in L_G$. Let $MG=\{MG_1, MG_2, .., MG_h\}$ be the set of connected components formed by removing all the edges labelled l and source and sink vertices labelled l from G. The label set for any MG_i in MG is $L_G-\{l\}$.

Lemma 3.1: MG_i is a Mesh Graph.

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

□

We next combine the Mesh Graphs in MG into classes as follows.

Let $CG=\{CG_1, CG_2, ..., CG_n\}$ be a family of sets of Mesh Graphs such that $CG_i=\{MG_q \in MG \mid$ if v_x is a computation vertex in MG_q then $F_l(v_x)=i\}$ (that is, Mesh Graphs in CG_i have the property that the l^{th} coordinate of their computation vertices 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 $l=l3$].

We next describe the algorithm to map a Cube Graph onto a linear array . Let $Ar=<N,L_{Ar},\Psi_{Ar}>$ denote the linear array onto which G is mapped. Without loss of generality, let $l=l3$. So the label set of any Mesh Graph within any set in CG 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, that is, 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.

The algorithm that maps G onto Ar 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 CG 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

Let $n_{l1}=w_1$, $n_{l2}=w_2$ and $n_{l3}=w_3$. For every computation vertex v_x in diagonal D_i, let $PA(v_x)=i$, that is, map the computation vertices in the i^{th} diagonal onto processor 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 CG_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 CG_i, let $TA(v_x)= t_i+x_{l1}d_{l1}+x_{l2}d_{l2}$.

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 $d_{l3}=h_1+1+2n_{l3}$,
   b. if $h_1-h_2+n_{l3}<0$ then choose $d_{l3}=h_2+1+n_{l3}$,
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 CG_i by letting $t_i=t_1+id_{l3}$.

We show that this mapping is syntactically correct in the Appendix.

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 Cube Graph needs to be used as we show in the following examples wherein we use the mapping algorithm described above to synthesize novel linear array algorithms for multiplying matrices that we reported in [10]. These algorithms multiply two $n \times n$ matrices using $O(n)$ processors in $O(n^2)$ time steps. The processors used require no control, no addressable memory and the array requires no loading and unloading circuitry.

Example 3.3: Consider multiplication of two matrices A and B as shown below.

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

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

$$\begin{aligned} \xi_{ij}^{(k+1)} &= \xi_{ij}^{(k)} + a_{ik}b_{kj} \qquad 1 \le i,k \le 2, \quad 1 \le j \le 3. \\ \xi_{ij}^{(1)} &= 0 \end{aligned}$$

The program graph in Figure 3.12 is a representation of this program. In Figure 3.12, 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 q_{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+ab respectively. In Figure 3.12, 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}, that is, $a_{i1}b_{1j}+a_{i2}b_{2j}$.

The program graph in Figure 3.12 is a Cube Graph as illustrated in Figure 3.13. 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 2, 1 and 1 respectively, that is, $h_1=2$, $h_2=1$ and $h_3=1$. We next map this graph onto a linear array using the mapping algorithm described earlier.
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 $so_t^3=si_t^3+si_t^1si_t^2$.

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 $l2$. Now $n_{l1}=n_{l2}$ and $h_1-h_2+n_{l3} \ge 0$ and so from phase three, we obtain $d_{l3}=h_1+1+2n_{l3}=2+1+2=5$ and hence $t_2=t_1+5$. The composed mapping for the entire graph is shown in Figure 3.15.

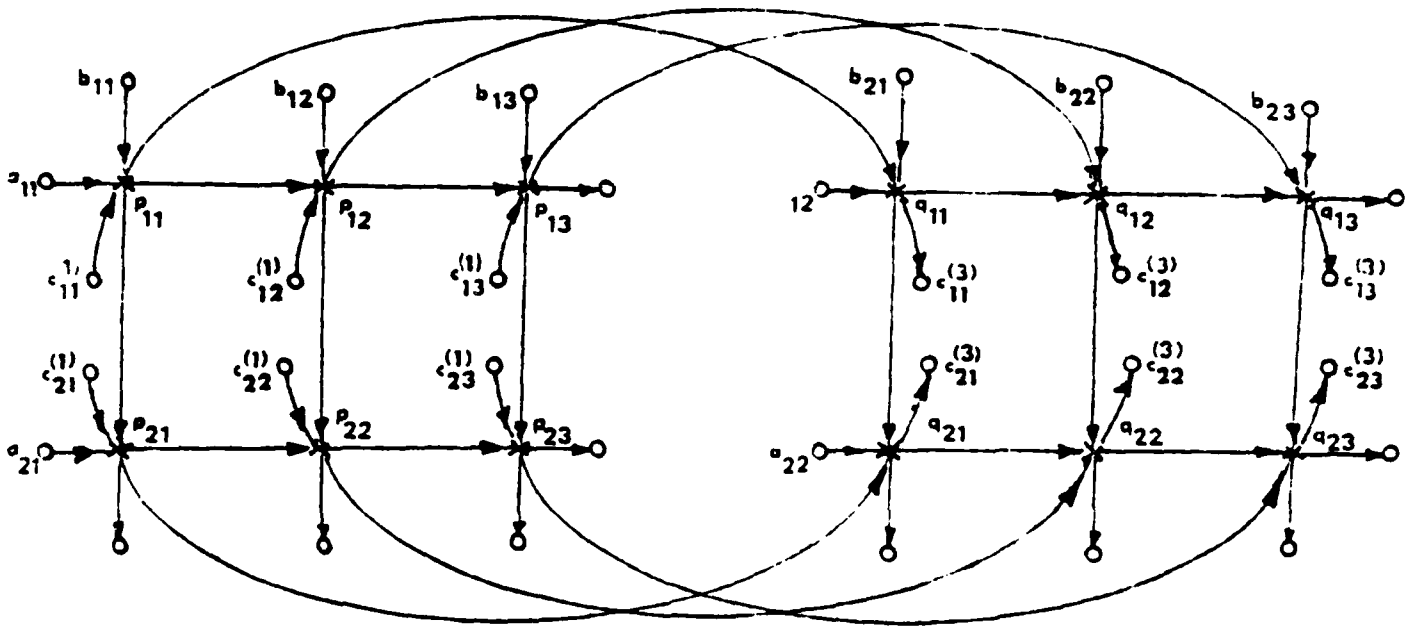


Figure 3-12

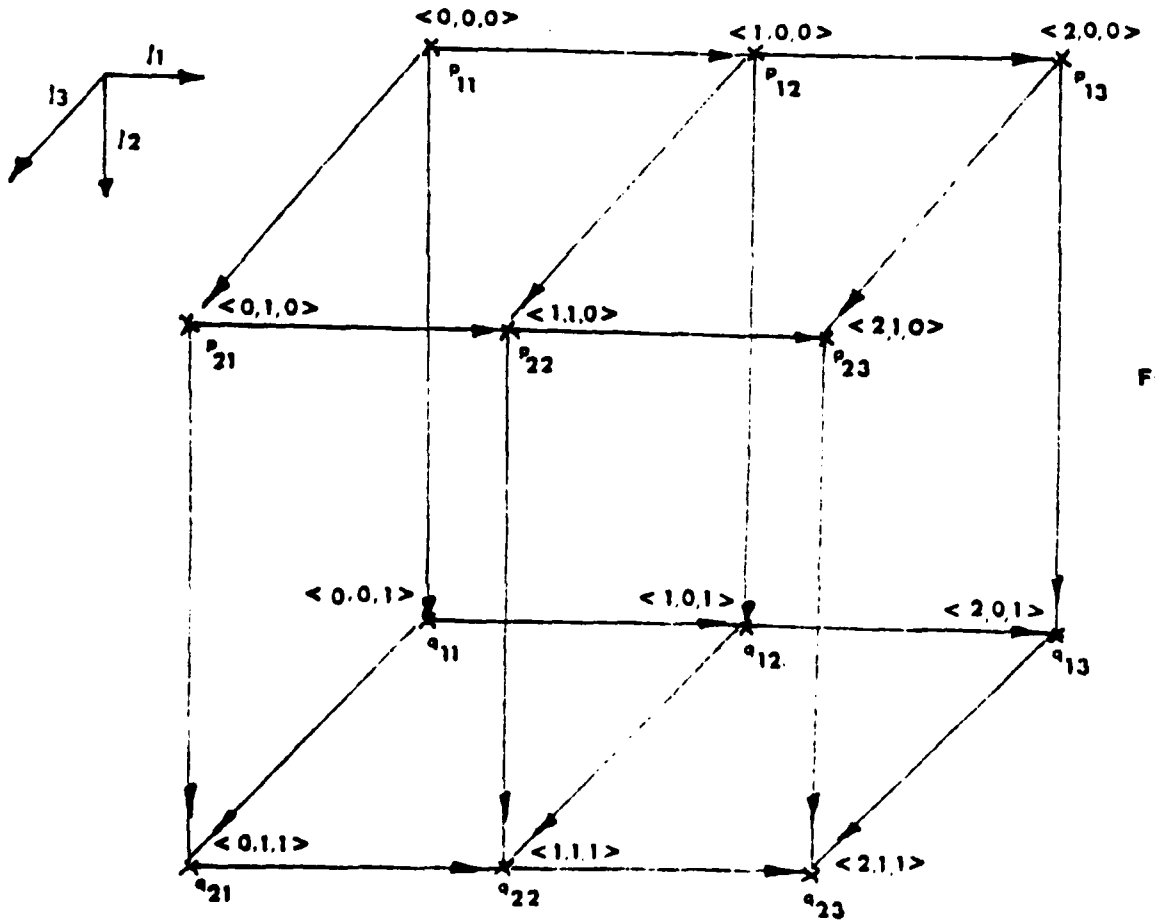


Figure 3.13

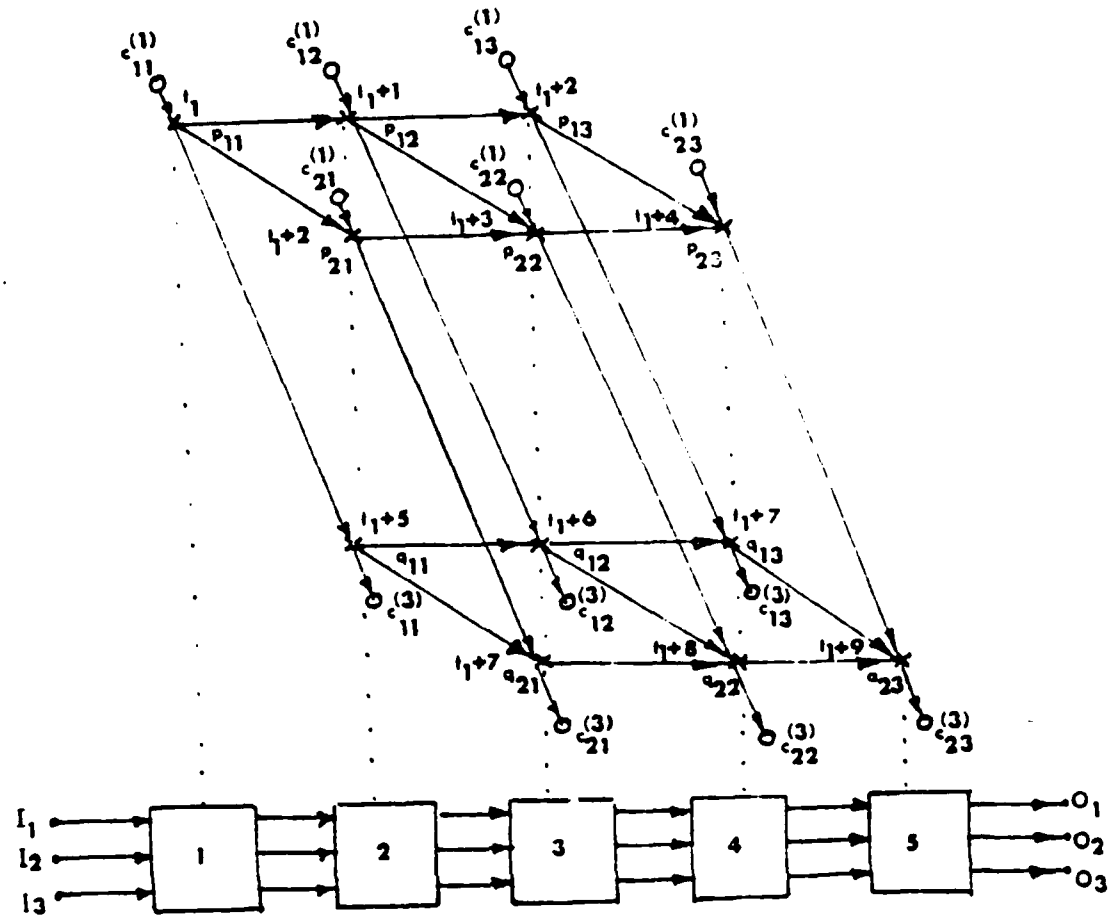


Figure 3.14

In Figure 3.14, 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 $c^{(1)}_{ij}$ 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 $c^{(3)}_{ij}$ 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 then we can compute the times at which the input value $c^{(1)}_{ij}$ appears at the input ports labelled $l3$ of processors indexed 1,2...,s-1. Similarly if $PA(q_{ij})$=s then we can compute the times at which the output value $c^{(3)}_{ij}$ appears at the input ports labelled $l3$ of s+1,s+2,...5. This is shown in Table 3.7. Consider some row - say row 5 in Table 3.7. 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 $c^{(1)}_{23}$ 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 $c^{(1)}_{23}$ is preserved. An element pumped into I_1 travels at the rate of 1 processor/cycle ($1/d_{l1}$). 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 3.7, we compute the times at which 0 must be pumped into I_1 and this is tabulated in Table 3.8. Consider some row in Table 3.8, 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 3.8 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 and t_1+16.

Table 3.7

$c_{12}^{(1)}$	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 3.8

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

Example 3.4: 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 3.15. In Figure 3.15, 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 3.7 and Table 3.8 are used to construct Table 3.9 and Table 3.10 respectively. From Table 3.10 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.

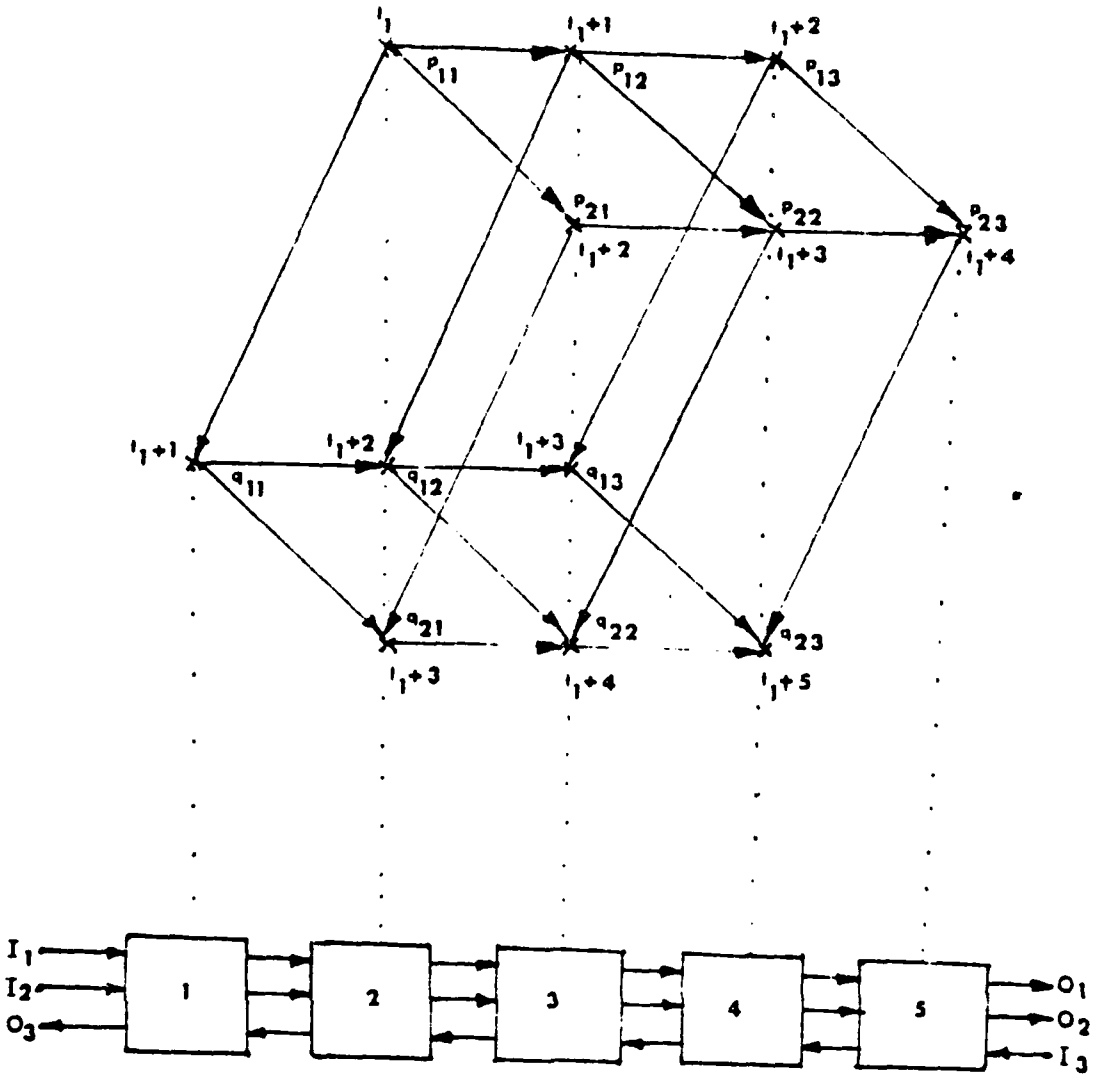


Figure 3.15

$c_{11}^{(1)}$			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 3.10

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 3.11

4. Conclusions

We presented a formal model of linear arrays suitable for VLSI, and introduced homogeneous graphs which are a natural representation of programs potentially executable on such arrays. We then introduced Θ graphs which are subsets of homogeneous graphs and provided a set of necessary and sufficient conditions on the structure of graphs in Θ for the existence of a syntactically correct mapping. As a practical consequence we developed a technique to synthesize linear-array algorithms for programs in Θ and synthesized a few published algorithms.

Subsequently, we examined Cube Graphs which are more general than graphs in Θ and showed a technique to map such graphs correctly onto linear arrays. As a consequence we synthesized some novel linear-array matrix multiplication algorithms.

The technique to correctly map a Cube Graph can be generalized to correctly map Hypercube Graphs (that is, Cube Graphs in Euclidean K-space where K>3) onto linear arrays. The details appear in [9]. However, Hypercube Graphs are only proper subsets of graphs that are not in Θ. The structure of any correctly mappable graph that is not in Θ is an open question.

References

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

[2] L.J. Guibas, and F.M. Liang, "Systolic Stacks, Queues and Counters," Proc. MIT Conf. on Advanced Research in VLSI, (January, 1982), pp. 155-164.

[3] L. Johnsson, and D. Cohen, "A Mathematical Approach to Modelling the Flow of Data and Control in Computational Networks," VLSI Systems and Computations, H.T. Kung, R.F. Sproull, and G.L. Steele, Jr., (editors), Computer Science Press, (1981), pp. 213-225.

[4] H.T. Kung, "Let's Design Algorithms for VLSI Systems," Proc. Caltech Conf. on Very Large Scale Integration: Architecture, Design, Fabrication, (January, 1979), pp. 65-90.

[5] H.T. Kung, and C.E. Leiserson, "Systolic Arrays (for VLSI)," Sparse Matrix Proceedings 1978, I.S. Duff, and G.W. Stewart, (editors), SIAM, (1979), pp. 256-282.

[6] S.Y. Kung, "VLSI Array Processor for Signal Processing," Proc. MIT Conf. on Advanced Research in Integrated Circuits. (January, 1980).

[7] H.T. Kung, "Why Systolic Architectures," IEEE Computer 15(1), (January, 1980), pp. 37-46.

[8] C. Mead, and L. Conway, Introduction to VLSI Systems, Addison-Wesley, (1980).

[9] I.V. Ramakrishnan, "A Theory of Mapping Program Graphs onto Linear Arrays," PhD Thesis, University of Texas at Austin, (May, 1983).

[10] I.V. Ramakrishnan, D.S. Fussell, and A. Silberschatz, "Systolic Matrix Multiplication on a Linear Array," Twentieth Annual Allerton Conf. on Computing, Control and Communication. (October, 1982).

[11] I.V. Ramakrishnan, D.S. Fussell, and A. Silberschatz, "On Mapping Cube Graphs onto Linear Systolic Arrays," Seventeenth Annual Conf. on Information Sciences and Systems. The Johns Hopkins University, (March, 1983).

[12] U. Weiser, and A. Davis, "A Wavefront Notation Tool for VLSI Array Design," VLSI Systems and Computations, H.T. Kung, R.F. Sproull, and G.L. Steele, Jr., (editors), Computer Science Press, (1981), pp. 226-234.

Appendix

Theorem 3.1 and Theorem 3.2 characterize the structure of graphs in Θ. We now develop the proofs for these two theorems. In addition we will also show that the algorithm to map a Cube Graph on a linear array is syntactically correct.

We first establish certain fundamental results on major paths and Mesh Graphs which we will use later on in the proofs of the two theorems.

We will continue to follow the notational conventions adopted in the beginning of section 3. Additionally, for any two computational vertices v_x and v_y, we will be using $\Delta_P(v_x, v_y)$ and $\Delta_T(v_x, v_y)$ to denote $PA(v_y)$-$PA(v_x)$ and $TA(v_y)$-$TA(v_x)$ respectively. Also all the labels in G will be assumed to be in L_1 unless mentioned otherwise and SG will denote a minimally labelled connected component of G.

A.1 Properties of Major Paths

A major path specifies some transformation that a data item undergoes and a correct mapping of a program graph preserves the transformations of all the major paths in the graph. The value represented by a major path will be either the input value represented by the source vertex or the output value represented by the sink vertex or an intermediate value represented by an edge between two pairs of computation vertices in the major path. All major paths in a program graph are unique as we have not assumed any properties of the function represented by the computation vertices in the graph. So a value represented by a major path is also unique. We use uniqueness to mean that the value represented by a major path is distinguishable from the value represented by any other major path.

The processor model that we have used in the linear array does not have any branching ability. This imposes certain restrictions on major paths labelled lj in mappings where the neighborhood constant n_{lj} is 0. These restrictions are captured in the following lemma.

Lemma A.1: Let $lj \in L_1$ and v_x and v_y be any two computation vertices. In any mapping, if $n_{lj}=0$ and $PA(v_y)=PA(v_x)$ (i.e., the neighborhood constant of label lj is 0 and v_x and v_y are mapped on the same processor) then v_x and v_y must be in the same major path labelled lj.

Proof: If $n_{lj}=0$ then in every processor a register serves as the processor's I/O port labelled lj and a value is preloaded into this register, and so if v_x and v_y are in different major paths labelled lj then two registers would be needed – one to hold the value of the first major path and the second to hold the value of the second major path. The processor would then require branching to choose one of the two registers whenever it is in active phase.

□

In the following lemma we relate the vertices and edges in a path to the processors and time steps at which they are mapped.

Lemma A.2: Let v_x and v_y be any pair of vertices in G. Consider any path p from v_x to v_y. For any label lj let k_j^1 and k_j^2 denote the number of edges labelled lj in p whose directions are consistent and not consistent respectively with the directed path from v_x to v_y through the same sequence of vertices as in p. Then in any correct mapping of G, $\sum_{\forall lj} (k_j^1 - k_j^2)n_{lj} = \Delta_P(v_x,v_y)$ and $\sum_{\forall lj} (k_j^1 - k_j^2)d_{lj} = \Delta_T(v_x,v_y)$

Proof: Let $\sum_{\forall lj} (k_j^1 + k_j^2) = n$. So n is the path length. The lemma is easily established by induction on n.

□

From the above lemma the following result on major paths is immediate.

Lemma A.3: Consider any major path labelled lj and let v_x and v_y be any two vertices in this major path. Then in any correct mapping of G, $\Delta_P(v_x,v_y)d_{lj}=\Delta_T(v_x,v_y)n_{lj}$.

Proof: Immediate from Lemma A.2.

□

We next show that if two major paths have the same set of computation vertices then they must be identical.

Lemma A.4: Let q_1 and q_2 be two major paths. Let V_1 and V_2 be the sets of computation vertices in q_1 and q_2 respectively. If $V_1=V_2$ and there exists a correct mapping for G then q_1 and q_2 must be identical.

Proof: Suppose q_1 and q_2 are not identical. Then there must exist two computation vertices v_x and v_y in q_1 and q_2 such that v_x precedes v_y in q_1 and v_y precedes v_x in q_2. Now consider any correct mapping of G. v_x precedes v_y in q_1 and so $\Delta_T(v_x,v_y)>0$. Likewise v_y precedes v_x in q_2 and so $\Delta_T(v_x,v_y)<0$ – a contradiction.

□

We are now in a position to show that there can be at most one label in L_1 whose neighborhood constant can be 0.

Lemma A.5: Let *li* and *lj* be any two labels. Then in any correct mapping of G, if $n_{li}=n_{lj}$ then $n_{li}\in\{1,-1\}$.

Proof: *li* and *lj* are in L_1, so there exists a major path q_r labelled *li* that is not identical to any of the major paths labelled *lj*. This implies that there exists a major path q_s labelled *lj* and,

1. either the computation vertices in q_s and q_r are the same,
2. or the computation vertices in q_s are a subset of the computation vertices in q_r,
3. or the computation vertices in q_r are a subset of the computation vertices in q_s.

Consider the first case. By Lemma A.4, q_r and q_s must be identical.

Next consider the second case. q_s passes through a subset of the vertices in q_r. Let v_x and v_y be two vertices in q_r such that v_x is in this subset and v_y is not. Clearly then, there is a major path q_t labelled *lj* distinct from q_s that passes through v_y as illustrated in Figure A.1.

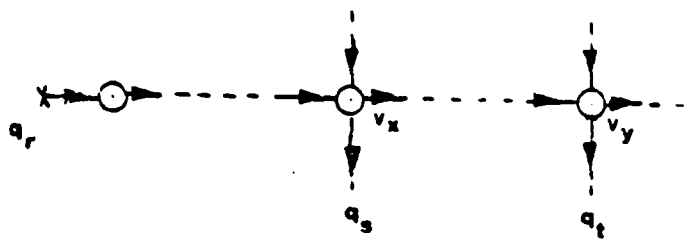


Figure A.1

Now assume $n_{li}=n_{lj}=0$. So $PA(v_x)=PA(v_y)$ and q_s and q_r are distinct major paths labelled *lj* violating Lemma A.1. So $n_{li}=n_{lj}\neq 0$.

We can similarly show that $n_{li}=n_{lj}\neq 0$ in the third case also.

□

A correct mapping must ensure that no two values appear simultaneously at the input port of any processor. As we see in the next lemma this forces some constraint on the structure of major paths.

Lemma A.6: For any label lj and for any pair of vertices v_x and v_y if $\Delta_P(v_x,v_y)d_{lj}=\Delta_T(v_x,v_y)n_{lj}$ in any correct mapping of G then there must be a major path labelled lj passing through v_x and v_y.

Proof: Assume that in a correct mapping there exists a pair of vertices v_x and v_y and a label lj such that $\Delta_P(v_x,v_y)d_{lj}=\Delta_T(v_x,v_y)n_{lj}$ and v_x and v_y are in different major paths labelled lj. Let q_1 and q_2 be the two major paths such that v_x is in q_1 and v_y in q_2. Using Lemma A.3 it can be easily shown that that for any pair of vertices v_u in q_1 and v_w in q_2, if $\Delta_P(v_x,v_y)d_{lj}=\Delta_T(v_x,v_y)n_{lj}$ then $\Delta_P(v_u,v_w)d_{lj}=\Delta_T(v_u,v_w)n_{lj}$. So assume without loss of generality that v_x and v_y are the first[a] computation vertices in q_1 and q_2 respectively. Now $n_{lj}\in\{1,-1,0\}$. We will arrive at a contradiction for each of the three values that n_{lj} assumes.

Case 1: $n_{lj}=0$. So $\Delta_P(v_x,v_y)=0$ as $d_{lj}>0$. Hence by lemma A.1 there must be a major path labelled lj passing through v_x and v_y – a contradiction.

Case 2: $n_{lj}=1$. Now $\Delta_P(v_x,v_y)$ must be either 0, positive or negative. Let $PA(v_x)=s_1$ and $PA(v_y)=s_2$. Let $TA(v_x)=t_1$ and $TA(v_y)=t_2$.

(A): $\Delta_P(v_x,v_y)=0$. So $s_1=s_2$. Now $n_{lj}\neq 0$ and so $\Delta_T(v_x,v_y)=0$. Hence the input value represented by source of q_1 and the input value represented by source of q_2 appear simultaneously at the input port labelled lj of s_1 – a contradiction.

(B): $\Delta_P(v_x,v_y) > 0$. So $s_2>s_1$. Now $n_{lj}=1$ and so $\Delta_T(v_x,v_y)>0$ and hence $t_2>t_1$. The input value represented by source of q_2 appears at the input port labelled lj of s_1 at time $t_2-(s_2-s_1)d_{lj}$. This reduces to $t_2-\Delta_T(v_x,v_y)n_{lj}$ which is $t_2-(t_2-t_1)$. So the input value represented by source of q_1 and that of q_2 appear simultaneously at the input port of s_1 at time t_1 – a contradiction.

(C): $\Delta_P(v_x,v_y) < 0$. So $s_1>s_2$. Now $n_{lj}=1$ and so $\Delta_T(v_x,v_y)<0$ and so $t_1>t_2$. The input value represented by source of q_1 appears at the input port labelled lj of s_2 at time $t_1-(s_1-s_2)d_{lj}$. This reduces to $t_1+\Delta_T(v_x,v_y)n_{lj}$ which is $t_1+(t_2-t_1)$. Hence the two input values appear simultaneously at the input port of s_2 at time t_2 – a contradiction.

Case 3: $n_{lj}=-1$. Using proofs similar to Case 2 we can show that the input values represented by sources of q_1 and q_2 appear simultaneously at the input port labelled lj of a processor.

□

We next show that if the neighborhood constants of any two labels in L_1 are equal then their delays cannot be the same.

Lemma A.7: Let li and lj be any two labels. In any correct mapping of G if $n_{li}=n_{lj}$ then $d_{li}\neq d_{lj}$.

Proof: Now li and lj are in L_1, so there exists a major path q_r labelled li that is not identical to any of the major paths labelled lj. This implies that there exists a major path q_s labelled lj and,

1. either the computation vertices in q_s and q_r are the same,
2. or the computation vertices in q_s are a subset of the computation vertices in q_r,
3. or the computation vertices in q_r are a subset of the computation vertices in q_s.

Consider the first case. By Lemma A.4, q_r and q_s must be identical - a contradiction.

Next consider the second case. q_s passes through a subset of the vertices in q_r. Let v_x and v_y be the two vertices in q_r such that v_x is in the subset and v_y is not. Then there is a major path q_t labelled lj distinct from q_s that passes through v_y as illutrated in Figure A.2.

---

[a] the vertex adjacent to a source vertex in a major path

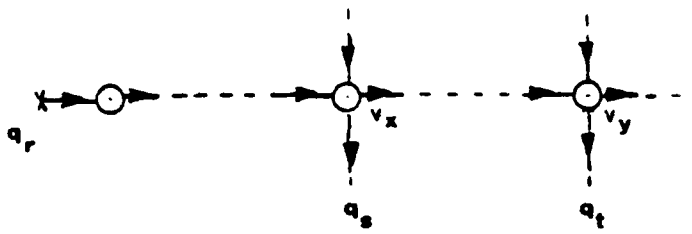


Figure A-2

By Lemma A.3, $\Delta_P(v_x,v_y)d_{li}=\Delta_T(v_x,v_y)n_{li}$. Assume $d_{li}=d_{lj}$ and hence $\Delta_P(v_x,v_y)d_{lj}=\Delta_T(v_x,v_y)n_{lj}$. By Lemma A.6, v_x and v_y must be in the same major path labelled lj – a contradiction.

We can similarly show that $n_{li}=n_{lj}\Rightarrow d_{li}\neq d_{lj}$ for the third case also.

□

A.2 Connected Components

We now examine the relationship between correct mapping and connected components. In particular let $l\mu\in L_1$ and $l\nu\in L_1$. Let S be a connected component obtained by removing all the edges and source and sink vertices from G whose labels are in $L_G-\{l\mu,l\nu\}$. In general several such components may result and S is one such component.

Let $S_{l\mu}=\{$major paths labelled $l\mu$ in S$\}$ and $S_{l\nu}=\{$major paths labelled $l\nu$ in S$\}$.

Let $G_s^\mu=<V_s^\mu,E_s^\mu>$ and $G_s^\nu=<V_s^\nu,E_s^\nu>$ be two directed graphs and F_s^μ and F_s^ν be two one-one functions such that

1. $F_s^\mu:S_{l\mu}\rightarrow V_s^\mu$ (the major paths in $S_{l\mu}$ are represented by the vertices in V_s^μ)
2. $E_s^\mu=\{<q_m,q_n>|\ q_m\in S_{l\mu},\ q_n\in S_{l\mu}$ and there exists a directed edge labelled $l\nu$ from some computation vertex in q_m to some computation vertex in $q_n\}$
3. $F_s^\nu:S_{l\nu}\rightarrow V_s^\nu$ (the major paths in $S_{l\mu}$ are represented by the vertices in V_s^ν)
4. $E_s^\nu=\{<q_m,q_n>|\ q_m\in S_{l\nu},\ q_n\in S_{l\nu}$ and there exists a directed edge labelled $l\mu$ from some computation vertex in q_m to some computation vertex in $q_n\}$

We are now in a position to establish the first fundamental result concerning the structure imposed on S by any correct mapping.

Lemma A.8: If there exists a syntactically correct mapping for G then S must satisfy the following conditions.

1. G_s^μ must be acyclic, and there must be a unique directed path between any pair of vertices in V_s^μ.
2. G_s^ν must be acyclic, and there must be a unique directed path between any pair of vertices in V_s^ν.

Proof: The proofs for (1) and (2) are similar and we thus only prove (1).

We will first show that G_s^μ is acyclic. Suppose there is a cycle in G_s^μ. Let $q_1, q_2, \ldots q_m$ be the set of vertices in V_s^μ that form a cycle in G_s^μ as shown in Figure A.3.

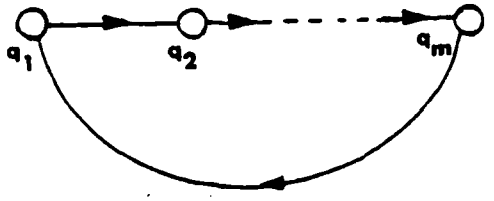


Figure A.3

This cycle implies that, between any pair v_x and v_y of not necessarily distinct computation vertices in q_1, there exists a path p between them through computation vertices in each of q_2, q_3, ..., q_m and through edges labelled $l\mu$ or $l\nu$ as shown in Figure A.4 wherein the "horizontal edges" are labelled $l\mu$ and the "non-horizontal edges" are labelled $l\nu$.

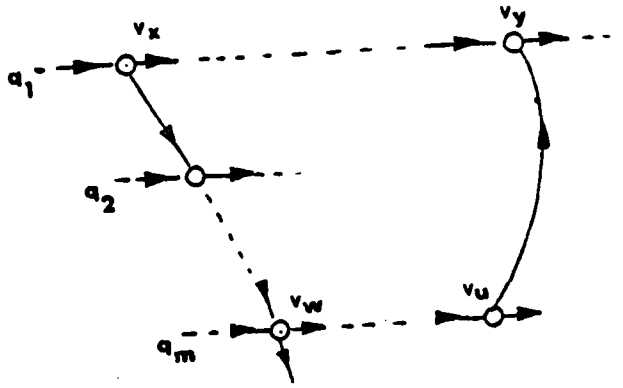


Figure A.4

Let k_μ^1 and k_μ^2 be the number of edges labelled $l\mu$ in p whose directions are consistent and not consistent respectively with the direction imposed on them by the directed path passing through the same sequence of vertices as in p. Let $k_\mu^1 - k_\mu^2 = h$.

Similarly let k_ν^1 and k_ν^2 be the number of edges labelled $l\nu$ in p whose directions are consistent and not consistent respectively with the direction imposed on them by the directed path passing through the same sequence of vertices as in p. As m vertices in G_s^μ form a cycle, $k_\nu^1 - k_\nu^2 = m$ and clearly $m \geq 1$. By Lemma A.2,

$$\Delta_p(v_x, v_y) = n_{l\mu}h + n_{l\nu}m$$

and $$\Delta_T(v_x, v_y) = d_{l\mu}h + d_{l\nu}m$$

Let the distance between v_x and v_y in the major path q_1 be k. Hence

$$\Delta_p(v_x, v_y) = n_{l\mu}k$$

and $$\Delta_T(v_x, v_y) = d_{l\mu}k$$

and so

$$n_{l\mu}k=n_{l\mu}h+n_{l\nu}m \quad \ldots(a)$$

$$d_{l\mu}k=d_{l\mu}h+d_{l\nu}m \quad \ldots(b)$$

By Lemma A.5, $n_{l\mu}=n_{l\nu} \Rightarrow n_{l\mu}=n_{l\nu}\neq 0$ and hence the possible values that $<n_{l\mu}, n_{l\nu}>$ can assume are $<1,0>$, $<1,1>$, $<1,-1>$, $<-1,0>$, $<-1,1>$, $<-1,-1>$, $<0,1>$ and $<0,-1>$. We will arrive at a contradiction for each of these values that $<n_{l\mu},n_{l\nu}>$ asuumes.

1. Consider the set of values $<1,1>$ and $<-1,-1>$, that is, $n_{l\mu}=n_{l\nu}$. From (a) and (b) $d_{l\mu}=d_{l\nu}$ – a contradiction since by Lemma A.7, $d_{l\mu}\neq d_{l\nu}$.
2. Consider the set of values $<0,1>$ and $<0.-1>$, that is, $n_{l\mu}=0$ and $n_{l\nu}\in\{1,-1\}$. From (a) $n_{l\nu}=0$ – a contradiction as $n_{l\nu}\neq 0$.
3. Consider the set of values $<1,0>$ and $<-1,0>$, that is, $n_{l\mu}\in\{1,-1\}$ and $n_{l\nu}=0$. From (a) and (b) $d_{l\nu}=0$ – a contradiction as $d_{l\nu}>0$.
4. Consider the set of values $<-1,1>$ and $<1,-1>$, that is, $n_{l\mu}\in\{1,-1\}$ and $n_{l\nu}\in\{1,-1\}$. From (a) and (b) $d_{l\mu}=-d_{l\nu}$ – a contradiction as $d_{l\mu}>0$ and $d_{l\nu}>0$.

So we have arrived at contradictions when G_s^μ has a cycle and hence G_s^μ must be acyclic.

We next show that there must be a directed path between any pair of vertices in V_s^μ. Suppose not. Then let q_s and q_t be two vertices in V_s^μ that do not have a directed path between them. Now G_s^μ is connected and so there must be a q_k in V_s^μ such that one of the following two cases must occur.

1. There are two vertex disjoint directed paths; one from q_s to q_k and the other from q_t to q_k.
2. There are two vertex disjoint directed paths; one from q_k to q_s and the other from q_k to q_t.

We will only consider the first case and the proof for the second case will be similar. Let q_m be the vertex adjacent to q_k in the directed path from q_s to q_k and q_n be the vertex adjacent to q_k in the directed path from q_t to q_k as shown in Figure A.5.

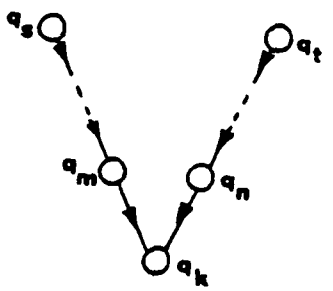


Figure A.5

Now q_m, q_n, q_k, q_s and q_t are all major paths labelled $l\mu$ in G. Existence of a directed edge from q_m to q_k in G_s^μ in Figure A.5 implies there exists computation vertices v_x in q_m and v_w in q_k and a directed edge e_a labelled $l\nu$ from v_x to v_w. Similarly existence of a directed edge from q_n to q_k in G_s^μ implies there exists computation vertices v_y in q_n and v_u in q_k and a directed edge e_b labelled $l\nu$ from v_y to v_u as illustrated in Figure A.6.

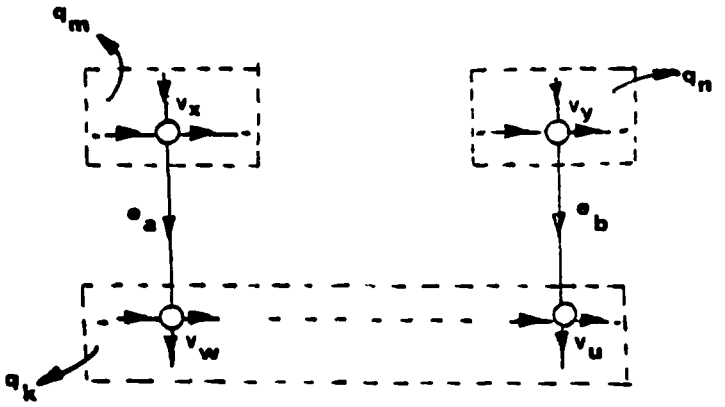


Figure A-6

In Figure A.6 each of the shaded boxes denote a major path labelled $l\mu$. Let the distance between v_w and v_u in q_k be h and hence in any correct mapping,

$$\Delta_P(v_w, v_u) = n_{l\mu} h$$

and $$\Delta_T(v_w, v_u) = d_{l\mu} h$$

As there is a directed edge from v_x to v_w,

$$\Delta_P(v_x, v_w) = n_{l\nu}$$

and $$\Delta_T(v_x, v_w) = d_{l\nu}$$

Also as there is a directed edge from v_y to v_u,

$$\Delta_P(v_y, v_u) = n_{l\nu}$$

and $$\Delta_T(v_y, v_u) = d_{l\nu}$$

From the above equations we obtain,

$$\Delta_P(v_x, v_y) = n_{l\mu} h$$

and $$\Delta_T(v_x, v_y) = d_{l\mu} h$$

Now by Lemma A.6, v_x and v_y must be in the same major path labelled $l\mu$. But q_m and q_n are distinct – a contradiction.

Lastly we must show that the directed path between any pair of vertices in G_s^μ is unique. Suppose not. Let q_m and q_n be two vertices in G_s^μ such that there are two distinct directed paths from q_m to q_n. Let $\{q_m, q_{m1}, \ldots, q_{mi}, q_s, q_t, \ldots\}$ and $\{q_m, q_{m1}, \ldots, q_{mi}, q_s, q_r, \ldots\}$ be the two sequence of vertices traversed by the first and second directed paths respectively. Let q_t and q_r be distinct. So the two sequences differ after q_s. We have already shown that there must be a directed path between any pair of vertices in G_s^μ. Without loss of generality let there be a directed path from q_r to q_t. So now there are two directed paths from q_s to q_t. The first directed path is a directed edge from q_s to q_t and the second directed path is through the

sequence of vertices $\{q_r, q_{r1}, .., q_{rj}\}$. These two directed paths imply the existence of computation vertices v_x and v_y in q_s and q_t respectively and two paths p_1 and p_2 between them as shown in Figure A.7

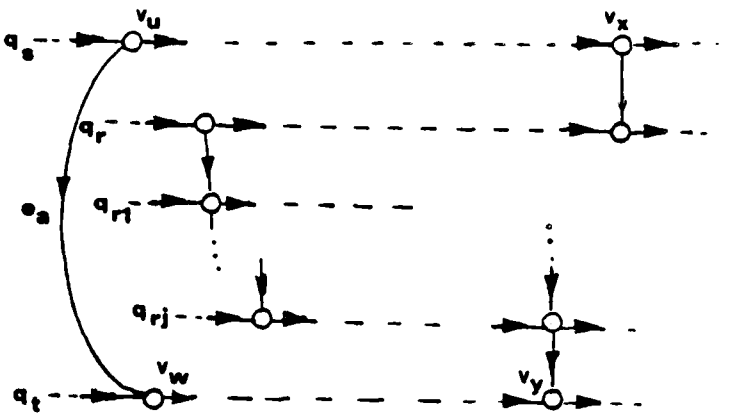


Figure A.7

The first path p_1 between v_x and v_y traverses the edge e_a labelled $l\nu$. The second path p_2 is through computation vertices in $q_r, q_{r1}, .. q_{rj}$. Let k_1^μ and k_2^μ be the number of edges labelled $l\mu$ in p_1 whose directions are consistent and not consistent respectively with the direction imposed on them by the directed path from v_x to v_y passing through the same sequence of vertices as in p_1. Let $k_1^\mu - k_2^\mu = h_1$. For this path from Lemma A.2, we obtain,

$\Delta_P(v_x, v_y) = h_1 n_{l\mu} + n_{l\nu}$

$\Delta_T(v_x, v_y) = h_1 d_{l\mu} + d_{l\nu}$

Let k_3^μ and k_4^μ be the number of edges labelled $l\mu$ in p_2 whose directions are consistent and not consistent respectively with the direction imposed on them by the the directed path from v_x to v_y passing through the same sequence of vertices as p_2. Let $k_3^\mu - k_4^\mu = h_2$. Also let k_1^ν and k_2^ν be the number of edges labelled $l\nu$ in p_2 whose directions are consistent and not consistent respectively with the direction imposed on them by the directed path from v_x to v_y passing through the same sequence of vertices as in p_2. Let $k_1^\nu - k_2^\nu = m$. The distance from q_r to q_t must be at least 1 and so $m > 1$. For the second path p_2, from Lemma A.2 again, we obtain,

$\Delta_P(v_x, v_y) = h_2 n_{l\mu} + n_{l\nu} m$

$\Delta_T(v_x, v_y) = h_2 d_{l\mu} + d_{l\nu} m$

and so

$(h_1 - h_2) n_{l\mu} = (m-1) n_{l\nu}$ (c)

$(h_1 - h_2) d_{l\mu} = (m-1) d_{l\nu}$ (d)

(c) and (d) are similar to (a) and (b) that resulted from a cycle in G_s^μ. Hence solution to (c) and (d) would lead to contradictions and hence the directed path between any pair of vertices in G_s^μ must be unique.

□

We establish the link between Mesh Graphs and S through the following lemma.

Lemma A.9: S is a Mesh Graph if and only if the following conditions are satisfied:

1. G_s^μ is acyclic, and there must exist a unique directed path between any pair of vertices in V_s^μ.

2. G_s^ν is acyclic, and there must exist a unique directed path between any pair of vertices in V_s^ν.

Proof:

(Only If): Simple.

(If Part): Let V_S be the set of computation vertices in S. Topologically sort the vertices of G_s^μ and G_s^ν. Assign indices ranging from 0 to $|V_s^\mu|$-1 to the topologically sorted vertices in V_s^μ. Let I_1 denote the sequence of indices ranging from 0 to $|V_s^\mu|$-1. Similarly assign indices ranging from 0 to $|V_s^\nu|$-1 to the topologically sorted vertices in V_s^ν. Let I_2 denote the sequence of indices ranging from 0 to $|V_s^\nu|$-1.

We next construct a set $B \subseteq I_1 \times I_2$ and a one-one function $F: V_G \rightarrow B$. To begin with let $B=\phi$. Let q_m and q_n be any two vertices in V_s^μ and V_s^ν respectively. Now q_m and q_n are major paths labelled $l\mu$ and $l\nu$ respectively in S. Let v_x be the computation vertex in q_m and q_n. Let a and b be the indices assigned to q_m and q_n respectively by the topological sort of vertices in V_s^μ and V_s^ν respectively. Then let $F(v_x)=<b,a>$ and $B=B \cup \{<b,a>\}$. Using conditions (1) and (2) of the lemma it can be easily shown that F is a one-one function that transforms S into a Mesh Graph.

□

We are now in a position to establish our fundamental result relating S, Mesh Graphs and correct mapping.

Theorem A.1: If there exists a syntactically correct mapping for G then S must be a Mesh Graph.

Proof: Straightforward from Lemma A.8 and Lemma A.9.

□

Theorem 3.1 captured the structure of the minimally labelled component SG of $G \in \Theta$ and its proof is an immediate consequence of Theorem A.1.

A.3 Properties of Mesh Graphs

We examine some properties of Mesh Graphs that we will be using later on. For purposes of examining these properties alone we will assume that the connected component S is a Mesh Graph.

Let v_x and v_y be any two computation vertices in S. Consider any path p between v_x and v_y. Let k_1^μ and k_2^μ be the number of edges labelled $l\mu$ in p whose directions are consistent and not consistent respectively with the direction induced on them by the directed path from v_x to v_y through the same sequence of vertices as in p. Similarly let k_1^ν and k_2^ν be the number of edges labelled $l\nu$ in p whose directions are consistent and not consistent respectively with the direction induced on them by the directed path from v_x to v_y through the same sequence of vertices as p. In the following lemma we relate $<x_{l\mu},x_{l\nu}>$ and $<y_{l\mu},y_{l\nu}>$ to k_1^μ, k_2^μ, k_1^ν, and k_2^ν.

Lemma A.10: $k_1^\mu - k_2^\mu = y_{l\mu} - x_{l\mu}$ and $k_1^\nu - k_2^\nu = y_{l\nu} - x_{l\nu}$.

Proof: The proof is by induction on the path length. Let n denote the path length. v_x and v_y are distinct and hence $n>0$.

Basis Step: $n=1$; so the path consists of only one edge. Hence only one of k_1^μ, k_2^μ, k_1^ν and k_2^ν can be 1 and the rest must be 0.

We will show for the case $k_1^\mu = 1$. So $k_2^\mu = k_1^\nu = k_2^\nu = 0$. This implies the path is a directed edge labelled $l\mu$ from v_x to v_y. By definition of a Mesh Graph then $y_{l\mu} - x_{l\mu} = 1$ and $y_{l\nu} - x_{l\nu} = 0$. Similarly we can prove the basis is true for the other three cases also.

Induction Step: Assume the lemma is true for paths of length $\leq n$. Consider any path fron v_x to v_y of length n+1. If n+1=1 then lemma holds by basis. So assume n+1>1 and let v_z be any intermediate vertex in this path. Let n_1 and n_2 denote the path length from v_x to v_z and v_z to v_y in this path. Clearly $n_1 \leq n$ and $n_2 \leq n$. By applying the induction hypothesis to each of these two paths it follows that the lemma is true for paths of length $\leq n+1$.

□

Now consider any correct mapping of S and let v_x and v_y be any two computation vertices in S. We relate the processors and the times at which they are mapped in the following lemma.

Lemma A.11: $\Delta_P(v_x,v_y)=(y_{l\mu}-x_{l\mu})n_{l\mu}+(y_{l\nu}-x_{l\nu})n_{l\nu}$ and $\Delta_T(v_x,v_y)=(y_{l\mu}-x_{l\mu})d_{l\mu}+(y_{l\nu}-x_{l\nu})d_{l\nu}$

Proof: Straightforward from Lemma A.10 and Lemma A.2

□

We next establish a fundamental property of Mesh Graphs. This property relates the existence of a directed path between two computation vertices in a Mesh Graph to certain relationships between their coordinates. This is useful in the proof of Theorem 3.2 wherein we show that certain graphs in Θ can never be mapped correctly.

To prove this property the following lemma is useful.

Lemma A.12: Let $li \in \{l\mu, l\nu\}$ and let q_m, q_n and q_k be three distinct major paths labelled li. If the indices m,n and k of q_m,q_n and q_k respectively are such that $m<k<n$, then any path between any computation vertex in q_m and any computation vertex in q_n must pass through a computation vertex in q_k.

Proof: Let $li=l\mu$ and let v_x,v_y and v_z be any three computation vertices in q_m,q_n and q_k respectively. Indices of q_m,q_n and q_k are m,n and k respectively and hence $F_{l\nu}(v_x)=m$, $F_{l\nu}(v_y)=n$ and $F_{l\nu}(v_z)=k$.

Now assume that the path does not pass through any computation vertex in q_k. Then the path must traverse an edge labelled $l\nu$ between two computation vertices in major paths q_s and q_r that are labelled $l\mu$ such that if s and r are the indices of q_s and q_r respectively then $s<k<r$. By Lemma A.10, the number of edges labelled $l\nu$ in any path from q_s to q_r is r-s. Since $s<k<r$ and k,r and s are integers, $r-s \geq 2$. But as there is also an edge labelled $l\nu$ between a computation vertex in q_s and a computation vertex in q_r it follows from the definition of a Mesh Graph that r-s=1–a contradiction.

Using similar arguments we can show that the lemma is true for $li=l\nu$.

□

The following result is a straightforward consequence of the previous lemma.

Corollary A.1: Let $li \in \{l\mu, l\nu\}$ and $lj \in \{l\mu, l\nu\}$ and let $li \neq lj$. Let q_m and q_n be two distinct major paths labelled li. If their indices m and n differ by 1 then any path between a computation vertex in q_m and a computation vertex in q_n must traverse an edge labelled lj between computation vertices in q_m and q_n respectively.

Proof: Without loss of generality let m=n+1, where m and n are the indices of q_m and q_n respectively. Now pick a path from some computation vertex in q_m, say v_x, to some computation vertex in q_n, say v_y, such that it does not traverse an edge between any pair of computation vertices in q_m and q_n. Then there must be a computation vertex v_z in this path distinct from v_x and v_y. Let v_z be in the major path q_s. Let s be the index of q_s. If $s>m$ then the path from v_x to v_z violates Lemma A.12 and if $s<m$ then the path from v_z to v_y violates Lemma A.12.

□

We are now ready to establish a fundamental property of Mesh Graphs.

Lemma A.13: Let v_x and v_y be any pair of computation vertices such that $y_{l\mu} \geq x_{l\mu}$ and $y_{l\nu} \geq x_{l\nu}$. Then there must exist a directed path from v_x to v_y.

Proof: Let $y_{l\mu}-x_{l\mu}=m$ and $y_{l\nu}-x_{l\nu}=n$. The proof is an induction on n.

Basis Step: We need to consider the case when $m=0$ and $n \geq 0$ and the case when $m \geq 0$ and $n=0$.

Case 1: $m=0$ and $n \geq 0$. By the definition of a Mesh Graph, there must be a directed path from v_x to v_y in some major path labelled $l\nu$.

Case 2: $m \geq 0$ and $n=0$. By the definition of a Mesh Graph again, there must be a directed path from v_x to v_y in some major path labelled $l\mu$.

Induction Step: Assume the lemma holds for any pair of vertices v_x and v_y such that $0 \leq y_{l\mu}-x_{l\mu} \leq m$ and $0 \leq y_{l\nu}-x_{l\nu} \leq n$. We will show that it holds for any v_x and v_y such that $0 \leq y_{l\mu}-x_{l\mu} \leq m+1$ and $0 \leq y_{l\nu}-x_{l\nu} \leq n+1$. To do this we have to consider the following three cases.

1. $y_{l\mu}-x_{l\mu} \leq m+1$ and $y_{l\nu}-x_{l\nu} \leq n$.
2. $y_{l\mu}-x_{l\mu} \leq m$ and $y_{l\nu}-x_{l\nu}=n+1$.
3. $y_{l\mu}-x_{l\mu}=m+1$ and $y_{l\nu}-x_{l\nu}=n+1$.

The following geometric picture comes in useful in understanding the proof.

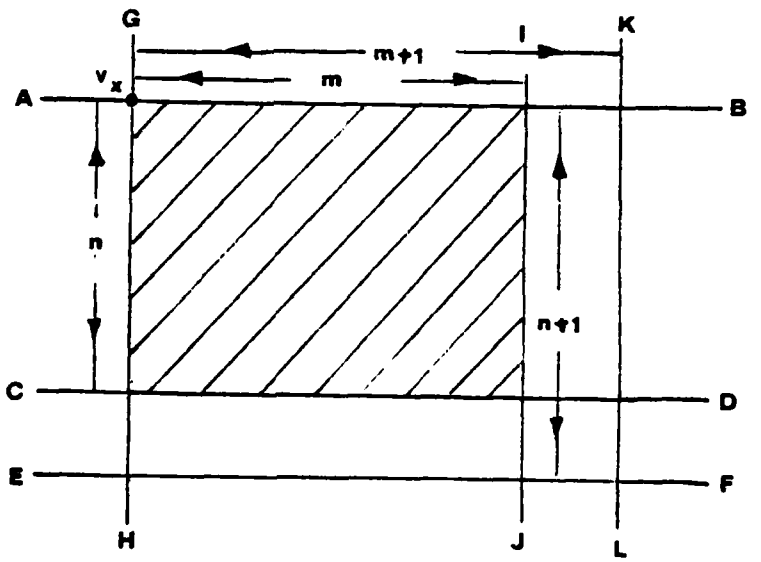


Figure A.8

The lines GH, IJ and KL denote major paths labelled $l\nu$. The index of GH is $x_{l\mu}$ and the indices of IJ and KL are $m+x_{l\mu}$ and $m+1+x_{l\mu}$ respectively. The lines AB, CD and EF denote major paths labelled $l\mu$. The index of AB is $x_{l\nu}$ and the indices of CD and EF are $n+x_{l\nu}$ and $n+1+x_{l\nu}$. The induction hypothesis holds for v_x and any v_y within the region enclosed by AB, CD, GH and IJ which is the shaded region in the above figure .

We first proceed to establish that the lemma holds for any v_y such that $y_{l\mu}-x_{l\mu}=m+1$ and $0\leq y_{l\nu}-x_{l\nu}\leq n$. Consider one such vertex v_y as shown in Figure A.9.

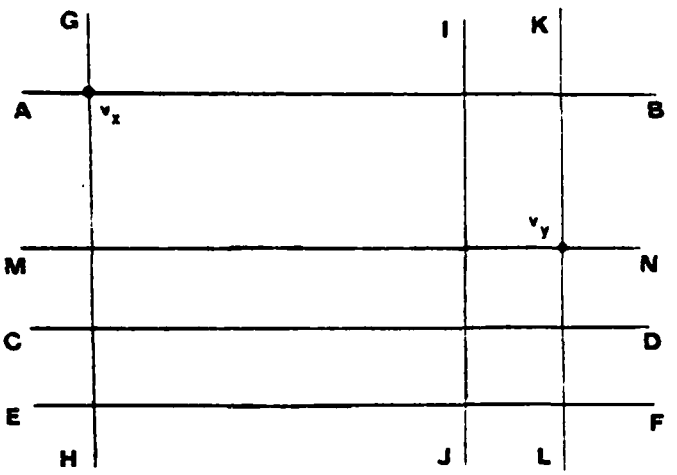


Figure A·9

From Corollary A.1, any path from v_x to v_y must traverse an edge labelled $l\mu$ between vertices in IJ and KL. Let v_u and v_w be the two vertices in IJ and KL respectively. Now v_u and v_w must appear in one of the following three regions in Figure A.9.

1. Above AB
2. Within AB and MN
3. Below MN

Figures A.10(a), A.10(b) and A.10(c) illustrate cases 1, 2 and 3 respectively.

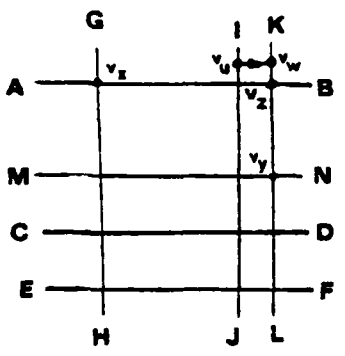


Figure A·10(a)

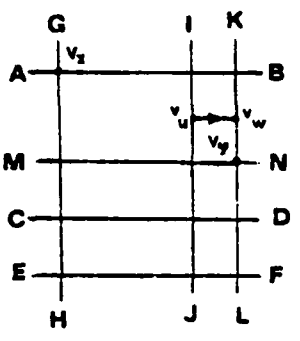


Figure A·10(b)

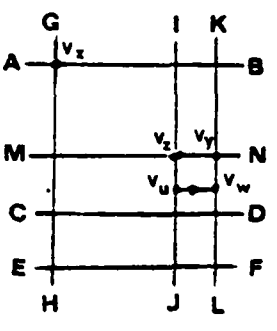


Figure A·10(c)

Case 1: By the definition of a Mesh Graph, v_z must exist in AB. Then there is a directed path from v_x to v_z and from v_z to v_y.

Case 2: By the inductive assumption, there is a directed path from v_x to v_u. The edge labelled $l\mu$ is directed from v_u to v_w. By the Mesh Graph definition there is a directed path from v_w to v_y.

Case 3: We now show that whenever v_u and v_w occurs below MN then v_z must always exist. Suppose not. Then by definition of Mesh Graph there cannot be any vertex on IJ above MN and on MN to the left of IJ. Consider any path p_1 between v_x and any vertex, say v_s on IJ. v_s must be below MN. Then by Lemma A.12, there must exist a vertex, say v_r on MN in the path p_1 and v_r precedes v_s in p_1. As v_z does not exist, v_r must be to the right of IJ. Consider any path p_2 between v_x and v_r. By Lemma A.12 again, there must exist a vertex, say v_t on IJ in the path p_2 and v_t precedes v_r in p_2. So there exists a path between v_x and v_t and no vertex on MN in this path -- a contradiction. So v_z must exist. Therefore by the inductive assumption there is a directed path from v_x to v_z. The edge labelled $l\mu$ is directed from v_z to v_y.

We can similarly show that the lemma holds $\forall v_x$ and $\forall v_y$ such that $y_{l\mu}-x_{l\mu} \leq m$ and $y_{l\nu}-x_{l\nu}=n+1$ and also holds $\forall v_x$ and $\forall v_y$ such that $y_{l\mu}-x_{l\mu}=m+1$ and $y_{l\nu}-x_{l\nu}=n+1$.

□

We have established all the relevant results to prove Theorem 3.2.

Proof: (of Theorem 3.2)

(Only If Part): Consider a correct mapping of G. Now by Theorem 3.1, SG must be a Mesh Graph. We construct a Main Diagonalization of SG as follows. If $\langle n_{l\mu},n_{l\nu}\rangle \in \{\langle 1,1\rangle,\langle 1,-1\rangle,\langle 1,0\rangle,\langle 0,1\rangle\}$ then let $w=\langle w_1,w_2\rangle=\langle n_{l\mu},n_{l\nu}\rangle$ and n_{lj} be the consistency constant of a_{lj}. Otherwise let $w=\langle -n_{l\mu},-n_{l\nu}\rangle$ and $-n_{lj}$ be the consistency constant of a_{lj}.

We will prove that each of the three conditions is necessary when $\langle n_{l\mu},n_{l\nu}\rangle=\langle 1,-1\rangle$ as the proof for any other value that it assumes is similar.

$\langle n_{l\mu},n_{l\nu}\rangle=\langle 1.-1\rangle$ and so by the above construction of a Main Diagonalization the diagonalization factor $w=\langle 1,-1\rangle$. Hence the complementary diagonalization factor $w_c=\langle 0,1\rangle$.

(1) Consider any edge labelled lj directed from v_x to v_y. Now $\Delta_P(v_x,v_y)=n_{lj}\in\{1,-1.0\}$. Also by Lemma A.11, $\Delta_P(v_x,v_y)=(y_{l\mu}-x_{l\mu})-(y_{l\nu}-x_{l\nu})$ and so $\Delta_P(v_x,v_y)=\Delta_D(v_x,v_y)$ and hence a_{lj} is consistent with respect to $\mathcal{T}_D$.
So consistency of a_{lj} with respect to $\mathcal{T}_D$ ensures that adjacent vertices are mapped on neighboring processors.

(2) Now $\Delta_P(v_x,v_y)=(y_{l\mu}-x_{l\mu})-(y_{l\nu}-x_{l\nu})$
$=n_{lj}$
Also $\Delta_T(v_x,v_y)=(y_{l\mu}-x_{l\mu})d_{l\mu}+(y_{l\nu}-x_{l\nu})d_{l\nu}$
and $\Delta_T(v_x,v_y)=d_{lj}$

As $n_{lj},d_{lj},d_{l\mu}$ and $d_{l\nu}$ are all constants, $(y_{l\nu}-x_{l\nu})$ is a constant. $w_c=\langle 0,1\rangle$ and so $\Delta_{Dc}(v_x,v_y)=(y_{l\nu}-x_{l\nu})$ and hence b_{lj} is consistent with respect to $\mathcal{T}_{Dc}$.
Consistency of b_{lj} with respect to $\mathcal{T}_{Dc}$ ensures that elements in a data stream travel at a constant velocity.

(3) Let $a=(y_{l\mu}-x_{l\mu})$ and $b=(y_{l\nu}-x_{l\nu})$. We have already proved that a_{lj} and b_{lj} are consistent with respect to $\mathcal{T}_D$ and $\mathcal{T}_{Dc}$ respectively and hence we easily obtain $d_{lj}=(m_{lj}+c_{lj})d_{l\mu}+c_{lj}d_{l\nu}$.

From Lemma A.11. $\Delta_P(v_x,v_y)=a-b=\Delta_D(v_x,v_y)$ and $\Delta_T(v_x,v_y)=d_{l\mu}a+d_{l\nu}b$.
Now $w_c=\langle 0,1\rangle$ and so $\Delta_{Dc}(v_x,v_y)=b$. Also $c_{lj}\Delta_D(v_x,v_y)=m_{lj}\Delta_{Dc}(v_x,v_y)$ and so $c_{lj}(a-b)=m_{lj}b$.

Now $\Delta_P(v_x,v_y)d_{lj}=(a-b)d_{lj}$
$=[(m_{lj}+c_{lj})d_{l\mu}+c_{lj}d_{l\nu}](a-b)$
$=m_{lj}\Delta_T(v_x,v_y)-(d_{l\mu}+d_{l\nu})[-ac_{lj}+b(m_{lj}+c_{lj})]$
$=m_{lj}\Delta_T(v_x,v_y)$

and so from Lemma A.6, there must be a major path labelled lj passing through v_x and v_y.

Satisfaction of this condition ensures that no two values appear simultaneously at the input port of any processor.

(If Part): Let $D=\{D_1, D_2, .., D_n\}$ be the set of main diagonals where i denotes the index of any $D_i \in D$. Construct a linear array L_{Ar} with $|N|=n$. Now construct a mapping through the following steps.

1. Choose two-phase clocking if there exists a transitive edge labelled lj such that $m_{lj}=0$ or else choose a single-phase clocking scheme.
2. Let D_q be any diagonal in D and let v_x be any computation vertex in D_q. Then, let $PA(v_x)=q$. This assigns computation vertices to processors.
3. Next fix the neighborhood constant n_{lj} and delay constant d_{lj} for every label lj in L_1. Let $n_{lj}=m_{lj}$. Let d_a and d_b be two constants which we will be using in the construction of the delays for the labels in L_1. If the main diagonalization factor w is $<1,-1>$ or there exists a transitive edge labelled lj such that $m_{lj}=0$ then let $d_a=2$ else let $d_a=1$. Let c_{min} be the minimum of all consistency constants among all the relations in S_{Dc}. If $c_{min}>0$ then set $d_b=1$ else set $d_b=1+|c_{min}|d_a$. Let $d_{lj}=m_{lj}d_b+c_{lj}d_a$.
4. Next construct the neighborhood and delay constant for the labels in L_2. By definition of L_2, if there exists a label lj in L_2 then there must exist some label li in L_1 such that for every major path in E_{lj} there is an identical major path in E_{li}. Hence let $n_{lj}=n_{li}$ and $d_{lj}=d_{li}$.
5. For every lj in L_3, let the neighborhood relation imposed by label lj on processors in N be empty and hence no processor's output port labelled lj is connected to the input port labelled lj of any processor.
6. Construct the function TA which assigns computation vertices to time steps. Let v_s be the computation vertex which is in $D_1 \in D$ and $Dc_1 \in Dc$. Let $TA(v_s)=t_0$. Let v_x be any computation vertex in $D_p \in D$ and $Dc_q \in Dc$. Then, let $TA(v_x)=t_0+(q-1)d_a+(p-1)d_b$.

Step 1 to step 6 described above completes the construction of a correct mapping which we establish as follows.

We begin by showing that for any label lj, n_{lj} and d_{lj} are constants. Consider an edge labelled lj from v_x to v_y and let v_x be in D_p and Dc_q and v_y in D_r and Dc_s respectively.

Now $\Delta_D(v_x,v_y)=\Delta_P(v_x,v_y)$
$=r-p$
$=m_{lj}\in\{1,-1,0\}$
$=n_{lj}$

Next $\Delta_T(v_x,v_y)=(s-q)d_a+(r-p)d_b$
$=\Delta_{Dc}(v_x,v_y)d_a+\Delta_D(v_x,v_y)d_b$
$=m_{lj}d_b+c_{lj}d_a$
$=d_{lj}$

Next we show that for any lj if $n_{lj}=0$, then all the vertices mapped onto the same processor belong to the same major path labelled lj. Suppose $n_{lj}=0$. Then $m_{lj}=0$. Consider any v_x and v_y such that $\Delta_D(v_x,v_y)=0$. Then $\Delta_P(v_x,v_y)=0$ and so $c_{lj}\Delta_D(v_x,v_y)=m_{lj}\Delta_{Dc}(v_x,v_y)$. But by condition (2) of the Theorem there must be a major path labelled lj passing through v_x and v_y. So whenever $n_{lj}=0$ and $PA(v_x)=PA(v_y)$, there is always a major path labelled lj passing through v_x and v_y.

We next show that no two values appear simultaneously at the input port of any processor. We have shown that for any label lj, if $n_{lj}=0$ then vertices mapped onto the same processor all belong to the same major path labelled lj and hence no two values of any two distinct major paths labelled lj appear simultaneously at the input port labelled lj of any processor. So we need to consider only major paths labelled lj whose neighborhood constant $n_{lj}\in\{1,-1\}$. Let $n_{lj}=1$ and let q_1 and q_2 be two major paths whose input or output values appear simultaneously at the input port of some processor in the array. Clearly the input values associated with these two paths must be fed simultaneously at the external input port associated with label lj and let t denote this time. Let v_x and v_y be the first vertices of q_1 and q_2 respectively. The time taken by the input value of q_1 to reach $PA(v_x)$ is $t+PA(v_x)d_{lj}$ and the time taken by the input value of q_2 to reach $PA(v_y)$ is $t+PA(v_y)d_{lj}$. Without loss of generality let $PA(v_y)\geq PA(v_x)$ and hence,

$\Delta_T(v_x,v_y) = \Delta_P(v_x,v_y)d_{lj}$ and so

$\Delta_T(v_x,v_y)n_{lj} = \Delta_P(v_x,v_y)d_{lj}$

Now $m_{lj}=n_{lj}$ and $d_{lj}=m_{lj}d_b+c_{lj}d_a$ and so

$[\Delta_{Dc}(v_x,v_y)d_a+\Delta_D(v_x,v_y)d_b]m_{lj}=\Delta_D(v_x,v_y)[m_{lj}d_b+c_{lj}d_a]$ and hence $\Delta_{Dc}(v_x,v_y)m_{lj}=\Delta_D(v_x,v_y)c_{lj}$. But by condition (2) of the Theorem, q_1 and q_2 must be the same major path labelled lj. We can arrive at a similar contradiction when $n_{lj}=-1$.

Lastly we show that $d_{lj}>0$ for any label lj. Consider the case when $w=<1,-1>$. So $w_c=<0,1>$. By construction $d_a>0$ and $d_b>0$. Now $d_{lj}=m_{lj}d_b+2c_{lj}$. We will show that $\forall lj$, $c_{lj}\geq 0$. Let v_x and v_y be vertices such that there is an edge labelled lj from v_x to v_y. So $\Delta_D(v_x,v_y)=(y_{l\mu}-x_{l\mu})-(y_{l\nu}-x_{l\nu})=m_{lj}$. $w_c=<0,1>$ and hence $\Delta_{Dc}(v_x,v_y)=y_{l\nu}-x_{l\nu}$.

Suppose $c_{lj}<0$. Then $y_{l\nu}<x_{l\nu}$. $m_{lj}\in\{1,-1,0\}$ and hence $y_{l\mu}-x_{l\mu}\leq 0$ and so by Lemma A.13, there must be a directed path from v_y to v_x causing a cycle. So $\forall lj$, $c_{lj}\geq 0$. Hence $d_b=1$ and $d_{lj}=m_{lj}+2c_{lj}$.

If $c_{lj}=0$ then we will show that $m_{lj}>0$. Suppose $m_{lj}\leq 0$. Then $\Delta_D(v_x,v_y)\leq 0$. $c_{lj}=0$ and hence $y_{l\nu}=x_{l\nu}$ and hence $y_{l\mu}-x_{l\mu}\leq 0$. So by Lemma A.13, there must be a directed path from v_y to v_x causing a cycle. So $m_{lj}>0$ and hence $d_{lj}>0$. $m_{lj}\in\{1,-1\}$ and hence if $c_{lj}>0$ then $d_{lj}\geq 1$.

For the cases $w=<0,1>$ and $w=<1,0>$ we can show by Lemma A.13 that (a) if $c_{lj}\leq 0$ then $m_{lj}>0$ and (b) if $m_{lj}\leq 0$ and $c_{min}<0$ then $c_{lj}\geq 1+|c_{min}|$. For both these cases we can easily show that $d_{lj}>0$.

□

A.4 Correctness of Mapping Cube Graphs

We had provided a technique for mapping Cube Graphs onto linear arrays. Herein we establish that the mapping is syntactically correct. We begin by first showing that the mapping preserves the neighborhood constant of the labels.

Theorem A.2: Let $l\in L_G$ and let n_l and d_l be its neighborhood and delay constants respectively. Then, if $e=<v_x,v_y>$ 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,

$$w_1x_{l1}+w_2x_{l2}+w_3x_{l3}=w_p$$
and $$w_1y_{l1}+w_2y_{l2}+w_3y_{l3}=w_q$$

Let $l=l1$. Since $e=<v_x,v_y>$ 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, $w_q-w_p=w_1=1$. Now p and q are the indices of D_p and D_q respectively. We next show that $q=p+1$. Suppose $q\neq p+1$. 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 $w_q-w_p=w_1=1$ ..... a contradiction. So $q=p+1=p+w_1$.

The mapping algorithm maps vertices in D_p onto processor p and those of D_q onto processor $p+w_1$ and hence $PA(v_y)=PA(v_x)+w_1$. Also from the mapping algorithm $n_{l1}=w_1$. 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 A.3: Let $l \in L_G$ and let n_l and d_l be its neighborhood and delay constants respectively. Then if $e=<v_x,v_y>$ 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 CG say CG_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 CG_i. Clearly, v_y must be a vertex in some mesh graph in CG_{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 A.14: 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 $PA(v_x)=s_1$, $PA(v_y)=s_2$, $TA(v_x)=t_1$ 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 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.

1. Let $n_l=1$. The input port labelled l of processor 1 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 1 to s_1 and 1 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 $(s_1-s)d_l+t$ 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 $(s_2-s)d_l+t$ which is $TA(v_y)$ and hence,

   $t_2-t_1=(s_2-s_1)d_l$ and so
   $(t_2-t_1)n_l=(s_2-s_1)d_l$
2. Let $n_l=-1$. The input port labelled l of processor $|N|$ is the external input port. So the input value represented by source vertex of P_1 travels from $|N|$ to s_1 passing through the intermediate processor s and the input value represented by source vertex of P_2 travels from $|N|$ 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 $t+(s-s_1)d_l$ and the time taken to reach s_2 by the input value represented by source vertex of P_2 is $t+(s-s_2)d_l$ and hence,

   $t_2-t_1=(s_1-s_2)d_l$ and so
   $(t_2-t_1)n_l=(s_2-s_1)d_l$

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 A.4: 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,

$PA(v_y)-PA(v_x)=\Delta P=k_1 n_{l1}+k_2 n_{l2}+k_3 n_{l3}$

$TA(v_y)-TA(v_x)=\Delta T=k_1 d_{l1}+k_2 d_{l2}+k_3 d_{l3}$

where $k_1=(y_{l1}-x_{l1})$ and $h_1 \leq k_1 \leq h_1$, $k_2=(y_{l2}-x_{l2})$ and $-h_2 \leq k_2 \leq h_2$, $k_3=(y_{l3}-x_{l3})$ and $-h_3 \leq k_3 \leq h_3$.

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 $l1$ of a processor. By lemma A.14,

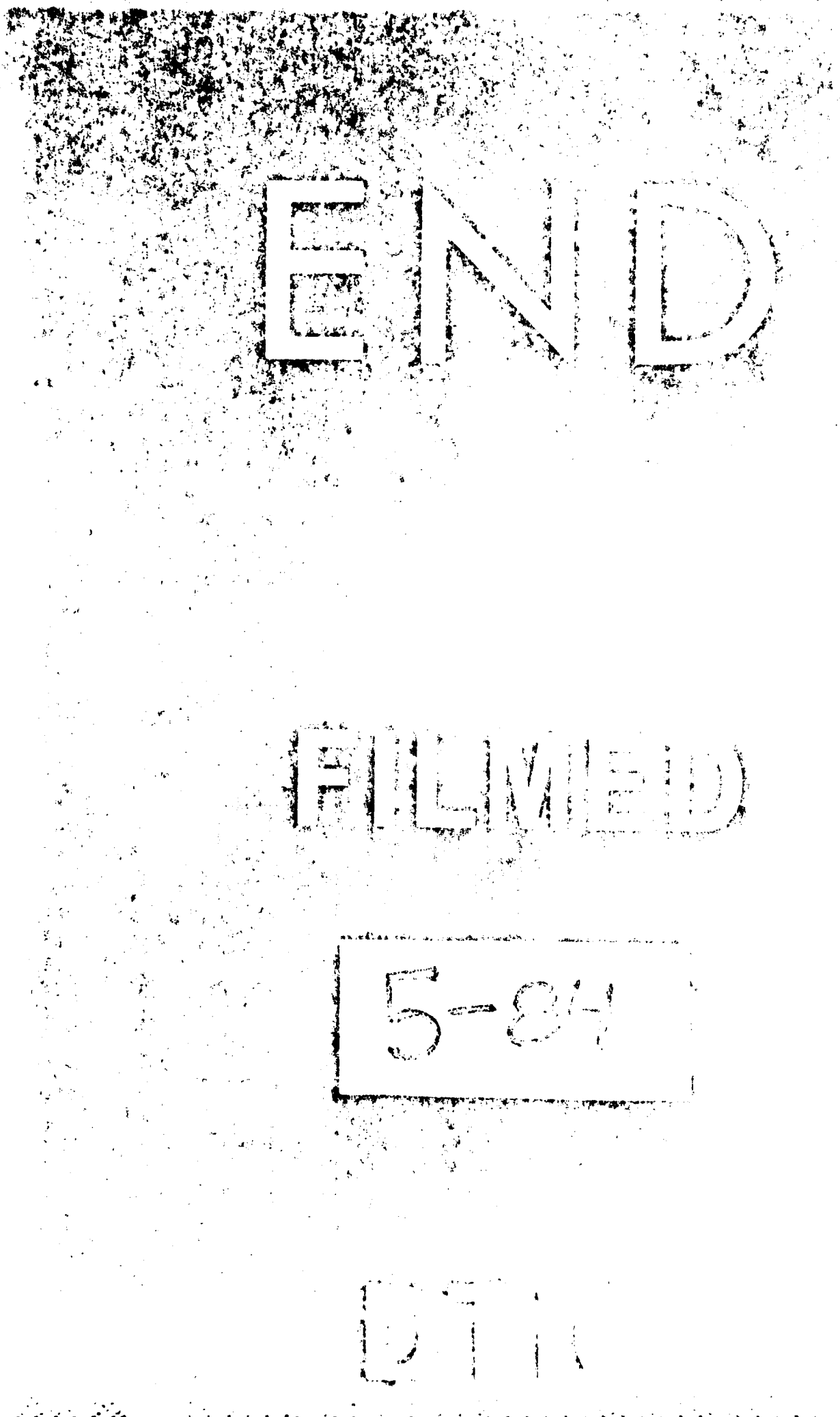
$d_{l1}\Delta P=n_{l1}\Delta T$ (*)

We next show that (*) cannot be satisfied.

1. Let $n_{l2}=1$ and so by the mapping algorithm, $d_{l1}=1$ and $d_{l2}=2$. P_1 and P_2 are distinct major paths labelled $l1$ and so $k_2=k_3 \neq 0$.
   a. Let $h_1-h_2+n_{l3} \geq 0$. So $d_{l3}=h_1+1+2n_{l3}$ and (*) reduces to $k_3(h_1+1+n_{l3})+k_2=0$. Now $h_1+1+n_{l3} \geq 1$ and so $k_2 \neq 0$ and $k_3 \neq 0$. Besides $h_2 \leq h_1+n_{l3}$ and $-h_2 \leq k_2 \leq h_2$ 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 $k_3(h_2+1)+k_2=0$. Now $h_2 \geq 1$ and so $k_2 \neq 0$ and $k_3 \neq 0$. Besides $-h_2 \leq k_2 \leq h_2$ and so (*) cannot be satisfied.
2. Let $n_{l2}=-1$. 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 $2k_2+k_3(2h_2+1)=0$. As $h_2 \geq 1$, so $2h_2+1 \geq 3$ and so $k_2 \neq 0$ and $k_3 \neq 0$. Besides $-h_2 \leq k_2 \leq h_2$ and so $-(2h_2+1) \leq 2k_2 < 2h_2+1$ and so (*) cannot be satisfied.
   b. Let $h_2-h_1+n_{l3}<0$ and so $d_{l3}=2h_1+1-n_{l3}$. So (*) reduces to $2k_2+k_3(2h_1+1-2n_{l3})=0$. Now $1 \leq h_2 < h_1-n_{l3}$. So $2h_1+1-2n_{l3}>1$ and hence $k_2 \neq 0$ and $k_3 \neq 0$. Besides $-h_2 \leq k_2 \leq h_2$ and so $-(2h_1+1-2n_{l3}) < 2k_2 < 2h_1+1-2n_{l3}$ and hence (*) cannot be satisfied.

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

□


END
FILMED
5-84
DTIC