TOWARDS A CHARACTERIZATION OF PROGRAMS FOR A MODEL OF VLSI ARRAY-PROCESSORS[1]

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

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

TR-202 July 1982

---

[1]This research was supported in part by the National Science Foundation under Grant MCS-8104017.

ABSTRACT

This paper is concerned with understanding the properties of programs correctly executable on a model of linear array processors suitable for VLSI. Such a model has been proposed as an attractive architecture to handle compute-bound problems in an efficient and cost-effective manner. We provide a complete syntactic characterization for a class of programs that are correctly executable on this model and show that among the class of syntactically characterized programs the sub-class of programs correctly executable without semantics is very limited. We then identify the semantic properties required of programs in this entire syntactic class inorder for them to be correctly executable.

1. Introduction

Interests in parallel processing were created by the emergence of parallel computers namely ILLIAC. The recent advent of large-scale integration technology has further stimulated interests in parallel processing [2, 3, 4, 6, 8]. VLSI offers the potential of realizing parallel computations in silicon. Such a realization can be made cost-effective and modular provided:

1. The processors used are simple and uniform.

2. The processors are connected by a modular, simple and regular interconnection network and this implies that programs executed on such a network exhibit simple and regular data and control flow.

The realization is rendered efficient by extensive use of pipelining and multiprocessing. These notions of simplicity and regularity will be made more precise in the subsequent sections.

In this paper we have characterized the properties of programs correctly executable on a model of linear array processors for VLSI. We have distinguished the roles of two types of properties in a correct mapping (execution) of programs onto such processor-arrays, namely-

1. Syntactic - i.e, the structure of programs

2. Semantic - i.e., some knowledge of what the programs do

The main results in this paper are lemma 4.0-5, theorem 5.1-1, theorem 5.2-1 and theorem 6.1-1. Our paper is organized as follows:

In section-2 we introduce the program and linear array models. In section-3 we define the problem of correct execution of programs on a linear array. In section-4 and section-5 we provide a syntactic characterization of correctly executable programs and in section-6 we demonstrate the importance of knowing the semantics of the syntactically characterized programs inorder for them to be correctly executable. In section-7 we illustrate the characterization by synthesizing algorithms for two important computational problems.

2. Program and Linear-Array Models

In this section we will describe our program and linear-array models. We will also make the notions of simplicity and uniformity of processors more precise.

2.1. Graph and Set-Theoretic Preliminaries

Let $A=\{a_1, a_2, .. , a_n\}$ be a set of elements. Let R be a binary relation on A. Let R^+ denote the transitive closure of R.

Definition 2.1-1: R is total on A iff for any a_i and a_j in A, either $a_i\ R^+\ a_j$ or $a_j\ R^+\ a_i$.

Let $S=\{s_1, s_2, .., s_m\}$ be a set of binary relations on A.

Let $G_S=(V, E)$ be the directed graph induced by all the relations in S on A with V=A and $E=\{\langle a_i, a_j\rangle |\ a_i\ s_x\ a_j$ for some s_x in S$\}$.

Let $G_x=(V_x, E_x)$ be the subgraph of G_S induced by the relation s_x in S on A with $V_x=V$ and $E_x=\{\langle a_i, a_j\rangle |\ a_i\ s_x\ a_j\}$.

Definition 2.1-2: S imposes a consistent order on A iff

1. there exists at least one relation s_x in S such that s_x is total and G_x is acyclic

2. for every s_y in S there is a constant c_y associated with s_y such that for any a_i and a_j in A, if a_i s_y a_j then $\pi_A(a_j)=\pi_A(a_i)+c_j$ where π_A is the indexing function that maps every element in A to its position in the total order imposed by a toplological sort[1] on the vertices in G_x.

We will call c_y the consistency constant of s_y.

Definition 2.1-3: A relation s_x in S imposes a linear chain on A iff S imposes a consistent order on A and $c_x=1$.

Example 2.1-1: Let $A=\{a_1, a_2, a_3, a_4\}$ and $S=\{s_1, s_2\}$ such that a_2 s_1 a_1, a_1 s_1 a_4, a_4 s_1 a_3, a_2 s_2 a_4 and a_1 s_2 a_3. Set $s_x=s_1$. G_S and G_x are

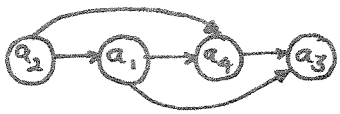


Figure 2.1-1: G_s

Figure 2.1-2: G_x

The indexing function π_A is $\pi_A(a_2)=1$, $\pi_A(a_1)=2$, $\pi_A(a_4)=3$ and $\pi_A(a_3)=4$. The consistency constants for s_1 and s_2 are 1 and 2 respectively. S

[1]In this paper we will be assuming that if V is a set of vertices then the total order imposed by a topological sort on V is a set of consecutive integers ranging from $\emptyset$ to $|V|-1$.

imposes a consistent order on A and s_1 imposes a linear chain on A.

Example 2.1-2: Let A, s_1 and s_2 be defined as in example 2.1-1. Let $S=\{s_1, s_2, s_3\}$ where $a_2\ s_3\ a_3$ and $a_1\ s_3\ a_4$. G_S is shown in figure 2.1-3.

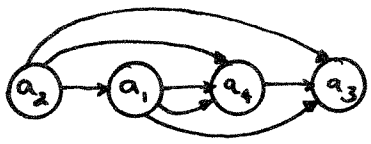


Figure 2.1-3

S does not impose a consistent order on A as s_3 does not have a consistentcy constant.

Definition 2.1-4: Consider a labelled directed graph $G=\langle V_G, E_G, SO_G, SI_G, L_G, G_E, G_{IO}\rangle$ where:

1. V_G, SO_G and SI_G are three distinct sets of vertices with SO_G as the set of source vertices, SI_G as the set of sink vertices and V_G as the set of remaining vertices called computation vertices.

2. E_G is a set of edges

3. L_G is a set of labels

4. G_E and G_{IO} are two many-one functions such that:

   a. G_E : E_G ---> L_G

b. $G_{IO} : SI_G \cup SO_G \longrightarrow L_G$

G is a uniform graph iff it satisfies the following properties

1. Every vertex in SO_G (SI_G) has exactly one edge directed from (to) it to (from) some vertex in V_G. The labels of any vertex in SO_G (SI_G) and the edge directed from (to) it are the same.

2. For any vertex in V_G all the edges directed to (from) the vertex have distinct labels and the number of edges directed to (from) the vertex is equal to $|L_G|$.

Example 2.1-3:

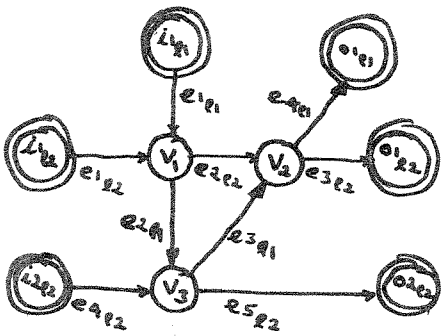


Figure 2.1-4: Uniform Graph

$L_G=\{11, 12\}$ and $V_G=\{v_j \mid 1<j<3\}$.

$E_{11}=\{ej_{11} \mid 1<i<5\}$, $E_{12}=\{ej_{12} \mid 1<i<4\}$ and $E_G=E_{11} \cup E_{12}$.

$SI_{11}=\{ij_{11} \mid 1<j<2\}$, $SI_{12}=\{il_{12}\}$ and $SI_G=SI_{11} \cup SI_{12}$

$SO_{11}=\{oj_{11} \mid 1<j<2\}$, $SO_{12}=\{ol_{12}\}$ and $SO_G=SO_{11} \cup SO_{12}$.

The label of edges in E_{11}, the vertices in SO_{11} and SI_{11} is 11. The

label of edges in E_{12}, the vertices in SO_{12} and SI_{12} is 12.

Henceforth G will always denote a uniform graph.

Definition 2.1-5: For any label l in G, a major path labelled l 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 l.

Let r_l be a binary relation on major paths where l is some label in G.

Definition 2.1-6: For any pair of major paths q_p and q_r in G, q_p r_l q_r iff there exist computation vertices v_x and v_y in q_p and q_r respectively such that there is an edge labelled l directed from v_x to v_y.

We will illustrate all the above graph-theorectic definitions by two examples.

Example 2.1-4: In example 2.1-3 the major path labelled 11 is the path directed from il_{11} to ol_{11} through vertices v_1, v_3 and v_2 in that order. The edges in this path are el_{11}, $e2_{11}$, $e3_{11}$ and $e4_{11}$. There are two major paths labelled 12. One path is directed from il_{12} to ol_{12} through vertices v_1 and v_2 in that order. The edges in this path are el_{12}, $e2_{12}$ and $e3_{12}$. The other major path is the path directed from $i2_{12}$ to $o2_{12}$ through v_3. The edges are $e4_{12}$ and $e5_{12}$ in this path.

Example 2.1-5:

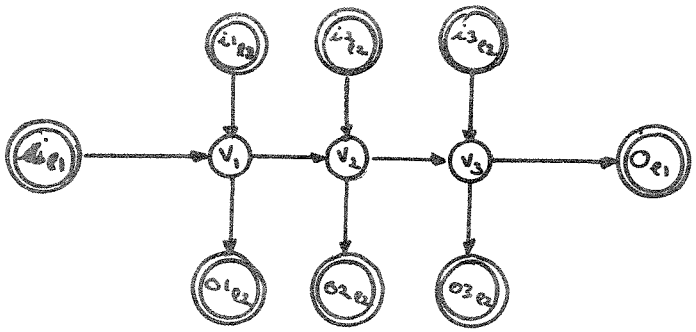


Figure 2.1-5

In figure 2.1-5, $L_G=\{11,12\}$, the set of computation vertices $V_G=\{v_1, v_2, v_3\}$, the set of source vertices $SO_G=\{i_{11}, i1_{12}, i2_{12}, i3_{12}\}$ and the set of sink vertices $SI_G=\{o_{11}, o1_{12}, o2_{12}, o3_{12}\}$. The vertices i_{11} and o_{11} are labelled 11 and the vertices $i1_{12}$, $i2_{12}$, $i3_{12}$, $o1_{12}$, $o2_{12}$ and $o3_{12}$ are labelled 12. The horizontal edges are labelled 11 and the vertical edges are labelled 12. The major path labelled 11 is the horizontal path and the major paths labelled 12 are the three vertical paths. The relation r_{11} is non-empty while r_{12} is empty.

Definition 2.1-7: Two computation vertices v_x and v_y are transitively related by an edge e_a directed from v_x to v_y iff there exists a computation vertex v_z and two edges such that one edge is directed from v_x to v_z and the other edge is directed from v_z to v_y.

Example 2.1-6:

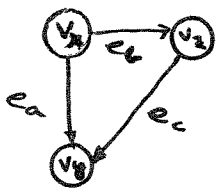


Figure 2.1-6

In figure 2.1-6 e_a relates v_x and v_y transitively.

Definition 2.1-8: Two major paths q_p and q_r are identical iff:

1. the computation vertices in q_p and q_r are identical.

2. for any computation vertex v_x in q_p and q_r, its index in any topological sort of the vertices in q_p is the same as its order in any topological sort of the vertices in q_r.

Example 2.1-7:

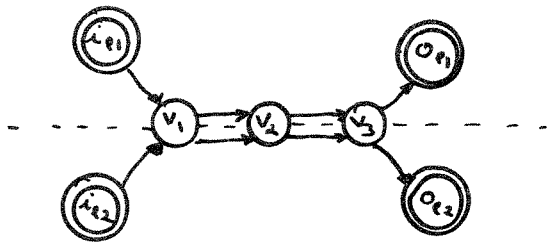


Figure 2.1-7

In figure 2.1-7 v_1, v_2 and v_3 are computation vertices, i_{11} and i_{12} are source vertices labelled 11 and 12 respectively. o_{11} and o_{12} are two sink vertices labelled 11 and 12 respectively. The edges above the

dotted line are labelled 11 and the edges below the dotted line are labelled 12. q_p is the major path labelled 11 and is directed from i_{11} to o_{11} through edges shown above the dotted line. q_r is the major path labelled 12 and is directed from i_{12} to o_{12} through edges shown below the dotted line. q_p and q_r are identical.

2.2. Program Model

Let Y_1, Y_2, .., Y_k be n sets of scalar elements. We do not wish to provide a formal definition of scalar elements. Suffice it is to say that a scalar element is a typical value held in a memory location of a machine like IBM/36Ø. Let $Y=Y_1 \times Y_2 \times .. \times Y_k$ be the cartesian product of k sets. Let x_i denote the i^{th} component of a tuple x in Y and so x_i is in Y_i.

Definition 2.2-1: A program is a one-one function Ψ : D ---> R where $D \subseteq Y$, $R \subseteq Y$ and

1. for any pair of tuples x and z in D, $x_i \neq z_i$

2. for any pair of tuples x and z in R, $x_i \neq z_i$

3. for any pair of tuples x and z where x is in D and z is in R and if $\Psi(x)=z$ then $x_i \neq z_i$

[Note: (1), (2) and (3) are not restrictive as multiple occurrences can be replaced by distinct elements.]

Let $D_i=\{x_i \mid x_i$ is the i^{th} component of tuple x in D $\}$ and $R_i=\{x_i \mid x_i$ is the i^{th} component of tuple x in R $\}$. Let $D_i'=\{x_i \mid x_i \notin D_i$ and

$x_i \notin R_i$ } and $R_i' = \{x_i \mid x_i \in R_i$ and $x_i \notin D_i$ }. Let Ψ_i : D ---> R_i be the projection function of Ψ -- i.e., if $\Psi(x)=z$ then $\Psi_i(x)=z_i$.

Let G be a uniform graph and Ψ be some program with domain D and range R. Let D and R be subsets of the cartesian product of k sets. Let $L_G=\{11, 12, .., 1k\}$.

Definition 2.2-2: A program Ψ is transformed to G by a set $TR=\{TR_1, TR_2, TR_3, TR_4\}$ of one-one functions where:

1. TR_1 : D ---> V_G.

2. TR_2 : $(D_1 \cup R_1) \cup (D_2 \cup R_2) \cup .. \cup (D_k \cup R_k)$ ---> E_G. If $x_i \in \{D_i \cup R_i\}$ and if $TR_2(x_i)=e_a$ then $G_E(e_a)=1i$.

3. TR_3 : $D_1' \cup D_2' \cup .. \cup D_k'$ ---> SO_G. If $x_i \in D_i'$ and if $TR_3(x_i)=v_x$ then $G_{IO}(v_x)=1i$.

4. TR_4 : $R_1' \cup R_2' \cup .. \cup R_k'$ ---> SI_G. If $x_i \in R_i'$ and if $TR_4(x_i)=v_x$ then $G_{IO}(v_x)=1i$.

5. for every x and z if $\Psi(x)=z$ and $TR_1(x)=v_x$, $TR_3(x_i)=e_a$ and $TR_3(z_i)=e_b$ then e_a is directed into v_x and e_b is directed out of v_x.

TR always transforms Ψ into a uniform graph. For any i, every element in Y_i is assigned either an edge labelled 1i or a source or sink vertex labelled 1i by the transformation functions. Henceforth we will

refer to uniform graphs as program graphs and we will denote a program graph as G. Our program graphs are restricted versions of data-flow graphs. Unlike in program graphs the computation vertices in data-flow graphs represent different computable functions. In the following example we will illustrate a program and it's transformation into a program graph.

Example 2.2-1: Consider the problem of multiplying a band matrix M by a vector X as shown in figure 2.2-1.

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

Figure 2.2-1: Band Matrix Multiplication by a Vector

The elements in the product vector Y can be computed by the following recurrence:

$$y_i^{(k+1)} = y_i^{(k)} + a_{ik} *^1 x_k$$

This recurrence can be rewritten as:

$$y_i^{(k+1)} = y_i^{(k)} + a_{ik}^{(1)} * x_k^{(i)}$$

---

[1] throughout this paper '*' denotes multiplication unless explicitly mentioned otherwise

$a_{ik}^{(2)} = a_{ik}^{(1)}$

$x_k^{(i+1)} = x_k^{(i)}$

Define Y_1, I_1 and O_1 as follows:

1. $Y_1=\{y_i^{(k)} \mid 1\leqslant i\leqslant 6,\ 1\leqslant k\leqslant 2+i$ for $1\leqslant i\leqslant 3$ and $i-2\leqslant k\leqslant 6$ for $4\leqslant i\leqslant 6\}$

2. $I_1=\{y_i^{(1)}$ and $y_j^{(j-2)} \mid 1\leqslant i\leqslant 3$ and $4\leqslant j\leqslant 6\}$. Every $y_i^{(1)}=y_j^{(j-2)}=\emptyset$

3. $O_1=\{y_i^{(2+i)}$ and $y_j^{(6)} \mid 1\leqslant i\leqslant 4,\ 5\leqslant j\leqslant 6\}$. For every $y_i^{(2+i)}$ and $y_j^{(6)}$ in O_1, $y_i^{(2+i)}=y_i$ and $y_j^{(6)}=y_j$

Define Y_2, I_2 and O_2 as follows:

1. $Y_2=\{a_{ik}^{(j)} \mid 1\leqslant j\leqslant 2,\ 1\leqslant i\leqslant 6,\ 1\leqslant k\leqslant i+1$ for $1\leqslant i\leqslant 3$ and $i-2\leqslant k\leqslant 5$ for $4\leqslant i\leqslant 6\}$

2. $I_2=\{a_{ik}^{(1)} \mid i$ and k vary as in $Y_2\}$. For every $a_{ik}^{(1)}$ in I_2, $a_{ik}^{(1)}=a_{ik}$.

3. $O_2=\{a_{ik}^{(2)} \mid i$ and k vary as in $Y_2\}$.

Define Y_3, I_3 and O_3 as follows:

1. $Y_3=\{x_k^{(i)} \mid 1\leqslant k\leqslant 5,\ i\leqslant i\leqslant 3+k$ for $i\leqslant k\leqslant 2,\ 2\leqslant i\leqslant 6$ for $k=3$ and $k-1\leqslant i\leqslant 7$ for $4\leqslant k\leqslant 5\}$.

2. $I_3=\{x_k^{(1)}$ and $x_j^{(j-1)} \mid 1\leqslant k\leqslant 2,\ 3\leqslant j\leqslant 5\}$. For every $x_k^{(1)}$ and

$x_j^{(j-1)}$ in I_3, $x_k^{(1)}=x_k$ and $x_j^{(j-1)}=x_j$.

3. $O_3=\{x_k^{(3+k)}$ and $x_5^{(7)} \mid 1\leqslant k\leqslant 4\}$.

Define Ψ, Ψ_1, Ψ_2 and Ψ_3 as follows:

1. $\Psi(\langle y_i^{(k)}, a_{ik}^{(1)}, x_k^{(i)}\rangle) = \langle y_i^{(k+1)}, a_{ik}^{(2)}, x_k^{(i+1)}\rangle$

2. $\Psi_1(\langle y_i^{(k)}, a_{ik}^{(1)}, x_k^{(i)}\rangle) = y_i^{(k+1)} = y_i^{(k)} + a_{ik}^{(1)} * x_k^{(i)}$

3. $\Psi_2(\langle y_i^{(k)}, a_{ik}^{(1)}, x_k^{(i)}\rangle) = a_{ik}^{(2)} = a_{ik}^{(1)}$

4. $\Psi_3(\langle y_i^{(k)}, a_{ik}^{(1)}, x_k^{(i)}\rangle) = x_k^{(i+1)} = x_k^{(i)}$

Define $D=\{\langle y_i^{(k)}, a_{ik}^{(1)}, x_k^{(i)}\rangle\}$ where the tuple exists iff $y_i^{(k)}$, $a_{ik}^{(1)}$, $x_k^{(i)}$ are all defined in Y_1, Y_2 and Y_3 respectively.

Now $Y_1=D_1\cup R_1$, $Y_2=D_2\cup R_2$, $Y_3=D_3\cup R_3$. Also $D_1'=I_1$, $R_1'=O_1$, $D_2'=I_2$, $R_2'=O_2$, $D_3'=I_3$ and $R_3'=O_3$.

Define a uniform graph G as follows:

$L_G=\{11, 12, 13\}$.

$V_G=\{v_{ik} \mid 1\leqslant i\leqslant 6,\ 1\leqslant k\leqslant i+1$ for $1\leqslant i\leqslant 3$ and $i-2\leqslant k\leqslant 5$ for $4\leqslant i\leqslant 6\}$.

$E_G=EY_1\cup EY_2\cup EY_3$ where:

1. $EY_1=\{ey_i^{(k)} \mid$ i and k vary as in $Y_1\}$ and for every $ey_i^{(k)}$, $G_E(ey_i^{(k)})=11$.

2. $EY_2=\{ea_{ik}^{(j)} \mid$ i, k and j vary as in Y_2 $\}$ and for every $ea_{ik}^{(j)}$, $G_E(ea_{ik}^{(j)})=12$.

3. $EY_3=\{ex_k^{(i)} \mid$ i and k vary as in Y_3 $\}$ and for every $ex_k^{(i)}$, $G_E(ex_k^{(i)})=13$.

$SO_G=SO_1 \cup SO_2 \cup SO_3$ where:

1. $SO_1=\{vy_i^{(1)}$ and $vy_j^{(j-2)} \mid$ i and j vary as in I_1 $\}$ and for every $vy_i^{(1)}$ and $vy_j^{(j-2)}$, $G_{IO}(vy_i^{(1)})=G_{IO}(vy_j^{(j-2)})=11$.

2. $SO_2=\{va_{ik}^{(1)} \mid$ i and k vary as in I_2 $\}$ and for every $va_{ik}^{(1)}$, $G_{IO}(va_{ik}^{(1)})=12$.

3. $SO_3=\{vx_k^{(1)}$ and $vx_j^{(j-1)} \mid$ k and j vary as in I_3 $\}$ and for every $vx_k^{(1)}$ and $vx_j^{(j-1)}$, $G_{IO}(vx_k^{(1)})=G_{IO}(vx_j^{(j-1)})=13$.

$SI_G=SI_1 \cup SI_2 \cup SI_3$ where:

1. $SI_1=\{vy_i^{(2+i)}$ and $vy_j^{(6)} \mid$ i and j vary as in O_1 $\}$ and for every $vy_i^{(2+i)}$ and $vy_j^{(6)}$, $G_{IO}(vy_i^{(2+i)}) = G_{IO}(vy_j^{(6)}) = 11$.

2. $SI_2=\{va_{ik}^{(2)} \mid$ i and k vary as in $O_2\}$ and for every $va_{ik}^{(2)}$, $G_{IO}(va_{ik}^{(2)})=12$.

3. $SI_3=\{vx_k^{(3+k)}$ and $vx_5^{(7)} \mid$ k varies as in O_3 $\}$ and for every $vx_k^{(3+k)}$, $G_{IO}(vx_k^{(3+k)})=G_{IO}(vx_5^{(7)})=13$.

Define TR_1, TR_2, TR_3 and TR_4 as follows:

1. $TR_1(\langle y_i^{(k)}, a_{ik}^{(l)}, x_k^{(i)} \rangle) = v_{ik}$

2. $TR_2(y_i^{(k)}) = ey_i^{(k)}$, $TR_2(a_{ik}^{(j)}) = ea_{ik}^{(j)}$ and $TR_2(x_k^{(i)}) = ex_k^{(i)}$.

3. $TR_3(y_i^{(1)}) = vy_i^{(1)}$ and $TR_3(y_j^{(j-2)}) = vy_j^{(j-2)}$; $TR_3(a_{ik}^{(1)}) = va_{ik}^{(1)}$; $TR_3(x_k^{(1)}) = vx_k^{(1)}$ and $TR_3(x_j^{(j-1)}) = vx_j^{(j-1)}$.

4. $TR_4(y_i^{(2+i)}) = vy_i^{(2+i)}$ and $TR_4(y_j^{(6)}) = vy_j^{(6)}$; $TR_4(a_{ik}^{(2)}) = va_{ik}^{(2)}$; $TR_4(x_k^{(3+k)}) = vx_k^{(3+k)}$ and $TR_4(x_5^{(7)}) = vx_5^{(7)}$.

The resulting program graph is shown in figure 2.2-2.

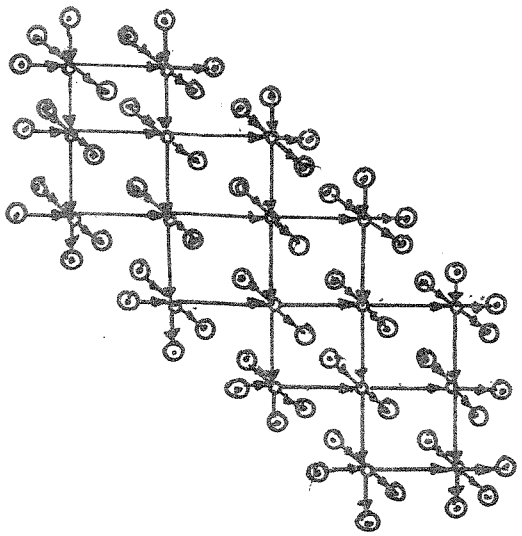


Figure 2.2-2

In figure 2.2-2 'o' represents computation vertices and '⊙' represents source or sink vertices. The horizontal, vertical and oblique edges are labelled 11, 13 and 12 respectively. If a source or sink vertex is connected to a computation vertex by a horizontal, vertical or oblique edge then that source or sink vertex is labelled 11, 13 and 12 respectively. A computation vertex in figure figure 2.2-2 is shown in

greater detail in figure 2.2-3.

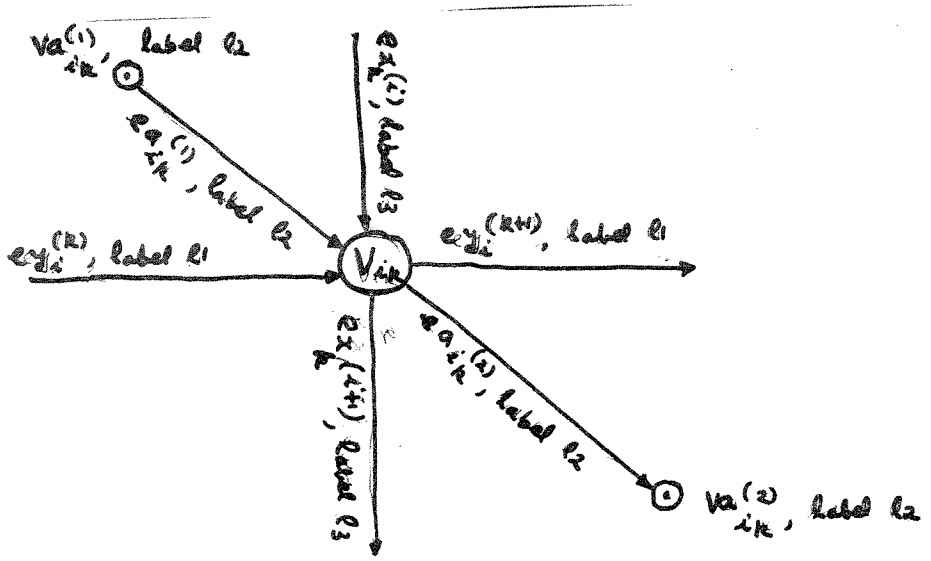


Figure 2.2-3

We will be often referring to the contents or the element represented by an edge or a source vertex or a sink vertex with the explicit understanding that the contents or the element represented refere to the element in some domain -- say Y_j that has been transformed to either an edge or a source vertex or a sink vertex by the transformation functions.

2.3. Processor and Linear-Array Models

A general-purpose processor model consists of an addressable-memory, a control-unit and an instruction-set. During a cycle the processor executes an instruction from its instruction-set where a cycle is the total time taken starting from the time to fetch an instruction to completion of the instruction. In every cycle the processor fetches an instruction from memory, decodes the instruction, fetches the operands from memory, executes the instruction and stores the results in memory. Obviously the processor needs decision-making ability to:

1. execute different instructions in different cycles

2. to access different operand addresses in memory during instruction execution

This decision-making ability is achieved by having states and making state-transitions. In every cycle the control-unit of a processor receives control-inputs (instruction and operand addresses) and determines on the basis of the control-inputs the state the processor should be in.

In this paper we will be using a very simple model of a processor. Our processor-model does not have any decision-making ability. The implications are:

1. the processor must execute the same instruction in every cycle

2. the operand addresses for the instruction is the same in every cycle

We give a formal definition of a processor in the following:

Definition 2.3-1: A processor is a 8-tuple $P=\langle I_P, O_P, D_P, L_P, F_I, F_O, F_D, \Psi_P \rangle$ where:

1. I_P is a set of input ports

2. O_P is a set of output ports

3. D_P is a set of non-zero positive integers

4. L_P is a set of labels

5. F_I, F_O and F_D are three one-one functions such that:

   a. $F_I : I_P \longrightarrow L_P$

   b. $F_O : O_P \longrightarrow L_P$

   c. $F_D : D_P \longrightarrow L_P$

6. Ψ_P is a $|L_P|$-ary function computed by the processor in every cycle, .i.e., if IN_c is the input to P in cycle c and OUT_c is the output computed by P in cycle c then $\Psi_P(IN_c)=OUT_c$ and IN_c and OUT_c are both $|L_P|$-tuples.

Let $L_P=\{l1, l2, .., lk\}$ for a processor P. If $x \in D_P$ and $F_D(x)=lj$ then we will denote x as d_{lj} and refer to it as the delay associated with label lj. Now $|L_P|=k$. Every input-port in I_P is assigned a unique label from L_P. Similarly every output-port in O_P is also assigned a unique label from L_P and so $|I_P|=|O_P|=k$. The function Ψ_P is the instruction executed by P in every cycle. Ψ_P is a 'simple' function, .i.e., it can be computed using a 'few' instructions from the instruction-set of a machine like IBM/360.

Let the k-tuple IN_c be the input to processor P in cycle c. The processor computes the k-tuple $OUT_c=\Psi_P(IN_c)$ in cycle c. Ψ_P can be

decomposed as the cross-product of k functions, .i.e., $OUT_c=\langle\Psi_1(IN_c)\ X\ \Psi_2(IN_c)\ X\ ..\ X\ \Psi_k(IN_c)\rangle$. Now each component of IN_c and OUT_c is a scalar value, .i.e., a value typically held in a memory location like IBM/360. Processor P receives every component of IN_c from a distinct input-port and outputs every component of OUT_c onto a distinct output-port. Let $IN_c=\langle in_{11}, in_{12}, .., in_{1k}\rangle$ and $OUT_c=\langle out_{11}, out_{12}, .., out_{1k}\rangle$. Now in_{1j} is the j^{th} component of IN_c that is received by the processor at the input-port labelled 1j and out_{1j} is the j^{th} component of OUT_c that the processor outputs on the output-port labelled 1j after a delay of d_{1j} cycles, .i.e., in cycle $c+d_{1j}$. Also $out_{1j}=\Psi_j(IN_c)$. However in every cycle between c and $c+d_{1j}$ a new output-tuple is computed by the processor. Now each of these output-tuples is comprised of a component labelled 1j (out_{1j}) and so the number of such components produced beteween c and $c+d_{1j}$ is d_{1j}. Each of these output component produced is outputted from the output-port labelled 1j only after cycle $c+d_{1j}$ and so all these values need to be buffered in a queue. For every label 1j such a queue is implemented by a shift-register of length $d_{1j}-1$. The rightmost end (output of shift-register) of the shift-register is connected to the output port labelled 1j. During every cycle c processor P does the following:

1. compute $OUT_c=\Psi_P\langle IN_c\rangle$

2. for all 1j, shift the contents of the shift-register to the right by one position. This creates a vacant position at the leftmost end (input of shift-register). Place out_{1j} in this vacant slot.

Example 2.3-1:

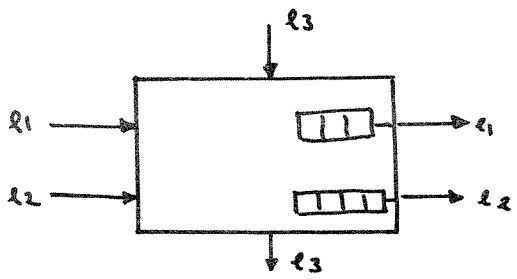


Figure 2.3-1: A Processor

In figure 2.3-1 L_P={11, 12, 13}. d_{11}=4, d_{12}=5 and d_{13}=1.

Definition 2.3-2: Two processors are identical iff:

1. the label-sets are the same

2. the functions computed are identical

3. identical labels in both the processors have the same delays

Let PR be a totally ordered set of identical processors. The processors are numbered {Ø, 1, .., |PR|-1 }. We will refer to a processor in this set by its index number in the ordering. For every label lj let B_{lj} be a binary relation on PR.

Definition 2.3-3: For any pair of processors m and n in PR, m B_{lj} n iff the output port labelled lj of m is connected to the input port labelled lj of n.

B_{lj} is a neighbourhood relation on processors. If two processors are

related by B_{1j} then they are neighbours with respect to the label 1j.

Definition 2.3-4: B_{1j} is empty iff there exists no pair m and n in PR such that m B_{1j} n and is non-empty otherwise.

Definition 2.3-5: A linear-array of interconnected processors is a 4-tuple LA=<PR, L_p, Ψ_p, N_p> where:

1. PR is a totally ordered set of identical processors

2. L_p is the label-set of every processor in PR

3. Ψ_p is the function computed by every processor in PR

4. N_p={n_{1j} iff B_{1j} is non-empty} is a set of constants such that every n_{1j} is only one of {+1, -1, $\emptyset$}.

If n_{1j}=1 then it means that for all processors m its neighbour with respect to label 1j is processor m+1.

If n_{1j}=-1 then it means that for all processors m its neighbour with respect to label 1j is processor m-1.

If n_{1j}=$\emptyset$ then it means that for all processors m its neighbour with respect to label 1j is processor m itself.

The linear array of processors is driven by a single-phase global clock. In every clock pulse all the processors compute Ψ_p. The cycle time for a processor is the width of the clock.

The input-ports and output-ports of some processors are designated to interact with the external environment. Communication with the external environment occurs only through these ports. These input-ports and output-ports are the distinguished input-ports and output-ports respectively. Data is driven into the array only through the distinguished input-ports and the results are obtained through the distinguished output-ports. For every label lj this designation depends on B_{lj}, namely:

1. if B_{lj} is empty the input-port and output-port labelled lj of every processor is the distinguished input-port and output-port respectively.

2. if $n_{lj}=1$ then the input-port labelled lj of processor numbered $\emptyset$ and the output-port labelled lj of processor numbered $|PR|-1$ are the distinguished input-ports and output-ports respectively.

3. if $n_{lj}=-1$ the the input-port labelled lj of processor numbered $|PR|-1$ and the output-port of processor numbered $\emptyset$ are the distinguished input-port and output-port respectively.

4. if $n_{lj}=\emptyset$ then there are no distinguished input-ports and output-ports labelled lj.

Example 2.3-2:

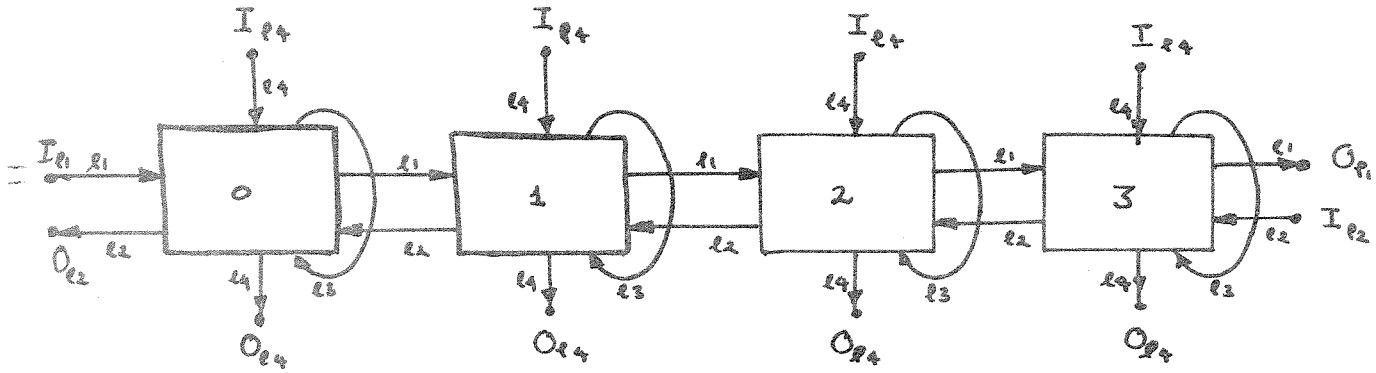


Figure 2.3-2: Linear Array of Processors

In figure 2.3-2 |PR|=4 and L_p={11, 12, 13, 14}. B_{14} is empty and so N_p={n_{11}, n_{12}, n_{13}} and n_{11}=1, n_{12}=-1 and n_{13}=$\emptyset$. I_{11} and O_{11} are the distinguished input-ports and output-ports for 11. I_{12} and O_{12} are the distinguished input-ports and output-ports for 12. I_{14} and O_{14} are the distinguished input-ports and output-ports for 14. 13 does not have any distinguished input-ports and output-ports. A simple register serves as the input/output port labelled 13.

3. Mapping Programs on Linear-Arrays

In this section we formally define the problem of mapping programs on linear arrays. We first provide an intuitive view of mapping.

Consider a program[1] whose domain D and range R are subsets of Y_1 X Y_2 X .. X Y_k. Let G be the corresponding program graph of Ψ which is obtained by transforming Ψ into G using the set TR={TR_1, TR_2, TR_3, TR_4} of transformation functions.. Let L_G={11, 12, .., 1k}. In particular consider a path in G as shown in figure 3.1-1 and whose path-label is

---

[1] we use the same notations as in section-2

lj.

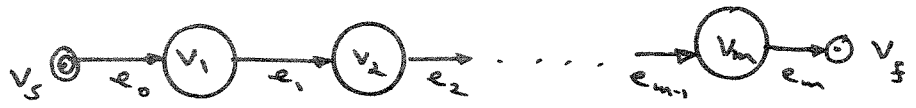


Figure 3.1-1: Path labelled lj

In figure 3.1-1, v_1, v_2, .., v_m are computation vertices. v_s and v_f are source and sink vertices respectively and are labelled lj. $e_\emptyset$, e_1, .., e_m are edges whose labels are lj.

Let $z_\emptyset$, z_1, .., $z_{(m-1)}$ and w_1, w_2, .., w_m be tuples in Y such that for every i , $1 \leq i \leq m$, $(z_{(i-1)})=w_i$ and $TR_1(z_{(i-1)})=v_i$. Let $x_{(i-1)}$ denote the j^{th} component of $z_{(i-1)}$ and hence $TR_2(x_{(i-1)})=e_{(i-1)}$. Also let $TR_3(x_\emptyset)=v_s$ and $TR_4(x_m)=v_f$.

Let $z_i(-j)$ denote z_i without its j^{th} component, .i.e., if $z_i=\langle u_1, u_2, .., u_{j-1}, x_i, u_{j+1}, .., u_k\rangle$ then $z_i(-j)=\langle u_1, u_2, .., u_{j-1}, u_{j+1}, .., u_k\rangle$.

Now $x_{i+1} = \psi_j(z_i(-j), x_i)$

$$= \psi_j(z_i(-j), \psi_j(z_{i-1}, \psi_j(.., \psi_j(z_\emptyset(-j), x_\emptyset)) ..).$$

So x_{i+1} is the result of an i-fold composition of ψ_j on the sequence of arguments $z_\emptyset$, z_1, .., z_i. In a sequential processing of the program ψ a register is associated with every labelled path. In particular for

the path in figure 3.1-1 the location is initialized with the element represented by the source vertex v_s which is $x_\emptyset$ and at the end of the computation the location contains the element represented by the sink vertex v_f which is x_m. If any veretex v_i is computed then the element represented by edge e_i which is x_i is stored in the location and is held there till the computation of v_{i+1} for which the tuple z_i is the input and x_i is the j^{th} component of z_i.

A mapping of Ψ on a linear-array of processors is an execution of every computation vertex in G by some processor in the array. However this execution is performed under some constraints.

Let LA be a linear-array of processors with L_P={l1, l2, .., lk}. Let Ψ and G be defined as in the beginning of this section. Let T={$\emptyset$,c,2c,.....} be a sequence of time steps where c is a basic processor cycle time. Let SO_G' and SI_G' be a subset of $(SO_G)x(T)$ and $(SI_G)x(T)$ respectively.

Definition 3.0-1: A mapping of G onto the processors in LA is a 4-tuple MP_G=<TA, PA, IOA> where TA, PA and IOA are many-one functions such that:

1. $TA:V_G \longrightarrow T$

2. $PA:V_G \longrightarrow N$

3. $IOA:(SO_G' \cup SI_G') \longrightarrow N$

If v_x is a source vertex such that IOA<v_x, t>=s then it means that v_x

is mapped onto the input-port of s at time t and if v_x is a sink vertex such that IOA<v_x, t>=s then it means that v_y is mapped onto the output-port of s at time t. SO_G', SI_G', and IOA are constructed as follows:

1. Initially SO_G' and SI_G' are empty.

2. For all lj if v_x is in SO_G and there exists an edge labelled lj directed from v_x to v_y then $SO_G'=SO_G' \cup \{<v_x,t>\}$ where $t=TA(v_y)-m*d_{lj}$ for all m such that $PA(v_y)-m*n_{lj}$ is a processor index. Define $IOA<v_x,t)=PA(v_x)-m*n_{lj}$

3. For all lj if v_x is in SI_G and there exists an edge labelled lj directed from v_x to v_y then $SI_G'=SI_G' \cup \{<v_x,t+d_{lj}>\}$ where $t=TA(v_y)+m*n_{lj}$ for all m such that $PA(v_y)+m*n_{lj}$ is a processor index. Define $IOA<v_x,t+d_{lj}>=PA(v_x)+m*n_{lj}$

Let t_s and t_f denote the start time and finish time respectively of a computation graph G that has been mapped onto LA.

t_s is less or equal to the minimum over all t in all tuples $<v_x,t>$ in SO_G' and for all labels lj such that $G_{IO}(v_x)=lj$ and $n_{lj}\neq\emptyset$.

t_f is greater or equal to the maximum over all t in all tuples $<v_x,t>$ in SI_G' and for all labels lj such that $G_{IO}(v_x)=lj$ and $n_{lj}\neq\emptyset$.

Definition 3.0-2: MP_G is syntactically correct iff:

1. $\Psi = \Psi_P$.

2. the three functions satisfy the following:

   a. For every label lj if v_x and v_y are computation vertices and there exists an edge labelled lj directed from v_x to v_y then $PA(v_y)=PA(v_x)+n_{lj}$ and $TA(v_y)=TA(v_x)+d_{lj}$.

   b. If v_x and v_y are computation vertices and if $PA(v_x)=PA(v_y)$ and $TA(v_x)=TA(v_y)$ then $v_x=v_y$.

   c. If v_x and v_y are either source or sink vertices and if $IOA\langle v_x, t\rangle=IOA\langle v_y, t\rangle$ and $G_{IO}(v_x)=G_{IO}(v_y)$ then $v_x=v_y$.

3. for all lj if $n_{lj}=\emptyset$ and $PA(v_x)=PA(v_y)$ then there must be a major path labelled lj that passes through these two vertices (i.e., a register is associated with the computation of the vertices in the path labelled lj and this register serves as the input/output port labelled lj for the processor) .

For any tuple $\langle v_x,t_m\rangle$ either in SI'_G or in SO'_G, let $t_m(v_x)$ denote the contents of v_x at time t_m.

Definition 3.0-3: MP_G correctly computes Ψ iff:

1. MP_G is syntactically correct

2. For any pair of tuples $\langle v_x,t_m\rangle$ and $\langle v_x,t_n\rangle$ either in SI'_G or in SO'_G, $t_m(v_x)=t_n(v_x)$

Intutively if there exists more than one tuple $\langle v_x, t\rangle$ either in SO'_G or in SI'_G it means that v_x is mapped onto the input/output ports of two distinct processors more than once. Hence the contents of the source and sink vertices that is mapped more than once must be invariant after every such mapping.

4. Preliminary Results

Let G be the program graph for some program . Let the label set L_G={l1, l2, .., lk}. Let LA be the linear array of processors whose label set L_P is the same as L_G and the function Ψ_P computed by any processor in LA is the same as Ψ.

For any label lj in L_G let E_{lj}={major paths labelled lj}.

Definition 4.0-1: For any label li and lj, E_{li} is identical to E_{lj} iff for every major path in E_{li} there is an identical path in E_{lj} and vice-versa.

Let H={E_{lj}| r_{lj} is not empty[1] and for any E_{lj} and E_{li} in H, E_{lj} is not identical to E_{li}}.

Let F={E_{lj}| r_{lj} is not empty and for every E_{lj} in F there exists some E_{li} in H such that E_{lj} and E_{li} are identical}.

Let D={E_{lj}| r_{lj} is empty}.

---

[1] definition 2.1-6, page 7

Let q be a undirected[2] path from a computation vertex v_x to another computation vertex v_y such that all the vertices in the path are computation vertices and all the edges in the path have the same label. Let lj be the label of the edges in q. A directed edge imposes an order on the pair of vertices it connects. If there is a directed edge from v_x to v_y then the order of v_x is less than the order of v_y. A path imposes an order on the vertices it vists. If the path vists v_x before v_y then the order of v_x is less than the order of v_y. In a directed path the order of vertices imposed by the path is consistent with the order imposed by the edges. In a undirected path there could be edges in the path such that the order imposed by these edges on the vertices is not consistent with the order imposed by the path. Let k_1 and k_2 be the number of edges in a undirected path that impose an order that is consistent and not consistent respectively with the order imposed by the undirected path. If v_x and v_y are two computation vertices in this path then the processors computing v_x and v_y along with the time at which they are computed can be easily related.

Proposition 4.0-1: In any syntactically correct mapping $PA(v_y)=PA(v_x)+(k_1-k_2)*n_{lj}$ and $TA(v_y)=TA(v_x)+(k_1-k_2)*d_{lj}$.

Let q be a major path labelled lj. Let v_x and v_y be two computation vertices in q such that the distance from v_x to v_y is k.

Corollary 4.0-1: In any syntactically correct mapping,

---

[2]in a undirected path the direction of edges in the path are ignored

$PA(v_y)=PA(v_x)+k*n_{1j}$ and $TA(v_y)=TA(v_x)+k*d_{1j}$.

Proposition 4.0-2: If a major path has k computation vertices then the indices of these computation vertices in the ordering imposed by a topological sort on the vertices in the major path range from 1 to k.

Proof: A topological sort on a directed path from v_x to v_y containing m vertices in the path (v_x and v_y included) imposes an ordering on these vertices and assigns indices to the vertices in the ordering ranging from $\emptyset$ to m-1. Index $\emptyset$ is assigned to v_x and index m-1 is assigned to v_y and indices ranging from 1 to m-2 are assigned to all the m-2 intermediate vertices in the path (v_x and v_y included).

In any major path the source vertex is v_x, the sink vertex is v_y and the intermediate vertices are computation vertices.

Proposition 4.0-3: G must be acyclic for any syntactically correct mapping.

Proof: Suppose G has a cycle. Then there must be a computation vertex v_x such that there is a directed path from v_x to itself and hence in any syntactically correct mapping $TA(v_x)>TA(v_y)$.

Proposition 4.0-4:

If there are two major paths in G such that the computation vertices in these two major paths are same then these two major paths must be identical for any syntactically correct mapping.

Proof: Let q_r and q_s be the two major paths such that the computation vertices in these two major paths are the same. Let k be the number of computation vertices in q_r and q_s. The range of the indices of the computation vertices (.i.e., 1 to k) in the ordering imposed by a topological sort on q_r is the same as the range of the indices of the computation vertices in the ordering imposed by a topological sort on q_s.

Suppose q_r is not identical to q_s. It can be easily shown that there must exist two computation vertices v_x and v_y such that q_r vists v_x before v_y and q_s visits v_y before v_x and hence in any syntactically correct mapping $TA(v_x) > TA(v_y)$ in q_s and $TA(v_y) > TA(v_x)$ in q_r.

Proposition 4.0-5:

For any lj in L_G if r_{lj} is not empty then in any syntactically correct mapping n_{lj} must be one of {1, -1, ∅}.

Lemma 4.0-1:

For any li and lj in L_G if E_{li} and E_{lj} are in H then in any syntactically correct mapping $n_{li}=n_{lj}\neq\emptyset$.

Proof: If E_{li} and E_{lj} are in E then E_{li} and E_{lj} are not identical. Consequently there is a major path q_r in E_{li} that is not identical to any major path in E_{lj} and this is because of one of the following:

1. there is a major path q_s in E_{lj} such that the computation

vertices in q_s and in q_r are the same but there exists a computation vertex v_x such that its index in the ordering imposed by a topological sort on q_r is different from its index in the ordering imposed by a topological sort on q_s. But by proposition 4.0-4 q_r and q_s must be identical.

2. there does not exist any major path q_s in E_{1j} such that the computation vertices in q_r and q_s are the same and hence,

   a. there exists more than one major path in E_{1j} and each of these major paths pass through a subset of the computation vertices in q_r.

   b. there exists a major path q_s in E_{1j} and a subset of the computation vertices in q_s is the same as the computation vertices in q_r.

First let us consider (2a). Without loss of generality let there be two major paths q_s and q_t in E_{1j} that pass through computation vertices in q_r as shown in figure 4.Ø-1.

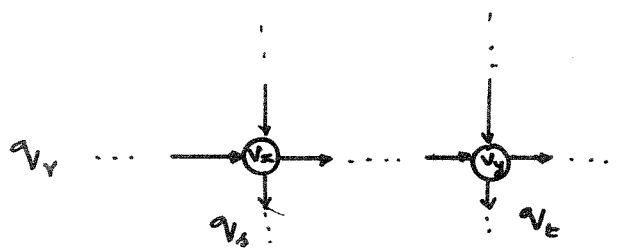


Figure 4.Ø-1

If $n_{1i}=n_{1j}=\emptyset$ then $PA(v_x)=PA(v_y)$. By definition of a syntactically correct mapping a single major path labelled 1j must pass through v_x and v_y. But in figure 4.Ø-1 v_x and v_y are on distinct major paths q_s and q_t respectively.

Next consider (2b). Let q_s be the major path in E_{1j} that passes through all the computation vertices in q_r as shown in figure 4.Ø-2.

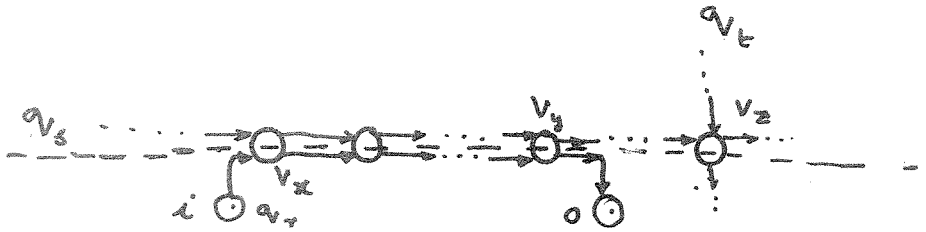


Figure 4.Ø-2

The major path q_r is directed from v_x to v_y through edges shown below the dashed line. q_s is the major path that visits v_x, v_y and v_z in that order. All the vertices in the path of q_s from v_x to v_y is the same as the vertices in q_r. The edges in the path of q_s from v_x to v_y are through edges shown above the dashed line. Now v_z is a computation vertex in the path of q_s which is also in the path of another major path q_t that is labelled 1i. Now q_t is distinct from q_r. So if $n_{1i}=n_{1j}=\emptyset$ then $PA(v_x)=PA(v_y)=PA(v_z)$. But in a syntactically correct mapping there must be a single major path labelled 1i which passes through v_y and v_z. But in figure 4.Ø-2 v_y and v_z are in distinct major paths q_r and q_t respectively.

Proposition 4.0-6:

In any syntactically correct mapping if v_u and v_w are the source and sink vertices of a major path then $IOA\langle v_u, t\rangle = IOA\langle v_w, t\rangle$.

Proof: There must be at least one computation vertex in between the source and sink vertex and hence the result.

Let $T_{\langle a,b\rangle}$ be a binary relation on computation vertices that have been mapped onto processors by some mapping MP_G.

Definition 4.0-2: $v_x \; T_{\langle a,b\rangle} \; v_y$ iff

1. $PA(v_y)=PA(v_x)+a$ and $PA(v_y)$ is a processor index.

2. $TA(v_y)=TA(v_x)+b$.

So $T^k_{\langle a,b\rangle}$ is the k-fold composition of $T_{\langle a,b\rangle}$ and hence if $v_x \; T^k_{\langle a,b\rangle} \; v_y$ then $PA(v_y)=PA(v_x)+k*a$ and $TA(v_y)=TA(v_x)+k*b$.

Now consider a mapping M that satisfies all the properties of a syntactically correct mapping except (2c) . The conditions under which M satisfies (2c) is stated in the following two lemmas.

Lemma 4.0-2:

For any label lj such that $n_{lj}=\emptyset$ if $G_{IO}(v_u)=G_{IO}(v_w)=lj$ for any v_u, v_w in $SI_G \cup SO_G$ and $IOA\langle v_u,t\rangle = IOA\langle v_w,t\rangle$ in M then $v_u=v_w$.

Proof: Suppose the hypothesis is true but $v_u \neq v_w$. Then by proposition 4.0-6 v_u and v_w must be in distinct major paths

labelled lj. This means there exists a computation vertex v_x in the major path in which v_u is a source vertex and another computation vertex v_y in the major path in which v_w is a source vertex respectively. $n_{lj}=\emptyset$ and hence $PA(v_x)=PA(v_y)$. But this violates property (3) of a syntactically correct mapping and we have assumed that M satisfies this property.

Lemma 4.0-3:

For any label lj such that $n_{lj}\neq\emptyset$ if $G_{IO}(v_u)=G_{IO}(v_w)=lj$ for any v_u and v_w in $SI_G \cup SO_G$ and $IOA\langle v_u,t\rangle=IOA\langle v_w,t\rangle$ in M then $v_u=v_w$ iff for any pair of computation vertices v_x and v_y if $v_x\ T^k_{\langle a,b\rangle}\ v_y$ where $a=n_{lj}$ and $b=d_{lj}$ then there must be a major path labelled lj passing from v_x to v_y in that order and the distance from v_x to v_y in this path must be k.

Proof:

Necessity:

Suppose there exist computation vertices v_x and v_y and a label lj such that $v_x\ T^k_{\langle a,b\rangle}\ v_y$ in M and there does not exist a single major path labelled lj passing through v_x and v_y. Let q_r and q_s be two distinct major paths labelled lj such that v_x is in q_r and v_y is in q_s respectively. Let v_u and v_w be the source vertices for q_r and q_s respectively. Let k_1 and k_2 be the distances from v_u to v_x and from v_w to v_y respectively. Let $PA(v_x)=s$ and $TA(v_x)=t$. Consequently v_u is mapped onto the input port of processor $s-k_1*n_{lj}$ at $t-k_1*d_{lj}$. Similarly v_w is mapped onto the input port of $s+(k-k_2)*n_{lj}$ at $t+(k-k_2)*d_{lj}$. Without

loss of generality let $(s-k_1*n_{1j})<(s+(k-k_2)*n_{1j})$. Since n_{1j} is one of {1,-1} there must exist a k_3 such that $s-k_1*n_{1j}=s+(k-k_2-k_3)*n_{1j}$ and so $k_1=-(k-k_2-k_3)$. Now by our construction of SI_G' there must exist a tuple $\langle v_w, t+(k-k_2-k_3)*d_{1j}\rangle$ in SI_G' such that $IOA\langle v_w, t+(k-k_2-k_3)*d_{1j}\rangle = s+(k-k_2-k_3)*n_{1j}=s-k_1*n_{1j}$ and hence $IOA\langle v_u, t-k_1*d_{1j}\rangle=IOA\langle v_w, t-k_1*d_{1j}\rangle$ and $v_u=v_w$.

Sufficiency

Suppose there exist vertices v_u and v_w in SI_G SO_G and a label 1j such that $IOA\langle v_u, t\rangle=IOA\langle v_w, t\rangle=s$ in M, $G_{IO}(v_u)=G_{IO}(v_w)=1j$ and $v_u\neq v_w$. By proposition 4.0-6 v_u and v_w must be in two distinct major paths labelled 1j. v_u and v_w in combination must be one of the following:

1. both are source vertices

2. both are sink vertices

3. one is a source vertex and the other is a sink vertex

Consider the first case where v_u and v_w are both source vertices. By lemma 4.0-2 $n_{1j}\neq\emptyset$. Let q_r and q_s be the two major paths labelled 1j such that v_u is the source in q_r and v_w is the source in q_s. Let v_x be the first[1] computation vertex in q_r and

[1]the first computation vertex in a major path is the vertex adjacent to the source, .i.e., the edge from source vertex is directed into this vertex.

v_y be the first computation vertex in q_s.

Let $PA(v_x)=s_1$, $TA(v_x)=t_1$, $PA(v_y)=s_2$ and $TA(v_y)=t_2$. By construction of SO'_G there must exist tuples $\langle v_u,t_1\rangle$, $\langle v_u,t_1-k_1*d_{1j}\rangle$, $\langle v_w,t_2\rangle$ and $\langle v_w,t_2-k_2*d_{1j}\rangle$ such that

$t=t_1-k_1*d_{1j}=t_2-k_2*d_{1j}$
$IOA\langle v_u,t_1\rangle=s_1$ and $IOA\langle v_w,t_2\rangle=s_2$
$IOA\langle v_u,t\rangle=IOA\langle v_w,t\rangle=s=s_1-k_1*n_{1j}=s_2-k_2*n_{1j}$
and hence, $s_1=s+k_1*n_{1j}$
$s_2=s+k_2*n_{1j}$
$t_1=t+k_1*d_{1j}$
$t_2=t+k_2*d_{1j}$
and hence, $s_2=s_1+(k_2-k_1)*n_{1j}$
$t_2=t_1+(k_2-k_1)*d_{1j}$

and so $v_x\ T^k_{\langle a,b\rangle}\ v_y$ where $a=n_{1j}$, $b=d_{1j}$ and $k=k_2-k_1$. But v_x and v_y are in distinct paths q_r and q_s respectively and we have assumed that they both are in the same major path labelled 1j. We can obtain a similar contradiction in the other two cases also.

Lemma 4.0-4:

In any syntactically correct mapping of G and for any E_{1i} and E_{1j} in H if $n_{1i}=n_{1j}\neq\emptyset$ then $d_{1i}\neq d_{1j}$.

Proof: Since E_{1i} and E_{1j} are in H they cannot be identical giving rise to the two cases similar to the two cases considered in the proof of lemma 4.0-1. For reasons similar to the one given in proof of lemma 4.0-1 case (1) cannot arise in any syntactically correct mapping.

Consider case (2a). Let the distance from v_x to v_y in figure

4.0-1 be k. Now $PA(v_y)=PA(v_x)+k*n_{1i}$ and $TA(v_y)=TA(v_x)+k*d_{1i}$. Suppose $d_{1i}=d_{1j}$ and hence $PA(v_y)=PA(v_x)+k*n_{1j}$ and $TA(v_y)=TA(v_x)+k*d_{1j}$ and hence $v_x\ T^k_{\langle a,b\rangle}\ v_y$ where $a=n_{1j}$ and $b=d_{1j}$ and so by lemma 4.0-3 there must be a single major path labelled 1j passing through v_x and v_y. But in figure 4.0-1 v_x and v_y are in q_s and q_t respectively and q_s and q_t are distinct major paths labelled 1j .

Consider case (2b). We can similarly show that $v_y\ T^k_{\langle a,b\rangle}\ v_z$ where $a=n_{1i}$ and $b=d_{1i}$. But v_y and v_z are in distinct major paths q_r and q_t labelled 1i.

Definition 4.0-3: A maximally-connected subgraph of a uniform graph G is a 5-tuple $SG=\langle V_{SG}, SO_{SG}, SI_{SG}, E_{SG}, L_{SG}\rangle$ where:

1. V_{SG}, SO_{SG}, SI_{SG}, E_{SG} and L_{SG} are subsets of V_G, SO_G, SI_G, E_G and L_G respectively.

2. and SG satisfies the following:

   a. the edges in E_{SG} are labelled with labels from L_{SG}

   b. SG is connected[1]

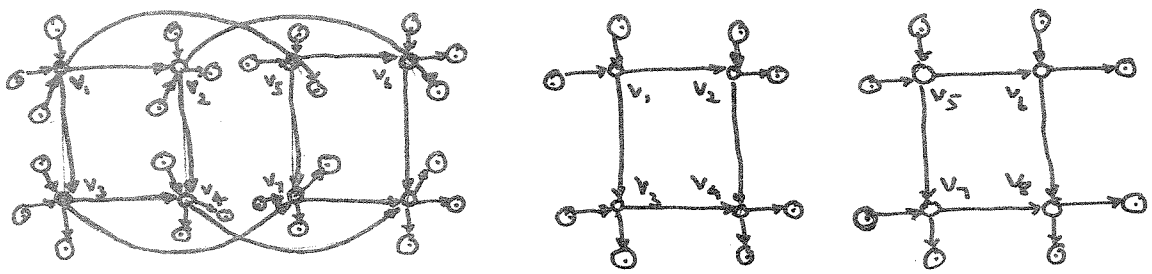
   c. there exist no pair of vertices v_x and v_y such that v_x is not in SG and v_y is in SG and there is a undirected

---

[1] there is an undirected path between every pair of computation vertices in SG such that the edges in the path are all from E_{SG}.

path from v_x to v_y through edges whose labels are in L_{SG}

Example 4.0-1:



G SG_1 SG_2

Figure 4.Ø-3

G is a uniform graph. Computation vertices are denoted by 'o', source and sink vertices are denoted by '⊙'. The horizontal, vertical and oblique (curved edges in G) are labelled l1, l2 and l3 respectively. The source and sink vertices attached to horizontal, vertical and oblique edges are labelled l1, l2 and l3 respectively. SG1 and SG2 are the two maximally-connceted subgraphs with $L_{SG1}=L_{SG2}=\{l1, l2\}$.

Let li and lj be any two labels in L_G such that E_{li} and E_{lj} are in H. Let SG be a maximally-connected subgraph with $L_{SG}=\{li, lj\}$. Let $SG_{li}=\{$major paths in SG labelled li$\}$ and $SG_{lj}=\{$major paths in SG labelled lj$\}$. Now SG_{li} and SG_{lj} are subsets of E_{li} and E_{lj} respectively.

Let $S_{li}=\{r_l^{\,1} \mid l$ is a label and r_l is binary relation on major paths in

---

[1] recall definition 2.1-6, page 7

$SG_{1i}\}$. Since $L_{SG}=\{1i, 1j\}$ and hence $S_{1i}=\{r_{1j}\}$. Similarly $S_{1j}=\{r_{1i}\}$. For any mapping of G to be syntactically correct r_{1i} and r_{1j} must impose a structure on the major paths in SG_{1i} and SG_{1j} which is stated formally in the following lemma.

Lemma 4.0-5:

For any mapping of G to be syntactically correct r_{1j} in S_{1i} and r_{1i} in S_{1j} must impose a linear chain on the major paths in SG_{1i} and SG_{1j} respectively.

Proof: We will first show that r_{1j} in S_{1i} must impose a linear chain on major paths in SG_{1i}. Inorder to show this we must first show that S_{1i} imposes a consistent order on major paths in SG_{1i}.

Let G_s and G_x be the directed graph induced by all the relations in S_{1i} and by the relation r_{1j} respectively on the major paths in SG_{1i}. $S_{1i}=\{r_{1j}\}$ and hence $G_s=G_x$. Consequently inorder to show that S_{1i} imposes a consistent order on the major paths in SG_{1i} we must show that r_{1j} is total on SG_{1i} and G_x is acyclic.

Suppose r_{1j} is not total. This means there exist atleast two major paths q_s and q_t in SG_{1i} such that neither $q_s\ r_{1j}^{+} q_t$ nor $q_t\ r_{1j}^{+} q_s$. Now SG is connected and hence G_x is connected and so there must be a major path q_k in SG_{1i} such that one of the following two cases hold:

1. $q_s\ r_{1j}^{+} q_k$, $q_t\ r_{1j}^{+} q_k$ and there does not exist any major path q_w in SG_{1i} such that $q_s\ r_{1j}^{+} q_w$, $q_t\ r_{1j}^{+} q_w$ and $q_w\ r_{1j}^{+} q_k$

2. $q_k\ r_{1j}^{+} q_s$, $q_k\ r_{1j}^{+} q_t$ and there does not exist any major path q_w in SG_{1i} such that $q_k\ r_{1j}^{+} q_w$, $q_w\ r_{1j}^{+} q_s$ and $q_w\ r_{1j}^{+} q_t$

Consider the first case. Let q_m and q_n be two major paths in SG_{1i} such that $q_s\ r_{1j}^{+} q_m$, $q_t\ r_{1j}^{+} q_n$, $q_m\ r_{1j}\ q_k$ and $q_n\ r_{1j}\ q_k$. Since q_w in (1) and (2) is non-existent and hence q_m and q_n are two distinct major paths. $q_m\ r_{1j}\ q_k$ and so there must exist a pair of computation vertex v_x and v_w in q_m and q_k respectively such that there is an edge e_a labelled 1j directed from v_x to v_w. Similarly since $q_n\ r_{1j}\ q_k$ and hence there must exist a pair of computation vertices v_y and v_u in q_n and q_k respectively such that there is an edge e_b labelled 1j directed from v_y to v_u. This is illustrated in figure 4.Ø-4 below:

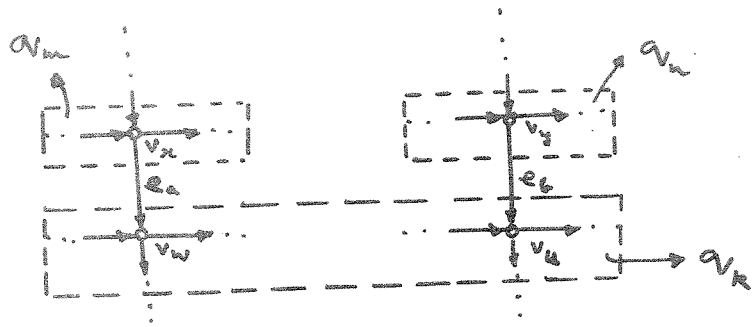


Figure 4.Ø-4

In Figure 4.Ø-4 each of the shaded boxes denote a major path labelled 1i. Let the distance from v_w to v_u in q_k be h and hence in any syntactically correct mapping,

$PA(v_u)=PA(v_w)+h*n_{1i}$

and $TA(v_u)=TA(v_w)+h*d_{1i}$

Now $v_x\ r_{1j}\ v_w$ and hence,

$PA(v_w)=PA(v_x)+n_{1j}$

and $TA(v_w)=TA(v_x)+d_{1j}$

Also $v_y\ r_{1j}\ v_u$ and hence,

$PA(v_u)=PA(v_y)+n_{1j}$

$TA(v_u)=TA(v_y)+d_{1j}$

From the above equations we obtain,

$PA(v_y)=PA(v_x)+h*n_{1i}$

$TA(v_y)=TA(v_x)+h*d_{1i}$

Hence $v_x\ T^k_{<a,b>}\ v_y$ where $a=n_{1i}$ and $b=d_{1i}$. If $n_{1j}=\emptyset$ then $PA(v_x)=PA(v_y)$ and by definition of a syntactically correct mapping v_x and v_y must both be in the same major path labelled 1i. If $n_{1j}\neq\emptyset$ then by lemma 4.0-3 v_x and v_y must both be in the same major path labelled 1i. But v_x and v_y are in distinct major paths q_m and q_n. We can arrive at a similar contradiction in the second case also.

We must next show G_x is acyclic. Suppose there is a cycle in G_x. Let q_1, q_2, .., q_m be the set of the m major paths in SG_{1i} that form a cycle in G_x as shown in figure 4.0-5.

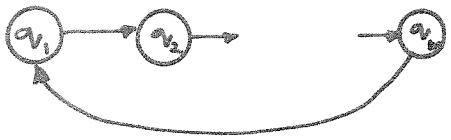


Figure 4.Ø-5: a cycle in G_x

This cycle implies that there are two computation vertices v_x and v_y in q_1 such that there is a undirected path from v_x to v_y that passes through computation vertices in each of q_2, q_3, ..., q_m and through edges labelled li or lj as shown in figure 4.Ø-6.

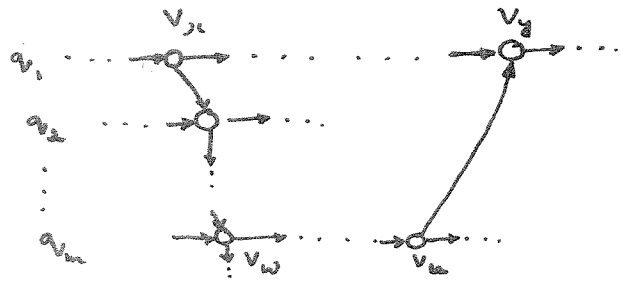


Figure 4.Ø-6: a cycle in G

Let k_1 and k_2 be the number of edges labelled li in this path such that the order on the vertices imposed by these edges are consistent and not consistent respectively with the order on the vertices imposed by the undirected path. Let $k_1-k_2=h$. The number of edges in this path that are labelled lj is m-1 and clearly (m-1)>1. By proposition 4.0-1,

$$PA(v_y)=PA(v_x)+h*n_{li}+(m-1)*n_{lj}$$

and $TA(v_y)=TA(v_x)+h*d_{1i}+(m-1)*d_{1j}$

Let the distance from v_x to v_y in the major path q_1 be k and hence,

$$PA(v_y)=PA(v_x)+k*n_{1i}$$

and $$TA(v_y)=TA(v_x)+k*d_{1i}$$

and so,

$$k*n_{1i}=h*n_{1i}+(m-1)*n_{1j} \quad \ldots\ldots\ldots\ldots\ldots (a)$$

$$k*d_{1i}=h*d_{1i}+(m-1)*d_{1j} \quad \ldots\ldots\ldots\ldots\ldots (b)$$

Now E_{1i} and E_{1j} are in H and hence by proposition 4.0-5 each of n_{1i} and n_{1j} must be one of {1, -1, $\emptyset$}. By lemma 4.0-1 $n_{1i}=n_{1j}\neq\emptyset$ and so the possible values that the tuple $\langle n_{1i}, n_{1j}\rangle$ can assume are: $\langle 1,\emptyset\rangle$, $\langle 1,1\rangle$, $\langle 1,-1\rangle$, $\langle -1,\emptyset\rangle$, $\langle -1,1\rangle$, $\langle -1,-1\rangle$, $\langle \emptyset,1\rangle$ and $\langle \emptyset,-1\rangle$.

1. Consider the set of values $\langle 1,1\rangle$ and $\langle -1,-1\rangle$. From (a) and (b) $d_{1i}=d_{1j}$. But by lemma 4.0-4 $d_{1i}=d_{1j}$.

2. Consider the set of values $\langle \emptyset,1\rangle$ and $\langle \emptyset,-1\rangle$. From (a) $n_{1j}=\emptyset$.

3. Consider the set of values $\langle 1,\emptyset\rangle$ and $\langle -1,\emptyset\rangle$. From (a) and (b) $d_{1j}=\emptyset$.

4. Consider the set of values $\langle -1,1\rangle$ and $\langle 1,-1\rangle$. From (a) and (b) $d_{1i}=-d_{1j}$.

and hence G_x must be acyclic.

Lastly we must show that the consistency constant of r_{1j} in S_{1i} is 1. Suppose otherwise. This means there exist major paths q_s and q_t in SG_{1i} such that $q_s\ r_{1j}^m q_t$ and $q_s\ r_{1j}\ q_t$ in G_x and $m>1$ as shown in figure 4.Ø-7.

Figure 4.Ø-7

Now $q_s\ r_{1j}\ q_t$ and hence there must exist computation vertices v_x and v_y in q_s and q_t respectively such that there is an edge e_b labelled 1j directed from v_x to v_y. Also $q_s\ r_{1j}^m q_t$ and so there must exist major paths q_{s1}, q_{s2}, ••, $q_{s(m-1)}$ in SG_{1i} such that $q_s\ r_{1j}\ q_{s1}$, $q_{s1}\ r_{1j}\ q_{s2}$, ••, $q_{s(m-1)}\ r_{1j}\ q_t$ and hence there must exist an undirected path from v_x to v_y through computation vertices in each of q_{s1}, q_{s2}, ••, $q_{s(m-1)}$ and through edges labelled 1i or 1j as shown in figure 4.Ø-8.

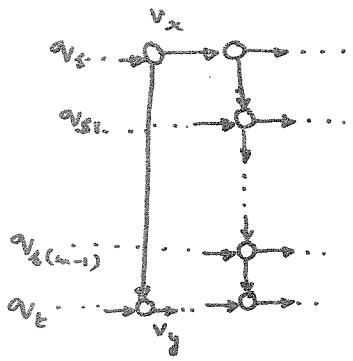


Figure 4.Ø-8

Let k_1 and k_2 be the number of edges labelled 1i in the

undirected path from v_x to v_y such that these edges impose an order on the vertices in the path that are consistent and not consistent respectively with the order imposed by the path on these vertices. The number of edges labelled lj in this path is m and hence,

$$PA(v_y)=PA(v_x)+m*n_{1j}+h*n_{1i}$$

and $$TA(v_y)=TA(v_x)+m*d_{1j}+h*d_{1i}$$

also $q_s\ r_{1j}\ q_t$ and hence

$$PA(v_y)=PA(v_x)+n_{1j}$$

$$TA(v_y)=TA(v_x)+d_{1j}$$

and so, $h*n_{1i}=(m-1)*n_{1j}$

and $h*d_{1i}=(m-1)*d_{1j}$

By reasons similar to that used in showing G_x is acyclic we conclude that the consistency constant of the relation r_{1j} in S_{1i} must be 1. Hence r_{1j} in S_{1i} must impose a linear chain on the major paths in SG_{1i}. Similarly we can prove that r_{1i} in S_{1j} must impose a linear chain on the major paths in SG_{1j}.

5. Syntactic Characterization

In this section we provide a characterization theorem for a class of program graphs such that there exists a syntactically correct mapping for every member in this class. As a prelude to the main result (theorem 5.2-1) we establish some preliminary results.

5.1. Mesh-graphs

Let MG be a uniform graph such that $L_{MG}=\{1i,1j\}$ and r_{1i} and r_{1j} are both non-empty.

Definition 5.1-1: MG is a mesh graph iff there is a one-one function $F:V_{MG} \longrightarrow I \times I$ where:

1. I is a set of integers

2. V_{MG} is the set of computation vertices in MG

3. F_{1i} and F_{1j} are projection functions of F, .i.e., if $F(v_x)=\langle m,n\rangle$ then $F_{1i}(v_x)=m$ and $F_{1j}(v_y)=n$

4. for any v_x and v_y in V_{MG}

   a. if there exists a major path labelled 1i passing through v_x and v_y such that the distance from v_x to v_y in this major path is k then $F_{1j}(v_y)=F_{1i}(v_x)+k$ and $F_{1j}(v_y)=F_{1j}(v_x)$ and conversely so[1]

   b. if there exists a major path labelled 1j passing through v_x and v_y such that the distance from v_x to v_y in this major path is k then $F_{1j}(v_y)=F_{1j}(v_x)+k$ and $F_{1i}(v_y)=F_{1i}(v_x)$ and conversely so.

---

[1].i.e., if $F_{1i}(v_y)=F_{1i}(v_x)+k$ and $F_{1j}(v_y)=F_{1j}(v_x)$ then there must exist a major path labelled 1i passing through v_x and v_y and the distance from v_x to v_y in this path must be k.

We will denote $F(v_x)$ by the tuple $\langle x_{1i}, x_{1j} \rangle$ where $F_{1i}(v_x)=x_{1i}$ and $F_{1j}(v_x)=x_{1j}$. We can think of x_{1i} and x_{1j} as the horizontal and vertical co-ordinates of the computation vertex v_x.

Example 5.1-1:

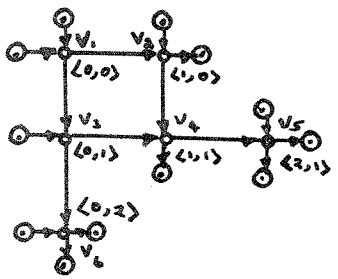


Figure 5.1-1: Mesh Graph MG

In figure 5.1-1 'o' denote computation vertices and '0' denote source and sink vertices.

$F(v_1)=\langle \emptyset, \emptyset \rangle$, $F(v_2)=\langle 1, \emptyset \rangle$, $F(v_3)=\langle \emptyset, 1 \rangle$, $F(v_4)=\langle 1,1 \rangle$, $F(v_5)=\langle 2,1 \rangle$ and $F(v_6)=\langle \emptyset, 2 \rangle$.

We can characterize a mesh graph in terms of the relations r_{1i} and r_{1j} on the major paths in E_{1j} and E_{1i} respectively. Consider a uniform graph MG with $L_{MG}=\{1i,1j\}$ and r_{1i} and r_{1j} are non-empty.

Proposition 5.1-1:

MG is a mesh graph iff each of the following two conditions hold:

1. r_{1j} imposes a linear chain on the major paths labelled 1i in MG

2. r_{1i} imposes a linear chain on the major paths labelled 1j in MG

If MG satisfies proposition 5.1-1 then the major paths in E_{1i} are totally ordered and similarly the major paths in E_{1j} are also totally ordered. So if v_x is a computation vertex in MG then $F(v_x)=\langle m,n\rangle$ where m is the index of the major path in E_{1j} which passes through v_x and n is the index of the major path in E_{1i} that passes through v_x.

Using proposition 5.1-1 we can characterize the property of any maximally-connected subgraph of a uniform graph G inorder that any mapping of G is syntactically correct.

Theorem 5.1-1: For any pair of labels 1i and 1j in L_G of a uniform graph G if SG is any maximally-connected subgraph of G with $L_{SG}=\{1i,1j\}$ then SG must be a mesh graph in order for any mapping of G to be syntactically correct.

Proof: Immediate from lemma 4.0-5 and proposition 5.1-1.

In any syntactically correct mapping of a mesh graph we can easily relate the processors computing any pair of computation vertices and also the times at which they are computed.

Proposition 5.1-2:

In any syntactically correct mapping of the mesh graph MG and for any pair of computation vertices v_x and v_y,

$$PA(v_y)=PA(v_x)+(y_{1i}-x_{1i})*n_{1i}+(y_{1j}-x_{1j})*n_{1j}$$