TERMINATION AND STORAGE REQUIREMENTS IN A MODEL FOR PARALLEL COMPUTATIONS

by

IVAN SCHNAPP
Dipl.Ing., Czech Technical University, Prague, 1966

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in the Department of
Electrical Engineering

We accept this thesis as conforming to the
required standard

Research Supervisor ....

Members of the Committee ....

....

Acting Head of the Department ....

Members of the Department
of Electrical Engineering

THE UNIVERSITY OF BRITISH COLUMBIA
June, 1970

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical Engineering

The University of British Columbia
Vancouver 8, Canada

Date July 27, 1970

ABSTRACT

This paper deals with a graph-theoretic model for parallel computations as formulated by Karp and Miller. A necessary condition and a sufficient condition for self-termination of loops with unit loop gain are presented. For the special case that W=U the necessary and sufficient condition is derived. A direct procedure for testing termination properties of strongly connected graphs is presented.

A method due to Reiter, for determining the maximum storage required for a computation graph, is extended to cover the general case.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

ACKNOWLEDGMENT

I wish to express my gratitude to Dr. E. L. Sigurdson for his guidance and encouragement in carrying out this research, and to Dr. G. F. Schrack for reading the manuscript.

I also wish to acknowledge the financial support of the National Research Council.

CHAPTER 1

INTRODUCTION

As signal propagation speeds represent a serious barrier to increasing the speed of strictly sequential computers more attention has been paid in recent years to the use of the parallelism intrinsic to most computational algorithms. A number of designs have appeared which utilize a number of processors which may simultaneously execute several steps of the computation (/1/-/5/), rather than overlapping of subfunctions in sequential processing.

In general, values to be used in a computation step are the results of previous computation steps. This establishes certain precedence constraints upon the computation steps.

A model of such a system, satisfying a particular class of precedence constraints, has been formulated by Karp and Miller /6/.

This thesis studies some problems arising in connection with this model, in particular termination properties (Chapter 2) and storage requirements (Chapter 3).

In this chapter we present the model and the results of previous research, and compare it with other approaches to the parallel processing.

1.1 The Karp-Miller Model

The model represents the sequencing of a parallel computation by a finite directed graph. Each node of the graph corresponds to an operation in the computation (or to a processor assigned to perform that operation). Each branch represents a first-in first-out queue of data directed from one processor to another. To describe data transformation by processors, with each node is associated a single-valued function determining the dependence of outputs on inputs. The eligibility for initiation of an operation is determined by the lengths of the queues on branches directed into its associated node.

Thus a computation is represented by a directed graph G called a <u>computation graph</u> which is given by:

(i) a set of nodes $n_1, n_2, \ldots, n_\ell$

(ii) a set of branches $d_1, \ldots d_t$, where any given branch d_p is directed from a specified node n_i to a specified node n_j,

(iii) four nonnegative integers A_p, U_p, W_p and T_p, where $T_p \geq W_p$, associated with each branch d_p.

Here, A_p gives the initial number of data words in the first-in first-out queue associated with d_p; U_p gives the number of words added to the queue whenever the operation O_i associated with n_i terminates; W_p gives the number of words removed from the queue whenever the operation O_j is initiated; and T_p is a threshold giving the minimum queue length of d_p which permits the initiation of O_j. Upon initiation of O_j only the first W_p of the T_p operands for O_j are removed from the queue.

The operation O_j associated with a given node n_j is eligible for initiation if and only if, for each branch d_p directed into n_j, the number of words in the queue associated with d_p is greater than or equal to T_p. After O_j becomes eligible for initiation, W_p words are removed from each branch d_p directed into n_j. The operation O_j

is then performed. When O_j terminates, U_q words are placed on each branch d_q directed out from n_j. The times required to perform the steps mentioned above are left unspecified by the original model as presented in /6/.

These constraints on initiation lead to the following definitions of the possible sequences of initiations associated with a given computation graph G.

Let E be a sequence of nonempty sets $S_1, S_2, .., S_N, ..$, such that each set S_N is a subset of $\{1, 2, .., \ell\}$, where ℓ is the number of nodes in G.

Let $x(j,0) = 0$, and, for $N > 0$, let $x(j,N)$ denote the number of sets S_m, $1 \leq m \leq N$, of which j is an element.

The sequence E is an <u>execution</u> of G if and only if for all N, the following conditions hold:

(i) if $j \in S_{N+1}$ and G has a branch d_p directed from n_i to n_j, then $A_p + U_p x(i,N) - W_p x(j,N) \geq T_p$;

(ii) if E is finite and of length R, then for each j there exists a node n_i and a branch d_p directed from n_i to n_j such that $A_p + U_p x(i,R) - W_p x(j,R) < T_p$.

An execution E of G is called a <u>proper execution</u> if the following implication holds:

(iii) if, for all n_i and for every branch d_p directed from n_i to n_j, $A_p + U_p x(i,N) - W_p x(j,N) \geq T_p$, then $j \in S_R$ for some $R > N$.

The sequence E may be interpreted as giving a possible temporal sequence of initiations of operations throughout the performance of the parallel computation specified by G; the occurence of S_N denotes the simultaneous initiation of O_j for all $j \in S_N$.

Condition (i) states that in order for node n_j to initiate for the $x(j,N + 1)$-th time, the queue lengths on its input branches must be greater than, or equal, to the respective branch thresholds. Condition (ii) defines the circumstance under which an execution terminates, i.e. under which the computation defined by G halts. This computation terminates when every node of G is unable to initiate. Condition (iii) requires that a node, if able to initiate, actually will do so after some finite number of initiations of other nodes.

The following example illustrates these ideas:

Example 1.1

Consider the Laguerre polynomials defined by the recurrence relation

$$L_{n+1}(x) = (2n + 1 - x)L_n(x) - n^2 L_{n-1}(x)$$

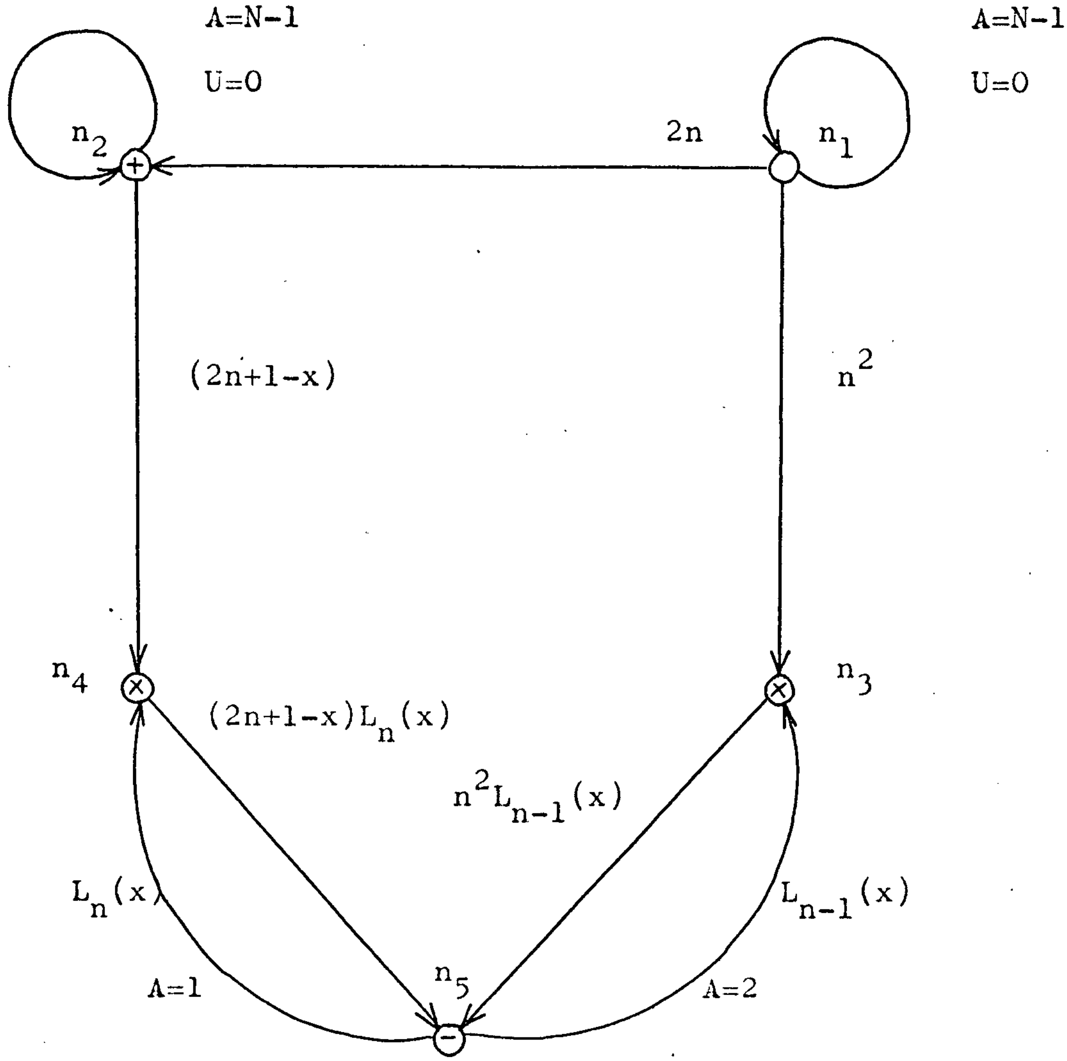
with initial conditions

$$L_0(x) = 1$$

$$L_1(x) = 1 - x$$

We want to compute the values of $L_n(x)$ for $n = 2,3,..,N$ and for a given x.

A computation graph for this calculation is in Fig.1. For each branch the intermediate result is shown. Branch coefficients are assumed to be A=0, U=W=T=1 unless otherwise shown. Branches (n_1,n_1) (i.e. the branch directed from n_1 to n_1) and (n_2,n_2) serve as counters; the computation is terminated by the depletion of queues associated with these branches. Node n_1 produces 2n and n^2, and places them on (n_1,n_2) and (n_1,n_3), respectively. Node n_2 takes



Computation graph for Laguerre polynomials

Fig. 1

$(1-x)$ from the branch (n_2,n_2) and adds it to the other input $2n$; the result $(2n+1-x)$ is placed on (n_2,n_4). Node n_4 forms the product of $(2n+1-x)$ and $L_n(x)$ and places the result on (n_4,n_5). Node n_3 multiplies $L_{n-1}(x)$ by n^2 and places the result on (n_3,n_5). Finally, n_5 produces the desired polynomials $L_2(x)$, $L_3(x),..,$ $L_N(x)$ and places them on (n_5,n_4) and (n_5,n_3). The initial data queue on (n_5,n_3) is $L_0(x)=1$, $L_1(x)=1-x$ and on (n_5,n_4) it is $L_1(x)=1-x$.

1.2 Research connected with parallel computation models

Karp and Miller /6/ show that for every proper execution the sequence of data words occuring on any branch of G is always the same thus ensuring the same computational result. This property is referred to as the determinacy of a computation graph. Also, they give an algorithm to determine whether a computation terminates, and a procedure for finding the number of performances of each operation in G. Finally, they give necessary and sufficient conditions for the lengths of data queues to remain bounded.

Reiter in his Ph.D. thesis /7/ addresses himself to the problems of storage, scheduling, and optimum assignment of operations to processing units. He gives an integer linear program for the determination of the maximum storage required by a computation graph G. He introduces a concept of an admissible schedule defining valid node initiation times and characterizes the class of all admissible schedules in the case $W_p=T_p=1$, $U_p=0$ or 1. He further shows that in this case it is possible to find a periodic admissible schedule which achieves the maximum computation rate (also see /8/). He also defines the cost of an assignment of node functions to processors and gives a method for determining feasible solutions when the maximum computation rate has a lower bound. Finally, he extends the model to incorporate a restricted form of data dependency without losing its determinacy.

A different approach to the graphical representation of a computation is taken by Martin /9/. He allows two types of node input control. In the case of conjunctive input control a node can initiate only if each branch directed into the node contains at least one data word. In the case of disjunctive input control a node can initiate only if at least one branch directed into the node contains at least one data word. Similarly, conjunctive output control places one word on each branch directed out from the node and disjunctive output control places one word on a branch directed out from the node according to an a priori probability. The latter represents a conditional transfer which is deterministic every time it occurs but over many possible data sets may be modelled probabilistically.

Note that the conjunctive input control and conjunctive output control correspond to the T=W=1 case and U=1 case in the Karp -Miller model, respectively. Also note, that because of conditional transfers, this model is not determinate.

In his work Martin studies the assignment of node computations to processors and tries to minimize the average computation time. Further research of this model can be found in /10/, where an approximative method for calculating the average computation time is given, and /11/, where procedures are given to determine a lower and an upper bound on the number of processors required for maximum parallelism.

An application of results of these studies to assembly-line balancing problems is given in /12/.

CHAPTER 2

TERMINATION PROPERTIES OF COMPUTATION GRAPHS

Part 2.1 of this chapter is devoted to a presentation of the Karp-Miller algorithm. The algorithm is used to determine whether the computation specified by a given computation graph terminates, and to find the number of performances of each operation in case the computation terminates.

Several theorems are given in parts 2.2 and 2.3 to improve the efficiency of the algorithm.

2.1 The Karp-Miller Algorithm

Theorems and Lemmas of section 2.1 are proved in /6/. Since the number of performances of an operation 0_j is independent of the execution considered only if this excution is proper, Karp and Miller restrict their attention to proper executions.

A node n_j of a computation graph is said to terminate if and only if (in the following iff) j occurs in only a finite number of the sets S_N of a proper execution of G. Naturally, this number is the same for all proper executions of G.

To further study the termination of properties of computation graphs we need to introduce a few concepts from graph theory.

A directed graph is called strongly connected iff given any pair of nodes n_i and n_j there exists a directed path from n_i to n_j. A special case of a strongly connected graph is the trivial graph which has only one node and no branches.

For any directed graph there exists a unique partition of its nodes into equivalence classes as follows:

Two nodes n_i and n_j are in the same class if there exists a directed path from n_i to n_j, and a directed path from n_j to n_i.

A subgraph consisting of the nodes of an equivalence class and the branches of the original graph connecting these nodes is then a strongly connected graph. The subgraphs corresponding to the node equivalence classes are maximal strongly connected subgraphs of the original graph and are called the <u>strong components</u> of the graph. These strong components play a crucial role in the Karp-Miller algorithm.

<u>Lemma 2.1</u>

Let G' be a strongly connected subgraph of a computation graph G. Then either every node of G' terminates or none does.

We say that <u>G' terminates</u> if every node of G' terminates.

Divide all the nodes of a computation graph G into two classes according to whether they terminate or not. Then by Lemma 2.1 the set S of terminating nodes is a union of sets of nodes of some strong components of G.

The strongly connected subgraph G' which would terminate if it were isolated from the rest of G is called <u>self-terminating</u>.

Let us examine when a strong component is self-terminating.

<u>Lemma 2.2</u>

Let $d_p=(n_i,n_j)$ be a branch of a computation graph. Then n_j terminates if n_i terminates.

This leads to the following concepts: Let G^r and G^s be strong components of G. Then define $G^r \geq G^s$ if either $G^r \equiv G^s$ or there exist such nodes $n_i \in G^r$ and $n_j \in G^s$ that there is a directed path from n_i to n_j in G.

<u>Theorem 2.3</u>

A strong component G^s of a computation graph G terminates iff there exists G^r such that G^r is self-terminating and $G^r \geq G^s$.

Thus G^s terminates iff it is self-terminating or some G^r is self-terminating and there is a directed path from some $n_i \in G^r$ to some $n_j \in G^s$. Therefore to determine the set S we examine strong components for self-termination in such order that if $G^r \geq G^s$, then G^r is examined first.

Now the problem is how to determine whether a strongly connected subgraph G' is self-terminating. For this purpose Karp and Miller use properties of the loops contained in G'.

A computation graph L is called a <u>loop</u> if it consists of distinct nodes $n_1, n_2, .., n_\ell$ and branches $d_1, d_2, .., d_\ell$ such that d_k is directed from n_k to n_{k+1}, $k=1,2,..,\ell-1$, and d_ℓ is directed from n_ℓ to n_1.

Here we note that any strongly connected subgraph G' except the trivial graph contains at least one loop.

<u>Theorem 2.4</u>

A strongly connected subgraph G' is self-terminating iff G' contains a self-terminating loop.

Given this theorem the only problem is how to establish that a particular loop is self-terminating.

Let L be a loop with branches $d_1, d_2, .., d_\ell$. The product $g = \prod_{i=1}^{\ell} \frac{U_i}{W_i}$ is called the <u>gain</u> of the loop. There are 3 cases: $g < 1$, $g = 1$, and $g > 1$.

<u>Loops with $g < 1$</u>

<u>Theorem 2.5</u>

Any loop L for which $g < 1$ is self-terminating.

<u>Loops with $g=1$</u>

<u>Theorem 2.6</u>

Let

$$P = \begin{bmatrix} 1 & | & \frac{U_{\ell-1}}{W_{\ell-1}} \frac{U_\ell}{W_\ell} & \frac{U_\ell}{W_\ell} \\ \frac{U_1}{W_1} & | & \frac{U_{\ell-1}}{W_{\ell-1}} \frac{U_\ell}{W_\ell} \frac{U_1}{W_1} & \frac{U_\ell}{W_\ell} \frac{U_1}{W_1} \\ \frac{U_1}{W_1} \frac{U_2}{W_2} & | & \frac{U_{\ell-1}}{W_{\ell-1}} \frac{U_\ell}{W_\ell} \frac{U_1}{W_1} \frac{U_2}{W_2} & \frac{U_\ell}{W_\ell} \frac{U_1}{W_1} \frac{U_2}{W_2} \\ \vdots & | & \vdots & \vdots \\ \frac{U_1}{W_1} \frac{U_2}{W_2} \cdots \frac{U_{\ell-1}}{W_{\ell-1}} & | & \frac{U_{\ell-1}}{W_{\ell-1}} & 1 \end{bmatrix}$$

$$\beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \cdot \\ \cdot \\ \beta_\ell \end{bmatrix} \qquad \text{where} \quad \beta_k = \frac{A_{k-1}-T_{k-1}+1}{W_{k-1}} \qquad k=2,3,..,\ell$$

$$\text{and} \quad \beta_1 = \frac{A_\ell - T_\ell + 1}{W_\ell}$$

Then the necessary condition for self-termination of a loop with g=1 is

$$P\beta \leq 0$$

Karp and Miller give a necessary and sufficient condition for self-termination in the following special case:

<u>Theorem 2.7</u>

If, for $1 \leq k \leq \ell$, $W_k = U_k = 1$, then the loop L is self-terminating iff $\sum_{k=1}^{\ell} \beta_k \leq 0$ (i.e. $\sum_{k=1}^{\ell} A_k \leq \sum_{k=1}^{\ell} (T_k - 1)$).

In case that Theorems 2.6 and 2.7 cannot be applied Karp and Miller derive an upper bound on the numbers of performances of nodes of a self-terminating loop.

<u>Theorem 2.8</u>

Let L be a self-terminating loop with g=1. Let $\underline{X}^*$ be a positive integer solution of the system

$$\alpha_1 = a$$

$$\alpha_2 = \frac{U_1}{W_1} a$$

$$\alpha_3 = \frac{U_1}{W_1} \frac{U_2}{W_2} a$$

- - - - - - -

$$\alpha_\ell = \frac{U_1}{W_1} \frac{U_2}{W_2} \cdots \frac{U_{\ell-1}}{W_{\ell-1}} a$$

where a is an arbitrary parameter. Let

$$\underline{X} = \begin{bmatrix} X_1 \\ X_2 \\ \cdot \\ \cdot \\ X_\ell \end{bmatrix}$$

where X_k is the number of performances of node n_k, $k=1,2,..,\ell$. Then at least one component of $\underline{X}$ is less than the corresponding component of $\underline{X}^*$.

Loops with $g > 1$

Theorem 2.6 is valid also for this case.

If L is self-terminating then we get the following upper bound:

$$\underline{X} \leq \frac{1}{1-g} P \beta$$

Having obtained an upper bound for $\underline{X}$ we can test self-termination of a loop by applying a procedure given in the following theorem.

Theorem 2.9 #

For all nodes $n_j \in S$ the following iteration scheme converges in a finite number of steps to $\underline{X}$;

$$x^{(0)}(j) = 0$$
$$x^{(n+1)}(j) = \max\left\{ x^{(n)}(j), \left\lceil \min_{(i,p)\in\Sigma_j,\ i\in S} \left(\frac{A_p - T_p + 1 + U_p x^{(n)}(i)}{W_p}\right)\right\rceil \right\}$$

Here Σ_j is the set of ordered pairs (i,p) such that d_p is a branch from node n_i to node n_j.

The results given so far may be organized into an algorithm for determining which nodes of a computation graph G terminate and, for the terminating nodes n_j, computing the number of performances $x(j)$. This algorithm may be outlined /6/ as follows:

Step 1. From among the strong components of the computation graph being considered (initially this graph is G), select one which is not covered by any other subgraph. Call it G'.

Step 2. By applying Steps 2A,...,2D given below, test whether G' is self-terminating and, when it is, determine $x(j)$ for each $n_j \in G'$.

Step 3. Form a new computation graph as follows: If G' is not self-terminating, remove G' and all branches incident with nodes of G'. If G' is self-terminating, replace each branch d_p from $n_i \in G'$ to $n_j \notin G'$ by an "equivalent" branch $d_{p'}$ from n_j to n_j, having $U_{p'}=0$, $A_{p'}=A_p+U_p x(i)$, $T_{p'}=T_p$, and $W_{p'}=W_p$. Then remove G'.

Step 4. If the new computation graph is nonempty, return to Step 1. Otherwise the analysis of termination is complete.

The symbol $\lceil x \rceil$ denotes "least integer greater than or equal to x."

The details of Step 2 are now described.

<u>Step 2A</u>. If G' contains a branch d_p with $U_p=0$, go to Step 2D. If not, determine whether G' contains a loop with $g<1$. This is equivalent to determining whether there is a loop L such that

$$\sum_{d_p \in L} \log (U_p/W_p) < 0.$$

This determination can be carried out by a shortest-route algorithm given in /13/. Enumeration of loops is not required in this procedure. If a loop with $g<1$ exists, go to Step 2D; otherwise, go to Step 2B.

<u>Step 2B</u>. Every loop of G' has $g \geq 1$. Determine whether there is a loop not previously considered such that $0 \geq P\beta$. If no such loop exists, G' is not self-terminating; return to Step 3. If such a loop L is found, determine upper bounds on the quantities $x_L(k)$ by the methods given above. These bounds hold, of course, only if L is self-terminating.

<u>Step 2C</u>. Continue applying the iteration scheme of Theorem 2.9, taking S to be set of nodes of G', until either

(a) it terminates, establishing that G' is self-terminating, and giving $x(j)$ for each $n_j \in G'$, or

(b) for some n and some k, $x^{(n)}(k)$ exceeds the upper bound on $x_L(k)$, establishing that L is not self-terminating. Return to Step 2B.

<u>Step 2D</u>. G' is self-terminating. Apply the iteration scheme of Theorem 2.9, taking S to be the set of nodes of G', to obtain $x(j)$ for each $n_j \in G'$. Return to Step 3.

2.2 Some necessary and sufficient conditions for self-termination of loops

As shown above, the Karp-Miller algorithm is based on termination properties of loops. Consequently its efficiency depends mainly on the means available for testing self-termination of loops. In the following we shall derive some theorems testing these properties.

Theorem 2.10

If, for $1 \leq k \leq \ell$, $\frac{W_k}{U_k} = 1$, then the loop L is self-terminating iff

$$\sum_{k=1}^{\ell} \lceil \beta_k \rceil \leq 0$$

PROOF: By Theorem 6 of /6/ the necessary and sufficient condition for self-termination of a loop is the existence of a nonnegative integer solution of the following system of inequalities:

$$x(1) \geq \frac{A_\ell - T_\ell + 1 + U_\ell\, x(\ell)}{W}$$

$$x(k+1) \geq \frac{A_k - T_k + 1 + U_k x(k)}{W_k} \qquad \text{for } k=1,2,..\ell-1$$

This system reduces to

$$x(1) \geq \beta_\ell + x(\ell)$$

$$x(k+1) \geq \beta_k + x(k) \qquad \text{for } k=1,2,..,\ell-1$$

Since all x's are integers, we have

$$x(1) \geq \lceil \beta_\ell \rceil + x(\ell)$$

$$x(k+1) \geq \lceil \beta_k \rceil + x(k) \qquad \text{for } k=1,2,..,\ell-1$$

By summing left and right-hand sides of the above inequalities, respectively, we get the necessary condition

$$\sum_{k=1}^{\ell} \lceil \beta_k \rceil \leq 0$$

To prove that it is also a sufficient condition we can show that if it is satisfied the system

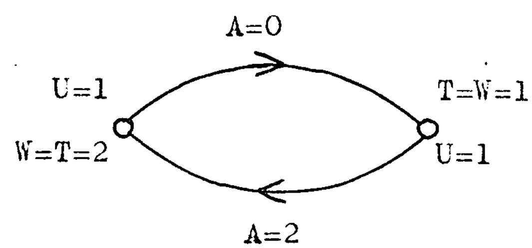
$$x(1)=C$$

$$x(k+1)=x(k)+\lceil \beta_k \rceil \qquad k=1,2,..,\ell-1$$

where C is a sufficiently large integer is a nonnegative integer solution of the above inequalities.

Consider the following examples

<u>Example 2.1</u>



Here g=1/2. The data distribution after each node performes once is

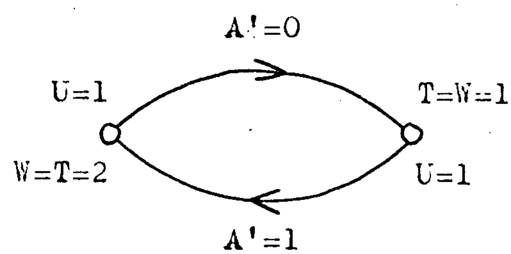
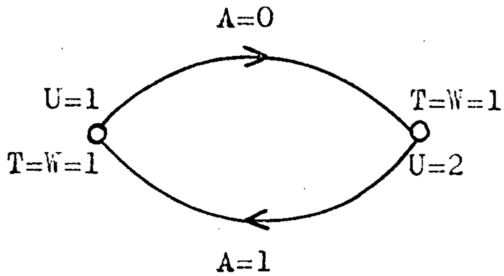
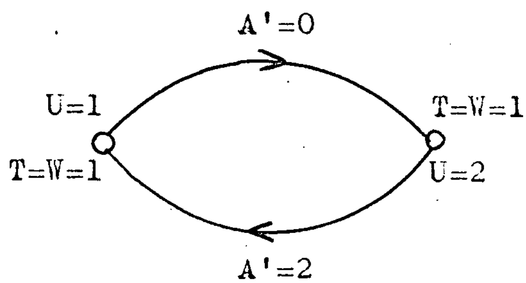


Fig.2

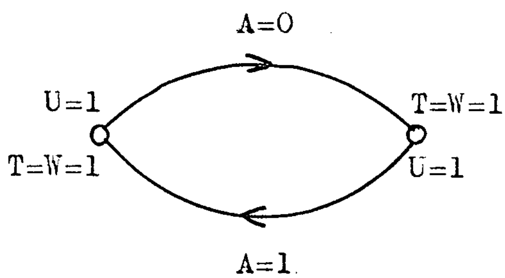
Example 2.2



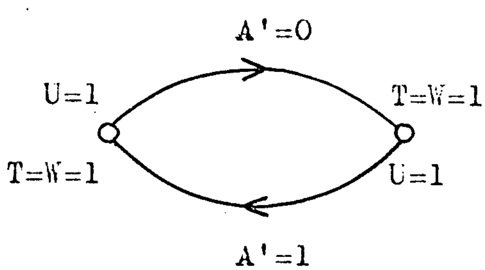
Here g=2. The data distribution after each node performs once is



Example 2.3



Here g=1. The data distribution after each node performs once is



Dependence of the amount of data on the loop gain

Fig.2

These examples indicate that (roughly speaking) for $g>1$ the "amount of data" increases, for $g<1$ it decreases, and for $g=1$ it remains constant. The following theorem gives this fact a precise form.

<u>Theorem 2.11</u>

Let L be a loop with gain $g = \prod_{i=1}^{\ell} \frac{U_i}{W_i}$. Let A_i^0 and A_i be the initial and current number of words on the i-th branch, respectively, for $i=1,2,..,\ell$. Then

$$\text{if } g<1 \qquad \sum_{i=1}^{\ell} c_i A_i \leq \sum_{i=1}^{\ell} c_i A_i^0$$

$$\text{if } g=1 \qquad \sum_{i=1}^{\ell} c_i A_i = \sum_{i=1}^{\ell} c_i A_i^0$$

$$\text{if } g>1 \qquad \sum_{i=1}^{\ell} c_i A_i \geq \sum_{i=1}^{\ell} c_i A_i^0$$

$$\text{where} \qquad c_1 \triangleq 1$$

$$\text{and} \qquad c_i \triangleq \prod_{j=1}^{i-1} \frac{W_j}{U_{j+1}} \qquad \text{for } i=2,3,..,\ell.$$

<u>PROOF</u>: Suppose that W_j words are taken from the j-th branch $(j\neq\ell)$, and U_{j+1} words are placed on the (j+1)-th branch. Taking into account that

$$c_{j+1} = c_j \frac{W_j}{U_{j+1}}$$

the change in the sum $\sum_{i=1}^{\ell} c_i A_i$ is

$$c_{j+1}U_{j+1} - c_jW_j = c_j \frac{W_j}{U_{j+1}} U_{j+1} - c_jW_j = 0$$

Now suppose that W_ℓ words are taken from the ℓ-th branch and U_1 words are placed on the 1-st branch. We have

$$c_1 = 1$$

$$c_\ell = \frac{W_1}{U_2}\frac{W_2}{U_3}\cdots\frac{W_{\ell-1}}{U_\ell} = g\,\frac{U_1}{W_\ell}$$

and the change in the sum

$$c_1U_1 - c_\ell W_\ell = U_1 - \frac{1}{g}\frac{U_1}{W_\ell}W = U_1(1 - \frac{1}{g})$$

is positive for $g > 1$, negative for $g < 1$, and zero for $g=1$.

As a corollary we get a necessary condition for self-termination of a loop, which is essentially equivalent to Theorem 2.6.

Corollary 2.12

A necessary condition for self-termination of a loop with $g \geq 1$ and initial number of words on the i-th branch A_i^0, $i=1,2,..,\ell$ is that

$$\sum_{i=1}^{\ell} c_iA_i^0 \leq \sum_{i=1}^{\ell} c_i(T_i-1)$$

PROOF: If the loop is self-terminating, then after a finite number of performances we must have

$$A_i \leq T_i - 1 \qquad \text{for } i=1,2,..,\ell$$

and

$$\sum_{i=1}^{\ell} c_i A_i \leq \sum_{i=1}^{\ell} c_i (T_i - 1)$$

By Theorem 2.11

$$\sum_{i=1}^{\ell} c_i A_i^0 \leq \sum_{i=1}^{\ell} c_i A_i \leq \sum_{i=1}^{\ell} c_i (T_i - 1)$$

Now we shall derive a sufficient condition for loops with g=1.

<u>Lemma 2.13</u>

Let L be a loop with g=1, and $W_i = T_i$ for $i=1,2,..,\ell$. Let A_i^0 be the initial number of data words on the i-th branch for $i=1,2,..,\ell$. If

$$\sum_{i=1}^{\ell} c_i A_i^0 < \max_{1 \leq j \leq \ell} c_j W_j$$

then the loop is self-terminating.

<u>PROOF</u>: Let k be such number that

$$\max_{1 \leq j \leq \ell} c_j W_j = c_k W_k$$

Then

$$\sum_{i=1}^{\ell} \frac{c_i}{c_k} A_i^0 < W_k$$

Since

$$\sum_{i=1}^{\ell} c_i A_i^0 = \sum_{i=1}^{\ell} c_i A_i$$

where A_i is the current number of words on the i-th branch we have

$$A_k \leq \sum_{i=1}^{\ell} \frac{c_i}{c_k} A_i = \sum_{i=1}^{\ell} \frac{c_i}{c_k} A_i^0 < W_k = T_k$$

Thus the number of words on the k-th branch will never reach the threshold and the number of performances of node n_k is zero. By Lemma 2.2 every node of the loop terminates. Hence the theorem.

Theorem 2.14

Let L be a loop with g=1. Let A_i^0 be the initial number of data words on the i-th branch, $i=1,2,..,\ell$. A sufficient condition for self-termination of L is that

$$\sum_{i=1}^{\ell} c_i A_i^0 < \sum_{i=1}^{\ell} c_i (T_i - W_i) + \max_{1 \leq j \leq \ell} c_j W_j$$

PROOF: Suppose that L does not terminate. Then after each node performed at least once,

$$A_i \geq T_i - W_i, \qquad \text{i.e.} \qquad A_i = (T_i - W_i) + A_i'$$

where

$$A_i' \geq 0 \qquad \text{for } i=1,2,..,\ell.$$

If we replace A_i and T_i by A_i' and $T_i' = W_i$, respectively, the resulting loop L' will have the same termination properties as L, i.e. will not terminate.

Then by Lemma 2.13

$$\sum_{i=1}^{\ell} c_i' A_i' \geq \max_{1\leq j\leq \ell} c_j' W_j'$$

Since $c_i'=c_i$ for $i=1,2,..,\ell$

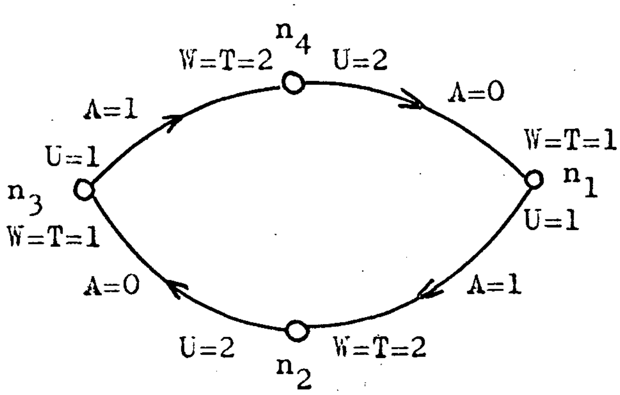
$$\sum_{i=1}^{\ell} c_i A_i^0 = \sum_{i=1}^{\ell} c_i A_i = \sum_{i=1}^{\ell} c_i (T_i - W_i) + \sum_{i=1}^{\ell} c_i A_i' = \sum_{i=1}^{\ell} c_i (T_i - W_i) + \sum_{i=1}^{\ell} c_i' A_i' \geq$$

$$\sum_{i=1}^{\ell} c_i (T_i - W_i) + \max_{1\leq j\leq \ell} c_j' W_j' = \sum_{i=1}^{\ell} c_i (T_i - W_i) + \max_{1\leq j\leq} c_j W_j$$

which is a contradiction.

To illustrate how strong the conditions of Corollary 2.12 and Theorem 2.14 are consider the following two simple examples;

<u>Example 2.4</u>



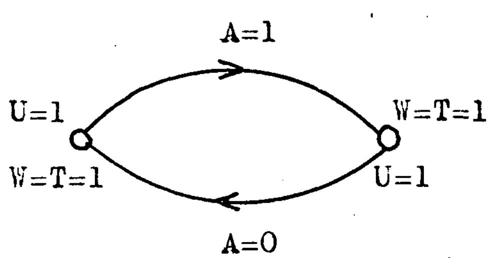
Here $c_1=c_3=1$

$c_2=c_4=2$

$\sum c_i A_i^0 = 2 = \sum c_i (T_i - 1)$

and the loop terminates

<u>Example 2.5</u>



Here $c_1=c_2=1$ and

$\max_{1\leq j\leq 2} c_j W_j = 1$

$\sum c_i A_i^0 = 1 = \sum c_i (T_i - W_i) + \max_{1\leq j\leq 2} c_j W_j$

and the loop does not terminate

Termination of loops

Fig.3

2.3 A direct method of testing termination properties of strong components.

As noted in /6/, the Karp-Miller algorithm requires the inspection of each loop of a strong component G' when G' is not self-terminating. If G' contains many loops a more direct way is desirable.

Consider the shortest-route algorithm given in /13/. It is used in Step 2A of the Karp-Miller algorithm for testing termiation properties (see part 2.1). If we use multiplication and associate U_p/W_p with branch d_p, rather than addition and $\log(U_p/W_p)$, then, in the absence of loops with $g<1$, the algorithm results in assigning a rational number to each node of G'.

On multiplying these numbers by the least common product of their denominators each node n_i will have an integer α_i assigned. If $d_p=(n_i,n_j)$ is a branch of a loop L with g=1, then $\alpha_j/\alpha_i=U_p/W_p$. If L has $g>1$, then for one and only one branch do we have $\alpha_j/\alpha_i<$ $<U_p/W_p$; for other branches of L $\alpha_j/\alpha_i=U_p/W_p$.

Consider now an arbitrary loop L of G' with g=1. If L is self-terminating then by Theorem 2.8 the number of performances is for at least one node $n_j \in L$ less than α_j.

We shall show that the same is valid for loops with $g>1$.

Theorem 2.15

If L is a self-terminating loop with $g>1$, then at least one component of $\underline{X}$ is less than the corresponding component of $\underline{X}^*$. Here

$$\underline{X} = \begin{bmatrix} X_1 \\ X_2 \\ \cdot \\ \cdot \\ X_\ell \end{bmatrix} \qquad \underline{X}^* = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \cdot \\ \cdot \\ \alpha_\ell \end{bmatrix} = \begin{bmatrix} a \\ a(U_1/W_1) \\ \cdot \\ \cdot \\ a(U_1/W_1)(U_2/W_2)\cdots(U_{\ell-1}/W_{\ell-1}) \end{bmatrix}$$

where a is an arbitrary parameter and X_i is the number of performances of node n_i, $i=1,2,..,\ell$.

<u>PROOF:</u> By Theorem 4 of /6/, $\underline{X}$ is the minimum nonnegative solution of $(E-A)\underline{X} \geq \underline{\beta}$, i.e.

$$\begin{bmatrix} 1 & 0 & 0 & . & . & 0 & -\frac{U_\ell}{W_\ell} \\ -\frac{U_1}{W_1} & 1 & 0 & . & . & . & 0 \\ 0 & -\frac{U_2}{W_2} & 1 & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & -\frac{U_{\ell-1}}{W_{\ell-1}} & & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ . \\ . \\ X_\ell \end{bmatrix} \geq \begin{bmatrix} \beta_1 \\ \beta_2 \\ . \\ . \\ \beta_\ell \end{bmatrix}$$

where $\beta_1 = \dfrac{A_\ell - T_\ell + 1}{W_\ell}$, and $\beta_i = \dfrac{A_{i-1} - T_{i-1} + 1}{W_{i-1}}$, $i=2,3,..,\ell$.

Then

$$(E-A)\underline{X}^* = \begin{bmatrix} 1 & 0 & 0 & . & . & 0 & -\frac{U_\ell}{W_\ell} \\ -\frac{U_1}{W_1} & 1 & 0 & . & . & . & 0 \\ 0 & -\frac{U_2}{W_2} & 1 & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & -\frac{U_{\ell-1}}{W_{\ell-1}} & & 1 \end{bmatrix} \begin{bmatrix} a \\ a\,\frac{U_1}{W_1} \\ a\,\frac{U_1}{W_1}\,\frac{U_2}{W_2} \\ . \\ a\,\frac{U_1}{W_1}\,\frac{U_2}{W_2}\cdots\frac{U_{\ell-1}}{W_{\ell-1}} \end{bmatrix}$$

$$(E-A)\underline{X}^* = a \begin{bmatrix} 1-g \\ 0 \\ 0 \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \leq \underline{0}$$

Now consider the vector $\underline{X}-\underline{X}^*$.

$$(E-A)(\underline{X}-\underline{X}^*) = (E-A)\underline{X} - (E-A)\underline{X}^* \geq \underline{\beta}$$

If no component of $\underline{X}$ is less than the corresponding component of $\underline{X}^*$, then $(\underline{X}-\underline{X}^*)$ is a smaller nonnegative integer solution of $(E-A)\underline{X} \geq \underline{\beta}$ than $\underline{X}$, which is a contradiction.

Theorems 2.15, 2.8, and 2.4 serve as a basis for a procedure for testing termination properties of strong components, which may be outlined as follows:

Step 1. Apply the shortest-route algorithm modified as shown above. If a loop with $g < 1$ is found go to Step 2; otherwise continue until each node n_i of the strong component is assigned a constant α_i.

Step 2. Apply the iteration scheme of Theorem 2.9 to the nodes of the strong component G' until either

a) the scheme terminates, establishing that G' is self-terminating and giving the number of performances x(j) for each $n_j \in G'$, or

b) for some n and some k, $x^{(n)}(k)$ exceeds the upper bound α_k on x(k), establishing that G' is not self-terminating.

CHAPTER 3

STORAGE REQUIREMENTS

In the Karp-Miller model each branch $d_p=(n_i,n_j)$ represents a queue of data words which may be an output of operation O_i associated with node n_i and which may serve as an input for operation O_j associated with node n_j. Each data word has to be stored in a memory location of a computer performing the computation, and the maximum number of memory locations required becomes of interest. Chapter 3 is devoted to this problem.

3.1 Maximum storage requirement - special case

In this part we present the results of /7/.

Let us introduce a branch parameter $\tau_p > 0$: If $d_p=(n_i,n_j)$, τ_p is the fixed time required by node n_i to fetch its input data from storage, process these data, and place outputs into memory locations associated with the queue on branch d_p. Thus if n_i initiates at time t, it places U_p data words upon branch d_p at time $t+\tau_p$.

Another parameter we introduce is $\tau_i=\max\{\tau_p\}$ where the maximum is taken over all branches d_p directed out from node n_i.

A <u>schedule</u> is a set $\sigma=\{\sigma_1,\sigma_2,..,\sigma_\ell\}$ where each σ_i is a function

$$\sigma_i : \{1,2,..,X_i\} \rightarrow R$$

such that for $1 \leq k < r \leq X_i$

$$\sigma_i(k) < \sigma_i(r)$$

Here R is the set of real numbers and X_i is the number of initiations of node n_i for any proper execution of G. If $X_i=0$, σ_i is undefined.

With each σ_i we associate a function

$$x_i : R \rightarrow \{0,1,2,..,X_i\}$$

$$x_i(t)=0 \quad \text{iff either } X_i=0 \text{ or } X_i \geq 1 \text{ and } \quad t<\sigma_i(1).$$

For $1 \leq k < X_i$, $\quad x_i(t)=k \quad$ iff $\quad \sigma_i(k) \leq t < \sigma_i(k+1)$

For $X_i \geq 1$, $\quad x_i(t)=X_i \quad$ iff $\quad \sigma_i(X_i) \leq t$.

For every branch $d_p=(n_i,n_j)$ define

$$b_p^{\sigma}(t)=A_p+U_p x_i(t-\tau_p)-W_p(x_j(t)-\varepsilon_j(t))$$

where

$\varepsilon_j(t)=1$ if there exists k, $\;1 \leq k \leq X_j\;$ such that $\;\sigma_j(k)=t$

$\varepsilon_j(t)=0$ otherwise.

A schedule σ is called an <u>admissible schedule</u> if, for $j=1,2,..,\ell$

$$\sigma_j(k)=t \;\Rightarrow\; b_p^{\sigma}(t) \geq T_p$$

for all branches d_p into n_j, and for all k, $1 \leq k \leq X_j$.

A schedule σ is <u>sequential</u> if for no nodes n_i, n_j, with $n_i \neq n_j$ do we have

$$\sigma_i(k) \leq \sigma_j(r) < \sigma_i(k)+\tau_i$$

for $1 \leq k \leq X_i$, $\;1 \leq r \leq X_j$.

These definitions are to be interpreted as follows:

$\sigma_i(k)=t$ means that node n_i begins its k-th initiation at time t under the schedule σ.

$x_i(t)$ is the number of initiations of node n_i, up to and including time t, under the schedule σ.

$b_p^{\sigma}(t)$ is the number of data words on branch d_p at time t. The quantity $\varepsilon_j(t)$ is introduced for the following reason: All data transmitted to node n_j by node n_i via the branch (n_i,n_j) must first pass through storage. Then if t is a time at which n_j does not initiate, the storage requirement at time t is

$$b_p^{\sigma}(t) = A_p + U_p x_i(t-\tau_p) - W_p x_j(t).$$

If n_j initiates at time t, the number of data words in storage at time t is

$$b_p^{\sigma}(t) = A_p + U_p x_i(t-\tau_p) - W_p(x_j(t)-1)$$

An admissible schedule specifies those node initiation times corresponding to a proper execution. Thus a node n_i initiates at time t ($\sigma_i(k)=t$ for some $1 \leq k \leq X_i$) only if each branch d_p directed into n_i contains at least T_p data words at time t, ($b_p^{\sigma}(t) \geq T_p$). Finally, a sequential schedule is a schedule under which no node initiates at the same time that some other node is executing.

For any admissible schedule σ, define

$$\mu_{\sigma} = \max_t \sum_p b_p^{\sigma}(t)$$

μ_{σ} thus defines the maximum amount of storage required by the admissible schedule σ.

<u>Lemma 3.1</u>

Let σ be an admissible schedule for a computation graph G. Then there exists an admissible sequential schedule (a.s.s.) σ' such that

$$\mu_{\sigma'} \geq \mu_{\sigma}$$

This Lemma shows that in general a parallel system is more economical than a sequential system in the sense that it needs fewer storage locations.

Let

$$\mu = \max\{\mu_\sigma / \sigma \text{ is an admissible schedule}\}.$$

μ is the maximum number of memory locations that a computation graph G can require.

Corollary 3.2

$$\mu = \max(\mu_\sigma / \sigma \text{ is an a.s.s.})$$

For any admissible schedule define

$$T_\sigma = \{t / t \notin (\sigma_i(k), \sigma_i(k)+\tau_i) \quad \text{for } 0 \leq k \leq X_i,\ i=1,2,..,\ell\}.$$

T_σ are those times during which no node of G is executing under the schedule σ; however, nodes of G may just be initiating or terminating under σ at some of the times $t \in T_\sigma$.

We then have the following result:

Corollary 3.3

$$\mu = \max_\sigma \max_t \{\mu_\sigma(t) / \sigma \text{ is an a.s.s. and } t \in T_\sigma\}.$$

Write $\underline{b}$ for the column vector with p-th component A_p, $\underline{W}$ for the column vector with p-th component W_p, $\underline{T}$ for the column vector with p-th component T_p, and, for any admissible schedule σ define $\underline{b}^\sigma(t)$ to be the column vector with p-th component $b_p^\sigma(t)$, p=1,2,..,t.

Define a $t \times \ell$ matrix A with elements

$a_{pj}=W_p$ if branch d_p is directed into n_j but not also out from n_j.
$a_{pj}=-U_p$ if branch d_p is directed out from n_j but not also into n_j.
$a_{pj}=W_p-U_p$ if branch d_p is directed out from n_j into n_j.
$a_{pj}=0$ otherwise.

Finally, let $\underline{X}$ be a column vector with i-th component X_i, $i=1,2,..,\ell$.

Theorem 3.4 #

Let G satisfy $\underline{b} \geq \underline{T}-\underline{W}$. Then μ is determined by the following integer linear program:

$$\mu = |\underline{b}| - \min|A\underline{y}|$$

subject to

$$\underline{0} \leq \underline{y} \leq \underline{X}$$

$$A\underline{y} \leq \underline{b}-\underline{T}+\underline{W}$$

where each y_i, $i=1,2,..\ell$, is an integer.

Theorem 3.4 enables us to determine the maximum amount of storage required for a given computation graph, provided $\underline{b} \geq \underline{T}-\underline{W}$. What if this is not satisfied? We quote from /7/:

"In those cases where this inequality is violated, the program /of Theorem 3.4/ is inapplicable. Under such contingency one possible course of action is to simulate all possible admissible schedules for G until a distribution of data is obtained which satisfies the above ineqality, and then apply the program /of Theorem 3.4/ to this data distribution. Then the maximum storage requirement is either that obtained through the simulation phase, or that obtained by the program, whichever is the greater. In general, however, such a scheme would be impractical due to the potentially large number of possible different simulations involved."

If $\underline{x}$ is a n-dimensional vector, then define $|\underline{x}| = \sum_{i=1}^{n} x_i$.

3.2 Maximum storage requirement - general case

A node n_i of a computation graph can initiate only if for every branch d_p directed into n_i the number of data words associated with this branch is not less than the corresponding threshold T_p. Consequently, the number of data words on this branch after the initiation is not less than $T_p - W_p$.

Consider a computation graph G which does not satisfy $\underline{b} \geq \underline{T}-\underline{W}$, i.e. there exists at least one branch $d_p=(n_i,n_j)$, $d_p \in G$ such that $A_p < T_p - W_p$. Let us assume that G contains only one node terminal to such a branch. Later we shall show how to extend the results to the case of more nodes terminal to such branches.

Clearly, the data distribution will satisfy the requirement $\underline{b} \geq \underline{T}-\underline{W}$ after the first initiation of n_j. In the following we derive a method for finding the maximum storage required prior to the first initiation of n_j. Then we apply methods of part 3.1 to determine the maximum storage required after the first initiation of n_j.

Given a computation graph G modify it as follows:
For the node n_j put

$$\text{all} \quad W_{rj}=T_{rj}=0$$
$$\text{all} \quad U_{js}=0$$

Note that the data distribution of the modified graph G' is not affected by initiations of n_j. In the following parameters of G' will be primed.

Lemma 3.5

Let $\sigma = \left\{\sigma_{i_1}(k_{i_1}), \sigma_{i_2}(k_{i_2}),..,\sigma_{i_m}(k_{i_m})\right\}$ be a schedule, where $\left\{i_1, i_2,.., i_m\right\} = I \subset \left\{1,2,..,j-1,j+1,.,\ell\right\}$ and

$$\sigma_{i_1}(k_{i_1})+\tau_{i_1} \leq \sigma_{i_2}(k_{i_2}) \ , \sigma_{i_2}(k_{i_2})+\tau_{i_2} \leq \sigma_{i_3}(k_{i_3}) \quad ,\ldots,$$
$$,\ldots,\sigma_{i_{m-1}}(k_{i_{m-1}})+\tau_{i_{m-1}} \leq \sigma_{i_m}(k_{i_m}) \ .$$

Then σ is admissible for G' iff it is admissible for G; moreover $\underline{b}^{\sigma}_{G'}(t)=\underline{b}^{\sigma}_{G}(t)$ for $0\leq t \leq \sigma_{i_m}(k_{i_m})$.

PROOF: By definition

$$x_i(t)=0 \text{ iff } t \not\geq \sigma_i(1)$$
$$x_i(t)=k \text{ iff } t \geq \sigma_i(k) \text{ and } t \not\geq \sigma_i(k+1)$$

Therefore $x_i(t)=x_i'(t)$ for $i \in I$ and $0 \leq t \leq \sigma_{i_m}(k_{i_m})$,

and $x_i(t)=x_i'(t)=0$ for $i \notin I$ and $0 \leq t \leq \sigma_{i_m}(k_{i_m})$.

Also $\varepsilon_i(t)=\varepsilon_i'(t)$ for $i \in I$ and $0 \leq t \leq \sigma_{i_m}(k_{i_m})$,

and $\varepsilon_i(t)=\varepsilon_i'(t)=0$ for $i \notin I$ and $0 \leq t \leq \sigma_{i_m}(k_{i_m})$.

Let us examine $b^{\sigma}_p(t)=A_p+U_p x_r(t-\tau_p)-W_p(x_s(t)-\varepsilon_s(t))$ for $d_p=(n_r,n_s)$. There are four different cases depending on whether r and s are elements of I or not.

(i) $r \in I$, $s \in I$

Here $U_p=U'_p$, $W_p=W'_p$

and $b^{\sigma}_p(t)=A_p+U_p x_r(t-\tau_p)-W_p(x_s(t)-\varepsilon_s(t))=b'^{\sigma}_p(t)$

(ii) $r \in I$, $s \notin I$

Here $U_p=U'_p$, $x_s(t)=x'_s(t)=0$, $\varepsilon_s(t)=\varepsilon'_s(t)=0$

and $b^{\sigma}_p(t)=A_p+U_p x_r(t-\tau_p)=b'^{\sigma}_p(t)$

(iii) $r \notin I$, $s \in I$

Here $W_p=W'_p$, $x_r(t)=x'_r(t)=0$

and $b^{\sigma}_p(t)=A_p-W_p(x_s(t)-\varepsilon_s(t))=b'^{\sigma}_p(t)$

(iv) $r \notin I$, $s \notin I$

Here $x_r(t)=x'_r(t)=x_s(t)=x'_s(t)=\varepsilon_s(t)=\varepsilon'_s(t)=0$

and $b^{\sigma}_p(t)=A_p=b'^{\sigma}_p(t)$

We have thus established that

$$\underline{b}_G^{\sigma}(t) = \underline{b}_{G'}^{\sigma}(t) \qquad \text{for} \quad 0 \leq t \leq \sigma_{i_m}(k_{i_m}).$$

Moreover, for all $d_p=(n_r,n_s)$, $s \in I$ we have $T_p'=T_p$ and

$$b_p^{\sigma}(t) \geq T_p$$

iff

$$b_p'^{\sigma}(t) \geq T_p' \qquad \text{for} \quad 0 \leq t \leq \sigma_{i_m}(k_{i_m}).$$

This proves that σ is admissible for G iff it is admissible for G'.

Lemma 3.6

The maximum storage S'_{max} required for the modified graph G' is the same as the maximum storage S_{max} required for G prior to the first initiation $\sigma_j(1)$ of n_j.

PROOF: By Corollary 3.2 we need only consider sequential schedules. The proof will consists of 3 parts.

(i) Every sequential schedule $\sigma=\{\sigma_{i_1}(k_{i_1}), \sigma_{i_2}(k_{i_2}),..,\sigma_{i_m}(k_{i_m}),..\}$ can be divided into time intervals

$$\langle 0,\sigma_{i_1}(k_{i_1})), \ \langle\sigma_{i_1}(k_{i_1}),\sigma_{i_2}(k_{i_2})),..,\langle\sigma_{i_m}(k_{i_m}),\sigma_{i_{m+1}}(k_{i_{m+1}})),..$$

From the definition of $\underline{b}^{\sigma}(t)$ and $\varepsilon_i(t)$ it follows that

$$S(t) \leq S(0) \qquad \text{if } t<\sigma_{i_1}(k_{i_1})$$

and

$$S(t) \leq S(\sigma_{i_m}(k_{i_m})) \qquad \text{if } \sigma_{i_m}(k_{i_m}) \leq t \leq \sigma_{i_{m+1}}(k_{i_{m+1}})$$

where

$$S(t)=|\underline{b}^{\sigma}(t)|$$

Thus to get S_{max} it is sufficient to examine S(t) at t=0, $\sigma_{i_1}(k_{i_1})$, $\sigma_{i_2}(k_{i_2})$, $\sigma_{i_3}(k_{i_3})$,..; in other words it is sufficient to examine the integer sequence S_m where $S_m=S(\sigma_{i_m}(k_{i_m})$.

(ii) Here we shall show that S'_{max} is infinite iff S_{max} is infinite. Let N be an arbitrary integer. If S'_{max} is infinite, then there exists a schedule σ' and integer M' such that

$$S_{M'}^{\sigma'} > N$$

Take the initial part of σ' up to $\sigma'_{i_{M'}}(k_{i_{M'}})$ and omit all initiations of n_j. Then by Lemma 3.5 we get an admissible schedule σ for G which requires the same storage as σ'.

Thus for some $M<M'$ we have

$$S_M^{\sigma}=S_{M'}^{\sigma'} > N.$$

In the same fashion we can show that S'_{max} is infinite if S_{max} is infinite.

(iii) Now suppose that S'_{max} is finite. Then sequences S'_m are bounded by S'_{max} and there exists such a schedule σ' and integer M' that $S'^{\sigma'}_{M'}=S'_{max}$. Take the initial part of σ' up to $\sigma'_{i_{M'}}(k_{i_{M'}})$ and omit all the initiations of n_j. This by Lemma 3.5 will give us an a.s.s. for G with the same storage requirements as σ. Thus

$$S'_{max}=S'_{M'}=S_M \leq S_{max}$$

On the other hand, since S_{max} is finite, there exists a schedule ρ and integer N such that

$$S_N^{\rho}=S_{max}$$

The initial part of this schedule up to $\rho_{i_N}(k_{i_N})$ is an a.s.s. ρ' for G' with the same storage requirements as ρ. This gives

$$S_{max}=S_N^{\rho}=S'^{\rho'}_N \leq S'_{max}$$

From this and $S'_{max} \leq S_{max}$ we obtain

$$S_{max}=S'_{max}$$

Corollary 3.7

Let all branches of G satisfy $A_p \geq T_p-W_p$ with a possible exception of branch (n_i,n_j). Modify G as follows:

Put all $W_{rj}=T_{rj}=0$, all $U_{js}=0$

Then the maximum storage S_{max} required for G prior to the first initiation of n_j is determined by the following integer linear program:

$$S_{max}=|\underline{b}|-\min|A'\underline{y}|$$

subject to

$$0\leq y_k\leq X'_k \qquad k\neq j$$

$$0\leq y_j\leq \infty$$

$$A'\underline{y}\leq\underline{b}-\underline{T}'+\underline{W}'$$

where A', $\underline{T}'$, $\underline{W}'$, $\underline{X}'$ are parameters of the modified graph G'.

PROOF: Proof follows from Lemma 3.6 and Theorem 3.4.

Lemma 3.8

Let G be a computation graph whose all branches satisfy $A_p\geq T_p-W_p$ with a possible exception of a branch $(n_i,n_j,)$. Let $X_j>0$. Then $\underline{c}=\underline{b}^{\sigma}(t)$, where $\underline{c}$ is defined below, for some a.s.s. σ and some $t\in T_\sigma$, $t\geq\sigma_j(1)$ iff there exist such integers y_i, $i=1,2,..,\ell$ which satisfy

(i) $0\leq y_k\leq X_k \qquad k\neq j$

$1\leq y_j\leq X_j$

(ii) $\underline{c}=\underline{b}-A\underline{y}\geq\underline{T}-\underline{W}$

PROOF: Analogous to that of Theorem 2.4 of /7/.

Corollary 3.9

Let G be a computation graph whose all branches satisfy

$A_p\geq T_p-W_p$ with a possible exception of a branch (n_i,n_j).

Let $X_j > 0$. Then the maximum storage S^1_{max} required after the first initiation of n_j is determined by the following integer linear program:

$$S^1_{max} = |\underline{b}| - \min|A\underline{y}|$$

subject to

$$0 \le y_k \le X_k \qquad k \ne j$$
$$1 \le y_j \le X_j$$
$$A\underline{y} \le \underline{b} - \underline{T} + \underline{W}$$

where each y_k is integer.

Theorem 3.10

Let G be a computation graph whose all branches satisfy $A_p \ge T_p - W_p$ with a possible exception of a branch (n_i, n_j). Let $X_j > 0$. Then the maximum storage required is

$$\mu = \max(S_{max}, S^1_{max})$$

PROOF: I. a) Suppose $S_{max} \ge S^1_{max}$

There exists an a.s.s. for the modified graph G' which requires S_{max} of storage. By Lemma 3.5 this schedule is admissible also for G and therefore is an initial part of some a.s.s. for G. Thus

$$\mu \ge S_{max} \ge S^1_{max}$$

and $$\mu \ge \max(S_{max}, S^1_{max})$$

b) Now suppose $S^1_{max} \ge S_{max}$

By Corollary 3.9 $$S^1_{max} = |\underline{b}| - |A\underline{y}^0|$$

where $$|A\underline{y}^0| = \min|A\underline{y}|$$

subject to $$0 \le y_k \le X_k \qquad k \ne j$$
$$1 \le y_j \le X_j$$
$$A\underline{y} \le \underline{b} - \underline{T} + \underline{W}$$

Then by Lemma 3.8 $S^1_{max}=|\underline{b}^{\sigma}(t)|$ for some a.s.s., and

$$\mu \geq S^1_{max} \geq S_{max}$$

i.e.

$$\mu \geq \max(S_{max}, S^1_{max})$$

II. Let σ be an arbitrary schedule for G. Then

$$|\underline{b}^{\sigma}(t)| \leq S_{max} \qquad \text{for} \quad t < \sigma_j(1)$$

and

$$|\underline{b}^{\sigma}(t)| \leq S^1_{max} \qquad \text{for} \quad t \geq \sigma_j(1)$$

Thus

$$\mu \leq \max(S_{max}, S^1_{max})$$

The results of I. and II. give $\mu = \max(S_{max}, S^1_{max})$, q.e.d.

Theorem 3.10 and Corollaries 3.7 and 3.9 thus provide means for finding the maximum storage for graphs where one and only one node is terminal to a branch which does not satisfy $A_p \geq T_p - W_p$.

In case there are r such nodes, we take a subset of these nodes $\{n_{i_1}, n_{i_2}, \ldots, n_{i_s}\}$, where $0 \leq s \leq r$. For all branches directed into the nodes of the subset we put W=T=0, and for all branches directed out from the nodes of the subset we put U=0. Then to this modified graph we apply the following integer linear program:

$$S_{max} = |\underline{b}| - \min |A\underline{y}|$$

subject to

$$
\begin{array}{ll}
0 \leq y_k \leq X_k & k \neq n_{i_1}, n_{i_2}, \ldots, n_{i_s} \\
1 \leq y_{i_1} \leq X_{i_1} & \\
1 \leq y_{i_2} \leq X_{i_2} & \\
\cdots\cdots\cdots\cdots & \\
1 \leq y_{i_s} \leq X_{i_s} &
\end{array}
$$

Since there are 2^r such subsets we have 2^r linear programs. The maximum required storage is the maximum of the 2^r partial maximum storage requirements.

CHAPTER 4

CONCLUSIONS

The Karp-Miller algorithm for testing termination properties of computation graphs is based on the termination properties of loops. It is, therefore, desirable to have means for testing loops.

A quantity related to the number of data words in a loop is introduced, which decreases for $g < 1$, increases for $g > 1$, and remains constant for $g=1$, in the course of computation. This concept makes it possible to derive a simple sufficient condition (Theorem 2.14) for self-termination of loops with $g=1$, and to give a shorter and intuitively more satisfying proof (Corollary 2.12) of necessary condition of Theorem 2.6 due to Karp and Miller. In the special case that $W=U$, the necessary and sufficient condition is given (Theorem 2.10). This condition has a simple form due to the fact that data propagation has local character in this case. Since this is invalid in the general case, one probably cannot hope for a simple form of the general necessary and sufficient condition.

The Karp-Miller algorithm is not well suited for computation graphs with many loops. Therefore, in part 2.3 of Chapter 2 a direct procedure for testing termination properties of strongly connected graphs is derived. The procedure does not require inspection of every loop as in the Karp-Miller algorithm. However, it also uses the iteration scheme of Theorem 2.9, which for large graphs may be too lengthy.

Reiter in /7/ gives a linear integer program for determining the maximum amount of storage required in the special case that

$\underline{b} \geq \underline{T}-\underline{W}$. Part 3.2 of chapter 3 extends his method to cover the general case. The number of linear programs required in our method increases as 2^i where i is the number of nodes terminal to branches which do not satisfy $A_p \geq T_p - W_p$, but it appears to be more efficient than simulation of all possible schedules /7/, especially for highly parallel computations.

REFERENCES

/1/ Gregory J. and McReynolds R., "The SOLOMON computer", IEEE Trans. on Electronic Computers, vol. EC-12, pp.774-781, Dec.1963.

/2/ Schwartz J., "Large parallel computers", J.ACM, vol.13, No.1, pp.25-32, Jan.1966.

/3/ Slotnick D.L. et al.,"The ILLIAC IV computer", IEEE Trans. on Computers, vol. C-17, No.8, pp.746-757, Aug.1968.

/4/ Thurber K.J. and Myrna J.W., "System design of a cellular APL computer", IEEE Trans. on Computers, vol. C-19, No.4, pp.291--303, April 1970.

/5/ Koczela L.J. and Wang G.Y., "The design of a highly parallel computer organisation", IEEE Trans. on Computers, vol.C-18, No.6, pp.520-529, June 1969.

/6/ Karp R.M. and Miller R.E., "Properties of a model for parallel computations: determinacy, termination, queuing", SIAM J. Appl. Math., vol.14, No.6, pp.1390-1411, Nov.1966.

/7/ Reiter R., "A study of a model for parallel computations", Techn. Report No. RADC-TR-68-350, Rome Air Development Center, Nov.1968.

/8/ Reiter R., "Scheduling parallel computations", J.ACM, vol.15, No.4, pp.590-599, Oct.1968.

/9/ Martin D.F., "The automatic assignment and sequencing of computations on parallel processor systems", Ph.D. Thesis, Dept. of Eng., UCLA, Jan.1966.

/10/ Martin D.F. and Estrin G., "Path length on graph models of computations", IEEE Trans. on Computers, vol.C-18, No.6, pp.530-536, June 1969.

/11/ Baer J.L.E. and Estrin G., "Bounds for maximum parallelism in a bilogic graph model of computations", IEEE Trans. on Comp., vol.C-18, No.11, pp.1012-1014, Nov.1969.

/12/ Reiter R., "On assembly-line balancing problems", Operations Research, vol.17, No.4, pp.685-700, July-Aug.1969.

/13/ Ford L.R. and Fulkerson D.P., Flows in networks, Princeton University Press, Princeton, 1962, pp.130-134.