Properties of a Model for Parallel Computations: Determinancy, Termination, Queueing
Author(s): Richard M. Karp and Raymond E. Miller
Source: *SIAM Journal on Applied Mathematics*, Vol. 14, No. 6 (Nov., 1966), pp. 1390-1411
Published by: Society for Industrial and Applied Mathematics
Stable URL: http://www.jstor.org/stable/2946247
Accessed: 11/05/2011 22:25

Your use of the JSTOR archive indicates your acceptance of JSTOR's Terms and Conditions of Use, available at http://www.jstor.org/page/info/about/policies/terms.jsp. JSTOR's Terms and Conditions of Use provides, in part, that unless you have obtained prior permission, you may not download an entire issue of a journal or multiple copies of articles, and you may use content in the JSTOR archive only for your personal, non-commercial use.

Please contact the publisher regarding any further use of this work. Publisher contact information may be obtained at http://www.jstor.org/action/showPublisher?publisherCode=siam.

Each copy of any part of a JSTOR transmission must contain the same copyright notice that appears on the screen or printed page of such transmission.

JSTOR is a not-for-profit service that helps scholars, researchers, and students discover, use, and build upon a wide range of content in a trusted digital archive. We use information technology and tools to increase productivity and facilitate new forms of scholarship. For more information about JSTOR, please contact support@jstor.org.

Society for Industrial and Applied Mathematics is collaborating with JSTOR to digitize, preserve and extend access to *SIAM Journal on Applied Mathematics.*

http://www.jstor.org

SIAM J. APPL. MATH.
Vol. 14, No. 6, November, 1966
Printed in U.S.A.

PROPERTIES OF A MODEL FOR PARALLEL COMPUTATIONS: DETERMINACY, TERMINATION, QUEUEING*

RICHARD M. KARP AND RAYMOND E. MILLER†

Abstract. This paper gives a graph-theoretic model for the description and analysis of parallel computations. Within the model, computation steps correspond to nodes of a graph, and dependency between computation steps is represented by branches with which queues of data are associated. First, it is shown that each such *computation graph* G represents a unique computation, determined independently of operation times. Next, methods of determining whether such a computation terminates and of finding the number of performances of each computation step are developed. The maximal strongly connected subgraphs of G and the loops within these subgraphs play a central role in this analysis. For example, use is made of the result that either every computation step within a strongly connected subgraph of G is performed an infinite number of times, or none is. Finally, necessary and sufficient conditions for the lengths of data queues to remain bounded are derived.

1. Introduction. There has recently been considerable interest in the use of parallel computation as a means of increasing computation speeds [1], [2], [4], [6], [7]. In contrast with conventional digital computer operation, in which only one computation step is initiated at any moment, parallel computation procedures are based on the simultaneous execution of several computation steps. In the development of parallel computation methods, problems arise concerning the derivation of appropriate numerical algorithms, the efficient scheduling of multiple facilities, and the logical design of circuits for sequence control. In formulating such problems, it is essential to develop models for the precise description of parallel computations. Such a model must include means for defining both the computation steps and the sequencing of the steps. Since existing notations of mathematics and programming seem adequate for the definition of computation steps in parallel computation, we will concentrate on developing a model for sequencing.

One natural way to depict the sequencing of a parallel computation is to associate each step of the algorithm with a node of a directed graph, and to specify the transmission paths for data and control information by branches. The initiation of a step depends only upon the branches directed into its node, and the only constraints on the initiation of a step are those intrinsic to the algorithm. Thus, several computation steps can be initiated simultaneously. This mode of operation may be contrasted with the execution of a conventional computer program, in which the instructions are

* Received by the editors October 16, 1964, and in final revised form January 24, 1966.

† Thomas J. Watson Research Center, International Business Machines Corporation, Yorktown Heights, New York.

arranged in a fixed order and executed one at a time under the control of a single program counter. In general, the intrinsic conditions for initiation of a computation step may be complex in their dependence upon input parameters and the outcomes of previous executions of other steps.

In this study we present a model for an important special class of computations in which the sequencing is particularly simple. Within this graph-theoretic model, intermediate results are placed in first-in first-out queues associated with branches of the graph. The eligibility for initiation of a computation step is determined by the lengths of the queues on branches directed into its associated node. Severe restrictions on the type of sequence control which can be represented within the model arise from the fixed queue discipline, the fact that each data queue has a unique "source" and a unique "sink", and the fact that data dependent conditional transfers cannot be represented. On the other hand, these restrictions yield a model with the convenient property that the results of computations are independent of individual operation times (cf. Theorem 2). Moreover, the model is capable of describing the sequencing of many "inner loops" encountered in numerical computation, including some which depend on several indices.

This paper deals with the analysis of computations described within the model. It is proved that all such computations are determinate, even when the speeds of the computation steps are variable and unspecified. Also, a test to determine whether a computation terminates, and a procedure for finding the number of performances of each computation step, are derived. Finally, the temporary storage requirements of a computation, represented by data queues associated with the branches of its graph, are considered, and conditions for the queue lengths to remain bounded are derived.

Other areas of interest to which this model may apply include the allocation and scheduling of operations when the number of units is limited, and the design of sequencing and control units for parallel computation. In addition to its relationship to parallel computation, there are other contexts in which the model may be useful. In particular, the results of this paper appear to apply to the analysis of the routing of materials and information in industrial processes. Also, the "loop gain" used in studying the model is a discrete variable analogue of "gain" in network theory.

2. The model. The model we consider represents the sequencing of a parallel computation by a finite directed graph. Each node of the graph corresponds to an operation in the computation, and each branch represents a queue of data directed from one node to another. With each node is associated a single-valued function determining the dependence of outputs on inputs. The only properties of these functions that are relevant for the

present analysis are the number of arguments they require on each input branch, and the number of results they determine on each output branch. In addition, the proof of Theorem 2 requires the concept of equality of data values.

The formal specification of a computation within the model is given by a computation graph, defined as follows.

DEFINITION 1. A *computation graph* G is a directed graph consisting of:

(i) nodes $n_1, \cdots, n_l$;

(ii) branches $d_1, \cdots, d_t$, 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 $T_p \geqq W_p$, associated with each branch d_p.

The parameters of a branch d_p directed from n_i to n_j may be given the following interpretation: A_p gives the number of data words initially 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 others, if any, remain available for later initiations of O_j.

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. It is assumed that no two performances of O_j can be simultaneously initiated. 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 of n_j. The times required to perform the steps mentioned above are left unspecified by the model. In fact, these times may differ for different initiations and performances of the same operation.

The constraints on initiation just stated lead to the following definitions of the possible sequences of initiations associated with a given computation graph G. Let $\mathcal{E}$ be a sequence of nonempty sets $S_1, S_2, \cdots, S_N, \cdots$, such that each set S_N is a subset of $\{1, 2, \cdots, l\}$, 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, $1 \leqq m \leqq N$, of which j is an element.

DEFINITION 2. The sequence $\mathcal{E}$ is an *execution* 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 from n_i to n_j, then $A_p + U_p x(i, N) - W_p x(j, N) \geqq T_p$;

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

DEFINITION 3. An execution $\mathcal{E}$ of G is called a *proper execution* if the following implication holds.

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

The sequence $\mathcal{E}$ is interpreted as a possible temporal sequence of initiations of operations throughout the performance of the parallel computation specified by G; the occurrence of S_N denotes the simultaneous initiation of O_j for all $j \in S_N$. There is no implication that successive initiations are equally spaced in time.

The inequality $A_p + U_p x(i, N) - W_p x(j, N) \geqq T_p$ of Definition 2 insures that the length of the queue associated with d_p will be greater than or equal to the threshold T_p upon the termination of all initiations indicated in $S_1, S_2, \cdots, S_N$. In those cases where the stronger inequality

$$A_p + U_p(x(i, N) - 1) - W_p x(j, N) \geqq T_p$$

holds it is possible that the $x(j, N + 1)$st initiation of O_j may actually precede the termination of the $x(i, N)$th performance of O_i. It is interesting to note that there is no need to represent terminations in the definition of an execution. Condition (ii) of Definition 2 states that a computation terminates only when no operations are eligible for initiation. The stronger condition (iii) states that an operation which is eligible for initiation will actually be initiated after some finite number of initiations of other operations.

Usually the number of executions and proper executions associated with a computation graph will be large. This is a consequence of the absence from the model of assumptions about timing. If, however, the precise values of all delays were known, it would be possible to select a unique execution representing the sequence of initiations which would actually occur.

As an example of a numerical computation process described by a computation graph, let us consider the numerical solution to the Dirichlet problem for the elliptic partial differential equation:

$$\nabla^2 u(x, y) - G^2(x, y)u(x, y) = f(x, y).$$

A five-point approximation formula based on point overrelaxation is

$$u_{i,j}^{(m+1)} = a u_{i,j}^{(m)} + g_{i,j}[f_{i,j} - (u_{i-1,j}^{(m+1)} + u_{i+1,j}^{(m)} + u_{i,j+1}^{(m)} + u_{i,j-1}^{(m+1)})],$$

where i and j denote the space coordinates of mesh points and m denotes the iteration number. The computation for fixed values of i and m, with j running from 1 to n, is given by the following computation graph. The values $u_{i,j}^{(m)}$, $u_{i+1,j}^{(m)}$, $u_{i-1,j}^{(m+1)}$, $f_{i,j}$ and $g_{i,j}$, for all j, are assumed to be available in some storage medium before the computation begins.

The quadruple (A_p, U_p, W_p, T_p) is shown on each branch d_p. See Fig. 1. The nonzero A_p correspond to initial data as follows. On the branch (n_1, n_1) are placed n arbitrary markers to indicate that j must be incremented n times. On the branch (n_2, n_4) is placed the value $u_{i,1}^{(m)}$. On the branch (n_4, n_4) is placed the constant a, and on the branch (n_7, n_3) is placed the boundary value $u_{i,0}$. The operations are defined as follows.

O_1 increments index j, and places the value of j on branches (n_1, n_2) and (n_1, n_8). It is assumed that the initial value of j is zero.

O_2 fetches operands from storage. It places $u_{i,j+1}^{(m)}$ on (n_2, n_4), $f_{i,j}$ on (n_2, n_5), $g_{i,j}$ on (n_2, n_6), and the three quantities $u_{i-1,j}^{(m+1)}$, $u_{i+1,j}^{(m)}$, and $u_{i,j+1}^{(m)}$ on (n_2, n_3).

O_3 takes operands $u_{i-1,j}^{(m+1)}$, $u_{i+1,j}^{(m)}$ and $u_{i,j+1}^{(m)}$ from (n_2, n_3) and operand $u_{i,j-1}^{(m+1)}$ from (n_7, n_3), forms their sum, and places the result on branch (n_3, n_5).

O_4 takes the operand $u_{i,j}^{(m)}$ from branch (n_2, n_4), reads the constant a from branch (n_4, n_4) and places their product on (n_4, n_7).

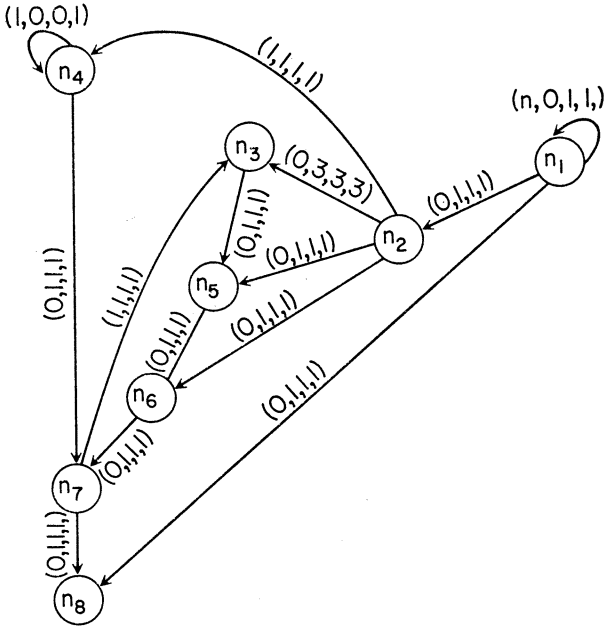

FIG. 1

O_5 subtracts the result of O_3 from $f_{i,j}$.

O_6 multiplies the result of O_5 by $g_{i,j}$.

O_7 adds the results of O_4 and O_6, and places the result on (n_7, n_3) and (n_7, n_8).

O_8 stores the result of O_7 as $u_{i,j}^{(m+1)}$.

It is easy to verify that the following sequence of sets is the beginning of some of the many possible executions of the computation graph.

$$\{1, 4\}, \{1\}, \{1, 2\}, \{2, 3\}, \{1, 4, 5\}, \{6\}, \{2, 7\}, \{8\}.$$

Inspection of this example suggests several questions.

1. How can we verify that every execution of a computation graph represents the desired computation?
2. What is the number of performances of each operation?
3. How do the lengths of the data queues behave during a computation?
4. With a limited number of units to perform operations and registers to hold temporary data, how can an allocation be found which yields an optimal schedule?
5. How can one design a simple sequence control unit to implement the sequencing of operations described by the computation graph together with a given allocation and schedule?

In the rest of this paper we derive certain properties of executions and proper executions which enable us to answer the first three of these questions. The model of a computation graph also appears to be helpful in formulating and answering the last two questions, and we hope to pursue this line of investigation further.

3. Determinacy of computation graphs. In this section we will show that the sequence of data words occurring on any branch is the same for all proper executions of a computation graph with given initial data. Thus a computation graph represents the same computation regardless of which proper execution occurs. We refer to this property as the *determinacy* of computation graphs. The derivation of this result is given in two parts. First, we prove that the number of performances of any operation, as well as the total number of words occurring in any data queue, is independent of the proper execution considered. Second, as a consequence of the fact that each operation O_j realizes a single-valued function, it is shown that the sequence of data values placed in any given queue is unique. In addition, we formulate a weakened form of determinacy applicable to all executions of a computation graph.

The following terminology will prove helpful. Let Σ_j be the set of ordered pairs (i, p) such that d_p is a branch from node n_i to node n_j.

THEOREM 1. *Let* $\mathcal{E} = S_1, S_2, \cdots, S_N, \cdots$ *and* $\mathcal{E}' = S_1', S_2', \cdots, S_N', \cdots$ *be two proper executions of a computation graph G. Let* $x(j, N)$

and $x'(j, N)$ denote the numbers of occurrences of j in $\{S_1, \cdots, S_N\}$ and $\{S_1', \cdots, S_N'\}$ respectively. Then, for all $j \in \{1, 2, \cdots, l\}$ and all S_N, there exists an S_R' such that $x(j, N) = x'(j, R)$.

Proof. The proof is by contradiction. Let $N_0 + 1$ denote the least positive integer such that, for some j (call it j_0), $x(j_0, N_0 + 1) > x'(j_0, R)$ for all R. Then, since $j_0 \in S_{N_0+1}$, $A_p + U_p x(i, N_0) - W_p x(j_0, N_0) \geqq T_p$ for all $(i, p) \in \Sigma_{j_0}$. By the minimality of $N_0 + 1$, there exists N such that, for all $(i, p) \in \Sigma_{j_0}$, $x'(i, N) \geqq x(i, N_0)$ and $x'(j_0, N) \geqq x(j_0, N_0)$. But, by assumption, $x(j_0, N_0 + 1) = x(j_0, N_0) + 1 > x'(j_0, N)$ so that $x'(j_0, N) = x(j_0, N_0)$. Thus, for all $(i, p) \in \Sigma_{j_0}$, $A_p + U_p x'(i, N) - W_p x'(j_0, N) \geqq T_p$. Therefore, since $\mathcal{E}'$ is a proper execution, there is an $R > N$ such that $j_0 \in S_R'$; giving $x'(j_0, R) \geqq x(j_0, N_0 + 1)$. This contradiction completes the proof.

Theorem 1 establishes that the number of performances of an operation O_j is the same in all proper executions of a given computation graph G. Of course, we admit the possibility that this number may be infinite for some operations.

Let us assume that each operation O_j is specified as a fixed single-valued function of the sequences of T_p operands on the branches d_p entering node n_j. Also, let d_{pv} denote the value of the vth data word to be placed on branch d_p in an execution $\mathcal{E}$ of G. Then $d_{p1}, d_{p2}, \cdots, d_{pA_p}$ are the initial data words on branch d_p. Also for $v > A_p$, Theorem 1 ensures that, if d_{pv} is defined for a proper execution $\mathcal{E}$ of G, then any other proper execution, say $\mathcal{E}'$, also produces a vth data word d'_{pv} on branch d_p.

THEOREM 2. *Let $\mathcal{E} = S_1, S_2, \cdots, S_N, \cdots$ and $\mathcal{E}' = S_1', S_2', \cdots, S_N', \cdots$ be two proper executions of a computation graph G. If for all p, $d_{pv} = d'_{pv}$, $1 \leqq v \leqq A_p$, then, for all p, $d_{pv} = d'_{pv}$ for any value of v such that d_{pv} is defined.*

Proof. Let N_0 be the least N with the following property: for some $i \in S_N$ the $x(i, N)$th performance of O_i produces an output d_{pv} such that $d_{pv} \neq d'_{pv}$. All the arguments d_{qw} of this performance of O_i satisfy $d_{qw} = d'_{qw}$, since such arguments are either initial data or are generated by operations performed earlier in $\mathcal{E}$. By assumption O_i is a single-valued function, which implies that $d_{pv} = d'_{pv}$. This contradiction completes the proof.

This theorem establishes that the initial data determine the entire sequence of data occurring on each branch, regardless of the proper execution considered. We call such behavior of the graph G *determinate*.

The following weakened form of Theorem 2 applies to all executions of G.

COROLLARY 1. *Let $\mathcal{E}$ and $\mathcal{E}'$ be two executions of a computation graph G. If, for all p, $d_{pv} = d'_{pv}$, $1 \leqq v \leqq A_p$, then, for all p, $d_{pv} = d'_{pv}$ for any value of v such that d_{pv} and d'_{pv} are both defined.*

4. Termination properties of executions. In §3 it was established that the number of performances of an operation O_j is independent of the proper execution considered. Our discussion, however, did not include a method for determining this number for each j. Sections 4 through 6 are concerned with developing results required for such a method.

It will be convenient to introduce some preliminary transformations on computation graphs which do not affect the number of performances of any operation. Consider a branch d_p of G from n_i to n_j, $i \neq j$, with $U_p = 0$. Since $U_p = 0$, no results are placed on d_p by O_i and, therefore, the number of performances of O_i cannot affect the number of performances of O_j. We may transform G by removing d_p and inserting a branch with the same parameters as d_p from n_j to n_j. Next, consider a branch d_p from n_i to n_j with $W_p = 0$ and $A_p \geqq T_p$. Since no data words are ever removed from d_p, the number of data words on d_p will always be at least T_p, and this branch will never prevent O_j from being eligible for initiation. In this case d_p can be removed from G. It can be verified directly from Definitions 2 and 3 that the executions and proper executions of the transformed graph are the same as for G. Henceforth, we consider only computation graphs which have been so transformed. In §§4 through 6 we also assume that G has no branches d_p with $W_p = 0$ and $A_p < T_p$. This restriction on G is easily removed as is shown in [5].

THEOREM 3. *Let G be a computation graph. Consider the system of inequalities*

$$x(j) \geqq \min_{(i,p)\in\Sigma_j} \left\lceil \frac{A_p - T_p + 1 + U_p x(i)}{W_p} \right\rceil \tag{1}$$

for $j = 1, 2, \cdots, l$.

If Σ_j is empty for any j, or if (1) has no solution in nonnegative integers, then every execution $\mathcal{E}$ of G is of infinite length. If (1) has a solution in nonnegative integers, then every execution $\mathcal{E}$ of G is of finite length, and for any j, the number of performances of O_j is $\mathbf{x}(j)$, the minimum value of $x(j)$ in any nonnegative integer solution of (1).

Proof. If Σ_j is empty, then condition (ii) of Definition 2 can never apply to n_j, and therefore every execution $\mathcal{E}$ of G is infinite. If any execution $\mathcal{E}$ is of length N, then from (ii) of Definition 2, for each j there exists an $(i, p) \in \Sigma_j$ such that

$$A_p + U_p x(i, N) - W_p x(j, N) < T_p,$$

which implies that the set of variables $x(j, N)$ is a solution of (1). This proves the first part of the theorem. For the second part consider an arbitrary execution $\mathcal{E}$ of G. We will show, by induction on N, that for all j and for all N, $x(j, N) \leqq \mathbf{x}(j)$. For $N = 0$, $x(j, 0) = 0 \leqq \mathbf{x}(j)$. For the in-

duction step suppose the induction hypothesis holds for $N = M$; then, $x(j, M) \leqq \mathbf{x}(j)$ for all j. Either $j \notin S_{M+1}$ or $j \in S_{M+1}$. If $j \notin S_{M+1}$, then $x(j, M+1) = x(j, M)$ so that $x(j, M+1) \leqq \mathbf{x}(j)$. Suppose $j \in S_{M+1}$; then $x(j, M+1) = x(j, M) + 1$. From (i) of Definition 2:

$$A_p + U_p x(i, M) - W_p x(j, M) \geqq T_p$$

for all $(i, p) \in \Sigma_j$. Also, from the definition of $\mathbf{x}(j)$ it follows easily that, for some $(i, p) \in \Sigma_j$,

$$U_p \mathbf{x}(i) \leqq T_p - 1 + W_p \mathbf{x}(j) - A_p .$$

Finally, from the induction hypothesis, $\mathbf{x}(i) \geqq x(i, M)$ and $\mathbf{x}(j) \geqq x(j, M)$. Combining these inequalities, we obtain

$$\mathbf{x}(j) \geqq x(j, M) + \frac{1}{W_p} .$$

Now since $\mathbf{x}(j)$ and $x(j, M)$ are integers,

$$\mathbf{x}(j) \geqq x(j, M) + 1 = x(j, M+1).$$

This completes the induction and shows that $\mathcal{E}$ is finite.

Further, if $\mathcal{E}$ is of length N, then $(x(1, N), x(2, N), \cdots, x(l, N))$ forms a solution to (1) by (ii) of Definition 2. Thus, for all j, $x(j, N) \geqq \mathbf{x}(j)$. Combining this with the induction result we obtain that $x(j, N) = \mathbf{x}(j)$, $j = 1, 2, \cdots, l$. Since $x(j, N)$ is the number of performances of O_j, this completes the proof.

From Theorem 3 it is clear that the executions of G are either all of finite length or else all of infinite length. When all executions are of finite length, every execution is proper so that the determinacy results of Theorems 1 and 2 apply to all executions. We now consider questions concerning the numbers of performances of operations in computation graphs having executions of infinite length.

DEFINITION 4. A node n_j is said to *terminate in an execution* $\mathcal{E}$ *of* G if and only if j occurs in only a finite number of the sets S_N. Equivalently, n_j terminates in $\mathcal{E}$ if and only if $\max_N x(j, N)$ exists.

The following lemma is easily established.

LEMMA 1. *The node* n_j *terminates in an execution* $\mathcal{E}$ *of the computation graph* G *if, for some* $(i, p) \in \Sigma_j$, n_i *terminates in* $\mathcal{E}$.

5. Termination properties of proper executions. In order to make further progress in studying the termination of nodes in executions, we shall henceforth restrict our attention to proper executions. It follows from Theorem 1 that, if n_j terminates in one proper execution, it terminates in every proper execution. This fact enables us to work with the following definitions in place of Definition 4 throughout the rest of the paper.

DEFINITION 5. A node n_j is said to *terminate* if it terminates in any proper execution of G.

DEFINITION 6. If G' is a subgraph of G, we say that G' *terminates* if every node of G' terminates in any proper execution of G.

When attention is restricted to proper executions, certain results concerning termination can be strengthened. For any terminating node n_j, let $\mathbf{x}(j)$ denote $\max_N x(j, N)$ in any proper execution of G. Note that this definition of $\mathbf{x}(j)$ is a consistent extension of the definition used in Theorem 3.

LEMMA 2. *The node $n_j \in G$ terminates if and only if, for some $(i, p) \in \Sigma_j$, n_i terminates. Also, if n_j terminates, there exists a pair $(i, p) \in \Sigma_j$ such that n_i terminates and*

$$\mathbf{x}(j) \geqq \frac{A_p - T_p + 1 + U_p \mathbf{x}(i)}{W_p}. \tag{2}$$

THEOREM 4. *Let S be the set of indices of nodes terminating in G. For each $j \in S$, $\mathbf{x}(j)$ is given by the minimum value of $x(j)$ in any nonnegative integer solution of*

$$x(j) \geqq \min_{(i,p)\in\Sigma_j, i\in S} \left[\frac{A_p - T_p + 1 + U_p x(i)}{W_p}\right] \qquad \text{for } j \in S. \tag{3}$$

Proof. Lemma 2 implies that the quantities $\{\mathbf{x}(j), j \in S\}$ constitute a solution of (3); thus $\mathbf{x}(j) \geqq x^*(j)$, where $x^*(j)$ represents the minimum value of $x(j)$ in any solution of (3). On the other hand, an induction on N identical to that used in the proof of Theorem 3 establishes that, for all N, $x(j, N) \leqq x^*(j)$. Thus $\mathbf{x}(j) = \max_N x(j, N) = x^*(j)$.

The proof of Theorem 4 establishes that the set of quantities $\mathbf{x}(j) = x^*(j), j \in S$, constitutes a solution of (3). Since this solution simultaneously minimizes all of the quantities $x(j)$ over the set of all nonnegative integer solutions of (3), we call it the *minimal solution* of (3).

We now have two equivalent characterizations of the number of performances of O_j, if $j \in S$. From Theorem 1, the number is given by $\max_N x(j, N)$ in any proper execution $\mathcal{E}$ we choose to consider. From Theorem 4, it is given as a component of the minimal solution of (3).

The following theorem gives an iterative method of finding the minimal solution of (3), provided that S is known.

THEOREM 5. *The following iteration scheme converges in a finite number of steps to the minimal solution of* (3); *for all $j \in S$,*

$$x^{(0)}(j) = 0,$$

$$x^{(n+1)}(j) = \max\left[x^{(n)}(j), \left[\min_{(i,p)\in\Sigma_j, i\in S} \left[\frac{A_p - T_p + 1 + U_p x^{(n)}(i)}{W_p}\right]\right]\right]. \tag{4}$$†

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

Proof. We prove by induction on n that, for all n and for $j \in S$, $\mathbf{x}(j) \geqq x^{(n)}(j)$. This is certainly true for $n = 0$. Assuming that it is true for $n = k$, we shall show that it is true for $n = k + 1$. If $x^{(k+1)}(j) = x^{(k)}(j)$, the result is immediate. Otherwise,

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

But, since $\mathbf{x}(i) \geqq x^{(k)}(i)$ for all i,

$$x^{(k+1)}(j) \leqq \left\lceil \min_{(i,p)\in\Sigma_j, i\in S} \left\lceil \frac{A_p - T_p + 1 + U_p\, \mathbf{x}(i)}{W_p} \right\rceil \right\rceil \leqq \mathbf{x}(j).$$

This completes the induction. On the other hand, if $x^{(n)}(j) = x^{(n+1)}(j)$ for all $j \in S$, then the set of values $x^{(n)}(j), j \in S$, certainly constitutes a solution of (3). This must occur for some n; otherwise, $\sum_{j\in S} x^{(n)}(j)$ grows without limit as $n \to \infty$, contradicting the fact that $\sum_{j\in S} x^{(n)}(j) \leqq \sum_{j\in S} \mathbf{x}(j)$ for all n.

Given Theorem 5, the only remaining difficulty in determining the number of performances of each operation lies in finding the set S. For those nodes which are not in S, the number of performances is infinite.

Some definitions and elementary concepts from graph theory will prove useful in characterizing S. In particular, we shall make use of the partition of the nodes of G into subsets corresponding to the so-called strong components of G. Using Lemma 1, we shall be able to show that, within such a subset, either all nodes are terminating in an execution $\mathcal{E}$, or none are; this result is stated in Lemma 3.

A directed graph is called *strongly connected* if and only if, given any pair of nodes i and j, there exists a directed path from i to j. For any directed graph there exists a unique partition of the nodes into equivalence classes as follows: two nodes i and j are in the same class if and only if there is a directed path from i to j, and a directed path from j to i. A subgraph consisting of the nodes of an equivalence class and the branches of the original graph between these nodes is strongly connected. Also, the subgraphs so defined are the maximal strongly connected subgraphs of the original graph, and are called the *strong components* of the graph.

LEMMA 3. *Let G' be a strongly connected subgraph of a computation graph G. Then, in any execution $\mathcal{E}$ of G, either every node of G' terminates or none do.*

Proof. Suppose $n_j \in G'$ does not terminate in $\mathcal{E}$. Let n_i be any other node of G'. Since G' is strongly connected, there exists some path of G' from n_i to n_j. By repeated use of Lemma 1, every node in this path, including n_i, does not terminate in $\mathcal{E}$. Since n_i is an arbitrary element of G', this completes the proof.

Lemma 3 establishes that S is a union of sets of node indices associated with strong components of G. Unfortunately, the problem of determining whether any particular strong component of G is terminating is rather complex. A direct determination of S for use in the iteration scheme (4) is not known. In place of this, we develop some methods for determining termination of strong components of G which depend upon properties of loops in these components. The remainder of this section and all of §6 are devoted to this development.

THEOREM 6. *Let G' be a strongly connected subgraph of a computation graph G. Then G' is terminating if and only if*:

(i) *for some $n_j \in G'$ there exists $(i, p) \in \Sigma_j$ such that n_i is terminating and $n_i \notin G'$; or*

(ii) *the system of inequalities*

$$x(j) \geqq \min_{(i,p)\in\Sigma_j, d_p \in G'} \left[\frac{A_p - T_p + 1 + U_p\, x(i)}{W_p}\right], \tag{5}$$

for all j such that $n_j \in G'$, has a solution in nonnegative integers.

Proof. Assume G' terminates. Then, by Lemma 2, for each j such that $n_j \in G'$, there exists an n_i such that $(i, p) \in \Sigma_j$ and n_i terminates. If, for any j, any one of these "terminating predecessors" is not an element of G', then (i) holds. Otherwise, by (2) of Lemma 2, there exists, for each j such that $n_j \in G'$, an ordered pair $(i, p) \in \Sigma_j$ such that n_i terminates, $n_i \in G'$, and

$$\mathbf{x}(j) \geqq \frac{A_p - T_p + 1 + U_p \mathbf{x}(i)}{W_p},$$

Thus, (ii) must hold, with $x(j) = \mathbf{x}(j)$ for all j. Conversely, if (i) holds, then some node n_j of G' has a terminating predecessor. By Lemma 1, n_j must terminate, and Lemma 3 establishes that G' terminates. If (ii) holds, then an induction on N, analogous to the induction used in the proof of Theorem 3, establishes that $x(j, N) \leqq x^*(j)$, where $x^*(j)$ is the minimum value of $x(j)$ in any nonnegative integer solution of (5). This completes the proof.

It should be noted that Theorem 6 holds when G' is the *trivial graph*, having only one node and no branches, if we adopt the convention that (5) has no nonnegative integer solution in this case.

Let $G^1, \cdots, G^t$ be the strong components of G.

DEFINITION 7. The subgraph G^r is said to *cover* G^s if there exist $n_i \in G^r$ and $n_j \in G^s$ such that, for some p, $(i, p) \in \Sigma_j$.

DEFINITION 8. $G^r \geqq G^s$ if $r = s$ or there exist $G^{r1}, G^{r2}, \cdots, G^{rl+1}$ such that $G^{r1} = G^r$, $G^{rl+1} = G^s$, and G^{ri} covers G^{ri+1}, $1 \leqq i \leqq l$.

It is readily shown that the relation $\geqq$ defines a partial ordering on $G^1, \cdots, G^t$; i.e., it is reflexive, antisymmetric, and transitive.

DEFINITION 9. The strongly connected subgraph G' is said to be *self-terminating* if (5) has a nonnegative integer solution.

Thus, G' is self-terminating if and only if, when it is considered by itself as a computation graph, it terminates. In particular, the trivial graph is not self-terminating.

THEOREM 7. *A strong component G^s of a computation graph G terminates if and only if there exists G^r such that G^r is self-terminating and $G^r \geqq G^s$.*

This theorem follows directly from Theorem 6. The theorem shows that the set S may be determined by examining the strong components of G in an order consistent with the partial ordering $\geqq$; if $G^r \geqq G^s$, then G^r is examined before G^s. If $G^r \geqq G^s$ and G^r has been found to be self-terminating, then G^s is terminating. If no such G^r exists, then G^s must be tested for self-termination.

6. Conditions for self-termination. At the conclusion of this section, we present an algorithm for determining which nodes of a computation graph G terminate and finding $\mathbf{x}(j)$ for each such node n_j. In view of Theorems 5 and 7, the principal problem to be solved before such an algorithm can be given is that of finding which strong components of G are self-terminating. The method that we present for testing self-termination of a given strongly connected subgraph G' uses properties of the loops contained in G'. The first part of this section is concerned with an investigation of these properties.

DEFINITION 10. A computation graph L is called a *loop* if it consists of distinct nodes $n_1, n_2, \cdots, n_l$ and branches $d_1, d_2, \cdots, d_l$ such that d_k is directed from n_k to n_{k+1}, $k = 1, 2, \cdots, l-1$, and d_l is directed from n_l to n_1.

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

For any strongly connected graph G' the condition for self-termination is the existence of a nonnegative integer solution of the system of inequalities (5). In the case of a loop, this system of inequalities reduces to:

$$
\begin{aligned}
x(1) &\geqq \frac{A_l - T_l + 1 + U_l\, x(l)}{W_l}, \\
x(k+1) &\geqq \frac{A_k - T_k + 1 + U_k\, x(k)}{W_k} \qquad \text{for } k = 1, 2, \cdots, l-1.
\end{aligned} \tag{6}
$$

THEOREM 8. *A strongly connected subgraph G' is self-terminating if and only if G' contains a self-terminating loop.*

Proof. Assume that G' is self-terminating and let $\{x(j)\}$ be a nonnegative integer solution to (5). Let $n_{k(j)}$ be a node such that:

$$\text{(i)} \qquad x(j) \geqq \frac{A_{h(j)} - T_{h(j)} + 1 + U_{h(j)}\, x(k(j))}{W_{h(j)}},$$

where $d_{h(j)}$ is a branch from $n_{k(j)}$ to n_j. For each n_j of G' such an $n_{k(j)}$ exists. For any node $n_r \in G'$ the sequence r, $k(r)$, $k(k(r))$, $\cdots$, $k^{[n]}(r)$, $\cdots$ must contain a repeated node. The loop determined by this repetition is self-terminating, since (i) is satisfied for every node n_j in the loop.

Conversely, let us assume that G' contains a self-terminating loop L. Since G' is strongly connected, one can construct a connected subgraph H containing all the nodes of G', all the branches of L, and with each node having exactly one branch directed into it. Let $d_{p(j)}$ be the branch directed into node n_j, and let $n_{h(j)}$ be the node out of which $d_{p(j)}$ is directed. Finally, let $\{x(j), j \text{ such that } n_j \in L\}$ be a nonnegative integer solution of the loop inequalities (6). Since L is the only loop in H, this solution may be uniquely extended to the remaining nodes of G' by the relation

$$x(j) = \left\lceil \frac{A_{p(j)} - T_{p(j)} + 1 + U_{p(j)}\, x(h(j))}{W_{p(j)}} \right\rceil.$$

This provides a solution, not necessarily the minimal one, to the system of inequalities (5).

Since the proof of this theorem establishes that any nonnegative integer solution of (6) may be extended to a nonnegative integer solution of (5), the following corollary is immediate.

COROLLARY 2. *Let L be a self-terminating loop contained in a strongly connected graph G'. Then the number of performances of a node $k \in L$ in any proper execution of L is greater than or equal to the number of performances of k in any proper execution of G'.*

In view of Theorem 8, we now turn our attention to methods for determining whether a strongly connected graph contains a self-terminating loop. Consider a loop L with nodes and branches as numbered in Definition 10. The system of inequalities (6) can be written in matrix form as follows:

$$\begin{bmatrix} 1 & 0 & \cdot & \cdot & \cdot & 0 & -\alpha_l \\ -\alpha_1 & 1 & 0 & & & & 0 \\ 0 & -\alpha_2 & 1 & \cdot & & & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & & & \cdot & \cdot & \cdot & 0 \\ 0 & \cdot & \cdot & \cdot & 0 & -\alpha_{l-1} & 1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x(l) \end{bmatrix} \geqq \begin{bmatrix} \beta_1 \\ \beta_2 \\ \cdot \\ \cdot \\ \cdot \\ \\ \beta_l \end{bmatrix}$$

or

$$\text{(7)} \qquad (I - A)X \geqq \beta,$$

where $\alpha_k = U_k/W_k$, $k = 1, 2, \cdots, l$, $\beta_1 = (A_l - T_l + 1)/W_l$, and $\beta_k = (A_{k-1} - T_{k-1} + 1)/W_{k-1}$, $k = 2, 3, \cdots, l$.

If L is self-terminating, let $\mathbf{X}$ be the minimum nonnegative integer solution to (7) and let $\mathbf{x}_L(k)$ be the kth component of $\mathbf{X}$. If (7) has no nonnegative integer solution, then L is not self-terminating.

The following inequality, based on an obvious lower bound on the number of words in the queue associated with d_k, is useful in obtaining criteria for self-termination of loops.

LEMMA 4. *Let* $\mathcal{E} = S_1, S_2, \cdots, S_N, \cdots$ *be an execution of a loop* L. *Then, for all* N,

$$A_k + U_k\, x(k, N) - W_k\, x(k+1, N) \geqq \min (A_k, T_k - W_k).$$

The proof is obtained by induction on N.

Thus, for any execution $\mathcal{E} = S_1, \cdots, S_N, \cdots$, and for any N,

$$(I - A)X_N \leqq \gamma, \tag{8}$$

where $x(k, N)$ is the kth component of X_N, $\gamma_1 = \max (0, A_l - T_l + W_l)/W_l$, and $\gamma_k = \max (0, A_{k-1} - T_{k-1} + W_{k-1})/W_{k-1}$, $k = 2, 3, \cdots, l$, and γ_k denotes the kth component of γ.

We now wish to investigate the consequences of (7) and (8). The determinant of $I - A$ is $1 - \alpha_1\alpha_2 \cdots \alpha_l$. We define the product $g = \alpha_1\alpha_2 \cdots \alpha_l$ to be the *gain* of the loop. The quantity g exhibits certain direct analogies to the concept of gain of a feedback loop in an electrical network. This will become apparent from our classification and analysis of loops.

LEMMA 5. *If the matrix inequality* (7) *holds, then, for any positive integer* n,

$$(I - A^n)X \geqq (A^{n-1} + A^{n-2} + \cdots + A + I)\beta. \tag{9}$$

If (8) *holds, then, for any positive integer* n,

$$(I - A^n)X_N \leqq (A^{n-1} + A^{n-2} + \cdots + A + I)\gamma. \tag{10}$$

Proof. We prove the first part of the lemma by induction on n. The result certainly holds for $n = 1$. Suppose the result holds for $n = k$; i.e.,

$$X \geqq A^kX + (A^{k-1} + A^{k-2} + \cdots + A + I)\beta.$$

Since A is a nonnegative matrix,

$$AX \geqq A(A^kX + (A^{k-1} + A^{k-2} + \cdots + A + I)\beta),$$

and

$$\begin{aligned} X \geqq AX + \beta &\geqq A(A^kX + (A^{k-1} + A^{k-2} + \cdots + A + I)\beta) + \beta \\ &= A^{k+1}X + (A^k + A^{k-1} + \cdots + I)\beta. \end{aligned}$$

The second part of the lemma is proved similarly.

Let P be used to denote the matrix $A^{l-1} + A^{l-2} + \cdots + A + I$; then

$$P = \begin{bmatrix} 1 & \cdot & \cdot & \cdot & \alpha_{l-2}\ \alpha_{l-1}\ \alpha_l & \alpha_{l-1}\ \alpha_l & \alpha_l \\ \alpha_1 & 1 & & & & \alpha_{l-1}\ \alpha_l\ \alpha_1 & \alpha_l\ \alpha_1 \\ \alpha_1\ \alpha_2 & \alpha_2 & 1 & & & & \alpha_l\ \alpha_1\ \alpha_2 \\ \cdot & & & \cdot & & & \\ \cdot & & & & \cdot & & \\ \cdot & & & & & 1 & \\ \alpha_1\ \alpha_2 \cdots \alpha_{l-1} & \cdot & \cdot & \cdot & & \alpha_{l-1} & 1 \end{bmatrix}.$$

Also, the matrix $I - A^l$ is equal to $(1 - g)I$. Setting $n = l$ in Lemma 5 we find that, for any solution X of (7),

$$(1 - g)X \geqq P\beta, \tag{11}$$

and, for any execution $\mathcal{E}$ and any N,

$$(1 - g)X_N \leqq P\gamma. \tag{12}$$

In applying these inequalities, we consider three cases: $g < 1$, $g = 1$, and $g > 1$. A general test for self-termination, based on the results obtained in these three cases, will then be given.

A. *Loops with $g < 1$.*

THEOREM 9. *Any loop L for which $g < 1$ is self-terminating.*

Proof. For any execution $\mathcal{E}$ of L and any N, (12) yields

$$X_N \leqq \frac{P\gamma}{1 - g}.$$

Since each component of X_N is bounded, $\mathcal{E}$ must terminate, and L is self-terminating.

B. *Loops with $g = 1$.* Our main purpose in this subsection is to obtain an upper bound on X for a self-terminating loop L with $g = 1$. We shall also present criteria for self-termination in some special cases.

When $g = 1$, (11) reduces to the following necessary, but not sufficient, condition for self-termination:

$$0 \geqq P\beta. \tag{13}$$

In this case, and also where $g > 1$, the inequality (12) becomes trivial.

When (13) is satisfied we need to consider the structure of (7) further to test for self-termination and determine the exact number of performances of each operation. When $g = 1$, $(I - A)$ is singular, and the homogeneous system

$$(I - A)X = 0$$

has a one-parameter family of solutions as follows:

$$x_1 = a,$$
$$x_2 = \alpha_1 a,$$
$$\vdots$$
$$x_l = \alpha_1 \alpha_2 \cdots \alpha_{l-1} a.$$

By proper choice of the parameter a, the minimum positive integer solution of this homogeneous system may be obtained; call it $\tilde{X}$.

THEOREM 10. *If L is a self-terminating loop with $g = 1$, then at least one component of $\mathbf{X}$ is less than the corresponding component of $\tilde{X}$.*

Proof. If each component of $\mathbf{X}$ were greater than or equal to the corresponding component of $\tilde{X}$, $\mathbf{X} - \tilde{X}$ would be a smaller nonnegative integer solution of (7).

In the special case where $\alpha_k = 1$, $1 \leqq k \leqq l$, we obtain the following corollary.

COROLLARY 3. *Let L be a self-terminating loop for which $\alpha_k = U_k/W_k = 1$, $1 \leqq k \leqq l$. Then $\mathbf{x}_L(k) = 0$ for some k.*

Proof. Since each of the α_k is equal to 1, each component of $\tilde{X}$ is equal to 1.

COROLLARY 4. *Let G' be a strongly connected computation graph such that $\alpha_p = U_p/W_p = 1$ for every branch d_p of G'. G' is self-terminating if and only if there is a node $n_j \in G'$ such that the number of performances of O_j in any execution of G' is zero.*

Proof. Suppose G' is self-terminating. Then by Theorem 8, G' contains a self-terminating loop L. By Corollary 3 there exists a k such that $\mathbf{x}_L(k) = 0$. Thus, by Corollary 2, the number of performances of O_k in any execution of G' is zero. Conversely, if some node n_k terminates, then G' is self-terminating.

If we make the further restriction that $W_k = U_k = 1$, $1 \leqq k \leqq l$, the following simple necessary and sufficient condition for self-termination of L may be given.

THEOREM 11. *If, for $1 \leqq k \leqq l$, $W_k = U_k = 1$, then the loop L is self-terminating if and only if $\sum_{k=1}^{l} \beta_k \leqq 0$ (i.e., $\sum_{k=1}^{l} A_k \leqq \sum_{k=1}^{l} (T_k - 1)$).*

Proof. In this case, $\alpha_k = 1$ for all k, and every element of the matrix P is 1. Accordingly, the necessary condition (13) reduces to $\sum_{k=1}^{l} \beta_k \leqq 0$. On the other hand, for any value of the parameter C the following is a solution of (7) whenever $\sum_{k=1}^{l} \beta_k \leqq 0$: $x(l) = C$, $x(1) = C + \beta_1$, $x(2) = C + \beta_1 + \beta_2, \cdots, x(l-1) = C + \beta_1 + \cdots + \beta_{l-1}$. The quantities β_k are all integers, and a nonnegative integer solution may be obtained by choosing a sufficiently large integer value for C.

C. *Loops with $g > 1$.* If L is self-terminating, $\mathbf{X}$ is a solution of (7), and therefore, the inequality (11) yields the following upper bound on $\mathbf{X}$:

$$\mathbf{X} \leqq \frac{1}{(1-g)} P\beta. \tag{14}$$

It follows that, in this case also, (13) is a necessary but not sufficient condition for self-termination.

D. *An algorithm.* 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 $\mathbf{x}(j)$. This algorithm may be outlined 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 $\mathbf{x}(j)$ 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 $U_{p'} = 0, A_{p'} = A_p + U_p \mathbf{x}(i)$, $T_{p'} = T_p$, 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 details of Step 2 are now described.

Step 2A. 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 [3, pp. 130–134]. 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.

Step 2B. Every loop of G' has $g \geqq 1$. Determine whether there is a loop not previously considered such that $0 \geqq P\beta$. 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 $\mathbf{x}_L(k)$ by the methods given in subsections B and C. These bounds hold, of course, only if L is self-terminating. When this is so, Corollary 2 establishes that the bound on $\mathbf{x}_L(k)$ is also an upper bound on $\mathbf{x}(k)$.

Step 2C. Continue applying the iteration scheme (4), taking S to be the set of nodes of G', until either

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

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

Step 2D. G' is self-terminating. Apply the iteration scheme (4), taking

S to be the set of nodes of G', to obtain $\mathbf{x}(j)$ for each $n_j \in G'$. Return to Step 3.

It should be noted that, when G' is not self-terminating, this algorithm requires the inspection of each loop of G'. A more direct way of establishing that G' is not self-terminating would be desirable, especially when G' contains many loops. No such procedure is presently known.

7. Properties of queue lengths. If one wishes to use a computer to perform a parallel computation specified by a computation graph, one must consider the number of temporary storage registers required and the allocation of these registers to queues. For this purpose, it is important to determine how the queue length on each branch varies and, in particular, to ascertain whether any queue length grows without limit. In what follows we derive necessary and sufficient conditions for the existence of upper bounds on queue lengths of branches in a computation graph G. These conditions are closely related to the termination properties of G.

Let G be a computation graph, and let d_p be a branch directed from n_i to n_j. Given an execution $\mathcal{E}$ of G, we define the quantity $q(p, N)$ as follows:

$$q(p, N) = A_p + U_p\, x(i, N) - W_p\, x(j, N).$$

The queue length on d_p immediately after the $x(j, N)$th initiation of O_j is $q(p, N)$ if the $x(i, N)$th performance of O_i has been completed; otherwise, the queue length is $q(p, N) - U_p$. Thus, any upper bound on $q(p, N)$ is also an upper bound on the queue length associated with d_p. We say that the queue length of d_p is *uniformly bounded by* M_p if, for every execution $\mathcal{E}$ and every N such that S_N occurs in $\mathcal{E}$, $q(p, N) \leqq M_p$.

THEOREM 12. *Let d_p be a branch of G from n_i to n_j. If $n_i \in G^r$ and $n_j \in G^s$, where G^r and G^s are different strong components of G, then the queue length of d_p is uniformly bounded if and only if n_i terminates.*

If n_i is a terminating node, $x(i, N)$ has $\mathbf{x}(i)$ as a uniform upper bound, and the queue length of d_p is certainly uniformly bounded. If n_i is nonterminating, it is easy to construct proper executions in which the number of performances of O_i before the first performance of O_j is arbitrarily large. This verifies the theorem.

It remains only to consider branches within a strong component G'. If G' is terminating, then the queue length of each branch of G' is certainly uniformly bounded.

THEOREM 13. *Let G' be a nonterminating strong component of G. Every loop of G' for which $g > 1$ contains at least one branch for which the queue length is not uniformly bounded. Also, the queue length for every branch which lies in a loop with $g = 1$ is uniformly bounded.*

Proof. Let L be a loop in G' consisting of the nodes $n_1, \cdots, n_l$ and branches $d_1, \cdots, d_l$, with d_k directed from n_k to n_{k+1}, $1 \leqq k \leqq l-1$, and d_l directed from n_l to n_1. Choose a proper execution $\mathcal{E}$, and consider the linear form

$$q(1,N) + \frac{W_1}{U_2} q(2,N) + \frac{W_1 W_2}{U_2 U_3} q(3,N) + \cdots + \frac{W_1 W_2 \cdots W_{l-1}}{U_2 U_3 \cdots U_l} q(l,N).$$

With the substitutions $q(k,N) = A_k + U_k x(k,N) - W_k x(k+1,N)$, $k = 1, 2, \cdots, l-1$, and $q(l,N) = A_l + U_l x(l,N) - W_l x(1,N)$, the linear form becomes

$$A_1 + \frac{W_1}{U_2} A_2 + \frac{W_1 W_2}{U_2 U_3} A_3 + \cdots + \frac{W_1 W_2 \cdots W_{l-1}}{U_2 U_3 \cdots U_l} A_l + \frac{W_1 W_2 \cdots W_{l-1}}{U_2 U_3 \cdots U_l} (-W_l x(1,N)) + U_1 x(1,N).$$

The coefficient of $x(1,N)$ may be rewritten as $U_1((g-1)/g)$. Thus, if $g = 1$, the value of the linear form is constant. In this case, since each $q(k,N)$ is bounded below by zero, a uniform upper bound on each of the quantities $q(k,N)$ must exist. On the other hand, if $g > 1$, the coefficient of $x(1,N)$ is positive. Since G' is not terminating, $x(1,N)$ grows without limit in any proper execution of G. The value of the linear form must also grow without limit, showing that some $q(k,N)$ must tend to infinity.

Theorem 13 shows that, if G' does not terminate in G, there is a branch in each loop of G' with $g > 1$ for which the queue length is not uniformly bounded. The theorem, however, does not identify exactly which branches have unbounded queue length. Conceivably, there may exist a branch of G' which does not occur in any loop with $g = 1$, but for which the queue length is nevertheless uniformly bounded. We do not know whether this is possible.

One general consequence of Theorems 12 and 13 which may be of practical value in the construction of computation graphs is the following statement. Any nonterminating computation graph in which each queue length is uniformly bounded must be strongly connected, and each loop which it contains must have gain equal to one.

8. Some possible extensions. If one considers performing a parallel computation described by a computation graph on a special-purpose or general-purpose computer, certain problems of allocation and scheduling arise. These problems involve the assignment of operations to a limited number of units available to perform them, and of data on queues to a limited amount of temporary storage. For example, if only one adder is available, no two addition operations may be in progress simultaneously.

Also, if the calculation is to be implemented with a finite amount of temporary storage, no queue length may be unbounded. A method of enforcing such constraints is to modify the computation graph by adding branches which carry control information, rather than operands or results. A simple example of this is the insertion of branches to make a nonterminating computation graph strongly connected. If all loops of the resulting graph have $g = 1$, these insertions will have the effect of making all queue lengths uniformly bounded. Another example is the insertion of branches to form a loop including all operations assigned to a given adder. By appropriate choice of the parameters of these branches, it is possible to ensure that only one of these operations is in progress at a time. It would be of interest to study the effect that such insertions have on the set of executions of the computation graph, the numbers of performances of operations, and the lengths of data queues.

Throughout this paper complete variability in the delays associated with the initiation and performance of operations has been assumed. In practice, however, these delays may be known exactly, or at least within certain bounds. It is of interest to consider how such bounds affect the set of executions actually possible, and whether they limit the amount of temporary storage required. When such bounds are given, it is possible to consider the time required for a computation, and investigate how this time is affected by the insertion of branches to enforce scheduling constraints.

There are a number of possible extensions to the computation graph model which would make its use practical for a larger class of computations. It would be desirable, for example, to incorporate data dependent decisions, queue disciplines other than the first-in first-out discipline, addressable temporary storage, or random access storage. It would be of particular interest to determine whether any such generalizations of the model preserve some form of determinacy.

Finally, it would be of interest to add to the model a specification of the particular operations allowed, to devise methods of translating from conventional programming languages into the language of computation graphs, and to characterize the class of computations that can be represented by computation graphs using a given operation set.

REFERENCES

[1] W. S. Dorn, N. C. Hsu, and T. J. Rivlin, *Some mathematical aspects of parallel computation*, RC-647, IBM Research Center, Yorktown Heights, New York, 1962.

[2] G. Estrin, B. Bussell, R. Turn, and J. Bibb, *Parallel processing in a restructurable computer system*, IEEE Trans. Electronic Computers, EC-12 (1963), pp. 747–755.

[3] L. R. FORD, JR. AND D. R. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, 1962.

[4] J. HOLLAND, *A universal computer capable of executing an arbitrary number of subprograms simultaneously*, Proc. Eastern Joint Computer Conference, 1959, pp. 108–113.

[5] R. M. KARP AND R. E. MILLER, *Properties of a model for parallel computations: determinacy, termination, queueing*, RC-1285, IBM Research Center, Yorktown Heights, New York, 1964.

[6] D. SLOTNICK, W. C. BORCK, AND R. C. MCREYNOLDS, *The SOLOMON computer*, Proc. Eastern Joint Computer Conference, 1962, pp. 97–107.

[7] E. G. WAGNER, *An approach to modular computers, I: Spider automata and embedded automata*, RC-1107, IBM Research Center, Yorktown Heights, New York, 1964.