LANGLEY GRANT
IN-61-CR
104434
71P

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
COLLEGE OF ENGINEERING & TECHNOLOGY
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23508

STRATEGIES FOR CONCURRENT PROCESSING OF COMPLEX ALGORITHMS IN DATA DRIVEN ARCHITECTURES

By

John W. Stoughton, Principal Investigator

and

Roland R. Mielke, Co-Principal Investigator

Interim Report
For the period June 1, 1986 to May 15, 1987

Prepared for the
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665

Under
Research Grant NAG-1-683
Mr. Paul J. Hayes, Technical Monitor
ISD-Information Processing Technology Branch

(NASA-CR-181435) STRATEGIES FOR CONCURRENT PROCESSING OF COMPLEX ALGORITHMS IN DATA DRIVEN ARCHITECTURES Interim Report, 1 Jun. 1986 - 15 May 1987 (Old Dominion Univ.) 71 p Avail: NTIS HC A04/MF A01 CSCL 09B G3/61

N88-10527

Unclas
0104434

November 1987

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
COLLEGE OF ENGINEERING & TECHNOLOGY
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23508

STRATEGIES FOR CONCURRENT PROCESSING OF COMPLEX ALGORITHMS IN DATA DRIVEN ARCHITECTURES

By

John W. Stoughton, Principal Investigator

and

Roland R. Mielke, Co-Principal Investigator

Interim Report
For the period June 1, 1986 to May 15, 1987

Prepared for the
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665

Under
Research Grant NAG-1-683
Mr. Paul J. Hayes, Technical Monitor
ISD-Information Processing Technology Branch

Submitted by the
Old Dominion University Research Foundation
P.O. Box 6369
Norfolk, Virginia 23508

November 1987

DISCLAIMER

The use of brand names in this document is for completeness and does not imply NASA endorsement.

ACKNOWLEDGEMENT

This research was completed under a project sponsored by the NASA-Langley Research Center, Hampton, Virginia, Grant No. NAG-1-683.

TABLE OF CONTENTS

TABLE OF CONTENTS- continued

LIST OF FIGURES

CONCURRENT PROCESSING OF COMPLEX ALGORITHMS

By

John W. Stoughton[1] and Roland R. Mielke[2]

CHAPTER 1

1.0 INTRODUCTION

This report presents the results of ongoing research directed at developing a graph theoretic model for describing data and control flow associated with the execution of large grained algorithms in a special distributed computer environment. This model is identified by the acronym ATAMM which represents Algorithm/Architecture Mapping Model. The purpose of such a model is to provide a basis for establishing rules for relating an algorithm to its execution in a multiprocessor environment. Symbolically this problem is illustrated in Figure 1.1

Specifications derived from the model lead directly to the description of a data flow architecture which is a consequence of the inherent behavior of the data and control flow described by the model. The purpose of the ATAMM based architecture is to optimize computational concurrency in the multiprocessor environment and to provide an analytical basis for performance evaluation. The ATAMM model and architecture specifications are demonstrated on a prototype system for concept validation.

The problem domain of the research reported herein consists of decision free algorithms with computationally complex primitive operations which are assumed to be implemented in a dedicated distributed multicomputer environment. The algorithms are such as may be found in (but not limited to) large

[1] Associate Professor, Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, Virginia 23508.
[2] Professor, Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, Virginia 23508.

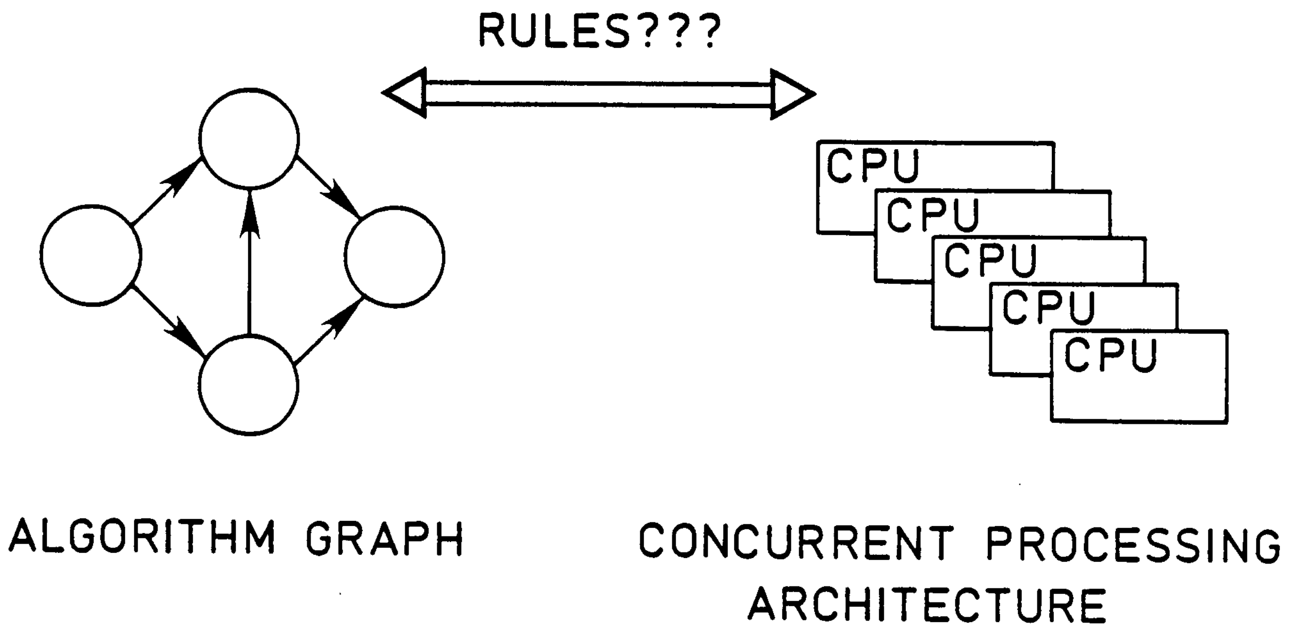


Figure 1.1 Algorithm to architecture mapping problem.

scale signal processing and control applications. The anticipated multiprocessor environment is assumed to consist of 2 to 20 processing elements for concurrent execution of the various algorithm primitives. Further, Very High Speed Integrated Circuit (VHSIC) technology incorporating the MIL-STD 1750A instruction set is the intended technology for the support of the multiprocessor environment.

From the given problem domain, the research products are the result of understanding two major areas. These areas are non Von Neumann multiprocessor architectures and Petri-net and marked graph theory which provides the theoretical basis for the ATAMM model.

Chapter 2 presents the ATAMM model development. From the model description, general specifications of a data flow architecture are generated. Chapter 3 presents an introductory discussion of performance measures. In Chapter 4, a data flow prototype of a multiprocessor architecture design based on the ATAMM specifications is described. Implementation of this prototype provides experimental verification of the ATAMM Model rules. Chapter 5 presents preliminary evaluation results from the data flow prototype.

CHAPTER 2

2.0 ATAMM MODEL DEVELOPMENT

2.1 Introduction

New computer architectures based upon multiple processor organizations for computation are motivated mainly by the desire to increase computer performance through the use of concurrency for computationally intensive applications. The development of parallel architectures composed of identical, special purpose computing elements is already a topic of great interest to many researchers. However, models for describing the behavior of algorithms in this setting do not appear to be adequate to address the complex issues of scheduling, coordination, and communication.

In this chapter, a modeling process to describe concurrent processing of decomposed algorithms is presented. The resulting model (ATAMM) consists of a Petri net marked graph which incorporates general specifications of communication and processing associated with each computational event in a multiprocessor data flow architecture. The availability of such a modeling process is important for two reasons. First, the model provides a hardware-independent context in which to investigate the relative merits of different algorithm decomposition and implementation strategies. Second, the model clearly displays the data flow and control flow which must be manifested by any data flow computer architecture implementing the decomposed algorithm. Thus the ATAMM Model provides the foundation for the development of design procedures for concurrent processing of complex algorithms.

In Section 2.2, a description of the class of problems under consideration is given. The directed graph representation of particular decomposed algorithms is described in Section 2.3. After a brief introduction to Petri-net and marked graphs in Section 2.5. The basic assumptions concerning the architectural enviroment are presented in Section 2.6. The development of the computational marked graph model in Section 2.7 completes the ATAMM model in Section 2.7 completes the ATAMM model development.

2.2 Problem Description

The computational problems of interest are decision-free computationally complex problems as are often found in signal processing and control applications. A problem description normally results in the definition of a function given by the triple (X,Y,F). The set X represents the set of admissible inputs, Y represents the set of admissible outputs, and $F: X \rightarrow Y$ is the rule of correspondence which unambiguously assigns exactly one element from Υ to each element of Ω. This functional problem statement is illustrated in Figure 2.1. Associated with a computational problem is an algorithm. An algorithm is composed of a sequentially order set of primitive operations and operands which represent the particular rule of correspondence $F: X \rightarrow Y$.

A given problem often decomposes into a number of different algorithms. In general, a given algorithm can be decomposed by several different primitive operator sets. Also, for a given primitive operator set, there are often different sequences of primitive operations which can be scheduled to carry out the algorithm. For illustration, consider the following problem. Suppose that $\Omega = \Upsilon$ is the set of (nxn) matrices with elements in R (set of real numbers.) Given a matrix $x \in \Omega$, it is desired to compute a matrix $y \in Y$ given by $y = f(x) = x^2 + ax + b$ where a and b are specified (nxn) matrices with elements in R. This algorithm can be decomposed in the two

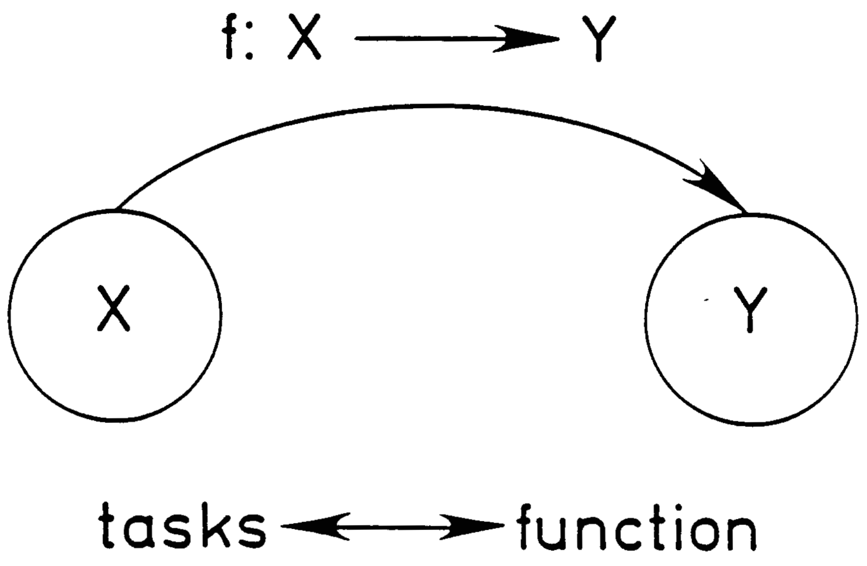


Figure 2.1 Functional correspondence.

sets of primitive operators stated below.

Primitive Operator Set One:

$$f_1(p,q) = p + q;\ f_2(p,q) = p \cdot q.$$

and

Primitive Operator Set Two

$$f_3(p,q,r) = (p \cdot q) + r.$$

Using primitive operator set one, the algorithm is represented by two different operator sequences:

$$y = f(x) = \{[(x \cdot x) + (a \cdot x)] + b)$$

$$= f_1\{f_1[f_2(x,x),\ f_2(a,x)],b\},$$

or

$$y = f(x) = \{[x \cdot (x + a)] + b\}$$

$$= f_1\{f_2[x,f_1(x,a),\ b\}.$$

Another decomposition is expressed using primitive operative set two:

$$y = f(x) = \{x \cdot [(1 \cdot x) + a] + b\}$$

$$= f_3\{x,\ f_3[1,a,x],\ b\}.$$

where the notation 1 is used to represent the (nxn) identity matrix.

2.3 Algorithm Directed Graph

An algorithm directed graph (ADG) is a directed graph which represents a specific algorithm decomposition. The graph provides a description of the operand data flow and operation sequence required by the algorithm decomposition. Vertices of the ADG are in a one-to-one correspondence with each occurence of a primitive operation. The algorithm graph contains an edge (i,j) directed from vertex i to vertex j if the output of primitive operation i is an input operand for primitive operation j. When constructing an algorithm graph, vertices (primitive operations) are displayed as circles, and edges (input-output signals) are displayed as directed line segments connecting appropriate vertices. Sources and sinks for input and output signals are represented as squares. Sources from constants are not usually included in the algorithm graph; however, triangles are used for this purpose when necessary.

To illustrate the construction of an algorithm directed graph, consider the problem of computing the output of a discrete linear system given a sequence of inputs to the system. Let the system be described by the partitioned state equation

$$\begin{matrix} x_1(k+1) \\ \\ x_2(k+1) \end{matrix} = \begin{bmatrix} A_{11} & A_{12} \\ \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1(k) \\ \\ x_2(k) \end{bmatrix} + \begin{bmatrix} B_1 \\ \\ B_2 \end{bmatrix} u(k+1)$$

and

$$y(k) = [C_1 : C_2] \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$$

where x_1 is a p-vector, x_2 is a q-vector, u is an m-vector, y is an r-vector, $p + q = n$, and A_{ij} and B_k are constant submatrices. The primitive operations are defined as matrix multiplication and vector addition, and the natural algorithm decomposition resulting from the state equation description is selected. The algorithm directed graph for this decomposed algorithm is shown in Fig. 2.2. Note that each edge is labled with the corresponding data and the nodes are labled to indicate the associated computational operation.

2.4 Petri Nets and Marked Graphs

Petri nets have been established as an appropriate models for describing or controlling systems defined by some sequence of events. Without argument, the algorithm directed graph satisfies this general aspect. Further, since computers need to communicate and be controlled on the occurence of certain events, the Petri net becomes a suitable tool to form the basis of the ATAMM model. Certain physical characteristics of the class of problems under consideration lead to a simplified Petri net representation. (For a formal description of Petri net features, the reader is referred to Appendix A.)

Considering the data flow in an algorithm directed graph, the execution of a primitive operation is preconditioned on the availability of input signals (or operands). This process may be directly modeled by a Petri-net "transition" which is "enabled" for "firing" when input "places" to the transition are marked with "tokens". Because the signal or data availability is a binary condition, it is appropriate that the tokens are limited to the set (0,1) in order to associate places (conditions) to transactions (events) in a binary way. A Petri net having such restricted input and

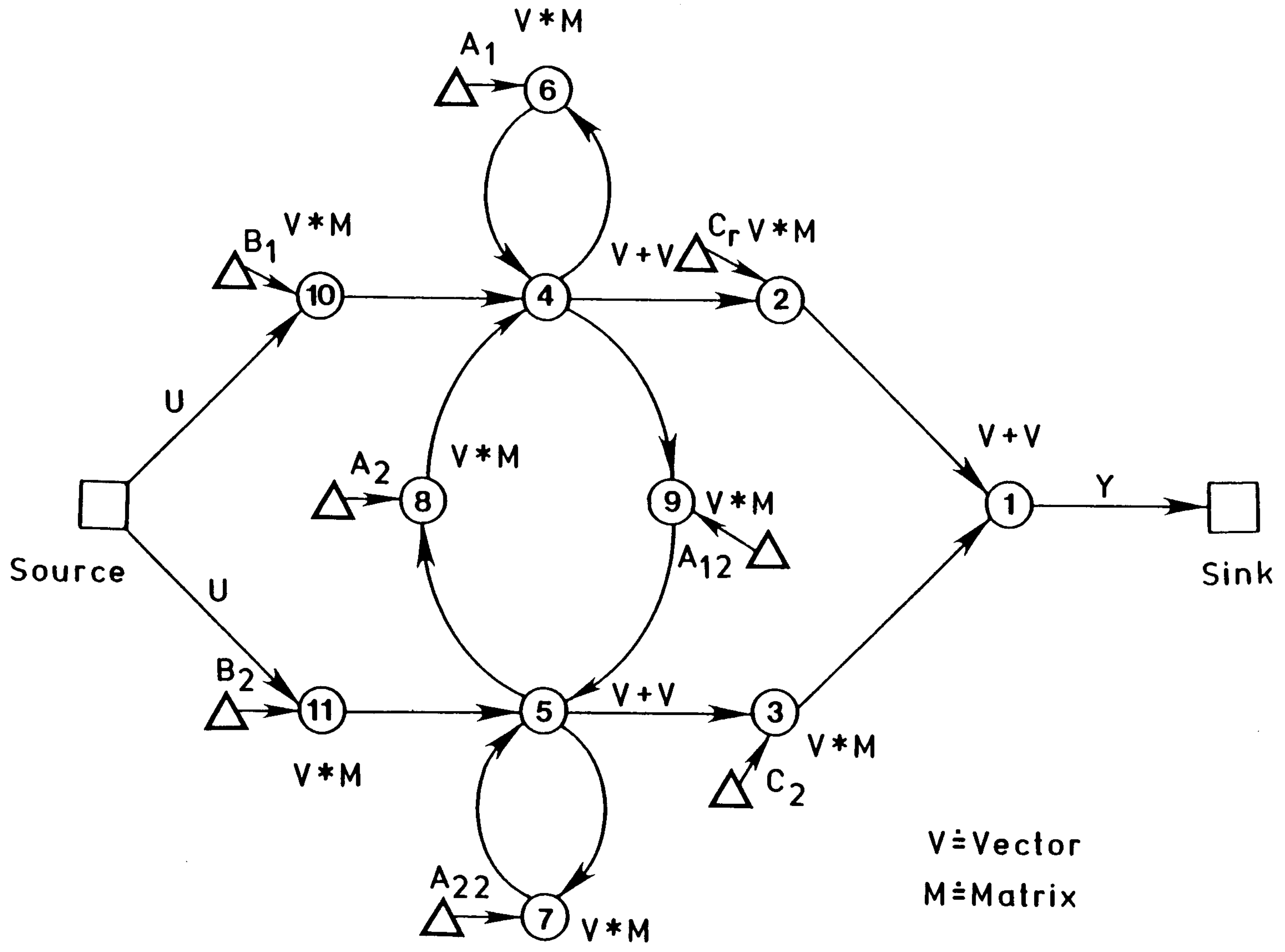


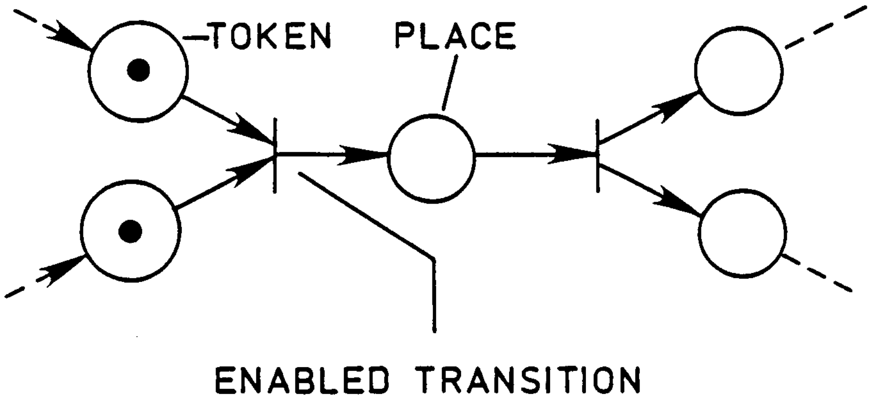
Figure 2.2 Algorithm directed graph-decomposed state equation.

output functions is called an ordinary Petri net. Figure 2.3 illustrates the ordinary Petri net features. The interpretation of places in the system model developed here is the availability of a signal. That is, the absence of a token indicates the absence of a data signal, and the presence of a token indicates the availability of a data sigal. Petri nets having such restricted markings are called safe or one-bounded Petri nets. Finally, the assumption is made that the algorithms under consideration contain no conflict or decision making such as "if then else" or "do while" statements, thus limiting the Petri net places to having one input transition and one output transition. This class of restricted Petri nets is called marked graphs. Therefore, the Petri nets used in this report are ordinary, safe marked graphs.

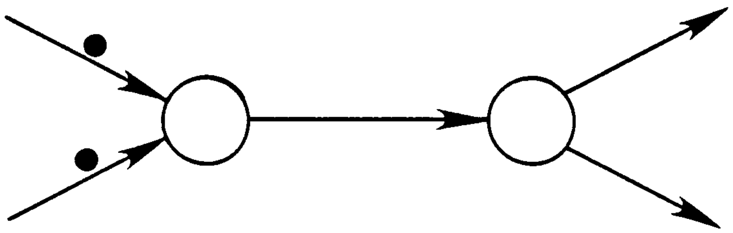
The decision to initially consider decision-free algorithms is made because the resulting marked graph models are better understood than general Petri nets. Well known properties of marked graphs hold the potential for the development of performance bounds for concurrent processing strategies. An interesting extension of this work is to admit algorithms which include conditional branching.

2.5 Algorithm Marked Graph

An algorithm marked graph (AMG) is a marked graph which represents a specific algorithm decomposition and is identical in topology to the corresponding algorithm directed graph. The AMG represents the first application of the Petri net structure to the development of the ATAMM model. The construction rules and symbols are the same as the ADG except that the edges are marked with tokens to represent the availability of data. That is, edge (i,j) is marked with a token if an output from primitive operator is is available as an input to primitive operator j. The presence



PETRI NET REPRESENTATION



MARKED GRAPH REPRESENTATION

Figure 2.3 Marked ordinary Petri net.

of a token on an edge is indicated by a solid dot placed on the edge. The vertices correspond to transitions which may fire after being enabled by the availability of all input data tokens.

The decomposed state equation represented in Fig. 2.2 is used to illustrate the AMG. The example AMG is shown in Fig. 2.4. It should be noted that the initial conditions for the recursion are represented by tokens on the loop edges.

The algorithm marked graph is a useful tool for representing decomposed algorithms and for displaying data flow within an algorithm. However, the AMG does not display procedures that a computing structure must manifest in order to perform the computing task. In addition, the issues of control, time performance, and resource management are not apparent in this graph.

2.6 Computational Environment

The computational environment for the ATAMM model is assumed to be a multiprocessor data-flow computer architecture. The data flow aspect is motivated by the algorithm directed graph which defines the data flow required to execute the algorithm.

The architecture is assumed to consist of R identical processors or functional units (FUNs) where R has a value in the range of two to twenty. This upper bound is suggested for practical reasons due to the large grained aspect of the algorithm decomposition and the need to maintain communication times small relative to process times. Therefore, little or no contention for access to communication paths occurs between functional units.

Each FUN is a processor having local memory for program storage and temporary input and output data containers. Each FUN has the capability to execute any algorithm primitive operation. The FUNs share a common global

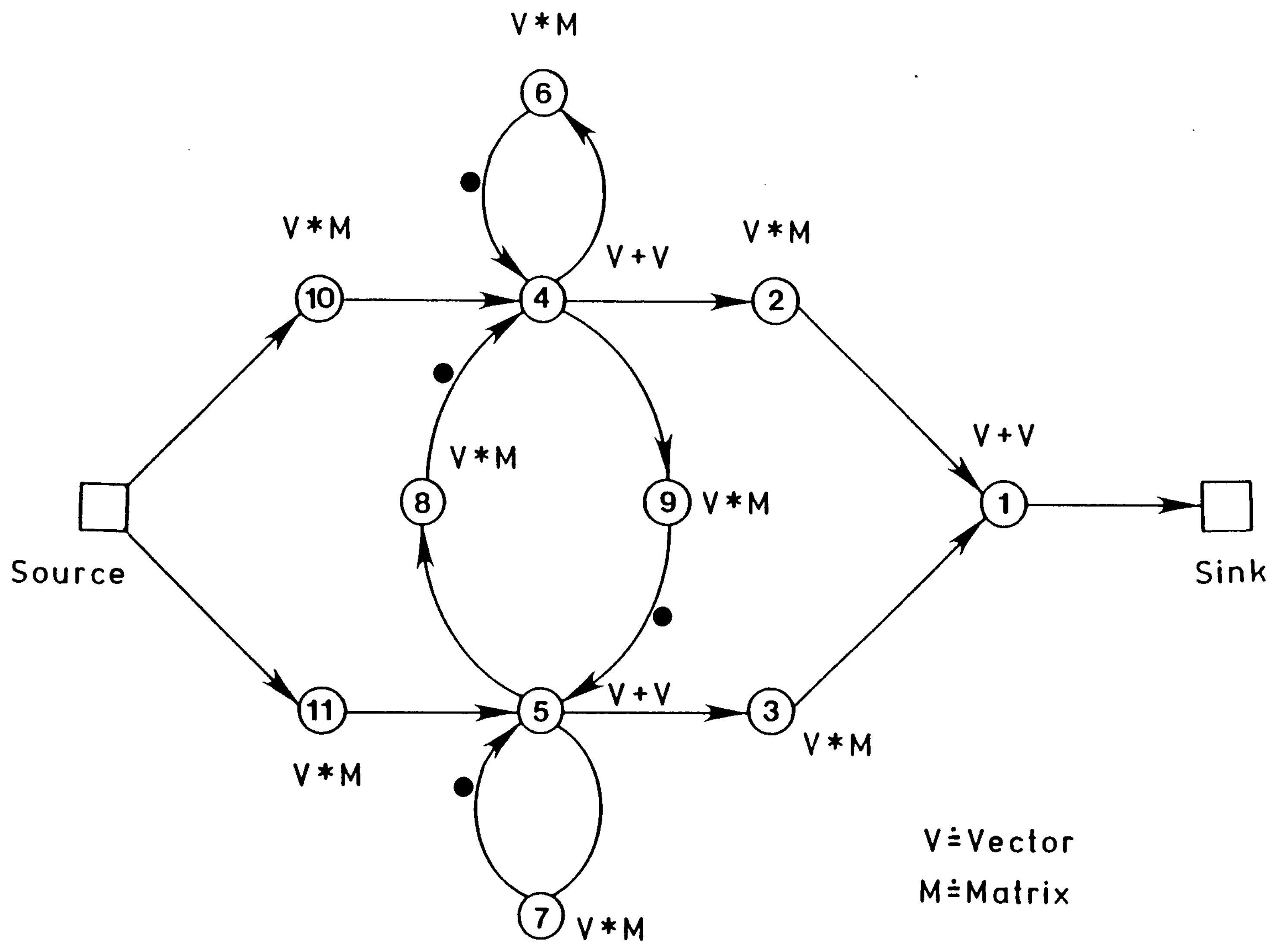


Figure 2.4 Algorithm Marked Graph-decomposed state equation.

memory (GLM) which may be either centralized or distributed. The coordination of FUNs in relation to data and control flow is directed by the graph manager (GRM). The GRM itself may be centralized or distributed.

Output created by the completion of a primitive operation is placed into global memory only after the output data containers have been emptied. That is, outputs must be consumed as inputs to successor primitive operations before allowing new data to fill the output locations.

Assignment of a functional unit to a specific algorithm primitive operation is made by the GRM only when all inputs required by the operation are available in global memory and a functional unit is available. A feature that will be developed later is that assignment of funcitonal units to primitive operations is performed continuously during run-time execution of the algorithm. This contrasts with static resource assignment procedures in which primitive operations are assigned to specific functional units during program development, and with dynamic resource assignment procedures in which primitive operations are assigned to specific funcitonal units during program compilation. One of many possible computer architectures consistent with these assumptions is shown in Fig. 2.5. Specific features of an experimental prototype architecture are described in Chapter 4.

2.6 Node Marked Graph

Algorithm requirements and the computing environment may now be integrated into a comprehensive Petri net model to complete the ATAMM model. The model consists of a Petri net marked graph called the computational marked graph (CMG). The CMG displays the data flow and control flow required to implement a decomposed algorithm in a multiprocessor data flow

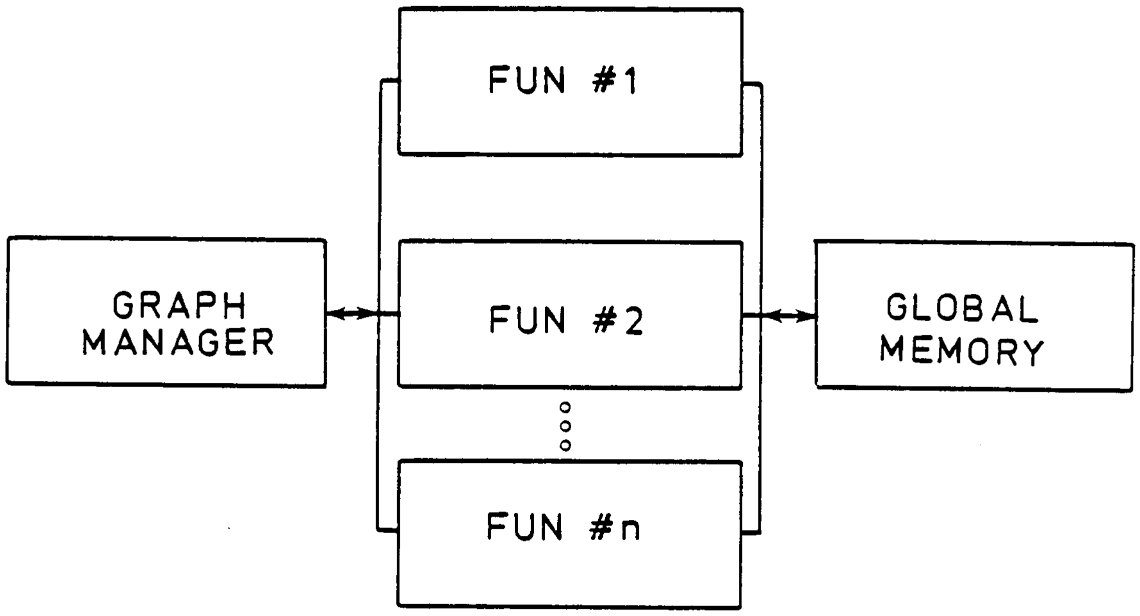


Figure 2.5 Candidate architecture.

computer architecture. Before defining this model, it is helpful to define an intermediate graph called the node marked graph (NMG).
A node marked graph is a Petri net representation of the computing behavior of a primitive operation by a functional unit. Three primary activities, reading of input data from global memory, processing of input data to compute an output, and writing of output data to global memory, are represented as transitions (vertices) in the NMG. Data and control flow paths are represented as places (edges), and the presence of signals is notated by tokens marking appropriate edges. The conditions for firing the process and write transitions of the NMG are as defined for a general Petri net, while the read transiiton has one additional condition for firing. In addition to having a token present on each incoming signal edge, a functional unit must be available for assignment to the primitive operation before the read node can fire. Once assigned, the functional unit is used to implement the read, process, and write operations before being returned to a queue of available FUNs.

Two different node marked graphs are defined to represent two different strategies. The first mode, called the three node model, requires that control signals indicating that empty data containers are available to receive new output are input edges to the write transition. Therefore, initiation of the primitive operation depends only on availablity of input data and availability of a functional unit. This strategy allows a primitive operation to commence without first having an output container available in global memory. This model is shown in Fig. 2.6. The second model, called the one node model, requires control signals indicating that empty data containers are available to recieve new output as input edges to the read transiton. Therefore, initiation of the primitive operation requires not only the availability of input data and a functional unit, but also the availability of empty output data containers in global memory. This model is shown in Fig. 2.7. It is noted that the three node model is used in for most of the examples of this report. However, it has been recently observed that the one node model has the inherent property of maintaining deadlock free CMG graphs. Thus, it is anticipated that the one node NMG will become prominent in future development and application of the ATAMM model.

2.7 Computational Marked Graph

A computational marked graph (CMG) is constructed from an algorithm marked graph according to the following rules.

1. Source and sink nodes in the algorithm graph are represented by source and sink nodes in the CMG.
2. Nodes corresponding to primitive operations in the algorithm graph are represented by NMGs in the CMG.
3. Edges in the algorithm graph are represented by edge pairs, one forward directed for data flow and one backward directed for control flow, in the CMG.

The play of the CMG proceeds according to the following graph rules.

1) A node is enabled when all incoming edges are marked with a token. An enabled node fires by encumbering one token from each incoming edge, delaying for some specified transition time, and then depositing one token on each outgoing edge.
2) A source node and a sink node fire when enabled without regard for the availability of a FUN.
3) A primitive operation is initiated when the read node of an NMG is enabled and a FUN is available for assignment to the NMG and thus fires the read node. A FUN remains assigned to an NMG until

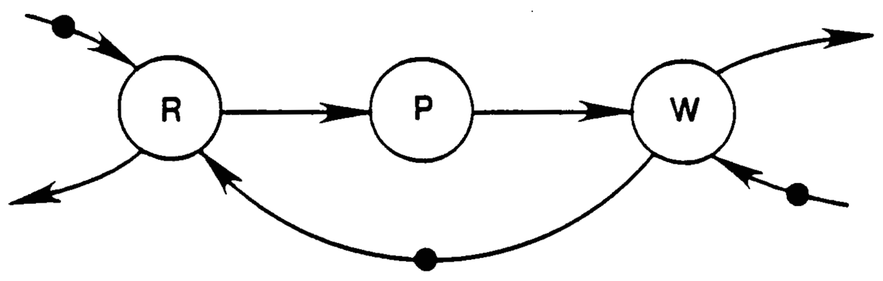


Figure 2.6 Node marked graph 3-node model.

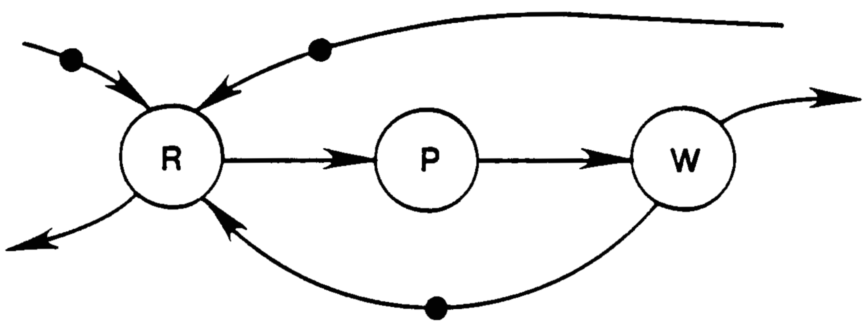


Figure 2.7 Node marked graph one-node model.

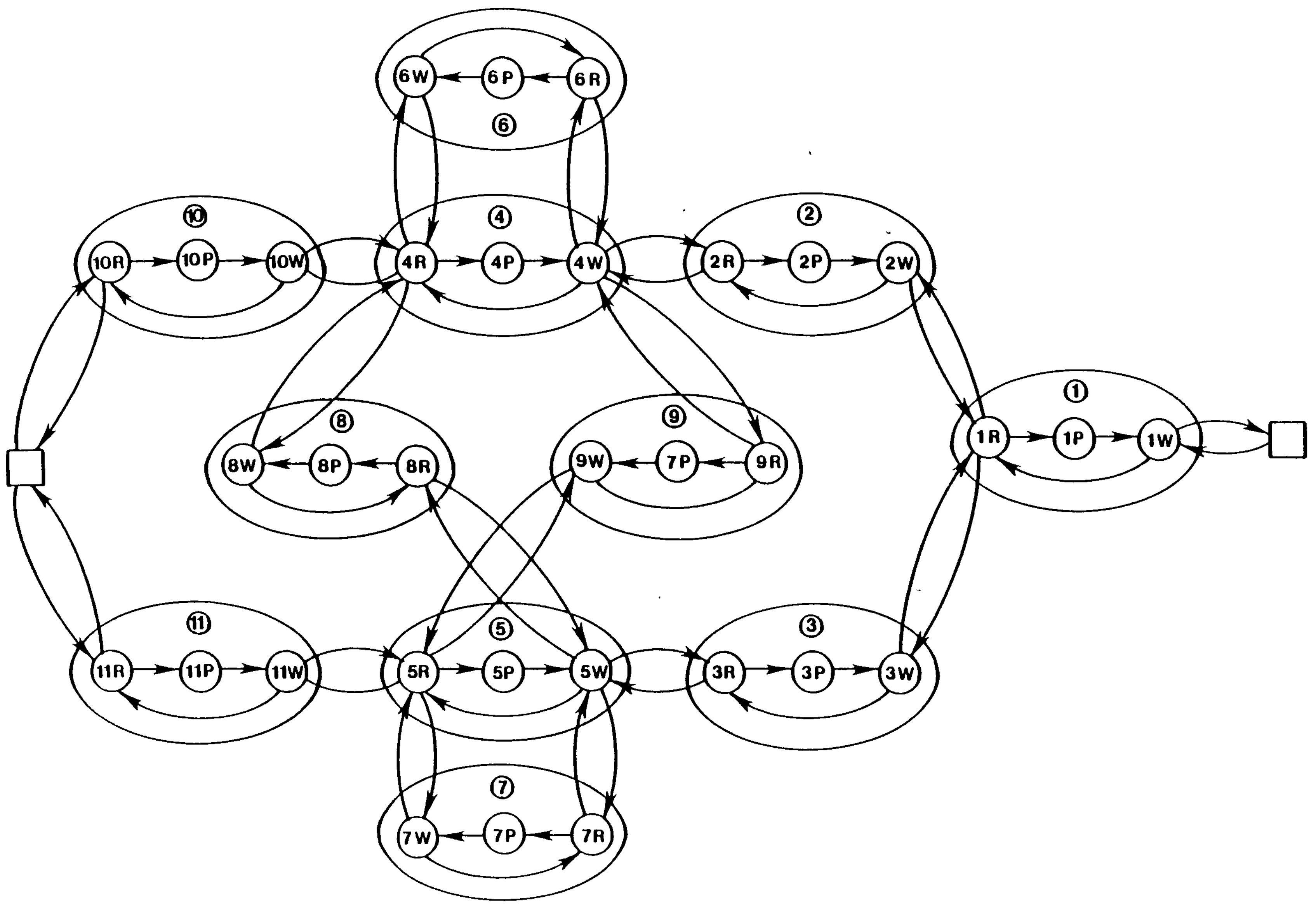


Figure 2.8 Computational Marked Graph of Decomposed State Equation.

completion of the firing of the write node of the NMG. Supervision of this logical assugnment of the fun is managed by the GRM.

In order to illustrate the construction of a computational marked graph, the CMG corresponding to the algorithm graph of Fig. 2.2 is shown in Fig. 2.8. The three node NMG is used in this CMG for convenience of presentation. The computational marked graph is important because it clearly displays the data and control flow which must occur in any hardware implementation of the model process, and because it provides a hardware independent context in which to evaluate process performance. Thus, the CMG becomes the theoretical vehicle for presenting the ATAMM model.

The ATAMM model consists of all the modeling steps which lead to the integration of the algorithm data flow with the data flow architecture. A pictorial description of the ATAMM model is shown in Fig. 2.9.

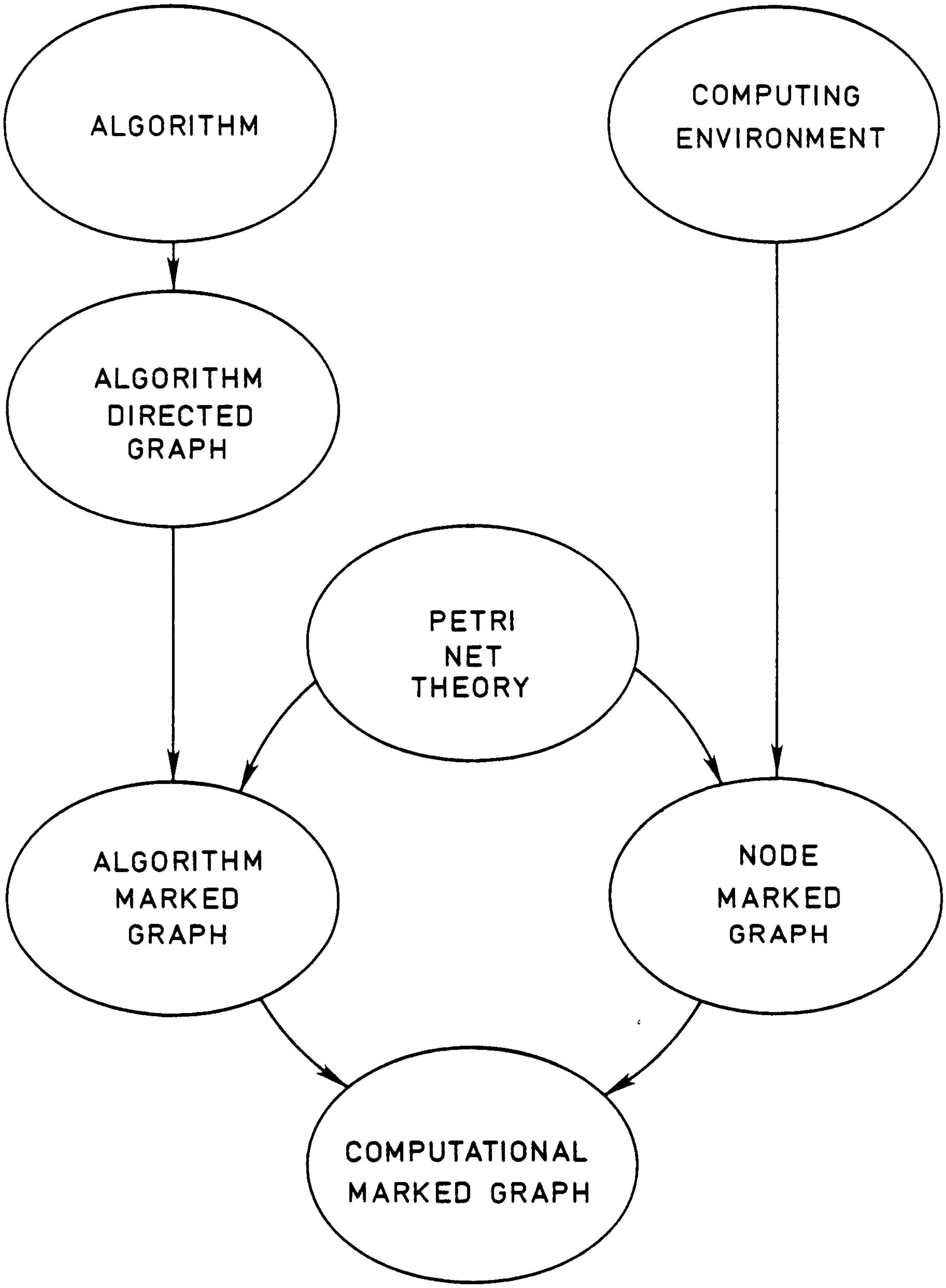


Figure 2.9 Relational diagram of ATAMM model.

CHAPTER 3

3.0 GRAPH MODEL OPERATING CHARACTERISTICS

3.1 Introduction

An important component of the ATAMM model, as previously described, is the CMG algorithm/architecture behavioral model. This model is important because it provides a hardware independent context in which to investigate the relative merits of different algorithm decompositions and different implementation strategies. In this chapter, properties of the CMG Petri net model are studied analytically to determine graph operating characteristics and to develop bounds on computational performance. Many of the properties presented here result from restricting the algorithms under consideration to be decision-free so that the graph models are marked graphs. An important extension of this work is to conduct a similar study admitting algorithms containing decision points (branching).

In Section 3.2, a state variable description is developed for the computational marked graph (CMG). This formulation expresses the next graph marking as a function of the present marking and a vector which indicates which transition is to be fired. Graph operating characteristics are developed analytically in Section 3.3. Among the properties considered are reachability, liveness and safeness. Then, in Section 3.4, performance bounds are investigated. Upper and lower bounds for computational time are established.

3.2 State Equation Description

In this section, a state equation formulation for computing the marking vector of a marked graph is presented. This development is easily extended to general Petri nets. Let G be a marked graph consisting of m places and n transitions. The m-vector M_k is the marking vector for G resulting from the firing of some sequence of k transitions. The following two definitions are necessary for the state equation formulation.

Complete incidence Matrix. The complete incidence matrix for a marked graph G is the (nxm) matrix $A = [a_{ij}]$ having rows corresponding to transitions and columns corresponding to places, and where

$$a_{ij} = \begin{cases} +1(-1) & \text{if place } j \text{ is incident at transition } i \text{ and directed out of (into) the transition} \\ 0 & \text{if place } j \text{ is not indicent at transition } i. \end{cases}$$

Elementary Firing Vector. An elementary firing vector u_k is an n-vector having all zero entries except for the ith component which is 1 denoting that transition i is the kth transition to fire in some transition firing sequence.

To gain insight to the state equation formulation, it is helpful to consider the firing of transition k. If $a_{ki} = -1$, place i is an input to transition k. Therefore, transition k is enabled if $M(i) = 1$ for each place i for which $a_{ki} = -1$. When transition k fires, one token is removed from each place i for which $a_{ki} = -1$, and one token is added to each place j for which $a_{kj} = +1$. These observations lead to the following state equation description for the marking vector of a marked graph.

<u>State Equation Description</u>. For a marked graph G with present marking M_{k-1} and elementary firing vector u_k, the next marking vector is given by

$$M_k = M_{k-1} + A^T u_k.$$

where T denotes transpose.

The state equation formulation can be used to express the graph marking resulting from the application of sequences of elementary firing vectors. This is done in the next two definitions.

Firing Count Vector. Let $(u_1,u_2,\ldots.,u_d)$ be a sequence of elementary firing vectors taking a marked graph G from an initial marking M_0 to a destination marking M_d. The firing count vector x_d for this elementary firing vector sequence is defined by

$$x_d = \Sigma\, u_k \quad , \quad k=1,2\ldots d$$

State Transitions. Consider a sequence of elementary firing vectors $(u_1,u_2,\ldots,u_d)$ taking marked graph G from marking M_0 to M_d. Then

$$M_1 = M_0 + A^T u_1$$
$$M_2 = M_1 + A^T u_2$$
$$\bullet$$
$$\bullet$$
$$M_d = M_{d-1} + A^T u_d$$

and repeated substitution yields the state transition equation

$$M_d = M_0 + A^T x_d$$

where x_d is the firing count vector.

This state equation description for the marking vector of a marked graph is used in the next section to investigate properties of the computational marked graph.

3.3 Marked Graph Properties

Several graph theoretic properties of the computational marked graph are developed in this section. The properties investigated include reachability, liveness, and safeness. This area of investigation should be viewed as a preliminary study only; additional properties are likely to be developed as more experience is gained with the computational marked graph model. It will also be important to attempt to extend these or similar properties to the more general Petri net model of concurrent processes.

The first graph property to be considered is reachability. We begin with a definition of this property.

Reachability. A marking M_d is reachable from a marking M_0 if there exists a sequence of elementary firing vectors that transforms M_0 to M_d. Before stating conditions for reachability, it is necessary to define a new matrix quantity called a fundamental circuit matrix. For simplicity, it is assumed that G is connected. That is, a path exists between every pair of vertices in G.

Fundamental Circuits. Let T be a tree of G. Then the set of (m-n+1) fundamental (or f) circuits, each uniquely formed by appending one cotree edge to the tree, are called the fundamental circuits of G for tree T.

Fundamental Circuit Matrix. The fundamental circuit matrix of a graph G for tree T is the (m-n+1) x (m) matrix $B_f = [b_{ij}]$ having rows corresponding to places, and where

$$bij = \begin{cases} +1(-1) & \text{if place j is contained in f-circuit} \\ & \text{i and the edge and the edge and circuit} \\ & \text{directions agree (disagree)} \\ 0 & \text{if place j in not contained in f-circuit i.} \end{cases}$$

The following property gives necessary and sufficient conditions for a marking M_d to be reachable from an initial marking M_0.

Property 1 (Reachability). In a computational marked graph G, a marking M_d is reachable from an initial marking M_0 if and only if $B_f M_d = B_f M_0$ where B_f is a fundamental circuit matrix for G.

Proof of Necessity. Suppose M_d is reachable from M_0. Then from the state transition equation, there exists a firing count vector x_d and incidence matrix A, such that

$$M_d - M_0 = \Delta M = A^T x_d .$$

It is known from linear algebra that this equation has a solution for x_d if and only if Δ_M is orthogonal to every solution of the transposed homogenous equation $A_y = 0$ (y is mxl vector). By the orthogonality of A and B_f, it is apparent that all possible solutions for y are contained in the space spanned by the columns of B_f^T. Thus $B_f \Delta M_d = 0$ and the property follows.

Proof of Sufficiency. Suppose $B_f M_d = B_f M_0$. Then $B_f \Delta_M = 0$ and it follows by the above argument that there exists a vector x_d satisfying the equation

$$M_d - M_0 = \Delta M = A^T x_d \quad .$$

It is known that x_d is an executable firing count vector if and only if G contains no token free directed circuits [4]. Since a CMG contains no token free directed circuits, x_d is executable so that M_d is reachable from M_o. This completes the proof.

The second graph property to be considered is liveness. Also presented is a discussion of another closely related property called consistency.

Liveness. A marked graph G is live for marking M_o if, for all markings reachable from M_o, it is possible to fire any transition of G by progressing through some firing sequence.

The following property gives necessary and sufficient conditions for a graph to be live.

Property 2 (Liveness). A marked graph G is live for marking M if and only if G has no token free directed circuits in marking M.

A proof of this property is given in [4] and is not repeated here. Since by the construction rules of the CMG there are no token-free directed circuits, it follows that the CMG is live.

A very important property which is closely related to liveness is a property called consistency. It is shown that the CMG is consistent.

Consistency. A marked graph G is consistent if there exists a marking M_o and a firing sequence Σ from M_o back to M_o such that every transition occurs at least once in Σ.

Property 3 (Consistency). A connected CMG is consistent. In addition, each transition of G occurs in Σ an equal number of times.

Proof. The incidence matrix for a marked graph G is an (n x m) matrix A. If G is connected, then it is shown [9] that the rank of A is n-1, and thus the null space of A^T has dimension one. It is observed that each row of A^T has dimension one. It is observed that each row of A^T has one (1), one (-1) and all remaining terms of (0)s; and, in terms of the columns, C_j, of A^T

$$\Sigma C_j = 0 \quad j=1,2,\ldots n_N.$$

It is readily shown that the homogeneous equations

$$\Sigma k_j C_j = 0 \quad j = 1,2,\ldots,n$$

has only one non zero solution for the k_j's. That is, $k_1=k_2=\cdots=k_n=1\cdot K$, where K is an arbitrary constant. The homogenous solution for the state equation

$$A^T x_d = \Delta M$$

where ΔM is zero, directly follows. That is, the firing vector, xd, has elements all equal to an arbitrary constant, K, or $x_d = [K,K,\ldots.K]^T$. Because x_d is a firing vector, K is restricted to non negative integers. By further restricting K to be non zero and eliminating the null firing vector, then $A^T x = 0$ implies that there exists a non trivial firing sequence such that $M_d = M_o$, and thus G is consistent. This completes the proof.

The consistency property is important because it shows that the CMG operates periodically as long as inputs are available. During each period, each transition of the CMG fires an equal number of times.

The third and final graph property considered in this section is safeness. This property is first defined, and then it is shown that the CMG is safe.

Boundedness. A marked graph G is K-bounded for marking M_0 if, for all markings reachable from M_0, no place contains more than K tokens.

Safeness. A marked graph G is safe for marking M_0 if it is 1-bounded for M_0.

Property 4 (Safeness). A live marking M_0 of a marked graph G is safe if every place of G belongs to a directed circuit with token count one.

Proof. Let $B_d = [b_{ij}]$ be the directed circuit matrix for G. Then the rows of G correspond to directed circuits of G, the columns correspond to directed circuits of G, and the entries of B_f are given by

$$b_{ij} = \begin{cases} +1 & \text{if place } j \text{ is in directed circuit } i \\ 0 & \text{if place } j \text{ is not in directed circuit } i \end{cases}$$

Consider the state transition equation for G. Since B_d is orthogonal to the incidence matrix A, it follows that for any marking M_d reachable from M_0,

$$B_d M_d = B_d M_0 + B_d A^T x_d = B_d M_0 \quad .$$

For any M, the pth component of vector $B_d M$ is equal to the number of tokens contained in directed circuit p. It follows that the number of tokens contained in a directed circuit is invariant. Therefore, if every place belongs to a directed circuit with token count one for marking M_0, it follows that every place belongs to a directed circuit with token count one for all markings reachable from M_0. It follows that no place of G contains more than one token. This completes the proof.

In summary, it has been shown that the computational marked graph is live, consistent, and safe. In addition, necessary and sufficient conditions for a marking M_d to be reachable from an initial marking M_0 has been given. It has also been established that when a CMG operates periodically, each transition fires an equal number of times during a period, and that the number of tokens contained in any directed circuit is invariant under transition firings.

3.4 Analytical Bounds on Computational Performance

In this section, bounds on the computational performance of the computational marked graph are developed. Included are formulations of an upper bound on the completion time for the performance of an algorithm, and a lower bound for the completion time of the performance of an algorithm. An objective of future research is to develop tighter bounds on operation performance as a function of the number of functional units available.

The time required to complete a computational task implemented according to the rules of the computational marked graph has been shown to be a function of the number of functional units available to carryout primitive operations, the priority schedule with which functional units are assigned to primitive operations, and the node marked graph strategy which is employed. At this time, it is not clearly understood how each of these operating parameters effects the computational time. However, as shown in Fig. 3.1 computational time is maximum when a single functional unit is used, and a minimum computational time is realized when the number of functional units is equal to the number of primitive operations, n. Properties of these bounds, identified as T_{max} and T_{min}, are presented in this section. Future research will address determining $N=N_{max}$ which is the minimum N_{max} required for optimal performance.

T_{max} is an upper bound on the time required to complete a computation (input to output). T_{max} is the actual computational time when only a single

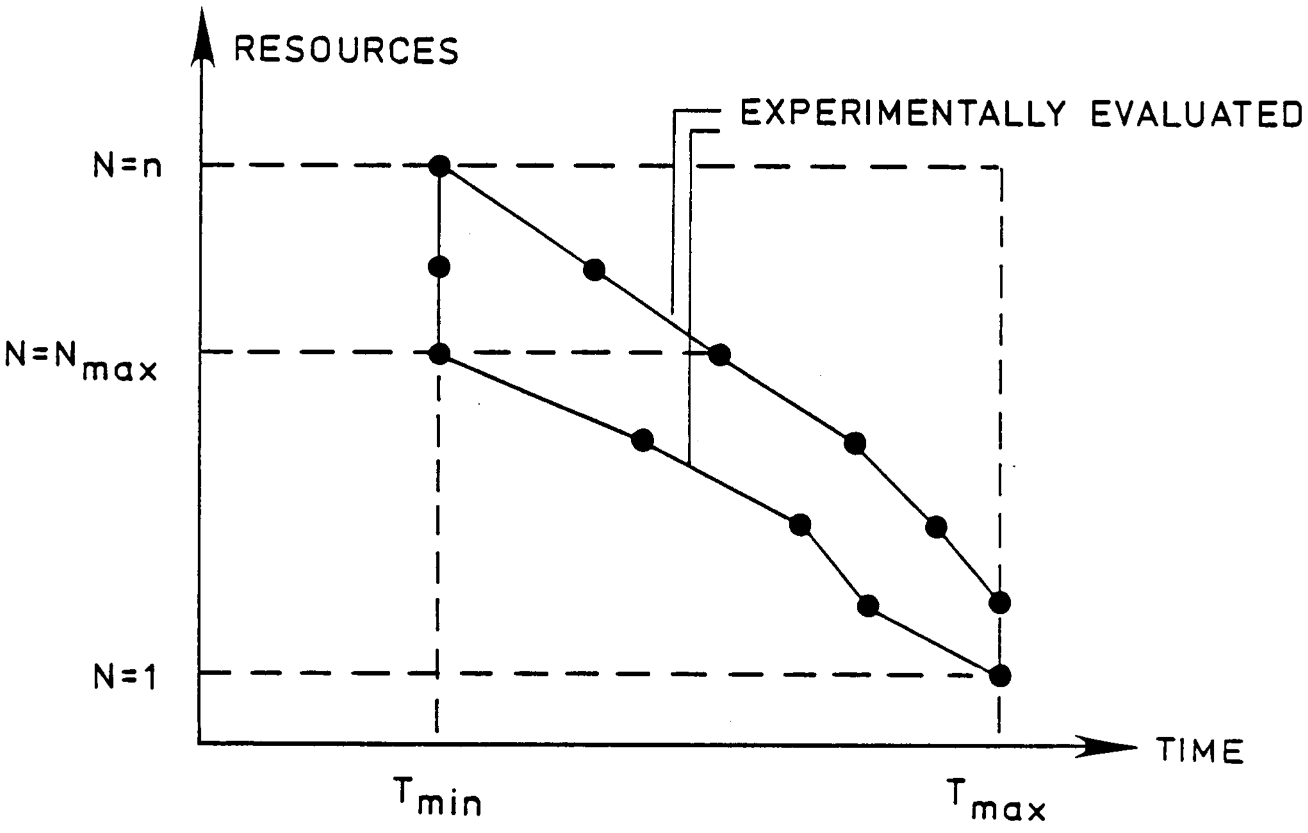


Figure 3.1 Performance Bounds.

functional unit is available. The following are properties of the operating bound.

1. T_{max} is an upper bound for all admissable operating conditions. Task performance is always completed within this time.

2. T_{max} is independent of node marked graph strategy. The same maximum time is required in the three-node model and the one-node model.

3. T_{max} is independent of priority schedule used to assign functional units to primitive operations.

4. $T_{max} = \Sigma\ T_k$, k=1,2,..,n

where T_k is the delay time associated with transition k.

T_{min} is a lower bound on the time required to complete a computation. The following are properties of this operating bound.

1. T_{min} is a lower bound for all admissable operating conditions. Task performance is never completed in a shorter period of time.

2. T_{min} is dependent on node marked graph operating strategy. It is anticipated that T_{min} (1-node model). However, this property requires further research for more specific assessment.

3. $T_{min} = \text{Max}\ \{T(C_i)/M_0(C_i)\}$

where $T(C_i)$ is the sum of transitions delays in directed circuit C_i, $M_0(C_i)$ is the number of tokens contained in directed circuit C_i, and the maximum is taken over all directed circuits.

In the next chapter, a prototype hardware implementation which operates according to the CMG rules is presented. The prototype is used to validate the CMG model, and as an experimental testbed to investigate computational performance.

CHAPTER FOUR

4.0 PROTOTYPE ARCHITECTURE

4.1 Introduction

A description of a prototype system which was used to implement the ATAMM Model is discussed in this chapter. An overview of the system is presented in Section 4.2. A description of the prototype graph manager is presented in Section 4.3. Discussion of the prototype functional unit and global memory are presented in Sections 4.4 and 4.5, respectively. A discussion of the relationship between design requirements and graph validation is discussed in Section 4.6.

4.2 Prototype Overview

The prototype realization is based on computing environment assumptions for the ATAMM model as described in Section 2.6. These assumptions are reiterated below.

1. The computing structure contains N functional units (FUN). FUNs are processors with local memory for program storage and temporary input and output data containers. The stored programs include all primitives to be executed.
2. The computing structure contains a global data memory accessible to all FUNs. Although the GLM could be distributed, the GLM was chosen to be centralized for implementation convenience. The input data for each primitive operation are found in fixed data containers in the global data memory.

3. A primitive operation is assigned for execution on a functional unit only when all inputs required by the operation are available in data memory, and a FUN is available to carry out the primitive operation.
4. Output created by the completion of a primitive operation may be placed into global memory only after the output data containers have been emptied. That is, outputs must be consumed as inputs to successor primitive operations before allowing new data to fill the output locations.

A prototype architecture, based upon the above requirements, has been implemented to provide hardware validation of the ATAMM model rules. The NMG that is used is the three node model. The prototype is not unique and is only one of several candidates which could have been used to perform the concurrent operations. The resulting structure is a data-flow architecture which is a natural consequence of meeting the requirements of the ATAMM model.

The hardware configuration of the prototype is shown in Fig. 4.1. A primary motivation for the particular design was the availability of hardware. The hardware used to implement the system consists of S-100 crates, each having an Intel 8088 CPU card, multiple serial I/O channels and 32K memory. An IBM PC/XT is used to host the system and to download algorithm graph descriptions to the system. A working prototype of the system has been developed with three FUNs employing serial communications in lieu of bus-level communications.

4.3 Prototype Graph Manager

The purpose of the graph manager (GRM) is to facilitate the assignment

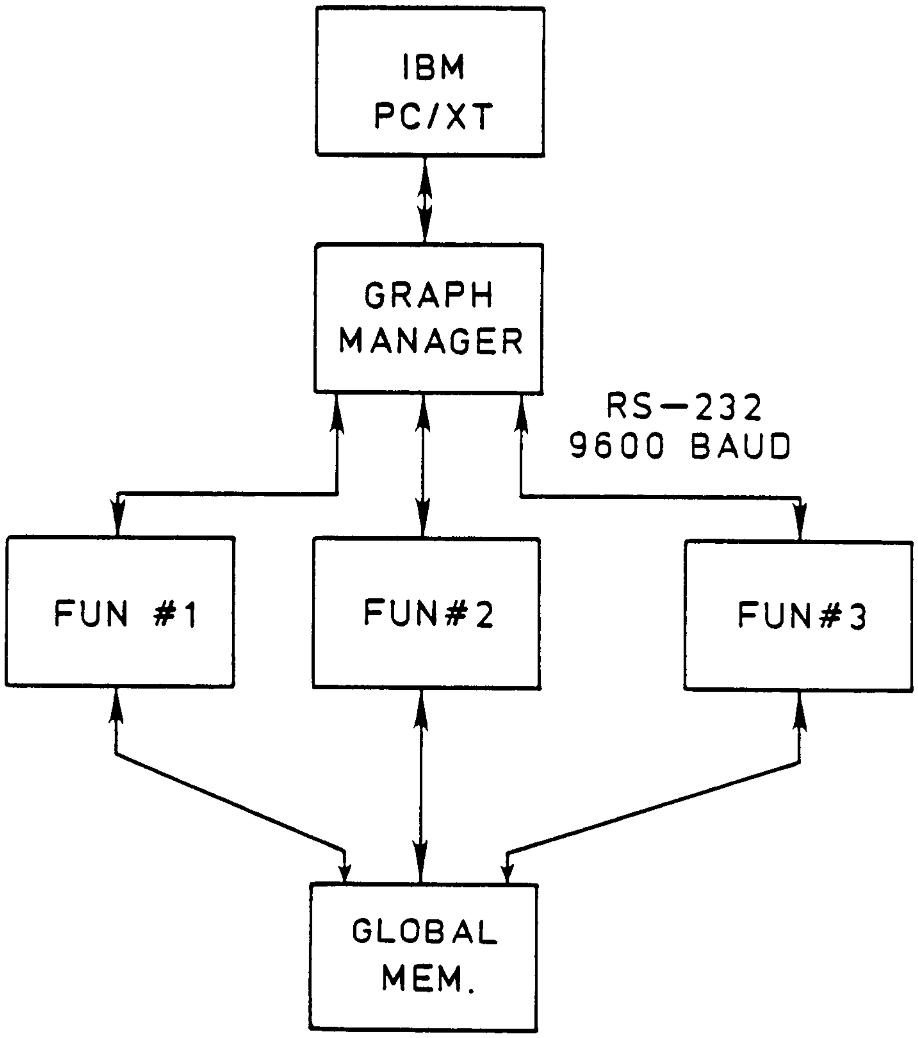


Figure 4.1 Experimental Protype Block Diagram.

of FUNs to the various algorithm graph node operations in relation to the advancement of tokens and transition firing in the CMG for the particular algorithm being executed. The NMG characteristics for each node are maintained by the GRM. The updating of token placement is facilitated by status information which is communicated to and/or from the active FUNs in their respective stages of computing activity. Node firings are actuated by the GRM when enabling information has been determined. Also, the GRM assigns the FUN which will execute the particular process. It is noted that the GRM manages the abstract properties of the graph through placement of tokens, but does not handle data, per se. The GRM only respond to the data flow conditions in the CMG and facilitates the firing of enabled transitions.

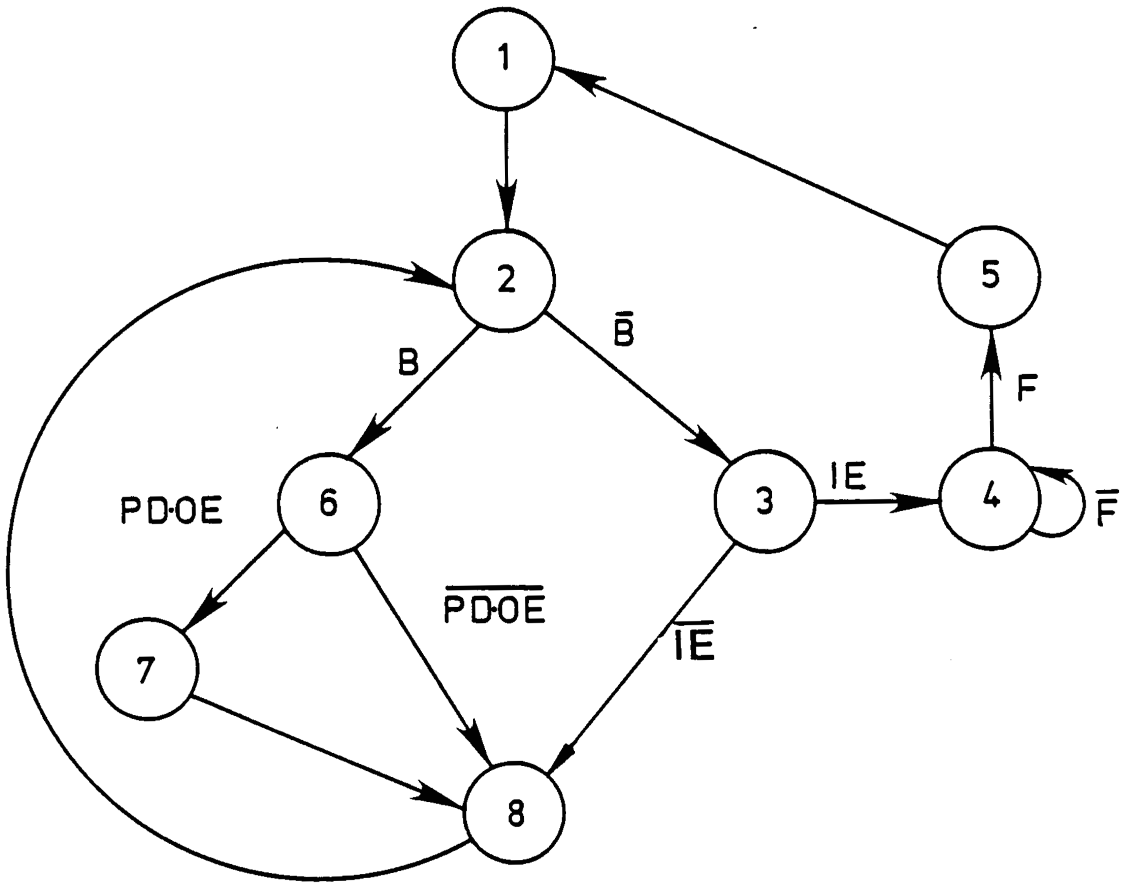
A simplified logical flow diagram for the prototype GRM operating system is shown in Fig. 4.2. Each node NMG attribute is scanned in a predetermined order which establishes a priority order among the nodes. For example, consider the following path in the control flow:

If a node is not busy, B, the inputs tokens are checked for enablement, IE. If a FUN is available, F, a FUN is assigned to the particular node computation, and the node pointer is reset.

The control flow is interrupted when new status conditions are being reported by the various FUNS. These status conditions are then recorded in the various node NMG attributes and control flow is resumed on the updated conditions.

4.4 Prototype Functional Unit

Each FUN must provide for communication handling as well as execution of the primitive. The FUN must communicate status conditions to the GRM



1. Reset node list pointer
2. Scan node busy condition
3. Check enabled inputs
4. Check available FUN's
5. Assign FUN to node
6. Check output empty
7. Send output labels
8. Increment node list pointer

B	- node busy
F	- FUN available
IE	- inputs enabled
OE	- outputs enabled
PD	- process done

Figure 4.2 Simplified Graph Manager control states.

in order that the GRM may track CMG token flow. The FUN must communicate with the GLM to facilitate the appropriate access of data containers. The GRM identifies an idle FUN to which is passed labels indicating primitive execution and data containers of input operands. Subsequent communication with the GRM provides output data containers labels (when they become available) and completion of the processing events. Thus the operating system of the FUN must manage graph attribute details with the GRM and actual data management with the GLM.

A control flow diagram of the prototype FUN operating system is shown in Fig. 4.3. The control state of the FUN operating system is denoted by "Z". The five control states are Wait (Z=1), Fetch Data (Z=2), Complete Task (Z=3), Wait for Empty Output Container, (Z=4), and Output Data (Z=5).

4.5 Prototype Global Memory

The GLM operating system responds to directives by the FUN to either fetch or write operands to the various data container labels in the global memory. A simplified operating system for the prototype GLM is shown in Figure 4.4. The operating system polls each FUN serial communication port to determine the request for transfer of data. If a transfer is requested, the type (input or output) and label is transfer. Then the appropriate data is transferred.

4.6 Synthesis Considerations

The synthesis procedure for a particular realization of the ATAMM based architecture must preserve the graph model requirements. Care must be exercised not to change the behavior of the ATAMM characteristics as represented by the NMG model. Thus communication/data exchange events built into the architecture must be modeled in accordance with graph expansion rules for

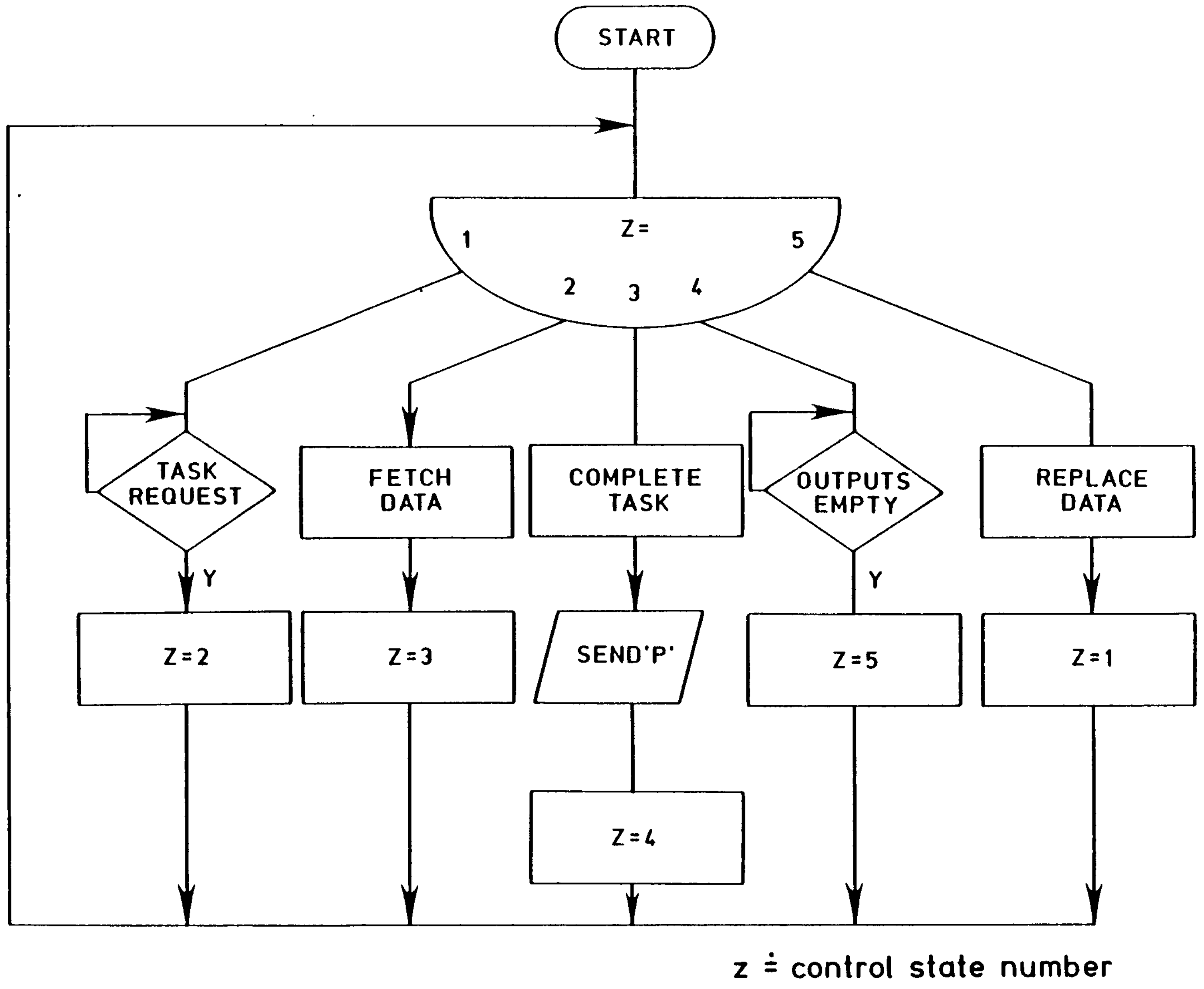


Figure 4.3 Functional Unit Control.

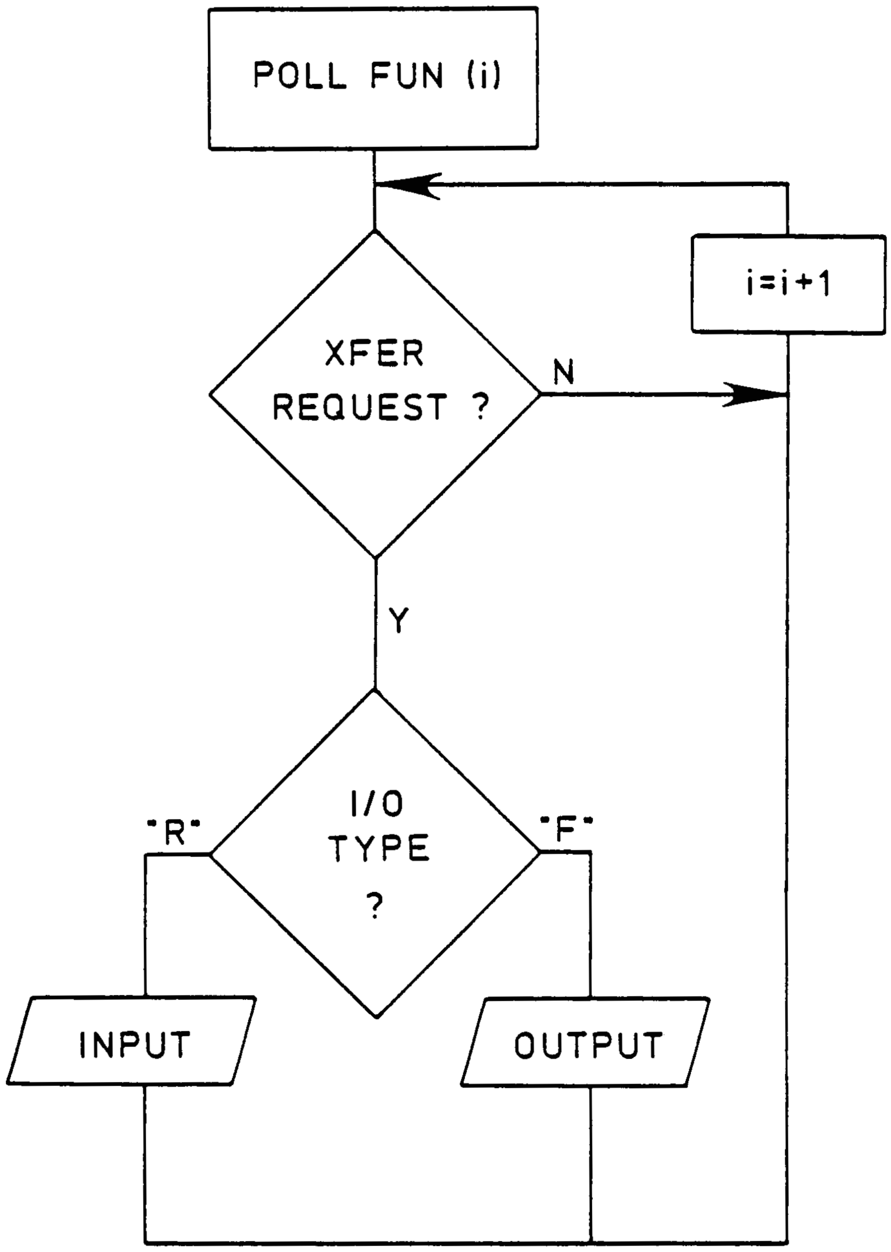
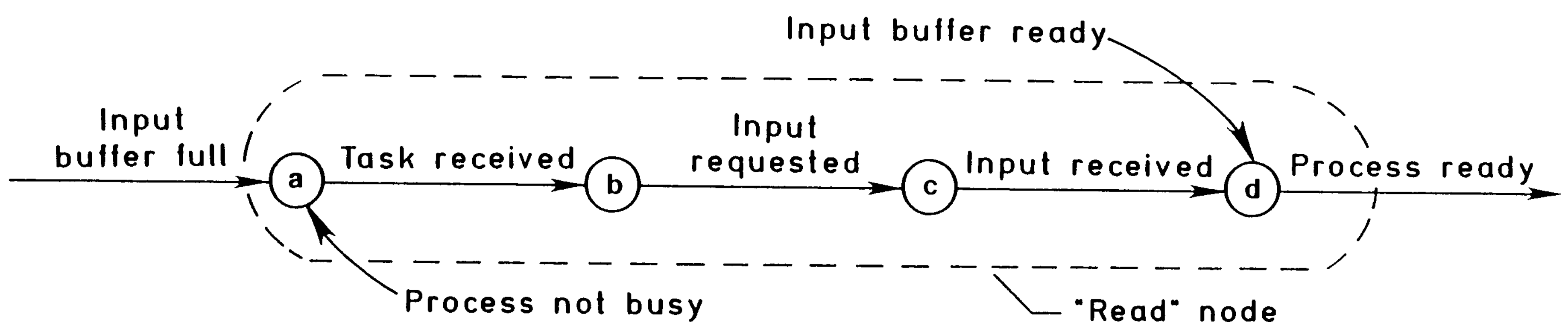


Figure 4.4 Global Memory Control Diagram.

marked graphs [6], [7]. Allowable additions to the NMG include additions of parallel edges, series edges and nodes, and Y-Δ transformations. The first level synthesis expansion of the read node of an NMG is conducted to exemplify the synthesis and modeling verification. The read node of the NMG requires that data be brought from the GLM to the assigned FUN. This transaction requires the data container labels (locations) and task assignment to be sent from the GRM to the FUN. The FUN in turn requests the data from the given locations in the GLM. When data has been placed in the FUN, the FUN must indicate to the GRM that the data container has been emptied so that the appropriate tokens can be placed in the graph description. The marked graph expansion of the read node is shown in Fig. 4.5.

The above synthesis process leads to the communication dialogue sequence shown in Fig. 4.6. The expanded three node NMG with the communication/data transactions and related handshaking is shown in Fig. 4.7. It should be noted that the topology of the graph reflects the physical layers in the architecture where the GRM activities occur on the top layer, the FUN activities occur on the middle layer, and the GLM activities occur on the bottom layer. The communication and requisite handshaking form links to the various layers, as should be expected.



a=Receive task assisgnment

b=Request input data

c=Receive input data

d=Acknowledge data reception

Figure 4.5 Expanded Read Node Marked Graph. (3-node model).

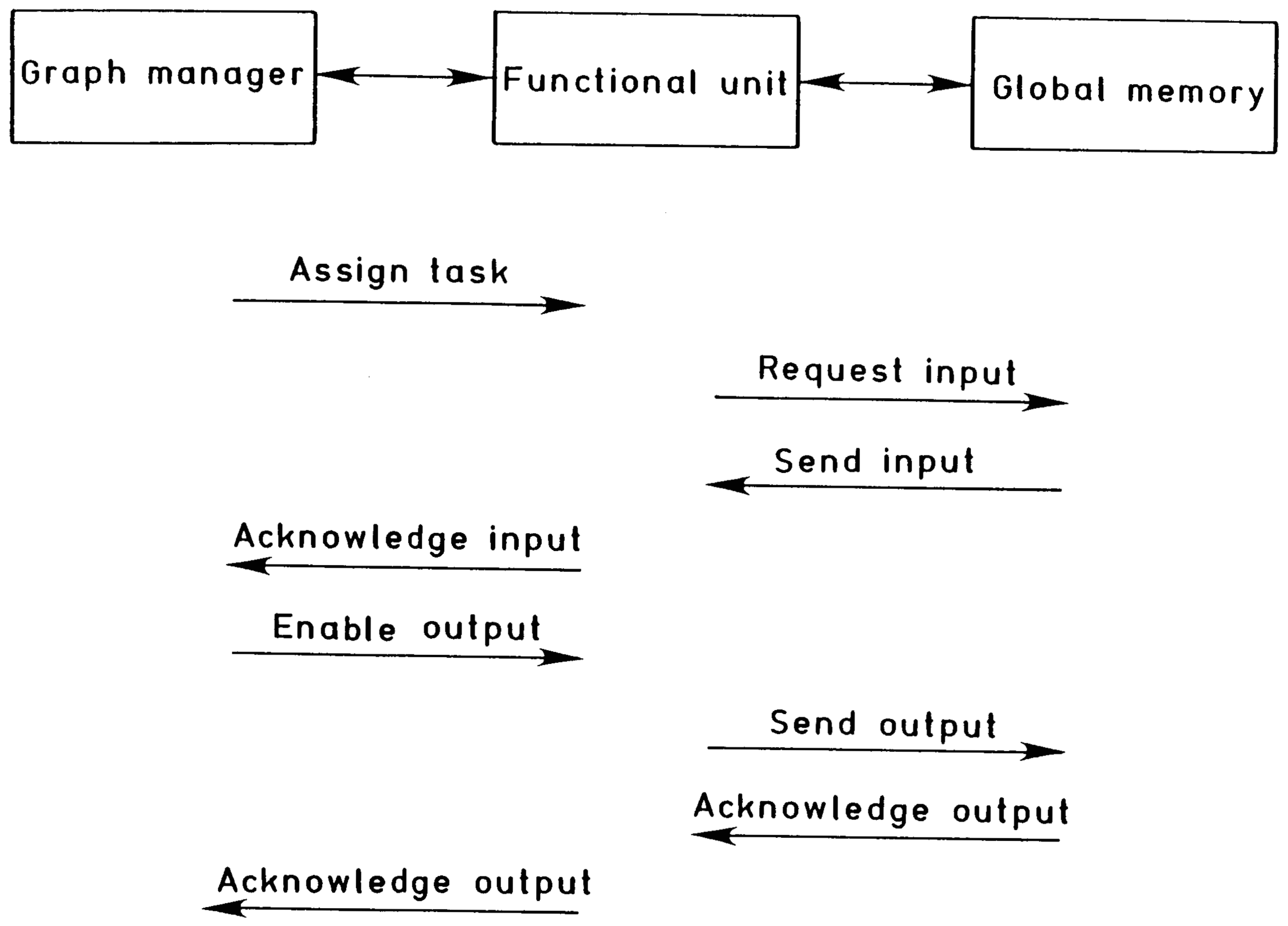


Figure 4.6 Prototype Communication Dialogue.

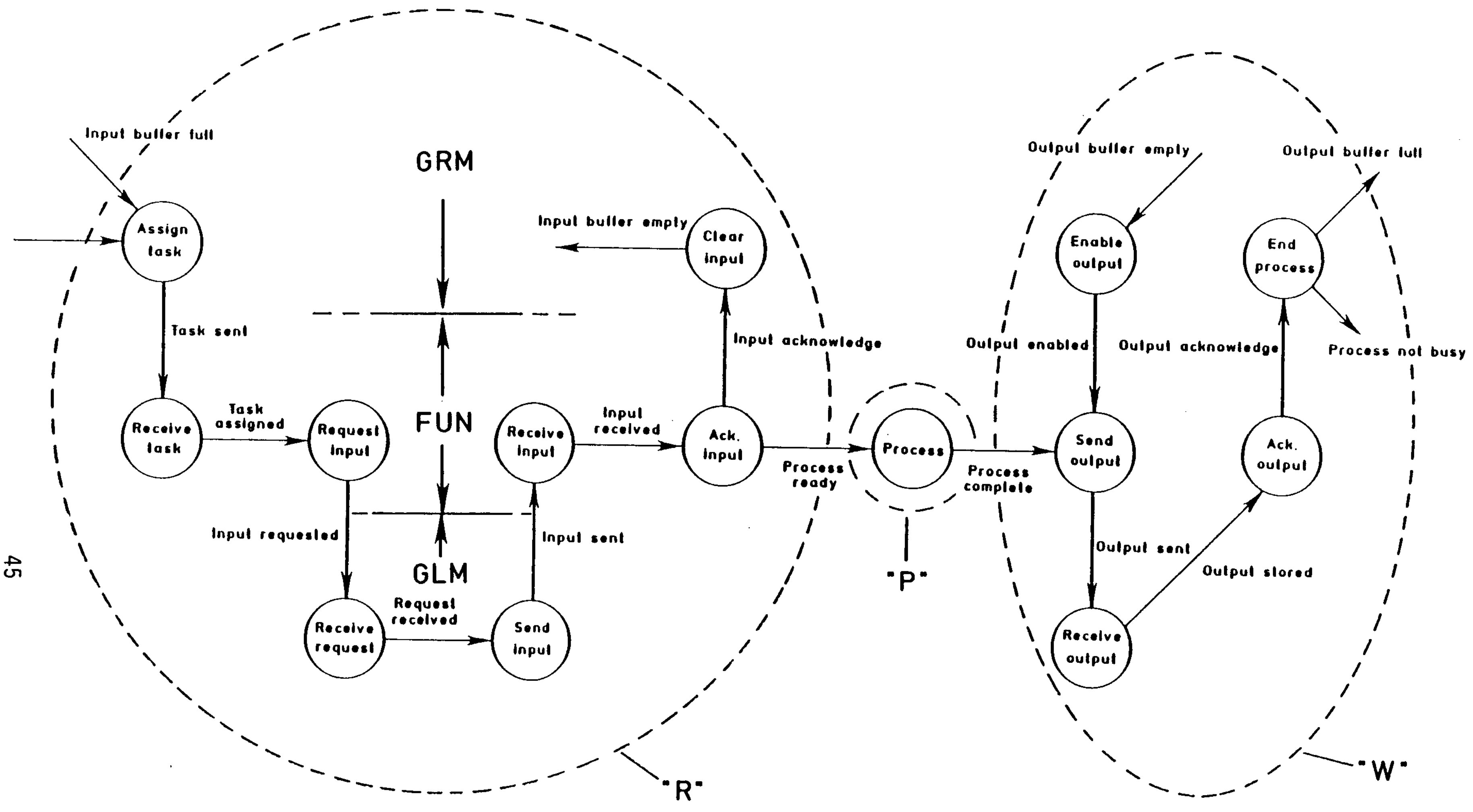


Figure 4.7 Expanded Node Marked Graph (NMG).

CHAPTER FIVE

5.0 EXPERIMENTAL EVALUATION

5.1 Introduction

Chapter five presents a preliminary evaluation of the prototype implementation described in Chapter Four. The evaluation is supported by the development of a diagnostic procedure which interacts with the GRM. The diagnostics are discussed in Section 5.2. An algorithm example is executed to illustrate both the behavior of the system and diagnostic attributes.

5.2 System Diagnostics

The evaluation of the prototype is important in order to determine if the system is behaving in accordance with the ATAMM model. Analysis is difficult due to the concurrent processing and communication events taking place. An appropriate diagnostic or analysis tool should make use of the properties of the Graph Manager in that all system events are known as a translation of the CMG token placement and node firings.

The Graph Manager has an internal real time clock which may be used to time mark each event. The events to be recorded include:

1. the assignment of a FUN to a particular node,
2. the acquisition of input data by the node being processed,
3. the completion of the node processing,
4. the full or empty condition of the data container labels,
5. the writing of the output data,

The format of each of the entries of the report contains the next items of information:

1. Time at which the event took place.
2. Node at which the event took place.
3. Type of event (any of the above).

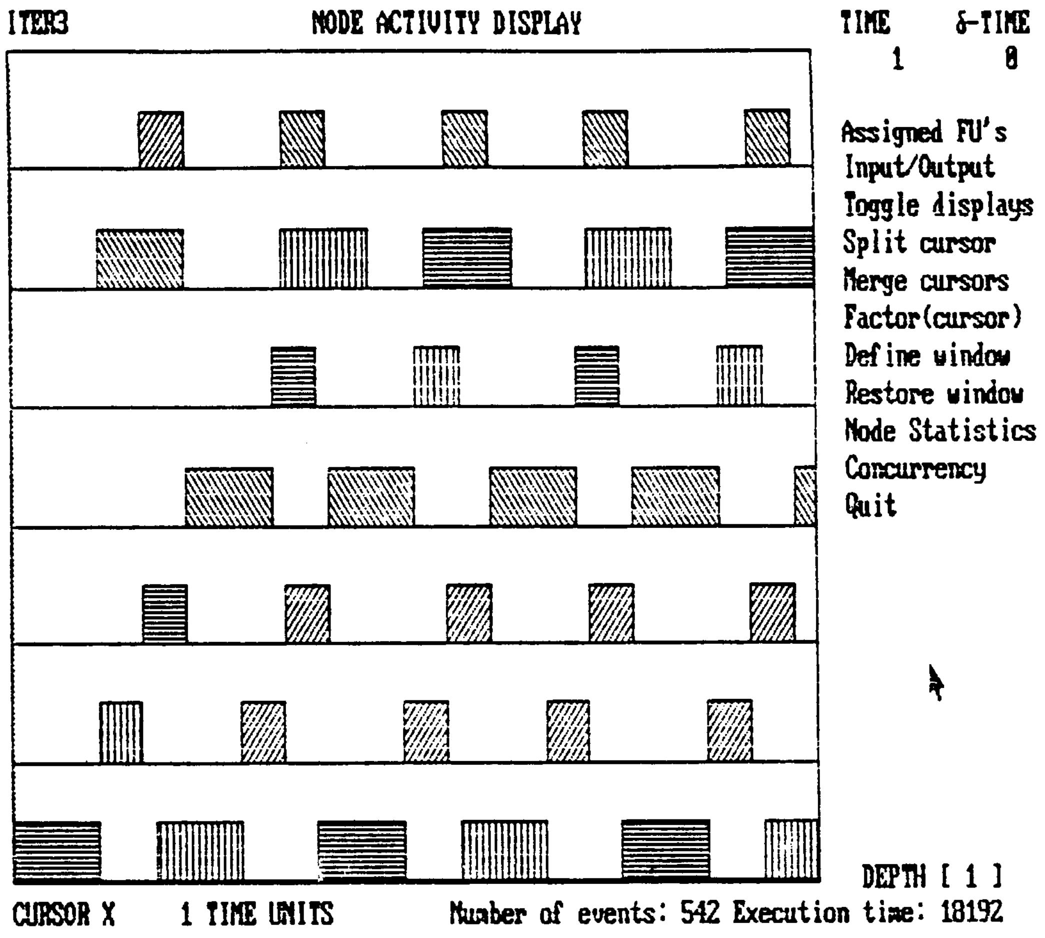
By recording the event time of every event of a particular graph execution, the system can be analyzed. The analysis yields information on how the various FUNs dispatch their respective assignments, how they are controlled by the data flow in the system, and how they compete for memory access. In terms of performance, information can be derived to evaluate data thruput parameters. For such an analysis, a program to translate this information to a more readable form is being developed. This software is called ANALYZER.

In order to demonstrate the general features of the ANALYZER program, an example was run in the prototype system. This example is the partitioned state equation algorithm that was previously described in Section 2.3. Recall that this particular graph has eleven nodes, one input to the algorithm and one output from the algorithm. The algorithm is presented witn a sequence of ten inputs.

Several figures are presented to illustrate the behavior of the algorithm and the diagnostic products of ANALYZER. Figure 5.1 is a display of the activity of the algorithm graph nodes 1 to 7. In these plots, the x-axes (time) are aligned in order to show the concurrent behavior of the various nodes. The lowest graph is a display of Node #1. The display indicates when that node becomes active and the duration of that state. For this example, three FUNs are available to the system. Whenever a box is filled with horizontal lines it indicates that the Functional Unit #1 iss connected to that particular task or node. Vertical lines indicate

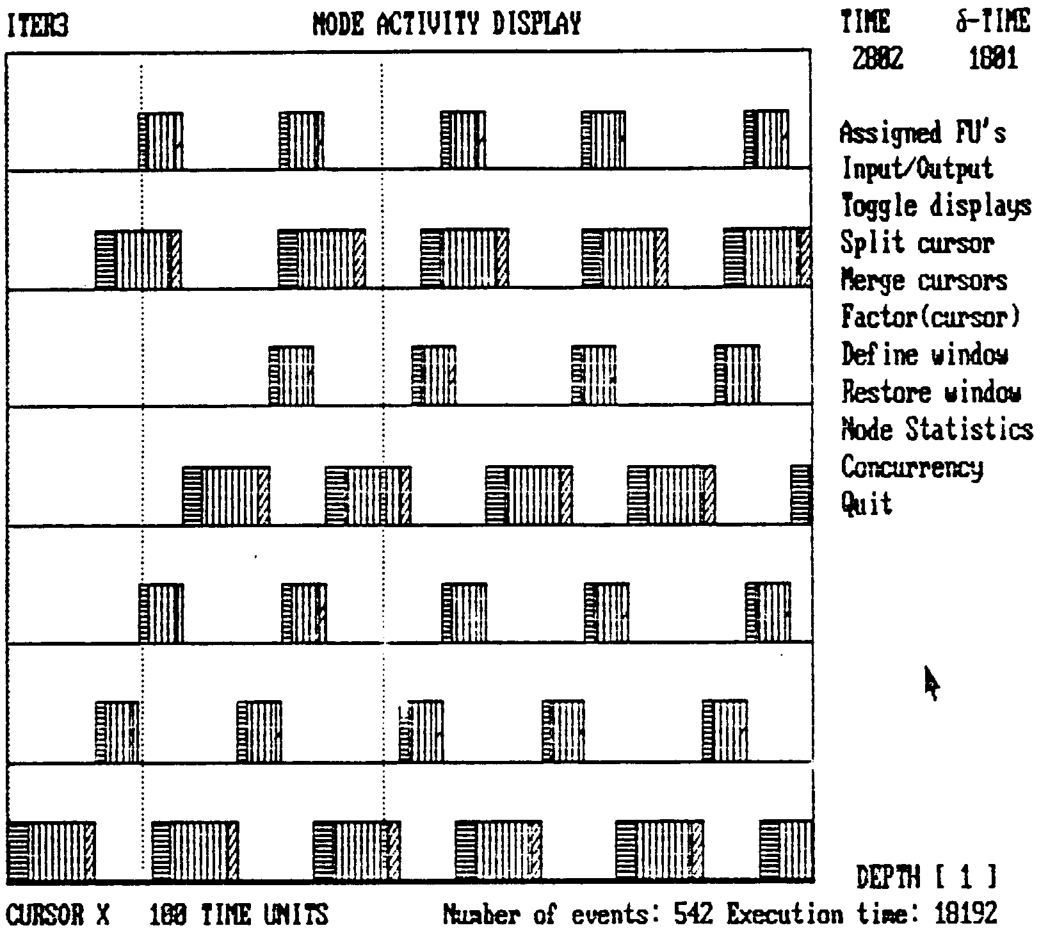
Functional Unit #2, lines running from up-left to down-right indicate Functional Unit #3, and so on. The display can be changed in order to show the amount of time required to execute the individual sub-processes (i.e. data input read time, process time, waiting for data output clear and data output write time) for every node. This presentation format is shown in Fig. 5.2. Horizontal lines indicate input read time, vertical lines indicate process time, lines running from up-left to down-right indicate waiting for data outputs to clear, and lines running from down-left to up-right indicate data output write time. A feature that helps the user to more closely examine the data presented by these displays is the capability to "zoom in" to a marked section. The region enclosed by the two cursors in Fig. 5.2 is enlarged in Fig. 5.3. Any other region can be defined in Fig. 5.3 and be enlarged again and so on. In this case the differences between the sub-processes marking are more evident. An additional display provides a time activity history of each individual FUN. This ANALYZER FUN activity display for the algorithm example is shown in Fig. 5.4. The bottom plot corresponds to Functional Unit #1. It is also possible to apply the "zoom" feature to this screen. The interpretation of the patterns is the same as shown in Fig. 5.2.

Of particular importance is the quantifying of the algorithm data performance. The ANALYSER program provides displays to indicate The Input to Output time (TBIO), Time between Inputs (TBI), and Time between Outputs (TBO). For the algorithm example having a sequence of ten inputs, Fig. 5.5 shows the tabulated values and a pictorial display. The solid line represents TBI, the dashed line represents TBIO, and the dotted line represents TBO. The graphs are not presented on the same scale, but are presented to provide qualitative information on the transient and steady state



```
CURSOR X     1 TIME UNITS          Number of events: 542 Execution time: 18192
```

Figure 5.1 Analyzer's Node Activity Display, Assigned FU's.



Number of events: 542 Execution time: 18192

Figure 5.2. Read/Process/Write Node Activity.

characteristics of these performance measures. An additional performance statistic provided by ANALYZER is the mean value of every sub-process time for a given node for the entire process. The boxes in the lower right corner of Figs. 5.1, 5.2, 5.4 and 5.4 contain this information for several nodes. The "concurrency" of a selected region in time is illustrated in Fig. 5.6. This plotting shows the number of nodes that are working at the same time versus time. The box in the lower right corner indicates the percentage of the total time in the viewport that a given number of nodes or FUNs are working at the same time. Time between any two points along the x-axis can be measured using a double cursor arrangement. One cursor is fixed and the other can be placed at any point in time. The difference between both is continuously reported in the upper right corner of the screen as shown in Fig. 5.2.

The SIMULATION program, as reported in [8], has been modified in order to report the same type of information as the hardware system. In this fashion, the execution of a specific graph can be compared to that of the simulated behavior using this analyzer program. This way the simulation program can be 'tuned' to the hardware for more accuracy. The ANALYZER program will run in an IBM PC or true compatible with at least 256k of memory, one disk drive and an Enhanced Graphics Adapeer with at least 64k of memory and either an Enhanced Color Display or Monochrome Display. The version used for the figures will run under these display restrictions using an Enhanced Color Display (640X350 pixels) or Monochrome Display using just four colors or tones. There is another version of the program that will run using a Color Display and showing up to sixteen colors (640X200 pixels) or with an Enhanced Color Display also with sixteen colors (640X350 pixels).

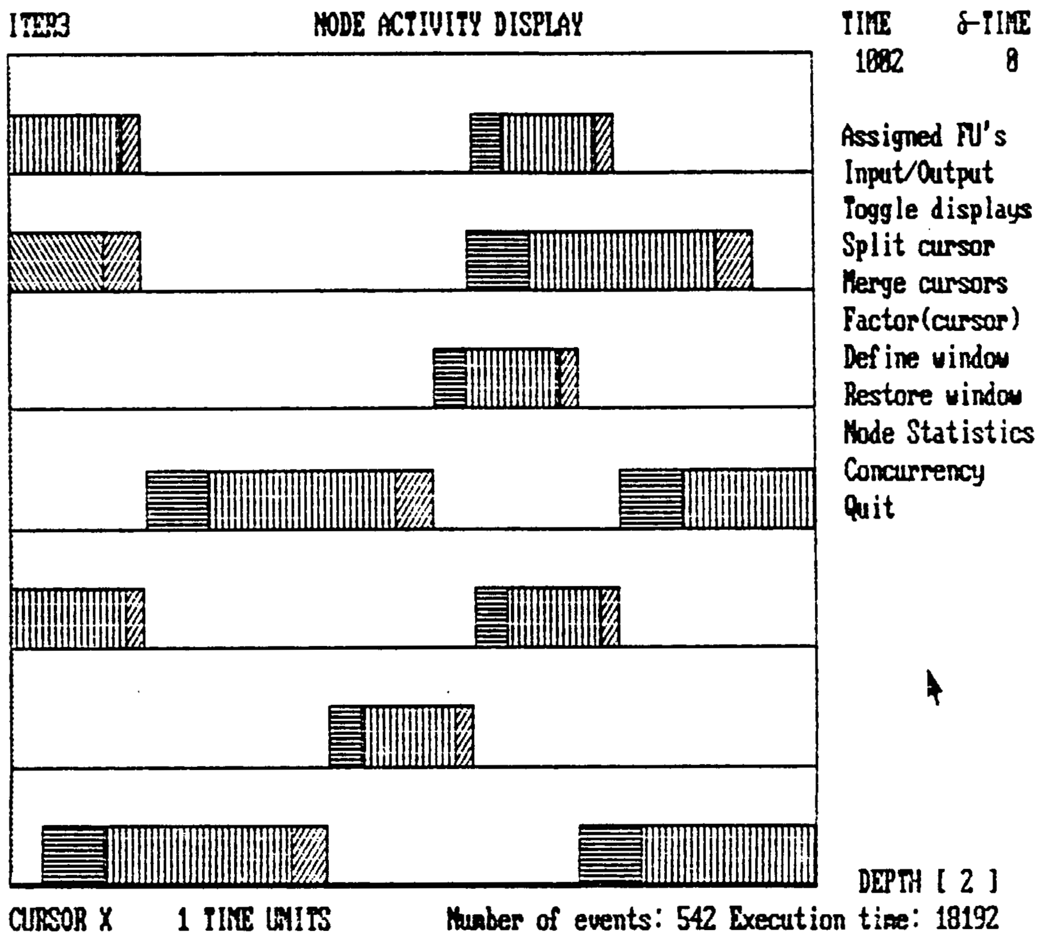


Figure 5.3 Enlargement of Read/Process/Write Display.

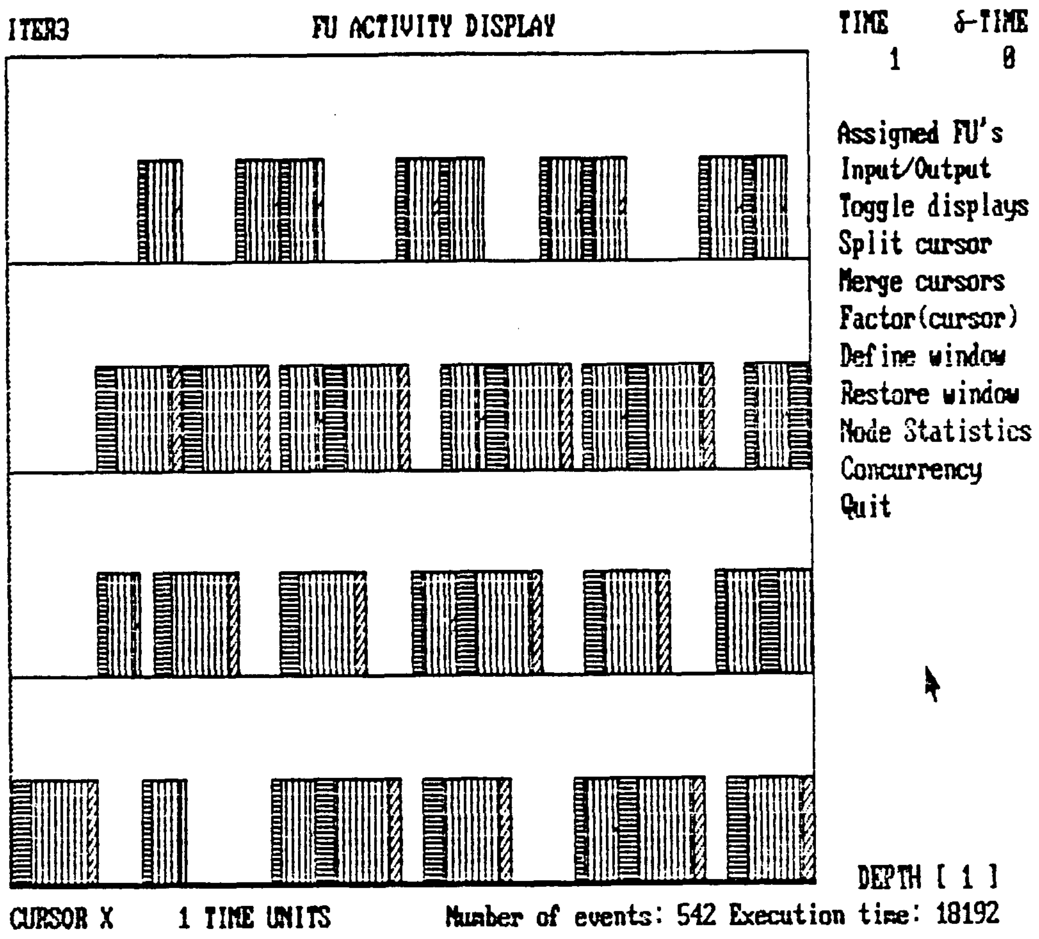
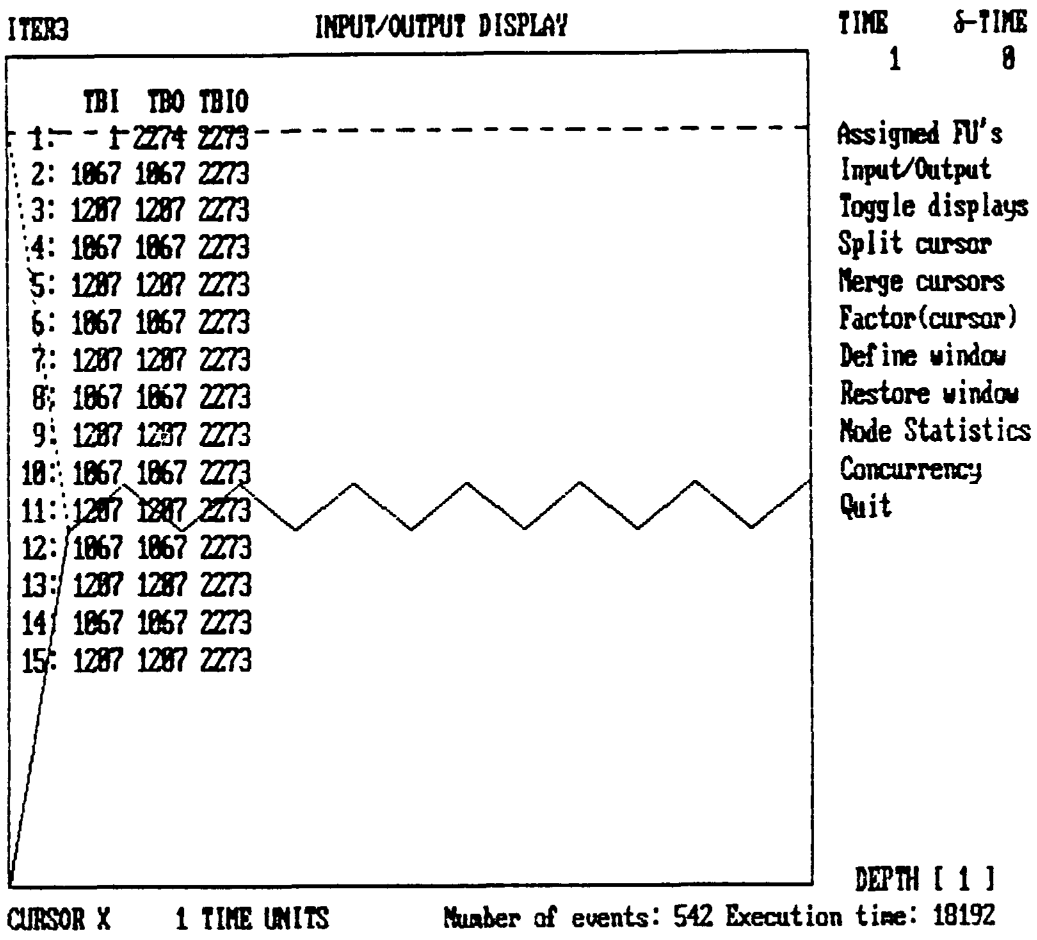


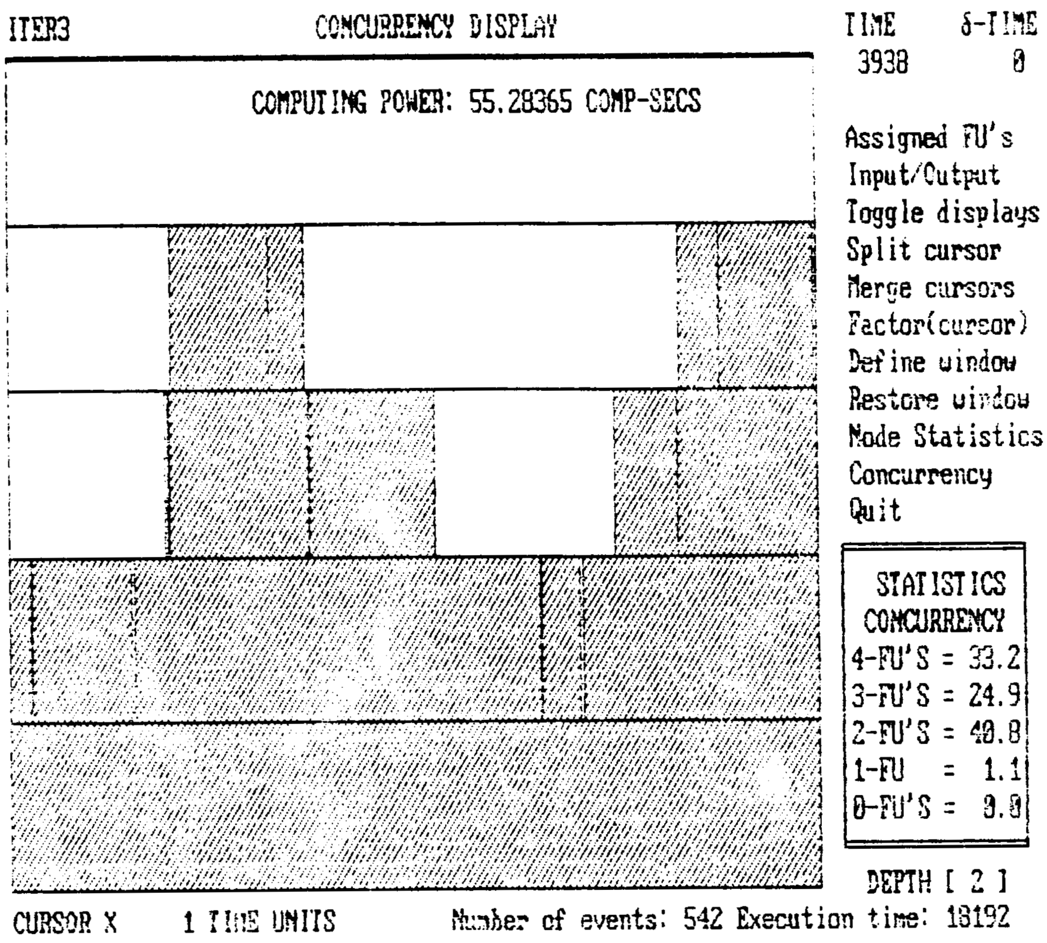
Figure 5.4 FUN Activity.



```
ITER3                 INPUT/OUTPUT DISPLAY            TIME    δ-TIME
                                                        1        0
     TBI  TBO  TBIO
 1:     1 2274 2273
 2: 1067 1067 2273
 3: 1287 1287 2273
 4: 1067 1067 2273
 5: 1287 1287 2273
 6: 1067 1067 2273
 7: 1287 1287 2273
 8: 1067 1067 2273
 9: 1287 1287 2273
10: 1067 1067 2273
11: 1287 1287 2273
12: 1067 1067 2273
13: 1287 1287 2273
14: 1067 1067 2273
15: 1287 1287 2273
                                                         DEPTH [ 1 ]
CURSOR X     1 TIME UNITS       Number of events: 542 Execution time: 18192
```

Figure 5.5 Timing Analysis Display.

ORIGINAL PAGE IS
OF POOR QUALITY


ITER3 CONCURRENCY DISPLAY TIME δ-TIME
3938 0

COMPUTING POWER: 55.28365 COMP-SECS

STATISTICS
CONCURRENCY
4-FU'S = 33.2
3-FU'S = 24.9
2-FU'S = 40.8
1-FU = 1.1
0-FU'S = 0.0

DEPTH [2]

CURSOR X 1 TIME UNITS Number of events: 542 Execution time: 18192

Figure 5.6 Analyzer's Concurrency Display.

REFERENCES

1. C. Petri, "Kommunikation mit Automaton," Ph.D Dissertation, University of Bonn, West Germany, 1962.

2. A. Holt and F. Commoner, "Event and Conditions" Applied Data Research, New York, 1970.

3. J. L. Peterson, Petri Net Theory and the Modeling of Systems, Englewood Cliffs, N. J., Prentice-Hall, 1981.

4. T. Murata, "Circuit Theoretic Analysis and Synthesis of Marked Graphs," IEEE Transactions on Circuits and Systems, Vol. CAS-24, No. 7, pp. 400-405, July 1977.

5. J. W. Stoughton and R. R. Mielke, "Petri Net Model for Concurrent Processing of Complex Algorithms," Proceedings 1986 Government Microcircuit Applications Conference, Vol. 12, pp. 11-14, November 1986.

6. T. Murata and J. Koh, "Reduction and Expansion of Live and Safe Marked Graphs," IEEE Transactions on Circuits and Systems, Vol. CAS-27, No. 1, January 1980.

7. R. Johnsonbaugh and T. Murata, "Additional Methods for Reduction and Expansion of Marked Graphs," IEEE Transactions on Circuits and Systems, Vol. CAS-28, No. 10, October 1981.

8. K. Jackson, R. Tymchyshyn, R. Mielke, J. Stoughton and R. Obando, "Simulation Software for Concurrent Processing," Proceedings of Southeastcon 87, Tampa, FL April 1987.

9. S. Seshu, and M. Reed, Linear Graphs and Electrical Networks, Assison-Wesley Publishing Co., Inc., 1961.

BIBLIOGRAPHY

PETRI-NETS

F. Commoner, "Deadlocks in Petri Nets," Report CA-7206-2311, Wakefield, Massachusetts, Computer Associates, (June 1972), 50 pages.

F. Commoner, A. Holt, S. Even, and A. Pnueli, "Marked Directed Graphs," Journal of Computer and System Sciences, Volume 5, Number 5, (October 1971), pages 511-523.

R. DeVillers, and G. Louchard, "Realization of Petri Nets Without Conditional Statements," Information Processing Letters, Volume 2, Number 4, (October 1973), pages 105-107.

H. Genrich and E. Stankiewicz-Wiechno, "A Dictionary of Some Basic Notions of Net Theory," Lecture Notes in Computer Science, Berlin: Springer-Verlag, (1980).

[Genrich and Lautenbach 1978] H. Genrich, and K. Lautenbach, "Facts in Place/Transition-Nets," Proceedings of the Seventh Symposium on Mathematical Foundations of Computer Science 1978, Lecture Notes in Computer Science, Volume 64, Berlin: Springer-Verlag, (September 1978), pages 213-231.

A. Holt, and F. Commoner, "Events and Conditions," Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, New York: ACM, (June 1970), pages 1-52.

M. Jantzen and R. Valk, "Formal Properties of Place Transition Nets," Lecture Notes in Computer Science, Berlin: Springer-Verlag. (1980).

L. Kinney , and Y. Han, "Reduction of Petri Nets," Proceedings of the 14th Allerton Conference on Circuits and Systems Theory, (September 1976).

T. Murata, "State Equations, Controllability and Maximal Matchings of Petri Nets," IEEE Transactions on Automatic Control, Volume AC-22, Number 3, (June 1977), pages 412-416.

T. Murata, "A Method for Synthesizing Marked Graphs from Given Markings," Proceedings of the Tenth Annual Asilomar Conference on Circuits, Systems, and Computers, (November 1976), pages 202-206.

T. Murata, "Petri Nets, Marked Graphs, and Circuit-System Theory," IEEE Circuits and Systems Society Newsletter, Volume 11, Number 3, (June 1977), pages 2-12.

T. Murata, "Circuit Theoretic Analysis and Synthesis of Marked Graphs," IEEE Transactions on Circuits and Systems, Volume CAS-24, Number 7, (July 1977), pages 400-405.

T. Murata, and R. Church, "Analysis of Marked Graphs and Petri Nets by Matrix Equations," Research Report MDC 1.1.8, Department of Information Engineering, University of Illinois, Chicago, Illinois, (November 1975).

T. Murata, and T. Shah, "On Liveness, Deadlock, and Reachability of E-Nets," Proceedings of the 14th Annual Allerton Conference on Circuits and Systems Theory, (September 1976), pages 597-605.

T. Murata, R. Church, and A. Amin, "Matrix Equations for Petri Nets and Marked Graphs," Proceedings of the Ninth Annual Asilomar Conference on Circuits, Systems, and Computers, (November 1975), pages 36-41.

C. Petri, "Introduction to General Net Theory," Lecture Notes in Computer Science, Berlin: Springer-Verlag, (1980).

J. Sifakis, "Structural Properties of Petri Nets," Proceedings of the Seventh Symposium on Mathematical Foundations of Computer Science 1978, Lecture Notes in Computer Science, Volume 64, Berlin: Springer-Verlag, (September 1978), pages 474-483.

S. Yu, and T. Murata, "PT-marked Graphs: A Reduced Model of Petri Nets," Proceedings of the 16th Allerton Conference on Communication, Control and Computing, (October 1978).

H. J. Genrich, K. Lautenbach, and P. S. Thiagarajan, "Elements of General Net Theory," Net Theory and Applications, Lecture Notes in Computer Science, vol. 84, Springer-Verlag, New York, 1980. pp. 21-164.

E. W. Mayr, "An Algorithm for the General Petri Net Reachability Problem, Procs, of the 13th Annual ACM Symposium on Theory of Computing, May 1981, pp. 238-246.

J. L. Peterson, Petri Net Theory and the Modeling of Systems, Prentice-Hall, Englewood Cliffs, N. J., 1981.

A. T. Amin and T. Murata, "A Characterization of Live and Safe Markings of Directed Graphs," Proceedings of the 1976 Conference on Info. Sci. and Syst., Johns Hopkins University, Baltimore, March 1976.

T. Murata and J. Y. Koh, "Reduction and Expansion of Live and Safe Marked Graphs," IEEE Transactions on Circuits and Systems, vol. CA -27, No. 1, Jan. 1980. pp. 68-70.

R. Johnsonbaugh and T. Murata, "Additional Methods for Reduction and Expansion of Marked Graphs," IEEE Transactions on Circuits and Systems; idem. "Analysis of Resource Requirements in Marked Graph Computation Models," Proceedings of the 1980 International Symposium on Circuits and Systems, Apr. 1980, pp. 342-345.

E. Pless and H. Plunnecke, "A Bibliography of Net Theory," Report ISF, ISF-79-04, GMD, Bonn, August 1979.

MODELS FOR CONCURRENT PROCESSING

D. Adams, "A Model for Parallel Computations," Parallel Processor Systems, Technologies and Applications, New York: Spartan Books, (1970), pages 311-334.

T. Agerwala, "Some Applications of Petri Nets," Proceedings of the 1978 National Electronics Conference, Volume 23, (October 1978), pages 149-154.

P. Azema, R. Valette, and M. Diaz, "Petri Nets as a Common Tool for Design Verification and Hardware Simulation," Proceedings 13th Design Automation Conference, New York: IEEE, (June 1976), pages 109-116.

J. Baer, "A Survey of Some Theoretical Aspects of Multiprocessing," Computing Surveys, Volume 5, Number 1, (March 1973), pages 31-80.

J. Cotronis, and P. Lauer, "Verification of Concurrent Systems of Processes," Proceedings of the International Computing Symposium 1977, Amsterdam North-Holland, (April 1977), pages 197-207.

S. Foo, and G. Musgrave, "Comparison of Graph Models for Parallel Computation and Their Extension," Proceedings of the 1975 International Symposium on Computer Hardware Description Languages and Their Applications, New York: IEEE, (September 1975), pages 16-21.

R. Lipton, L. Snyder, and Y. Zalcstein, "A Comparative Study of Models of Parallel Computation," Proceedings of the 15th Annual Symposium on Switching and Automata Theory, New York: IEEE, (October 1974), pages 145-155.

R. Miller, "A Comparison of Some Theoretical Models of Parallel Computation," IEEE Transactions on Computers, Volume C-22, Number 8, (August 1973), pages 710-717.

J. Peterson, and T. Bredt, "A Comparison of Models of Parallel Computation," Information Processing 74, Proceedings of the 1974 IFIP Congress, Amsterdam: North-Holland, (August 1974), pages 466-470.

F. Rammig, "Petri-Net Based Description, Analysis and Simulation of Concurrent Processes," Proceedings 14th Design Automation Conference, New York: IEEE, (June 1977).

K. Lautenbach and H. A.Schmid, "Use of Petri nets for proving correctness of concurrent systems," Proceedings of IFIP Congress 74, North Holland Pub. Co., 1974, pp. 187-191.

R. E. Miller, "Some relationships between various models of parallelism and synchronization," Report RC-5074, IBM T. J. Watson Res. Center, Yorktown Heights, N. Y., Oct. 1974.

T. Murata, "Relevance of Network Theory to Models of Distributed/Parallel Processing," Journal of the Franklin Institute, vol. 310, no. 1, 1980, pp. 41-50.

C. Ramchandam, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets," Report TR-120, Project MAC, MIT, Cambridge, Mass., Feb. 1974.

L. Cox, Jr., "Predicting Concurrent Computer System Performance Using Petri Net Models," Proceedings of the 1978 ACM,(December 1978), pages 901-913.

Y. Han, "Performance Evaluation of a Digital System Using a Petri Net-like Approach," Proceedings of the National Electronics Conference, Volume 23, (October 19780, pages 166-172.

W. Heimerdinger, "A Petri Net Approach to System Level Fault Tolerance Analysis, Proceedings of the National Electronics Conference, Volume 23, (October 1978), pages 161-165.

J. Sifakis, "Use of Petri Nets for Performance Evaluation, Measuring, Modeling and Evaluation of Computer Systems: Proceedings of the Third International Workshop on Modeling and Performance Evaluation of Computer Systems, Amsterdam: North Holland, (1977), pages 75-93.

J. Sifakis, "Realization of Fault-Tolerant Systems by Coding Petri Nets," Proceedings of the Eighth Annual International Conference on Fault-Tolerant Computing, New York: IEEE, (June 1978), page 205; Also (revised) Journal of Design Automation and Fault-Tolerant Computing, Volume 3, Number 2, (1979).

J. Sifakis, "Use of Petri Nets for Performance Evaluation," Advanced Course on General Net Theory of Processes and Systems, Hamburg, (October 1979); Also Lecture Notes in Computer Science, Berlin: Springer-Verlag. (1980).

G. S. Ho and C. V. Ramamoorthy, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," IEEE Transactions on Software Engineering, vol. SE-6, no. 5, Sept. 1980, pp. 440-449.

T. Murata, "Synthesis of Decision-Free Concurrent Systems for Prescribed Resources and Performance," IEEE Transactions on Software Engineering, vol. SE-6, no. 6, Nov. 1980, pp. 525-530.

DATA FLOW ARCHITECTURE

J. Dennis, "Modular, Asynchronous Control Structures for a High Performance Processor," Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, New York; ACM, (June 1970), pages 55-80.

C. Seitz, "Asynchronous Machines Exhibiting Concurrency," Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, New York: ACM, (June 1970), pages 93-106.

M. Sowa and T. Murata, "A data flow computer architecture with program and token memories," Procs. of the 14th Asilomar Conf. on Circuits, Systems, and Computer, IEEE Comp. Society, Long Beach, Cal., Nov. 1980. Also IEEE Trans. on Computers, Sept. 1982, pp. 820-824.

S. H. Yu and T. Murata, "Modeling and simulating data flow computations at machine language level," Procs. of Conf. on Simulation, Measurement and Modeling of Computer Systems, ACM, New York, Aug. 1979, pp. 207-213.

J. B. Dennis "Data Flow Supercomputers," Computer, vol. 13, Nov. 1980, pp. 48-56.

S. S. Reddi, "A Parallel Computer with Centralized Control," IEEE Comp. So. Repository, R76-22, Feb. 1976.

APPENDIX A

Petri Net Background

A useful mathematical tool for modeling systems with interacting concurrent components is the Petri net. Petri nets were first developed by Carl Petri [1] in 1962, and later were identified as a useful system analysis tool in the work of Holt and Commoner [2]. A comprehensive introductory treatment of Petri nets is presented in Peterson [3].

A Petri net is a bipartite directed multigraph G described by a five tuple, $G=(P,T,\alpha,\beta,M_0)$. The set P is a set of $|P|=m$ objects called places. Places are used to represent the condition or status of a system. T is a set of $|T|=n$ objects, disjoint from elements of P, called transitions. Transitions are used to represent events or actions in a system. The terms $\alpha : P \times T \to N$ (set of nonnegative integers) is called the input function. The term $\alpha(p_i,t_j)$ is the number of arcs directed from place p_i into transition t_j. Arcs directed from a place p_i to a transition t_j indicates that the status represented by place p_i is a precondition for the event represented by transition t_j. The expression $\beta : P \times T \to N$ is called the output function. $\beta(p_k,t_j)$ is the number of arcs directed out of transition t_j to place p_k.

Certain physical characteristics of the class of problems under consideration lead to a simplified Petri net representation. In a decomposed algorithm, the performance of a primitive operation is either preconditioned on the availability of a signal or it is not. That is, arcs associate places (conditions) to transactions (events) in a binary way. Therefore, $\alpha : P \times T \to (0,1)$ and $\beta : P \times T \to (0,1)$. A Petri net having such restricted inputs and output functions is called an ordinary Petri net.

Arcs directed from a transition t_j to a place p_k indicates that the action represented by transition t_j to a place p_k indicates that the action represented by transition t_j results in the status represented by place p_k. A condition may exist and is indicated by marking the corresponding place with one or more tokens. M_0:P -> N is called the initial marking vector. The components of M_0 identify the number of tokens marking each place.

The placement of tokens in a Petri net, and the status of the corresponding system, evolve according to the following rules. A transition t_i is enabled if all input places contain at least as many tokens as input arcs. That is, $M(p) > \alpha(p,t_i)$ for all $p \epsilon P$. When an enabled transition t_i fires, tokens in each input place p_j equal in number to the number of input arcs $\alpha(p_j,t_i)$ are removed. Tokens in each output place p_k equal in number to the number of output arcs $\beta(p_k,t_i)$ are deposited. Transition firings continue as long as at least one transition is enabled. When there are no enabled transitions, the execution of the net halts.

The concept of time is not explicitly included in the definition of Petri nets. However, for performance evaluation and scheduling problems, it is necessary and useful to define timed delays associated with the performances of events. Such a Petri net is called a timed Petri net and is defined by the six-tuple $G = (P,T,\alpha,\beta, M_0,\lambda)$. The first five parameters are as previously defined. The function λ : T -> R (nonnegative real numbers) is called the firing time function. The components of λ identify the time delay associated with each transition. The placement of tokens in a timed Petri net evolve according to the following rules. Tokens have two states called reserved and non-reserved. A transition t_i is enabled if all input places contain at least as many non-reserved tokens as input arcs. As before, an enabled transition may or may not fire. When an enabled transition t_i fires, the firing process commences by changing the status of tokens in each input place p_j, equal in number to the number of input arcs $\alpha(p_j,t_i)$, from non-reserved to reserved. Firing of transition t_i terminates $\lambda(t_i)$ time units after initiation by removing $\alpha(p_j,t_i)$ reserved tokens from each input place p_j, and depositing $\beta(p_k,t_i)$ non-reserved tokens at each output place p_k.

Two very important subclasses of Petri nets are state machines and marked graphs. A state machine is a Petri net in which each transition is restricted to having exactly one input place and one output place. A marked graph is the dual of a state machine. A marked graph is a Petri net in which each place is restricted to having exactly one input transition and one output transition. Thus, a state machine can represent conflicts by a place with several output transitions, but cannot model the creation and destruction of tokens required to model concurrency or the waiting which characterizes synchronization. Marked graphs, on the other hand, can not model conflicts or data-dependent decisions, but can model concurrency.

APPENDIX B

PUBLICATIONS AND PRESENTATIONS

1. John W. Stoughton and Roland R. Mielke, "Petri Net Model for Concurrent Processing of Complex Algorithms," Government Microcircuit Applications Conference, Vol. 12 pp. 11-14, Nov. 1986.

2. K. Jackson, R. Tymchyshyn, John W. Stoughton and Roland R. Mielke, "Simulation Software for Concurrent Processsing," Proceedings of Southeastcon 1987. Tampa, FL. Apr. 1987.