NASA Contractor Report 4339

Algorithm to Architecture Mapping Model (ATAMM) Multicomputer Operating System Functional Specification

R. Mielke, J. Stoughton,
S. Som, R. Obando,
M. Malekpour, and B. Mandala

COOPERATIVE AGREEMENT NCC1-136
NOVEMBER 1990

NASA

[illegible]

NASA Contractor Report 4339

Algorithm to Architecture Mapping Model (ATAMM) Multicomputer Operating System Functional Specification

R. Mielke, J. Stoughton,
S. Som, R. Obando,
M. Malekpour, and B. Mandala
Old Dominion University Research Foundation
Norfolk, Virginia

Prepared for
Langley Research Center
under Cooperative Agreement NCC1-136

NASA

National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Division

1990

TABLE OF CONTENTS

LIST OF FIGURES

TABLE OF CONTENTS (continued)

LIST OF FIGURES (continued)

TABLE OF CONTENTS (concluded)

LIST OF FIGURES (concluded)

ALGORITHM TO ARCHITECTURE MAPPING MODEL (ATAMM) MULTICOMPUTER OPERATING SYSTEM FUNCTIONAL SPECIFICATION

I. INTRODUCTION

The purpose of this document is to describe the ATAMM Multicomputer Operating System. ATAMM, an acronym for Algorithm To Architecture Mapping Model, is a marked graph model which describes the implementation of a decomposed algorithm on a data flow computer architecture. AMOS, the ATAMM Multicomputer Operating System, is an operating system which implements the ATAMM rules. A first generation version of AMOS has been developed for the Advanced Development Module (ADM), a four processor architecture based on the Westinghouse VHSIC 1750A instruction set processor. A second generation version of AMOS is being developed for the Generic VHSIC Spaceborne Computer (GVSC), a spaceborne four processor breadboard which also is based on the VHSIC 1750A instruction set architecture.

AMOS is a special purpose operating system suitable for periodic execution of decomposed signal processing and control algorithms in real-time on a variety of multicomputer architectures. An algorithm is expressed as a directed graph where vertices represent algorithm operations and edges represent data sets or operands. It is assumed that the algorithms are decision-free and large-grained. Decision-free refers to the absence of data dependent paths in the algorithm graph representation. Large-grained refers to the assumption that the time required to perform algorithm operations is large compared to the time required to move data from one graph node to another. The computer architecture is assumed to consist of two or more identical functional units or computing resources, each having a capability of processing, communication and memory. Functional units communicate with each other through an interconnecting bus. The functional units share a common global memory which may be either centralized or distributed. Coordination of resources in relation to data and control flow is directed by a graph manager which also may be centralized or distributed. Assignment of a functional unit to a specific algorithm operation is made by the graph manager according to ATAMM rules. In a specific hardware setting, the graph manager, global memory and functional unit activities together form the ATAMM Multicomputer Operating System or AMOS.

The ATAMM model is important because it specifies criteria for a multicomputer operating system to achieve predictable and highly reliable performance. When sufficient resources are available, the system executes algorithms with maximum throughput and minimum computing time. When only limited resources are available or resources fail, performance degrades gracefully and predictably. The user is able to specify off-line tradeoffs between decreasing throughput or increasing computing time, and the operating system is able to implement these changes on-line in real time as the number of resources decreases. The ATAMM model also provides a platform for investigating the effect on performance of different algorithm decompositions and implementation strategies in a hardware independent context.

In Section II, the ATAMM model is defined and illustrated by example. Time performance of algorithms executed according to the ATAMM rules is considered in Section III. Strategies are described for generating operating conditions for predictable performance based on the number of available computing resources. In Section IV, the ADM implementation of AMOS is presented. Finally, in Section V, enhancements planned for the GVSC implementation of AMOS are described. The AMOS functional features described in Section IV, as modified to include the enhancements described in Section V, are to be implemented in the GVSC breadboard system.

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

II. THE ATAMM MODEL

The ATAMM model consists of a set of Petri net marked graphs which incorporates general specifications of communication and processing associated with the implementation of a decomposed, large-grained algorithm in a data flow architecture. In this section, the execution of a computational problem is represented by the ATAMM model. Some familiarity with Petri nets and marked graphs is assumed [1]. A more detailed description of the ATAMM model and its characteristics are found in [2, 3].

An algorithm marked graph (AMG) is a marked graph which represents a specific algorithm decomposition. Transitions and places are represented as nodes (vertices) and directed edges, respectively. Vertices of the algorithm marked graph are in a one-to-one correspondence with each occurrence of an algorithm operation. The transition times represent the computational times of the respective algorithm operations. The algorithm marked graph contains an edge (i, j) directed from vertex i to vertex j if the output of vertex i is an input for vertex j. Edge (i, j) is marked with a token if an output from vertex i is available as an input to vertex j. Source transitions and sink transitions for input and output signals are represented as squares.

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

$$x(k) = Ax(k-1) + Bu(k)$$

and the output equation

$$y(k) = Cx(k),$$

where x is a p-vector, u is an m-vector, and y is a r-vector. The algorithm operations are defined as matrix multiplication and vector addition, and the natural algorithm decomposition resulting from the state equation description is selected. The algorithm marked graph for this decomposed algorithm is shown in Figure 1. The initial marking indicates that initial condition data are available.

The algorithm marked graph is a useful tool for representing decomposed algorithms and for displaying data flow within an algorithm. However, the algorithm marked graph 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. These important aspects of concurrent processing are included in the ATAMM model through the definition of two additional graphs. These additional graphs are defined in the following.

The node marked graph (NMG) is a Petri net representation of the performance of an algorithm operation by a functional unit.

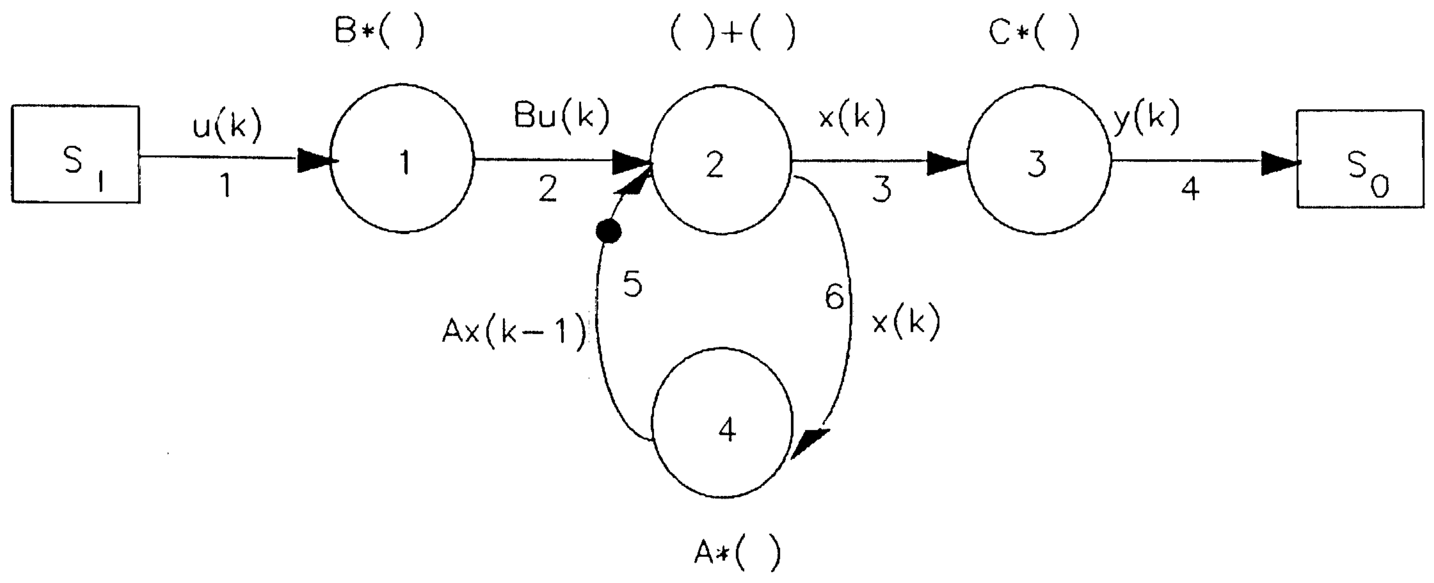


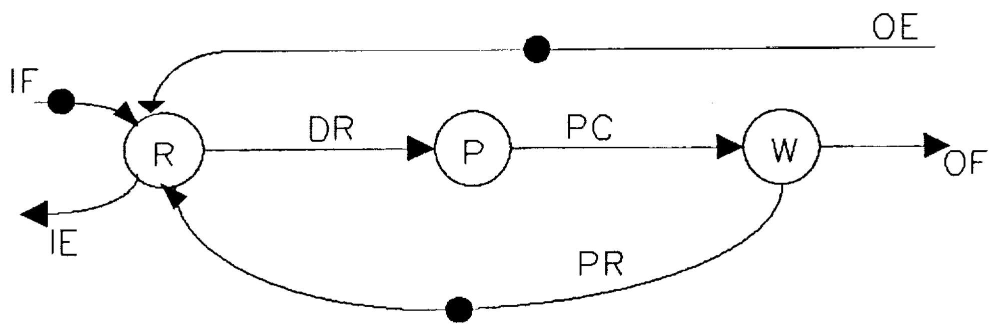
Figure 1. Algorithm marked graph for discrete system equation.

Three primary activities, reading of input data from global memory, processing of input data to compute output data, 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 transition has one additional condition for firing. In addition to having a token present on each incoming signal edge, a functional unit must be available in a queue of available functional units for assignment to the algorithm 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 functional units. The initial marking for an NMG consists of a single token in the Process Ready place. The NMG model is shown in Figure 2.

A computational marked graph (CMG) is constructed from the AMG and the NMG by the following rules:

1) Source and sink nodes in the algorithm marked graph are represented by source and sink nodes in the CMG.

2) Nodes corresponding to algorithm operations in the algorithm marked graph are represented by NMGs in the CMG.



NMG EDGE LABELS

IF	Input Buffer Full
IE	Input Buffer Empty
DR	Data Read
PC	Process Complete
PR	Process Ready
OE	Output Buffer Empty
OF	Output Buffer Full

Figure 2. ATAMM node marked graph model.

3) Edges in the algorithm marked graph are represented by edge pairs, one forward directed for data flow and one backward directed for control flow, in the CMG.

A forward directed edge goes from a predecessor write transition to a successor read or sink transition. Forward edges are also shown as part of the NMG in Figure 2 where they are labeled OF and IF edges of the predecessor and successor transitions, respectively. A backward directed edge goes from a successor read transition to a predecessor read or source transition. Backward edges are also shown as part of the NMG where they are labeled OE and IE edges of predecessor and successor transitions, respectively. The initial marking for the edge pair consists of a single token in the forward directed place if data are available, or a single token in the backward directed place if data are not available. In order to illustrate the construction of a computational marked graph, the CMG corresponding to the algorithm marked graph of Figure 1 is shown in Figure 3.

The complete ATAMM model consists of the algorithm marked graph, the node marked graph, and the computational marked graph. A pictorial display of the components of the ATAMM model are shown in Figure 4.

Graph execution based on the ATAMM rules has several useful and important properties [4]. Execution is live, reachable,

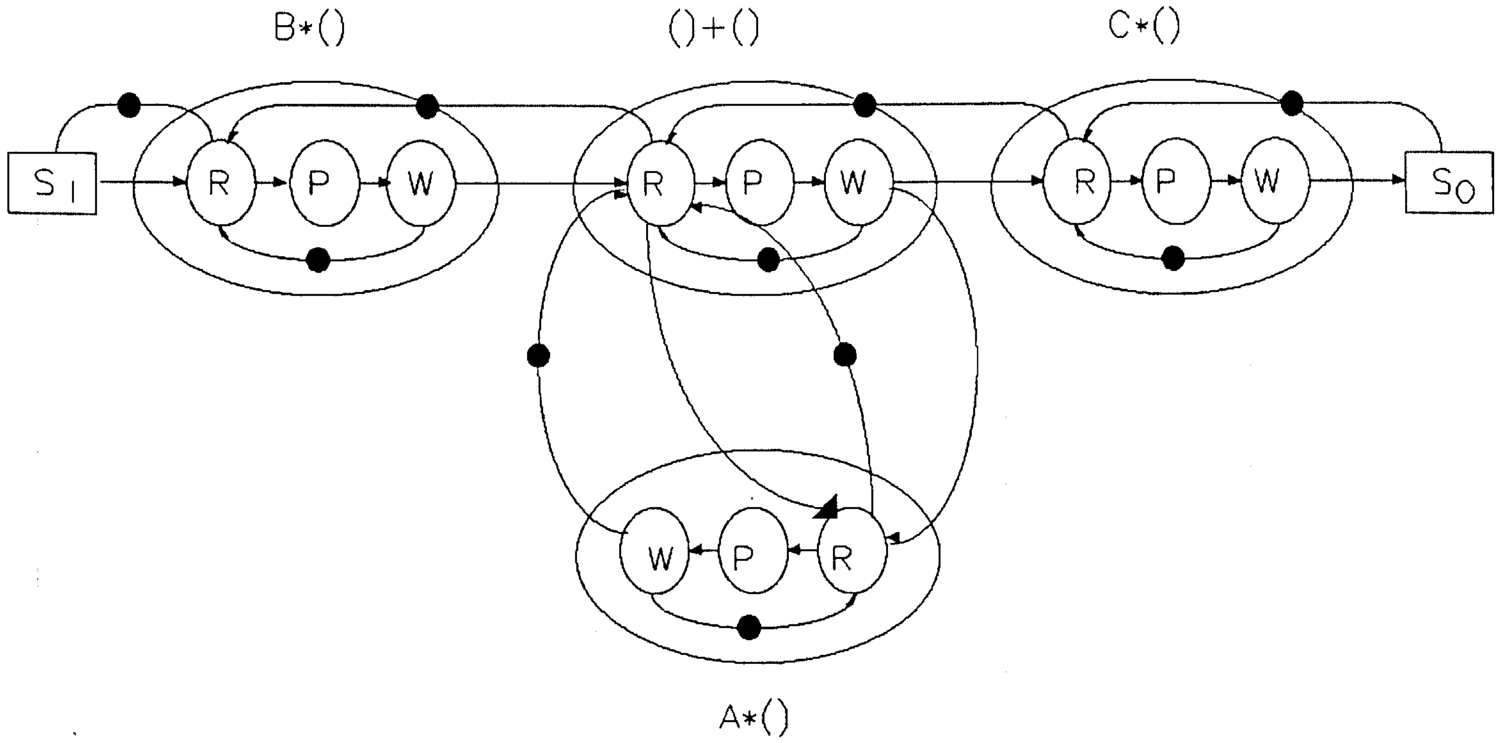


Figure 3. ATAMM computational marked graph model for discrete system equation.

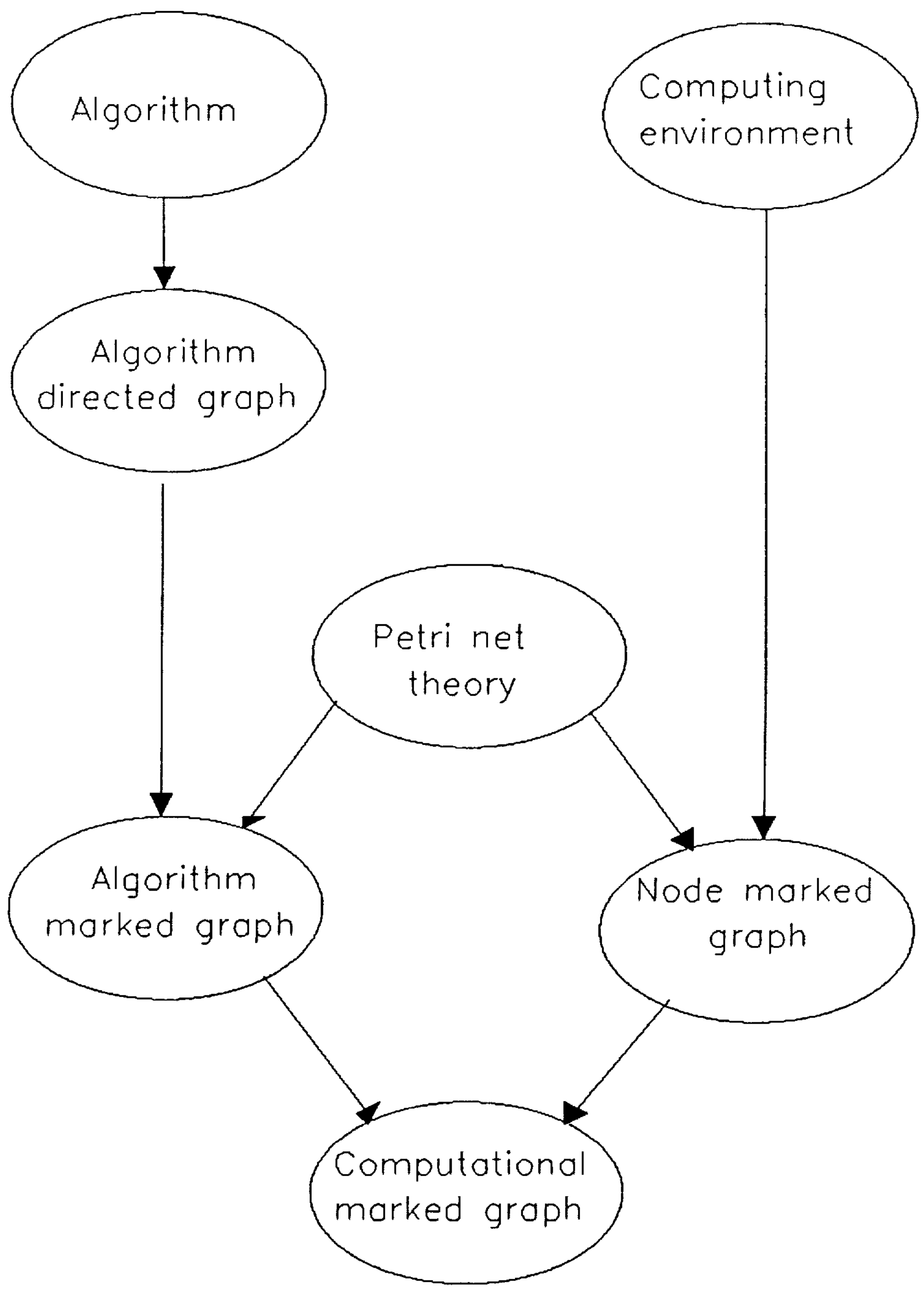


Figure 4. ATAMM model components.

safe, deadlock-free, and consistent [3]. Liveness indicates that all transitions in the CMG are firable from the initial marking, whereas reachability ensures that the CMG will generate an output for each input. Safeness guarantees that output of an algorithm operation will not be overwritten before it is picked up by a successor algorithm operation or sink. This property is a result of including backward control places in the CMG and is necessary for safe periodic operation. The necessary and sufficient condition for avoidance of deadlock in the graph play is to ensure that once assigned, a functional unit always is able to complete execution of an algorithm operation. A computation can not enter deadlock because no read transition is executed unless the output edges of the corresponding NMG are empty and a functional unit is available. The consistency property implies that computations are repeated periodically when input are applied periodically.

III. PERFORMANCE ANALYSIS AND DESIGN

In this section, the time performance of algorithms implemented in data flow architectures according to the ATAMM rules is investigated. First, performance measures for computing speed and throughput are defined. It is shown that the ATAMM model is useful for analytically calculating bounds for these measures. Then, graph play is described and used to determine resource requirements necessary to achieve a specified time performance. Finally, the ATAMM performance plane is defined. This diagram displays possible operating strategies with resource requirements shown as a parameter. Using this display, a system operator is able to specify quantitatively system time performance.

III.1. Performance Measures

Two measures of time performance, TBIO and TBO, are defined in this section. The performance measure TBIO (time between input and output) is the elapsed computing time between an algorithm input and the corresponding algorithm output. Therefore, TBIO is an indicator of computing speed. It is shown in [5] that the algorithm imposed lower bound for TBIO, denoted $TBIO_{LB}$, is given by the sum of transition times for nodes contained in the longest directed path from the input source to the output sink in the AMG.

The performance measure TBO (time between outputs) is the elapsed computing time between successive algorithm outputs when the AMG is operating periodically at steady-state. Therefore, the inverse of TBO is an indicator of output per unit time or throughput. It is shown in [5,6] that the algorithm imposed lower bound for TBO is given by the largest time per token of all directed circuits in the CMG. A second bound on TBO is imposed by the availability of resources. It is shown in [3] that the resource imposed lower bound for TBO is TCE/R where TCE (total computing time) is the sum of transition times for all nodes in the AMG and R is the number of available functional units. The lower bound for TBO, denoted TBO_{LB}, is the greater of the algorithm bound and the resource bound.

To illustrate the calculation of these performance bounds, consider as an example the AMG shown in Figure 5 and the corresponding CMG shown in Figure 6. The AMG contains four directed paths from the input source to the output sink. These paths, identified by included transitions, are (1, 2, 6, 7), (1, 3, 6, 7), (1, 4, 6, 7) and (1, 5, 6, 7). The sum of transition times of nodes in each path is 7 so that $TBIO_{LB} = 7$. The largest time per token of any directed circuit in the CMG is 2. There are several directed circuits which yield this result; one such directed circuit is the circuit containing the read, process and write transitions of node 6 and the read transition of node 7. Therefore, $TBO_{LB} = 2$.

III.2. Graph Play and Resource Requirements

Two diagrams which display graph play and are useful for determining the number of resources needed to achieve specified performance measures are defined next. The SGP (single graph play) diagram is a diagram which displays the execution of each

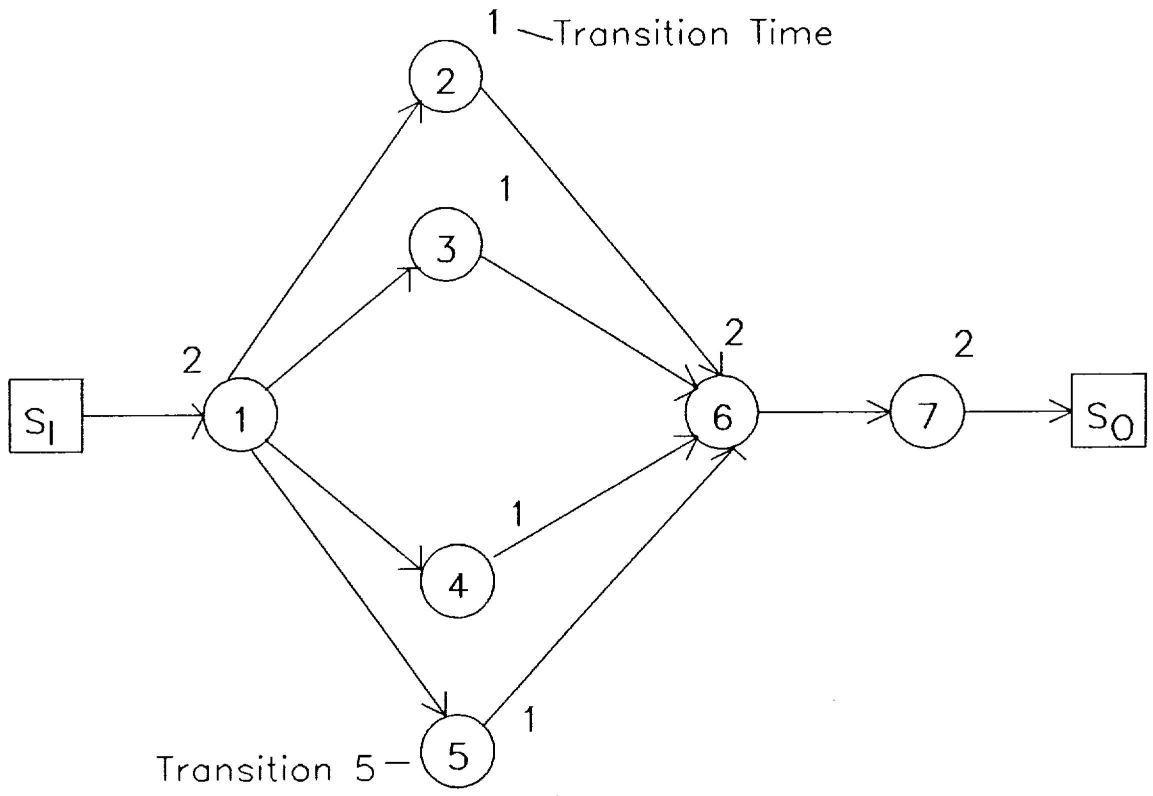


Figure 5. AMG for performance example.

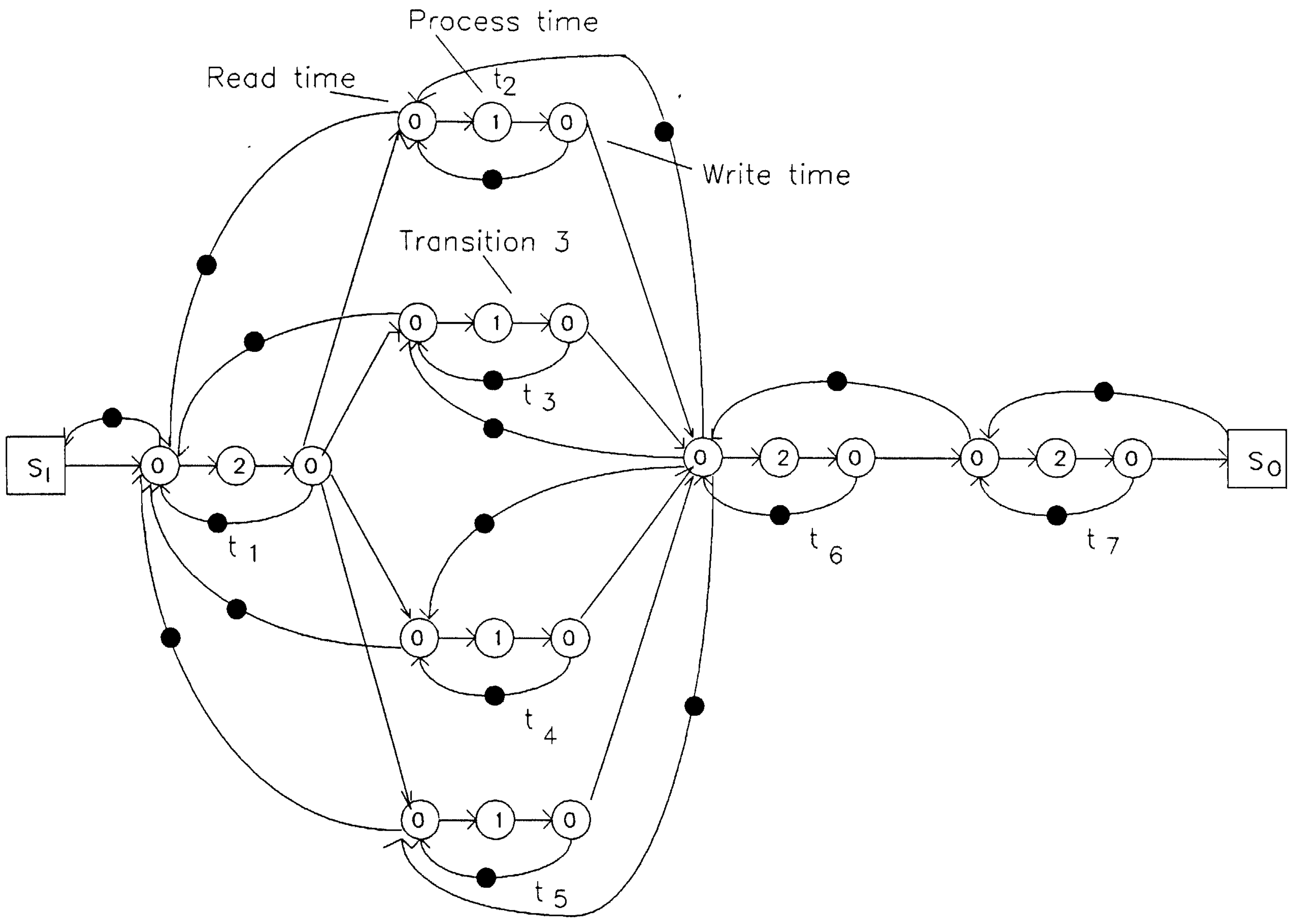


Figure 6. CMG for Figure 5.

node of the AMG as a function of time. The diagram is constructed for a single input data packet under the assumption that unlimited resources are available to play the graph. Node activity is denoted by a solid line and the symbols (<, >) are used to indicate the beginning and end of execution. When several nodes are active at the same time, lines indicating node activity are stacked vertically so that computing concurrency is apparent. The SGP diagram for the AMG of Figure 5 is shown in Figure 7.

The resource requirements to execute a single data packet are obtained by counting the number of active nodes during each time interval in the SGP diagram. The peak resource requirement is denoted by R_{min} and represents the minimum number of resources necessary to achieve operation at $TBIO = TBIO_{LB}$. For the AMG in Figure 5, $R_{min} = 4$ is the minimum number of resources necessary to execute the graph with $TBIO = TBIO_{LB} = 7$.

The TGP (total graph play) diagram is a diagram which displays the execution of each graph node when the graph is operating periodically in steady-state with period TBO. As with SGP, the diagram is constructed under the assumption that unlimited resources are available to play the graph, and a different diagram results for each value of TBO. The TGP diagram is drawn using information from SGP. SGP is divided into segments of width TBO, and these segments are overlaid to form

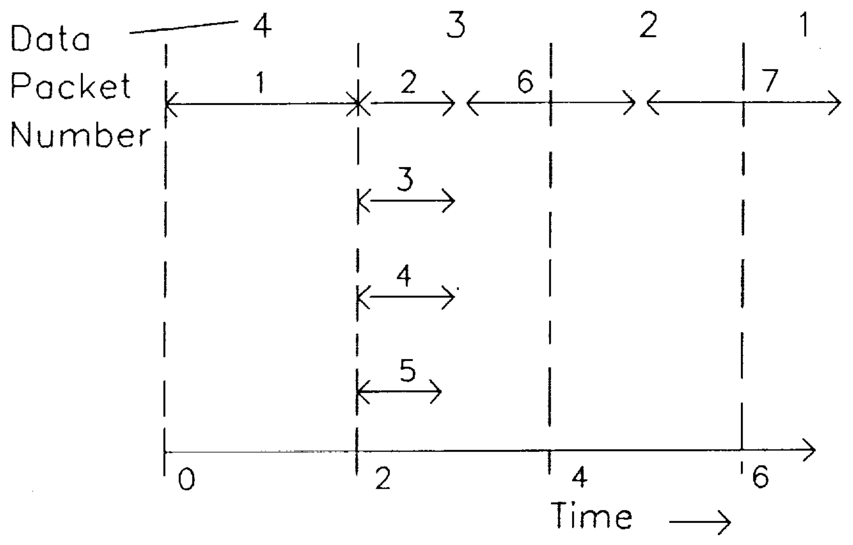


Figure 7. SGP diagram for Figure 5.

TGP. Each segment from SGP represents a new input data packet. Data packets are numbered sequentially so that the packet numbered i+1 is the data packet which is input to the graph TBO time units after the packet numbered i. To illustrate the construction of this diagram, TGP for the AMG of Figure 5 is shown in Figure 8.

The resource requirements to execute multiple data packets injected with period TBO are obtained by counting the number of active nodes during each time interval in the TGP diagram. The peak resource requirement necessary to execute the graph periodically with period greater than or equal to TBO is denoted by R_{max}. R_{max} is determined by finding the largest resource requirement in all TGP diagrams drawn for injection intervals greater than or equal to TBO. For example, the TGP diagram drawn for TBO = TBO_{LB} = 2 shown in Figure 8 indicates that a minimum of 7 resources is required. If this same TGP diagram is drawn for all values of TBO > 2, it can be shown that the required number of resources does not exceed 7. Therefore, R_{max} to achieve TBO = 2 for the AMG shown in Figure 5 is equal to 7.

III.3. ATAMM Performance Plane

For a given algorithm decomposition, the parameters TBIO, TBO and R define an operating point for ATAMM. The display of all operating points on a graph of TBO versus TBIO with R

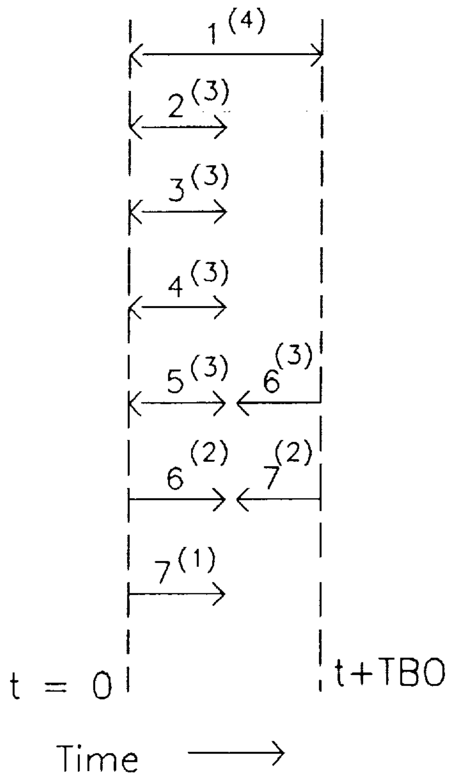


Figure 8. TGP diagram for Figure 7 with TBO = 2.

indicated as a parameter is called the ATAMM performance plane. The ATAMM performance plane, illustrated in Figure 9, is extremely useful for selecting system operating strategies. The use of this diagram is described in this section.

The best system performance is achieved by operating at point B where TBIO = $TBIO_{LB}$ and TBO = TBO_{LB}. The resource requirement associated with this operating point is the value of R_{max} computed from the TGP diagram drawn for $TBIO_{LB}$ and TBO_{LB}. Operation at point B is obtained by the use of injection control as shown in Figure 10. Injection control is a control procedure which limits the maximum rate at which new input data packets can be injected. When presented with continuously available input data packets, the natural behavior of a data flow architecture results in operation where data packets are accepted as rapidly as available resources and the input transition permit. This leads to operation at a steady-state operating point where TBO = TBO_{LB} but TBIO > $TBIO_{LB}$. This occurs because the pipeline from input to output becomes congested with extra data packets which must wait for free resources to be processed. Injection control eliminates data packet congestion and thus preserves operation at $TBIO_{LB}$.

When there are not sufficient resources to operate at point B, the operating point must be shifted to a new location having a smaller resource requirement. Using injection control

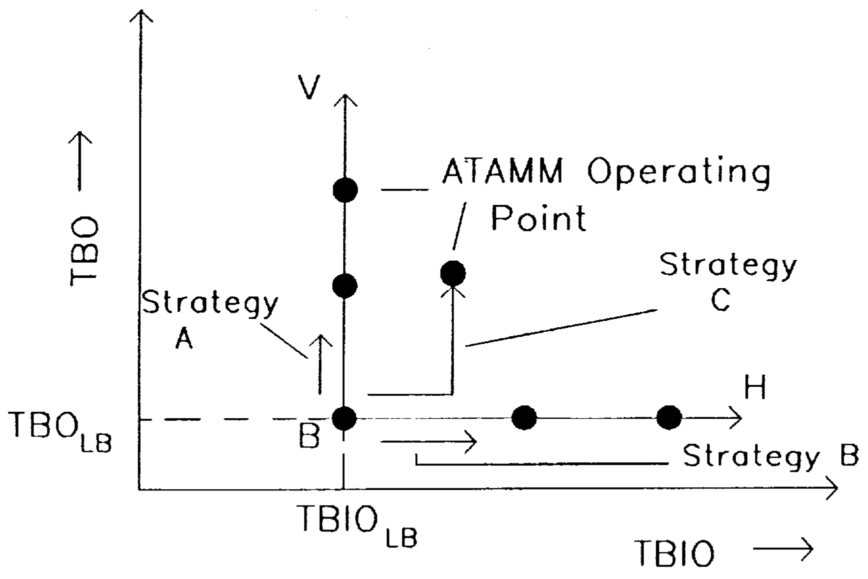


Figure 9. ATAMM performance plane.

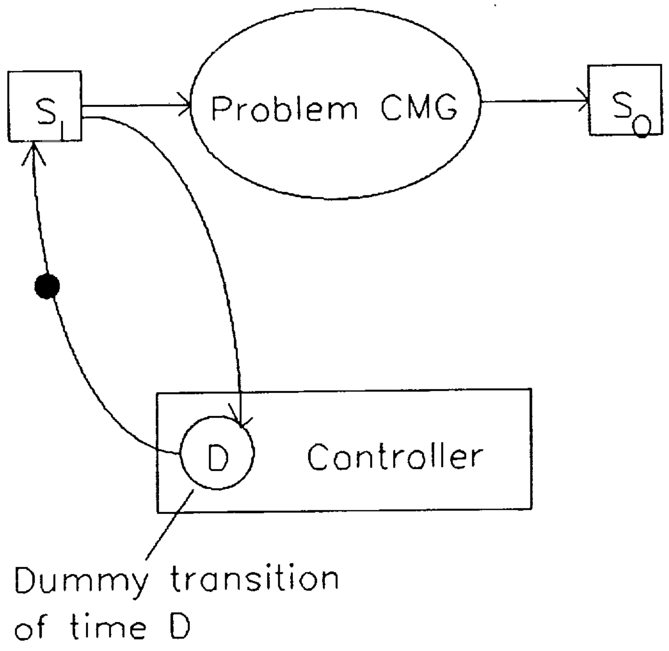


Figure 10. Injection control implementation.

procedures, it is possible to shift the operating point vertically along line B-V. This strategy preserves TBIO while degrading throughput performance. Such a strategy is useful for real-time control and signal processing applications where maintaining high computing speed is very important. Operating points on line B-V for lower resource requirements are calculated from the TGP diagram by increasing TBO until the number of active nodes in any time interval decreases by one from the previous operating point. These operating points are implemented by adjusting the minimum input injection control interval. As an example, consider the AMG shown in Figure 5. Operation at TBIO = 7 and TBO = 2 requires 7 resources. By increasing TBO to 3, the number of required resources decreases to 5. This can be observed by increasing the value of TBO in the TGP diagram of Figure 8 until the number of concurrently active nodes decreases. Increasing TBO to 5 further reduces the resource requirement to 4 resources. These operating points are displayed in the ATAMM performance plane shown in Figure 11.

It is also possible to shift the operating point horizontally along the line B-H to reduce resource requirements. This strategy preserves TBO while degrading computing speed performance. Such a strategy is useful for number crunching applications where maintaining throughput is important. Operating points on line B-H for lower resource requirements are

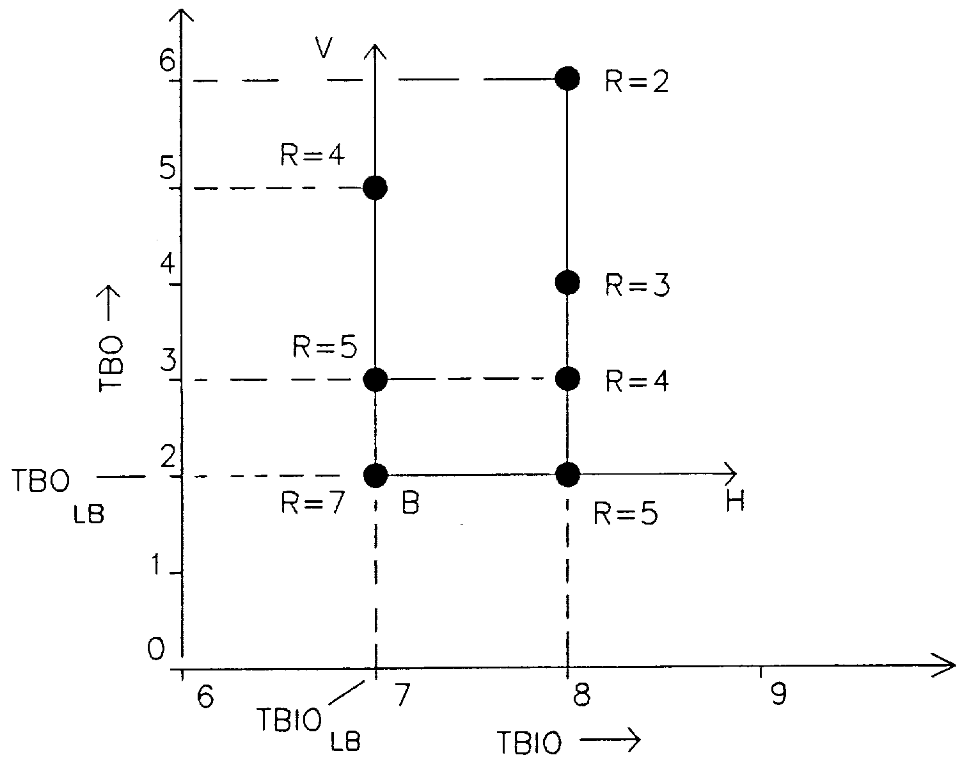


Figure 11. ATAMM performance plane for Figures 5 and 12.

obtained by adding control edges to the original AMG. A control edge is an AMG place which imposes a precedence relation among two transitions, but does not imply data dependency. When such an edge is added to an AMG so that the longest directed path from the input source to the output sink is increased, the resulting new graph has an increased TBIO value but still describes the same algorithm.

The addition of a control edge can create new directed circuits having increased time per token values so that TBO is also increased. This potential problem is avoided by adding dummy nodes to the AMG. A dummy node is an AMG transition which implements an identity operation and requires zero computation time. The dummy node serves as a buffer to provide additional storage for the output data of a graph node. Implementation of a dummy node is a memory operation and thus does not require a resource. Using the dummy node, it is possible to increase the token count on circuits formed by adding control edges, thus preserving the value of TBO in the original graph. Control edges and dummy nodes also can be used to improve performance bounds and to balance resource requirements. Operating point design using control edges and dummy nodes is explained in more detail in [3].

To illustrate shifting the operating point horizontally, consider again the AMG shown in Figure 5. Adding a control edge directed from node 3 to node 4 creates a new directed path from input source to output sink which contains nodes (1, 3, 4, 6, 7). Therefore, $TBIO_{LB}$ for the new graph is equal to 8. However, the control edge also creates a new directed circuit containing the read, process and write transitions of nodes 1 and 3, and the read transition of node 4. This directed circuit has a time per token value of 3 so that TBO_{LB} is increased to 3. The time per token value of this circuit is reduced by adding a dummy node to the edge directed from node 1 to node 4. The new AMG and corresponding CMG are shown in Figures 12 and 13. A second control edge and dummy node are also added in this figure for the purpose of reducing the peak resource requirement. The SGP diagram and the TGP diagram for TBO = 2 are shown in Figure 14. The new operating point having TBIO = 8, TBO = 2, and R = 5 is shown on the performance plane diagram in Figure 11. Also shown are additional operating points on the constant TBIO = 8 line which are implemented by injection control as described previously.

The performance plane diagram provides information essential for the selection and control of the time performance of algorithms executing under ATAMM rules. Operating points are selected by identifying R points in the performance plane, one point corresponding to each resource number. The point associated with a specific value of R identifies the value of TBIO and TBO

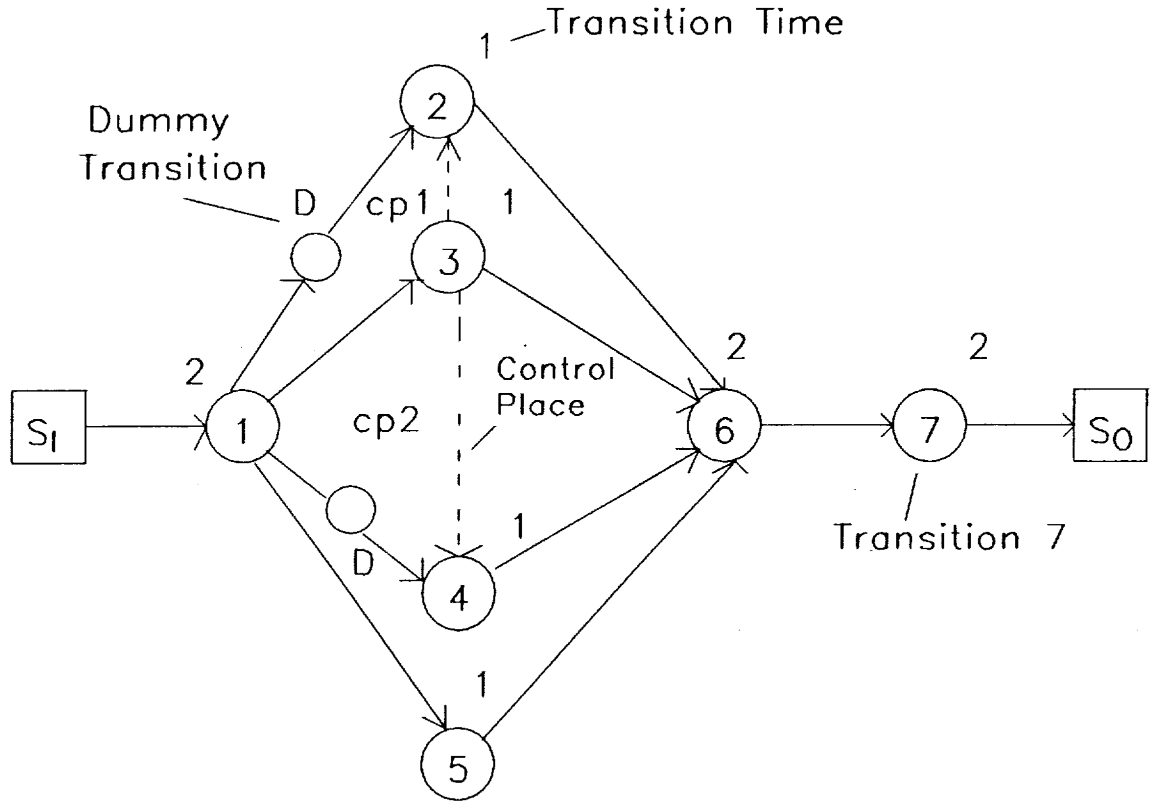


Figure 12. Modified AMG for Figure 5.

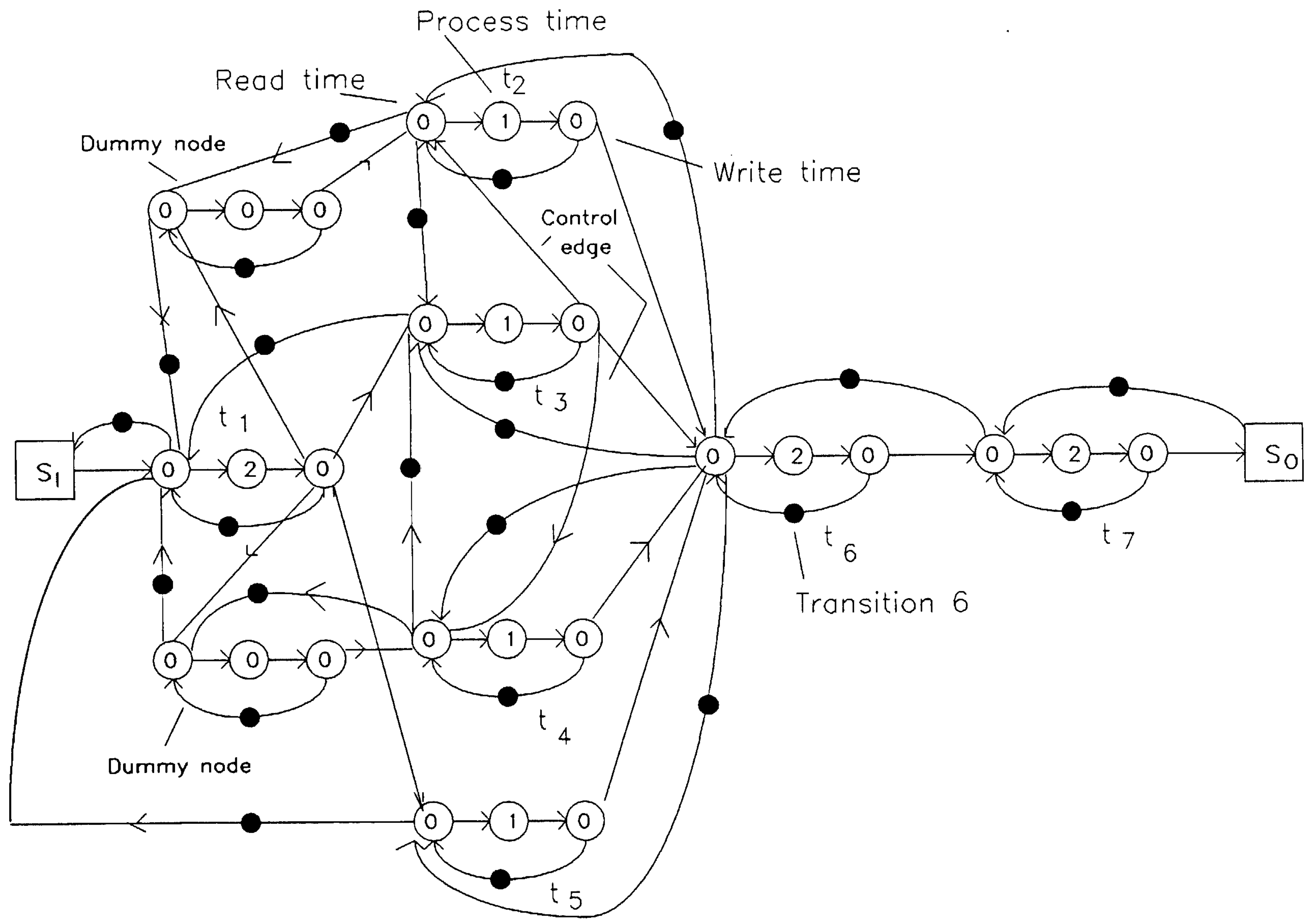
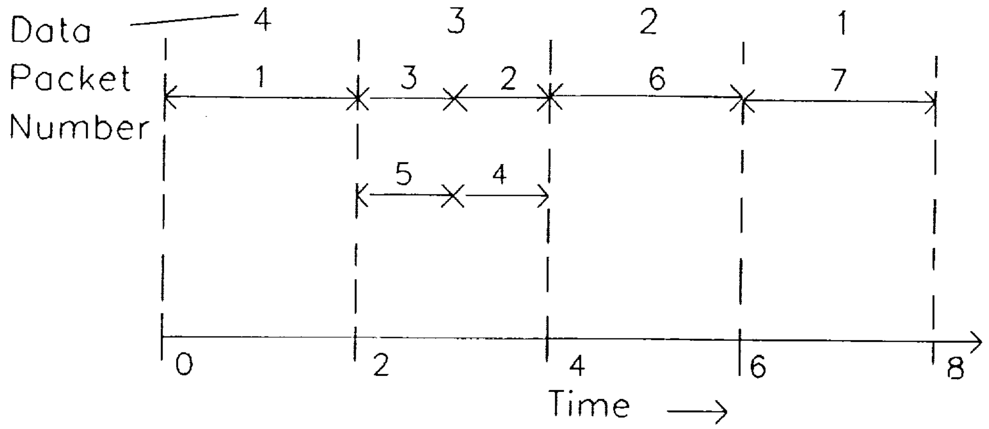
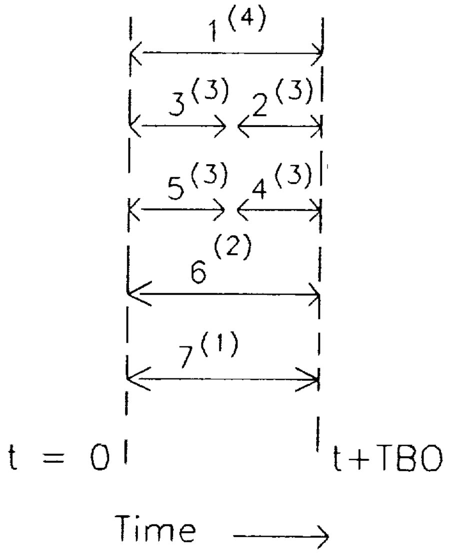


Figure 13. Modified CMG for Figure 12.



(a)



(b)

Figure 14. (a) SGP diagram and (b) TGP diagram for Figure 12 with TBO = 2.

implemented when the system is functioning with R resources. If the number of resources changes, then a new operating point is identified. Operation at the new point is realized by modifying the graph with control edges and dummy nodes, and adjusting the input injection interval. The calculation of performance bounds and the construction of the SGP, TGP and performance plane diagrams have been automated in a software package called the ATAMM Design Tool. The software is constructed to operate on IBM PC/AT compatible computers in a Microsoft Windows environment.

IV. AMOS IMPLEMENTATION IN THE ADM

In this section, the adaptation of the ATAMM model to the ADM system is described. The architecture of the ADM system is discussed first. Next, the major components of the ATAMM Multicomputer Operating System (AMOS) are identified and AMOS operation is explained using a state diagram description. Then, system capabilities of the ADM system running under AMOS are described. Included is a description of the strategy used to implement triple modular redundancy (TMR).

IV.1. ADM Description

A VHSIC ATAMM data flow architecture, called the Advanced Development Module (ADM), is under development and will be completed prior to the start of the GVSC implementation. The ADM architecture is being constructed by Westinghouse Electric Corporation and is shown in Figure 15. This system consists of four identical VHSIC 1750A processors which communicate over a dual PI-bus. Also connected to the PI-bus is a 1553B communication module which serves as a gateway for input and output data flow from an IBM PC/AT. Communications over the PI-bus are accomplished by broadcasting and use of direct memory access. All processors also communicate over an IEEE-488 bus to a Microvax computer which is used to download application programs and files for debugging activities. The 1553B module is connected to the IBM PC/AT by a single line communication link. Data are transferred between the 1553B module and the IBM PC/AT by synchronous communications. In addition to input and output, this link is used for fault injection, fault recovery, modification of the algorithm graph in real-time, and passing information back to IBM PC/AT for testing purposes. The 1553B acts as a source and sink for the algorithm graph and thus is capable of controlling the input injection rate to the 1750A processors and collecting output from the PI-bus.

IV.2. AMOS Description

The ATAMM Multicomputer Operating System (AMOS) is the operating system of the ADM hardware and its operation is based upon ATAMM rules. It consists of three logical components, the

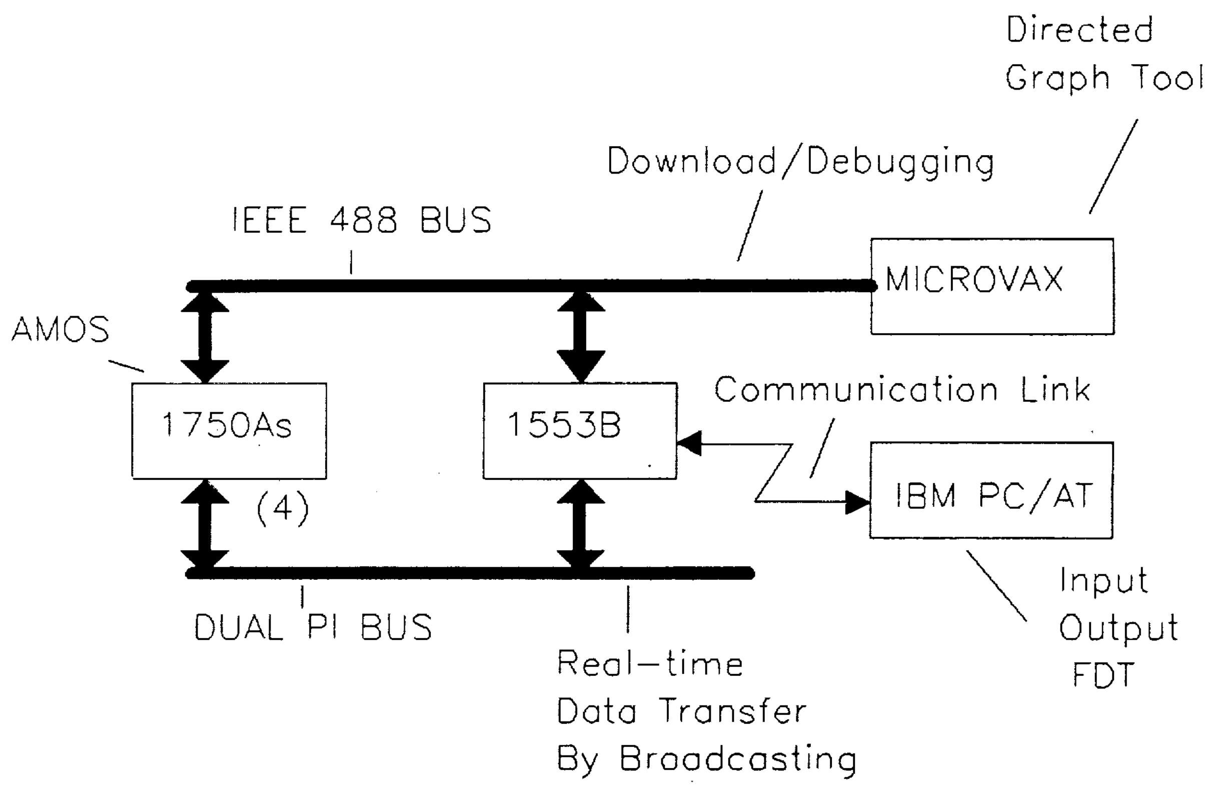


Figure 15. Advanced Development Module (ADM) System.

graph manager, the global memory, and a set of functional units or resources. The graph manager updates and monitors the status of the computational marked graph. When a read transition of this graph is enabled, the graph manager assigns a functional unit from the queue of available functional units to perform the corresponding algorithm operation. Each of the 1750A processors is a functional unit of the operating system. At the completion of the computation, a message is sent from the functional unit to the graph manager to update the computational marked graph. Therefore, the graph manager requires a communication path to each functional unit. The global memory stores data corresponding to input and output signals for each algorithm operation of the algorithm marked graph. Thus, this unit also requires a communication path to each functional unit. The functional unit is the logical component which executes all three node marked graph (NMG) transitions of each algorithm operation. It is assumed that each functional unit has the code required to perform the assigned algorithm operation. The functional unit communicates with the graph manager to update the status of the CMG, and with the global memory to read and write data. The communications between graph manager, global memory, and functional units are asynchronous and are carried out by direct memory access transfer over the PI-bus. In order to ensure that all functional units have an identical copy of the graph data structure, a functional unit grabs the PI-bus before changing the graph data structure. The updated graph data structure is transmitted to all functional units by a broadcast, and only then the functional unit releases the PI-bus for other communication.

The graph manager and global memory are distributed among all the functional units. However, only the graph manager residing in the functional unit on top of the queue of available functional units is active at a given time. This distribution of activities has the advantage of increasing the number of functional units in the system and at the same time improving the potential for achieving a higher degree of fault tolerance to processor failure. Also, a distributed global memory eliminates the need for shared memory among functional units. The major components of AMOS are shown pictorally in Figure 16.

The operation of AMOS is represented by the state diagram shown in Figure 17. Initially, all functional units awake in the state labeled Idle. A functional unit remains in this state until its identifier appears at the top of the resource queue. When this occurs, the functional unit undergoes a state transition to the Examine Graph state. In this state, the functional unit actively monitors the status of the CMG until a read transition for an algorithm operation becomes enabled. When an enabled read node is identified, the functional unit assigns itself to perform the algorithm operation, grabs the PI-bus, and

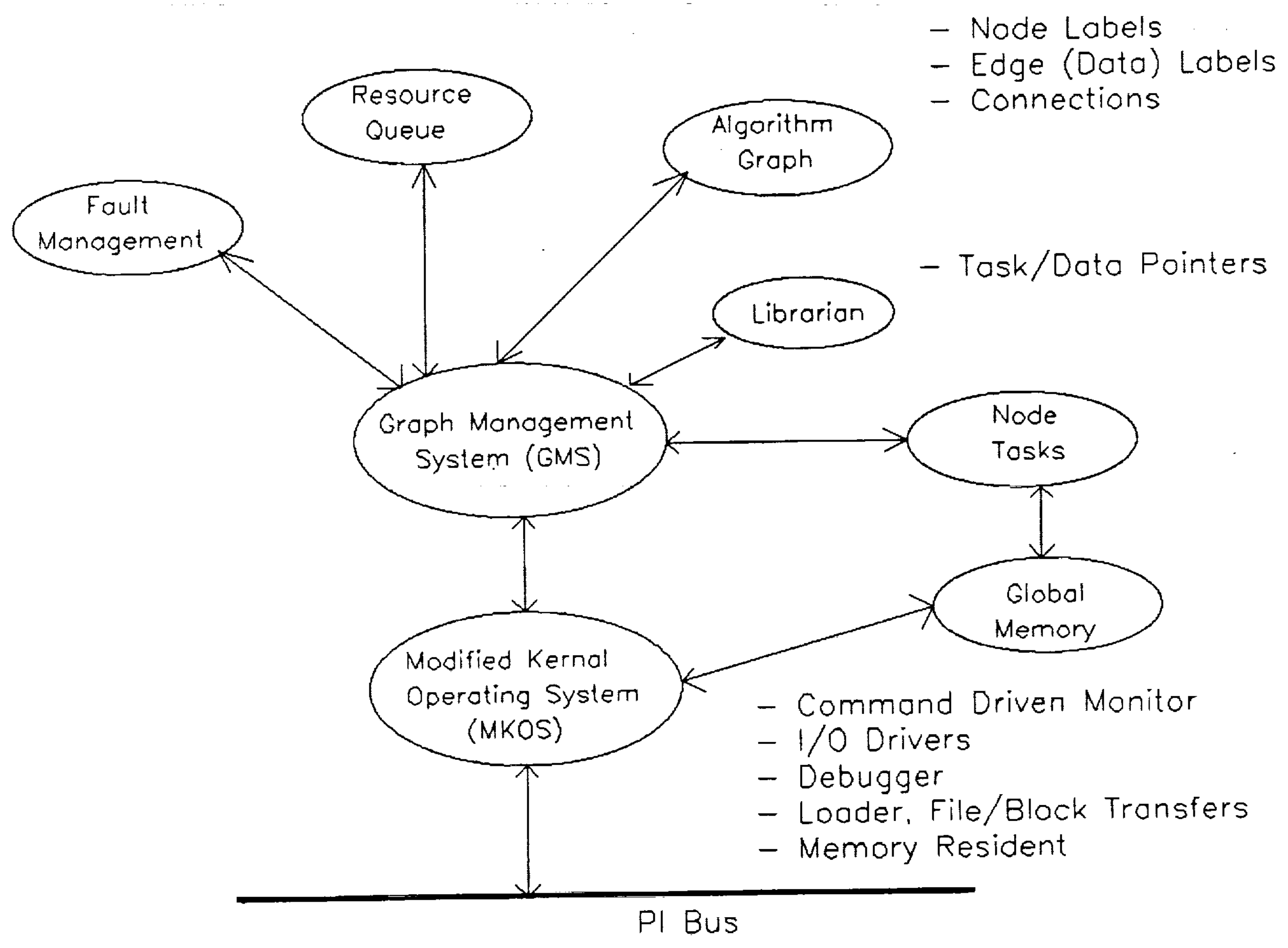


Figure 16. Major components of the ATAMM Multicomputer Operating System (AMOS).

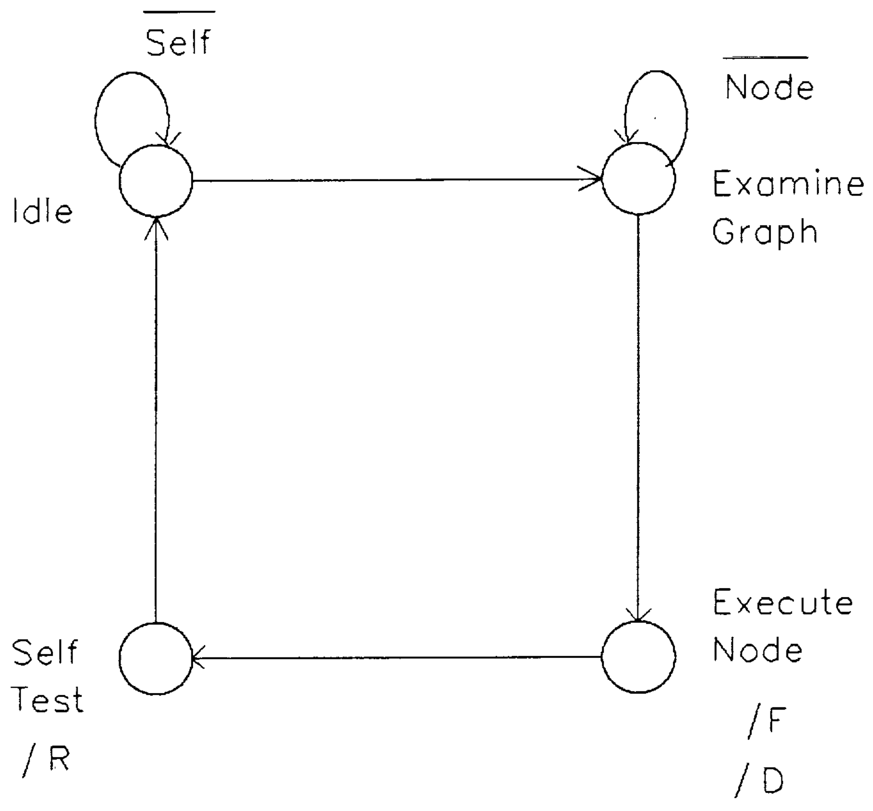


Figure 17. AMOS functional unit state diagram.

undergoes another state transition to the Execute state. During this state transition, the functional unit identifier is removed from the top of the resource queue and a bus communication "F" announcing that an algorithm operation has been initiated is broadcast on the bus. Then, the functional unit releases the PI-bus. The functional unit remains in the Execute state until the algorithm operation is complete. At the completion of the algorithm operation, the functional unit grabs the PI-bus and initiates a second bus communication "D" which includes a broadcast of the algorithm operation output data to all other functional unit global memories. At this time the functional unit again changes state to the Self Test state and releases the PI-bus. The Self Test state corresponds to a diagnostic check of the functional unit. After a successful self test, the functional unit returns to the initial Idle state. This state transition is accompanied by a grabbing of the PI-bus, a third bus broadcast communication "R" announcing that the functional unit identifier should be returned to the bottom of the resource queue, and a release of the PI-bus. While in any state, the CMG and resource queue in the global memory of a functional unit can be updated by F, D, or R commands from other functional units.

IV.3. System Capabilities

The capabilities of the ADM now are described briefly. The ADM is capable of implementing one algorithm graph at a time. The graph has a single input source and a single output sink. Graph play is assumed to be periodic, and the time between inputs is controlled by the 1553B module. Algorithms can be implemented either in simplex, duplex or TMR form. The operating system can correct a computational error in a functional unit by the Triple Modular Redundancy (TMR) method in which every algorithm operation is performed by three functional units and the result is selected by voting. AMOS can recover from a single fault in which a functional unit detects an error in the self-test state. It is also possible to modify graph structure and input injection period in real-time based upon knowledge of the number of functional units. These features of fault tolerance and operating point modification can be tested by injecting faults or changing the number of functional units in real-time.

It is possible to implement TMR in a number of different ways. However, in view of the graph theoretic basis for the ATAMM model, the TMR strategy is considered in a purely graph context. The approach used is to triplicate each node and each edge of the AMG. The elements of each triplicated graph component are identified by a color extension, red (R), blue (B) and green (G). The triplication of an AMG node pair and the associated edges is shown in Figure 18. Each of the colored nodes receives each input label triplicated and coded as R, B and G. After all data are acquired, a vote is conducted and the majority accepted data are used as the node process input. The three colored nodes of a triplicated AMG node are enabled simultaneously. Given the availability of three functional units to fire these enabled nodes, the triplicated processing occurs in parallel and produces triplicated output data. The requirement that all elements of a triplicated node fire simultaneously preserves the node process timing in accordance with the simplex model.

V. AMOS ENHANCEMENT FOR THE GVSC

The focus of present research is to extend the capabilities of the ATAMM Multicomputer Operating System, and thus expand the class of problems to which the ATAMM rules can be applied. The enhanced AMOS is to be implemented in the Generic VHSIC Spaceborne Computer (GVSC), a spaceborne four processor breadboard which is also based on the VHSIC 1750A instruction set architecture. The GVSC architecture and associated consoles are shown in Figure 19. Because of the similarity of this hardware to the ADM system, the structure and features of the present version of AMOS, as presented in Section IV, will be retained. In addition, several new features will be added. In this

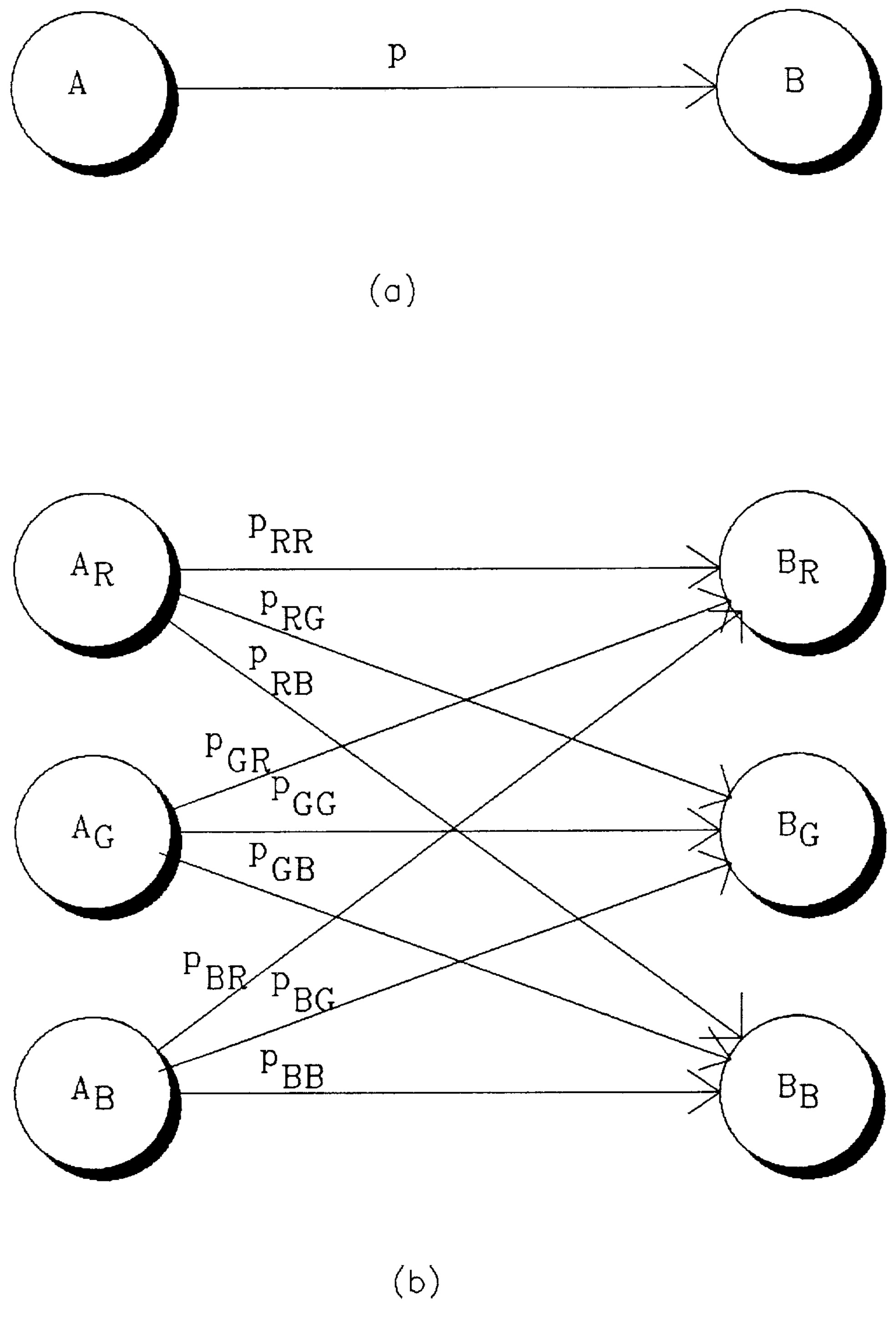


Figure 18. (a) Simplex and (b) TMR AMG node pair representation.

MULTIPROCESSOR INTERFACE DIAGRAM

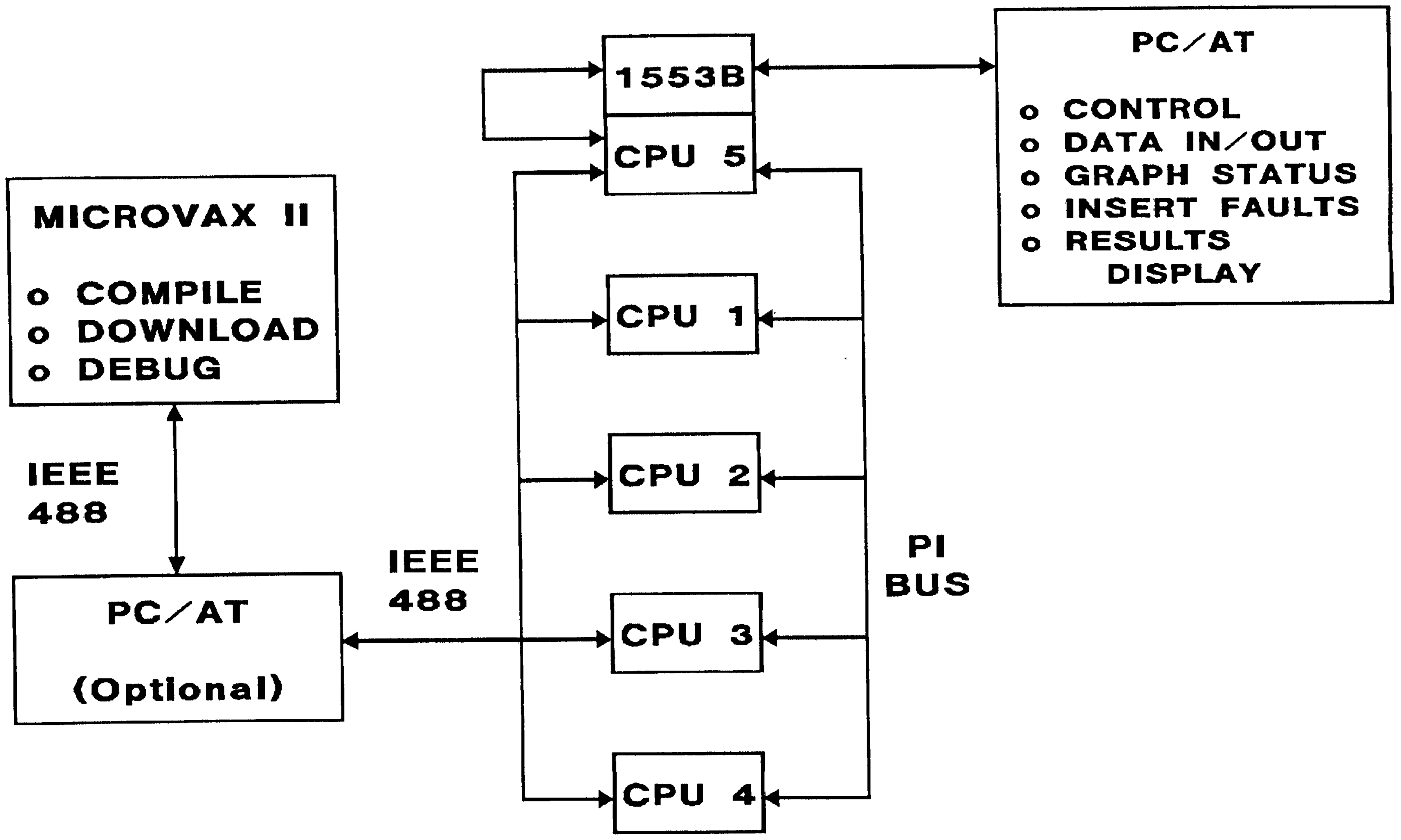


Figure 19. Generic VHSIC Spaceborne Computer (GVSC) System

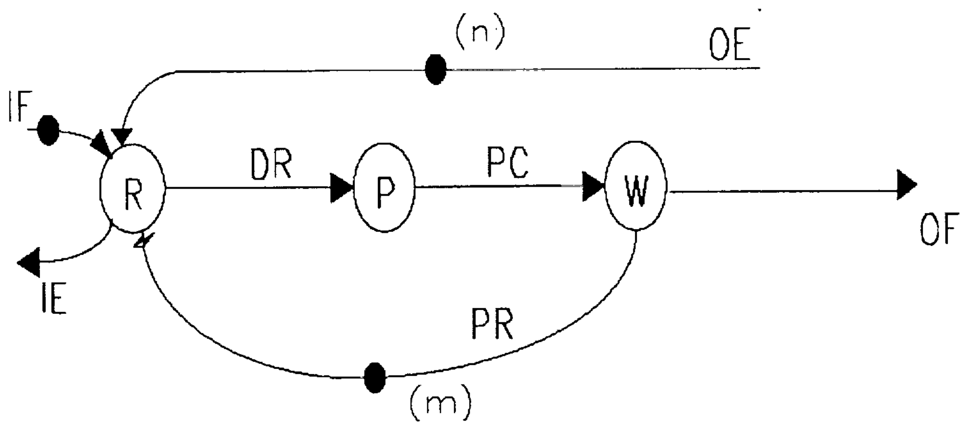
section, the proposed enhancements to AMOS are described. First, the ATAMM model is generalized to permit multiple concurrent instantiations of selected graph nodes. This enhancement to the ATAMM model is desirable because, in many algorithm decompositions, it provides an opportunity to improve throughput performance and resource utilization. Second, the simultaneous play of multiple algorithm graphs, each having a distinct source node and sink node, is considered. Three strategies for implementing multiple graphs are proposed. Then, a method for distributing the source and sink nodes in the functional units is presented. This enhancement is desirable to reduce the workload and complexity of the 1553B communication module. Finally, improvements to the fault tolerance capabilities of AMOS are described. Proposed are new methodologies for recovering from the failure of a functional unit during the complete AMOS operating envelope.

V.1. Multiple Concurrent Instantiations of Node Operations

The present version of the ATAMM model requires that all computing associated with a node operation must be completed before the node can be initiated another time. However, in decomposed algorithms where throughput is not limited by recursion circuits in the AMG, it is possible to improve throughput by allowing multiple concurrent instantiations of node operations. Therefore, it is proposed that the ATAMM model be enhanced to include multiple concurrent instantiations of AMG nodes. A strategy for this generalization of ATAMM and an illustrative example are presented in this section.

The generalization of ATAMM to include multiple concurrent instantiations of AMG nodes is accomplished by modification of the NMG. Multiple tokens, n tokens on the OE labeled edge and m tokens on the PR labeled edge, are placed as initial tokens in the NMG as shown in Figure 20. The m tokens on edge PR indicate that as many as m functional units can be assigned concurrently to perform the node operation. The n tokens on edge OE provide up to n memory locations in which output data from the node operation can be deposited. Since the model requires a vacant output data container as a prerequisite to node firing, the number n is chosen to be at least as large as the number m.

It can be shown that the modified ATAMM model, like the present model, is live, reachable and consistent. However, the enhanced NMG and CMG are no longer safe (1-bounded), but are n-bounded. For this reason, additional graph management is required to ensure that the model is deadlock free and that tokens belonging to different data packets are not mixed in order. To guarantee these essential properties, all graph tokens will be tagged to indicate an input data packet identity. The firing rule for each read node will be modified so that a read



NMG EDGE LABELS

IF	Input Buffer Full
IE	Input Buffer Empty
DR	Data Read
PC	Process Complete
PR	Process Ready
OE	Output Buffer Empty
OF	Output Buffer Full

Figure 20. The NMG for the enhanced ATAMM model.

node is enabled only when each incoming edge contains a token having the tag identifier for the next appropriate node instantiation. In this way, proper data sequencing is assured. The graph is made deadlock free by allowing concurrent processes for the same node operation to complete in any order. Procedures for managing the tagging of graph tokens are being developed and should not add significantly to the overhead of the operating system.

An example is presented to demonstrate the significance of this enhancement to the ATAMM model. Consider the AMG shown in Figure 21. If each AMG node operation must complete before being initiated a second time, then TBO for this graph is limited to 5 by the node labeled 2. However, if three concurrent instantiations of node 2 are allowed, then a TBO of 2 can be achieved. This throughput limitation results from the presence of the AMG recursion circuit consisting of nodes 4 and 5. In general, nodes which occur in recursion circuits can not be multiply instantiated because each node calculation depends on output data from the previous node in the recursion circuit. Therefore, in the enhanced ATAMM model, recursion circuits are the only theoretical limiting factor for throughput performance.

The SGP diagram and the TGP diagram for TBO = 2 are shown in Figure 22. The number of concurrent instantiations required of node 2 to achieve a TBO value of 2 is determined by counting the

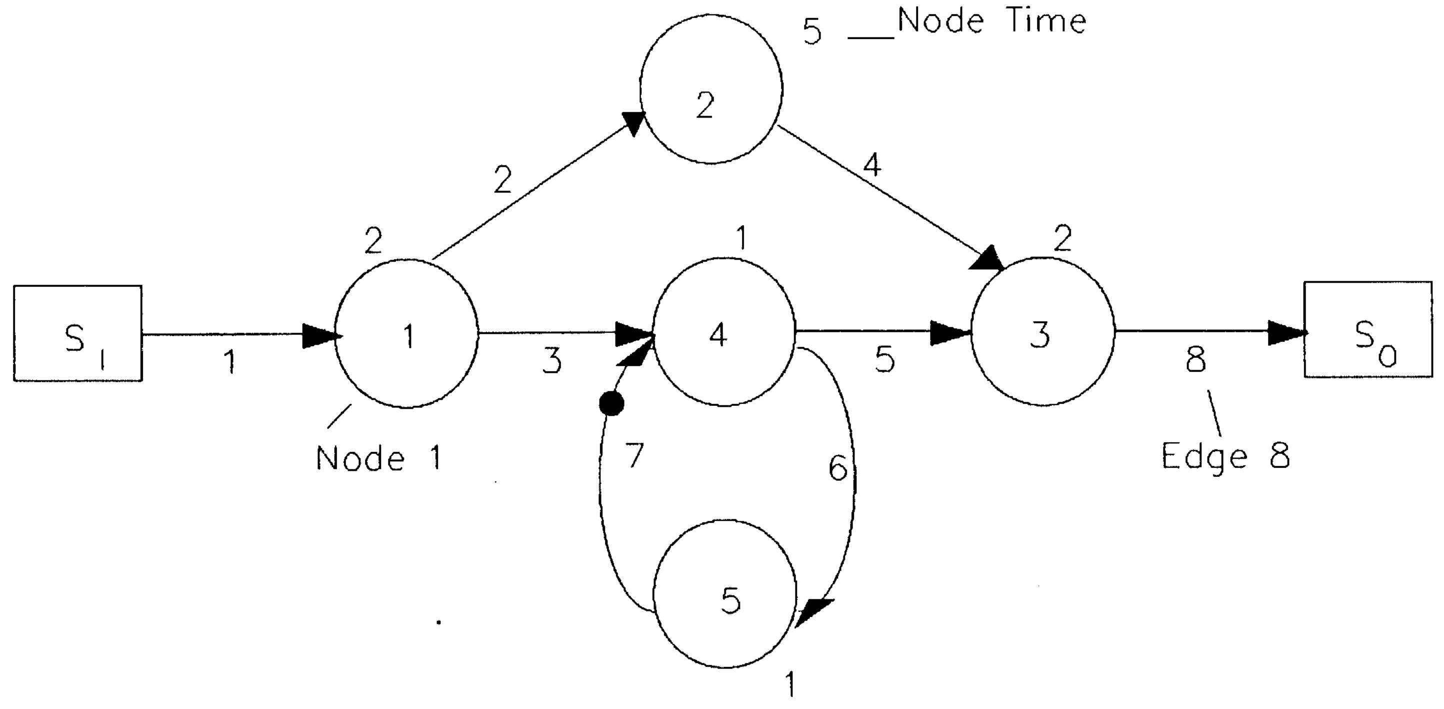
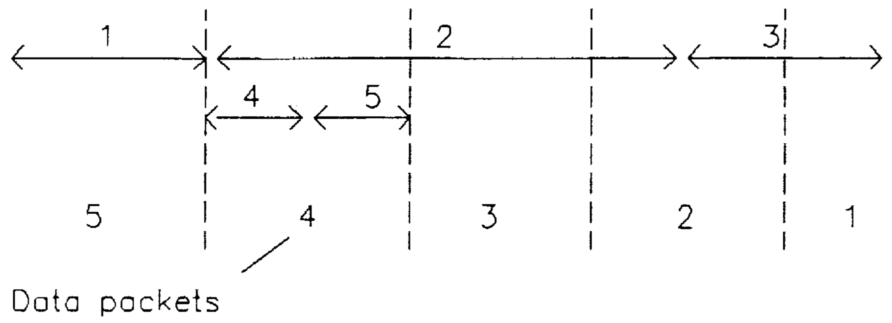
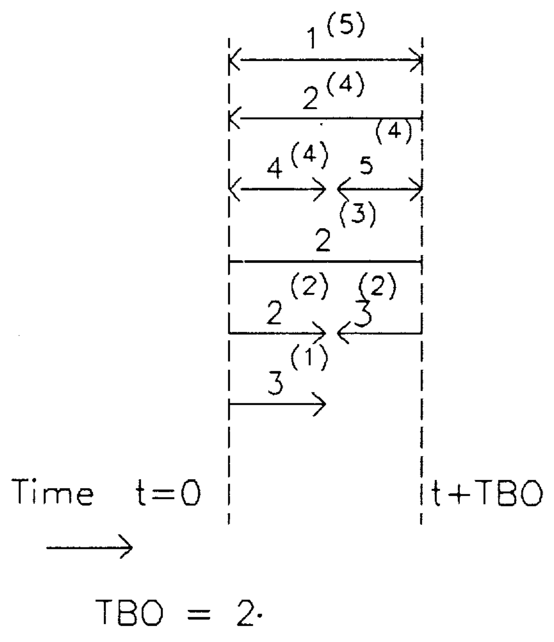


Figure 21. AMG example to illustrate the enhanced ATAMM model.



(a) Single Graph Play



(b) Total Graph Play

Figure 22. (a) SGP diagram (b) TGP diagram for TBO = 2 for the AMG of Figure 21.

number of times node 2 appears in the TGP diagram with different data packet numbers. The number of output data containers required at the output of each graph node also is calculated from this diagram by monitoring the predecessor-successor relations among the node operations. In this example, because nodes 2 and 4 are processing data packet 4 while node 3 is processing data packet 1, three data containers are required on the edge connecting nodes 2 and 3 and the edge connecting nodes 4 and 3. This information is used to place the correct number of initial tokens in the enhanced CMG, as shown in Figure 23.

A comparison of throughput performance and resource ultiization for the present ATAMM model and the enhanced ATAMM model is shown in Figure 24. In many problems, significantly better throughput performance and resource utilization are possible with the enhanced ATAMM model. As is the case with the initial ATAMM model, the design procedure to obtain predictable performance, including both TBO and TBIO, can be automated in a software design tool. Such a design tool is being developed by NASA Langley Research Center and Old Dominion University for the GVSC system.

V.2. Multiple Graph Strategies

In order to expand the class of problems which can be addressed in the ATAMM context, it is desirable to enhance the

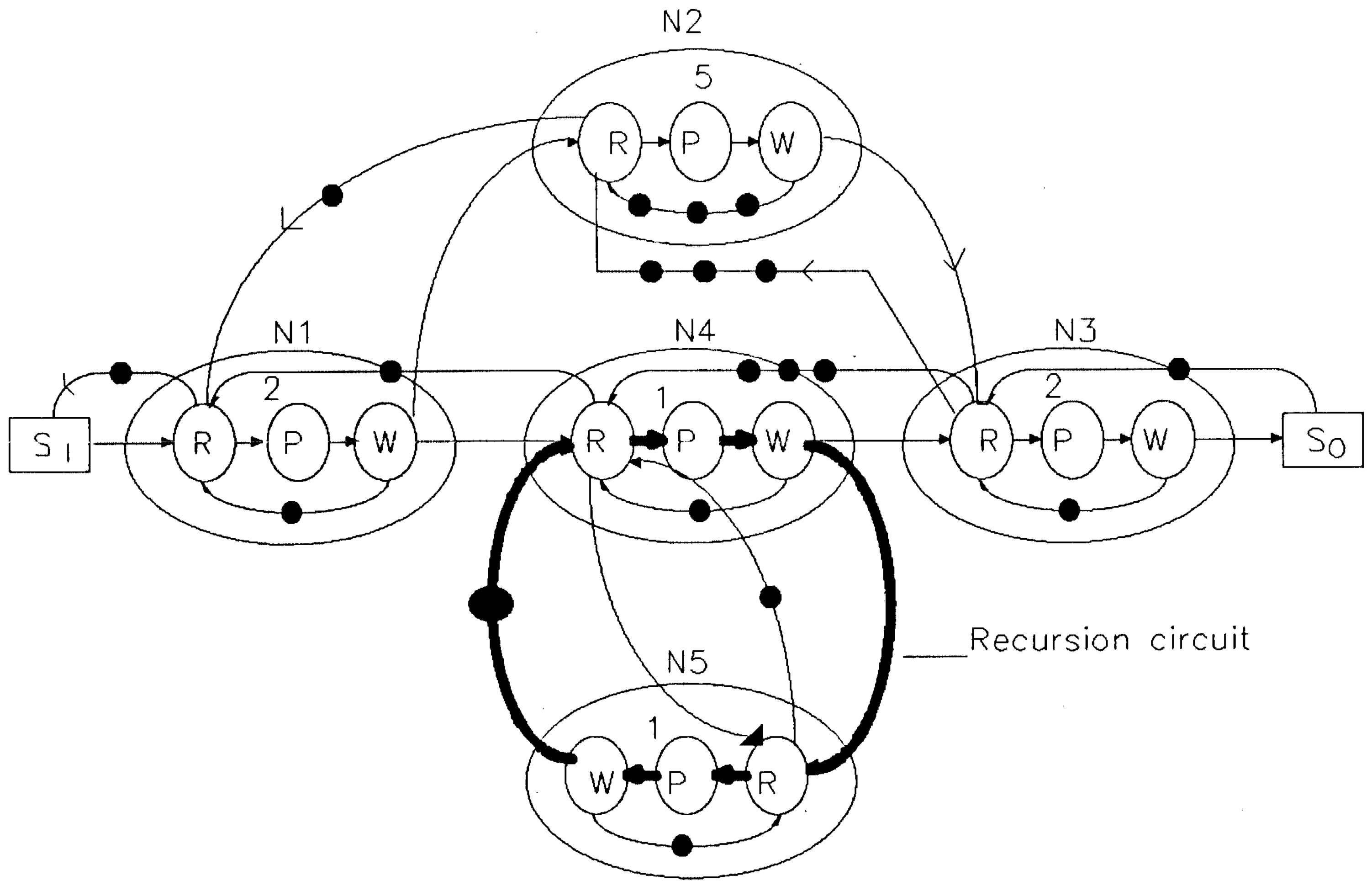


Figure 23. Enhanced CMG for the AMG of Figure 21.

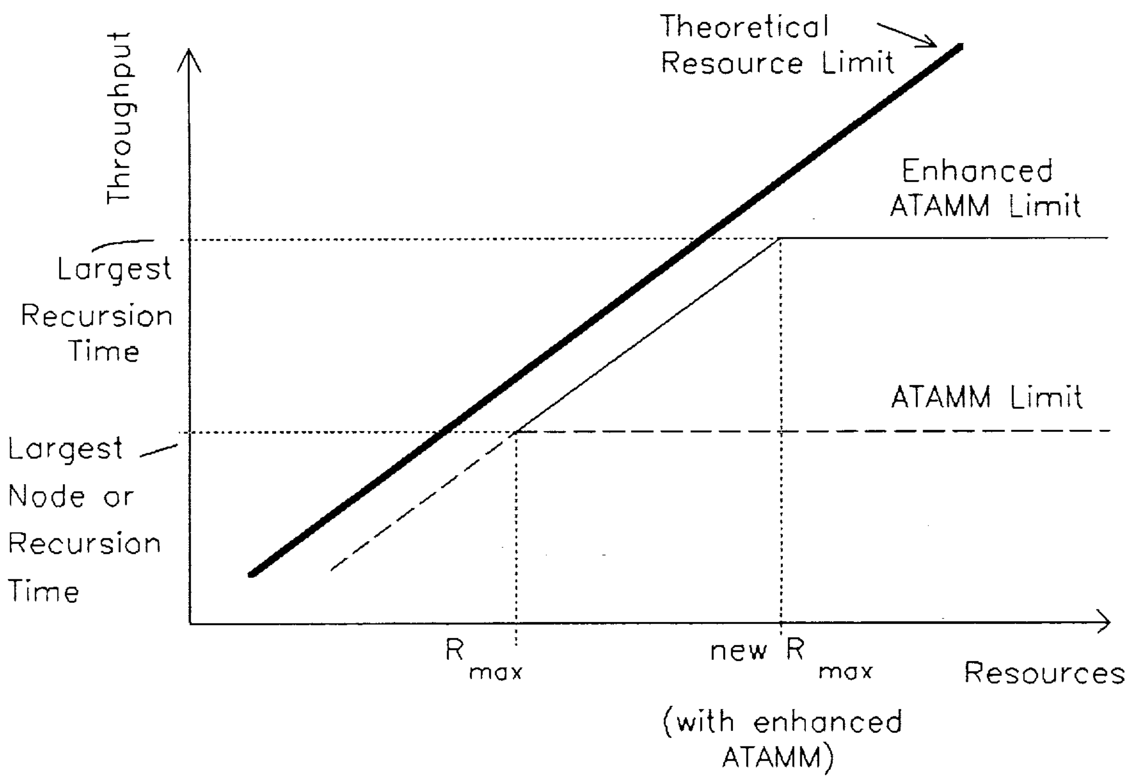


Figure 24. Ideal throughput improvement using the enhanced ATAMM model.

ATAMM model so that multiple graphs can be implemented. The purpose of this section is to propose several strategies for implementing the simultaneous play of multiple graphs under ATAMM rules. Three separate strategies for implementing multiple graphs are presented. These strategies are referred to as the parallel execution strategy, the time multiplexing strategy, and the priority interrupt strategy. The different strategies are selected to address different classes of problems which commonly arise in real-time applications. Equally important, the proposed strategies are selected to insure that graph-theoretic based analytical procedures for predicting time performance can be formulated, just as is done now for single graphs.

Parallel Execution Strategy

Let G denote a collection of algorithm graphs G_1, G_2, ... , G_n, each having its own input source and output sink, as shown in Figure 25. Each graph G_k has a unique computation time TBIO(k) and is played repetitively in period TBO(k), where it is assumed that these performance times are of the same order of magnitude. It also is assumed that the phase relationship between the various graphs is unpredictable and can not be accurately controlled. In the parallel execution strategy, all graphs are played in parallel. Just as in the single graph case, an operating point is determined for each graph G_k. The operating

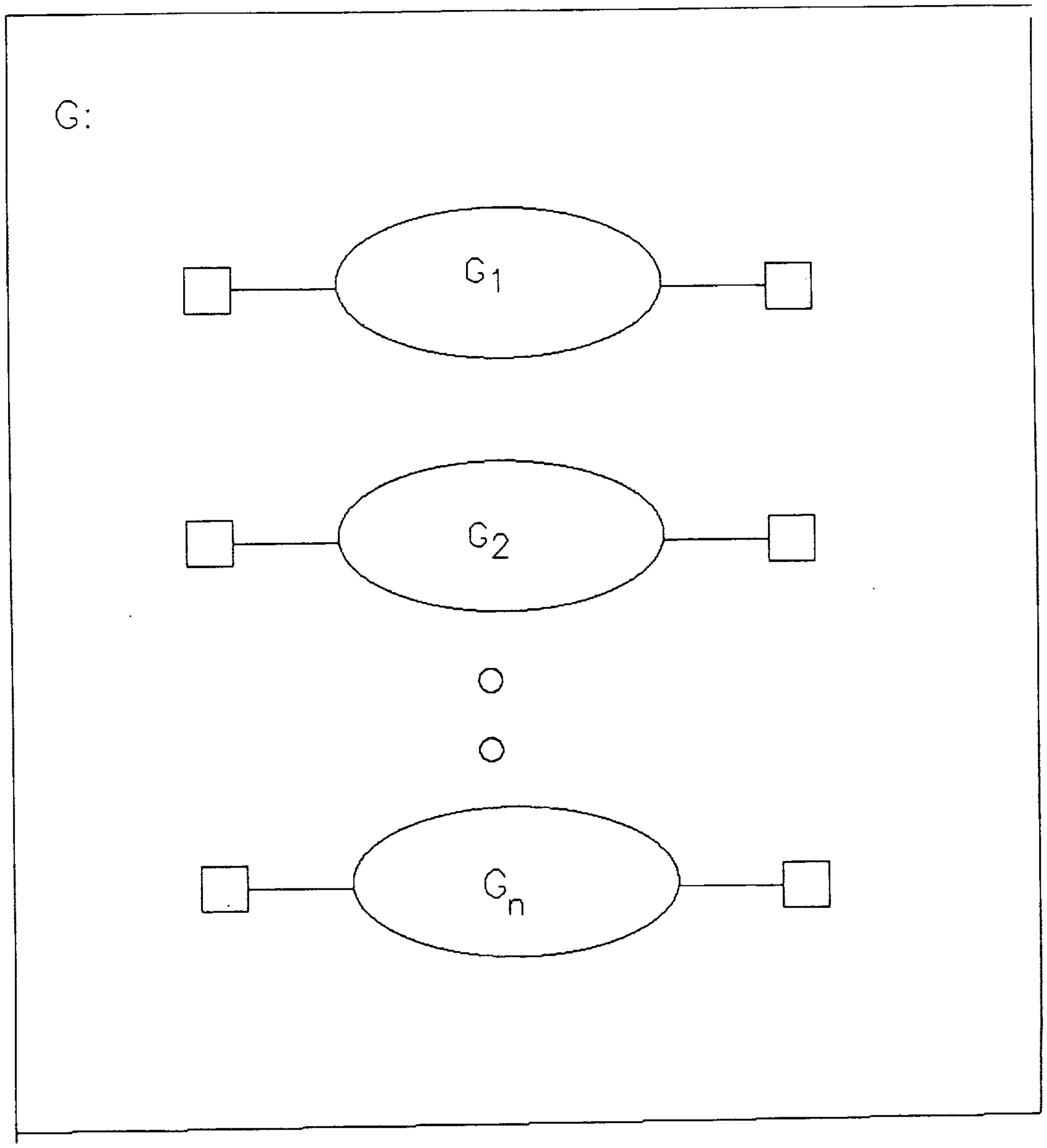


Figure 25. A collection of algorithm graphs (G).

point is characterized by the performance times TBIO(k) and TBO(k), and the minimum number of resources R(k) necessary to produce this performance. Since there is no knowledge concerning the phase relationships of the graphs, the total number of available resources required to insure that all graphs play at the desired operating point is the sum of the resources needed for each individual graph. A pictoral display of the parallel execution strategy is shown in Figure 26.

The parallel execution strategy is useful where it is necessary to play several graphs simultaneously while achieving a very high repetition rate for each graph. Such problems often occur in data processing and data reduction applications. For efficient use of resources, each graph operating point must be selected to avoid high peak resource requirements.

When operating with limited resources, the user is able select which graphs will operate with decreased time performance. It also is possible to suspend completely the play of graphs according to a priority ranking. A possible disadvantage of this strategy is that it can lead to low levels of resource utilization when one or more graphs have large peak resource requirements.

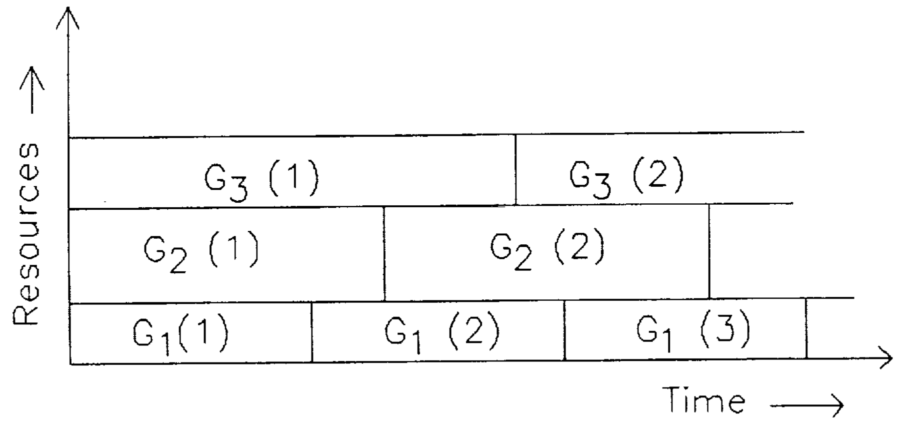


Figure 26. Parallel execution strategy for multiple graphs.

Time Multiplexing Strategy

Again let G be a collection of algorithm marked graphs G_1, G_2, ..., G_n, each having its own input source and output sink. It is assumed that each graph has a unique computation time TBIO(k), but that all graphs share a common repetition time TBO which is large compared to all computation times. It also is assumed that the time between inputs to the different graphs is known and can be controlled. In the time multiplexing strategy, system resources are time shared between the various graphs. An operating point is selected for each graph, thus specifying a resource requirement envelope. The input injection time for each graph is selected so that the individual graph resource requirement envelopes fit together like pieces of a puzzle and in summation at each instant total no more than the available number of resources. The time multiplex strategy is shown pictorially in Figure 27.

The time multiplexing strategy is useful when it is necessary to play several graphs with a common repetition period which is long relative to the computation time of each graph, and it is important to achieve short computation times. Such problems often occur in real-time control applications requiring the use of multiple sensors and actuators. This strategy naturally results in high utilization of system resources. As with the parallel execution strategy, the user is able to specify which

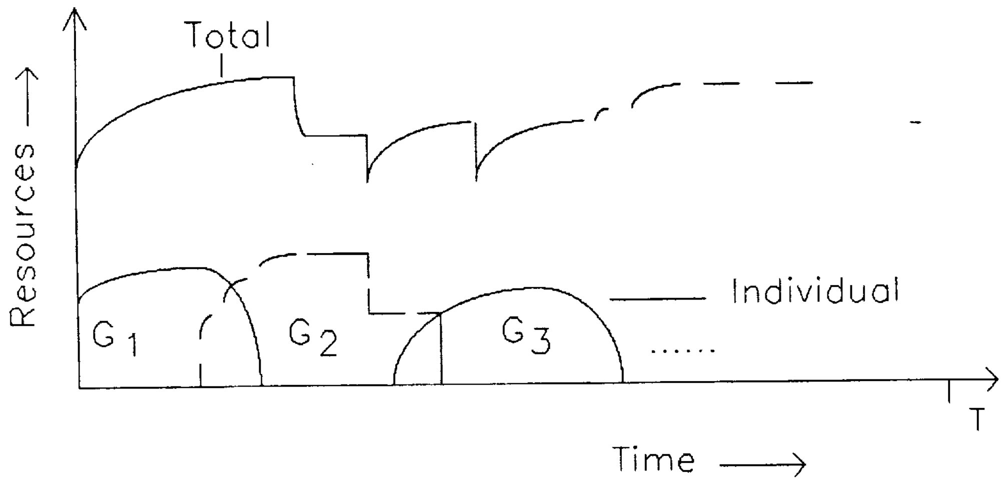


Figure 27. Time multiplexing strategy for multiple graphs.

graphs will degrade in performance as the number of system resources decreases.

Priority Interrupt Strategy

Let G denote a collection of n graphs playing on an ATAMM defined architecture according to the parallel execution strategy or the time multiplexing strategy. Let H denote a second collection of m graphs, also defined to play according to one of the two execution strategies, but which is inactive due to the absence of input data. In the priority interrupt strategy, it is assumed that input data to graph H occur only occasionally. When data become available, graph collection H is allocated highest priority for access to system resources, and graph collection G is terminated or allowed to play in a "resource available" status only. When input data to H are exhausted, system resources again are allocated to G, and play of graph G resumes. Therefore, this strategy is useful for addressing emergencies or unexpected high-priority computational tasks.

Time performance in all three operation strategies will be controlled as in the present ADM system. The operating system will be designed to continuously monitor the number of available system resources. Whenever the number of resources changes, a new operating point will be obtained from an operating point table. This table is constructed off-line by the user with the support of design and simulation software tools. A new operating point is achieved by reconfiguring the collection of graphs (setting control edges and dummy nodes), and by adjusting the injection rates for each graph. The feedback control process, shown in Figure 28, is to be integrated into the ATAMM Multicomputer Operating System (AMOS).

In the present ADM implementation of ATAMM, the single graph source node and the single graph sink node are realized in the 1553B Communication Module. It is proposed that in the GVSC implementation of ATAMM, the multiple source nodes and sink nodes be distributed over the set of functional units. This is done to limit the number of activities assigned to the 1553B Module, and to reduce the complexity of this component. In the GVSC, the 1553B module simply will pass data and performance information between the 1750A units and the PC/AT via the PI-bus. The 1553B also will time stamp performance information prior to transmission to the PC/AT. In the next section, a procedure for implementing the source and sink nodes in the functional units is described.

V.3. Distributed Sources and Sinks

Each of the concurrently processed multiple graphs has a distinct source node and a distinct sink node. The activities of a source node are to supply new input data at specified intervals

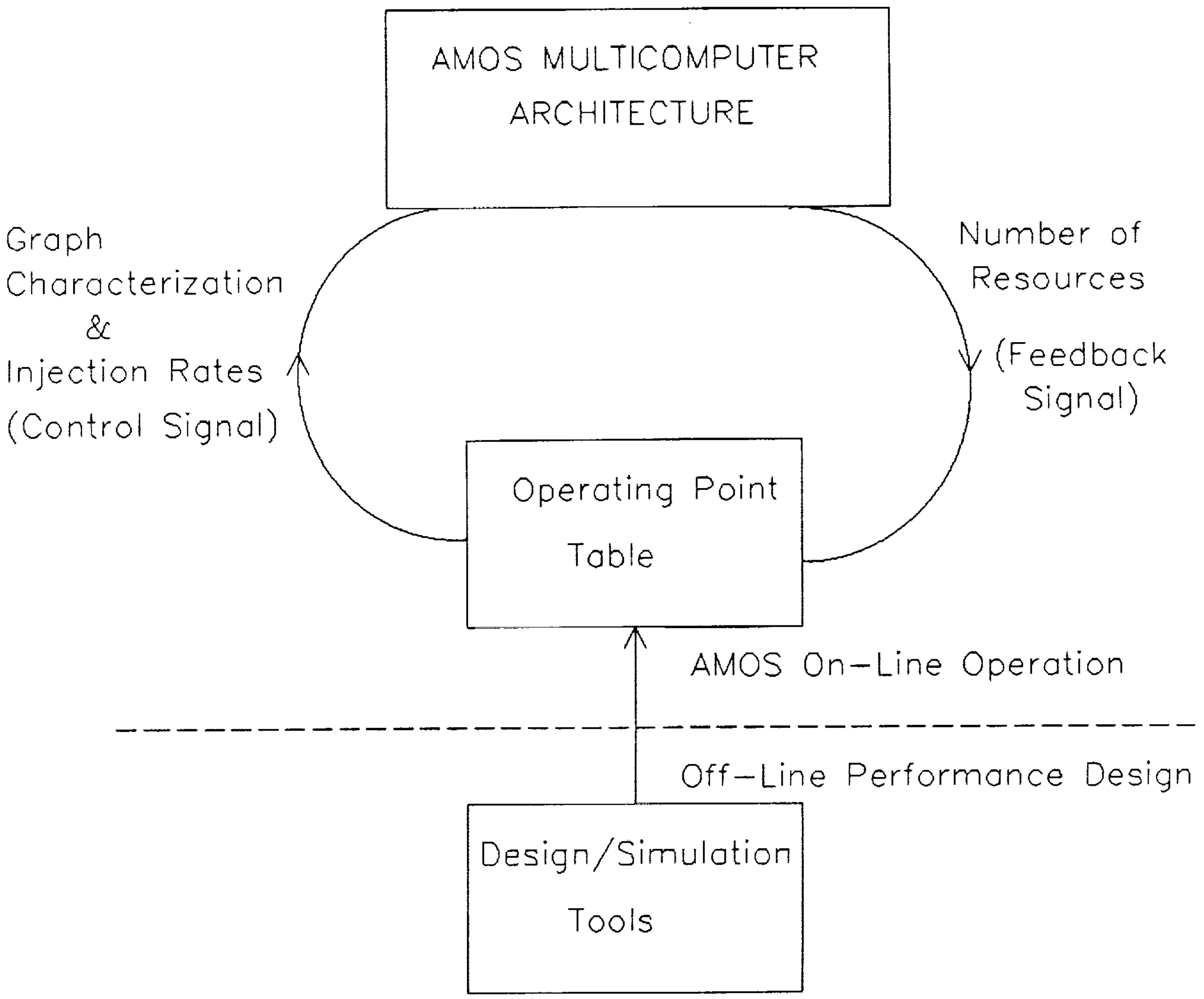

AMOS MULTICOMPUTER ARCHITECTURE
Graph Characterization & Injection Rates (Control Signal)
Number of Resources (Feedback Signal)
Operating Point Table
AMOS On-Line Operation
Off-Line Performance Design
Design/Simulation Tools

Figure 28. Control strategy for GVSC.

of time and to update the graph data structure to indicate the presence of new input data. The activities of a sink node are to collect output data for transmission to the PC/AT, and to update the graph data structure to indicate that a new output can be generated. In order to simplify the 1553B code and to reduce the amount of activity required of the communication module, it is felt that the implementation of sources and sinks should be done in the 1750A processors. It is proposed that input data for each graph be broadcast to all the 1750A processors. The functional unit at the top of the resource queue examines the graph for firable sources as well as firable nodes. Whenever a functional unit completes the processing of a node, it also checks for firable sinks. In this way, the source and sink activities are distributed over the set of functional units.

This new strategy is implemented by modifying the Examine Graph and Execute states of the AMOS functional unit state diagram as shown in Figure 16. The modified Examine and Execute states are described with the help of flow charts shown in Figure 29. As the functional unit acting as the graph manager enters the Examine state, it searches for firable sources first. If there is no firable source, the functional unit searches for the highest priority firable node. If there is no such node, it repeats the search for a firable source. If a firable node is found, the functional unit grabs the PI-bus and enters the

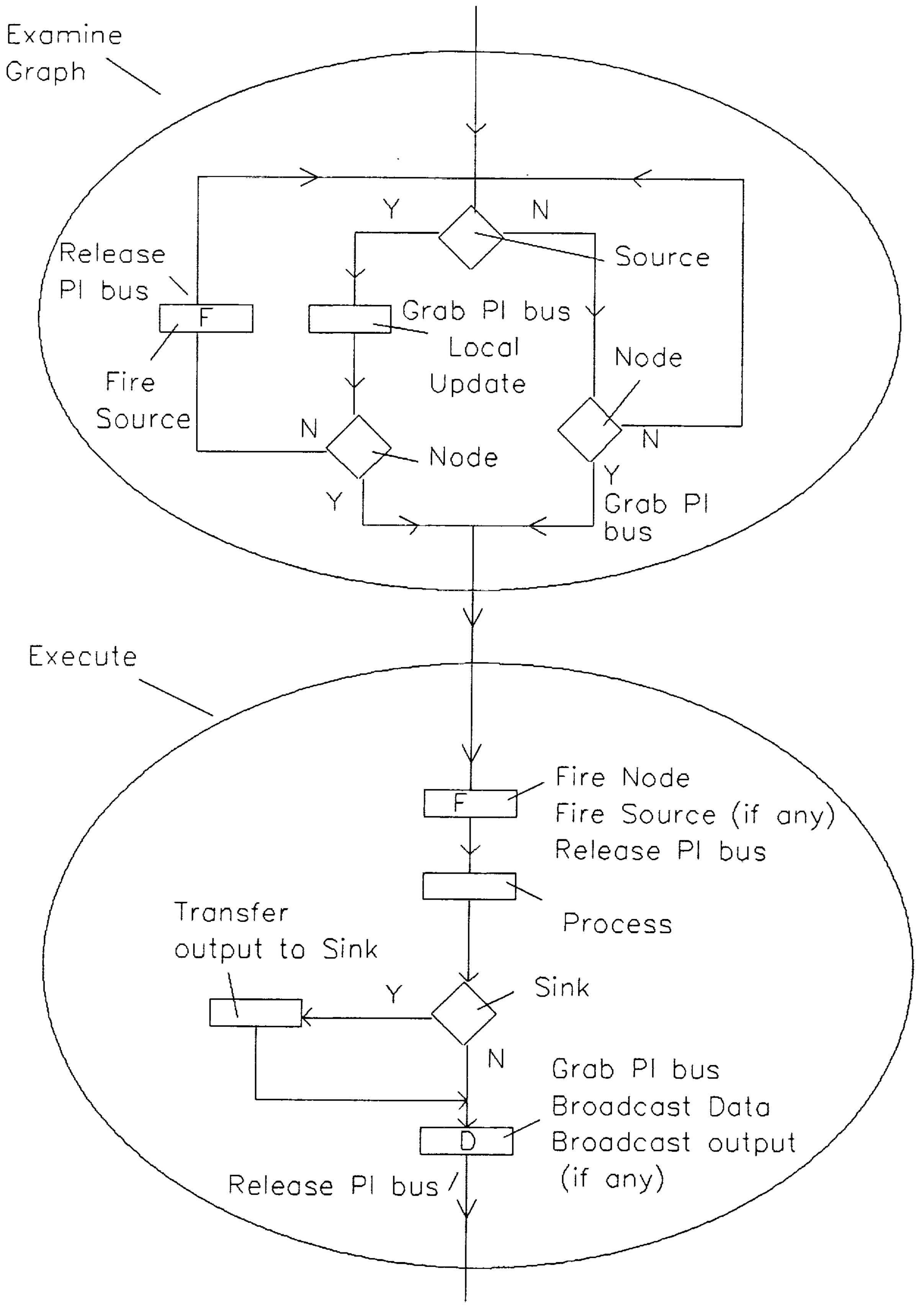


Figure 29. Modified Examine and Execute states for GVSC.

Execute state. If a firable source is found, the functional unit grabs the PI-bus, updates its copy of the graph data structure to indicate firing of the source, and then looks for a firable node. If a firable node is found, the functional unit enters the Execute state. Otherwise, an "F" command is issued by broadcasting the updated graph data structure, the PI-bus is released, and the functional unit returns to the beginning of the Examine state. The change in the Execute state to provide sink activity is to require that the functional unit search for firable sinks at the completion of each node process. If a sink can be fired, the sink output also is broadcast as part of the "D" command. The broadcast also is used to update the graph data structure to indicate that a new output can be deposited in the sink.

V.4. Fault Tolerant Strategies

The ADM system is capable of dealing with two types of faults, a fault which occurs during the processing of a node and a fault which occurs during the self-test of a resource. The first is detected and its effects discarded by the use of TMR. The second is detected in the Diagnostic State of a resource and the resource purges itself from the system. The present version of AMOS is to be enhanced so that the system is able to recover from the death of a functional unit in each of the four AMOS states. The purpose of this section is to propose a strategy to detect and recover from this type of system fault. The death of a resource occurs when the resource stops executing instructions in any of the states of AMOS (e.g., due to internal fault detection or total power failure) and does not generate a PI-bus broadcast. It is assumed that the resource does not fail while in possession of the communication bus, since it is not clear whether the bus will be released or maintained in the grabbed state after the fault. The overall view of the technique is divided into the following phases: fault detection, damage assessment, system recovery and system back to service, as shown in Figure 30. A brief explanation of each of these phases follows.

Fault Detection

One of the goals of effective fault detection in this context is to detect a fault in a time less than or equal to the TBO at which the system is operating. Due to the nature of real-time systems, it is important to detect a fault in the shortest period of time to avoid any significant loss of system performance. This fault detection can be achieved by monitoring the graph execution. By using the graph status, it is possible to detect a fault before the system has stopped completely due to the consequences of the fault. A strategy is being developed to

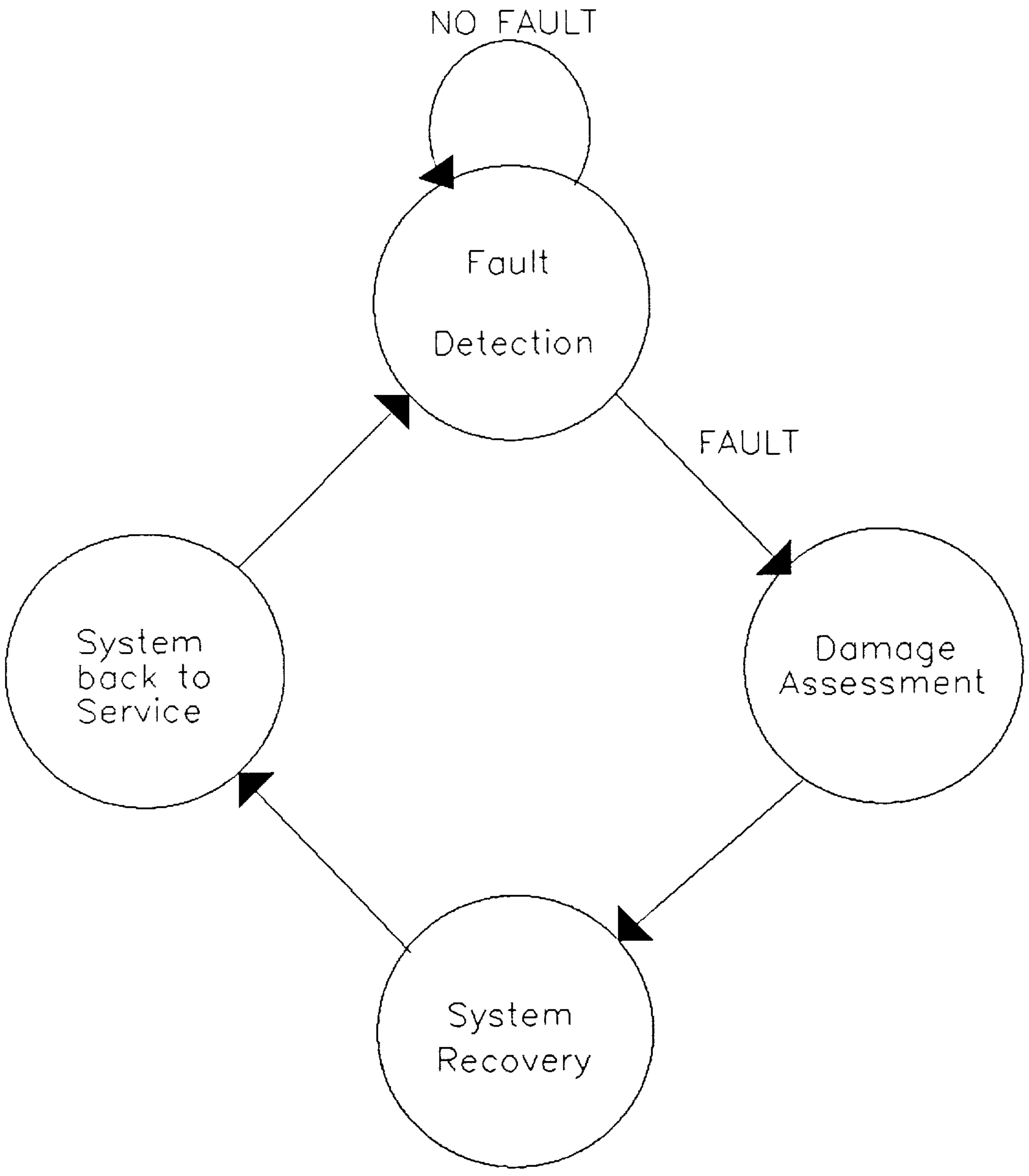


Figure 30. AMOS fault management process for GVSC.

use the graph status for fault detection.

Damage Assessment

Once a fault is detected, system damage must be assessed and confined. The damage assessment is derived from the fault detection scheme. The fault detection strategy should be specific enough to help isolate the resources that possibly may have failed under the given circumstances. Therefore, the assessment can be achieved by observation of the status of the graph, the available resource queue, and the resources themselves. The type of fault that is assumed will render a resource totally unresponsive to any type of inquiry from other resources; hence, detection of this condition is ultimately confirmed by direct inquiries and time-outs to the possibly faulty resources.

System Recovery

After the damage has been confined, the system must be purged of all unwanted data and graph status information that may have been generated by the fault (e. g., an invalid queue update or an unfinished process node). The available resource queue may have to be restored to a valid state by purging the faulty resource. The graph may have to be returned to the last valid state prior to the fault. The system recovery phase is used to perform the appropriate rollback of the system to a valid status.

System Return To Service

Having recovered or repaired the system, the next step is to bring the system back to service. This normally would be done by firing a node that was left unfinished by a faulty resource. After the system is returned to a valid state in the system recovery phase, the system is considered repaired and ready to continue with normal operation.

All of these detection and recovery phases can be carried out by a single resource while the remaining healthy resources are working on the remainder of the graph. The system is not placed in a state of recovery; instead, a resource is placed in a recovery state. This means that the system may be capable of concurrently working on an algorithm graph while it recovers from a fault in the system. This feature insures that the system performance is highly fault tolerant and very reliable.

REFERENCES

[1] J.L. Peterson, Petri Net Theory and the Modeling of Systems, Englewood Cliffs, NJ, Prentice Hall, 1981.

[2] Roland R. Mielke, John W. Stoughton and Sukhamoy Som, "Modeling and Optimum Time Performance for Concurrent Processing," NASA Contractor Report 4167, Grant NAG1-683, August 1988.

[3] Sukhamoy Som, "Performance Modeling and Enhancement for the ATAMM Data Flow Architecture," Ph.D. Dissertation, Old Dominion University, Norfolk, VA, May 1989.

[4] W.R. Tymchyshyn, "ATAMM Multicomputer System Design," Master's Thesis, Old Dominion University,, Norfolk, VA, August 1988.

[5] S. Som, J.W. Stoughton, and R.R. Mielke, "Performance Modeling in the ATAMM Data Flow Architecture," Presented at the Ninth IEEE International Phoenix Conference on Computers and Communications, Scottsdale, Arizona, March 21-23, 1990.

[6] T. Murata, "Modeling and Analysis of Concurrent Systems," Handbook of Software Engineering, C. Vick and C. Ramamoorthy Editors, pp. 39-63, Van Nostrand Reinhold, 1984.

NASA National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA CR-4339	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle Algorithm to Architecture Mapping Model (ATAMM) Multicomputer Operating System Functional Specification	5. Report Date November 1990 6. Performing Organization Code
7. Author(s) R. Mielke, J. Stoughton, S. Som, R. Obando, M. Malekpour and B. Mandala	8. Performing Organization Report No. 10. Work Unit No. 590-32-31-01
9. Performing Organization Name and Address Old Dominion University Research Foundation P.O. Box 6369 Norfolk, Virginia 23508-0369	11. Contract or Grant No. NCC1-136 13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics & Space Administration Langley Research Center Hampton, Virginia 23665-5225	14. Sponsoring Agency Code

15. Supplementary Notes
Langley Technical Monitor: Paul J. Hayes

Interim Report
February 1990 - August 1990

16. Abstract

A functional description of the ATAMM Multicomputer Operating System is presented. ATAMM, an acronym for Algorithm To Architecture Mapping Model, is a marked graph model which describes the implementation of large-grained, decomposed algorithms on data flow architectures. AMOS, the ATAMM Multicomputer Operating System, is an operating system which implements the ATAMM rules. A first generatic version of AMOS which has been developed for the Advanced Development Module (ADM) is described. A second generation version of AMOS being developed for the Generic VHSIC Spaceborne Computer (GVSC) is also presented.

17. Key Words (Suggested by Author(s)) Data flow computers Multicomputer operating system Petri nets Concurrent Processing	18. Distribution Statement UNCLASSIFIED - UNLIMITED Subject Category 33

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 76	22. Price A05

NASA FORM 1626 OCT 86