DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
OLD DOMINION UNIVERSITY
COLLEGE OF ENGINEERING AND TECHNOLOGY
NORFOLK, VIRGINIA 23529

STRATEGIES FOR CONCURRENT PROCESSING OF COMPLEX ALGORITHMS IN DATA DRIVEN ARCHITECTURES

By

John W. Stoughton, Principal Investigator

Roland R. Mielke, Co-Principal Investigator

and

Sukhamony Som, Research Assistant Professor

Final Report
For the period ended August 15, 1989

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

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

April 1990

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
OLD DOMINION UNIVERSITY
COLLEGE OF ENGINEERING AND TECHNOLOGY
NORFOLK, VIRGINIA 23529

STRATEGIES FOR CONCURRENT PROCESSING OF COMPLEX ALGORITHMS IN DATA DRIVEN ARCHITECTURES

By

John W. Stoughton, Principal Investigator

Roland R. Mielke, Co-Principal Investigator

and

Sukhamony Som, Research Assistant Professor

Final Report
For the period ended August 15, 1989

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

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

Submitted by the
Old Dominion University Research Foundation
P.O. Box 6369
Norfolk, Virginia 23508-0369

April 1990

STRATEGIES FOR CONCURRENT PROCESSING OF COMPLEX ALGORITHMS IN DATA DRIVEN ARCHITECTURES

By

John W. Stoughton[1], Roland R. Mielke[2], and Sukhamoy Som[3]

ABSTRACT

This research report is concerned with performance modeling and performance enhancement for periodic execution of large-grain, decision-free algorithms in data flow architectures. Applications include real-time implementation of control and signal processing algorithms where performance is required to be highly predictable. The mapping of algorithms onto the specified class of data flow architectures is realized by a marked graph model called ATAMM (Algorithm To Architecture Mapping Model). Performance measures and bounds are established. Algorithm transformation techniques are identified for performance enhancement and reduction of resource (computing element) requirements. A systematic design procedure is described for generating operating conditions for predictable performance both with and without resource constraints. An ATAMM simulator is used to test and validate the performance prediction by the design procedure. Experiments on a three resource testbed provide verification of the ATAMM model and the design procedure.

---

[1]Associate Professor, [2]Professor, [3]Research Assistant Professor
Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, Virginia 23529

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

SYMBOL	DESCRIPTION
AOP	ATAMM Operating Point
AMG	Algorithm Marked Graph
ATAMM	Algorithm To Architecture Mapping Model
b	Section number of GPST and data packet number
C_i	i^{th} directed circuit
CMG	Computational Marked Graph
CC	Computing Capacity
CE	Computing Effort
DR	Data Ready
FUN	Functional Unit
G	An algorithm marked graph
G_C	A computational marked graph
G_M	Modified algorithm marked graph for G
GLM	Global Memory
GPST	Graph Play for a Single Task Input
GM	Graph Manager
IE	Input Buffer Empty
IF	Input Buffer Full
MAMG	Modified Algorithm Marked Graph
$M(C_i)$	Number of tokens in circuit i
NMG	Node Marked Graph

OE	Output Buffer Empty
OF	Output Buffer Full
p_j	Place j
P_i	i^{th} directed path
PC	Process Complete
PR	Process Ready
REST	Resource Envelope for a Single Task Input
RU	Resource Utilization
t_j	Transition j
$T(C_i)$	Total transition times in C_i
$T(P_i)$	Total transition times in P_i
TBI	Input data injection interval
TBO	Time Between Outputs
TBO_{op}	TBO at the operating point
TBO_{ALB}	Absolute lower bound for TBO
TBO_{LB}	Lower bound for TBO
TBIO	Time Between Input and Output
$TBIO_{ALB}$	Absolute lower bound for TBIO
$TBIO_{LB}$	Lower bound for TBIO
TGP	Total Graph Play
TRE	Total Resource Envelope
TBC	Total Backward Computation
TC	Total Computation
TCE	Total Computing Effort
TFC	Total Forward Computation
TFCE	Total Forward Computing Effort
TT	Task Time

TT_{ALB}	Absolute lower bound for TT
TT_{LB}	Lower bound for TT

PREFACE

The purpose of this report is to document research to develop strategies for concurrent processing of complex algorithms in data driven architectures performed under Grant NAG-1-683 during the period May 16, 1988 to May 15, 1989. In this overview, the problem domain is described, the motivation for this research is explained, and a summary of research activities are presented. The detailed description of the investigation is taken from the doctoral dissertation by Dr. Sukhamoy Som entitled "Performance Modeling and Enhancement for the ATAMM Data Flow Architecture".

During earlier grant periods, a computational model called the Algorithm To Architecture Mapping Model (ATAMM) was formulated for mapping large-grain, decision-free algorithms to a multicomputer data flow architecture. Major applications are expected to be real-time implementation of control and signal processing algorithms where performance is required to be highly predictable and fault tolerant. Of interest is the periodic execution of algorithms. For our purposes, an algorithm is expressed as a directed graph where vertices (nodes) represent algorithm operations and edges represent data sets or signals. Large-grain refers to the assumption that the time required to perform algorithm operations is large compared to the time required to move data from one node to another. Decision-free refers to the absence of data dependent paths in the algorithm graph

representation. The architecture is assumed to consist of two to twenty functional units or resources each having a capability of processing, communication, and memory. The resources share a common global memory which is centralized or distributed. The coordination of resources in relation to data and control flow is directed by a graph manager. The graph manager also is centralized or distributed. Assignment of a functional unit to a specific algorithm node is made by the graph manager according to ATAMM rules and a priority ordering of algorithm nodes. All assignments are non-preemptive for minimum communication cost. 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 a criteria for a multicomputer operating system to achieve predictable and highly fault tolerant performance, and it creates a platform for investigating different algorithm decompositions and implementation strategies in a hardware independent context. In earlier reports, the use of the ATAMM model is described for determining analytically performance bounds and developing an operating strategy for optimum time performance. In addition, the construction of an ATAMM defined data flow architecture and development of simulation and analysis tools are reported. During the present grant period, research is carried out for performance modeling and performance enhancement for the ATAMM data flow architecture. In order to have a predictable performance, it is necessary that assignment of algorithm nodes to functional units be as much priority independent as possible. This is done to avoid the priority inversion problem. Even for small run-time

variations of communication delays and execution time variations, a low priority algorithm node may be enabled before a high priority algorithm node. As the assignment is non-preemptive, this may completely change the graph execution pattern and resource requirements. In order to overcome this problem, it is suggested that the operating system (AMOS) transform the algorithm graph and control input data injection interval so that a functional unit always is available for every enabled algorithm node. In other words, even if priority inversion changes the order of execution of algorithm nodes, graph execution patterns and resource requirements will not be changed drastically. Two performance measures, TBIO and TBO, are defined for periodic processing of algorithms. TBIO is an indicator of computing speed for an algorithm. TBO is a measure of the time interval between algorithm outputs, and the inverse of TBO indicates throughput. The time performance (TBIO, TBO) and the number of required resources define an operating point for AMOS. If enough functional units are available, optimum TBIO and TBO can be achieved. However, if a limited number of resources is available, one must increase either TBO or TBIO, or a combination of both. Two key methods for shifting the operating point are control of the input injection interval and transformation of the algorithm graph. Transformation of the algorithm grapg is achieved by adding dummy nodes (transitions) and control edges (places) as described below. A dummy node is an algorithm node which implements an identity operation and requires zero time. It is used as a buffer to provide additional storage space for the output of an algorithm node. A dummy node is a pure memory operation and does not require a resource. A control edge is an

algorithm edge which imposes a precedence relation among two algorithm nodes but does not imply data dependency. This type of edge is used to delay the execution of a node. Thus, predictable performance is achievable even if the number of functional units decreases to 1. An ATAMM simulator and experiments on a three resource testbed provide verification of performance modeling and graph transformation methods.

CHAPTER ONE

INTRODUCTION

1.0 Preface

Algorithm To Architecture Mapping Model (ATAMM) is a new graph theoretic model from which the rules for data and control flow in a homogeneous, multicomputer, data flow architectures may be defined [1, 2]. The subject of this dissertation is the investigation of concurrent processing in such an ATAMM defined architecture for large-grain, decision-free algorithms. Performance modeling, performance enhancement, and the development of operating strategies for periodic execution of such algorithms are the key research objectives. Chapter One is an introduction of ATAMM and a discussion of the motivation behind the research. Background for the ATAMM model and this research is presented in Section 1.1. The computational problem representation by the ATAMM model is presented in Section 1.2. The objectives and organization of this dissertation are described in Section 1.3.

1.1 Background

The principles of computer architecture design historically have been based upon von Neumann organization [3]. These principles have lead to architectures consisting of a single computer in which low level machine language instructions perform simple operations on elementary operands, and centralized, sequential control of

computation is employed. Despite the fact that electronic components are becoming increasingly faster, the desired computer performance has always been much more than that which is obtainable with von Neumann organization. Advances in the solid state technology alone are not expected to be enough to produce computers to meet the computational needs of the future. There is a growing agreement that the next (fifth) generation of computers will be based upon non-von Neumann structures.

Recently, a number of new computer architectures have been proposed from which a number of computer systems have been built [3]. A few examples are Texas Instruments Distributed Data Processor (USA), Cellular Tree Machine of the University of North Carolina-Chapel Hill (USA), and Manchester Data Flow Computer (England) [3]. This work has been motivated mainly by three objectives. First, there is the desire to increase computer performance through the use of concurrency. Second, there is the desire to more fully exploit very large scale integration (VLSI) in the design of computers. Third, there is interest in new programming methods which facilitate the mapping of algorithms onto architectures. These ideas suggest a decentralized computer architecture in which a number of independent computers are to work together. These independent computers, each having a capability for processing, communication, and memory, can be as large as a geographically distributed mainframe computer or as small as microcomputers on a single VLSI chip. Unfortunately, strategies for interconnecting and programming such architectures based upon von Neumann principles have not evolved. It appears that von Neumann organization principles are not adequate to address the complex issues of scheduling, coordination, and communication.

Strategies for control of computations on decentralized computer architectures can be classified broadly as control flow, demand driven, and data driven. In control flow computers, explicit flows of control cause the execution of instructions. In demand driven architectures, the execution of operations are triggered by the requirements of outputs or results. In data driven architectures (also known as data flow computers), the availability of operands trigger the execution of operations. Data flow architectures are the primary interest of this research because of their suitability for concurrent processing of complex algorithms.

A useful mathematical tool for modeling execution of complex algorithms on a data flow decentralized architecture is the Petri net. Petri nets were first developed in 1962 by Carl Petri [4], and later were identified as a useful analysis tool in the work of Holt and Commoner [5]. A comprehensive treatment of Petri nets is presented in [6]. One problem with the Petri net model is that it tends to be too complicated to analyze. An important subclass of Petri net is the marked graph where each place has exactly one incoming and one outgoing arc. Marked graphs can be used to model the processing of decision-free algorithms [7]. Properties such as liveness, safeness, and reachability can be achieved for marked graph models [6]. Procedures also exist for expanding and reducing marked graphs while preserving these properties [8]. These graph features are suitable for modeling the succession of single events such as data and status conditions. In this dissertation, the marked graph is used as a modeling tool for data driven computations.

The data flow concept has already attracted the attention of a great many researchers. Starting with the work on data flow at MIT by

Jack Dennis, a number of data flow computers have been built [9]. The best strategy for executing an algorithm in these data flow computers is machine dependent. However, only a few researchers have tried to develop a theoretical model for evaluating computation in a data driven architecture [10]. These models do not appear to be adequate to address the complex issues of scheduling, coordination, and communication.

There is a need for a simple, but effective, model for data driven computations in order to investigate the relative merits of different algorithm decompositions and implementation strategies in a hardware independent context. Ongoing research effort at Old Dominion University has lead to the development of a new marked graph model for describing data and control flow associated with the execution of algorithms in data flow architectures [2]. The model is identified by the acronym ATAMM which represents Algorithm To Architecture Mapping Model [11]. Specifications derived from the model lead directly to the description of a data flow architecture and will be called the ATAMM data flow architecture henceforth. The availability of the ATAMM model is important for at least three reasons. First, it provides a context in which to investigate algorithm decomposition strategies without the need to specify a specific ATAMM data flow architecture. Second, the model identifies the data flow and control dialogue required of any ATAMM data flow architecture which implements the algorithm. Third, the model provides a basis for analytically calculating performance bounds and developing a methodology for improvement in performance.

The problem domain addressed by the ATAMM data flow architecture and this research consists of decision-free, large-grain, complex algorithms which are assumed to be executed periodically in a multicomputer environment. The algorithms are assumed to require large computations which would include such computations as matrix addition, multiplication, etc. The anticipated multicomputer environment is assumed to consist of two to twenty identical computers or functional units each having a capability of processing, communication and memory. The primary reason for such assumptions is the objective of implementing control and signal processing algorithms in fifth generation multicomputer architectures for real time applications on board the proposed Space Station. The granularity level of the algorithm decomposition is kept high to avoid communication bottlenecks as observed in many fine-grain data flow architectures [12]. The range of functional units is suggested due to the large-grained aspect of the algorithm decomposition. Of interest is the definition of a performance model so that the performance of the algorithms can be evaluated and improved. Also an operating procedure is needed for obtaining predictable performance with respect to available computing elements.

1.2 Problem Representation by the ATAMM Model

The ATAMM model consists of a set of Petri net marked graphs which incorporate general specifications of communication and processing associated with each computational event in a data flow architecture. In this section, the computational problem is represented by the ATAMM model. First of all a detailed description

of the problem context is stated. This is followed by the definition of the ATAMM model consisting of the algorithm marked graph, the node marked graph, and the computational marked graph. Some familiarity with Petri nets [6] and marked graphs [13] is assumed.

A problem description normally results in the definition of a function given by the triple (X, Y, F), where X represents the set of admissible inputs, Y the set of admissible outputs, and F: X -> Y the rule of correspondence which unambiguously assigns exactly one element from Y to each element of X. Associated with a computational problem is one or more algorithms. An algorithm is an explicit mathematical statement, expressed as an ordered set of primitive operations, which explains how to implement the rule of correspondence F. A primitive operation is a complex computation. Matrix multiplication and addition are examples of primitive operations. In general, a given problem can be decomposed by several different primitive operator sets. Also, for a given primitive operator set, there are often different ordering of primitive operations which can be specified to carry out the problem. Of special interest are algorithm decompositions in which two or more primitive operations can be performed concurrently. For such decompositions, the potential exists for decreasing the computational time required to solve the problem by increasing the computational resources which implement the primitive operations.

The hardware environment for executing the decomposed algorithms is assumed to consist of R identical computers or functional units (FUN's), where R has a value in the range of two to twenty. These computers or functional units are also denoted by the terms

"computing element" or "resource". Each functional unit is a processor having local memory for program storage and temporary input and output data containers. Each functional unit can execute any algorithm primitive operation. The functional units share a common global memory (GLM), which may be either centralized or distributed. The coordination of functional units in relation to data and control flow is directed by the graph manager (GM). The graph manager also 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 graph manager only when all inputs required by the operation are available in global memory and a functional unit is available.

An algorithm marked graph (AMG) is a marked graph which represents a specific algorithm decomposition. Transitions and places are represented as vertices and directed edges respectively. Vertices of the algorithm marked graph are in a one-to-one correspondence with each occurrence of a primitive operation. The transition times represent the computation times of the respective primitive 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. By the rules of the marked graph, the computation of a vertex can only be done when all the incoming edges have a token on them. When constructing an

algorithm marked graph, vertices (transitions) are displayed as circles, and edges (places) are displayed as directed line segments connecting appropriate vertices. The presence of a token on an edge is indicated by a solid dot placed on the edge. Source transitions and sink transitions for input and output signals are represented as squares. Sources for constants are not usually included in the algorithm marked graph; however, triangles are used for this purpose when necessary.

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 an r-vector. 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 marked graph for this decomposed algorithm is shown in Figure 1.1. 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

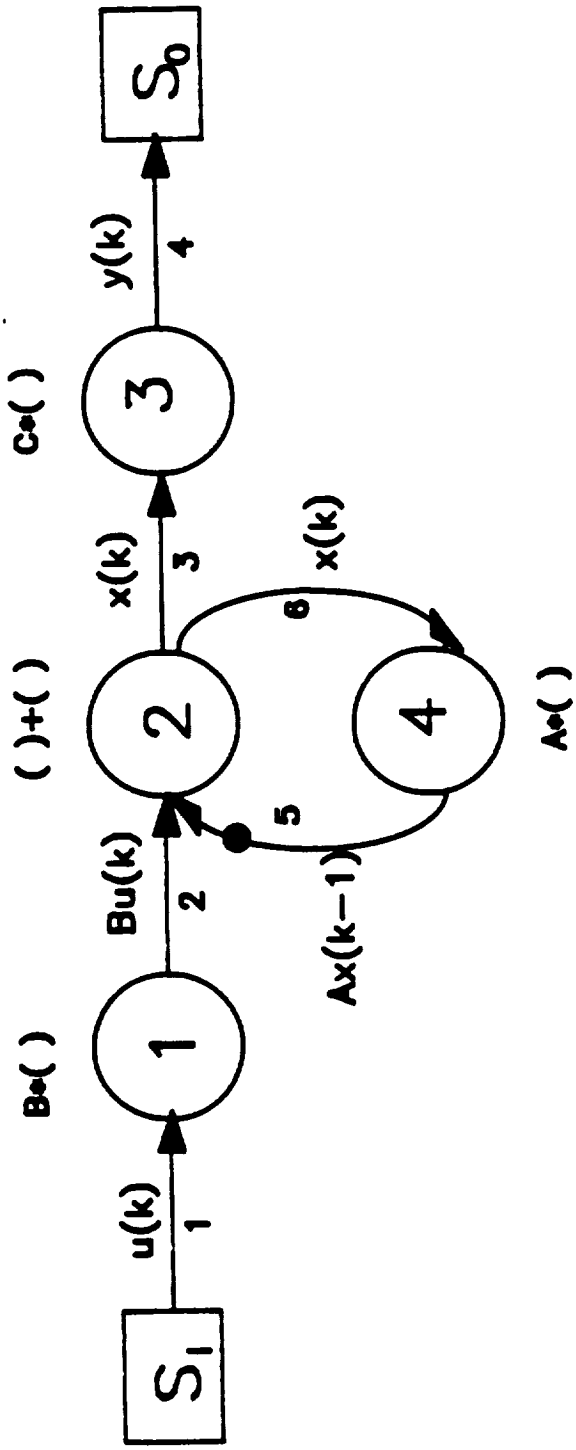


Figure 1.1. Algorithm marked graph for discrete system equation.

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. The node marked graph (NMG) is defined to model the execution of a primitive operation. The computational marked graph (CMG), obtained from the AMG and the NMG by a set of construction rules, integrates both the algorithm requirements and the computing environment requirements into a comprehensive graph model. These additional marked graphs are defined below.

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

units. The initial marking for an NMG consists of a single token in the Process Ready place. The NMG model in shown in Figure 1.2.

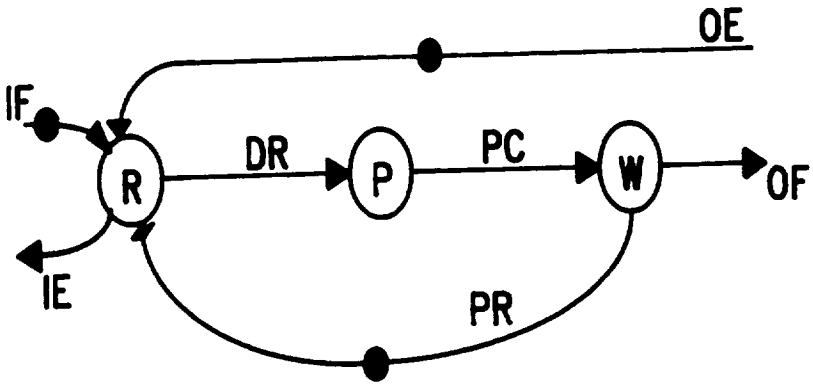
A computational marked graph (CMG) is constructed from the AMG and the NMG by the following rules:

1) Source and sink nodes in the algorithm marked graph are represented by source and sink nodes in the CMG.
2) Nodes corresponding to primitive operations in the algorithm marked graph are represented by NMG's in the CMG.
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.

The forward directed edge goes from predecessor write transition to successor read or sink transition. This forward edge is also shown as part of the NMG where it is the OF and IF edge of the predecessor and successor respectively. The backward directed edge goes from successor read transition to predecessor read or source transition. This backward edge is also shown as part of the NMG where it is OE and IE edge of predecessor and successor 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.

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.



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 1.2. ATAMM node marked graph model.

2) A source node and a sink node fire when enabled without regard for the availability of a functional unit.

3) A primitive operation is initiated when the read node of an NMG is enabled and a functional unit is available for assignment to the NMG. A functional unit remains assigned to an NMG until completion of the firing of the write node of the NMG.

In order to illustrate the construction of a computational marked graph, the CMG corresponding to the algorithm marked graph of Figure 1.1 is shown in Figure 1.3. The computational marked graph is useful because it clearly displays the data and control flow which must occur in any hardware implementation of the algorithm, and because it provides a hardware independent context in which to evaluate algorithm performance.

The complete ATAMM model consists of the algorithm marked graph, the node marked graph, and the computational marked graph. A pictorial display of this model is shown in Figure 1.4. ATAMM model characteristics are described in detail in the Appendix.

1.3 Objectives and Organization of Dissertation.

The behavior and performance for periodic execution of complex algorithms in the ATAMM data flow architecture is investigated in this dissertation. The problem domain consists of large-grain, decision-free algorithms. The major research objectives are threefold. First, a performance model is established. Second, rules for transformation of algorithms for performance enhancement and reduction of computing element requirements are identified. Third,

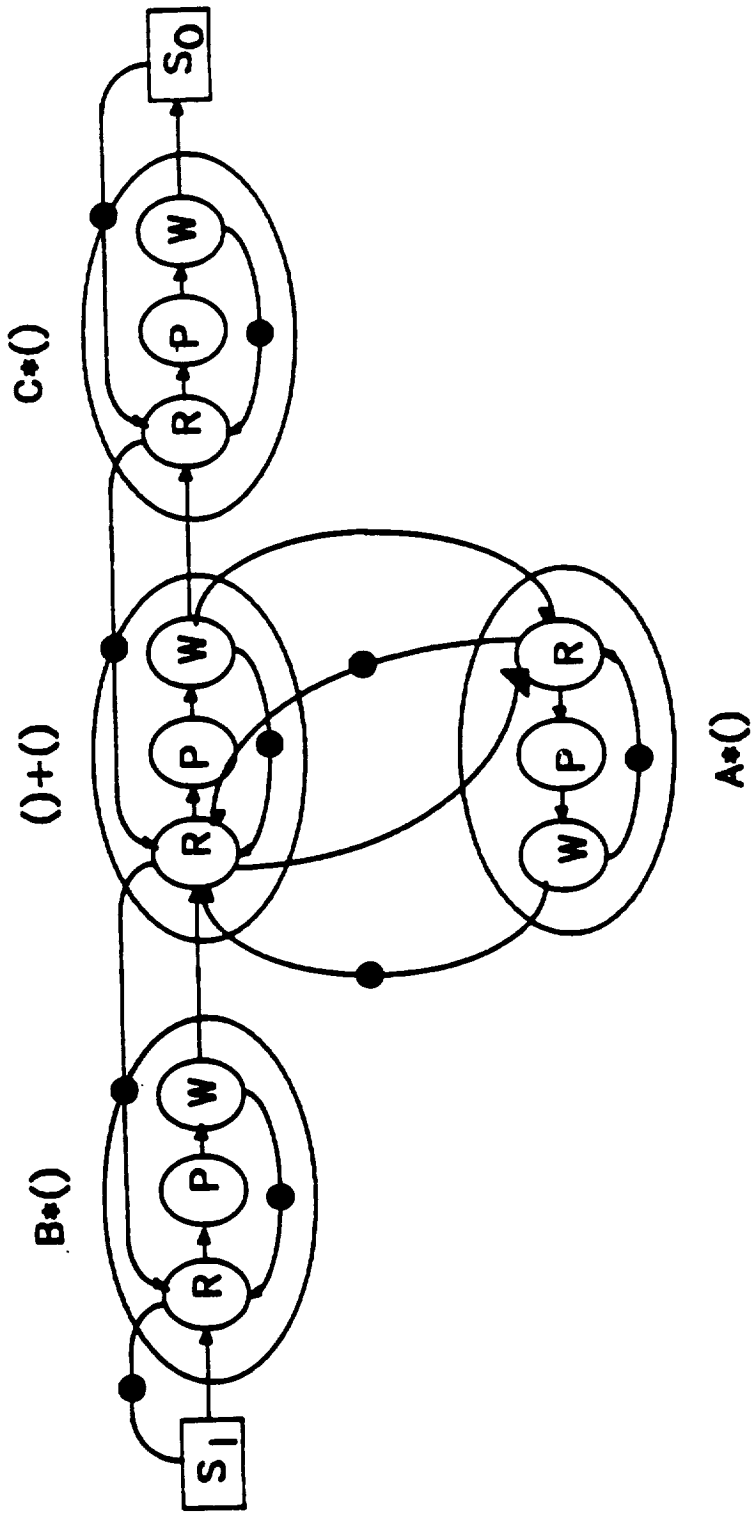


Figure 1.3. ATAMM computational marked graph model for discrete system equation.

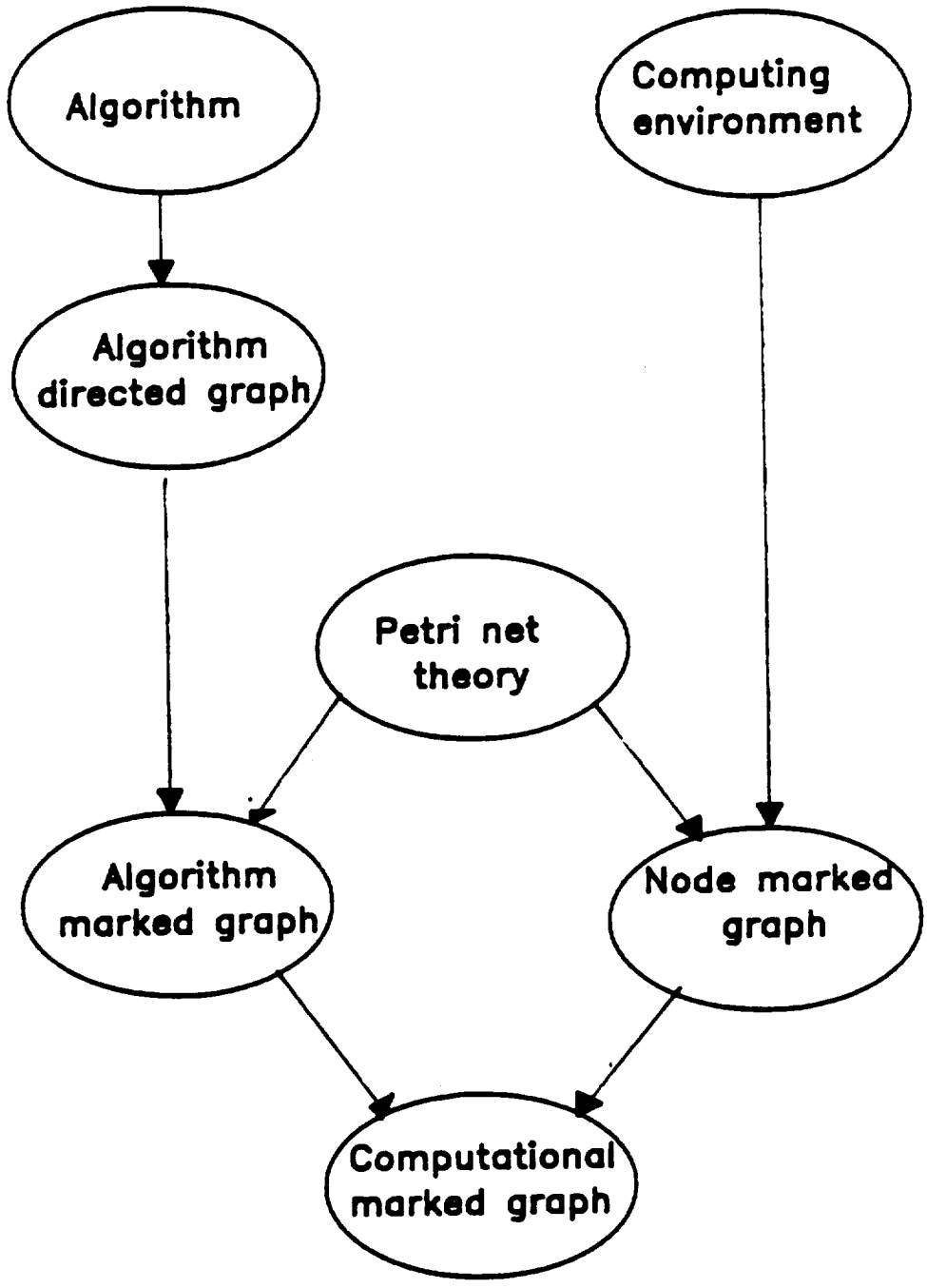


Figure 1.4. ATAMM model components.

operating strategies are developed for optimum time performance and for sub-optimum time performance under limited availability of computing elements.

The dissertation is organized in five chapters and an appendix. In the Appendix ATAMM model characteristics, some of which are used in this dissertation, are described in detail. Definitions of the computing environment, performance measures, and evaluation of performance bounds and resource requirements are presented in Chapter Two. In Chapter Three, algorithm transformations for improving performance, and methods for enforcing desired resource envelope and inducing structural changes in algorithm marked graph are described. Definition, characteristics, and design procedure of operating point along with simulation and experimental results are presented in Chapter Four. Finally conclusions from this research and future research topics are presented in Chapter Five.

CHAPTER TWO

PERFORMANCE MODEL

2.0 Introduction

A performance model for the ATAMM (Algorithm To Architecture Mapping Model) data flow architecture is described in this chapter. The objective is to determine computing speed, throughput capacity and resource (computing element) need for implementing decision-free large-grain algorithms on the ATAMM data flow architecture. The computing environment and performance measures are defined in Section 2.1. In Section 2.2, characteristics of marked graphs, which are needed to establish the performance model, are described. Graph theoretic lower bounds for the time performance of algorithm marked graphs operated in the ATAMM data flow architecture are established in Section 2.3. Resource needs are predicted and performance bounds in the presence of resource limitations are evaluated in Section 2.4. A summary of the chapter is presented in Section 2.5.

2.1 Performance Measures

The importance of the ATAMM model is that it provides a hardware independent context in which to investigate the performance of decomposed algorithms as long as the architecture obeys the rules of CMG. It is assumed that a decomposed algorithm is implemented in a ATAMM data flow architecture containing R identical resources or

functional units. Each functional unit is capable of performing any of the primitive operations whose sequence defines the decomposition. The tokens on the CMG indicate the data and control flow that must occur in any hardware implementation of the algorithm. A task is a sequence of computations as described by the AMG. The computational task is applied on all input data from the source node. Task output occurs when a corresponding output data token is deposited at the output sink node. A task is completed when all computing associated with the task is completed. It should be noted that task output and task completion do not always coincide. In many iterative signal processing algorithms, computing to generate initial conditions for the next iteration often occurs after the output has been calculated. Task completion is usually indicated in the AMG or the CMG by the return of the graph to some steady state initial marking. To use the output of an algorithm for control and signal processing applications, it is assumed that the task is repeated periodically with new input data sets (data packets). New data sets are injected as input tokens from the input source node at a finite interval of time so that computing time and resource needs are identical for all data sets. Included in this problem class are iterative algorithms where the present task requires input data from previous task calculations.

Computational concurrency occurs in two ways. First, several transitions of the task on individual data set may be performed simultaneously. We have referred to this type of concurrency as parallel concurrency because it is the result of inherent parallelism in the algorithm. Parallel concurrency has a direct effect on task computing speed. It is limited by the number of transitions that can

be performed simultaneously for the given task and by the number of functional units available to perform the transitions. Second, transitions of the task belonging to different data sets can be performed simultaneously in the computing system. This type of concurrency is referred to by us as pipeline concurrency because the task is repeated for successive data sets, like a pipeline. This type of concurrency has a direct effect on throughput capacity. It is limited by the capacity of the graph to accommodate additional data sets and by the number of functional units available to implement the algorithm periodically.

Three performance measures, TBIO, TT, and TBO, are now defined for concurrent processing of complex algorithms in ATAMM data flow architectures. TBIO and TT are indicators of computing speed for a task and thus reflect the degree of parallel concurrency. TBO is a measure of time interval between task outputs. The inverse of TBO indicates throughput, and thus reflects the degree of pipeline concurrency.

Definition 2.1: TBIO. The performance measure TBIO (time between input and output) is the elapsed computing time between a task input and the corresponding task output.

Definition 2.2: TT. The performance measure TT (task time) is the elapsed computing time between a task input and the completion of all computation associated with that task input.

Definition 2.3: TBO. The performance measure TBO (time between outputs) is the elapsed computing time between successive task outputs when the graph is operating periodically at steady state.

To illustrate, an algorithm marked graph for an aircraft flight simulation is shown in Figure 2.1. S_I is the input source

representing flight plan data. S_O is the output sink representing moving map and flight instruments data. Transitions of the graph represent activities. Places represent data dependency or precedence relation. Tokens on places are initial tokens representing initial condition data. As an example, transition 3 represents inertial navigation computation and requires ten time units for processing. Time units associated with transitions are relative and are measured with respect to a reference. Transition 7 (zero processing time) is used to combine outputs of the coordinate transform computation (moving map) and the auto-pilot computation (control for flight instruments). TBIO is the time to produce the outputs in S_O for a flight plan data. TT is the time to finish all processing for a task input. TBIO and TT need not be the same for all problems although they are related. TBO is the time between arrival of successive output tokens in the output data sink when the algorithm is executed periodically at steady state.

2.2 Marked Graph Characteristics

Marked graphs, a class of Petri nets, are used as a device for expressing the ATAMM. A marked graph is viewed as a directed graph where the vertices are the transitions and the edges are directed places. In this section, concept of path and circuit for the marked graph is developed. Only directed paths and circuits are of interest to this dissertation. If not mentioned, a path or a circuit of a marked graph should always be understood to be a directed path or a directed circuit respectively. Some properties of the marked graph

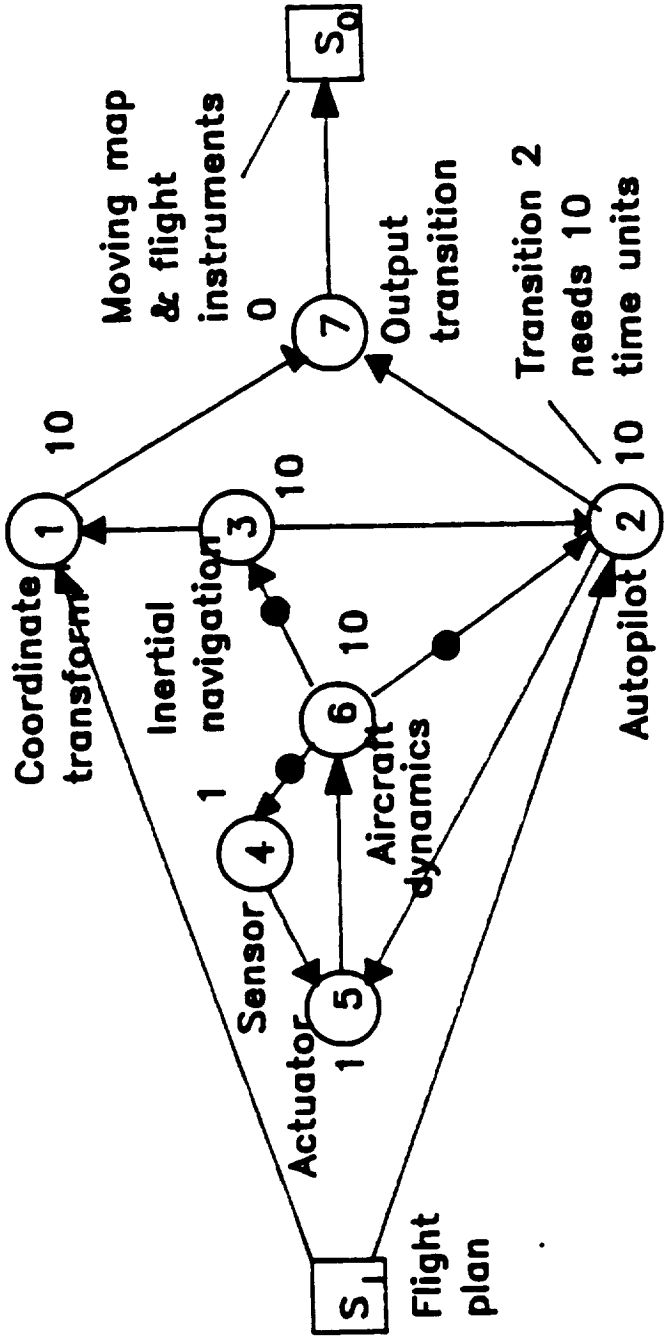


Figure 2.1. An algorithm for flight simulation plan.

which are needed to establish a performance model are stated. Also, circuits of the CMG are classified. Let t_i and p_i denote transition i and place i respectively.

Definition 2.4: Directed Path. A directed path in a marked graph is a finite alternating sequence of distinct transitions and distinct directed places with the following property. The sequence begins and ends with transitions and every place originates from the immediate predecessor transition and ends on the immediate successor transition in that sequence.

To illustrate, the sequence S_I, p_1, t_1, p_2, t_2, p_3, t_3, p_4, and S_O is a directed path in Figure 1.1. But the sequence t_1, p_2, t_2, p_6, t_4, p_5, t_2, p_3, and t_3 is not a directed path in Figure 1.1 as transition 2 is repeated twice in that sequence.

Definition 2.5: Directed Circuit. A directed circuit in a marked graph is the same as a directed path except that beginning and end transitions are the same in a directed circuit.

To illustrate, the sequence t_2, p_6, t_4, p_5 and t_2 is a directed circuit in Figure 1.1.

Definition 2.6: Parallel Paths. Parallel paths are directed paths which have identical beginning and ending transitions; however, all other transitions and places on all directed paths are distinct.

In Figure 2.2, the sequence t_1, p_2, t_2, p_3, t_3, p_4, t_4, p_5, and t_5 and the sequence t_1, p_6, t_6, p_8, and t_5 are parallel paths.

Definition 2.7: Group Of Paths. Group of paths are a finite number of directed paths from a marked graph.

To illustrate, the sequences t_2, p_7, t_7, p_9, t_4 and t_1, p_6, t_6, p_8, t_5 form a group of paths in Figure 2.2.

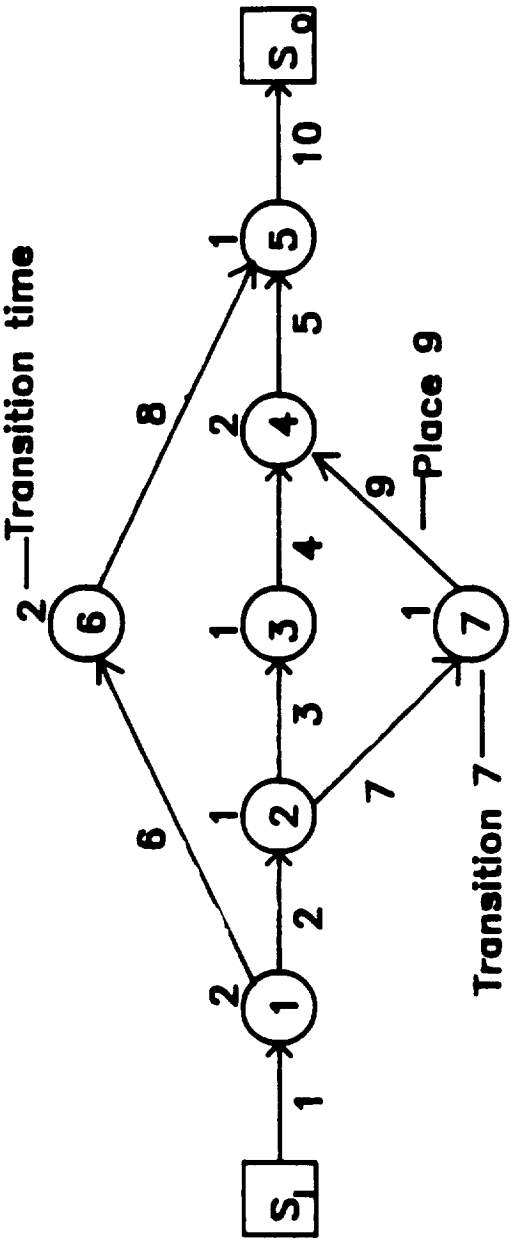


Figure 2.2. Example algorithm marked graph.

Definition 2.8: Path Length. The length of a directed path in a marked graph is defined to be the summation of all the times for transitions in that directed path.

Definition 2.9: Circuit Length. The length of a directed circuit in a marked graph is defined to be the summation of all the times for transitions in that directed circuit.

Definition 2.10: Critical Path. The critical path among a group of paths is the one which has the highest path length.

This definition of critical path is identical to the one used in task scheduling [14, 15] and project management [16, 17].

To illustrate, let T(i) stand for the time of the i^{th} transition. In Figure 1.1, let T(1) = 4, T(2) = 1, T(3) = 5 and T(4) = 6, $T(S_I) = 0$ and $T(S_O) = 0$. Then, the directed circuit t_2, p_6, t_4, p_5, and t_2 has length 7. The directed path used to illustrate Definition 2.4 has length 10. The directed path S_I, p_1, t_1, p_2, t_2, p_6, and t_4 has length 11. These two directed paths form a group of paths. In that group of paths, the directed path from S_I to t_4 is the critical path. It is to be noted that there can be more than one critical path in a group of paths.

Property 2.1. The critical path length of a group of paths is the lowest possible time to move tokens from the input of the beginning transition to the output of the end transition on all directed paths of that group.

This is a property of the critical path known from critical-path scheduling [14] and project management [17]. In the context of a marked graph, as the token has to move through all the transitions of the directed path in order to reach the output of the end transition

from the input of the beginning transition, the minimum time required is the length of the directed path. Considering all the directed paths of the group, the lowest possible time to move tokens on all directed paths from the input of the beginning transition to the output of the end transition is the critical path length.

Property 2.2. With unlimited resources, tokens always take time equal to critical path length to complete the move from the input of the beginning transition to the output of the end transition on all directed paths of the group.

This is another property of the critical path known from task scheduling [14] and project management [17]. In the context of the marked graph, with unlimited resources, a transition can always be fired as soon as it is enabled by input data. Therefore, the lowest possible time can actually be achieved. Hence, the critical path length is the time to move all tokens from the input of the beginning transition to the output of the end transition.

Directed circuits are created in the computational marked graph in four different ways. They are node, process, recursion and parallel path circuits. Formal definitions of each kind of directed circuit are presented below along with examples.

Definition 2.11: Node Circuit. This is a directed circuit in the CMG which is the only internal directed circuit of an NMG.

To illustrate, the sequence t_R, p_{DR}, t_P, p_{PC}, t_W, p_{PR}, and t_R is a node circuit in the ATAMM node marked graph model of Figure 1.2. One such node circuit in the CMG of Figure 1.3 is shown in Figure 2.3(a). This is the node circuit of transition 1 in the AMG of Figure 1.1. Node circuits always have one token, as described in the Appendix.

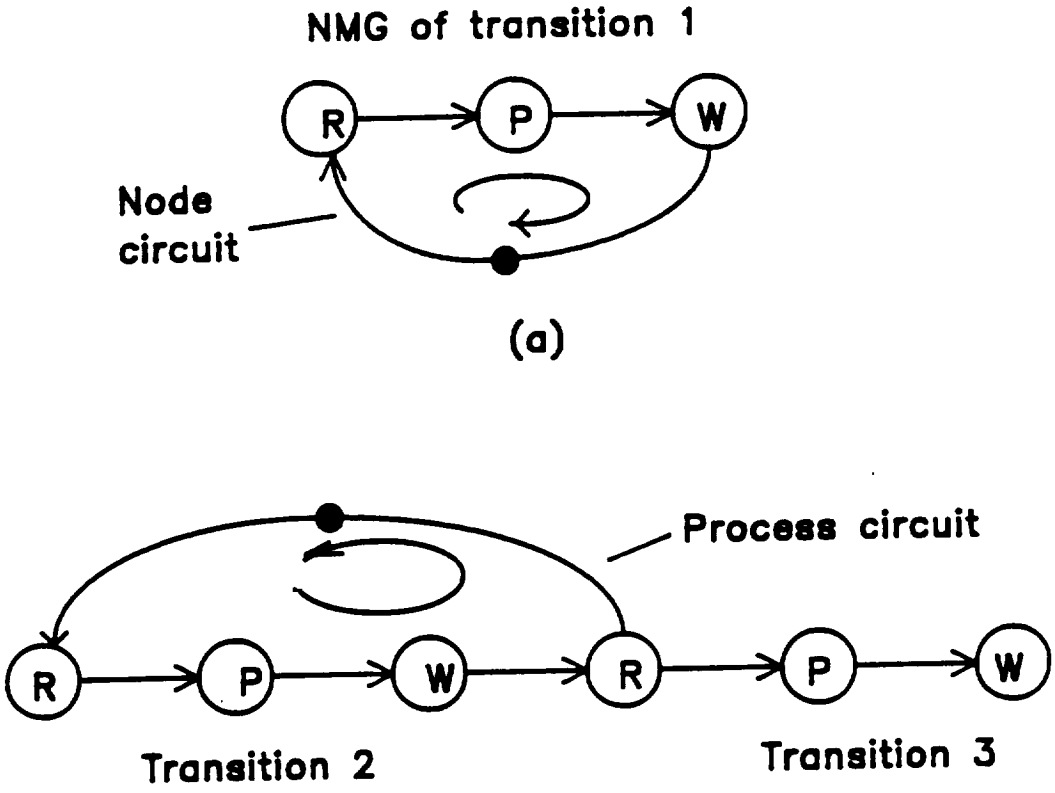


Figure 2.3. Example of node and process circuits.

Definition 2.12: Process Circuit. This is a directed circuit in the CMG which is formed each time an NMG or source is linked to another NMG or sink. The backward directed place from successor read or sink transition to predecessor read or source transition, along with forward directed places from predecessor to successor create the process circuit.

A process circuit of Figure 1.3 is shown in Figure 2.3(b). This process circuit is formed when node marked graphs of transition 2 and 3 are linked. Process circuits always have one token as described in the Appendix.

Definition 2.13: Parallel Path Circuit. This is a directed circuit in the CMG which is created by any two parallel paths in the AMG. The circuit is formed by the forward directed places through the NMG'S of one directed path and backward directed places from the successor read to the predecessor read transition from the NMG's of the other directed path.

To illustrate, the CMG of Figure 2.2 is shown in Figure 2.4. The parallel paths of the AMG form parallel path circuits in the CMG. One such parallel path circuit is shown in Figure 2.5(a). This circuit is created by two parallel paths in the Figure 2.2 between transition 1 and transition 5.

Definition 2.14: Recursion Circuit. This is a circuit in the CMG which is created due to a directed circuit in the algorithm marked graph.

To illustrate, the recursion circuit of Figure 1.3 is shown in Figure 2.5(b). The directed circuit t_2, p_6, t_4, p_5, and t_2 in Figure 1.1 translates itself into a recursion circuit in the CMG of

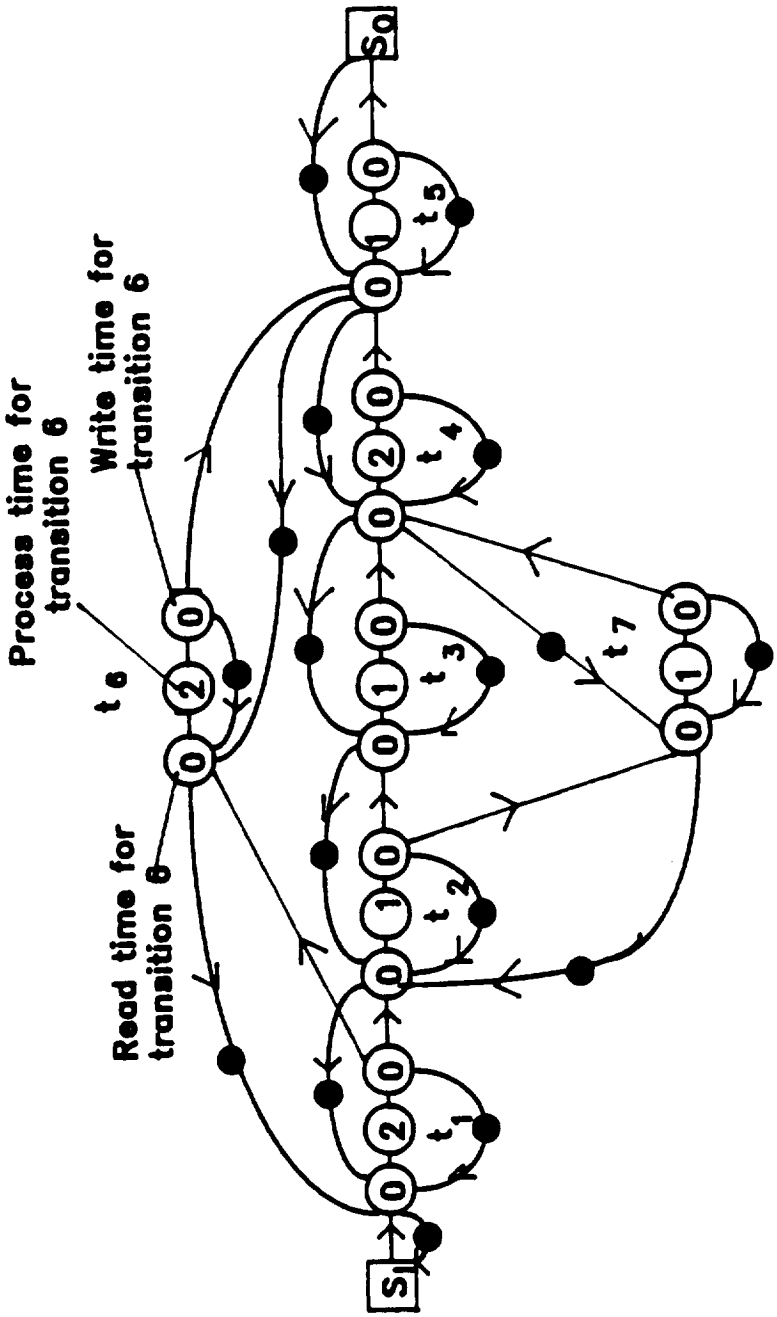


Figure 2.4. Computational marked graph for the AMG.

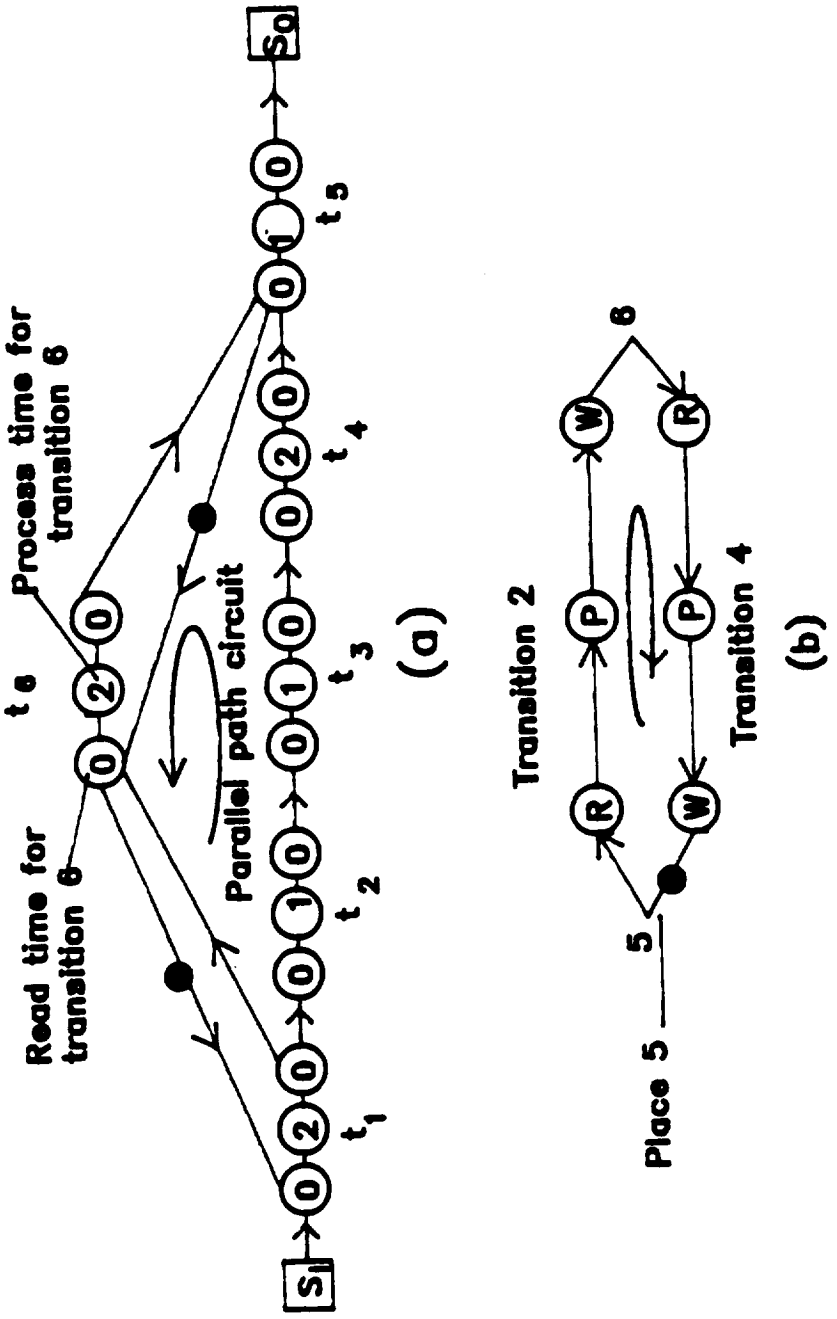


Figure 2.5. Example of recursion and parallel path circuits.
(a) A parallel path circuit from the CMG of Figure 2.4.
(b) A recursion circuit from Figure 1.3.

Figure 1.3. Directed circuits are created in the AMG mainly due to a recursion in computation and hence the corresponding circuits in the CMG are called recursion circuits.

2.3 Graph Theoretic Performance Bounds

The process of algorithm decomposition imposes bounds on the amount of parallel concurrency and pipeline concurrency possible in a given problem. If sufficient computing resources are available, operation at these bounds can be achieved. In this section, graph theoretic lower bounds on three performance measures are established for decomposed algorithms to be operated in ATAMM data flow architectures. These lower bounds are only a function of the algorithm marked graph and the node marked graph. Therefore, performance cannot be improved beyond these bounds by increasing the number of resources. The remainder of this section is devoted to developing lower bounds for these performance measures.

Let G denote an algorithm marked graph representing a decomposed algorithm. The lower bound for TBIO is the shortest time required for a data token from the data input source to propagate through the graph to the data output sink. Similarly the lower bound for TT is the shortest time required to complete all computing activity initiated by the injection of a data from the input source. These shortest times are the actual performance times when only a single data set is present in the graph during any time interval (no pipeline concurrency), and as many computing resources as are required are available (maximum parallel concurrency). Under these operating conditions, lower bounds for TBIO and TT are calculated by identifying

certain longest paths in a graph obtained from the algorithm marked graph. This new graph, called the modified algorithm marked graph G_M, is defined and then used to determine lower bounds for TBIO and TT.

Definition 2.15: Modified Algorithm Marked Graph. Let p_i be a place of G, directed from transition t_r to transition t_s, which contains a token of the initial marking. The modified algorithm marked graph G_M is obtained from the graph G by the following construction rules:

1) Place p_i is deleted from G.
2) A new place, p_{i1}, directed from the data input source to transition t_s, is added to G.
3) A new output sink S_i different from all other output sinks, and a new place p_{i2}, directed from transition t_r to S_i, are both added to G.
4) The above rules are repeated for each place of G containing a token of the initial marking.

Example: The recursion problem of Figure 1.1 is used to generate a modified algorithm marked graph as shown in Figure 2.6. Only place 5 from transition 4 to 2 has an initial token in the algorithm marked graph of Figure 1.1. According to rule 1, place 5 is deleted. A new place 5-1 is inserted from data input source to transition 2 by rule 2. Rule 3 is then used to generate a new output sink (S_5) and a new place 5-2 as shown in Figure 2.6. As there are no more places with initial tokens, this completes the procedure to generate a modified algorithm marked graph.

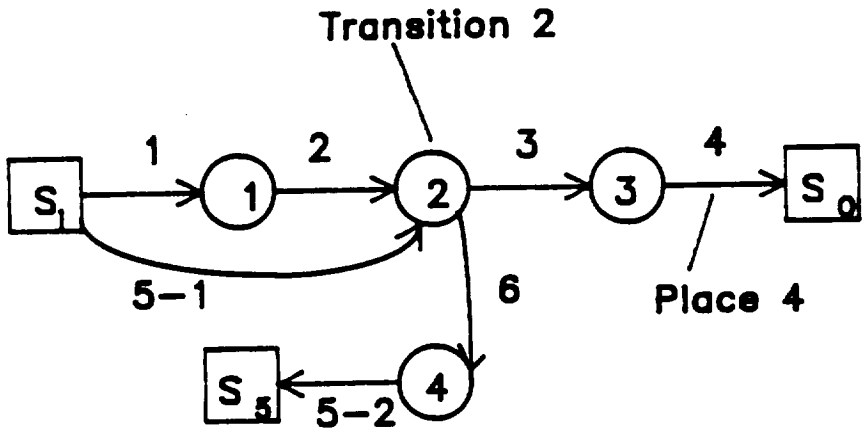


Figure 2.6. Modified algorithm marked graph for Figure 1.1.

Theorem 2.1: Graph Theoretic Lower Bound for TBIO. Let P_i be the i^{th} directed path in G_M from the data input source to the data output sink, and let $T(P_i)$ denote the sum of transition times for transitions contained in P_i. Then,

$$TBIO_{LB} = Max\ \{T(P_i)\},$$

where the maximum is taken over all paths P_i between the data input source and the data output sink in graph G_M.

Proof. $T(P_i)$ is the length of path P_i; therefore, Max $(T(P_i))$ is the length of the critical path from the data input source to the data output sink. From the properties of the critical path [14, 17], $TBIO_{LB}$ = Max $(T(P_i))$. This completes the proof.

Theorem 2.2: Lower Bound for TT. Let P_i be the i^{th} directed path in G_M from the data input source to any output sink, and let $T(P_i)$ denote the sum of transition times of transitions contained in P_i. Then,

$$TT_{LB} = Max\ \{T(P_i)\},$$

where the maximum is taken over all paths P_i in graph G_M.

Proof. By the construction rules for graph G_M, a task is initiated with an input from the data input source, and is completed when all output sinks have accepted tokens. Therefore, TT is the time which elapses from injection of input tokens to the arrival of a token at the last fired output sink. Let $T(P_j)$ = Max $(T(P_i))$, among all P_i in G_M. P_j is the longest path among all paths from the

data input source S_I to any output sink. Therefore, P_j is the critical path among all paths from the data input source to any output sink. Hence, by the properties of the critical path [14, 17], $TT_{LB} = T(P_j) = Max(T(P_i))$, where the maximum is over all paths P_i in G_M. This completes the proof.

To illustrate the application of Theorem 2.1 and Theorem 2.2, $TBIO_{LB}$ and TT_{LB} are computed for the algorithm marked graph shown in Figure 1.1. For this example, the following transition times are assumed: $T(1) = 4$, $T(2) = 1$, $T(3) = 5$, and $T(4) = 6$. The modified algorithm marked graph corresponding to Figure 1.1 is shown in Figure 2.6. The modified algorithm marked graph contains two paths directed from the data input source S_I to the data output sink S_O. Path P_1 is the sequence t_1, p_2, t_2, p_3, and t_3 with $T(P_1) = 10$. Path P_2 is the sequence t_2, p_3, and t_3 with $T(P_2) = 6$. Since $T(P_1) > T(P_2)$, path P_1 determines the lower bound for TBIO and $TBIO_{LB} = 10$. The modified algorithm marked graph contains two additional directed paths from the data input source S_I to the output sink S_5. Path P_3 is the sequence t_1, p_2, t_2, p_6, and t_4 with $T(P_3) = 11$. Path P_4 is the sequence t_2, p_6, and t_4 with $T(P_4) = 7$. Since $T(P_3)$ is the highest, path P_3 determines the lower bound for TT and $TT_{LB} = 11$.

Next a lower bound for the performance measure TBO may be determined. Let G be an algorithm marked graph representing a decomposed algorithm. It is assumed that the operating conditions for G are set to maximize pipeline concurrency. That is, data tokens are continuously available at the data input source, and as many computing resources as needed can be called to perform primitive operations. The graph G is executed periodically and TBO_{LB} is the shortest time possible between successive outputs.

Theorem 2.3: Graph Theoretic Lower Bound for TBO. Let G_C be a computational marked graph and let C_i be the i^{th} directed circuit in G_C. The notation $T(C_i)$ denotes the sum of transition times of transitions contained in C_i, and $M(C_i)$ denotes the number of tokens contained in C_i. Then,

$$TBO_{LB} = \text{Max } (T(C_i) \ / \ M(C_i)),$$

where the maximum is taken over all directed circuits in G. The circuits which determine TBO_{LB} will be called critical circuits of the CMG.

Proof. Without loss of generality, let t_f be the output transition in G_C so that an output is produced each time t_f completes firing. TBO_{LB} is then the minimum firing period of transition t_f. By consistency property of the Appendix, G_C is consistent so that all transitions of G_C fire periodically with minimum period TBO_{LB}. It is shown in [18] (pp. 58-60) that the minimum firing period of each transition of a marked graph is given by Max $(T(C_i)/M(C_i))$, where the maximum is taken over all directed circuits C_i in G. Therefore, the theorem follows.

The algorithm marked graph shown in Figure 1.3 is used to illustrate Theorem 2.3. The CMG contains many directed circuits. However, the recursion circuit which contains all NMG nodes of transitions 2 and 4 has only one token and maximizes the ratio $T(C_i)/ M(C_i)$. Therefore, the shortest time possible between successive outputs in this graph is $TBO_{LB} = 7$.

2.4 Resource Requirements

The performance bounds of the last section assume availability of a resource for each transition to fire when enabled. Therefore, graph theoretic performance bounds are absolute bounds provided sufficient resources are available to meet the firing requirements. However, for insufficient resources, performance cannot reach the graph-theoretic bounds. The number of resources (R) of an ATAMM data flow architecture imposes bounds on performance of an algorithm marked graph. In this section, characteristics of resource usage, maximum resource requirement, and resource imposed performance bounds are investigated. Formal definitions of computation, graph execution, and resource requirements are stated. Definitions and results are illustrated with examples.

Definition 2.16: TC. Total Computation (TC) is the sum of all transition times of an algorithm marked graph.

Definition 2.17: TFC. Total Forward Computation (TFC) is the sum of all transition times that appear in the forward paths from the data input source to the data output sink of the modified algorithm marked graph.

Definition 2.18: TBC. Total Backward Computation (TBC) is the sum of all transition times that do not appear in the forward paths from the data input source to the data output sink of the modified algorithm marked graph.

Lemma 2.1. TC is the sum of TFC and TBC of an algorithm marked graph.

Proof. With the notation of Definitions 2.16, 2.17, and 2.18, transitions which constitute TFC and TBC are mutually exclusive and collectively exhaustive of all transitions of the algorithm marked

graph. Hence, the sum of all transition times of the algorithm marked graph equals the sum of transition times for both transitions on the forward paths and not on the forward paths from the data input source to the data output sink of the modified algorithm marked graph. Therefore, TC equals the sum of TFC and TBC. This completes the proof.

Definition 2.19: Computer Time. A unit of Computer Time is defined to indicate one functional unit available over one unit of time.

To illustrate, if two functional units are used for three units of time, six units of computer time are used.

Definition 2.20: Computing Capacity (T). Computing Capacity (CC) is the total available units of computer time over an interval of time T.

To illustrate, for a time interval of T, the computing capacity of an ATAMM data flow architecture with R functional units is given by R * T. Thus CC (T) = R * T.

Definition 2.21: Computing Effort (T). Computing Effort (CE) is the total used units of computer time over an interval of time T.

To illustrate, for a time interval of T and R functional units, let T_i be the number of time units the i^{th} functional unit is used. Then $T_i * 1 = T_i$ units of computer time is the computing effort due to the i^{th} resource in interval T. Thus the computing effort due to R resources is given by

$$CE\ (T) = \sum_{i=1}^{R} (T_i)$$

units of computer time.

Lemma 2.2. For any number of functional units and any interval of time, computing effort is always less than, or equal to, computing capacity.

Proof. With the notation of definitions 2.20 and 2.21,

$$CC\ (T) = R * T$$

$$CE\ (T) = \sum_{i=1}^{R} (T_i),$$

where T_i is the number of time units the i^{th} functional unit was used in time interval T. So T_i cannot be more than T [15]. Hence, $CE(T) \leq CC(T)$. This completes the proof.

Definition 2.22: Resource Utilization (T). The Resource Utilization (RU) of functional units over a time interval T is given by the ratio of computing effort to computing capacity over that time interval. Thus,

$$RU\ (T) = CE\ (T) / CC\ (T).$$

Lemma 2.3. Resource Utilization (RU) over a time interval T is always greater than, or equal to, zero but less than, or equal to, 1.

Proof. By definition, resource utilization is a ratio of computing effort to capacity. With the notation of Definitions 2.20 and 2.21, $T_i \geq 0$, $T > 0$. So $CE(T) \geq 0$. $CC(T) = R * T > 0$ as the ATAMM data flow architectures must have at least one functional unit. So $RU(T) \geq 0$. Also as $CE\ (T) \leq CC\ (T)$, $RU\ (T) \leq 1$. This completes the proof.

Definition 2.23: Total Computing Effort (TCE). TCE is defined to be the computing effort required to execute once all transitions of an algorithm marked graph.

Lemma 2.4. TCE equals TC units of computer time.

Proof. With the notation of Definitions 2.16, 2.21, and 2.23,

$$TCE = CE(T) = \sum_{i=1}^{R} (T_i)$$

$$= TC$$

units of computer time as total computation to execute all transitions of the AMG once is TC. This completes the proof.

Definition 2.24: Total Forward Computing Effort (TFCE). TFCE is defined to be the computing effort required to execute once all transitions on forward paths from the data input source to the data output sink of the modified algorithm marked graph.

Lemma 2.5. TFCE equals TFC units of computer time.

Proof. The proof is similar to that of Lemma 2.4.

With the above definitions and lemmas regarding computation of a task, it is now intended to establish resource imposed bounds on the computing time of a task. The following two theorems state the minimum possible value of TT and TBIO for an ATAMM data flow architecture of R resources.

Theorem 2.4: Minimum TT for R Resources. The minimum value of TT for an algorithm marked graph operated with R resources is always greater than, or equal to, TCE / R.

Proof. TT is the computing time to complete all computation associated with a task input. For a time interval of TT, the

computing capacity of R resources is R * TT. The total computation for any task input is the execution of all transitions of the algorithm marked graph once and hence, equals TC. The corresponding computing effort is TCE. By Lemma 2.2, R * TT $\geq$ TCE, or TT $\geq$ TCE / R [19]. This completes the proof.

Theorem 2.5: Minimum TBIO for R Resources. The minimum value of TBIO for an algorithm marked graph operated with R resources is always greater than, or equal to, TFCE / R.

Proof. TBIO is the computing time to generate data output for a task. For a time interval of TBIO, the computing capacity of R resources is given by R * TBIO. In order to generate data output, all transitions on all the forward paths from the data input source to the data output sink in the modified algorithm marked graph must be executed once. The computation involved is TFC and the corresponding computing effort is TFCE. By Lemma 2.2, R * TBIO $\geq$ TFCE [19], or TBIO $\geq$ TFCE / R. This completes the proof.

Two graph execution features (GPST and TGP) and two hardware usage measures (REST and TRE) are now defined for predicting resource requirements. GPST describes the execution of transitions of the algorithm marked graph for a single data packet. REST is the description of the resource usage to process one data packet. TGP and TRE are the graph execution description and resource usage envelope when the algorithm marked graph is executed repeatedly and periodically.

Definition 2.25: GPST. GPST (graph play for a single task input) is a drawing depicting beginning, duration, and end of execution for each transition of the task when operated for a single data packet.

Definition 2.26: TGP. TGP (total graph play) is a drawing depicting beginning, duration, and end of execution for each transition of each task input at steady state when the AMG is executed periodically with an input data injection interval of TBO.

Definition 2.27: REST. REST (resource envelope for a single task input) is an envelope of resource usage by a single data packet between the time of task input and the completion of all computation associated with that task.

Definition 2.28: TRE. TRE (total resource envelope) is an envelope of resource usage to execute the graph at steady state with input period TBO.

Definition 2.29: Construction of GPST and REST. GPST and REST are generated by firing every transition in the algorithm marked graph at the earliest possible moment assuming unlimited resources and a single task input. Graph play is generated by depicting execution of all transitions in every time interval. Symbols (<, >) are used to show the beginning and the end of execution for a transition respectively. The resource usage envelope is obtained by counting the number of computing resources used during each time interval.

Example. Consider the algorithm marked graph of Figure 2.7. Transitions 1, 2, and 4 have duration of one time unit. Transitions 3, 5, and 6 have duration of two time units. The graph is played according to Definition 2.29 and the GPST is shown in Figure 2.8(a). The need for resources is the same as the number of active transitions in each time interval. The REST is computed by counting the number of resources used in each time interval and is shown in Figure 2.8(b).

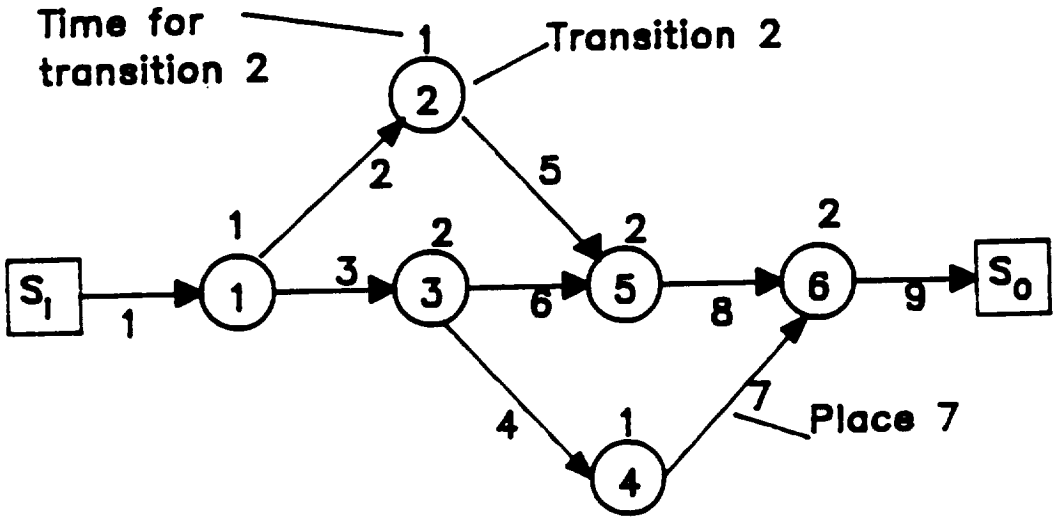
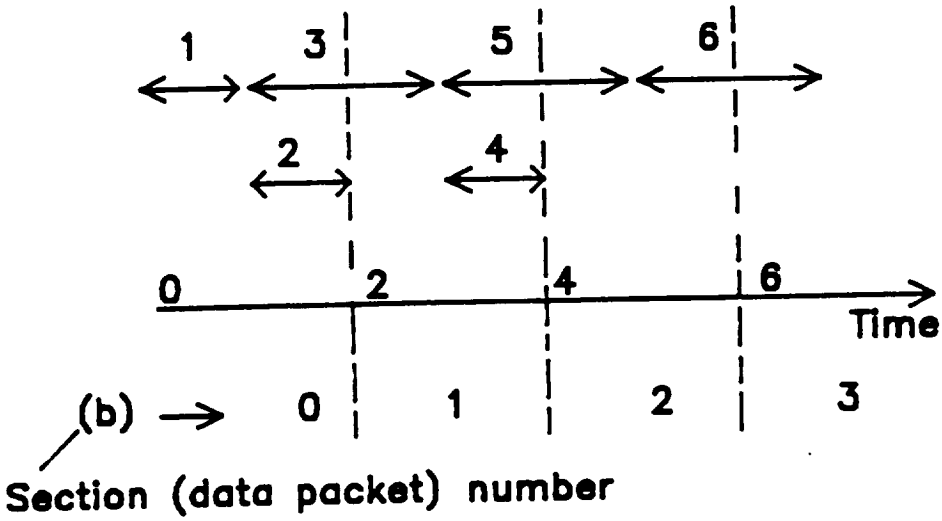
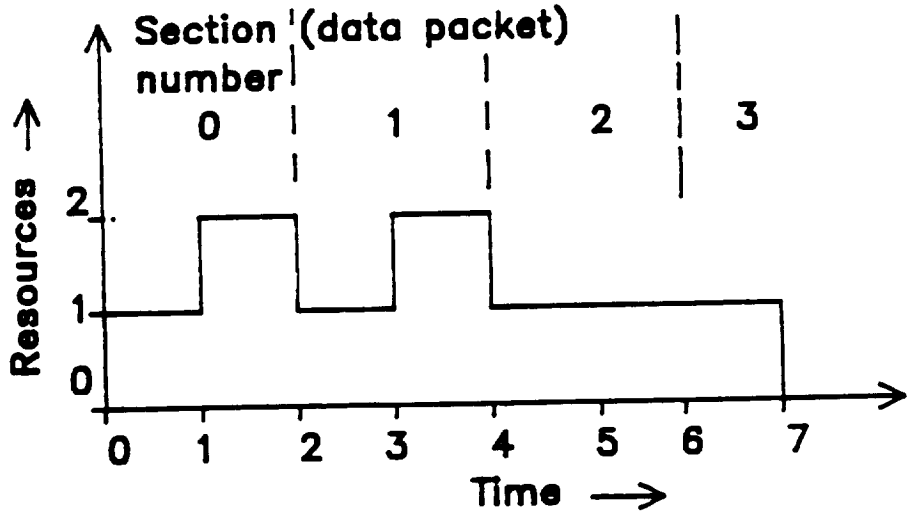


Figure 2.7. Algorithm marked graph for illustration of GPST and REST.



(a)



(b)

Figure 2.8. (a) GPST. (b) REST.

Now suppose the algorithm is executed periodically. Assume that the input data injection interval is long enough so that every data packet executes the graph as the GPST and needs resources over the task time as given by the REST. As a result, the algorithm is executed with a input period equal to output period TBO. The total resource envelope (TRE) is to be determined then by adding the resource needs of the concurrently processed data packets. The total graph play (TGP) is generated by drawing the execution of transitions from all the concurrently processed data packets. It is shown in the following two theorems that TRE and TGP are periodic with period TBO. If REST and GPST are divided from the beginning in sections of TBO time units, these sections are shown to be the contributions from the consecutive concurrent data packets towards a period of TRE and TGP. As an example, GPST and REST of Figure 2.8, are divided in sections of TBO = 2 time units. Section as well as data packet numbers are represented by the integer variable b. To illustrate, data packet 2 has been injected two time units before data packet 1. Moreover, transition 3, 2 for data packet 0, transition 5, 4 for data packet 2 and transition 6 for data packet 3 are executed concurrently at steady state requiring a total of five resources.

Theorem 2.6. When the algorithm marked graph is operated periodically for input period TBO with all data packets requiring resource envelopes identical to REST, the total resource envelope at steady state is periodic with period TBO and one period of TRE is generated by the summation of sections of REST of width TBO as follows.

Let REST (x) represent the resource envelope for a single task input where REST (x) = 0 for $x \geq TT$. Let the origin of time axis (t)

at steady state be the injection of a data packet. Let TRE (t) be the value of total resource requirement at time t. Let b represent the concurrently processed data packets at time t. A period of TRE(t) is then given by

$$TRE\ (t) = \sum_{b} REST\ (t + b * TBO),$$

where

$$0 \leq t < TBO$$

$$0 \leq b < \lceil TT\ /\ TBO \rceil .$$

Proof. By the rules of operation, data packets are injected and outputs are generated at the interval of TBO at steady state. Consider three consecutive data packets P, Q, and R injected at t = K * TBO, (K+1) * TBO and (K+2) * TBO respectively, where K is a positive integer. Let d be a time unit in which the total resource requirement is desired. Let s denote the time between d and time for the last data packet injection. Suppose d is a time between the injection of data packets P and Q. Thus K * TBO $\leq$ t < (K+1) * TBO, and s = t - (K * TBO). TRE(t) in this interval is made of REST's due to data packet P and previous data packets whose computations are completed after P has started. As all data packets have resource envelope identical to REST of duration TT, any data packet which is injected TT or more time before P has no effect on TRE in this interval. Consequently, the total number of concurrently processed data sets creating TRE(t) in this interval is given by $\lceil TT\ /\ TBO \rceil$.

Hence, let the range of b be $0 \leq b < \lceil TT / TBO \rceil$; b is an integer. TRE(t) for time interval between P and Q is then the summation of the resource requirements for these concurrently processed data packets. Let b = 0 identify task input P whose contribution to TRE (t) is REST (s). The data packet which has started TBO time units before P will contribute REST (s + TBO) and is identified by b = 1. In general, a data packet which is injected b * TBO time units before P is identified by the data packet number b and contributes REST (s + b * TBO) to TRE (t). Therefore, summing REST (s + b * TBO) over the entire range of b for the concurrently processed data packets will give the corresponding TRE (t). The data packet corresponding to the largest b may contribute to TRE(t) for only a partial interval. As REST (x) = 0 for $x \geq TT$, REST (s + b * TBO) properly represents the contribution due to the data packet corresponding to the largest b. Therefore, TRE (t) at d between P and Q is given by the following equation,

$$\begin{aligned} TRE\ (t) &= \sum_{b} REST\ (s + b * TBO) \\ &= \sum_{b} REST\ (t - K * TBO + b * TBO) \end{aligned} \qquad (2.4.1)$$

where

$$K * TBO \leq t < (K+1) * TBO$$

$$0 \leq b < \lceil TT/TBO \rceil .$$

Now let d be a time unit t + TBO from the origin. As d now is a time unit between data packet injection Q and R, s = (t+TBO) - (K+1)*TBO.

By similar arguments as before,

$$\begin{aligned} TRE\ (t + TBO) &= \sum_{b} REST\ (s + b * TBO) \\ &= \sum_{b} REST\ \{(t+TBO) - (K+1)*TBO + b * TBO\} \\ &= \sum_{b} REST\ (t - K*TBO + b*TBO) \\ &= TRE\ (t), \end{aligned}$$

from equation (2.4.1). Thus, TRE(t) is periodic with period TBO. Hence, it is sufficient to specify TRE (t) for one period only; let s = t, or K = 0. Modifying equation (2.4.1) we get,

$$TRE(t) = \sum_{b} REST\ (t + b * TBO)$$

where

$$0 \leq t < TBO$$

$$0 \leq b < \lceil TT/TBO \rceil .$$

Thus, one period of TRE(t) is generated by the summation of the sections of REST (x) of width TBO, starting from x = 0. The sections are identified by the corresponding value of b. This completes the proof.

Theorem 2.7. When the algorithm marked graph is operated periodically for input period TBO with all data packets executing the AMG as GPST, total graph play at steady state is periodic with period TBO and one

period of TGP is generated by the overlapping of sections of GPST of width TBO as follows.

Let GPST (x) represent the graph play for a single task input where $0 \leq x < TT$. Let the origin of time axis (t) at steady state be the injection of a data packet. Let TGP (t) be the total graph play at time t. Let b represent the concurrently processed data packets at time t. A period of TGP (t) is then given by,

$$TGP(t) = \sum_{b} GPST\ (t + b * TBO)$$

where

$$0 \leq t < TBO$$

$$0 \leq b < \lceil TT / TBO \rceil .$$

Proof. The proof is similar to Theorem 2.6 with one exception. Unlike REST, sections of GPST of width TBO represent portions of graph play for successive data packets which overlap to form TGP at steady state. Hence, instead of adding sections of GPST, one period of TGP should be constructed by overlapping sections of GPST with each section being identified separately by the value of b. If two values of b are i and i+1, it means data packet i+1 is injected TBO time units before data packet i. This completes the proof.

Example. One period of TGP and TRE is constructed for the AMG of Figure 2.7 according to Theorem 2.6 and 2.7 with an input period TBO of two time units. GPST and REST of Figure 2.8 are divided in sections of width two time units as shown in Figure 2.8 by the dotted

lines. Figure 2.9 shows the TGP and TRE for input period TBO of 2. Time t is any time when a new data packet is injected at steady state. In the TGP, the superscript of transitions indicate the value of b (data packet number). Data packet 1 is injected TBO time units before data packet 0. $1^{(0)}$ and $5^{(1)}$ represent the execution of transition 1 and 5 for the data packet 0 and 1 respectively in Figure 2.9(a). The TGP indicates that $5^{(1)}$ begins after the completion of $1^{(0)}$. As in GPST, (<, >) arrow symbols indicate the beginning and end for execution of a transition respectively. In Figure 2.9(a), transitions $3^{(0)}$, $5^{(1)}$, and $6^{(2)}$ have started in this period but did not end. Similarly $3^{(1)}$, $5^{(2)}$, and $6^{(3)}$ have been completed in this period but did not start in it. The resource usage in the four sections of REST in order of increasing b are (1, 2), (1, 2), (1, 1), and (1, 0). One period of TRE is calculated by adding the four sections of REST. The total resource need in one period of TRE is (4, 5) as shown in Figure 2.9(b). It is to be noted that TRE could also have been calculated from TGP by counting the number of active transitions in each time interval.

Lemma 2.6. Computing effort in one period of TRE is TCE at steady state when the algorithm marked graph is operated periodically with an input period of TBO.

Proof. As the algorithm marked graph is operated periodically, computing effort in every period is the same. Computing effort in a period TBO of TRE will equal TCE as one task output is generated in every TBO time units. This completes the proof.

Lemma 2.7. Resource Utilization (RU) in one period (TBO) of TRE is given by (TCE / (R * TBO)).

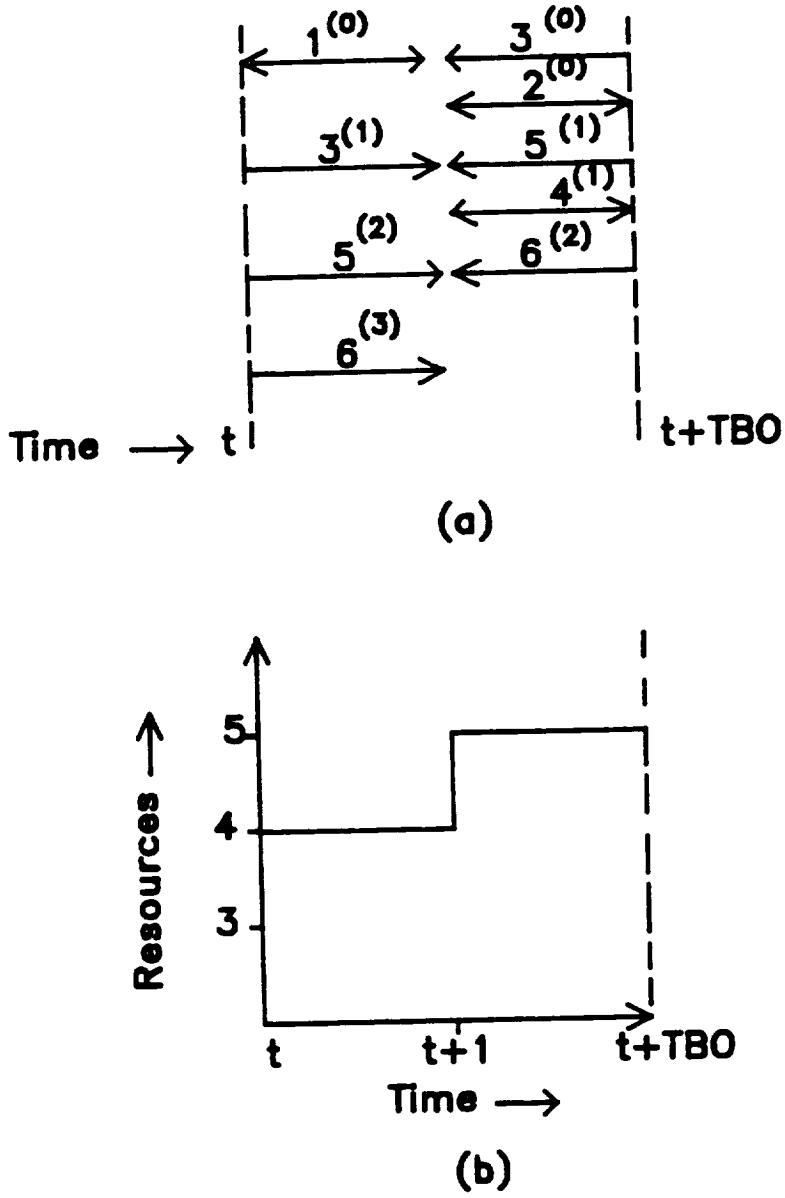


Figure 2.9. For TBO=2, (a) Total graph play. (b) Total resource envelope.

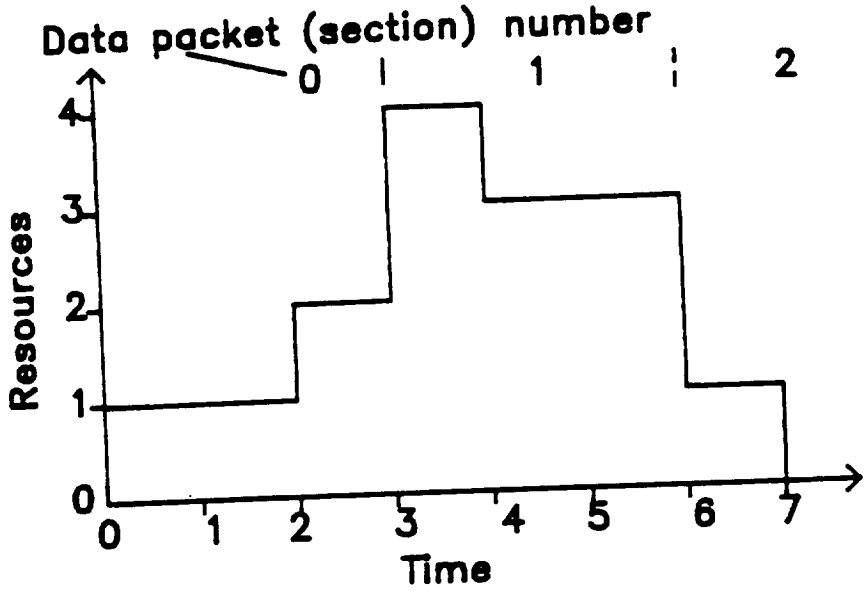
Proof. By Lemma 2.6, computing effort in one period (TBO) of TRE is TCE. Computing capacity in the TBO time interval is R * TBO. By definition then, resource utilization is (TCE /(R * TBO)). This completes the proof.

Example. Consider the REST as shown in Figure 2.10(a) with TT = 7, TC = 15 (ignore the dotted lines). The peak of REST is 4 which indicates that the ATAMM data flow architecture requires at least four functional units to process the task according to the REST in seven time units. Let TBO = 3. Tasks are initiated and outputs are generated at the interval of three time units with all having identical REST at steady state. TRE is calculated from Theorem 2.6. Dividing REST from the beginning in sections of width TBO, as in Figure 2.10(a), with the dotted lines, (1, 1, 2), (4, 3, 3), and (1, 0, 0) are the contributions of three overlapping task inputs to a period of TRE. Adding three sections of REST, a period of TRE is given by (6, 4, 5) and is shown in Figure 2.10(b). The computing effort in three time units of TRE is 15 as claimed by Lemma 2.6. Since the peak of TRE is 6, a minimum of six functional units is required to operate an algorithm marked graph with REST of Figure 2.10(a) and TBO = 3. By Lemma 2.7, resource utilization (RU) for six functional units is given by (15 / (6 * 3)) = .833.

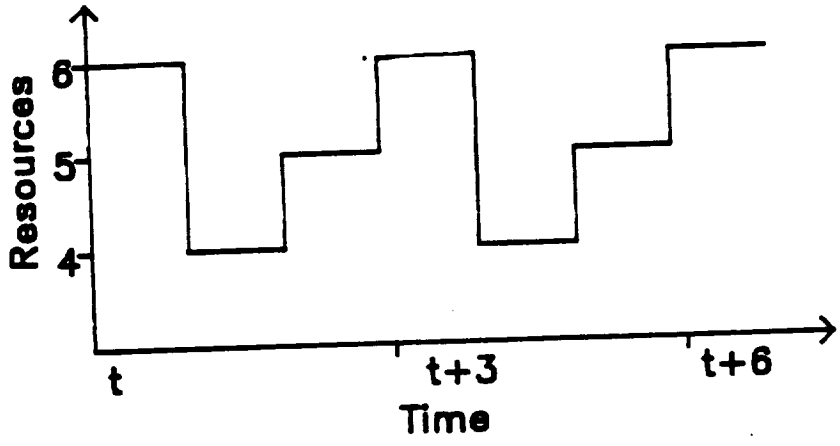
With the help of above lemmas, the resource imposed bound on TBO is established in the following theorem.

Theorem 2.8: Minimum TBO for R Resources. The minimum value of TBO for an algorithm marked graph operated periodically with R resources is always greater than, or equal to, TCE / R.

Proof. By Theorem 2.6, the total resource envelope is periodic. By Lemma 2.6, the computing effort needed in period TBO is TCE. The



(a)



(b)

Figure 2.10. (a) Resource envelope for a single task input. (b) Total resource envelope for TBO=3.

computing capacity for time interval of TBO is R * TBO. By Lemma 2.2, R * TBO ≥ TCE. Hence, TBO ≥ TCE / R. This completes the proof.

Corollary 2.8.1. The minimum value of resource requirements (R) for a desired TBO is bounded by ⌈TCE / TBO⌉ when the graph is operating periodically at steady state.

Proof. As TBO ≥ TCE / R, it follows that R ≥ TCE / TBO. Since R is an integer, R ≥ ⌈TCE / TBO⌉. This completes the proof.

Example. Consider the algorithm marked graph of Figure 1.1 and the corresponding modified algorithm marked graph of Figure 2.6. Let T(1) = 4, T(2) = 1, T(3) = 5, and T(4) = 6. The sum of all transition times are 16. Hence, TC = 16. TFC and TBC are calculated from the modified algorithm marked graph. Transitions 1, 2, and 3 appear in the forward paths from S_I to S_O. Therefore, TFC = T(1) + T(2) + T(3) = 10. As only transition 4 does not appear in any of the forward paths from data input source to data output sink, TBC = T(4) = 6. Also, TFC and TBC add up to TC. If only two functional units are available, the minimum values of TT, TBIO, and TBO are 8, 5, and 8 respectively. For a TBO of 7, the minimum R is ⌈TCE / TBO⌉ = 3.

2.5 Summary

The computing environment and performance measures in the ATAMM data flow architecture are established. Graph time performance is expressed by time between input and output (TBIO), task time (TT), and time between outputs (TBO). The modified algorithm marked graph is defined to compute lower bounds for TT and TBIO. Lower bounds for the performance measures are calculated analytically from the modified algorithm marked graph and the computational marked graph with the

assumption that a functional unit is available for every enabled transition to fire. The availability of a limited number of functional units is then considered. The modified algorithm marked graph is used to distinguish between forward computation (TFC) and backward computation (TBC) and to establish their relation to total computation (TC). Computing capacity, computing effort, and resource utilization are defined. The range of values for performance measures are established assuming that the ATAMM data flow architecture has only R functional units. The algorithm marked graph execution for a single task input or data packets periodically are defined in terms of GPST and TGP. The requirements of functional units to process a single task input or data packets periodically are expressed by REST and TRE. Resource utilization is defined; construction rule for GPST and REST are defined; and properties of TRE are described. Methodologies for generating TRE and TGP are established. All definitions and results are illustrated with examples.

CHAPTER THREE

ALGORITHM TRANSFORMATION

3.0 Introduction

The lower bounds for performance measures of an algorithm marked graph are developed in Chapter Two. One of the two remaining important problems concerning performance measures is considered in Chapter Three. Of interest is the potential of transforming an algorithm marked graph, with or without decomposition, in order to decrease lower bounds for performance. Investigation is also carried out to use transformations to reduce resource requirements, enforce periodicity in execution, and provide structural changes in the algorithm marked graph. All required transformation techniques, including an investigation of their usefulness and limitations, are described in this chapter. Algorithm transformation techniques are defined and elaborated in Section 3.1. Applications of algorithm transformations for performance improvements and reduction of resource requirements are discussed in Section 3.2. A steady state periodic execution of algorithm marked graphs is realized in Section 3.3. Structural changes of algorithm marked graphs are considered in Section 3.4. A summary of the chapter is presented in Section 3.5.

3.1 Algorithm Transformation Guidelines

The aim of this section is to define algorithm transformation techniques and illustrate their significance. Algorithm

transformation is defined to be a process to change some features of an algorithm marked graph while preserving its equivalence in computations. In other words, algorithm transformations produce a new AMG which is equivalent to the original AMG but better in some respect. The primary objectives are to improve time performance and lower resource requirements through algorithm transformation. Therefore, algorithm transformation techniques which can lower critical path length, lower time per token for the critical circuit of the CMG, lower resource requirements, and enforce periodicity in the execution of the AMG are of great interest. A formal definition of equivalency of two algorithm marked graphs and algorithm transformation techniques are stated and explained below.

Definition 3.1: Equivalency Of Two Algorithm Marked Graphs. Two algorithm marked graphs are equivalent if they map any set of input variables into the same set of output variables and produce an identical output sequence for an input sequence.

Definition 3.1 specifies the allowable transformations. An algorithm marked graph can be transformed as long as the new AMG is input-output equivalent with the old one. It is to be noted that if the computations of transitions and data dependency among the transitions of the original AMG are not altered, the transformed AMG will remain input-output equivalent with the original AMG. Definitions 3.2 through 3.5 describe four transformation techniques which are based on this observation.

Definition 3.2: Control Place. A control place is any place in the algorithm marked graph whose deletion generates an equivalent algorithm marked graph.

A control place is an artificial place in the sense it is not necessary for the correctness of an algorithm. A control place imposes a precedence relation among two transitions. The control place needs to be initialized by an initial token if it creates a circuit in the algorithm marked graph. The designer inserts a control place in the algorithm marked graph to delay the firing of a transition. All places in the AMG other than control places will be called active places henceforth. If broadcasting is used to transmit data between transitions, insertions of control places are not going to change read and write times of transitions. Also, control places need not transmit data vectors; therefore they can be implemented at very low communication cost. Thus for analyses purposes, insertion of control places in an AMG will be assumed not to increase read and write times of transitions.

Definition 3.3: Dummy Transition. A dummy transition is any transition in the algorithm marked graph which is not required for executing a primitive operation.

A dummy transition is a redundant transition in the sense that it is not required for computation. However, it can be used to control operation or improve performance. All transitions other than dummy transitions will be called active transitions henceforth. A dummy transition can act as a buffer to provide storage for the output of any transition. Such buffers will be shown to be needed at times when the algorithm marked graph is operated periodically. A dummy transition can be used to combine input or output data vectors in order to create single input or output vectors respectively. Another application of a dummy transition is as a delay operator for holding

firing of one, or a group of, transitions. Read and write time for the NMG of a dummy transition depend on implementation and data length, but should be less, or equal to, read or write times of an active transition of equal data length respectively. A dummy transition has zero process time when it is used as a buffer; it has a very small process time when it is used for combining data vectors. A dummy transition as a delay operator has a process time corresponding to the amount of delay needed. As operations are restricted to large-grain algorithms, read and write times are expected to be significantly smaller than the process time of an active transition. Thus for analyses purposes, a dummy transition will be assumed to have zero time when it is used as a buffer or for combining data vectors. Also, it will be assumed that a dummy transition for applications other than a delay operator does not require a resource because a resource is required to implement such a dummy transition for a very short time. A dummy transition for delay application has not been explored in detail in this dissertation, but poses an interesting topic for future research.

Definition 3.4: Predefined Token. A predefined token is any initial token on a place of the algorithm marked graph.

A predefined token indicates the presence of a precomputed initial data or initial control. A predefined token is necessary at times for execution of the task and for forward flow of data.

Definition 3.5: Decomposition of a Transition. Decomposition of a transition in the AMG is to replace the transition by an equivalent marked graph of a group of transitions.

The transition decomposition of Definition 3.5 is to distribute the computation of a transition among a group of transitions in order

to reduce the original transition time. This is important because large transition times are major contributors to critical path length and time per token of critical circuits. It should also be noted that the decompositions of transitions are not always reasonable or possible due to added communication cost, higher resource requirements, and transition characteristics. Serial, or a combination of serial and parallel, decompositions of a transition tends to decrease TBO_{LB} significantly while $TBIO_{LB}$ does not improve much and can even increase due to added serial communication time. In those cases, a proper decomposition is dependent upon the relative importance of TBO and TBIO. Pure parallel decomposition of transitions decrease both TBO_{LB} and $TBIO_{LB}$.

Subsequent sections of this chapter will develop a theoretical basis for the applications of control places, dummy transitions, initial token and decomposition. A software program, called Ttime [20], will be used for determining lower bounds for TBO, TT, and TBIO. This program constructs the CMG from the specified AMG to determine TBO_{LB}. Two examples are presented to illustrate the transformation of an AMG through the use of control places and dummy transitions.

Example. Consider the algorithm marked graph of Figure 2.2. The corresponding CMG is shown in Figure 2.4. A transformed AMG and corresponding CMG are shown in Figures 3.1 and 3.2 respectively. A dummy transition of zero time is used as buffer between transition 1 and 6. The AMG's of Figures 2.2 and 3.1 are equivalent as they produce the same output sequence for identical input sequences. The dummy transition provides an additional storage space for the output

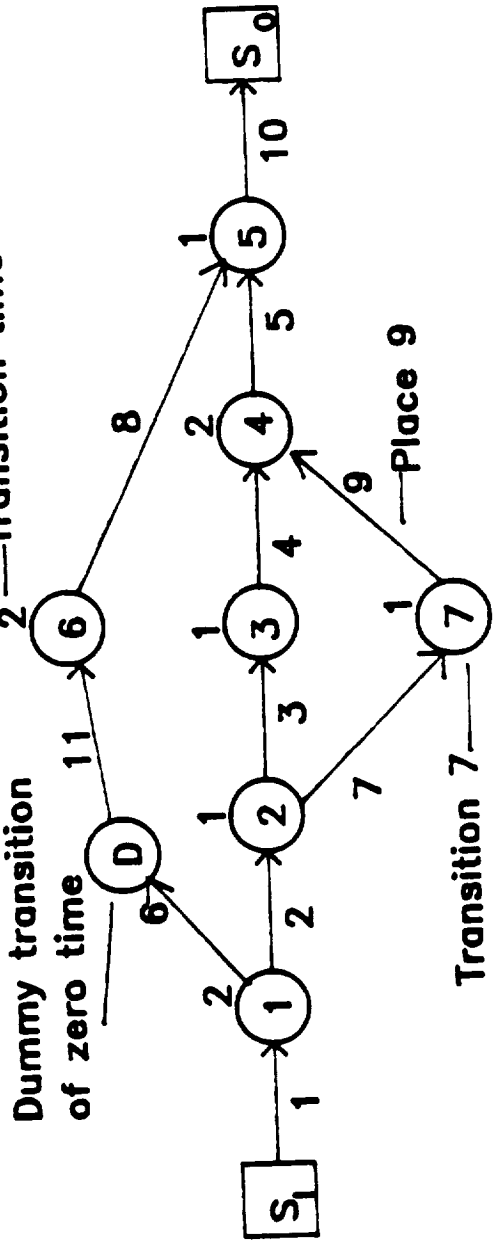


Figure 3.1. Transformed algorithm marked graph in Application 1.

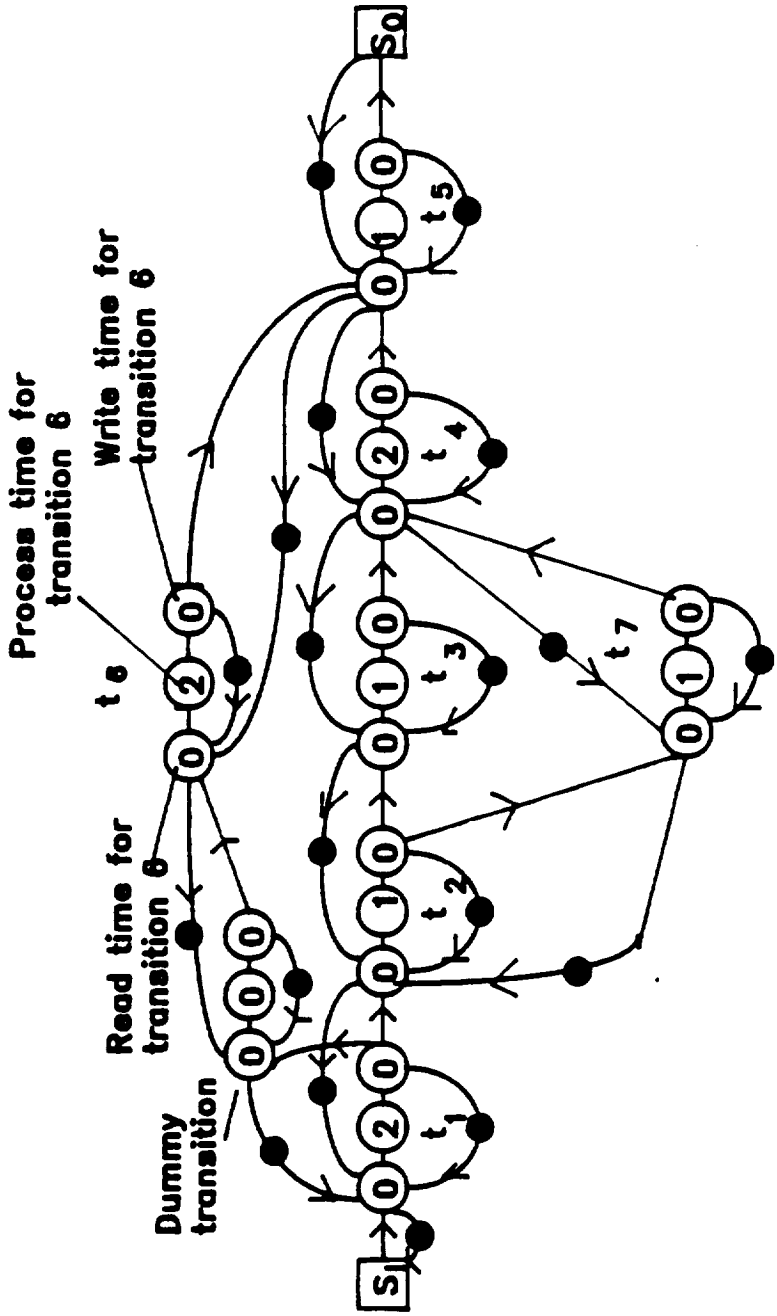


Figure 3.2. Computational marked graph for the transformed AMG.

of transition 1, which is to be used as an input of transition 6. Without this dummy transition, transition 1 can fire only once before transition 6 fires; however, with the dummy transition, transition 1 can fire again before transition 6 fires. Application of this transformation will be described later.

An example of transformation by control places is shown in Figures 3.3 and 3.5. Control places delay firing of selective transitions and therefore modify REST and TRE. The dummy transition is used again as a buffer. Improvement due to this transformation will be described later.

3.2 Performance Improvements by Transformation

Applications of dummy transitions and control places for improving time performance and reduction of resource requirements are discussed in this section. New results are stated in Application 1 and 2. Application 1 describes how dummy transitions can reduce TBO_{LB} of an AMG to the largest time/token among the process and recursion circuits. Application 2 describes how the REST of an AMG can be modified to give a lower peak TRE through the use of control places.

Application 1. This is an application where a dummy transition is used as a buffer. A dummy transition can provide storage space for the output of a transition. This can increase the firing rate of transitions as ATAMM does not allow firing of an active transition unless its outputs are read by successor transitions. In terms of the CMG, a dummy transition can increase the number of tokens in the circuits of an CMG created by parallel paths in the AMG. This is the basis for Theorem 3.1.

Theorem 3.1: Reduction of TBO_{LB} to the Largest Time Per Token Among the Process and Recursion Circuits by Dummy Transition. Any AMG can be transformed by using dummy transitions as buffers so that

$$TBO_{LB} = \text{Max } \{T(C_i)/M(C_i)\} \qquad (3.2.1)$$

where $T(C_i)$ and $M(C_i)$ denote the sum of transition times and the number of tokens contained in C_i of the CMG respectively. Circuit C_i is a process or recursion circuit.

Proof. There are four kinds of circuits in a CMG, as mentioned in Section 2.2. They are node circuits, process circuits, recursion circuits, and parallel path circuits. Theorem 2.3 has proved equation (3.2.1) when C_i is any directed circuit of the CMG. From ATAMM model characteristics, as described in the Appendix, both node and process circuits always have only one token. Also the sum of transition times for process circuits are always greater than, or equal to, that of their corresponding node circuits as process circuits include the successor read transition. Consequently, the largest time/token ratio of process circuits is always greater than, or equal to, the largest time/token ratio of node circuits. The remaining task is to show that the time/token ratio for circuits in a CMG due to parallel paths in the AMG can be reduced sufficiently to make them insignificant in determining TBO_{LB}. Consider any two parallel paths P_i and P_j of the AMG which begin and end at transitions S and E respectively. Consider the parallel path circuit in the CMG created by forward directed places (for data flow) from NMG

transitions of path P_i and backward directed places (for control flow) from NMG's of path P_j. Each of these backward directed places has a token in the initial marking. The number of such backward directed places are one more than the number of transitions on path P_j, excluding transitions S and E. Inserting a dummy transition of zero time on path P_j will increase the number of tokens in this circuit by one. As this dummy transition does not have any time, it cannot increase the $T(C_i)$ of this circuit or any other. Hence, the time/token ratio of this circuit will decrease while not increasing the time/token ratio of any other circuit. By inserting more dummy transitions on path P_j, the time/token ratio for this circuit can be arbitrarily reduced. If the time/token ratio for this circuit is greater than the largest time/token ratio from process or recursion circuits, dummy transitions can be used to reduce the time/token ratio to a value lower than, or equal to, the largest time/token ratio among process or recursion circuits without increasing the time/token ratio of any other circuit. Following this procedure, sufficient dummy transitions may be added so that the time/token ratio for any parallel path circuit in the CMG is smaller than, or equal to, the largest time/token ratio among process or recursion circuits. The procedure is guaranteed to terminate as dummy transitions, when used as buffers, never increase the time/token ratio of any circuit. This completes the proof.

Example. Consider again the AMG of Figure 2.2. The corresponding CMG is drawn in Figure 2.4 assuming zero time for read and write transitions. Therefore, TBO_{LB} is 3. There is no recursion circuit in the AMG. The largest time/token ratio among all process circuits

is 2 and the largest time/token ratio among node circuits is 2. However, the largest time/token ratio among all directed circuits is 3 due to two parallel path circuits as shown in Figure 2.4. For both of these circuits, parallel paths in the AMG start and end in transitions 1 and 5 respectively. Let t_i denote transition i and p_j denote place j. Path P_j for both circuits is the forward path t_1, p_6, t_6, p_8, and t_5. Path P_i for the two parallel path circuits are t_1, p_2, t_2, p_3, t_3, p_4, t_4, p_5, and t_5, and t_1, p_2, t_2, p_7, t_7, p_9, t_4, p_5, and t_5 respectively. Both of these circuits have two tokens from backward directed places from the NMG transitions of path P_j, as shown in the CMG. Now the AMG is transformed by inserting a dummy transition on path P_j as shown in Figure 3.1. The corresponding CMG is shown in Figure 3.2. The number of tokens on the parallel circuits are now 3 and therefore the time/token ratio is 2. Time/token ratio for any other circuit does not increase as the dummy transition has zero time. The largest time/token ratio over all directed circuits is now 2. However, TBO_{LB} for the AMG of Figure 3.1 is 2, and transformation by a dummy transition has improved throughput performance.

Application 2. This is an application to demonstrate a procedure for reducing resource requirements. Control places and dummy transitions are the two transformation techniques which are used. Suppose that all the data sets of an AMG require resource envelope, as given by REST, and data sets are injected at the interval of TBO time units. The total resource envelope will then be given by TRE and the peak value of TRE will be the required number of functional units. From Chapter Two, TRE is periodic and one period of TRE is made by

additions of sections of REST of width TBO. This immediately leads to the possibility that the peak value of TRE might be lowered by adjusting the shape of REST if the peak value of TRE is more than the minimum requirement $\lceil TCE/TBO \rceil$. REST can be modified by delaying active transitions selectively with the help of control places. This may or may not lead to an increase in TT_{LB} (thereby duration of REST) or $TBIO_{LB}$ depending on the "float" of delayed active transitions. Float is the amount of time an active transition can be delayed without increasing $TBIO_{LB}$ and TT_{LB}.

A desired result is to modify REST without increasing $TBIO_{LB}$ and TT_{LB} to achieve TBO_{LB} with a minimum number of resources. Unfortunately, this problem is equivalent to a class of scheduling problems which is known to be NP complete [12]. Thus, REST must be modified heuristically by control places. Judicious insertion of control places may reduce the resource requirement for the same TBO_{LB}, but perhaps at the expense of $TBIO_{LB}$. A control place is useful if it can reduce resource requirements by delaying transitions with float or by sacrificing parallel concurrency to some extent. Lastly, insertion of control places in the AMG can create dominant parallel path circuits in the corresponding CMG which are made insignificant following the procedure of Application 1.

The methodology for lowering the resource requirement is now stated. First, construct REST and TRE for the AMG at specified TBO. The peak value of TRE is the resource requirement for an input data injection interval of TBO. If the peak value of TRE is more than $\lceil TCE/TBO \rceil$, heuristically modify REST by transforming the AMG with control places with as small an increase in $TBIO_{LB}$ and

TT_{LB} as possible. Make all dominant parallel path circuits created by control places insignificant by adding dummy transitions. An example is given below to illustrate Application 2.

Example. Consider the algorithm marked graph of Figure 3.3. From the AMG, TCE = 12, TBO_{LB} = 2, and $TBIO_{LB}$ = TT_{LB} = 6. The minimum resources to achieve TBO_{LB} are $\lceil TCE / TBO_{LB} \rceil$ = 6. REST is shown in Figure 3.4. Adding sections of REST of width TBO_{LB}, a period of TRE is computed and is shown in Figure 3.4. The peak value of TRE is 9. Hence, nine functional units are required for implementing this AMG for optimum time performance. As the minimum resource requirement for TBO_{LB} is 6, Application 2 is considered. The AMG is transformed heuristically, as shown in Figure 3.5. The dotted lines are control places 1 through 4. Ignore control places 2, 3, and 4 initially. The justification of control place 1 is as follows. It is noted that transition 5 is the only transition which has a float in the AMG. Transition 5 can be delayed up to two time units without delaying the output. Considering section 1 of REST as shown in Figure 3.4, transition 5 should be delayed one time unit so that the peak value of TRE is reduced to 8. This is accomplished by control place 1. The modified REST and TRE are shown in Figure 3.6. Unfortunately, control place 1 creates a parallel path circuit among transitions 1, 4, and 5 whose time/token ratio is more than 2. The time/token ratio of this circuit is made less than 2 by inserting a dummy transition on the place between transition 1 and 5. Now consider section 2 of REST as shown in Figure 3.6. It contributes (4, 1) to a period of TRE. In order to reduce the peak value of TRE, a more equal distribution of transitions among the time intervals (t, t+1) and (t+1, t+2) of TRE

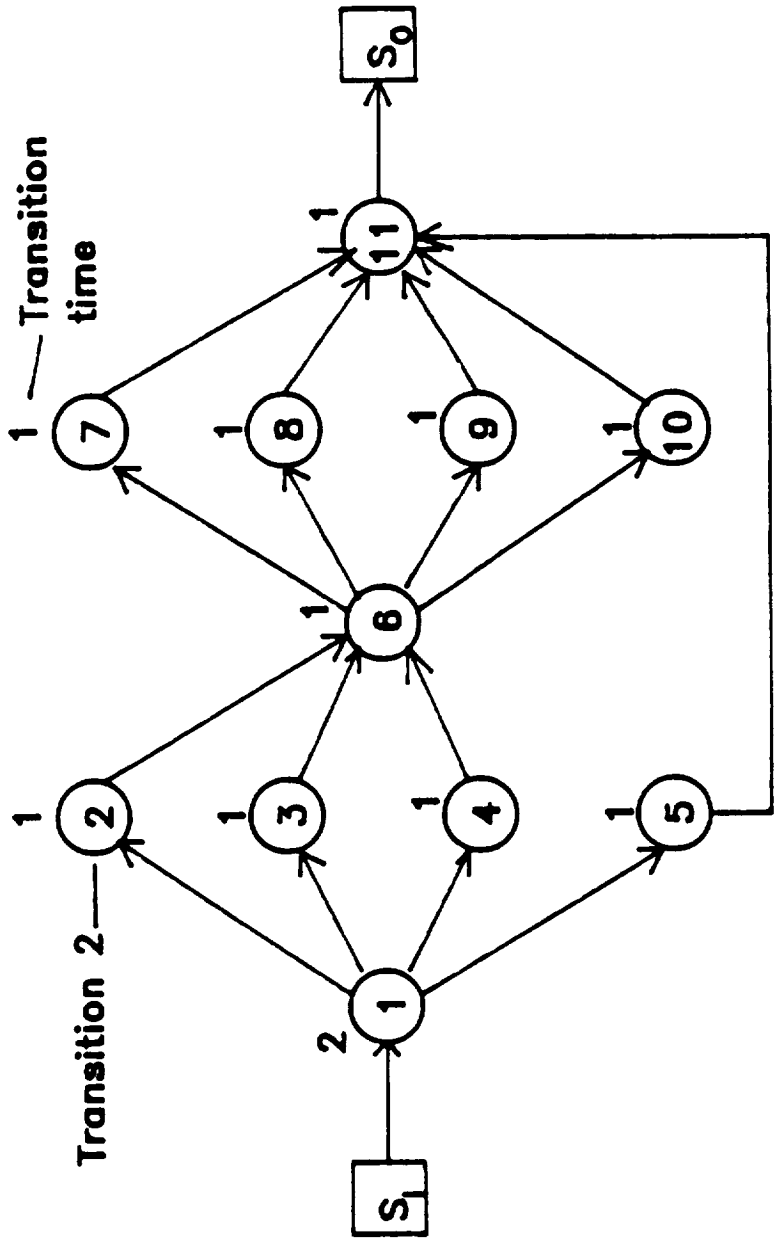


Figure 3.3. AMG for illustration of Application 2.

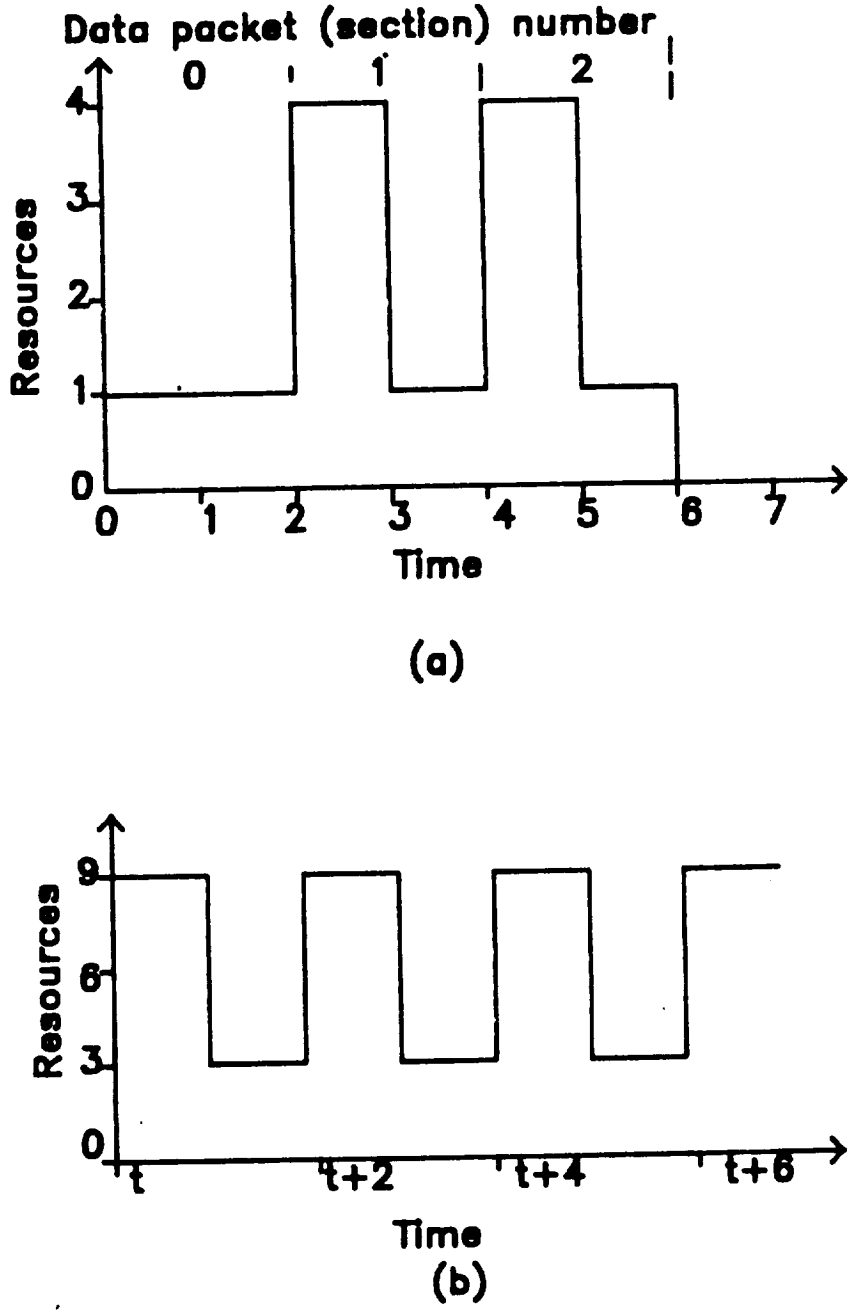


Figure 3.4. For Figure 3.3, (a) REST.
(b) TRE for TBO=2.

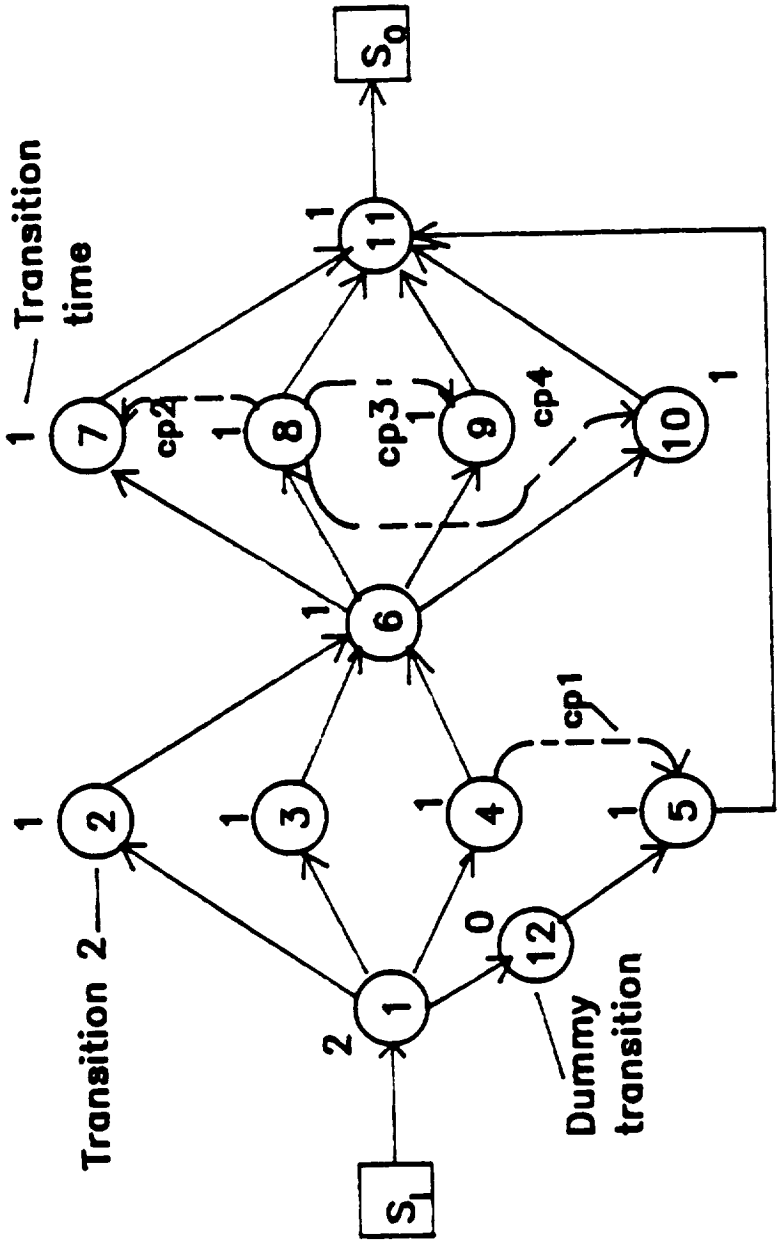
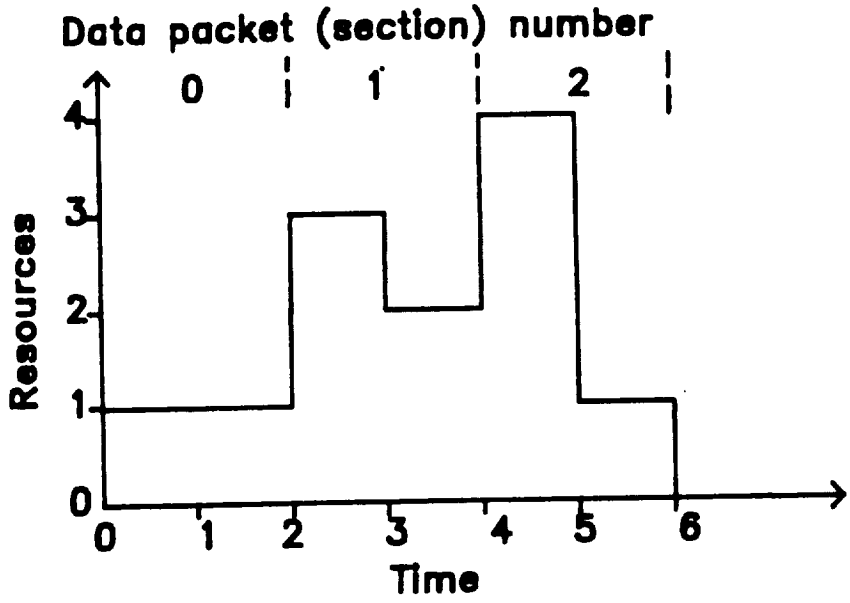


Figure 3.5. Transformed AMG for Figure 3.3.



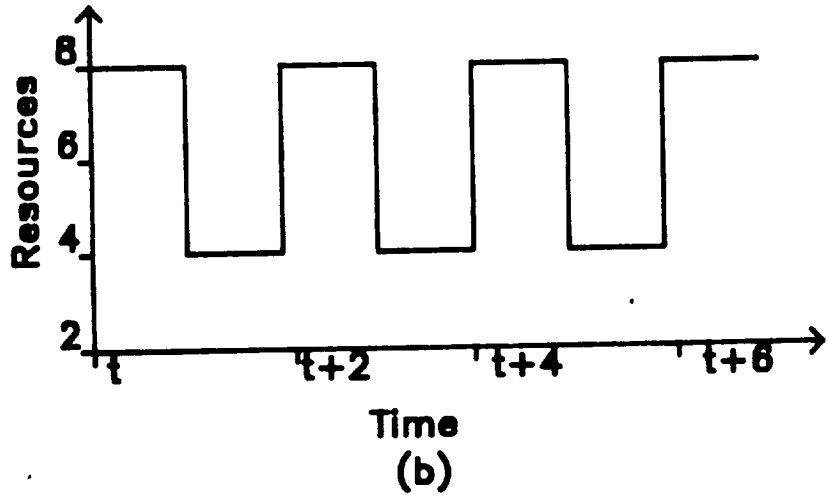


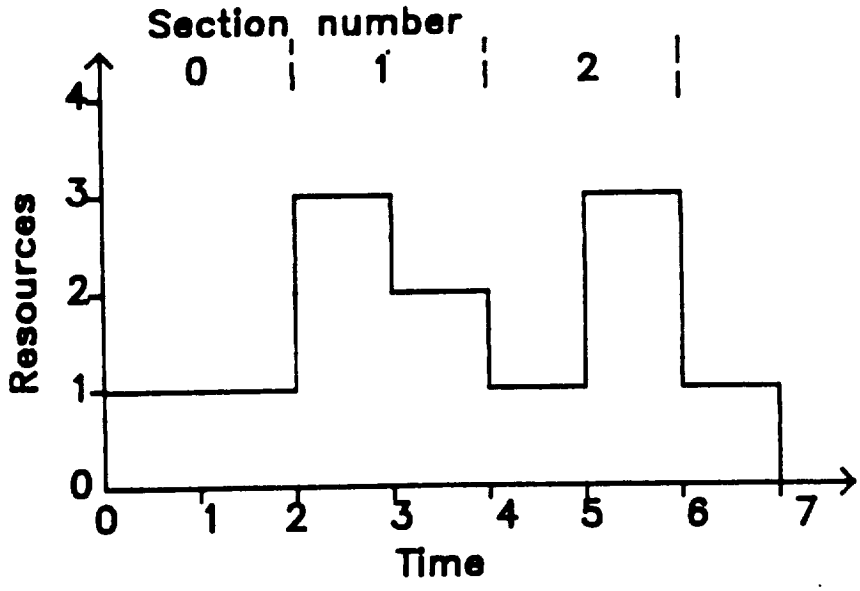
Figure 3.6. For the AMG transformed by control place 1, (a) REST. (b) TRE for TBO=2.

is needed. Control places 2, 3, and 4 do this job at the expense of increasing $TBIO_{LB}$ and TT_{LB} by one time unit. The REST and TRE of the fully transformed AMG of Figure 3.5 are shown in Figure 3.7. Now only six functional units are required, which is the minimum for a TBO_{LB} of 2. It is to be noted that the maximum utilization of resources may not be achievable by use of control places in all cases unless the AMG is turned into a complete chain.

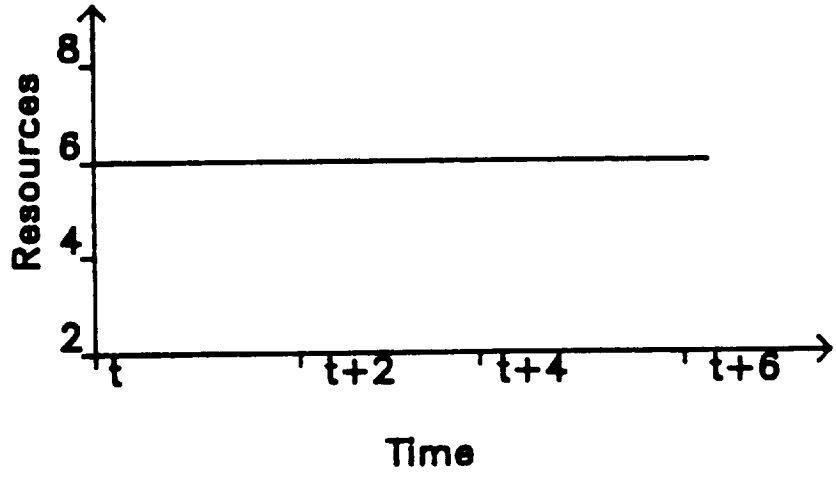
3.3 Implementation Of Periodicity By Transformation

This section describes a procedure for enforcing periodicity in the execution of an algorithm marked graph for successive data sets. It is desired that performance and resource needs be identical for all data sets for two reasons. First, input data should not experience a waiting time on the critical path of a task so that $TBIO_{LB}$ is achieved for all data sets. Second, the resource envelopes for all data sets should be identical so that the total resource need can be predicted. It will be shown in Application 3 and 4 of this section that by controlling input data injection and transforming the AMG by dummy transitions, periodicity can be realized in the execution of the AMG. The need and methodology for injection control of input data is explained in Application 3. Application 4 describes the conditions for operating an AMG periodically with each data packet having identical resource envelopes.

Application 3. When presented with continuously available input data sets, the natural behavior of a data flow architecture results in operation where new data sets are accepted as rapidly as the available resources and the input transition of the AMG permits. From Chapter



(a)



(b)

Figure 3.7. For the transformed AMG with all control places, (a) REST. (b) TRE for TBO=2.

Two, the output of the AMG cannot be generated at a higher rate than $1/TBO_{LB}$ or R/TCE. Therefore, if the data sets are continuously available, they experience a waiting time inside the architecture which increases TBIO from $TBIO_{LB}$. That is, the architecture will naturally operate at high levels of pipeline concurrency with the possible loss of capability for achieving high levels of parallel concurrency. This will result in performance characterized by high throughput rates, but relatively poor task computing speed. In many control and signal processing applications, it is important to achieve both a high throughput rate and high task computing speeds. Therefore, it is necessary to control injection rate of data sets so that input data never waits on the critical path. The input data injection interval must always be greater than, or equal to, TBO_{LB} and it should be such that all task inputs always have a resource available to fire transitions on the critical path to the data output sink. This can be accomplished by either adjusting the time for the source transition or as shown in Figure 3.8. It is not always easy to adjust the source transition time as this will be the sampling interval of sensors in a real system. All that is required is to limit the rate at which new input data are presented to the CMG. This is done in Figure 3.8 by adding a dummy transition in a directed circuit with the data input source. The predefined token on the directed circuit is for initialization. The dummy transition imposes a minimum delay of D time units between inputs. D is chosen to be the designer specified TBO.

Application 4. It is necessary that all data sets have the same resource envelope so that the total resource requirement can be

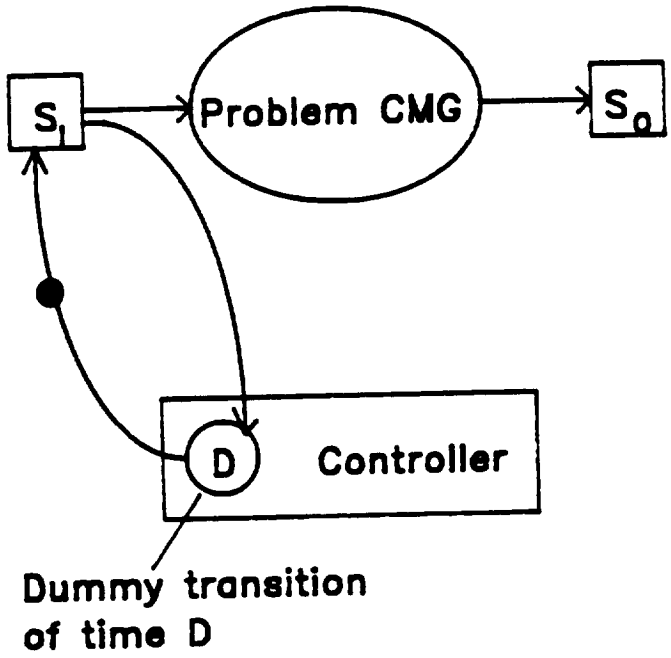


Figure 3.8. Injection control by Application 3.

predicted. Also at steady state, it is desirable that all data sets require resource envelopes identical to REST as REST can be modified to lower the peak value of TRE as described in Application 2. In order to achieve such a resource envelope, all transitions of the AMG should fire as soon as there is a token on every input place. The first step is to control the data injection interval as discussed in Application 3. If this condition is satisfied, then it can be guaranteed that a data token never waits on the critical path from the data input source to the data output sink for all data sets. Hence, $TBIO_{LB}$ is achieved for all data sets. Secondly, the resource envelope for a data set of an AMG at steady state may not be identical to the REST even though injection is controlled for the following reason. Whenever there are parallel paths in the algorithm marked graph, the transitions on non-critical paths of the algorithm marked graph will have a float associated with them. The float of a transition is the time by which a transition can be delayed without increasing $TBIO_{LB}$ and TT_{LB}. If there is not enough storage space for previous data, transitions in the AMG with float may not fire even though all the input places have token. The reason is that one or more output places of the transition contain previous data. This will change the steady state resource envelope from the REST. One way to prevent this from happening is to use control places to eliminate all floats from the AMG. However, this may not be always possible as any control place has to be generated from the completion of execution for a transition. Also, use of control places may require dummy transitions to prevent TBO_{LB} from increasing, which will make the AMG more complex. A better way of enforcing REST for all data sets

is to use dummy transitions as buffers in the output of transitions with float which need more storage space for previous data. The position and number of dummy transitions can be determined from TGP based on GPST. As the input injection interval is greater than, or equal to, TBO_{LB}, REST should be enforced for the injection interval of TBO_{LB}. This will also guarantee REST for all data sets with any higher injection interval. The reason is that transitions are executed at a lower rate for a higher injection interval and need for storage space at the output of floating transitions will be lower. The detailed procedure is now stated below.

Construct the TGP based on REST for TBO = TBO_{LB}. Locate all transitions with float and identify their corresponding task input number. By inspection of TGP, check whether all the successors of a floating transition for the previous task inputs have fired before the floating transition fires. If not, the floating transition needs dummy transitions as buffers at its output. The number of required dummy transitions equals the number of previous task inputs for which at least one of the successor transition has not fired at the time of firing of the floating transition.

Example. Consider the algorithm marked graph of Figure 3.9. From the AMG, TBO_{LB} = 2 and $TBIO_{LB}$ = TT_{LB} = 5. Only transition 5 has a float of two time units. GPST and TGP for TBO = TBO_{LB} = 2 are shown in Figure 3.10. Task input 1 has started TBO_{LB} before task input 0, and task input 2 has started another TBO_{LB} before task input 1. The successor of floating transition 5 is transition 4. Another predecessor of transition 4 is transition 3. Notice from the TGP that $4^{(2)}$ has started before $5^{(0)}$; $3^{(1)}$ begins with $5^{(0)}$. As $4^{(1)}$ is

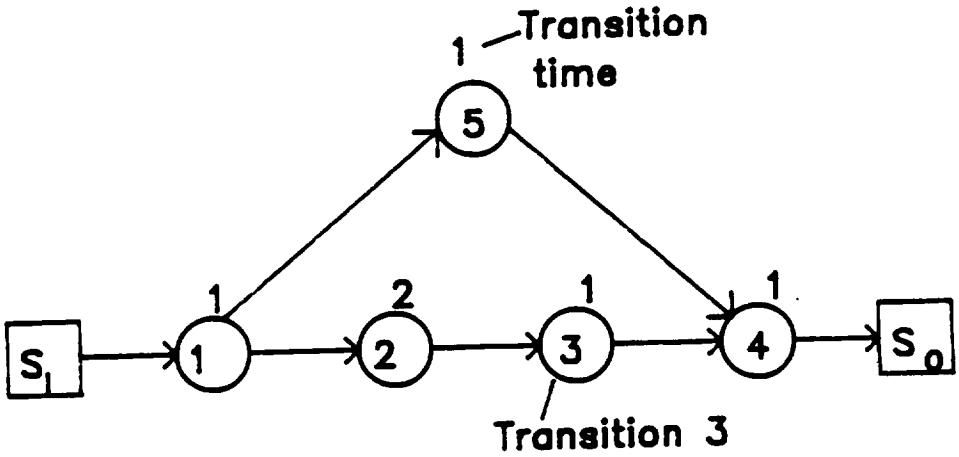


Figure 3.9. Example AMG for illustration of Application 4.

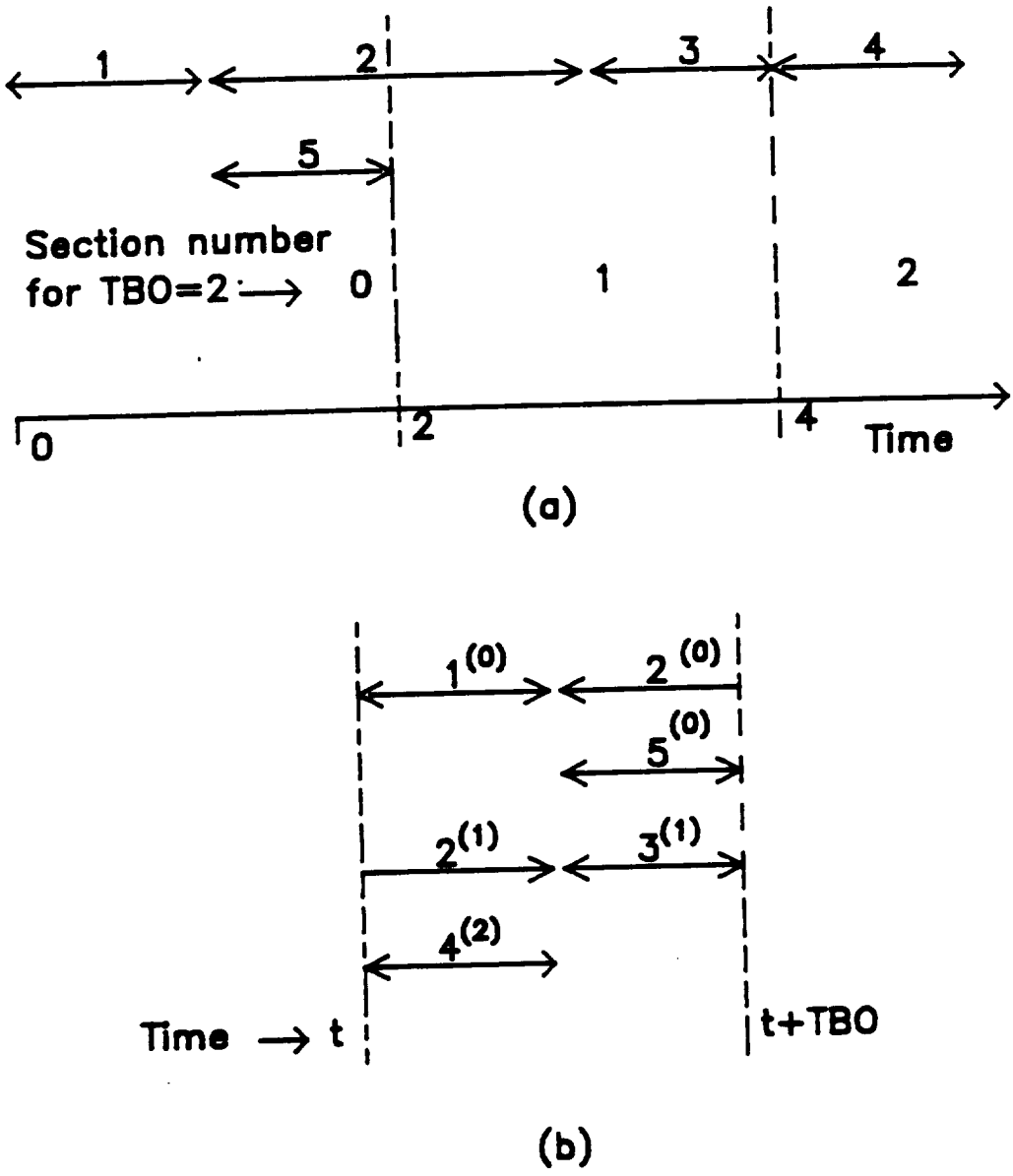


Figure 3.10. (a) GPST. (b) TGP for TBO=2.

executed after $3^{(1)}$ in the GPST, $4^{(1)}$ has not started before $5^{(0)}$. Hence, one dummy transition is needed at the output of transition 5 to store $4^{(1)}$ so that $5^{(0)}$ can fire according to the GPST. Otherwise, the firing of $5^{(0)}$ will be delayed as the NMG model of a transition does not allow the firing of a transition unless the output buffer is empty. The transformed AMG is shown in Figure 3.11(a). The TGP for TBO = 3 is shown in Figure 3.11(b). Transition 5 no longer needs a dummy transition in the output for enforcing REST. Hence, the transformed AMG of Figure 3.11(a) enforces REST for TBO equal to both 2 and 3.

3.4 Structural Changes In Algorithm by Transformation

The transformations considered so far try to preserve the original structure of an algorithm marked graph. In certain conditions that may not be possible, or desirable. For example, it is possible to improve TBO_{LB} of linear time invariant systems by modifying the state equations. In this section, three kinds of structural changes of algorithms are considered in Application 5 through 7. Application 5 explains how multiple input-output algorithms or a group of algorithms can be combined into a single input-output algorithm. This is necessary because the analysis tools developed in this dissertation are based on single input-output algorithms. Improvement of throughput by modifying the state equations of linear time-invariant systems is demonstrated in Application 6. Application 7 considers the parallel decomposition of transitions as a way of improving performance.

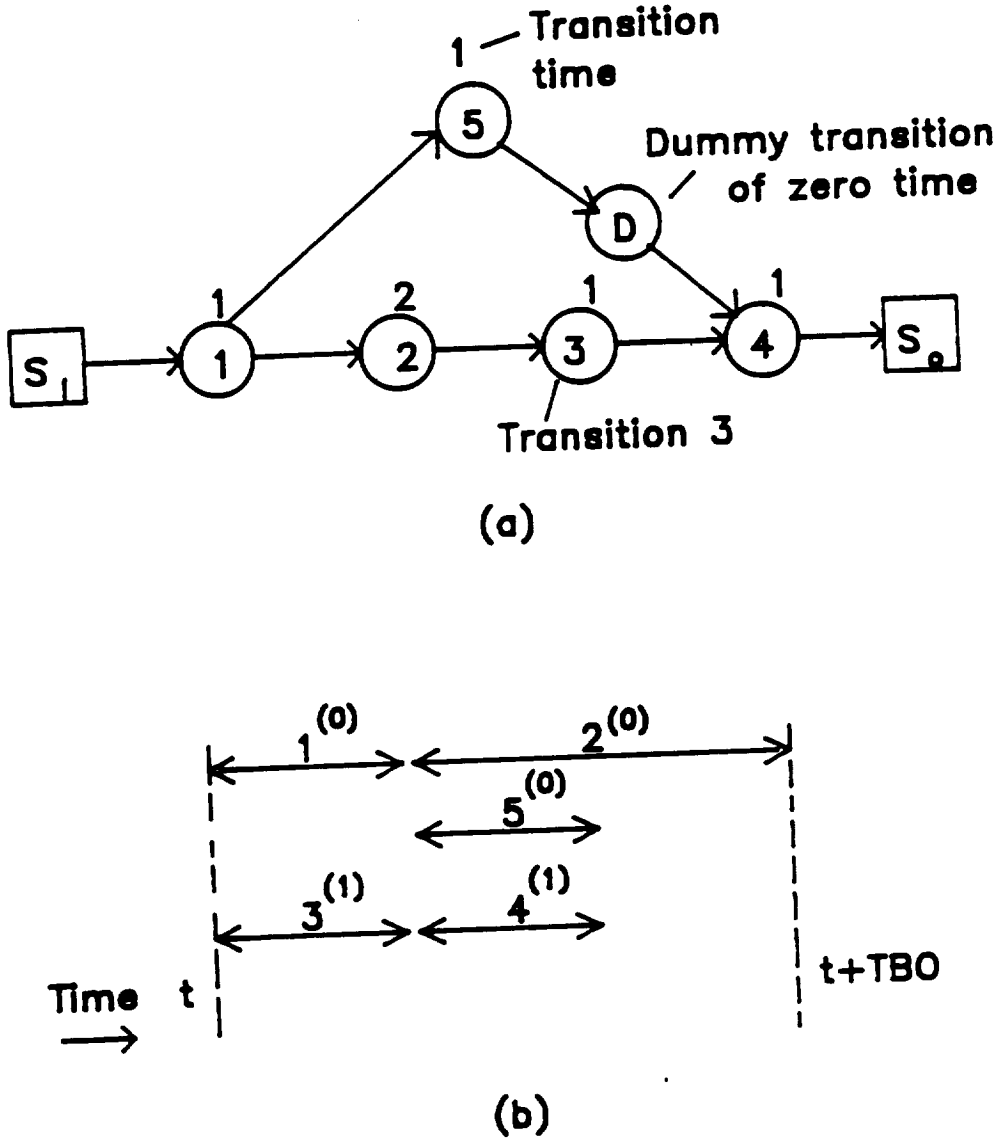


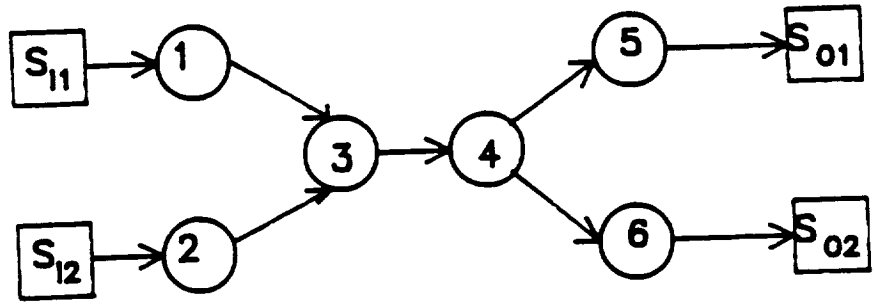
Figure 3.11. (a) Transformed AMG. (b) Total graph play for TBO=3.

Application 5. The performance model of Chapter Two considers only single input and single output algorithms. The addition of dummy transitions provides a way of converting multiple input-output algorithms or a number of algorithms into one single input-output algorithm. A dummy transition is used to combine input data vectors or output data vectors. All the inputs are synchronized and fed to the dummy transition at the same rate. Performance is evaluated from the combined algorithm which represents the total task. Two examples are shown in Figures 3.12 and 3.13. In Figure 3.12, AMG A_1 has two inputs and two outputs. It is transformed into a single input-output algorithm A_2 by dummy transitions. Figure 3.13 shows how dummy transitions can be used to combine two algorithms into one algorithm.

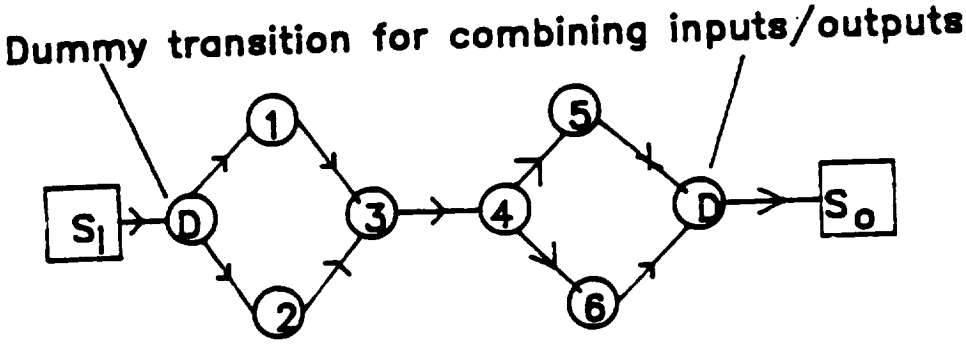
Application 6. This is an application of increasing throughput of linear time invariant systems by increasing the number of tokens in the circuit. Linear time invariant systems are described by the state equations as stated below.

$$x(k) = Ax(k-1) + Bu(k)$$
$$y(k) = Cx(k) + Du(k) \qquad (3.4.1)$$

where x is the state vector, y is the output vector, and u is the input vector. A, B, C, and D are time-invariant system matrices. The corresponding algorithm marked graph is shown in Figure 3.14. Usually, Ax(k-1) is the most time consuming computation in the AMG. In such a system, the recursion circuit determines the TBO_{LB}. It is shown that it is possible to reduce the time/token ratio of this recursion circuit by doubling the number of tokens so that TBO_{LB} is



(a)



(b)

Figure 3.12. (a) AMG A_1. (b) Transformed AMG A_2.

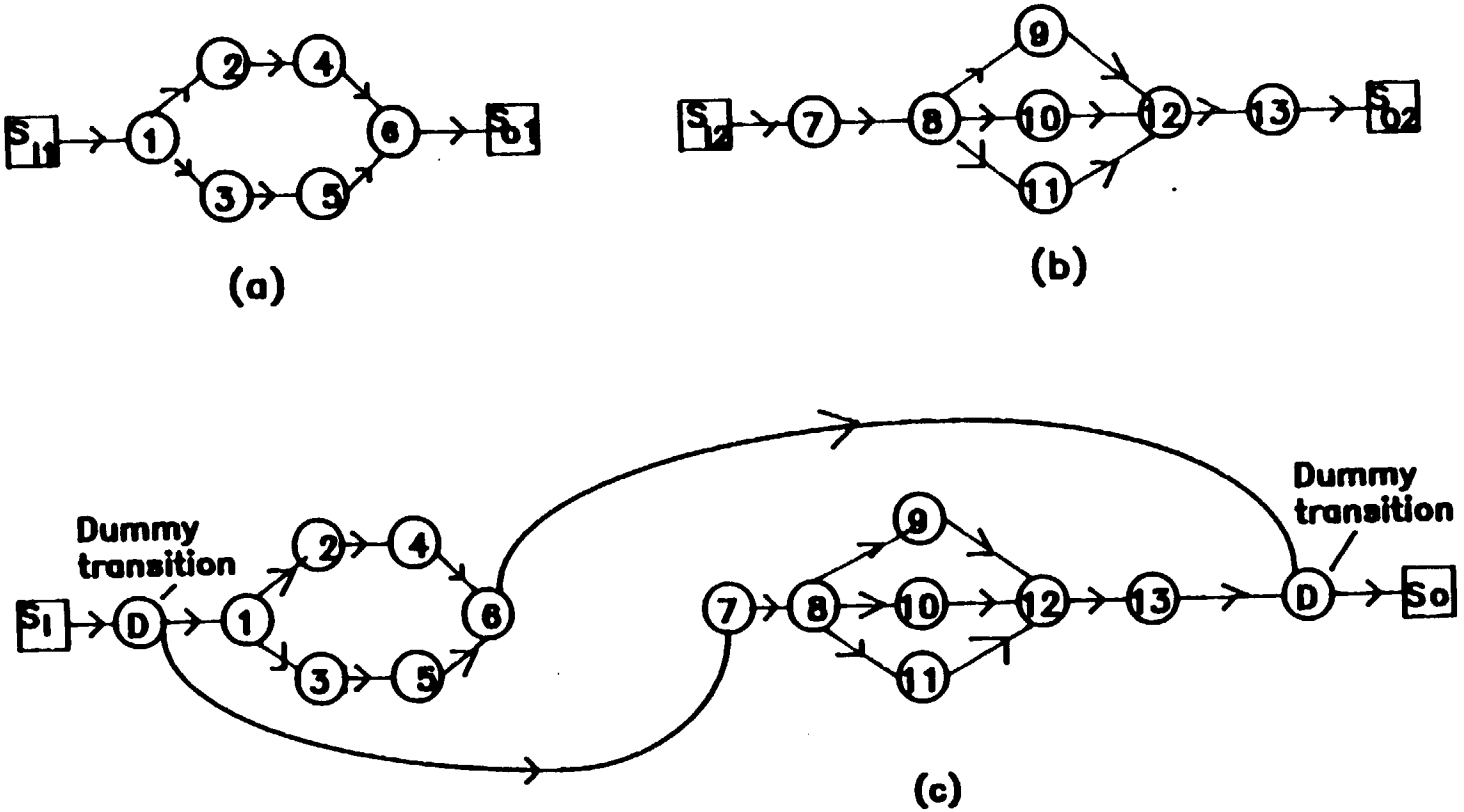


Figure 3.13. (a) Algorithm 1. (b) Algorithm 2. (c) Algorithms 1 and 2 are combined by dummy transitions.

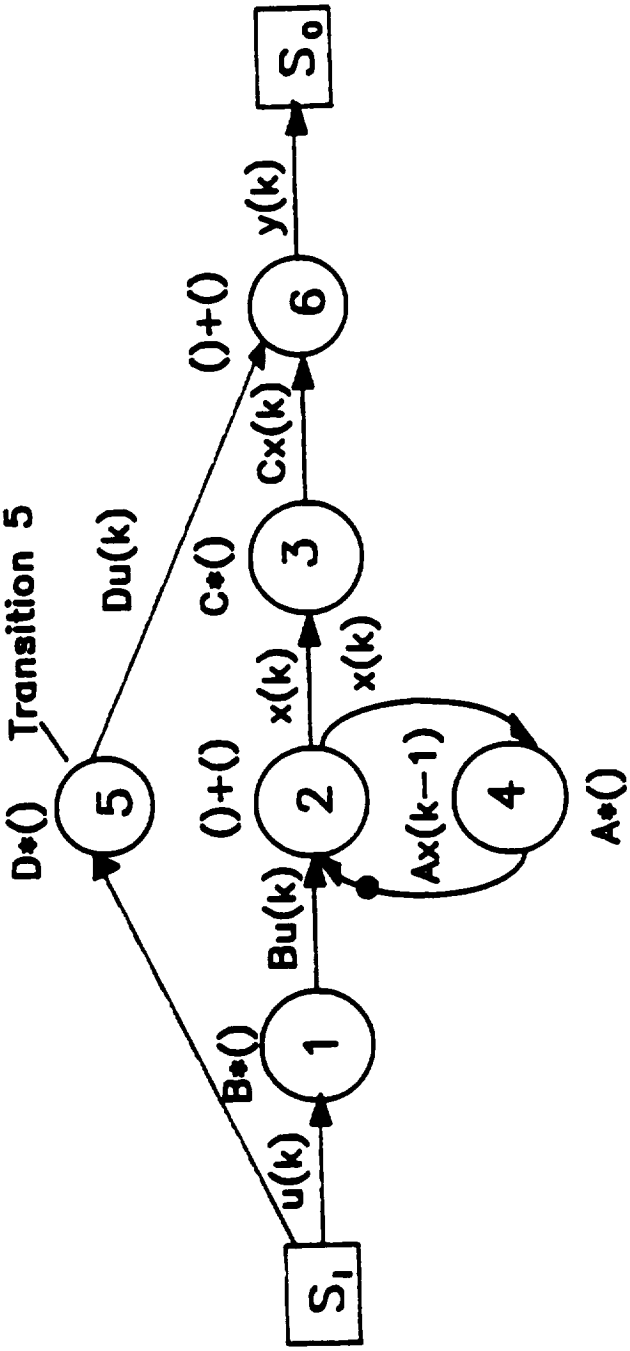


Figure 3.14. AMG for the linear time invariant system.

improved to the largest time/token ratio of the process circuits in the CMG. This is useful if decomposition is not desirable and TBO_{LB} needs to be reduced approximately to the largest transition time of the AMG. The methodology for reducing the time/token ratio of the recursion circuit is expressed below by the statement and proof of Theorem 3.2 with the assumption that Ax(k-1) is the largest transition in the AMG representing the state equation.

Theorem 3.2. It is possible to improve TBO_{LB} to the largest time/token ratio of the process circuits of a linear time invariant system by reducing the time/token ratio of the recursion circuit by doubling the number of tokens in the recursion circuit.

Proof. Theorem is proved by construction. Assuming Ax(k-1) (transition 4) to be the largest transition of Figure 3.14, TBO_{LB} is determined from the recursion circuit. Application 1 has shown that any AMG can be transformed so that TBO_{LB} is determined by only process circuits and recursion circuits. Thus, the statement of Theorem 3.2 will be proved if the AMG for the state equation can be transformed so that the time/token ratio of the recursion circuit is smaller than that of the largest process circuit. Let the state equation represent a 1-input, m-output, and n-element state vector system. The dimensions of A, B, C, and D are then (n, n), (n, 1), (m, n), and (m, 1) respectively. Now

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

$$x(k-1) = Ax(k-2) + Bu(k-1);$$

$$x(k) = A(Ax(k-2) + Bu(k-1)) + Bu(k).$$

It follows from the linearity of the system that

$$x(k) = (A * A)x(k-2) + (A * B)u(k-1) + Bu(k).$$

Let $A * A = E$ and $A * B = F$. Then,

$$x(k) = Ex(k-2) + Fu(k-1) + Bu(k). \qquad (3.4.2)$$

Notice that the dimension of E and A and F and B are the same. Therefore, the amount of computation of $Ax(k-1)$ and $Ex(k-2)$ and $Fu(k-1)$ and $Bu(k)$ are the same. However, if equation (3.4.2) is used instead of equation (3.4.1) for representing a linear time-invariant system, the recursion circuit has two initial tokens as $x(k)$ is generated from $x(k-2)$. The new AMG based on equation (3.4.2), and the original output equation, is shown in Figure 3.15. The dummy transitions are inserted to act as buffers so that transitions are not blocked from firing because output buffers are never empty. T_1, T_2, and T_3 are predefined tokens. $T_1 = F * u(k-1)$, $T_2 = E * x(k-2)$, and $T_3 = x(k-1)$. Let $k = 1, 2, 3...$ and the initial state vector be $x(0)$. Therefore, the first input and output are $u(1)$ and $y(1)$ respectively. That is, $u(s) = 0$ for s equal to zero or negative. Therefore, the initial values of T_1, T_2, and T_3 correspond to $k = 1$. Hence, the initial values of T_1 and T_3 are $T_1 = F * u(0) = 0$ and $T_3 = x(0)$. From (3.4.2),

$$T_2 = Ex(k-2) = x(k) - Fu(k-1) - Bu(k).$$

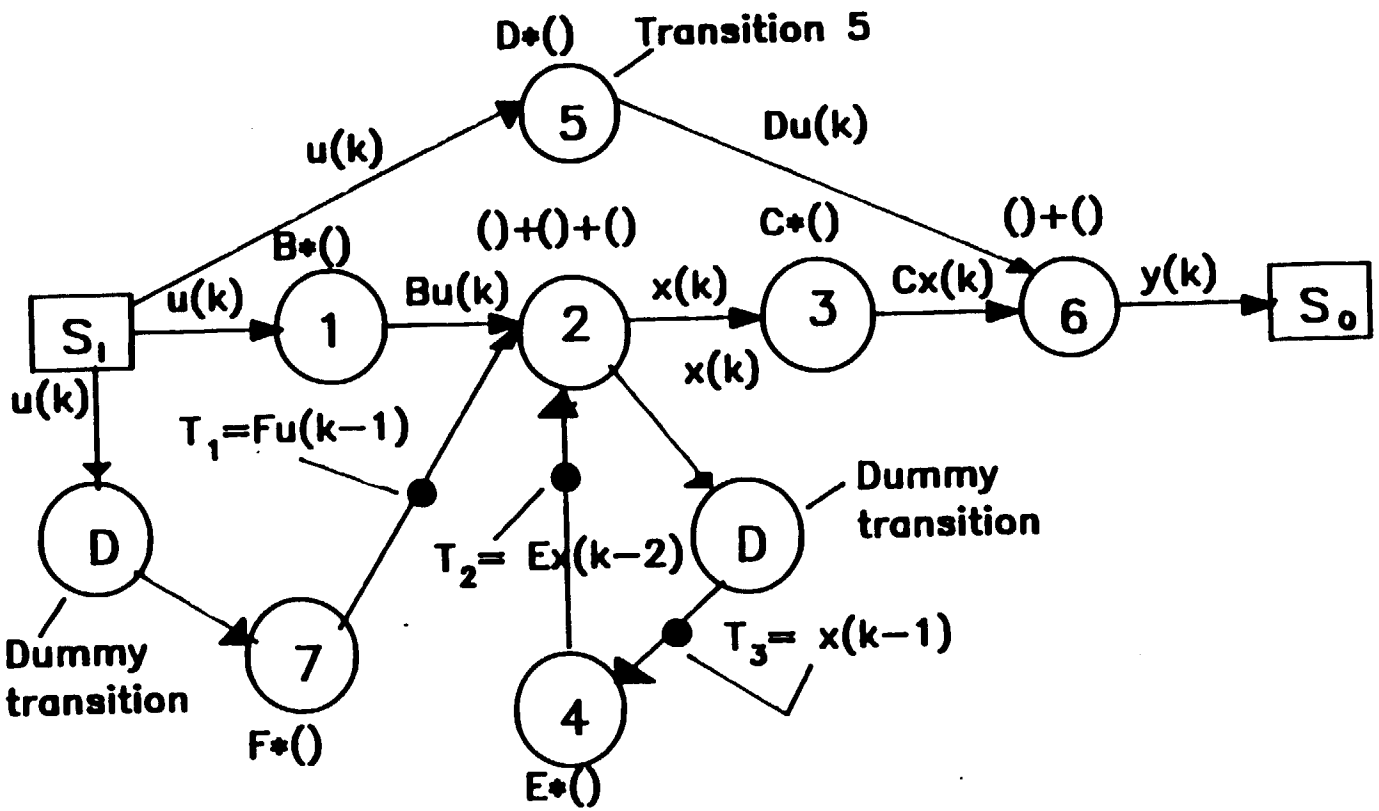


Figure 3.15. Transformed AMG for the linear time invariant system.

Therefore, the initial value of T_2 is given by $x(1) - Fu(0) - Bu(1)$. As $u(0) = 0$, the initial value of $T_2 = x(1) - Bu(1)$. Hence, it follows from the equation (3.4.1) that the initial value of $T_2 = Ax(0) + Bu(1) - Bu(1) = Ax(0)$. Therefore, all the initial values of the predefined tokens can be calculated from the initial state vector. The recursion circuit now consists of transitions 2 and 4 and there are two tokens in that circuit. The computation level of transition 4 has not changed, although that of transition 2 has doubled. Thus, the new time/token ratio of the recursion circuit is $T(4)/2 + T(2)$, where $T(4)$ and $T(2)$ are the times for transition 4 and 2 of the original algorithm marked graph. Assuming $T(4)$ is much greater than $T(2)$, the TBO_{LB} of the new algorithm marked graph of Figure 3.15 is given by the process circuit of transition 4 whose time/token ratio is the same as in Figure 3.14.

Application 7. This application establishes a method for finding the maximum level of parallel decomposition of a transition in an AMG for the best computing speed of the transition. Decomposition reduces process times of transitions; unfortunately, it also increases the communication cost due to an increase in number of transitions and places in the graph. Therefore, computing speed is improved with decompositions up to a certain level. For the lowest process time, transitions are decomposed uniformly. The maximum level of decomposition of the transition is determined from the condition for the fastest completion of the computation represented by the original transition.

Let T be the computation time of a transition which can be decomposed in parallel arbitrarily without changing T. Let this

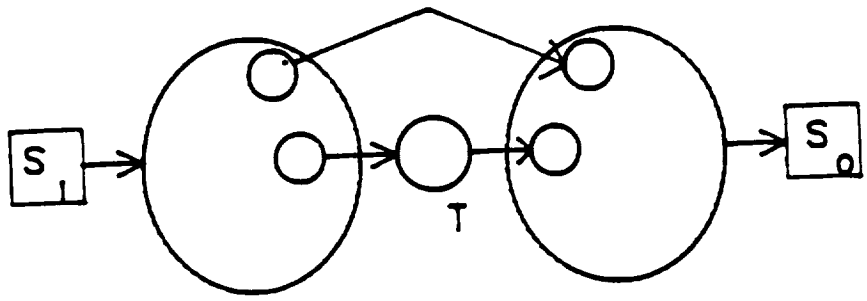
transition be decomposed into N equal parallel transitions as shown in Figure 3.16. Each T_i is T/N. The time to complete the total computation (A) for T in the worst case is then given by

$$A = R + T/N + C_0 + W. \qquad (3.4.3)$$

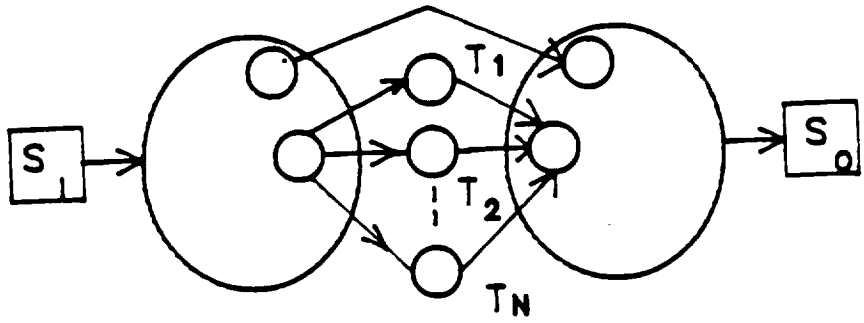
R and W are the read and write times to complete reading and writing of data for all T_i transitions. When this set of N transitions is computing T, some other transitions of the AMG may be concurrently processed. C_0 is the time required by each functional unit to receive data from the transitions of the rest of the AMG during the computing of T. C_0 is assumed to be independent of N and i. Any data are assumed to be broadcast to all functional units by a transmission medium. It is assumed that one data packet can be broadcasted at a time to all functional units. It is also assumed that total transmission time for output data for all N transitions together does not change with N. The worst case value of read and write time for all N transitions together can then be expressed by the following equation:

$$R + W = C_1 + N*L*C_2 + C_3, \qquad (3.4.4)$$

where C_1 is the time that the transmission medium has to be used to serve the rest of the AMG during the read and write operations for N transitions of T. C_1 is assumed to be independent of N. C_2 is the average access time for the transmission medium and L is the number of times a functional unit has to access the transmission



(a)



(b)

Figure 3.16. (a) An AMG with a large transition T. (b) T is decomposed in N parallel transitions.

medium for computing a transition. C_3 is the time to transmit output data over the transmission medium for all N transitions together and is assumed to be independent of N. Therefore, from 3.4.3 and 3.4.4,

$$A = T/N + C_0 + C_1 + N*L*C_2 + C_3.$$

For minimizing A, $dA/dN = 0$; d^2A/dN^2 = positive. Now

$$dA/dN = (- T/N^2) + (L*C_2);$$
$$d^2A/dN^2 = 2 * (T/N^3).$$

As T and N are always positive, d^2A/dN^2 is positive. Equating $dA/dN = 0$,

$$0 = (- T/N^2) + (L*C_2);$$
$$N = [\{T / (L*C_2)\}^{.5}]$$

As N has to be an integer and higher N will mean higher communication cost,

$$N = \lfloor [\{T / (L*C_2)\}^{.5}] \rfloor. \qquad (3.4.5)$$

Also as $N \geq 2$ for any decomposition,

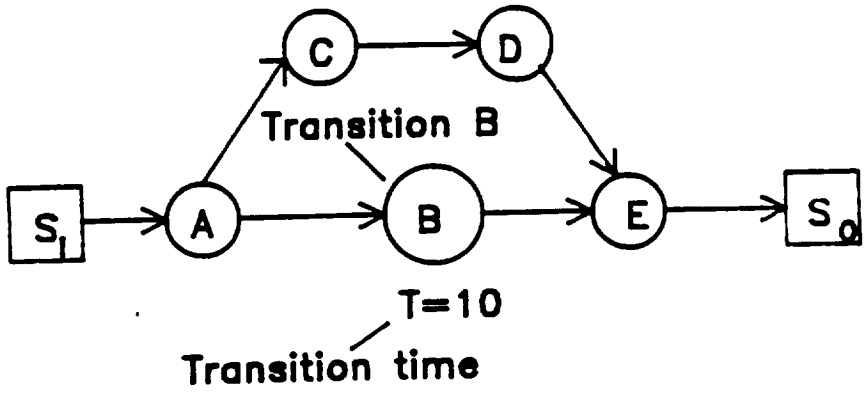
$$T \geq 4 * L * C_2. \qquad (3.4.6)$$

Thus knowing C_2, which is an architecture dependent parameter, the minimum value of T for decomposition can be evaluated from (3.4.6). Equation (3.4.5) provides the maximum level of decomposition.

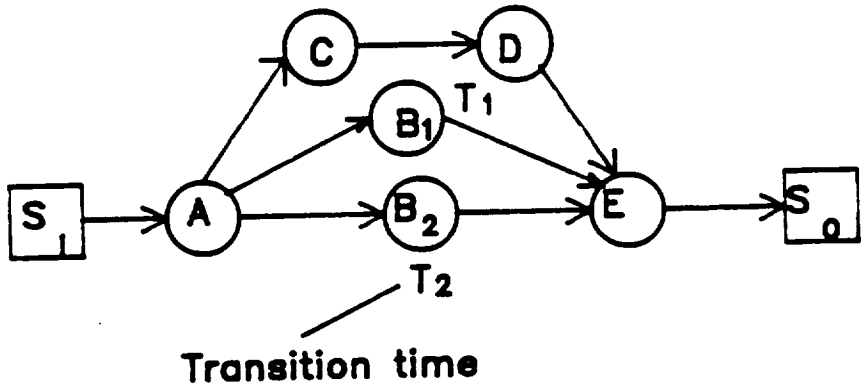
Example. Let T be the processing time for transition B in an AMG as shown in Figure 3.17. Suppose B can be arbitrarily decomposed in parallel. Let T = 10, C_2 = 0.25 and L = 2. As T $\geq$ (4*2*.25 = 2), B can be decomposed to improve performance. Let B be decomposed in N transitions in parallel. Hence, $N \geq \lfloor [\{10/(2*.25)\}^{.5}] \rfloor = 4$. In order to maintain process time for computation T reasonably higher than communication time for large granularity, a level of decomposition, less than or equal to, half the maximum level is assumed to be appropriate in the following example. Thus N is chosen to be 2. The decomposed transition B is shown in Figure 3.17.

3.5 Summary

Applications of algorithm transformation are discussed in this chapter and transformation techniques are defined. Improvements of TBO_{LB} are achieved by dummy transitions. Resource requirements may be lowered by control places and dummy transitions. Input data injection is controlled by predefined token and dummy transition. Periodicity in the resource envelope is enforced by dummy transitions. The methodology for transforming algorithms into single input-output algorithm is described. The TBO_{LB} of linear time-invariant systems is improved by predefined tokens. Lastly, parallel decomposition of transitions are considered to illustrate the trade-off between decreased granularity and increased communication cost.



(a)



(b)

Figure 3.17. (a) AMG before decomposition of B.
(b) B is decomposed.

CHAPTER FOUR

ATAMM OPERATING POINT DESIGN

4.0 Introduction

The ATAMM operating point (AOP) describes the specification of the input data injection interval (latency), resource requirements and the time performance of an algorithm marked graph operated on an ATAMM data flow architecture. The design of operating points based on the number of resources of the ATAMM data flow architecture is investigated in this chapter. The methodology is demonstrated through examples, simulations, and experiments. Properties of the ATAMM operating point under the allowable transformations and implementation strategies are discussed in Section 4.1. In Section 4.2, AOP design methodology is developed. Performance model, transformation techniques and the AOP design methodology are verified by simulations and experiments on test algorithms in Section 4.3. A summary of the chapter is presented in Section 4.4.

4.1 Characteristics of Operating Point

The ATAMM operating point is the parameter set (TBI, R, TBIO, TT, and TBO) for an algorithm execution where TBI is the input data injection interval (latency) and R is the minimum number of resources required by the ATAMM data flow architecture. The design problem is to specify an operating point for executing an AMG in the ATAMM data flow architecture which achieves optimum time performance with a

minimum number of computing resources. Unfortunately, this problem is equivalent to a class of scheduling problems which is known to be NP complete [12]. Thus, there exists no methodology for obtaining an optimum solution which is better than enumerating all possible solutions and then choosing the best one. However, it is possible to develop a procedure for generating sub-optimal solutions. This is the objective of this chapter. The design objective is to determine an operating point given the number of resources, and to provide the guidelines for generating a new operating point should the number of resources change. Also, the expected time performance for TBIO and TT should remain the same with any input data injection interval greater than that of the operating point as long as the number of resources are not decreased. The following properties are assumed in the operating point design:

a) Input data from the source are injected into the ATAMM data flow architecture at a constant rate, and hence the time between successive inputs (TBI) is always the same.

b) For all input data of the task, TBIO = $TBIO_{LB}$ and TT = TT_{LB}.

c) Each data set requires a resource usage envelope identical to REST.

All of these properties are realized by the use of Applications 3 and 4 of Section 3.3. These properties are needed for achieving the best task computing speed for all task inputs and to accurately predict resource requirements. As stated in Application 3, the time between successive data inputs (TBI) is adjusted to be greater than, or equal to, TBO_{LB} so that input data never wait on the critical

path to the data output sink. The algorithm marked graph is transformed as in Application 4 so that the resource envelope for each task input is REST. The design procedure must determine the allowable range of TBI so that the ATAMM data flow architecture has sufficient resources to meet the resource requirements of all task inputs. Let R_{min} be the peak value of REST. Therefore, any task input requires at least R_{min} resources to meet properties b and c. Let R_{max} be the largest peak value of TRE for any TBI $\geq TBO_{LB}$. Hence, with R_{max} or more functional units, any ATAMM data flow architecture can execute the AMG while achieving TT_{LB} and $TBIO_{LB}$ for any injection interval greater than, or equal to, TBO_{LB}. It is to be noted that TBI and TBO are the same for any AMG at steady state. Finally, let the number of resources of the ATAMM data flow architecture be denoted by R.

The operating point for various numbers of resources can be displayed on a graph of TBO versus TT. Every point in the graph is associated with a value of TBIO and R. From Chapter Two, TT $\geq$ TCE/R and TBO $\geq$ TCE/R. Also TBI and, hence, TBO need not be increased beyond TT as $R_{max} = R_{min}$ on the TBO = TT line. Therefore, the AOP is expected to lie in a triangular area of the graph determined by the number of functional units of the ATAMM data flow architecture. The characteristics of the operating point are shown in Figure 4.1.

Let the problem be specified by an algorithm marked graph. Let the best possible performance under the rules of operating point design be defined as the absolute lower bounds for the time performance. Formal definitions of the absolute lower bounds for TT, TBIO, and TBO are now stated.

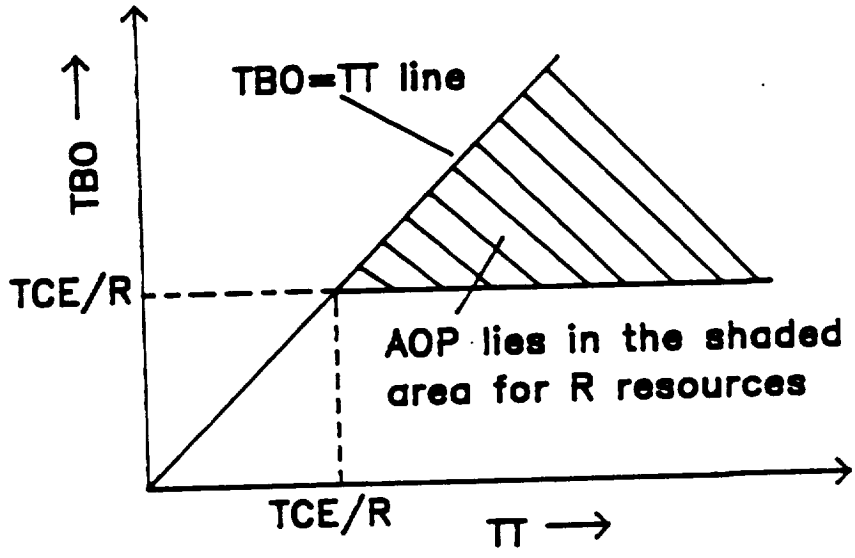


Figure 4.1. ATAMM operating point characteristics.

Definition 4.1: Absolute Lower Bound for TBIO. The absolute lower bound for TBIO ($TBIO_{ALB}$) is defined to be the lowest $TBIO_{LB}$ for the algorithm marked graph with or without any transformations.

Definition 4.2: Absolute Lower Bound for TT. The absolute lower bound for TT (TT_{ALB}) is defined to be the lowest TT_{LB} for the algorithm marked graph with or without any transformations.

Definition 4.3: Absolute Lower Bound for TBO. The absolute lower bound for TBO (TBO_{ALB}) is defined to be the lowest TBO_{LB} with or without any transformations.

Let the transformation be restricted such that only dummy transitions (of zero time) and control places (with no initial token) are used for transforming the algorithm marked graph. Theorems are now described to determine the absolute lower bounds under the above transformations.

Theorem 4.1. The absolute lower bound for TBIO is equal to the lower bound without any transformations.

Proof. Control places can create new paths in an algorithm marked graph but do not alter existing paths in the AMG. Dummy transitions of zero time increase the number of transitions on a path in the AMG but do not increase the path length. Therefore, any path in the original AMG is also a path in the transformed AMG with equal path length. The critical path from the data input source to the data output sink in the MAMG of the original algorithm marked graph is also a path from the data input source to the data output sink in the MAMG of the transformed AMG. Hence, $TBIO_{LB}$ of any transformed AMG under the stated transformations cannot be lower than that of the original one. Therefore, the $TBIO_{ALB}$ of an algorithm marked graph is

determined by the $TBIO_{LB}$ of the AMG without any transformations. This completes the proof.

Theorem 4.2. The absolute lower bound for TT is equal to the lower bound without any transformations.

Proof. The proof is similar to that of Theorem 4.1. However, TT_{LB} is determined by the critical path among all paths from the data input source to any output sink in the MAMG. By the arguments of Theorem 4.1, this critical path in the MAMG of the original AMG is also present with equal path length in the MAMG of the transformed AMG. Thus, TT_{LB} cannot be reduced by transformation with dummy transitions (zero time) and control places (no initial token). Hence the TT_{ALB} of an AMG is determined by the TT_{LB} of the AMG without any transformations. This completes the proof.

Theorem 4.3. The absolute lower bound for TBO is equal to the largest time/token ratio among the process and recursion circuits in the CMG of the original algorithm marked graph without any transformations.

Proof. Theorem 3.1 has proved that the TBO_{LB} of an algorithm marked graph can be reduced to the largest time/token ratio of the process and recursion circuits by transforming with dummy transitions of zero time. Because of the way process and recursion circuits are created, dummy transitions do not alter their time/token ratio. Control places, on the other hand, can create new parallel path circuits in the CMG but do not change the time/token ratio value of the circuits in the CMG of the original AMG. Therefore, the lowest TBO_{LB} and TBO_{ALB} is determined by the largest time/token ratio among the process and recursion circuits in the CMG of the original AMG. This completes the proof.

Any operating point will have TBIO, TT, and TBO values greater than, or equal to, those specified by the respective absolute lower bounds. Figure 4.2(a) displays the characteristics of the operating point when designed with only dummy transitions (zero time) and control places (no initial token). Any operating point resides in the area BVWH. The point B corresponds to the operating point which achieves the absolute lower bounds for TBIO, TT, and TBO. Lines BV and BH represent operating points which achieve the absolute lower bounds in task computing speeds (TT and TBIO) and the output interval (TBO) respectively. With the specified transformations, TT_{LB} cannot be more than TC. Any operating point on line HW has TT_{LB} = TC, which indicates the absence of any parallel concurrency. Point W is characterized by $TT_{LB} = TBO_{LB}$ = TC and represents complete sequential operation with no concurrency. ATAMM is most appropriate for problems which require both vertical and horizontal concurrency. It is assumed that $TBIO_{LB}$ and TT_{LB} are achieved for any TBI greater than, or equal to, the data injection interval at the operating point. Therefore, the minimum resource requirement at any operating point is the greatest peak value of TRE for any TBI $\geq$ TBO_{op}, where TBO_{op} is the data output interval and the input data injection interval at the operating point.

4.2 Operating Point Design

Let the problem be specified by an algorithm marked graph for which the ATAMM operating point is to be determined. The only allowable algorithm transformations are dummy transitions of zero time and control places. Predefined tokens and decomposition will not be

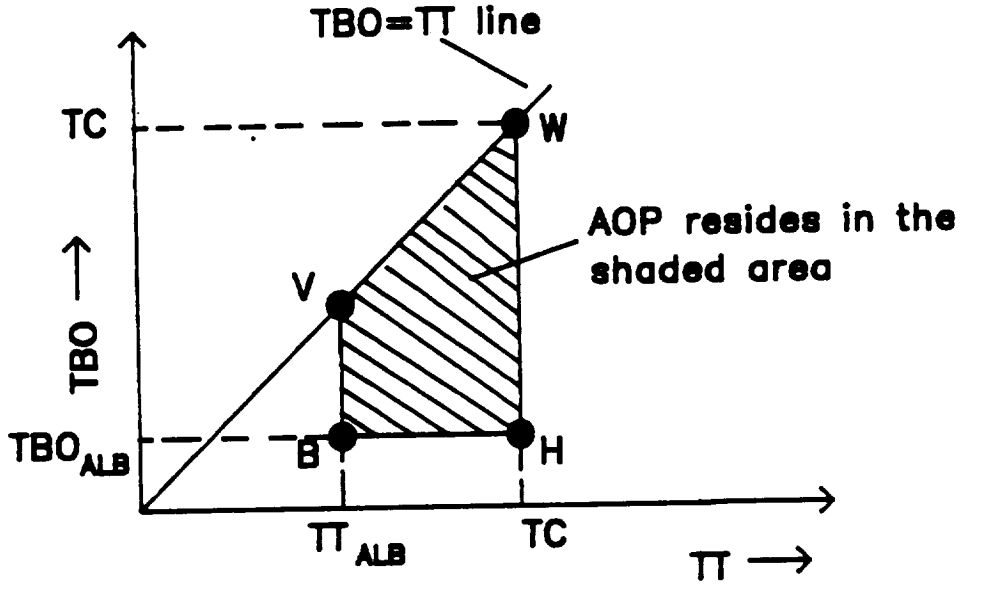


Figure 4.2(a). AOP characteristics under specific transformations.

considered for operating point design. The AOP design consists of six steps. These steps are described in the remainder of this section. The operating points are determined corresponding to different number of resources for the algorithm marked graph of Figure 3.3 to illustrate each step as it is presented.

Step 1. Construct the CMG from the AMG. Determine lower bounds and absolute lower bounds for TBIO, TT, and TBO for the AMG. If TBO_{LB} is greater than TBO_{ALB}, transform the AMG with dummy transitions to achieve TBO_{ALB}, as in Application 1 of Section 3.2. Determine R_{max} and R_{min}. If $R_{max} > \lceil TCE/TBO_{ALB} \rceil$, heuristically transform the AMG with control places and dummy transitions to reduce R_{max} without increasing $TBIO_{LB}$, TT_{LB}, and TBO_{LB}, as in Application 2 of Section 3.2. Determine new R_{max} and R_{min} values. Lower bounds of performance for the resultant AMG are also the absolute lower bounds for TT, TBIO, and TBO under the specified transformations.

From the AMG of Figure 3.3, $TBIO_{LB} = 6$, $TT_{LB} = 6$, $TBO_{LB} = 2$. Also $TBIO_{ALB} = 6$, $TT_{ALB} = 6$, and $TBO_{ALB} = 2$. REST and TRE corresponding to TBO = 2 are shown in Figure 3.4. Checking all TBI $\geq$ 2, $R_{max} = 9$. The AMG of Figure 3.3 is now transformed heuristically to lower R_{max} without increasing $TBIO_{LB}$, TT_{LB}, and TBO_{LB}, as described in Application 2 of Section 3.2. The transformed AMG is shown in Figure 3.5 (ignore control places 2, 3, and 4). REST and TRE corresponding to TBI = TBO_{LB} = 2 are shown in Figure 3.6 for the resultant AMG. By checking all TBI $\geq$ 2, it is determined that R_{max} = 8, $R_{min} = 4$.

Step 2. Choose a convenient transition firing rule. A rule to determine when an enabled transition in the CMG fires must be specified in the graph manager. The rule usually used is that enabled transitions fire when computing resources are available. If contention exists, such as when there are more enabled transitions than computing resources, firing occurs according to a priority ordering of the transitions. For the algorithm marked graph of Figure 3.5, the highest to lowest priority ordering of the transitions is chosen as (11, 10, 9, 7, 8, 5, 6, 4, 3, 2, 12, and 1).

Step 3. If $R \geq R_{max}$ functional units are available, operate at TBI = TBO_{ALB}. Use Application 3 and 4 of Section 3.3 to adjust TBI to TBO_{ALB} and to transform the AMG by dummy transitions in order to realize REST as the resource envelope for all task inputs. Eliminate all unnecessary dummy transitions. The operating point time performance is the absolute lower bound values for TBIO, TT, and TBO. The AMG can also be operated for any TBI > TBO_{ALB} while maintaining TBIO and TT at absolute lower bound values. When $R < R_{max}$, determine the operating point from one of the following strategies:

Strategy A: Strategy A is applicable when $R_{max} > R \geq R_{min}$. Preserve TBIO and TT at their respective absolute lower bounds at the expense of increasing TBI and TBO above TBO_{ALB}.

Strategy B: Strategy B is applicable for the following range of R. $R_{max} > R \geq \lceil TCE/TBO_{ALB} \rceil$. Preserve TBO to its absolute lower bound at the expense of increasing one, or both, of $TBIO_{LB}$ and TT_{LB}.

Strategy C: Strategy C is applicable when $R_{max} > R \geq 1$. The operating point is determined by first following Strategy B so that $R_{max} > R \geq R_{min}$, and then increasing TBI above TBO_{ALB}. The strategy tries to minimize performance degradation in TBIO, TT, and TBO from their respective absolute lower bound values.

These three strategies of the AOP design under resource constraints are illustrated in Figure 4.2(b). Strategy A maintains TT and TBIO at their respective absolute lower bound values and reduces pipeline concurrency to lower resource requirements. Strategy B reduces resource requirements by decreasing parallel concurrency resulting in a higher lower bounds for one or both of TBIO and TT. Strategy C sacrifices both pipeline and parallel concurrency to some extent for lowering resource requirements.

If the ATAMM data flow architecture has eight or more functional units, the algorithm marked graph of Figure 3.5 can be operated at TBIO = TT = 6 and TBO = 2 by adjusting TBI = 2 using Application 3 of Section 3.3. GPST and TGP corresponding to TBI = 2 are shown in Figure 4.3 which suggest that no new dummy transitions are required to enforce REST and GPST. Resource utilization over a period TBO is given by {TCE/(R*TBO)} = 12/16 = .75.

Step 4. Execute this step if strategy A is appropriate. Increase TBI to TBO_{op} such that TBO_{op} is the lowest time interval between overlapping REST's for the peak value of TRE to be less than, or equal to R, for all $TBI \geq TBO_{op}$. TBO_{op} is guaranteed to lie in the range $\lceil TCE/R \rceil \leq TBO_{op} \leq TT_{ALB}$. Operate at $TT = TT_{ALB}$,

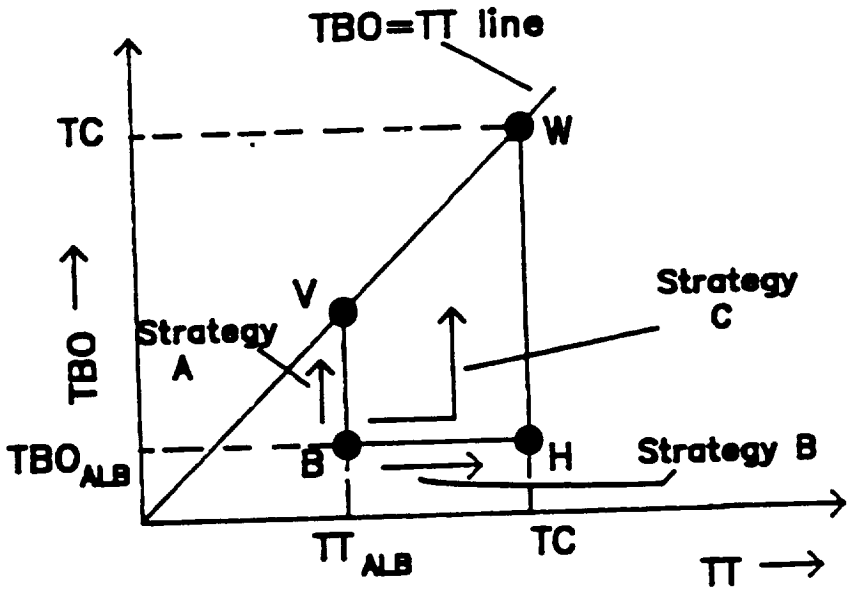


Figure 4.2(b). The strategies for AOP design under resource constraints.

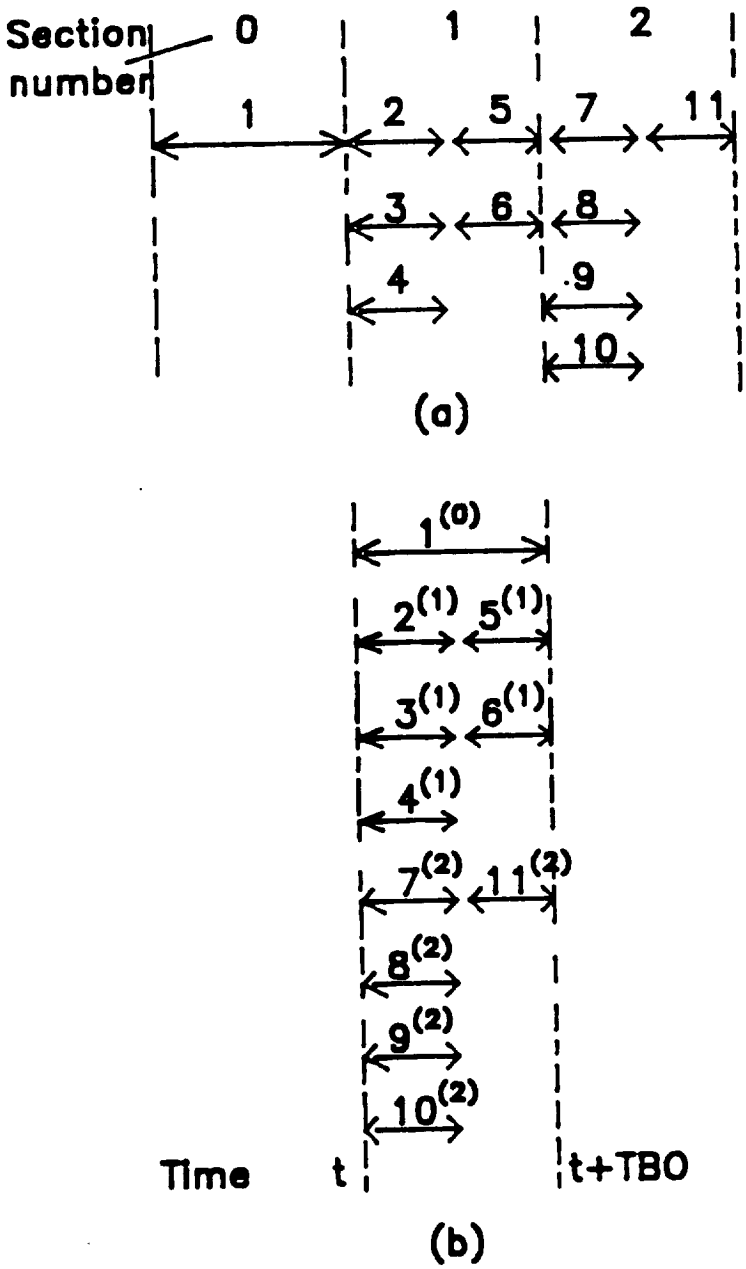


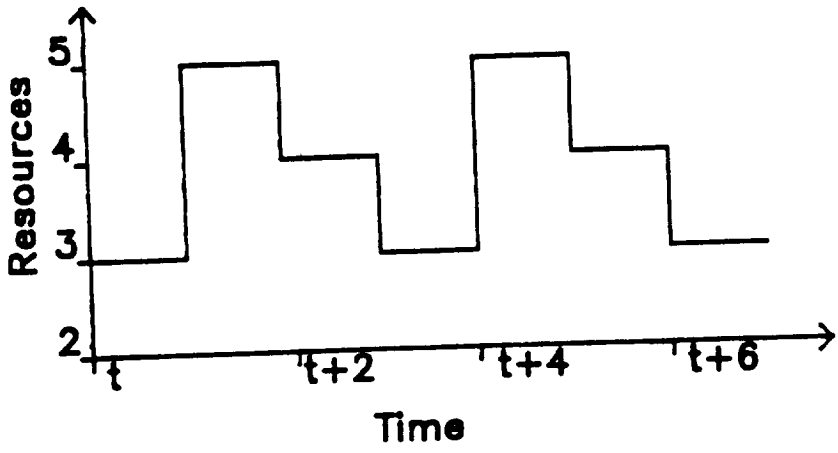
Figure 4.3. (a) GPST. (b) TGP for TBO=2.

$TBIO = TBIO_{ALB}$, and $TBO = TBI = TBO_{op}$ using Application 3 of Section 3.3. $TBIO_{ALB}$ and TT_{ALB} are also achieved for any $TBI > TBO_{op}$.

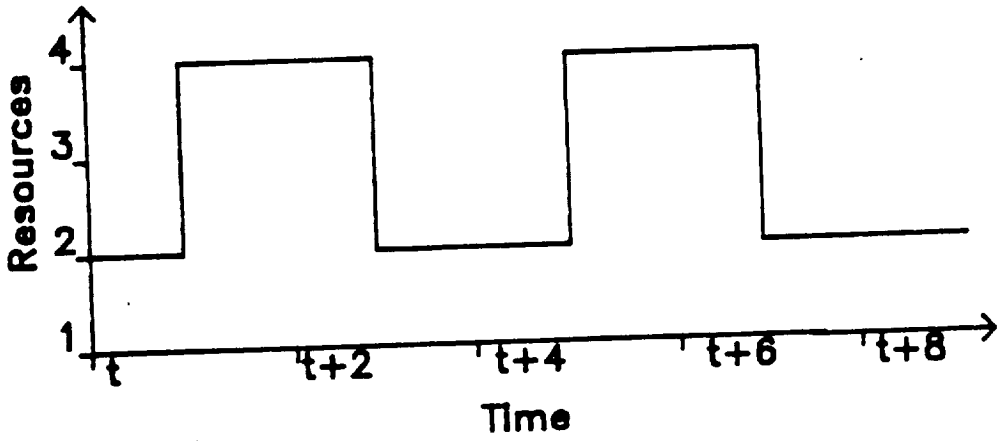
Assume, the ATAMM data flow architecture has five functional units. As $R_{min} = 4$, Strategy A can be applied. Following Strategy A, it is found that $TBO_{op} = 3$. Overlapping of REST's for TBI = 3 is shown in Figure 4.4(a). The operating point is given by TT = TBIO = 6 and TBI = TBO = 3 and RU(TBO) = (12/(5*3)) = .8.

Step 5. Execute this step if Strategy B is appropriate. Heuristically transform the AMG to reduce R_{max} using control places, as in Application 2 of Section 3.2. Maintain TBO_{LB} at TBO_{ALB} by using dummy transitions. A good heuristic is to reduce R_{min} significantly. There is a guaranteed solution at $TT_{LB} = TC$, $TBIO_{LB} = TFC$, and $TBO_{LB} = TBO_{ALB}$ by transforming the AMG into a complete chain. Eliminate all unnecessary dummy transitions. Operate the transformed AMG for $TBI = TBO_{ALB} = TBO$, $TT = TT_{LB}$, and $TBIO = TBIO_{LB}$ using Applications 3 and 4 of Section 3.3.

Suppose the ATAMM data flow architecture has six resources. TCE = 12 units of computer time. As $R \geq \lceil TCE/TBO_{ALB} \rceil = 6$, Strategy B can be applied. R_{max} is reduced to 6 by control places 2, 3, and 4 as shown in Figure 3.5. New REST and TRE for TBI = 2 are shown in Figure 3.7. The peak value of TRE is 6. $TT_{LB} = TBIO_{LB} = 7$. By checking all $TBI \geq 2$ for this AMG, it is found that $R_{max} = 6$ and $R_{min} = 3$. GPST and TGP for the transformed AMG are shown in Figure 4.5. Only transition 5 has a float associated with it. The successor of transition 5 is transition 11. By inspection of the TGP, transition $5^{(1)}$ fires before transition $11^{(2)}$, which is impossible



(a)



(b)

Figure 4.4. (a) TRE for TBO=3 in Step 4.
(b) TRE for TBO=4 in Step 6.

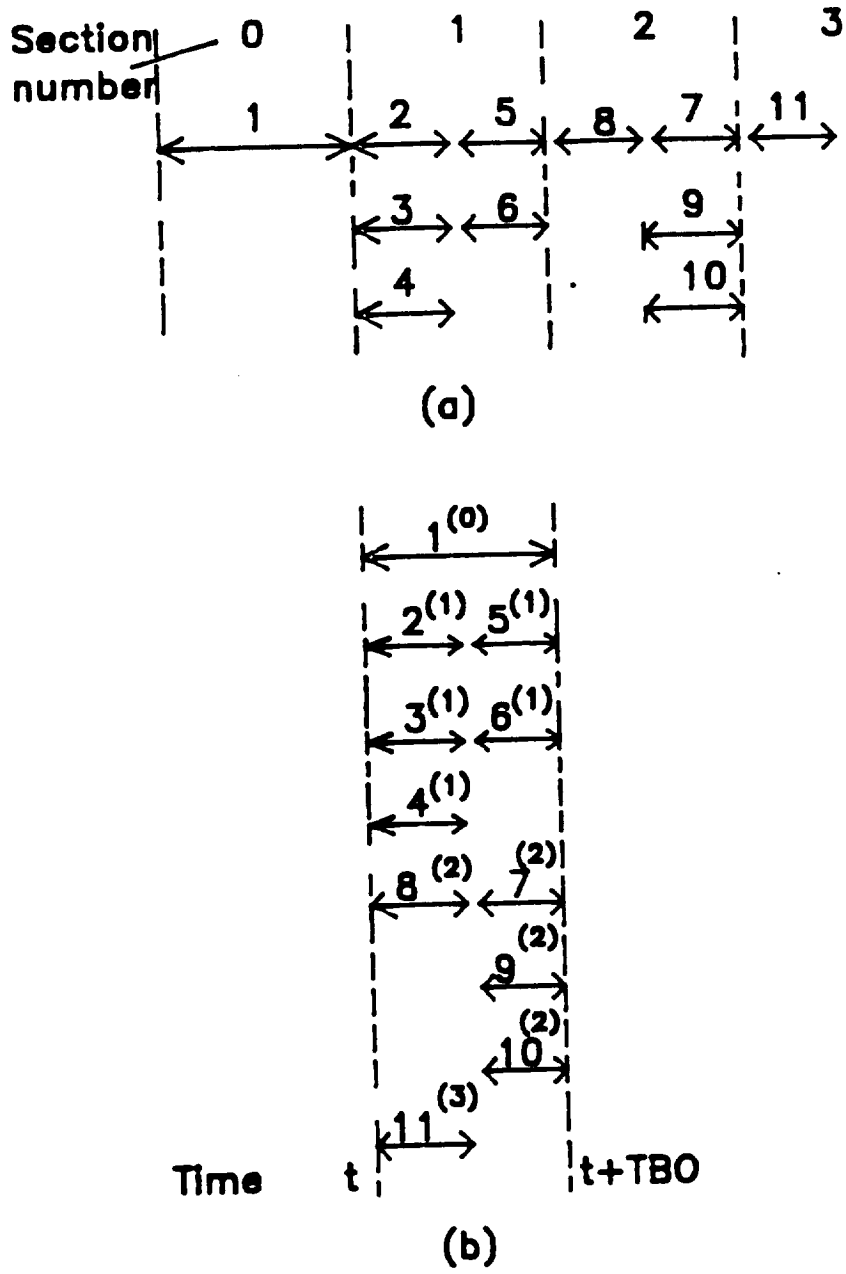


Figure 4.5. (a) GPST. (b) TGP for TBO=2.

in an ATAMM unless there is a buffer between transitions 5 and 11. Hence one dummy transition is required between transitions 5 and 11 as shown in Figure 4.6 to enforce REST as the resource envelope for all task inputs. The operating point is given by TT = TBIO = 7 and TBO = TBI = 2; RU(TBO) = 1.

Step 6. Execute this step if Strategy C is appropriate. Transform the AMG by Strategy B until $R_{max} > R \geq R_{min}$ and then increase TBI to determine TBO_{op}, as in Strategy A.

Let R = 4. The AMG is transformed by Strategy B as described in Step 6. Now $R_{max} = 6$ and $R_{min} = 3$. As R is within the range of R_{max} and R_{min}, the operating point can be determined by increasing TBI as in Strategy A. Increasing TBI, $TBO_{op} = 4$. Overlapping of REST's and TRE for TBI = 4 are shown in Figure 4.4(b). The operating point is given by TT = TBIO = 7 and TBI = 4. Adjust TBI to 4 for the AMG of Figure 4.6 to implement the operating point. RU(TBO) = .75.

These operating points for the AMG of Figure 3.5 are shown in Figure 4.7. Operating point B is the only operating point which achieves the absolute lower bounds for TT, TBIO, and TBO and is achieved in Step 3. OP_A, OP_B, and OP_C are the operating points developed by Strategies A, B, and C respectively.

4.3 Test Results

The performance model, transformation techniques, and the ATAMM operating point design procedures are tested by simulations and experiments. Simulations on the test algorithms are done by a software simulator developed to simulate the execution of an algorithm in the ATAMM environment [21]. The input parameters for the simulator

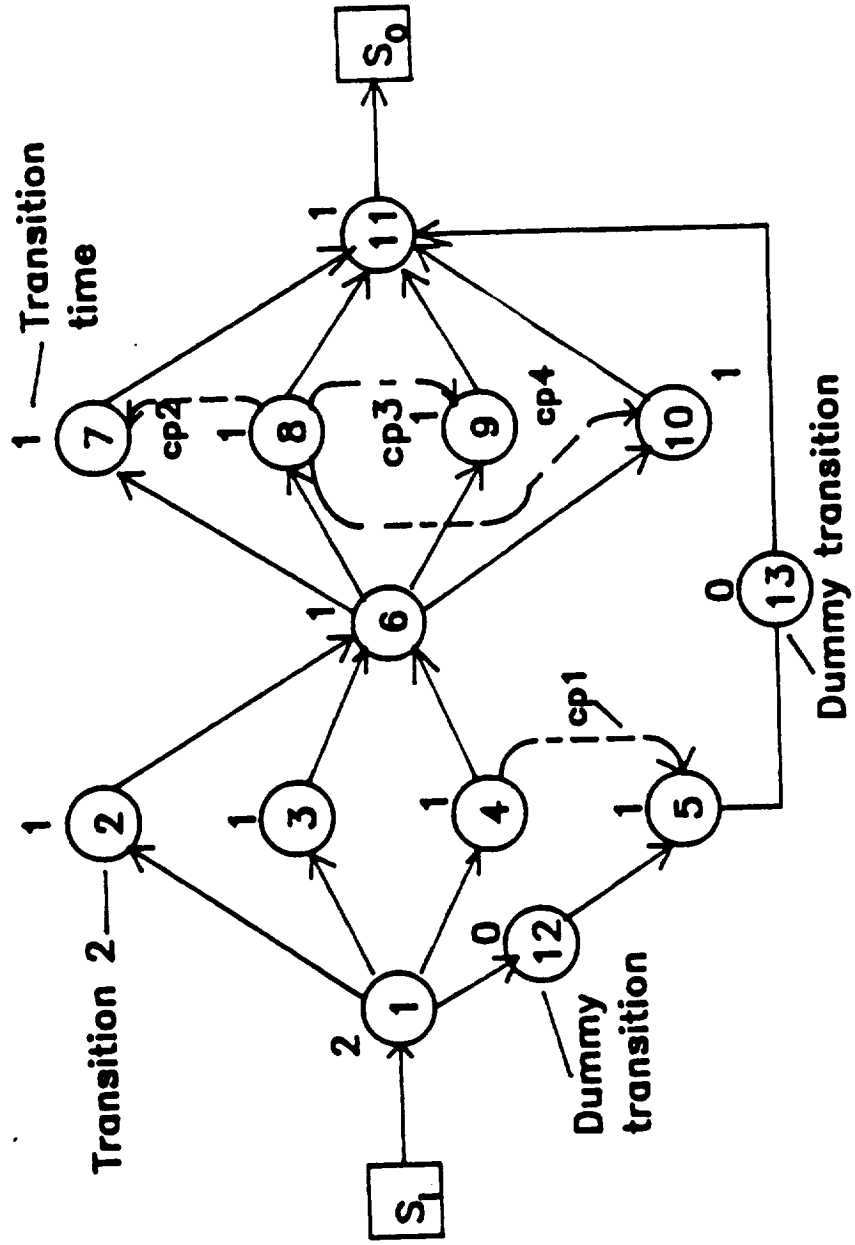


Figure 4.6. Transformed AMG for Steps 5 and 6.

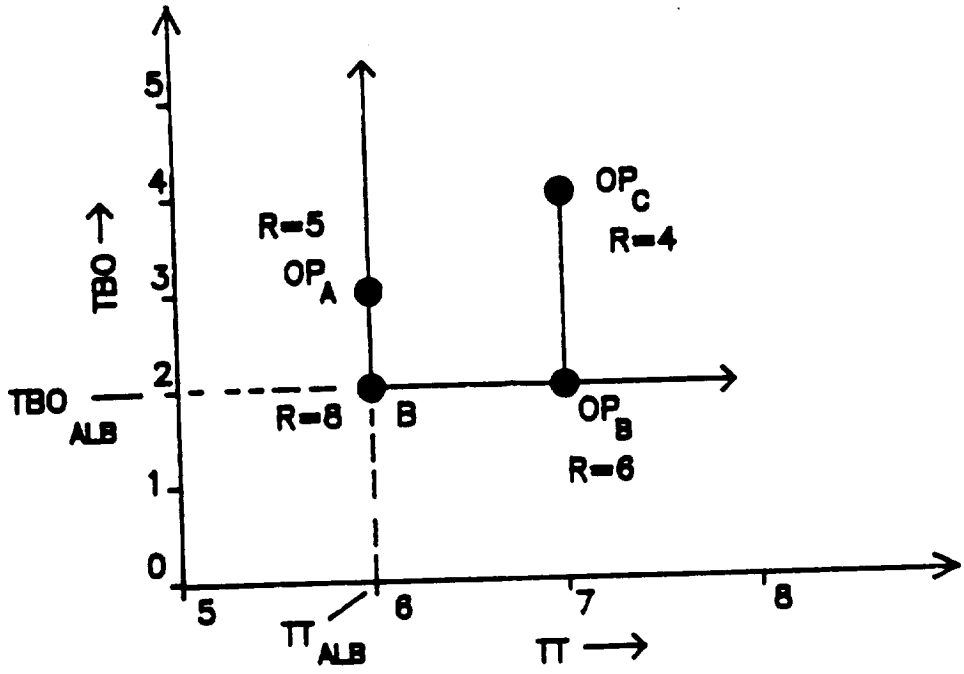


Figure 4.7. ATAMM operating points for the example algorithm marked graph.

are the algorithm marked graph including all NMG transition times, the number of resources, and a priority ordering for the transitions of the AMG. The input data injection interval is controlled by adjusting the source transition time. The simulator detects and writes all events associated with the execution of transitions for each task input on a graph diagnostic file. The analyzer is a program developed to analyze this graph diagnostic file [21]. The two features of the analyzer used in this dissertation are the node activity display and the input/output display. The node activity display shows the execution of transitions as a function of time. The input/output display shows TBI, TBO, and TBIO for each task input and also plots these quantities as a function of time. Detailed information about the simulator and the analyzer are found in [21]. Another useful program developed is called Ttime which determines the lower bounds for TT, TBIO, and TBO in an algorithm marked graph by constructing the CMG and MAMG [20].

A testbed is developed to run test algorithms in the ATAMM environment [20]. The ATAMM data flow architecture consists of, at most, three functional units with a distributed global memory and graph manager. Figure 4.8 shows the architecture. Functional units are realized by IBM Personal Computer AT's. Functional units communicate between each other by a Ethernet communication bus. In addition, another IBM PC AT which implements the source and sink transitions of the AMG is connected on the Ethernet bus. This IBM PC AT is used to begin and end the execution of the test algorithm and to generate a graph diagnostic file recording all events during the execution of the AMG. At the present stage, the source transition time cannot be adjusted to control the injection rate and this rate is

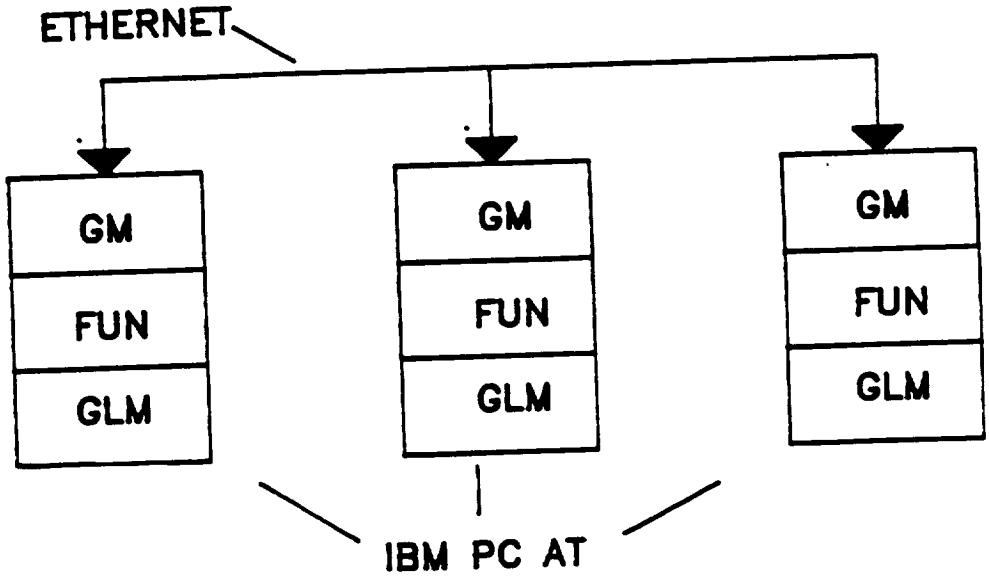
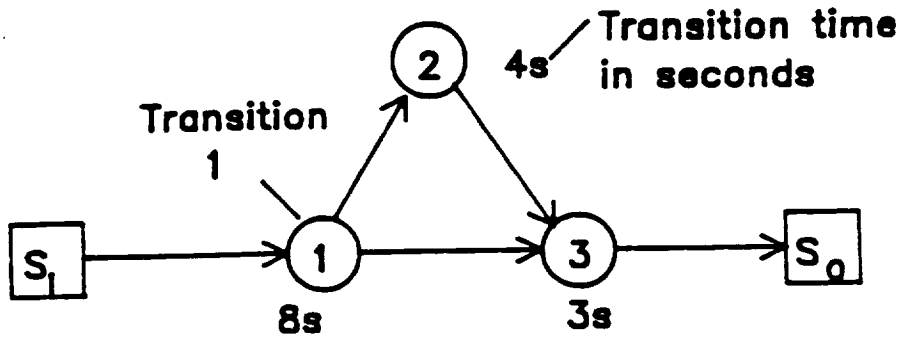


Figure 4.8. The testbed ATAMM data flow architecture.

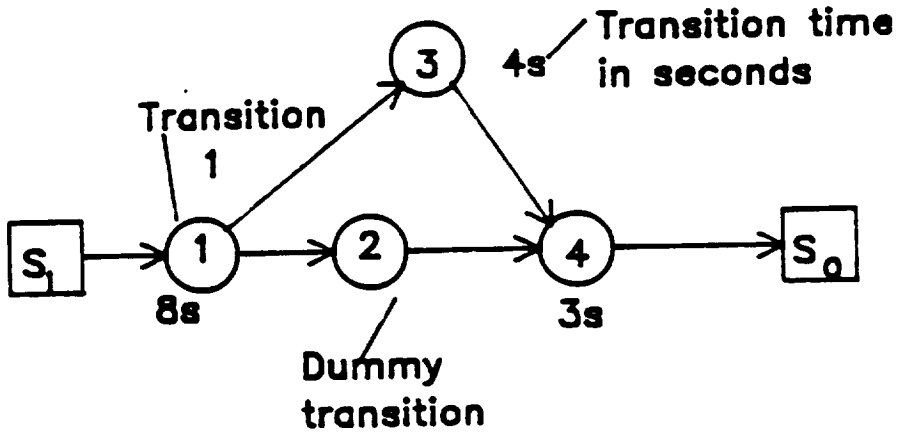
always equal to a small write time. Thus, it is not possible to check the entire ATAMM operating point design procedure on the testbed. However, two experiments are carried out to show the effect of dummy transitions in improving TBO_{LB} and the use of control places to reduce resource requirements. The analyzer is used to determine the performance of the test algorithm from the graph diagnostic file. Detailed information about the testbed can be found in [20].

Five test algorithms are chosen to test the design procedure, performance model, and transformation techniques on algorithms with a wide range of structural characteristics. Execution of all five algorithms were simulated but only two algorithms were actually implemented on the testbed, mainly due to the resource limitations and inability to control the input data injection interval. The results are stated and analyzed for each of the test algorithm execution in the following discussion.

Test 1. The primary objective of this test is to show the use of a dummy transition as buffer in reducing the time/token ratio of a parallel path circuit. Experimental time performance is also compared with the theoretical time performance predicted by the performance model. The test AMG and a transformed test AMG are shown in Figure 4.9(a) and (b) respectively. The purpose of the dummy transition is to reduce the time/token ratio of the parallel path circuit for the parallel paths between transition 1 and 3 in Figure 4.9(a) so that TBO_{LB} is improved to the time/token ratio of the largest process circuit. All the transition times are expressed in seconds. Priority ordering from highest to lowest in the test AMG and transformed test AMG are (3, 2, 1) and (4, 3, 2, 1) respectively. The dummy



(a)



(b)

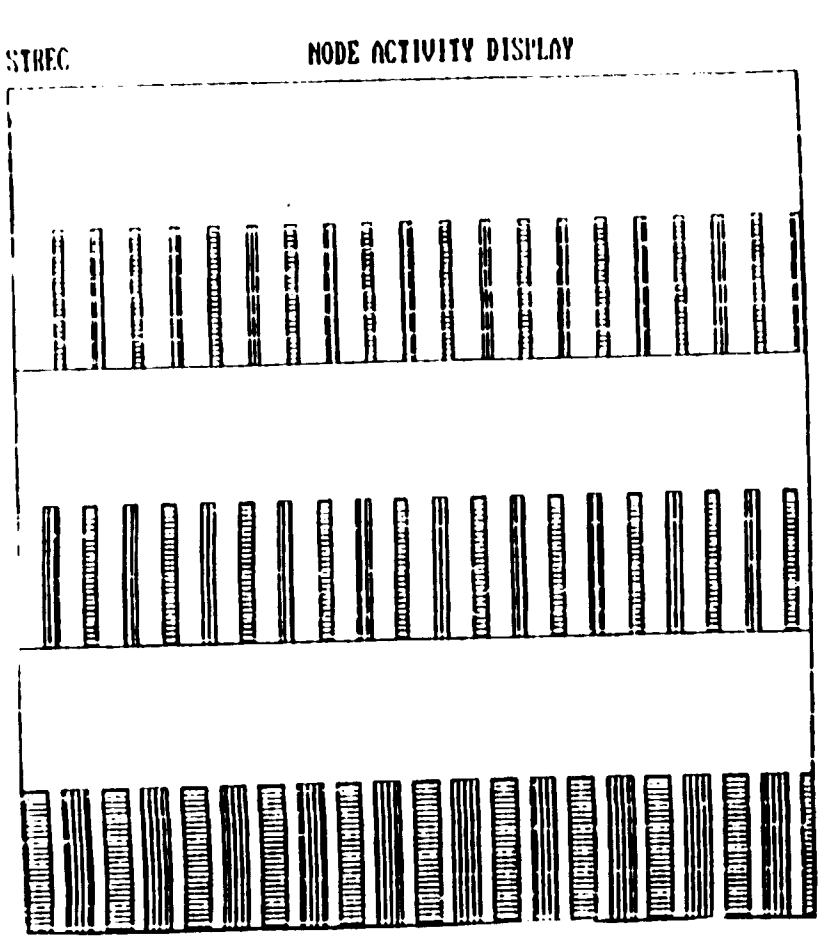
Figure 4.9. (a) AMG for Test 1. (b) Transformed AMG for Test 1.

transition is implemented as an active transition of zero process time. Read and write times of the transitions are assumed to be 220 ms and 255 ms for simulation and theoretical performance evaluation (these communication times were measured for the testbed in [20] for two functional units). Lower bounds for TBIO and TBO are calculated for both the test AMG and the transformed test AMG. It is assumed in simulations and experiments that no resource is needed to implement a dummy transition. Both the AMG's are executed and simulated for two functional units which are the maximum resource requirements to achieve TBO_{LB} and $TBIO_{LB}$ in either case. Although experimental and simulated time performance are expected to be $TBIO_{LB}$ and TBO_{LB}, there can be some differences due to the following reasons. The simulated performance measures are always a little higher than the theoretical expected performance. This is due to lost clock cycles in assigning transitions to resources and the fact that even a dummy transition will also require a resource, though only for a small duration. Experimental time performance values are higher in some cases from the theoretical expected time performance due to one or more of the following reasons. First, Ethernet cannot implement more than one read or write operation at the same time. Second, as the dummy transition is nonideal, it requires a resource. Third, read and write times for NMG transitions were measured with no contention, which is not true when a number of transitions try to communicate at the same time. Fourth, there is a slight increase in actual process times for transitions due to interrupt from other functional units. Experimental and simulation results for both AMG's are presented in Figures 4.10 through 4.13 and compared with theoretical performance lower bounds in Table 4.1. The node activity display shows the

TABLE 4.1

COMPARISON OF RESULTS FOR TEST 1

Algorithms	Experimental Results (s)		Simulation Results (s)		Theoretical LB's (s)	
	Av. TBO	Av. TBIO	Av. TBO	Av. TBIO	TBO_{LB}	$TBIO_{LB}$
AMG for Test 1	13.13	16.41	13.28	16.53	13.17	16.425
Transformed AMG for Test 1	9.23	16.43	9.1	16.53	8.695	16.425



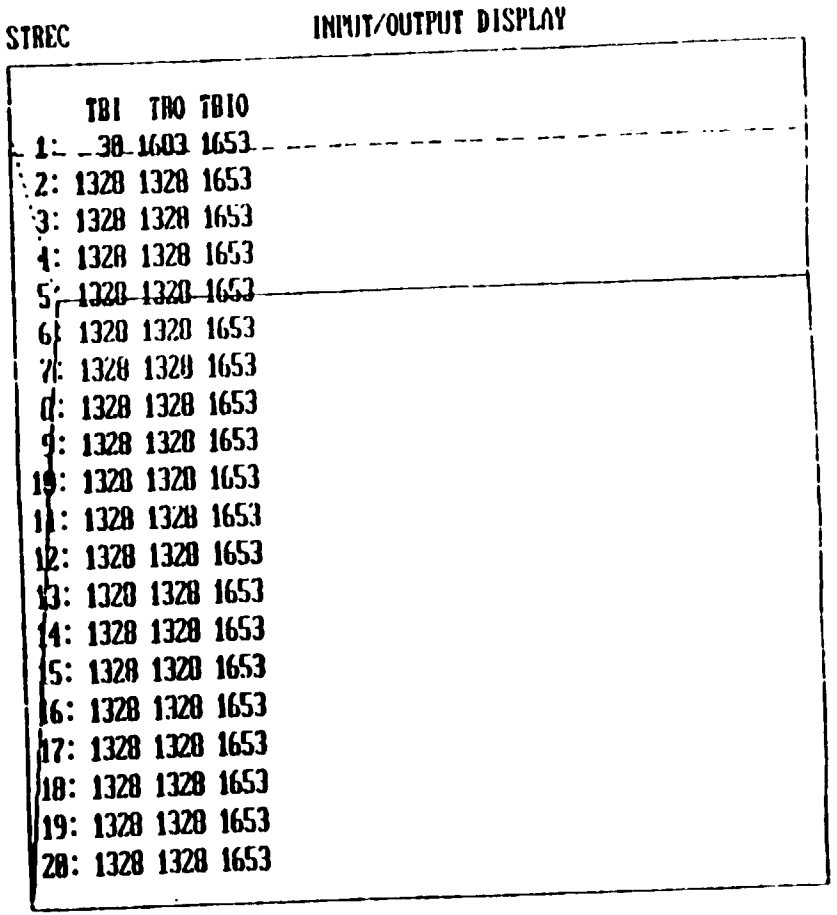


Figure 4.10. Simulation results for the AMG in Test 1.

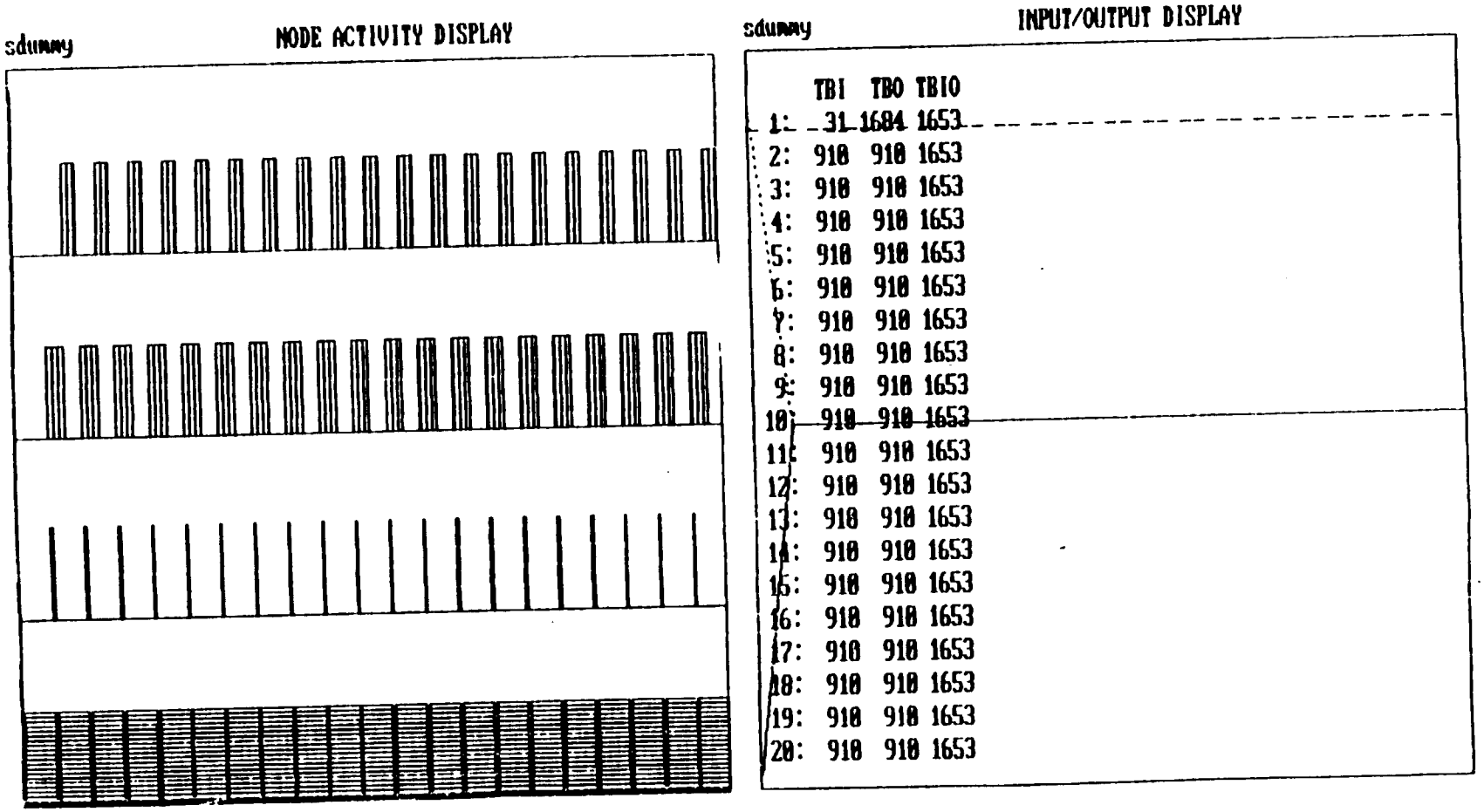


Figure 4.11. Simulation results for the transformed AMG in Test 1.

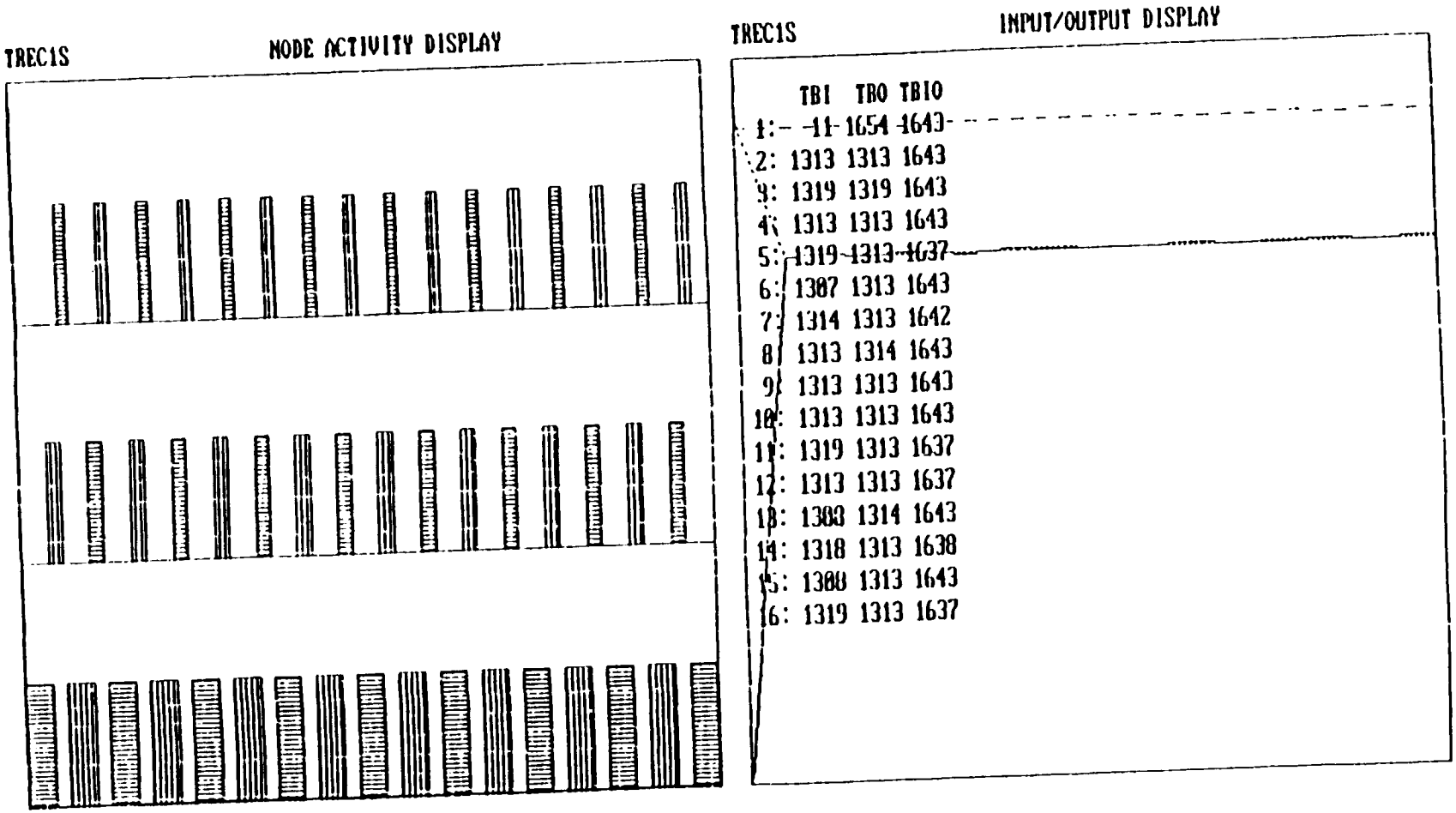


Figure 4.12. Experimental results for the AMG in Test 1.

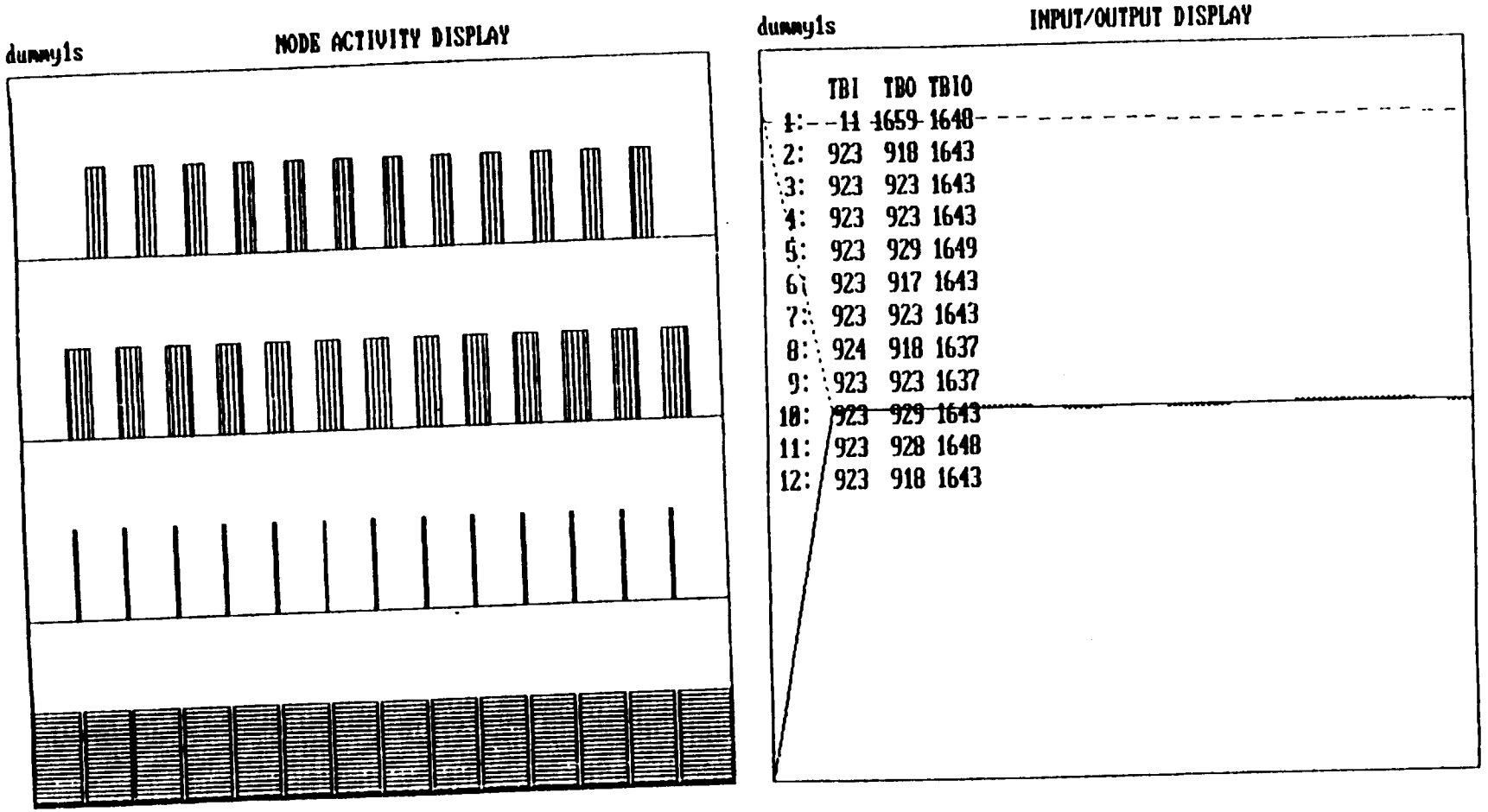


Figure 4.13. Experimental results for the transformed AMG in Test 1.

execution of transitions with time in the order of transition numbers, with transition 1 being the lowest. TBI, TBO, and TBIO of the input/output display are to be divided by 100 for converting all times to seconds. From the input/output display there is a significant gain in TBO by the transformation. Performance varies very little with task inputs. From the table, it can be seen that TBO_{LB} is improved from 13.17s to 8.695s by the dummy transition. It can also be seen that the experimental and simulated performances are very close to the theoretical lower bounds of performance, except for the TBO of the transformed test AMG. This is primarily due to the fact that the read of transition 3 and that of the dummy transition in Figure 4.9(b) cannot occur at the same time. Also, as there are only two resources with the priority of transition 1 being the lowest, no new task input will be accepted until the operation of the dummy transition is completed. All other results are as expected.

Test 2. This test illustrates the use of control place to reduce resource requirements (peak of TRE) while maintaining TBO_{LB}. Also, theoretical and experimental time performances are compared. The test AMG and the transformed AMG are shown in Figures 4.14(a) and 4.14(b) respectively. The test AMG of Figure 4.14(a) requires three resources to operate at $TBIO_{LB}$ and TBO_{LB}. The AMG is transformed as shown in Figure 4.14(b) which achieves TBO_{LB} with only two resources at the expense of increasing $TBIO_{LB}$ (assuming that no resources are required for the dummy transition). All the transition times are expressed in seconds. Priority ordering from highest to lowest for the AMG of Figures 4.14(a) and 4.14(b) are 4, 2, 3, 1 and

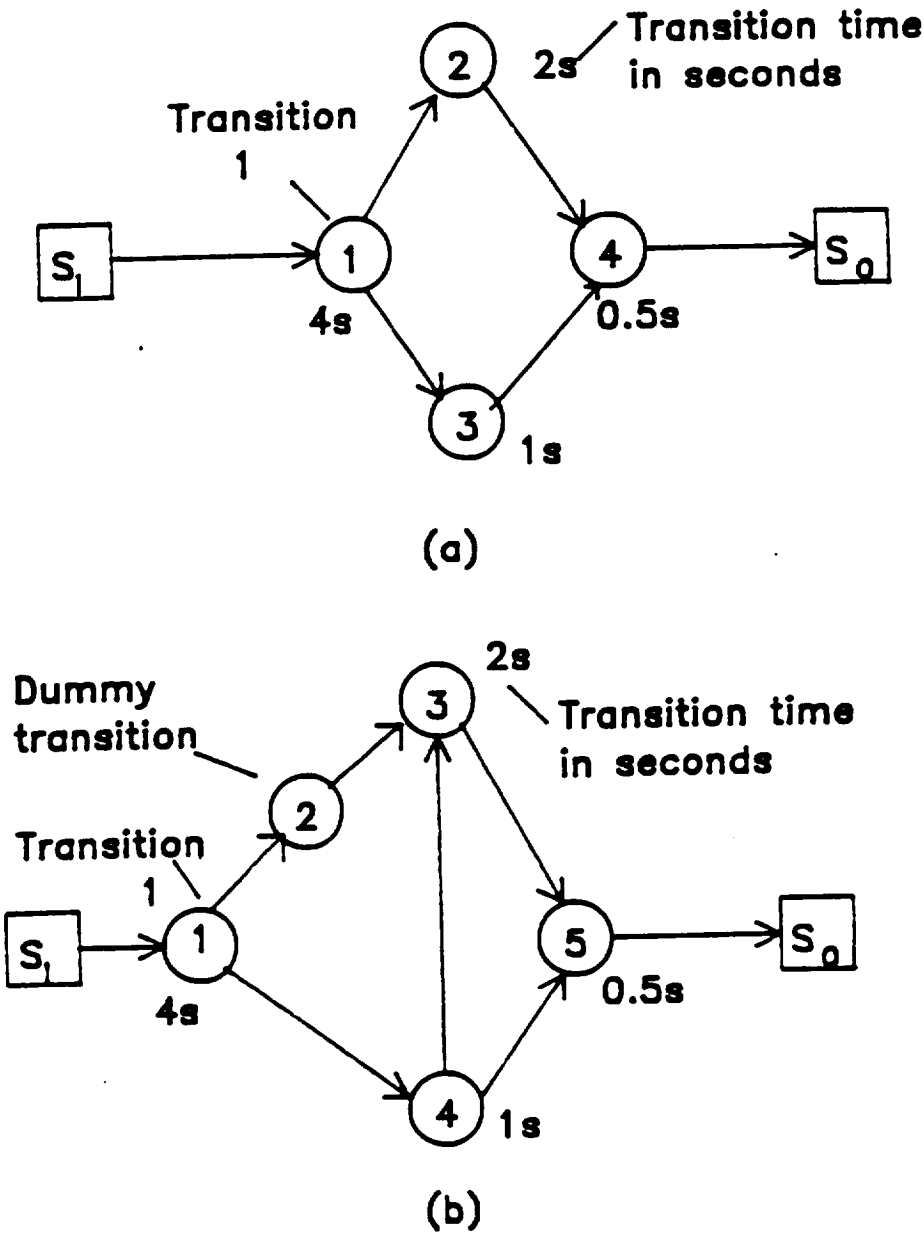


Figure 4.14. (a) AMG for Test 2.
(b) Transformed AMG for Test 2.

5, 3, 4, 2, 1 respectively. Read and write times for each NMG transition were measured in [20] to be 0.275s and 0.31s respectively for three resources. The test AMG of Figure 4.14(a) and the transformed AMG of Figure 4.14(b) are run on the testbed and simulated with three and two resources respectively. Experimental and simulation results are described in Figures 4.15 through 4.18 and compared with theoretical lower bounds in Table 4.2. In Figures 4.15 through 4.17, TBI, TBIO, and TBO are divided by 100 to get time in seconds. The times in the input/output display of Figure 4.18 are divided by 18.2 to get time in seconds. It can be observed that the transformed AMG achieves almost the same TBO as the original AMG; however, TBIO is increased by nearly the time for transition 3 of Figure 4.14(a) in the experiment and simulation. The differences in experimental results from theoretical lower bounds for both the AMG's are primarily due to nonideal dummy transition and Ethernet communication problems, as described in Test 1. The difference in the simulation results from the theoretical expected performance is mainly due to lost clock cycles in assigning transitions to resources and due to nonideal dummy transitions. The experimental performance for the transformed AMG unexpectedly went through a wide variation initially. One probable reason is the lack of proper injection control, which may cause the communication software (for implementing Ethernet communication) to be unpredictable. All other results are as expected.

Test 3. This is a simulation for the execution of a test algorithm shown in Figure 4.19(a) to check the ATAMM operating point design procedure. Let T = 1000 time units. The read and write times of the

TABLE 4.2

COMPARISON OF RESULTS FOR TEST 2

Algorithms	Experimental Results (s)		Simulation Results (s)		Theoretical LB's (s)	
	Av. TBO	Av. TBIO	Av. TBO	Av. TBIO	TBO_{LB}	$TBIO_{LB}$
AMG for Test 2	5.00	8.25	4.98	8.36	4.86	8.255
Transformed AMG for Test 2	5.16	9.81	5.13	9.56	4.70	9.4

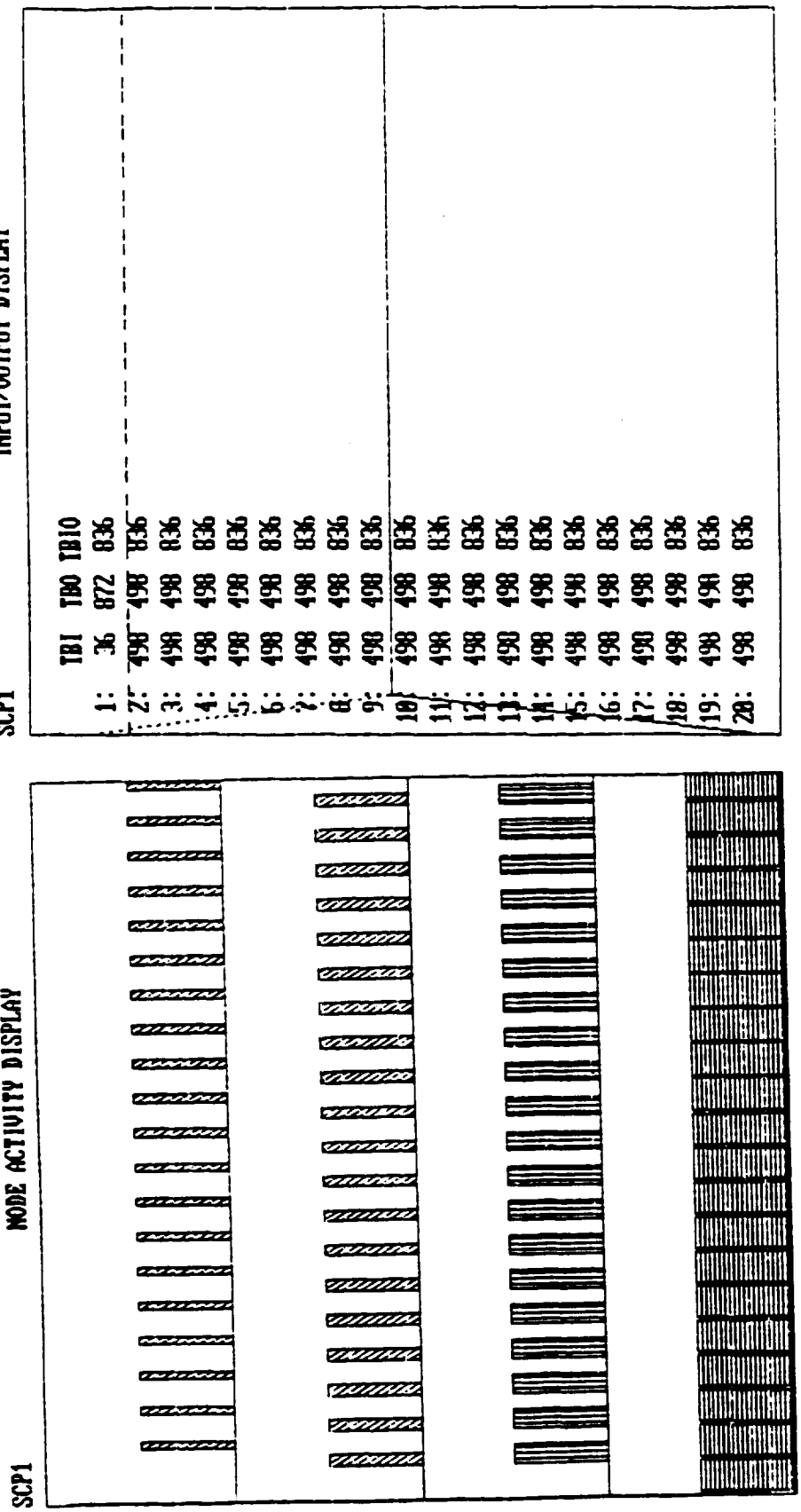


Figure 4.15. Simulation results for the AMG in Test 2.

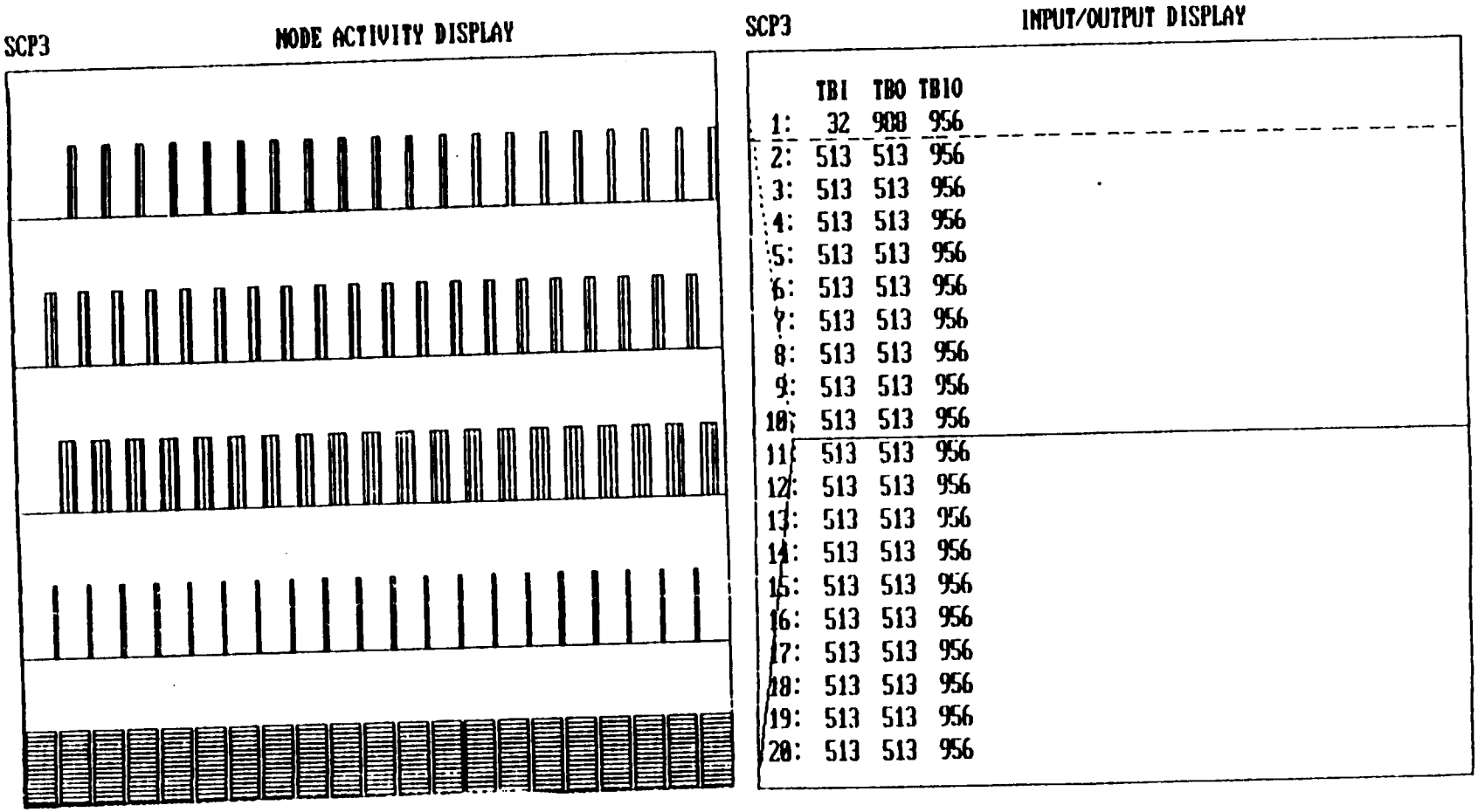
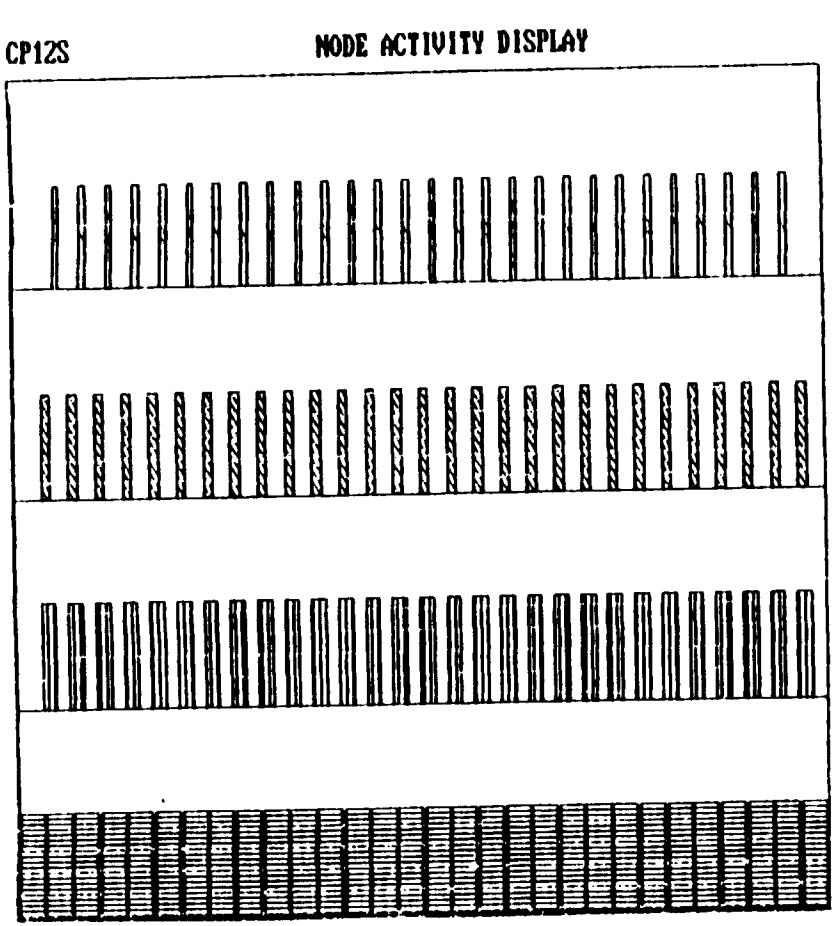


Figure 4.16. Simulation results for the transformed AMG in Test 2.



CP12S INPUT/OUTPUT DISPLAY

	TBI	TBO	TBIO		TBI	TBO	TBIO
1:	16	846	830	21:	500	500	825
2:	506	500	824	22:	500	500	825
3:	500	500	824	23:	500	500	825
4:	500	500	824	24:	500	505	830
5:	500	500	824	25:	506	500	824
6:	500	500	824	26:	500	500	824
7:	500	500	824	27:	500	500	824
8:	500	500	824	28:	500	500	824
9:	500	500	824				
10:	500	506	830				
11:	505	500	825				
12:	500	500	825				
13:	500	500	825				
14:	500	500	825				
15:	500	500	825				
16:	500	500	825				
17:	500	500	825				
18:	500	500	825				
19:	500	500	825				
20:	500	500	825				

Figure 4.17. Experimental results for the AMG in Test 2.

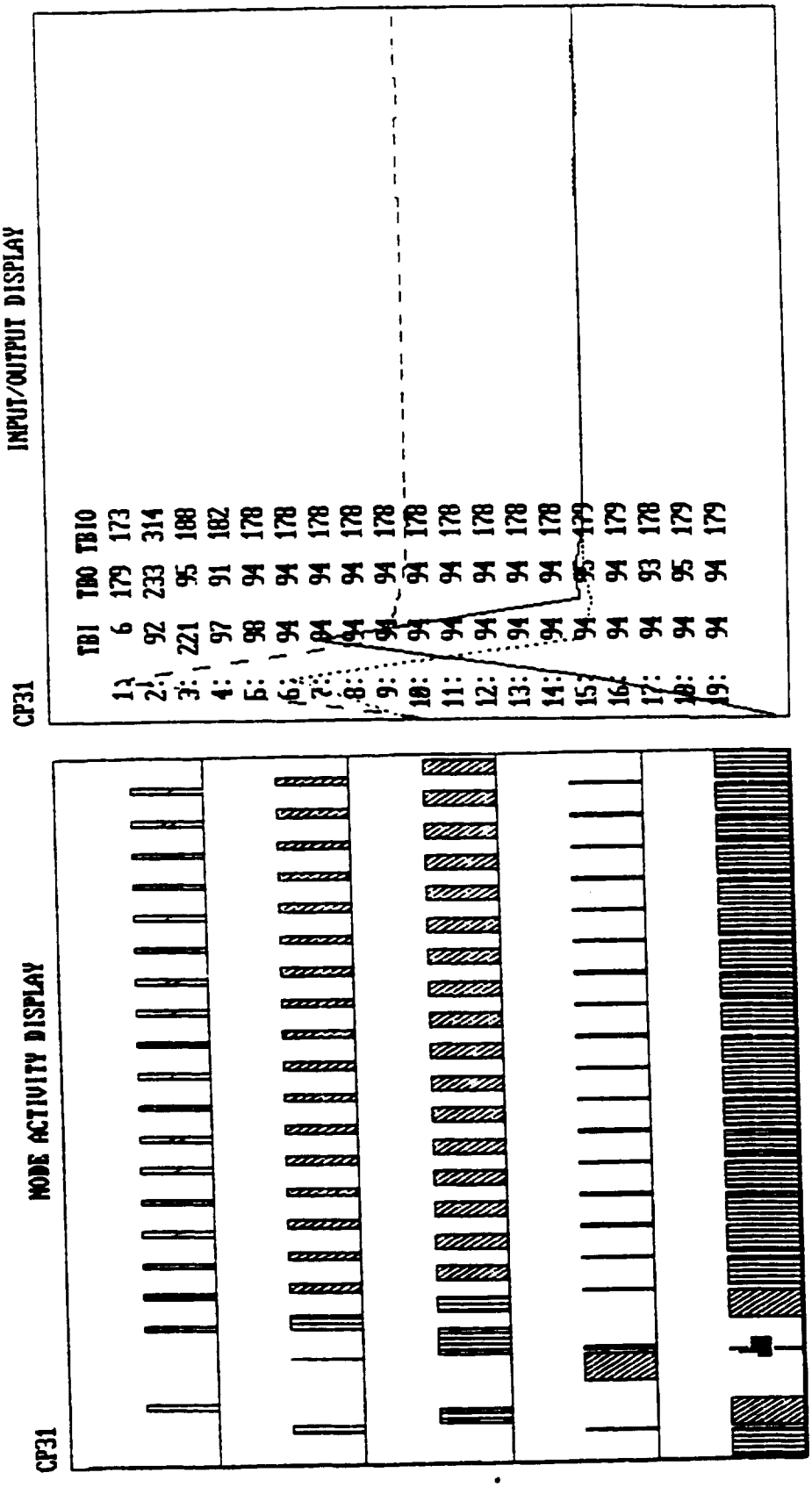
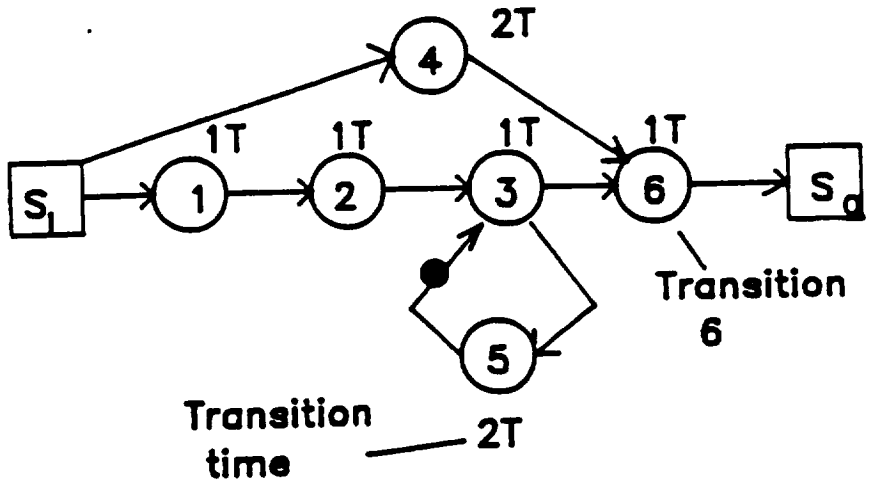
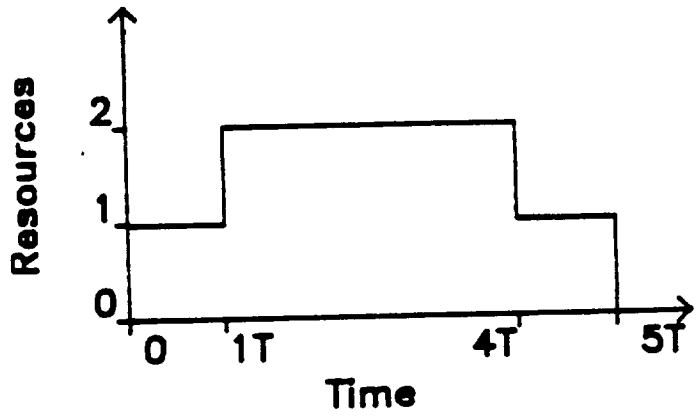


Figure 4.18. Experimental results for the transformed AMG in Test 2.



(a)



(b)

Figure 4.19. For Test 3, (a) AMG. (b) REST.

NMG transitions are assumed to be zero. Then $TBIO_{LB}$ = 4T, TT_{LB} = 5T, and TBO_{LB} = 3T. No further improvement of TBO_{LB} is possible as it is determined by the time/token ratio of the recursion circuit. Hence, $TBIO_{ALB}$ = 4T, TT_{ALB} = 5T, and TBO_{ALB} = 3T. REST is shown in Figure 4.19(b). By checking out all TBO $\geq$ TBO_{ALB}, R_{max} = 3, and R_{min} = 2. Also TC = 8T, TCE = 8T units of computer time. As $\lceil TCE/TBO_{ALB} \rceil$ = 3, R_{max} cannot be improved any further and Strategies B and C cannot be applied. So if R $\geq$ 3, the ATAMM operating point is determined by Step 3 as TBI = 3T, TBIO = 4T, TT = 5T, and TBO = 3T for all task inputs. As there are no floating transitions, Application 4 is not required. For R = 2, Strategy A of Step 4 in the ATAMM operating point design determines TBI = 4T, TBIO = 4T, TT = 5T, and TBO = 4T for all task inputs. The AMG execution at the operating points determined by Steps 3 and 4 are simulated and results are described in Figures 4.20 and 4.21 respectively. The achieved time performance in simulation is very close to the predicted theoretical time performance of the ATAMM operating point design. In the simulation of the operating point given by Step 3, TBI = 3.02T is used instead of 3T because TBO_{ALB} is slightly higher in the simulation due to lost clock cycles.

Test 4. The algorithm of Test 4 is a subsystem of a Space Surveillance System and is described in Figure 4.22(a) (ignore the dotted line). Let T = 100 time units. The read and write times of NMG transitions are assumed to be zero. Then, $TBIO_{LB}$ = TT_{LB} = $TBIO_{ALB}$ = TT_{ALB} = 18T and TBO_{LB} = TBO_{ALB} = 10T. REST is shown in Figure 4.22(b). By checking out all TBI $\geq$ TBO_{ALB}, R_{max} = 4, and R_{min} = 3. Now TCE = 25T units of computer time. As

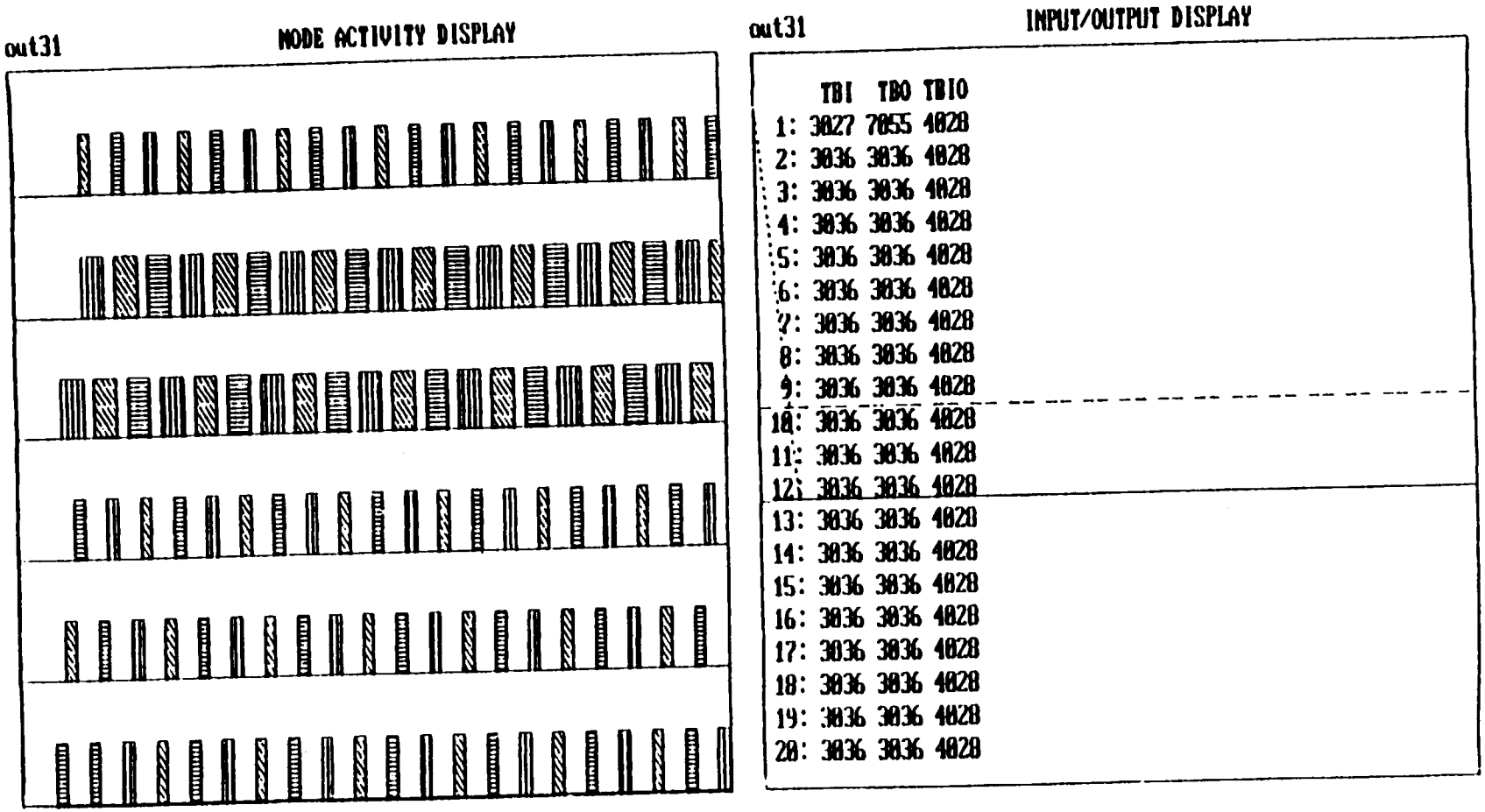


Figure 4.20. Simulation results for AOP of Step 3 in Test 3.

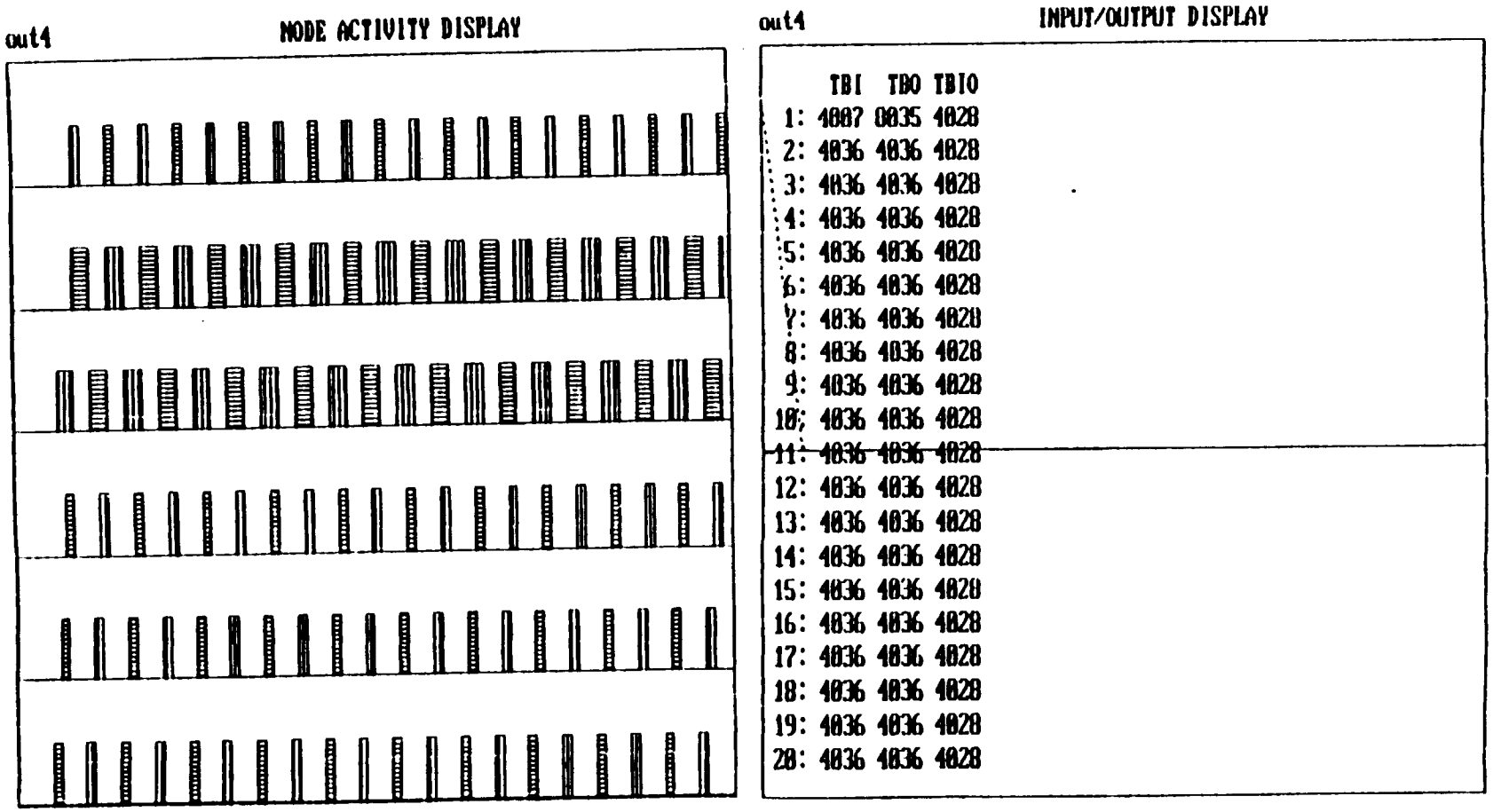
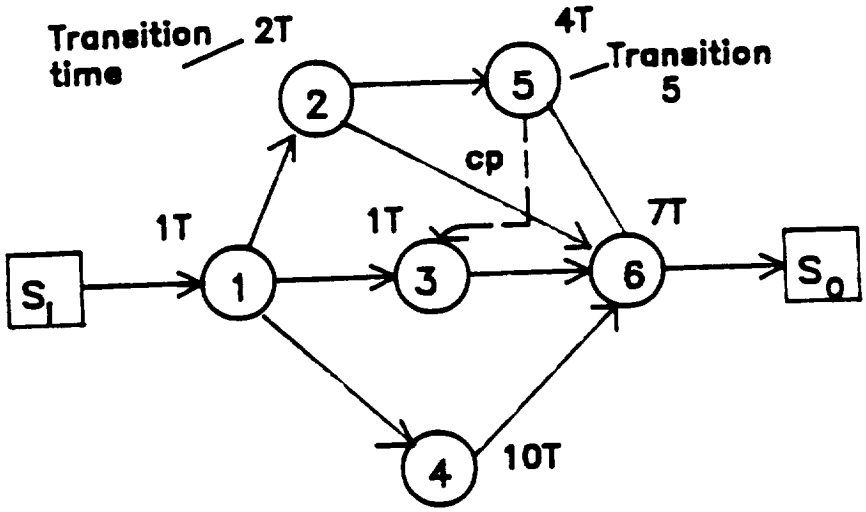
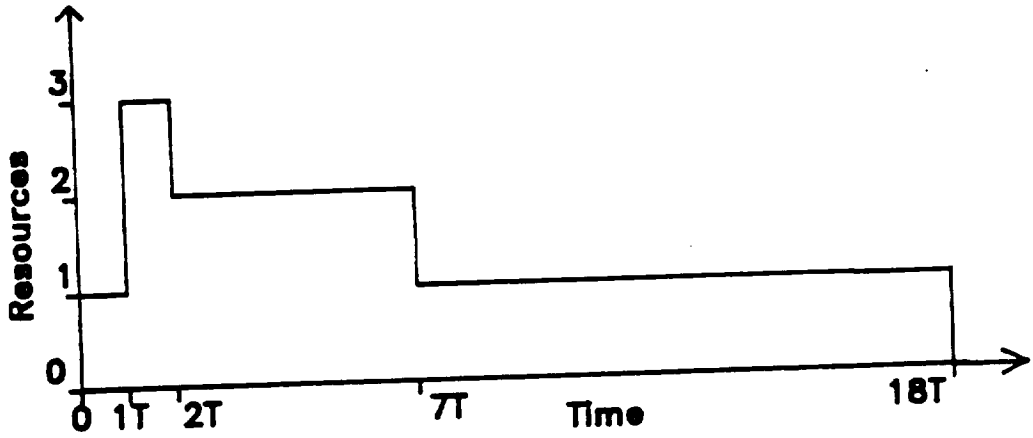


Figure 4.21. Simulation results for AOP of Strategy A in Test 3.



(a)



(b)

Figure 4.22. (a) AMG for Test 4. (b) REST for the AMG of Test 4.

$\lceil TCE/TBO_{ALB} \rceil = 3$, it may be possible to lower R_{max} to 3. A control place is placed from transition 5 to 3 for that purpose, as shown by the dotted line in Figure 4.22(a). The new REST is shown in Figure 4.23(a). It was checked by the Ttime program that $TBIO_{LB}$, TT_{LB}, and TBO_{LB} were unchanged by the control place. By checking all TBI $\geq$ 10T, $R_{max} = 3$, and $R_{min} = 2$. Hence, Strategies B and C of the ATAMM operating point design are not appropriate as R_{max} will always be equal or more than 3. For $R \geq 3$, Step 3 of the ATAMM operating point design determines TBI = 10T and TBIO = TT = 18T for all task inputs. For R = 2 Strategy A of the ATAMM operating point design determines TBI = 17T, TBO = 17T, and TBIO = TT = 18T. The graph play for a single task and the total graph play for TBO = 10T is shown in Figures 4.23(b) and 4.24 respectively. By inspection of TGP, no dummy transition is required to enforce GPST and REST. The AMG execution at the operating points, determined by Steps 3 and 4, are simulated and the results are described in Figures 4.25 and 4.26 respectively. The achieved time performance in simulation is very close to the predicted time performance of the ATAMM operating point design.

Test 5. Execution of the algorithm marked graph in Figure 3.3 is simulated for all the operating points developed in Section 4.2. All the process times for the transitions of the AMG are multiplied by T (T = 1000 time units) in the simulation. The read and write times of the NMG transitions are assumed to be zero. The results of the simulation for the operating points of Steps 3 through 6 are described in Figures 4.27 through 4.30 respectively. It is to be noted that the TBI's used in the simulation for the operating points in Steps 4

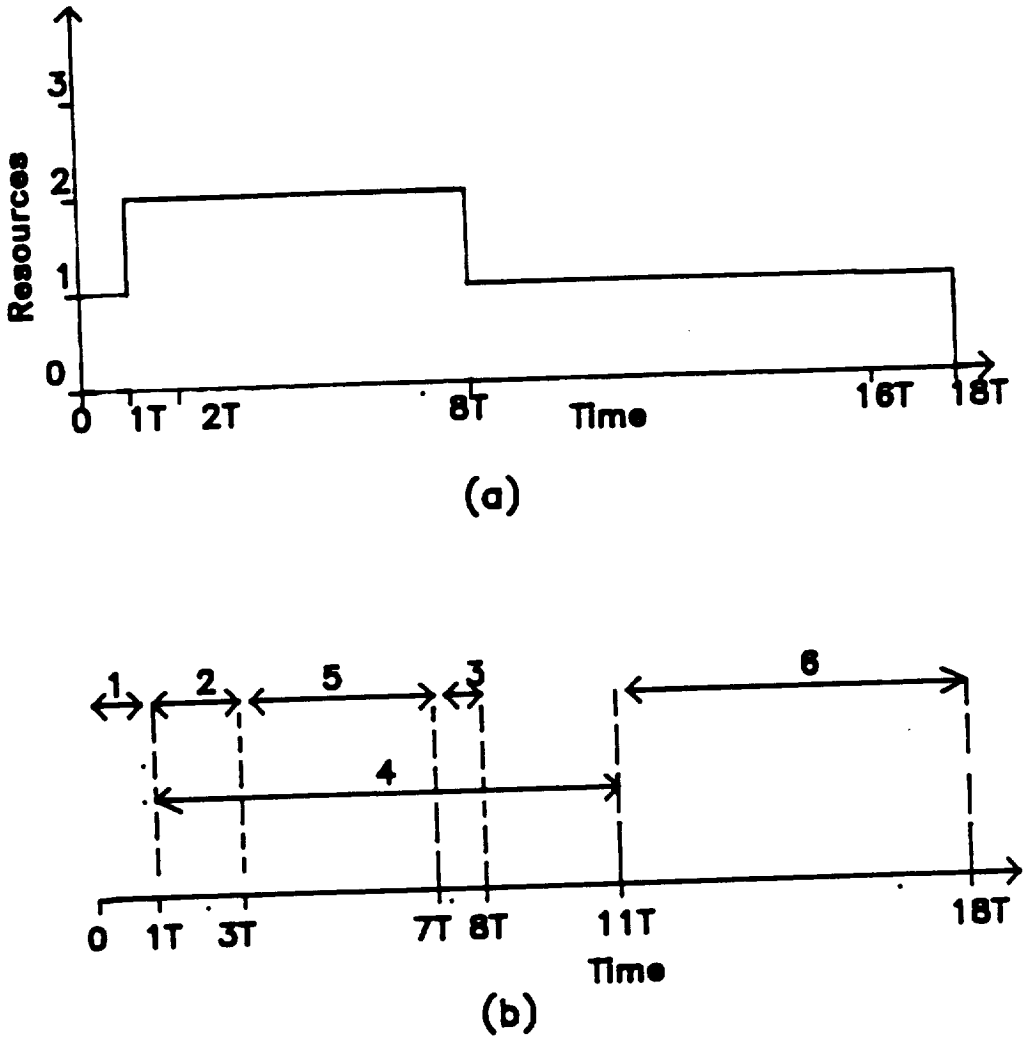


Figure 4.23. For the transformed AMG, (a) REST. (b) GPST.

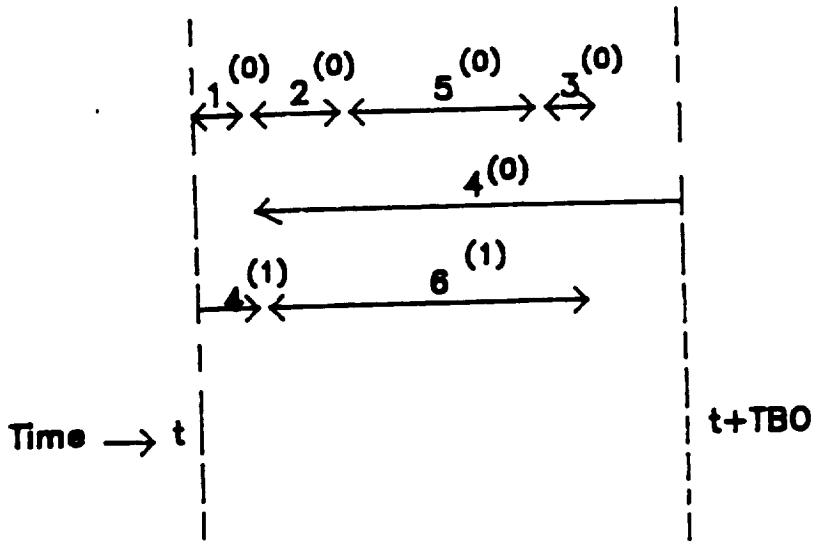


Figure 4.24. TGP of the transformed AMG for TBO=10T.

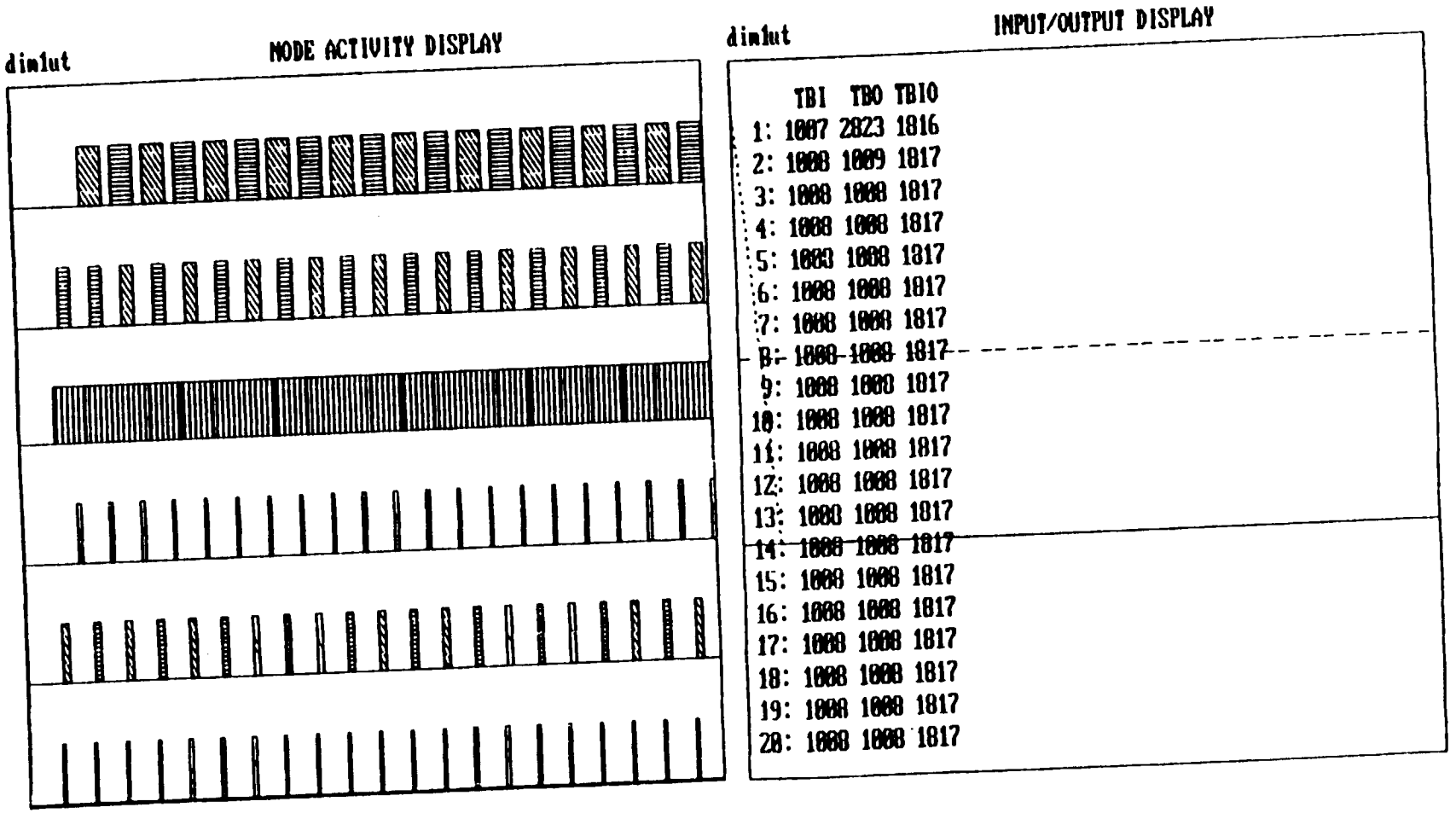
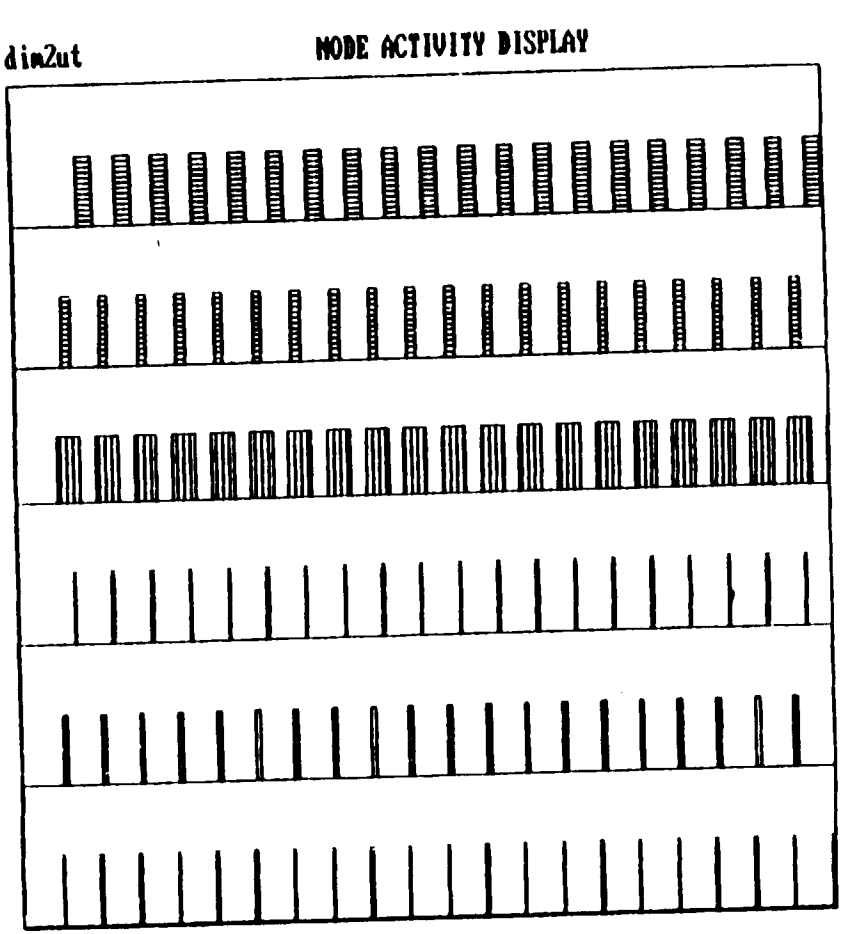


Figure 4.25. Simulation results for AOP of Step 3 in Test 4.



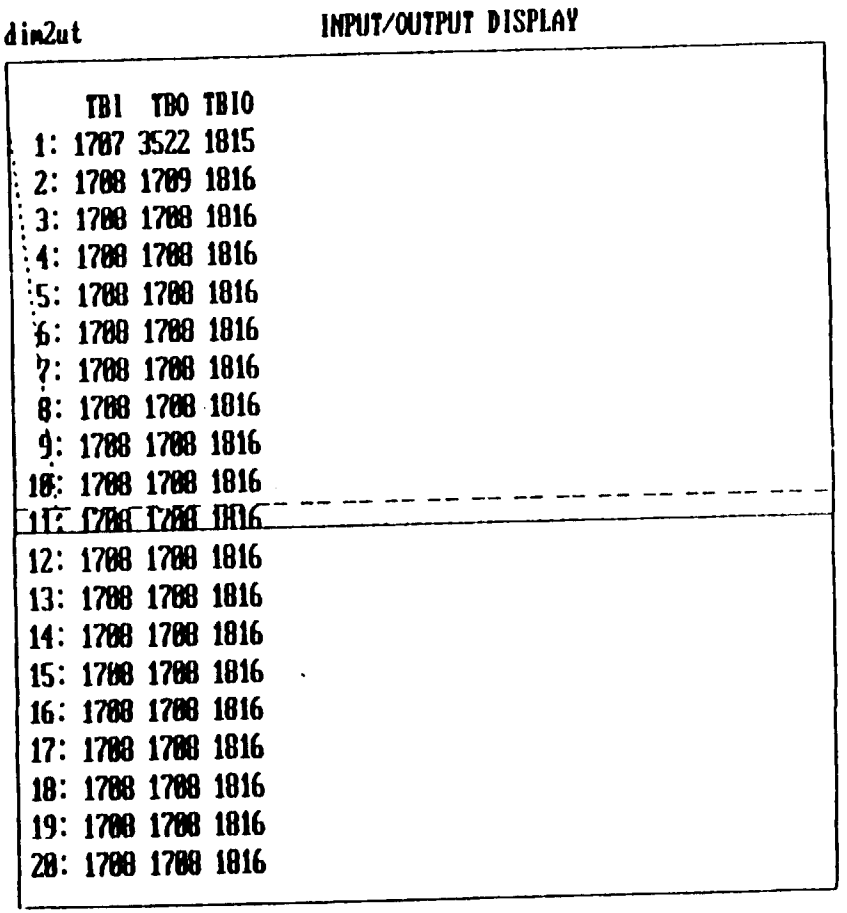


Figure 4.26. Simulation results for AOP of Strategy A in Test 4.

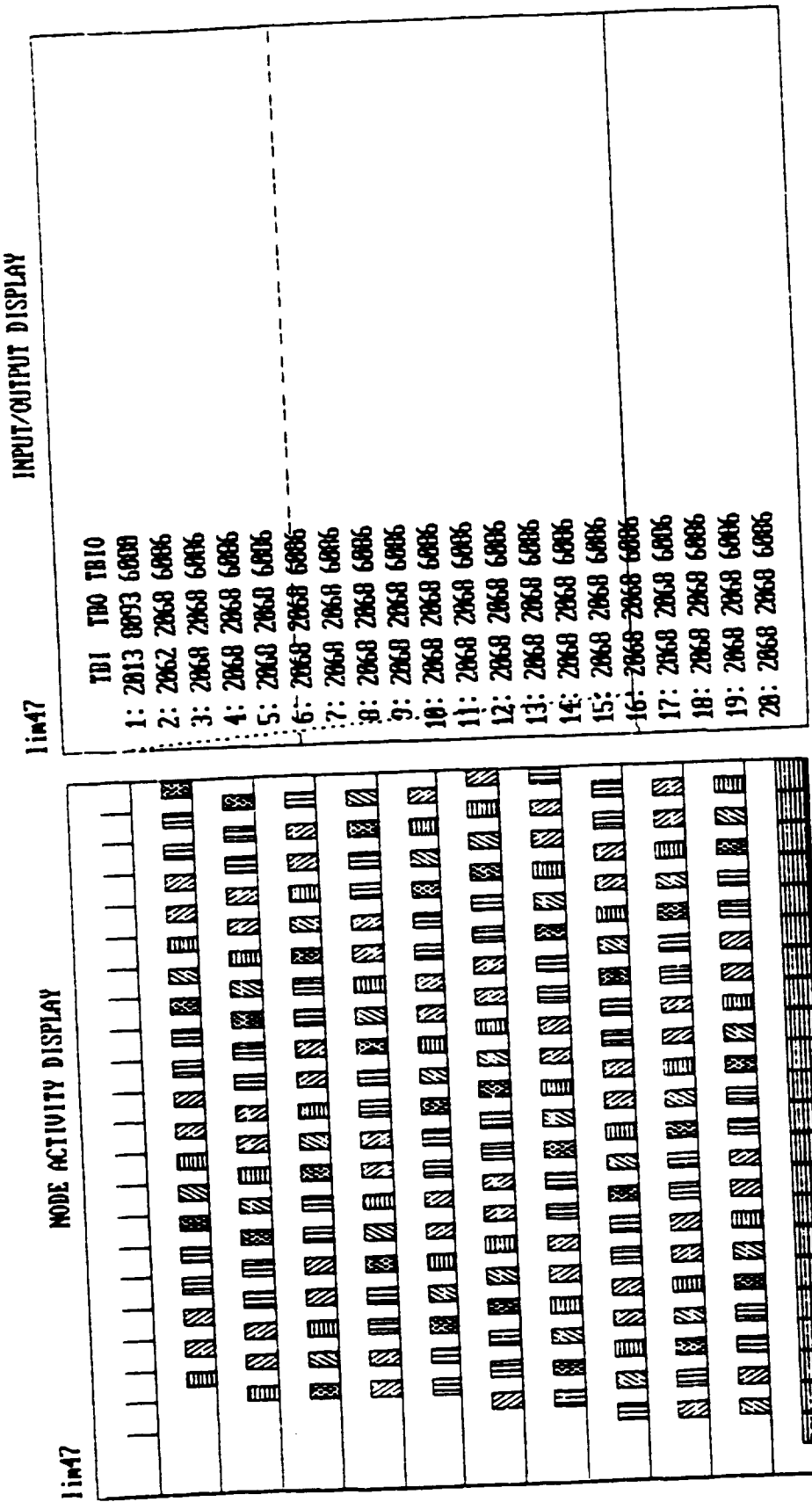


Figure 4.27. Simulation results for AOP of Step 3 in Test 5.

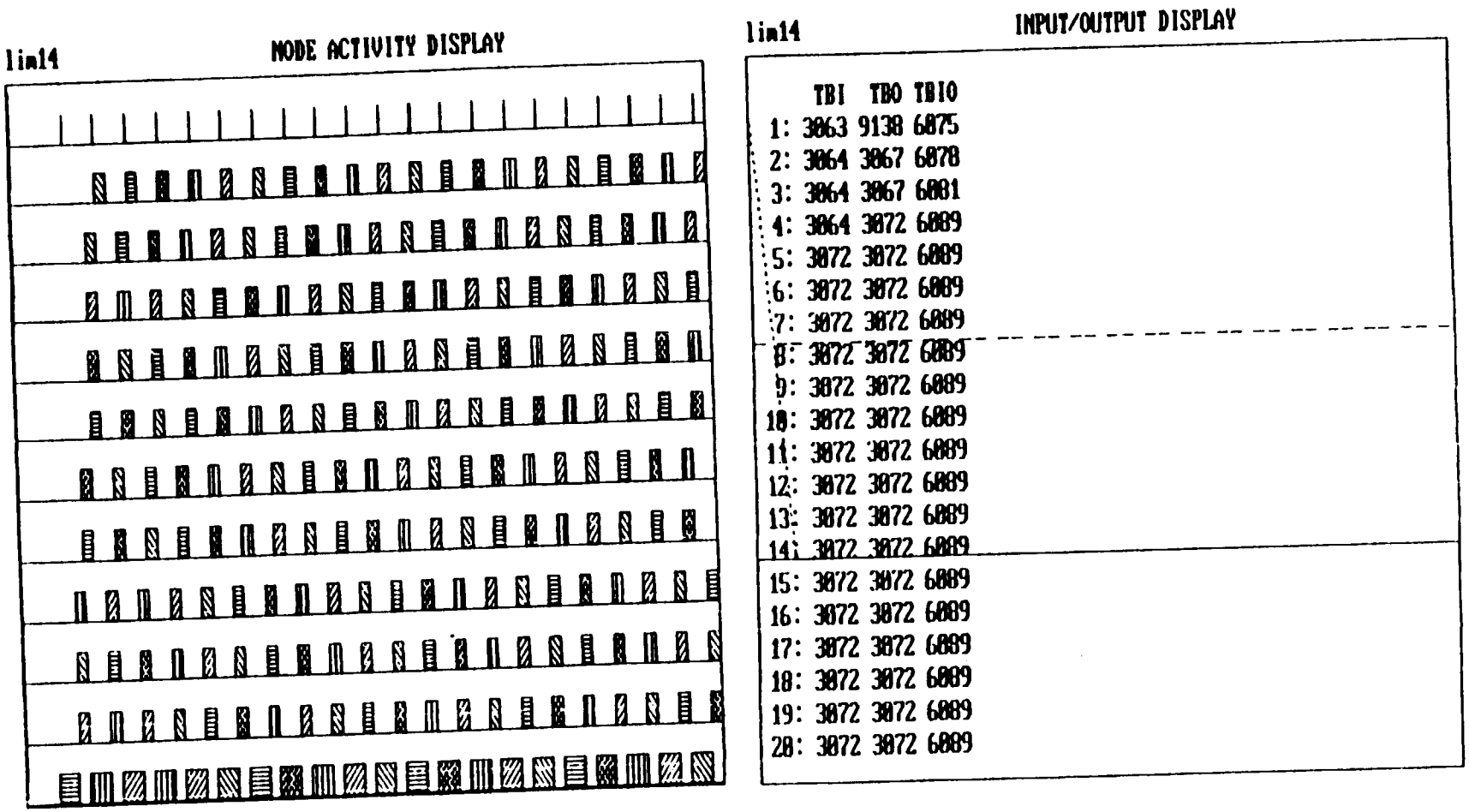


Figure 4.28. Simulation results for AOP of Strategy A in Test 5.

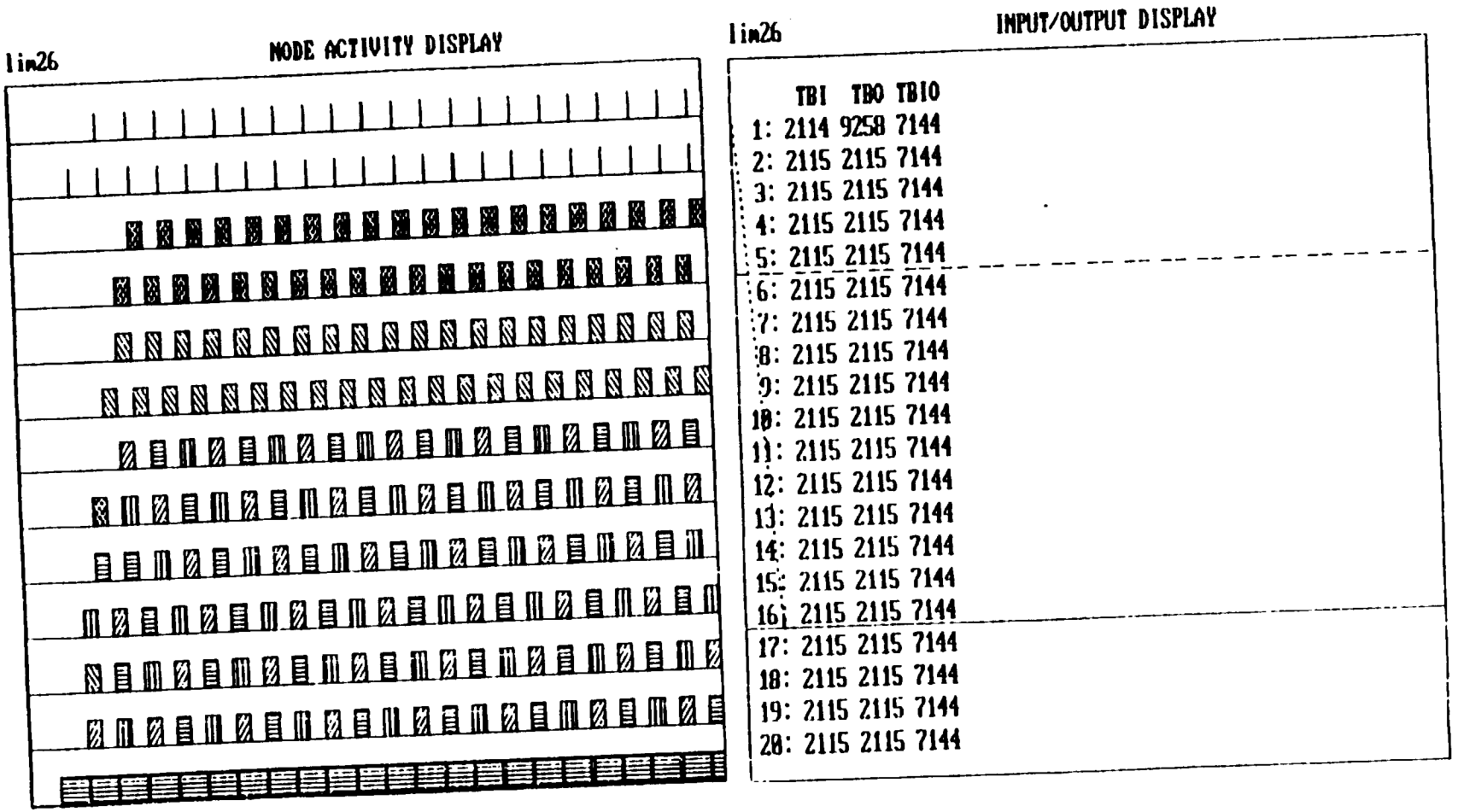
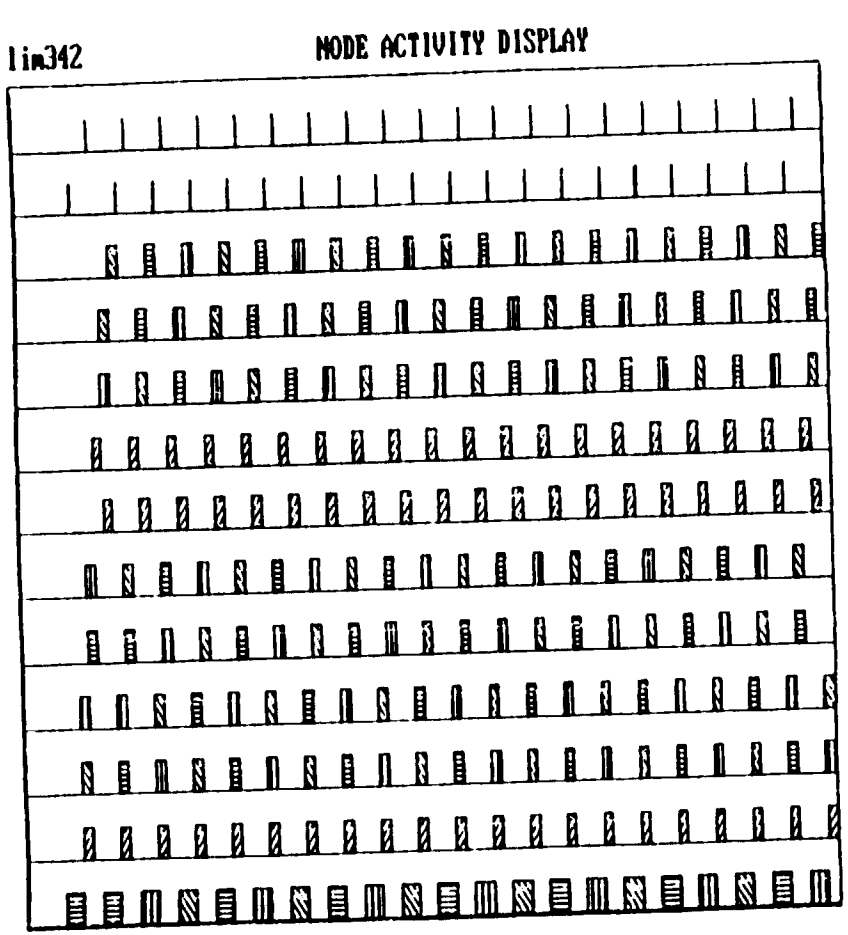


Figure 4.29. Simulation results for AOP of Strategy B in Test 5.



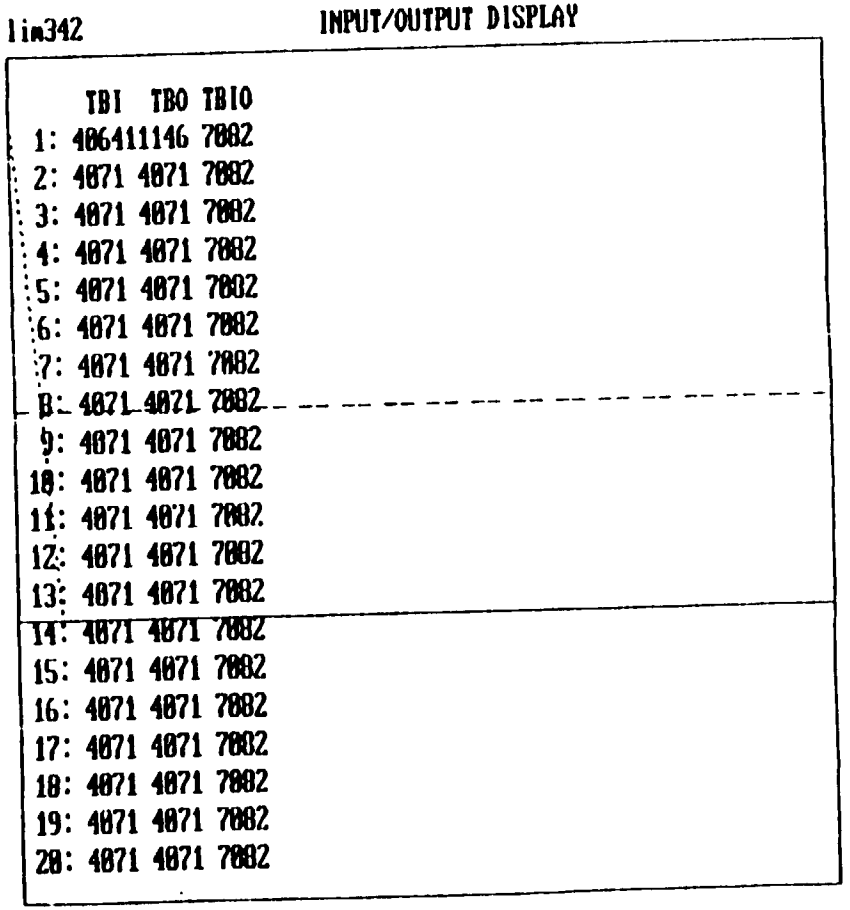


Figure 4.30. Simulation results for AOP of Strategy C in Test 5.

through 6 are slightly higher than the value predicted in the ATAMM operating point design. The reason is, again, a slight increase in the transition times of the AMG in the simulation due to the time needed to assign transitions to resources.

4.4 Summary

A new term, the ATAMM operating point (AOP), is defined to express all the parameters of an algorithm execution in the ATAMM data flow architecture. The characteristics of an AOP are explored for finite resources and under specified transformations. The absolute lower bounds for performance measures are defined. $TBIO_{ALB}$, TT_{ALB}, and TBO_{ALB} are determined under transformations by control places and dummy transitions. A procedure is developed for operating point design given the number of functional units. The performance model and the use of dummy transitions and control places for improving time performance and resource requirements are illustrated through experiments and simulations. The ATAMM operating point design methodology is checked by simulations on test algorithms.

CHAPTER FIVE

CONCLUSION

Performance modeling and enhancement for concurrent processing in the ATAMM data flow architecture have been the primary thrust for this research. Several key results are achieved in that respect. First, a performance model is developed to determine performance of an algorithm executed periodically in the ATAMM data flow architecture. Second, algorithm transformation techniques are identified and their applications are illustrated in improving time performance and resource (computing element) requirements. Third, an ATAMM operating point design procedure is developed to specify time performance and input data injection control for periodic execution of an algorithm on an ATAMM data flow architecture. Significant results in these three areas have been discussed. Finally, future research topics are suggested.

The starting point of this research has been to define the computing environment and performance measures for the periodic execution of algorithms in the ATAMM data flow architecture. The architecture is assumed to have R identical computers, or functional units, and executes algorithms according to the rules of ATAMM. These computers, or functional units, are also denoted by the terms resource and computing element. The performance of an algorithm is measured by the time between input and output (TBIO), task time (TT), and time between outputs (TBO). Graph theoretic and resource imposed bounds

are developed for these performance measures. Also, the graph execution pattern and resource requirements are defined through GPST, REST, TGP, and TRE. These results establish a new model for evaluating performance of algorithms in a hardware independent context as long as the architecture obeys the rules of ATAMM. Hence, it is now possible to compare the relative merits of different algorithm decompositions with respect to performance and resource requirements for the ATAMM data flow architecture.

The performance model enables the user to identify the cause of performance limitations. It is observed that the critical circuits of the CMG and the critical paths of the MAMG are the determining factors for the graph theoretic lower bounds of time performance. Also, the total resource requirement (the peak value of TRE) is determined by the shape of the resource envelope (REST) and TBO. Hence, it may be possible to enhance performance or reduce resource requirements by transforming the algorithm marked graph while maintaining its equivalency. Algorithm transformation techniques are identified which can be used to improve time performance or aid resource envelope modification. Transformation of an AMG may, or may not, involve decomposition of transitions. This research has concentrated on two of the transformation techniques, namely dummy transitions and control places. Concentration on these techniques is due to their wide range of applications, ease of implementation, and negligible increase in communication time by transformation. The most important contribution of this research is the application of dummy transitions which provide storage space for output of transitions. Dummy transitions have made parallel path circuits in the CMG insignificant for determining

TBO_{LB}. Thus, it is now possible to use control places and dummy transitions together to change the REST without increasing TBO_{LB}. Dummy transitions can improve TBO_{LB} by reducing the time/token ratio of dominant parallel path circuits. Another application of dummy transition is to enforce the REST as the resource envelope for all task inputs. Hence, it is now possible to enhance the throughput of an algorithm execution in the ATAMM data flow architecture. Also, the algorithm marked graph can be transformed according to the resource capability of the architecture or to make the resource need for periodic operation predictable.

The ATAMM operating point (AOP) design procedure uses the knowledge of the performance model and algorithm transformation to specify an operating point for executing an algorithm in the ATAMM data flow architecture. The only transformations used for the AOP design are dummy transitions as buffer and control places. The AOP design describes the procedure to achieve the absolute lower bound of time performance under these transformations. It proposes three strategies corresponding to sacrificing pipeline concurrency, parallel concurrency, and a combination of both to meet the limited availability of resources. Pipeline and parallel concurrency can be reduced by reducing input data injection rate or by transforming the AMG to modify the shape of REST respectively. Although the design procedure is partially heuristic because of the NP completeness of the problem, it allows the user to make a trade-off between pipeline and parallel concurrency for limited availability of resources.

Test algorithms are simulated by a PC-based simulator [21] to validate the ATAMM operating point design procedure. The read and

write times of transitions are assumed to be zero. Process times of transitions are in the order of hundreds of clock cycles to keep the algorithms at a large-grain level. This order of transition times are appropriate as the simulator takes less than ten clock cycles for assigning transitions to resources. Dummy transitions and control places are realized as regular active transitions (of zero process time) or active places respectively. It is assumed that a dummy transition does not require a resource. Simulated performance of algorithms are always very close to that predicted by the AOP design (within 2.1% for TBIO and within 5.8% for TBI and TBO). One significant observation is that the proper input data injection interval in the simulation is slightly higher than that predicted by the AOP design (within 5.8%). These differences between theoretical and simulated results are mainly due to a slight increase in transition times by the unaccounted clock cycles in assigning transitions to resources.

Test algorithms are executed on a testbed ATAMM data flow architecture [20] to verify the performance model and the use of dummy transitions and control places for transformation of algorithms. Dummy transitions and control places are implemented as active transitions of zero process time and active places respectively. Read and write times for the transitions in the experiments are assumed to be those measured in [20]. The largest process time among the transitions of the test algorithm is kept at least ten times higher than read or write times for maintaining algorithms in the large-grain level. The performance model is verified as experimental time performances are close to theoretical time performances (within 4.4%

for TBIO and within 9.8% for TBO). The use of dummy transitions for making parallel path circuits insignificant is verified in Test 1. The TBO of the transformed AMG in Test 1 is determined by the time/token ratio of the largest process circuit (experimental TBO is 6.15% more). A control place and a dummy transition together in Test 2 have reduced the total resource requirement from 3 to 2 while maintaining the change in TBO within 3%. The larger difference between the experimental and theoretical results compared to the simulation can be attributed mainly to two reasons. First, implementing a dummy transition as an active transition has a much greater effect in the testbed. The dummy transition requires read and write times in the experiments and hence, requires a resource for a considerable amount of time contrary to the assumption. Second, as pointed out in [20], Ethernet cannot implement concurrent read or write operations. This fact is not taken into account in the measurement of read and write times. The experimental results suggest that a better method of implementing a dummy transition and a more accurate communication model for read and write times are necessary.

There are several topics that can be the subject of future research. On the theoretical side, the following problems need attention. In order to properly decompose an algorithm, a specific definition of large granularity is needed corresponding to the communication time of an ATAMM data flow architecture. The first step is to develop a general and more accurate model for read and write times. The use of dummy transitions of finite time, control places with initial tokens, and predefined tokens in performance improvement and reduction of resource requirements needs to be explored.

Experiments and simulations have shown that the proper input data injection interval is slightly higher than the predicted value. This observation and the possibility of slight variation in transition times suggest that automatic injection control may be necessary. Execution of multiple AMG's or AMG's with multiple input and output transitions provide a complex, but interesting, topic of future research. Finally, the performance of algorithms with conditional data flow need to be analyzed. On the implementation side, realizing dummy transitions as buffers in the functional unit or graph manager, a better technique for measuring communication times, a fully automated ATAMM operating point design procedure, and transformations of algorithms by dummy transitions and control places in real time are useful topics for future research.

LIST OF REFERENCES

1. J. W. Stoughton and R. R. Mielke, "Petri-Net Model for Concurrent Processing of Complex Algorithms," Proceedings of Government Microcircuit Applications Conference, San Diego, CA, November 1986.

2. R. R. Mielke, John W. Stoughton, and Sukhamoy Som, "Modeling and Performance Bounds for Concurrent Processing," Proceedings of the 8th International Conference on Distributed Computing Systems, San Jose, CA, June 1988.

3. J. Tiberghien, New Computer Architectures, Academic Press, London, 1984.

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

5. A. Holt and F. Commoner, Events and Conditions, Applied Data Research, NY, 1970.

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

7. Tadao Murata, "Synthesis of Decision-Free Concurrent Systems for Prescribed Resources and Performance," IEEE Transactions on Software Engineering, pp. 525-530, November 1980.

8. T. Murata and J. Koh, "Reduction and Expansion of Live and Safe Marked Graphs," IEEE Transactions on Circuits and Systems, vol. CAS-27, pp. 68-70, January 1980.

9. T. Agerwala and Arvind, "Data Flow Systems," Computer, pp. 10-13, February 1982.

10. Tadao Murata, "Relevance of Network Theory to Models of Distributed/Parallel Processing," Journal of Franklin Institute, pp. 41-49, 1980.

11. R. R. Mielke, John W. Stoughton, and Sukhamoy Som, "Modeling and Optimum Time Performance for Concurrent Processing," NASA Contractor Report, Grant NAG-1-683, August 1988.

12. M. Granski, I. Koren, and G. Silberman, "The Effect of Operation Scheduling on the Performance of a Data Flow Computer," IEEE Transactions on Computers, vol. 36, pp. 1019-1029, September 1987.

13. Tadao Murata, "Circuit Theoretic Analysis and Synthesis of Marked Graphs," IEEE Transactions on Circuits and Systems, vol. 24, pp. 400-405, July 1977.

14. E. G. Coffman, Computer and Job-Shop Scheduling Theory, pp. 190-194, John Wiley & Sons, NY, 1976.

15. E. G. Coffman, Jr. and P. J. Denning, Operating System Theory, Prentice-Hall, Inc., NJ, 1973.

16. K. G. Lockyer, An Introduction to Critical Path Analysis, Pitman Publishing Limited, London, 1969.

17. J. J. Moder and C. R. Philips, Project Management with CPM and PERT, pp. 63-83, Van Nostrand Reinhold, NY, 1964.

18. 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.

19. Dennis B. Gannon and John Van Rosendale, "On the Impact of Communication Complexity on Design of Parallel Numerical Algorithms," IEEE Transactions on Computers, vol. 33, pp. 1180-1191, December 1984.

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

21. R. Obando, "Software Tools for Performance Evaluation of Concurrent Processing," Master's Thesis, Old Dominion University, Norfolk, VA, August 1987.

22. R. Agrawal and H. V. Jagadish, "Partitioning Techniques for Large-Grained Parallelism," IEEE Transactions on Computers, vol. 37, pp. 1627-1634, December 1988.

23. S. H. Bokhari, "Partitioning Problems in Parallel, Pipelined, and Distributed Computing," IEEE Transactions on Computers, vol. 37, pp. 48-57, January 1988.

24. Z. Cvetanovic, "The Effect of Problem Partitioning, Allocation, and Granularity on the Performance of Multiple-Processor Systems," IEEE Transactions on Computers, vol. 36, pp. 421-432, April 1987.

25. R. Johnsonbaugh and T. Murata, "Additional Methods for Reduction and Expansion of Marked Graphs," IEEE Transactions on Circuits and Systems, vol. CAS-28, pp. 1009-1014, October 1981.

26. Dan I. Moldovan and Jose A. B. Fortes, "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays," IEEE Transactions on Computers, vol. C-35, pp. 1-12, January 1986.

27. C. V. Ramamoorthy and Gary S. Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," IEEE Transactions on Software Engineering, vol. 6, pp. 440-449, September 1980.

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

29. M. Sowa and T. Murata, "A Data Flow Computer Architecture with Program and Token Memories," IEEE Transactions on Computers, pp. 940-948, November 1986.

30. V. Srini, "An Architectural Comparison of Dataflow Systems," Computer, pp. 68-88, March 1986.

31. J. W. Stoughton and R. R. Mielke, "Petri Net Model for Analysis of Concurrently Processed Complex Algorithms," Proceedings of Southeastcon Conference, March 1986.

32. J. W. Stoughton and R. R. Mielke, "Strategies for Concurrent Processing of Complex Algorithms," Proceedings of Workshop on Future Directions in Computer Architecture and Software, Army Research Office, May 1986.

33. M. N. S. Swamy and K. Thulasiraman, Graphs, Networks, and Algorithms, John Wiley & Sons Publication, NY, 1981.

34. D. F. Vrsalovic, D. P. Siewiorek, Z. Z. Segall, and E. F. Gehringer, "Performance Prediction and Calibration for a Class of Multiprocessors," IEEE Transactions on Computers, vol. 37, pp. 1353-1365, November 1988.

35. P. M. Kogge, The Architecture of Pipelined Computers, Advanced Computer Science Series, McGraw-Hill, NY, 1981.

36. H. Tokuda, C. W. Mercer, Y. Ishikawa, T. E. Marchok, "Priority Inversions in Real-Time Communication," Proceedings of the Real-Time Systems Symposium, Santa Monica, California, December 5-7, 1989.

APPENDIX

This appendix is an excerpt from [11]. The ATAMM model is studied analytically to determine important graph operating characteristics. First, a state description which expresses the next graph marking as a function of the present marking and a vector indicating which transition is to be fired is developed. Then the marked graph properties of reachability, liveness, and safeness are considered for the CMG. Two excellent papers by Murata [13, 18] on properties of marked graphs are the sources for much of the material presented in this appendix.

Let G be a marked graph consisting of m places and n transitions. The m-vector M_k denotes the marking vector for G resulting from the firing of some sequence of k transitions. The following two definitions are necessary to develop the state description of the CMG.

Definition A.1: Complete Incidence Matrix. The complete incidence matrix for a marked graph G is the (n x m) matrix $A = [a_{ij}]$ having rows corresponding to transitions and columns corresponding to places and where

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

Definition A.2: Elementary Firing Vector. An elementary firing vector u_k is an n-vector having all zero entries except for the i^{th} component, which is 1 denoting that transition i is the k^{th} transition to fire in some transition firing sequence.

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

Property A.1: Next State Description. For a marked graph G with present marking vector M_{k-1} and elementary firing vector u_k, the next marking vector is given by

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

The next state description can be used to express the graph marking resulting from the application of sequences of elementary firing vectors. This is done in the next definition and property.

Definition A.3: 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 firing sequence is defined by

$$x_d = \sum_{k=1}^{d} u_k.$$

Property A.2: State Equation Description. For a marked graph G with initial marking vector M_0, the marking vector resulting from the application of an elementary firing vector sequence $(u_1, u_2, \ldots, u_d)$ is given by

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

Using the state description of a marked graph as a basis, the property of reachability is investigated. Necessary and sufficient conditions for a CMG marking vector to be reachable from an initial marking are established, and it is shown that the number of tokens contained in any directed circuit of the CMG is invariant under transition firings.

Definition A.4: Reachability. A marking M_d is reachable from an initial marking M_0 if there exists a sequence of elementary firing vectors that transforms M_0 to M_d.

The following definition is required to state the reachability conditions for a CMG.

Definition A.5: Fundamental Circuit Matrix. Let T be a tree of a connected marked graph G. The set of (m-n+1) circuits, each uniquely formed by appending one cotree edge to the tree, is called the set of fundamental circuits of G for tree T [28]. The fundamental circuit matrix for G for tree T is the (m-n+1) x (m) matrix $B_f = [b_{ij}]$ having rows corresponding to fundamental circuits and columns corresponding to places, and where b_{ij} is determined by the rules as described on the next page.

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

Property A.3: Reachability in the CMG. 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. It is shown in [13] (Theorem 3) that the property is true for marked graphs containing no token-free directed circuits. By the construction rules for the CMG, directed circuits occur in exactly four ways. First, each NMG consists of a directed circuit which contains an initial marking token in the Process Ready place. Second, a directed circuit is formed each time an NMG is linked to another NMG. Since one of the two linking places contains an initial marking token and both places are contained in the circuit, this circuit is never token free. Third, directed circuits exist in the CMG corresponding to interconnected feedforward paths in the algorithm marked graph. Each such circuit contains one or more backward directed control edge containing one initial marking token. Fourth, directed circuits exist in the CMG corresponding to directed circuits in algorithm marked graph. Each such circuit contains exactly one forward directed edge containing one initial marking token which represents initial condition data. Therefore, the CMG contains no token-free directed circuits and the property follows.

As a direct consequence of the reachability property of the CMG, it can be shown that the number of tokens in any directed circuit is constant. This characteristic is stated as Property A.4.

Property A.4: Token Count Invariance. In a CMG, the number of tokens contained in a directed circuit is invariant under transition firing.

Proof. Consider a directed circuit C of a CMG. The entries in the row of a circuit matrix B corresponding to C are +1 in columns representing edges in C and are 0 otherwise. If M is a marking vector, the component of BM corresponding to C is equal to the number of tokens in directed circuit C marking M. Therefore, if M_d is any marking reachable from an initial marking M_0, it follows from Property A.3 that $BM_d = BM_0$. That is, the number of tokens in directed circuit C under initial marking M_0 is equal to the number of tokens under any marking M_d reachable from M_0. This completes the proof.

Next, liveness and a closely related property called consistency are considered. It is shown that the CMG is live and consistent.

Definition A.6: Liveness. A marked graph G is said to be live for a marking F if, for all markings reachable from M, it is possible to fire any transition of G by progressing through some transition firing sequence.

Property A.5: Liveness in the CMG. The computational marked graph is live for all appropriate initial marking vectors.

Proof. It is shown in [18] (Property 2) that a marked graph G is live for a marking M, if and only if, G contains no token-free directed circuits in marking M. As stated in the proof of Property A.3, for

all appropriate initial markings M_0, the CMG contains no token-free directed circuits. Therefore, the property follows.

Definition A.7: Consistency. A marked graph G is said to be consistent if there exists a marking M and a transition firing sequence S from M back to M such that every transition occurs at least once in S.

Property A.6: Consistency in the CMG. A connected computational marked graph G is consistent. In addition, each transition of G occurs an equal number of times in a firing sequence from a marking M back to M.

Proof. From Property A.2, if a CMG is consistent then there exists a marking $M_d = M_0$ and a firing count vector $x_d > 0$ such that $A^T x_d = 0$. The converse is also true. The incidence matrix for a marked graph G is an (n x m) matrix A. If G is connected, then it is known [28] 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 one (1), one (-1), and all remaining terms are zero (0). Therefore, if C_j denotes the j^{ih} column of A^T, it follows that

$$\sum_{j=1}^{n} C_j = 0.$$

Thus, there exists a vector $x_d = [k\ k....k]^T$, $k > 0$, which uniquely satisfies $A^T x_d = 0$. This completes the proof.

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

Definition A.8: Safeness. A marked graph G is said to be safe for marking M if, for all markings reachable from M, no place contains more than one token.

Property A.7: Safeness in the CMG. The computational marked graph is safe for all appropriate initial marking vectors.

Proof. By Property A.4, the token count for each directed circuit of the CMG is invariant under transition firing. Therefore, it is sufficient to show that each edge of the CMG belongs to at least one directed circuit containing a single token. By the construction rules for the CMG, all CMG edges can be classified into two groups NMG edges and linking edges. NMG edges occur in groups of three and always form a directed circuit containing one token. Linking edges occur in pairs, one forward directed and one backward directed, and also form a directed circuit with the forward directed edges of the NMG. One of the linking edges, but not both, always contains one token while the forward directed edges of the NMG contain no tokens. Therefore, each edge of the CMG is contained in a directed circuit with one token, and the property follows.