

Old Dominion University

ODU Digital Commons

Electrical & Computer Engineering Theses & Dissertations

Electrical & Computer Engineering

Fall 1990

A Software Design Tool for Predictable Performance in Real-Time, Data Flow Architectures

Brij Mohan V. Mandala
Old Dominion University

Follow this and additional works at: https://digitalcommons.odu.edu/ece_etds



Part of the [Computer and Systems Architecture Commons](#), [Software Engineering Commons](#), and the [Theory and Algorithms Commons](#)

Recommended Citation

Mandala, Brij M.. "A Software Design Tool for Predictable Performance in Real-Time, Data Flow Architectures" (1990). Master of Science (MS), Thesis, Electrical & Computer Engineering, Old Dominion University, DOI: 10.25777/js1c-h114
https://digitalcommons.odu.edu/ece_etds/426

This Thesis is brought to you for free and open access by the Electrical & Computer Engineering at ODU Digital Commons. It has been accepted for inclusion in Electrical & Computer Engineering Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.

A SOFTWARE DESIGN TOOL FOR PREDICTABLE PERFORMANCE
IN
REAL-TIME, DATA FLOW ARCHITECTURES

by

Brij Mohan V. Mandala
B.Tech. August 1988, V.R.S. Engineering College, Vijayawada, India

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfilment
of the Requirements for the Degree of

MASTER OF SCIENCE
ELECTRICAL ENGINEERING

OLD DOMINION UNIVERSITY
December, 1990

Approved by:

Roland R. Mielke (Director)

David L. Livingston

John W. Stoughton

Sukhamoy Som

ABSTRACT

A software design tool which aids in the performance evaluation and selection of operating points for an algorithm implemented in ATAMM defined data flow architectures is presented in this thesis. ATAMM (Algorithm To Architecture Mapping Model) is a new graph theoretic model developed by researchers at Old Dominion University and the NASA-Langley Research Center. ATAMM is capable of modeling the execution of large-grained algorithms on distributed data flow architectures. A software tool is required for predicting the performance, determining the resource requirements and for selecting suitable operating points for an ATAMM based system. The ATAMM Design Tool presented in this thesis is capable of determining the computing speed, throughput, and resource requirements for an algorithm. Case studies are performed on two real algorithms to demonstrate the applicability of the ATAMM Design Tool.

To My Mother, Father, and Brothers

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Roland R. Mielke, for his encouragement, guidance, patient understanding and invaluable advice during the development of this thesis. I wish to express my deep gratitude to Dr. Sukhamoy Som for many discussions regarding this research and for several hours of his valuable time. I would also like to thank Dr. John W. Stoughton, Rod Obando, Rob Jones, and Mahyar Malekpour for their constructive suggestions during the software development. My special thanks are due to Dr. David L. Livingston for being on my thesis committee. Finally, I wish to thank my family and friends for their encouragement in my pursuit for higher studies.

TABLE OF CONTENTS

	Page
LIST OF FIGURES	vi
LIST OF SYMBOLS	xii
Chapter	
1. Introduction	1
1.1 Research Problem	1
1.2 Overview	1
1.3 Research Objective	4
1.4 Thesis Organization	5
2. ATAMM Performance Model	7
2.1 Introduction	7
2.2 ATAMM Model	7
2.3 Performance Measures	16
2.4 Graph Play and Resource Requirements	17
2.5 ATAMM Performance Plane	24
2.6 An Example Implementation of ATAMM	29
3. ATAMM Design Tool	35
3.1 Introduction	35
3.2 Software Environment	36

3.3 Graph Editor Window	37
3.4 Data Generation	41
3.5 Bounds and Buffer Windows	51
3.6 SGP and SRE Windows	54
3.7 TGP and TRE Windows	61
3.8 Resources and Throughput Windows	65
3.9 Performance Plane Window	66
4. Case Studies	71
4.1 Introduction	71
4.2 Design Procedure	72
4.3 Space Surveillance Algorithm	72
4.4 Decomposed State Equation	97
5. Conclusion	129
5.1 Summary	129
5.2 Topics for Future Research	130
REFERENCES	132
APPENDIX	134

LIST OF FIGURES

FIGURE	PAGE
2.1 Marked Graph	9
2.2 An AMG Representation	11
2.3 An Example of NMG	12
2.4 An Example CMG	14
2.5 ATAMM Model Components	15
2.6 Injection Control	19
2.7 AMG Illustrating the definitions of Section 2.4.	20
2.8 An Example AMG	22
2.9 SGP Diagram for Figure 2.8.	23
2.10 TGP Diagram for Figure 2.8.	25
2.11 ATAMM Performance Plane	26
2.12 ATAMM Performance Plane for Figure 2.8.	28
2.13 Layout of ADM System	30
2.14 AMOS State Diagram	33
3.1 Graph Editor Window	40
3.2 Graph Structure	42
3.3 Graph File Format	43
3.4 Partial Graph File for Figure 3.1.	44

3.5 Organization of Routines	45
3.6 Example AMG and corresponding Modified AMG	48
3.7 Bounds Window	52
3.8 Example AMG	53
3.9 SGP and TGP Diagrams for the AMG of Figure 3.8.	56
3.10 Buffer Window	57
3.11 SGP Window	59
3.12 SRE Window	60
3.13 TGP Window	63
3.14 TRE Window	64
3.15 Resources Window	67
3.16 Throughput Window	68
3.17 Performance Plane Window	70
4.1 Space Surveillance Algorithm	73
4.2 Buffer Window for the Graph of Figure 4.1.	75
4.3 Bounds Window for the Graph of Figure 4.1.	75
4.4 SGP Display for the Graph of Figure 4.1.	76
4.5 SGP Display showing Floats	77
4.6 Sliced View of the SGP Display showing Test Times	78
4.7 SRE Display corresponding to the SGP Display of Figure 4.4.	79
4.8 TGP Display for the Graph of Figure 4.1.	80
4.9 TGP Display showing Floats	81

4.10 Sliced View of the TGP Display showing Test Times	82
4.11 TRE Display corresponding to the TGP Display of Figure 4.8.	83
4.12 Resources Window for the Graph of Figure 4.1.	85
4.13 Performance Plane Display for the Graph of Figure 4.1.	85
4.14 Throughput Display for the Graph of Figure 4.1.	86
4.15 Space Surveillance Algorithm with Control Edge from Node 4 to Node 2	87
4.16 Buffer Window for the Graph of Figure 4.15.	88
4.17 Bounds Window for the Graph of Figure 4.15.	88
4.18 SGP Display for the Graph of Figure 4.15 showing Floats	90
4.19 TGP Display for the Graph of Figure 4.15 showing Floats	91
4.20 TRE Display for the Graph of Figure 4.15	92
4.21 Resources Window corresponding to the Graph of Figure 4.15.	92
4.22 Performance Plane corresponding to the Graph of Figure 4.15.	93
4.23 Throughput Display for the Graph of Figure 4.15.	94
4.24 Graph with Control Edges from Nodes 4 to 2, 4 to 3, and 3 to 2	94
4.25 Buffer Window for the Graph of Figure 4.24.	95
4.26 Bounds Window for the Graph of Figure 4.24.	96
4.27 SGP Display corresponding to the Graph of Figure 4.24.	98
4.28 TGP Display corresponding to the Graph of Figure 4.24.	99
4.29 TRE Display for the Graph of Figure 4.24.	100
4.30 Performance Plane Display showing Modify Table	101
4.31 Throughput Display for the Graph of Figure 4.24.	102

4.32	Decomposed State Equation Algorithm	104
4.33	Buffer Window for the Graph of Figure 4.32.	105
4.34	Bounds Window corresponding to the Graph of Figure 4.32.	105
4.35	SGP Display for the Graph of Figure 4.32.	107
4.36	SRE Display corresponding to the SGP Display of Figure 4.35.	108
4.37	TGP Display for the Graph of Figure 4.32.	109
4.38	TRE Display corresponding to the TGP Display of Figure 4.37.	110
4.39	Resources Window for the Graph of Figure 4.32.	111
4.40	Performance Plane Display for the Graph of Figure 4.32.	111
4.41	Throughput Display for the Graph of Figure 4.32.	112
4.42	Graph with Control Edge from Node 1 to 2	113
4.43	Buffer Window for the Graph of Figure 4.42.	114
4.44	Bounds Window for the Graph of Figure 4.42.	114
4.45	SGP Display for the Graph of Figure 4.42.	116
4.46	TGP Display for the Graph of Figure 4.42.	117
4.47	TRE Display for the Graph of Figure 4.42.	118
4.48	Resources Window for the Graph of Figure 4.42.	118
4.49	Performance Plane Display for the Graph of Figure 4.42.	119
4.50	Throughput Display corresponding to the Graph of Figure 4.42.	120
4.51	Graph with Control Edges from Nodes 1 to 2, 7 to 8, and 10 to 8	121
4.52	Buffer Window for the Graph of Figure 4.51.	122
4.53	Bounds Window corresponding to the Graph of Figure 4.51.	122

4.54 SGP Display for the Graph of Figure 4.51.	123
4.55 TGP Display for the Graph of Figure 4.51.	124
4.56 TRE Display for the Graph of Figure 4.51.	125
4.57 Resources Window corresponding to the Graph of Figure 4.51.	125
4.58 Performance Plane Display showing Modify Table	127
4.59 Throughput Display for the Graph of Figure 4.51.	128

LIST OF SYMBOLS

SYMBOL	DESCRIPTION
ACT	Algorithm Completion Time
ADM	Advanced Development Model
AMG	Algorithm Marked Graph
AMOS	ATAMM Multicomputer Operating System
ATAMM	Algorithm To Architecture Mapping Model
C_i	i th Circuit in the CMG
CMG	Computational Marked Graph
$D[n]$	Number of data packet associated with node n
DR	Data Read
EF	Earliest Finish
ES	Earliest Start
FDT	Fire Data Time
FU	Functional Unit
G	Algorithm Marked Graph
GVSC	Generic VHSIC Spaceborne Computer
IE	Input buffer empty
IF	Input buffer full
I/O	Input/Output

LF	Latest Finish
LS	Latest Start
$M(C_i)$	Number of tokens in C_i
N	Number of nodes in the AMG
NMG	Node Marked Graph
OE	Output buffer empty
OF	Output buffer full
PC	Process Complete
p_i	Place of G
PR	Process Ready
R	Resources
R_{max}	Maximum number of Resources
R_{min}	Minimum number of Resources
SGP	Single Graph Play
s_i	Sink of Modified AMG
SRE	Single Resource Envelope
TBI	Time Between Inputs
TBIO	Time Between Input and Output
$TBIO_{LB}$	Lower bound limit of TBIO
TBO	Time Between Outputs
TBO_{LB}	Lower bound limit of TBO
TCE	Total Computing Effort

T(C_i)	Sum of transition times in C_i
TGP	Total Graph Play
t_r, t_s	Transitions
TRE	Total Resource Envelope
TMR	Triple Modular Redundancy
VHSIC	Very High Speed Integrated Circuit

CHAPTER ONE

Introduction

1.1 Research Problem

This thesis research is concerned with the development of a software design tool which aids in the performance evaluation of an algorithm implemented in an ATAMM defined data flow architecture. 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 architecture may be defined [1, 2]. The ATAMM model was developed by researchers at Old Dominion University.

1.2 Overview

Over the past few years, a number of different computer architectures have been proposed of which several computer systems have been built [3]. A few examples are the Texas Instruments Distributed Processor (USA), the Cellular Tree Machine of the University of North Carolina-Chapel Hill (USA), and the Manchester Data Flow Computer (England) [3]. The motivation for the development of these systems comes mainly from three objectives [4]. First is the desire to increase computer performance through the use of concurrency. Second is the desire to more fully exploit very large scale integration (VLSI) in the design of computers. Third is the interest in new programming methods which facilitate the mapping of algorithms onto new architectures.

These ideas suggest a decentralized computer architecture in which a number of independent computers are to work together. Unfortunately, strategies for interconnecting and programming such architectures based upon von Neumann principles have not evolved [4].

Strategies for the control of computations on decentralized computer architectures can be classified broadly as control flow, demand driven, and data driven [4]. 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 for outputs. In data driven architectures (also known as data flow computers), the availability of operands trigger the execution of operations.

The data flow architecture 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 [5]. The best strategy for executing an algorithm in these data flow computers is machine dependent [4]. However, only a few researchers have tried to develop a theoretical model for evaluating computation in a data driven architecture [6]. These models do not appear to be adequate to address the complex issues of scheduling, coordination, and communication [4].

The Algorithm-To-Architecture Mapping Model (ATAMM) is a new graph model developed by researchers at Old Dominion University [7]. The ATAMM model describes the data and control flow associated with the execution of algorithms in data flow architectures [2]. The ATAMM model consists of a set of three Petri net marked graphs

[8, 9] which incorporate general specifications of communication and processing associated with each computational event in a data flow architecture.

The problem domain addressed by the ATAMM data flow architecture consists of decision-free, large grain, complex algorithms which are assumed to be executed periodically in a multicomputer environment [4]. The granularity level of the algorithm decomposition is kept high to avoid communication bottlenecks as observed in many fine grain data flow architectures [10].

The ATAMM model specifies data and control flow dialog necessary for any data flow architecture that implements a given algorithm [4]. The ATAMM model also provides the means to investigate different algorithm decompositions without having to consider the hardware [4]. A basis for determining analytically the performance bounds for computing speed and throughput capacity is also provided by the ATAMM model [4].

The ATAMM model is being adapted to a VHSIC (Very High Speed Integrated Circuit) data flow architecture called the Advanced Development Model (ADM). The ADM system consists of four identical VHSIC 1750A processors communicating over a dual PI bus. A 1553B communication module, also connected to the PI bus, serves as a gateway for input and output data flow from an IBM PC/AT. The 1553B communication module provides to the AT the information regarding the number of available functional units. Accordingly, the AT chooses an operating point that matches the resource requirements and modifies the graph to operate at that particular operating point. An automated software tool is necessary to determine the modifications to be made to the graph to operate at various operating points for different values of resources. The

tool must also provide a means to the user to select operating points of his choice in the performance plane.

1.3 Research Objective

The objective of this thesis research is to develop a supporting software design tool for the ATAMM model. The tool is capable of predicting the time performance and resource requirements of an algorithm implemented in an ATAMM defined data flow architecture and can provide a means to select optimal operating points for the execution of the algorithm. Prediction of time performance, resource requirements, and specification of operating points manually for an algorithm are in general very time consuming and tedious processes. The ATAMM Design Tool is developed as an automated tool to make these processes simple and less time consuming for the user. Three major aspects constitute this thesis research. First is the development of required algorithms. Second is the development of code for the software tool. Third is the evaluation of the tool's capability through case studies.

Algorithms are developed for the construction of displays representing graph play and resource envelopes. Included in the displays are the options for adding test times to functional units and floats to algorithm nodes. Algorithms for generating operating points in the performance plane are also developed as part of this thesis work. The complete software tool is coded in C under the Microsoft Windows environment. As a demonstration of the application capabilities of the ATAMM Design Tool, case studies

are performed on real algorithms. These algorithms include the space surveillance algorithm and the state equation for linear, time invariant systems.

1.4 Thesis Organization

A brief description of the ATAMM model and the related performance issues are presented in Chapter Two. A performance model for the ATAMM data flow architecture is presented in Section 2.2. A detailed description of ATAMM performance measures is given in Section 2.3. The theory concerning the ATAMM performance plane and the operating point selection strategies is presented in Section 2.5. In Section 2.6, the ATAMM based operating system (AMOS) for the ADM is presented as an example implementation of ATAMM. A state diagram description of AMOS is used as a means to discuss system operation.

The development of the ATAMM Design Tool software is presented in Chapter Three. A brief description of system requirements and the programming environment is presented in Section 3.2. The development of the Graph Editor window is described in Section 3.3. The details of data generation for the tool are described in Section 3.4. The construction of the SGP and SRE windows is related in Section 3.5. The development details for the TGP and TRE windows are presented in Section 3.6. In Section 3.7, Bounds and Buffer windows are discussed. Throughput and Resource requirement windows are presented in Section 3.8. The construction details of Performance Plane and Modify Table windows is presented in Section 3.9.

The results of the two case studies performed using the ATAMM Design Tool are presented in Chapter Four. Some real algorithms are chosen for case studies to prove the practical applicability of the ATAMM Design Tool. The case studies conclude with the selection of suitable operating points for efficient operation of the system.

A summary of the research performed and topics for future research are presented in Chapter Five.

CHAPTER TWO

ATAMM Performance Model

2.1 Introduction

In this chapter, a performance model for the ATAMM data flow architecture is presented. The ATAMM model is briefly discussed in Section 2.2. The performance measures are introduced in Section 2.3. The concepts of injection control, earliest start, earliest finish, latest start, latest finish, critical path(s), TCE, graph play, and resource requirements are discussed in Section 2.4. These concepts are the crucial factors in the development of the software tool described in Chapter 3. The ATAMM performance plane and related concepts are described in Section 2.5. In Section 2.6, an example of an ATAMM implementation is presented and the capabilities of the system are described.

2.2 ATAMM Model

Over the past few years, multiprocessor and distributed-processing systems have become a subject of intensive research. The development of parallel architectures composed of identical, special purpose computing elements is of particular interest [11]. The computing elements of a distributed system must share resources and information. Therefore, there is a need to synchronize and control this sharing in order to obtain correct overall system operation [12].

The ATAMM model is a result of research by Stoughton and Mielke at ODU, in conjunction with NASA-Langley Research Center, to develop a multicomputer operating strategy for implementing large-grained, decomposed algorithms on data flow architectures. This 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 computer architecture. Second, the model identifies the data flow and control dialog required of any data flow architecture which implements the algorithm. Third, the model provides a basis for analytically calculating performance bounds for computing speed and throughput capacity [7].

The ATAMM model consists of three Petri net marked graphs called the algorithm marked graph (AMG), the node marked graph (NMG), and the computational marked graph (CMG). A Petri net is a special kind of directed graph capable of describing data and control flow of a system [12]. Petri nets serve as both a graphical and mathematical tool. Some familiarity with Petri nets [8] and marked graphs [9] is assumed in this presentation. An example marked graph is shown in Figure 2.1. Circles represent nodes (transitions) and line segments represent edges (places). The black dots on the edges represent tokens which indicate availability of data. A node is "enabled" by the presence of tokens on all incoming edges.

The AMG is a representation of a specific algorithm decomposition. Operations and operands are represented as nodes and directed edges, respectively. Data availability is represented by the presence of tokens on incoming edges. Source and sink transitions for input and output signals are represented as squares. An example illustration of an

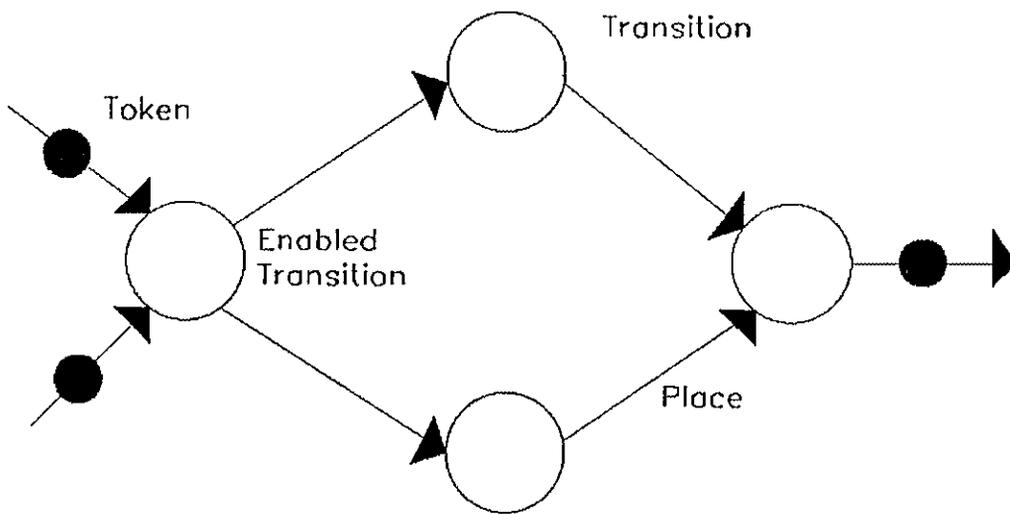


Figure 2.1. Marked Graph.

AMG is shown in Figure 2.2. The AMG, however, 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.

The NMG is a Petri net graph that represents the performance of an algorithm 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 in the NMG. Data and control flow paths are represented as places (edges), and the presence of data is shown by tokens marking appropriate edges. A read transition can be fired only if a functional unit is available in a queue of available functional units and a token is present on each incoming edge. 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. An NMG describing these activities is shown in Figure 2.3. The edge labels stand for the following:

IF Input Buffer Full

IE Input Buffer Empty

DR Data Read

PC Process Complete

PR Process Ready

OE Output Buffer Empty

OF Output Buffer Full

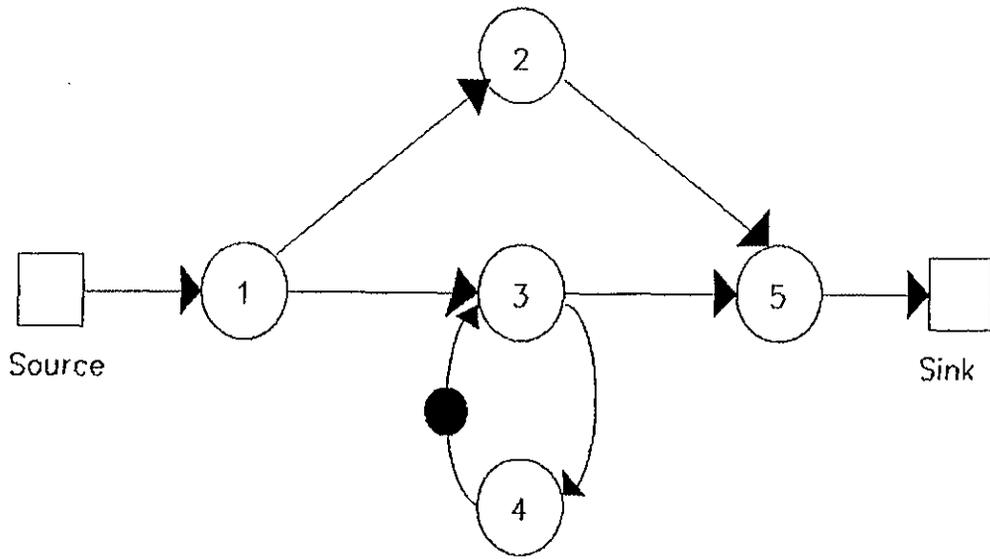


Figure 2.2. An AMG Representation.

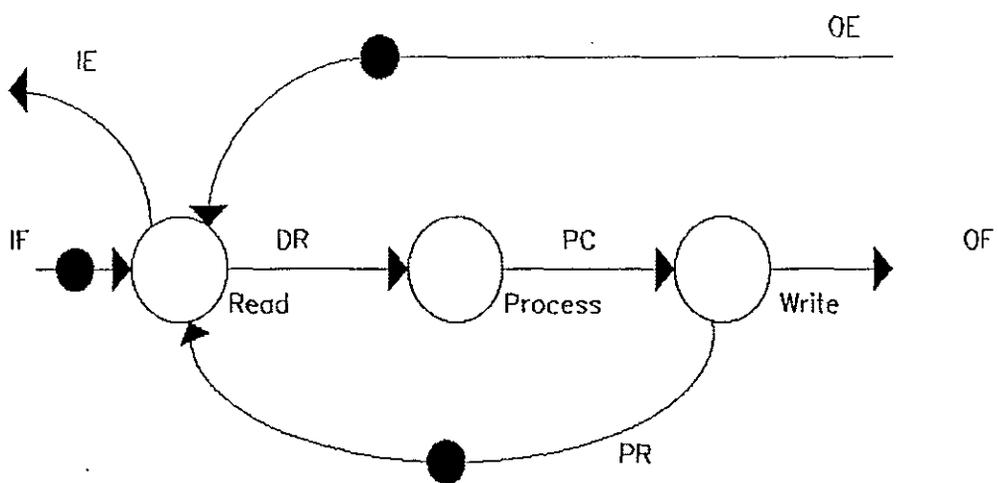


Figure 2.3. An Example of NMG.

The 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 respectively in the CMG.
2. Nodes corresponding to algorithm operations in the algorithm marked graph are represented by NMGs in the CMG.
3. Edges in the algorithm marked graph are represented by edge pairs, one forward directed edge for data flow and one backward directed edge for control flow, in the CMG.

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

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

A CMG representation of the AMG of Figure 2.2 is shown in Figure 2.4. The complete ATAMM model consists of the AMG, the NMG, and the CMG. A pictorial display of this model is shown in Figure 2.5.

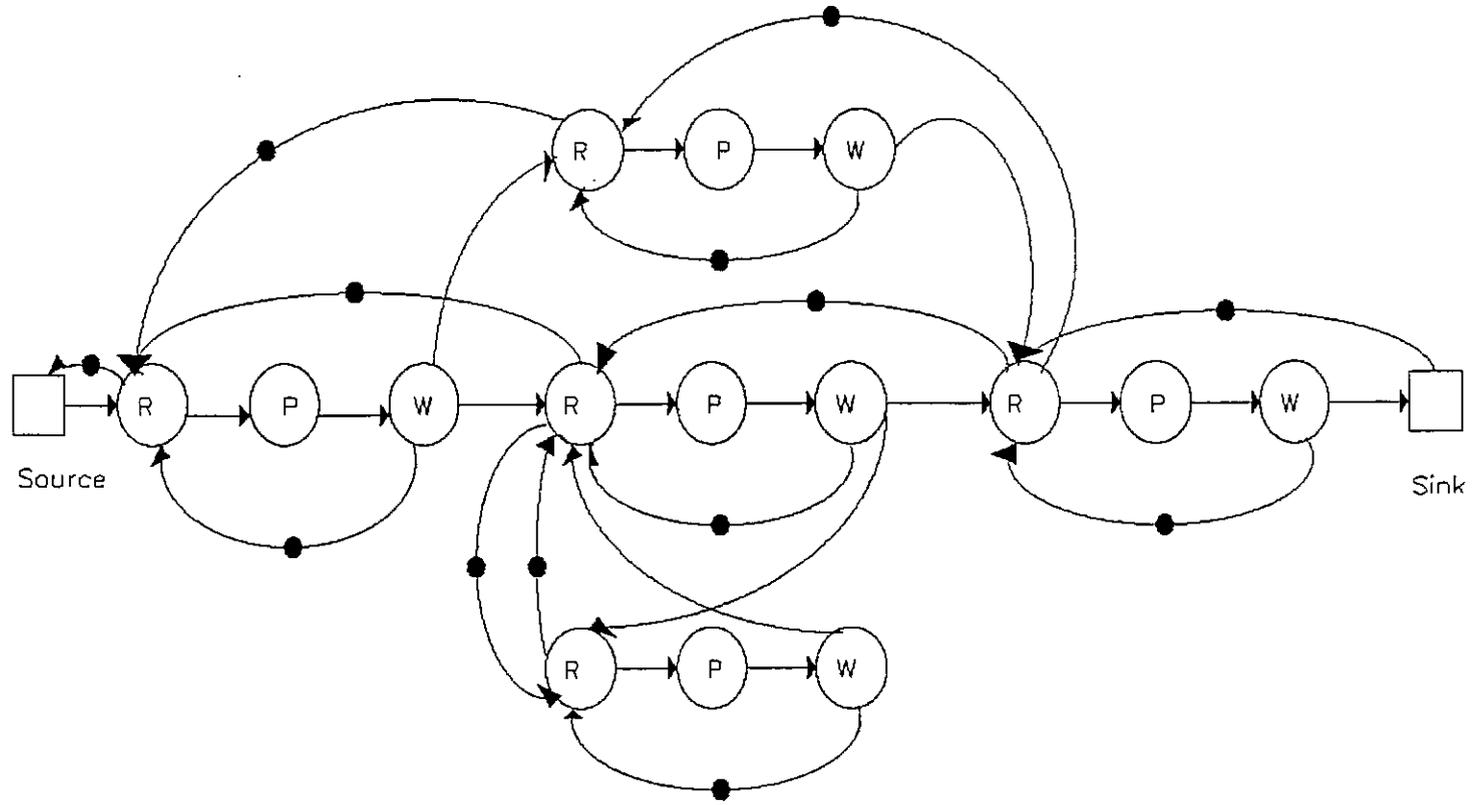


Figure 2.4. An Example CMG.

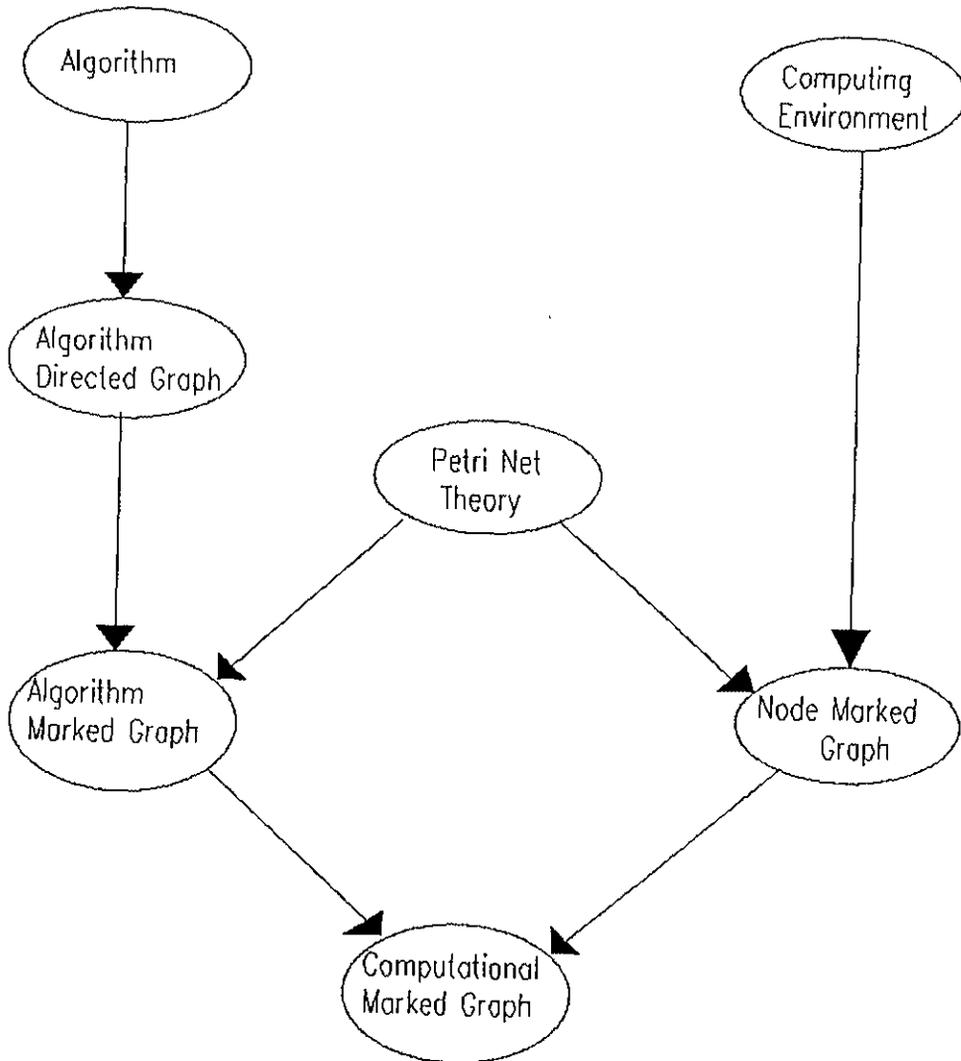


Figure 2.5. ATAMM Model Components.

The CMG of Figure 2.4 has certain note-worthy characteristics. Execution of the CMG results in live, reachable, safe, deadlock free, and consistent behavior. Liveness indicates that every transition of the graph can be fired from the initial marking [7]. Reachability implies that an output will be produced for every input. The CMG is safe because the backward control edges prevent data from being overwritten. The backward control edges prevent enablement of a transition until previous output data are picked up. The CMG is also deadlock free, because once assigned to an transition, a functional unit always is able to complete execution. Consistency implies that the CMG periodically produces outputs when inputs are applied periodically [7].

There are two types of concurrency possible during the execution of an algorithm as specified by the CMG. Transitions belonging to the same data set and which are independent of each other may be executed simultaneously. This is referred to as parallel concurrency and has a direct effect on computing speed. It is limited by the number of transitions that can be performed simultaneously for the given algorithm graph and by the number of functional units available. Also, transitions belonging to different data sets can be performed simultaneously in the computing system. This is referred to as pipeline concurrency [4]. 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.

2.3 Performance Measures

In this section, two measures of time performance, TBIO and TBO, are defined. The performance measure TBIO is the elapsed time between an algorithm input and the

corresponding output. TBIO is an indicator of computing speed. The lower bound for TBIO, denoted as $TBIO_{LB}$, is given by the sum of transition times for nodes contained in the longest directed path from input source to the output sink in the AMG. This is shown in [4]. The performance measure TBO, for the time between outputs, is the elapsed time between successive algorithm outputs when the AMG is operating periodically at steady-state. The inverse of TBO is an indicator of output per unit time or throughput. The algorithm imposed lower bound for TBO is given by the largest time per token of all directed circuits in the CMG [4]. TBO is also bounded by the number of available resources. It is shown in [4] that the resource imposed lower bound for TBO is TCE/R where TCE (total computing effort) is the sum of transition times for all nodes in the AMG and R is the number of available functional units. The lower bound for TBO, denoted as TBO_{LB} , is the greater of the algorithm bound and the resource bound.

2.4 Graph Play and Resource Requirements

In this section, a brief description of injection control is presented. Also, the parameters describing graph play, earliest start, earliest finish, latest start, and latest finish, are defined. Then, two diagrams which display graph play and are useful for determining the number of resources needed to achieve specified performance measures are defined.

Injection control is a control procedure which limits the maximum rate at which new input data packets can be injected. A data packet is an input data set. For control and signal processing applications, the algorithm is repeated periodically with new input

data sets [4]. When presented with continuously available input data packets, the natural behavior of a data flow architecture results in operation where data packets are accepted as rapidly as available resources and the input transition permit. This leads to a steady-state operating point where $TBO = TBO_{LB}$ but $TBIO > TBIO_{LB}$. This occurs because the pipeline from input to output becomes congested with extra data packets which must wait for free resources to be processed. Injection control eliminates data packet congestion and thus preserves operation at $TBIO_{LB}$. An example implementation of injection control is shown in Figure 2.6.

Each node in the AMG has four associated time parameters. These parameters are the earliest start (ES), earliest finish (EF), latest start (LS), and latest finish (LF). ES is the earliest time at which a node can be fired. EF is the earliest time at which a node execution can complete, and is equal to the sum of ES and the node time associated with the node. LS is the latest time at which a node can be fired. This parameter is equal to or greater than the parameter ES. LF is the latest execution completion time of a node and is equal to the sum of LS and the node time of that particular node [13].

An example AMG is given in Figure 2.7 along with the values of the parameters defined above. The difference between latest start and earliest start or latest finish and earliest finish is called the float of a node. Float is the greatest increment of time by which a node execution can be delayed without increasing TBIO.

In the AMG, the longest path from the input source to the output sink, measured in terms of time, is defined as the critical path. There can be more than one critical path

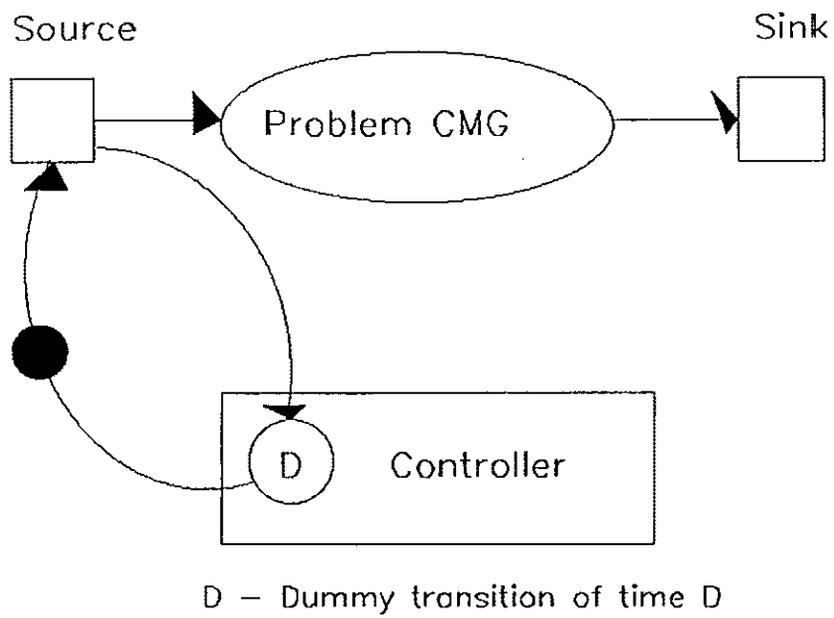
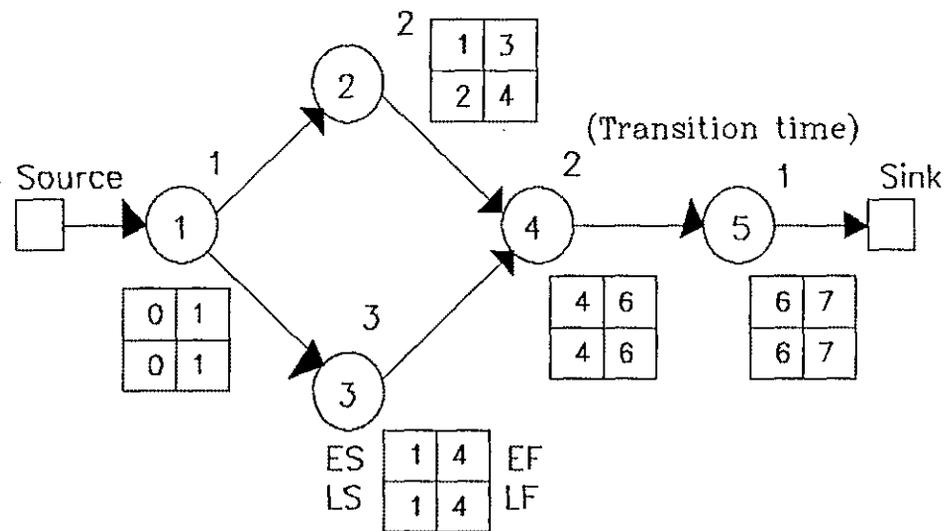


Figure 2.6. Injection Control.



ES – earliest start EF – earliest finish
 LS – latest start LF – latest finish

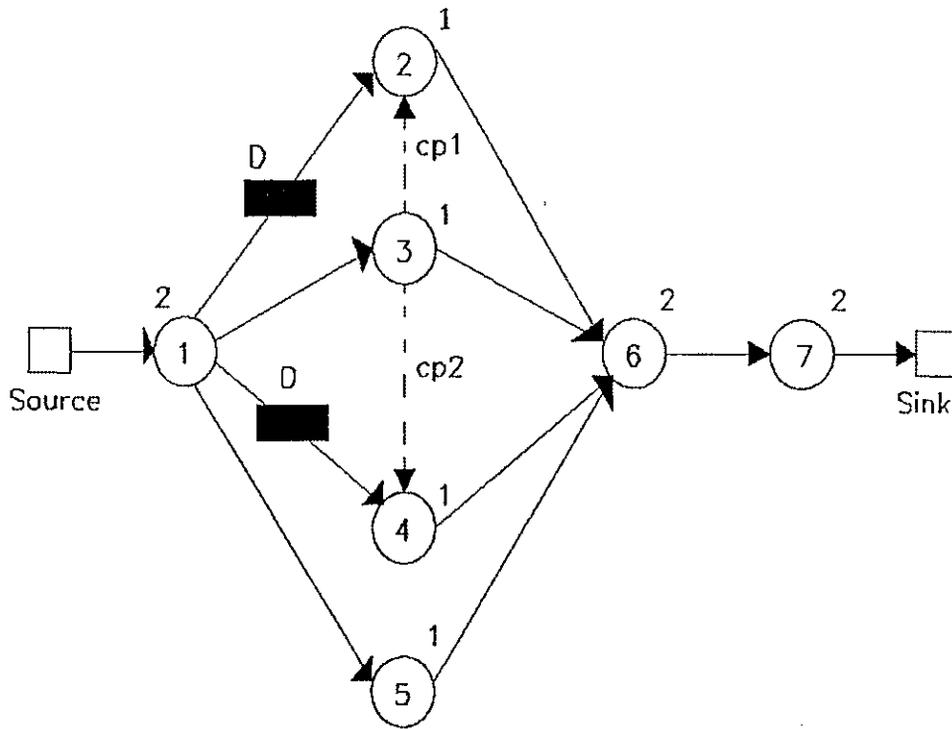
Figure 2.7. AMG Illustrating the definitions of Section 2.4.

for a given AMG. In the AMG of Figure 2.7, nodes 1, 3, 4, and 5 form a critical path. In this graph, there is no other critical path.

The single graph play (SGP) diagram is a diagram which displays the execution of each node of the AMG as function of time. The diagram is constructed for a single input data packet under the assumption that unlimited resources are available to play the graph. Node activity is denoted by a solid line and the symbols (<, >) are used to indicate the beginning and end of node execution. When several nodes are active at the same time, lines indicating node activity are stacked vertically so that computing concurrency is apparent. The SGP diagram for the AMG shown in Figure 2.8 is given in Figure 2.9. The data packets are numbered in the same sequence in which they are injected.

The number of resources required to execute a single data packet is obtained by counting the number of active nodes during each time interval in the SGP diagram. The peak resource requirement is denoted by R_{\min} and represents the minimum number of resources necessary to achieve operation at $TBIO = TBIO_{LB}$.

The total graph play (TGP) diagram displays the execution of each graph node when the graph is operating periodically in steady-state with period TBO. The TGP diagram is constructed using information from the SGP diagram. The SGP diagram is divided into segments of width TBO, and these segments are overlaid to form the TGP diagram. Each segment from the SGP diagram represents a new input data packet. Data packets are numbered sequentially so that the packet numbered $i+1$ is the data packet which is input to the graph TBO time units after the packet numbered i . The TGP diagram for the AMG of Figure 2.8 is shown in Figure 2.10.



D – Dummy Transition
 cp1, cp2 – Control Places

Figure 2.8. An Example AMG.

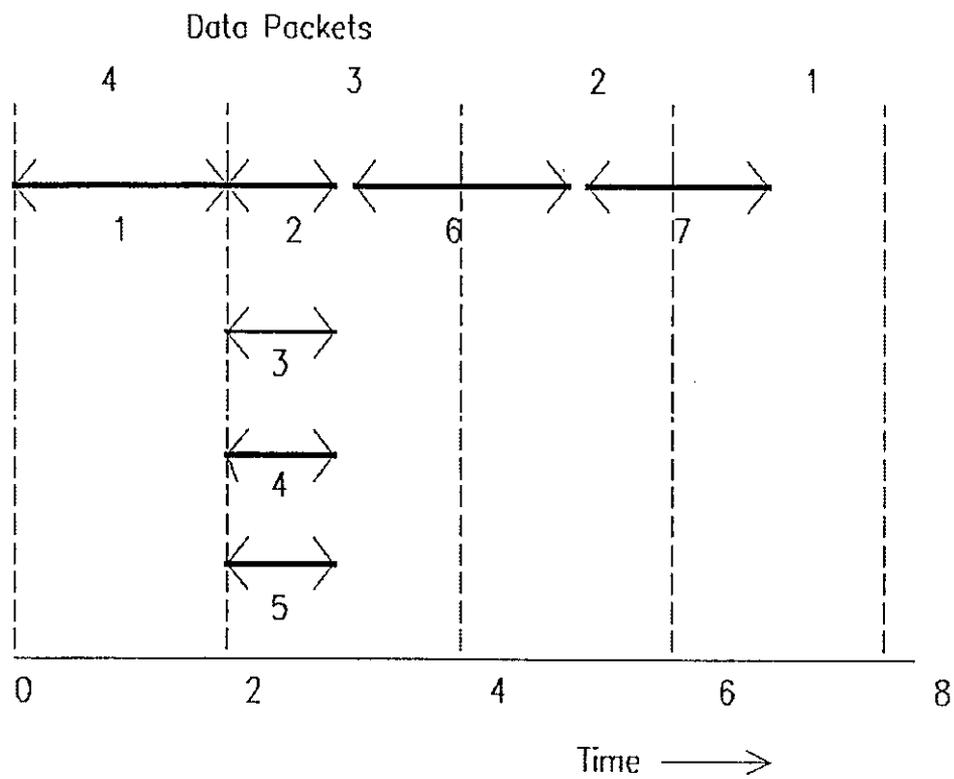


Figure 2.9. SGP Diagram for Figure 2.8.

The resource requirements to execute multiple data packets injected with period TBO are obtained by counting the number of active nodes during each time interval in the TGP diagram. The peak resource requirement R_{\max} is determined by finding the largest resource requirement in all TGP diagrams drawn for injection intervals greater than or equal to TBO. From Figure 2.10, it is evident that a minimum of 7 resources is required for $TBO_{LB} = 2$. It can be shown that if the TGP diagram is drawn for values of $TBO > 2$, the resource requirement does not exceed 7. Therefore, the peak resource requirement R_{\max} is 7.

2.5 ATAMM Performance Plane

The display of all operating points on a graph of TBO versus TBIO with R as a parameter is called the ATAMM performance plane. An example performance plane is shown in Figure 2.11.

The system exhibits the best time performance when operated at the lower bounds of TBO and TBIO. Operation of the algorithm graph at these lower bounds is achieved using input injection control. The resource requirement at this point is the value R_{\max} obtained in the TGP diagram drawn for $TBIO_{LB}$ and TBO_{LB} . Under conditions of non-availability of sufficient resources, the operating point must be shifted so that fewer resources are required. By using injection control, the operating point can be moved along the vertical line B-V. This operating strategy preserves TBIO but degrades throughput performance [14]. The operating points on the vertical line B-V are calculated

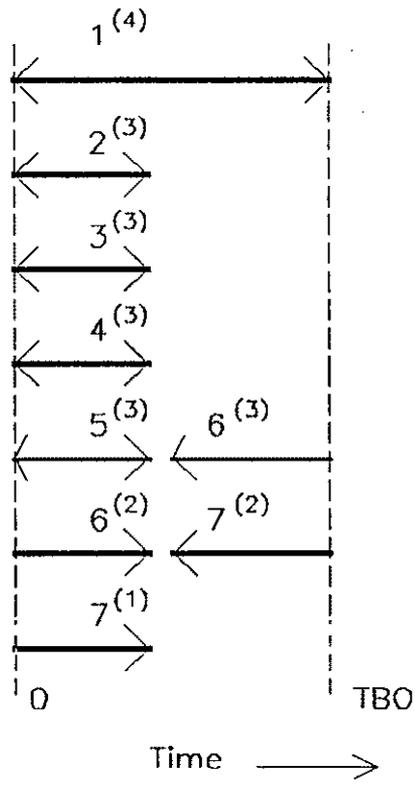


Figure 2.10. TGP Diagram for Figure 2.8.

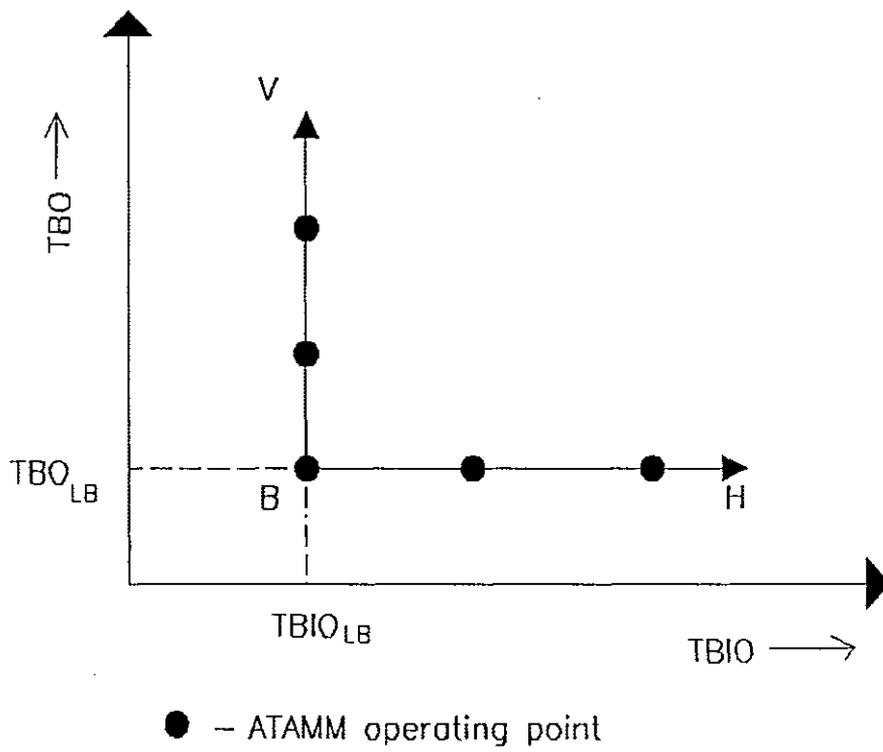


Figure 2.11. ATAMM Performance Plane.

from the TGP diagram by increasing TBO until the number of active nodes in any time interval decreases by one from the previous operating point. As an example, consider the AMG of Figure 2.8. By increasing TBO from 2 to 3, the number of required resources decreases to 5. Increasing TBO to 5 further reduces the resource requirement to 4. These points are shown in Figure 2.12.

To reduce resource requirements, the operating point also can be moved along the horizontal line B-H. This operating strategy degrades computing speed but preserves TBO [15]. This strategy is implemented by adding control edges to the original AMG. A control edge is an AMG place which imposes a precedence relation among two transitions, but does not imply data dependency [15]. When such an edge is added to an AMG, the longest path from input to output increases thus increasing TBIO. The addition of control edges can create new directed circuits having increased time per token values so that TBO is also increased. This can be avoided by increasing the number of buffers on an edge in the AMG. Every edge has an initial buffer size of one which serves as a storage for the output of a node. By increasing the number of buffers on an edge, the token count on circuits formed by adding control edges can be increased so that the value of TBO is preserved. Operating point design using control edges and buffer spaces is explained in more detail in [4]. As an illustrative example, consider Figure 2.8. Adding control edges from node 3 to node 4 and node 3 to node 2 requires that buffer size be increased between nodes 1 and 4 and nodes 1 and 2, respectively. The new operating point at $TBO = 2$ and $TBIO = 8$ for $R = 5$ is shown in Figure 2.12. Additional operating points on the $TBIO = 8$ line are obtained by using injection control.

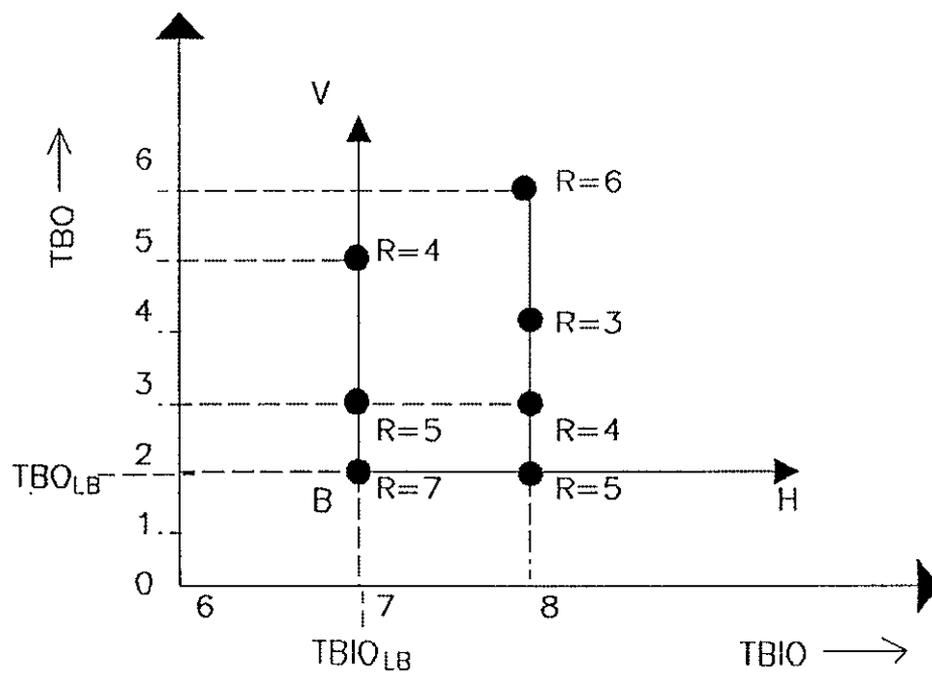


Figure 2.12. ATAMM Performance Plane for Figure 2.8.

The performance plane diagram provides information essential for the selection and control of the time performance of algorithms executing under ATAMM rules. Operating points are selected by identifying R points in the performance plane, one point corresponding to each resource number. The point associated with a specific value of R identifies the value of T_{BIO} and T_{BO} when the system is operating with R resources. If the number of resources changes, then a new operating point is identified. Operation at the new point is realized by modifying the graph with control edges and buffers, and adjusting the input injection interval.

2.6 An Example Implementation of ATAMM

In this section, an example implementation of the ATAMM is described. The host architecture is the ADM (Advanced Development Model) system. The operating system is the AMOS (ATAMM multicomputer operating system).

The ADM system consists of four VHSIC 1750A processors that are identical, communicating over a dual PI-bus. A 1553B communication module, also connected to the PI-bus, serves as a gateway for input and output data flow from an IBM PC/AT. The single line communication link between the 1553B module and the PC/AT is also used for fault injection, fault recovery, and modification of the algorithm graph in real-time [15]. The 1553B is capable of controlling the input injection rate to the 1750A processors and collecting output from the PI-bus. All the processors also communicate over an IEEE-488 bus to a Microvax computer used to download application programs and files for debugging activities. The layout of ADM system is shown in Figure 2.13.

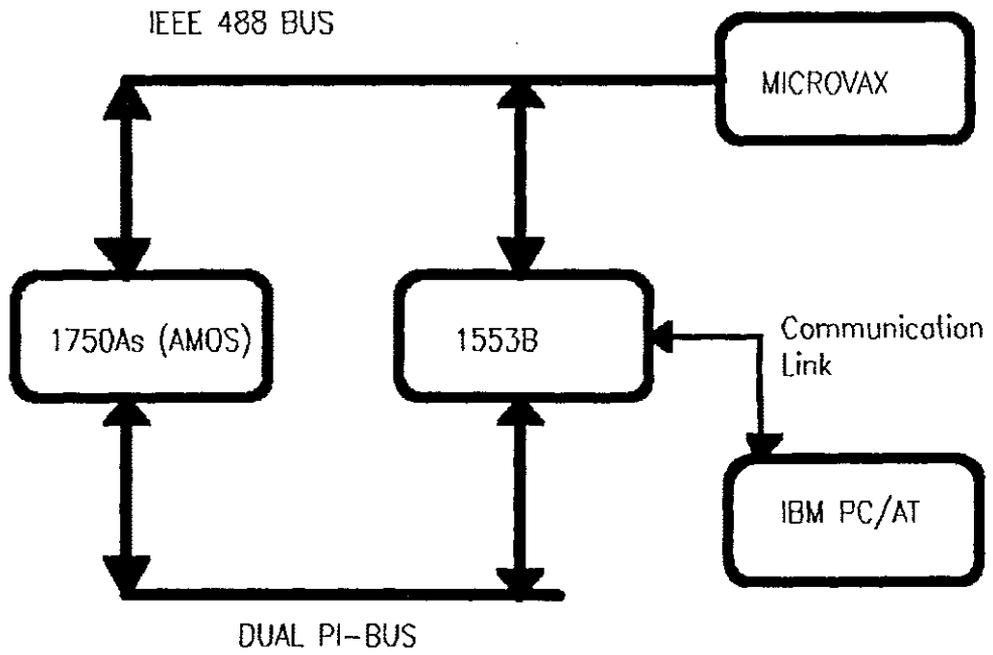


Figure 2.13. Layout of ADM System.

One operation of the AT shown in the ADM system which is important from the perspective of design tool is the following. If there is any change in the number of available resources, the information must be passed on to the AT. The reason for passing the information to the AT is that the AT is responsible for modifying the graph according to the number of available resources. In addition, all the information specifying the necessary modifications to be made to the graph, must be initiated by the AT. This information includes placement of buffers, control edges, and the corresponding injection interval. The AT is provided with this information by the design tool. The design tool has been developed to maintain a modify table for resources. The modify table contains information concerning buffer sizes, control edges, and injection interval necessary to modify the original algorithm to match the functional unit availability. When an operating point is selected in the performance plane, the corresponding modify table is made available to the AT. The modify table information is stored in the AT and is used for modification of the graph.

The AMOS consists of three logical components, the graph manager, the global memory, and a set of functional units or resources. The graph manager updates and monitors the status of the computational marked graph to assign functional units to enabled nodes (according to priority if more than one node is enabled). The global memory stores data corresponding to input and output signals for each algorithm operation of the AMG. The functional unit communicates with the graph manager to update the CMG status, and with the global memory to read and write data. A functional unit releases the PI-bus only after the updated graph data structure is broadcast to all functional units. This is to ensure that all functional units have an identical copy of the

graph data structure [15]. The graph manager and global memory are distributed among all the functional units [15].

The operation of the AMOS is described using the state diagram of Figure 2.14. Initially, a functional unit is in the Idle state and remains there until its identifier appears at the top of the resource queue. Then the functional unit enters the Examine Graph state. Here, it monitors the status of the CMG and when an enabled read node is identified, it assigns itself to perform the algorithm operation, grabs the PI-bus, and enters the next state called Execute. At this time, the functional unit identifier is removed from the top of the queue and an "F" command is broadcast on the bus to announce that an algorithm operation has been initiated. The PI-bus is then released but the functional unit remains in the Execute state until the algorithm operation is complete. At the completion of the operation, the functional unit again grabs the PI-bus and a "D" command is broadcast, along with the output data of the operation, to all the other functional units. At this time, the functional unit enters the Self Test state and releases the PI-bus. After successfully completing the self test, the functional unit returns to the Idle state. This final state transition includes broadcasting of the "R" command. The purpose of this broadcast is to notify all functional units that the particular functional unit identifier has been returned to the bottom of the resource queue. Also included in the final state transition are the operations of grabbing and releasing of the PI-bus. The CMG and resource queue in the global memory of any functional unit can be updated by F, D, or R commands from any

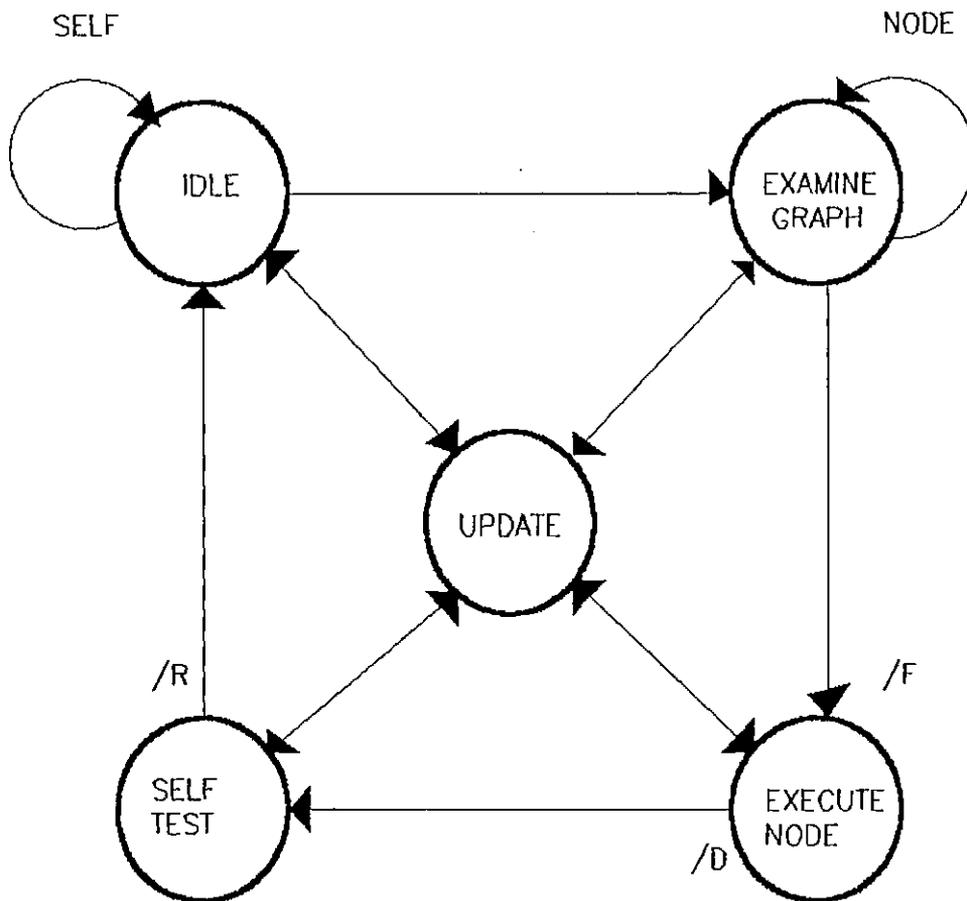


Figure 2.14. AMOS State Diagram.

other functional unit while in any state [15]. This information is shown as the fifth state in the state diagram, and is called Update.

Several operating features of AMOS make the ADM system operation reliable and efficient. The ADM is capable of implementing one algorithm graph with a single input source and a single output sink [15]. ADM is also developed with fault tolerant features. The operating system is capable of correcting a computational error in a functional unit. The approach used is the TMR (triple modular redundancy) method in which every algorithm operation is performed by three functional units and the result is selected by voting [15]. AMOS can recover from a single fault in which a functional unit detects an error in the self test state [15]. It is also possible to modify graph structure and input injection period in real-time based upon knowledge of the number of functional units. These features of fault tolerance and operating point modification can be tested by injecting faults or changing the number of functional units in real-time.

CHAPTER THREE

ATAMM Design Tool

3.1 Introduction

This chapter describes the development of the ATAMM Design Tool software. This tool aids the user in predicting the performance and resource requirements of an algorithm implemented in an ATAMM defined data flow architecture. The tool also provides the ADM system with the Modify Table consisting of injection interval, buffer size, and control edge information for each selected operating point in the Performance Plane. The entire tool is developed under the Microsoft Windows environment described in Section 3.2.

The ATAMM Design Tool consists of several component pieces called windows. Each window is constructed to solve a particular section of the total design problem. One such window is the SGP window which displays the single graph play diagram for an algorithm. There are several other windows which constitute the ATAMM Design Tool. The TGP window displays the total graph play diagram for an algorithm. The SRE and the TRE windows construct the single resource envelope diagram and the total resource envelope diagram respectively, corresponding to the SGP and the TGP displays. The Buffer window specifies the number of buffers needed on each edge in the graph. The Bounds window calculates the values of the lower bounds for performance measures, the earliest start time, the earliest finish time, the latest start time, and the latest finish time

resources required for different values of the injection interval. The Throughput window displays the throughput performance for different values of resources in the form of a bar chart. The Performance Plane window shows all operating points for a particular graph and enables the user to choose a particular set of points for operating the system.

The ATAMM Design Tool obtains the required input information from a graph drawn using the Graph Editor window. The Graph Editor is presented in Section 3.3. In Section 3.4, a description of how the data required for all of the application windows are generated from the graph information is presented. In Section 3.5, the construction details for the Bounds and Buffer windows are described. The SGP and the SRE windows are discussed in Section 3.6. The TGP and the TRE windows are presented in Section 3.7, and the Throughput window is presented in Section 3.8. Lastly, in Section 3.9, a description of the Performance Plane and the Modify Tables windows are presented.

3.2 Software Environment

The software environment used for the ATAMM Design Tool is the Microsoft Windows. The complete code is written in Microsoft C. The Software Development Kit was used to develop the code for compatibility with the Windows environment. The various built-in functions of the kit greatly reduce the complexity of the program and provide more user-friendly features.

From the viewpoint of a programmer, Windows programming is relatively easy because the overhead involved in interfacing with the mouse, the keyboard, and external

routines is eliminated. For a user, the environment is extremely friendly because none of the commands need to be remembered. All options are menu driven and minimum keyboard interaction is required. Another major advantage of the Windows environment is the feature which enables parallel execution of more than one program or more than one copy of the same program. This presents the user an opportunity to run different graphs at the same time so that comparisons can be made. All windows share global data and interact with each other dynamically for information transfer. Several dialogue boxes and message boxes are provided for the convenience of the user when running the tool. A hard copy of any of the windows can be obtained. If a color printer is not available, then a black and white printout can be obtained. All colors in the displays are mapped to different hatch patterns for printing in black and white.

3.3 Graph Editor Window

An algorithm that runs under the specifications of ATAMM is represented as a graph as discussed in Chapter 2. The Graph Editor window in the ATAMM Design Tool allows the user to create and store the graph. The Graph Editor window is one of the several application windows in the tool and is invoked from the main window.

In the Graph Editor window, several different menu options are provided for creating, editing, saving, and retrieving a graph. Under the menu called "Nodes", options for drawing source nodes, sink nodes, and operator nodes are provided. Also, an option for specifying the source node, sink node, and operator node times is provided. The default

times are zero. In order to create an item, the item is selected from the menu. If a source (sink) node is created, a small rectangle with the number of source (sink) node displayed inside appears. If an operator node is created, a circle with the node number displayed inside and the node time displayed outside appears. The numbering of each of these items increases consecutively, starting with one.

To draw the graph edges, another menu called "Edge" is provided. Under this menu, different options for creating regular edges, control edges, buffers, and tokens are available. In order to draw a regular edge or a control edge, the user selects the appropriate option from the menu. An edge can have as many as three segments. Every edge contains a small square box, and numbers appear inside and above the box. The number inside the box indicates the number of tokens and the number above the box indicates the edge buffer size. The default value for the number of tokens is zero and that for the buffer size is one. The terminal node of an edge is identified by a small, black circle. A control edge is distinguished from a regular edge by color. The control edge appears in red and the regular edge appears in black.

Another menu available in the Graph Editor window is the "Modify" menu. This menu contains options to delete any item drawn in the Editor. Each item is provided with an individual delete option.

The final menu available in the Graph Editor window is the "File" menu. This menu provides an option to save the created graph under a file name with ".gph" extension. Another option is to exit from the Graph Editor. The ".gph" extension for the graph file name is the default extension and cannot be changed. In order to load a

previously saved graph file, the user selects the "Open" option under the "Graph_Items" menu in the main window of the ATAMM Design Tool. An example Graph Editor window is shown in Figure 3.1.

Internally, the source node, the sink node and the operator node are defined in a RECTANGLE structure. When any of these items are created in the Graph Editor window, the point where the item is intended to appear on the screen is considered the center of a rectangle. The item appears within the rectangular region. When writing the graph structure in a file, the two main diagonal points of the rectangle are saved, corresponding to a particular item number. The associated times are defined as elements under item structures. The node structure has provision for specifying read, process, and write times for a node. But in the Graph Editor window, only the total time, which is the sum of read, process, and write times, appears. The source node structure has provision for write time and the sink node structure has provision for process and read times. For an edge, each edge segment is defined in a POINT structure. Both end coordinates of a segment are stored in an array defined as POINT type. The edge structure has elements "terminal" and "initial" which store the terminal point and the initial point respectively, of a particular edge. Both the initial and the terminal points of an edge are written as elements of a two dimensional array called the "connection matrix". This connection matrix is used for identifying the connected items in the graph. The size of the matrix is 20 by 20. The elements of this matrix are zeroes and ones. If an edge connects node 'm' to node 'n', then the element corresponding to row 'm' and column 'n' is a one. Otherwise, the element is a zero. The square boxes which appear on the edges are stored

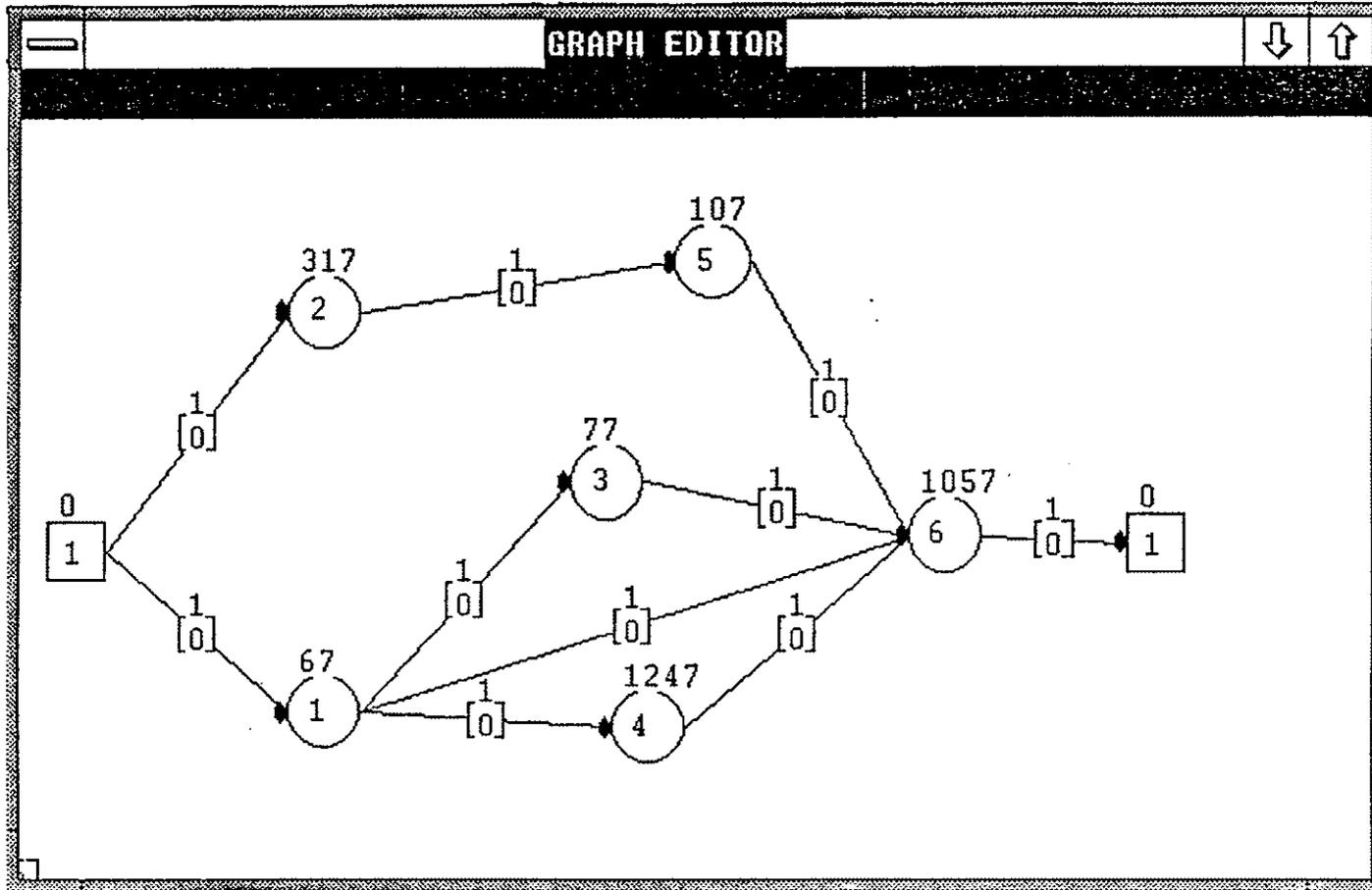


Figure 3.1. Graph Editor Window.

in rectangular structures. The small, black circle which identifies the terminal end of an edge is also defined in a rectangular structure. The number of tokens and the number of buffers associated with a particular edge are stored in arrays, defined under edge structure as "token" and "dummy", respectively. The default number of tokens is zero and the default number of buffers is one. A detailed specification for graph structure is shown in Figure 3.2. The general format in which the graph information is saved in a file is shown in Figure 3.3. Figure 3.4 shows partial graph information stored in a file for the graph of Figure 3.1.

If there are any recursions or directed circuits in the AMG, an initial token must be created on the last edge of each circuit. To achieve operation at a particular operating point in the performance plane, the initial number of buffers may have to be made greater than one. The required buffer information is displayed in the "Buffer" window of the ATAMM Design Tool and the user is directed to run the Buffer window once he is done with the creation of the graph.

3.4 Data Generation

In order to run the ATAMM Design Tool application windows, the graph information must be made available to all of the other windows. The generation of data for all windows is performed by a number of routines. Figure 3.5 shows the organization of these routines, and the interaction of all windows with the routines. When a graph file is opened, the graph information is loaded into a routine called "ASSIGN3". The graph file data is read only one time. Any subsequent changes made to the algorithm graph are

```
Struct nodess
{
    int          time,
                read,
                process,
                write,
                test;
    POINT        position;
    RECT         location;
};

Struct edges
{
    int          token,
                dummy,
                terminal,
                leader,
                first,
                last;
    RECT         location[3];
};

Struct sources
{
    int          write;
    RECT         location;
};

Struct sinks
{
    int          process,
                read;
    RECT         location;
};
```

Figure 3.2. Graph Structure.

```

Nodes
  { number
    time
    test
    read
    location } repeat for over "n"
Sources
  { number
    write
    location } repeat over "k"
Sinks
  { number
    read
    process
    location } repeat over "k"
Token
  { Edge points
    tokens
    initial
    final } repeat over "l"
Buffer
  { Edge points
    Queue size } repeat over "l"
Edge
  { Edge points
    Start.x
    Start.y
    Finish.x
    Finish.y
    Ellipse.x
    Ellipse.y
    Box.x
    Box.y } repeat over "l"
Ctrl. Edge
  { Edge points
    Ctrl. edge } repeat over "l"
Connection Matrix
  { Edge count
    Leader
    Terminal } repeat over "m"
Performance Plane
  { op.x
    op.y
    Max-R } repeat over "i"

```

Figure 3.3. Graph File Format.

```

Nodes
{
  1
  67
  0
  2
  82, 102, 61, 41 }
Sources
{
  1
  0
  14, 30, 101, 85 }
Sinks
{
  1
  0
  0
  304, 320, 101, 85 }
Token
{
  20
  0
  0
  1 }
Buffer
{
  20
  1 }
Edge
{
  20          (for edge 0 to 1)
  30
  93
  82
  51
  82
  51
  56
  72 }
Ctrl. Edge
{
  20
  0 }
Connection Matrix
{
  10
  0
  1 }
Performance Plane
{
  2371
  1247
  4 }

```

Figure 3.4. Partial Graph File for Figure 3.1.

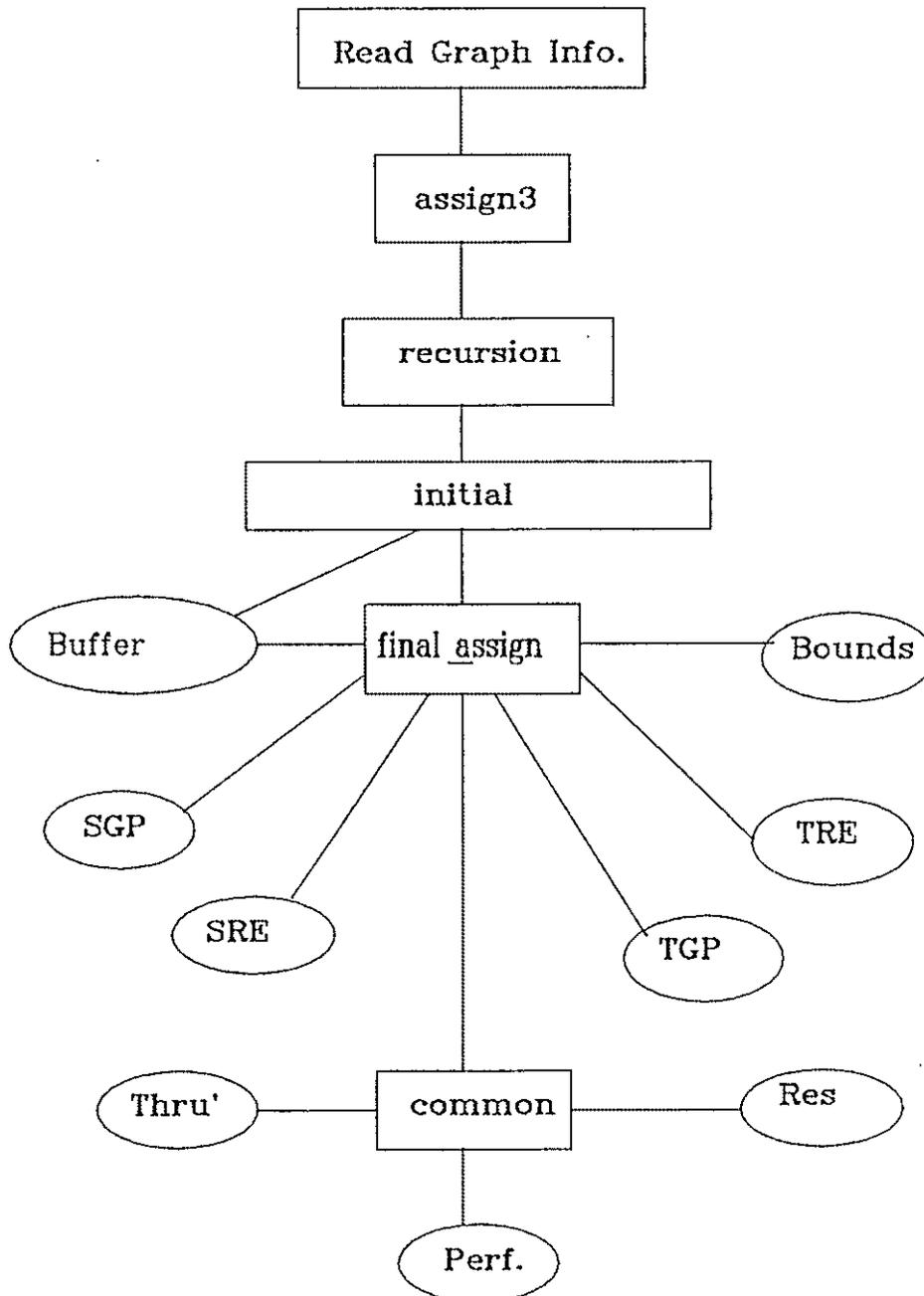


Figure 3.5. Organization of Routines.

passed directly to routine ASSIGN3. The routine ASSIGN3 is called by all the other routines which perform calculations for the application windows. These other routines are described below.

The routine "RECURSION", which calculates TBO_{LB} , calls the routine ASSIGN3 first. Initially, TBO is assigned the value of the largest node time. The routine consists of a search process for detecting directed circuits in the graph. This search is initiated by the presence of an initial token on the last edge in a directed circuit. If no initial tokens are detected, the routine is not executed further.

Let an initial token be detected on an edge k-m, having an initial node k and a terminal node m. The search starts at node m. Nodes k and m are stored in an array of circuit elements. Then, all possible paths from node m are scanned until either node k or the output sink is encountered. If node k is encountered, the search process is terminated and the path from k to m and back to k is recorded as a directed circuit. If the output sink is encountered, then the search process is continued along a different path until node k is reached again. A flag is set to identify a path that has already been traversed in the search process.

All nodes and node times encountered in the search process are saved in different arrays. The number of tokens in the circuit is also saved in an array. Once a directed circuit is identified, the routine calculates the circuit time for that particular circuit. The circuit time is the sum of all node times in the circuit. Once the above information is available, TBO is calculated as

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

where C_i is the i^{th} directed circuit, $T(C_i)$ is the sum of the node times of all nodes in C_i , and $M(C_i)$ is the number of tokens contained in C_i [4]. Finally, the value of TBO_{LB} is calculated as the maximum of all TBO values.

The RECURSION routine also generates the Modified AMG from the AMG. The generation of the Modified AMG is necessary for calculating TBIO, the latest start time, and the latest finish time.

Let G be an AMG. Let p_i be a place of G , directed from transition t_i to transition t_j , which contains an initial token. The Modified AMG is obtained from the AMG as follows.

1. Place p_i is deleted from G .
2. A new place p_{i1} , directed from the data input source to transition t_j , 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_i to s_i , are added to G .
4. The above rules are repeated for each place of G containing an initial token.

An example AMG and the corresponding Modified AMG are shown in Figure 3.6. Only place 5 directed from transition 4 to transition 2 has an initial token in the AMG. According to the first rule, place 5 is deleted. A new place 5-1 is inserted from the data input source to transition 2 by the second rule. Also, a new output sink s_5 and a new place 5-2 are generated according to the third rule. As there is no additional place with initial tokens, this completes the procedure to generate the Modified AMG. The execution of the RECURSION routine completes with the generation of the Modified AMG.

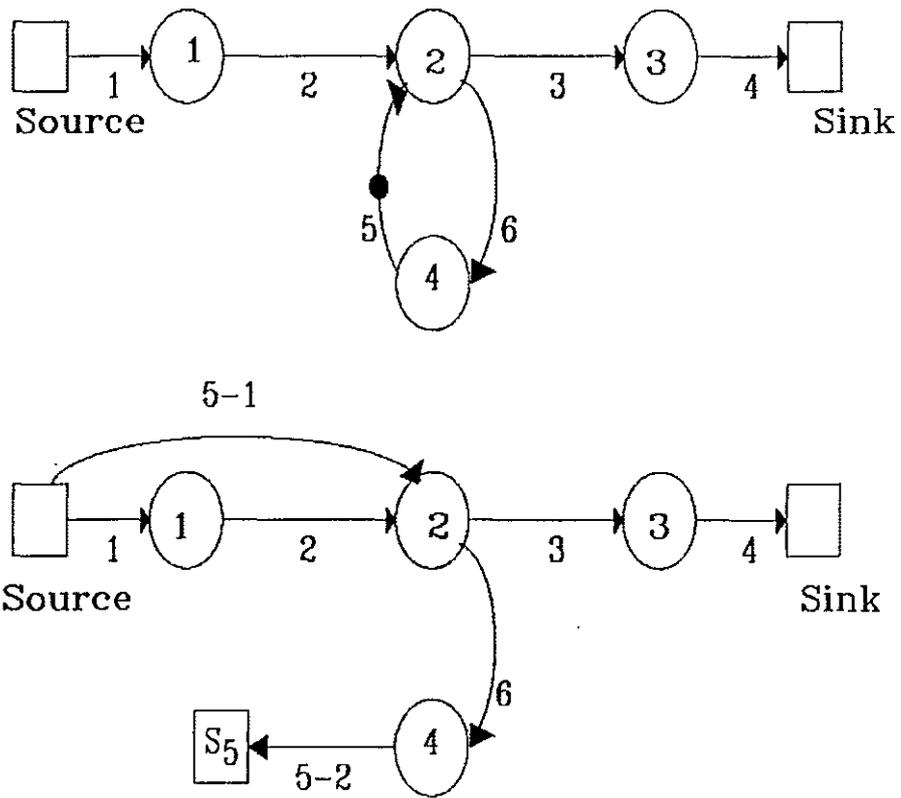


Figure 3.6. Example AMG and corresponding Modified AMG.

Another routine called as "INITIAL" takes care of all the remaining calculations including determination of TBIO, TCE, earliest start time, earliest finish time, latest start time, latest finish time, and identification of critical path(s). The INITIAL routine calls the RECURSION routine to obtain TBO_{LB} and all information pertaining to the Modified AMG.

As a first step, the INITIAL routine calculates TCE, earliest start time, earliest finish time, critical path(s), and TBIO. TCE is calculated by adding all node times in the graph. The earliest start time and earliest finish time of all nodes are calculated through a search process that identifies different directed paths in the graph starting from the input source and ending at the output sink.

The search process starts at the input source and identifies a node n_1 connected to the input source. Node n_1 is stored in an array as an element of a possible path from the input source to the output sink. Then, the next node n_2 connected to node n_1 is identified as the next element in a possible path. This process is continued till the output sink is reached. Once the output sink is encountered, the search process is repeated to identify a different path. A flag is set to identify each path traversed during the search process.

Once all the paths are detected, all the transition times in each path are added to obtain the path time. The largest of all the path times is identified as TBIO, and the corresponding path is labeled as the critical path. If a node has more than one predecessor node, then the earliest start time of that particular node is the largest of all the earliest finish times of the predecessor nodes. The earliest finish time of a node is

obtained by adding the earliest start time and the node time associated with that particular node.

As a last step, the INITIAL routine also calculates the latest start and latest finish times. The procedure involves a search process in which the paths directed from the input source to the output sink are traversed in a backward direction. The backward search process is similar to the forward search process except that it starts with the output sink and ends with the input source. For the output sink, the latest finish time is the same as the earliest finish time, because the output collected from the output sink must not be delayed. In the Modified AMG model, the latest finish time of the sink s_i is equal to the sum of earliest start time of node t_i and TBO_{LB} . Calculating the latest finish time of the sink s_i in this way ensures that the recursion is completed before a new data packet is injected into the node t_i . If a node has more than one successor, then the latest finish time of that node is equal to the smallest of all the latest start times of the successor nodes. The INITIAL routine is called only once by the Buffer window when a new graph is created, an old graph file is opened, or a graph is edited.

Once the INITIAL routine completes all calculations, it calls another routine called "FINAL_ASSIGN". The FINAL_ASSIGN routine is assigned all the results of the calculations performed in the INITIAL routine. Figure 3.5 shows that all application windows call FINAL_ASSIGN when they are run. Calling FINAL_ASSIGN avoids having to repeat all of the calculations, thus reducing execution time and saving memory.

3.5 Bounds and Buffer Windows

The Bounds window is another application window similar to the Graph Editor window and is invoked from the main window menu. The quantities earliest start time, earliest finish time, latest start time, latest finish time, TCE, TBO_{LB} , $TBIO_{LB}$, and the critical path(s) are displayed in this window. The window provides the user an opportunity to actually view the results of several computations performed internally by the routines in the ATAMM Design Tool. It also computes the start and the end times of float associated with each node. The Bounds window for the AMG of Figure 3.1 is presented in Figure 3.7.

The Buffer window is the first window that must be opened when a new graph is created, an old graph file is opened, or any changes are made to the existing graph. Initially, every edge is assumed to have a buffer space of one to hold the output data of a node. If the number of buffers on any edge is required to be greater than the default value of one, then the Buffer window displays the number of buffers required on that particular edge. Otherwise, the window displays a message saying "No Extra Buffers Required". The number of buffers on each edge is determined by the following procedure.

Consider an edge k - m having initial node k and terminal node m . Let $ES[k]$ and $ES[m]$ be the earliest start times of nodes k and m , respectively. Let $D[k]$ be the number of the data packet associated with node k and $D[m]$ be the number of the data packet associated with node m . The data packet numbers associated with each node are determined from the TGP diagram constructed for the lower bound of TBO. Also, the

BOUNDS				
NODES	ES	EF	LS	LF
1	0	67	0	67
2	0	317	890	1207
3	67	144	1237	1314
4	67	1314	67	1314
5	317	424	1207	1314
6	1314	2371	1314	2371
TCE TBIO(LB) TBO(LB)				
2872 2371 1247				
CRITICAL PATH(S)				
1 4 6				

Figure 3.7. Bounds Window.

start times for the predecessor and the successor nodes are determined from the TGP diagram constructed for the lower bound of TBO. If $ES[m]$ is less than or equal to $ES[k]$, the number of buffers on the edge directed from node k to node m is equal to the difference between $D[k]$ and $D[m]$, provided $D[k]$ is not equal to $D[m]$. Otherwise, the number of buffers is equal to the default value. On the other hand, if $ES[m]$ is greater than $ES[k]$, the number of buffers on the edge directed from node k to node m is equal to one more than the difference between $D[k]$ and $D[m]$.

The algorithm used for determining the number of buffers required on each edge is derived as follows. All edges must have at least one storage location. The data packets are numbered in the same order in which they are injected. Therefore, $D[k]$ is always greater than or equal to $D[m]$. If node k and node m are processing the same data packet, then $D[k]$ is equal to $D[m]$. This means that node k is initiated only one time $D[k]-D[m]+1$ before node m consumes a data token from the edge connecting nodes k and m . Hence only one data packet needs to be stored on the edge between nodes k and m . If $D[k]$ is greater than $D[m]$, it means that node k is initiated $D[k]-D[m]+1$ times before node m consumes a data token from the edge between nodes k and m . If $ES[k]$ is greater than or equal to $ES[m]$, it means node m has already removed data packet $D[m]$ from the edge between node k and node m by the time node k begins execution of data packet $D[k]$. Hence, if $D[k]$ is greater than $D[m]$, $D[k]-D[m]$ buffers are required on the edge between nodes k and m to provide storage for $D[k]-D[m]$ number of data packets. However, if $ES[k]$ is less than $ES[m]$, a storage location is also required on the edge

between nodes k and m for the data packet numbered $D[m]$. Therefore, the number of buffers needed is equal to $D[k]-D[m]+1$.

As an example, consider the AMG of Figure 3.8. The corresponding SGP and TGP diagrams are shown in Figure 3.9. Consider transition 1 and transition 6 in the TGP diagram. Transition 1 and transition 6 have data packet numbers 4 and 2, respectively, and have the same start time. Hence, the number of buffers required on the edge from transition 1 to transition 6 is equal to $4 - 2 = 2$. Considering the edge directed from transition 2 to transition 7, the start time of transition 2 is greater than the start time of transition 7 for data packet number 3. Hence, the number of buffers is equal to the default value of one. Similarly, no other edge requires more than one buffer.

The Buffer window for the AMG of Figure 3.1 is presented in Figure 3.10.

3.6 SGP and SRE Windows

The SGP window displays the single graph play diagram for a given graph. The total window width is defined to be 100 units greater than TBIO, and the total window height is defined to be 20 units greater than ten times the total number of transitions. Each transition is drawn on a row as a green rectangular box of height 5 units. The width of the box represents the transition time. The start time of a transition is identified by a red bar and the completion time is identified by a blue bar. A transition is drawn starting at the earliest start time and ending at the earliest finish time. The associated transition numbers are displayed on the left hand side of the window in consecutive order.

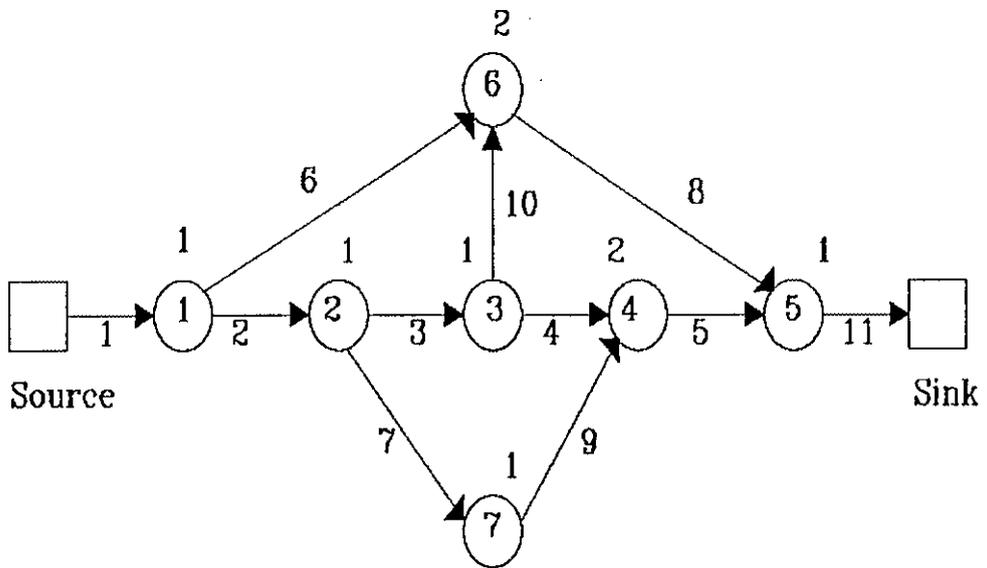


Figure 3.8. Example AMG.

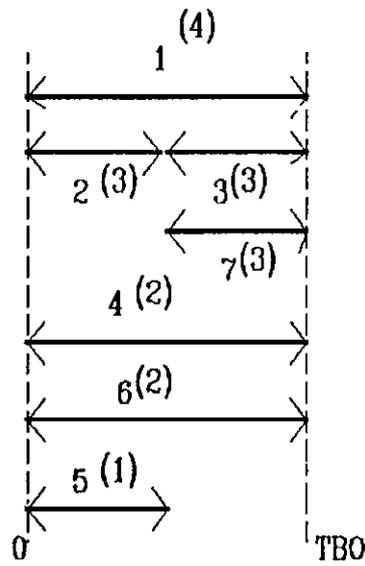
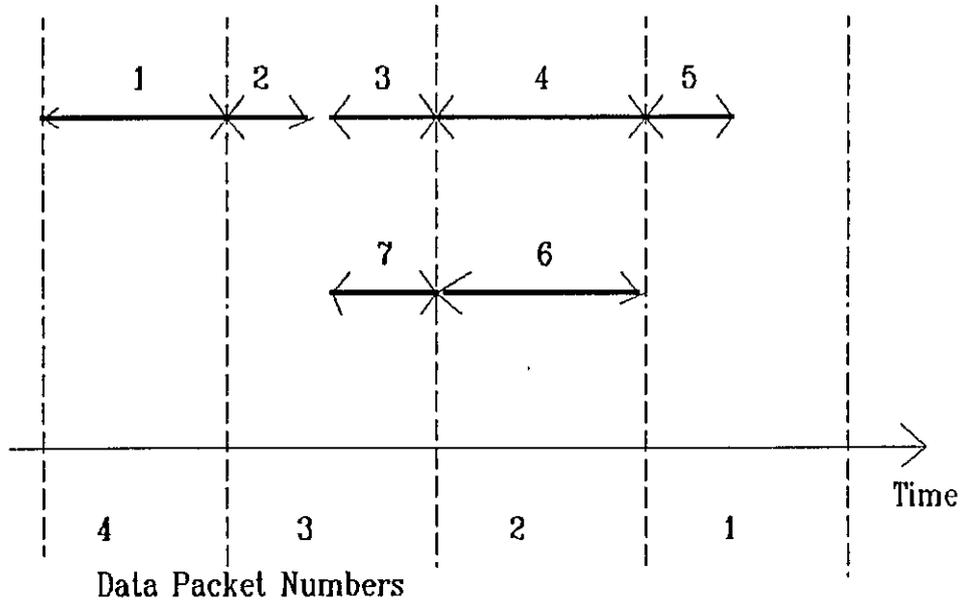


Figure 3.9. SGP and TGP Diagrams for the AMG of Figure 3.8.

The image shows a window titled "BUFFER" with a standard graphical user interface. The window contains a table with the following data:

EDGE		# BUFFERS(DEFAULT 1)
FROM	TO	
1	6	2

Figure 3.10. Buffer Window.

The float time associated with each node is displayed by choosing the float option from the SGP window menu. The float time is displayed in yellow. Similarly, the test times associated with functional units are displayed by choosing the test menu option. Test time is the time during which a functional unit remains in the self-test mode. The test times are displayed in light blue. The SGP window display for the AMG of Figure 3.1 is shown in Figure 3.11.

The SRE window displays the single graph play resource envelope for a graph. The number of resources are plotted vertically and time is plotted horizontally. Blue rectangular boxes are used to represent the number of resources utilized at any instant in the selected time interval. The SRE window uses a search procedure between the limits from zero to ACT (Algorithm completion time) to identify changes in resource requirements. A step size of $ACT/200$ units is chosen for the search procedure. The algorithm used for the search process is described as follows.

1. Initially, the number of resources, RE, is zero at the zero time marker.
2. When the time marker is incremented by the step size, the value of RE is set equal to the total number of transitions active at that time instant. The value of RE remains constant until there is a change in the number of resources.
3. A rectangle is plotted with height equal to the value of RE and width equal to the amount of time over which RE remained at that particular value.
4. The process is repeated over the range from zero to ACT.

The maximum height of the SRE diagram is the minimum resource requirement

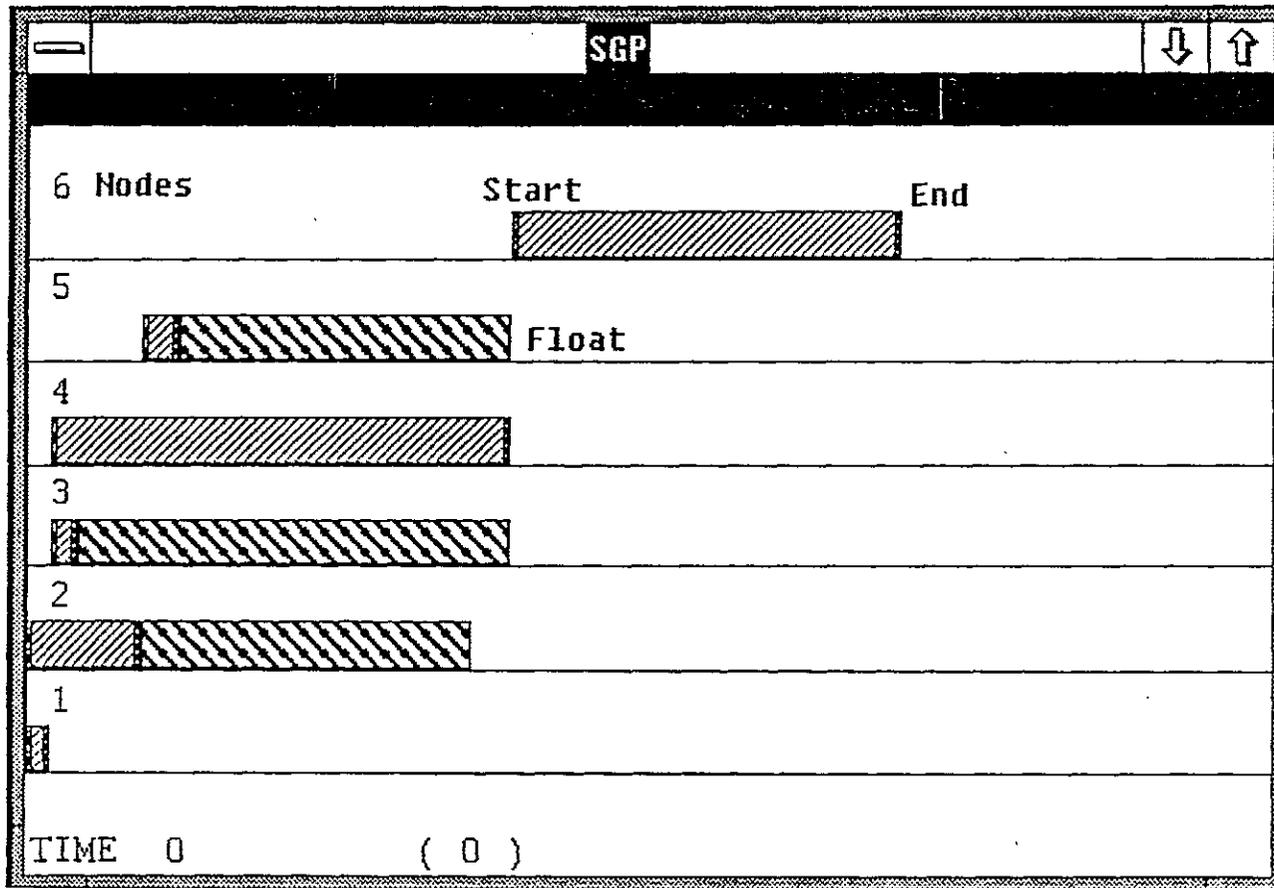


Figure 3.11. SGP Window.

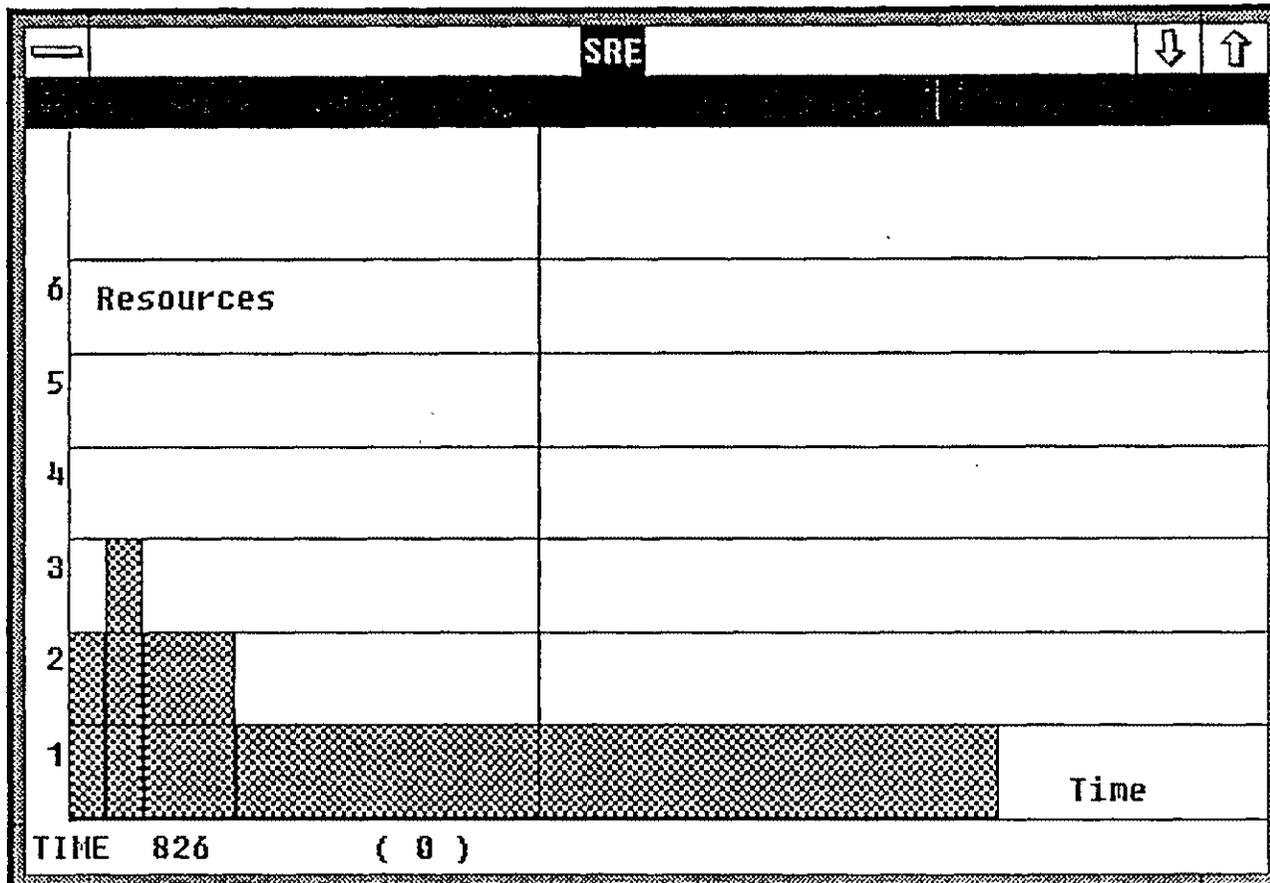


Figure 3.12. SRE Window.

for a graph. An option to include test times in the count of required resources is provided. The SRE window display for the example AMG of Figure 3.1 is presented in Figure 3.12.

The SGP and SRE windows are provided with two vertical cursers each. The left vertical cursor is invoked by clicking the left mouse button. The right vertical cursor is optional and is operated with the right mouse button. The position of the left cursor is displayed in time units at the bottom left corner of the window. If the right cursor is active, the time difference between the two cursor positions is also displayed.

3.7 TGP and TRE Windows

The TGP window displays the total graph play diagram for a graph. The window is organized similar to the SGP window. Each transition is displayed on a separate row in a sequential order. The transition numbers appear on the left side of the window. The window width is defined to be 100 units greater than TBO_{LB} . Each transition is displayed as a green rectangular box. The width of the box represents the transition time. The start time is identified by a red bar and the finish time is identified by a blue bar. The float times are displayed in yellow and the test times are displayed in light blue color. The steps involved in plotting the TGP diagram are stated below.

Let ES be the earliest start time and EF be the earliest finish time.

1. If $ES \geq TBO_{LB}$ for a node, then $ES = ES - TBO_{LB}$ and $EF = EF - TBO_{LB}$.

This step is continued until $ES < TBO_{LB}$.

2. The transition is displayed starting at ES and ending at EF. If $EF > TBO_{LB}$, then the transition is displayed from ES to TBO_{LB} as one single block. The remaining block, which extends from TBO_{LB} to EF is wrapped around and displayed starting from zero.
3. The above two steps are repeated for all the transitions in the graph.

The float times and the test times are displayed optionally as smaller rectangles, appended to the transition rectangles at the bottom. Float times and test times are also wrapped around and displayed in one period of TBO_{LB} , in case they extend beyond TBO_{LB} . An example TGP window for the AMG of Figure 3.1 is given in Figure 3.13.

The TRE window displays the total resource envelope for a graph. The display techniques used in the TRE window are exactly the same as the display techniques used in the SRE window, except that the search is carried out over the interval from zero to TBO_{LB} . The maximum height of the TRE diagram is the maximum resource requirement. The TRE window display for the AMG of Figure 3.1 is shown in Figure 3.14.

Two vertical cursers are provided in both of the TGP and the TRE windows. In addition, the TGP window is provided with a third cursor, displayed in red, which indicates TBO_{LB} in the default position. The third cursor is moved by pressing the left mouse button and the shift-key simultaneously. The current position of the third cursor defines the value of TBO, and the TGP window display is updated corresponding to that value.

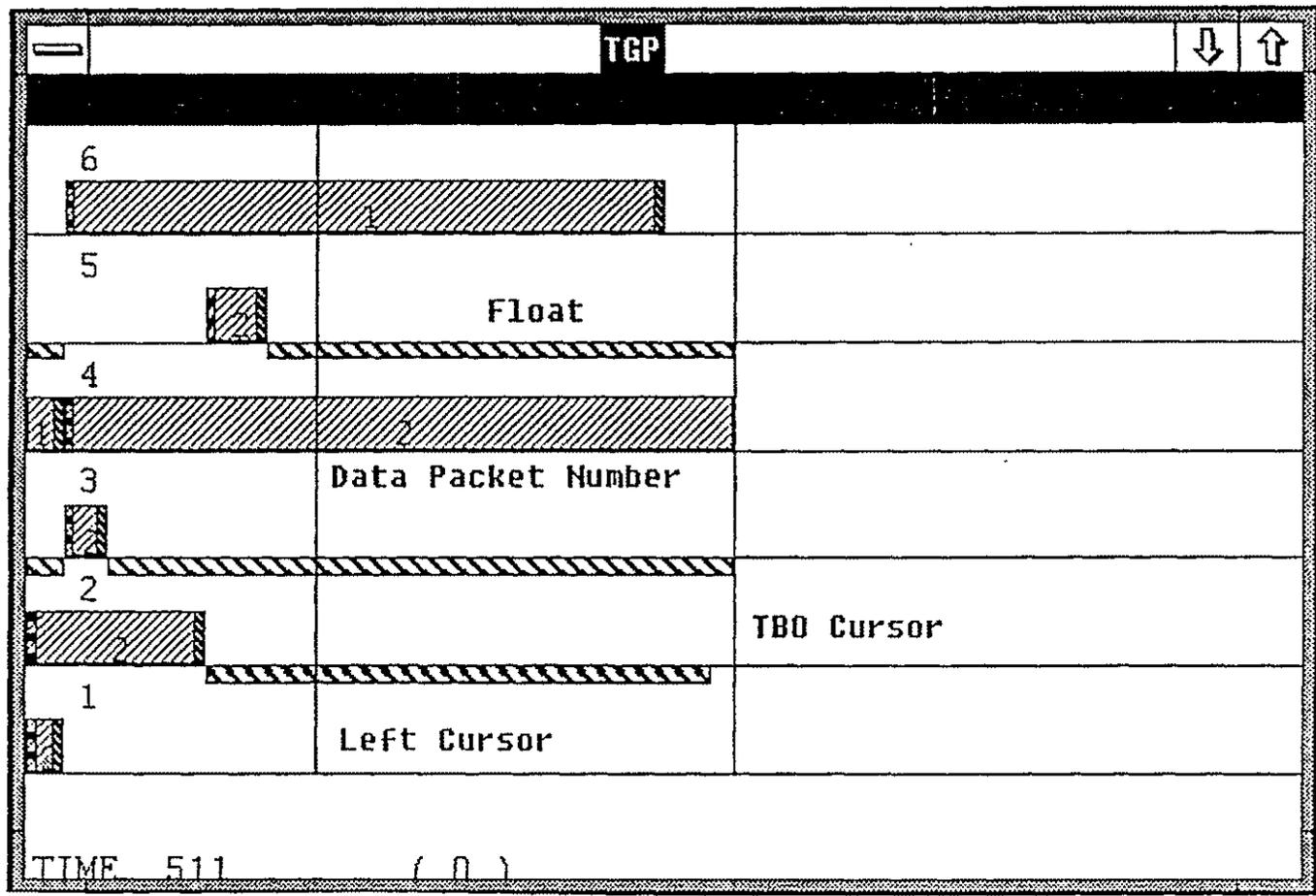


Figure 3.13. TGP Window.

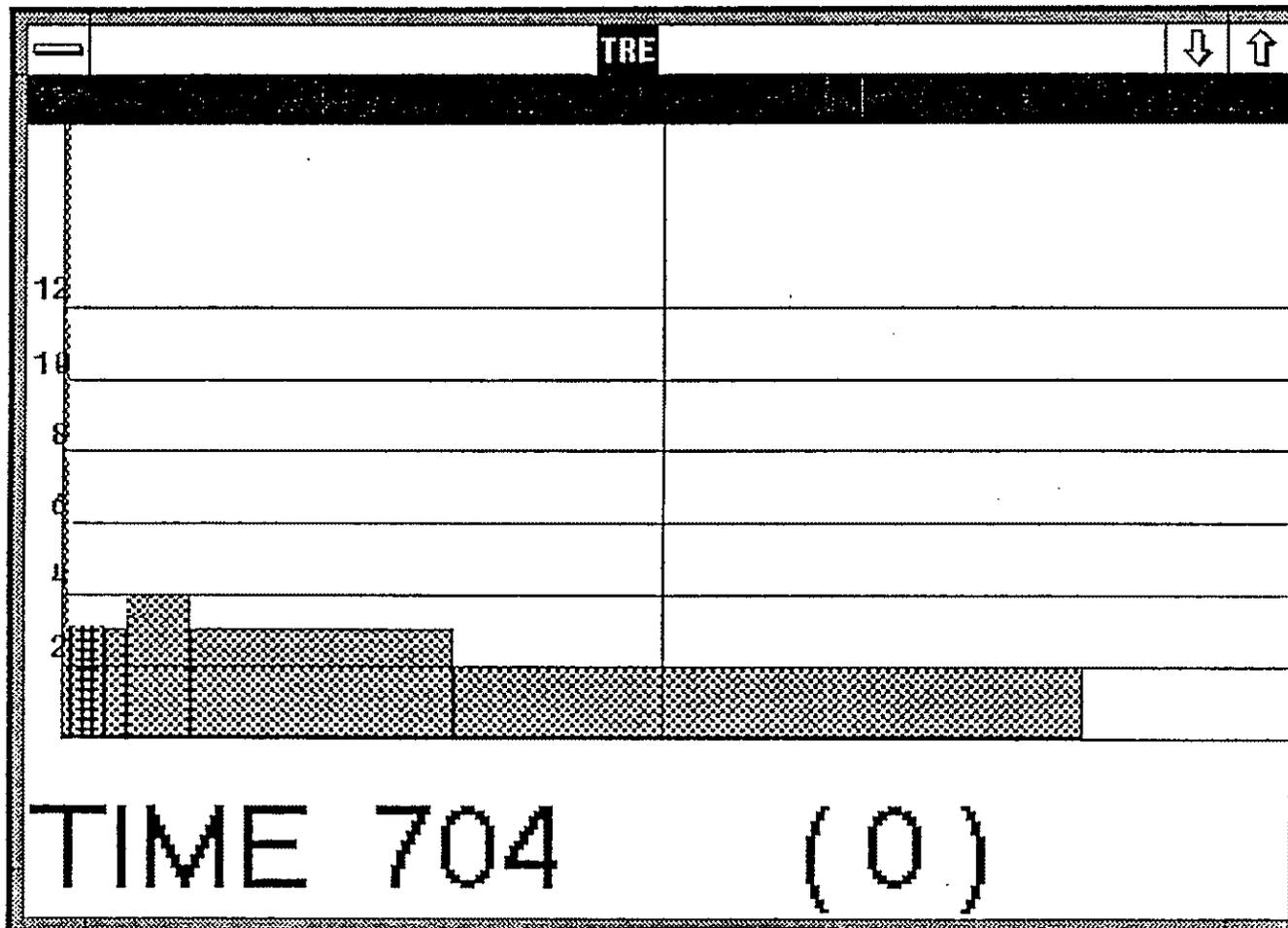


Figure 3.14. TRE Window.

3.8 Resources and Throughput Windows

The Resources window displays the number of resources required as a function of time for values of TBO ranging from TBO_{LB} to ACT (Algorithm Completion Time). If there are no recursions, then ACT equals $TBIO_{LB}$. If there are any recursions, ACT may be more than $TBIO_{LB}$ and is calculated as the largest of all the earliest finish times of the nodes. The algorithm used is described in the following.

1. The initial value of TBO is set to TBO_{LB} . The TGP and the TRE diagrams corresponding to TBO are constructed.
2. The maximum value of the resource requirement and the corresponding value of TBO are stored in separate arrays.
3. The value of TBO is then increased by a step size equal to two percent of the difference between ACT and TBO_{LB} and step one is repeated. If the latest maximum value of resource requirement is different from the previous value, step 2 is implemented. Otherwise, the value of TBO is incremented again and step 1 is repeated.
4. The above three steps are repeated until TBO equals ACT.

The four steps stated above are performed in a routine called "COMMON". The different TBO values and the corresponding resource numbers are stored in a separate routine called "ASSIGN4", which is called in the COMMON routine. The Resources window, the Throughput window and the Performance Plane window call the ASSIGN4 routine and share the information for their respective displays. All three windows have

the capability to call the COMMON routine for the first time. COMMON is called again only when there is a change in the graph.

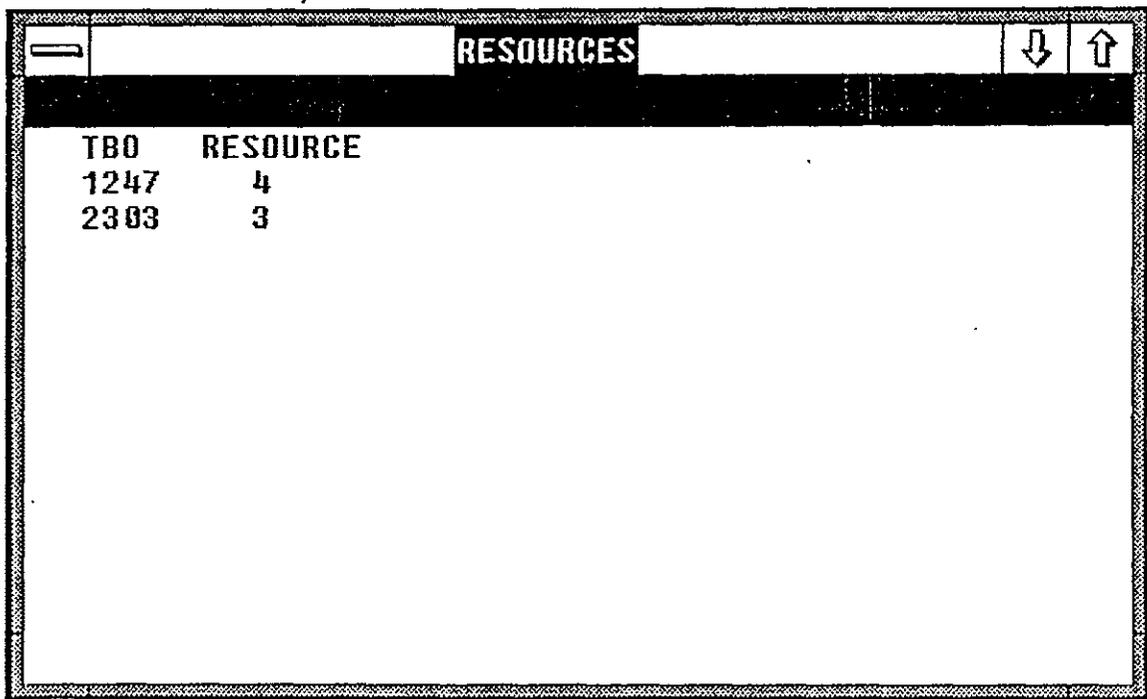
The Resources window displays the different maximum values of resources and the corresponding TBO values. Figure 3.15 shows an example Resources window for the AMG of Figure 3.1.

The Throughput window displays the normalized throughput versus the number of required resources as a bar chart. The normalized throughput is expressed as a percentage ratio of TBO_{LB} and different values of TBO obtained over the range from TBO_{LB} to TBIO. For each value of resource, the throughput is displayed as a bar or a rectangle. Two additional rectangles for higher values of resources are displayed to show that the throughput cannot increase above the maximum determined value, even if resources increase. The bars are displayed in magenta color. The Throughput window corresponding to the AMG of Figure 3.1 is shown in Figure 3.16.

3.9 Performance Plane Window

The Performance Plane window displays the operating points for different values of injection intervals. The width of the window is made slightly larger than the TCE value. The height of the window is variable and depends on the maximum value of the TBO to be displayed. Menu options are provided to select the operating points and to view the modify tables. The operating points are displayed as magenta ellipses. A cross-hair cursor is provided and the coordinates of the cursor position are displayed as TBO and TBIO.

The Performance Plane window calls the COMMON routine or uses information available in the ASSIGN4 routine to display the operating points. Also, the Performance

A screenshot of a window titled "RESOURCES". The window has a title bar with a minimize button on the left and two arrow buttons (down and up) on the right. The main content area displays a table with two columns: "TBO" and "RESOURCE".

TBO	RESOURCE
1247	4
2303	3

Figure 3.15. Resources Window.

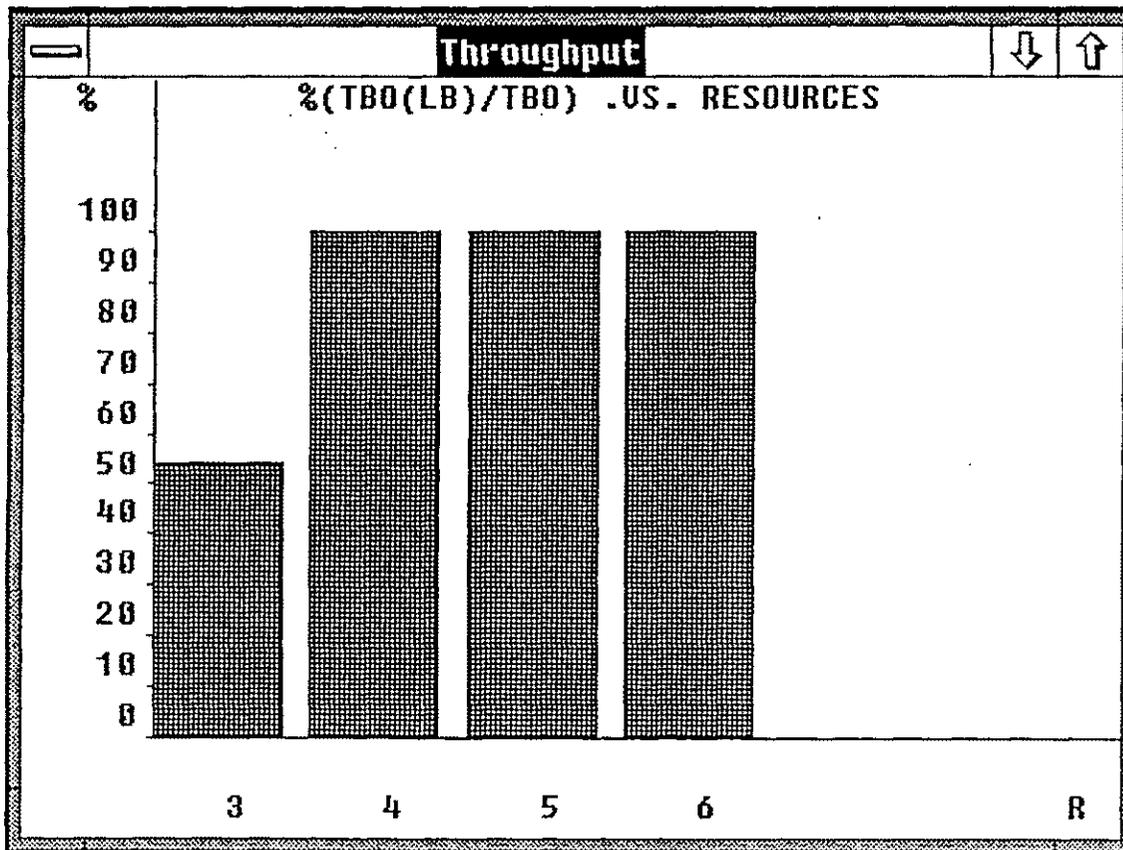


Figure 3.16. Throughput Window.

Plane calls another routine called "OP_SAVE". The OP_SAVE routine is dedicated to the Performance Plane window. If the graph is changed by insertion of a control edge, then OP_SAVE stores the old operating points in an array and appends the new operating points to them. It also saves the old values of TBO and TBIO. The ASSIGN4 routine is called only once by the Performance Plane window. After the first call, only the OP_SAVE routine is called.

The operating points are saved in a POINT structure. The point in turn is defined as the center of a rectangle. The coordinates of the point are the values of TBO and TBIO. When an operating point is selected, the color of the operating point is changed to yellow.

The Modify Table window display appears as a child window of the Performance Plane window display, when selected from the menu. The Modify Table display shows the values of TBIO, TBO, number of resources, control edge locations and the buffer size information corresponding to the operating points selected in the Performance Plane window display.

A display of the Performance Plane for the AMG of Figure 3.1 is presented in Figure 3.17.

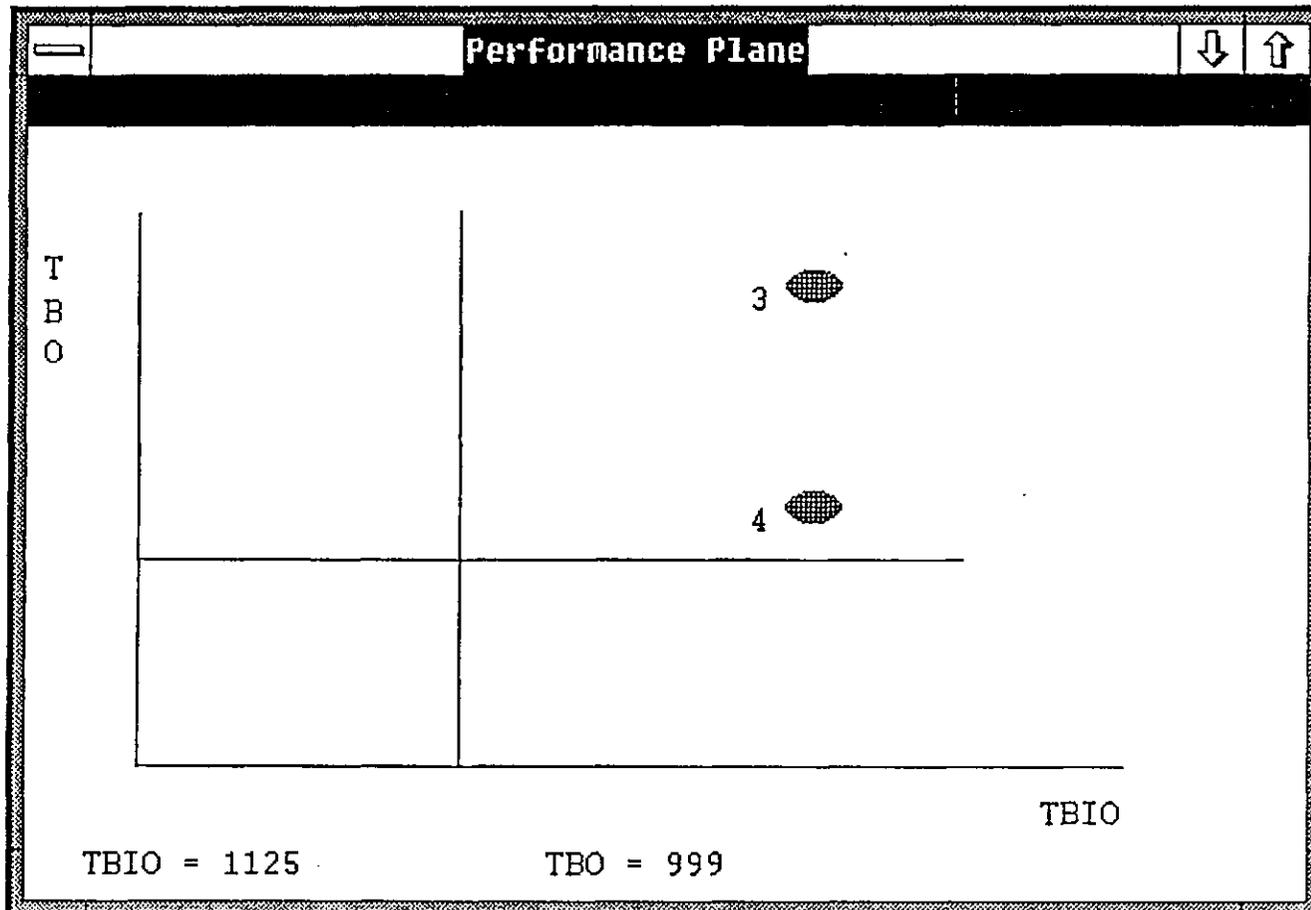


Figure 3.17. Performance Plane Window.

CHAPTER FOUR

Case Studies

4.1 Introduction

In this chapter, results of two case studies are presented as a demonstration of the application capabilities of the ATAMM Design Tool. These case studies are conducted and presented in a manner that typically would take the user of the ATAMM Design Tool through the procedural steps for generating a Performance Plane with several operating points and the corresponding Modify Table. The first algorithm chosen for the case study is the space surveillance algorithm. This algorithm is to be run on the ADM system as the primary demonstration algorithm to illustrate system capabilities. The second algorithm is the decomposed state equation for discrete linear systems. The criteria for choosing this algorithm are its real world applicability and the presence of recursion circuits. The capability of the ATAMM Design Tool to handle recursion circuits is tested through this algorithm.

In Section 4.2, an outline description of the design procedure using the ATAMM Design Tool is presented. In Section 4.3, the case study results for the space surveillance algorithm are presented. In Section 4.4, the decomposed state equation algorithm is discussed. Numerous window display printouts have been included throughout the presentation of the two case study results.

4.2 Design Procedure

The first step in using the ATAMM Design Tool is the creation of the algorithm graph in the Graph Editor window. Once the graph is created, the initial buffer requirements are observed by running the Buffer window. The earliest start time, the earliest finish time, the latest start time, the latest finish time, critical path(s), TCE, and the lower bounds for TBO and TBIO are obtained from the Bounds window. The graph play displays show the float times and test times. The minimum and maximum resource requirements to run the algorithm at the lower bounds of TBO and TBIO are determined from the SRE and the TRE display windows, respectively. The Resources window presents the variations in the resource requirements for different values of TBO over the range of TBO_{LB} to ACT (Algorithm Completion Time). The operating points are selected in the Performance Plane window. The corresponding Modify Table window presents the control edge information, the buffer size information and the details of the selected operating points. The percentage throughput is determined from the Throughput window.

4.3 Space Surveillance Algorithm

The space surveillance algorithm is presented in the Graph Editor window display of Figure 4.1. Node labels describe algorithm operations, and time units required for each operation are shown above the nodes. Signals which are transferred from one node to another node are shown as directed edges. The nodes have a read time of 2 units and a write time of 5 units. The self test time for functional units has been assigned as 5 units. The read, process, and write times appear as total node time on the graph.

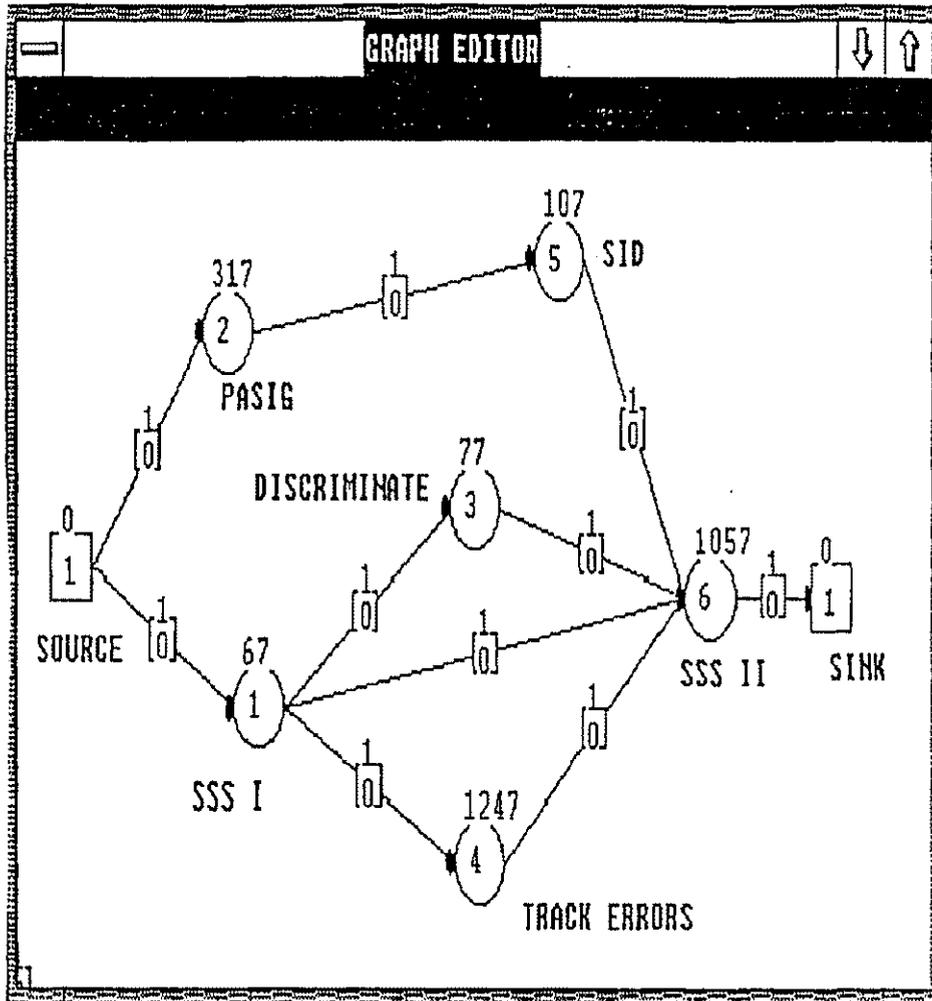


Figure 4.1. Space Surveillance Algorithm.

The Buffer window display for the space surveillance algorithm is presented in Figure 4.2. It shows that two buffers are needed on the edge from node 1 to node 2. The Bounds window display shown in Figure 4.3 displays the earliest start time, earliest finish time, latest start time, latest finish time, TCE, lower bounds for TBIO and TBO, and the critical path. The value of TBO_{LB} is 1247 units which corresponds to the node time of node 4. The critical path is the path containing nodes 1, 4, and 6.

Figure 4.4 shows the SGP display window for the graph of Figure 4.1. Nodes 2, 3, and 5 have float times associated with them. The SGP display window showing these float times is presented in Figure 4.5. In Figure 4.6, a section of the SGP display window is expanded to show the test times appended to the node times of nodes 2, 3, 4, and 5. The time displayed at the bottom indicates the cursor position as shown in the figure. The SRE window corresponding to the SGP diagram of Figure 4.4 is shown in Figure 4.7. The peak value of the resource envelope as displayed in this figure is 3. This is the minimum number of resources required for running the algorithm with TBIO equal to $TBIO_{LB}$.

The TGP display window for the graph of Figure 4.1 is presented in Figure 4.8. The rightmost cursor is the marker for TBO_{LB} , which in this case is equal to 1247 units. Figures 4.9 and 4.10 show respectively the TGP display windows with float times and test times. From Figure 4.9, it is observed that node 3 of data packet 2 begins execution after completion of node 1 of data packet 2. Also, after node 4 of data packet 1 is completed, node 4 of data packet 2 is started. The TRE display window for the TGP display of Figure 4.8 is displayed in Figure 4.11. From this figure, it is evident that the maximum

EDGE		
FROM	TO	BUFFER SIZE(DEFAULT 1)
1	6	2

Figure 4.2. Buffer Window for the Graph of Figure 4.1.

NODES	ES	EF	LS	LF
1	0	67	0	67
2	0	317	890	1207
3	67	144	1237	1314
4	67	1314	67	1314
5	317	424	1207	1314
6	1314	2371	1314	2371

TCE	TBD(LB)	TBD(LB)
2872	2371	1247

CRITICAL PATH(S)

1 4 6

Figure 4.3. Bounds Window for the Graph of Figure 4.1.

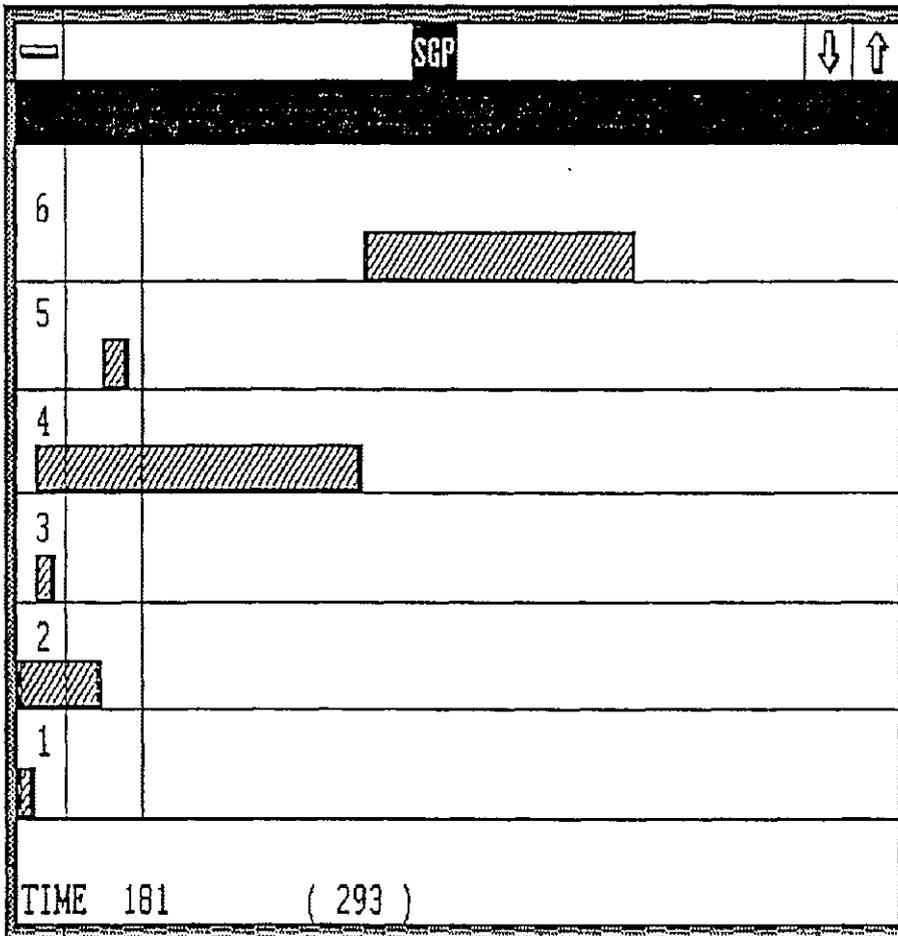


Figure 4.4. SGP Display for the Graph of Figure 4.1.

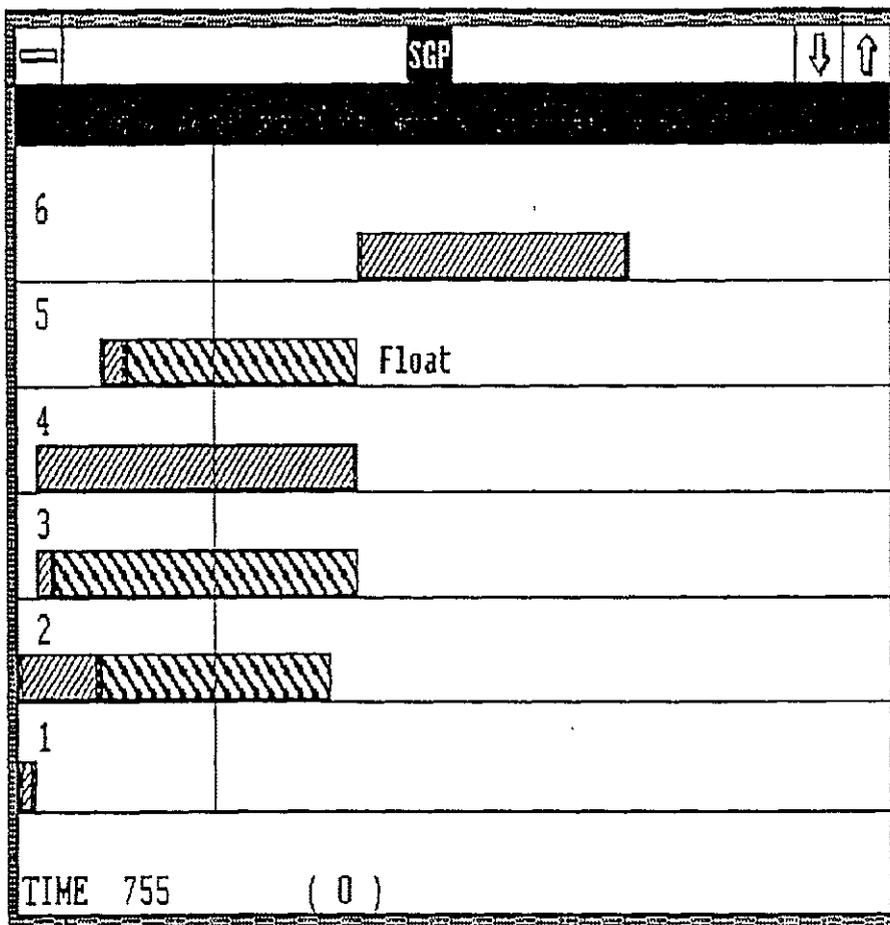


Figure 4.5. SGP Display showing Floats.

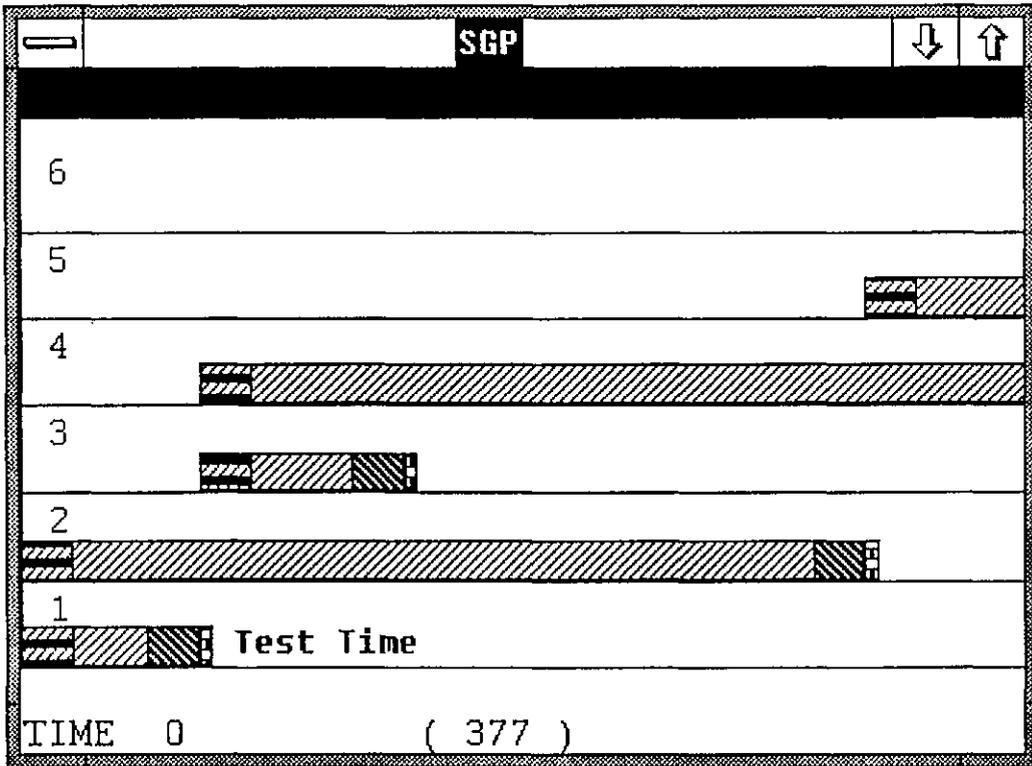


Figure 4.6. Sliced View of the SGP Display showing Test Times.

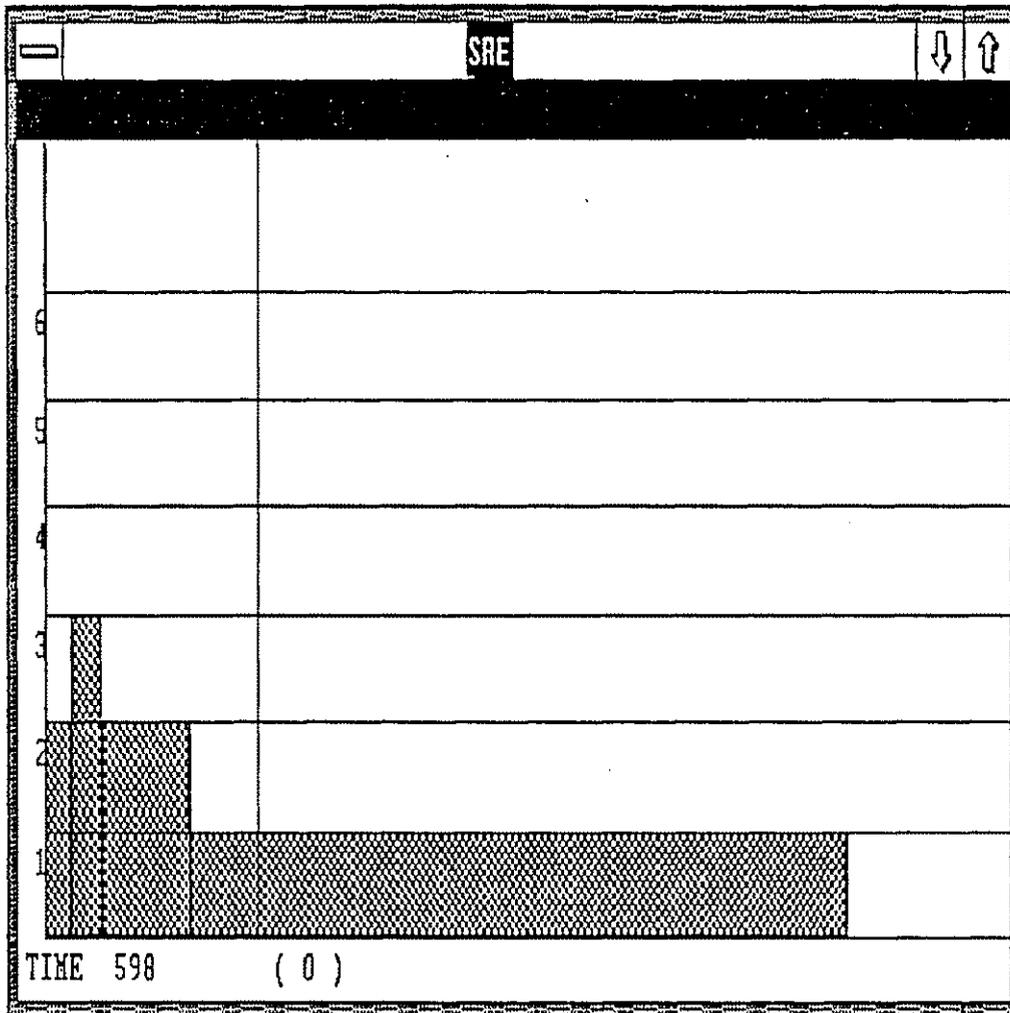


Figure 4.7. SRE Display corresponding to the SGP Display of Figure 4.4.

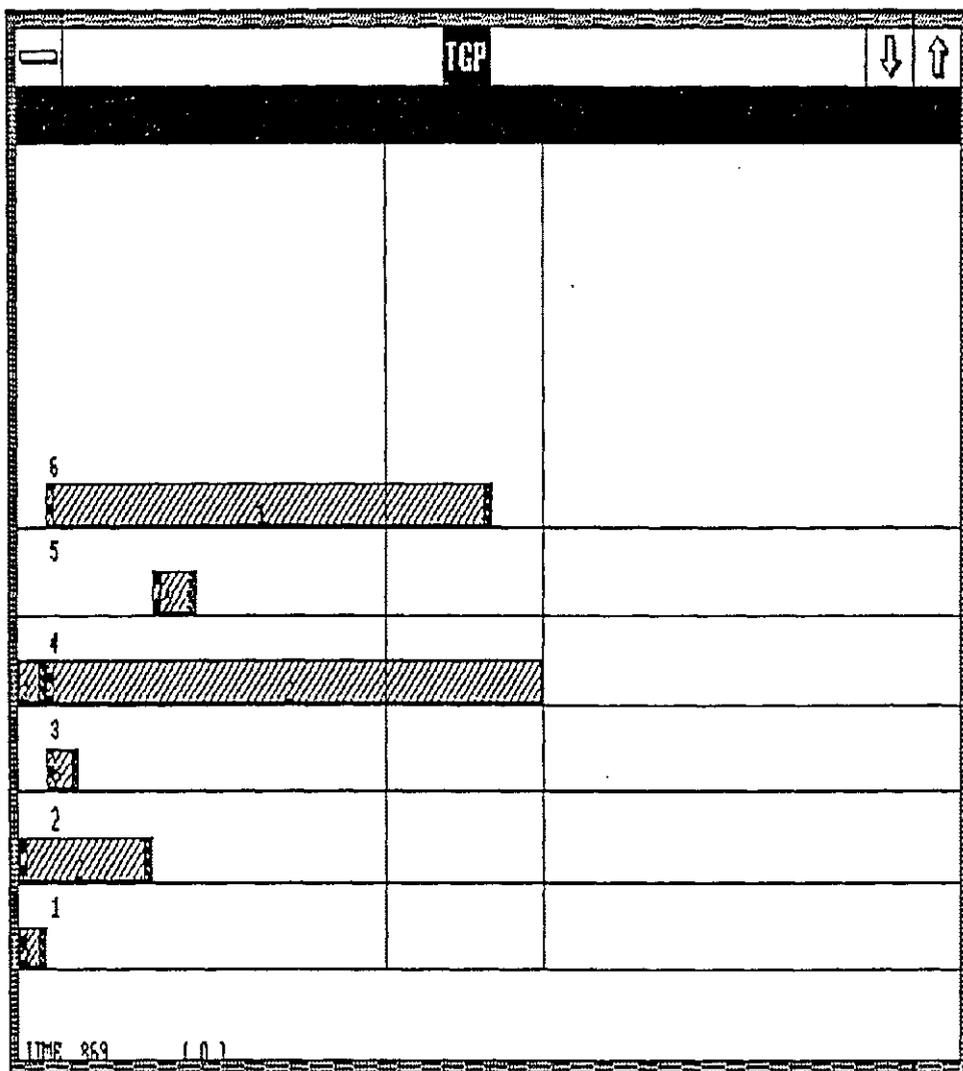


Figure 4.8. TGP Display for the Graph of Figure 4.1.

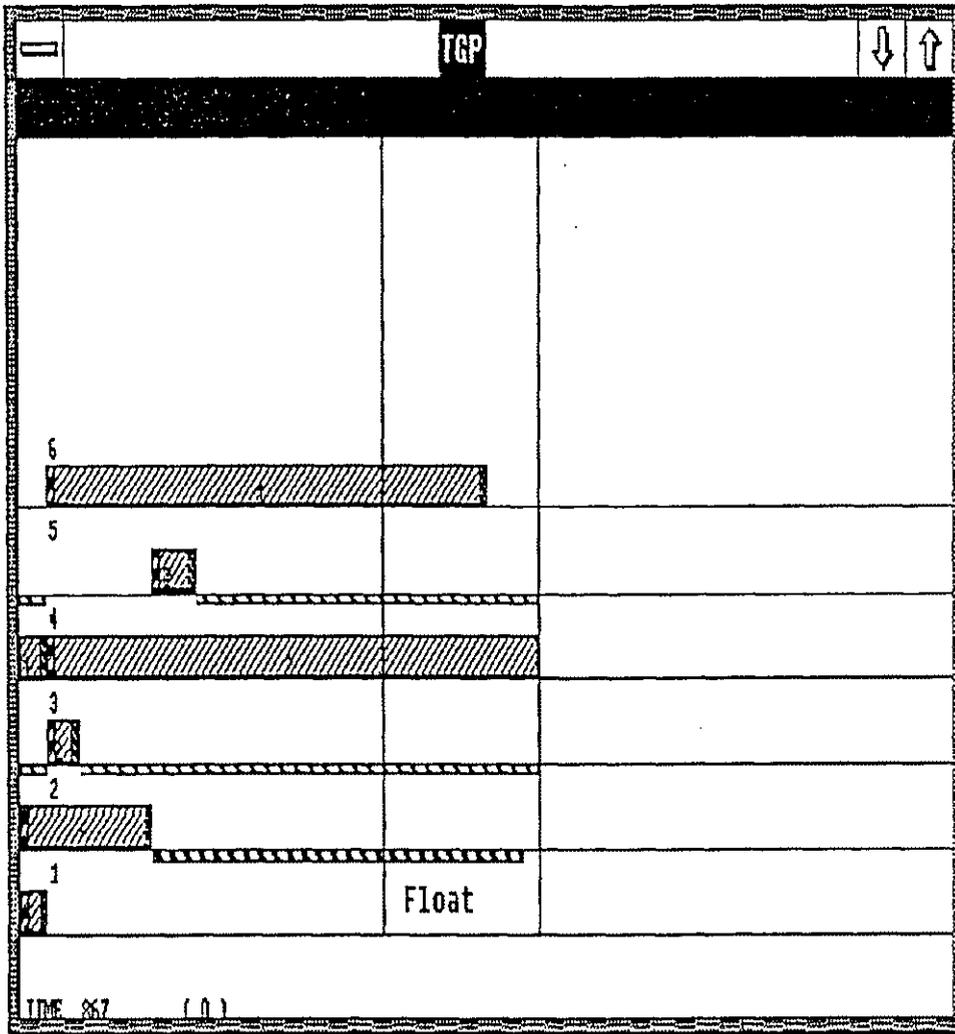


Figure 4.9. TGP Display showing Floats.

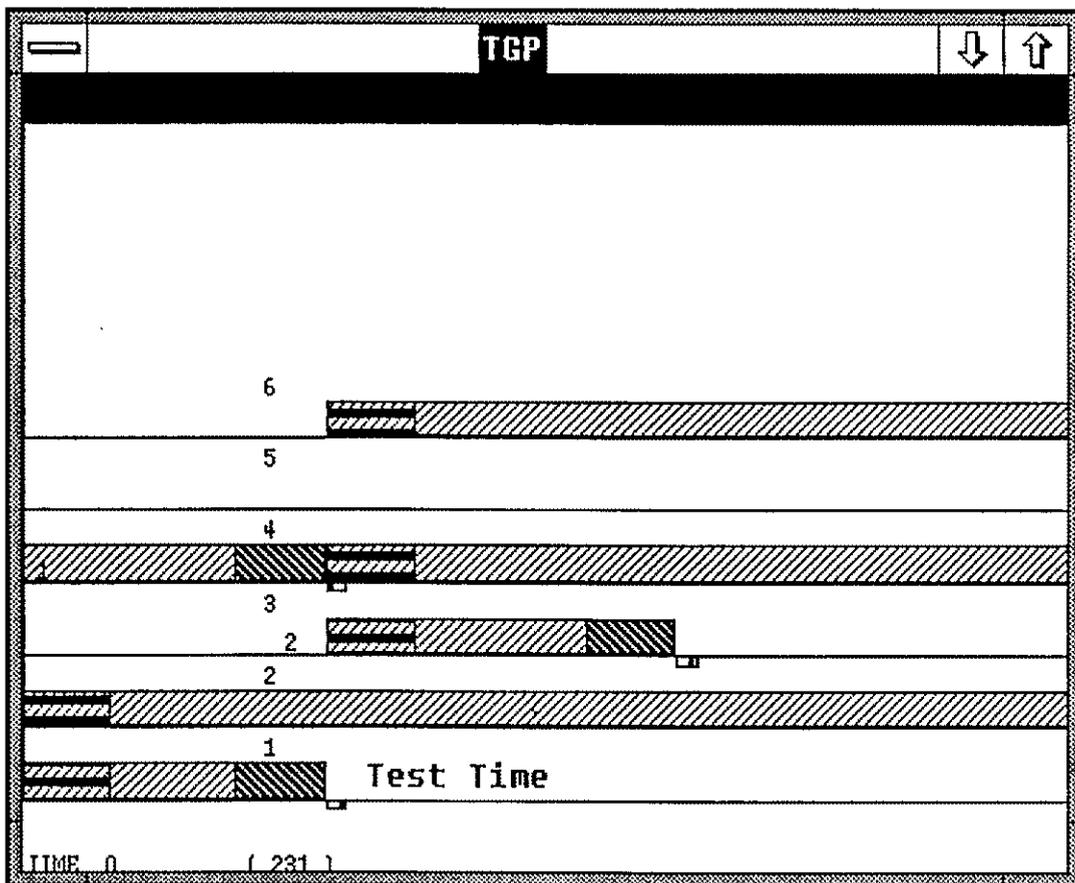


Figure 4.10. Sliced View of the TGP Display showing Test Times.

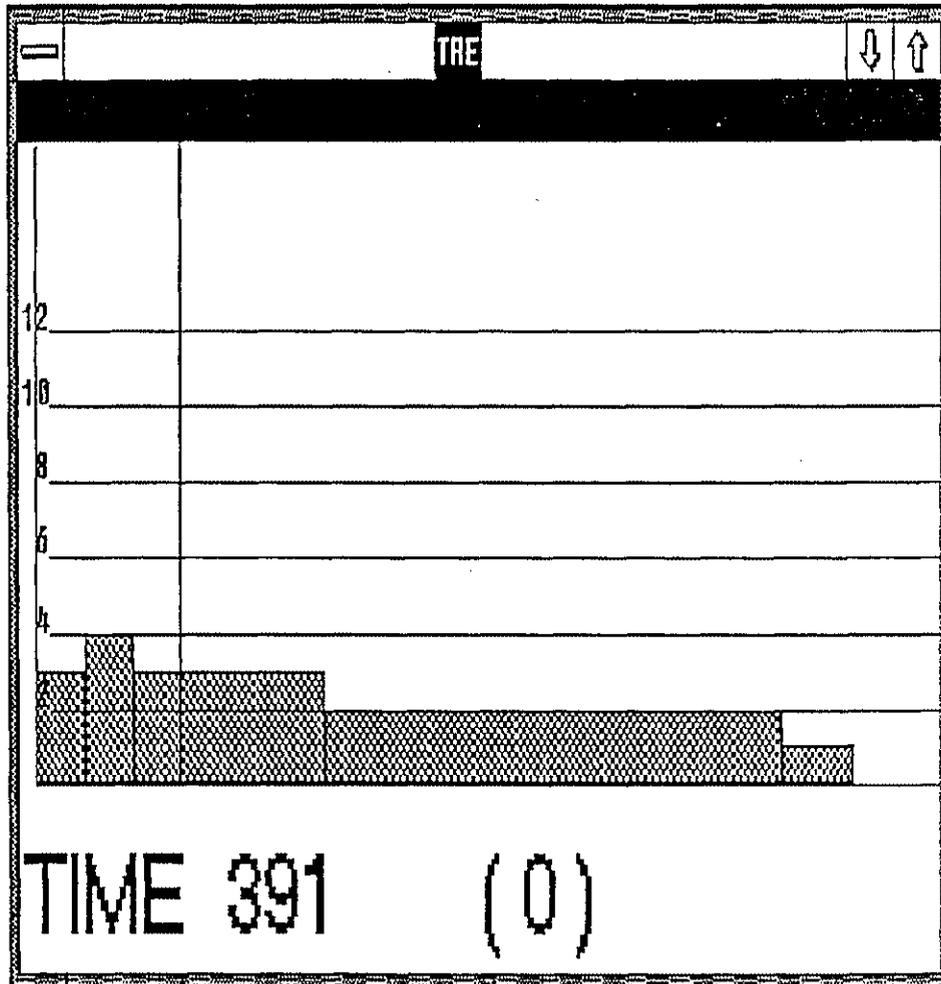


Figure 4.11. TRE Display corresponding to the TGP Display of Figure 4.8.

value of resources required for running the algorithm of Figure 4.1 is 4 corresponding to the peak value of the envelope.

The Resources window of Figure 4.12 shows that there are two variations in the number of resources for different values of TBO over the range of TBO_{LB} to ACT. One value shows $R = 4$ for $TBO = 1247$ units. This is the lower bound of TBO. The second value of R is detected as 3 at a TBO value of 2303 units. These two values of R are the R_{max} and R_{min} as observed in the SRE and the TRE display windows, respectively. The Performance Plane display window presented in Figure 4.13 shows the two operating points corresponding to the values of TBO and R as determined by the Resources window. As seen from the figure, these two points are located at TBO_{LB} . The position of the cross-hair cursor is indicated in time units and is shown at the bottom of the window. The Throughput display window is shown in Figure 4.14. It is seen from this figure that the throughput is 100 percent when operating with 4 resources and is about 55 percent of the maximum throughput with 3 resources. There is no variation in throughput for higher values of resources and this is evident from the two bars drawn for $R = 5$ and $R = 6$.

A control edge is now added to the original space surveillance algorithm between node 4 and node 2 to lower the resource requirements. The resulting graph is shown in Figure 4.15. The Buffer window shown in Figure 4.16 indicates that two buffers are required on the edges directed from node 0 to node 2, node 1 to node 6, node 3 to node 6, and node 4 to node 6. As seen from the Bounds window display shown in Figure 4.17,

RESOURCES	
TBO	RESOURCE
1247	4
2363	3

Figure 4.12. Resources Window for the Graph of Figure 4.1.

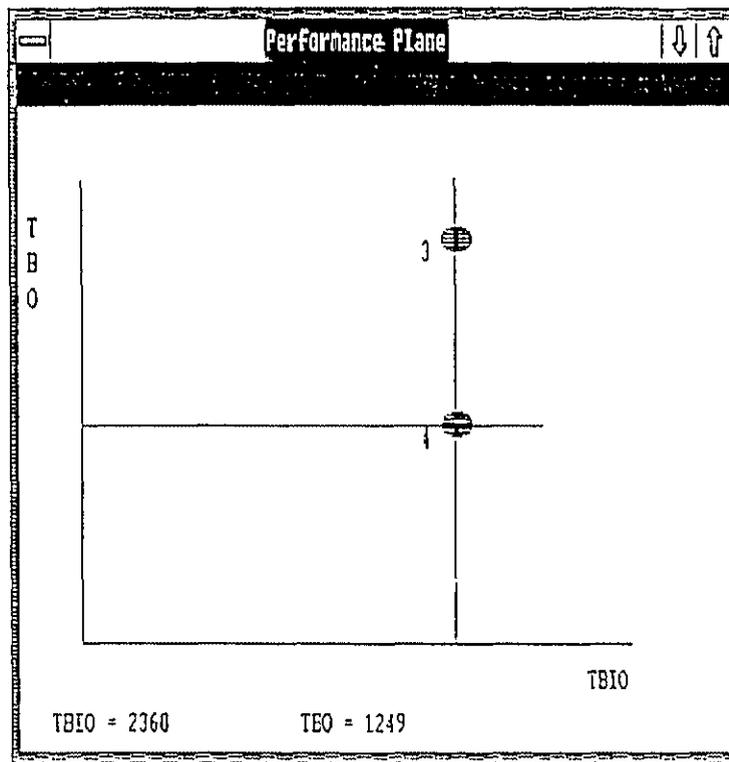


Figure 4.13. Performance Plane Display for the Graph of Figure 4.1.

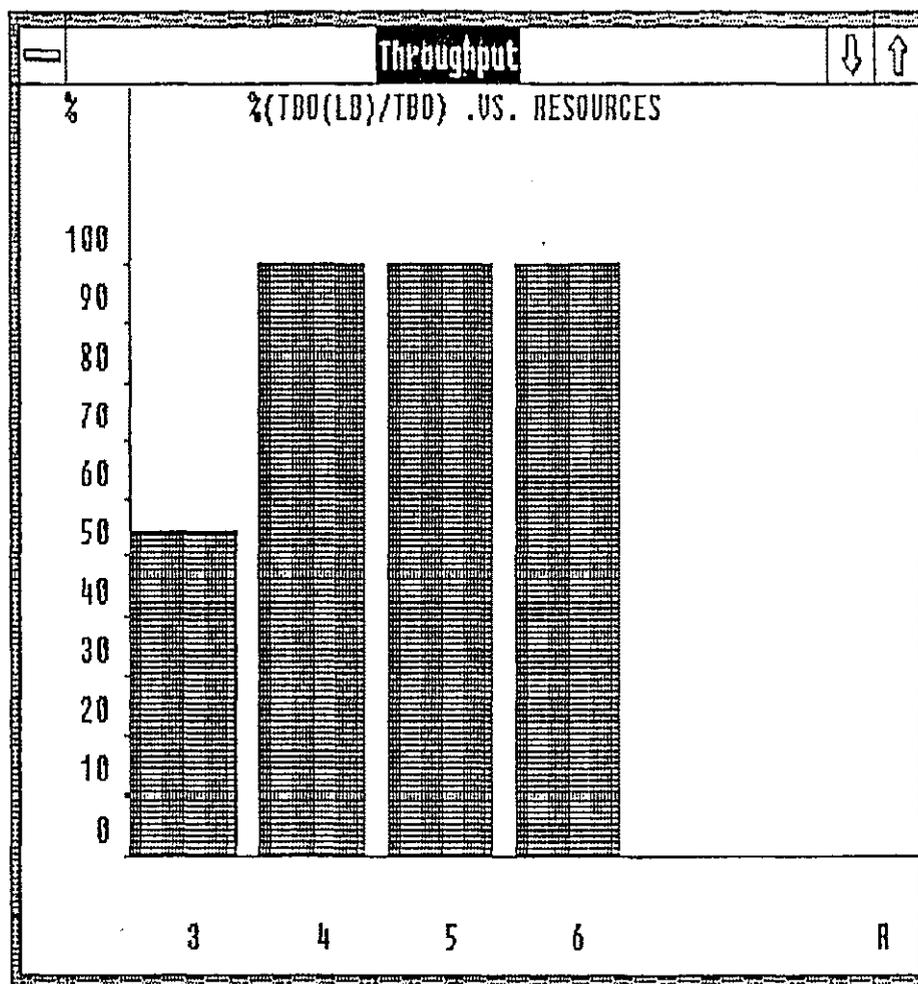


Figure 4.14. Throughput Display for the Graph of Figure 4.1.

BUFFER		
EDGE		
FROM	TO	BUFFER SIZE(DEFAULT 1)
0	2	2
1	6	2
3	6	2
4	6	2

Figure 4.16. Buffer Window for the Graph of Figure 4.15.

BOUNDS				
NODES	ES	EF	LS	LF
1	0	67	0	67
2	1314	1631	1314	1631
3	67	144	1661	1738
4	67	1314	67	1314
5	1631	1738	1631	1738
6	1738	2795	1738	2795
TCE	TBTO(LB)	TBO(LB)		
2872	2795	1247		
CRITICAL PATH(S)				
1 4 2 5 6				

Figure 4.17. Bounds Window for the Graph of Figure 4.15.

TBO_{LB} remains the same as it is for the graph of Figure 4.1. However, the $TBIO_{LB}$ has increased from 2371 to 2795, and the new critical path includes nodes 1, 4, 2, 5, and 6.

From the SGP display window shown in Figure 4.18, it is seen that only node 3 has float time and that node 2 is now executed after node 4 due to the control edge added from node 4 to node 2. The corresponding TGP display window is shown in Figure 4.19. This display also shows the float time associated with node 3. Figure 4.20 shows the TRE display window.

Figure 4.21 shows the Resources window for the graph of Figure 4.15. It is quite evident from this window that there are three variations in the values of resources. The three different values of R are 4, 3, and 2 with corresponding values of TBO equal to 1247, 1367, and 2747, respectively. The Performance Plane window is presented in Figure 4.22. It shows the new set of operating points generated along the $TBIO = 2795$ time units line due to the control edge added from node 4 to node 2. This window also displays the two operating points corresponding to the original algorithm graph given in Figure 4.1. The Throughput window is shown in Figure 4.23. It shows that throughput is 100 percent for $R = 4$. It also shows that throughput is about 90 percent of the maximum value for $R = 3$, and 45 percent of the maximum value for $R = 2$.

To further reduce the resource requirement, control edges are added to the graph of Figure 4.15 directed from node 4 to node 3 and from node 3 to node 2. The resulting graph is shown in Figure 4.24. According to the Buffer window display of Figure 4.25, two buffers are needed on each of the edges directed from node 0 to node 2, node 1 to node 3, node 1 to node 6, from node 4 to node 2, and node 6. The critical path is now

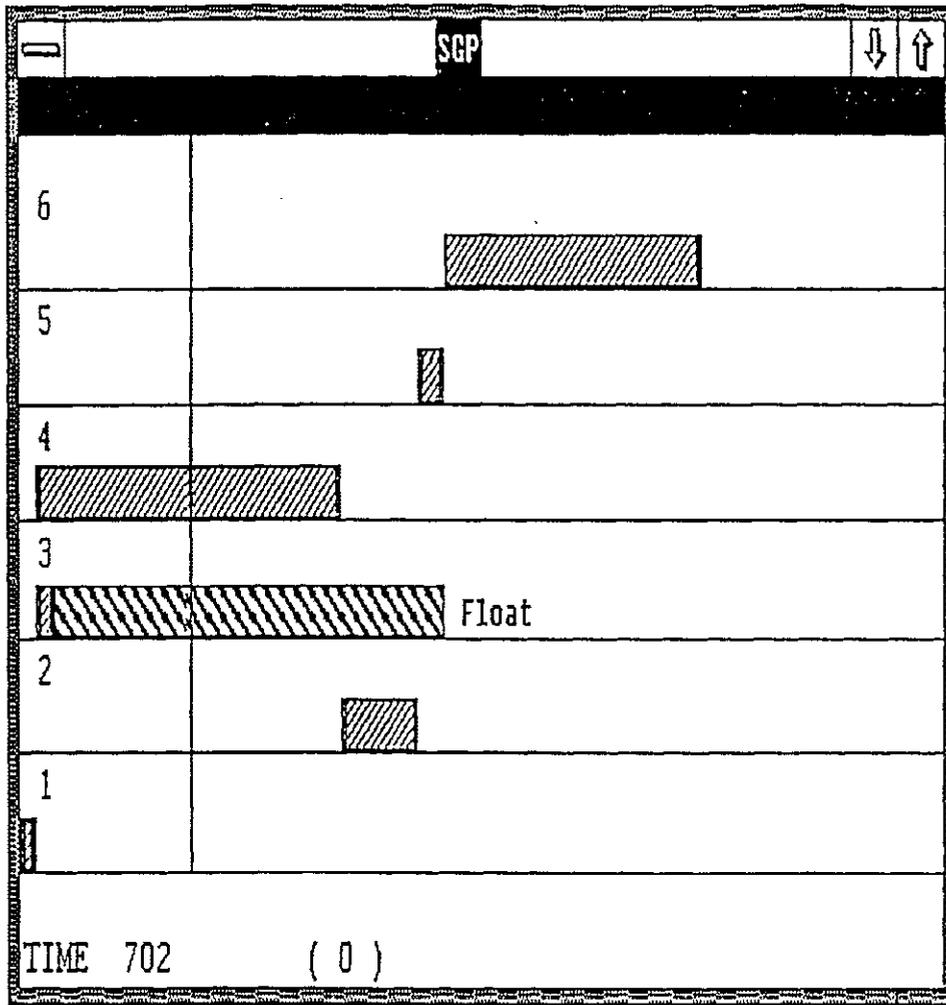


Figure 4.18. SGP Display for the Graph of Figure 4.15 showing Floats.

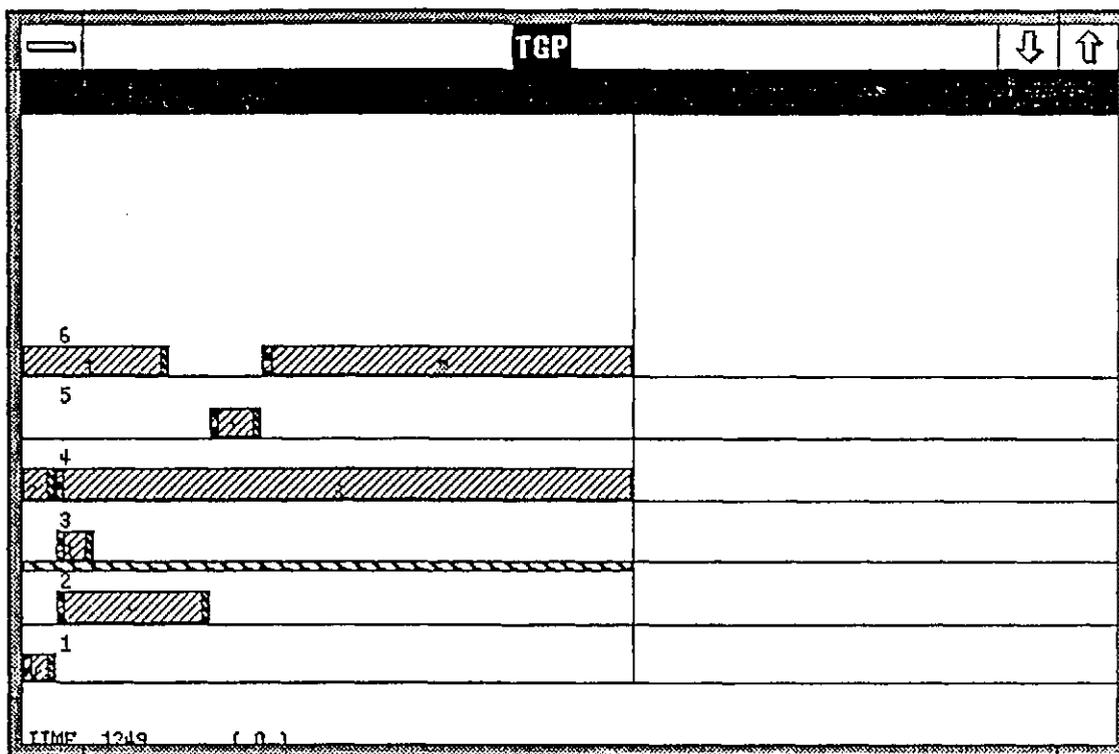


Figure 4.19. TGP Display for the Graph of Figure 4.15 showing Floats.

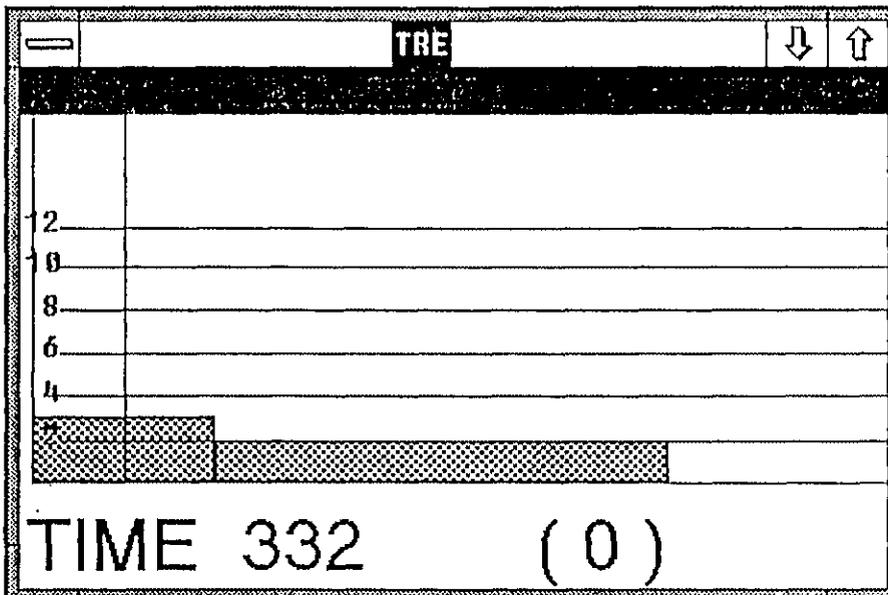


Figure 4.20. TRE Display for the Graph of Figure 4.15.

The RESOURCES window displays a table with two columns: T80 and RESOURCE. The table contains three rows of data.

T80	RESOURCE
1247	4
1367	3
2747	2

Figure 4.21 Resources Window corresponding to the Graph of Figure 4.15.

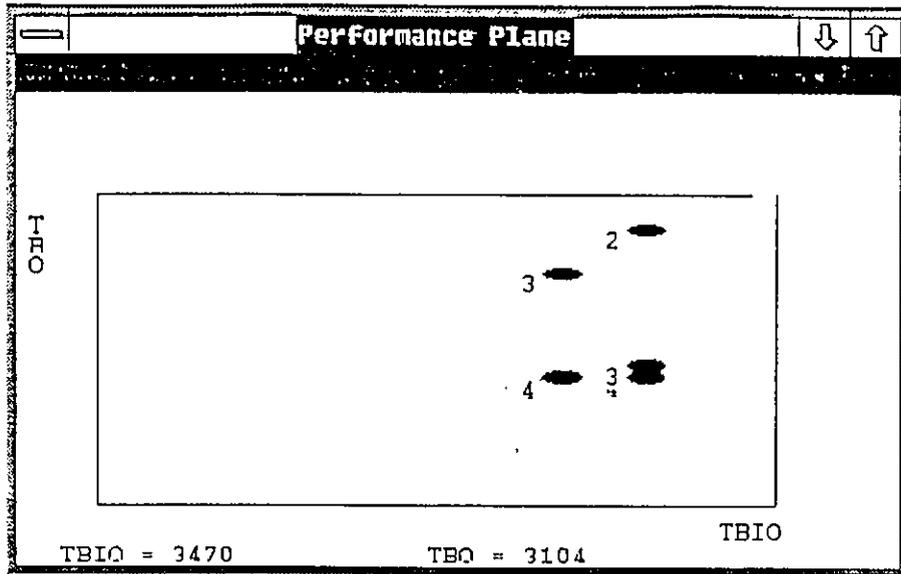


Figure 4.22 Performance Plane corresponding to the Graph of Figure 4.15.

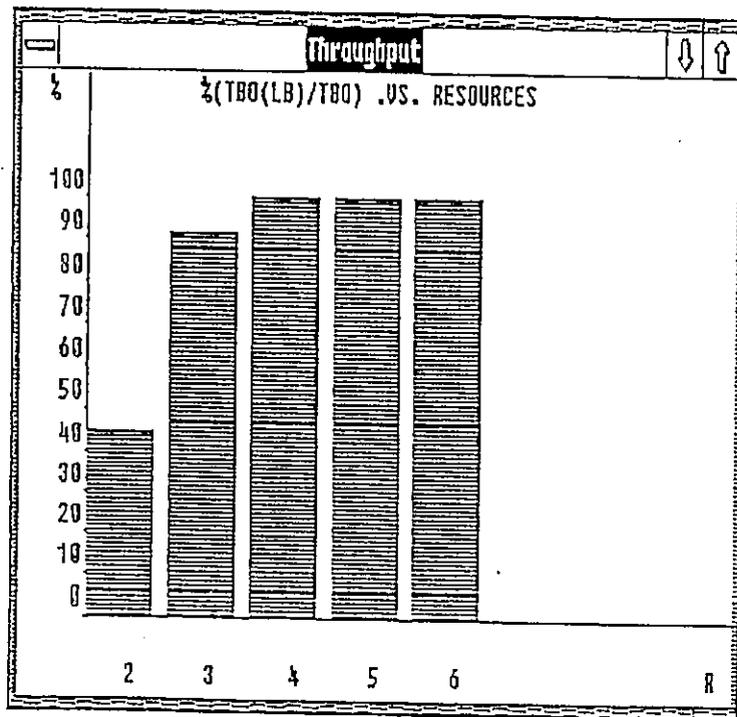


Figure 4.23. Throughput Display for the Graph of Figure 4.15.

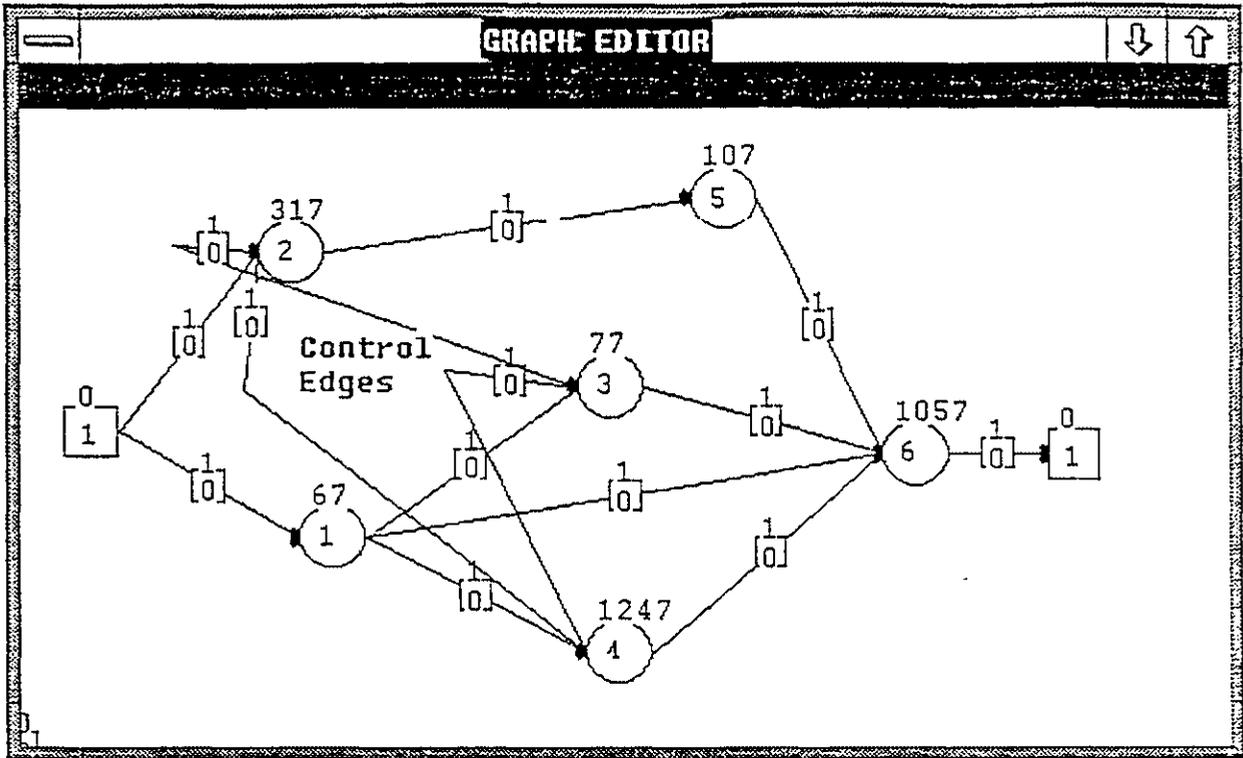


Figure 4.24. Graph with Control Edges from Nodes 4 to 2, 4 to 3, and 3 to 2.

EDGE		
FROM	TO	BUFFER SIZE(DEFAULT 1)
0	2	2
1	3	2
1	6	2
4	2	2
4	6	2

Figure 4.25. Buffer Window for the Graph of Figure 4.24.

BOUNDS				
NODES	ES	EF	LS	LF
1	0	67	0	67
2	1391	1708	1391	1708
3	1314	1391	1314	1391
4	67	1314	67	1314
5	1708	1815	1708	1815
6	1815	2872	1815	2872
TCE	TBIO(LB)	TBO(LB)		
2872	2872	1247		
CRITICAL PATH(S)				
1 4 3 2 5 6				

Figure 4.26. Bounds Window for the Graph of Figure 4.24.

observed to include all the nodes in the graph in the order 1, 4, 3, 2, 5, and 6. Hence, the graph is now reduced to a chain graph. The associated Bounds window is as shown in Figure 4.26. The corresponding SGP display window is shown in Figure 4.27. There are no float times associated with the nodes, as is expected. The minimum number of resources required is 1 as can be seen from the SGP diagram. Figure 4.28 shows the TGP diagram for the graph of Figure 4.24. Figure 4.29 shows the corresponding TRE display window. It is evident that the maximum value of resources required is 3 to operate at the lower bound value of TBIO.

The Performance Plane display window, along with the Modify Table display are shown in Figure 4.30. Each operating point is selected corresponding to a different value of resource number. The operating point with $R = 4$ is the operating point corresponding to the lower bounds of TBO and TBIO. The operating point with $R = 3$ maintains the same value of TBIO at the expense of increasing TBO. Operating at $R = 2$, the TBIO is increased considerably though the increase in TBO is not significant. To operate the system with only one resource, TBIO must equal TCE.

The Throughput display window is shown in Figure 4.31. It can be seen that the throughput is 100 percent for $R = 3$. It is about 86 percent of the maximum value with $R = 2$ and is about 42 percent of the maximum value with $R = 1$.

4.4 Decomposed State Equation

Consider the problem of computing the output of a discrete linear system given a sequence of inputs to the system. Let the system be described by the state equation

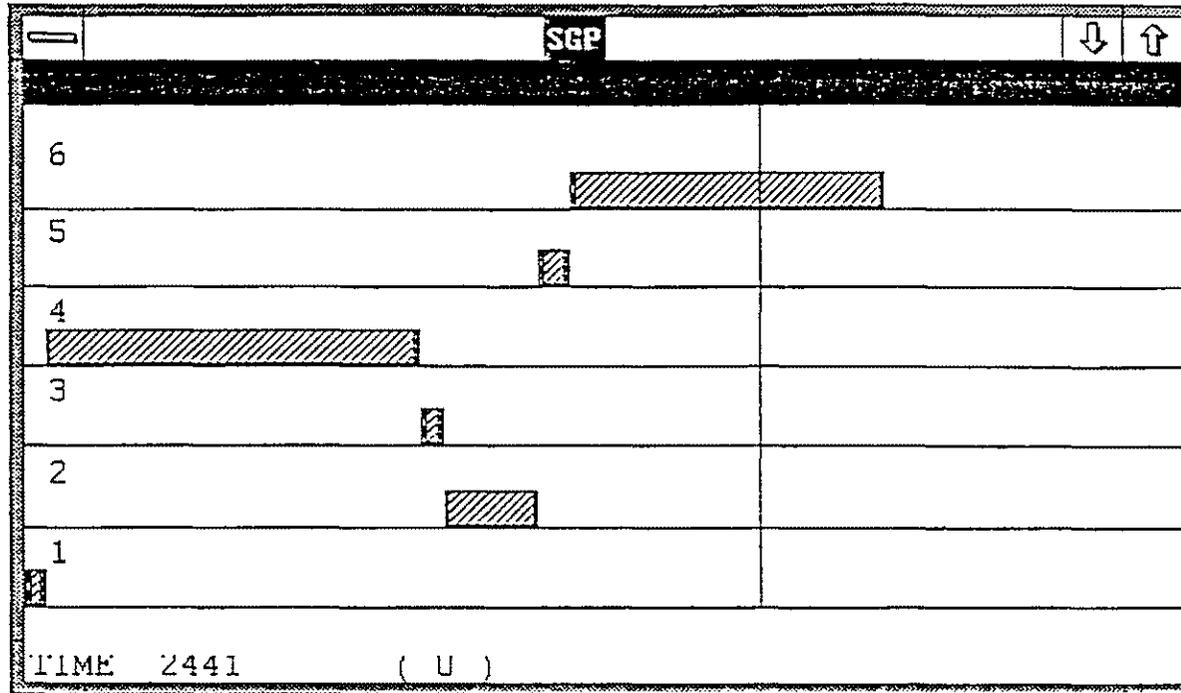


Figure 4.27. SGP Display corresponding to the Graph of Figure 4.24.

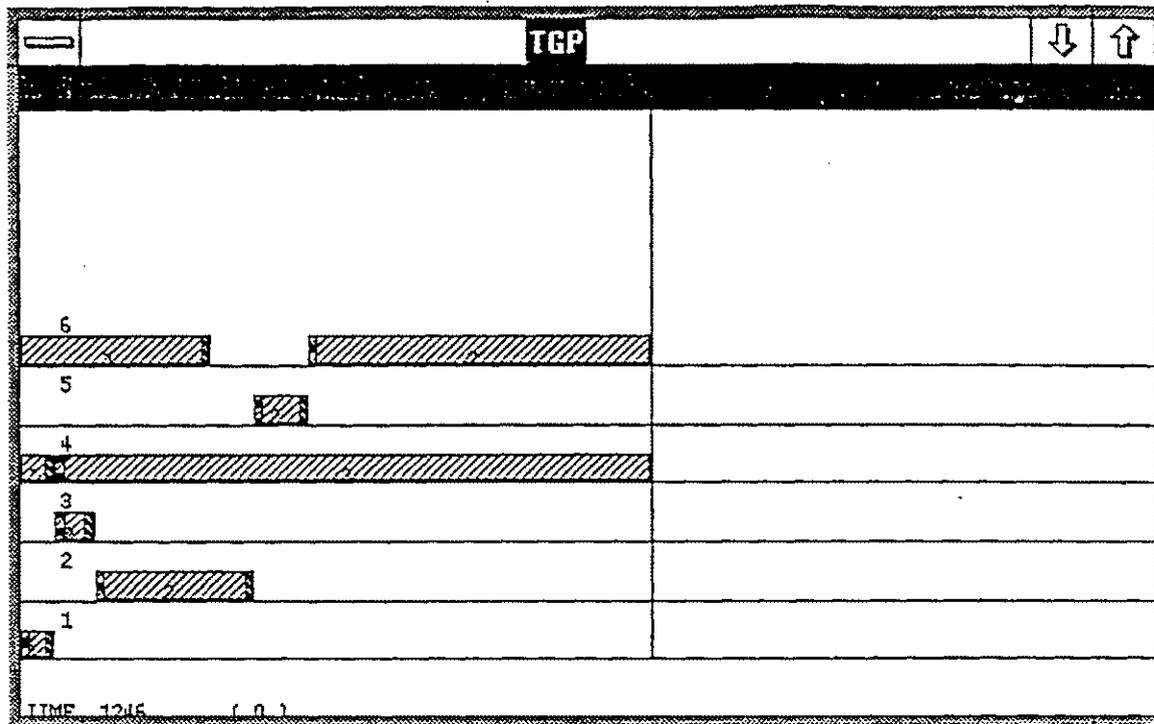


Figure 4.28. TGP Display corresponding to the Graph of Figure 4.24.

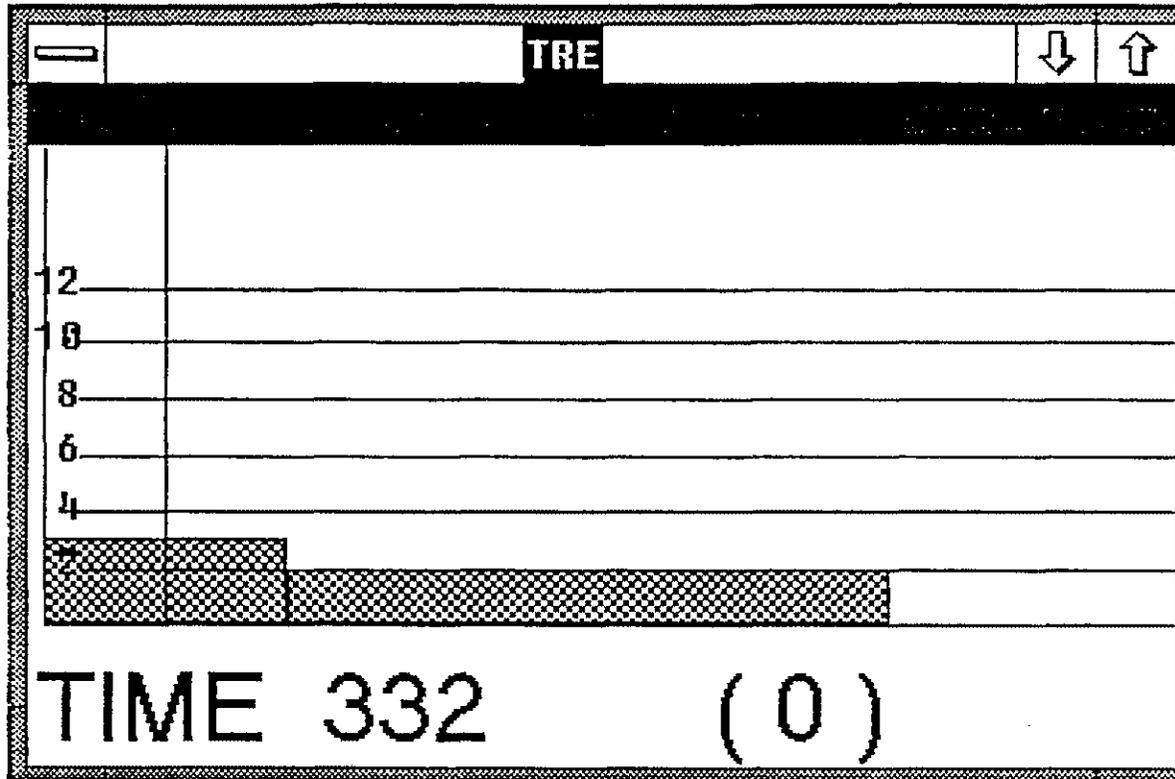
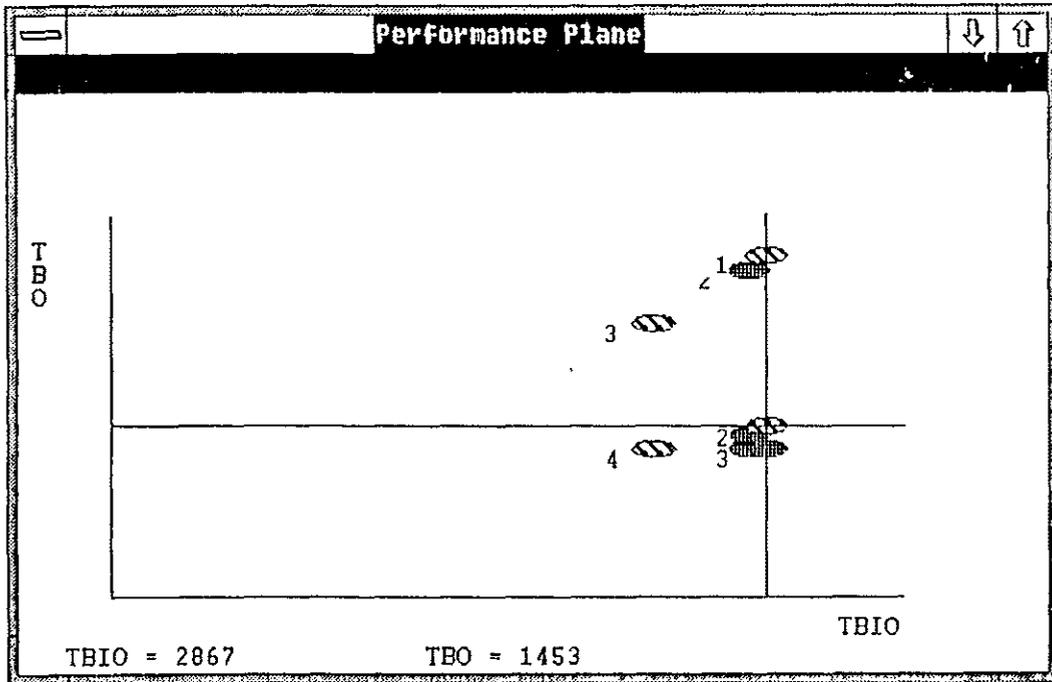


Figure 4.29. TRE Display for the Graph of Figure 4.24.



R	TBO	TBIO
4	1247	2371
3	2303	2371
1	2879	2872
2	1439	2872

C-EDGE		RESOURCES			
FROM	TO	4	3	1	2
4	2	0	0	0	0
4	3	0	0	1	1
3	2	0	0	1	1

BUFFERS		
FROM	TO	SIZE
0	2	2
1	3	2
1	6	2
4	2	2
4	6	2

Figure 4.30. Performance Plane Display showing Modify Table.

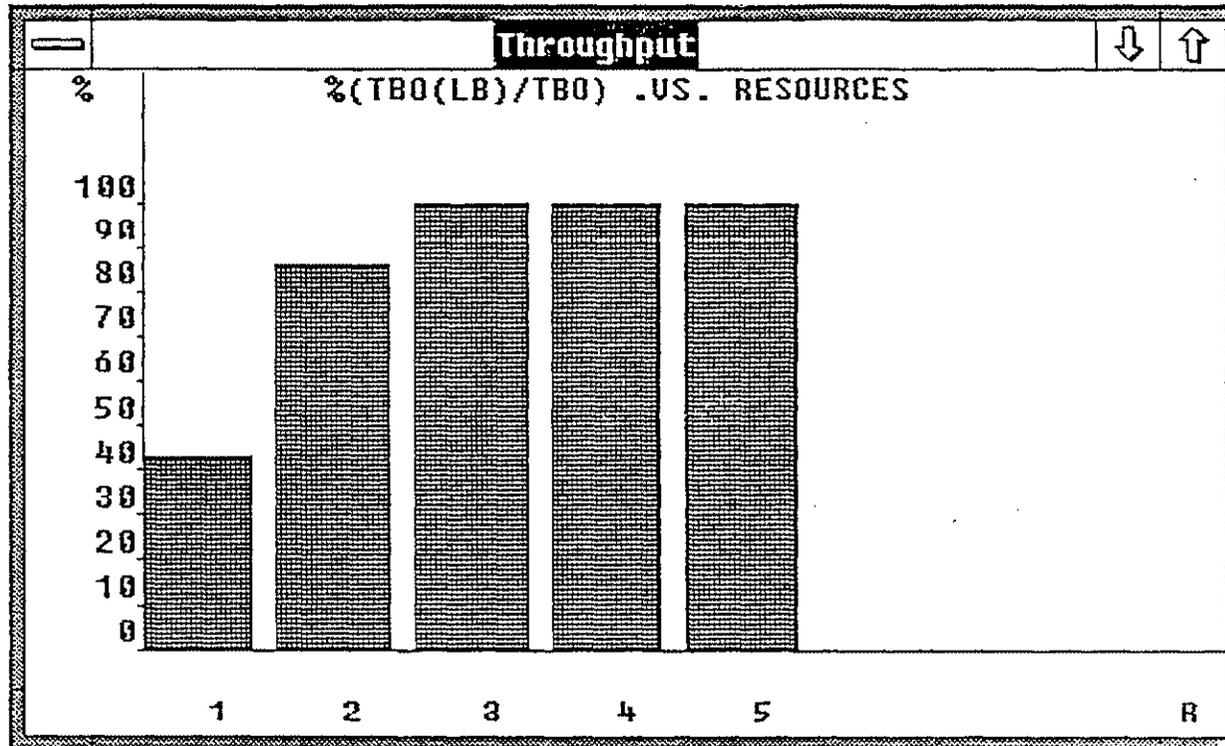


Figure 4.31. Throughput Display for the Graph of Figure 4.24.

$$X(K + 1) = A X(K) + B U(K + 1)$$

and output equation

$$Y(K + 1) = C X(K + 1)$$

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.

For the purpose of this case study, the state equation is decomposed so that the node times are reduced and parallel execution of nodes is possible. This decomposition lowers the value of TBO thus increasing throughput. The decomposition of the state equation is performed as follows,

$$\begin{bmatrix} X_1(K+1) \\ X_2(K+1) \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X_1(K) \\ X_2(K) \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} [U(K+1)],$$

$$Y(K+1) = [C_1 \ C_2] \begin{bmatrix} X_1(K+1) \\ X_2(K+1) \end{bmatrix}.$$

The AMG representing the decomposed state equation is as shown in the Graph Editor display window of Figure 4.32. The nodes and the edges are labeled in accordance with the above two equations.

Figure 4.33 shows the Buffer display window. It is evident from this figure that no extra buffers are needed on any of the edges of the graph of Figure 4.32. From the

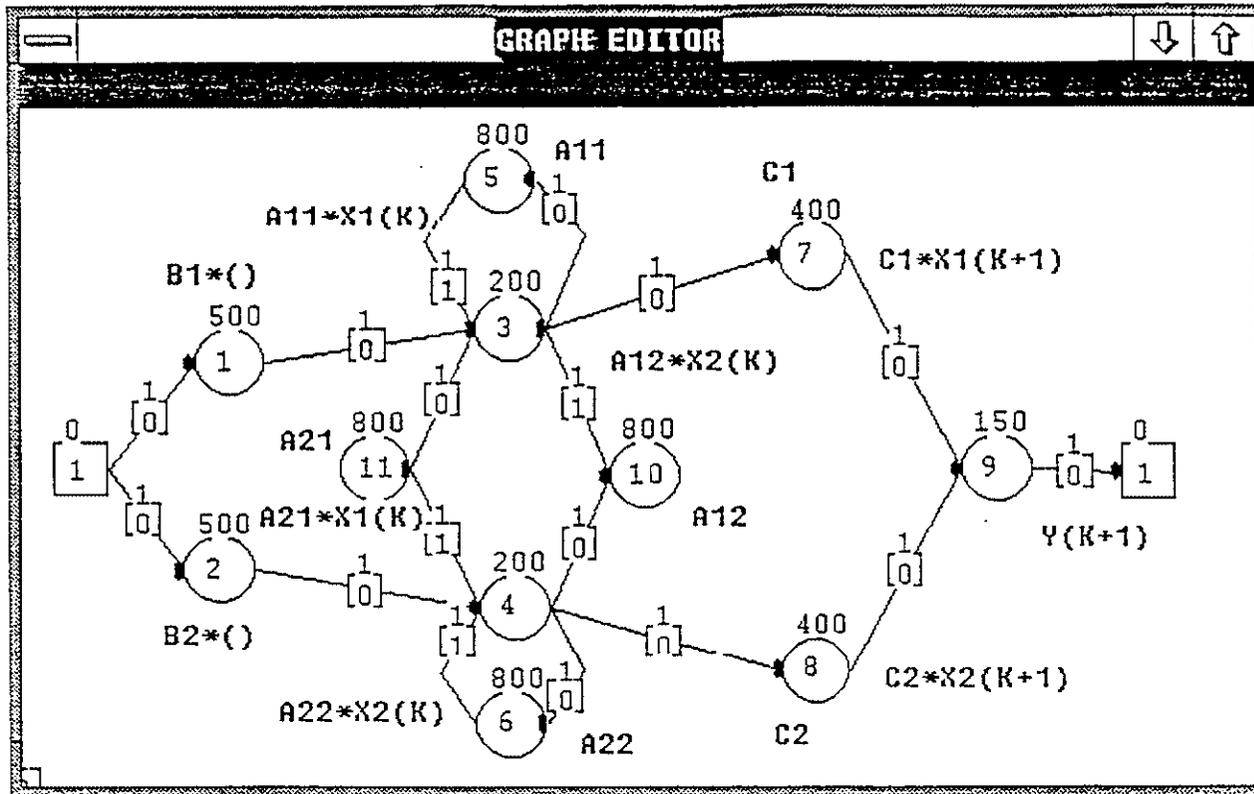


Figure 4.32. Decomposed State Equation Algorithm.

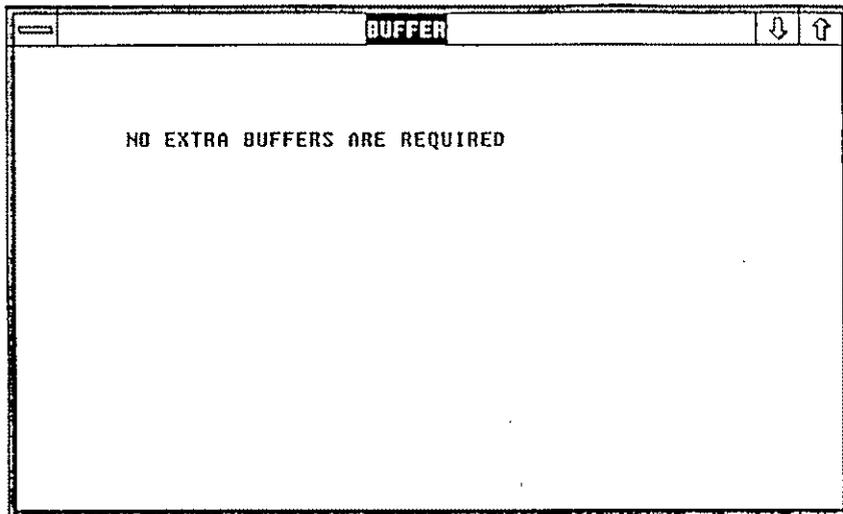


Figure 4.33. Buffer Window for the Graph of Figure 4.32.

Figure 4.34 shows a window titled "BOUNDS". The window displays the following data:

NODES	ES	EF	LS	LF
1	0	500	0	500
2	0	500	0	500
3	500	700	500	700
4	500	700	500	700
5	700	1500	700	1500
6	700	1500	700	1500
7	700	1100	700	1100
8	700	1100	700	1100
9	1100	1250	1100	1250
10	700	1500	700	1500
11	700	1500	700	1500

TCE	TDI0(LD)	TDO(LD)
5550	1250	1000

CRITICAL PATH(S)

2 4 8 9

1 3 7 9

Figure 4.34. Bounds Window corresponding to the Graph of Figure 4.32.

Bounds window of Figure 4.34, it is seen that the value of TBO_{LB} is 1000 units, which is equal to one-half of the total time of the circuit formed by nodes 4, 10, 3, and 11 since there are two tokens in the circuit. Also, it is observed that there are two critical paths in the graph of Figure 4.32, one formed by nodes 2, 4, 8, and 9 and the other formed by nodes 1, 3, 7, and 9.

The SGP display window is presented in Figure 4.35. As is shown, there are no float times associated with any of the nodes. The corresponding SRE display window is shown in Figure 4.36. The peak value of the resource envelope is 6 and hence the minimum number of resources required to achieve a $TBIO = TBIO_{LB}$ is 6.

Figure 4.37 shows the TGP diagram and the corresponding TRE diagram is shown in Figure 4.38. The peak value of resources as shown by the TRE display window is 8 and hence the maximum number of resources required to operate at $TBIO_{LB}$ is 8. The Resources window of Figure 4.39 shows three different values of resources for three different values of TBO with the minimum resource requirement being 6 and the maximum being 8. The corresponding Performance Plane window and the Throughput window are shown in Figure 4.40 and Figure 4.41 respectively. The throughput, as seen from the figure is 100 percent for $R = 8$. It is about 90 percent of the maximum value for $R = 7$ and is about 80 percent of the maximum value for $R = 6$.

Figure 4.42 shows the decomposed state equation algorithm with a control edge added from node 1 to node 2. The addition of the control edge requires no additional buffers on any of the edges as is displayed by the Buffer window of Figure 4.43. The

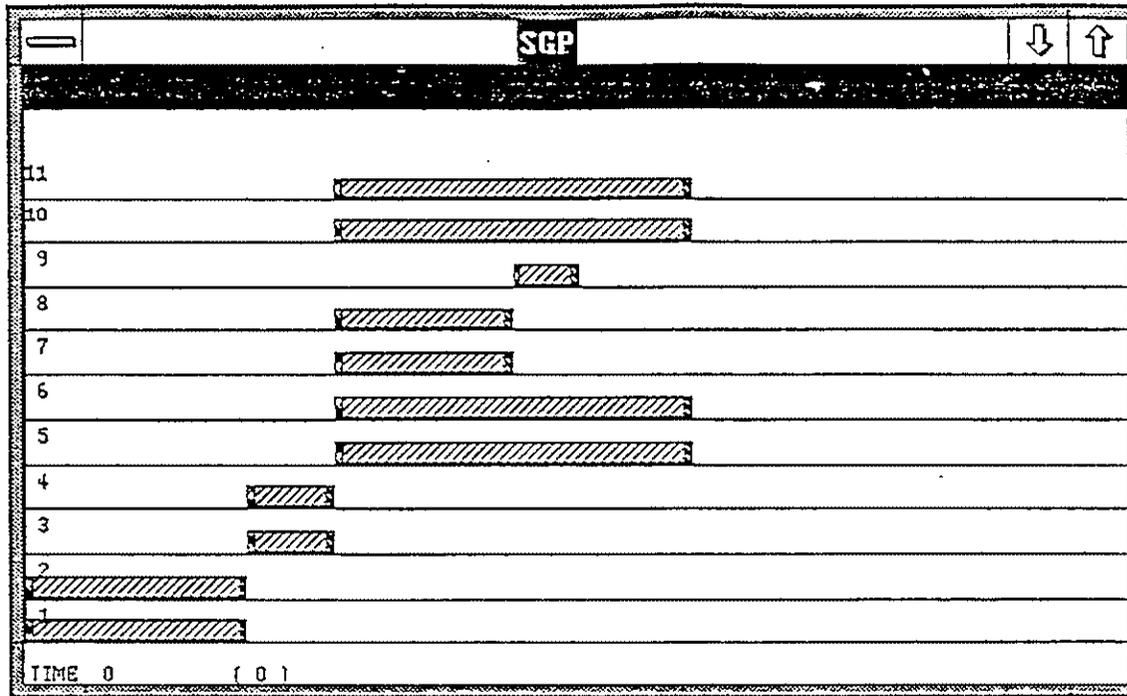


Figure 4.35. SGP Display for the Graph of Figure 4.32.

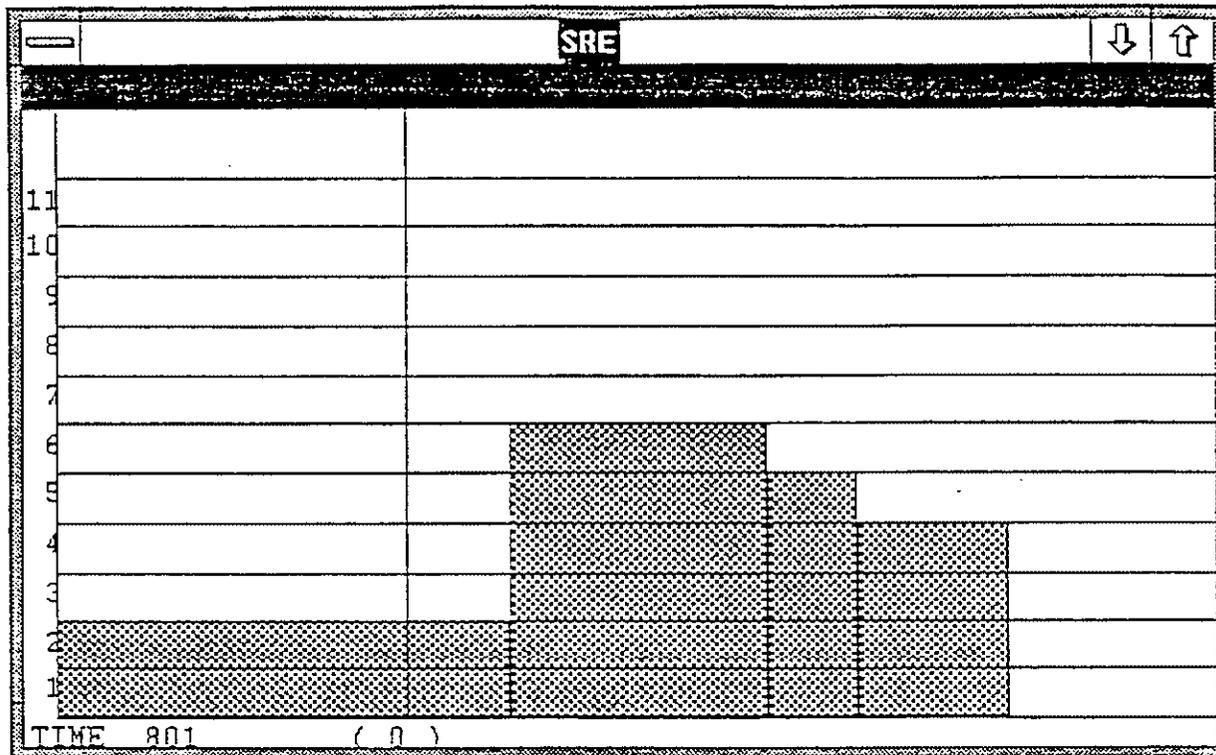


Figure 4.36. SRE Display corresponding to the SGP Display of Figure 4.35.

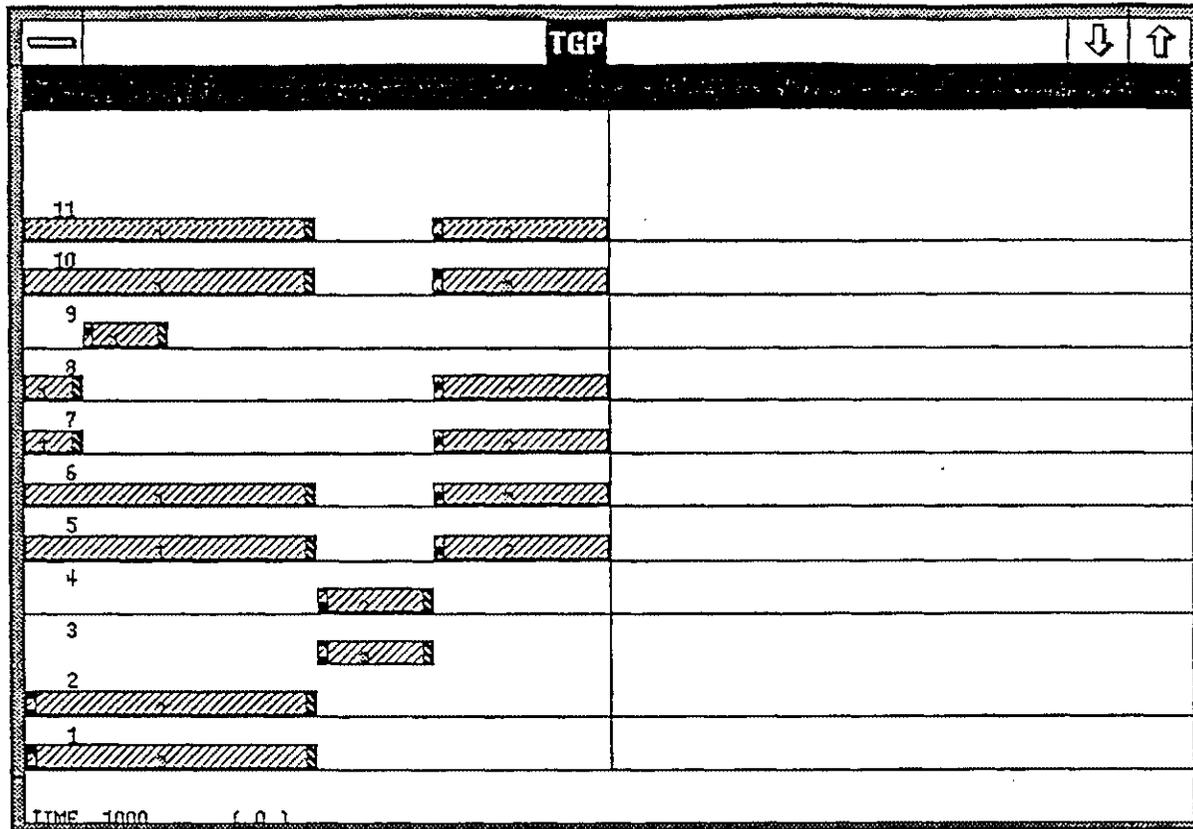


Figure 4.37. TGP Display for the Graph of Figure 4.32.

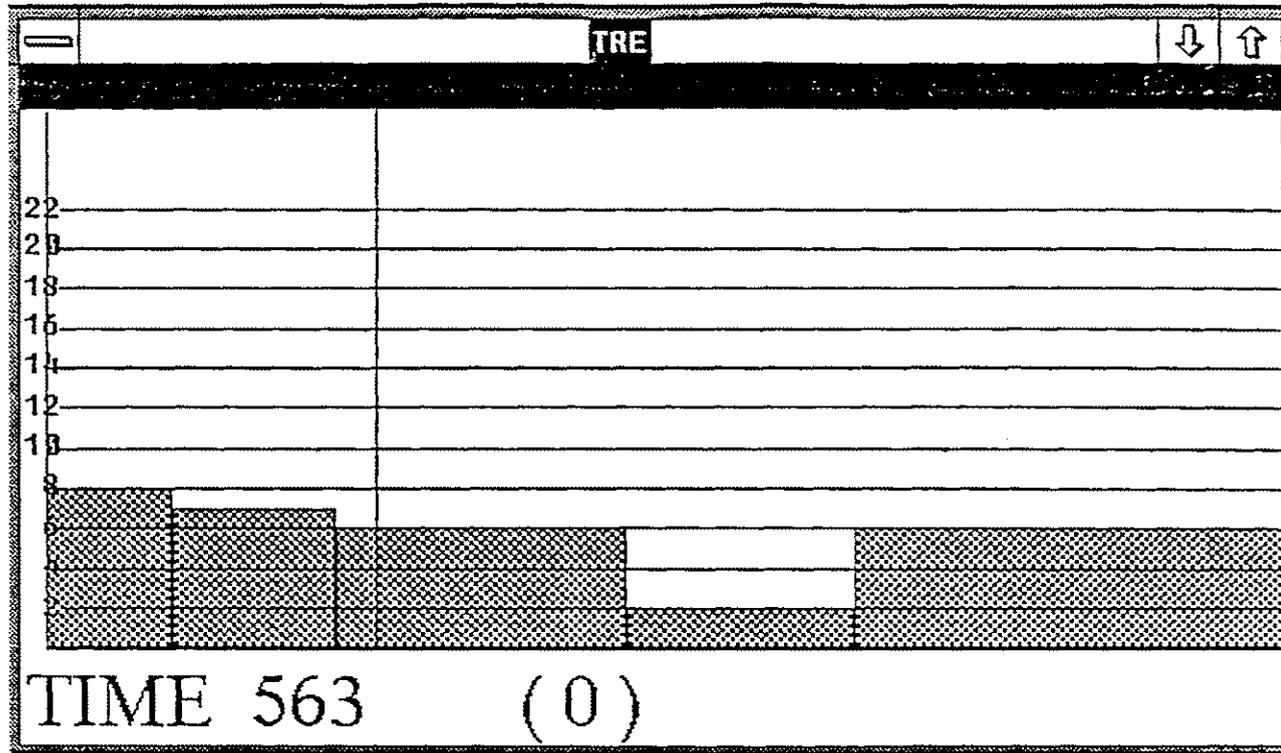


Figure 4.38. TRE Display corresponding to the TGP Display of Figure 4.37.

RESOURCES	
TBO	RESOURCE
1000	8
1100	7
1250	6

Figure 4.39. Resources Window for the Graph of Figure 4.32.

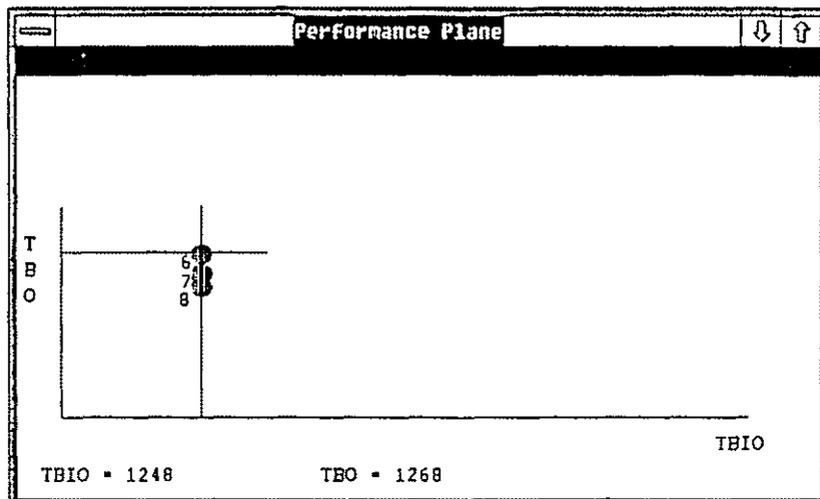


Figure 4.40. Performance Plane Display for the Graph of Figure 4.32.

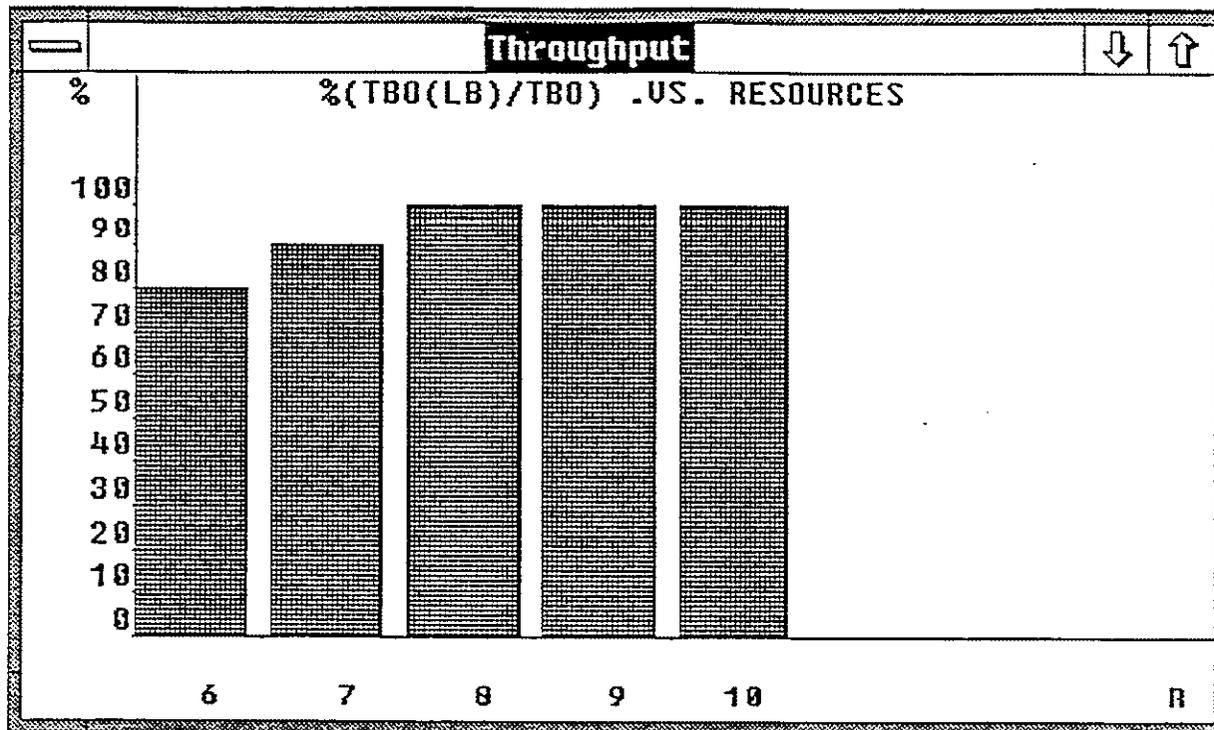


Figure 4.41. Throughput Display for the Graph of Figure 4.32.

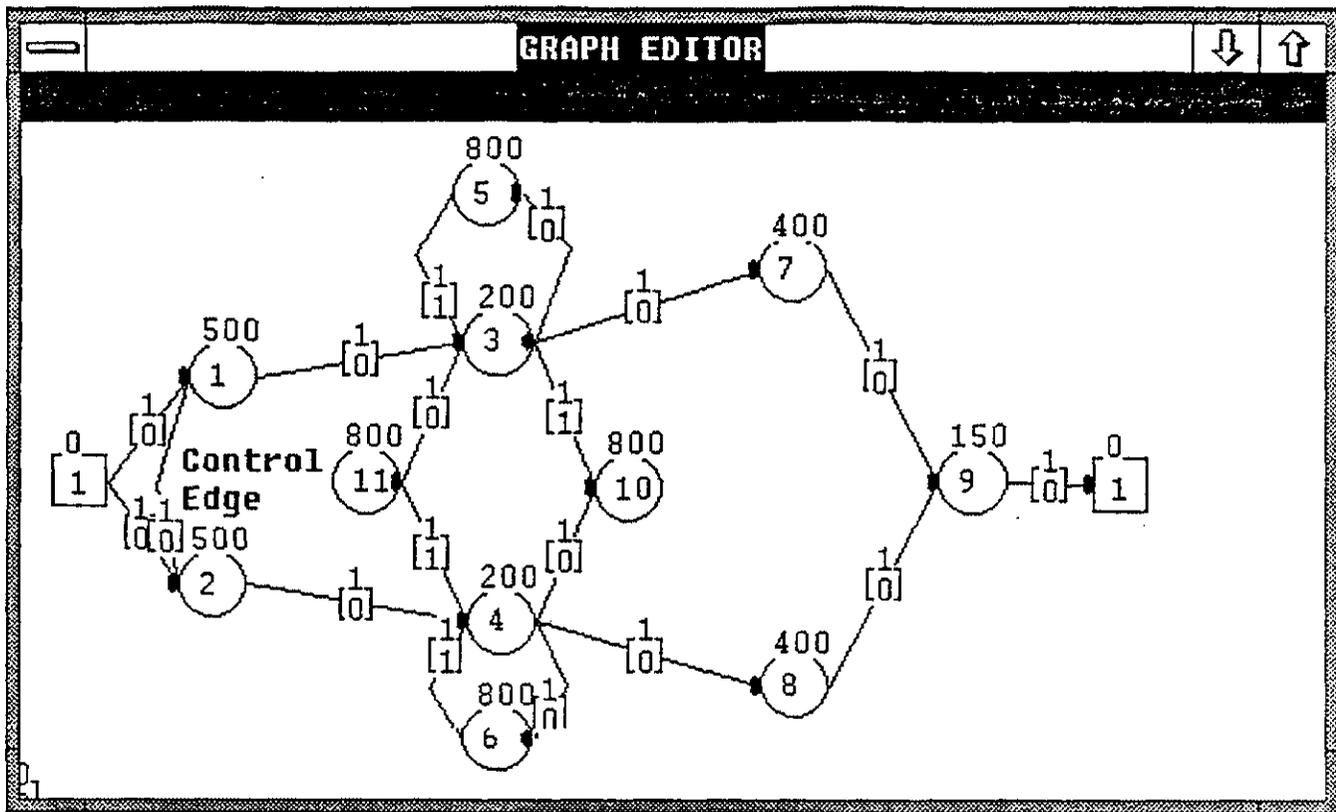


Figure 4.42. Graph with Control Edge from Node 1 to Node 2.

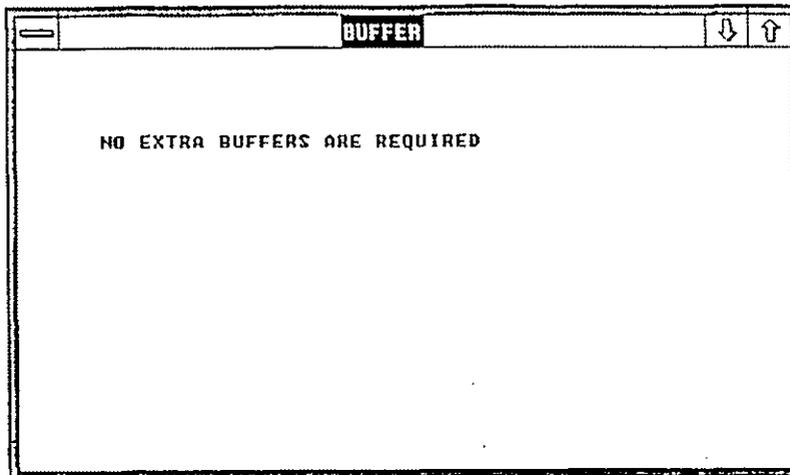


Figure 4.43. Buffer Window for the Graph of Figure 4.42.

The image shows a window titled "BOUNDS" with a standard window control bar and scroll arrows. The window contains a table of node values and critical path information.

NODES	ES	EF	LS	LF
1	0	500	0	500
2	500	1000	500	1000
3	500	700	1000	1200
4	1000	1200	1000	1200
5	700	1500	1200	2000
6	1200	2000	1200	2000
7	700	1100	1200	1600
8	1200	1600	1200	1600
9	1600	1750	1600	1750
10	1200	2000	1200	2000
11	700	1500	1200	2000

TCE	TB10(LB)	TB0(LB)
5550	1750	1000

CRITICAL PATH(S)

1 2 4 8 9

Figure 4.44. Bounds Window for the Graph of Figure 4.42.

Bounds window is shown in Figure 4.44. It is observed that the number of critical paths in the graph is now reduced to one, and includes the nodes 1, 2, 4, 8, and 9. The value of TBIO is increased to 1750 units. However, the addition of the control edge introduces float times to nodes 3, 5, 7, and 11 as shown in the SGP display window of Figure 4.45. The corresponding TGP display window is presented in Figure 4.46. Figure 4.47 shows the TRE display window for the TGP of Figure 4.46. From Figures 4.45 and 4.47, the minimum and the maximum numbers of resources required to operate at $TBIO_{LB}$ are calculated as 5 and 7, respectively. The Resources window is presented in Figure 4.48. Figure 4.49 shows the Performance Plane with two sets of operating points corresponding to two different values of TBIO. It is seen from the Throughput display window of Figure 4.50 that the throughput is 100 percent for $R = 7$. For $R = 6$, the throughput is about 90 percent of the maximum value and for $R = 5$, it is about 68 percent of the maximum value.

To reduce the resource requirement further, control edges are added to the graph of Figure 4.42 from node 7 to node 8 and from node 10 to node 8. The resulting graph is shown in Figure 4.51. It is evident from the Buffer display window of Figure 4.52 that two buffers are required on each of the edges directed from node 7 to node 8 and from node 7 to node 9. The critical path now includes nodes 1, 2, 4, 10, 8, and 9 as displayed in the Bounds display window of Figure 4.53. The SGP display window is presented in Figure 4.54. It can be seen that the nodes 3, 5, 7, and 11 still have float times. The corresponding TGP display window is shown in Figure 4.55, and Figure 4.56 is the TRE display window. The minimum and the maximum values of the resources are 4 and 6,

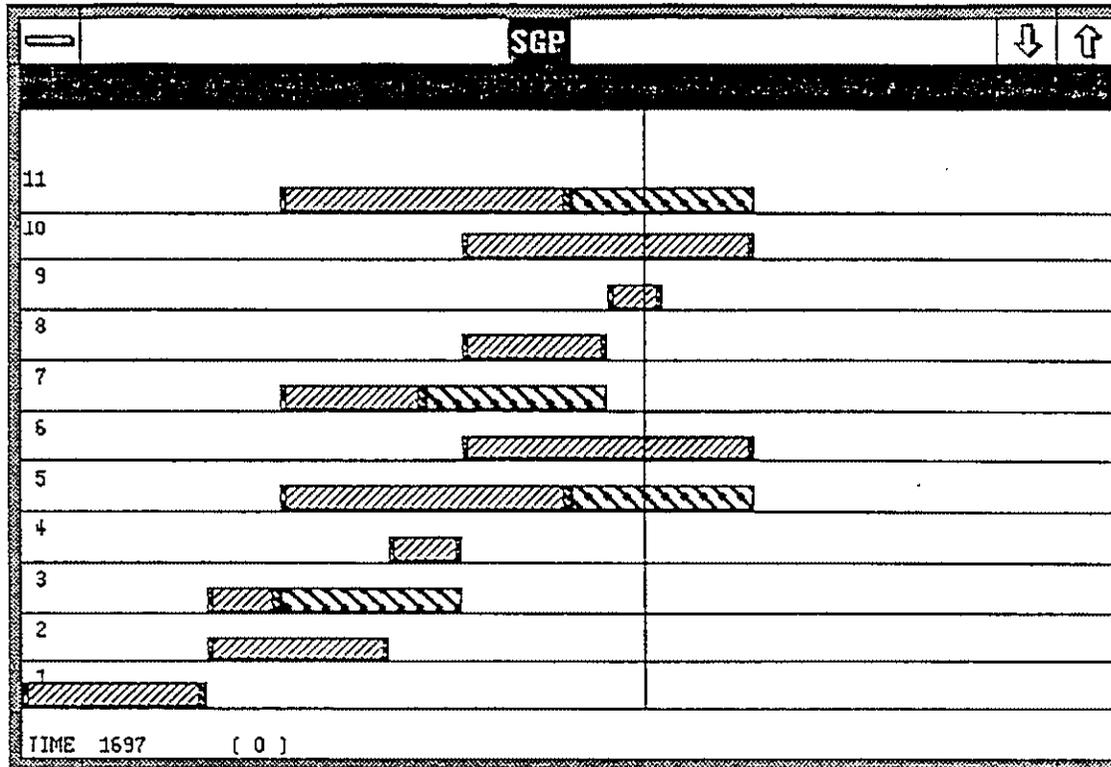


Figure 4.45. SGP Display for the Graph of Figure 4.42.

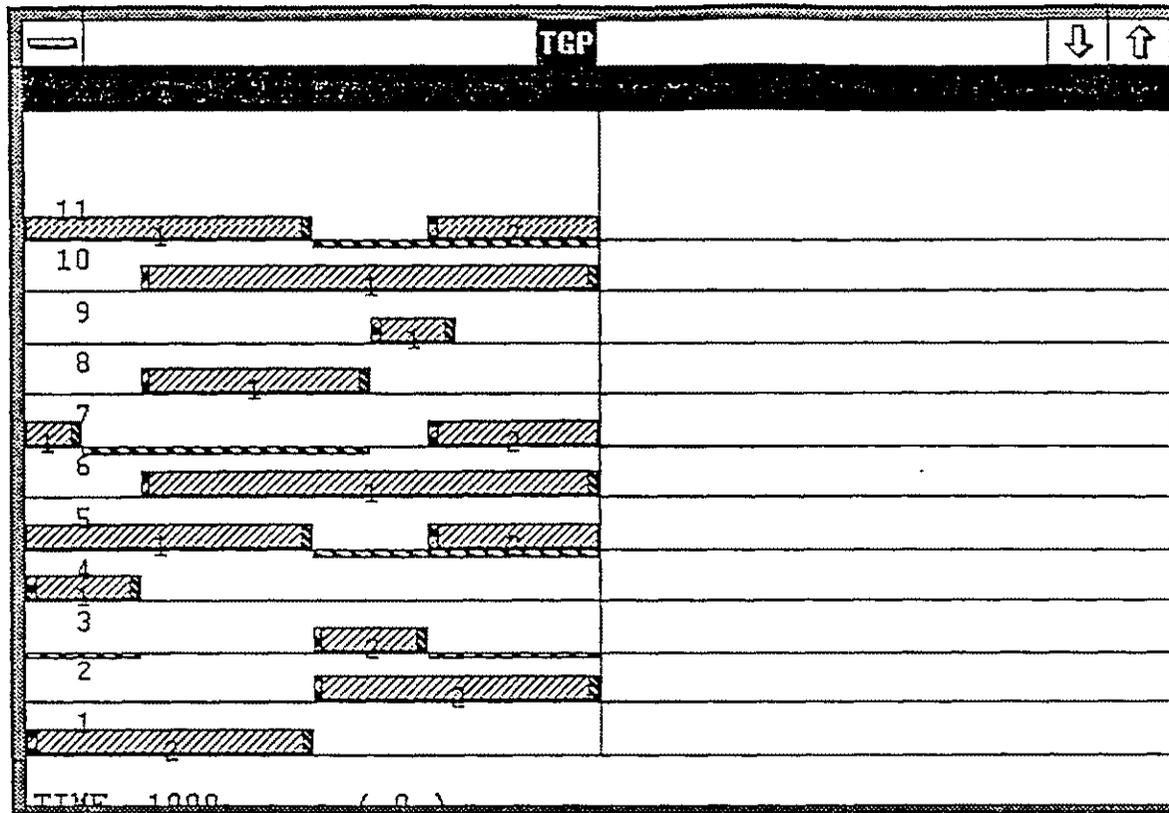


Figure 4.46. TGP Display for the Graph of Figure 4.42.

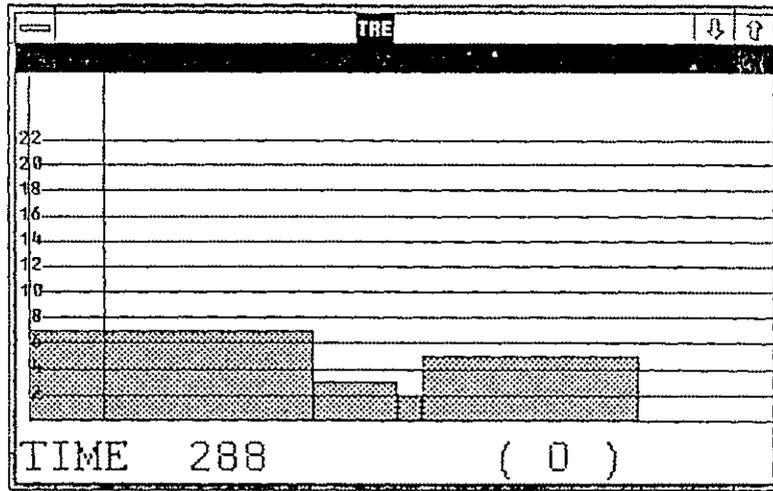


Figure 4.47. TRE Display for the graph of Figure 4.42.

TBO	RESOURCE
1000	7
1060	6
1500	5

Figure 4.48. Resources Window for the Graph of Figure 4.42.

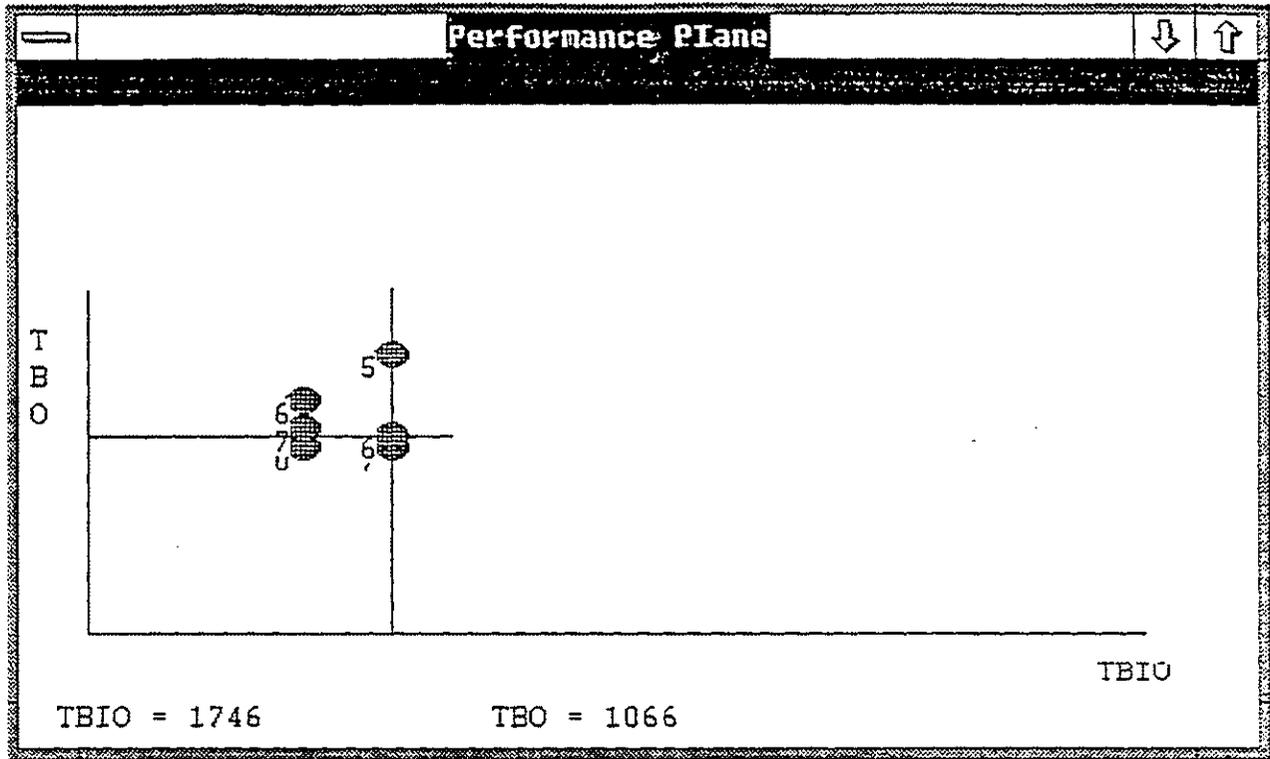


Figure 4.49. Performance Plane Display for the Graph of Figure 4.42.

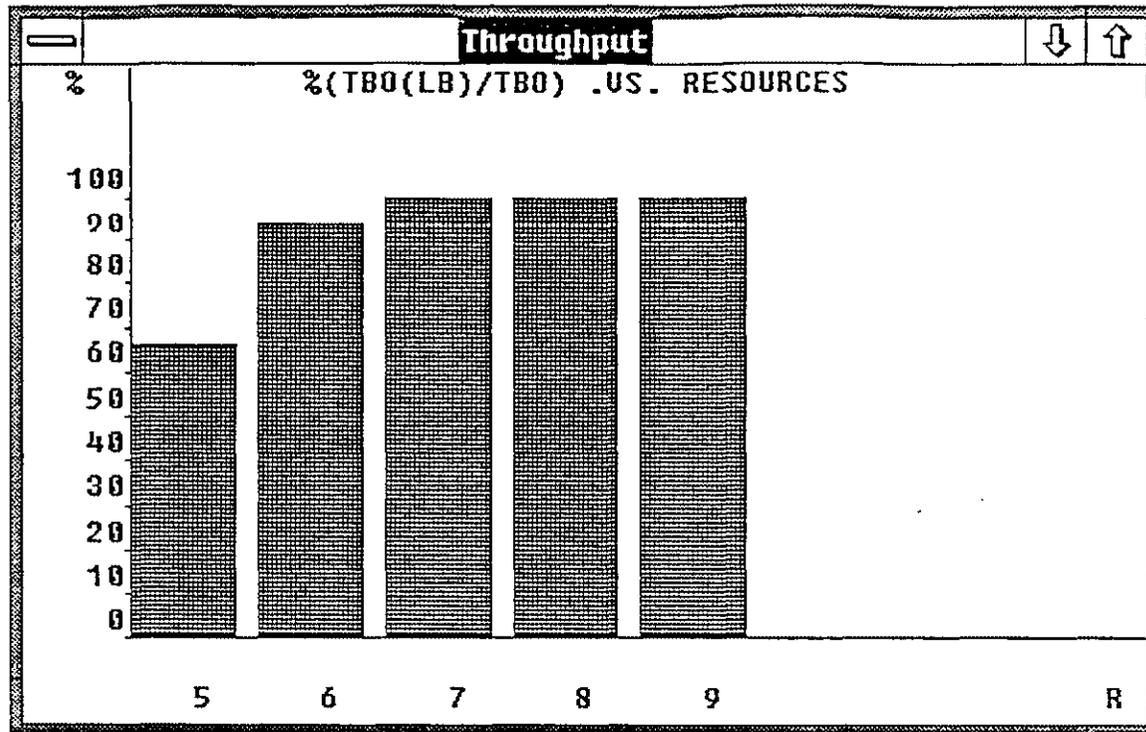


Figure 4.50. Throughput Display corresponding to the Graph of Figure 4.42.

EDGE		
FROM	TO	BUFFER SIZE(DEFAULT 1)
7	8	2
7	9	2

Figure 4.52. Buffer Window for the Graph of Figure 4.51.

NODES	ES	EF	LS	LF
1	0	500	0	500
2	500	1000	500	1000
3	500	700	1000	1200
4	1000	1200	1000	1200
5	700	1500	1600	2400
6	1200	2000	1200	2000
7	700	1100	1600	2000
8	2000	2400	2000	2400
9	2400	2550	2400	2550
10	1200	2000	1200	2000
11	700	1500	1200	2000

TCC	TDIO(LD)	TDO(LD)
5550	2550	1000

CRITICAL PATH(S)

1 2 4 10 8 9

Figure 4.53. Bounds Window corresponding to the Graph of Figure 4.51.

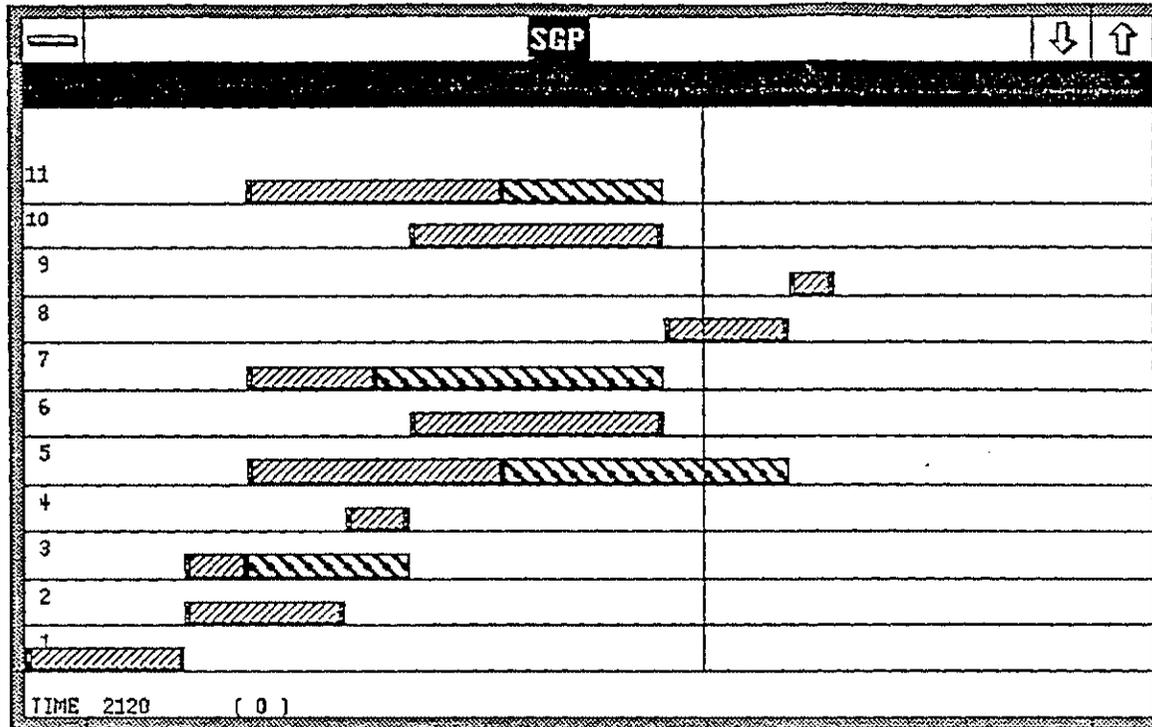


Figure 4.54. SGP Display for the Graph of Figure 4.51.

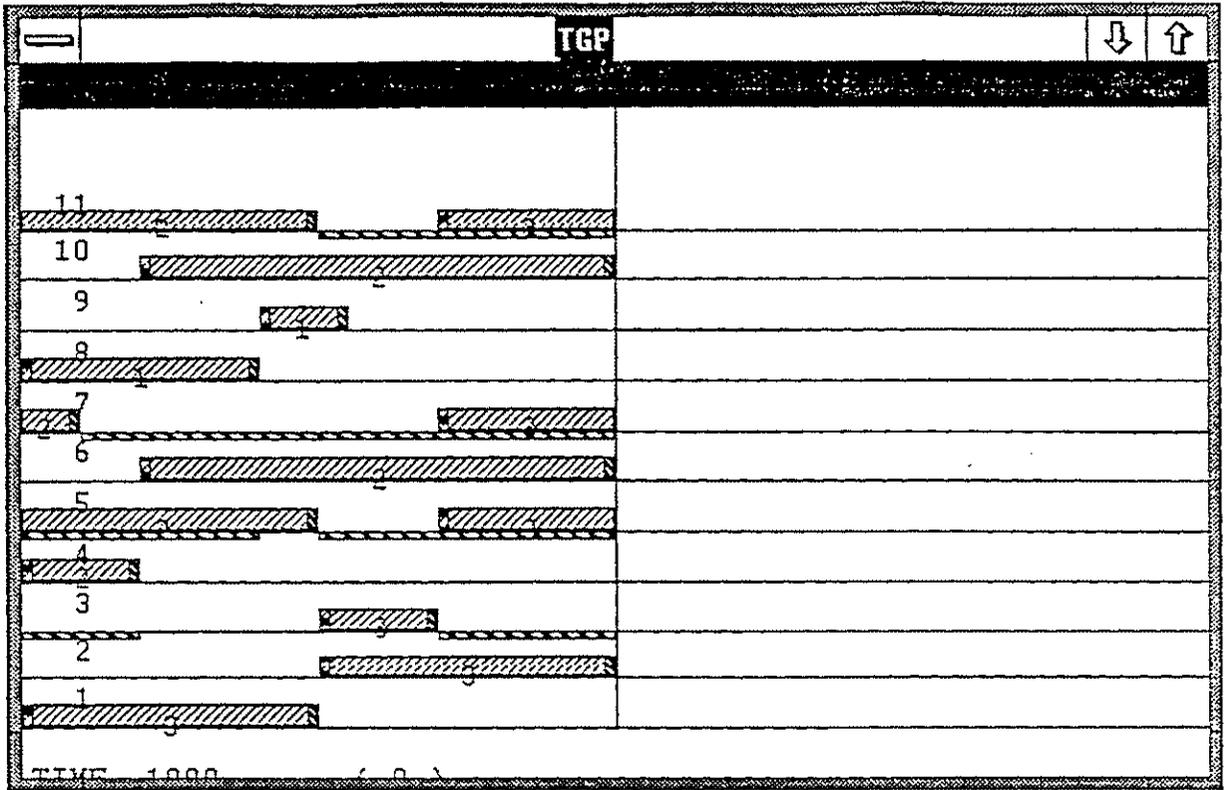


Figure 4.55. TGP Display for the Graph of Figure 4.51.

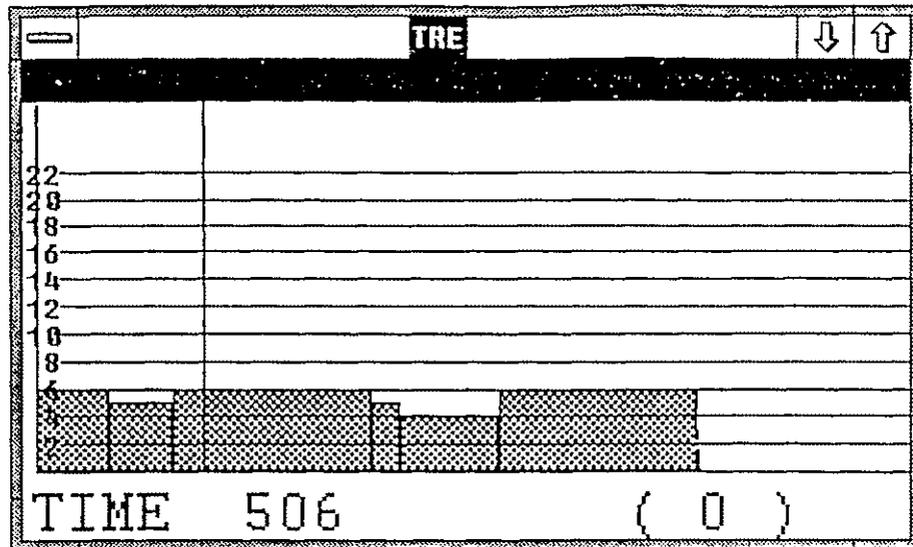


Figure 4.56. TRE Display for the Graph of Figure 4.51.

The screenshot shows a window titled "RESOURCES" containing a table with two columns: "YBO" and "RESOURCE". The data is as follows:

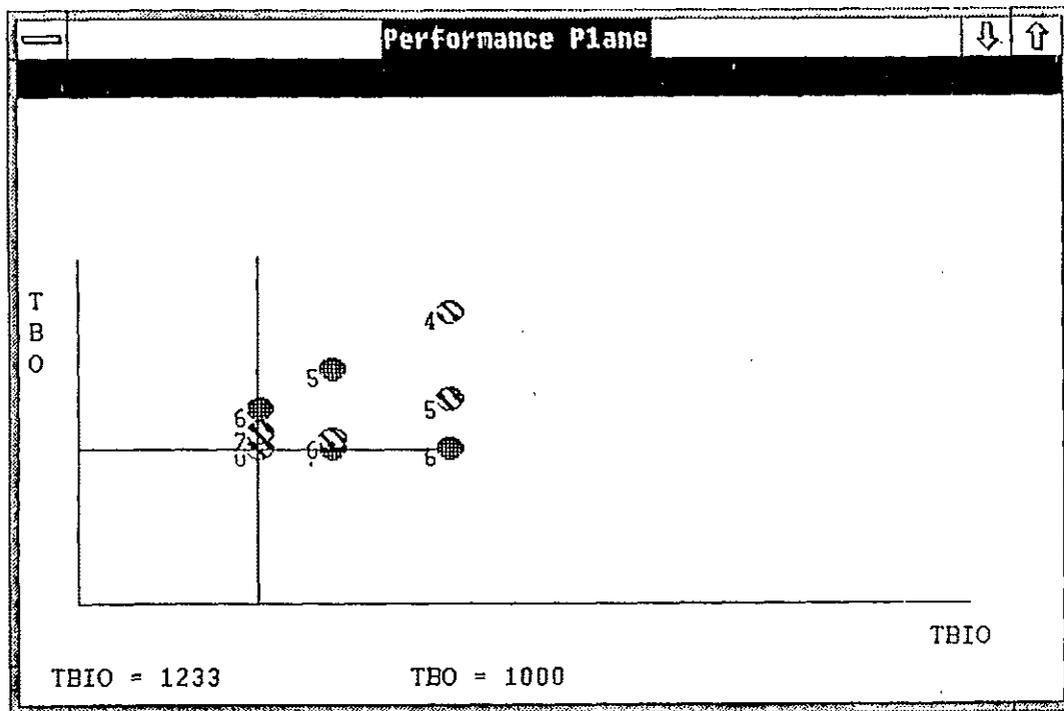
YBO	RESOURCE
1000	6
1310	5
1868	4

The window also features a minus sign on the left and up/down arrow icons on the right.

Figure 4.57. Resources Window for the Graph of Figure 4.51.

respectively, to operate at $TBIO_{LB}$. The Resources window is shown in Figure 4.57. The complete Performance Plane display window along with the Modify Table is presented in Figure 4.58. Each operating point selected in the Performance Plane corresponds to a different value of resource. The $R = 8$ point corresponds to the lower bounds of TBO and TBIO. The $R = 7$ point is selected to keep the value of TBIO at the lower bound at the expense of increasing the TBO value. The points corresponding to $R = 6$ and $R = 5$ are chosen to keep the value of TBO low, although the value of TBIO is increased. The point corresponding to $R = 4$ is chosen for operating with fewer number of resources. For the selected operating points, the Modify Table window displays the status of the control edges. If a control edge is active for a selected operating point, then a one is entered in the table corresponding to that particular control edge for the given value of R . Otherwise, a zero is entered in the table to show that the control edge is inactive for the selected operating point.

The throughput is 100 for $R = 6$. It is about 75 percent of the maximum for $R = 5$, and about 51 percent of the maximum value for $R = 4$. This information is presented in the Throughput display window of Figure 4.59.



R	TBO	TBIO
8	1000	1250
7	1100	1250
6	1060	1750
5	1310	2550
4	1868	2550

C-EDGE		RESOURCES				
FROM	TO	8	7	6	5	4
1	2	0	0	1	0	0
10	8	0	0	0	1	1
7	8	0	0	0	1	1

BUFFERS		
FROM	TO	SIZE
7	8	2
7	9	2

Figure 4.58. Performance Plane Display showing Modify Table.

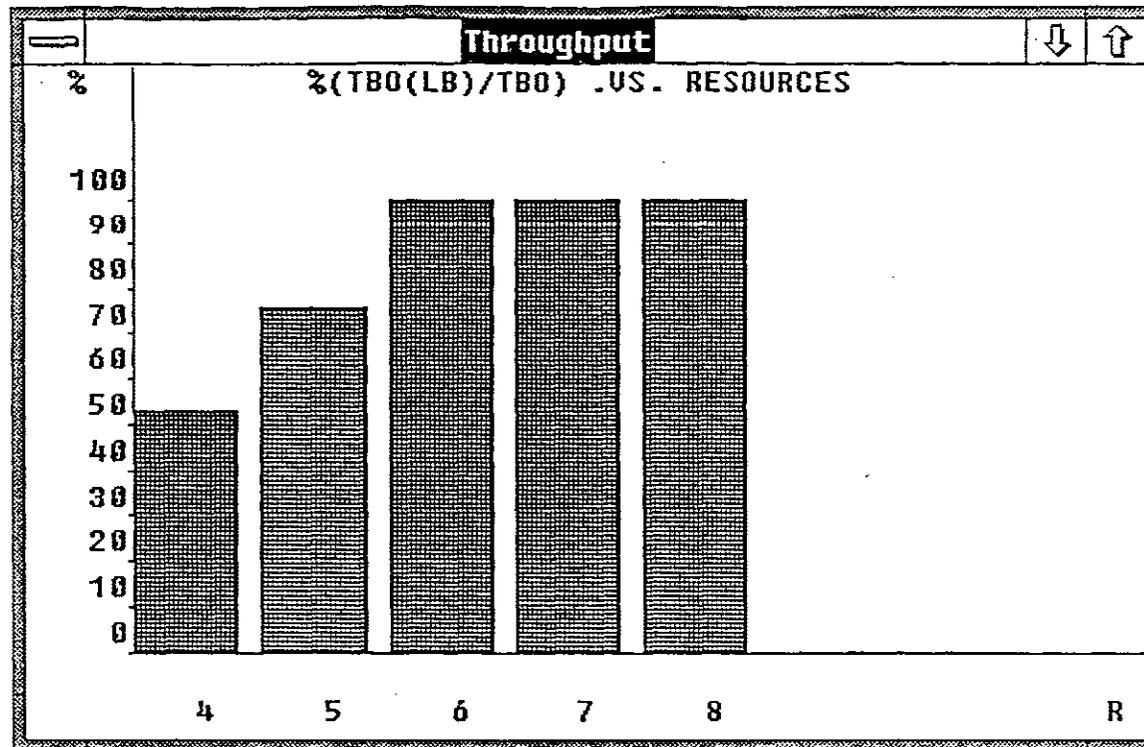


Figure 4.59. Throughput Display for the Graph of Figure 4.51.

CHAPTER FIVE

Conclusion

5.1 Summary

ATAMM, a new Petri net based model developed by researchers at Old Dominion University, is capable of describing the execution of large-grained algorithms on data flow architectures. The ATAMM model provides the necessary data and control flow dialog for predictable execution of an algorithm by a data flow architecture. The ATAMM model also facilitates the investigation of different algorithm decompositions without having to consider the hardware. For a selected hardware, the model can be used to match the algorithm requirements with the hardware capability in order to achieve optimum time performance.

In this thesis, a software design tool is developed to aid the user of ATAMM based distributed-processing system in the design process of selecting operating strategies. The ATAMM Design Tool is developed as an automated tool to make the processes of predicting time performance, resource requirements, and specifying operating points, simple and fast. Algorithms are developed for construction of displays representing graph play and resource utilization. Algorithms for automating generation of operating points in the performance plane are also developed as part of this thesis work. The tool is found to be very useful in determining the buffer requirements accurately for a given graph.

It aided in detecting those extra buffer requirements for the space surveillance algorithm which were not apparent to manual calculations.

As a demonstration of the application capabilities of the ATAMM Design Tool, case studies are performed on two real algorithms. One is the space surveillance algorithm and the other is the decomposed state equation for discrete linear systems. The results of these two case studies are presented through a set of ATAMM Design Tool window displays.

5.2. Topics for Future Research

There are several topics which are the subject of continuing and future research. First, the ATAMM Design Tool could be used to investigate several other real world algorithms and to evaluate their performance. Second, allowances could be made to include the non-ideal nature of algorithm play.

Currently, research is being done at ODU to extend the capabilities of the AMOS, and thus expand the class of problems to which the ATAMM rules can be applied. The enhanced AMOS is to be implemented in the Generic VHSIC Spaceborne Computer (GVSC), a spaceborne four processor breadboard which is also based on the VHSIC 1750A instruction set architecture. In this regard, the ATAMM model is being generalized to permit multiple concurrent instantiations of selected graph nodes. Also, the simultaneous play of multiple algorithm graphs, each having a distinct source node and sink node, is being developed. Three separate strategies for implementing multiple graphs are being considered. These strategies are referred to as the parallel execution

strategy, the time multiplexing strategy, and the priority interrupt strategy. The different strategies are selected to address different classes of problems which arise in real-time applications. Efforts are also being made to incorporate the features of fault-tolerance and branching in the ATAMM model. These enhancements will thus require future modifications to the ATAMM Design Tool. In addition, any other enhancements or refinements to the ATAMM model might provide additional interesting research topics for future consideration.

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, J.W. Stoughton, and S. 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 Computing Architectures," Academic Press, London, 1984.
- [4] S. Som, "Performance Modeling and Enhancement for the ATAMM Data Flow Architectures," Ph.D. Dissertation, Old Dominion University, Norfolk, Virginia, May 1989.
- [5] T. Agerwala, and Arvind, "Data Flow Systems," *Computer*, pp. 10-13, February 1982.
- [6] T. Murata, "Relevance of Network Theory to Models of Distributed/Parallel Processing," *Journal of Franklin Institute*, pp. 41-49, 1980.
- [7] R.R. Mielke, J.W. Stoughton, and S. Som, "Modeling and Optimum Time Performance for Concurrent Processing" NASA Technical Paper 4167, Grant NAG1-683, August 1988.
- [8] J.L. Peterson, "Petri Net Theory and the Modeling of Systems," Englewood Cliffs, NJ, Prentice Hall, 1981.
- [9] T. Murata, "Circuit Theoretic Analysis and Synthesis of Marked Graphs," *IEEE Transactions on Circuits and Systems*, vol. 24, pp. 400-405, July 1977.
- [10] 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.

- [11] J.W. Stoughton and R.R. Mielke, "Strategies for Concurrent Processing of Complex Algorithms in Data Driven Architectures," NASA Technical Paper 181657, Grant NAG1-683, February 1988.
- [12] R.L. Jones, "Diagnostics Software for Concurrent Processing Computer Systems," M.S. Thesis, Old Dominion University, Norfolk, Virginia, April 1990.
- [13] K. G. Lockyer, "An Introduction to Critical Path Analysis," Pitman Publishing Limited, London, 1969.
- [14] S. Som, J.W. Stoughton, and R.R. Mielke, "Performance Modeling in the ATAMM Data Flow Architecture," Technical Paper Presented at the *Ninth IEEE International Phoenix Conference on Computers and Communications*, Scottsdale, Arizona, March 1990.
- [15] R.R. Mielke, J.W. Stoughton, S. Som, R. Obando, M. Malekpour, and B. Mandala, "ATAMM Multicomputer Operating System Functional Specification" Progress Report Prepared for NASA for the period February 1990 - August 1990, Grant NCC1-136, August 1990.
- [16] Charles Petzold, "Programming Windows," Redmond, Washington, Microsoft Press, 1988.

APPENDIX
USER MANUAL FOR THE ATAMM DESIGN TOOL
TABLE OF CONTENTS

A.1 Overview

A.2 System Requirements

A.3 Using the ATAMM Design Tool

A.3.1 Window Management

A.3.2 Creating, Editing, and Saving a Graph

A.3.3 Viewing the Buffer Requirements for a Graph

A.3.4 Viewing the Bounds Window

A.3.5 Viewing the Displays for SGP and TGP

A.3.6 Viewing the SRE and the TRE Displays

A.3.7 Viewing the Resources and the Throughput Windows

A.3.8 Viewing the Performance Plane Display

A.3.9 Controlling the Cursors

A.3.10 Getting Help

A.3.11 Selecting Colors or Patterns

A.1 Overview

The ATAMM Design Tool is a software tool that provides an efficient means to predict the performance and resource requirements of an algorithm implemented in an ATAMM defined data flow architecture. The tool is capable of generating the Modify Table information which includes the input injection interval, the control edge settings, and the buffer size details for selected operating points in the Performance Plane. The Modify Table is generated for the ADM system to modify the graph for matching the resource availability. Performance evaluation is made possible by providing displays for SGP, SRE, TGP, and TRE. Also, displays for throughput, buffer size and resource variations are provided by the ATAMM Design Tool.

A.2 System Requirements

The ATAMM Design Tool program requires an IBM AT (286/386) or compatible, the Microsoft Windows 286/386 multitasking environment, and a mouse. The ATAMM Design Tool is divided into several modules, each occupying its own code segment. The largest code segments are moveable and discardable. This feature helps in saving memory. If the code segments are not discardable, then the entire system would require approximately 120k bytes of memory. But with the discardable feature, the system's code will manage with much less memory as it discards from the memory those code fragments which are not being used.

As long as the Microsoft Windows environment has the necessary driver files, any printer and monitor can be used with the system.

A.3 Using the ATAMM Design Tool

When the ATAMM Design Tool is executed, a small window (called as Design Tool) appears on the screen. This window is the main window for the program. The main window provides three different menus. These menus have options to open a graph file, create a new graph, and view various application windows. Also provided in the main window are the help files for each of the windows. The user can invoke any one of the options by clicking the left mouse button on that particular option. The user may exit the program anytime by selecting the "Exit" option from the main window menu or by closing the ATAMM Design Tool window from the system menu. A more convenient way of closing any window is to double-click in the system menu area, which is provided on the top left side corner of the window.

A.3.1 Window Management

The ATAMM Design Tool has several independent display windows. More than one window can be invoked at any time. These windows can overlap. This allows the viewing of more than one display simultaneously. More detailed information concerning the use of windows is available in [16].

A.3.2 Creating, Editing, and Saving a Graph

The Graph Editor window is invoked by selecting "Graph_Editor" option in the Design Tool window. The Graph Editor window is provided with four menus namely, "File", "Node", "Edge", and "Modify". The "File" menu has options for saving a graph

created in the Graph Editor window and exiting the window. When the "Save" option is invoked, a message box appears asking the user to type the file name. The program automatically assigns a ".gph" extension to the file name typed. The program does not accept any other file with a different extension.

The "Node" menu has the options to create source nodes, sink nodes and operator nodes. An option to specify the various node times is also provided. In order to create an item, the user must select the item from the menu. A check mark appears by the side of the selected item in the menu as soon as the user clicks on the item. After selecting an item, the user must click the left mouse button to create the item. The "Time" option in the menu allows the user to specify the times for the items created in the Graph Editor window. After selecting the "Time" option, the user must click the left mouse button on the item of interest. A message box appears asking the user to enter the time for that item. If the mouse button is clicked in an invalid region on the window, the program gives a message beep.

The "Edge" menu provides the options to draw regular edges and control edges. Options to create buffers and tokens on the edges are also provided. After selecting the "Edge" option, the user must click the left mouse button on an item to specify the initial point for an edge. To specify the terminal point, the right mouse button must be clicked. A message beep is given if the mouse button is clicked in an invalid region. The program allows an edge to have as many as three segments. The left mouse button must be clicked for creating a segment. After creating two segments, the edge must be ended by clicking the right mouse button. Otherwise, a message beep is given. The user must

click the left mouse button on the small square boxes on the edges in order to vary the number of tokens and the number of buffers. A control edge is drawn like a regular edge. But, a control edge appears in red color and a regular edge appears in black color.

The last menu in the Graph Editor is the "Modify" menu. This menu consists of options to delete any item drawn in the Graph Editor. The user must click the left mouse button on the item selected in order to delete that item. An edge is deleted by clicking in the small square box on that edge.

A.3.3 Viewing the Buffer Requirements for a Graph

Whenever the Graph Editor window is opened, the user has to execute the Buffer window before opening any other window. The Design Tool window has an option, namely "BufferSize", to open the Buffer window. This window displays the number of initial buffers required on an edge, if that value is more than the default value of one. Otherwise, the window displays the message "No Extra Buffers Required". There are no menu options provided in the Buffer window.

A.3.4 Viewing the Bounds Window

The Bounds window is invoked from the Design Tool window by selecting the "Bounds" option. The Bounds window displays the earliest start time (ES), the earliest finish time (EF), the latest start time (LS), and the latest finish time (LF). The window also displays values of TCE, $TBIO_{LB}$, and TBO_{LB} . In addition, a list of critical paths in the graph is displayed. The Bounds window has no menu option.

A.3.5 Viewing at the Displays for SGP and TGP

The "SGP" and "TGP" options in the Design Tool Window allow the user to view the Single Graph Play and the Total Graph Play displays respectively. Both these windows are provided with menu options. The option "Float" in the menu enables the user to look at the float times associated with the nodes. The "Test" option provides a means to view the node test times. The SGP and the TGP windows are provided with vertical cursors to note the time. In addition, the TGP window has a cursor displayed in red color which indicates TBO_{LB} in default position. The red colored cursor is moved by pressing the left mouse button and the shift-key simultaneously.

A.3.6 Viewing the SRE and the TRE Displays

The SRE display window is opened by selecting the "SRE" menu option from the Design Tool window. The TRE display window is invoked by choosing the "TRE" option. The SRE and the TRE windows are provided with the option "IncludeTest" to include test times in the count of required resources. Vertical cursors in both the SRE and the TRE windows give the user an opportunity to note the time at any selected position in the respective windows.

A.3.7 Viewing the Resources and the Throughput Windows

The Resources window is opened from the Design Tool window by selecting the "Resources" option. This window performs a considerable amount of computations internally and hence there is a time delay before the window is displayed if it is invoked

the first time and if the Throughput or the Performance Plane windows are not invoked previously. The reason is that the Resources window, the Throughput window, and the Performance Plane window make use of the same routine to perform the resource variation calculations. The routine involves an iterative search process which consumes a certain amount of time. The time step for the search process is equal to two percent of the time difference between T_{BIO} and $T_{BO_{LB}}$. But if any one of the Resources, the Throughput, or the Performance Plane windows calls the common routine once, the results of the computations are readily made available to the other two windows. The computations are repeated only if any modifications are made in the Graph Editor window.

The Resources window has a menu option "IncludeTest", which allows the user to include the test times in the search for resource variations. The Design Tool window has an option "Throughput" to open the Throughput window. There are no menu options in the Throughput window.

A.3.8 Viewing the Performance Plane Display

The Performance Plane window displays the operating points in terms of T_{BO} , T_{BIO} , and R (resources). The user must select the "PerformancePlane" option from the Design Tool window in order to view the operating points. The Performance Plane window is provided with cross-hair cursors and a menu called "Select". The user may select an operating point in the Performance Plane by choosing the "SelectPoint" option and then clicking the right mouse button on the operating point of interest. The color of

the selected operating point changes to yellow. The Modify Table for the selected operating points may be viewed by selecting the "ModifyTable" option. This displays the Modify Table as a child window of the Performance Plane display window. The child window is displayed within the parent window, which in this case is the Performance Plane display window. The child window can be resized and closed like any other overlapped window.

A.3.9 Controlling the Cursors

Selecting the "Split" option from the menu creates two time cursors that measure time differentials. The time difference between the two cursors is displayed in parentheses at the bottom of the window. Clicking the left mouse button at the point of interest moves the left most time cursor to this new location. The right most time cursor is moved by clicking the right mouse button. In order to merge the two cursors, the "Split" option must be selected again. When the two time cursors are active in a window, the user is provided with the opportunity to look at an enlarged view of the display bounded between the cursors by selecting the "Slice" option. Selecting the "Total" option returns the view of the normal display.

A.3.10 Getting Help

A help window providing general help, help on SGP, TGP, SRE, TRE, Bounds, Bufferize, Graph Editor, Resources and Performance Plane is available in the ATAMM Design Tool. Help on any desired window is provided on selecting the related option from the "Help" menu in the Design Tool window.

A.3.11 Selecting Colors or Patterns

A printout of all the Design Tool window displays is possible through a screen capture technique within the windows environment. It is possible to switch from colors to black and white patterns for the purpose of printing. Selecting the "Pattern" option from the Design Tool window "paints" the graphic images with patterns instead of colors. Selecting this option again allows the use of colors once again. The printing procedure involves capturing the display through the "snap" program provided by the Software Development Kit and pasting the captured image in the "paint" program. It is a straight forward procedure to print from the "paint" program.