Old Dominion University

ODU Digital Commons

Spring 1993

A Performance Prediction Model for a Fault-Tolerant Computer During Recovery and Restoration

Rodrigo A. Obando
Old Dominion University

A PERFORMANCE PREDICTION MODEL FOR A FAULT-TOLERANT COMPUTER DURING RECOVERY AND RESTORATION

by

Rodrigo A. Obando
B.S. December 1978, Instituto Tecnológico de Costa Rica
M.E. December 1987, Old Dominion University

A Dissertation submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
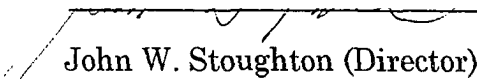Requirement for the Degree of

DOCTOR OF PHILOSOPHY

IN

ENGINEERING

OLD DOMINION UNIVERSITY

March, 1993

Approved by:

John W. Stoughton (Director)

ABSTRACT

A PERFORMANCE PREDICTION MODEL FOR A FAULT-TOLERANT COMPUTER DURING RECOVERY AND RESTORATION

Rodrigo A. Obando

Old Dominion University, 1993

Director: Dr. John W. Stoughton

The modeling and design of a fault-tolerant multiprocessor system is addressed in this dissertation. In particular, the behavior of the system during recovery and restoration after a fault has occurred is investigated. Given that a multicomputer system is designed using the Algorithm to Architecture To Mapping Model (ATAMM) model, and that a fault (death of a computing resource) occurs during its normal steady-state operation, a model is presented as a viable research tool for predicting the performance bounds of the system during its recovery and restoration phases. Furthermore, the bounds of the performance behavior of the system during this transient mode can be assessed. These bounds include: time to recover

from the fault (t_{rec}), time to restore the system (t_{res}) and whether there is a permanent delay in the system's Time Between Input and Output (TBIO) after the system has reached a steady state. An implementation of an ATAMM based computer was developed with the Generic VHSIC Spaceborne Computer (GVSC) as the target system. A simulation of the GVSC was also written based on the code used in ATAMM Multicomputer Operating System (AMOS). The simulation is in turn used to validate the new model in the usefulness and accuracy in tracking the propagation of the delay through the system and predicting the behavior in the transient state of recovery and restoration. The model is validated as an accurate method to predict the transient behavior of an ATAMM based multicomputer during recovery and restoration.

DEDICATION

To my wife, Jacquelline, for believing in me and for her support and understanding; and to my daughter and son, Jacquelline and Rodrigo, who gave up many hours with their Dad. And to my mom, Trinidad, who always believed in me.

To my Lord, Jesus Christ, the Son of God; to Him be the Glory, and the Honor for ever and ever, Amen.

ACKNOWLEDGEMENTS

I want to thank my advisor, Dr. J. W. Stoughton for his support during the research and writing of this dissertation. I also thank the members of my dissertation committee, Dr. R. R. Mielke, Dr. D. L. Livingston and Dr. L. Wilson for their comments, suggestions and help on my dissertation.

I also thank the personnel at NASA Langley Research Center for their support in the investigation of this subject. Special thanks to Paul Hayes and Rob Jones. Thanks to Asa Andrews and Dr. Sukhamoy Som.

I want to thank James O. Crawford, Jr. for his help in getting it finished and printed.

TABLE OF CONTENTS

Chapter/Section   **Page**

iii

iv

v

LIST OF FIGURES

viii

LIST OF TABLES

ix

CHAPTER ONE

INTRODUCTION

This dissertation addresses the modeling and design of a fault-tolerant multiprocessor system. In particular, the behavior of the system during recovery and restoration is investigated. Fault-tolerant computing has become an increasingly important facet of real-time computing systems. Real-time computing is a rapidly growing field in its own right. The demand for real-time computing systems in industries such as aeronautics and defense has produced an increase in research in this area. A real-time computing system designer cannot afford to omit fault-tolerant capabilities in systems designs. It is not realistic to consider that the system will never fail. When a real-time system requires that the output meet deadlines with high reliability, then the system has to be fault-tolerant.

A computing system with fault-tolerant capabilities has to deal with recovery and restoration of the system after an error has been detected. A fault-tolerant computer requires redundancy of some kind to recover from a fault. Such redundancy may be temporal or hardware based. An example of temporal redundancy is the process of *retry* after a failure of a task. An

1

example of hardware redundancy is the use of multiple computing resources to perform the same task concurrently [1], [2], [7].

A real-time computing system with fault-tolerant capabilities has greater requirements than other computing systems. One of these requirements is that in addition to recovering from a fault it must deliver its output before or on a given deadline. Meeting such requirements is critical in the design and implementation of real-time fault-tolerant systems. A fault in any computing system causes the system to go through a transient in its behavior. A fault-tolerant system is designed to prevent any further damage in the system and to repair the damage that has been done. A system without fault-tolerant facilities is doomed to fail to perform to specifications if a fault occurs. A fault-tolerant real-time computing system, on the other hand, should be able to go through the transient meeting also the data deadlines imposed on the system. Determining the system feature of transient behavior is crucial in the design of fault-tolerant real-time computing systems [3].

Prediction of performance during a transient phase, such as that caused by a fault, requires first the prediction during the steady-state phase of the system. Fortunately, there exists a model that allows such a prediction: the Algorithm To Architecture Mapping Model (ATAMM) [4], [5], [6] developed at Old Dominion University. The development of ATAMM has made performance prediction of a class of multiprocessor

systems possible. Steady-state performance prediction is possible for systems designed to follow the ATAMM design guidelines. It is possible to calculate bounds for such measures as throughput (TBO: Time Between Outputs) or system delay (TBIO: Time Between Input and Output).

1.1 Fault-Tolerant Computing

One taxonomy of the phases that a fault-tolerant computing system goes through is error detection; damage confinement and assessment; error recovery; and fault treatment and continued system service [1]. An error is the deviation from specifications caused by a fault and can be detected by hardware mechanisms or software procedures. Damage confinement and assessment is the process of reducing the spread of damage and estimating its extent in the system. The kind of process to be used depends on the type of fault detection used. If the damage is beyond repair it may be necessary to restart the system from its initial state. After having assessed the damage, recovery from the damage needs to take place. Error recovery is a very important aspect of fault-tolerant computing since the system depends on a proper mechanism to recover the lost work to continue reliably. The phase of fault treatment and continued service is used to locate the fault and remove it and to leave the system in a healthy state. After this stage the system may continue normal operation until another fault occurs [1], [2], [7].

1.2 ATAMM Context

Of interest is the behavior of a multiprocessor system operating periodically on a set of inputs. The application of ATAMM has been limited to large grain, decision-free algorithms. The number of computing resources the system contains is on the order of twenty. The communication-to-computing effort ratio is small.

It is of interest to use the ATAMM model to design fault-tolerant computers. A system designed to follow the ATAMM rules is a good candidate for a fault-tolerant computer since the model predicts how the system should be changed to continue after a failure of a computing resource. The system can be designed to follow a given performance path from n computing resources to 1 computing resource. For every number of computing resources, a different performance operating point can be obtained. In a graceful degradation scenario, ATAMM lends itself to predict a particular degradation path, optimized for the application at hand [4], [5], [6].

1.3 Current Research Areas

Three of the current research areas in fault-tolerant computing are faulty processor detection, performance prediction in a fault-tolerant system, and estimation of software reliability.

Faulty processor detection in a multiprocessor system can be accomplished in several ways. A method that has been used for many

years and has been the subject of much research is the testing of

computing processors by other computing processors [9], [10], [11], [12],

[13], [14], [15]. This method involves the use of the computing processors

themselves testing other computing resources in the system or a master

controller that tests all processors. The number of tests that are necessary

to test a given system is an issue of interest. Other issues are the topology

of the test graph to test effectively all computing resources and the validity

of tests given that a computing resource is identified as faulty by others

that may not all be healthy. For these methods to succeed, it is necessary

to design test algorithms for specific systems or architectures and

sometimes they are tied very closely to the interconnection network of the

computing resources. This approach is based in the work described in [10]

which is deterministic in principle. A more recent procedure of diagnosing

faulty processors is explained in [25]. This procedure deals with arbitrarily

connected processors providing faulty processor diagnosis for a wider

variety of connection networks. This approach is not deterministic but

involves probability theory and, therefore, provides diagnosis with high

probability instead that probability one as the methods directly derived

from [10].

Performance prediction in a fault-tolerant real-time system has been

addressed by Kant [16]. His model assumes that there are recovery blocks

(RB) and N-version programming (NVP) [17] involved in the design of the

system. The model is a hierarchical one. Starting from the top level of the program or main routine, it decomposes the system into different levels of complexity. The top level is designated as 1 and the subsequent levels are labeled in increasing order. This model is stochastic and also addresses some reliability measures of the system. It assumes a separate master processor running the supervisor procedure. The analysis is performed on the processes of the system instead of the processors. The processors are assigned to processes but some processes can be left dormant while other processes are spawned in the system. This model specifically addresses the software reliability of the system and not explicitly that of the hardware.

The model mentioned above is based on N-version programming [17] which assumes that the different software versions are uncorrelated, i.e., statistically independent. A recent technique that addresses a correlated set of software versions is explored in [26]. This technique uses the mutation analysis to create a data set to test software modules. Mutation analysis refers to the mutation of software modules. The idea is based on the fault-injection techniques used to test hardware. Mutation is performed by artificially introducing software errors in a module. A test data set is produced that identifies the mutant modules as faulty. The underlying assumption is that if a mutant module is not identified as faulty, a potential software fault in in the original module. This method has proven to provide better reliability than the N-version programming.

1.4 Research Objective

The restrictions that apply to a real-time, fault-tolerant multicomputer system make this class of systems stand out from the rest due to the delicate balance of performance versus reliability. Increasing the system's performance may impair its reliability and vice versa. The models in the current research areas do not address the question of whether a system ever restores to a state within specifications. This question is important to real-time systems since a state that is reached after recovery from a fault may be operational but not within the system specifications. An example may be a computer which provides control for a given system. After a fault occurs in the system, the computer performs the required fault-tolerant phases and may continue evaluating input data and delivering output data, but the system delay may not be within the specified limits. This is an undesirable characteristic since, although the system may take care of the damage caused by a fault, it may never reach the desired operating point or state.

The problem domain of interest is that of systems executing single-input single-output graphs, with or without recursive circuits, designed under ATAMM. It is assumed that the system processes multiple data packets, i.e., input data is presented to the system periodically. The systems have a time limit to deliver their outputs; therefore, they are considered real-time systems. It is also assumed that only one computing

resource fails at a time and that no other computing resource fails before the system fully recovers. The latter suggests that the system is fault-tolerant to some extent.

The application of ATAMM to real-time computing systems has opened the possibilities of predicting the requirements of a multiprocessor system to meet data deadlines. Under steady-state conditions, performance of an algorithm processing repetitive input data sets can be predicted. By the argument explained above, there is a need for fault-tolerant capabilities for a highly reliable real-time system. Therefore, a system designed under the ATAMM rules should have fault-tolerant capabilities added. If this is the case, the issues at stake are the behavior of the system upon the occurrence of a fault; the time it takes to recover and restore the system; and whether or not the system still meets the deadlines it was designed for after the fault. Addressing these and other issues are crucial in the design of fault-tolerant, real-time computing systems [3].

An analysis procedure is necessary to predict performance during transients due to faults and to define a minimum set of requirements of a fault-tolerant real-time system designed to follow the ATAMM rules. However, the ATAMM model has not been previously used to predict performance during transient phases, although it has been used successfully to predict performance in the steady-state phase of the system.

A candidate model and an analysis procedure are presented in this dissertation.

Given that a multicomputer system is designed to comply with the ATAMM design guidelines [8], and that a fault (death of a computing resource) occurs during its normal steady-state operation, a model and an analysis procedure are proposed as candidates to predict the performance bounds of the system during its recovery and restoration phases. Furthermore, the system requirements to comply with such bounds can be assessed and system design specifications can be gathered. The bounds are time to recover from the fault (t_{rec}), time to restore the system (t_{res}), and permanent or temporary delay of the output from its expected time.

The time to recover from the fault is related to the time to restart the process or node that was not completed due to the fault occurrence. The time to restore the system is the time from when the error is detected to the time that the system reaches the target operating point. In the case of the experiments presented in this dissertation, the target operating point is the same as the operating point before the fault. The bound of temporary or permanent delay injected to the system output is related to the bound of time to restore the system. If the system is not able to restore to the target operating point then the delay injected to the system is permanent and vice versa.

1.5 Dissertation Organization

The fundamental model and analysis procedure along with the necessary theoretical background are presented in Chapter Two. This model addresses the introduction and propagation of delay in the system under study. An overview of the implementation of an ATAMM operating system with the fault-tolerant attributes to test the model developed in Chapter Two is explained in Chapter Three. This chapter is used to expound the implementation of ATAMM Multicomputer Operating System (AMOS) in the IBM Generic VHSIC Spaceborne Computer (GVSC). Experiments to demonstrate the use of the model and the validity of the ATAMM implementation are presented in Chapter Four. These experiments are intended to compare the experimental results against the calculated results obtained according to the procedures developed in Chapter Two. Chapter Five contains conclusions and suggestions concerning future research.

CHAPTER TWO

THEORY

The objective of this chapter is to extend the ATAMM model to

investigate the transient behavior a multicomputer system subject to a

fault and during the recovery process. Of interest is how delay introduced

by a node is propagated in the graph. When a computing resource fails

while working in a node in the graph, it is detected and the node is

reassigned to a healthy computing resource. The effect of this error

detection and reassignment of the node is to delay the output from the

faulty node. The propagation of this delay through the graph is examined

and a model is developed in this chapter. The extension to the ATAMM

graph model to study this behavior is presented in Section 2.1. Node model

definitions are presented in Section 2.2. The fault node model to describe

the introduction of delay in the graph is developed in Section 2.3. The

delay propagation model to describe how the delay propagates in the graph

is developed in Section 2.4. Section 2.5 contains general purpose

definitions. A summary of the chapter is presented in Section 2.6.

11

2.1 Graph Model

The ATAMM model consists of the Algorithm Marked Graph (AMG) and the Computational Marked Graph (CMG). These graphs describe the data dependency of nodes in an algorithm. The AMG is a dataflow description of the algorithm and does not show control flow. The CMG describes both data flow and control flow [6]. The CMG is constructed with the use of another graph that describes the internal behavior of a node in the AMG. This graph is called the Node Marked Graph (NMG).

The AMG is described by two sets, a set of nodes N and a set of directed edges E. The set of nodes N is

$$N = \{n_i\} \text{ , for } i = 1 .. k$$

where k is the number of nodes in the graph. The set of directed edges E is

$$E = \{e_{i,j}\} \text{ , for } i,j = 1 .. k$$

where $e_{i,j}$ is a directed edge from initial node i to terminal node j.

The CMG is constructed using the AMG and the NMG according to [4], [7], and [27]. These graphs are used to obtain performance bounds as explained below.

There are two performance bounds derived from this graph model. These bounds are the maximum system throughput or Lower Bound of Time Between Outputs (TBO$_{LB}$) and the minimum system delay or the

Lower Bound of the Time Between Input and Output (TBIO$_{LB}$) [5]. The first bound Time Between Outputs (TBO) refers to the minimum time between outputs at which a given algorithm graph is capable of working. This indicates the minimum Time Between Inputs (TBI) at which a graph should be driven. This bound is calculated by finding the circuit with the largest amount of time per token. This is done by first finding all circuits in the CMG. For each one of the circuits, the execution times in the circuit are added, and the sum is divided by the number of tokens in the circuit. The second bound (TBIO) refers to the time that a data packet or input takes to be transformed by the algorithm and reach the output sink. This bound is calculated by finding all paths from the source to the sink and adding the execution times in each one of them. The path with the largest time defines TBO$_{LB}$.

An AMG shows the data flow and data dependency among the computational nodes but it does not explicitly show data packet interdependency. Every node processes every input datum that is presented for every data packet that is input to the graph. The AMG explicitly demonstrates the data dependency of one data packet; it displays the different stages or transformations a data packet goes through until it is delivered at the sink node. Ideally, each node finishes executing in a fixed amount of time. However, in the event of a node requiring a longer amount of time, a transient occurs.
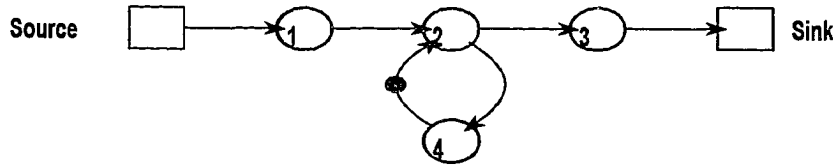
Figure 2.1. Example of an AMG

2.1.1 XAMG

The following graph model is an extension of an AMG and is called an

eXtended Algorithm Marked Graph (XAMG). This graph model uncovers

the relationship that exists between data packets. Every node in this

graph is associated with one and only one data packet. The XAMG can be

obtained from the AMG by indexing the nodes of the AMG with the data

packet number.

The XAMG is described by two sets. The set of nodes N^X and the set

of directed edges E^X. The set of nodes N^X is

$$N^X = \{n_{p,i}\} \text{, for } p = 1 \mathinner{..} l, i = 1 \mathinner{..} k,$$

where l is the number of data packets and k is the number of nodes in the

set of nodes N that describes the AMG from which the XAMG is obtained.

The total number of nodes in N^X is the product of l and k. The set of

directed edges E^X is

$$E^X = \{e_{p,i,n,j}\} \text{, for } p,n = 1 \mathinner{..} l; i,j = 1 \mathinner{..} k$$

where the edge $e_{p,i,n,j}$ is a directed edge from node $n_{p,i}$ to node $n_{n,j}$.

Therefore, node X in the AMG becomes an array of nodes indexed by the data packet number i, so X becomes X_i for all i in the XAMG. To illustrate this, the AMG in Figure 2.1 can be transformed into the XAMG in Figure 2.2. The latter graph shows the relationship between the data packet i and the data packets $i + 1$, $i + 2$, $i + 3$ and so on.

The XAMG model requires the redefinition of some measures of the AMG such as TBIO and the addition of new measures relating to the input and output of different data packets. These measures will be presented in Section 2.5. One feature worth mentioning here is that there are no directed circuits in the XAMG, therefore most of the analysis is carried out on directed paths instead. In fact, every circuit in the AMG unfolds as a path that goes from data packet to data packet.

2.1.2 XCMG

An immediate extension of the XAMG is the XCMG (eXtended Computational Marked Graph). The XCMG is an unfolded view of the CMG and it provides a way to account for interpacket data and control relationships. The XCMG does not have directed circuits as the CMG does. For example, the directed circuit in every computational node of a CMG does not exist in a XCMG. This directed circuit is transformed into a directed path from the write transition for node X, data packet i to the read transition of node X, data packet $i + 1$, and so on. The XCMG is obtained

similarly as in [27]. The XCMG related to the XAMG in Figure 2.2 is
shown in Figure 2.3.
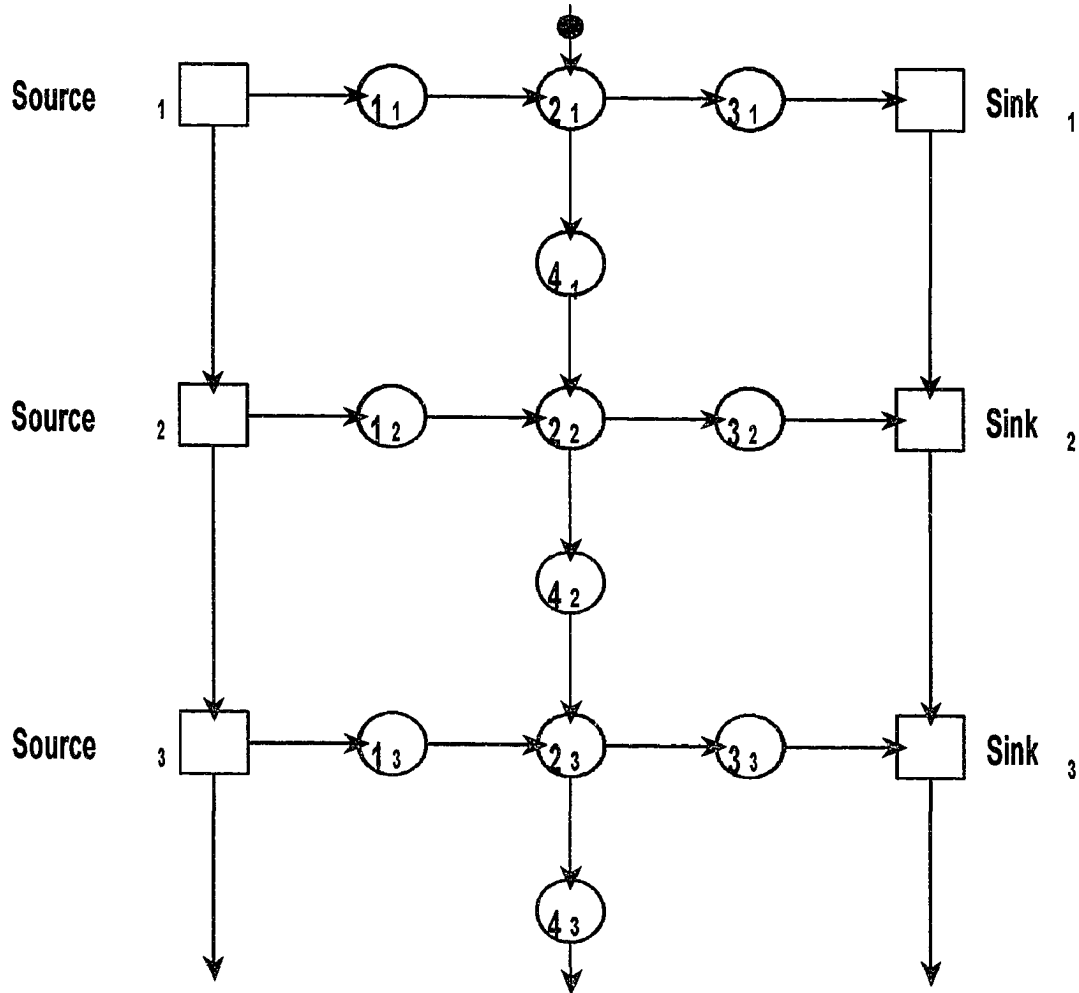
Figure 2.2. Example of an XAMG
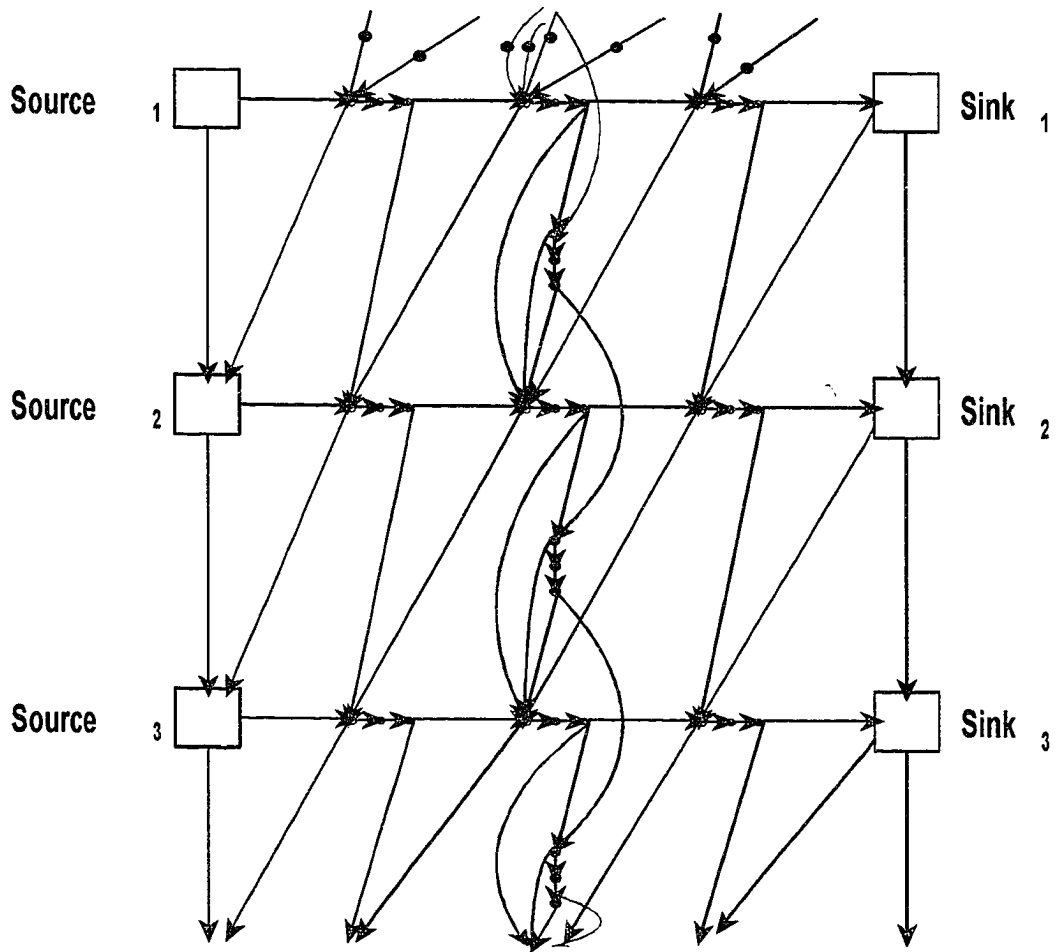
Figure 2.3. Example of an XCMG

2.2 Node Model Definitions

Of interest is the development of fault models for a system designed with the ATAMM strategy. These models can be used to analyze the effect of a fault on the system at run-time. The only fault that is considered in this analysis is the death of a resource during the execution of a node.

Consider a node in an XAMG with index n in data packet p. This node has two sets of edges associated with it. These sets are the input edge set $I_{p,n}$ and the output edge set $O_{p,n}$, where and are given by

$$I_{p,n} = \{e_i \text{ is an input edge to } N_{p,n}\}$$

and

$$O_{p,n} = \{e_i \text{ is an output edge to } N_{p,n}\}$$

respectively. Assume that the read and write times are zero, so that the only time associated with a node is the process period $p_{p,n}$, which is the time necessary to execute the node.

When fired, node $n_{p,n}$ encumbers tokens from all edges in $I_{p,n}$ and enters the execution stage of the node. The time when this is done is represented by the fire time $f_{p,n}$. The node $n_{p,n}$ deposits tokens in all edges in $O_{p,n}$ after the execution of the node has been completed. This is represented by the deposit time $d_{p,n}$. By using the ATAMM design

procedure it is possible to ensure that nodes fire as soon as all edges in $I_{p,n}$ have tokens. Thus the node is enabled when the last token arrives at the input edges. The fire time can be calculated by

$$f_{p,n} = max\{d_{q,i}, d_{q,j}, d_{q,k}, ...\},$$ (2.1)

where $d_{q,i}, d_{q,j}, d_{q,k}, ...$ are the deposit times of the nodes $n_{q,i}, n_{q,j}, n_{q,k}, ...$ which are the predecessor nodes to $n_{p,n}$.

Each node $n_{p,n}$ in the graph has a predetermined execution time $p_{p,n}$. Therefore, the deposit time $d_{p,n}$ can be expressed by

$$d_{p,n} = f_{p,n} + p_{p,n}.$$ (2.2)

The $max\{\}$ operator in the expression for $f_{p,n}$ indicates that tokens in the input edges may stay in the edges for a finite period of time before they are encumbered. Only the last token to arrive will stay for zero amount of time in its edge. The time a token stays in an edge can be represented by $\tau_{q,m,p,n}$. This expression represents the lifetime of a token in the edge between node $n_{p,n}$ and node $n_{q,m}$ and is calculated by

$$\tau_{q,m,p,n} = f_{p,n} - d_{q,m}.$$ (2.3)

2.3 Fault Node Model

In a very broad sense, computing systems may be classified into two categories. There can be systems without fault detection and there can be fault-tolerant systems, or systems with an error detection mechanism. When a computing resource fails while executing a node, the node is called faulty node. A faulty node requires different models for these two types of computing systems mentioned above.

In a system without fault detection, if a resource is executing a node and it fails prior to the deposit of the node output tokens, the process period $p_{p,n}$ can be estimated to be infinite. Thus, if $p_{p,n}$ is infinite, then $d_{p,n}$ is also infinite. Since there is no fault detection in the system, this event of a resource failure will lock the graph because successor nodes to $n_{p,n}$ will not receive input data. Hence, the model for a faulty node in a system without fault detection is straightforward: the process period $p_{p,n}$ for the faulty node is estimated to be infinite.

In a fault-tolerant system, if a resource is executing node $n_{p,n}$ and a failure occurs prior to the deposit of the node's output tokens, it is desirable that the process period $p_{p,n}$ and hence $d_{p,n}$ be finite. Since this is a fault-tolerant system, a fault detection mechanism is required to flag the fault with attendant application of fault-recovery techniques. One of these techniques is rollback which involves the restarting of the task that failed

to complete. Another technique is to discard the process of that task and declare a failure of the entire data packet. In this research it is desired that there is to be no loss of data, thus only the first technique is of interest. Once a node is restarted with a healthy resource and assuming there are no more faults in the system, the node will complete its process in time which will exceed the original process period $p_{p,n}$. The resultant time can be calculated by adding a delay to the original process period $p_{p,n}$.

This delay is represented by $\Delta^p_{p,n}$, and it is defined by

$$\Delta^p_{p,n} = p_{p,n} - p^*_{p,n} \tag{2.4}$$

Assuming there are no faults in a system, the fire time $f_{p,n}$ and deposit time $d_{p,n}$ for every node may be determined. These times are denoted by $f^*_{p,n}$ and $d^*_{p,n}$, respectively. However, if there is a delay in a node, i.e., it is a faulty node, there is a difference between $d_{p,n}$ and the actual $d^*_{p,n}$. This difference is called the delay to finish $n_{p,n}$, $(\Delta^d_{p,n})$, where

$$\Delta^d_{p,n} = d_{p,n} - d^*_{p,n} \tag{2.5}$$

In conclusion, a faulty node can be modeled by expressing its deposit time by considering the delay added to its execution time. Therefore a faulty node can be characterized by the delay $\Delta^d_{p,n}$.

The delay to fire node $n_{p,n}$ may be defined by

$$\Delta^f_{p,n} = f_{p,n} - f^*_{p,n},$$ (2.6)

where $\Delta^f_{p,n}$ is called the delay to fire node $n_{p,n}$.

If there is no delay introduced in node $n_{p,n}$, $\Delta^d_{p,n}$ can be calculated by

$$\Delta^d_{p,n} = d_{p,n} - d^*_{p,n} = f_{p,n} + p_{p,n} - f^*_{p,n} - p_{p,n} = f_{p,n} - f^*_{p,n} = \Delta^f_{p,n}$$

or

$$\Delta^d_{p,n} = \Delta^f_{p,n}.$$ (2.7)

Therefore, a node that does not introduce any delay propagates the delay that is applied to the firing of the node.

If a node introduces a delay $\Delta^p_{p,n}$ in the process, then the delay to deposit $\Delta^d_{p,n}$ is

$$\Delta^d_{p,n} = d_{p,n} - d^*_{p,n} = f_{p,n} + p_{p,n} + \Delta^p_{p,n} - f^*_{p,n} - p_{p,n} = f_{p,n} - f^*_{p,n} + \Delta^p_{p,n} = \Delta^f_{p,n} + \Delta^p_{p,n}$$

or

$$\Delta^d_{p,n} = \Delta^f_{p,n} + \Delta^p_{p,n},$$ (2.8)

where $\Delta^p_{p,n}$ is the delay introduced in the process time.

The general assumption made in the treatment of faults in this document is that there is only one fault at a time. This precludes the possibility of $0 < \Delta^f_{p,n}$ when $\Delta^f_{p,n} < \Delta^d_{p,n}$. Therefore, for a node that

introduces a delay, the delay to deposit is restricted to

$$\Delta^d_{p,n} = \Delta^p_{p,n}. \tag{2.9}$$

The node in which the delay is introduced is called the delay generator

node where

$$\Delta^d_{p,n} > \Delta^f_{p,n} \tag{2.10}$$

and any other node is a delay propagator node, such that

$$\Delta^d_{p,n} = \Delta^f_{p,n}. \tag{2.11}$$

2.4 Delay Propagation Model

Without considering the loss of the resource, the only damage

introduced in a system after a fault is the delay $\Delta^d_{p,n}$. Although this delay

may be considered local to the node $n_{p,n}$, it may also affect other nodes in

the system. For example, if $d_{p,n}$ determines when $n_{q,m}$ fires, then a delay

in $d_{p,n}$ may cause a delay in the firing of $n_{q,m}$. It is important to consider

how delay introduced by a faulty node propagates in a graph. A model of

delay propagation in a graph is presented in this section.

2.4.1 Fire-Equivalent Node Model

A fire-equivalent node model is developed in this section to facilitate

modeling of delay propagation through directed paths. Consider a node

$n_{p,n}$ with one input edge on the directed path of interest. The time to fire this node is

$$f_{p,n}^* = max\left\{d_{q,i}^*, d_{q,j}^*, d_{q,k}^*, \cdots\right\}. \tag{2.12}$$

It is assumed that only the edge associated with $d_{q,i}^*$ is in the directed path of interest. By the properties of the $max\{\}$ operator, the set of input edges $I_{p,n}$ can be partioned such that

$$f_{p,n}^* = max\left\{d_{q,i}^*, max\left\{d_{q,j}^*, d_{q,k}^*, \cdots\right\}\right\} \tag{2.13}$$

$$= max\left\{d_{q,i}^*, d_{q,x}^*\right\} \tag{2.14}$$

where

$$d_{q,x}^* = max\left\{d_{q,j}^*, d_{q,k}^*, \cdots\right\}. \tag{2.15}$$

This allows the time to fire the node to be decomposed into two terms. One term representing the directed path of interest and another representing the remaining input deposit times by the maximum of predecessor nodes deposit times. Let $d_{q,i} \geq d_{q,i}^*$, so

$$f_{p,n} = max\left\{d_{q,i}, d_{q,x}^*\right\} \tag{2.16}$$

where

$$f_{p,n} \geq f_{p,n}^* \tag{2.17}$$

Thus the time to fire a node on a directed path from the faulted node is potentially delayed.

2.4.2 Delay Propagator Nodes and Delay Absorbant Edges

How delay is introduced in a system is discussed in Section 2.3. It has been shown that the simplest element that propagates delay is a node. This section is used to present how an edge and a node propagate delay. It is shown that whereas a node propagates delay, an edge may absorb all or part of the delay introduced or propagated by a predecessor node. In Section 2.4.2, it is found how delay is propagated through the graph. These developments eventually are extended to the entire graph in the next sections.

It has been defined that if a delay propagator node $n_{p,n}$ is such that

$$\Delta^d_{p,n} = \Delta^f_{p,n},$$
(2.18)

then, the delay to deposit can be expressed by

$$\Delta^d_{p,n} = f_{p,n} - f^*_{p,n}.$$
(2.19)

Using the concept of fire-equivalent node, this equation can be transformed into

$$\Delta^d_{p,n} = max\{d_{q,i}, d_{q,x}\} - f^*_{p,n}.$$
(2.20)

Recall that the terms with * denote times that do not change between both estimated and actual transition times.

By the properties of the *max*{} operator, the term $f^*_{p,n}$ can be brought

inside the *max*{} term, so that

$$\Delta^d_{p,n} = max\left\{d_{q,i} - f^*_{p,n}, d^*_{q,x} - f^*_{p,n}\right\}. \tag{2.21}$$

Transforming only the left term in the *max*{} operator:

$$\Delta^d_{p,n} = max\left\{d_{q,i} - max\left\{d^*_{q,i}, d^*_{q,x}\right\}, d^*_{q,x} - f^*_{p,n}\right\}$$

$$= max\left\{d_{q,i} + min\left\{-d^*_{q,i}, -d^*_{q,x}\right\}, d^*_{q,x} - f^*_{p,n}\right\}$$

$$= max\left\{min\left\{d_{q,i} - d^*_{q,i}, d_{q,i} - d^*_{q,x}\right\}, d^*_{q,x} - f^*_{p,n}\right\}$$

$$= max\left\{min\left\{\Delta^d_{q,i}, \Delta^d_{q,i} + d^*_{q,i} - d^*_{q,x}\right\}, d^*_{q,x} - f^*_{p,n}\right\}$$

$$= max\left\{\Delta^d_{q,i} + min\left\{0, d^*_{q,i} - d^*_{q,x}\right\}, d^*_{q,x} - f^*_{p,n}\right\},$$

yielding,

$$\Delta^*_{p,n} = max\left\{\Delta^d_{q,i} - max\left\{d^*_{q,x} - d^*_{q,i}, 0\right\}, d^*_{q,x} - f^*_{p,n}\right\}. \tag{2.22}$$

Solving equation (2.22) requires considering two cases.

Case 1:

$$d^*_{q,x} < d^*_{q,i} \Rightarrow f^*_{p,n} = d^*_{q,i} \Rightarrow d^*_{q,x} - f^*_{p,n} < 0 \Rightarrow \tau_{q,i,p,n} = 0.$$

Thus,

$$\Delta^d_{p,n} = max\left\{\Delta^d_{q,i} - max\left\{d^*_{q,x} - f^*_{p,n}, 0\right\}, d^*_{q,x} - f^*_{p,n}\right\}$$

$$= max\left\{\Delta^d_{q,i} - max\left\{< 0, 0\right\}, < 0\right\},$$

so that,

$$\Delta^d_{p,n} = \Delta^d_{q,i}. \tag{2.23}$$

Case 2:

$$d^*_{q,x} \geq d^*_{q,i} \Rightarrow f^*_{p,n} = d^*_{q,x} \Rightarrow d^*_{q,x} - f^*_{p,n} = 0 \Rightarrow \tau_{q,i,p,n} \geq 0.$$

Thus,

$$\Delta^d_{p,n} = max\left\{\Delta^d_{q,i} - max\left\{f^*_{p,n} - d^*_{q,i}, 0\right\}, 0\right\}$$

so that,

$$\Delta^d_{p,n} = max\left\{\Delta^d_{q,i} - \left(f^*_{p,n} - d^*_{q,i}\right), 0\right\}. \qquad (2.24)$$

If $f^*_{p,n} = d^*_{q,i}$ in equation (2.24), then

$$\Delta^d_{p,n} = \Delta^d_{q,i}.$$

Hence, case 2 contains case 1 and the general result is Equation 2.24. This equation can also be written in terms of the static token lifetime as follows:

$$\Delta^d_{p,n} = max\left\{\Delta^d_{q,i} - \tau^*_{q,i,p,n}, 0\right\}. \qquad (2.25)$$

As it can be seen in this expression, the delay propagated by node $n_{p,n}$ can be less than or equal to the delay propagated by node $n_{q,j}$. Thus delay may be absorbed in between nodes or by the edges that connect them.

The delay that is absorbed is equal to the static token lifetime, $\tau^*_{q,i,p,n}$, of the edge. Therefore, the token lifetime may also be called the delay absorption property of an edge.

2.4.3 Lifetime Equivalent Paths

The next step in understanding how delay propagates through directed paths is to develop a model for two nodes in series. This model is then generalized to an n-node series and the delay propagation model for any directed path is presented.

Consider two fire-equivalent nodes $n_{q,i}$ and $n_{p,n}$ connected in a directed path as in Figure 2.4. Node $n_{q,i}$ is one predecessor to $n_{p,n}$ and the only directed path between $n_{q,i}$ and $n_{p,n}$ is one edge.
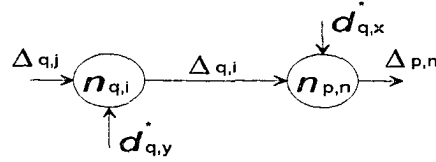
Figure 2.4. Nodes Connected in Directed Path.

By the results of section 2.3.2 and by using Equation 2.24, the delay to deposit $\Delta^d_{p,n}$ is

$$\Delta^d_{p,n} = max\left\{\begin{matrix} \Delta^d_{q,i} - \left(max\{d^*_{q,i}, d^*_{q,x}\} - d^*_{q,i}\right), \\ 0 \end{matrix}\right\};$$
(2.26)

similarly, the delay to deposit $\Delta^d_{q,i}$ is

$$\Delta^d_{q,i} = max\left\{\begin{matrix} \Delta^d_{q,j} - \left(max\{d^*_{q,j}, d^*_{q,y}\} - d^*_{q,j}\right), \\ 0 \end{matrix}\right\}.$$
(2.27)

By substituting Equation 2.27 into Equation 2.26,

$$\Delta_{p,n}^{d} = max\left\{ \begin{array}{l} max\left\{\Delta_{q,j}^{d} - \left(max\{d_{q,j}^{*},d_{q,y}^{*}\} - d_{q,j}^{*}\right),0\right\} - \left(max\{d_{q,i}^{*},d_{q,x}^{*}\} - d_{q,i}^{*}\right), \\ 0 \end{array} \right\}$$

$$= max\left\{ \begin{array}{l} max\left\{ \begin{array}{l} \Delta_{q,j}^{d} - \left(max\{d_{q,j}^{*},d_{q,y}^{*}\} - d_{q,j}^{*}\right) - \left(max\{d_{q,i}^{*},d_{q,x}^{*}\} - d_{q,i}^{*}\right), \\ -\left(max\{d_{q,i}^{*},d_{q,x}^{*}\} - d_{q,i}^{*}\right) \end{array} \right\}, \\ 0 \end{array} \right\}$$

$$= max\left\{ \begin{array}{l} \Delta_{q,j}^{d} - \left(max\{d_{q,j}^{*},d_{q,y}^{*}\} - d_{q,j}^{*}\right) - \left(max\{d_{q,i}^{*},d_{q,x}^{*}\} - d_{q,i}^{*}\right), \\ max\left\{ \begin{array}{l} -\left(max\{d_{q,i}^{*},d_{q,x}^{*}\} - d_{q,i}^{*}\right), \\ 0 \end{array} \right\} \end{array} \right\}$$

so that,

$$\Delta_{p,n}^{d} = max\left\{ \begin{array}{l} \Delta_{q,j}^{d} - \left(max\{d_{q,j}^{*},d_{q,y}^{*}\} - d_{q,j}^{*}\right) - \left(max\{d_{q,i}^{*},d_{q,x}^{*}\} - d_{q,i}^{*}\right), \\ 0 \end{array} \right\}. \quad (2.28)$$

Equation 2.28 may be rewritten in terms of the static token lifetimes as follows

$$\Delta_{p,n}^{d} = max\left\{\Delta_{q,j}^{d} - \left(\tau_{q,j,q,i}^{*} + \tau_{q,i,p,n}^{*}\right),0\right\}. \quad (2.29)$$

In conclusion, the two-node series can be reduced to a one-node path where the token lifetime at the input of interest of $n_{p,n}$ is equal to the sum of the token lifetimes of the two-node series. In other words, the new one-node path is said to be lifetime equivalent to the two-node series.

A similar procedure can be carried out for three, four or more nodes connected by only one directed path in series. It can be shown by induction that a one-node path is lifetime equivalent to an n-node path if the edge of

interest to the node has a static token lifetime equal to the sum of all token lifetimes of the edges in the path. This conclusion may be expressed as follows. A node $n_{p,n}$ connected to n-1 predecessor nodes in a series or chain has a one-node lifetime equivalent path with

$$\Delta_{p,n}^{d} = max\left\{\Delta_{q,j}^{d} - \sum_{i} \tau_{i}^{*}, 0\right\} \qquad (2.30)$$

where

τ_{i} = static token lifetime of edge $e_{i} \in$ n - node path, and

$\Delta_{q,j}^{d}$ = delay introduced at the input of the n-node path.

This set of edges is the only path from the node $n_{q,j}$ that injected the delay $\Delta_{q,j}^{d}$ to the node $n_{p,n}$. The delay propagation model for a directed path was presented in the last section. This result holds only when this path is the only path between the nodes $n_{q,j}$ and $n_{p,n}$.

2.4.4 Dominant Lifetime Equivalent Paths

The delay propagation model for a set of parallel independent directed paths between nodes $n_{p,r}$ and $n_{p,n}$ is developed in this section. The paths are independent in the sense that there is no interconnection between them except at the start and at the end of the paths. This discussion leads to the definition of dominant lifetime equivalent paths.
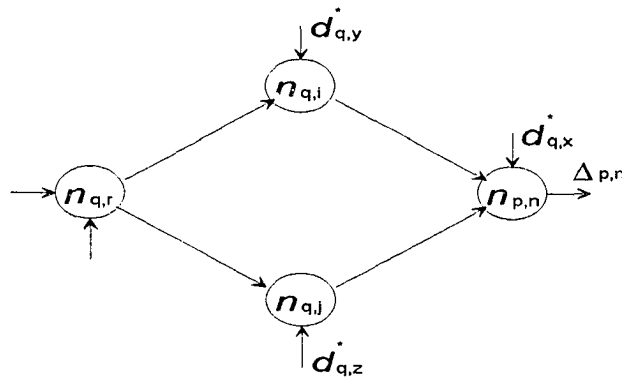
Figure 2.5. Nodes Connected Through Parallel Paths.

Consider two nodes $n_{p,r}$ and $n_{p,n}$ that are connected by only two

lifetime equivalent directed paths as shown in Figure 2.5. Each one of

these directed paths contain only one node each, namely $n_{q,i}$ and $n_{q,j}$.

Employing Equation 2.24 and considering the input edges from $n_{q,i}$ and

$n_{q,j}$ into $n_{p,n}$, the fire equivalent model becomes

$$\Delta_{p,n}^{d} = max \begin{cases} \Delta_{q,i}^{d} - \left(max\left\{ d_{q,i}^{*}, d_{q,j}^{*}, d_{q,x}^{*} \right\} - d_{q,i}^{*} \right), \\ \Delta_{q,j}^{d} - \left(max\left\{ d_{q,i}^{*}, d_{q,j}^{*}, d_{q,x}^{*} \right\} - d_{q,j}^{*} \right), \\ 0 \end{cases}. \tag{2.31}$$

In the same manner, the expression for the delay to deposit $\Delta_{q,i}^{d}$ and

$\Delta_{q,j}^{d}$ are, respectively,

$$\Delta_{q,i}^{d} = max \begin{cases} \Delta_{p,r}^{d} - \left(max\left\{ d_{p,r}^{*}, d_{q,y}^{*} \right\} - d_{p,r}^{*} \right), \\ 0 \end{cases} \tag{2.32}$$

and

$$\Delta_{q,j}^{d} = max \begin{cases} \Delta_{p,r}^{d} - \left(max\left\{ d_{p,r}^{*}, d_{q,z}^{*} \right\} - d_{p,r}^{*} \right), \\ 0 \end{cases}. \tag{2.33}$$

Substituting Equations 2.32 and 2.33 into Equation 2.31 the

expression for the delay to deposit $\Delta^d_{p,n}$ is

$$\Delta^d_{p,n} = max\left\{ \begin{array}{l} max\left\{ \begin{array}{l} \Delta^d_{p,r} - \left(max\{d^*_{p,r},d^*_{q,y}\} - d^*_{p,r}\right), \\ 0 \end{array}\right\} - \left(max\{d^*_{q,i},d^*_{q,j},d^*_{q,x}\} - d^*_{q,i}\right), \\ max\left\{ \begin{array}{l} \Delta^d_{p,r} - \left(max\{d^*_{p,r},d^*_{q,z}\} - d^*_{p,r}\right), \\ 0 \end{array}\right\} - \left(max\{d^*_{q,i},d^*_{q,j},d^*_{q,x}\} - d^*_{q,j}\right), \\ 0 \end{array}\right\}.$$

Manipulating the internal $max\{\}$ operators, the equation can be

transformed into

$$\Delta^d_{p,n} = max\left\{ \begin{array}{l} \Delta^d_{p,r} - \left(max\{d^*_{p,r},d^*_{q,y}\} - d^*_{p,r}\right) - \left(max\{d^*_{q,i},d^*_{q,j},d^*_{q,x}\} - d^*_{q,i}\right), \\ \Delta^d_{p,r} - \left(max\{d^*_{p,r},d^*_{q,z}\} - d^*_{p,r}\right) - \left(max\{d^*_{q,i},d^*_{q,j},d^*_{q,x}\} - d^*_{q,j}\right), \\ 0 \end{array}\right\}.$$

By further manipulation and extracting the common term $\Delta^d_{p,r}$, the result is

$$\Delta^d_{p,n} = max\left\{ \begin{array}{l} \Delta^d_{p,r} - min\left\{ \begin{array}{l} \left(max\{d^*_{p,r},d^*_{q,y}\} - d^*_{p,r}\right) + \left(max\{d^*_{q,i},d^*_{q,j},d^*_{q,x}\} - d^*_{q,i}\right), \\ \left(max\{d^*_{p,r},d^*_{q,z}\} - d^*_{p,r}\right) + \left(max\{d^*_{q,i},d^*_{q,j},d^*_{q,x}\} - d^*_{q,j}\right) \end{array}\right\}, \\ 0 \end{array}\right\}.$$

(2.34)

This equation can be expressed in terms of the static token lifetime

$$\Delta^d_{p,n} = max\left\{ \begin{array}{l} \Delta^d_{p,r} - min\left\{ \begin{array}{l} \tau^*_{p,r,q,i} + \tau^*_{q,i,p,n}, \\ \tau^*_{p,r,q,i} + \tau^*_{q,j,p,n} \end{array}\right\}, \\ 0 \end{array}\right\}$$

(2.35)

It is indicated by this expression that the delay to deposit $\Delta^d_{p,n}$ can be calculated based only on the delay to deposit $\Delta^d_{p,r}$ and the path with the minimum total token lifetime. It is said that the path with the minimum total token lifetime dominates the other path. In conclusion, the path with the minimum total token lifetime is called the dominant lifetime equivalent path.

By a similar procedure, it can be shown that for k independent parallel paths between nodes $n_{p,r}$ and $n_{p,n}$, the delay to deposit $\Delta^d_{p,n}$ is

$$\Delta^d_{p,n} = max\left\{ \begin{array}{l} \Delta^d_{p,r} - min\left\{ \sum_i \tau_i^{*1}, \sum_i \tau_j^{*2}, \cdots, \sum_i \tau_j^{*k}, \right\}, \\ 0 \end{array} \right\} \qquad (2.36)$$

or

$$\Delta^d_{p,n} = max\left\{ \begin{array}{l} \Delta^d_{p,r} - \sum_i \tau_j^{dom}, \\ 0 \end{array} \right\}, \qquad (2.37)$$

where τ_i^{dom} is the static token lifetime in the edge e_i, and edge e_i exists in the dominant lifetime equivalent path of the set of k independent parallel paths.

2.4.5 Path Construction

Any loop-free network between two nodes $n_{p,r}$ and $n_{p,n}$ can be constructed be connecting or concatenating series of nodes or independent series in parallel. This section is used to demonstrate this construction method. By using this method any loop-free network between two nodes $n_{p,r}$ and $n_{p,n}$ can be reduced to a lifetime equivalent path (LEP). It will also be shown how delay introduced at $n_{p,r}$ is propagated to $n_{p,n}$.

2.4.5.1 Concatenation

Consider two lifetime equivalent paths LEP_1 and LEP_2 with token lifetimes τ_1 and τ_2, respectively. If these two paths are concatenated, following the procedure outlined in Section 2.4.3, it can be shown that the resultant LEP_c has lifetime τ_c,

$$\tau_c = \tau_1 + \tau_2.$$

Concatenation of paths is indicated as follows:

$$LEP_1 : LEP_2 \equiv LEP_c.$$

It can be shown that the concatenation operator (:) is associative, i.e.,

$$(LEP_1 : LEP_2) : LEP_3 \equiv LEP_1 : (LEP_2 : LEP_3)$$

and, therefore, n number of paths can be concatenated and are lifetime

equivalent to a single path LEP_c with token lifetime τ_c

$$\tau_c = \sum_{i=1}^{i=n} \tau_i,$$

where τ_i is the lifetime of LEP_i.

2.4.5.2 Parallel Paths

Consider two lifetime equivalent paths LEP_1 and LEP_2 with token lifetimes τ_1 and τ_2, respectively. If these two paths are connected in parallel, i.e., both have a common predecessor node and a common successor node, following the procedure outlined in Section 2.4.4, it can be shown that a resultant LEP_p has lifetime τ_p:

$$\tau_p = min\{\tau_1, \tau_2\}.$$

Connecting paths in parallel is indicated as follows:

$$LEP_1 \| LEP_2 \equiv LEP_c$$

It can be shown that the parallel operator ($\|$) is associative, i.e.,

$$\left(LEP_1 \| LEP_2\right) \| LEP_3 \equiv LEP_1 \| \left(LEP_2 \| LEP_3\right)$$

and, therefore, n number of paths can be connected in parallel and they are lifetime equivalent to a single path LEP_p with token lifetime

$$\tau_p = min\{\tau_1, \tau_2, ..., \tau_n\}$$

where τ_1 is the lifetime of LEP_1, τ_2 is the lifetime of LEP_2, etc. It can also be expressed as

$$LEP_p = domLEP .$$

2.4.5.3 Distributivity and Commutativity

It can be shown that the concatenation operator distributes over the parallel operator. Consider lifetime equivalent paths LEP_1, LEP_2, and LEP_3 then

$$LEP_1 \cdot \left(LEP_2 \| LEP_3 \right) \equiv \left(LEP_1 : LEP_2 \right) \| \left(LEP_1 : LEP_3 \right)$$

and

$$\left(LEP_1 \| LEP_2 \right) \cdot LEP_3 \equiv \left(LEP_1 : LEP_3 \right) \| \left(LEP_2 : LEP_3 \right).$$

Both concatenation and parallel operators are commutative such that

$$LEP_1 : LEP_2 \equiv LEP_2 : LEP_1$$

and

$$LEP_1 \| LEP_2 \equiv LEP_2 \| LEP_1.$$

These properties stem from the properties of the operator '+' in the concatenation operation and the operator $min\{\}$ in the parallel operation.

2.4.5.4 Identifying the Dominant LEP

By the use of these two operators, it is possible to identify the

dominant LEP between any two nodes $n_{p,r}$ and $n_{p,n}$ in an XAMG. These

operators can be applied to sections of the network to reduce it to a single

edge representing the dominant *LEP* between $n_{p,r}$ and $n_{p,n}$. This is

demonstrated with a practical example as shown in Figures 2.6.a to 2.6.i.

Consider two nodes $n_{p,r}$ and $n_{p,n}$ connected by a loop-less network. This is

a subgraph contained in another graph where edges going out or coming in

are not shown for simplicity. The only directed paths between $n_{p,r}$ and

$n_{p,n}$ are the ones shown in Figure 2.6.a.

By applying the concatenation operator to the path that contains node

B, and to the path that contains node E, and applying the parallel

operator to node E results the graph as in Figure 2.6.b. Using the

distribution of the concatenation operator over the parallel operator to the

paths that contain the node A, it yields the graph in Figure 2.6.c. By

applying the parallel operator between node $n_{p,r}$ and node C, the graph is

transformed as shown in Figure 2.6.d. By applying distribution of the

concatenation operator over the parallel operator to the paths that contain

node C, the graph is further reduced as shown in Figure 2.6.e. The

application of the parallel operator to the paths between the node $n_{p,r}$ and

node D yields the the graph in Figure 2.6.f. By applying the concatenation
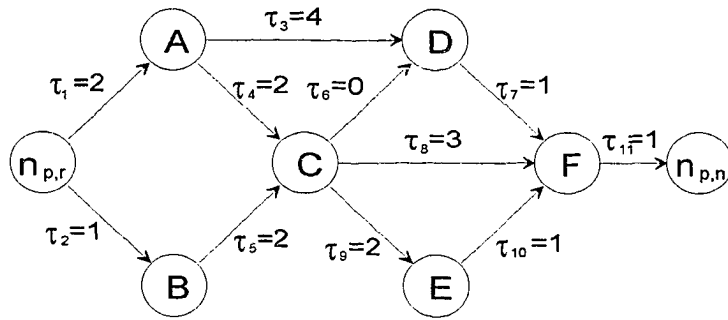
Figure 2.6.a Example of Graph Reduction.
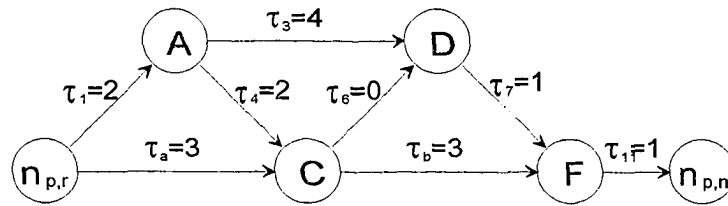
Figure 2.6.b.
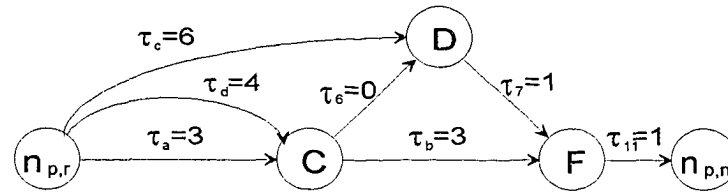
Figure 2.6.c.

Figure 2.6.d.

Figure 2.6.e.

$\tau_h=3$

D

$\tau_7=1$

$n_{p,r}$

$\tau_g=6$

F $\tau_{1}=1$ $n_{p,n}$

Figure 2.6.f.

$\tau_i=4$

$\tau_g=6$

$n_{p,r}$

F $\tau_{1}=1$ $n_{p,n}$

Figure 2.6.g.

$n_{p,r}$ $\tau_j=4$

F $\tau_{1}=1$ $n_{p,n}$

Figure 2.6.h.

$n_{p,r}$ $\tau_k=5$ $n_{p,n}$
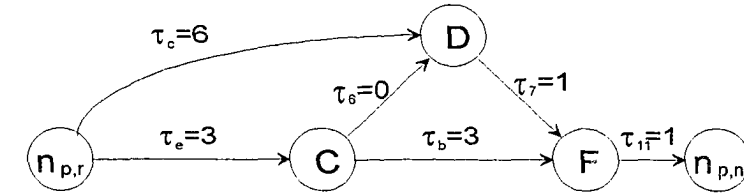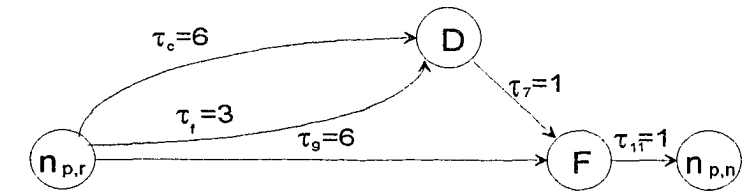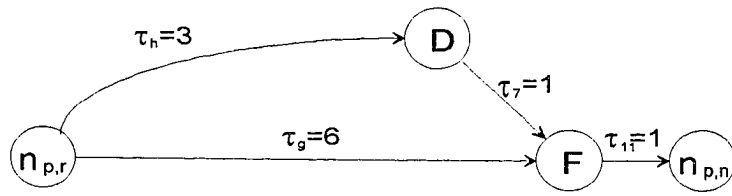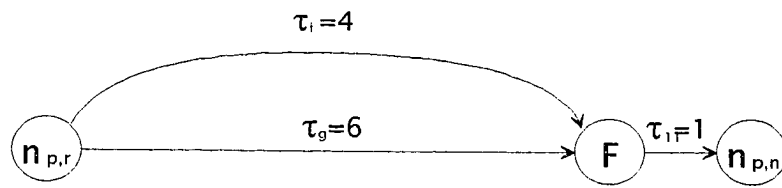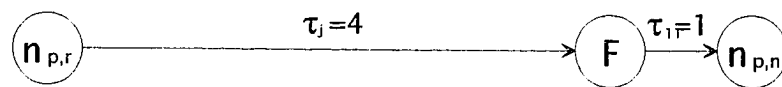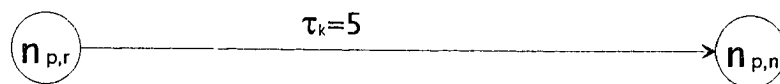
Figure 2.6.i.

operator to the path that contains node D, the graph is transformed as shown in Figure 2.6.g. The application of the parallel operator to the paths between node $n_{p,r}$ and node F produces the graph in Figure 2.6.h. Finally the use of the concatenation operator in the path that contains node F reduces the graph as shown in Figure 2.6.i. It can be observed that the path with τ_k is lifetime equivalent to the path between $n_{p,r}$ and $n_{p,n}$ with the minimum token lifetime.

2.4.6 Alternate Method for Identifying the Dominant LEP

Another method to identify the dominant LEP between two nodes is to identify the path with the largest sum of node process times. This approach follows directly from the result of Section 2.4.5.4 and is developed below.

Consider two nodes $n_{p,r}$ and $n_{p,n}$ connected by a network. Let the time interval between the output of $n_{p,r}$ and the output of $n_{p,n}$ be represented by T. Any path k between $n_{p,r}$ and $n_{p,n}$ must comply with the following equation:

$$T = \sum_i p_i^k + \sum_j \tau_j^k$$

where p_i^k is the node process time of the ith node in path k and τ_j^k is the jth token lifetime in path k.

Regardless of the path that is traversed, the addition of the node process times and token lifetimes is equal to T, i.e., T is a constant. If the dominant LEP is the path with minimum token lifetime, it can be concluded that the dominant LEP is also the path with the largest sum of node process times. This path is also known as the time critical path between nodes $n_{p,r}$ and $n_{p,n}$.

In conclusion, the dominant LEP between $n_{p,r}$ and $n_{p,n}$ can be identified by finding the time critical path between $n_{p,r}$ and $n_{p,n}$.

2.5 Definitions

The subsequent definitions relate to the graph model in general. They are used to aid the understanding of results in Chapter Four.

2.5.1 $TBIO_{n,m}$

The definition of $TBIO$ in the AMG is the time associated with the path with the longest time between the input source and the output sink. This is straightforward since there is only one source and one sink in the graph. However, in the XAMG (XCMG), there are as many sources/sinks as there are data packets. Therefore there can be many paths between sources and sinks. If there are n number of data packets, there can be n longest directed paths between source 1 and all sinks. To distinguish each of these paths they need to be labelled according to the source and sink at

both ends of the path. This can be accomplished by subscripting $TBIO$ with the corresponding source and sink. That is, $TBIO_{n,m}$ is the time associated with the longest directed path between source n and sink m. The original $TBIO$ of the AMG then becomes $TBIO_{i,i}$, i.e., the time associated with the longest directed path between the source i and its corresponding sink i.

2.5.2 $CP_{n,m}$

The path that characterizes $TBIO$ in the AMG (CMG) is called the Critical Path (CP). The path associated with $TBIO_{n,m}$ is then called the Critical Path between source n and sink m or $CP_{n,m}$.

2.5.3 System Slack

The systems are assumed to work at a $TBI \geq TBO_{LB}$ Define the difference between TBI and TBO_{LB} as the system slack, σ, where

$$\sigma = TBI - TBO_{LB}. \tag{2.38}$$

2.5.4 $TBIO_{LB(i,i+1)}$

In the AMG, the token lifetime of the edges in the CP, the longest path between the source and the sink, is zero. However, in the XAMG, that is not necessarily true for all $CP_{i,i+1}$, the critical paths between the possible source/sink pairs. Since the actual value of $TBIO_{i,i+1}$ may be

affected by the token lifetimes in the edges, there is the need to define a

lower bound for $TBIO_{i,i+1}$. The lower bound for $TBIO_{i,i+1}$ is defined by

$TBIO_{LB(i,i+1)}$:

$$TBIO_{LB(i,i+1)} = TBIO + TBO_{LB}.$$ (2.39)

The value for $TBIO_{i,i+1}$ in steady state is then

$$TBIO_{i,i+1} = TBIO + TBI.$$ (2.40)

By substituting TBI expressed in terms of the system slack, it results in

$$TBIO_{i,i+1} = TBIO + TBO_{LB} + \sigma.$$ (2.41)

By further combination of $TBIO$ and $TBIO_{LB}$, the final result is

$$TBIO_{i,i+1} = TBIO_{LB(i,i+1)} + \sigma.$$ (2.42)

In conclusion, the system slack σ is the total token lifetime in the

path $CP_{i,i+1}$.

It can be shown that for an arbitrary value of k, $TBIO_{i,i+k}$

$$TBIO_{i,i+k} = TBIO_{LB(i,i+k)} + k\sigma, \qquad k = 1, 2, ..$$ (2.43)

As a generalization, the total token lifetime in the path $CP_{i,i+k}$ is k times

the system slack σ.

2.6 Summary

The graph model to investigate the transient behavior of a multicomputer system has been presented in this chapter. The transient of interest is that of the effect of a delay introduced into one of the nodes of the algorithm graph due to a fault. The model has been shown to be useful to study the propagation of delay through a graph. A measure of importance has been the token lifetime to be found in the paths between any two nodes connected by at least one directed path. The existence of token lifetime in a given path between two nodes expresses the amount of delay that such path is able to absorb. When there is more than one path, there is a dominant path with respect to the token lifetime. This path is called the dominant lifetime equivalent path between two nodes. Two methods to calculate the dominant lifetime equivalent path were shown. These methods are to be used in Chapter Four in the testing of the model against simulated system behavior. Finally some definitions were presented to help in the understanding of the graph model. Some of these definitions refer to the extension of measures of the steady state analysis. Others pertain only to the transient model and do not have application on the steady state model. The results of this chapter present the usefulness of the model to start investigating the transient behavior of a multicomputer system designed with the ATAMM strategy.

CHAPTER THREE

DEVELOPMENT

The theory to study the behavior of a multicomputer system in a transient due to a fault is detailed in Chapter Two. This theory may be exposed either by system simulation or by real system implementation. One way to simulate a system is by using a general purpose computer. This method should simulate normal operation of the system along with failure of computing resources and the recovery and restoration of the system. Real system implementation, on the other hand, requires more sophisticated hardware since it is used to test real-time software. An available hardware system to validate the theory is the Generic VHSIC Spaceborne Computer (GVSC). The implementation of a system that complies with ATAMM in the GVSC as well as its fault-tolerant features is described in this chapter. The development of the ATAMM Multicomputer Operating System is described in Section 3.1. The fault-tolerant system phases are presented in Section 3.2. In Section 3.3 the operating system additions to make the system fault-tolerant are explained.

45

3.1 ATAMM Multicomputer Operating System (AMOS)

One purpose of an operating system is to "allocate hardware resources among tasks"[18]. The objective of this section is to identify the hardware resources and the tasks in the system of interest. The system under discussion is not a general-purpose computer but an embedded system for applications such as control and signal processing, among others. Therefore, this embedded system does not have all attributes of a general-purpose computer. For example, an embedded system may not have 'human' users, i.e., the system processes a signal from a servo and passes the results onto another system. Another attribute of the systems under study is that they are real-time systems where the time to fulfill a service request is as important as the data processing itself. These systems are also multicomputer systems and, as such, impose another difficulty on their design, namely how to obtain maximum system throughput.

Many computing systems have been designed from the point of view of hardware needs. This perspective has imposed a handicap on the software development for these systems. Software design around system architecture in order to take full advantage of the system limits the reusability of the software package. Often such software depends on a given number of computing resources which are to be connected in a certain fashion in the system. For example, if software is to be developed for a hypercube computer, the software engineer considers the hypercube

connection not only a feature of the system but a requirement for the software to work as desired. If the software is to be used or moved to another computing system that is not a hypercube computer, most probably the package will have to be rewritten and retargeted to the new machine.

On the other hand, there are computing systems that are designed from the point of view of software needs. This perspective imposes restrictions on the hardware field. Achievement of optimization in the hardware is not easy since software drives the hardware design and development. Such systems ensure that the software for which they are designed run optimally. A typical example of this kind of system is a vector processor or math processor. The disadvantage is that these systems are highly specialized and all too often not reusable for another type of software.

3.1.1 AMOS Overview

The purpose of AMOS is to take a multicomputer system architecture and a software system, both independently designed, and create a common interface between them. The objective is that the system architecture should look optimally designed for the software system and the software system should look optimally designed for the system architecture. The AMOS operating system becomes a common ground for the hardware designer as well as for the software developer.

The system architecture is the hardware factor in the overall system. The different hardware elements of this system are the individual computing resources, the communication channels and the memory resources. These elements should be managed for efficient use in the execution of the application program. The system architecture designer is concerned with this set of components without necessarily knowing the application software that will run in the system.

The algorithm graph is the software factor in the overall system. This algorithm includes the graph node code and the data interdependency or connection between the nodes. For each graph that the system is expected to run, there is a set of nodes and their interconnection to each other that constitute the data path, along with the appropriate node code to process the data. The basic ATAMM system has the property that a node in the algorithm graph becomes enabled when the following conditions are met: all input edges contain tokens (data or control), the node is available or not busy, and there are empty places on the output edges to deposit the output data (see Figure 3.1). The software application designer is concerned with the logical interconnection of the nodes (processes) through data edges. This concern refers to the data flow of the algorithm and not to how to map the algorithm to a given architecture. This feature an architecture that is transparent to the software designer.
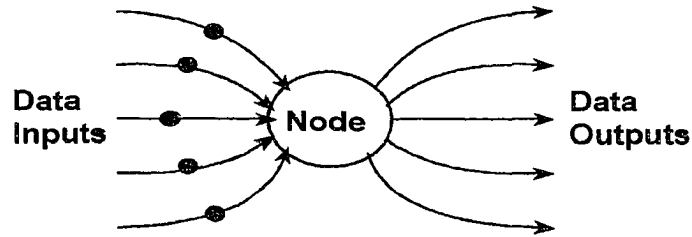
Figure 3.1. An Enabled Node in a Marked Graph.

One attribute that is highly desirable in systems design is the effective use of multiple processors or computers in a system. The AMOS operating system becomes the multicomputer connection in that it manages the interprocessor communications. Second, AMOS manages the interprocess communications. Third, AMOS manages the communications between the processes and the processors.

AMOS can be regarded as a two-way translator. It translates the requests from the processes to the processors. One of those requests can be 'execute this node.' It also translates the requests from the processors to the processes. Again, one of such requests can be 'code execution has generated data.' This two-way translation posses a challenge since the entities that should communicate are vastly different. On one hand, the processors are physical units and they can be defined and located in space. On the other hand, the processes are abstract entities and they cannot be defined in physical terms as are the processors. Recall that the processes are represented by the graph and its data interconnections and the code that should be executed. For the processors and processes to communicate there is a need for a common medium and to address this need concepts are

borrowed from object-oriented programming (OOP) or object-oriented

systems (OOS). The following section is a very brief summary of OOP.

3.1.2 Object-Oriented Programming Paradigm

The object-oriented programming paradigm has grown in its

acceptance and application in the last few years. In simple terms, object-

oriented programming may be considered an extension of structured

programming and a programming philosophy in the sense that it imposes a

particular structure on the software development. This structure combines

data and code into one package where the code is optimized to manipulate

the accompanying data. Structured programming, the predecessor of OOP,

was developed with the idea of creating code that is generic enough to be

used in many applications. One disadvantage was that the code that was

generated was highly dependent on the data structure. If the data

structures had to be modified to enhance the capabilities of the system, the

routines that manipulated the data structures had to be modified as well.

Since the data structure was known throughout the program, all levels of

the software had to be modified also. The data dependency aspect can be

alleviated by using one of the attributes of OOP, namely, data

encapsulation. This is achieved by creating a more sophisticated data type

or data structure, one that contains not only the declarations of the

primitive data types but also the functions that directly manipulate such

data. These functions are called member functions, the data variables are called data members and the whole unit is called a class.

An instance of a data structure is called a variable, but an instance of a class is called an object, hence the name object-oriented programming. An object is then an instance of a class and contains a set of data variables and member functions that manipulate the data. According to good object-oriented programming practice, only the member functions should directly manipulate the data in the object. When a member function is called or invoked in an object it is said that a message is passed to the object. This message indicates the type of data manipulation that is requested, either read, write or initialize a given data member. This allows for programs that are object-oriented to be more generic in the data manipulation, since only the object knows how the data is represented internally. A simplified view of an object is depicted in Figure 3.2. These principles are used to explain which of the features were borrowed from OOP and placed in AMOS.

The data encapsulation concept may be used to isolate the details of how a specific piece either of hardware or software is manipulated. The
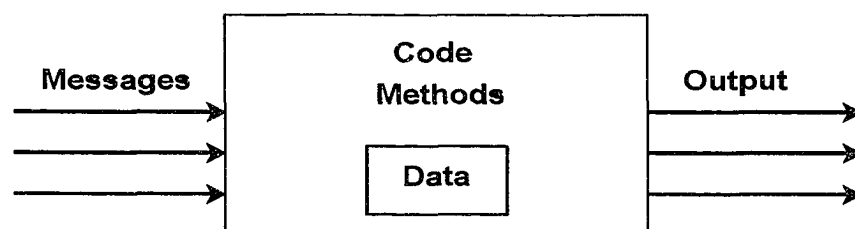
Figure 3.2. Simplified Structure of an Object.

AMOS operating system should not be unnecessarily involved with the direct manipulation of either one or another. Instead of directly manipulating the data that represents the graph, AMOS should send a 'message' to the graph requesting a particular service. In the same fashion, AMOS should send a 'message' to a computing resource to request or assign a particular task. Changes in the internal behavior of any of these objects, hardware or software, should not impact the general behavior of AMOS. A set of message queues is implemented for this purpose in AMOS, to isolate requests from one side of the system to the other. The logical structure of AMOS is explained in the following section.

3.1.3 AMOS Organization

The AMOS operating system is targeted for a multiprocessor environment, but the code may be used in a system with only one processor. The system uses as many computing resources as the graph requires for its designed operation. The status of the system at any moment would be the status of the combination of all its computing resources. This fact leads to an overwhelming task in explaining how the system operates. Therefore, the point of view of the operation of a single processor or computing resource is taken instead. An overview of the resource logical structure is shown in Figure 3.3.

The different components can be divided into the following categories: the message handler, the graph, the queues and the semaphore. The
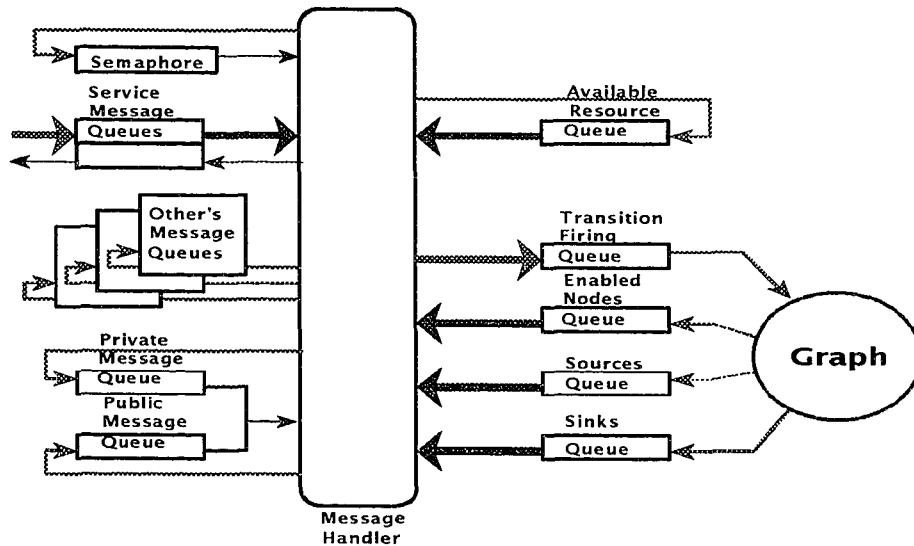
Figure 3.3. Resource Logical Structure.

message handler is the component that moves messages from one queue to

another. It may take a message from the 'Enabled Nodes Queue' to a

resource 'Public Message Queue'. The graph contains the representation of

the algorithm graph to be executed in the system. A description of the

internal graph representation or data structure is irrelevant at this level,

although it is important to say that it contains information such as the

connection of the nodes, from where each node input data are read, where

each node output data should be stored and what code should be executed

with what data, among others.

The queues are the pipeline connection to hold the messages or other

information for the systems components such as computing resources and

the graph. They are the abstract means by which the hardware and

software components communicate with each other. Finally, the

semaphore is a logical variable which provides a means for arbitrating access to the graph. The semaphore is used to permit access to the graph since there may be multiple computing resources in the system, and for redundancy purposes, a copy of the graph exists in each and every one of the processors. Every time a message is to be handled that affects the status of the graph, the semaphore should be requested. Only by possessing this semaphore (setting the variable to true), is a computing resource is permitted to manage the graph and its queues. In practice, the semaphore regulates access to the communications channel of the system, and hence to updating the copy of the graph in each and every other computing resource. Because of the boolean nature of the semaphore, only one computing resource can have access to the graph at a time.

The Enabled Nodes, Sources and Sinks queues are the means through which the graph requests a service from the system hardware. The Enabled Nodes queue contains messages which have all the information that identifies an enabled node to be executed. If a message exists in this queue, there is an associated node in the graph which is enabled and ready to be executed. Similarly, the Sources queue contains all the sources, or system inputs, that are available to the system and each source in this queue has a variable with a time relative to the global clock. This time indicates when that source is ready to be executed and input data is ready to be brought into the system. The Sinks queue contains all the sinks, or

system outputs, that are ready to be executed. The Transition Firing queue is the message input to the graph. This queue contains all the transitions that should be fired in the graph. It is used to pass the messages from the computing resources to indicate the activity that is taking place with respect to the nodes assigned to them. It is used to update the graph and reflect the state of the data and nodes in the system.

The Private Message and Public Message queues are used by the computing resources to receive messages. The Private Message queue is used exclusively for private or intraprocessor messages. These messages are written to and read by the computing resource itself. This is for consistency so that the computing resource is message driven even if the messages are written by itself. The Public Message queue is used exclusively for public or interprocessor messages. These messages may be written by any of the computing resources in the system, but they are only read by the computing resource for which the queue is intended. A computing resource needs to obtain the semaphore to send a public message, even if it is targeting its own queue.

The Available Resources queue contains the ID's of all computing resources that are available for executing nodes in the graph. Every time a computing resource starts executing a node, its ID is removed from this queue. When the node is finished, the computing resource ID that executed the node is placed at the bottom of the queue. When an enabled

node is removed from the Enabled Nodes queue, it is assigned to the

computing resource that is at the top of the Available Resources.

3.1.4 AMOS Messages

The Service Messages queues are used to communicate messages to

and from the outside world. These are messages that are used for testing

purposes and to change certain parameters in the system. The messages

can be written or read only by the computing resource that possesses the

semaphore.

The messages that are passed among the elements in the system can

then be classified into private and public messages. These messages are

tabulated in Table 3.1. A brief explanation of the purpose of these

messages follows. The message 'None' is used to describe when no other

type of message is found in the private or public message queues. This

message is returned as the default message from the queue message

reader. The message 'Register' is used to inform the system that a resource

has become available for normal use. This initializes the private and

public queues of the computing resource and pushes the computing

resource ID into the Available Resource queue. The message 'Fire'

indicates that a node has been fired, that it has started the processing of

its input data. The message 'Data' indicates that a node has finished its

processing and has deposited its output data. The message 'Self-Test'

indicates that the computing resource should start the self test routine.

This requires that the computing resource be removed from the system in

case it does not return from the self-test routine. There are more

parameters in a message that are not shown in Table 3.1 and are not

relevant to this overview.

Message	First Parameter	Second Parameter
None[*]	N/A	N/A
Register[*]	ID	N/A
Fire[**]	Node	Color
Data[*]	Node	Color
Self-Test[**]	Pass/No Pass	N/A

* Private Message

** Public Message

Table 3.1. AMOS Messages.

3.1.5 State Diagram

The behavior of any of the computing resources, being a finite-state

machine, can be described using a state diagram. This diagram is depicted

in Figure 3.4. The states are connected by arcs that indicates the

conditions or 'messages' by which the computing resource goes from one

state to the next. The following is a brief description of the states of the

diagram. The state 'System Init' is the state where the resource 'registers'

itself to the system by pushing its ID into the Available Resources queue.

The state 'Graph Init' is where the graph is initialized for processing, all

initial tokens are placed in the appropriate data edges and the state of the

graph is brought to its initial conditions. The state 'Idle' is where the

computing resource reads its own message queues. According to every

message read from the message queues, the computing resource moves to

another state. The message 'None' takes the computing resource to the

'Bus Mgt' state where it scans the different graph or system queues,

Available Resources queue, Sources queue and Service queue. If any of

these queues is non-empty, the computing resource tries to capture the

semaphore. If it succeeds, it moves onto the 'Graph Mgt' state to carry out

the duty indicated in the non-empty queue. If it fails, it goes back to 'Idle'

and starts the process again. The message 'Fire' takes the computing

resource to the state 'Exec' where it will execute the appropriate node on its

input data. The message 'Data' indicates to the computing resource to

jump to the state 'Bus Mgt' In this state it does not query the queues,

instead, it simply tries to capture the semaphore. If it succeeds in

capturing the semaphore, it goes onto the 'Graph Mgt' state, otherwise it

goes back to 'Idle' and tries again. The 'Self-Test' message takes the

computing resource to the 'Self Test' state. In this state, the computing

resource executes a predefined self test program or routine. If the

computing resource passes its own test, it 'registers' again to the system. If

it fails, it does not return to the system. Instead, it is considered a malfunctioning computing resource and is not allowed to grab the semaphore anymore.

3.2 Fault Tolerance Scope

A fault-tolerant system is such that the occurrence of a fault does not lead the system to failure. This means that the system should be able to deal with a well-defined class of faults. Anderson and Lee [19] describe a suitable collection of elemental phases to provide fault tolerance to prevent faults from leading to system failures. These phases are: error detection; damage confinement and assessment; error recovery; and fault treatment and continued system service. The succeeding sections are used to describe these phases in somewhat more detail.

3.2.1 Error Detection

The first phase in a fault-tolerant system is error detection. Although a fault cannot be directly detected by a computing system, the consequences or effects of such an event can be tracked. After a fault has taken place in a system, the system eventually enters an erroneous system state. It is this erroneous state that can be detected and used to raise a system exception. An erroneous state is a state that will lead to system failure if it remains undetected and no action is taken to return the system to a valid state [20].
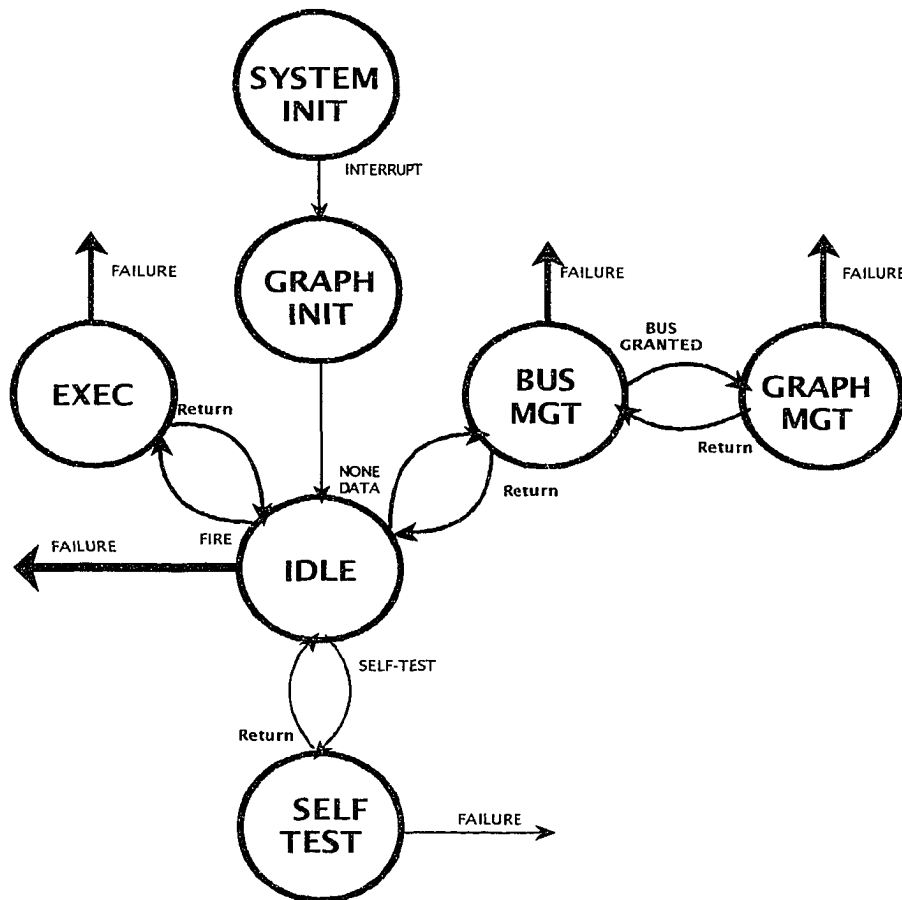
Figure 3.4. Resource State Diagram.

Measures and mechanisms need to be incorporated in a system for proper error detection. Measures for error detection are system components that help convey the necessary information for error detection. Mechanisms are actual implementation techniques that use such measures to raise exceptions in a system when an error is detected. The error detection measures for a computer system can be broadly classified into the following categories: replication checks; timing checks; reversal checks; coding checks; reasonableness checks; structural checks; and diagnostic

checks. Of interest in this section are the replication checks, the timing checks and the diagnostic checks.

Replication checks are among the most effective measures although they are also among the most expensive ones. As their name suggests, replication checks reproduce certain components in a system so that there are multiple copies of such components in the system. The results from an operation with the replicated components are compared against each other. Discrepancy in the comparison indicates that there is an error in the system. One common example of a replication check is Triple Modular Redundancy (TMR). This check employs triplication of a subsystem to provide detection of one or two errors in the subsystem. After a signal or input is processed by these components, the individual results from each replica are compared against each other. Any discrepancy in such a comparison indicates an error in the subsystem. A more general example of this measure is N-Modular Redundancy (NMR) which makes use of n number of replicas instead of only three.

Timing checks are a class of limited replication checks. Limited replication checks are checks that only verify the correct operation of certain parts of a subsystem and from that limited knowledge the operational health of the whole subsystem is determined. Timing checks only provide verification that a certain component has accomplished an operation or task within a time limit or restriction. They do not provide

information as to whether the operation is correct, reasonable or within specifications. These timing checks are also known as time-out checks. The raising of an exception due to a nonpassing timing check indicates that a fault of some kind has taken place in the system. However, the absence of such exception rising does not indicate that the system is fault-free.

Diagnostic checks are used primarily to test the behavior of subsystems under known and controlled inputs. Under these conditions, a subsystem is examined and a particular reaction or output is expected. Implementation of these checks have a tendency to be expensive in terms of resources and time required to execute them. Due to these characteristics, diagnostic checks are better used as secondary error detection measures.

Error detection mechanisms are largely dependent upon the measures used in the system. For duplication checks, for instance, a simple comparator may be sufficient to accomplish the task and raise an exception. With the timing checks a timer may be adequate to achieve the error detection. Finally, the diagnostic checks are heavily contingent on the system in which they are to be employed.

3.2.2 Damage Confinement and Assessment

After an error has been detected in a system, the damage caused by the fault needs to be assessed. The nature of the damage that is assumed

in this dissertation is that of unprocessed data and the loss of a computing resource. Since there is a time delay between the arrival of the fault and the detection of an error produced by the fault, the existent damage may be more than the error detection indicates. How the damage is assessed depends on the considerations taken by the system designer on damage confinement. Damage confinement relates to the restrictions imposed on the information flow in the system design. How these restrictions are fulfilled in the system directly affects the damage assessment after error detection [21].

Damage confinement measures are meaningful for choosing damage assessment measures for a system. The information flow in a system relates to the system structure and a practical concept to structure the system activity is the notion of atomic actions [24]. Atomic actions refer to actions or activities in a system that are said to be not divisible into smaller units. Examples of atomic actions can be opening a file, closing a file, reading from a file and writing to a file. This concept of atomic actions is applied at a given level in the system. It is clear that the atomic action of opening a file could be subdivided into other atomic actions such as allocate memory buffers for file information, fill the buffers with information about the file, associate buffer with a handle, and so forth. Whenever a set of atomic actions is defined it is understood at what level

they are defined and they are assumed to pertain to the same level of complexity.

By defining atomic actions in a system, the designer breaks the system down into modules which interact among themselves. When an error is detected at run-time in a module it is possible to assume that the fault has only affected the module where the error was detected. This is true if the module has not interacted with any other module since it started working. If the module has interacted with other modules prior to the error detection, these other modules may be considered in error as well. Structuring the system with atomic actions helps in the definition of the damage assessment measures.

Damage assessment may be performed in two ways: static assessment and dynamic assessment. Static assessment pertains to the assessment at design time, i.e., the damage assessment is defined *a priori* based only on the knowledge of the system at design time. Dynamic assessment involves the exploration of the system at run-time. This exploration or examination of the system should determine the extent of the damage caused by the fault and hence affects the recovery of the system.

3.2.3 Error Recovery

Error recovery techniques should be applied after error detection and damage assessment. These techniques bring the system from an erroneous state into a valid and healthy state. The damage that was caused by the fault is removed and the system is brought into a state from which it can continue functioning normally. Observe that the phases of error detection and damage assessment are passive in nature, i.e., they do not affect the state of the system but collect information about the erroneous state. On the other hand, the error recovery and the fault treatment phases are active in the sense that they do change the state of the system. The system removes errors during the error recovery phase and faults during the fault treatment phase. Error recovery is one of the areas where much research has been done due to its importance in restoring a system from an erroneous state [22].

Error recovery may be broadly classified into two categories: forward error recovery and backward error recovery. Backward error recovery attempts to reverse time in that it tries to restore the system to a healthy state prior to the fault. An example is that of resetting the system, knowing that the initial state was healthy. This is carried out disregarding the current state of the system, in other words, the portions of the system state that are not erroneous are not taken into consideration when the state is restored. Forward error recovery encompasses all types of error recovery that are not backward error recovery. Forward error

recovery techniques make use of the current state of the system and change those portions that are erroneous in search of a valid and healthy state.

Forward error recovery is dependent on damage assessment whereas backward error recovery is independent of damage assessment. The former is not appropriate to handle any arbitrary faults although its implementation may be simple since the information about the present state is used to recover the system. The latter is suitable for handling arbitrary types of faults but application of the techniques may be complicated since the entire system state is to be recovered. The use of one type of recovery instead of the other is up to the system designer and is also dependent on the system specifications and the specific application.

3.2.4 Fault Treatment and Continued System Service

After a system has undergone error recovery and has hence removed all error from the system, it is necessary to identify the fault that caused the erroneous state in the first place. Continued system service can only be insured by removing the faulty component if it can be identified, otherwise the fault may reoccur. This phase is partitioned into two stages, namely fault location and system repair [23].

Once an exception has been raised due to an error, the error may be removed from the system. The removing of the error does not necessarily indicate where the fault is located. During fault location the system relies

on information provided by the error exception. The disadvantage is that

the mapping of faults to errors may be many-to-one. Many faults may

generate the same error making it difficult to identify the faults based

solely on the error detection and damage assessment. Diagnostic checks

may be used to help locate the fault more accurately but often by taking

more time in the process and making it more expensive.

The system needs to be repaired once the fault has been located. The

system repair is carried out by reconfiguration. The system is reconfigured

so that the faulty component is not allowed to infuse any more faults in the

system. There are three kinds of reconfigurations: manual, dynamic and

spontaneous. The manual reconfiguration requires external or human

intervention in all stages of the reconfiguration. The dynamic

reconfiguration is accomplished by the system in response to external

signals. The spontaneous reconfiguration is done by the system under the

control of the system itself. The last two kinds of reconfiguration are the

most expensive and difficult to implement and are reserved mostly for

applications where there cannot be operator intervention.

3.3 Fault-Tolerant AMOS

The goal in designing AMOS with fault tolerance capabilities is to be

able to recover from the death of a computing resource while executing a

node in the graph. Death of a computing resource is defined as the state

where the resource does not finish executing the task that was assigned
and hence does not report back to the system. In the succeeding sections
the development of each of the fault tolerance phases with respect to
AMOS is presented.

3.3.1 Error Detection in AMOS

Replication checks are inherently expensive. They require the use of
system resources as backups in case of faults. These checks are used in
AMOS only for the implementation of Triple Modular Redundancy (TMR).
The employment of this technique is at the graph level where nodes are
triplicated and their outputs are voted upon when read on the successor
nodes. At the AMOS level there is basically no intervention in the process.
The triplicated nodes are treated as any other node in a non-replicated
graph. The voting is performed in the node shell or just prior to calling the
node procedure code. This information is included here for completeness
only and it does not directly affect the detection of the death of a computing
resource in the system.

Timing checks on the other hand are relatively inexpensive in terms
of system resources. The system resources are used only if an error is
detected and therefore the use of timing checks is very attractive to the
system designer. Among the input variables of the ATAMM design
procedure is the knowledge of execution node times. These node times can

be used to time the nodes that are executed in the system and serve as as time-out limits in timers for error detection. If a computing resource does not complete the execution of its assigned node within the time limit, an error exception is raised and proper action is taken. The following paragraphs contain a description of how the timing checks are performed.

If timing checks are fulfilled by hardware timers, the system hardware would limit how many nodes can work concurrently. Therefore timing checks are carried out in AMOS by the use of a software timer queue. A timer queue is defined as a queue that stores integers sorted by magnitude. The sorting order requires the head of the queue to be the lowest number and the tail of the queue the largest. Every time a node gets assigned to a computing resource, the global clock is read and the node execution time is added to it. This time value indicates when, with respect to the global clock, the associated node is expected to be completed by the computing resource. The value is then inserted in the timer queue and sorted accordingly. When the computing resource concludes executing the node, the associated queue entry is removed from the timer queue. This timer queue can be checked by comparing the global clock against the the head entry of the queue. If the head entry is less than or equal to the global clock, the node is said to be within the proper time margin. If the head entry is greater than the global clock, the node is said to be overdue in its execution and an error exception is raised. The augmented AMOS
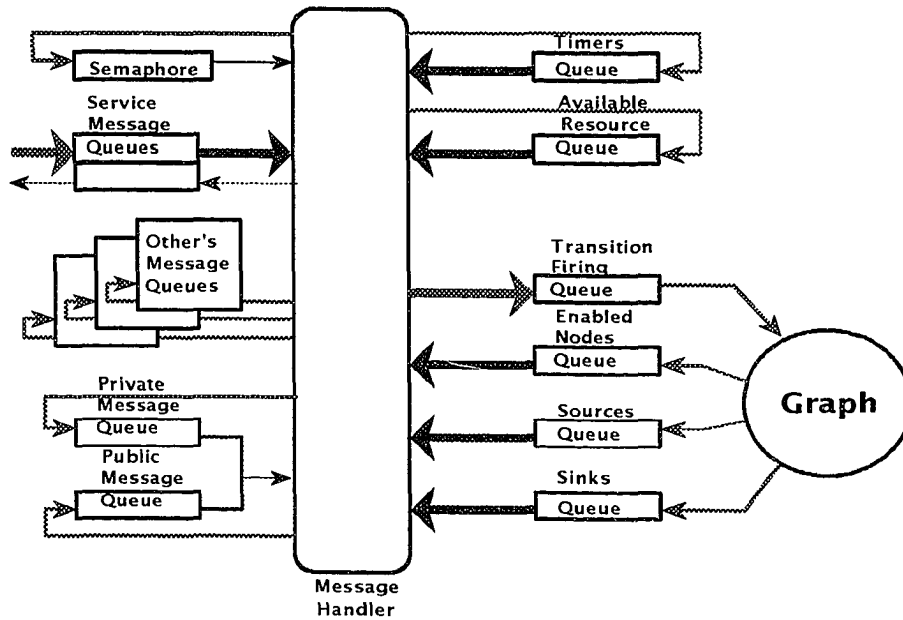
Figure 3.5. Augmented Resource Logical Structure.

logical structure is presented in Figure 3.5. This structure has the new

timers queue incorporated in the system.

Due to the nature of the timers implementation, when the timers

queue is checked becomes important. Every time a computing resource

checks its private and public message queues and does not find a pending

message, the computing resource checks the timers queue. This

requirement takes advantage of the computing resource idle time.

Although this may be sufficient in many cases, there are instances when

the computing resources always find a message in their message queues

making it impossible to check the timers queue. As a protection against

these possible cases, a second requirement is imposed in the system. Every

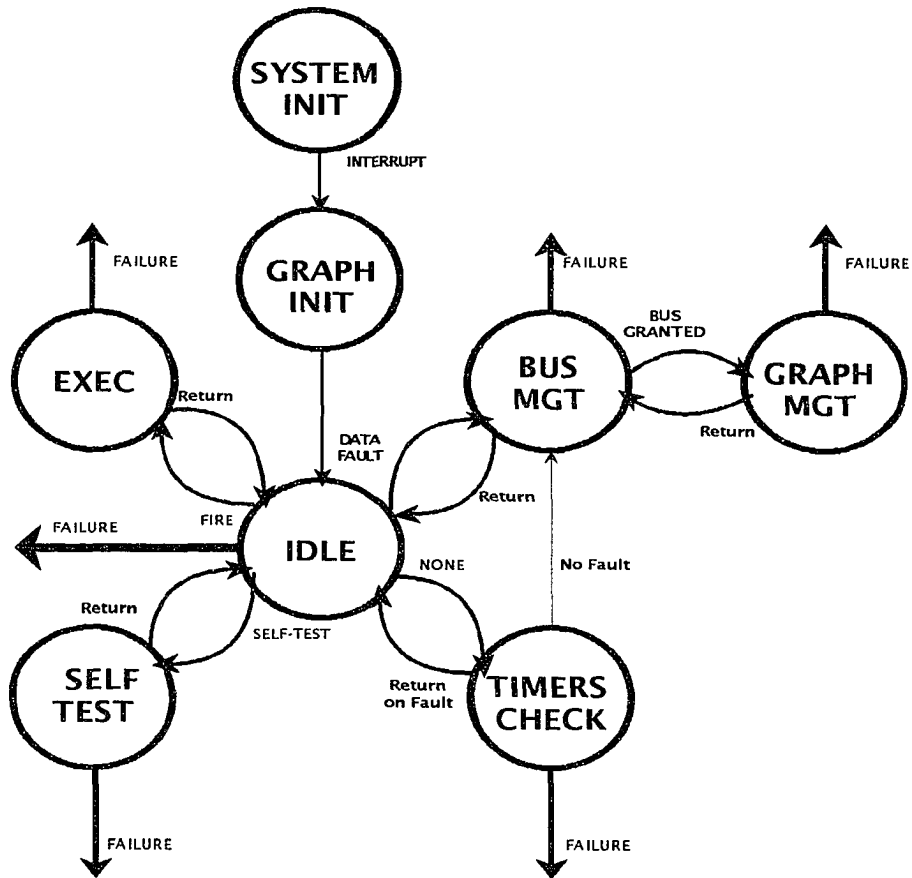time a computing resource completes execution of a node, it checks the

Figure 3.6. Augmented Resource State Diagram.

timers queue. This requisite ensures that the timers queue is examined at

least a number of times equal to the number of nodes in the graph in a

TBO period of time. The augmented resource state diagram is shown in

Figure 3.6. This diagram includes the new state of 'Timers Check'.

When an error is detected in the process of timers checking, an error

exception is raised by the means of sending a new private message. This

message is the FAULT message as depicted in Table 3.2. This message

identifies the node that has not completed. Its information is important for

the next phase of the fault tolerance process, namely 'Damage Assessment'.

The Color attribute refers to the mode of operation, as in the case of TMR,

Duplex or Simplex. For all the experiments presented in this dissertation,

this parameter is a constant and hence irrelevant.

Message	First Parameter	Second Parameter
Fault*	Node	Color

*Private Message

Table 3.2. New AMOS Message.

Finally, diagnostic checks are used to detect errors in different

components in the computing resources. These diagnostic checks take the

form of test programs and are only called upon when a computing resource

receives the public message SELF TEST. At this time, the computing

resource has been removed from the system and it returns back only on the

successful completion of its self-test. The message SELF TEST is

originated outside the system and it is injected as a service message. Most

of the times that a computing resource is removed, the system performance

degrades accordingly as explained in Section 3.3.4 when a computing

resource fails. Hence, this self-testing is reserved for use under special

circumstances due to its penalty on the system performance.

3.3.2 Damage Confinement and Assessment in AMOS

The graph model used in ATAMM is an ideal means for damage

confinement. The Algorithm Marked Graph (AMG) determines the data

flow and structures how information traverses the system. The input data

or data packets enter the graph by the source node. They are directed

through the nodes by data directed paths and leave the system by the sink

node. This arrangement allows the nodes in the AMG to be the system

atomic actions. Starting by its firing, a node does not interact with any

other node until it completes executing the code and delivers its output.

The damage is confined to a node if the fault occurs after its firing and

before its completion.

The system damage confinement lends itself to static damage

assessment. After the system detects an error, a computing resource

generates a FAULT message. This message contains the information

about the node that did not complete its execution. The system damage

may be assessed as the partially processed data in the node since, after the

node firing, the assigned computing resource did not interact with any

other computing resource. The system uses this information in the 'Error

Recovery' phase to remove the error caused by the fault.

3.3.3 Error Recovery in AMOS

The technique used for error recovery is a forward error recovery

technique. The data that is partially processed in the node that did not

complete is discarded. The node is considered as if it was not started and gets reassigned to a healthy computing resource, in particular, to the one at the top of the available resources queue. The input data to the node has been kept in a reserved cache in the event of a fault. This data are then read by the newly assigned computing resource as the data to be used in the execution of the node.

This technique is called 'rollback' of the node. It basically restarts a process anew, with all its conditions as they were in the first attempt. For this reason, it may seem possible to classify the technique as a backward error recovery technique, but the conditions that are restored pertain only to the node and not to the entire system. The system is then placed into a healthy state from which it continues to work and delivers its services.

3.3.4 Fault Treatment and Continued System Service in AMOS

The fault that is assumed is death of a resource during execution of a node. The error detection technique chosen is sufficient to detect this fault. Nevertheless, other types of faults may manifest in the same fashion. If the code in a computing resource gets corrupted by any means, the manifestation of this fault may as well be the same as the computing resource death. This fact leads to the conclusion that the timing checks used may detect errors caused not only by the death of a computing resource but also by other types of faults that lead to the same erroneous state. Regardless of this expansion in the types of faults that are covered,

the only assumption when an error is detected is that the computing resource has ceased to function. Trying to single out the specific fault that caused the error is out of the scope of this dissertation and out of the original specifications of the system. Although it is possible to perform a detailed examination of the resource that has failed, it is not exercised in the current implementation of AMOS.

After the location of the fault, the resource that was originally assigned to the node that did not complete is identified and its ID is removed from the healthy resources queue. By doing so, the fault is purged from the system and the computing resource cannot engage in any interaction with the semaphore to gain access to the graph. The computing resource is not allowed to participate in any global updates to the system.

Under the ATAMM design procedure, a system has different operating points that depend on the number of computing resources present. Since the number of resources changes after an error is detected, an appropriate change in the operating point is carried out. This operating point change is applied to the graph so that the graph works optimally with the new number of computing resources. All operating points are precalculated and downloaded along with the graph. There exists a table, known as the operating point table, that contains all the necessary information to take the graph from one operating point to another. This

process of changing the operating point is the last stage of the reconfiguration required for the system to return back to service.

As the foregoing description suggests, the reconfiguration used in AMOS is spontaneous as explained in Section 3.2.4. The system initiates the reconfiguration after the fault has been removed. It accomplishes the reconfiguration prompted totally by the internal event of error detection. It prepares the system to go back to service by using data available already in the system. This feature is valuable in the design of highly reliable computer systems. Hence it is desirable in such applications as space probes, real-time control systems and deep-sea unmanned submarines.

3.4 Summary

This chapter has been used to present the development of the multicomputer operating system AMOS. This development targeted the IBM GVSC system developed for NASA Langley Research Center. This operating system is message based and highly modular.

An overview of the different phases for a fault-tolerant computer system was presented. These phases are error detection, damage confinement and assessment, error recovery and fault treatment and continued service. AMOS was upgraded to a fault-tolerant system by adding these phases to the system kernel. The addition was relatively simple due to the operating system architecture and modularity.

CHAPTER FOUR

EXPERIMENTS

This chapter is intended to demonstrate the application of the theory developed in Chapter Two relating to the transient behavior of an ATAMM system under a fault. In this chapter experiments are run in which delay is introduced into the system by means of a fault. The evaluation is carried out by both simulation and actual GVSC hardware implementation.

This chapter is divided into five sections. The simulation development is explained in Section 4.1. Demonstration of the simulation is presented in Section 4.2. Performance and behavior corresponding to the hardware system when there are no faults introduced is presented. Two graphs are examined to evaluate the steady state behavior of the GVSC and simulation. The transient operation of the simulation is tested in Section 4.3 and compared to that of the hardware behavior. There is only one graph used in the testing and there are three faults introduced in the system. The simulation is also subjected to the same conditions and the output of both systems are compared. The comparisons performed in Section 4.2 and Section 4.3 are of two types: micro and macro. The micro comparison deals with the ordering of individual events in the execution of

77

the graph. The macro comparison uses the information at a more global

level. This comparison is at the level of TBO and TBIO for each data packet

that the systems generate. Section 4.4 is used to present twelve

experiments which are used to test the theory developed in Chapter Two. A

comparison between the theoretical and simulated experimental results is

shown at the end of Section 4.4. A summary of the chapter is presented in

Section 4.5.

4.1 Simulation Development

The objectives of this section are to present the features of the

simulation for the GVSC AMOS. These features should help in the

simulation of systems that use the ATAMM design approach, in particular

the GVSC. A useful characteristic of the simulation is that of using the

same information input as the hardware system, i.e., the simulation and

the hardware use a common graph or information language. This helps to

quickly use an algorithm of interest to go from operating the hardware to

using the simulation. Along with this feature is the reporting of the system

actions in a tractable format by both the hardware and the simulation with

the prospect of comparison of both outputs.

Another feature of the simulation is that of investigating the behavior

of algorithms that require more computing resources than available on the

hardware. Under the ATAMM design procedures there can be graphs

optimized to work with many more computing resources than available in an implementation of AMOS. Thus, it is useful to have a simulator that does not have the limitations of expensive hardware. In a simulation, the adding of more computing resources does not impose a high price tag; it simply requires more computing power in simulating the large number of necessary computing resources.

The GVSC AMOS code was required to be implemented in the ADA language. This requirement did not necessarily imply that the unique ADA language features or ADA run-time module had to be used. With this prerequisite in mind, the original code was generated in ANSI PASCAL. The choice of PASCAL derives from the fact that ADA is a superset of PASCAL, therefore, PASCAL is a common minimum denominator between both languages. Originally the code was translated into ADA as it was generated or updated. Near the end of completion of the code, it was decided that only the ADA version be used as the fully working code. This implied that all the changes had to be made directly to the ADA version. As a by-product of this arrangement, a working PASCAL version of the system was available to be used as an integral part of the simulation. This version was surrounded by objects in the sense of object-orientation. The version was moved to Turbo Pascal for Windows by Borland, which is a hybrid language. A hybrid language, as it is the case here, is a procedural language with object-oriented features.

Essentially the AMOS logic and data structures were preserved. The code that handles the message passing, the data structures and bus management were left intact. The addition to the AMOS code has been the simulation of a multicomputer environment in a single-processor system. This simulation has been achieved by creating window objects that contain the original AMOS code. These window objects are run one at a time by executing a method by the name 'run'. The parameter that is passed down to the method is the value of the global clock. The window object has the same basic states as a computing resource and it moves from one state to another according to internal parameters and the value of the global clock. The internal parameters that are used are the estimated times that the computing resource should spend in the different states and on the various operations in the system. Examples of these operations are the bus request, graph update and timers checking.

As part of the NASA contract, which was to integrate the AMOS code into the IBM GVSC hardware, IBM was required to generate a build tool. This tool creates the graph data structures as well as links the node code to be downloaded into the GVSC computers. This program takes a graph file and generates the AMOS internal data structures that represent the graph in the file. It also links the pieces of node code to the data structures. The node code is not necessary for a simulation, but the generation of the data structures is extremely useful. The build tool source code has been used to

create a modified build tool for the simulation. It creates the same data structures as the original build tool does from the same graph file. This has become another advantage of using the original AMOS code and helped the quick development of a reliable simulation.

The main purpose of a simulation is to explore the behavior of a system without using the system itself. Another use is to be able to change parameters in the simulation, an otherwise expensive or lengthy process in the hardware counterpart, and to observe the effect on the system output. One of these parameters is the number of computing resources accessible to the system. The expense of adding computing resources in the hardware is high, whereas in the simulation it is a matter of changing a parameter and using additional computing power in the simulation host system. This benefit allows the simulation of large graphs and the examination of the performance predictions derived from the ATAMM design procedure.

As a whole, the simulation has the potential to be used as a generic simulation for a multicomputer system executing a version of AMOS. There are parameters that are unique to the GVSC environment but they can be adjusted to simulate other different environments. The types of systems that can be simulated with this program are those that use the same logical structure and the same state diagram as explained in Chapter Three. This is considered potentially useful since currently there is only one AMOS implemented in a multicomputer system.

One added use of the simulation was as a debugging tool for helping in the hardware integration. During development, pieces of code were integrated one at a time. The code to change the operating point was integrated last. This code involves the changing of the graph at run-time and therefore is critical to the fault tolerance phase of bringing the system back to service. The simulation was used to debug the operating point change code that eventually went into the GVSC system. The complexity of debugging this piece of code in the target system would have been an arduous and long enterprise because it is an embedded system. Debugging the code with the simulation was user-friendly because it could be run instruction by instruction through the operating point change. A particular feature was programmed for this purpose. The entire graph data structure can be examined any time during simulation. Also a snapshot of these data structures can be written or appended to a file any time. With this property, it was possible to examine the graph data structures before and after an operating point change was carried out. As a result of this simulation process the code was highly debugged when it was integrated into the hardware system code.

4.2 Simulation Validation Experiments

The objective of this section is to validate the simulation of the GVSC system running under normal conditions, i.e., no fault is injected to the

system. The validation process is carried out by taking output from the

hardware system and comparing it to the output from the simulation

running under the same conditions. Two graphs are used to accomplish

this task and are shown in Figures 4.1 and 4.2. The first graph is referred

to by the name of 'Intermediate Graph' and the second by the name of

'Application Algorithm', as they were used internally at NASA.

The process of comparison is accomplished by comparing the output

file of both hardware and simulation. The output files from both systems

are called fdt files. These files contain the sequence of events that took

place in the system while executing an algorithm graph. These events refer

to the firing of transitions in the graph. A typical event is:

10904 Fire Task4 1 Proc1 4.
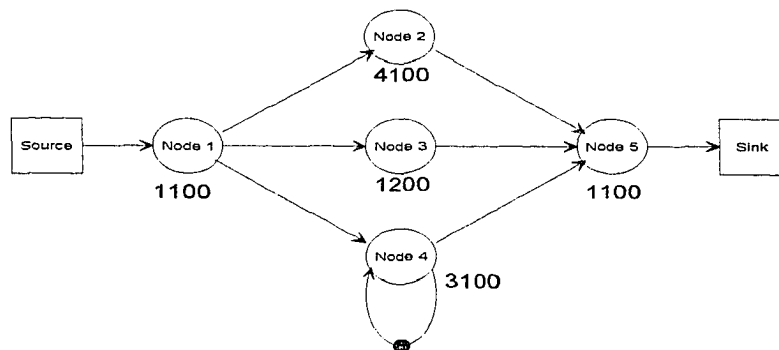
Figure 4.1. Intermediate Graph.

The first number is the value of the system clock when the event took

place. The 'Fire' keyword is the name of the event. The word 'Task4' refers

to the name of the node in the graph where the event was carried out. The

following number is the position or color designation in a TMR (Triple

Modular Redundancy) configuration. The word 'Proc1' identifies the name
of the processor that executed the event. The final figure identifies the data
packet number.
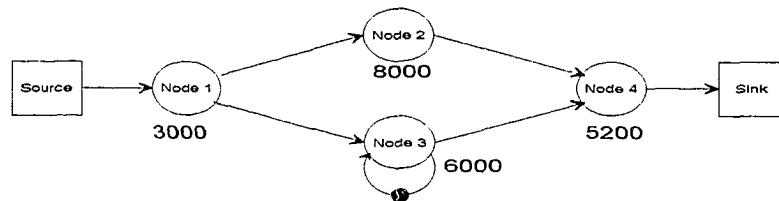
Figure 4.2. Application Algorithm Graph.

The events that concern the nodes are Fire, BeginNode, EndNode,
SentOutData and Data. There are other events that signal the request and
release of the communications channel. Other events are Fault and Refire
which are used in the process of detecting a faulty processor and restarting
the affected node.

There are two types of comparison, micro comparison and macro
comparison. The micro comparison is the comparison that is carried out at
the event level. The macro comparison is the one that is performed at the
performance level by comparing TBO and TBIO for each data packet. For
the micro comparison a program was written to compare two fdt files at the
event level. For the macro comparison, the Analyzer, written by Rob Jones
from NASA Langley Research Center, was used to measure the system
performance at every data packet.

The micro comparison is performed with the help of a C program that,
by using an fdt file as a reference and disregarding the time, reads an

event and tries to find the same event in the subject fdt file. For each event

in the reference file an output is generated that specifies the relative

position of the same event in the subject file. For example, if the event X

is found at position 52 in the reference file and the same event is found at

position 52 in the subject file an output of 0 is generated for that event. If

the event were to be found in position 53 then the output would be 1; if it

were in position 51, it would be -1 and so on. The range that the events are

searched on the subject file is limited to ±5 for the first two comparisons of

nonfault conditions and to ±20 for the comparison in a faulted condition. If

an event is not found within the specified range it is considered a miss. The

match of events should be perfect, i.e., everything, except the time, should

be identical. If an event is carried out in the reference file by processor 1

but on the subject file is carried out by processor 2, it is considered a miss.

A looser comparison is performed in the next section. The processor

assignment is made a don't care condition. The reason for such a

comparison is explained in the next section.

The macro comparison is done with the help of the Analyzer. This

comparison is performed at the performance level of the system for each

data packet. The measures used are TBO and TBIO for each data packet.

TBO is measured with respect to the predecessor data packet, i.e., the

difference between a data packet output time and the output time of the

predecessor data packet. The data is tabulated per data packet in an Excel

worksheet and graphed in an Operating Point Plane Figure. The Operating Point Plane Figure has TBIO as the x axis and TBO as the y axis. For each data packet a point is plotted at the intersection of the values of TBO and TBIO. A line is traced from that plotted point to the next point until all points are plotted. This Figure shows the dynamic nature of the system as it moves from data packet to data packet in the operating point plane.

4.2.1 First Comparison

The first comparison is performed using the Intermediate Graph of Figure 4.1 as the testing graph. The fdt file from the hardware is named inter1c and the one from the simulation is named inter1cs. There are only 281 events present and 6 data packets are output from the system. The file inter1c was generated when the IBM GVSC system was under test. The simulation was set with the same graph and the same time values as the hardware test. In the following sections the two comparisons are explained, first the macro comparison and then the micro comparison.

4.2.1.1 Micro Comparison

In the micro comparison, the file inter1cs was compared against inter1c. The results of the comparison are shown in Table 4.1 and a graph of the data is shown in Figure 4.3. It should be noted that there were no misses in the comparison. This means that all events in the reference file

Positio	Number	Percent
-5	0	0.00%
-4	0	0.00%
-3	0	0.00%
-2	8	2.85%
-1	18	6.41%
0	235	83.63%
1	11	3.91%
2	6	2.14%
3	2	0.71%
4	0	0.00%
5	1	0.36%

Table 4.1. Results of Micro Comparison #1.

Comparison of Hardware and Simulation #1
Intermediate Graph

Reference file: inter1c.fdt
Subject file: inter1cs.fdt
Total # of events = 281
Total # of misses = 0
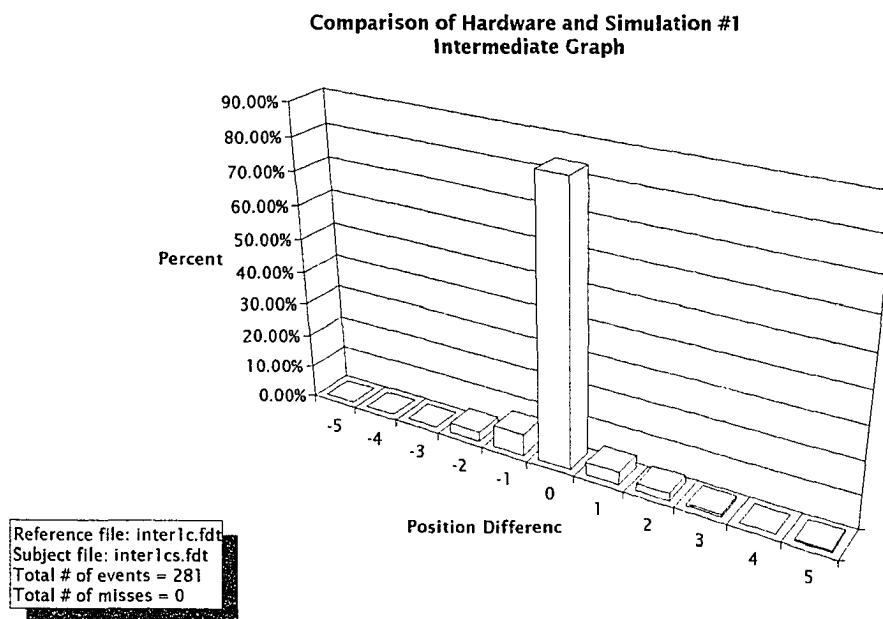
Figure 4.3. Results of Micro Comparison #1.

were found in the subject file within the specified range of ±5 positions.

Almost 84% of the events were found in the same position in both files.

Another 10% of the events were found in the -1 and +1 positions. Another

5% of the events were found in the -2 and +2 positions. In summary, 99% of

the events were located in the ±2 range. This comparison indicates that the

simulation is extremely close to the hardware in the order that the events are generated. The difference lies in that for a given set of events, they may take place in certain order in the hardware and in another order in the simulation program. For instance, if two or more nodes are assigned to computing resources in a graph update, the nodes may start executing in a different order in time in the hardware compared to the nodes in the simulation program. This can be observed in the fact that the differences are mostly in one or two positions.

4.2.1.2 Macro Comparison

The macro comparison involves the comparing of the values of TBO and TBIO at every data packet produced by the systems. The Table 4.2 contains the performance measures for both files. The Figure 4.3 shows the plotting of the values of TBO and TBIO. It should be noted that for the very first of the data packets the difference is large due to initialization differences. The hardware was programmed to start injecting input data at 10000 clock ticks, whereas the simulation started at 0 clock ticks. After the first data packet the largest difference is in the order of only 1.38%. The comparison yields a great similarity of the output of the simulation to that of the hardware.

After these two comparisons it can be seen that the simulation results are in very close agreement to the results from the hardware. It does so at

Packet	Inter1c			Inter1cs		
	TBI	TBO	TBIO	TBI	TBO	TBIO
1	10258	16855	6597	351	6891	6540
2	3054	3019	6562	3012	2983	6511
3	3053	3050	6559	3042	3042	6511
4	3059	3061	6561	3027	3027	6511
5	3060	3058	6559	3042	3042	6511
6	3061	3063	6561	3027	3027	6511

Table 4.2. Results of Macro Comparison #1.
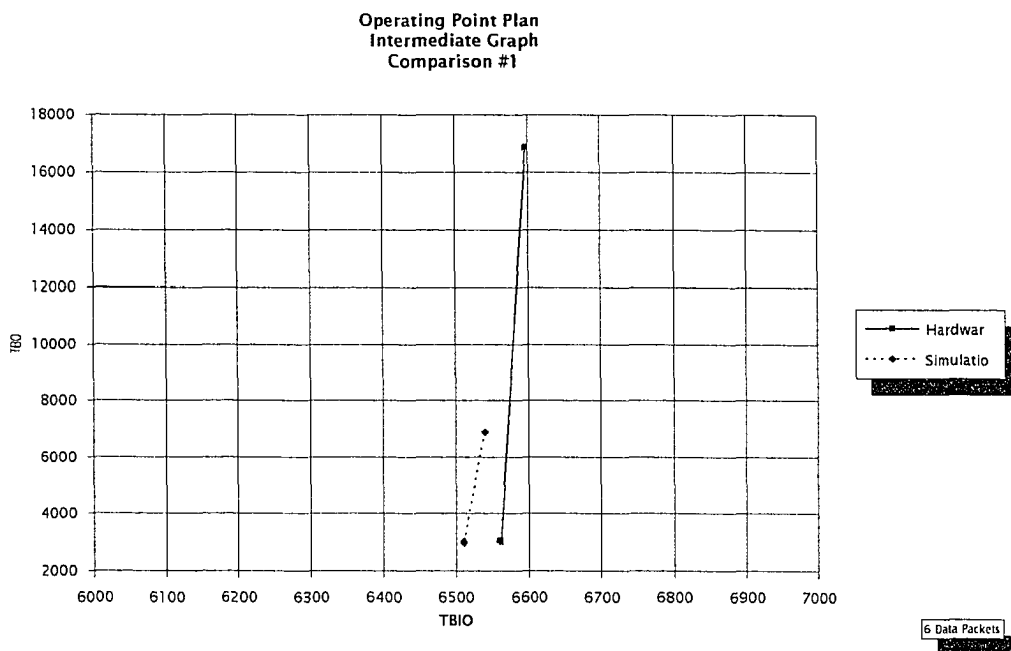
Figure 4.4. Results of Macro Comparison #1.

both the micro and the macro levels for the intermediate graph. The files

have been generated without introducing any faults into the system.

4.2.2 Second Comparison

The second comparison is performed using the Application Algorithm

as the testing graph. The fdt file from the hardware is named aatest2 and

the one from the simulation is named aatest2s. There are only 224 events

Positio	Number	Percent
-5	0	0.00%
-4	0	0.00%
-3	0	0.00%
-2	2	0.89%
-1	5	2.23%
0	215	95.98%
1	0	0.00%
2	1	0.45%
3	0	0.00%
4	0	0.00%
5	0	0.00%

Table 4.3. Results of Micro Comparison #2.
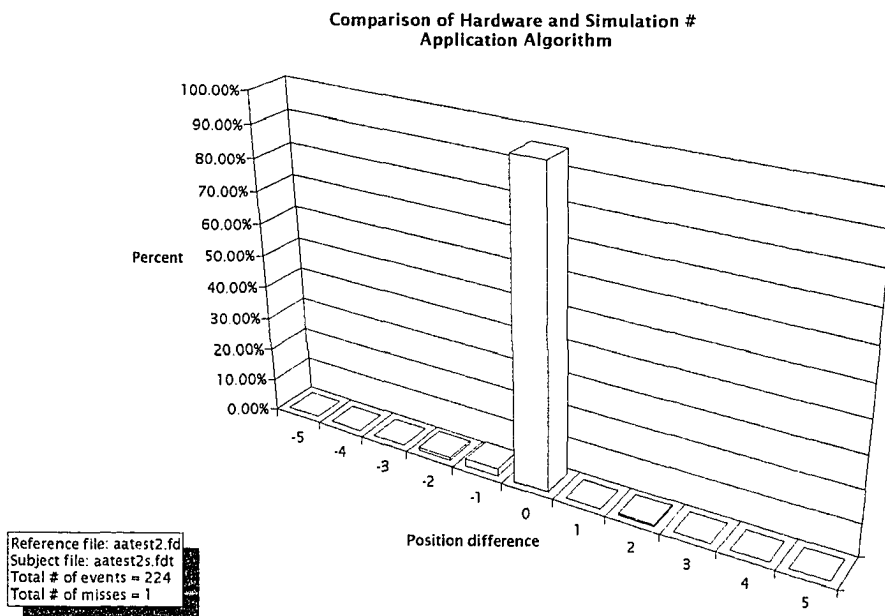
Figure 4.5. Results of Micro Comparison #2.

present and 8 data packets are output from the system. The file aatest2

was also generated when the IBM GVSC system was under test. The

simulation was set with the same graph and the same time values as the

hardware test. In the following sections the two comparisons are explained,

first the macro comparison and second the micro comparison.

4.2.2.1 Micro Comparison

In the micro comparison, the file aatest2 was compared against

aatest2s. The results of the comparison are shown in Table 4.3 and a graph

of the data is shown in Figure 4.5. It should be noted that there was only

one miss in the comparison. This means that of all events in the reference

file only one was not found in the subject file within the specified range of

±5 positions. Almost 96% of the events were found in the same position in

both files. Another 2% of the events were found in the -1 position. Another

1.15% of the events were found in the -2 and +2 positions. In summary,

Packet	Aatest2			Aatest2s		
	TBI	TBO	TBIO	TBI	TBO	TBIO
1	10288	27098	16810	351	17001	16650
2	6033	5884	16661	6018	5995	16627
3	6035	6036	16662	6004	6004	16627
4	6037	6037	16662	6039	6039	16627
5	6040	6041	16663	6004	6004	16627
6	6034	6031	16660	6039	6039	16627
7	6035	6040	16665	6004	6004	16627
8	6038	6036	16663	6039	6039	16627

Table 4.4. Results of Macro Comparison #2.

Operating Point Plane
Application Algorith
Comparison #2
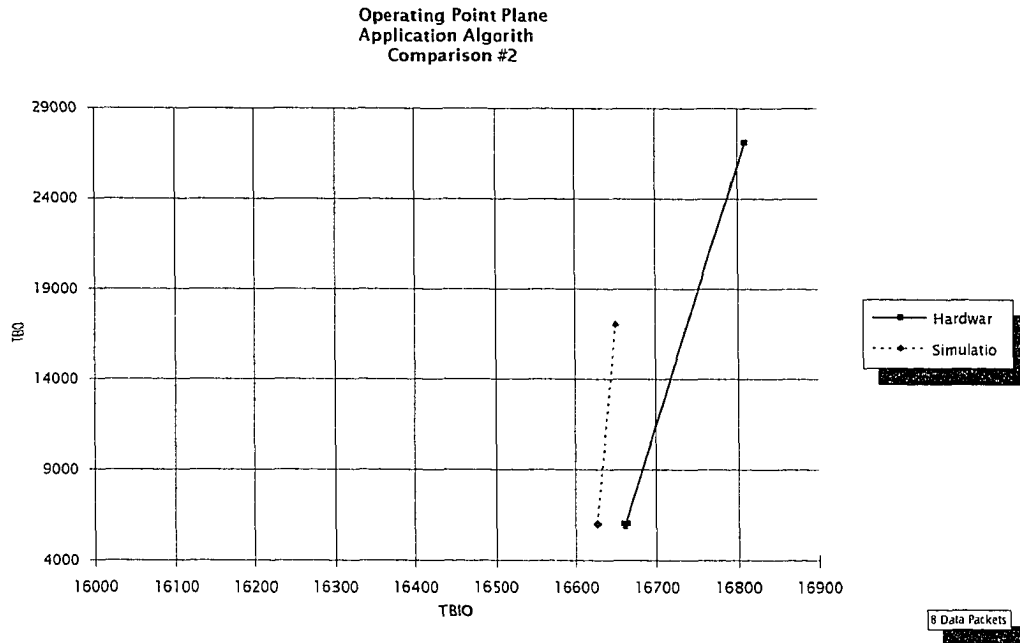
Figure 4.6. Results of Macro Comparison #2.

over 99% of the events were located in the ±2 range. This comparison

indicates that the simulation is extremely close to the hardware in the

order that the events are generated.

4.2.2.2 Macro Comparison

The macro comparison involves the comparing of the values of TBO

and TBIO at every data packet produced by the systems. Table 4.4 contains

the performance measures for both files. Figure 4.6 shows the plotting of

the values of TBO and TBIO. It should be noted that for the very first of

the data packets the difference is large due to initialization differences.

The hardware was programmed to start injecting input data at 10000 clock

ticks, whereas the simulation started at 0 clock ticks. After the first data

packet the largest difference is in the order of only 1.89%. The comparison

yields a great similarity of the output of the simulation to that of the

hardware.

After these two comparisons it can be seen that the simulation

generates very close results to the hardware. It does so at both the micro

and the macro levels for the intermediate graph. The files have been

generated without introducing any faults into the system.

4.2.3 Summary

The two comparisons presented in this section validate the simulation

program as a close simulation of a system with a multicomputer operating

system such as AMOS. The results were very close considering the many

variables that are used to represent the system's behavior. The maximum

difference was in the order of less than 2% in the macro comparison and

99% of the events were in the range of ±2 in the micro comparison. This

only validates the program for the normal conditions where there is no

fault introduced into the system during execution.

4.3 Fault Transient Validation

The objective of this section is to validate the simulation under

transient conditions as those encountered during fault detection and

correction. The procedure is similar to the preceding section. There is a

micro comparison and a macro comparison. The only difference is that

there is only one graph compared due to lack of suitable data at the present. The graph to be used is the Application Algorithm and there are three faults introduced in the system. The system executed for 35 data packets. The first fault was introduced in node 2 at data packet 10. The second fault was introduced also in node 2 at data packet 15. The third and last fault was introduced in node 1 at data packet 25. There were a total of 1167 events.

The original test was executed in the hardware to debug the code to detect and recover from a fault. The original file is named aatest and the simulation output is named aatests. The system had an optimized operating point table that was generated by the team at NASA. Each one of the operating points was optimal for the given number of processors present. The system underwent an operating point change each time a resource failed and was removed from the system. The system started with 4 resources and dropped down to 1 resource after the third fault.

The simulation was set with the same graph and timing information as the hardware. It also contained the same operating point table the hardware had during the test. This test is more critical since it validates the transient behavior of the simulation.

4.3.1 Comparison

After the first fault, assignment of processors to nodes changed with respect to that in the hardware experiment. This is not considered as critical since the assignment is performed dynamically and on-line. It should not make a difference which process gets assigned to what processor since what is important that the nodes get executed in the same way as in the hardware. For this reason the micro comparison disregards the processor assignment and seeks only the sequence of the firing of the transitions in the graph.

Comparison of Hardware and Simulation #
Application Algorithm

Reference file: aatest.fdt
Subject file: aatests.fdt
Total # of events = 116
Total # of misses = 0
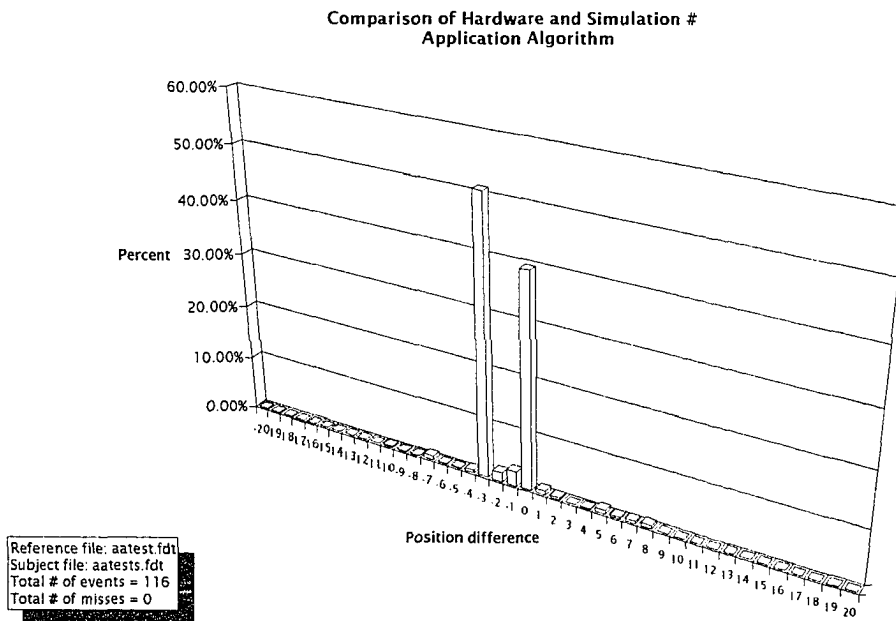
Figure 4.7. Results of Micro Comparison #3.

Position	Number	Percent
-20	0	0.00%
-19	0	0.00%
-18	0	0.00%
-17	0	0.00%
-16	0	0.00%
-15	0	0.00%
-14	0	0.00%
-13	0	0.00%
-12	0	0.00%
-11	0	0.00%
-10	3	0.26%
-9	1	0.09%
-8	1	0.09%
-7	9	0.77%
-6	1	0.09%
-5	1	0.09%
-4	6	0.51%
-3	594	50.90%
-2	20	1.71%
-1	32	2.74%
0	460	39.42%
1	9	0.77%
2	4	0.34%
3	0	0.00%
4	1	0.09%
5	9	0.77%
6	4	0.34%
7	5	0.43%
8	7	0.60%
9	0	0.00%
10	0	0.00%
11	0	0.00%
12	0	0.00%
13	0	0.00%
14	0	0.00%
15	0	0.00%
16	0	0.00%
17	0	0.00%
18	0	0.00%
19	0	0.00%
20	0	0.00%

Table 4.5. Results of Micro Comparison #3.

4.3.1.1 Micro Comparison

In the micro comparison, the file aatest was compared against aatests. The results of the comparison are shown in Table 4.5 and a graph of the data is shown in Figure 4.7. It should be noted that there were no misses in the comparison. This means that all events in the reference file were found in the subject file within the specified range of ±20 positions. Almost 39.5% of the events were found in the same position in both files. Another 3.5% of the events were found in the -1 and +1 positions. Another 2% of the events were found in the -2 and +2 positions. Another 50.9% of the events were found in the -3 position. In summary, 96% of the events were located in the ±3 range. This comparison indicates that the simulation is extremely close to the hardware in the order that the events are generated in spite of the three transients introduced in the form of faults. The difference is found mostly in the sections of events close to the time of the faults. Accurate simulation of when the fault is detected and when other processor finish is critical. While a processor is changing operating point and removing the faulty processor from the system, other processor may have finish executing assigned nodes. These other processors cannot access the graph since the semaphore has been acquired by the processor that responded to the FAULT message. Which processor grabs the semaphore is released, affects the order in which the events are registered in the system. These displaced events may upset the order of the events on the simulation with

respect to the hardware behavior. After this phenomenon takes place, the difference will remain through the rest of the execution since there is no resetting of the ordering while comparing both files. This may be observed in Table 4.5, position -3.

4.3.1.2 Macro Comparison

The macro comparison involves the comparing of the values of TBO and TBIO at every data packet produced by the systems. Table 4.6 contains the performance measures for both files. Figure 4.8 shows the plotting of the values of TBO and TBIO. After the first data packet and the last data packet the largest difference is on the order of only 3.47%. The comparison yields a great similarity of the output of the simulation to that of the hardware.

The simulation is validated by this comparison since, after being subject to three transients, the performance measures are still in very close agreement with the hardware. These tests are enough for the purposes of this dissertation because the Application Algorithm graph is the one to be used in the section where the experiments are carried out. The simulation follows the behavior of the hardware even under faults.

Figure 4.6 contains an additional line and set of points. This line is identified as Theoretical. These points are the operating points theoretically calculated to generate the operating point table that went into

Packet	aatest			aatests		
	TBI	TBO	TBIO	TBI	TBO	TBIO
1	10288	27098	16810	351	17001	16650
2	6033	5884	16661	6018	5995	16627
3	6035	6036	16662	6004	6004	16627
4	6037	6037	16662	6039	6039	16627
5	6040	6041	16663	6004	6004	16627
6	6034	6031	16660	6039	6039	16627
7	6035	6040	16665	6004	6004	16627
8	6038	6036	16663	6039	6039	16627
9	6037	6036	16662	6004	6083	16706
10	6035	14892	25519	6039	14654	25321
11	6041	5466	24944	6004	5435	24752
12	11108	8188	22024	10722	8160	22190
13	8038	8688	22674	8009	8646	22827
14	8040	8190	22824	8008	8160	22979
15	8043	17620	32401	8038	17382	32323
16	8039	6132	30494	8020	6059	30362
17	8040	11653	34107	8106	11572	33828
18	17067	11661	28701	16896	11572	28504
19	11654	11653	28700	11572	11572	28504
20	11661	11661	28700	11572	11572	28504
21	11653	11653	28700	11572	11572	28504
22	11661	11678	28717	11572	11572	28504
23	11653	11633	28697	11572	11572	28504
24	11661	11783	28819	11572	11572	28504
25	11502	26036	43353	11572	25978	42910
26	11809	22912	54456	11572	22842	54180
27	26167	22912	51201	25978	22842	51044
28	22912	22912	51201	22842	22842	51044
29	22912	22912	51201	22842	22842	51044
30	22912	22912	51201	22842	22842	51044
31	22912	22912	51201	22842	22842	51044
32	22912	22912	51201	22842	22842	51044
33	22912	22911	51200	22842	22842	51044
34	22912	22870	51158	22842	22842	51044
35	22912	19754	48000	22842	22842	51044

Table 4.6. Results of Macro Comparison #3.

Operating Point Plane
Application Algorith
Comparison #3
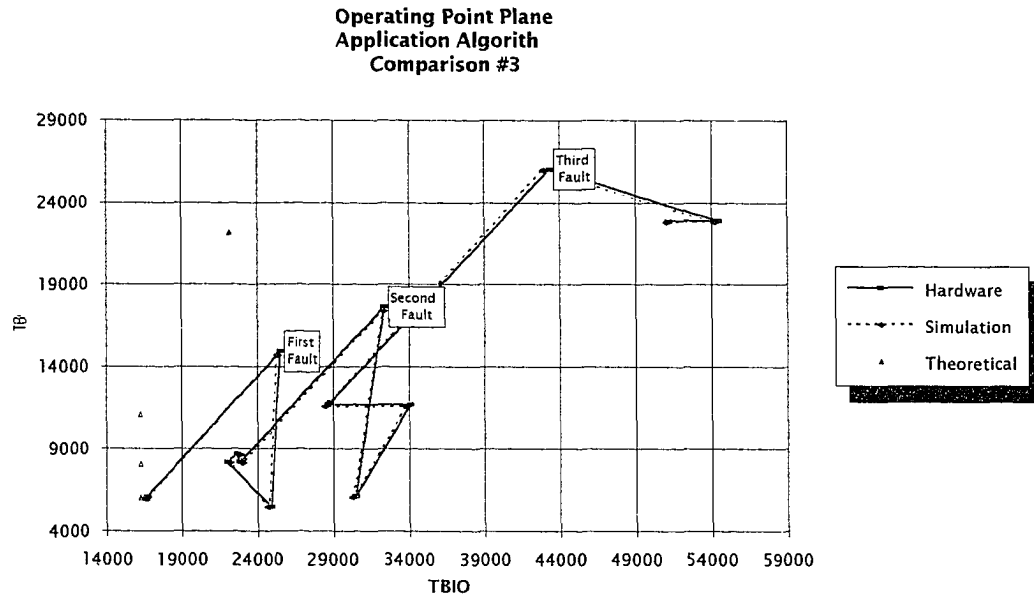
Figure 4.8. Results of Macro Comparison #3.

the hardware and the simulation for these tests. As it can be observed, the

behavior of the hardware as well as that of the simulation deviated

drastically from the desirable operation. It is important to highlight that

this is a good example of the significance of the findings in this

dissertation. The system did not operate as it was expected because it did

not have the means to absorb the delays introduced by the faults into the

graph and hence did not reach the target operating points. As explained in

Section 4.4, there are ways to alleviate this anomalous operation and is

demonstrated with examples how the delay can be absorbed in the system.

It should be pointed out that in the examples in Section 4.4 there is no

change of an operating point to another operating point with less

resources. The operating point is maintained between faults to highlight the effect of the delay introduced by the fault and the effect of the token lifetimes in the graph.

4.4 Simulation Examples

4.4.1 Overview

The objective of this section is to present examples of a graph

undergoing a single fault, recovering from the fault, and continuing with

the execution of the algorithm. The comparison of calculated and

simulated results are performed at a macro level, i.e., at the TBI, TBO and

TBIO level. Therefore, lifetime dominant paths are found by the means

developed in Chapter 2. The paths are found from the node where the

fault occurs to every single sink in the XCMG to calculated delay to fire the

sinks. With the values of the lifetime of these paths the values for TBO

and TBIO for every subsequent data packet are calculated. Since these

computations are rather involved to be presented in this section, an

example is detailed in Appendix A. All experiments were run under the

same conditions and the actual TBO and TBIO were retrieved with the

help of the Analyzer. These values are compared against the calculated

values and conclusions are drawn from the comparisons.

The graph used in the experiments is the Application Algorithm.

There are twelve experiments where the system is operated at TBI = 7000

TBI	Failure @			
	Node 2	Node 3	Node 1	Node 4
7000	Exp. #1	Exp. #2	Exp. #3	Exp. #4
6200	Exp. #5	Exp. #6	Exp. #7	Exp. #8
8000	Exp. #9	Exp. #1	Exp. #1	Exp. #1

Table 4.7. Summary of All Experiments.

for the first four, at TBI = 6200 for the second four, and at TBI = 8000 for the third four. A different node is made to fail in each one of the experiments in the groups of four. Table 4.7 is a summary of the conditions of all the experiments.

All experiments are run for thirty data packets and all faults are injected at data packet ten. An overview of the experimental results is presented in a table in Section 4.4.2. For each one of the experiments there is a chart showing the experimental values of TBI, TBO and TBIO for each data packet. Also, a table is presented with experimental and calculated values of TBI, TBO and TBIO, and the percentage of error of the calculated with respect to the experimental for each one of the experiments.

There are three comparison charts of all experiments with the same input rate or TBI. These charts are placed after all the charts and tables for the experiments. These comparison charts show the Operating Point Plane of all four experiments with the same TBI. These figures show the behavior of the system under the fault and the recovery process. There are also comparison charts of all experiments with the same faulty node. These charts present the individual data packets plotted in an Operating Point Plane for the system.

The value of TBO_{LB} of the system is 6200 which was found experimentally. It was necessary to know this bound so that the system would be driven at a higher or equal TBI. The resolution of the parameters

for source time in the hardware as well as in the simulation is in hundreds

of units. The actual value of TBO_{LB} is between 6100 and 6200. Thus for

the purpose of these experiments TBO_{LB} is considered at 6200.

The experimental value of $TBIO_{LB}$ of the system is 16650. The

amount of delay that is introduced into the system is approximately equal

to the process time of the faulty node plus 500. The 500 units extra is the

timeout delay that every node is allowed before it is declared faulty. Thus

the delay introduced by a fault at node 1 is 3500, by a fault at node 2 is

8500, by a fault at node 3 is 6500, and by a fault at node 4 is 5700.

4.4.2 Experiment # 1, an Example

This experiment is explained in detail to show the general process

that was followed for each one the experiments. Although the actual

numbers may be different for each experiment, the procedure followed to

calculate the values of TBI, TBO and TBIO are the same. The tables in the

experiments contain the values of TBI, TBO and TBIO that were measured

with the help of the Analyzer for each data packet. They also contain the

calculated values for TBI, TBO and TBIO, followed by a percentage of

error of the calculated versus the experimental values.

In experiment #1, node 2 fails executing data packet 10 and

introduces a delay of 8500. As it can be observed in Figure 4.7, both TBO

and TBIO are increased for that data packet. After the data packet 10, the

Application Algorithm,
Fault at Node 2, Data packet 1
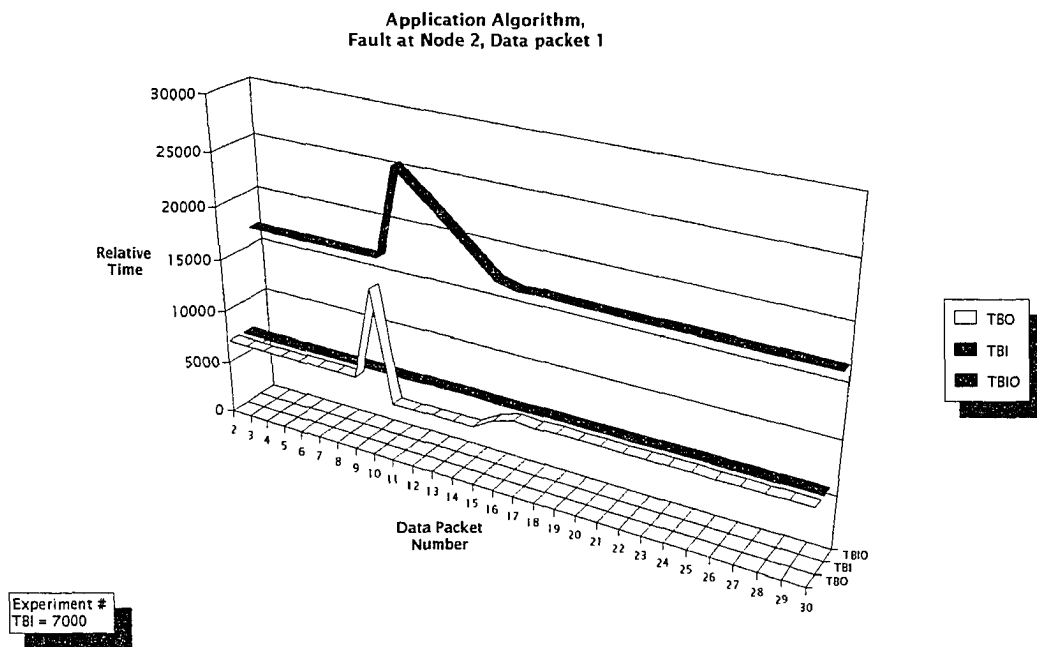
Figure 4.9. TBI, TBO and TBIO for Experiment #1.

system delivers outputs at a lower TBO than the TBI that is being used in

the system. This behavior continues for 6 data packets and the system

returns back to a TBO equal to the system TBI. From here on, this lower

TBO is referred to by the name of recovery TBO. TBIO decreases on each

data packet by the difference between the system TBI and the recovery

TBO. The value of TBI is 7022 and the recovery TBO is approximately

5385 which gives a reduction on TBIO of 1637 per data packet. This

response may be interpreted as the recovery process the system goes

through to reduce the delay introduced by the fault. Both TBO and TBIO

eventually return to the original values they had before the fault was

injected.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	351	17001	16650	0	16650	16650	100.00%	2.06%	0.00%
2	7030	7033	16653	7022	7022	16650	0.11%	0.16%	0.02%
3	7013	7010	16650	7022	7022	16650	-0.13%	-0.17%	0.00%
4	7030	7030	16650	7022	7022	16650	0.11%	0.11%	0.00%
5	7010	7013	16653	7022	7022	16650	-0.17%	-0.13%	0.02%
6	7033	7030	16650	7022	7022	16650	0.16%	0.11%	0.00%
7	7010	7010	16650	7022	7022	16650	-0.17%	-0.17%	0.00%
8	7033	7033	16650	7022	7022	16650	0.16%	0.16%	0.00%
9	7010	7010	16650	7022	7022	16650	-0.17%	-0.17%	0.00%
10	7033	15676	25293	7022	15665	25293	0.16%	0.07%	0.00%
11	7010	5409	23692	7022	5385	23656	-0.17%	0.44%	0.15%
12	7038	5386	22040	7022	5385	22019	0.23%	0.02%	0.10%
13	7010	5383	20413	7022	5385	20382	-0.17%	-0.04%	0.15%
14	7061	5386	18738	7022	5385	18745	0.55%	0.02%	-0.04%
15	7010	5383	17111	7022	5385	17108	-0.17%	-0.04%	0.02%
16	7006	6548	16653	7022	6564	16650	-0.23%	-0.24%	0.02%
17	7014	7122	16761	7022	7022	16650	-0.11%	1.40%	0.66%
18	7029	6921	16653	7022	7022	16650	0.10%	-1.46%	0.02%
19	7034	7034	16653	7022	7022	16650	0.17%	0.17%	0.02%
20	7009	7009	16653	7022	7022	16650	-0.19%	-0.19%	0.02%
21	7037	7034	16650	7022	7022	16650	0.21%	0.17%	0.00%
22	7006	7009	16653	7022	7022	16650	-0.23%	-0.19%	0.02%
23	7037	7037	16653	7022	7022	16650	0.21%	0.21%	0.02%
24	7006	7006	16653	7022	7022	16650	-0.23%	-0.23%	0.02%
25	7037	7037	16653	7022	7022	16650	0.21%	0.21%	0.02%
26	7006	7006	16653	7022	7022	16650	-0.23%	-0.23%	0.02%
27	7040	7037	16650	7022	7022	16650	0.26%	0.21%	0.00%
28	7003	7006	16653	7022	7022	16650	-0.27%	-0.23%	0.02%
29	7040	7040	16653	7022	7022	16650	0.26%	0.26%	0.02%
30	7003	7003	16653	7022	7022	16650	-0.27%	-0.27%	0.02%

Table 4.8. Experimental and Calculated Values of TBI,
TBO and TBIO for Experiment #1.

In order to calculate the values of TBI, TBO and TBIO, the paths between node 2 and all sinks were identified and their token lifetimes were computed. The value of TBI was estimated to be the mean value of the experimental values of TBI. TBIO for each data packet was calculated by first estimating the delay to fire each one of the sinks in the XCMG and adding this delay to $TBIO_{LB}$. The values of TBO were calculated by

finding the difference between the times when the sinks fired using the first data packet before the fault as reference. An example of this procedure is detailed in Appendix A. The experimental and calculated values and the percentage of error of TBI, TBO and TBIO are tabulated in Table 4.8. Discarding the error for packet one, the maximum difference is about 1.46% and most of the values are below 0.5%.

4.4.3 Experimental Results

A summary of the experimental results is tabulated in Table 4.9 and Table 4.10. Table 4.9 contains, for each one of the experiments, the node that failed; the average value of TBI, the recovery TBO; the delay introduced by the fault and the delay to the first output after the fault. Table 4.10 contains, for each one of the experiments, the node that failed; the reduction on TBIO per data packet; the number of packets the system

Exp. Number	Faulty Node	TBI	Recovery TBO	Introduced Delay	First Output Delay
1	2	7023	5385	8643	8643
2	3	7023	6198	6520	4720
3	1	7023	5385	3577	3577
4	4	7023	5385	5811	5811
5	2	6236	5385	8727	8727
6	3	6236	6236	6614	4814
7	1	6236	5385	3628	3628
8	4	6236	5385	5846	5846
9	2	8036	5385	8659	8659
10	3	8036	6185	6455	4655
11	1	8036	5385	3569	3569
12	4	8036	5385	5831	5831

Table 4.9. Summary #1 of Experimental Results.

takes to recover; the value of a permanent delay after reaching steady

state; and the time to restore the system to the target operating point.

The recovery TBO values in Table 4.9 refer to the TBO at which the

system delivered outputs while it was recovering from the fault. After

faults at all nodes, except at node 3, the recovery TBO reflects

approximately the value of process time for node 4, which is 5200. This is

because the lifetime dominant path for faults at these nodes is the path

that traverses node 4 through several data packets in the XCMG. The

lifetime dominant path when node 3 fails traverses node 3 and the recovery

TBO is higher; its value is TBO_{LB}.

The introduced delay, as shown in Table 4.9, is the effective delay

introduced into the system by the fault at the node. As can be seen, it is

approximately the value of the process time for the node that failed plus

Exp. Number	Faulty Node	TBIO Reduction	Recovery Packets	Permanent Delay	Time to Restore
1	2	1638	6	0	42138
2	3	825	6	0	42138
3	1	1638	3	0	21069
4	4	1638	4	0	28092
5	2	851	N/A(8)	2083	N/A(6560)
6	3	0	N/A(0)	4720	N/A(0)
7	1	851	N/A(3)	1432	N/A(2460)
8	4	851	7	0	43652
9	2	2651	4	0	32144
10	3	1851	3	0	24108
11	1	2651	2	0	16072
12	4	2651	3	0	24108

Table 4.10. Summary #2 of Experimental Results.

500 units. Any difference can be attributed to the effective timeout at execution time. It is possible that an error may have been detected but the communications channel was being used at the time and the actual time to operate on the graph may have been longer. This effectively adds delay to the estimated time.

The column of first output delay in Table 4.9 reflects the delay on the first data packet or the data packet 10 in which the fault occurred. In all nodes and TBI, except for node 3, the first output delay is equal to the introduced delay. The difference in the rows of node 3 is due to a token lifetime of approximately 1800 units in the path from node 3 to the sink for data packet 10. The theoretical value of this token lifetime in the path is 2000 units.

The TBIO reduction column in Table 4.10 indicates the amount of delay reduction that is applied to the TBIO for each data packet while the system is in recovery. This value can be calculated by subtracting recovery TBO from TBI.

The column of recovery packets in Table 4.10 denotes the number of packets that the system requires to reach the target operating point. This target operating is the initial operating point before the fault, i.e., the same values of TBO and TBIO. For faults at nodes 2, 3 and 1, and TBI of 6236 there is no value in this column. This is because the system never reaches

the target operating point. Instead it reaches an operating point with an offset in TBIO, hence a permanent delay in TBIO. The system does absorb some of the delay that is introduced by the fault, but the path that contains node 3 becomes lifetime dominant after a given number of data packets. This number of data packets is expressed between parentheses.

The column of permanent delay indicates the amount of delay that exists in TBIO after the system has reached a steady state. Most of the entries are zero, except for the related entries denoted by N/A in the column of recovery packets.

The time the system takes to restore the target operating point is indicated in the column of time to restore. The entries indicated by N/A are the ones where the system never reaches the target operating point. Instead the value between parentheses is the time the system took to reach steady state.

As has been shown, the model can be used to predict the behavior of a multicomputer system under recovery and restoration. The issues that have been raised in the Introduction of this dissertation can be addressed. These issues are whether a system fully recovers when it undergoes a fault; the time it takes to recover (t_{rec}) from the fault and to restore the system (t_{res}); and the existence of a permanent delay in the system after it reaches steady state.

The data from experiments #2 to #12 are graphed and tabulated in figures and tables in the rest of this section. The last seven charts are Operating Point Planes of the experiments grouped by TBI or by faulty node. They serve to highlight different aspects of the system under study.

By observing these charts and tables, some conclusions may be drawn that highlight performance aspects of the system. The first conclusion is that if

$$TBI > TBO_{LB}$$

the system recovers from the fault and reaches the target operating point. The token lifetime in the paths from the faulty nodes to the sinks is used to absorb the delay introduced by the fault.

Another conclusion is that is if

$$TBI = TBO_{LB}$$

the system may not recover from the fault and may not reach the target operating point. If it does not reach the target operating point, there is a permanent delay added to TBIO.

The value of the recovery TBO depends on the node that fails and not on TBI. This value is related to the time in the nodes in the paths and not to the token lifetime as it is the case of the TBIO reduction. It may also be observed that the higher the TBI, the faster the system recovers. This information may be easily observed in Figures 2.21 to 2.27.
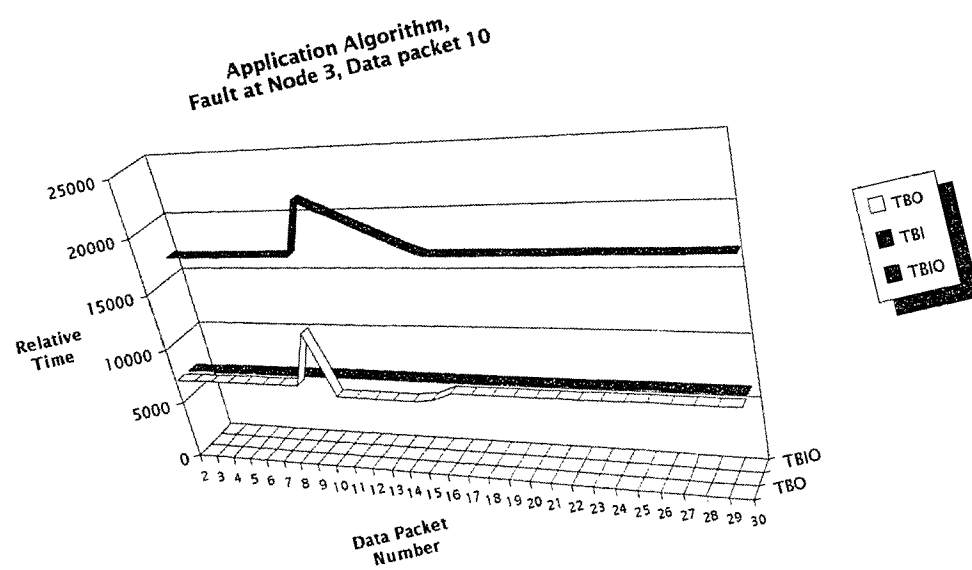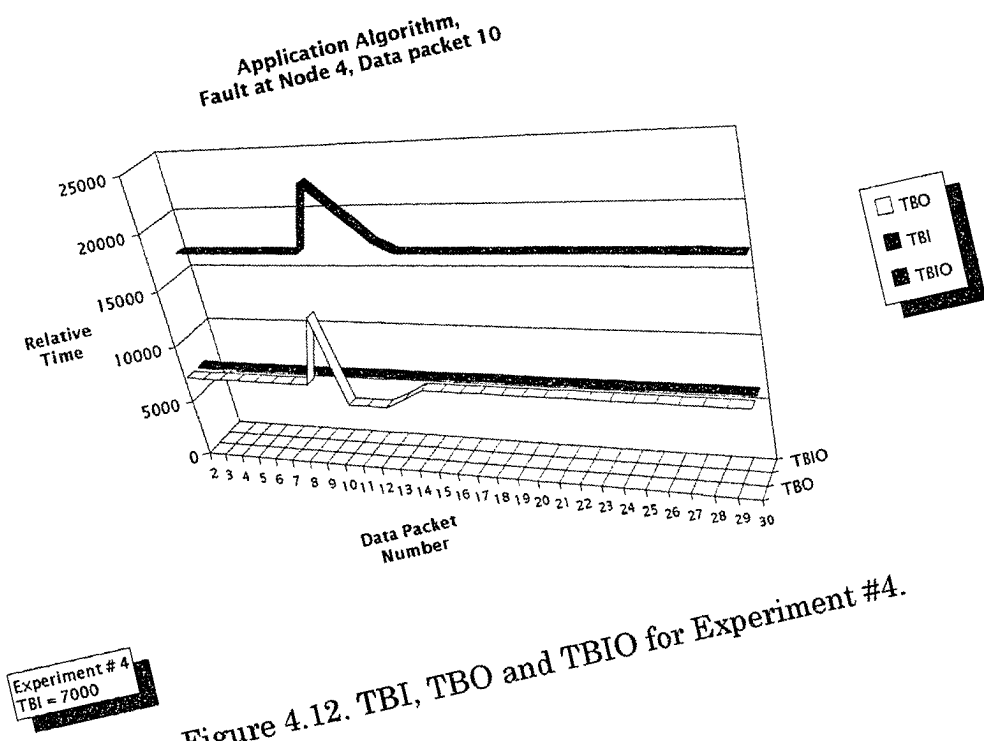
Figure 4.10. TBI, TBO and TBIO for Experiment #2.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	271	16924	16653	0	16650	16650	100.00%	1.62%	0.02%
2	7030	7027	16650	7023	7023	16650	0.10%	0.06%	0.00%
3	7016	7016	16650	7023	7023	16650	-0.10%	-0.10%	0.00%
4	7027	7030	16653	7023	7023	16650	0.06%	0.10%	0.02%
5	7016	7016	16653	7023	7023	16650	-0.10%	-0.10%	0.02%
6	7030	7027	16650	7023	7023	16650	0.10%	0.06%	0.00%
7	7016	7016	16650	7023	7023	16650	-0.10%	-0.10%	0.00%
8	7027	7030	16653	7023	7023	16650	0.06%	0.10%	0.02%
9	7016	7016	16653	7023	7023	16650	-0.10%	-0.10%	0.02%
10	7030	11750	21373	7023	11743	21370	0.10%	0.06%	0.01%
11	7016	6212	20569	7023	6198	20545	-0.10%	0.23%	0.12%
12	7004	6209	19774	7023	6198	19720	-0.27%	0.18%	0.27%
13	7020	6186	18940	7023	6198	18895	-0.04%	-0.19%	0.24%
14	7007	6183	18116	7023	6198	18070	-0.23%	-0.24%	0.25%
15	7035	6200	17281	7023	6198	17245	0.17%	0.03%	0.21%
16	7025	6368	16624	7023	6428	16650	0.03%	-0.94%	-0.16%
17	7019	7022	16627	7023	7023	16650	-0.06%	-0.01%	-0.14%
18	7042	7042	16627	7023	7023	16650	0.27%	0.27%	-0.14%
19	7019	7045	16653	7023	7023	16650	-0.06%	0.31%	0.02%
20	7042	7042	16653	7023	7023	16650	0.27%	0.27%	0.02%
21	7004	7001	16650	7023	7023	16650	-0.27%	-0.31%	0.00%
22	7039	7042	16653	7023	7023	16650	0.23%	0.27%	0.02%
23	7004	7004	16653	7023	7023	16650	-0.27%	-0.27%	0.02%
24	7042	7042	16653	7023	7023	16650	0.27%	0.27%	0.02%
25	7004	7004	16653	7023	7023	16650	-0.27%	-0.27%	0.02%
26	7039	7039	16653	7023	7023	16650	0.23%	0.23%	0.02%
27	7007	7004	16650	7023	7023	16650	-0.23%	-0.27%	0.00%
28	7036	7039	16653	7023	7023	16650	0.18%	0.23%	0.02%
29	7007	7007	16653	7023	7023	16650	-0.23%	-0.23%	0.02%
30	7039	7039	16653	7023	7023	16650	0.23%	0.23%	0.02%

Table 4.11. Experimental and Calculated Values of TBI,
TBO and TBIO for Experiment #2.
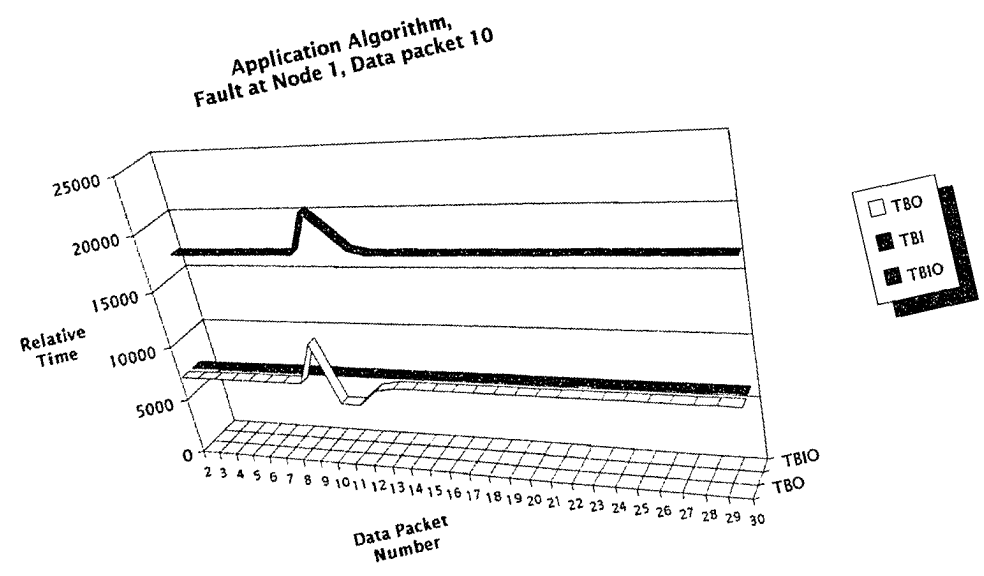
Figure 4.11. TBI, TBO and TBIO for Experiment #3.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	351	17001	16650	0	16650	16650	100.00%	2.06%	0.00%
2	7030	7033	16653	7000	7000	16650	0.43%	0.47%	0.02%
3	7013	7010	16650	7000	7000	16650	0.19%	0.14%	0.00%
4	7030	7033	16653	7000	7000	16650	0.43%	0.47%	0.02%
5	7013	7013	16653	7000	7000	16650	0.19%	0.19%	0.02%
6	7033	7033	16653	7000	7000	16650	0.47%	0.47%	0.02%
7	7013	7010	16650	7000	7000	16650	0.19%	0.14%	0.00%
8	7030	7033	16653	7000	7000	16650	0.43%	0.47%	0.02%
9	7013	7013	16653	7000	7000	16650	0.19%	0.19%	0.02%
10	7033	10610	20230	7000	10577	20227	0.47%	0.31%	0.01%
11	7002	5386	18614	7000	5385	18612	0.03%	0.02%	0.01%
12	7040	5383	16957	7000	5385	16997	0.57%	-0.04%	-0.24%
13	7028	6698	16627	7000	6653	16650	0.40%	0.67%	-0.14%
14	7014	7014	16627	7000	7000	16650	0.20%	0.20%	-0.14%
15	7003	7029	16653	7000	7000	16650	0.04%	0.41%	0.02%
16	7014	7014	16653	7000	7000	16650	0.20%	0.20%	0.02%
17	7035	7032	16650	7000	7000	16650	0.50%	0.46%	0.00%
18	7011	7011	16650	7000	7000	16650	0.16%	0.16%	0.00%
19	7032	7035	16653	7000	7000	16650	0.46%	0.50%	0.02%
20	7011	7011	16653	7000	7000	16650	0.16%	0.16%	0.02%
21	7032	7032	16653	7000	7000	16650	0.46%	0.46%	0.02%
22	7011	7011	16653	7000	7000	16650	0.16%	0.16%	0.02%
23	7038	7035	16650	7000	7000	16650	0.54%	0.50%	0.00%
24	7008	7008	16650	7000	7000	16650	0.11%	0.11%	0.00%
25	7035	7038	16653	7000	7000	16650	0.50%	0.54%	0.02%
26	7008	7008	16653	7000	7000	16650	0.11%	0.11%	0.02%
27	7035	7035	16653	7000	7000	16650	0.50%	0.50%	0.02%
28	7008	7008	16653	7000	7000	16650	0.11%	0.11%	0.02%
29	7041	7038	16650	7000	7000	16650	0.58%	0.54%	0.00%
30	7005	7005	16650	7000	7000	16650	0.07%	0.07%	0.00%

Table 4.12. Experimental and Calculated Values of TBI, TBO and TBIO for Experiment #3.

**Application Algorithm,
Fault at Node 4, Data packet 10**

Figure 4.12. TBI, TBO and TBIO for Experiment #4.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	351	17001	16650	0	16650	16650	100.00%	2.06%	0.00%
2	7030	7033	16653	7023	7023	16650	0.10%	0.14%	0.02%
3	7013	7010	16650	7023	7023	16650	-0.14%	-0.19%	0.00%
4	7030	7033	16653	7023	7023	16650	0.10%	0.14%	0.02%
5	7013	7013	16653	7023	7023	16650	-0.14%	-0.14%	0.02%
6	7033	7033	16653	7023	7023	16650	0.14%	0.14%	0.02%
7	7013	7010	16650	7023	7023	16650	-0.14%	-0.19%	0.00%
8	7030	7033	16653	7023	7023	16650	0.10%	0.14%	0.02%
9	7013	7010	16650	7023	7023	16650	-0.14%	-0.19%	0.00%
10	7033	12844	22461	7023	12834	22461	0.14%	0.08%	0.00%
11	7010	5386	20837	7023	5385	20823	-0.19%	0.02%	0.07%
12	7033	5383	19187	7023	5385	19185	0.14%	-0.04%	0.01%
13	7028	5383	17542	7023	5385	17547	0.07%	-0.04%	-0.03%
14	7003	6114	16653	7023	6126	16650	-0.29%	-0.20%	0.02%
15	7008	7008	16653	7023	7023	16650	-0.21%	-0.21%	0.02%
16	7035	7035	16653	7023	7023	16650	0.17%	0.17%	0.02%
17	7011	7011	16653	7023	7023	16650	-0.17%	-0.17%	0.02%
18	7035	7032	16650	7023	7023	16650	0.17%	0.13%	0.00%
19	7011	7011	16650	7023	7023	16650	-0.17%	-0.17%	0.00%
20	7032	7035	16653	7023	7023	16650	0.13%	0.17%	0.02%
21	7011	7011	16653	7023	7023	16650	-0.17%	-0.17%	0.02%
22	7032	7032	16653	7023	7023	16650	0.13%	0.13%	0.02%
23	7014	7014	16653	7023	7023	16650	-0.13%	-0.13%	0.02%
24	7032	7029	16650	7023	7023	16650	0.13%	0.09%	0.00%
25	7014	7014	16650	7023	7023	16650	-0.13%	-0.13%	0.00%
26	7029	7032	16653	7023	7023	16650	0.09%	0.13%	0.02%
27	7014	7014	16653	7023	7023	16650	-0.13%	-0.13%	0.02%
28	7029	7029	16653	7023	7023	16650	0.09%	0.09%	0.02%
29	7017	7017	16653	7023	7023	16650	-0.09%	-0.09%	0.02%
30	7029	7026	16650	7023	7023	16650	0.09%	0.04%	0.00%

Table 4.13. Experimental and Calculated Values of TBI,
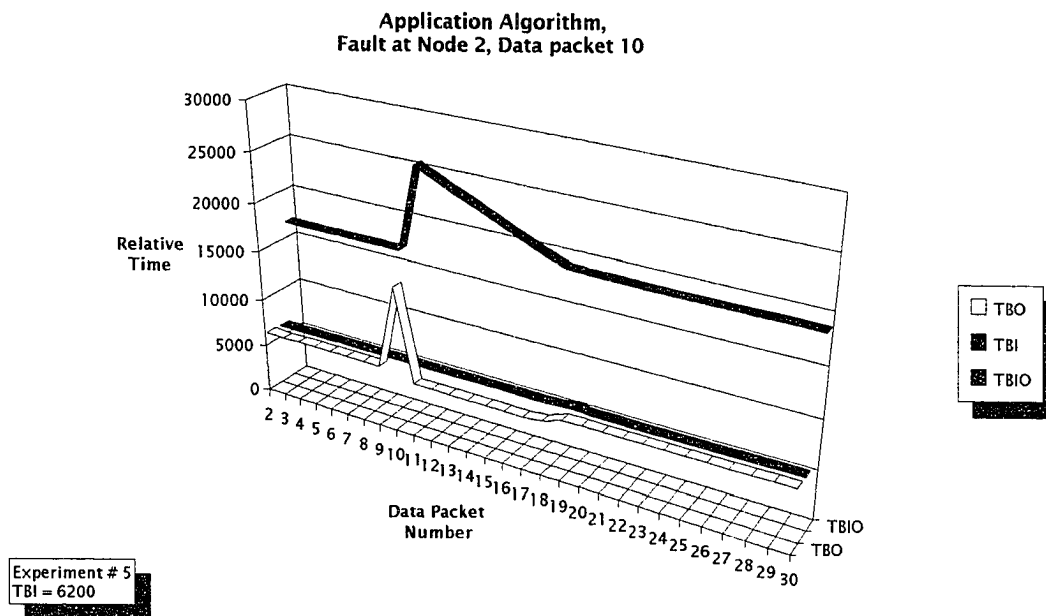TBO and TBIO for Experiment #4.

**Application Algorithm,
Fault at Node 2, Data packet 10**

Figure 4.13. TBI, TBO and TBIO for Experiment #5.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	351	17001	16650	0	16650	16650	100.00%	2.06%	0.00%
2	6238	6284	16696	6236	6236	16650	0.03%	0.76%	0.28%
3	6221	6149	16624	6236	6236	16650	-0.24%	-1.41%	-0.16%
4	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
5	6240	6237	16624	6236	6236	16650	0.06%	0.02%	-0.16%
6	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
7	6243	6240	16624	6236	6236	16650	0.11%	0.06%	-0.16%
8	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
9	6240	6237	16624	6236	6236	16650	0.06%	0.02%	-0.16%
10	6237	14964	25351	6236	14963	25377	0.02%	0.01%	-0.10%
11	6240	5412	24523	6236	5385	24526	0.06%	0.50%	-0.01%
12	6231	5412	23704	6236	5385	23675	-0.08%	0.50%	0.12%
13	6220	5409	22893	6236	5385	22824	-0.26%	0.44%	0.30%
14	6213	5383	22063	6236	5385	21973	-0.37%	-0.04%	0.41%
15	6241	5386	21208	6236	5385	21122	0.08%	0.02%	0.41%
16	6204	5383	20387	6236	5385	20271	-0.52%	-0.04%	0.57%
17	6214	5386	19559	6236	5385	19420	-0.35%	0.02%	0.71%
18	6237	5563	18885	6236	5549	18733	0.02%	0.25%	0.80%
19	6330	6212	18767	6236	6236	18733	1.48%	-0.39%	0.18%
20	6227	6215	18755	6236	6236	18733	-0.14%	-0.34%	0.12%
21	6230	6206	18731	6236	6236	18733	-0.10%	-0.48%	-0.01%
22	6209	6209	18731	6236	6236	18733	-0.43%	-0.43%	-0.01%
23	6215	6215	18731	6236	6236	18733	-0.34%	-0.34%	-0.01%
24	6206	6206	18731	6236	6236	18733	-0.48%	-0.48%	-0.01%
25	6212	6212	18731	6236	6236	18733	-0.39%	-0.39%	-0.01%
26	6212	6215	18734	6236	6236	18733	-0.39%	-0.34%	0.01%
27	6209	6206	18731	6236	6236	18733	-0.43%	-0.48%	-0.01%
28	6209	6209	18731	6236	6236	18733	-0.43%	-0.43%	-0.01%
29	6215	6215	18731	6236	6236	18733	-0.34%	-0.34%	-0.01%
30	6206	6206	18731	6236	6236	18733	-0.48%	-0.48%	-0.01%

Table 4.14. Experimental and Calculated Values of TBI,
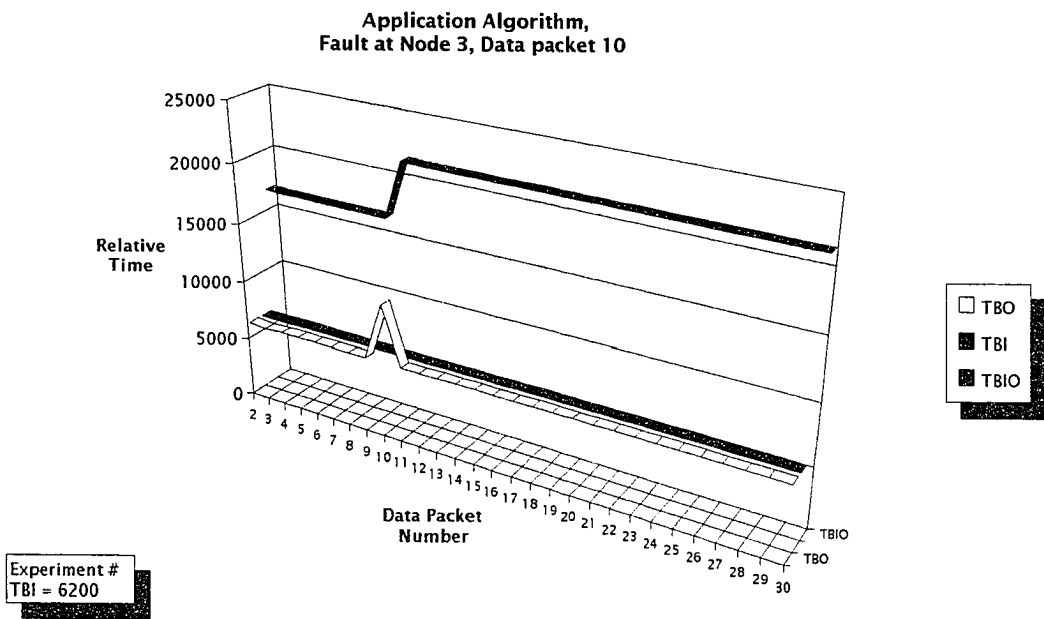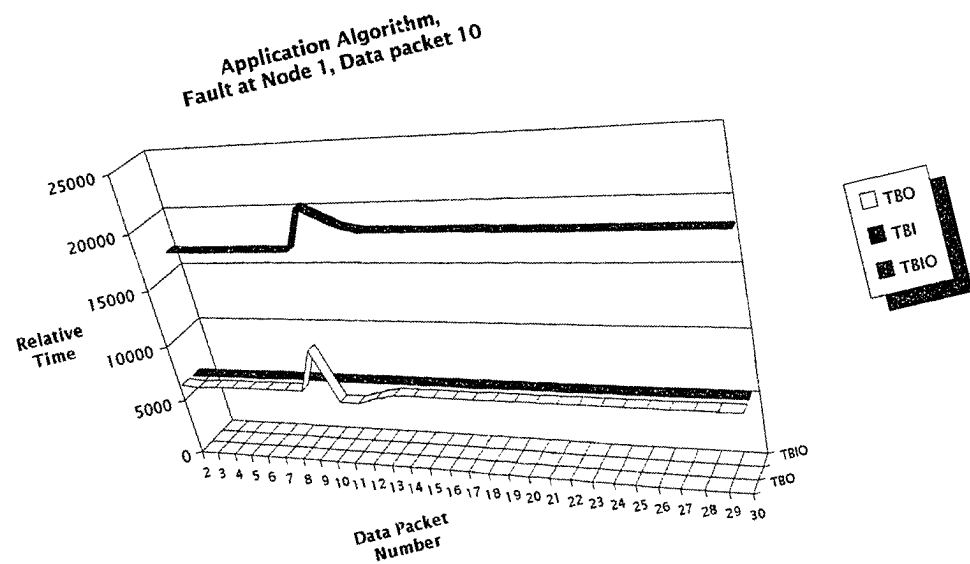TBO and TBIO for Experiment #5.

Figure 4.14. TBI, TBO and TBIO for Experiment #6.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	351	17001	16650	0	16650	16650	100.00%	2.06%	0.00%
2	6238	6284	16696	6236	6236	16650	0.03%	0.76%	0.28%
3	6221	6149	16624	6236	6236	16650	-0.24%	-1.41%	-0.16%
4	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
5	6240	6237	16624	6236	6236	16650	0.06%	0.02%	-0.16%
6	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
7	6243	6240	16624	6236	6236	16650	0.11%	0.06%	-0.16%
8	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
9	6240	6237	16624	6236	6236	16650	0.06%	0.02%	-0.16%
10	6237	11051	21438	6236	11050	21464	0.02%	0.01%	-0.12%
11	6240	6212	21410	6236	6236	21464	0.06%	-0.39%	-0.25%
12	6214	6212	21408	6236	6236	21464	-0.35%	-0.39%	-0.26%
13	6237	6209	21380	6236	6236	21464	0.02%	-0.43%	-0.39%
14	6220	6209	21369	6236	6236	21464	-0.26%	-0.43%	-0.44%
15	6212	6209	21366	6236	6236	21464	-0.39%	-0.43%	-0.46%
16	6209	6212	21369	6236	6236	21464	-0.43%	-0.39%	-0.44%
17	6209	6212	21372	6236	6236	21464	-0.43%	-0.39%	-0.43%
18	6209	6212	21375	6236	6236	21464	-0.43%	-0.39%	-0.42%
19	6212	6209	21372	6236	6236	21464	-0.39%	-0.43%	-0.43%
20	6212	6209	21369	6236	6236	21464	-0.39%	-0.43%	-0.44%
21	6212	6209	21366	6236	6236	21464	-0.39%	-0.43%	-0.46%
22	6209	6212	21369	6236	6236	21464	-0.43%	-0.39%	-0.44%
23	6209	6212	21372	6236	6236	21464	-0.43%	-0.39%	-0.43%
24	6209	6212	21375	6236	6236	21464	-0.43%	-0.39%	-0.42%
25	6212	6209	21372	6236	6236	21464	-0.39%	-0.43%	-0.43%
26	6212	6209	21369	6236	6236	21464	-0.39%	-0.43%	-0.44%
27	6212	6209	21366	6236	6236	21464	-0.39%	-0.43%	-0.46%
28	6209	6212	21369	6236	6236	21464	-0.43%	-0.39%	-0.44%
29	6209	6212	21372	6236	6236	21464	-0.43%	-0.39%	-0.43%
30	6209	6212	21375	6236	6236	21464	-0.43%	-0.39%	-0.42%

Table 4.15. Experimental and Calculated Values of TBI,
TBO and TBIO for Experiment #6.

**Application Algorithm,
Fault at Node 1, Data packet 10**

Figure 4.15. TBI, TBO and TBIO for Experiment #7.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	351	17001	16650	0	16650	16650	100.00%	2.06%	0.00%
2	6238	6284	16696	6236	6236	16650	0.03%	0.76%	0.28%
3	6221	6149	16624	6236	6236	16650	-0.24%	-1.41%	-0.16%
4	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
5	6240	6237	16624	6236	6236	16650	0.06%	0.02%	-0.16%
6	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
7	6243	6240	16624	6236	6236	16650	0.11%	0.06%	-0.16%
8	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
9	6240	6237	16624	6236	6236	16650	0.06%	0.02%	-0.16%
10	6237	9865	20252	6236	9864	20278	0.02%	0.01%	-0.13%
11	6240	5386	19398	6236	5385	19427	0.06%	0.02%	-0.15%
12	6214	5383	18541	6236	5385	18576	-0.35%	-0.04%	-0.19%
13	6237	5838	18157	6236	5742	18082	0.02%	1.64%	0.41%
14	6220	6209	18136	6236	6236	18082	-0.26%	-0.43%	0.30%
15	6212	6212	18105	6236	6236	18082	-0.39%	-0.39%	0.13%
16	6209	6209	18080	6236	6236	18082	-0.43%	-0.43%	-0.01%
17	6209	6212	18080	6236	6236	18082	-0.43%	-0.39%	-0.01%
18	6209	6212	18083	6236	6236	18082	-0.43%	-0.39%	0.01%
19	6212	6212	18086	6236	6236	18082	-0.39%	-0.39%	0.02%
20	6212	6209	18083	6236	6236	18082	-0.39%	-0.43%	0.01%
21	6212	6212	18083	6236	6236	18082	-0.39%	-0.39%	0.01%
22	6209	6209	18080	6236	6236	18082	-0.43%	-0.43%	-0.01%
23	6209	6212	18080	6236	6236	18082	-0.43%	-0.39%	-0.01%
24	6209	6212	18083	6236	6236	18082	-0.43%	-0.39%	0.01%
25	6212	6212	18086	6236	6236	18082	-0.39%	-0.39%	0.02%
26	6212	6209	18083	6236	6236	18082	-0.39%	-0.43%	0.01%
27	6212	6212	18083	6236	6236	18082	-0.39%	-0.39%	0.01%
28	6209	6209	18080	6236	6236	18082	-0.43%	-0.43%	-0.01%
29	6209	6212	18080	6236	6236	18082	-0.43%	-0.39%	-0.01%
30	6209	6212	18083	6236	6236	18082	-0.43%	-0.39%	0.01%

Table 4.16. Experimental and Calculated Values of TBI,
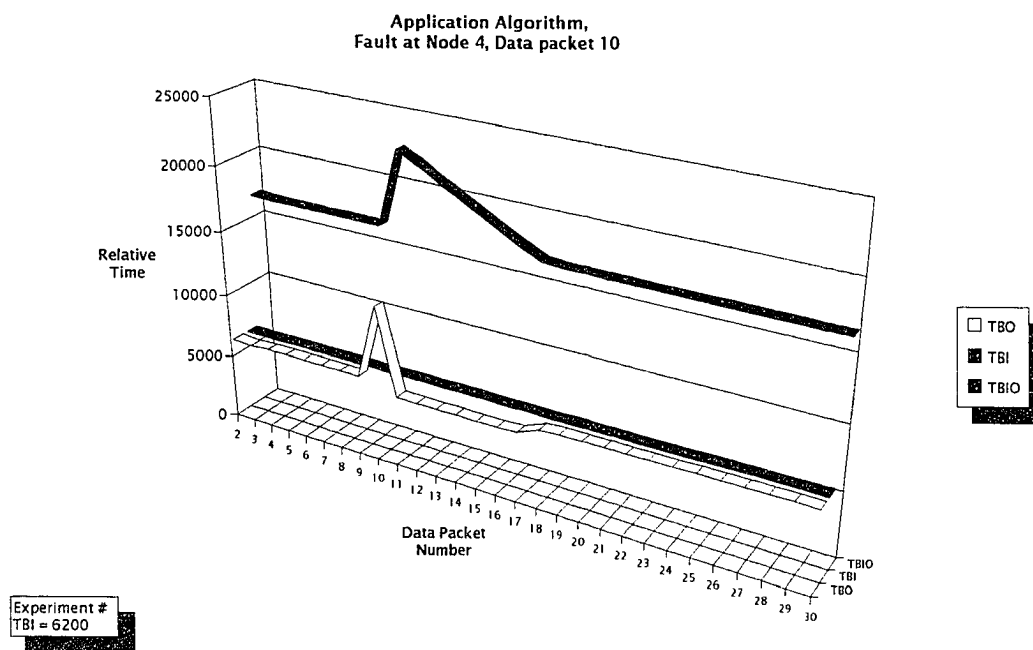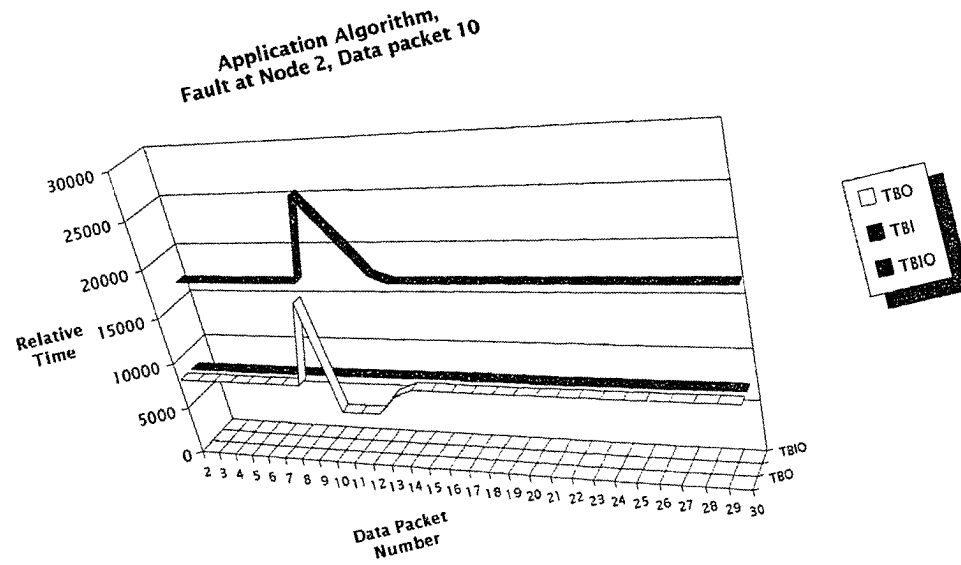TBO and TBIO for Experiment #7.

Application Algorithm,
Fault at Node 4, Data packet 10

Figure 4.16. TBI, TBO and TBIO for Experiment #8.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	351	17001	16650	0	16650	16650	100.00%	2.06%	0.00%
2	6238	6284	16696	6236	6236	16650	0.03%	0.76%	0.28%
3	6221	6149	16624	6236	6236	16650	-0.24%	-1.41%	-0.16%
4	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
5	6240	6237	16624	6236	6236	16650	0.06%	0.02%	-0.16%
6	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
7	6243	6240	16624	6236	6236	16650	0.11%	0.06%	-0.16%
8	6237	6240	16627	6236	6236	16650	0.02%	0.06%	-0.14%
9	6240	6237	16624	6236	6236	16650	0.06%	0.02%	-0.16%
10	6237	12083	22470	6236	12082	22496	0.02%	0.01%	-0.12%
11	6240	5438	21668	6236	5385	21645	0.06%	0.97%	0.11%
12	6237	5383	20814	6236	5385	20794	0.02%	-0.04%	0.10%
13	6217	5386	19983	6236	5385	19943	-0.31%	0.02%	0.20%
14	6237	5383	19129	6236	5385	19092	0.02%	-0.04%	0.19%
15	6213	5383	18299	6236	5385	18241	-0.37%	-0.04%	0.32%
16	6214	5386	17471	6236	5385	17390	-0.35%	0.02%	0.46%
17	6240	5487	16718	6236	5496	16650	0.06%	-0.16%	0.41%
18	6204	6113	16627	6236	6236	16650	-0.52%	-2.01%	-0.14%
19	6232	6229	16624	6236	6236	16650	-0.06%	-0.11%	-0.16%
20	6204	6204	16624	6236	6236	16650	-0.52%	-0.52%	-0.16%
21	6232	6232	16624	6236	6236	16650	-0.06%	-0.06%	-0.16%
22	6204	6207	16627	6236	6236	16650	-0.52%	-0.47%	-0.14%
23	6229	6229	16627	6236	6236	16650	-0.11%	-0.11%	-0.14%
24	6204	6204	16627	6236	6236	16650	-0.52%	-0.52%	-0.14%
25	6229	6226	16624	6236	6236	16650	-0.11%	-0.16%	-0.16%
26	6207	6207	16624	6236	6236	16650	-0.47%	-0.47%	-0.16%
27	6229	6229	16624	6236	6236	16650	-0.11%	-0.11%	-0.16%
28	6204	6207	16627	6236	6236	16650	-0.52%	-0.47%	-0.14%
29	6229	6229	16627	6236	6236	16650	-0.11%	-0.11%	-0.14%
30	6204	6204	16627	6236	6236	16650	-0.52%	-0.52%	-0.14%

Table 4.17. Experimental and Calculated Values of TBI,
TBO and TBIO for Experiment #8.

**Application Algorithm,
Fault at Node 2, Data packet 10**

Figure 4.17. TBI, TBO and TBIO for Experiment #9.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	351	17013	16662	0	16650	16650	100.00%	2.13%	0.07%
2	8042	8037	16657	8036	8036	16650	0.07%	0.01%	0.04%
3	8037	8030	16650	8036	8036	16650	0.01%	-0.07%	0.00%
4	8034	8037	16653	8036	8036	16650	-0.02%	0.01%	0.02%
5	8033	8034	16654	8036	8036	16650	-0.04%	-0.02%	0.02%
6	8037	8037	16654	8036	8036	16650	0.01%	0.01%	0.02%
7	8034	8034	16654	8036	8036	16650	-0.02%	-0.02%	0.02%
8	8037	8037	16654	8036	8036	16650	0.01%	0.01%	0.02%
9	8034	8037	16657	8036	8036	16650	-0.02%	0.01%	0.04%
10	8037	16696	25316	8036	16695	25309	0.01%	0.01%	0.03%
11	8034	5383	22665	8036	5385	22658	-0.02%	-0.04%	0.03%
12	8003	5386	20048	8036	5385	20007	-0.41%	0.02%	0.20%
13	8033	5420	17435	8036	5385	17356	-0.04%	0.65%	0.45%
14	8041	7259	16653	8036	7330	16650	0.06%	-0.98%	0.02%
15	8026	8031	16658	8036	8036	16650	-0.12%	-0.06%	0.05%
16	8044	8037	16651	8036	8036	16650	0.10%	0.01%	0.01%
17	8034	8037	16654	8036	8036	16650	-0.02%	0.01%	0.02%
18	8034	8037	16657	8036	8036	16650	-0.02%	0.01%	0.04%
19	8037	8037	16657	8036	8036	16650	0.01%	0.01%	0.04%
20	8037	8031	16651	8036	8036	16650	0.01%	-0.06%	0.01%
21	8037	8037	16651	8036	8036	16650	0.01%	0.01%	0.01%
22	8034	8037	16654	8036	8036	16650	-0.02%	0.01%	0.02%
23	8034	8037	16657	8036	8036	16650	-0.02%	0.01%	0.04%
24	8037	8037	16657	8036	8036	16650	0.01%	0.01%	0.04%
25	8037	8031	16651	8036	8036	16650	0.01%	-0.06%	0.01%
26	8037	8037	16651	8036	8036	16650	0.01%	0.01%	0.01%
27	8034	8037	16654	8036	8036	16650	-0.02%	0.01%	0.02%
28	8034	8037	16657	8036	8036	16650	-0.02%	0.01%	0.04%
29	8037	8037	16657	8036	8036	16650	0.01%	0.01%	0.04%
30	8037	8031	16651	8036	8036	16650	0.01%	-0.06%	0.01%

Table 4.18. Experimental and Calculated Values of TBI,
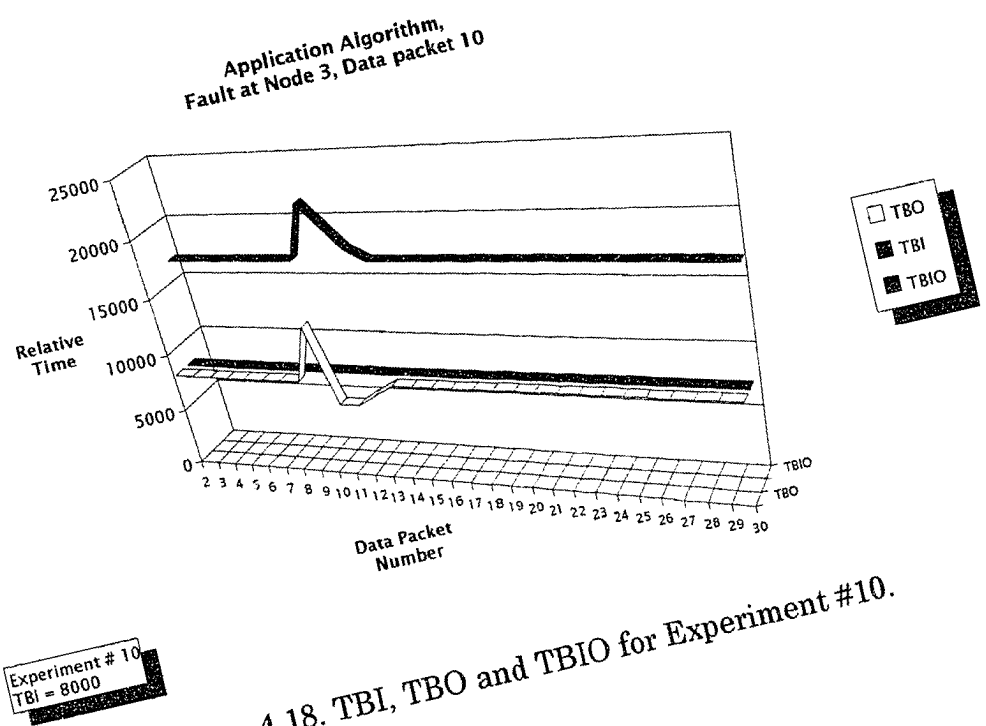TBO and TBIO for Experiment #9.

Figure 4.18. TBI, TBO and TBIO for Experiment #10.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	351	17013	16662	0	16650	16650	100.00%	2.13%	0.07%
2	8042	8037	16657	8036	8036	16650	0.07%	0.01%	0.04%
3	8037	8030	16650	8036	8036	16650	0.01%	-0.07%	0.00%
4	8034	8037	16653	8036	8036	16650	-0.02%	0.01%	0.02%
5	8033	8034	16654	8036	8036	16650	-0.04%	-0.02%	0.02%
6	8037	8037	16654	8036	8036	16650	0.01%	0.01%	0.02%
7	8034	8034	16654	8036	8036	16650	-0.02%	-0.02%	0.02%
8	8037	8037	16654	8036	8036	16650	0.01%	0.01%	0.02%
9	8034	8037	16657	8036	8036	16650	-0.02%	0.01%	0.04%
10	8037	12692	21312	8036	12691	21305	0.01%	0.01%	0.03%
11	8034	6183	19461	8036	6185	19454	-0.02%	-0.03%	0.04%
12	8072	6186	17575	8036	6185	17603	0.45%	0.02%	-0.16%
13	8034	7113	16654	8036	7083	16650	-0.02%	0.42%	0.02%
14	8034	8037	16657	8036	8036	16650	-0.02%	0.01%	0.04%
15	8037	8037	16657	8036	8036	16650	0.01%	0.01%	0.04%
16	8034	8034	16657	8036	8036	16650	-0.02%	-0.02%	0.04%
17	8037	8034	16654	8036	8036	16650	0.01%	-0.02%	0.02%
18	8037	8037	16654	8036	8036	16650	0.01%	0.01%	0.02%
19	8034	8037	16657	8036	8036	16650	-0.02%	0.01%	0.04%
20	8037	8037	16657	8036	8036	16650	0.01%	0.01%	0.04%
21	8034	8034	16657	8036	8036	16650	-0.02%	-0.02%	0.04%
22	8037	8034	16654	8036	8036	16650	0.01%	-0.02%	0.02%
23	8037	8037	16654	8036	8036	16650	0.01%	0.01%	0.02%
24	8034	8037	16657	8036	8036	16650	-0.02%	0.01%	0.04%
25	8037	8037	16657	8036	8036	16650	0.01%	0.01%	0.04%
26	8034	8034	16657	8036	8036	16650	-0.02%	-0.02%	0.04%
27	8037	8034	16654	8036	8036	16650	0.01%	-0.02%	0.02%
28	8037	8037	16654	8036	8036	16650	0.01%	0.01%	0.02%
29	8034	8037	16657	8036	8036	16650	-0.02%	0.01%	0.04%
30	8037	8037	16657	8036	8036	16650	0.01%	0.01%	0.04%

Table 4.19. Experimental and Calculated Values of
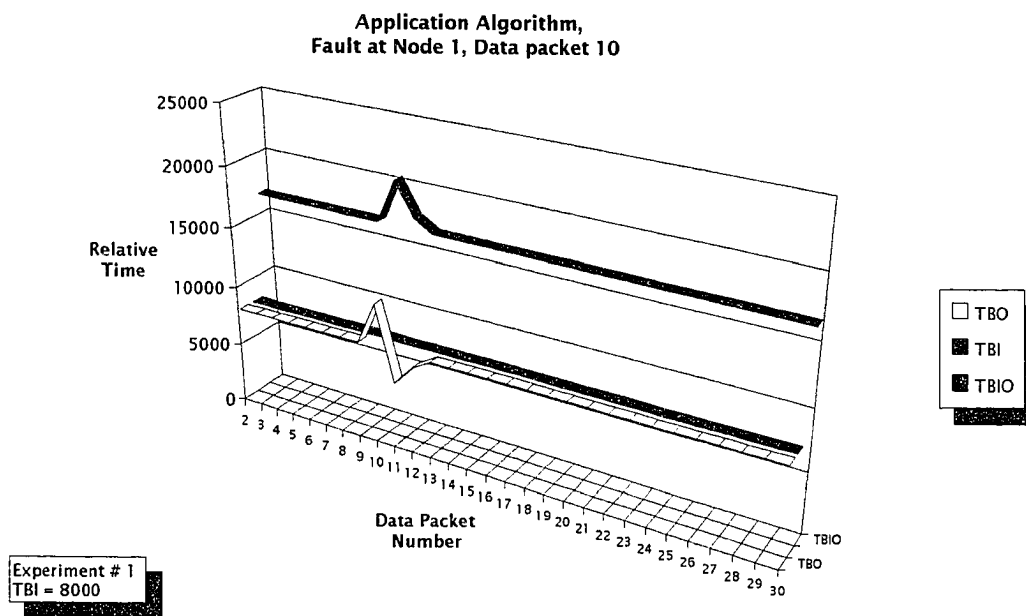TBI, TBO and TBIO for Experiment #10.

Application Algorithm,
Fault at Node 1, Data packet 10

Figure 4.19. TBI, TBO and TBIO for Experiment #11.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	351	17013	16662	0	16650	16650	100.00%	2.13%	0.07%
2	8042	8037	16657	8036	8036	16650	0.07%	0.01%	0.04%
3	8037	8030	16650	8036	8036	16650	0.01%	-0.07%	0.00%
4	8034	8037	16653	8036	8036	16650	-0.02%	0.01%	0.02%
5	8033	8034	16654	8036	8036	16650	-0.04%	-0.02%	0.02%
6	8037	8037	16654	8036	8036	16650	0.01%	0.01%	0.02%
7	8034	8034	16654	8036	8036	16650	-0.02%	-0.02%	0.02%
8	8037	8037	16654	8036	8036	16650	0.01%	0.01%	0.02%
9	8034	8033	16653	8036	8036	16650	-0.02%	-0.04%	0.02%
10	8037	11606	20222	8036	11605	20219	0.01%	0.01%	0.01%
11	8003	5383	17602	8036	5385	17568	-0.41%	-0.04%	0.19%
12	8028	7079	16653	8036	7118	16650	-0.10%	-0.55%	0.02%
13	8023	8023	16653	8036	8036	16650	-0.16%	-0.16%	0.02%
14	8003	8003	16653	8036	8036	16650	-0.41%	-0.41%	0.02%
15	8023	8023	16653	8036	8036	16650	-0.16%	-0.16%	0.02%
16	8003	8003	16653	8036	8036	16650	-0.41%	-0.41%	0.02%
17	8020	8020	16653	8036	8036	16650	-0.20%	-0.20%	0.02%
18	8003	8003	16653	8036	8036	16650	-0.41%	-0.41%	0.02%
19	8023	8023	16653	8036	8036	16650	-0.16%	-0.16%	0.02%
20	8003	8003	16653	8036	8036	16650	-0.41%	-0.41%	0.02%
21	8020	8023	16656	8036	8036	16650	-0.20%	-0.16%	0.04%
22	8003	8000	16653	8036	8036	16650	-0.41%	-0.45%	0.02%
23	8023	8023	16653	8036	8036	16650	-0.16%	-0.16%	0.02%
24	8003	8003	16653	8036	8036	16650	-0.41%	-0.41%	0.02%
25	8023	8023	16653	8036	8036	16650	-0.16%	-0.16%	0.02%
26	8003	8003	16653	8036	8036	16650	-0.41%	-0.41%	0.02%
27	8020	8020	16653	8036	8036	16650	-0.20%	-0.20%	0.02%
28	8003	8003	16653	8036	8036	16650	-0.41%	-0.41%	0.02%
29	8023	8023	16653	8036	8036	16650	-0.16%	-0.16%	0.02%
30	8003	8003	16653	8036	8036	16650	-0.41%	-0.41%	0.02%

Table 4.20. Experimental and Calculated Values of
TBI, TBO and TBIO for Experiment #11.

Application Algorithm,
Fault at Node 4, Data packet 10
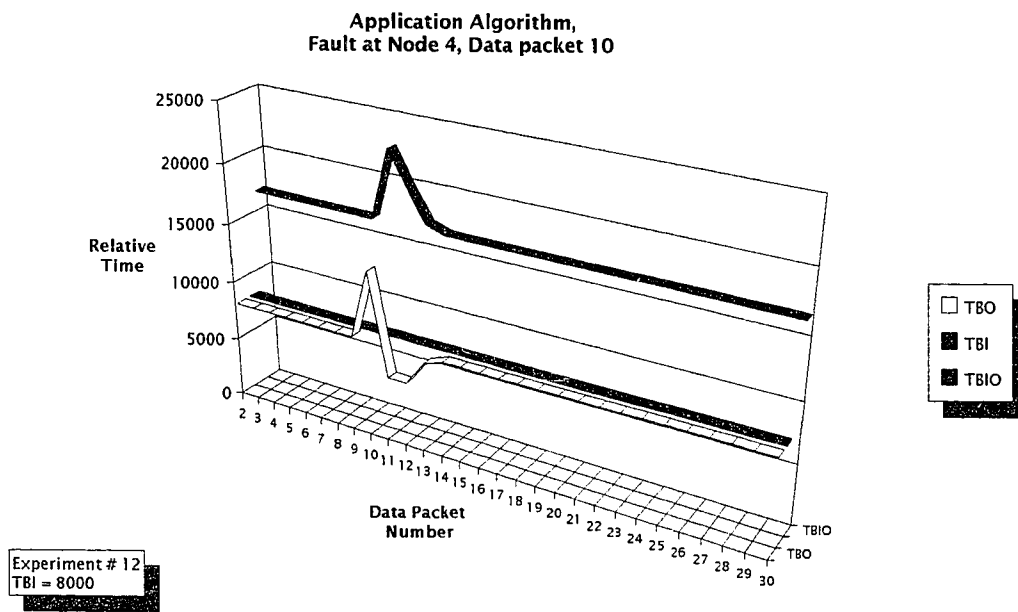
Figure 4.20. TBI, TBO and TBIO for Experiment #12.

Packet	Experimental			Calculated			Difference in %		
	TBI	TBO	TBIO	TBI	TBO	TBIO	TBI	TBO	TBIO
1	351	17013	16662	0	16650	16650	100.00%	2.13%	0.07%
2	8042	8037	16657	8036	8036	16650	0.07%	0.01%	0.04%
3	8037	8030	16650	8036	8036	16650	0.01%	-0.07%	0.00%
4	8034	8037	16653	8036	8036	16650	-0.02%	0.01%	0.02%
5	8033	8034	16654	8036	8036	16650	-0.04%	-0.02%	0.02%
6	8037	8037	16654	8036	8036	16650	0.01%	0.01%	0.02%
7	8034	8034	16654	8036	8036	16650	-0.02%	-0.02%	0.02%
8	8037	8037	16654	8036	8036	16650	0.01%	0.01%	0.02%
9	8034	8037	16657	8036	8036	16650	-0.02%	0.01%	0.04%
10	8037	13868	22488	8036	13867	22481	0.01%	0.01%	0.03%
11	8034	5386	19840	8036	5385	19830	-0.02%	0.02%	0.05%
12	8033	5383	17190	8036	5385	17179	-0.04%	-0.04%	0.06%
13	8009	7472	16653	8036	7507	16650	-0.34%	-0.47%	0.02%
14	8026	8034	16661	8036	8036	16650	-0.12%	-0.02%	0.07%
15	8044	8034	16651	8036	8036	16650	0.10%	-0.02%	0.01%
16	8034	8040	16657	8036	8036	16650	-0.02%	0.05%	0.04%
17	8034	8030	16653	8036	8036	16650	-0.02%	-0.07%	0.02%
18	8037	8037	16653	8036	8036	16650	0.01%	0.01%	0.02%
19	8033	8034	16654	8036	8036	16650	-0.04%	-0.02%	0.02%
20	8037	8034	16651	8036	8036	16650	0.01%	-0.02%	0.01%
21	8034	8040	16657	8036	8036	16650	-0.02%	0.05%	0.04%
22	8034	8030	16653	8036	8036	16650	-0.02%	-0.07%	0.02%
23	8037	8037	16653	8036	8036	16650	0.01%	0.01%	0.02%
24	8033	8034	16654	8036	8036	16650	-0.04%	-0.02%	0.02%
25	8037	8034	16651	8036	8036	16650	0.01%	-0.02%	0.01%
26	8034	8040	16657	8036	8036	16650	-0.02%	0.05%	0.04%
27	8034	8030	16653	8036	8036	16650	-0.02%	-0.07%	0.02%
28	8037	8037	16653	8036	8036	16650	0.01%	0.01%	0.02%
29	8033	8034	16654	8036	8036	16650	-0.04%	-0.02%	0.02%
30	8037	8034	16651	8036	8036	16650	0.01%	-0.02%	0.01%

Table 4.21. Experimental and Calculated Values of TBI,
TBO and TBIO for Experiment #12.

**Operating Point Plane
Application Algorith**
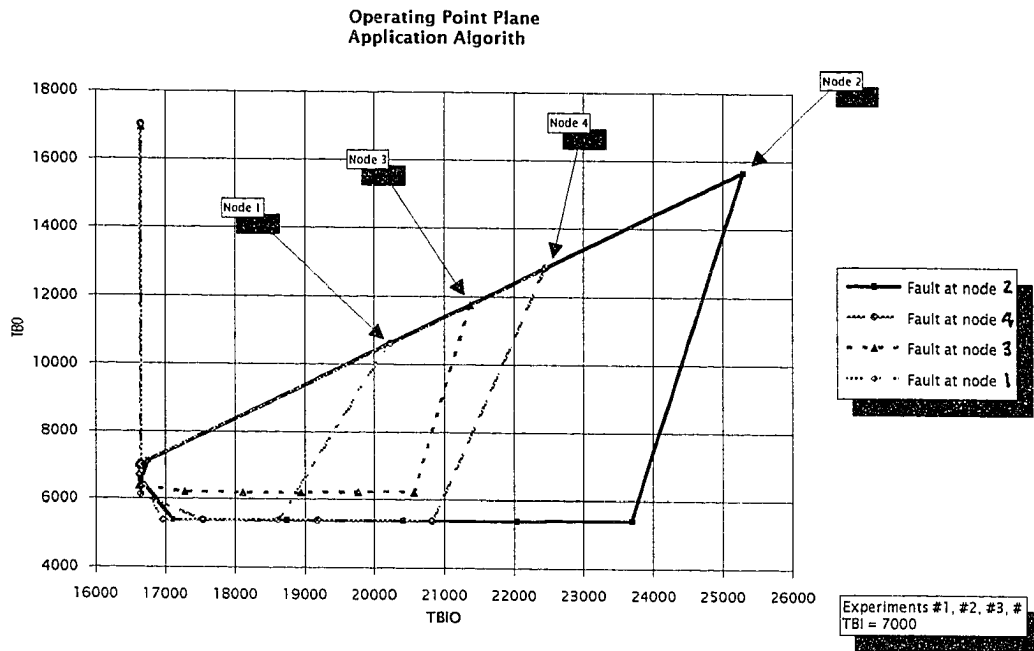
Figure 4.21. Operating Point Plane, TBI = 7000.

**Operating Point Plane
Application Algorith**
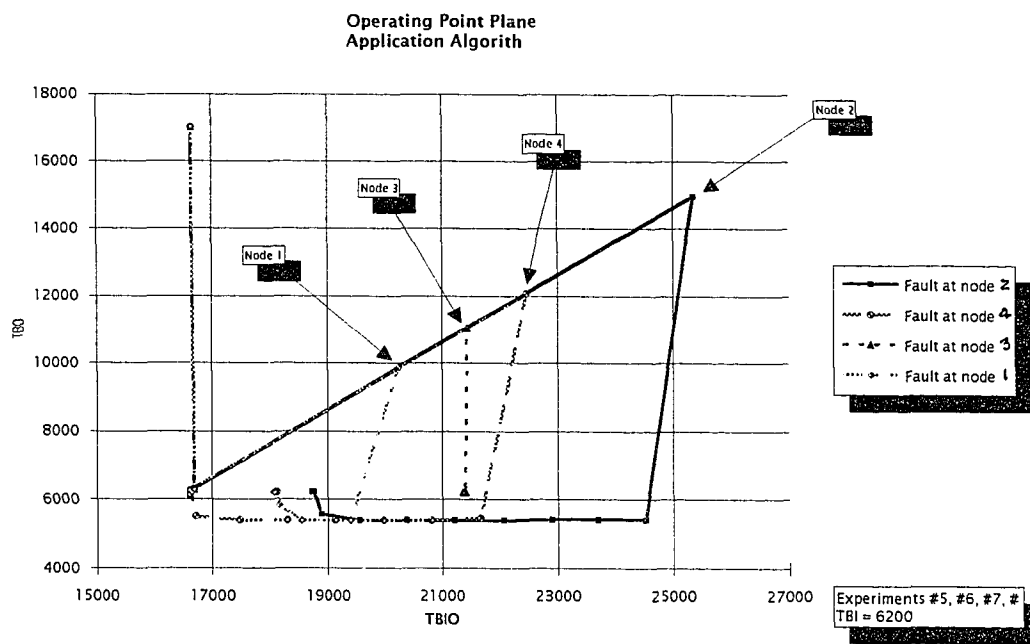
Figure 4.22. Operating Point Plane, TBI = 6200.

Operating Point Plane
Application Algorith
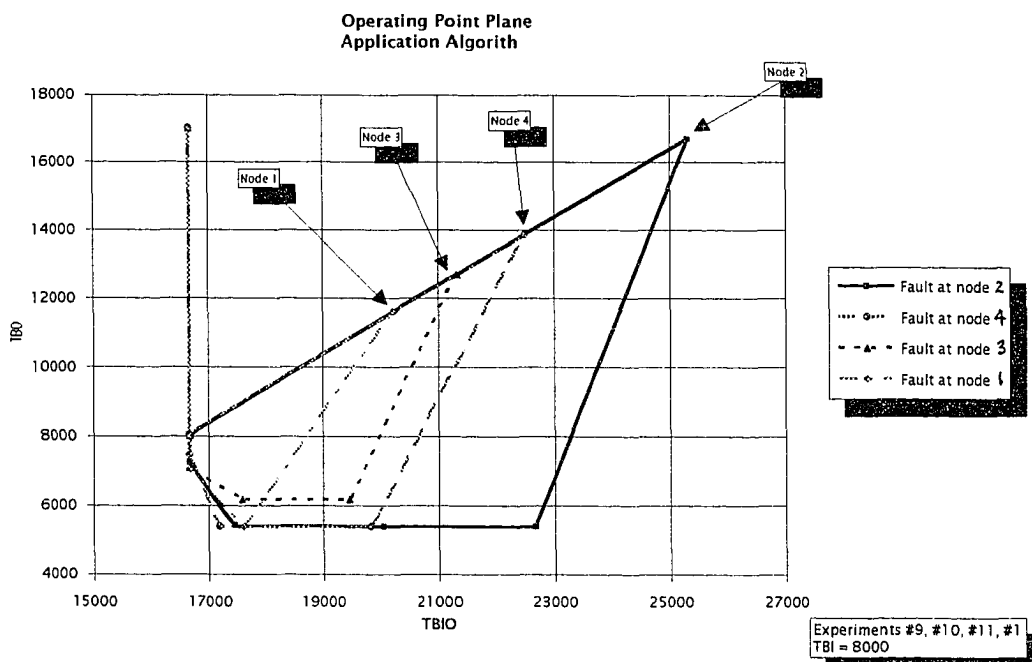
Figure 4.23. Operating Point Plane, TBI = 8000.

**Operating Point Plane
Application Algorithm
Fault at node 2, data packet 10**
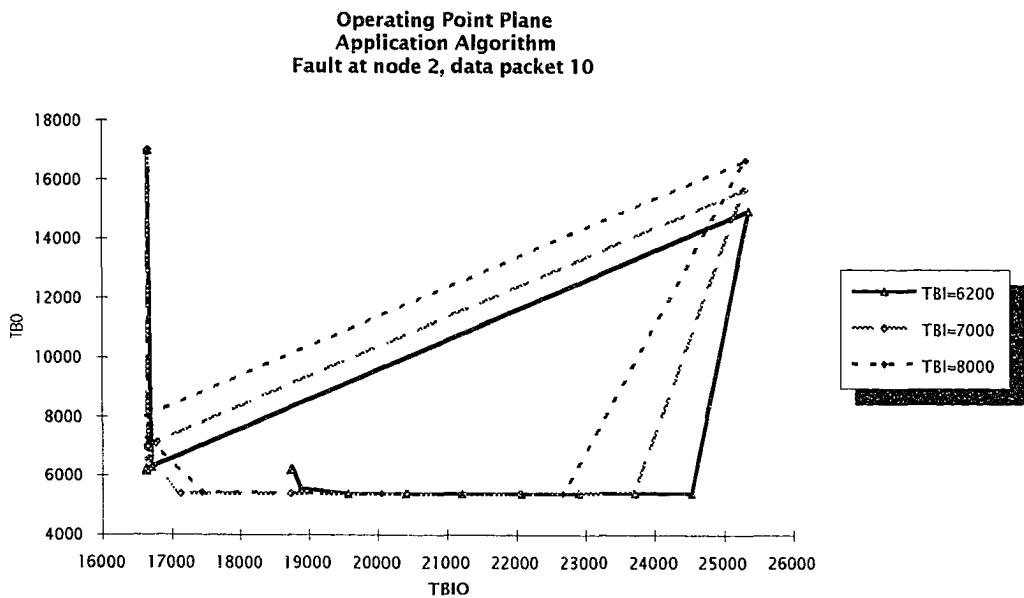
Figure 4.24. Operating Point Plane, Fault at node 2.

**Operating Point Plane
Application Algorithm
Fault at node 3, data packet 10**
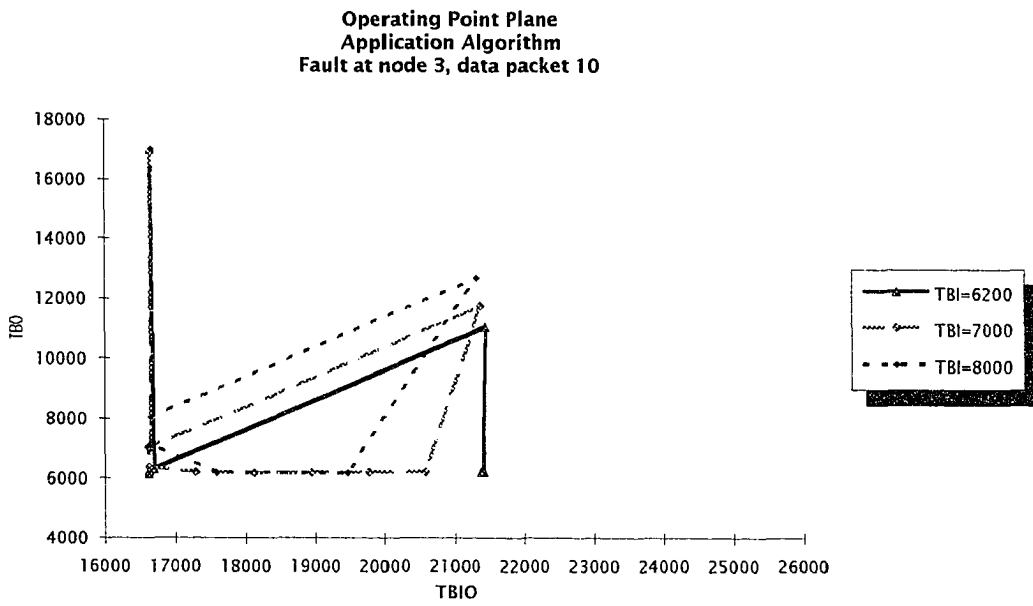
Figure 4.25. Operating Point Plane, Fault at node 3.

Operating Point Plane
Application Algorithm
Fault at node 1, data packet 10
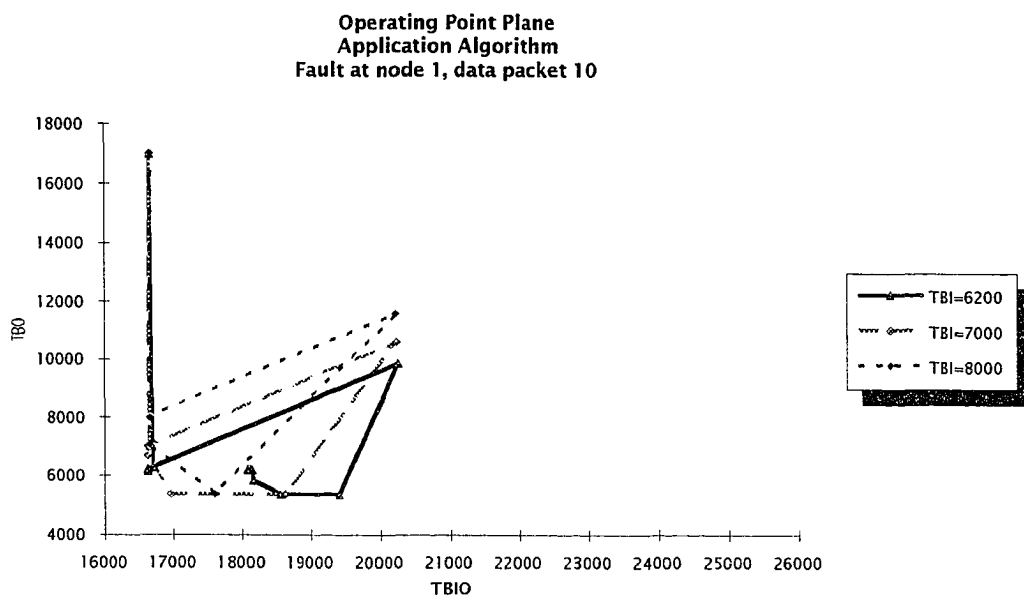
Figure 4.26. Operating Point Plane, Fault at node 1.

**Operating Point Plane
Application Algorithm
Fault at node 4, data packet 10**
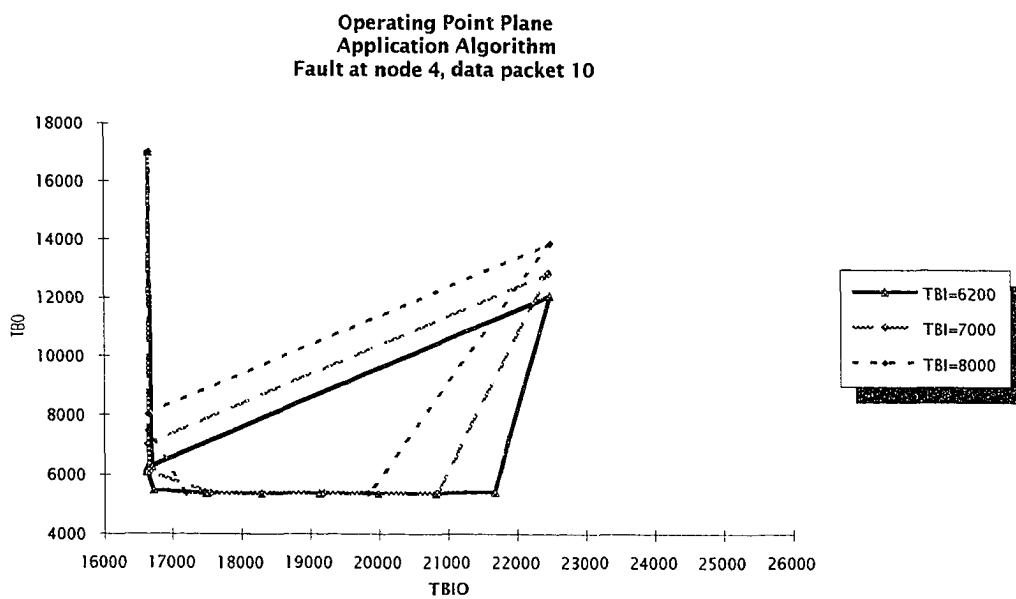
Figure 4.27. Operating Point Plane, Fault at node 4.

4.5 Chapter Summary

The following objectives have been accomplished in this chapter. The development of the simulation program has been explained to demonstrate its validity as an AMOS simulation. Furthermore, the validation of the simulation by comparison with hardware experiments has been attained in two phases. The first phase was to validate the simulation under normal conditions, i.e., no fault introduced in the system. This is called the steady state validation of the simulation. The second phase was to validate the simulation under the effects of a fault in the system. This is called the fault transient validation of the simulation. Lastly, having the simulation validated as a sound means to test the theory developed in Chapter Two, twelve experiments were carried out. Along with these experiments, all calculations and paths were found to retrieve the token lifetimes in the paths of the graph. These calculations were compared against the experimental data and were found to be, for all practical purposes, accurate within 1%. Considering that the execution time of the nodes and sources are not exactly the same for every data packet, the calculated data should be valid for the study of these systems. The data show that the model can be used as a tractable and a valid method to investigate the behavior of these systems under any transient conditions due to delays in the system and to delays due to faults in particular.

The dissertation objectives to provide a model that should furnish the time to recover from a fault and whether it recovers at all have been reached. Also the answer to whether a permanent delay has been introduced into the system can be answered with the model presented in this dissertation. It has been shown that the model is adequate for the analysis of the transient behavior of a fault-tolerant multicomputer system under recovery and restoration. An example of how to use the analytical model developed in Chapter Two to generate the data shown in this chapter is presented in Appendix A. The process is rather involved and detailed to be shown in this chapter.

CHAPTER FIVE

CONCLUSIONS

Since the inception of computers, there has been a drive to make computer systems faster. At the verge of physical limitation, although not for the only reason, there has been an interest in speeding up computation by dividing the work among more than one computer. Instead of loading a single computer with all the tasks that have to be executed, multiple computers are set out to work on separate portions of the work.

With the advent of fast systems, real-time systems have come of age. Computers have come to play an important role in situations where operators alone could not handle. Real-time systems have benefited with the coming of multicomputer systems. The added performance from the multicomputer systems is well used in real-time systems due to the demand on faster and more intelligent systems.

One added advantage of multicomputer systems is that they are suitable to be implemented as fault-tolerant systems. The redundancy in the number of processing units is an asset that can be used to increase the system's reliability. This leads to the idea of graceful degradation, i.e., if a computing resource fails, the system does not shut down completely.

144

By combining all three aspects, multicomputer systems, real-time systems and fault-tolerant systems, the resulting system is greater than the sum of all its parts. The complexity of a system in this category may only be compared to the complexity of the tasks that it is required to handle. This area requires an extensive study to provide analysis methods and tools to understand the behavior of such systems. One set of such tools has been provided by Old Dominion University in the body of knowledge of ATAMM. This model has been used to study the behavior of real-time multicomputer systems in steady state operation. It has been successful in uncovering performance bounds and operation strategies. It has also provided a route to follow in case of failure of computing resources. It provides a set of operating points suitable to be used for different numbers of computing resources available in the system at run time. Although the model has been available for some years, it has not been used to study the transient behavior of a system in the event of a fault.

When a fault occurs in a system, there is a transient in the system's performance. The error detection and error recovery phases require time to repair the system and to bring it back to a steady state. There are certain issues that should be addressed. The behavior of the system while it is recovering from a fault is of interest to system designers. The time the system takes to return to a steady state is important in the design of real-time systems. The question of whether the system ever reaches the target

state after a fault occurs is of importance in evaluating the system's reliability. These questions have been answered in this dissertation as it is explained in the following paragraphs.

The objective of this dissertation is to develop a method to estimate the time a system takes to recover and to restore the system to an operational state after a fault has occurred. It is also of interest to find whether the target operational state is reached at all. This objective has been attained successfully and the process is outlined next.

ATAMM for steady state was used as a starting point. The AMG (Algorithm Marked Graph), which expresses the data flow within one data packet, was unfolded to uncover the data flow dependency across data packets. This unfolding provides a view of the data flowing not only from source to sink but also across subsequent data packets. This unfolded AMG is called XAMG (eXtended Algorithm Marked Graph). Also the CMG (Computational Marked Graph), which expresses the data and control flow within one data packet, was unfolded in the same manner to obtain a view of the data and control flow across data packets. This unfolded CMG is called XCMG (eXtended Computational Marked Graph).

Based on these unfolded graphs, the times when the nodes fire and deposit were defined. These times were used to develop the concept of token lifetime. The token lifetime of an edge is the time a token spends in the edge. It is the time from when the token is deposited by the initial

node connected to the edge to when is encumbered by the terminal node connected to the edge. This token lifetime is of importance since it is time that data or control tokens spend without being processed by the successor node and it was used to estimate the amount of delay that could be absorbed in the edge. The notion of delay was introduced to express the effect caused by a fault at a given node. The delay introduced in the process time period effectively increases the time a node takes to process the data at the input. This delay is introduced due to the fact that a node that has failed has to be restarted after the error is detected. This delay is transformed in a delay to deposit the node's output data.

Once delay has been introduced into the graph, it is important to study how it propagates to other nodes and it effectively delays the time to deposit of other nodes. A fire-equivalent node model was developed to express a node's delay to deposit in terms of introduced delay and token lifetime. The notion of lifetime equivalent paths was introduced to characterize one path between nodes by their token lifetime. This was further developed to find dominant lifetime equivalent paths when there are more than one path connecting two nodes. The construction of paths between two nodes was used to show that the paths connecting any two nodes can be reduced to a dominant lifetime equivalent path.

The benefit of this model is that it may be used to find the effect of the delay to deposit of a node r on the deposit time of another node p. If there

exists one or more directed paths from node r to node p, the dominant life-time equivalent path can be found and the delay to deposit of node p can be expressed in terms of the path's token lifetime and the delay to deposit of node r. Since this can be performed between any two nodes, the propagation of delay from the faulted node to any node in the graph may be determined. The delay to deliver system outputs at the expected time may be calculated by finding the dominant paths between the faulted node and the sinks in the XAMG or XCMG. This model may used to predict system performance during recovery by calculating the delays to deposit of the nodes in the graph.

The drawback of the model is that it is computationally intensive. In order to estimate the delay that propagates to a node, all paths need to be searched between the node that generates the delay and the node of interest. After this is performed, the token lifetimes are to be calculated and a dominant lifetime equivalent path is to be found. This is no small task even in the case of a simple graph since an exhaustive search has to be performed every time.

The purpose of the model was to evaluate the behavior of ATAMM based systems undergoing a fault. The target system that was developed was the Generic VHSIC Spaceborne Computer (GVSC). The system required the design of an operating system to follow the ATAMM design guidelines and to be fault-tolerant. This operating system was written for

an IBM GVSC host system developed for NASA Langley Research Center. ATAMM requirements allowed the system to be designed in a modular fashion. The main entities of interest were the computing resources and the nodes to be executed in the graph. The implementation requires the communication between computing resources, which are hardware entities, and nodes in the graph, which are software entities. The idea of data encapsulation in object-oriented programming was used to isolate the communication between these two types of entities. A message passing scheme was used to relay a uniform type of message between nodes and computing resources. This allowed the system's structure to be highly modular and provided a better setting for testing and debugging as well as for software updates and modifications. The hardware dependency was restricted to low level software modules so that the system may be easily portable to other platforms. This makes the system hardware architecture independent at the highest level of operation, i.e., at the message handling level.

A simulation was developed from the same code used for the target GVSC system. This simulation was written to run under the Microsoft Windows environment on the IBM PC and compatibles. It was designed by surrounding the code from the GVSC system with objects as in object-oriented programming. The data structure that was used in the hardware system was also used in the simulation so that both systems may be

initialized by the same data file. This also allows updates to the hardware to be moved to the simulation more easily.

The simulation was the ideal tool to test the new model since the hardware was not available at the time of experimentation. Furthermore, the simulation is easier to use since it only requires an MS DOS system running Microsoft Windows. Therefore, the simulation needed to be validated to be used as an authoritative tool to test the model. This validation was performed in two phases. First it was validated to simulate steady state behavior of the hardware system. Second it was validated to simulate transient behavior of the hardware system.

Two graphs were used to validate the steady state behavior. Using these graphs, two types of comparison were performed between the hardware and the simulation, namely micro and macro comparison. The micro comparison involved the comparison of the sequence of internal events that both systems go through while executing a graph. This comparison was performed directly at the fdt files generated by the systems. An fdt file is a log of all the firing of the internal transitions in the graph. The macro comparison implied the calculation of the the values of TBI, TBO and TBIO for individual data packets for both systems. Values from both, the hardware and the simulation, were compared data packet by data packet. The micro comparison for the first graph indicated that 99% of the events in the fdt from the simulation were in the ±2 positions range with respect to the

fdt from the hardware. The macro comparison for the first graph showed that the maximum difference between the values of TBI, TBO and TBIO from both systems was of 1.38%. The micro comparison for the second graph indicated that 99% of the events in the fdt from the simulation were in the ±2 positions range with respect to the fdt from the hardware. The macro comparison for the second graph indicated that the maximum difference between the values of TBI, TBO and TBIO from both systems was of 1.89%. These results denoted the deterministic nature of a system designed along the ATAMM guidelines. This also validated the simulation for steady state of the hardware.

One graph was used to validate the transient behavior. The system was subject to three faults while executing this graph and degraded from four computing resources to one computing resources. There were 35 data packets executed during the test. The simulation was set up to execute the same graph for the same number of inputs. It was also subject to faults on the same nodes at the same data packets as the hardware system was. The actual processor assignment was disregarded in the micro comparison since after the first fault the actual processor assignment was different in both systems. The micro comparison yielded a 96% of the events in the ±3 positions range. The macro comparison presented a maximum difference of 3.47% in the values of TBI, TBO and TBIO. These comparisons validated

the simulation in the transient behavior of the simulation with respect to the hardware system.

With the simulation validated, it was possible to run twelve experiments to validate the model presented in Chapter Two. This model was developed to describe the transient behavior of a multicomputer system. There was only one graph used for all the experiments. Each experiment was run for 30 data packets. All faults were injected at the data packet ten. Three different TBI were used: 7000, 6200 and 8000. For each one of the experiments in the groups of common TBI a different node was set to fail. The results were gathered in tables showing the values for TBI, TBO and TBIO for each of the data packets. The paths between the faulted node and the sinks were identified, their token lifetimes were computed and the dominant lifetime paths were identified. The delays to the sinks were computed for each one of the data packets in each of the experiments. With these delay values, the TBO and TBIO for each data packet were calculated and gathered along with the simulated experimental values. The calculated values of TBO and TBIO were mostly in the 0.5% difference with respect to the experimental values. The number of individual comparisons is near 700. These results show that the model is extremely accurate in predicting the transient behavior of a multicomputer system designed along the ATAMM guidelines. This accuracy shows the high

performance predictability of the ATAMM model which now extends to the transient behavior of the system.

One conclusion that can be drawn from the collected data is that if the system is driven at a TBI above TBO_{LB}, the system recovers and reaches the target operating point. In three out of four experiments where the system was driven at TBI equal to TBO_{LB}, the system did not reach the target operating point and a permanent delay was added to TBIO in subsequent data packets. It can be seen that if the system is driven at TBO_{LB}, the system does not have enough token lifetime to absorb the delay introduced by the fault.

Another conclusion is that depending on which node the fault is injected, the value of recovery TBO is different. This is a value that depends not on TBI but exclusively on the node where the fault is injected. It can be seen in the results that when nodes 1, 2 or 4 fail the recovery TBO is of approximately 5385. When the fault is at node 3 the recovery TBO is of approximately 6200. These values are dependent on where the fault is injected and not on the value of the system's TBI. The same conclusion can be reached for the values of introduced delay and first output delay as well.

It is interesting to note that the value of TBIO reduction, i.e., the value by which TBIO is reduced while the system is recovering, depends on

the value of TBI and which node failed. This TBIO reduction is equal to TBI minus recovery TBO. It should be noted that when the fault is at node 2 the TBIO reduction is equal to the system slack as defined in Chapter Two.

Another conclusion is that the higher the TBI, the faster the system reaches the target operating point. For example, when node 4 fails, the system reaches the target operating point in 24,108 time units for a TBI of 8000, whereas it takes 43,652 time units for a TBI of 6200. This is an important factor to consider when designing systems to withstand faults and to reach an operating point within certain critical time. The penalty that is paid by increasing the TBI is that the system does not operate at optimal steady state throughput. The model helps in the decision making at design time by allowing the designer to choose the most suitable solution for the application at hand, with the full knowledge of advantages and disadvantages of a given operating point. The designer is able to balance steady state performance versus transient state performance.

As a final conclusion, it should be observed that the objectives of the dissertation were successfully achieved. The model has been used to estimate the time the system takes to recover from a fault and reaches the target operating point. It has also been used to determine whether the system reaches the target operating point at all. If the system does not reach the target operating point, it can be determined what operating point

it reaches. If a permanent delay is introduced into the system's TBIO, its value can be calculated as well. The determinism of the original steady state ATAMM model has been carried over to the transient state ATAMM extension.

The research presented in this dissertation is original and it has not been pursued before under the conditions exposed here. ATAMM had been used only to predict steady state performance in a multicomputer data flow system. With the extension to ATAMM, this research adds to the multicomputer systems analysis the capability of predicting the transient behavior of a system during recovery and restoration. It adds to the fault-tolerant computer the ability to evaluate whether a particular fault-tolerant technique applied to the system yields the expected results of bringing the system to the target state.

5.1 Future Research

The model presented in this dissertation has opened other avenues to explore multicomputer systems designed with the ATAMM model. In particular, the transient behavior of the systems can be explored to design and deliver highly reliable and robust multicomputer systems. The model may not be only used to highlight potential problems but it may also be used to solve them.

The delay propagation model may be applied not only to this kind of problem but also to project management. This application to project

management is restricted to projects with the characteristics of the systems

analyzed here. The projects should be of iterative nature for this model to

be successfully applied. Another area where the model may be applied is

to assembly-line type of systems as is the case of car factories. This

generality makes the model powerful and versatile.

The analysis may be easily extended to accept more than one fault at

a time. The assumption throughout the dissertation has been that there is

only one fault present in the system until the system reaches a steady

state. If two faults were assumed to occur close to each other in time, such

that the effect of the first has not disappeared from the system before the

second arrives, the effect of both faults may be said to be overlapping. If

the faults do not overlap, the analysis is simplified since each one can be

explored individually as has been presented. If both faults overlap, it

might be possible to estimate the effect of each fault separately and the

effects might be combined to obtain the overall effect. This combination

might be performed at points of interest such as the sinks. For a given

sink in the graph, the value of delay caused by the first fault and the value

of delay caused by the second fault may be compared by a given function

determine the effective delay to the sink. This method might also be

extended to study more than two faults that overlap.

The model may be used to characterize systems in their transient

behavior. As it can be seen in the data presented in Chapter Four, systems

may be studied based on their characteristics to recover from a fault. The graph's topology and time information determine how a system responds to the introduction of delay into one of its nodes. Previous research in the ATAMM model has been focused on the system's steady state behavior. The model presented in this dissertation extends the usefulness of the ATAMM model into the analysis of the system's transient modes.

It has been shown that when the systems are used at top performance the system may not reach the target operating point after a fault. A solution to this problem is to drive the system at a higher TBI as in some of the experiments, providing extra token lifetime throughout the graph. This solution gives a good recovery time but the steady state performance is not optimum. If steady state system performance were at premium, it may be possible to discard one or more data packets at the input and to artificially increase the token lifetime but only in the event of a fault. This might provide good performance during steady state but it would be a performance degradation during recovery in the form of data loss. This might be a viable solution depending on the specific application and the model provides an avenue to explore alternative strategies to solve the potential problems, providing the knowledge of the trade-off if one or another solution is implemented.

One possible use of the model is to help in the investigation of the dominant paths in the graphs. The search of all paths and their corre-

sponding token lifetimes is tedious and time consuming. Therefore, it is desirable to explore the possibility of formulating theorems that would help in finding the token lifetime between two nodes in the system without having to do such an exhaustive search and computation. The formulation of such theorems will help in the study of graphs and implementation of efficient research software tools. It was observed during the calculations in the experiments in Chapter Four that there are patterns and cyclic behavior due to the unfolding nature of the model. These characteristics can be further explored using this model as a the main tool and to help classify graphs by their topology, node times and recovery behavior.

The model may also be useful to explore the variable time node problem. As is, the model helps in the understanding of the propagation of delay in a system. This delay is not restricted to be produced by a fault in a node. The delay may be normal to the operation of a system as is the case of a variable time node graph. If every node in the graph were to have a different time every instance it is run, this variation may be considered as a delay introduced with respect to a mean value of the process time of the nodes. Since in such a scenario a node may not only introduce a delay but also introduce a speedup in the process, an extension to the definition of token lifetime might be performed. The idea of speedup propagation might be pursued as a symmetric measure to delay propagation. It was observed that, in parallel paths, the path with the minimum process time

would have the maximum token lifetime. In the case of a node that finishes earlier than expected, it adds token lifetime to the path. If the node is in the lifetime dominant path between two nodes it effectively increases the token lifetime and the ability to absorb delay increases between the two nodes. These ideas might be worth exploring using the model presented in this dissertation.

In the case of variable time nodes, the values for the node's process times might be expressed by random variables. In such instance, the values for token lifetime in the paths would also become random variables. These values of token lifetime would be dependent of several other random variables so their probability density functions become dependent on the pdf's of the node's process times. The lifetime dominant path between two nodes would be expressed by a random variable dependent on the various token lifetimes of the paths between the nodes. Among the questions that could be addressed is that of whether a system is stable under certain conditions such as being driven at a given TBI. As an example, if the system is intended to be run at best average performance there might be a probability that the system become unstable once a certain value of internal delay has been reached. The value of the probability that this critical delay is reached may be found by extending this model to include variable time nodes. These and other questions may be addressed by the

model by extending its usefulness beyond the deterministic value of the node's times.

Considering the behavior under the conditions of variable time nodes, it is also possible to expect that the system's operating point would not be contained in a region about a stable point in the operating point plane. It may be of interest to know if there are unstable regions that a system's operating points would fall into under these conditions. The question of whether there is a strange attractor in the data derived from these operating points is of interest to dynamical systems analysts.

The study of the multicomputer systems as dynamical systems may be achieved by extending this performance model. Although the systems are deterministic in nature, under certain conditions the systems may seem unpredictable and of random behavior. It is these cases where it is inter-esting to study the possibility that a system may be stable within a region, i.e., chaotic or unpredictable about sequence of instantaneous operating points but predictable about the confined region within which all operating points would fall. This may be used to broadly classify systems as either stable or unstable.

BIBLIOGRAPHY

[1]   T. Anderson and P. Lee, *Fault Tolerance Principles and Practice.* Englewood Cliffs, NJ; Prentice-Hall, 1981.

[2]   David A. Rennels, "Fault-Tolerant Computing - Concepts and Examples," *IEEE Transactions on Computers,* Volume C-33, Number 12, pages 1116-1129, Dec. 1984.

[3]   John A. Stankovic and Krithi Ramamritham, "Editorial: What is Predictability for Real-Time Systems?," *Real-Time Systems,* Kluwer Academic Publishers, Netherlands, 1990.

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

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

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

[7]   Victor P. Nelson, "Fault-Tolerant Computing: Fundamental Concepts," *Computer,* Vol. 23, Number 7, pages 19-25, July 1990.

[8]   R. Mielke, J. Stoughton, S. Som, R. Obando, M. Malekpour, and B. Mandala, "Algorithm to Architecture Mapping Model (ATAMM) Multicomputer Operating System Functional Specification," NASA Contractor Report 4339, Cooperative Agreement NCC1-136, November 1990.

[9]   Herbert Y. Chang, "An Algorithm for Selecting an Optimum Set of Diagnostic Tests," *IEEE Transactions on Electronic Computers,* Vol. EC-14, No. 5, pages 706-711, October 1965.

[10] Franco P. Preparata, Gernot Metze, and Robert T. Chien, "On the Connection Assignment Problem of Diagnosable Systems," *IEEE Transactions on Electronic Computers*, Vol. EC-16, No. 6, pages 848-854, December 1967.

[11] Ferruccio Barsi, Fabrizio Grandoni, and Piero Maestrini, "A Theory of Diagnosability of Digital Systems," *IEEE Transactions on Computers*, Vol. C-25, No. 6, pages 585-593, June 1976.

[12] Kyung-Yong Chwa and S. Louis Hakimi, "On Fault Identification in Diagnosable Systems," *IEEE Transactions on Computers*, Vol. C-30, No. 5, pages 414-422, June 1981.

[13] Anton T. Dahbura and Gerald M. Masson, "An $O(n^{2..5})$ Fault Identification Algorithm for Diagnosable Systems," *IEEE Transactions on Computers*, Vol. C-33, No. 6, pages 486-492, June 1984.

[14] Fred J. Meyer, and Dhiraj K. Pradhan, "Dynamic Testing Strategy for Distributed Systems," *IEEE Transactions on Computers*, Vol. 38, No. 3, pages 356-365, March 1989.

[15] Tein-Hsiang Lin, and Kang G. Shin, "Location of a Faulty Module in a Computing System," *IEEE Transactions on Computers*, Vol. 39. No. 2, pages 182-194, February 1990.

[16] Krishna Kant, "Performance Analysis of Real-Time Software Supporting Fault-Tolerant Operation," *IEEE Transactions on Computers*, Vol. 39, No. 7, pages 906-918, July 1990.

[17] A. Avizienis, "The N-Version approach to fault-tolerant software," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 12, pages 1491-1501, December 1985.

[18] A. Ralston & E. D. Reilly, Jr. Editors, *Encyclopedia of Computer Science and Engineering, Second Edition*, New York, N.Y., Van Nostrand Reinhold, pages 1060-1075, 1983.

[19] T. Anderson & P.A. Lee, *Fault Tolerance: Principles and Practice*, London, England, Prentice-Hall International, Inc., pages 63-67, 1981.

[20] T. Anderson & P.A. Lee, *Fault Tolerance: Principles and Practice*, London, England, Prentice-Hall International, Inc., pages 113-142, 1981.

[21] T. Anderson & P.A. Lee, *Fault Tolerance: Principles and Practice*, London, England, Prentice-Hall International, Inc., pages 147-169, 1981.

[22] T. Anderson & P.A. Lee, *Fault Tolerance: Principles and Practice*, London, England, Prentice-Hall International, Inc., pages 173-225, 1981.

[23] T. Anderson & P.A. Lee, *Fault Tolerance: Principles and Practice*, London, England, Prentice-Hall International, Inc., pages 231-247, 1981.

[24] D.B. Lomet, "Process Structuring, Synchronization and Recovery Using Atomic Actions," SIGPLAN Notices vol. 12, No. 3, pages 128-137, March 1977.

[25] S. Rangarajan, and D. Fussell, "Diagnosing Arbitrarily Connected Parallel Computers with High Probability," IEEE Transactions on Computers, Vol. 41, No. 5, pages 606-615, May 1992.

[26] R. Gerst, A. Jefferson Offutt, and F. C. Harris, "Estimation and Enhancement of Real-Time Software Reliability through Mutation Analysis," IEEE Transactions on Computers, Vol. 41, No. 5, pages 550-558, May 1992.

[27] J. W. Stoughton, R. R. Mielke, S. Som, R. Obando, M. R. Malekpour, R. L. Jones, Brij Mohan V. Mandala, "ATAMM Enhancement and Multiprocessor Performance Evaluation," NASA Langley Year End Report for 1990, Grant NCC1-136, pages 33-38, June 1991.

APPENDIX A

This appendix is used to illustrate how TBO and TBIO are calculated

for the experiments in Section 4.4 using the material developed in Chapter

Two. The values of the token lifetime used in this illustration do not

necessarily reflect a particular experiment. The XCMG for the 'Application

Algorithm' is shown in Figure A.1. The values shown on some edges are

the values for the corresponding token lifetime. The node denoted with the

letter A is where the delay is introduced. The sink denoted with the letter

B relates to the data packet for which the values of TBO and TBIO are to

be calculated. $LEP_{1,2}$ indicates the edge between nodes 1 and 2.

The first step is to identify all the directed paths from node A to sink

B. These paths are highlighted in Figure A.2. These are the paths that

are of interest in the computation of the dominant equivalent path from

node A to sink B.

The concatenation operator is applied to edges $e_{10,11}$ and $e_{11,8}$; $e_{2,12}$,

$e_{12,13}$, and $e_{13,8}$; and $e_{8,9}$ and $e_{9,B}$. The resultant graph is shown in Figure

A.3. The operations are

$$LEP_{10,11} + LEP_{11,8} = LEP_{10,8},$$
$$LEP_{2,12} + LEP_{12,13} + LEP_{13,8} = LEP_{2,8}$$
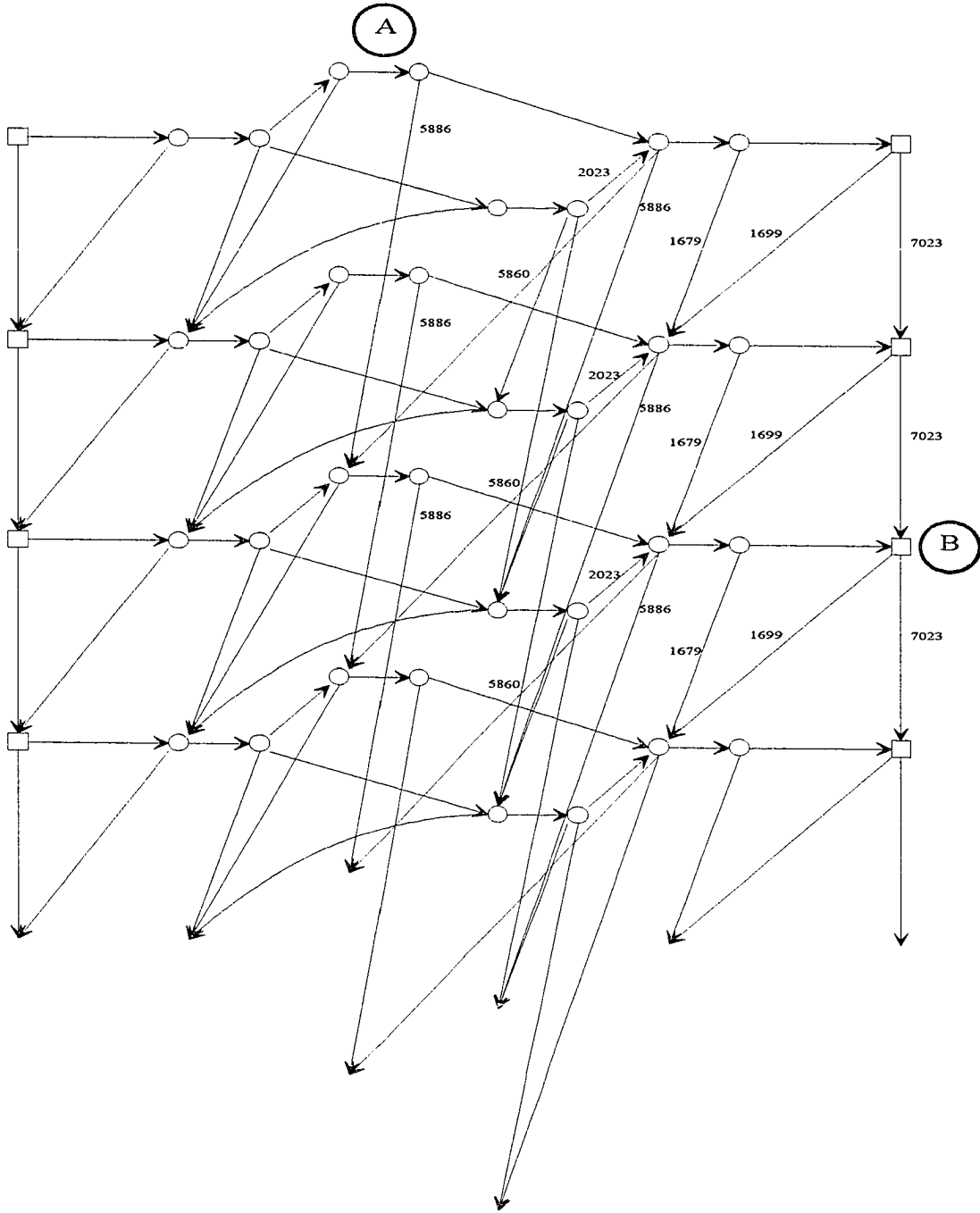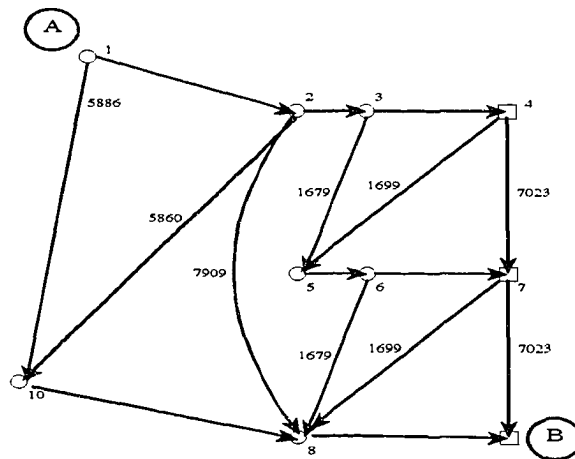$$LEP_{8,9} + LEP_{9,B} = LEP_{8,B}.$$
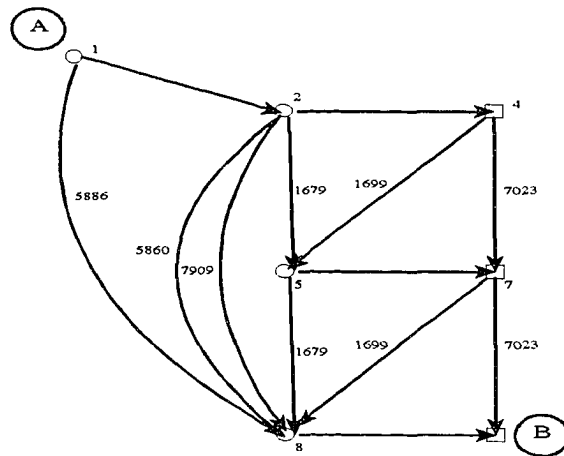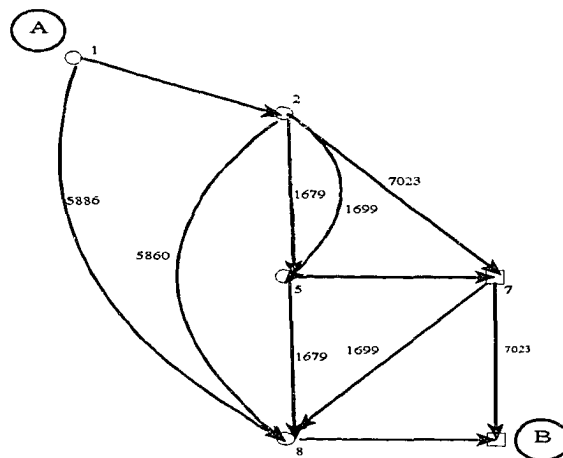
163

Figure A.1.

Figure A.2.

Figure A.3

Figure A.4

Figure A.5

Distribution of the concatenation operator over the parallel operator

is applied to edges $e_{2,3}$, $e_{3,4}$ and $e_{3,5}$; $e_{5,6}$, $e_{6,7}$ and $e_{6,8}$; and $e_{1,10}$, $e_{2,10}$ and

$e_{10,8}$. After the operations

$$LEP_{2,3} + LEP_{3,4} \| LEP_{3,5} = LEP_{2,4} \| LEP_{2,5},$$

$$LEP_{5,6} + LEP_{6,7} \| LEP_{6,8} = LEP_{5,7} \| LEP_{5,8}, \text{ and}$$

$$LEP_{1,10} \| LEP_{2,10} + LEP_{10,8} = LEP_{1,8} \| LEP_{2,8}$$

the graph in Figure A.4 is obtained.

The dominant path between 2 and 8 is obtained. Distribution of the concatenation operator over the parallel operator

$$LEP_{2,4} + LEP_{4,5} \| LEP_{4,7} = LEP_{2,5} \| LEP_{2,7}$$

is applied; and the graph in Figure A.5 results.

The dominant path between 2 and 5 is obtained. Distribution of the concatenation operator over the parallel operator is applied,

$$LEP_{1,2} + LEP_{2,5} \| LEP_{2,7} \| LEP_{2,8} = LEP_{1,5} \| LEP_{1,7} \| LEP_{1,8}$$

resulting in the graph shown in Figure A.6.

The dominant path between 1 and 8 is obtained. Distribution of the concatenation operator over the parallel operator is applied,

$$LEP_{1,5} + LEP_{5,8} \| LEP_{5,7} = LEP_{1,8} \| LEP_{1,7}$$
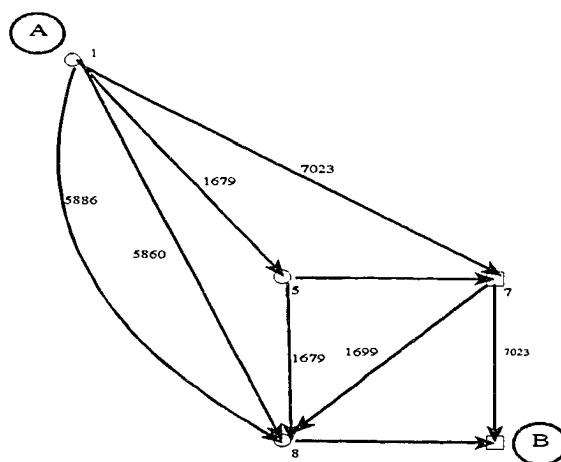
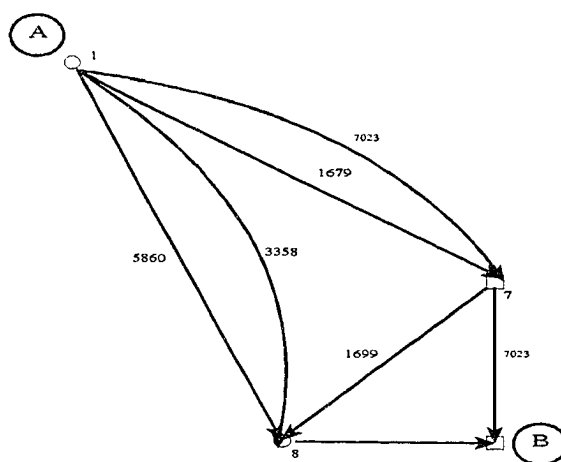resulting in the graph in Figure A.7.
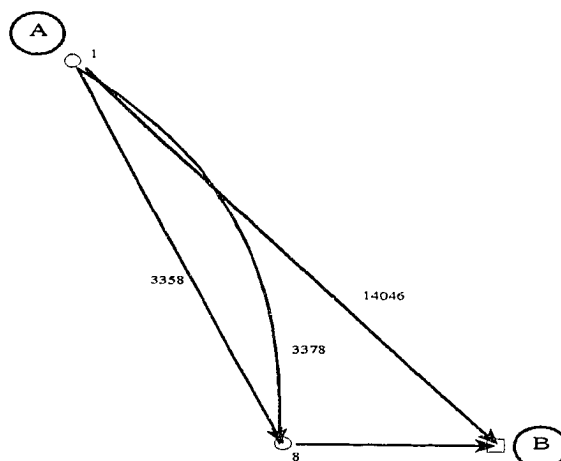
Figure A.6

Figure A.7

Figure A.8

Dominant paths between 1 and 8, and 1 and 7 are found.

Distribution of the concatenation operator over the parallel operator is

applied,

$$LEP_{1,7} + LEP_{7,8} \| LEP_{7,B} = LEP_{1,8} \| LEP_{1,B}$$

resulting in the graph in Figure A.8.

The dominant path between 1 and 8 is found. Concatenation operator

is applied,

$$LEP_{1,8} + LEP_{8,B} = LEP_{1,B},$$

resulting in the graph in Figure A.9.

The dominant path between 1 and B is found. The dominant path is

shown in Figure A.10.

The token lifetime between node A and sink B is equal to 3358.

Assuming that the delay introduced is 8643, the delay to fire the sink is

8643 - 3358 = 5285. If the value for TBIO is normally 16650, the value of

TBIO for that particular data packet to be delivered at sink B is 16650 +

5285 = 21935.

Assuming that the delay to fire the sink prior to sink B is 8643 - 1679

= 6994, the value for that data packet TBIO is 16650 + 6994 = 23644.

Assuming that the sink prior to sink B should have fired at time t_{B-1},

and that sink B should have fired at time t_B, the value of TBO between
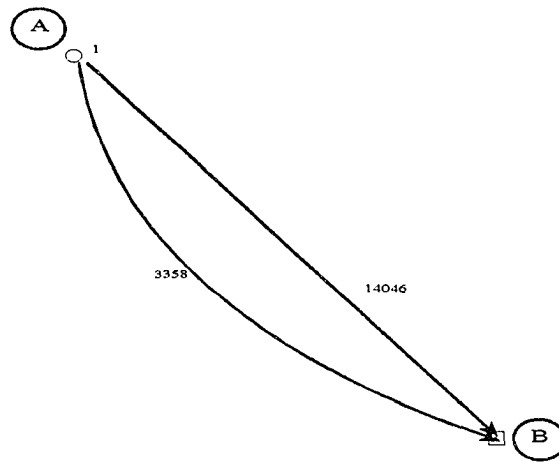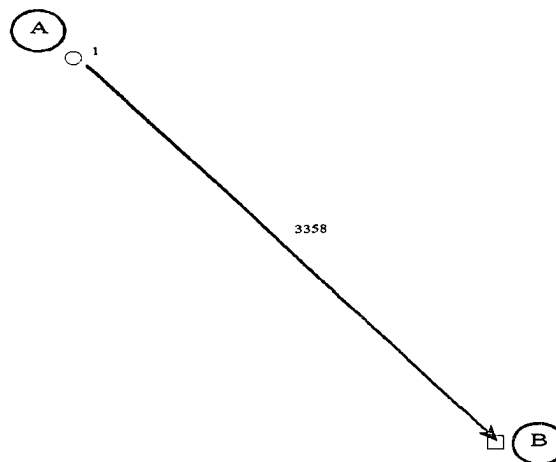
these two sinks is calculated by

Figure A.9

Figure A.10

$$TBO = t_B + 5285 - (t_{B-1} + 6994)$$
$$= t_B + 5285 - t_{B-1} - 6994$$
$$= t_B - t_{B-1} + 5285 - 6994$$
$$= 7023 + 5285 - 6994$$
$$= 5314,$$

where $t_B - t_{B-1}$ = 7023 or TBI under normal operation.

By using this procedure for each one of the 30 sinks in the XCMG,

each TBO and TBIO can be calculated. Every entry in the tables in Section

4.4 were calculated this way.

Autobiographical Statement

Rodrigo A. Obando was born in San José, Costa Rica, in June, 8th,

1957. He attended the Instituto Tecnológico de Costa Rica from January,

1975 to December, 1978, where he obtained his B.S. in Electronics

Engineering. Rodrigo attended Old Dominion University from August,

1985 to December, 1987, where he obtained his M.E. in Electrical

Engineering. Since then he has worked on his Ph.D. in Electrical

Engineering also at Old Dominion University.

PUBLICATIONS

Master's Thesis

"**Software Tools for Performance Evaluation of Concurrent Processing,**" August, 1987.

Papers

1. S. Som, R. Obando, R. Mielke, and J. Stoughton, "ATAMM: A Computational Model for Real-Time Data Flow Architectures," Accepted for Publication in the International Journal of Mini and Microcomputers of ACTA Press, Paper No. 202-662, ISMM, Canada.

2. S. Som, R. Mielke, R. Obando, J. Stoughton, P. J. Hayes, and R. L. Jones, "Throughput Enhancement by Multiple Concurrent Instantiations in the ATAMM Data Flow Architecture," Proceedings of the ISMM International Symposium on Computer Applications in Design, Simulation, and Analysis, pp. 71-74, Las Vegas, Nevada, March 19-21, 1991.

3. S. Som, R. Obando, R. R. Mielke, and J. W. Stoughton, "ATAMM: A Computational Model for Real-Time Data Flow Architectures," <u>Proceedings of the ISMM International Conference on Parallel and Distributed Computing, and Systems</u>, ACTA Press, pp. 241-245, Held in New York City, New York, October 10-12, 1990.

4. M. Malekpour, R. Obando, R. R. Mielke, and J. W. Stoughton, "ATAMM Simulation Tool for Data Flow Architectures," <u>Proceedings of the 21st Annual Pittsburg Conference on Modeling and Simulation</u>, pp. 953, May 3-4, 1990.

5. Alvertos, J. Champaneri, and R. Obando, "An Algorithm for Image Analysis Using Pattern Connectivity and Cluster Compactness," <u>Procedings of the 6th Scandinavian Conference on Image Analysis</u>, pp. 898-901, Oulu, Finland, June 19-22, 1989.

6. Alvertos, J. Champaneri, and R. Obando, "An Algorithm for for Determining Pattern Connectivity and Cluster Compactness," <u>Procedings of the IEEE SOUTHEASTCON '89</u>, Vol. 2, pp. 614-616, Columbia, South Carolina, April 9-12, 1989.

7. K. Jackson, W. R. Tymchyshyn, R. R. Mielke, John W. Stoughton, and R. Obando, "Simulation Software for Concurrent Processing," *Proceedings of Southeastcon Conference 87*, April, 1987.

Reports

1. J. W. Stoughton, R. R. Mielke, S. Som, R. Obando, M. R. Malekpour, R. L. Jones, Brij Mohan V. Mandala, "ATAMM Enhancement and Multiprocessor Performance Evaluation," <u>NASA Langley Year End Report for 1990</u>, Grant NCC1-136, June 1991.

2. R. R. Mielke, J. Stoughton, S. Som, R. Obando, M. Malekpour, and B. Mandala, "ATAMM Multicomputer Operating System Functional Specification," <u>NASA Contractor Report 4339</u>, Grant NCC1-136, November 1990.

3. J. W. Stoughton, R. R. Mielke, S. Som, R. Obando, R. Tymchyshyn, "Strategies for Concurrent Processing of Complex Algorithms in Data Driven Architectures," <u>NASA Langley Progress Report</u>, Grant NAG1-683, May 16, 1987 to May 15, 1988.

Societies

Member of Institute of Electrical and Electronics Engineers (IEEE),

and IEEE Computer Society since 1986.

Awards

Team Excellence Award, given by National Aeronautics and Space

Administration, Langley Research Center, on December 3rd,1992, for

the successful design, implementation, and performance verification of

a real-time dataflow strategy in a spaceborne multiprocessor

configuration.