RADC-TR-88-41
Final Technical Report
February 1988

EVALUATION OF PARALLEL ARCHITECTURES FOR BM/C^3 APPLICATIONS

Pennsylvania State University

C. R. Das, W. Lin, M. J. Thazhuthaveetil and T. Feng

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC TR-88-41 has been reviewed and is approved for publication.

APPROVED:

DAVID F. TRAD
Project Engineer

APPROVED:

RAYMOND P. URTZ, Jr., Technical Director
Directorate of Command and Control

FOR THE COMMANDER:

JOHN A. RITZ
Directorate of Plans and Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTC) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

ADA197357

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS: N/A
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S): RADC-TR-88-41
6a. NAME OF PERFORMING ORGANIZATION: Pennsylvania State University; 6b. OFFICE SYMBOL (If applicable):	7a. NAME OF MONITORING ORGANIZATION: Rome Air Development Center (COTC)
6c. ADDRESS (City, State, and ZIP Code): Computer Engineering Program, Dept of Electrical Engineering, University Park PA 16802	7b. ADDRESS (City, State, and ZIP Code): Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Rome Air Development Center; 8b. OFFICE SYMBOL (If applicable): COTC	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F30602-81-C-0169
8c. ADDRESS (City, State, and ZIP Code): Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 63223C; PROJECT NO. B413; TASK NO 03; WORK UNIT ACCESSION NO. P6

11. TITLE (Include Security Classification)
EVALUATION OF PARALLEL ARCHITECTURES FOR BM/C^3 APPLICATIONS

12. PERSONAL AUTHOR(S)
C.R. DAS, W.Lin, M.J. Thazhuthaveetil, T. Feng

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Apr 87 TO Sep 87	February 1988	98

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Computer Architecture, Evaluation, Performance
12			

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Several parallel computer systems are commercially available today. They could be divided into three main classes based on the technique used to connect the processing and memory elements of the system together - multistage interconnection network (MIN) based systems, bus based systems, and hypercube systems. Commercial examples of these system types are the BBN ACI Butterfly, the Encore Corp. Multimax, and the Intel Corp. iPSC respectively. The task of deciding which kind of parallel system is best suited for a particular programming application domain is a complex one; no well defined guidelines or decision assisting tools are currently available. This report describes a series of parallel system evaluation efforts being conducted with the BM/C^3 application domain in mind. Emphasis is placed on the BBN ACI Butterfly Parallel Processor.

One aspect of the research is the development of a software tool that can be used to conduct application dependent performance evaluation studies on the Butterfly. Called the Butterfly Performance Predictor, this tool consists of a system simulator and a code simulator. Using user provided descriptions of the algorithms of interest in terms of a small set of parameters, (Cont'd)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
DAVID F. TRAD	(315) 330-2925	RADC/COTC

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

AIR FORCE 86145/ 27-5-88 - 168

UNCLASSIFIED

19. (Cont'd) the tool generates estimates of various performance based on basic instruction execution speeds obtained from metrics processor data books. The structure of the tool, which is under development, is described.

A second aspect of the research effort described in this report is the mapping of a specific battle management algorithm onto the Butterfly parallel processor. The goal of this mappping procedure is to minimize the amount of contention for shared memory and communication links by the individual processing components. A tree-shaped process structure is suggested and evaluated using a simplified analytical technique. The mapping procedure is applicable to other algorithms with similar data flow properties.

No performance evaluation study would be truly complete without a study of the dependability of the underlying system. None of the existing reliability evaluation tools are capable of computing the reliability of the Butterfly or Hypercube systems. An analytical model for computing Butterfly reliability is presented. The model is based on the decomposition technique. A recursive equation is derived to compute the reliability of a 4^i system from four 4^{i-1} subsystems. Analytical results are given for 16-node, 64-node, and 256-node Butterfly configurations.

A new analytical technique to compute the reliability of n-dimensional hypercube systems is also described. A recursive equation is derived to compute the n-cube reliability from a 2-cube or 3-cube base model. Analytical results are presented for up to 8-dimensional hypercubes.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The availability of a variety of commercial multiprocessor computers today makes it difficult to decide on the optimal machine for any specific parallel application area. In addition to the strengths and weaknesses of each candidate machine, the characteristics of programs in the target application domain must be taken into account in making this selection. Unfortunately, neither formal techniques nor software tools are currently available to assist in this decision process. The development effort described in this report addresses the problem of deciding which class of parallel computer systems is best suited to the BM/C3 problem domain. In particular, this report describes the development of software tools for application dependent performance and dependability estimation of available parallel computers.

Most commercially available parallel computer systems can be classified as **bus based** systems, **multi-stage interconnection network** (MIN) based systems, or **hypercube** systems. Bus based multiprocessors consist of processors, memory modules and other devices, connected to each other through a simple computer bus. Examples of this class of system include the Encore Multimax, the Sequent Balance and Symmetry series, and the Synapse N+1 system, with new products announced regularly. These systems are typically restricted in size to a maximum of a few tens of processors due to the performance limitations of current computer buses. In MIN based multiprocessors, the processors, memory modules and other devices are connected through a network of stages of switching elements. The BBN ACI Butterfly system is one commercially available example of a MIN based multiprocessor. The power of the Butterfly MIN makes multiprocessor systems with hundreds of processors cost effective. Hypercube multiprocessors are a relatively new entrant in the parallel processor arena. A hypercube system consists of 2**n processor-memory modules, with each module directly connected to n-1 neighbors, forming an n-dimensional cube.

Hypercube systems with up to 1024 processor modules are commercially available from Intel Corp, NCUBE Corp, and Ametek. In this report the Butterfly, Hypercube, and Multimax machines are considerd as candidate architectures for BM/C3 applications.

A major goal in the design of parallel architectures is to provide high computing power with assured dependability. High computing power can be provided by exploiting the parallelism in the application algorithms and by mapping these parallel algorithms onto the candidate architectures. Performance evaluation of multiprocessors using the above three types of interconnection topologies have been addressed by various researchers using analytic and simulation models [Bhuyan 84, Das 85, Dias 81, Kruskal 83, Lang 82, Lin 88, Marsan 82, Mudge 84, Read 87, Wittie 81, Wu 84]. However, most of these studies are restricted to evaluation of the architecture. Consideration of both architecture and algorithms in performance evaluation has received little attention to date. We discuss this aspect of performance evaluation in Section 2 of this report.

The second requirement, "assured dependability" of parallel architectures, stems from the critical applications in which these machines are used. The performance analysis of the parallel systems outlined above implicitly assumes that the components of a system are fault free. These results give the so called "ideal" performance of a system. However, in a real situation the components of a system fail at random depending on the failure rates of the components. At the system level, a multiprocessor consists of two subsystems. One subsystem is the computation facility which is provided by processors (nodes) and memories. The second subsystem is the communication network, used to support interprocessor communication. The failure of a processor (node) or a memory unit reduces the hardware resources available on the system. The failure of the interconnection switches or links degrades the communication capability of the network. All these faults affect the dependability and performance of the system to varying degrees. A common approach to improve the fault-tolerance of these parallel systems is to provide graceful degradation as an inherent attribute of a system.

Following Laprie [Laprie 82], dependability is defined as "the quality of service

delivered by the system such that reliance can be justifiably placed on the service." Dependability is a generic concept that encompasses reliability, availability, maintainability, and safety as distinct facets of system specification. It has been reported [Avizienis 78] that there is a clear need for quantitative measurement of dependability parameters.

At the system level specification, fault-tolerant systems are categorized as either highly reliable or highly available [Siewiorek 82]. Most of the work in fault-tolerant evaluation of parallel computers is confined to reliability modeling. This is mainly because availability evaluation is more complex than reliability evaluation.

Reliability evaluation of parallel systems has been studied under two different approaches, namely: terminal reliability and task based reliability [Ingle 77, Raghavendra 84]. Terminal reliability is defined as the probability that at least one communication path exists between a pair of nodes. This may be an oversimplified estimate for parallel systems where a job (task) is executed concurrently over several nodes. The task based reliability, on the other hand, assumes that a system remains operational as long as a task can be executed with the available resources on the system. This is a more appropriate measure of reliability in a parallel processing domain.

Task based dependability evaluation of some parallel computers have been addressed by diferent researchers [Arlat 83, Das 85, Das 87, Hwang 82, Ingle 77]. These studies are not complete from different perspectives. For example, none of the models combine architecture, algorithm requirements, and software issues, to model the system behavior completely. This has been handicapped mostly due to the complexity of the parallel machine architecture. In particular, the exact reliability modeling of the communication networks, such as the MIN, is quite complex and can lead to NP-hard problems [Provan 86]. Hence, very little research effort has been directed to model the dependability of parallel computers combining both the computation and communication degradation.

While classical dependability measures such as reliability and availability are suitable to evaluate uniprocessor systems, these measures may not be good indicators of

parallel system behavior. Dependability measures specify only the operational status of a system at any time **t**. No performance statistics can be gathered from the reliability or availability study. High performance being the main objective of parallel architectures, performance-related-dependability evaluation is essential to evaluate these architectures. This evaluation will specify, for example, the execution time of a job in a real environment when all kinds of component failure and repairs are possible.

Performance related dependability measures are relatively new compared to classical dependability theory. A number of performance-related dependability measures such as computation reliability, computation availability [Beaudry 78], performability [Meyer 80], capacity and workload characterization [Gay 79], have been proposed for degradable multiprocessors. However, none of these models have been applied in a real sense to the candidate architectures in consideration.

There are several automatic program packages such a ARIES [Makam 82], CARE III [Stiffler 82], HARP [Geist 83], SAVE [Goyal 87], and SHARPE [Sahner 87] available for computing the dependability of complex systems. Markov models of a system are used to compute the reliability/availability of the system using numerical techniques. However, these packages are not general enough to handle the parallel architectures under investigation. The difficulty lies in generating the Markov states of a system such as the Butterfly or Hypercube. To our knowledeg there is no tool available today that can generate the Markov chain of the above systems automatically. The capabilities and weaknesses of some of the packages are reported in Section 4.

CHAPTER 2

PERFORMANCE EVALUATION OF PARALLEL ARCHITECTURES

It was decided to commence this study by concentrating on the MIN based Butterfly parallel processor. This section describes the development of tools to assist in the evaluation of the performance of such a MIN based computer system; these tools are referred to as the Butterfly Performance Predictor.

2.1. Butterfly Performance Predictor Overview

The general operation of the Performance Predictor is illustrated in Figure 2.1. It estimates performance metrics based on two kinds of data: architectural parameters (detailed information about the parallel machine architecture), and algorithm parameters (information about algorithms from the application domain under study). Theoretically, such a performance predictor could be used to study different parallel processor architectures by merely varying the architectural parameters. In practice, it is difficult to conceive of a set of parameters powerful enough to categorize bus-based systems, MIN based systems, and hypercube systems in sufficient detail to allow reasonable accuracy of performance prediction. A more conservative design goal was employed in this effort; the architectural parameters were chosen to enable the user to study parallel machines "similar" to the Butterfly.

Figure 2.2 shows the Performance Predictor in more detail; its main component is a **Butterfly Simulator** - a program that simulates program execution on a Butterfly while accumulating performance measures. To drive the simulator under conditions representative of the target application domain, two strategies are considered. In the first strategy, real Butterfly programs are used. This scheme has obvious drawbacks: it requires the simulator to be sophisticated enough to process actual Butterfly machine code and also requires access to BM/C3 programs coded specifically for the Butterfly. A more flexible and user-friendly strategy is to drive the simulator with synthetically

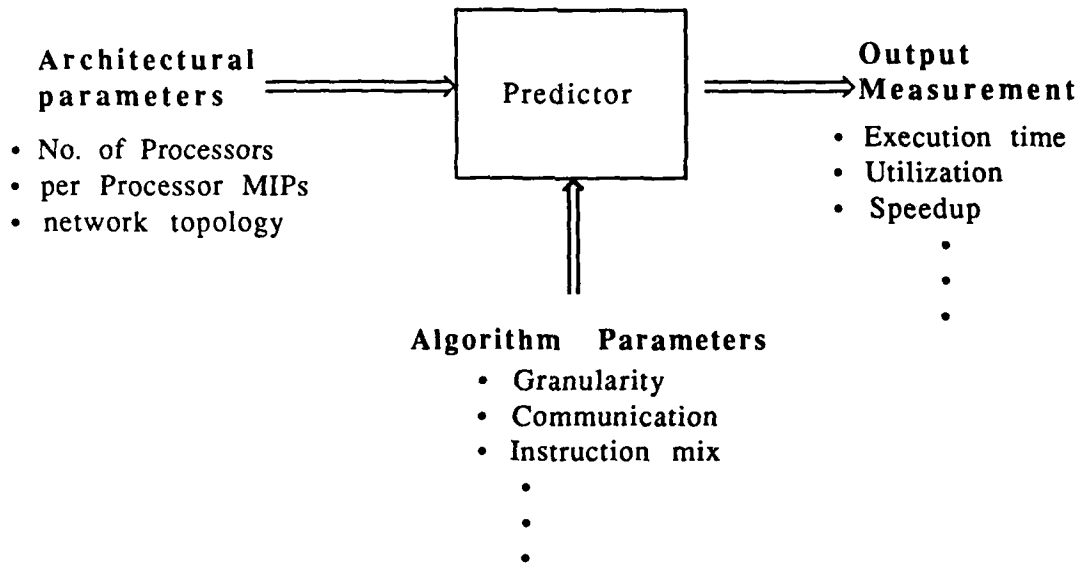


Figure 2.1 Butterfly Performance Predictor

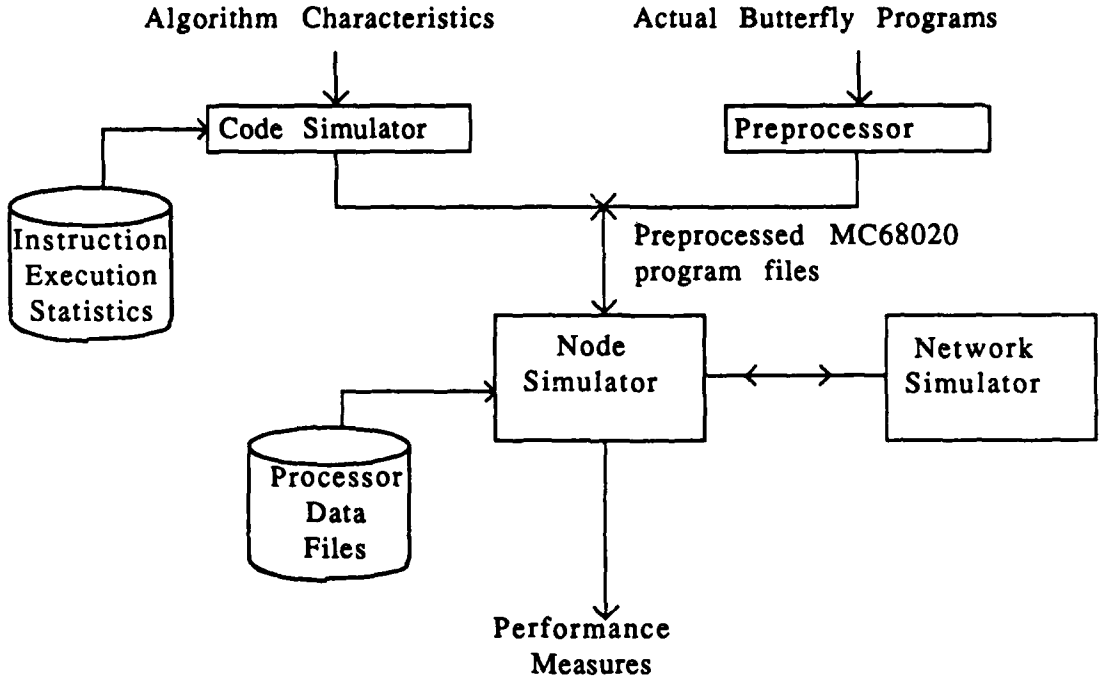


Figure 2.2 Structure of Performance Predictor

generated instruction streams that are representative of the target application domain. The second key component of the Performance Predictor, therefore, is a program (called the **Code Simulator**) that generates these instruction streams.

2.2. Butterfly Parallel Processor

The Butterfly multiprocessor system is made up of processor nodes and a Butterfly interconnection network as shown in Figure 2.3. The network is depicted as a cylinder since both its inputs and outputs are processor nodes, unlike a conventional "dance-hall" multiprocessor architecture, which would have processors at one end and memory modules at the other. All of the distributed memory is globally accessible. Remote memory accesses are conducted through the network. Each processor node contains a processor (currently a Motorola 68020), an arithmetic co-processor (MC68881), 1-4 Megabytes of memory, memory management hardware, and an interface to the network, as illustrated in Figure 2.4

2.3. Butterfly Simulator

The Butterfly Simulator is to contain two components: a **network simulator** and a **node simulator**. The network simulator maintains the status of the Butetrfly network while producing timing estimates of how long it takes to traverse the network. The node simulator accepts Buterfly programs as input and estimates their execution time. It uses the network simulator for timing information related to the Butterfly MIN, and uses a set of files of architectural information to do its own timing estimation. These files contain the "architectural parameters" mentioned earlier in this report, and are referred to as the processor files.

The program execution timing estimates are made at the instruction level. The execution of the program is traces instruction by instruction, and the time taken for each instruction is computed based on timing information obtained from Motorola data books for the MC68020 and the MC68881 and accumulated in the Performance Predictor's processor files: these files contain, for each instruction-addressing mode

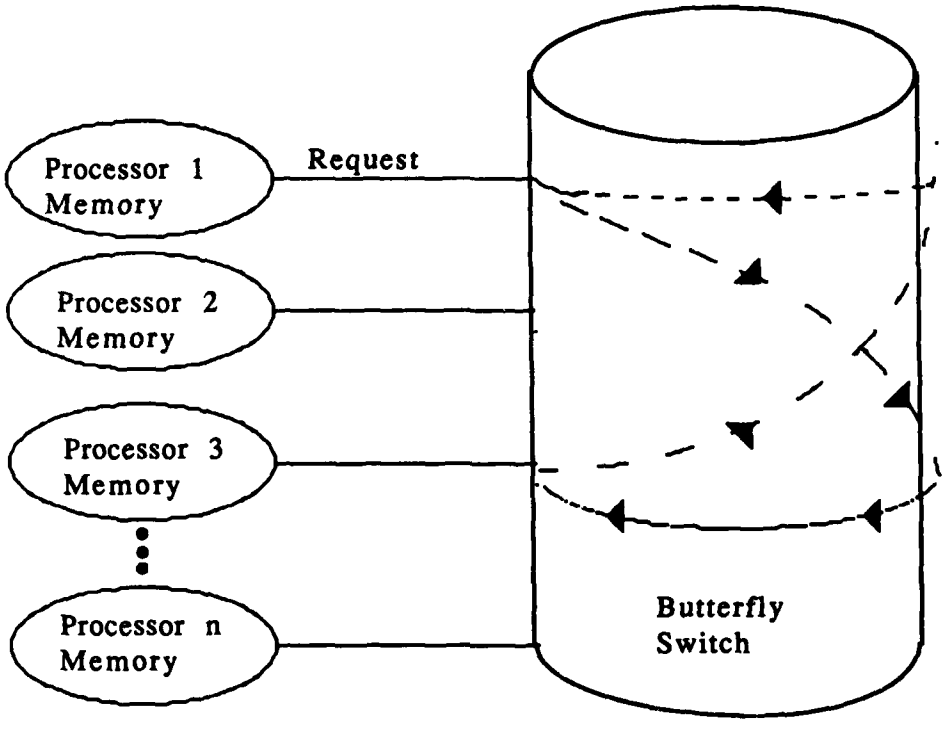


Figure 2.3 The Butterfly Parallel Processor.

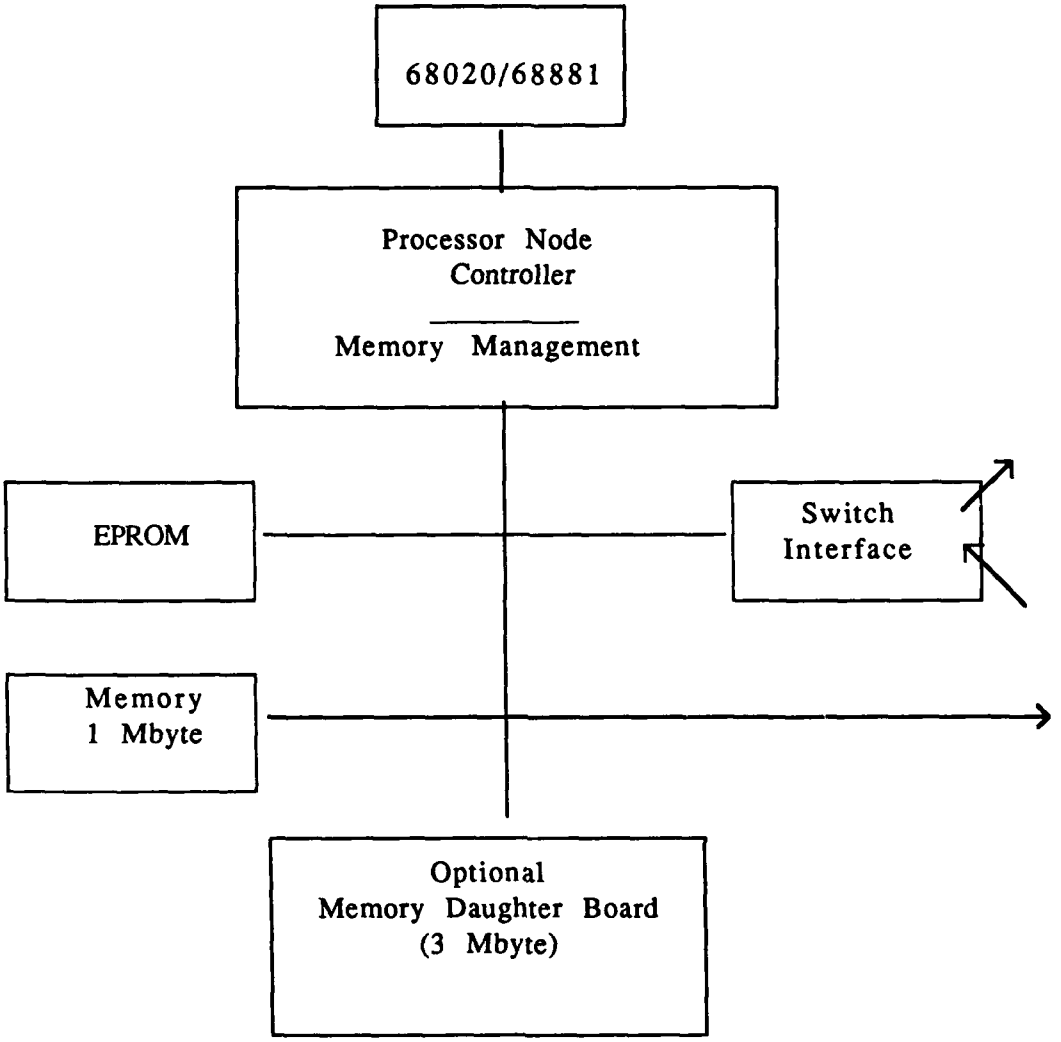


Figure 2.4 A Butterfly Processor-Memory Node.

pair, the time that it takes to execute the instruction on the processor. Figure 2.5 shows sample processor files.

This approach has one serious drawback - it can not take data-dependent (conditional) branches into account in producing its timing estimates. This would have been possible if the Butterfly simulator simulated the actual execution of the input Butterfly programs, which would be slow, costly, and difficult to implement. As an alternative, statistics from the literature on research into branch prediction [DeRosa 87, Lee 84, McFarling 86, Smith 81] were utilized to take conditional branches into account. The approach used in the first version of the Butterfly simulator was as follows: on encountering a given conditional branch for the first time, it would be taken with a probability of 0.5. When the same conditional branch is encountered again in the processing of the program, the simulator assumes that the branch goes the same way as it did the previous time with a probability of 0.9; it goes the opposite way with a probability of 0.1.

The first version of the Butterfly simulator is under development in the programming language C on a SUN 3/50 workstation running 4.2BSD UNIX. The development has proceeded as follows: a timing simulator for a single MC68020 was first developed, and extended into a simulator for multiple 68020s with one task running on each, by early September 1987. This simulator is now being extended to time multiple tasks running on multiple MC68020s - closer to the actual Butterfly environment. At the same time, efforts to refine the timing estimation procedure are underway, as are efforts to condense the processor information files, which currently occupy several Megabytes of disk file space.

The only performance metric that the Butterfly simulator currently measures is total execution time. Accumulation of other metrics, such as MIPS (millions of of instructions executed per second), MFLOPS (millions of floating point instructions executed per second), processor idle time, and network related metrics are also being incorporated.

2.4. Butterfly Network Simulator

Processor File Example:

```
16% clock rate in nanoseconds
** % section delimiter
ari% section with mnemonics for operand modes
ard
arid
. . .
**
move % section with two operand instructions
add
sub
. . .
**
neg % section with one operand instructions
load
. . .
**
nop 4 % section with no operand instruction and times
. . .
**
bcc 10 15 % section with conditional branch instructions and times
. . .
**
bra 10 % section with unconditional branch instructions and times
. . .
**
jsr 10 % section with subroutine call instructions and times
. . .
**
rtr 5 % section with subroutine return instructions and times
. . .
**
frk 20 % section with fork instruction(s) and time(s)
**
snd 8 % section with send instruction(s) and time(s)
**
rcv 3 % section with receive instruction(s) and times
EOF
```

Figure 2.5 Sample Processor File

Instruction Time File Example:

```
move % insruction mnemonic
3,4,5;6,7,8; . . . ; 7,8,9;
. . .
3,4,5;6,7,8; . . . ; 7,8,9;
add
3,4,5;2,4,8; . . . ; 7,8,9;
. . .
23,24,25;36,37,38; . . . ; 57,58,59;
. . .
** % section for one operand instructions
neg
3,4,5;
. . .
5,6,8;
. . .
EOF
```

Input Program File Example:

```
loop:  move ari, ard
add arid, ari
bcc loop
bsr inc
jmp end
inc:  add ard, arid
rtr
end:  nop
EOP
```

Figure 2.5 (Cont'd) Sample Data Files

The objective of this part of the simulator project is to implement a software module in C to simulate the dynamics of the Butterfly network. The module is intended to interface with the node simulator. The two modules interact in a function call manner. At simulation time, the node simulator keeps issuing requests for network services by calling the network simulator. These requests correspond to non-local memory references generated in chronological order by processor nodes of the Butterfly parallel processor. For each request, the network simulator will figure out the response time required by the request through the network by considering latency due to propagation delay and network contention.

The network simulator comprises two major components: a network switching mechanism and a conflict resolution mechanism. The former is used for directing a request for accessing a specific memory through the switches and communication links according to the routing rules of the Butterfly network. The latter is responsible for detecting network contentions where many requests compete for the same switch outputs or communication links, and for arranging them in order through the outputs or communication links in contention. Besides, it will add time penalty to the response time of a deferred request. The two mechanisms are associated with two essential data structures - a switch matrix and a collision matrix - for keeping track of network status and for recording network contentions, respectively. Both are of the form of a 3-dimensional array, reflecting the topology of the Butterfly network.

The implementation of the network simulator is expected to complete by the end of December 1987.

2.5. Code Simulator

The Performance Predictor, as shown in Figure 2.2, is being designed to accept two forms of input: actual Butterfly program files, and synthetically generated program files representative of algorithms in the application domain of interest. The generation of these synthetic traces is the duty of the Code Simulator.

The Code Simulator is still under development. Its operation is based on a set

of input parameters that characterizes the algorithms of interest. Examples of such parameters are granularity, parallelism, and communication/computation ratio. Granularity describes the size of the individual parallel tasks comprising the input program; it will be used by the Code Simulator to determine how large the synthetic tarce files are to be. Degree of parallelism describes the number of parallel tasks; the Code Simulator will use this parameter to determine how many synthetic program files to generate, as well as in task activation. The computation/communication ratio would be used to determine how many computation instructions should be incorporated per communication instruction in the synthectic program files. Other parameters will clearly b eneeded to adequately characterize parallel algorithms; these three examples represent a starting point.

The generation of synthetic program files is to be driven by tables of static and dynamic statistics of typical high level language program contents. These tables have been compiled based on the vast literature on instruction execution frequencies, operand addressing mode frequencies, instruction transition frequencies, etc. [Alexander 75, Brookes 82, DePrycker 82, Ditzel 80, Elshoff 76a, Elshoff 76b, Foster 71, Knuth 71, Tanenbaum 78,Wiecek 82]. Since no statistics are available relating directly to the MC68020 instruction set, these tables were derived based on equivalent features reported for other systems in the papers mentioned above.

CHAPTER 3
MAPPING THE BATTLE MANAGEMENT ALGORITHM TO THE BUTTERFLY PARALLEL PROCESSOR

This chapter is concerned with mapping the Battle Management Algorithm onto a BBN $Butterfly^{TM}$ shared-memory multiprocessor. An efficient mapping method for the algorithm is presented. The proposed method in fact is a general approach to tailoring and fitting a class of numerical and non-numerical algorithms heavily in need of global search and broadcast, into the $Butterfly^{TM}$ Parallel Processor.

It is known that the overall performance of a parallel algorithm on a multiprocessor system depends largely on how well the *communication structure* of a parallel algorithm is matched with the *system interconnection structure*. In a shared-memory environment, there are two major factors that have adverse effects on achieving the match of the two structures. These two factors are: (1) contentions in shared memories, and (2) conflicts in communication links. Performance analysis on a $Butterfly^{TM}$ multiprocessor has been presented in [Crowther 85], [Tomas 86]. They point out if contention problems in shared memories and communication links become dominant, the speedup curve goes to saturation as more processors are added. LcBlanc also examines the effect of memory and switch contention by adding extra memories and extra switches in the system network [Leblanc 86]. He concludes that an implementation based on very efficient communication (e.g., shared memory) may *perform worse* than that based on a less efficient mechanism if such efficiency causes too much communication overhead due to memory and switch contention. Several previous works have been done in reducing the memory contention problems. Worthy of notice are the works done in IBM RP3 and NYU Ultracomputer [Pfister 85], [Lee 86], in which hardware message-combining techniques are used. Since the hardware combining networks are expensive, Yew proposes an effective software combining tree for decreasing memory

contention and preventing tree saturation in the interconnection network [Yew 87].

We propose an algorithm-based method for reducing the above contention problems to the minimum. Unlike previous works, the proposed method does not require hardware augmentation or mediation in communication networks. Contention costs both in shared memories and in communication links are minimized by an efficient mapping method with two different phases of tree-shape communication structures, one for searching and the other for broadcasting. The tree structures allow us to rapidly determine and broadcast a critical data without concern for memory and link contentions.

3.1. Characteristics of The Butterfly Network

The core of the *Butterfly*^{TM} Parallel Processor is a multistage switching network, called the Butterfly Network, through which processor nodes access remote memories in a packet switching manner. Major characteristics of a Butterfly Network with 2^m inputs and 2^m outputs, where m is a positive even number, are enumerated below:

(1) the number of stages = $log_4 2^m = \frac{m}{2}$,

(2) the number of SEs in one stage = $\frac{2^m}{4}$, and

(3) the total number of SEs = $\frac{m}{2} * \frac{2^m}{4} = m * 2^{m-3}$.

If we let $N = 2^m$ represent the number of processors, the switch has the advantages that the total number of SEs needed is $O(N log_4 N)$ and the bandwidth of the network is $O(N)$. Figure 3.1 shows a spacial case of the BBN *Butterfly*^{TM} parallel processor with m = 4.

An m-bit binary representation of the source nodes (processors) and destination nodes (memories) of the Butterfly network can be expressed as follows:

$$a_{m-1} a_{m-2} \ldots a_2 a_1 a_0$$

where m = 4, 6, 8, ... The establishment of a connection from a source(S) to its destination(D) is based on a *self-routing scheme*. That is, to establish a connection from S to D, the binary address of D is used as a *routing tag* to direct the connection. If

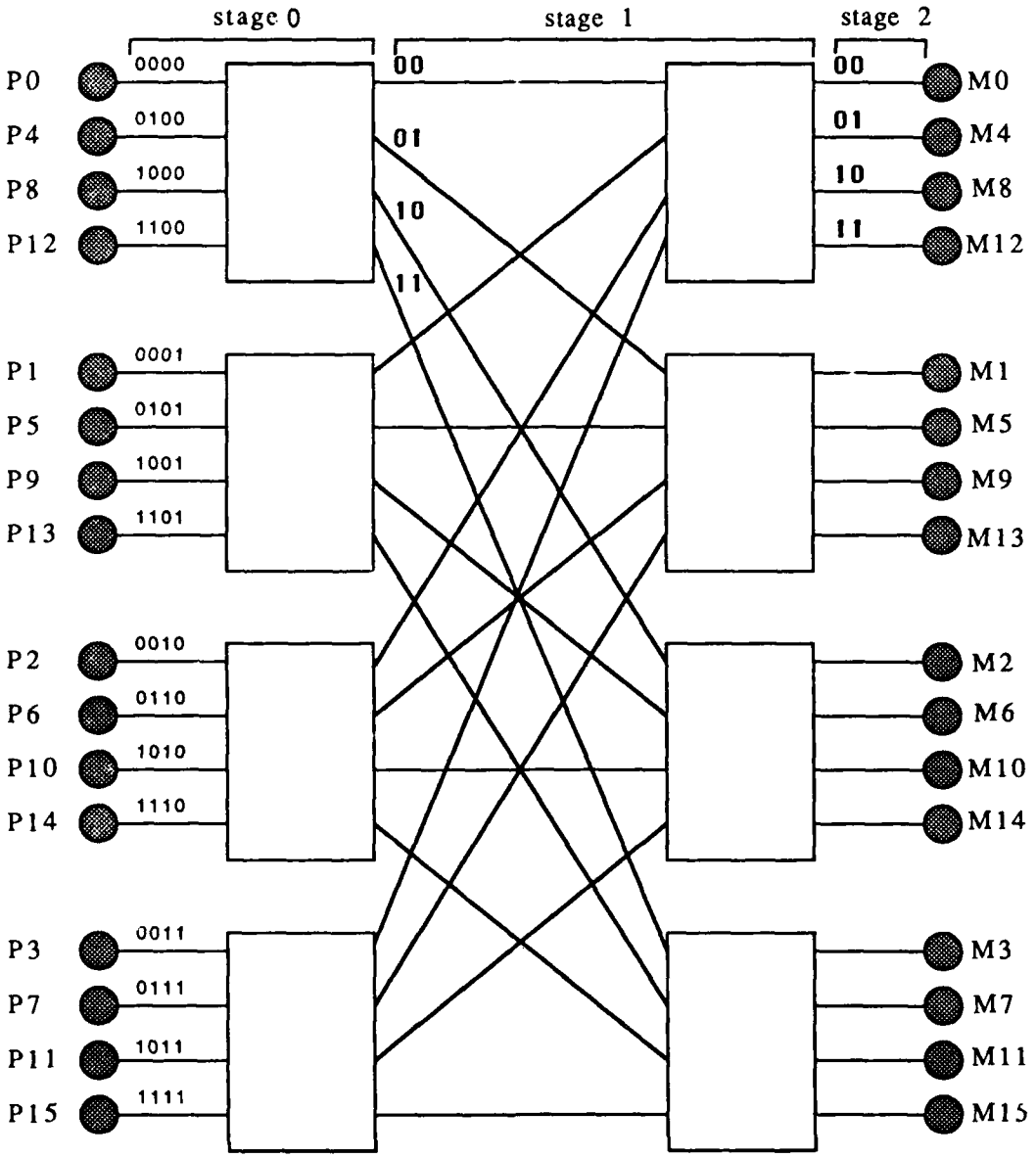


Figure 3.1. A 16x16 Butterfly Parallel Processor with 8 switches.

we let S $s_{m-1}s_{m-2}....s_1s_0$ and D $d_{m-1}d_{m-2}....d_1d_0$, every two binary bits $d_{2i+1}d_{2i}$ is corresponding to the setting of the switching element at stage i, where i=0, 1, 2, ..., $(\frac{m}{2} - 1)$, is the stage number. The communication link traversed by the connection from source to destination at stage i is described as

$$(s_1s_0s_3s_2...s_{m-1-2i}s_{m-2-2i}d_{2i-1}d_{2i-2}...d_3d_2d_1d_0)_i,$$

where $0 \le i \le \frac{m}{2} - 1$.

3.2. Problem Formulation and Algorithm Parallelism

3.2.1 Nature of The Battle Management Algorithm

The Battle Management Algorithm belongs to a class of linear programming problems. These problems are concerned with the optimization of a set of linear functions subject to some linear constraints and to the condition that all the variables must assume nonnegative values. The function to be optimized is called the *objective function.* For example, a general formulation of linear programming problems is as follows: To optimize the objective function

$$C = V_{0,0}X_0 + V_{0,1}X_1 + ... + V_{0,d}X_d + ... + V_{0,r}X_r$$

subject to the linear constraints

$$V_{1,0}X_0 + V_{1,1}X_1 + ... + V_{1,d}X_d + ... + V_{1,r}X_r\{\le, =, \ge\}W_1;$$

$$V_{2,0}X_0 + V_{2,1}X_1 + ... + V_{2,d}X_d + ... + V_{2,r}X_r\{\le, =, \ge\}W_2;$$

.....

$$V_{k,0}X_0 + V_{k,1}X_1 + ... + V_{k,d}X_d + ... + V_{k,r}X_r\{\le, =, \ge\}W_k;$$

.....

$$V_{n-1,0}X_0 + V_{n-1,1}X_1 + ... + V_{n-1,d}X_d + ... + V_{n-1,r}X_r\{\le, =, \ge\}W_{n-1};$$

and to the nonnegative condition $X_0 \ge 0$, $X_1 \ge 0$, ..., $X_r \ge 0$.

A set of values of the variables $X_0, X_1, ..., X_r$ that satisfy the linear constraints and the nonnegative condition is called a *feasible solution.* A *feasible solution* that can optimizes the objective function is called an *optimal feasible solution.* The region

that contains all the feasible solution is called *feasible region*, which is always a convex polygon.

For the problems of optimizing linear functions subject to linear constraints, it has been known that an optimal feasible solution is always at a vertex of the feasible region. Hence, we need to examine the value of the objective function at each vertex of the feasible region. The problem of finding an optimal feasible solution may become a very tedious job, as the number of variables and the number of linear constraints increase. *Simplex method* is one of the most useful methods for finding an optimal feasible solution without examining exhaustively the values of the objective function at all vertices. It is an iterative search procedure. Starting from an artificial candidate solution, it first finds a basic feasible solution, which is represented by a vertex of the convex polygon. From there, it searches for another vertex at which the value of the objective function will be improved. This search is repeated iteratively until the optimal solution is found. Since there are only a finite number of vertices, and the objective function value is improved everytime a new vertex is reached, the search process will eventually converge to the optimal solution.

The theory of searching another vertex at which the value of the objective function is better, and of knowing an optimal solution has been reached are described in detail in [Liu 68]. Here, we only focus on the procedures of the simplex method. The objective function and linear constraints can be rewritten as follows.

$$C - V_{0,0}X_0 - V_{0,1}X_1 - \ldots - V_{0,d}X_d - \ldots - V_{0,r}X_r = W_0;$$

$$X_{r+1} + V_{1,0}X_0 + V_{1,1}X_1 + \ldots + V_{1,d}X_d + \ldots + V_{1,r}X_r = W_1;$$

$$X_{r+2} + V_{2,0}X_0 + V_{2,1}X_1 + \ldots + V_{2,d}X_d + \ldots + V_{2,r}X_r = W_2;$$

.....

$$X_{r+k} + V_{k,0}X_0 + V_{k,1}X_1 + \ldots + V_{k,d}X_d + \ldots + V_{k,r}X_r = W_k;$$

.....

$$X_{r+n-1} + V_{n-1,0}X_0 + V_{n-1,1}X_1 + \ldots + V_{n-1,d}X_d + \ldots + V_{n-1,r}X_r = W_{n-1};$$

where $W_0 = 0$, and X_{r+1}, X_{r+2}, ... X_{r+n-1} are the added slack variables (or basic variables, initially), X_0, X_1, ..., X_r are the non-basic variables. Except the coefficients

of the slack variables, an nx(r + 2) coefficient matrix is generated.

$$C = \begin{pmatrix} -V_{00} & -V_{01} & \cdots & -V_{0d} & \cdots & -V_{0r} & W_0 \\ V_{10} & V_{11} & \cdots & V_{1d} & \cdots & V_{1r} & W_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ V_{k0} & V_{k1} & \cdots & V_{kd} & \cdots & V_{kr} & W_k \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ V_{(n-1)0} & V_{(n-1)1} & \cdots & V_{(n-1)d} & \cdots & V_{(n-1)r} & W_{n-1} \end{pmatrix}$$

For coefficients of the equation of objective function, any negative value will produce one pivot column. For example, if $-V_{0,d} < 0$, then the d-th column is called the *pivot column*. To determine which of the variables $X_{r+1}, X_{r+2}, \ldots, X_{r+n-1}$ will become a non-basic variable, the ratios $\frac{W_1}{V_{1,d}}, \frac{W_2}{V_{2,d}}, \ldots, \frac{W_k}{V_{k,d}}, \ldots, \frac{W_{n-1}}{V_{n-1,d}}$ are computed. Suppose the ratio $\frac{W_k}{V_{k,d}}$ is the smallest of all the positive quantities, then the coefficient $V_{k,d}$ is called the *pivot*. The row that contains the pivot is called the *pivot row*. As soon as the pivot is determined, the operations are continued as follows.

(1) The pivot is replaced by its reciprocal.

(2) The other entries in the pivot row are divided by the pivot.

(3) The other entries in the pivot column are divided by the pivot with their signs reversed.

(4) For the other entries, $V_{i,j}$ $(i \neq k, j \neq d)$ is replaced by $V_{i,j} - V_{k,j} * \frac{V_{i,d}}{V_{k,d}}$. W_i $(i \neq k)$ is replaced by $W_i - W_k * \frac{V_{i,d}}{V_{k,d}}$.

The above operations complete one iteration. The iterations are continued until the coefficients in the expression for the objective function are all positive. At this point, the optimal solution is found.

3.2.2 Task Decomposition

This subsection deals with decomposing the sequential algorithm of the simplex method into several concurrently executed subtasks, then these subtasks are assigned to the processors on $Butterfly^{TM}$ network. In general, there are two correlative factors influencing the decomposition : *granularity* and *interprocessor communication cost*. To achieve a maximum degree of parallelism, we attempt to distribute computations to as many processors as possible -- *fine grain*. However, overhead due to

interprocessor communication drives the tasks allocation strategy to cluster modules to as few processors as possible *large grain*. Obviously, it is not easy to satisfy these two conflicting factors simultaneously; therefore, a compromise must be made to find the optimal arrangement for a task such that the maximum system performance can be achieved.

We illustrate the data flow and process structure of the simplex method in Figure 3.2. In each iteration, a pivot column is arbitrarily selected corresponding to one negative coefficient of the objective equation. The following parallel algorithms consist of four sequential computational phases in one iteration:

Phase 1: To compute all the *ratios* simultaneously by accessing the local data.

Phase 2: To determine *the smallest ratio* and *the pivot*

Phase 3: To modify the data elements on pivot column by accessing the pivot.

Phase 4: To modify the data elements other than those on pivot column.

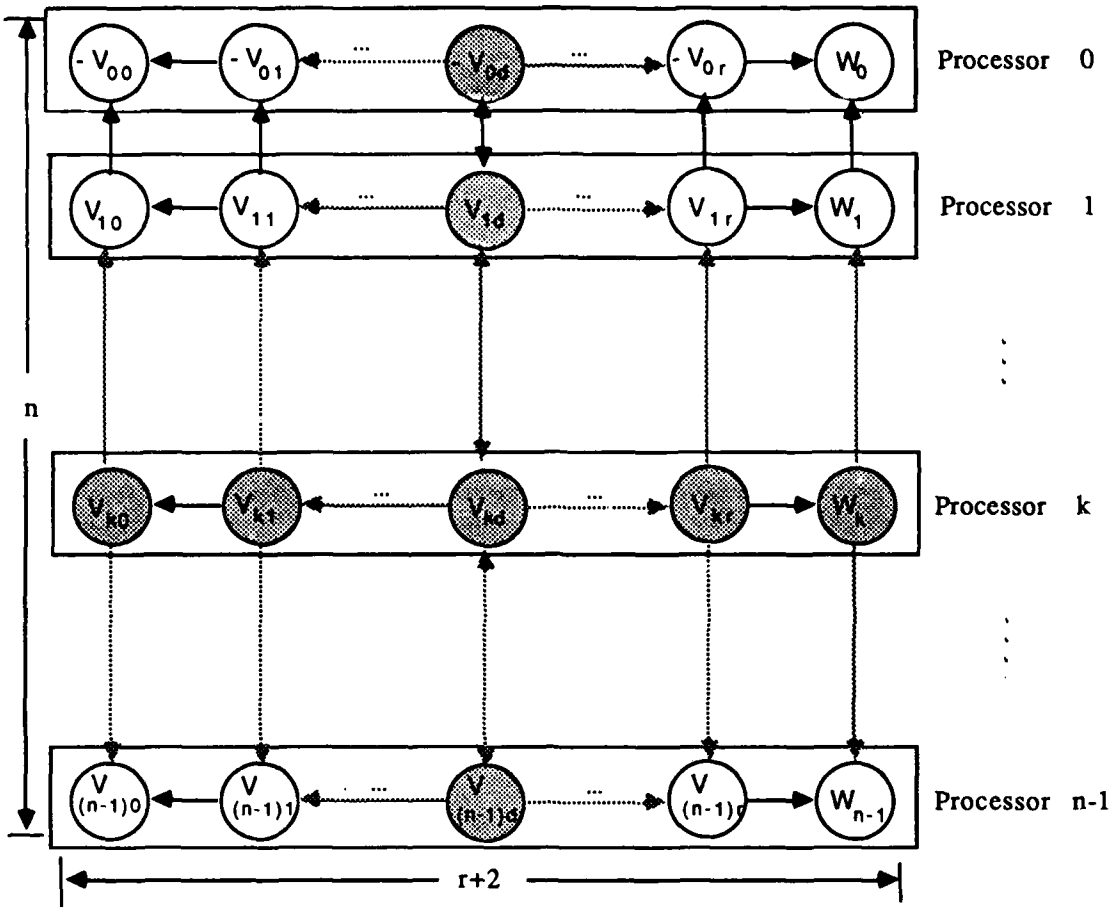
To obtain the optimal balance in the competition between granularity and interprocessor communication cost, we arrange one row elements of the coefficient matrix to be performed by one processor as shown in Figure 3.2. Hence, we are dealing with mapping a problem with nx(r + 2) data elements onto a *Butterfly*^{TM} network with 2^m processors and 2^m shared memories. The relation between the problem size and the network size is

$$2^{m-2} < n \leq 2^m.$$

Before the computation starts, we assume data elements are assigned to processors as the following way: The data elements on row 0, row 1, ..., row (n-1) as shown in the coefficient matrix are assigned to the processors P_0, P_1, .., P_{n-1} on the Butterfly network respectively. That means, the data elements on row 0, row 1, ..., row (n-1) are stored in the memory modules M_0, M_1, ..., M_{n-1} respectively.

3.3. Algorithm Mapping

A significant aspect of parallel algorithms is that in many cases, the model on which they run is not physically realizable directly in present day hardware. Typically,



n : Number of Constraints
r+2 : Number of Variables

Figure 3.2. Process structure and mapping of the simplex method.

for an ideal parallel computer, each processor can access (read from or write into) one memory *in one step*. Simultaneous read or write on a memory by more than one processor may result in competition problems in that memory and in communication links. As far as a better system performance is concerned, hence, it is highly demanded that an efficient mapping for the linear programming algorithms on the $Butterfly^{TM}$ network should be designed to reduce the contention problems both in shared memories and communication links.

Based on the fact that the same arithmetic operations of the linear programming algorithms always reside in the same level, in the following we design the conflict-free connection strategies to prevent from collision of the communication links at the same level.

3.3.1 Conflict-Free Connections

Prior to describing the parallel algorithm and mapping strategies, we first introduce Definitions and Theorems relevant to the algorithm mapping. A routing scheme must be used to set up the connections between processors and shared memories. However, the simultaneous connections may result in conflicts, since the $Butterfly^{TM}$ network belongs to a class of blocking interconnection network.

Definition: A *connection conflict* is defined as a situation in which two connections use the same communication links at some stages at the same time. On the other hand, two connections which do not result in connection conflicts are said to be *conflict-free.*

Definition: Let X and U represent two different n-bit binary numbers, then $\varphi(X,U)$ is the maximum number of consecutively *two identical low-order bits* of X and U.

For example, if we consider m=6, the number of stage=m/2=3, and X=01 10 10, U=10 10 10, then $\varphi(X,U)=2$.

Theorem 3.1: *In a BBN* $Butterfly^{TM}$ *network of size* $N=2^m$ *with the number of stage* $\frac{m}{2}$, *two connections* $X \rightarrow Y$ *and* $U \rightarrow V$ ($X \neq U$ *and* $Y \neq V$) *are conflict-free if and only if*

$$\varphi(X,U) + \varphi(Y,V) < \frac{m}{2}.$$

proof: The communication links traversed by X $\rightarrow$ Y and U $\rightarrow$ V at stage i are described as

$(x_1x_0x_3x_2...x_{m-1-2i}x_{m-2-2i}y_{2i-1}y_{2i-2}...y_3y_2y_1y_0)_i$ and

$(u_1u_0u_3u_2...u_{m-1-2i}u_{m-2-2i}v_{2i-1}v_{2i-2}...v_3v_2v_1v_0)_i$

respectively, where $0 \leq i \leq \frac{m}{2} - 1$.

For sufficient condition: since the two connections X $\rightarrow$ Y and U $\rightarrow$ V are conflict-free, at every stage i,

$(x_1x_0x_3x_2...x_{m-1-2i}x_{m-2-2i}y_{2i-1}y_{2i-2}...y_3y_2y_1y_0)_i$

$\neq (u_1u_0u_3u_2...u_{m-1-2i}u_{m-2-2i}v_{2i-1}v_{2i-2}...v_3v_2v_1v_0)_i$.

This implies

$(x_1x_0x_3x_2...x_{m-1-2i}x_{m-2-2i}) \neq (u_1u_0u_3u_2...u_{m-1-2i}u_{m-2-2i})$ or

$(y_{2i-1}y_{2i-2}...y_3y_2y_1y_0) \neq (v_{2i-1}v_{2i-2}...v_3v_2v_1v_0)$.

Hence, we have $\varphi(X,U) + \varphi(Y,V) < \frac{m}{2}$.

For necessary condition: we assume $\varphi(X,U) = k$ and $\varphi(Y,V) = q$, then $k + q < \frac{m}{2}$.

(1) if $i = 0$ or $\frac{m}{2} - 1$, then

$(x_1x_0x_3x_2...x_{m-1-2i}x_{m-2-2i}y_{2i-1}y_{2i-2}...y_3y_2y_1y_0)_i \neq$

$(u_1u_0u_3u_2...u_{m-1-2i}u_{m-2-2i}v_{2i-1}v_{2i-2}...v_3v_2v_1v_0)_i$ since X $\neq$ U and Y $\neq$ V.

(2) if $1 \leq i \leq q$, then

$(y_{2i-1}y_{2i-2}...y_3y_2y_1y_0) = (v_{2i-1}v_{2i-2}...v_3v_2v_1v_0)$.

Since φ(X,U) = k, and $k < \frac{m}{2} - q \leq \frac{m}{2} - i$, we have

$(x_1x_0x_3x_2...x_{m-1-2i}x_{m-2-2i}) \neq (u_1u_0u_3u_2...u_{m-1-2i}u_{m-2-2i})$.

(3) if $q < i < \frac{m}{2} - 1$, then

$(y_{2i-1}y_{2i-2}...y_3y_2y_1y_0) \neq (v_{2i-1}v_{2i-2}...v_3v_2v_1v_0)$.

Consequently, at every stage i,

$(x_1x_0x_3x_2...x_{m-1-2i}x_{m-2-2i}y_{2i-1}y_{2i-2}...y_3y_2y_1y_0)_i$

$\neq (u_1u_0u_3u_2...u_{m-1-2i}u_{m-2-2i}v_{2i-1}v_{2i-2}...v_3v_2v_1v_0)_i$

Hence, X $\rightarrow$ Y and U $\rightarrow$ V are conflict-free. □

The operation of sending messages from processors to memories (or vice versa) is

called a *permutation routing*. Shortly, we relax that the processor-to-memory assignment is a *permutation*.

Definition: A *permutation* of *bit reversal* is defined as that a binary representation is reversed in every one bit unit. For example, if we define the permutation function of bit reversal to be ρ, an m-bit binary number $A = a_{m-1}a_{m-2}a_{m-3}....a_2a_1a_0$, can be transformed as the following way, $\rho(A) = a_0a_1a_2....a_{m-3}a_{m-2}a_{m-1}$.

Theorem 3.2: *The permutation of bit reversal ensures that the communication links are conflict-free, when it is used to communicate all the source nodes to their respective destination nodes.*

Proof: Assume S_i and S_j are the binary representation of any two source nodes,

$$S_i = a_{m-1}a_{m-2}a_{m-3}....a_2a_1a_0$$

$$S_j = b_{m-1}b_{m-2}b_{m-3}....b_2b_1b_0$$

The permutation of bit reversal completes two connections: $S_i \rightarrow D_i$ and $S_j \rightarrow D_j$, we have

$$D_i = \rho(S_i) = a_0a_1a_2....a_{m-3}a_{m-2}a_{m-1}$$

$$D_j = \rho(S_j) = b_0b_1b_2....b_{m-3}b_{m-2}b_{m-1}$$

Since $S_i \neq S_j$, if we assume $\varphi(S_i, S_j) = \omega$, then $0 \leq \omega \leq \frac{m}{2} - 1$. This allows,

(1) $\varphi(D_i, D_j) = 0$, if $\omega \neq 0$ (Note that $max\{\omega\} = \frac{m}{2} - 1$), or

(2) $max\{\varphi(D_i, D_j)\} = \frac{m}{2} - 1$, if $\omega = 0$

In either case, $\varphi(S_i, S_j) + \varphi(D_i, D_j) < \frac{m}{2}$. Based on Theorem 3.1, any of the two connections are conflict-free. □

We now describe the *conflict-free connection strategies* as follows. In the beginning, the 2^m processor space is divided into two subspaces, source-1 and destination-1, each space has 2^{m-1} processors. For the processors in the source-1 space, the most significant bit of binary representation is zero. On the other hand, the processors in the destination-1 space have the most significant bit one. The connections between the source-1 and destination-1 space are established by performing the permutation of bit reversal on the remaining $m - 1$ binary bits of the processors in the source-1 space.

Secondly, the processors in destination-1 space are now divided into another two subspaces, source-2 and destination-2, each has 2^{m-2} processors. For the processors in the source-2 space, the second significant bit is zero, and the second significant bit of the processors in the destination-2 space is one. Similarly, the connections between the space of source-2 and destination-2 are established by performing the permutation of bit reversal on the remaining $m - 2$ binary bits of the processors in the source-2 space. The similar connection strategies are continued until the number of processors in the destination-m space is equal to one. Since the permutation of bit reversal ensures a one-to-one connection, the connections established as above also preserve the property of one-to-one permutation.

The above connection strategies can be formulated. First we define a permutation ρ^h which performs partial bit-reduction and partial bit-reversal. For example, if

$$A = a_{m-1}a_{m-2}...a_{m-h}a_{m-h-1}...a_2a_1a_0, \text{ then}$$

$$\rho^h(A) = a_0a_1a_2...a_{m-h-1}.$$

Suppose we let $S_i^h \rightarrow D_i^h$ and $S_j^h \rightarrow D_j^h$ represent any two connections from source-h space to destination-h space. The m-bit binary representations of S_i^h, S_j^h, D_i^h and D_j^h can be expressed as follows, note $h = 1, 2, 3, ..., m$.

$$S_i^h = \overbrace{11...1}^{h-1}0a_{m-h-1}...a_2a_1a_0;$$

$$S_j^h = \overbrace{11...1}^{h-1}0b_{m-h-1}...b_2b_1b_0;$$

$$D_i^h = \overbrace{11...11}^{h}\rho^h(S_i^h) = \overbrace{11...11}^{h}a_0a_1a_2...a_{m-h-1};$$

$$D_j^h = \overbrace{11...11}^{h}\rho^h(S_j^h) = \overbrace{11...11}^{h}b_0b_1b_2...b_{m-h-1}.$$

If $h = m$, only one connection is established. The m-bit binary representation of S_i^h and D_i^h become $S_i^m = 1111...10$ and $D_i^m = 1111...11$.

Theorem 3.3: *For every h, h = 1 to m, the connections $S^h \rightarrow D^h$ are conflict-free.*

proof: If we let $A^h = a_{m-h-1}...a_2a_1a_0$ and $B^h = b_{m-h-1}...b_2b_1b_0$, then we have $\rho^h(S_i^h) = \rho(A^h)$ and $\rho^h(S_j^h) = \rho(B^h)$. Based on Theorem 3.1 and 3.2, since the permutation of bit reversal ensures that the connections are conflict-free for $h = 1, 2, ..., m$, we have

$$\varphi(A^h, B^h) + \varphi(\rho(A^h), \rho(B^h)) < \frac{m}{2}.$$

Also, because of $S_i^h \neq S_j^h$ and $D_i^h \neq D_j^h$, the following two equations are valid,

(1) $\varphi(S_i^h, S_j^h) = \varphi(A^h, B^h)$, and

(2) $\varphi(D_i^h, D_j^h) = \varphi(\rho(A^h), \rho(B^h))$.

Hence, we have $\varphi(S_i^h, S_j^h) + \varphi(D_i^h, D_j^h) < \frac{m}{2}$. According to the necessary condition of Theorem 3.1, we know for every h, the connections established as above are all conflict-free. □

3.3.2 Reduction of Memory Contentions

In shared-memory environment, memory contention problems frequently incur extra execution time and consequently decay the system performance. Hence, the contention in shared memories needs to be reduced *in addition to minimizing* the conflict in communication links. We propose the tree-shape communication structure which can reduce memory contention problems from $O(n)$ to $O(log_2(n))$ (where n is the problem size). The set up of a tree-shape communication structure for searching is described as follows.

A *deposit-access mode*, which requires only *one-path* communication cost, is used for processors to access the shared memories in the communication structure. For example, if $P_i(P_k)$ is connected to the local memory of $P_j(P_l)$ by the conflict-free connection strategies, the *deposit-access mode* means $P_i(P_k)$ will fetch the compared result from its local memory and store it, through the interconnection network, to the local memory of $P_j(P_l)$. $P_j(P_l)$ performs an arithmetic operation to compare the deposited result by $P_i(P_k)$ with its own result both stored in the local memory of $P_j(P_l)$. After the comparison operation, P_j in turn deposits the compared result, through the interconnection network, to the local memory of P_l with the same procedure. Figure

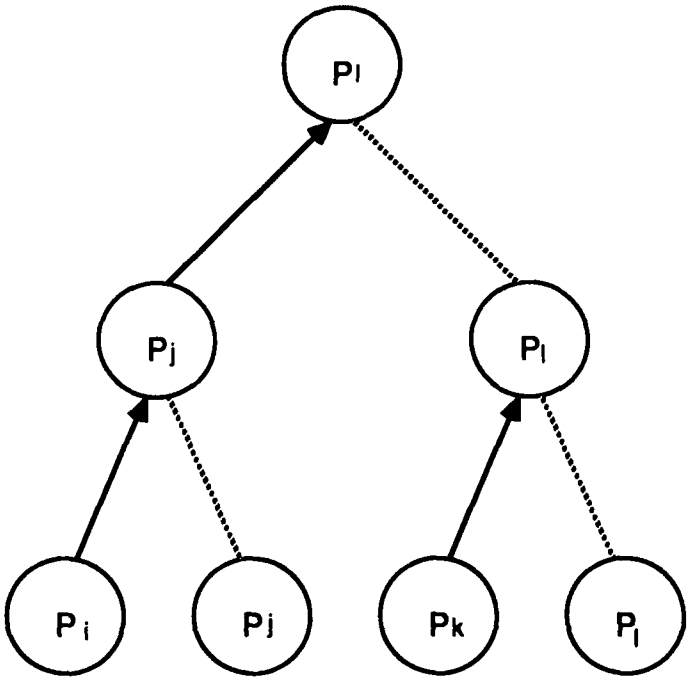


Figure 3.3. A tree-shape communication structure with deposit-access mode.

3.3 shows the tree-shape communication structure built with P_i, P_j, P_k, and P_l. In this case, P_i and P_j (P_k and P_l) are called *pair node*, and P_i (P_k) is referred as left child, P_j (P_l) as right child. In our design, the right child will become the father in next level.

Definition: The number of *level* of a tree structure is defined as the distance from the farthest leave node to the root. Hence, for a tree structure with 2^m nodes, the number of levels equals m. We let $h = 1, 2, 3, ..., m$ represent each level of the tree structure.

The subsequent following mapping algorithms serve the purpose of setting up a tree-shape communication structure on the $Butterfly^{TM}$ network. Meanwhile, at each level, the conflict in communication links is avoided and the contention in shared memories is reduced.

Algorithm 3.1: To compute all the ratios concurrently for a picked pivot column.

FOR $i = 0$ TO $n - 1$ (all processors execute in parallel)

(1) P_i reads data $V_{i,d}$ and W_i from its local memory;

(2) P_i computes the ratio = $\frac{V_{i,d}}{W_i}$;

END; □

The binary representation of processor P_i ($i = 0$ to $n - 1$) is assumed to be $a_{m-1}a_{m-2}...a_1a_0$. For a tree level h, bit a_{m-h} ($h = 1, 2, ..., m$) is an indicator to divide the processor space into two subspaces, source and destination.

(1) If $a_{m-h} = 0$, then P_i is in the source subspace.

(2) If $a_{m-h} = 1$, then the local memory of P_i (that is, M_i) is in the destination subspace.

The source node P_i then connects to its destination by performing the permutation of bit reversal on its remaining binary digits $a_{m-h-1}a_{m-h-2}...a_1a_0$.

Algorithm 3.2: To search the smallest ratio and the pivot, we build the *searching tree* structure from the bottom level ($h = 1$) to the top level ($h = m$).

Step 1: *For level $h = 1$ on the tree-shape structure*

If $a_{m-1} = 0$, then P_i connects to its destination by performing the permutation of bit reversal on its remaining binary digits $a_{m-2}a_{m-3}...a_1a_0$.

Step 2: *For level $h = 2$ on the tree-shape structure*

The destination nodes in level $h = 1$ now are divided into two subspaces. If $a_{m-2} = 0$, then P_i connects to its destination by performing the permutation of bit reversal on its remaining binary digits $a_{m-3}a_{m-4}...a_1a_0$.

Step 3: Repeat the same procedures until reach level $h = m$, the tree-shape communication structure can be set up.

Step 4: The smallest ratio is determined from the root processor, and the pivot is found from the numerator of the smallest ratio. The pivot row can also be decided from the pivot.

Based on Theorem 3.3, the searching tree can be mapped onto the *Butterfly*^{TM} network without any conflict in communication links for every level h, and the contention in shared memories is reduced to $O(log_2 n)$. □

An example of the communication structure of a searching tree shown in Figure 3.4 (the communication direction is indicated by upward arrow) is built with 16 processor nodes (m = 4 and n = 16). Once the tree-shape communication structure has been set up(also, the pivot has been determined), the second computational phase is concerned with broadcasting the pivot from the root processor to all other memory modules such that directly access for other processors becomes possible.

Definition: A *memory replication technique* is a technique to duplicate a critical data in as many memory locations as needed by using the *deposit-access mode*.

The same tree-shape communication structure built in Algorithm 3.2 is used to perform the *memory replication* just reverse the access procedure. The direction of communication is indicated by the downward arrow as shown in Figure 3.4. This tree structure is referred as a *broadcasting tree*.

Theorem 3.4: *A broadcasting tree structure can be mapped onto the Butterfly*^{TM} *network with conflict-free connections at each level, on the condition that the con-*

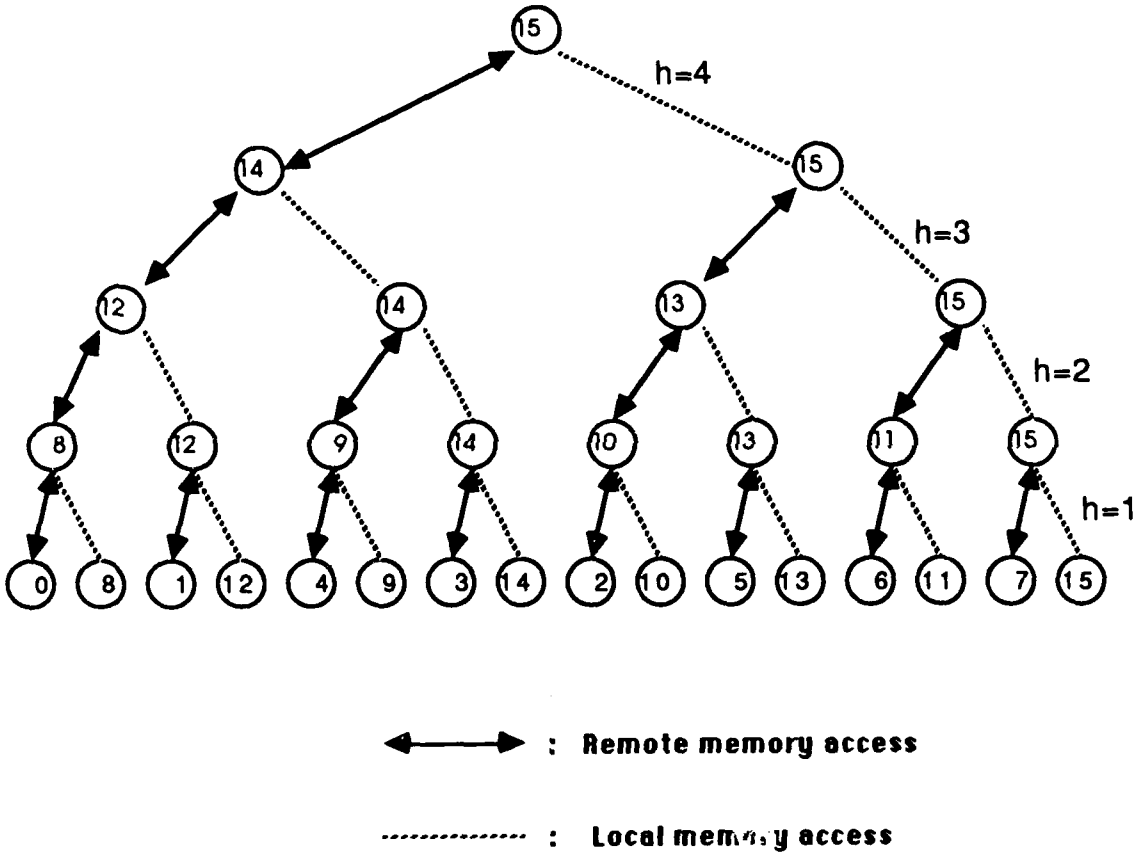


Figure 3.4. Communication structure of message search and broadcast through the Butterfly network.

nections in a searching tree structure are conflict-free at the same level on the same network.

Proof: Assume any two connections $X \rightarrow Y$ and $U \rightarrow V$ belong to the connections in the searching tree structure at level h. Since they are conflict-free, according to the sufficient condition of Theorem 3.1, we have $\varphi(X,U) - \varphi(Y,V) < \frac{m}{2}$. Now, these two connections in a broadcasting tree structure become $Y \rightarrow X$ and $V \rightarrow U$. Since the inequality $\varphi(Y,V) - \varphi(X,U) < \frac{m}{2}$ is also true, based on the necessary condition of Theorem 3.1, the connections in a broadcasting tree at level h are all conflict-free. □

Algorithm 3.3: *Memory replication technique* is used to broadcast the pivot from the root processor to all other processors' local memories. Then the data elements on the pivot column are modified simultaneously. The *pivot processor* is defined as one, in which its stored element $V_{i,d}$ is equal to the pivot. For example, if pivot $= V_{k,d}$, then P_k is the pivot processor.

Step 1: Memory replication

The broadcasting tree structure is used to replicate the pivot from the root processor to all other processors' local memories. Based on Theorem 3.4, the memory replication can be accomplished with the minimization of contention in shared memories and with the conflict-free in communication links at each level.

Step 2: Now, every processor receives the pivot ($V_{k,d}$).

FOR $i = 0$ TO $n - 1$ (all processors execute in parallel)

IF $i = k$ THEN P_i performs the following operation

$V_{i,d} = \frac{1}{V_{k,d}}$ (modify the pivot);

ELSE P_i performs

$V_{i,d} = \frac{V_{i,d}}{V_{k,d}}$ (modify other data elements);

END;

Step 3: Pivot processor P_k sends its local data $V_{k,j}$, j=0 to r, and W_k to the root processor. With the data flow similar to a wave, the root processor broadcasts these data to other processors' local memories by using the broadcasting tree

communication structure. □

Algorithm 3.4: To simultaneously modify the data elements other than those on the pivot column.

FOR $i = 0$ TO $n - 1$ (all processors execute in parallel)

IF $i = k$ THEN P_i performs $\frac{V_{k,j}}{V_{k,d}}$ ($j = 0$ to r, and $j \neq d$) and $\frac{W_k}{V_{k,d}}$

ELSE P_i performs

$V_{i,j} = V_{i,j} - V_{k,j} * \frac{V_{i,d}}{V_{k,d}}$ ($j = 0$ to r, and $j \neq d$); and

$W_i = W_i - W_k * \frac{V_{i,d}}{V_{k,d}}$;

END. □

As soon as the first iteration is completed, the second pivot column is selected according to the next negative coefficient of the objective equation. This begins the second iteration with the same procedures from Algorithm 3.1 to Algorithm 3.4. The number of iteration is equal to the number of negative coefficient of the objective equation.

3.4. Performance Evaluation

It is widely known that performance analysis of an iterative algorithm on the MIMD multiprocessor is a very complex and difficult job, since many factors jointly determine algorithm performance and the modification of a certain factor may affect others. For simplicity, we make a few assumptions in an attempt to approximately predict the system performance by complexity analysis. Note that the real system performance should be better than the following analysis, since we consider the worst case. It is assumed that execution of identical arithmetic operations on different processor nodes requires the same response time. In the following discussion, α denotes the time for completing one multiplication, β denotes the time for completing one division, η for completing one addition, σ for completing one logic comparison operation, and μ for completing one remote memory access with deposit mode. We also assume that the time required for a local memory access is so small that it can be neglected. Let $T(i)$ represent the execution time for algorithm i in one iteration, T_{para} represent

the execution time of the parallel algorithms on the $Butterfly^{TM}$ multiprocessor in one iteration, and T_{unit} represent the execution time of the sequential algorithm on unitprocessor in one iteration. we derive the following three inequalities:

$$T_{para} \leq \sum_{i=1}^{4} T(i) =$$
$$\beta + ((\mu + \sigma)log_2 n) + (\mu(log_2 n) + \beta + \mu(r+2)(1 + log_2 n)) + (r+1)(\alpha + \eta); \quad (3.1)$$

$$T_{unit} \geq n\beta + (n-1)\sigma + n\beta + (r+2)\beta + n(r+2)(\alpha + \eta); \quad (3.2)$$

$$Speedup = \frac{\sum_{k=1}^{r+1} T_{unit}}{\sum_{k=1}^{r+1} T_{para}} >$$

$$\frac{\sum_{k=1}^{r+1} (n\beta + (n-1)\sigma + n\beta + (r+2)\beta + n(r+2)(\alpha + \eta))}{\sum_{k=1}^{r+1} (\beta + ((\mu + \sigma)log_2 n) + (\beta + \mu(r+2) + \mu(r+3)log_2 n) + (r+1)(\alpha + \eta))}; \quad (3.3)$$

where the maximum number of iterations is equal to $r + 1$. If we assume the iterative parallel algorithm is homogeneous, then the time for completing each iteration is almost the same. When r approaches to n, we have

$$speedup \geq \frac{O(n^2)}{O(nlog_2 n)}.$$

This expression indicates that the speedup is a first-order increasing function of the problem size n as n becomes a reasonable large number. The result also verifies our claims that the parallel algorithms can achieve a higher system performance by taking advantages of the mapping method.

CHAPTER 4
EVALUATION OF EXISTING DEPENDABILITY TOOLS

It was pointed out in the introduction that there exists no tool that can be used for the exact dependability evaluation directly. By exact, we mean a Markovian or Semi Markovian approach to capture the component failure and repair processes accurately. Although there has been some attempt to model a MIN-based multiprocessor using Markovian technique [Arlat 83, Blake 87], these models are very simple and are also not easily extendable to large systems. In the absence of an exact analytical model, an approximate model is preferred if it can give acceptable results. Moreover, in the process of developing an approximate model one can get better insight to go for the exact technique. In sections 5 and 6, we present approximate techniques for the dependability evaluation of Butterfly and hypercube systems. In this section we first present a brief summary of some of the existing tools with highlighting their limitations for the dependability evaluation of candidate parallel computers.

There are a number of existing tools available for computing dependability of redundant systems. Tools such as ARIES, CARE III, HARP, and SHARPE can be used for reliability analysis where as tools such as HARP, SHARPE and SAVE can be used for both reliability and availability analysis. SHARPE, on the other hand, can be used for performability evaluation. In this section a brief summary of CARE III, HARP, and SHARPE is given. The conclusions regarding the applicability of these models to candidate architectures also apply to other tools not summarized here.

4.1 CARE III

CARE III (Computer Aided Reliability Estimation, Version Three) is a program designed to estimate reliability of complex redundant systems [Stiffler 82]. It was

developed specifically for fault-tolerant avionics systems, CARE III features are summarized below.

Capabilities

Predict the unreliability (1-reliability) of a system consisting of up to 70 stages with each stage composed of one or more identical modules.

Can handle hardware/software faults of various types such as permanent, transient and intermittent.

User must specify the number of modules in each stage, the minimum number of modules needed in each stage for the system to operate properly, the various combination of stage failures that constitute a system failure and the probability that a specific module from stage i forms a critical pair(system failure) with a specific modules from stage j. Hence, a system tree specification involving the critical pairs must be given as input to the program. The lower level faults in the fault tree specification are stage failures.

Modules imperfect fault handling (coverage) using Markovian technique.

Fault distribution is given by a Weibull function.

Disadvantages

Can not model availability.

Fault tree in terms of critical pairs of a MIN-based system or hypercube is very difficult. The number of each critical failure combination can be too large to specify for a medium or large size system. Particularly various combination of switch failures that can lead to system failure in a Butterfly type system is extremely difficult to specify.

Can not model performance-related dependability.

4.2 HARP

HARP (Hybrid automated Reliability Predictor) [Bavuso 87, Geist 83] is a software package that implements dependability modeling techniques. Its advantages and disadvantages are given below.

Capabilities

Can compute both reliability and transient availability of computer systems using behavioral decomposition along temporal lines. The overall model is decomposed into fault-occurrence/repair (FORM) and fault/error handling (FEHM) submodules to analyze the fault-occurrence and coverage effects effectively.

Can handle various types of faults as described in CARE III.

User must input either the Markov chain of the system or a Petri-net model, which can be converted to Markov chain automatically for computing dependability. The other alternative input can be a fault tree specification of the system.

Can model systems with sequence dependant failures.

Gives guaranteed bounds on reliability.

Weibull distribution for reliability modeling.

Disadvantages

Cannot compute MTTF or steady state behavior for repairable systems.

Cannot guarantee the Markov chain automatically. As has been pointed out earlier, generation of the Markov chain is complex for systems like Butterfly or Hypercube. Also, a fault-free specification of the candidate parallel system is not simple. Hence, the difficulty of finding the input model restricts the usefulness of HARP to parallel architectures under consideration.

4.3 SHARPE

SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) is currently under development at Duke University [Shaner 87]. In addition to dependability evaluation, it has the capability to include performance with dependability, such as performability. It's advantage are the following.

Capabilities

Supports seven model types such as reliability block diagram, fault tree without repeated nodes, acyclic Markov chains and irreducible cyclic Markov chains to be combined hierarchically in a flexible manner.

Allows to use either combinatorial or Markov/Semi-Markov submodules.

Uses Symbolic computation.

Input to the model is in the format of reliability block diagram, fault-tree, or Markov chain.

Disadvantages

As like HARP, construction of the fault-tree or Markov chain is again the challenging problem. Development of a reliability block diagram is also not simple to model task based evaluation.

CHAPTER 5
BUTTERFLY DEPENDABILITY MODELING

We have developed a preliminary analytical model for computing the reliability of Butterfly network based multiprocessors [Tien 88]. The model is preliminary in the sense that the actual Butterfly node failure/repair behavior is not included in the model to address the coverage accurately. Also, it does not address the details of a Butterfly system such as extra stage of switches, software failures, and system sizes which are not powers of four. It captures graceful degradation of a Butterfly system by considering the failure of processors, memories, and 4x4 switching elements (SES) that constitute the Butterfly network.

The modeling approach is based on system decomposition. Since, the Butterfly system uses 4x4 switches, the system size is generally given by 4^i. Although systems available today can be configured with any number of nodes n, $n \leq 256$, the configurations that are not powers of 4 do not use all the $N/4log_4N$ switches used for the interconnection network. Hence, this model addresses 4^i systems first. Extension of the model to other configuration when $N \neq 4^i$ is under investigation now. The modeling approach is based on system decomposition and combinational techniques. The reliability of a 4^i system is obtained from four 4^{i-1} subsystems and the connection pattern between those subsystems. The reliability model assumes a homogenous multiprocessor system. The PEs, MMs, and SEs are homogenous and have identical exponential failure distributions. We define λ_p, λ_m, and λ_s as the failure rate of a PE, MM, and SE, respectively. The corresponding component reliabilities are given by $R_p(t) = e^{-\lambda_p t}$, $R_m(t) = e^{-\lambda_m t}$, and $R_{se}(t) = e^{-\lambda_s t}$. Task based reliability is computed by looking for a connected system with at least I processors and J memories

5.1 16x16 System Reliability

A 16x16 multiprocessor with 8 switches is shown in Figure 5.1. We will call it a 16 node system since a PE_i and a MM_i are assembled on a single board. The 16 node system can be decomposed into four 4x4 subsystems while keeping the communication between the subsystems undisturbed. A subsystem with 4 processors, 4 memories, and 2 switches is shown in Figure 5.2. While a 4x4 configuration requires only one SE, Figure 5.2 uses 2 SEs. One switch connects the 4 PEs, and the second switch connects the 4 memories. We call these the input and output switches respectively. A system with more than 4 PEs and 4 MMs needs at least one input SE and one output SE to establish the connection.

5.1.1 **4x4 Analysis**

We first compute the probability of having exactly i PEs and j MMs connected at time t, for $i, j \leq 4$, in Figure 5.2. This is given by

$$P_{4(i,j)}(t) = I_{\{i=0 \wedge j=0\}}(1 - R_{se}^2(t))$$

$$+ \binom{4}{i} R_p^i(t)(1 - R_p(t))^{4-i} \binom{4}{j} R_m^j(t)(1 - R_m(t))^{4-j} R_{se}^2(t) \tag{5.1}$$

The subscript $4(i,j)$ stands for selecting i PEs and j MMs from a 4x4 or 4 node system. $I_{\{i=0 \wedge j=0\}}$ is an indicator function given by

$$I_{\{i=0 \wedge j=0\}} = \begin{cases} 1 & if \;\; i = 0 \wedge j = 0 \\ 0 & if \;\; i \neq 0 \vee j \neq 0 \end{cases} \tag{5.2}$$

The first term in equation (5.1) represents the situation where any number of processors and memories are working when at least one SE has failed. This term contributes to the $P_{4(0,0)}$ probability. The second term in equation (5.1) denotes the connection of i working processors with j working memories when both the SEs are fault free. For example, the probability of 2 processors connected to 3 memories is given by

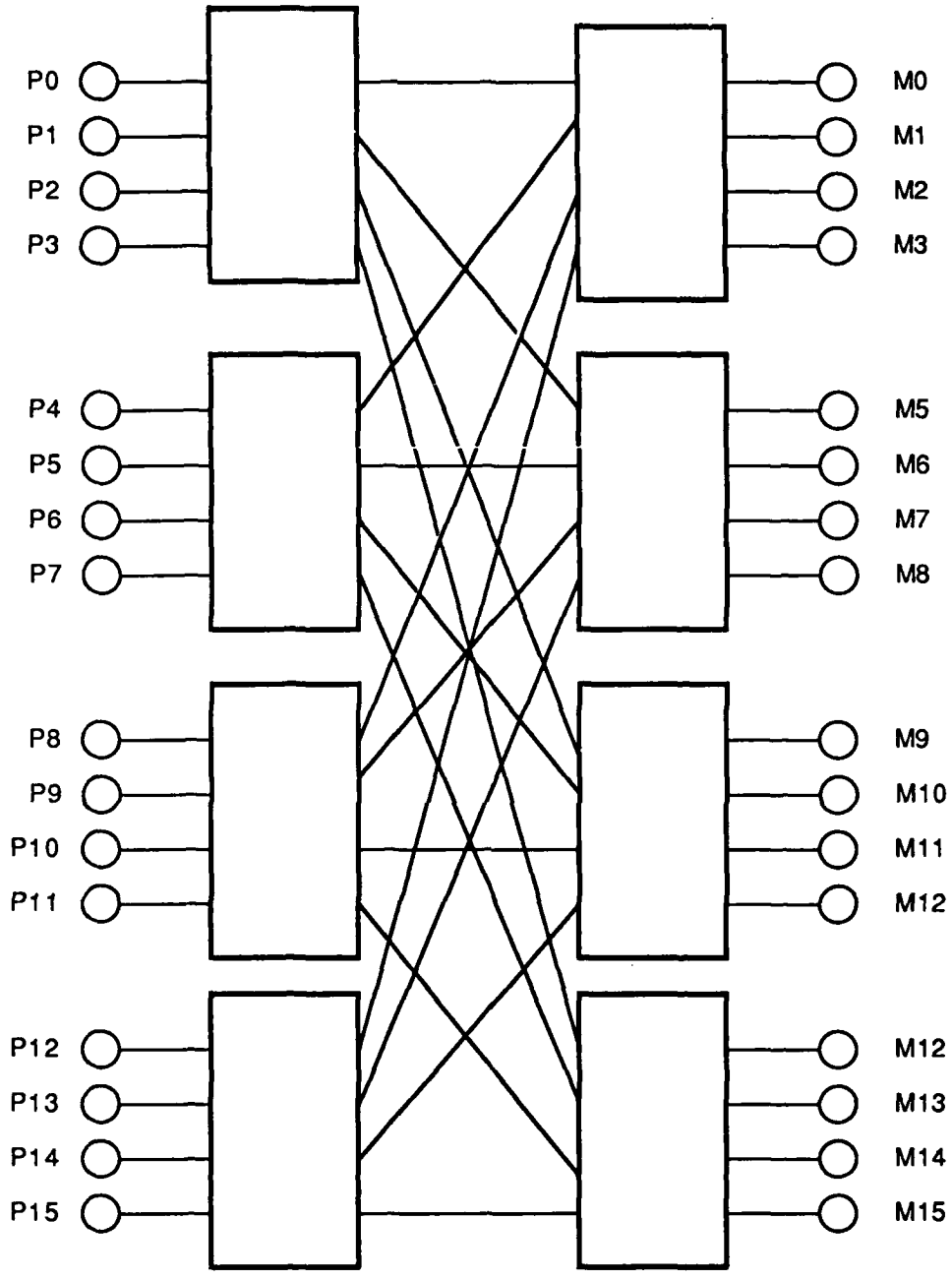


Fig 5.1 A 16x16 multiprocessor with 8 switches.

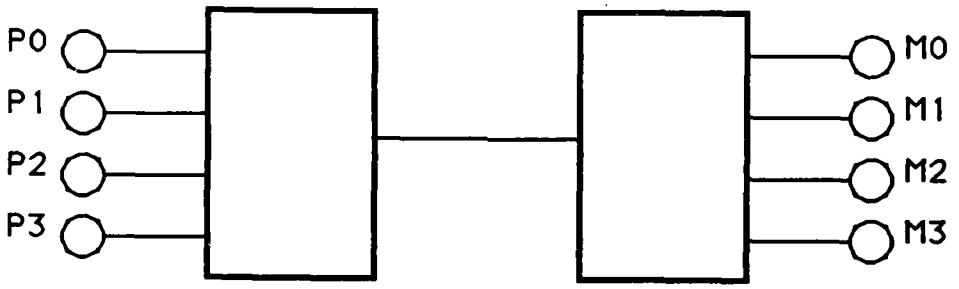


Figure 5.2 A 4x4 Multiprocessor with 2 switches.

$$P_{4(2,3)}(t) = \binom{4}{2} R_p^2(t)(1 - R_p(t))^2 \binom{4}{3} R_m^3(t)(1 - R_m(t)) R_{se}^2(t)$$

It should be observed that $R_{se}^2(t)$ is included for terms like $P_{4(0,j)}(t)$ or $P_{4(i,0)}(t)$. This is because the input and output SEs are utilized for any reference to be satisfied.

5.1.2 16x16 Analysis

The reliability of a 16x16 system is derived from the basic 4x4 model. Since the 16 processors and 16 memories can be divided into four groups each with their associated SEs, we need to distribute the required number of components among 4 groups. For example, all possible distributions of i PEs within four processor groups and j MMs within four memory groups must be considered. The probability of selecting i PEs and j MMs (ixj) from a 16x16 system at time t is given by

$$P_{16(i,j)}(t) = \sum_{i_0=0}^{k_0} \sum_{i_1=0}^{k_1} \sum_{i_2=0}^{k_2} \sum_{j_0=0}^{l_0} \sum_{j_1=0}^{l_1} \sum_{j_2=0}^{l_2} P_{4(i_0,j_0)}(t) P_{4(i_1,j_1)}(t) P_{4(i_2,j_2)}(t) P_{4(i_3,j_3)}(t) \tag{5.3}$$

where

$$k_0 = min(4, i)$$
$$k_1 = min(4, i - i_0)$$
$$k_2 = min(4, i - i_0 - i_1)$$
$$k_3 = i - (i_0 + i_1 + i_2)$$
$$l_0 = min(4, j)$$
$$l_1 = min(4, j - j_0)$$
$$l_2 = min(4, j - j_0 - j_1)$$
$$l_3 = j - (j_0 + j_1 + j_2)$$

The distribution of i PEs among 4 groups is such that $i_0 + i_1 + i_2 + i_3 = i$ and is controlled by the last term i_3. Since $\binom{4}{i_3} = 0$ for $i_3 > 4$, all possible valid distributions of i PEs among four processors groups are generated by the first three summation expressions. The k_is control the maximum number of PEs to 4 in a group. The last

three summation terms with the corresponding l_is, generate all the distribution j MMs among four groups.

Any valid distribution of the processors and memories are combined into four processor memory pairs. The $P_4(i_x, j_y)$ $(0 \leq x, y \leq 3)$ is the same as in equation (5.1). It should be observed that by including the input and output SEs in a 4x4 group, we guarantee connection among the i PEs and j MMs from the four groups. For example, a $P_4(i_x, o)$ group can access a $P_4(0, j_y)$ group as the required four SE reliabilities are included in the expression.

The reliability of a 16x16 multiprocessor with at least I PEs and J MMs working connected is then given by

$$R_{s(N,N)}(t) = \sum_{i=I}^{N} \sum_{j=J}^{N} P_{N(i,j)}(t) \tag{5.4}$$

where N is the size of the system. In this case N = 16. The reliability variation is plotted in Figure 5.3. The results are given for PE failure rate $\lambda_p = 0.0001$, MM failure rate $\lambda_m = 0.0001$, and SE failure rate $\lambda_s = 0.00002$. We have assumed a perfect coverage in this study. However, coverage parameters for the PEs, MMs, and SEs can be included in the model directly. The solid lines express the analytical results. The model is validated by plotting the simulation results, shown by dotted lines. It can be observed that system reliability increases by allowing graceful degradation.

5.2 64x64 System Reliability

The system size grows in powers of 4 when 4x4 switches are used for the network. Therefore, a 4^ix4^i system can always be decomposed into 4 4^{i-1}x4^{i-1} systems without disturbing connections. Figure 5.4 shows the decomposition of a 64x64 architecture into four 16x16 groups. A 64x64 configuration has three stages with 16 switches in each stage. As mentioned in the previous section, the input stage switches (stage 0) are included with the processors, and the output stage switches are included with

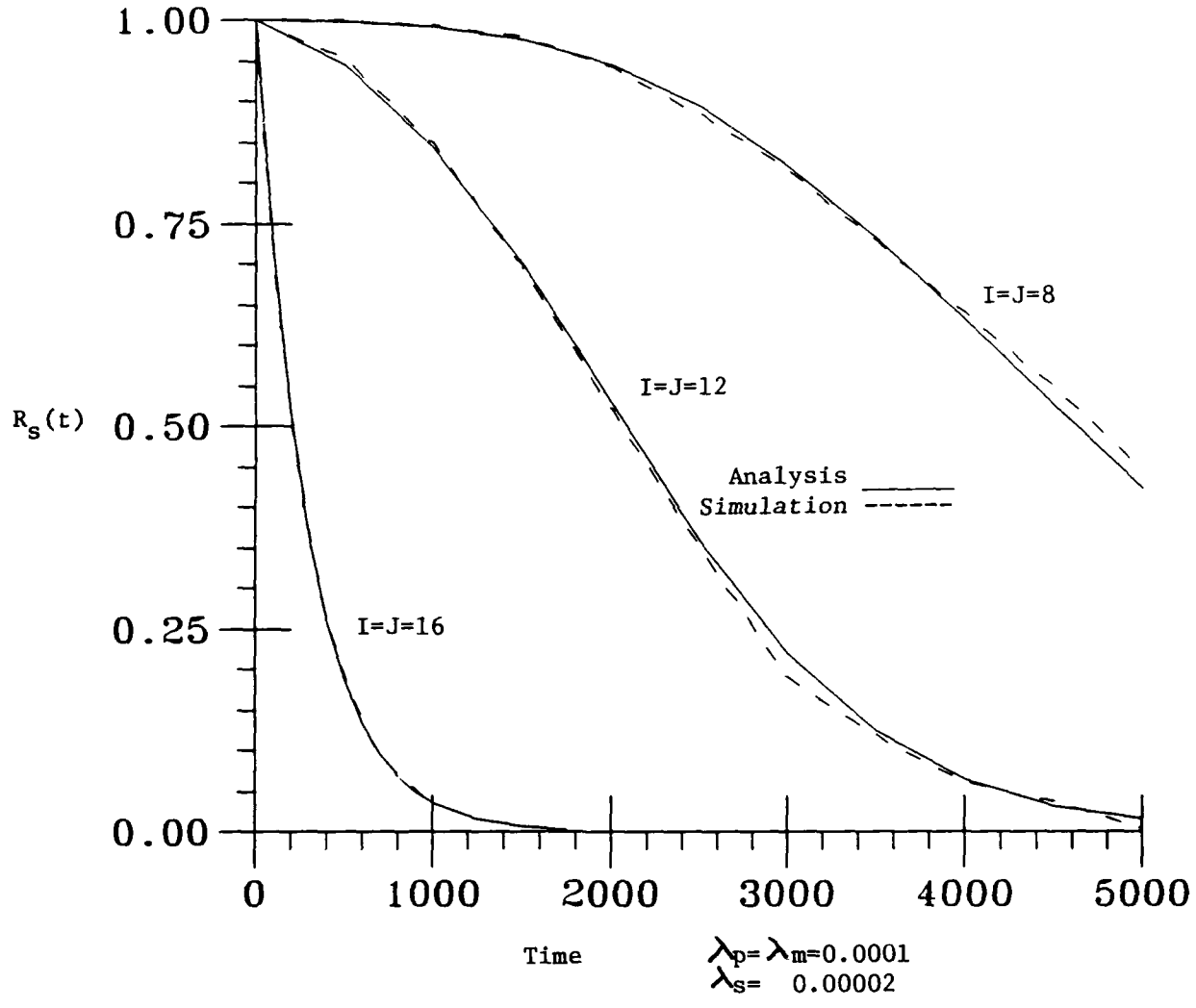


Fig. 5.3 Reliability Variation of a 16x16 Multiprocessor.

the memories. Hence, each group of 16 PEs (PG_x), or 16 MMs (MG_y), has four SEs associated with it. We represent a processor-memory group by (PG_x,MG_y) for $0 \leq x, y \leq 3$.

The distribution of i PEs and j MMs between the four groups can be done in the same way as for the 16x16 system. However, the middle stage switch (stage 1) controls the access between various processor-memory groups. It can be observed from Figure 5.4 that PG_x $(0 \leq x \leq 3)$ can access MG_x using only one of the stage 1 switches, whereas, PG_x uses two switches in stage 1 for a round trip communication with MG_y when $x \neq y$. We number the stage 1 switches by a 2-tuple notation S_{xy}. The first number, x, represents the processor group, and the second number, y, represents the memory group for which a switch is used. For example, switch 10 is used for a request from PG_1 to MG_0. The round trip path is established through switch (01). The connection between various processor-memory groups is represented by a switching node table given in Figure 5.5. It should be observed that the upper and lower triangular entries in the table are exactly the same.

The switching node table is used to calculate the number of stage 1 switches required for connection between a group of processors and memories. For example, if PG_0 and PG_1 need all the four memory groups, 12 SEs are required. This is because SEs (01) and (10) are common to both the groups. These two switches should be included only once to calculate the total number of SEs required to establish connection.

5.2.1 Processor Memory Distribution

There are two different ways a connected group of i PEs and j MMs can be available on the system. The first is the case where exactly i PEs and j MMs are working, and at least the required number of stage 1 SEs are perfect for providing connection between any PE and MM. In the second situation, more than the required number of processors and/or memories may be working on the system, but, the total connectivity is (ixj). This is possible when the number of stage 1 working switches

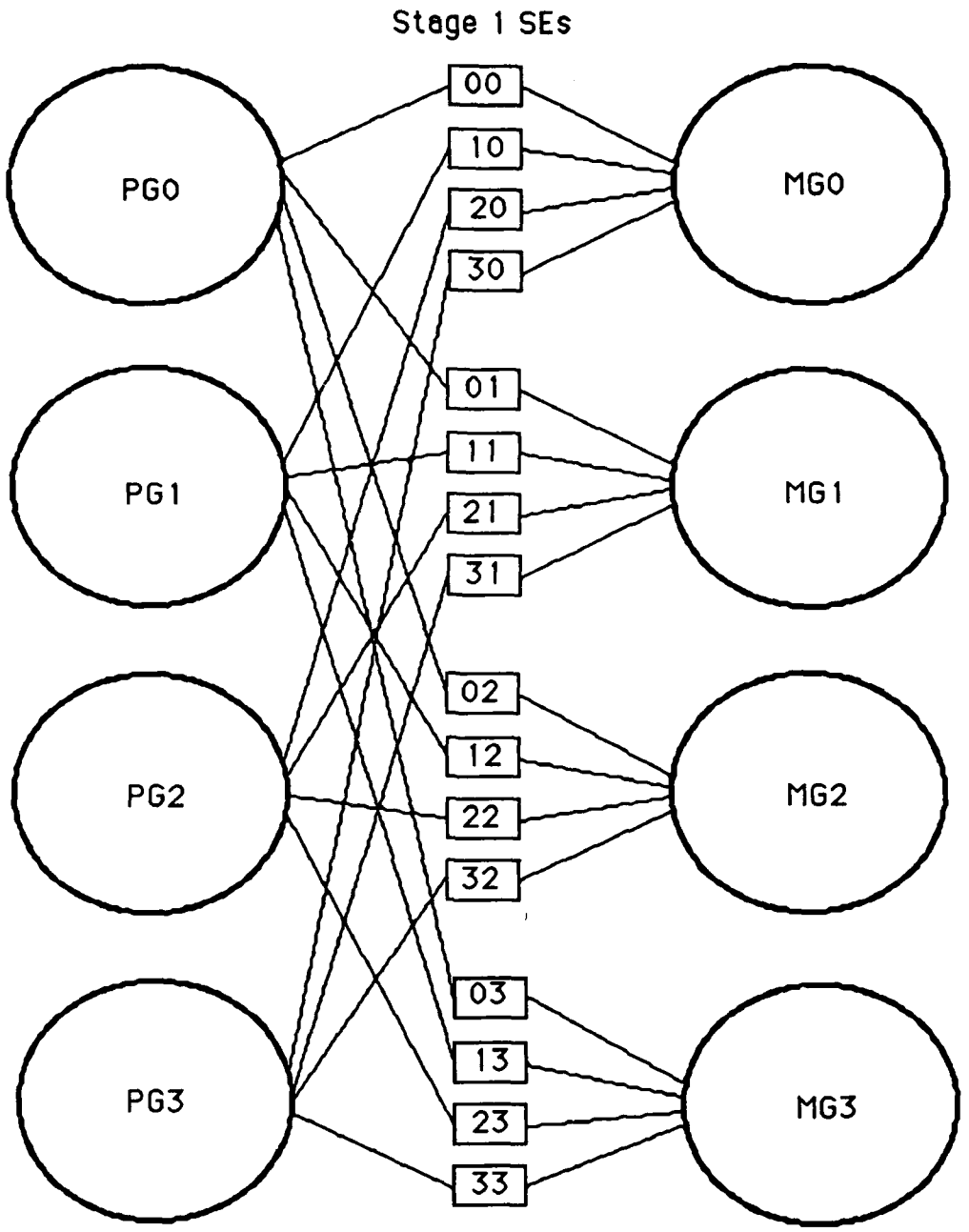


Fig 5.4 The decomposition of the 64x64 architecture to four 16x16 groups.

MG / PG	0	1	2	3
0	00	01 10	02 20	03 30
1	10 01	11	12 21	13 31
2	20 02	21 12	22	23 32
3	30 03	31 13	32 23	33

Fig. 5.5 The switching node table .

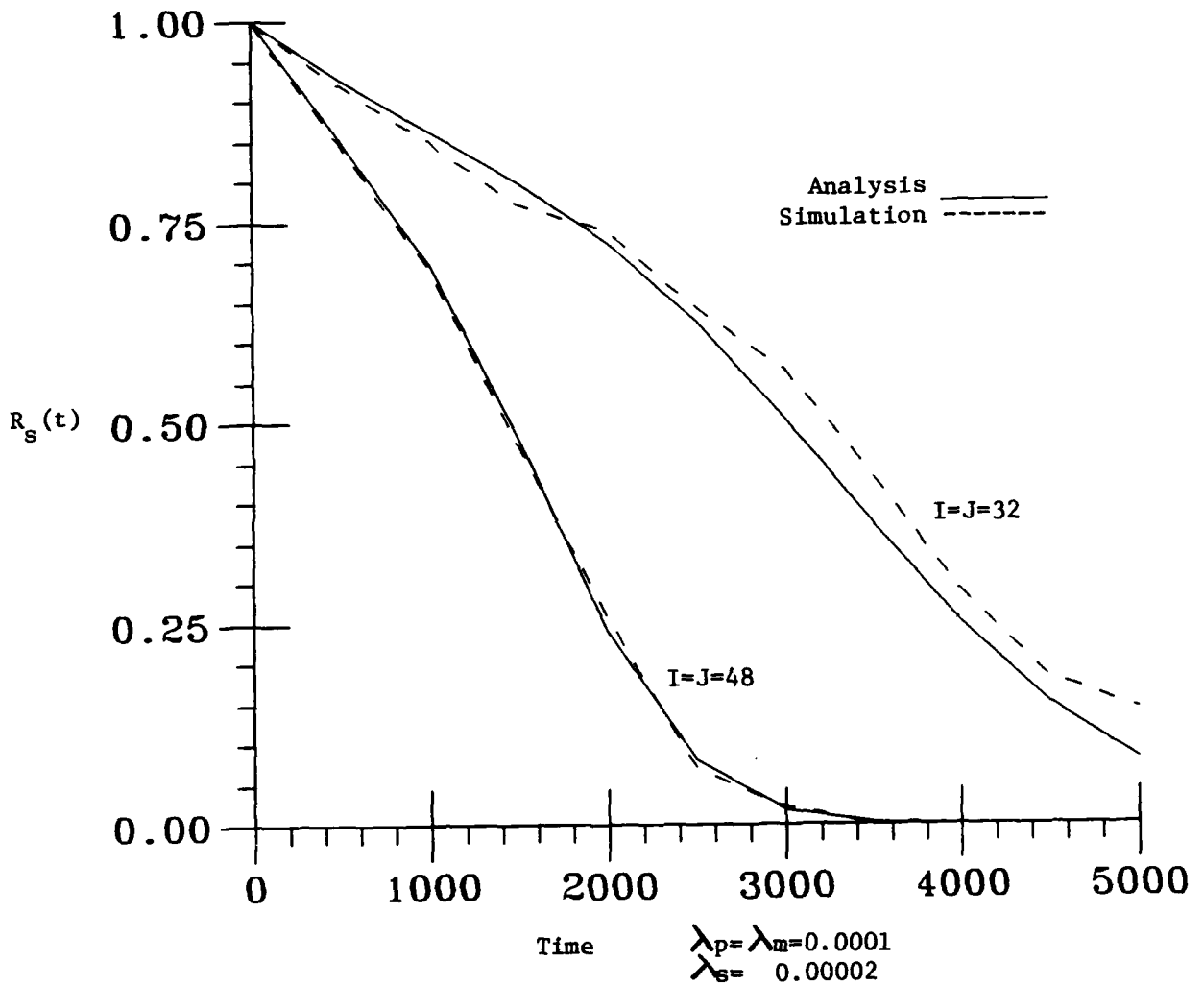


Fig. 5.6 Reliability Variation of a 64x64 Multiprocessor.

are just sufficient to provide a connectivity (ixj). Both the cases are analyzed in detail below.

5.2.2 Exactly (ixj) elements working

Since the i PEs and j MMs can be distributed in upto four groups, the first step in computing this probability is to find the number of stage 1 switches required for connection. Let Nc represents the number of stage 1 switches required to connect i PEs and j MMs. Nc is given by

$$Nc = \sum_{x=0}^{3} \sum_{y=0}^{3} N_{xy} | (PG_x \vee MG_y \neq 0 \ \& \ N_{xy} \ not \ included) \tag{5.5}$$

where N_{xy} is the number of SEs required to connected PG_x to MG_y. N_{xy} is obtained from the switching node table of Figure 5.5. $N_{xy} = 2$ when $x \neq y$, and $N_{xy} = 1$ when $x = y$. It should be observed that equation (5.5) is a conditional expression. The first condition says that if there are no working elements either from a processor or memory group, then $N_{xy} = 0$. The second condition ensures that the same switch should not be included twice. For example, (PG_0, MG_1) connection and (PG_1, MG_0) connection need the same two switches (01) and (10). Therefore, N_{10} should not be included in Nc, as N_{01} is already included. The Nc calculation is illustrated below by an example.

Example-1

Let the i PEs be distributed in groups PG_0, PG_1, and PG_2. The j MMs are selected from MG_1, MG_2, and MG_3. Then

$$Nc = N_{00} + N_{01} + N_{02} + N_{03} + N_{10} + N_{11} + N_{12} + N_{13} + N_{20} + N_{21} + N_{22} + N_{23}$$
$$= 0 + 2 + 2 + 2 + 0 + 1 + 2 + 2 + 0 + 0 + 1 + 2 = 14$$

As exactly Nc SEs are required to connect i PEs and j MMs,the stage 1 SEs are now divided into two groups. The first group is the required number of SEs Nc. The second group is the additional SEs (16-Nc). The state of the additional switches does not affect the working group (ixj). Therefore, the probability of i PEs and j MMs working at time t given by

$$P_{64(i,j)}(t) = \sum_{i_0=0}^{k_0} \sum_{i_1=0}^{k_1} \sum_{i_2=0}^{k_2} \sum_{j_0=0}^{l_0} \sum_{j_1=0}^{l_1} \sum_{j_2=0}^{l_2} P_{16(i_0,j_0)}(t)$$

$$P_{16(i_1,j_1)}(t) P_{16(i_2,j_2)}(t) P_{16(i_3,j_3)}(t) R_{se}(t)^{Nc} \tag{5.6}$$

Equation (5.6) is identical to equation (5.3) except that the subgroup sizes are 16 instead of 4. Evaluation of the term $P_{16(i_x,j_y)}(t)$, for $0 \leq x, y \leq 3$, is done using equation (5.3).

5.2.3 More than (ixj) elements working

This is the situation where the number of connected processors and memories are limited by the failure of the stage-1 switching elements. We illustrate this situation by an example.

Example-2

Consider the distribution $P_{16(16,16)}(t)$, $P_{16(16,16)}(t)$, $P_{16(16,0)}(t)$, and $P_{16(0,0)}(t)$. All the PEs and MMs from group 0 and group 1 are working. Group 2 has only 16 PEs working but memory connection is zero. Group 3 has all elements 0. The system size is given by (48x32). Let N_p be the number of $(0,MG_x)$ groups and N_m be the number of $(PG_x,0)$ groups in the distribution. For the above case NP=0 and NM=1. Now,as long as S_{22} has failed, the number of MMs working in the third group is immaterial. PG_2 and MG_2 are disconnected when S_{22} has failed. PG_2 and MG_2 are individually connected to the first and second group through S_{20}, S_{02} and S_{21}, S_{12}, but not connected as a (PG_2, MG_2) group. Hence, the system size remains the same, (48x32), with working MMs in group 2.

When at least one of the SEs from each group N_{30}, N_{31}, and N_{32} has failed, the number of working processors and memories from group 3 does not increase the system size from (48x32). In other words, the failure of at least one SE from N_{xy} disconnects processor group x from memory group y. The above two distributions, (0,0) and $(PG_x,0)$ or $(0,MG_x)$, are combined to give the maximum numbers of switches

N_f that can fail to disconnect the failed groups from the rest of the system. N_f is expressed as

$$N_f = N'_{f1} + N_{f2} \tag{5.7}$$

The first term N'_{f1} gives the maximum number of SEs that can fail to disconnect a (0, 0) group from the rest of the system. The second term N_{f2} denotes the number of SEs that must fail to keep a group size (PG_x, 0) or (0, MG_x) even though there is at least a memory or processor working in the null groups respectively. Let N_{f1} denotes the minimum number of SEs that must fail to disconnect a (0,0) group from the rest of the system. N'_{f1}, N_{f1}, and N_{f2} can be expressed as

$$N'_{f1} = \sum_{x=0}^{3}\sum_{y=0}^{3}\{N_{xy}|_{(PG_x \wedge MG_x=0),\ (x\neq y),\ \&(N_{xy}\ not\ included)}\} \tag{5.8.a}$$

$$N_{f1} = \frac{N'_{f1}}{2} - 1 \tag{5.8.b}$$

and

$$N_{f2} = \{I_{\{N_p>N_m\}}((N_p - 1) + N_m) + I_{\{N_p<N_m\}}((N_m - 1) + N_p)\}+$$

$$\sum_{x=0}^{3}\{N_{xx}|_{(PG_x \vee MG_x=0),\ (PG_x \wedge MG_x\neq 0),\ \&(N_{xx}\ not\ included}\} \tag{5.8.c}$$

Both these terms are conditional to avoid the inclusion of the same switches twice. The first term N'_{f1} counts the total number of SEs that disconnects a (0,0) group from the working groups of the system. It does not include the SE N_{xx}. The third term N_{f2} counts only the N_{xx} switches for a (PG_x,0) or (0,MG_x) group. The first term in (8.c) counts the switches that should fail between more than one (PG_x,0) or (0,MG_x) groups. The indicator function $I_{\{N_p>N_m\}}$ and $I_{\{N_p<N_m\}}$ are used to select the proper switches. The second term in (8.c) counts the N_{xx} switches for a (PG_x,o) or (0,MG_x) group. The third term does not include N_{xx} for a (0,0) group. On the other hand, the minimum number of SEs that must fail to keep the system size (ixj) is given by

$N_{fm} = (N_{f1} + N_{f2})$. This is because 1 out of each 2 SEs in $N_{xy(x \neq y)}$ is sufficient for disconnection of a (0,0) group. Also 2 out of 3 group switches are sufficient to disconnect a (0,0) group. For the above example, $N_{f1} = 2$, $N_{f2} = 1$, and $N_{fm} = 3$. To keep the SE failure model simple, we consider only the minimum number of SEs required to disconnect the groups. Since 1 out of 2 SEs for each N_{xy} is required for a (0,0) group, the total number of ways the N_{fm} can be selected is given by

$$X = \binom{N'_{f1}}{N_{f1}} \cdot 2^{N_{f1}} \times \{I_{\{N_p > N_m\}} \binom{N_p}{N_p - 1} + I_{\{N_p < N_m\}} \binom{N_m}{N_m - 1}\} \qquad (5.9)$$

For the above example $X = \binom{3}{2} \times 2^2$.

The situation where more than i PEs and j MMs are working but the total connectivity is (ixj) is then given by

$$P_{64(i,j)}(t) = \sum_{i_0=0}^{k_0} \sum_{i_1=0}^{k_1} \sum_{i_2=0}^{k_2} \sum_{j_0=0}^{l_0} \sum_{j_1=0}^{l_1} \sum_{j_2=0}^{l_2} P^*_{16(i_0,j_0)}(t) P^*_{16(i_1,j_1)}(t)$$

$$P^*_{16(i_2,j_2)}(t) P^*_{16(i_3,j_3)}(t) R_{se}(t)^{N_c} (1 - R_{se}(t))^{N_{fm}} X \qquad (5.10)$$

$R_{se}(t)^{N_c}$ in equation (5.10) gives the probability that the required number of SEs N_c are working to keep the connectivity (ixj). $(1 - R_{se}(t))^{N_{fm}} X$ represents the probability that the minimum number of SEs has failed so that the connectivity is (ixj) while there are actually more than i PEs and/or j MMs working in the system. The term $P_{16(i_x,j_x)}(t)$ in equation (5.10) stands for

$$P^*_{16(i_x,j_x)}(t) = \begin{cases} P_{16(i_x,j_x)}(t) & if\ i_x \wedge j_x \neq 0 \\ \sum_{x=1}^{16} P_{16(x,j_x)}(t) & if\ i_x = 0\ \&\ j_x \neq 0 \\ \sum_{x=1}^{16} P_{16(i_x,x)}(t) & if\ i_x \neq 0\ \&\ j_x = 0 \\ \sum_{x=1}^{16} \sum_{y=1}^{16} P_{16(x,y)}(t) & if\ i_x = j_x = 0 \end{cases} \qquad (5.11)$$

Evaluation of the term $P^*_{16(i_x,j_x)}(t)$ depends on the distribution. When neither a processor nor a memory group is zero in group x, $P_{16(i_x,j_x)}(t)$ is the same as equation

(5.3). When either the processor or memory group is zero, the corresponding probabilities are added for $1 \leq x \leq 16$ to compute $P^*_{16(i_x,j_x)}(t)$. The $P_{16(0,0)}(t)$ is computed by the fourth term of equation (5.11). It should be observed that the minimum value of x is 1 for the summations in equation (5.11), since all failed element probabilities are included in equation (5.6).

5.2.4 Reliability Computation

The reliability of a 64x64 system can be computed by combining equations (5.6) and (10). It should be observed that all valid working groups are generated by the above two equations. The only probabilities that are not included are in equation (5.10) where more than N_{fm} SEs can also fail while keeping the system size (ixj). However, the contribution of this expression is negligible compared to equation (5.6). This is mostly because of the term $(1 - R_{se}(t))N_{fm}$. When we take more than the minimum number, N_{fm}, this probability decreases even faster. This argument is valid when the required system size is about 50% of the original size. With i and j less that 32, the contribution from equation (5.10) is about 10%.

The computation of equation (5.10) is very costly in terms of time. Equations (5.6) and (10) both generate all the distributions and compute N_c. In addition, equation (5.10) generates N_{fm}, and whenever there is a PE and/or MM group 0, it computes either one or more of the last three expressions of equation (5.11). When (ixj) size is close to (NxN), the possibility of an $(i_x \vee j_y = 0)$ is negligible. With lower (ixj) values the probability of finding a null processor and/or memory group increases.

It can be observed that both equations (5.6) and (5.10) are combinatorial expressions. All possible distributions of (ixj) are generated in these equations. Probability computation for all of these combinations is time consuming. However, the equations can be evaluated efficiently by avoiding the regeneration of the similar of distributions. For example, consider a system of size (48x48). Four possible distributions are :

Processor	Memory
(16, 16, 16, 0)	(16, 16, 16, 0)

(16, 16, 0, 16)	(16, 16, 0, 16)
(16, 0, 16, 16)	(16, 0, 16, 16)
(0, 16, 16, 16)	(0, 16, 16, 16)

All of these combinations have the same probability, since N_c is 9 for all four cases. Hence, we need compute only one of these terms. By avoiding the recomputation of similar distributions, the computation of reliability becomes faster.

Figure 5.6 shows the reliability variation for a 64x64 multiprocessor with I=J=48 and I=J=32. The I=48 result is plotted using only equation (5.6). The analytical results match closely with simulation without including equation (5.10), since with i and j equal to 48, only one group of PEs and MMs can be 0 at a time. Hence, the value from equation (5.10) are negligible. The results for I=32 are plotted by combining equations (5.6) and (5.10). We have observed that using only equation (5.6), the results differ from the simulation less than 10%. So, if the reliability requirements are not stringent, equation (5.6) should be sufficient to give a close lower bound on reliability.

5.3 Generalization to Higher Systems

It is possible to extend the analysis of 64x64 system to 256x256 multiprocessor. The basic nature of the equations (5.6) and (10) remain the same except that each process or memory group has now 64 elements. A unique path 256 node configuration has 4 stages of SEs : stages 0, 1, 2, and 3. Each stage has 64 SEs. The decomposition of the 256x256 system into 4 64x64 groups is done by associating the stage 0-(input) SEs with processors, and stage 2, and 3 SEs with the memory side. Hence, a group of 64 PEs has 16 SEs associated with it. A group of 64 MMs has 16 SEs of stage 2 and 16 SEs of stage 3 associated with it. These 64 PEs and 64 MMs have the identical connection of a 64x64 system.

The four groups of 64 PEs and four groups of 64 MMs are connected through 64 stage 1 switches. These 64 stage 1 SEs can be divided into 16 groups, each having 4 switches. We can then represent the stage 1 connection of the 256 node system by the same switching node table of Figure 5.5. Now, each 2-tuple notation S_{xy} represents a

group of 4 SEs. For example, (00) will stand of 4 SEs that connect 64 PEs of group 0 to 64 MMs of group 0. Similarly, 8 SEs (01) and (12) are needed for communication between PG_0 and MG_1. The required number of SEs N_c for any system size (ixj) can be found by counting the number stage 1 of groups and multiplying this number by four. N_{fm} can also be computed similarly using equation (5.7). Hence, equations (5.6) and (5.10) can be used by changing each $P_{16(i_x,j_y)}(t)$ notation to $P_{64(i_x,j_y)}(t)$. The (64x64) system results are used to compute (256x256) system reliability.

It is theoretically possible to use equations (5.6) and (5.10) for a 256 node reliability computation. But the computation time is prohibitive. This will be illustrated by an example. Let us assume that we want simply to compute the probability of (192x192) distribution. One possible processor grouping is (64, 64, 64, 0). The memory combinations for this processor grouping vary from (64, 64, 64, 0) to (0, 64, 64, 64). The generation and computation of this large number of memory distributions for each processor distribution make this model unattractive for higher order systems such as the 256 node system. One can avoid recomputation of similar combinations, as discussed in section 4.2, to save computation time. Using these simplification techniques, we have computed the reliability of a 256x256 multiprocessor requiring at least 192 PEs and 192 MMs. The result is plotted in Figure 5.7. We also have written a simulation program for 256 node system to verify this analytical results. The results are compared in Figure 5.7.

The disadvantage of equation (5.6) and (5.10) for higher order systems is mainly because of the generation of all distributions, and in finding the numbers N_c and N_{fm}. Therefore, there is no approximation involved in the model except in neglecting the terms N_{fm+1} to N_f. As mentioned in section 4.2, contribution of these terms is very small. We are currently looking at approximation techniques that can be used to compute 256 node system reliability efficiently.

One such approach is to use a recursive computation of higher order systems starting from the 4x4 model. The first and last stage SEs are always included with the processors and memories. Starting from the 4x4 model, we can compute the

reliabilities of (8x8), (16x16), (32x32), (64x64), and (128x128) without considering the middle stage SEs. A (256x256) system reliability can then be computed by considering two (128x128) systems. An approximate number of middle stage SEs will be included in these expressions to provide connections between the PEs and MMs. For example, a (64x64) system working with (48x48) configuration needs at least 9, 12, or 16 SEs from stage 1 depending on the processor memory distribution. We should then be able to get a fairly accurate result for (64x64) system by including an average value for the reliability of stage 1 SEs with two (32x32) system reliability. The same principle can be applied for a 256 node multiprocessor.

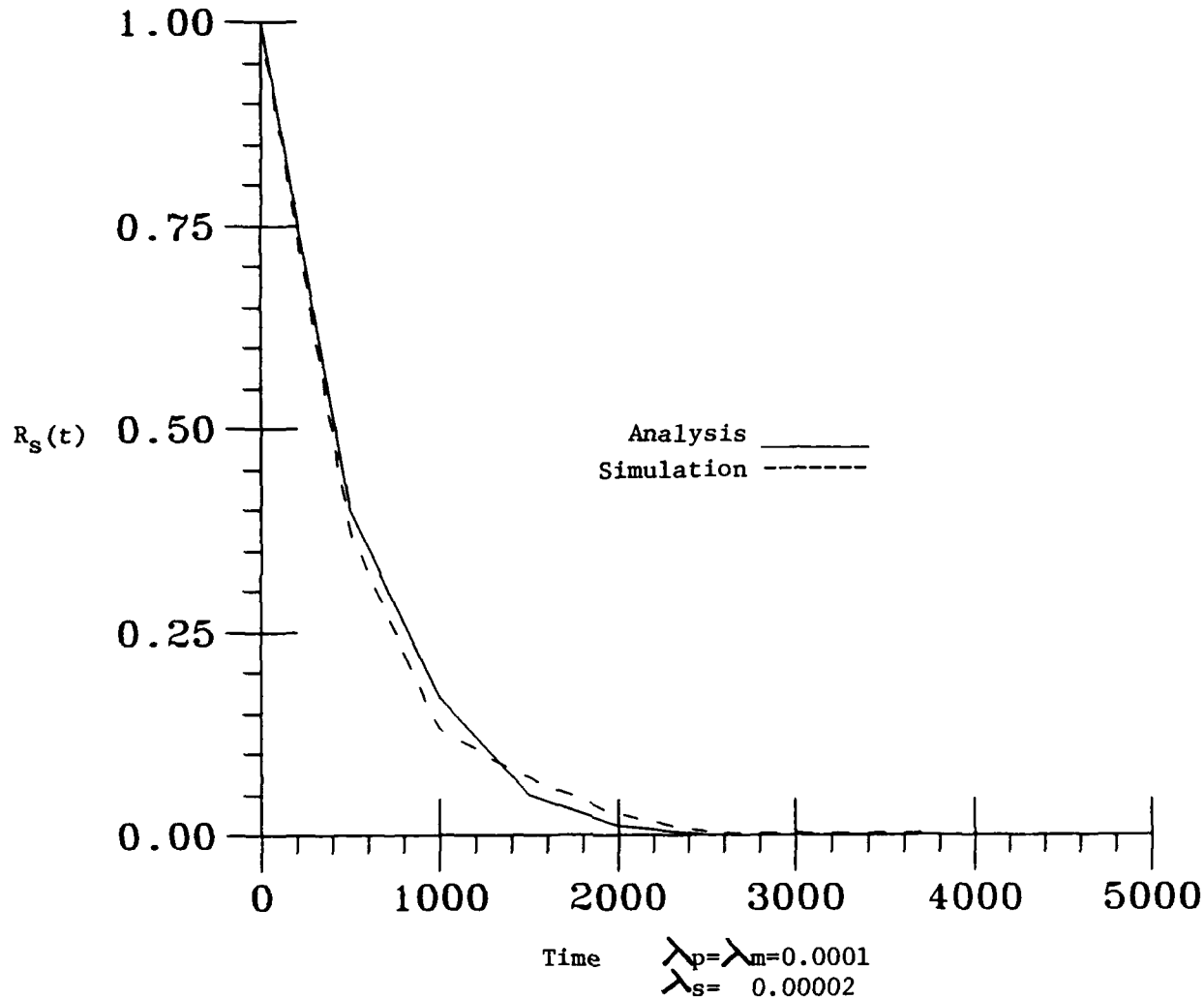


Fig. 5.7 Reliability Variation of a 256x256 Multiprocessor for I=J=192.

CHAPTER 6

HYPERCUBE DEPENDABILITY MODELING

We have developed an approximate techniques to compute the reliability of Hypercube multiprocessors. The model is based on the decomposition principle, where a hypercube of a higher dimension is recursively decomposed into smaller hypercubes, until the reliability of the smallest cube is modeled exactly. The reliability of the large n-cube is then obtained from this smallest base model using a recursive equation. The reliability model used is task based - it is assumed that the system is operational if the task can be executed on the system. Analytical results are given for n-dimensional hypercubes with upto 75% system degradation. The model is validated by comparing analytical results with simulation results.

6.1. Modeling Technique

We use a 2-cube (4 nodes) or 3-cube (8 nodes) system as the base model in this analysis. The exact task based reliability analysis of the base model is first done for various numbers of required nodes, I, where $I \leq 2^n$ for $n = 2$ or 3. The reliability of a higher dimension cube is obtained recursively from the base model results. We decompose an n-cube into 2 (n-1)-cubes, each (n-1)-cube in turn into 2 (n-2)-cubes, etc., until a 4-cube is divided into 2 base model 3-cubes. We start with the exact base model equations and derive results for a higher dimension system by considering the connectivity between two (n-1)-cube systems. One possible decomposition of the problem for a 5-cube system with 20 nodes working is given in Figure 6.1.

In this report, we shall assume that the failure rate of the links is negligible compared to the node failure rate. Thus, only processor failure is considered. While this is an optimistic assumption, it is widely used in the modeling of parallel architectures to keep the analysis simple. Further, if we include the failure rate of the common I/O

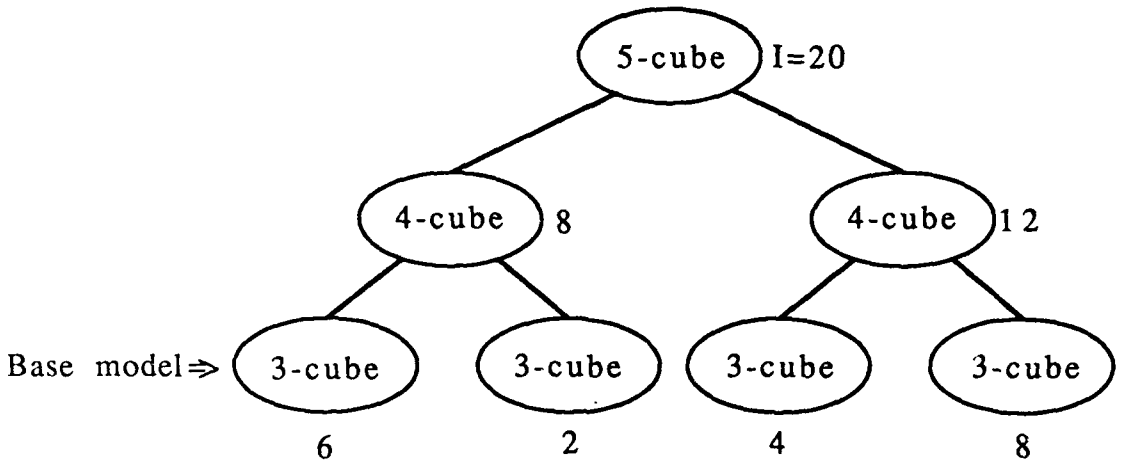


Fig. 6.1 A decompositiono of a 5-cube with 20 connected nodes.

bus along with the processor failure rate, the failure probability of individual channels becomes very small. In this case, the link failure assumption becomes less critical.

We assume homogeneity of processing nodes, with identical and exponential distribution of failure time. We define λ_n as the failure rate of a node. We consider reliability evaluation of only non-repairable hypercube systems in this report. A separate front end host processor is assumed to perform all the maintenance action. Detection and isolation of the failed nodes, and reconfiguration of the system to a degraded mode, are all done by the host processor. Host processor failure probability is not considered in this report. However, this can be included into the model without much difficulty.

We use the following notation in this analysis:

Notation:

N : Number of nodes in the hypercube, $N=2^n$.

X_n : Random variables that represent the number of processors (nodes) in the n-cube.

$G(N,i,p)$: $\binom{N}{i}p^i(1-p)^{N-i}$, the probability of having exactly i good units out of N units, where p is the unit reliability.

λ_n : Node failure rate.

$R_n(t)$: Node reliability at time t, given by $e^{-\lambda_n t} = p$

$P(X_n = i)$:The probability of having i good connected units in the n-cube.

$R_s(t)$: The n-cube transient reliability.

$C_{n-1}(i,j)$: Connectivity of two (n-1)-cubes; one cube with i connected processors and the other cube with j connected processors.

$P(C_{n-1} \neq i | X_{n-1} = i)$: The conditional probability of having i disconnected processors working in the (n-1)-cube.

$D_{n-1}(i,j)$: Connectivity of two (n-1)-cubes. One group with i connected nodes and the second group with j disconnected nodes.

$G_c(N,x)$: Number of x connected nodes from N.

$G_d(N,x)$: Number of x disconnected nodes from N.

$C_c(x,y)$: Number of y connected nodes from x connected nodes

6.2 The Base Model

In this section exact analyses for 2-cube and 3-cube configurations are presented. We assume perfect coverage in all this analyses. However, an appropriate value for coverage could be included in the model without changing its basic structure.

6.2.1 2-Cube analysis

A 2-dimensional hypercube (with N=4) is shown in Figure 6.2.

From simple combinatorics, we have the exact probability for various numbers of connected working nodes at time t :

$$P(X_n = 4) = R_n(t)^4 \tag{6.1.a}$$
$$P(X_n = 3) = 4R_n(t)^3(1 - R_n(t)) \tag{6.1.b}$$
$$P(X_n = 2) = 4R_n(t)^2(1 - R_n(t))^2 \tag{6.1.c}$$
$$P(X_n = 1) = 4R_n(t)(1 - R_n(t))^3 + 2R_n(t)^2(1 - R_n(t))^2 \tag{6.1.d}$$
$$P(X_n = 0) = (1 - R_n(t))^4 \tag{6.1.e}$$

It should be observed from $P(X_n = 1)$ that a situation such as nodes $\{0,3\}$ working, or $\{1,2\}$ working, in Figure 6.2 gives effectively only one working node, as the diagonal elements are not connected.

6.2.2 3-Cube analysis

A 3-dimensional hypercube (with N=8) is shown in Figure 6.3. While the probability expressions below could be obtained using the 2-cube model, we derive them directly due to their simplicity.

The probability of exactly I connected processors working in the 3-cube, for $0 \leq I \leq 8$, are given below.

$$P(X_n = 8) \quad G(4,4,p)G(4,4,p) = R_n(t)^8 \tag{6.2.a}$$
$$P(X_n = 7) \quad 2G(4,4,p)G(4,3,p) = 8R_n(t)^7(1 - R_n(t)) \tag{6.2.b}$$
$$P(X_n = 6) \quad 2G(4,2,p)G(4,4,p) + G(4,3,p)G(4,3,p)$$
$$28R_n(t)^6(1 - R_n(t))^2 \tag{6.2.c}$$

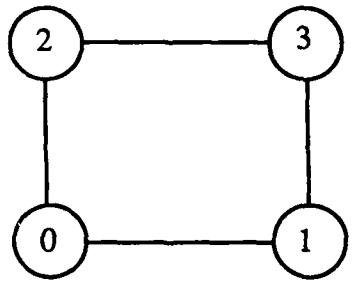


Fig. 6.2 A 2-dimensional hypercube.

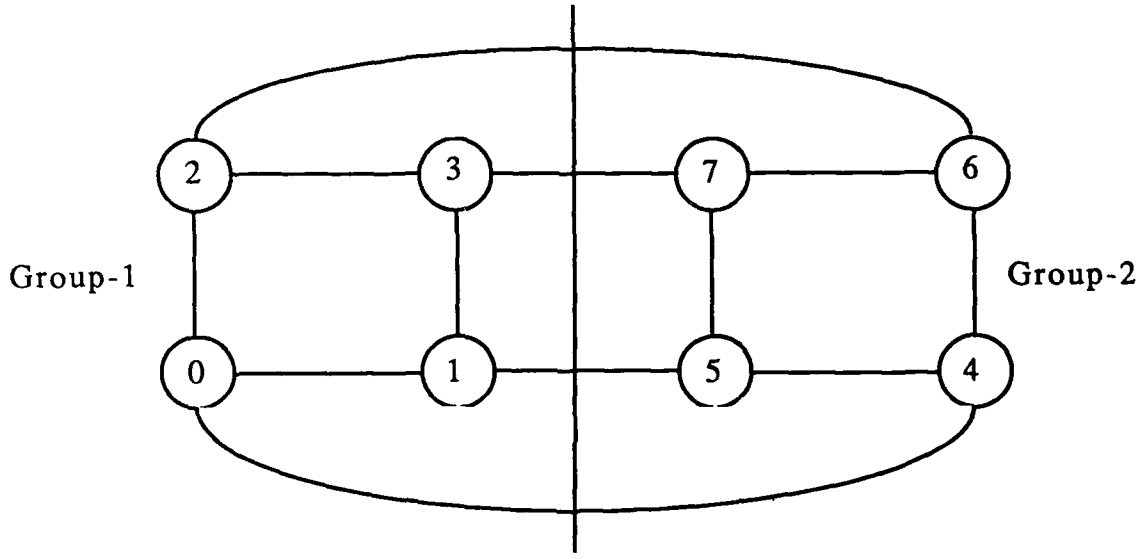


Fig. 6.3 A 3-dimensional hypercube.

$$
\begin{aligned}
P(X_n = 5) &= 2G(4,1,p)G(4,4,p) + 2G(4,2,p)G(4,3,p) - P(X_n = 5)_{disc} \\
&= 56R_n(t)^5(1 - R_n(t))^3 - 2 * 4R_n(t)^5(1 - R_n(t))^3 \\
&= 48R_n(t)^5(1 - R_n(t))^3
\end{aligned} \tag{6.2.d}
$$

The disconnected probability $P(X_n = 5)_{disc}$ appears in the above expression to take care of the situations where 4 out of 5 working nodes are connected. An example is when nodes $\{0,2,3,5,6\}$ are working. There will be 8 such cases in the 3-cube.

$$
\begin{aligned}
P(X_n = 4) &= P(X_n = 5)_{disc} + 2G(4,0,p)G(4,4,p) \\
&\quad + 2G(4,1,p)G(4,3,p) + G(4,2,p)G(4,2,p) - P(X_n = 4)_{disc} \\
&= 8R_n(t)^5(1 - R_n(t))^3 + 2R_n(t)^4(1 - R_n(t))^4 + 32R_n(t)^4(1 - R_n(t))^4 \\
&\quad + 36R_n(t)^5(1 - R_n(t))^4 - 32R_n(t)^4(1 - R_n(t))^4 \\
&= 8R_n(t)^5(1 - R_n(t))^3 + 38R_n(t)^4(1 - R_n(t))^4
\end{aligned} \tag{6.2.e}
$$

The $P(X_n = 5)_{disc}$ is the same as $P(X_n = 5)_{disc}$ terms of the previous equation (2.d). With 4 nodes working, there can be 1, 2, or 3 connected nodes which should be subtracted from the $P(X_n = 4)$. For example, working nodes $\{1,2,4,7\}$ are all disconnected and contribute only to $P(X_n = 1)$. Nodes $\{0,2,5,7\}$ give 2 connected groups and add to $P(X_n = 2)$. Finally, nodes $\{0,2,3,5\}$ contribute only to $P(X_n = 3)$. All of these disconnected terms are included in the term $32R_n(t)^4(1 - R_n(t))^4$.

$$
\begin{aligned}
P(X_n = 3) &= P(X_n = 4)_{disc} + 2G(4,0,p)G(4,3,p) \\
&\quad + 2G(4,1,p)G(4,2,p) - P(X_n = 3)_{disc} \\
&= 24R_n(t)^4(1 - R_n(t))^4 + 8R_n(t)^3(1 - R_n(t))^5 \\
&\quad + 48R_n(t)^3(1 - R_n(t))^5 - 32R_n(t)^3(1 - R_n(t))^5
\end{aligned}
$$

The first term represents the 24 cases where, out of 4 working nodes, only 3 are connected. The last term represents the 32 cases where $C_{n-1}(1,2) \neq 3$. The expression simplifies to

$$
\begin{aligned}
P(X_n = 3) &= 24R_n(t)^4(1 - R_n(t))^4 + 24R_n(t)^3(1 - R_n(t))^5 \qquad (6.2.f) \\
P(X_n = 2) &= P(X_n = 4)_{disc} + P(X_n = 3)_{disc} + 2G(4,0,p)G(4,2,p) \\
&\quad + G(4,1,p)G(4,1,p) - P(X_n = 2)_{disc}
\end{aligned}
$$

$$= 6R_n(t)^4(1 - R_n(t))^4 + 24R_n(t)^3(1 - R_n(t))^5$$
$$+ 12R_n(t)^2(1 - R_n(t))^6 + 16R_n(t)^2(1 - R_n(t))^6 - 16R_n(t)^2(1 - R_n(t))^6$$

The first term represents the cases where 2 out of 4 working nodes are connected. The second term represents the 24 cases where 2 out of 3 working nodes are connected. The last term is subtracted to take care of the situations where 2 working nodes are disconnected. After simplification we get

$$P(X_n = 2) = 6R_n(t)^4(1 - R_n(t))^4 + 24R_n(t)^3(1 - R_n(t))^5 + 12R_n(t)^2(1 - R_n(t))^6 \quad (6.2.g)$$

$$P(X_n = 1) = P(X_n = 4)_{disc} + P(X_n = 3)_{disc} + P(X_n = 2)_{disc} + 2G(4,0,p)G(4,1,p)$$
$$= 2R_n(t)^4(1 - R_n(t))^4 + 8R_n(t)^3(1 - R_n(t))^5$$
$$+ 16R_n(t)^2(1 - R_n(t))^6 + 8R_n(t)(1 - R_n(t))^7 \quad (6.2.h)$$

The first term represents the two cases where two diagonal elements from each 2-cube are working, but are disconnected. The second term is for the 8 cases where all 3 working nodes are isolated. The third term is for the cases where 2 working nodes are disconnected

$$P(X_n = 0) = (1 - R_n(t))^8 \quad (6.2.i)$$

6.3 Generalized Model

In this section we develop a generalized reliability model for an n-cube, for $n > 3$. We assume that the system works as long as I connected processors are working in the hypercube. The system reliability is expressed as

$$R_s(t) = \sum_{j=I}^{N} P_j(t) \quad (6.3)$$

where $P_j(t)$ is the probability that j connected processors are working in the system at time t. At any time t, $P_j(t)$ is given by $P(X_n = j)$. Hence, dropping the time parameter, system reliability can be written as

$$R_s(X_n \geq I) = \sum_{j=I}^{N} P(X_n = j) \quad (6.4)$$

6.3.1 System Decomposition

To calculate the probability $P(X_n = j)$ we divide the n-cube into two (n-1)-cubes (groups). There are two situations under which there will be j connected processors in the n-cube. In the first situation, exactly j connected nodes are working in the hypercube, with k nodes in one group and (j-k) nodes in the second group. In the second situation there are more than j nodes working in the hypercube, but the actual connectivity is only j. For either of these two cases, there are two possibilities for the nature of the connectivity between the two (n-1)-cubes. Either the k and (j-k) nodes working in the two (n-1)-cubes are all connected in their individual groups and the total connectivity is j, or one of the two groups is not internally connected but the total connectivity is still j. For the second case, where a total of more than j nodes are working in the two groups, the two working groups may or may not be connected. These four possible working node distributions are discussed below.

Distribution I

This is the case where there are two connected groups with a total of exactly j *connected nodes* for a given $j > 1$. Let us divide these nodes into two groups such that k connected nodes are working in one of the (n-1)-cubes, and the remaining (j-k) connected nodes are working in the second group. These two groups must be connected such that the total connectivity is j. This situation can be represented as

$$P(X_{n-1} = k)P(X_{n-1} = j-k)(1 - P(C_{n-1}(k, j-k) \neq j)) \qquad (6.5)$$

The first two terms give the probabilities that k and (j-k) nodes are connected in their respective groups. $P(C_{n-1}(k, j-k) \neq j))$ is the probability that the total connectivity in the n-cube, which is given by the connectivity of k and (j-k) nodes in the two (n-1)-cubes, is not j. Since we are interested in exactly j connections, $(1 - P(C_{n-1}(k, j-k) \neq j))$ is used in equation (6.5). If $j > 2^{n-1}$, then $P(C_{n-1}(k, j-k) \neq j) = 0$.

Distribution II

The other possibility is that k connected nodes are working in one (n-1)-cube, and (j-k)-nodes are working in a disconnected fashion in the second (n-1)-cube, but all of the j nodes happen to be connected. For example, assume that node 3 has failed in group-1 and nodes 4 and 7 have failed in group-2 in Figure 6.3. Hence, the two working nodes 5 and 6 are disconnected in group-2. But all the five nodes are working connected due to the hypercube topology. This situation can be expressed as

$$P(X_{n-1} = k)P(C_{n-1} \neq j-k | X_{n-1} = j-k)(1 - P(D_{n-1}(k, j-k) \neq j)) \qquad (6.6)$$

The second term in equation (6.6) gives the conditional probability that (j-k) working nodes are disconnected. The term $P(D_{n-1}(k, j-k) \neq j)$ gives the probability that the total connectivity of the k connected nodes from one (n-1)-cube and (j-k) disconnected nodes from the second (n-1)-cube is not j. The probability of connectivity being exactly j is obtained by subtracting this value from 1.

We assume that $k \geq (j-k)$, i.e., the group with fewer nodes is the disconnected group, since the probability of disconnection decreases with increasing number of nodes in a group.

Distribution III

The third possibility is that j connected nodes are working in one (n-1)-cube, s processors are working in the second (n-1)-cube, but the two groups are totally disconnected. For example, assume that nodes $\{1, 3\}$ have failed in group-1 and nodes $\{4, 6\}$ have failed in group-2 in Figure 6.3. If j=2, group-1, with 2 connected nodes $\{0, 2\}$, satisfies the task requirement. There are three possibilities in group-2: only 5 works, only 7 works, or both 5 and 7 work. Any of these three possibilities can not increase the total number of connected nodes in the system, as the two groups are always disconnected. Obviously, this kind of situation occurs only if $j < 2^{n-1}$. If $j = 2^{n-1}$, the two groups are always connected. Also distribution III can not occur if $j > 2^{n-1}$. From the 3-cube in Figure 6.3, we find that there are $min(j, 2^{n-1} - j)$ positions for s in group-2 that are always disconnected from the j positions in group-1.

For example, if j=3 in group-1, there is only one position in group-2 that is disconnected from group-1. For j=2, there are only 2 nodes in group-2 that are disconnected from group-1. The probability of distribution III can be approximated as

$$2 \sum_{s=1}^{min(j,2^{n-1}-j)} P(X_{n-1}=j)\binom{2^{n-1}-j}{s} R_n(t)^s(1-R_n(t))^{2^{n-1}-s}, \quad for\ j < 2^{n-1} \tag{6.7}$$

This equation gives nearly exact probability when j is close to but less than 2^{n-1}. As the difference between j and 2^{n-1} increases, equations (6.7) becomes less accurate, since we are not choosing s from all possible $(2^{n-1}-j)$ positions. The factor 2 appears in equation (6.7) since the j connected nodes can be in either of the two (n-1)-groups.

Distribution IV

This last case depicts a situation where some k nodes, $k > j$, are working in the n-cube, but only j of them are connected with the two groups not totally disconnected. This is the reverse case of distribution III, where the two groups are disconnected. For example, suppose that nodes $\{0,3,6\}$ have failed in Figure 6.3 of the 3-cube. This leaves 5 working processors in the system, of which only 4 are connected; node 2 is disconnected. We represent this case as

$$\sum_{k=j+1}^{N} P(C_n = j | X_n = k) \tag{6.8}$$

where N is the total number of nodes in the hypercube. Equation (6.8) represents the probability that j nodes are connected from k working nodes.

Now, by combining all the four cases, the approximate equation for j connected nodes is given by

$$\begin{aligned} P(X_n=j) = & \sum_{k=m}^{M} P(X_{n-1}=k)P(X_{n-1}=j-k)(1-P(C_{n-1}(k,j-k)\neq j)) \\ & + \sum_{k=m}^{M} P(X_{n-1}=k)P(C_{n-1}\neq j-k | X_{n-1}=j-k)(1-P(D_{n-1}(k,j-k)\neq j)) \end{aligned}$$

$$+ 2 \sum_{s=1}^{min(j,2^{n-1}-j)} P(X_{n-1} = j)\binom{2^{n-1}-j}{s} R_n(t)^s (1 - R_n(t))^{2^{n-1}-s}$$

$$+ \sum_{k=j+1}^{N} P(C_n = j | X_n = k) \qquad (6.9)$$

where m and M are given by $m = max(0, j - 2^{n-1})$ and $M = min(2^{n-1}, j)$. These two values determine the lower and upper bounds of k in an (n-1)-cube.

The second term in equation (6.9) is used if $k \geq j - k$. Otherwise, this expression is evaluated as

$$\sum_{k=m}^{M} P(X_{n-1} = j - k) P(C_{n-1} \neq k | X_{n-1} = k)(1 - P(D_{n-1}(j - k, k) \neq j))$$

Also, as explained under distribution III, the third term is evaluated if $j < 2^{n-1}$. It should be observed that equation (6.9) is a recursive expression; the n-cube probability is derived from (n-1)-cube probability. The recursion is continued until (n-1)=2 or 3.

6.3.2 Term Evaluation

There are four different probability terms in equation (6.9) that need to be quantified. These are $P(C_{n-1}(k, j - k) \neq j)$, $P(C_{n-1} \neq j - k | X_{n-1} = j - k)$, $P(D_{n-1}(k, j - k) \neq j)$, and $P(C_n = j | X_n = k)$. We address these terms below.

I) $P(C_{n-1}(k, j - k) \neq j)$

If $j > 2^{n-1}$, the probability of disconnection between two connected groups of k and (j-k) nodes from the two (n-1)-cubes is zero. For example, let j=5, k=3, and n=3. Because k and (j-k) nodes are connected in the two groups, there must be at least one link that connects the two groups.

The probability of disconnection is non-zero when $j \leq 2^{n-1}$. If we choose j-k connected processors from one (n-1)-cube such that $(j - k) \leq 2^{n-2}$, then the remaining $(2^{n-1} - j + k)$ nodes are also connected, considering no failure. On the other hand, if $(j - k) > 2^{n-1}$, the $(2^{n-1} - j + k)$ nodes are not always connected. In other words,

when more than half of the nodes are connected in a hypercube, the rest of the nodes may be dispersed on the various vertices of the cube without being connected. Hence, if we assume that (j-k) is a small number, $k \geq j - k$, then $(2^{n-1} - j + k)$ nodes will be connected.

Now, if we assume that (j-k) nodes are connected in one (n-1)-cube, then there are $(2^{n-1} - j + k)$ counterpart positions in the second (n-1)-cube which have no direct connection to the j-k nodes of the first group. For example, if $\{5, 7\}$ are the j-k nods in group-2 of Figure 6.3, then nodes $\{0, 2\}$ are not connected to 5, and 7. We will refer to nodes 0 and 2 as the counterparts of nodes 5 and 7. Nodes $\{1, 3\}$ have connection to 5 and 7 directly.

Since we are looking for k and (j-k) nodes to be disconnected, we can guarantee this situation if the k connected nodes are now chosen from the $(2^{n-1} - j + k)$ counterpart positions in the second group. Hence, we can write

$P(C_{n-1}(k, j-k) \neq j) =$

$$\begin{cases} 0 & if\ j > 2^{n-1} \\ \dfrac{\substack{(no\ of\ (j-k)\ processors\ connected) * \\ (no.\ of\ k\ processors\ connected\ from\ (2^{n-1}-j-k)\ connected)}}{\substack{(no\ of\ (j-k)\ processors\ connected) * \\ (no.\ of\ k\ processors\ connected)}} & if\ j \leq 2^{n-1} \end{cases}$$

$$= \begin{cases} 0 & if\ j > 2^{n-1} \\ \dfrac{C_c(2^{n-1} - j + k, k)}{G_c(2^{n-1}, k)} & if\ j \leq 2^{n-1} \end{cases} \qquad (6.10)$$

where $G_c(2^{n-1}, k)$ gives the number of k connected processors from one (n-1)-cube and $C_c(2^{n-1} - j + k, k)$ gives the number of k connected processors from among $(2^{n-1} - j + k)$ connected processors. We determine the disconnected probability by dividing the number of disconnected combinations by the total number of possible connections from among two (n-1)-cubes.

The exact evaluation of the term $C_c(2^{n-1} - j + k, k)$ is extremely difficult. However, after examining several cases, we approximate this term as

$$C_c(2^{n-1} - j + k, k) \approx 2^{n-1} - j + 1 \quad if\ j \leq 2^{n-1} \qquad (6.11)$$

The validity of this approximation will be discussed when we analyze this analytical results in section 5.

Evaluation of the denominator in equation (6.10) is also complex, since a simple $\binom{2^{n-1}}{k}$ does not guarantee connected nodes. We know $P(X_{n-1} = j)$ both for the base model and for higher order models. Hence, we approximate

$$P(X_{n-1} = j) \approx G_c(2^{n-1}, j) R_n(t)^j (1 - R_n(t))^{2^{n-1}-j} \tag{6.12}$$

This is an approximation, since we could also get j connected working nodes from more than j working nodes, as discussed in distributions III and IV.

Finally, if the $k < j < k$ condition is not satisfied, we can change the order of evaluation as $(j - k) > k$ in order to satisfy all values of k from m to M in equation (6.9).

II) $P(C_{n-1} \neq j - k | X_{n-1} = j - k)$

Here we are interested in determining the probability that there are (j-k) processors working in the (n-1)-cube, but all the (j-k) nodes are not connected. This probability can be expressed as

$$P(C_{n-1} \neq j - k | X_{n-1} = j - k) = \binom{2^{n-1}}{j-k} R_n(t)^{j-k} (1 - R_n(t))^{2^{n-1}-j+k} - P(X_{n-1} = j - k) \tag{6.13}$$

The first term in equation (6.13) represents the probability of all possible combinations of (j-k) nodes from among 2^{n-1} nodes. The second term gives the probability that all the (j-k) nodes are connected. Subtracting the second term from the first term, we get the probability when connectivity is not equal to j-k.

III) $P(D_{n-1}(k, j - k) \neq j)$

Here, we are interested in finding the probability that k connected nodes from one (n-1) cube and (j-k) disconnected nodes from the second cube are not j connected. If $k = 2^{n-1}$, this probability is 0, since all the disconnected (j-k) nodes of one group are connected to k nodes of the other group. Also, if j-k=0 or 1, then there is no

disconnection possibility. It is only when $k < 2^{n-1}$ and $(j-k) > 1$ that this probability is non-zero.

Since k nodes are working in one (n-1)-cube, there are $(2^{n-1} - k)$ counterpart positions in the second (n-1)-cube that have no connection with the k nodes of the first group. If we choose s nodes from these positions, then the remaining (j-k-s) nodes must be disconnected from s to satisfy the condition that (j-k) nodes be disconnected. Hence, we write $P(D_{n-1}(k, j-k) \neq j)$

$$= \left\{ \sum_{s=1}^{min(2^{n-1}-k, j-k-1)} \frac{(s \text{ nodes in } (2^{n-1}-k) \text{ positions}) * (j-k-s \text{ nodes from available positions}) * (\text{probability that } s \text{ and } (j-k-s) \text{ are disconnected})}{(\text{no. of } (j-k) \text{ nodes disconnected})} \right.$$

$$= \left\{ \sum_{s=1}^{min(2^{n-1}-k, j-k-1)} \frac{\binom{2^{n-1}-k}{s} \binom{total}{j-k-s} I_{s,j-k-s}}{G_d(2^{n-1}, j-k)} \right. \tag{6.14}$$

where $total = min(2^{n-1} - s*n, k)$ is the number of available positions from which to choose the (j-k-s) positions. We choose the minimum of the two terms, because if $k < (2^{n-1} - s*n)$, there are some positions in $2^{n-1} - s*n$ which are disjoint to k positions in the first group. This argument is based on the fact that k positions in one (n-1)-cube are connected to exactly k positions in the second (n-1)-cube. Hence, if we choose (j-k-s) from $2^{n-1} - s*n$, where $(2^{n-1} - s*n) > k$, there is a possibility that we can have three disjoint groups: k, s, (j-k-s). Since we consider the complete disjoint case in distribution III, these terms should not be included in distribution II. The term $G_d(2^{n-1}, j-k)$ gives the total number of cases where (j-k) nodes are disconnected. This can be obtained from equation (6.13) by the following approximation.

$$P(C_{n-1} \neq j-k | X_{n-1} = j-k) \approx G_d(2^{n-1}, j-k) R_n(t)^{j-k} (1 - R_n(t))^{2^{n-1}-j+k} \tag{6.15}$$

The parameter s can vary from 1 to (j-k-1) so that s and (j-k-s) can be considered as two disconnected groups.

The term $I_{s,j-k-s}$ represents the disconnected probability of s and (j-k-s) nodes. If we choose s nodes from $(2^{n-1} - k)$ counterpart positions, all the neighboring nodes of s can not be selected for disconnected positions. For example, if we choose node 0

as s in Figure 6.3, then connected nodes 1 and 2 to 0 can not be used as disconnected positions. Only node 3 can be selected for (j-k-s). Hence, the number of available disconnected positions must be greater than or equal to (j-k-s) for a connection between s and (j-k-s) to exist. On the other hand, when the number of available disconnected positions is less than (j-k-s), some of the connected positions of s will be used for (j-k) positions. In this case, there is no disconnection. We write this function as

$$I_{s,j-k-s} = \begin{cases} 0 & if \quad 2^{n-1} - s*n < j-k-s \\ 1 & if \quad 2^{n-1} - s*n \geq j-k-s \end{cases} \tag{6.16}$$

It should be observed that for a given s, s*n positions are not available for disconnection where n is the size of the cube. This leaves only $(2^{n-1} - s*n)$ position in the (n-1)-cube for choosing the remaining (j-k-s) positions. $2^{n-1} - s*n \geq (j-k-s)$ assures that s and j-k-s are disconnected.

IV) $P(C_n = j | X_n = k) \quad , \quad j \leq k \leq 2^{n-1}$

Let us assume that there are s connected processors working in group-1 and (k-s) processors working group-2. Out of the (k-s), only (j-s) are connected to group-1, and (k-j) are disconnected from group-1. The upper value of s is given by $U = min(2^{n-1}-1, j-1)$, since if $s = 2^{n-1}$, all the working nodes from group-1 are connected to all the (k-s) working nodes of group-2. If s=j, or j=1 then all the working nodes are in one group. This distribution then becomes identical to distribution III. Hence, the lesser of $2^{n-1} - 1$ and j-1 is the upper bound for one working group. The lower bound for s is given by $L = max(1, k-(2^{n-1}-1))$, since no more than $(2^{n-1} - 1)$ nodes can be working in the second group if the two groups are disconnected. This leaves us with $k - (2^{n-1} - 1)$ nodes to start within the first group. Therefore, the minimum of 1 and $k - (2^{n-1} - 1)$ gives the lower bound for s.

Now, the situation is that s connected nodes are in group-1, (k-s) disconnected nodes are in group-2 and the connectivity between the two groups is j. This is expressed as

$$P(C_n = j | X_n = k) = 2 \sum_{s=L}^{U} P(X_{n-1} = s) P(C_{n-1} \neq k-s | X_{n-1} = k-s)$$

$$P(D_{n-1}(s,k-s) - j)) \tag{6.17}$$

The first two terms of equation (6.17) are already known. We have to evaluate only the third term. This can be derived using the same argument as for term III, since $P(D_{n-1}(s,k-s) \neq k)$ can be written as the summation of all $P(D_{n-1}(s,k-s)=i)$ terms for $s < i < k$. However, we are interested only in j connections between s and k-j.

Since there are s processors working in the first group, there are $(2^{n-1}-s)$ positions in the second group that are not connected to those s positions. As (k-j) positions are always disconnected from j positions, we choose (k-j) from $(2^{n-1}-s)$. The number of ways in which we can choose (k-s) disconnected nodes from 2^{n-1} is given by $G_d(2^{n-1}, k-s)$. We then write the third term in equation (6.17) as

$$P(D_{n-1}(s,k-s)=j) = \frac{\binom{2^{n-1}-s}{k-j}\binom{total1}{j-s} I_{k-j,j-s}}{G_d(2^{n-1},k-s)} \tag{6.18}$$

where $total1 \quad min(2^{n-1} - (k-j)*n, s)$ is the number of available positions to choose the remaining (j-s) positions which are connected to the s positions of the first group. The term total1 is computed based on the same argument as for the term total in equation (6.14). $I_{k-j,j-s}$ gives the probability that there is no connection between (k-j) and (j-s) nodes in the second group. If we can guarantee that there is no connection between (k-j) and (j-s) nodes, then we have j connected nodes from k. This is because the (j-s) nodes in the second group are selected such that there is always some connection to the first group s nodes. More specifically, for each s positions in one (n-1)-cube, there are s positions in the second (n-1)-cube which are connected.

The term $I_{k-j,j-s}$ is given from equation (6.16) as

$$I_{k-j,j-s} \quad \begin{cases} 0 & if\ 2^{n-1} \quad (k-j)*n < j-s \\ 1 & if\ 2^{n-1} \quad (k-j)*n \geq j-s \end{cases} \tag{6.19}$$

Equation (6.19) shows that when the number of disconnected positions are less then required (j-s), the disconnection probability is 0. Otherwise (j-s) and (k-j) are always disconnected.

All the terms in equation (6.9) are now quantified to derive $P(X_n = j)$. The system reliability can be computed from equation (6.4).

We expect that equation (6.9) will give fairly accurate results when the value of j is close to N/2, since most of the working nodes are connected when j is large. Equations (6.12) and (6.15) give better approximation in this case. When j is less than N/2, the probability of a larger number of disconnected working nodes increases. Equations (6.12) and (6.15) deviate more from the reality in this situation.

6.3.3 Modified Method

We can divide equation (6.9)into two parts depending on the actual number of working nodes. The first two terms of equation (6.9) give the probability of j connected nodes when exactly j node are working in the system. The second two terms give j connected nodes from k working nodes where $k > j$. Hence, we can rewrite $P(X_n = j)$ as

$$P(X_n = j)^* = P(X_n = j) + \sum_{k=j+1}^{N} P(C_n = j | X_n = k)^* \tag{6.20}$$

We evaluate the first term of equation (6.20) for all the j connected nodes. This is computed recursively from the first two terms of equation (6.9). After expanding the first two terms of equation (6.9) and simplifying, we get

$$\begin{aligned} P(X_n = j) = & \sum_{k=m}^{M} P(X_{n-1} = k) G(2^{n-1}, k-j, p) \\ & - \sum_{k=m}^{M} P(X_{n-1} = j-k) C_c(2^{n-1} - j + k, k) R_n(t)^k (1 - R_n(t))^{2^{n-1}-k} \\ & - \sum_{k=m}^{M} P(X_{n-1} = k) P(D_{n-1}(k, j-k) \neq j) G_d(2^{n-1}, j-k) \\ & * R_n(t)^{j-k} (1 - R_n(t))^{2^{n-1}-j+k} \end{aligned} \tag{6.21}$$

The second term of equation (6.20) is computed assuming that all the disconnected nodes are present only one level lower than the n-cube, i.e., the computation is done

at the (n-1)-cube level. It does not go recursively down to the base model. We write this as

$$P(C_n = j\ X_n = k)^* = P(C_n = j|X_n = k) + 2P(X_{n-1} = j)\binom{2^{n-1}-j}{k-j} R_n(t)^{k-j}(1 - R_n(t))^{2^{n-1}-k+j} \quad (6.22)$$

The first term of equation (6.22) is the same as equation (6.17). However, this is required only at the (n-1)-cube level. The second term is for distribution III, given by equation (6.7). This term is valid for $j < 2^{n-1}$ and $k \leq 2^{n-1}$.

Since all the terms required for equation (6.22) at the (n-1)-cube level, are available from the previously computed terms of equation (6.21), this modified method is faster than equation (6.9).

6.4 Results and Discussion

The original equation (6.9) or the modified equation (6.20) can be used to compute $P(X_n = j)$. Most of this results are based on a node failure rate of 250 in 10^6 hours; i.e. $\lambda = 0.00025$. While analytical results for hypercubes of any size can be computed using this model, we are including here a few results because of space limit. Figure 6.4 shows the reliability variation with time of a 6-dimensional hypercube for different task requirements. The solid curves are for the analytical results using the modified technique. The dotted curves are for simulation results. The analytical results are computed using 2-cube base model. However, the results are almost the same for both the 2-cube and 3-cube. It can be observed from Figure 6.4 that the analytical and simulation results match closely for I=48, 32, and 16.

Figure 6.5 shows the reliability variation of a 7-cube system under different task requirements. The analytical and simulation curves match closely for 25%, 50%, and 75% node degradation (I=96, 64, 32). The difference between the analytical and simulation results is less than 6%. In Figure 6.6, the reliability variation of 8-cube system for I=64, 128, and 192 is given. The results match very closely with that of simulation.

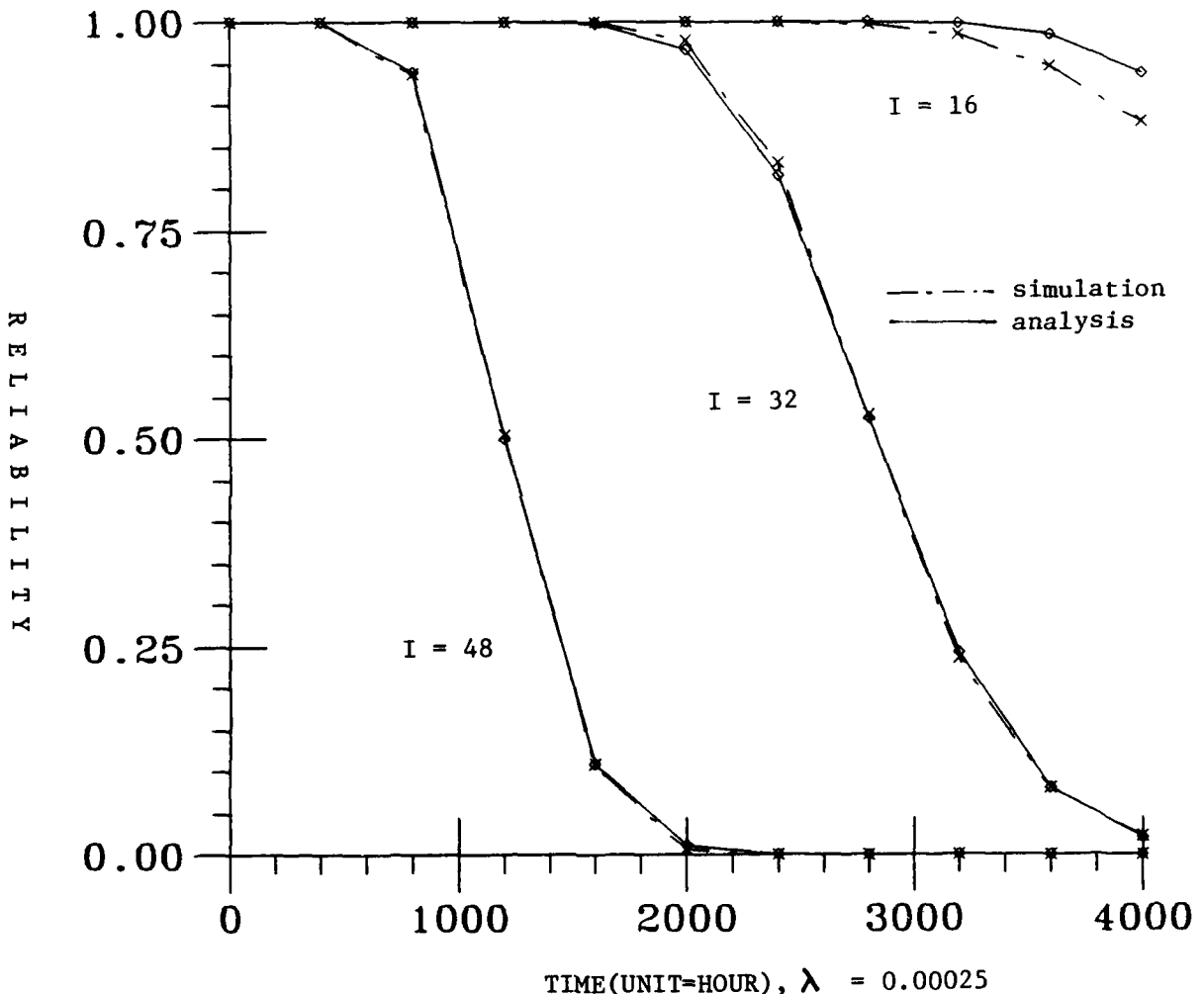


Fig.6.4 6-cube Reliability Comparison for a Task requiring I processors

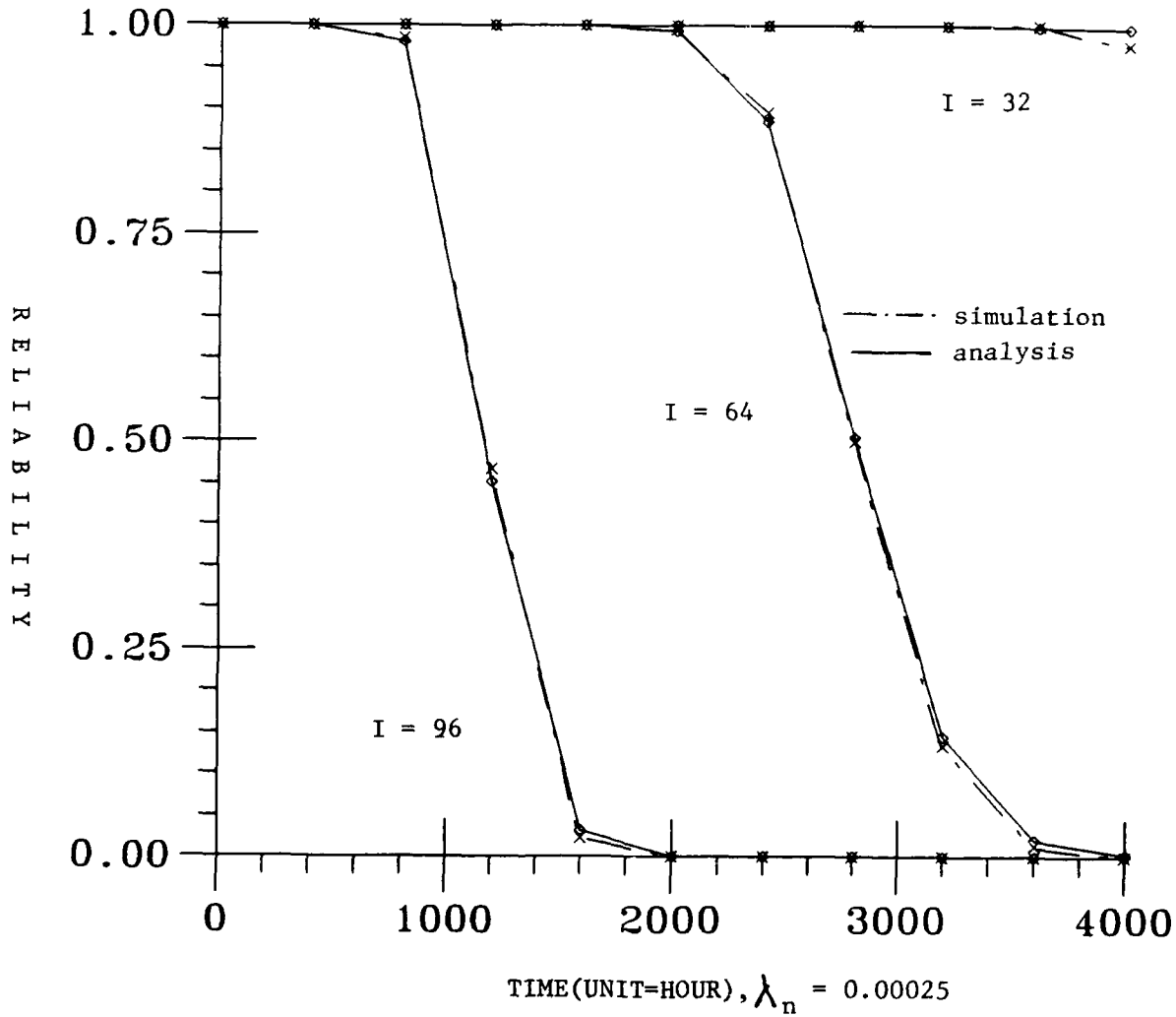


Fig.6.5 7-cube Reliability Comparison for a Task requiring I processors

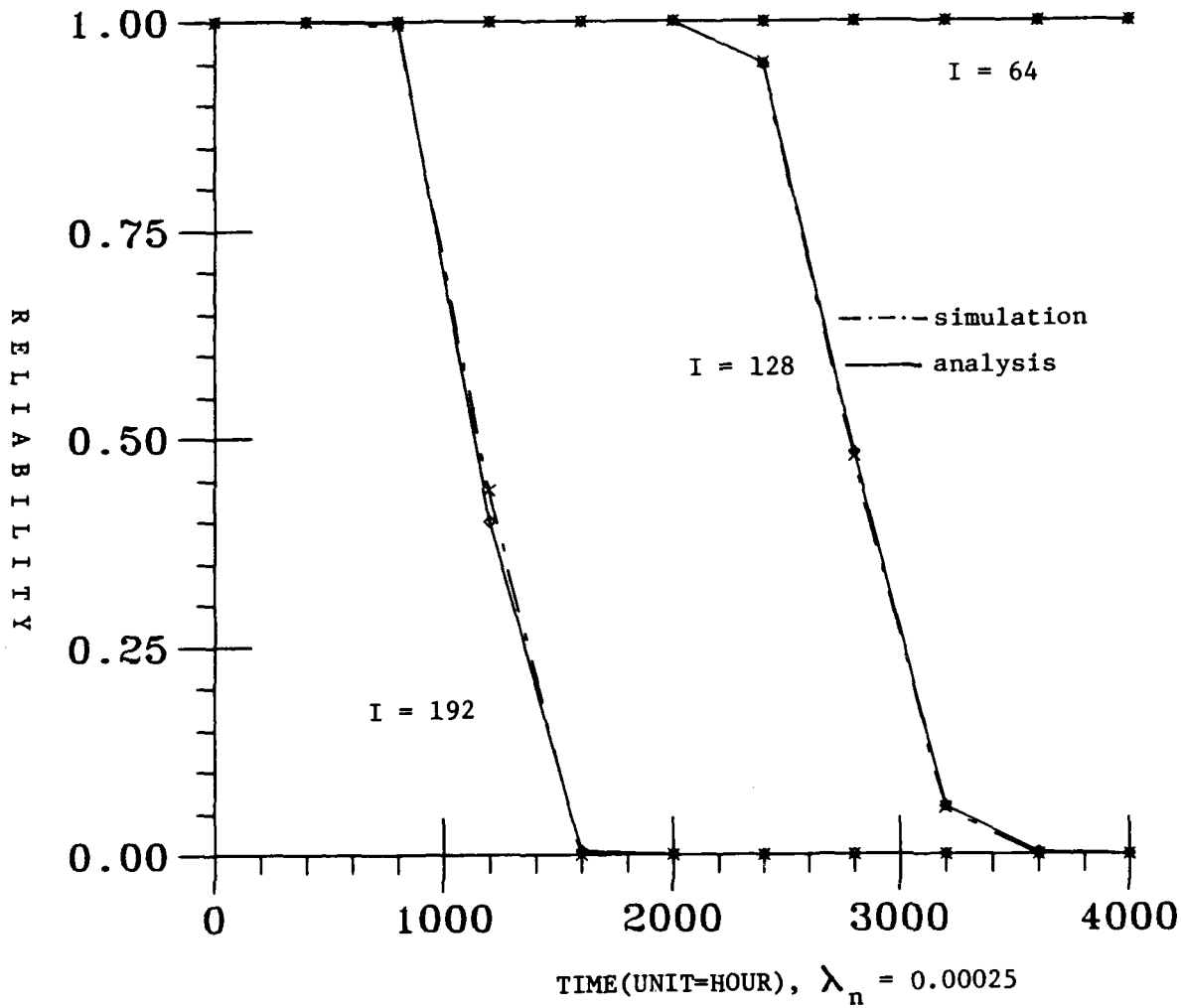


Fig.6.6 8-cube Reliability Comparison for a Task requiring I processors

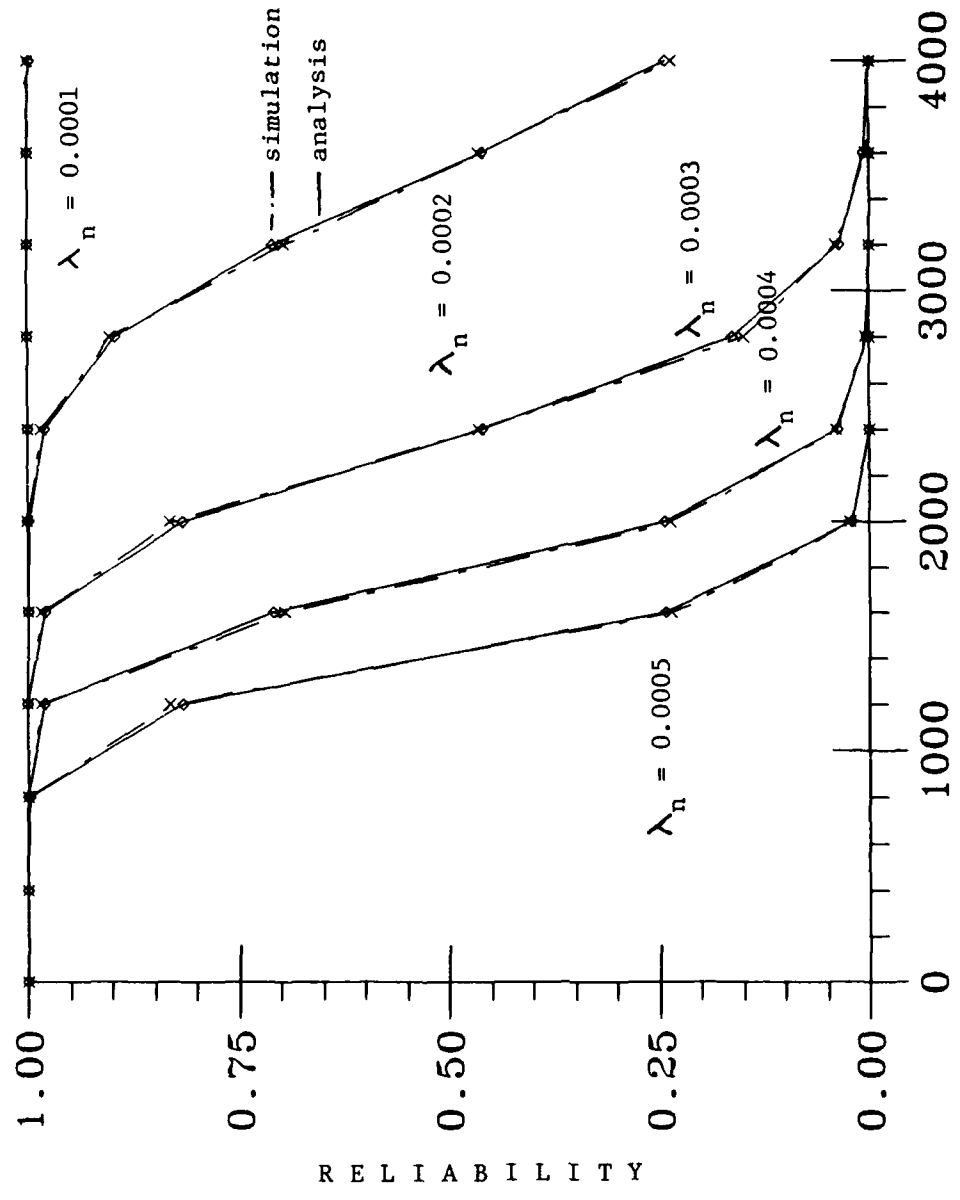


Fig.6.7 6-cube Reliability Comparison with Different Failure Rate

In Figure 6.7, the validity of the approximate technique for different node failure rate is analyzed for a 6-cube system with at least 32 connected nodes. It is observed that results from the approximate technique agree with simulation results for all λ_n values.

CHAPTER 7
CONCLUSIONS

This report is intended to summarize our research efforts in evaluating parallel architectures for BM/C^3 applications. The efforts are focused on defining evaluation criteria, developing tools and performing analysis in determining the suitability of existing parallel computer architectures to provide optimal processing environments for the BM/C^3 applications. Our research work consists of three major components: (1) development of a BBN Butterfly performance predictor, (2) mapping of the Battle Management Algorithm to the Butterfly Parallel Computer, and (3) performance and dependability evaluators for the Butterfly parallel computer and the Hypercube multiprocessor.

The main component of this Butterfly Performance Predictor is a program that simulates program execution on a Butterfly while accumulating performance measures. Performance estimates are made based on instruction execution rate information derived from Motorola data books relating to the basic hardware components of the Butterfly computer. To derive the simulator under conditions representative of the target application domain, a second program generates synthetic instruction streams that are representative of the target application domain. The first version of the Butterfly simulator is under development in the programming language C on a SUN 3/50 workstation running 4.2BSD UNIX.

The Battle Management Algorithm, formulated as a linear programming algorithm, is a problem that in nature requires intensive interprocess communication for simultaneous process execution. Our attempt is to minimize the contention costs both in shared memories and in communication links by setting up a tree-shape communication structure among processor nodes. The tree-shape communication structure is used for searching a minimum value in one computational phase, and for broadcasting

it in another computational phase. The proposed method is also applicable to other algorithms with similar data-flow graphs. Therefore, with the advantages of minimizing contention costs, the algorithm-based method leads to a new technique for mapping algorithms onto to-date parallel processors. Evaluation of the parallel algorithms is accomplished by a simplified mathematical analysis for approximately predicting the system performance. The result indicates that if the proposed method is used to map the linear programming algorithms of size n onto the $Butterfly^{TM}$ parallel processor, an $\frac{O(n^2)}{O(nlog_2 n)}$ speedup can be achieved.

The novelty of the reliability evaluation of the Butterfly network mainly lies in the use of an analytical model for precise definition and accurate analysis. Reliability evaluation of multiprocessor systems using Butterfly type network is addressed. The Butterfly network is a multistage network designed out of 4x4 switches. The novelty of the evaluation technique is the development of an analytical model for reliability computation. The model is based on a decomposition technique. Using this technique, the reliability of a 64 processor and 64 memory configuration, (64x64), is computed from four (16x16) system reliability. The (16x16) reliability in turn is computed from four (4x4) reliability. The reliability model is known as task based reliability, where a system remains operational as long as a task can be executed on the system. The failure of the PEs, MMs, and SEs are included in the analysis to consider a complete system. The model is suitable for the analysis of medium size systems, such as (16x16) and (64x64) multiprocessors. While a (256x256) configuration could be analyzed using our model, the computation time is a major concern unless some approximation technique is used to simplify the switch connection. We are currently investigating in applying approximate methods to improve the computation efficiency.

We have also devised a new analytical technique to compute the reliability of an n-dimensional hypercube. The model is based on the decomposition principle. A recursive equation is derived to compute n-cube reliability based on 2-cube or 3-cube base models. The model is developed by considering four different situations, where the required number of connected nodes are working on the system. Analytical results

for various hypercubes are compared with simulation results to show that they are in close agreement. Several extensions of this model are presently under investigation. The immediate extension is to apply this model to repairable hypercubes to compute transient and steady state availability. Also, inclusion of link failure in the base model and between two base models should allow us to consider both the node and link failures.

Performance and dependability evaluations are essential for any system characterization. Two types of parallel systems, namely; Butterfly and Hypercube are studied here. It is pointed out that there is no existing tool available for the performance evaluation of these machines considering both the architecture and application algorithms. Similarly none of the existing dependability packages can be applied to the above systems directly. These observations clearly dictate the necessity of developing evaluation tools for these architectures. Preliminary results of the Butterfly performance predictor, Butterfly and Hypercube reliability tools are included in this report.

REFERENCES

[Alexander 75]
Alexander, W.G. and D.B. Wortman, "Static and Dynamic Characteristics of XPL Programs," *IEEE Computer*, August 1975, pp. 41-46.

[Allik 87]
Allik, H., S. Moore, E. O'Neil and E. Tenenbaum, "Finite Element Analysis on a Shared-Memory Multiprocessor," 1987 International Journal of Computers and Structures.

[Arlat 83]
Arlat, J. and J.C. Laprie, "Performance-Related Dependability Evaluation of Supercomputer Systems," Proc. 13th FTCS, June 1983, pp. 276-283.

[Avizienis 78]
Avizienis, A., "Fault-Tolerance; The Survival Attibute of Digital Systems," Proc. IEEE, October 1978, pp. 1109-1125.

[Bavuso 87]
Bavuso, S.J., J.B. Dugan et. al., "Analysis of Typical Fault-Tolerant Architectures using HARP," IEEE Trans. on Reliability, Vol. R-36, June 1987, pp. 176-185.

[Beaudry 78]
Beaudry, M.D., "Performance Related Reliability Measures for Computing Systems," IEEE Transactions on Computers, Vol. C-27, Jan. 1978, pp. 540-547.

[Bhuyan 84]
Bhuyan, L.N. and D.P. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network," IEEE Transactions on Computers, Vol. C-33, April 1984, pp. 323-333.

[Blake 87]
Blake, J., and K. Trivedi, "Multistage Interconnection Network Reliability," Submitted to IEEE Transactions on Computers.

[Brookes 82]
Brookes, G.R., and I.R¿ Wilson, "A Static Analysis of Pascal Program Structures," Software Practice and Experience, Vol. 12, 1982, pp. 959-963.

[Crowther 85]
Crowther, W., J. Goodhue, E. Starr, R. Thomas, W. Milliken, and T. Blackadar, "Performance Measurements on a 128-Node *Butterfly*^{TM} Parallel Processor," 1985 Int'l Conf. on Parallel Processing, pp. 531-540.

[Das 85]

Das, C.R. and L.N. Bhuyan. "Bandwidth Availability of Multiple-bus Multiprocessors," IEEE Trans. on Computers, Special Issue on Parallel Processing, Oct. 1985, pp. 910-926.

Das 87
Das, C.R. and L.N. Bhuyan, "Dependability Evolution of Interconnection Networks," Information Sciences, and international Journal, Special Issue on Parallel Processing, October 1987.

Das 88
Das, C.R., J. Kim et. al., "An Analytical Model for Computing Hypercube Availability," Submitted to 18th FTCS.

Deprycker 82
Deprycker, M., "On the Development of a Measurement System for High Level Language Program Statistics," IEEE Trans. on Computers, Vol. C-31, No. 9, pp. 883-891, 1982.

Derosa
Derosa, J.A., and H.M. Levy, "An Evaluation of Branch Architecture," Proceedings of the 14th Annual International Symposium on Computer Architecture, 1987.

Dias 81
Dias, D.M., and J.R. Jump, "Analysis and Simulation of Buffered Detla Networks," IEEE Trans. on Computer, Vol. C-30, April 1981, pp. 273-282.

Ditzel 80
Ditzel, D.M., "Program Measurements on a High Level Computer," IEEE Trans. on Computers, 1980, pp. 62-72.

Elshoff 76a
Elshoff, J.L., "A Numerical Profile of Commercial PL/1 Programs," Software Practice and Experience, Vol. 6, 1976, pp. 505-526.

Elshoff 76b
Elshoff, J.L., "An Analysis of some Commercial PL/1 Programs," IEEE Trans. on Software Engineering," Vol. SE-2, pp. 113-120, 1976.

Foster 71
Foster, C., and R. Gonter, "Conditional Interpretation of Operation Codes," IEEE Trans. on Computers, Vol. C-20, No. 1, pp. 108-111, 1971.

Gay 79
Gay, F.A. and M.L. Ketelsen, "Performance Evaluation at Gracefully Degrading Systems," Proc. FTCS-9, Madison, June 1979, pp. 51-57.

Goyal 87
Goyal, A., W.C. Carter, et. al., "The System Availability Estimator," Proc. 16th FTCS June 1986, pp. 84-89.

[Hwang 82]
Hwang, k. and T.P. Chang, "Combinatorial Reliability Analysis at Multiprocessor Computers," IEEE Trans. on Reliability, Vol. R-31, Dec. 1982, pp. 469-473.

[Ingle 77]
Ingle, A.D. and D.P. Siewiorek, "Reliability Models for Multiprocessor Systems With and Without Periodic Maintenance," Proc. 7the FTCS, June 1977, pp. 3-9.

[Kim 88]
Kim, J., C.R. Das, et. al., "Reliability Evaluation of Hypercube Multicomputers," submitted to 8th ICDS.

[Knuth 71]
Knuth, D.E., "An Empirical Study of Fortran Programs," Software Practice and Experience, Vol. 1, pp. 105-133.

[Kruskal 83]
Kruskal, C.P. and M Snir, "The Performance of Multistage Interconnection Networks for Multiprocessors," IEEE Transactions on Computers, Vol. C-32, Dec. 1983, pp. 1091-1098.

[Lang 82]
Lang, T. et. al., "Bandwidth of Crossbar and Multibus Connection for Multiprocessors," IEEE Trans. on Computers, Vol. C-31, December 1982, pp. 1227-1234.

[Laprie 82]
Laprie, J.C. and A. Costes, "Dependability: A Unifying Concept for Reliable Computing," Proc. FTCS-12, June 1982, pp. 18-21.

[LeBlanc 86]
LeBlanc, T.J., "Shared Memory Versus Message Passing in a Tightly Coupled Multiprocessor: A Case Study," 1986, Int'l Conf. on Parallel Processing, pp. 463-466.

[Lee 86]
Lee, G., C.P. Kruskal and D.J. Kuck, "The Effectiveness of Combining in Shared Memory Parallel Computers in the Presence of Hot Spots," 1986, Int'l Conf. on Parallel Processing, pp. 35-41.

[Lee 84]
Lee, J.K.F., and A.J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," IEEE Computer, January 1984.

[Lin 88]
Lin, W. and C.L. Wu, "A Distributed Resource Management Mechanism for a Partitionable Multiprocessor System," to appear, IEEE Transactions on Computers, January 1988.

[Liu 68]
Liu, C.L., Introduction to Combinatorial Mathematics, Computer Science Series, 1968, chapter 12.

[Makam 82]
Makam, S.V. and A. Avizienis. "ARIES 81: A Reliability and Life Cycle Evaluation Tool for Fault Tolerance Systems." Proc. 12th FTCS June 1982, pp. 269-274.

[Marsan 82]
Marsan, M.A. and M. Gerla, "Markov Models for Multiple-bus Multiprocessors," IEEE Trans. on Computers, Vol. C-31. March 1982, pp. 239-248.

[McFarling 86]
McFarling, S. and J. Hennessy, "Reducing the Cost of Branches," Proceedings of the 13th Annual International Symposium on Computer Architecture, 1986.

[Meyer 80]
Meyer, J.F., "On Evaluating the Performability of Degradable Computing Systems." IEEE Trans. on Computers, Vol. C-29, Aug. 1980, pp. 720-731.

Mudge 84
Mudge, T., J.P. Hayes, G.D. Buzzard, and D.C. Winsor, "Analysis of Multiple-Bus Interconnection Networks," Proc. Int. Conf. on Parallel Processing, Aug. 1984, pp. 228-232.

[O'Neil 87]
O'Neil, E., E. Tenenbaum, H. Allik and S. Moore, "Finite Element Analysis on the BBN *Butterfly*^{TM} Multiprocessor," 1987 The Second International Conference on Supercomputing, Santa Clara, CA.

[Pfister 85]
Pfister, G.F. and V.A. Norton. ""Hot Spot" Contention and Combining in Multistage Interconnection Networks," IEEE Trans. on Computer, Vol. c-34, No. 10, Oct. 1985, pp. 943-948.

[Provan 86]
Provan, J.S., "Bounds on the Reliability of Networks," IEEE Trans. on Rel., Vol. 4-35, Aug. 1986, pp. 228-232.

[Raghavendra 84]
Raghavendra, C.S. and D.S. Parker, "Reliability Analysis of an Interconnection Network," Proc. 4th ICDS, May 1984, pp. 461-471.

[Reed 87]
Reed, D.A. and D.C. Grunwald, "The Performance of Multicomputer Interconnection Networks," IEEE Computer, June 1987, pp. 63-73.

[Sahner 87]
Sahner, F.A. and K.S. Trivedi, "Reliability Modeling Using SHARPE," IEEE Trans. on Reliability, Vol. R-36, June 1987, pp. 63-73.

[Sheu 88]
Sheu, T.L. and W. Lin, "Mapping Linear Programming Algorithms onto The Butterfly Parallel Processor," Submitted to 1988 Third International Conference on Supercomputing.

[Siewiorek 82]
Siewiorek, D.P. and R.S. Swartz, "The Theory and Practice of Reliable System Design," Digital Press, 1982.

[Smith 81]
Smith, J.E., "A Study of Branch Prediction Strategies," Proceedings of the 8th Annual International Symposium on Computer Architecture, 1981.

[Stiffler 82]
Stiffler, J.J. and L.A. Bryant, "CARE III Phase III Report- Mathematical Description," NASA Contractor Report 3566, November 1982.

[Tanenbaum 78]
Tanenbaum, A.S., "Implications of Structured Programming for Machine Architecture," CACM, Vol. 21, pp. 237-246, 1978.

[Tien 88]
Tien, L., C.R. Das et. al., "Reliability Evaluation of Butterfly Networks Based Multiprocessor Systems," Submitted to 8th ICDS.

[Tomas 86]
Tomas, R.H., "Behavior of the *Butterfly*^{TM} Parallel Processor in the Presence of Memory Hot Spots," 1986 Int'l Conf. on Parallel Processing, pp. 46-50.

[Wiecek 82]
Wiecek, C.A., "A Case Study of VAX-11 Instruction Set Usage for Compiler Execution," ACM, 1982, pp. 177-184.

[Wittie 81]
Wittie, L.D., "Communication Structures for a Large Multimicrocomputer Systems," IEEE Trans. on Computers, Vol. C-30, April 1981, pp. 264-293.

[Wu 84]
Wu, C.L. and T.Y. Feng, "A Tutorial on Interconnection Networks for Parallel and Distributed Processing," IEEE, 198D4.

[Yew 87]
Yew, P., N. Tzeng and D.H. Lawrie, "Distributing Hot-Spot Addressing in Large-Scale Multiprocessors," IEEE Trans. on Computer, Vol. c-36, No. 4, April 1987, pp. 388-395.

☆ U.S. Government Printing Office 1988 511-117/64100