NAG 1-613

August 1990

UILU-ENG-90-2231
CRHC-90-3

Center for Reliable and High-Performance Computing

P-136

ANALYSIS AND DESIGN OF ALGORITHM-BASED FAULT-TOLERANT SYSTEMS

V. S. Sukumaran Nair

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS None	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-90-2231 (CRHC-90-3)		5. MONITORING ORGANIZATION REPORT NUMBER(S) NASA NAG 1-613	
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois	6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION NASA	
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801		7b. ADDRESS (City, State, and ZIP Code) NASA Langley Research Center, Hampton, VA 23665 and Arlington, VA 22217	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION NASA	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code) 7b.		10. SOURCE OF FUNDING NUMBERS	

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
"Analysis and Design of Algorithm-Based Fault-Tolerant Systems"

12. PERSONAL AUTHOR(S)
NAIR, V. S. SUKUMARAN

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM ______ TO ______	1990 August	135

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	fault-tolerance, concurrent error detection, ABFT, Matrix-based model, detectability, fault locatability, FTMP systems hierarchical design

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

An important consideration in the design of high performance multiprocessor systems is to ensure the correctness of the results computed in the presence of transient and intermittent failures. Concurrent error detection and correction have been applied to such systems in order to achieve reliability. Algorithm Based Fault Tolerance (ABFT) has been suggested as a cost-effective concurrent error detection scheme.The research reported in this thesis has been motivated by the complexity involved in the analysis and design of ABFT systems. To that end, a matrix-based model has been developed and, based on that, algorithms for both the design and analysis of ABFT systems are formulated. These algorithms are less complex than the existing ones. In order to reduce the complexity further, a hierarchical approach is developed for the analysis of large systems.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

ANALYSIS AND DESIGN OF ALGORITHM-BASED FAULT-TOLERANT SYSTEMS

BY

V. S. SUKUMARAN NAIR

B.Sc. Engg., University of Kerala, 1984
M.S., University of Illinois, 1988

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1990

Urbana, Illinois

ANALYSIS AND DESIGN OF ALGORITHM-BASED FAULT-TOLERANT SYSTEMS

V. S. Sukumaran Nair
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, 1990

An important consideration in the design of high performance multiprocessor systems is to ensure the correctness of the results computed in the presence of transient and intermittent failures. Concurrent error detection and correction have been applied to such systems in order to achieve reliability. Algorithm Based Fault Tolerance (ABFT) has been suggested as a cost-effective concurrent error detection scheme. The research reported in this thesis has been motivated by the complexity involved in the analysis and design of ABFT systems. To that end, a matrix-based model has been developed and, based on that, algorithms for both the design and analysis of ABFT systems are formulated. These algorithms are less complex than the existing ones. In order to reduce the complexity further, a hierarchical approach is developed for the analysis of large systems.

DEDICATION

To my Parents and Brothers

And to the memory of my Uncle K. N. Pillai

ACKNOWLEDGEMENTS

I am deeply grateful to my thesis advisor, Professor Jacob A. Abraham, for his patient guidance and helpful suggestions. His encouragement, concern, and insight in academic as well as nonacademic matters were invaluable sources of support throughout the course of this work. I would also like to thank Professors Prithviraj Banerjee, Ravishankar K. Iyer, W. Kent Fuchs, and C. L. Liu for being members of my dissertation committee and for their time and support. I gratefully acknowledge Robert Mueller-Thuns and Professor Daniel G. Saab for many interesting discussions and helpful suggestions. The friendship of Madhav Desai, Rabindra Roy, Subhodev Das, and Abbas Butt deserves special mention. I am also thankful to my colleagues and friends in the Center for Reliable and High Performance Computing (CRHC) at the Coordinated Science Laboratory. A big thank you to: Biju, Leena, James, Kunjumol, Thomas Panthaplam, G'mon, Thomas, Abe, and Manoj Franklin for making me feel at home away from home. Finally, I would like to thank my parents and brothers for their everlasting love and support which made this thesis a reality.

This research was supported by the National Aeronautics and Space Administration (NASA) under Contract NAG 1-613 at the University of Illinois.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1.

INTRODUCTION

1.1. Fault-Tolerant Multiprocessor Systems

Multiprocessing has become a viable alternative to serial computing to meet the high-performance requirements in various scientific, engineering, medical, military, and basic research areas. High speed of computation, high throughput, large volumes of processed data, and long periods of reliable operation are some of the common requirements in most of these applications. With the help of modern VLSI technology, complex processor chips containing up to 10^6 transistors have been designed and marketed to meet the high computation requirements.

Unfortunately, performance and reliability are two contradicting requirements. As the rate of computation increases, the probability of an error in the computed result also increases. There are various reasons for this. First of all, the complexity of the processor increases with its computation capability; it has been observed that the failure rate increases exponentially with the complexity of the chip [1]. Another observation in this regard is that as the computation and the communication load increase, the failure rate in the system also increases [2]. (Note that an increased computation rate has to be supplemented with increased communications between the processors.)

Long periods of reliable computing are necessary in areas such as medical instrumentation where a failure may lead to fatalities. Another scenario may be where the system is inaccessible for repair; for instance, a space satellite, unattended after its launch, is expected to deliver accurate data from space for a long period of time. To meet these acute reliability requirements, the computer should be able to withstand failures.

Two methods have been suggested for handling failures in an electronic system: fault avoidance and fault tolerance [3]. In fault avoidance, the system tries to evade faults by design as well as by protection against fault inducing environments. However, it is applicable only when there is an a priori knowledge of all the possible faults. Quite often that is not the case. Furthermore, the cost involved in fault avoidance techniques is high. Therefore, fault tolerance has been accepted as the cost effective choice.

Two approaches to achieve fault tolerance have been the static or masking redundancy techniques and the dynamic redundancy techniques. In the former, failures are tolerated by masking their effects; triplication and voting [4], duplication and comparison [5], and quadded logic [6] are some examples. In the dynamic redundancy approach, first the presence of a fault is detected and then a corrective action is taken in the form of replacing the failed unit, recomputing the result, or reconfiguring the system to isolate the faulty module from the rest of the system. Systems with dynamic redundancy are preferred to systems with static redundancy due to their greater mean lifetime gains, greater isolation against catastrophic faults, ability to survive until all spares are exhausted, and their potential to utilize the lower failure rate of the redundant (usually unpowered) unit. However, the fault tolerance capabilities of the system are highly dependent on the quality of the fault detection and recovery schemes.

Various recovery schemes, especially reconfiguration schemes, have been studied extensively in the past [7, 8, 9, 10]. The area of fault detection seems to be less attended. One observes that fault detection is a more difficult problem than the reconfiguration problem. With the potential of microelectronic technology to provide more redundant processing nodes along with sophisticated switching networks interconnecting them, reconfiguration and replacement have become less complex issues. In contrast, detection of a fault in the system has become all the more complicated due to the complex interaction between the component processors. In order to harness fully the fault tolerance potentials of modern VLSI architecture, one must have efficient and high quality fault detection schemes. The main theme of discussion in this thesis is the detection of faults in multiprocessor systems.

A fault can be detected either by off-line checking or by concurrent checking. In the first method, the system is brought off-line and checked for the presence of faults. Even though this approach has the advantage that it does not affect the real-time performance of the system, its application is limited since it can detect only permanent faults. Unfortunately, studies show that [11] more than 85% of major system failures are transient in nature. Furthermore, a strong relationship has been observed between the occurrence of transients and the level of system activity. Therefore, it becomes imperative to check for faults in a system while it is in operation. The current trend is to include Concurrent Error Detection (CED) capability in the design of digital systems.

1.2. Concurrent Error Detection (CED)

Traditionally, systems with CED are implemented using self-checking circuits [12] or by hardware duplication and comparison of their results [5]. Self-checking circuits are specially designed to operate on data elements encoded using error-detecting codes. Duplication of circuits can be considered as a special type of self-checking circuits that employ the duplication code. Since these traditional techniques require 200 to 300% hardware redundancy, they are usually very expensive. This puts the pressure on the system designer to come up with cost effective schemes.

The quality of CED techniques depends heavily upon the level at which checking is implemented: the gate, functional or system level. Gate level techniques such as those using error detecting/correcting codes usually assume the conventional stuck-at fault model. Studies show, however, that there are faults which cannot be covered by the stuck-at fault model [13]. Further, due to the shrinking device dimensions, a physical defect affecting a small local area of a chip can result in faults in several gates. This points to the need for a higher-level fault model instead of the stuck-at fault model.

Algorithm-based fault tolerance (ABFT), proposed by Huang and Abraham [14], is a fault tolerance scheme that uses CED techniques at a functional level. System level applications of ABFT techniques have also been investigated [15]. These techniques assume a general fault model which allows any single module in the system to be faulty [14]. Even though the faults are modeled at a high level, they cover all the lower-level stuck-at faults; also the techniques are independent of the logic design and the type of the IC used.

ABFT is widely applicable and it has proved its cost-effectiveness especially when applied to array processors [16]. A detailed description of ABFT techniques may be found in Chapter 2. The objective of this thesis is to develop efficient analysis and design algorithms for ABFT systems.

1.3. Previous Research

The problem of locating faulty processors within a multiple processor system by temporarily halting normal operation and placing the system in a diagnostic mode has originally been studied using the PMC model [17] which assumed that the processors can individually test other processors. A test may be any sort of check by one processor on the operation of some other, including applying test vectors and checking the resulting outputs. On the basis of the test responses, the test outcome is classified as "pass" or "fail." The test evaluation is always accurate if the testing unit is fault-free.

The PMC model is limited to systems in which each unit alone can test some other units; also, different failure rates for the units in the system are not characterized. Russel and Kime generalized this model by broadly interpreting the concepts of faults and test [18, 19]. In this model, a complete testing of a unit requires combined operation of more than one unit. An algebraic approach to digital system fault diagnosis was suggested by Adham and Friedman [20]. Here, a set of fault patterns is described by a Boolean expression. To be applied to large systems, this approach requires tools for efficiently manipulating Boolean expressions containing large number of variables. Another generalization of the PMC model has been suggested by Maheswari and Hakimi [21]. Their model incorporates the probabilistic nature of fault occurrence. This model was further extended by Fujiwara and Kinoshita [22].

The analysis of ABFT systems is much harder than the analysis of systems considered in the above mentioned studies. In the PMC model and in its generalizations, researchers assume that complete tests are available for individual processors [18, 19]. That is, if the tested unit is faulty and the tester is fault-free, then the test is guaranteed to fail. However, in systems using ABFT, a particular fault pattern can produce a number of different error patterns. The checking operations detect the errors directly and the faults indirectly. Since the error detectability of the checks is finite, even if the check evaluating processor is fault-free, a fault in the checked unit may be undetected if the number of errors caused by that fault is larger than the error detectability of the check. (We denote these kinds of checks as incomplete checks.) Therefore, fault analysis in such systems is much more complex than conventional fault analysis. It may be observed that the systems using incomplete checks are supersets of systems using complete checks. This is because a complete check can be viewed as an incomplete check with infinite error detectability.

The first attempt towards modeling ABFT systems was made by Banerjee and Abraham [23] who proposed a graph-theoretic model. In this model, the system is represented by a tripartite graph having three groups of nodes: nodes of type F corresponding to the possible faulty processors, nodes of type E corresponding to the output data elements on which the errors may occur, and nodes of type C corresponding to the checks. There is an edge from an F node i to an E node j if data element d_j is affected by processor P_i. There is an edge from node j of type E to node k of type C if the data element d_j is checked by check c_k. For the analysis of faults in the system, a generalized error table (GET array) is constructed from the graph model [23]. The GET array contains all possible error combinations of the faults under consideration. The detectability or locatability of a fault is determined by observing whether all the error patterns produced by that fault are detected or located by the checks provided.

Even though this model can be used for the accurate analysis of systems using ABFT, it has some limitations. The complexity of the analytical algorithms based on this model is exponential in the number of data elements in the system. This leads to enormous memory and time requirements. Inefficient handling of invalidation of checks, performed by faulty processors, is another drawback of the model. However, the model gives a theoretical framework for representing fault-tolerant systems.

1.4. Thesis Outline

This thesis is organized in the following way. A detailed description of ABFT systems is given in Chapter 2. A general description of the multiprocessor systems which are candidate architectures for the application of ABFT is provided. The concept of (g, h) checks is discussed and examples are given. We consider fault-tolerant matrix multiplication in detail and derive a general set of real-number codes for fault-tolerant matrix operations on processor arrays.

In Chapter 3, first we briefly describe the graph-theoretic model. Then we present the new matrix-based model. In this model, the relationship between processors, data, and the checking operations are represented in terms of three matrices, the PD matrix, the DC matrix, and the PC matrix. The physical significance of the model matrices is explained with examples. The problem of invalidating the checks performed by faulty processors is transformed into a problem of error detection at the output of the faulty processor. This eventually simplifies the complexity of the analysis algorithms.

Based on this model, algorithms are developed for determining the fault detectability and locatability of ABFT systems. Unlike the algorithms based on the graph model, these algorithms do not need exhaustive enumeration of errors in order to analyze the system completely; instead, we propose an error collapsing technique which reduces the complexity of the analytical algorithms from exponential to linear in the number of data elements, and polynomial in the number of processors. Application of these algorithms for the analysis of ABFT systems is illustrated with some realistic examples. Finally we propose an alternative method for the invalidation of checks performed by faulty processors.

Chapter 4 deals with the design of ABFT systems. We propose a straightforward methodology for designing such systems. The advantage of this technique is that it can handle error detectability and locatability simultaneously. Also, when the processors in the system are producing large volumes of data, the new technique results in a smaller number of checks when compared to those for the existing algorithms.

Even though the complexities of the analysis algorithms are less than the complexities of the previous algorithms [23], the computation may require a large amount of time and memory when the system has a large number of processors producing huge volumes of data. In contrast, a hierarchical approach will reduce the complexity of the algorithms to a polynomial in the logarithm of the processors in the system. In Chapter 6 we illustrate a particular hierarchical approach to build large fault-tolerant multiprocessor systems. Based on this approach a hierarchical analysis procedure is outlined.

In Chapter 7 we give a summary of the results in the thesis. Finally, some pointers are given towards future research in the related area. In order to make it easy for the reader to place the thesis in the vast area of reliable computing, a relational tree diagram is shown in Figure 1.1. The area enclosed in the dotted rectangle represents the area covered in this thesis. Even though the figure suggests that the analysis and design techniques developed in this thesis are pertinent to ABFT systems, it should be noted that these techniques are applicable to other types of fault-tolerant systems as well.

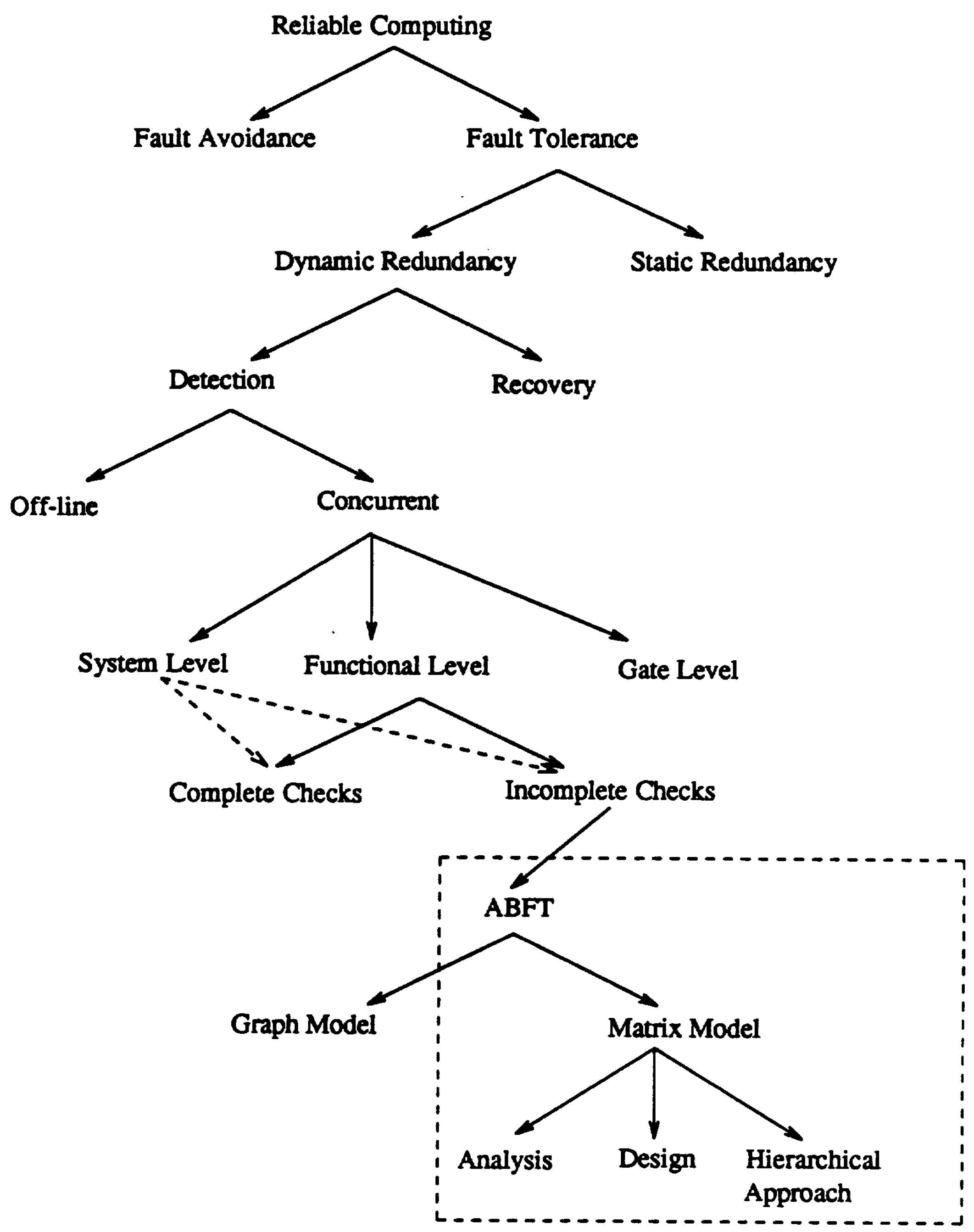


Figure 1.1. Scope of this thesis.

CHAPTER 2.

ALGORITHM-BASED FAULT TOLERANCE

2.1. Introduction

As discussed in the preceding chapter, fault detection and diagnosis are integral parts of any fault tolerance scheme. There are two ways to detect faults: (1) by off-line checking and (2) by concurrent checking. In an off-line checking scheme, the computer (processor) is checked for its correctness while it is not performing any useful computation. This approach has the advantage that the performance of the computer will be unaffected by the checking operation; however, this kind of checking can detect only permanent faults. Transient faults, which constitute 75-80% of faults in a computer system [11], will not be detected by off-line checks. In order to detect transient faults, concurrent error detection schemes such as duplication and comparison have been suggested. These schemes suffer from 200-300% hardware or time redundancy. In many application areas this amount of overhead is unaffordable. This motivated researchers to develop new schemes that require less overhead.

A concurrent error detection scheme called algorithm-based fault tolerance (ABFT) has been suggested by Huang and Abraham for attaining the above objectives [14]. In ABFT the input data elements are encoded in the form of error detecting or correcting codes. The original non-fault-tolerant algorithm is modified to operate on encoded data and produce encoded outputs, from which useful information can be recovered easily. The modified algorithm will take more time to operate on the encoded data when compared to the original algorithm, and this time overhead must not be excessive. The task distribution among the processing elements is done in such a way that any malfunction in a processing element will affect only a small portion of the data, which can be detected and corrected using the properties of the encoding.

It has been observed that ABFT techniques are very cost effective when applied to processor arrays. In this chapter we give a general description of systems which are good candidates for the application of ABFT. The concept of algorithm based fault tolerance will be illustrated with some application examples.

2.2. General System Description

In this section, we describe the general features of multiprocessor systems which are candidate architectures for the application of ABFT techniques. It may be noted, however, that the application of ABFT techniques is not limited to multiprocessor systems; they are also applicable to algorithms running on uniprocessors, probably with less efficiency.

An algorithm executing on a multiple processor system is specified as a sequence of operations performed on a set of processors in some discrete time steps. Each processor has a local memory on which it can perform reads and writes. It can also communicate with other processors in the system through buffers at various input and output ports. A processor cannot read or write from any other processor's local memory even in the presence of a fault. This is not an unrealistic assumption since most of the existing fault-tolerant multiprocessor systems are of the message passing type rather than the shared memory type. This is because in a shared memory architecture, error confinement is difficult, often, impossible. However, the concept of distributed shared virtual memory has been developed to support shared memory programming models in *loosely coupled* distributed multiprocessor systems [24]. These architectures have the advantages of a distributed memory parallel machine in a hardware point of view, whereas, in a software point of view they have the additional advantages such as ease in process migration, ease in passing complex data structures among processors and ease in object synchronization in object-oriented systems. Error recovery in such systems is described in [25]. In this thesis we deal exclusively with machines using message passing paradigm for communication among the processors.

2.2.1. Faults and errors

A *fault* is any condition that causes a malfunction in a single processor while performing operations. Some of the major causes which result in faults are: (1) manufacturing defects such as photolithography errors, deficiencies in process quality and improper designs; (2) wear out in the field due to electromigration, hot electron injection etc.; (3) environmental effects such as alpha particles and cosmic radiations [26, 27]. The manifestations of these faults are called errors [28].

An *error* is any discrepancy between the expected result of an operation and the actual result of the operation. Since a processor performs different types of operations, a fault in the processor may result in errors in any of those operations. For example, if the processor is performing some data computation, a fault in the processor may produce a wrong value of the data. If the processor is trying to read an address location, a fault may cause wrong address selection (addressing fault). However, certain types of faults may not produce any error at all.

Algorithm-based fault tolerance schemes are based on functional fault models that allow any single module in the system to be faulty [14]. Even though the faults and errors are treated at a high level, the model covers all the stuck-at faults and the corresponding errors in the lower gate and circuit levels. In addition, the model is independent of the type of design or technology used in the IC. In summary, we assume Byzantine type of faults [29].

In order to detect the presence of a fault in a processor, we resort to a technique called *data value checking* [30]. Here, a fault is detected by detecting errors in the final data value generated by the processor. One observes that the problem of detection of various faults such as addressing faults can be translated to the problem of detecting errors in the computed results [31]. Therefore, all the faults are treated uniformly as those corrupting the final, computed result.

On the other hand, if a particular fault does not necessarily produce any errors in the final data value computed by that processor, we may disregard the presence of that fault. The computed result of a processor may be checked by one or more other processors in the system. Processors which check the output of one or more processors are called *check evaluating* processors or, in short, *check* processors. The evaluation of a fault in a check processor can also be translated to the problem of error detection at the output of that processor as we show in Chapters 3 and 4.

We assume that any processor in the system is capable of performing useful computations, check evaluation, or both. A *check* on the data element is any combination of hardware and software procedures performed on the data by processors which use the encoding of the data to generate a "pass" or "fail" output.

Let q be the total number of checks that are applied on the data to perform the system level checking and $C = \{c_1, c_2, ..., c_q\}$ denote the set of checks. Let n be the total number of data and pseudo-data elements and $\lambda = \{e_1, e_2, ..., e_n\}$ be the set of errors in the data and pseudo-data elements. The set E represents the sets of error patterns = $\{E^1, E^2, ..., E^{2^n}\}$, consisting of all subsets of λ. Let N be the number of processors in the system which includes both the processors performing useful computations as well as the processors performing the evaluation of the checks. Faults in the processors can be denoted by the set $\nu = \{f_1, f_2, ..., f_N\}$, where f_i denotes a fault in processor p_i. The set $F = \{F^1, F^2, ..., F^{2^N}\}$ consists of all subsets of ν, and each fault pattern, $F^k \in F$, is permissible in the system. Fault patterns consisting of t or fewer faults are called *t-faults*.

DEFINITION 2.1. *DATA*(P_i) is the set of data elements affected by processor P_i. □

DEFINITION 2.2. *CHECK*(d_i) is the set of checks that evaluates the correctness of the data element d_i. □

2.2.2. The concept of (g, h) checks

Formally, a (g, h) check is one which is defined on g data elements, d_1, d_2, ..., and d_g, and evaluated by a check-evaluating processor such that

(1) the check passes (outputs 0) if

(1.1) the check-evaluating processor is not faulty, and none of the data elements is in error;

(2) the check fails (outputs 1) if

(2.1) at least one data element is erroneous and the number of erroneous elements among the g data elements does not exceed h and the check-evaluating processor is not faulty;

(3) the check is invalid (may output 0 or 1) if either

(3.1) more than h data elements are erroneous, or

(3.2) the check evaluating processor is faulty.

The variable h is referred to as the *error detectability* of the check.

Note that these checks are different from the *complete* checks defined in [17, 19]. In those works, the authors assume that whenever a checked unit is faulty and at least one of the checked units is fault-free, the fault in the checked unit will always be detected. The (g, h) checks are incomplete in this sense. In other words, even when all the checking units are fault-free and the checked unit is faulty, the fault may go undetected. Condition 3.1 covers this possible incompleteness of (g, h) checks in the sense that even if the check evaluating processor is fault-free, it may not detect a fault in another processor if the number of erroneous data elements, checked by that processor, exceeds h. We illustrate another important property of (g, h) checks in the following example.

EXAMPLE 2.1. Consider a check C which checks the equality of n data elements when they are all correct. Since the checking operation is done on n data elements, $g = n$. Any error on up to $n-1$ number of data elements will be detected by the check. However, if the error occurs on all the n data elements in such a way that the resulting numbers are still the same, the check will not detect that error. Therefore, the error detectability h of the check is $n-1$. It may be noted that, even though the check can detect a multiple number of faults, it cannot locate an error. In general, this is an important distinction between (g, h) checks and error detecting/correcting codes such as Hamming codes where the error detectability of t implies an error correctability (locatability) of $\left\lfloor \frac{t}{2} \right\rfloor$ [12]. □

Having described the general features of a system supporting algorithm-based fault tolerance, we will present the salient features of ABFT techniques and illustrate them with some application examples.

2.3. Characteristics of ABFT

This technique is distinguished by three characteristics:

(1) Encoding the input data stream.

(2) Redesign of the algorithm to operate on the coded data.

(3) Distribution of the additional computational steps among the various computational units in order to exploit maximum parallelism.

The input data are encoded in the form of error detecting or correcting codes. The modified algorithm operates on the encoded data and produces encoded data output, from which useful information can be recovered very easily. Obviously, the modified algorithm will take more time to operate on the encoded data when compared to the original algorithm; this time overhead must not be excessive. The task distribution among the processing elements should be done in such a way that any malfunction in a processing element will affect only a small portion of the data, which can be detected and corrected using the properties of the encoding.

Signal processing has been the major application area of ABFT until now, even though the technique is applicable in other types of computations as well. Since the major computational requirements for many important real-time signal processing tasks can be formulated using a common set of matrix computations, it is important to have fault tolerance techniques for various matrix operations [32]. Coding techniques based on ABFT have already been proposed for various computations such as matrix operations [14,33], FFT [34], QR factorization, and singular value decomposition [35]. Real-number codes such as the Checksum [14] and Weighted Checksum codes [16] have been proposed for fault-tolerant matrix operations such as matrix transposition, addition, multiplication and matrix-vector multiplication. Application of these techniques in processor arrays and multiprocessor systems has been investigated by various researchers [36,15,37]. In order to illustrate the application of ABFT techniques, we discuss fault-tolerant matrix operations in detail. We present some previous results in the area and then present some new results related to encoding schemes for fault-tolerant matrix operations.

2.4. ABFT Techniques for Matrix Operations

As mentioned in the preceding chapter, various methods such as checksum encoding, weighted checksum encoding and average checksum codes have been proposed for fault-tolerant matrix operations. These encoding schemes are especially suitable for computations in processor arrays [38].

EXAMPLE 2.2. Consider multiplying two 2×2 matrices A and B.

$$A = \begin{bmatrix} 2 & 1 \\ -1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 \\ 3 & 2 \end{bmatrix}.$$

We append an additional row (checksum row) to matrix A and an additional column (checksum column) to matrix B. Now the product of these appended matrices will have an additional row and an additional column that satisfy the checksum property.

$$\begin{bmatrix} 2 & 1 \\ -1 & 0 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 3 & 2 & 5 \end{bmatrix} = \begin{bmatrix} 5 & 2 & 7 \\ -1 & 0 & -1 \\ 4 & 2 & 6 \end{bmatrix}.$$

The implementation of this multiplication on a mesh-connected processor array is shown in Figure 2.1. Here the encoded A matrix is broadcasted among the processors in a horizontal direction and the encoded B matrix is broadcasted vertically as shown in the figure. The resultant matrix entries are shown within the rectangles, representing the processors. It has been shown that this kind of computational setup can detect three simultaneous faults or locate a single fault in the array. □

The use of the checksum codes is limited due to the inflexibility of the encoding schemes and also due to potential numerical problems. Numerical errors may also be misconstrued as errors due to physical faults in the system. A generalization of the existing schemes has been suggested as a solution to these shortcomings [39]. In order to complement those results, we prove that for every linear code defined over a finite field, there exists a corresponding linear real-number code with similar error detecting and correcting capabilities.

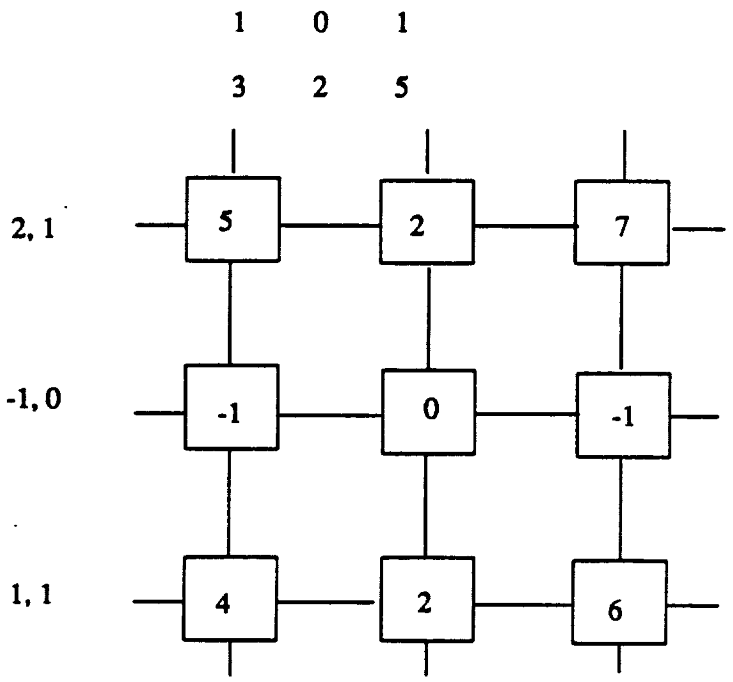


Figure 2.1. Matrix multiplication on a mesh-connected processor array.

2.4.1. Real-number codes for fault-tolerant matrix operations

Real-number codes are codes defined over the field of real numbers. This is a high level encoding scheme. In this section, we develop a general set of real-number codes for fault-tolerant matrix operations. We use the general definition of encoded matrices as given in [38].

DEFINITION 2.3. An *encoder vector* is a vector whose inner product with a column/row vector will produce a column/row check element. □

DEFINITION 2.4. An *encoder vector* is said to be a *Valid Encoder Vector* (*VEV*) if it produces check elements whose properties will be preserved during matrix multiplication, addition, transposition and LU-decomposition.

It has been proved that linearity is a necessary and sufficient condition for an encoding vector to be a *VEV*. Therefore, in the following discussion we consider only linear encoding schemes.

2.4.1.1. General description of linear codes

A data sequence $\{x_i\}$ over any finite field can be divided into blocks of k symbols which are processed independently. A typical block may be represented as a row vector of length k

$$x = [x_1, x_2, \ldots x_k]$$

and the corresponding code vector is given as

$$y = [y_1, y_2, \ldots, y_n].$$

Here x and y are related by

$$y = x\, G$$

where G is an $k{\times}n$ matrix called the generator matrix [40, 41]. Thus the row space of G is the linear code Y, and a vector is a code if and only if it is a linear combination of the rows of G. Such a code is called an $(n,\ k)$ code. Error detection is accomplished with the help of the *parity check* matrix H which satisfies the condition

$$G\, H^T = 0$$

The number of errors which can be detected and corrected by a code can be described in terms of the Hamming weight [12, 41, 40] of the code. A code of Hamming weight $d+1$ can detect at most d errors and correct at most $\left\lfloor \frac{d}{2} \right\rfloor$ errors [12, 42]. Error detectability may also be expressed in terms of the linear independence of columns of the matrix H^T. A code is t error detectable if and only if any set of $\leq t$ number of columns of

H^T are linearly independent [41]. In order to derive a correspondence between finite-field codes and real-number codes, we make use of the second definition of error detectability.

2.4.2. Systematic codes

Systematic codes are a special class of linear (n, k) codes. Here, (n-k) check elements are appended to k actual data elements. If the actual data word is

$$x = [x_1, x_2, \ldots, x_k]$$

the corresponding code word is

$$y = [x_1, x_2, \ldots, x_k, c_1, c_2, \ldots, c_{n-k}]$$

The generator matrix G of the systematic codes is of the form

$$G = [I_k \mid P], \tag{1}$$

where I_k is a k-dimensional unit matrix and P is a $(k \times n-k)$ matrix. A matrix H of the form $[-P^T \mid I_{n-k}]$ will form a parity check matrix.

In most of the high speed processing techniques, systematic encoding is preferred because once the received (or computed) result is found to be error free, retrieval of the actual information from the code vector is straightforward. Checksum and weighted checksum encodings are examples of systematic encoding. However, it has been proved that any linear encoding is equivalent to a systematic encoding scheme, in the sense that any linear generator matrix can be transformed into another combinatorially equivalent generator matrix [41] of the form given in Equation (1). Therefore, in the following discussion we will not make any distinction between a linear code and a systematic linear code.

Lemma 2.1. Vectors which are linearly independent over a finite field are also linearly independent over the field of real numbers.

Proof: Let us consider a finite field GF(q) where the additions and multiplications are done (modulo q). Suppose $v_1, v_2, \ldots, v_k$ are linearly independent over the field GF(q). Let A be the matrix whose columns/rows are the vectors $v_1, v_2, \ldots, v_k$. By definition of linear independence [43], there exists an $k \times k$ submatrix D of A such that

$$|D| \ (mod\ q) \neq 0,$$

where $|D|$ is the determinant of the submatrix D. For determining the linear dependence or independence of these vectors over the field of real numbers, we take the linear combination of the rows of A, where the rows are multiplied by real numbers rather than by elements from GF(q). If r_i is the real number multiplicand of vector v_i, in the place of $|D|$, we will have $(\prod_{i=1}^{i=l} r_i)\ |D|$, which is not equal to zero, since $|D| \ (mod\ q) \neq 0$. Therefore, the vectors v_1 through v_k are linearly independent over the field of real numbers. □

Lemma 2.2. If vectors $v_1, v_2, \ldots, v_k$ are linearly dependent over a finite field GF(q), they are not necessarily linearly dependent over the field of real numbers.

Proof: If $v_1, v_2, \ldots, v_k$ are linearly dependent, it implies that any submatrix D of A is such that

$$|D| \ (mod\ q) = 0,$$

which does not imply that $(\prod_{i=1}^{i=l} r_i)\ |D| = 0$; therefore, the vectors need not be linearly dependent over the field of real numbers. □

THEOREM 2.1. For any *t–error detecting* code defined over a finite field, there exists a corresponding code over the field of real numbers, with the same generator matrix and the same parity check matrix, whose error detectability is $\geq t$.

PROOF: Let C_f be a *t–error detecting* code defined over a finite field with generator matrix G_f and parity check matrix H_f. From the previous discussion, we know that every set of t, or smaller number, of columns of H_f^T will be linearly independent over the finite field. Then, by Lemma 2.1, these columns are also linearly independent over the field of real numbers, which implies that for a code C_r over the field of real numbers having generator matrix $G_r = G_f$ and parity check matrix $H_r = H_f$, the error detectability will be at least equal to t. By Lemma 2.2, it may be possible that a larger number of columns of H_r^T are linearly independent which effectively increases the error detecting capability of the code. Thus, the error detectability of C_r is greater than or equal to t. □

The set of *single–error correcting* linear real-number codes presented in [44] is one special case of the general sets of codes established by Theorem 2.1.

EXAMPLE 2.3. Consider the finite field GF(7) employing symbols $\{-3, -2, -1, 0, 1, 2, 3\}$. A matrix with all distinct columns of length two will define the parity matrix H of a Hamming code over the finite field GF(7). Let

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ -3 & -2 & -1 & 1 & 2 & 3 & 0 & 1 \end{bmatrix}.$$

This will also define a real-number code by regarding H as being over the real numbers.

The corresponding generator matrix is

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -1 & 3 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 2 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & -2 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & -3 \end{bmatrix}.$$

This real-number code can detect at least two errors or correct one error. □

EXAMPLE 2.4. Let us consider simple parity encoding over the field of binary numbers. It is known that parity codes are single error detecting [40], (that is, the Hamming distance is two) with a generator matrix

$$G = [\, I_k \mid P \,]$$

where $P = [\,1, 1, .., .., .., 1]^T$. It can be observed that the corresponding code (as in Theorem 2.1) over the field of real numbers is the simple row checksum code. □

The one to one correspondence between finite-field codes and real-number codes is a powerful result from an implementation point of view: (1) since most of the existing codes are proposed for finite fields, adapting those codes for real-number computations will be easier than inventing new codes for real-numbers; (2) the real number codes lend themselves to implementation in digital signal processors employing standard arithmetic units; (3) furthermore, they can be conveniently implemented in software which does not efficiently admit the bit by bit representation and manipulation required by finite field codes.

The application of these general sets of codes greatly improves the numerical performance of the fault tolerance scheme [32]. Details may be found in [38].

2.5. Conclusions

We discussed the salient features of ABFT techniques. A detailed description of systems supporting ABFT was presented with examples. The concept of (g,h) checks was elaborated and the distinctions between these checks and the Hamming codes were highlighted. Finally, we considered fault-tolerant matrix operations using ABFT on a processor array. In the process of developing a general set of codes for fault-tolerant matrix operations, we proved a fundamental theorem relating the error detectability of finite field codes and the error detectability of the corresponding real-number codes.

CHAPTER 3.

A MODEL FOR ALGORITHM-BASED FAULT TOLERANCE

3.1. Introduction

As discussed in the previous chapter, ABFT techniques are being more and more widely applied. Due to the critical nature of most of the application areas, it is necessary to know the fault tolerance capabilities of the computer system before it is put to the application. This requires an analytical procedure, which in turn requires a good model to represent the system in general.

The analysis of ABFT systems is difficult when compared to the analysis of conventional fault-tolerant systems such as TMR and TTR. In conventional designs of fault-tolerant systems, designers assume that complete tests are available for individual processors [18, 19]. That is, if the tested unit is faulty and the tester is fault-free, then the test is guaranteed to fail. However, in ABFT systems, errors in computed results are detected directly and the faults are detected indirectly. Most of the time there does not exist a one-to-one correspondence between errors and faults. One fault may produce multiple errors. If a processor is computing more than one data element, a fault in that processor may or may not produce an error in one or more of those data elements. For instance, a processor computing 3 data elements may generate 8 different error patterns (including the case where it does not cause any error in any three of the computed results) when it becomes faulty. In order to detect a fault in a processor, the checking operations done on the processor must be able to detect all the possible error combinations. The error detectability of the checks in the system is limited and hence the checks can detect an error only if the size of the error pattern does not exceed the error detectability of the checks. Therefore, situations may arise such that there are fault free processors checking a faulty processor, and still the fault is not being detected. This incomplete nature of the checks adds to the complexity of the analysis of ABFT systems.

The first attempt towards modeling ABFT systems was made by Banerjee and Abraham [23] who proposed a graph-theoretic model. In this model, the system is represented as a tripartite graph having three groups of nodes: nodes of type F corresponding to the possible faulty processors, nodes of type E corresponding to the output data elements on which the errors may occur, and nodes of type C corresponding to the checks. Even though the model is especially suitable for the analysis of faults in systems using ABFT, the analysis of conventional redundancy techniques such as duplication, triplication, or NMR can easily be done using this model. The limitation of the model is that the complexity of the analytical algorithms based on this model is exponential in the number of data elements in the system. This leads to enormous memory and time requirements for the analysis of complex systems with a large number of processors, with each processor producing large volumes of data. However, the model forms a theoretical framework for representing fault-tolerant systems.

In order to assuage the complexity of the analysis algorithms, we propose a matrix-based model. In this model, we define three matrices, the PD (Processor-Data) matrix, the DC (Data-Check) matrix and the PC (Processor-Check) matrix, which describe the system as a whole. The PD matrix represents the relationship between the processors and the data elements computed by them. The DC matrix contains the information regarding which check is checking which data element. The PC matrix is the product of the PD and the DC matrices.

If a *check processor* becomes faulty, the checking operations performed by that processor should be invalidated. To that end we introduce pseudo-data elements associated with every *check processor.* A fault in the *checkprocessor* will always produce an error in the pseudo-data element since an infinite weight is assigned to that data element. Thus, check invalidation is translated to a problem of error detection at the output of a faulty processor.

In this chapter we first give a brief description of the graph model. For completeness of the thesis, we discuss various fault detection and location constraints based on the model. The motivation for developing a new model is given by highlighting some of the limitations of the graph model. Then the matrix model is developed and the significance of the model matrices is explained. The modeling of ABFT systems using both the models is illustrated with examples. Finally, in the conclusion, we provide a critical comparison between the models.

3.2. Graph Representation of a System

In this model, the system is represented as an undirected graph with four sets of nodes and edges between them. The first set of nodes (called processor nodes) represent the processors performing useful computations. The results of the useful computations of the algorithm form the second set of nodes (called data nodes). The set of checks form the third set of nodes (called check nodes). The checks are performed on a set of checking processors, which form the fourth set of nodes (called evaluator nodes).

Edges between processor and data nodes represent dependencies of the result data elements on the processors. There is an edge from a processor node p_i to a data node d_j if $d_j \in DATA(p_i)$. Edges between data and check nodes represent the definitions of the checks on the data elements. If check c_k operates on data element d_i, then there is an edge between data node d_i and check node c_k. Edges between check and evaluator nodes model the check evaluation process. If an evaluator p_j participates in the evaluation of a check c_k, there is an edge between the evaluator node p_j and check node c_k.

A fifth set of nodes, the "pseudo-data" nodes, is introduced to facilitate a uniform network to treat faults in processors performing useful computations and faults in processors performing check evaluations. Every check has associated with it a number of processors involved in the evaluation of the check. For every check-evaluator pair, (check c_k, processor p_i), there is a pseudo-data node. Since there is a one-to-one correspondence between an evaluating processor and a pseudo-data node for a given check, *a fault in a processor evaluating a check* means the same as *an error in the corresponding pseudo-data element.*

The notion of invalidation of checks has been extended as errors in the pseudo-data elements. In the ordered set of errors, whenever there is an error in a pseudo-data element, the corresponding checking operation is considered to be invalid. The errors in pseudo-data elements and actual data elements are treated identically so that faults in processors performing useful computations and faults in check-evaluating processors can be considered without any distinction. With these observations, the system graph can be simplified by merging the data and pseudo-data nodes and the processor and evaluator nodes. The resulting graph has three sets of nodes: processor nodes, data nodes and check nodes.

EXAMPLE 3.1. Consider a hypothetical system having 4 processors P_1 through P_4. Processors P_1 and P_3 produce useful data elements whereas processors P_3, and P_4 performs check evaluations. The relationships among the processors, data, and checking operations are as given in the following.

$DATA(P_1) = \{d_1, d_2, d_3\}$

$DATA(P_2) = \{d_2, d_4\}$

$DATA(P_3) = \{d_5\}$

$DATA(P_4) = \{d_6, d_7\}$

$CHECK(d_1) = \{C_1\}$

$CHECK(d_2) = \{C_2\}$

$CHECK(d_3) = \{C_1\}$

$CHECK(d_4) = \{C_2, C_3\}$

$CHECK(d_5) = \{C_1\}$

$CHECK(d_6) = \{C_2\}$.

$CHECK(d_7) = \{C_3\}$.

It may be noted that data elements d_5, d_6, and d_7 are pseudo-data elements corresponding to checks C_1, C_2, and C_3, respectively. Figure 3.1 shows a graphical representation of the system. □

The model can be easily extended for systems having fault-secure checking units. In such a case, a check is invalid if and only if the corresponding pseudo-data element is

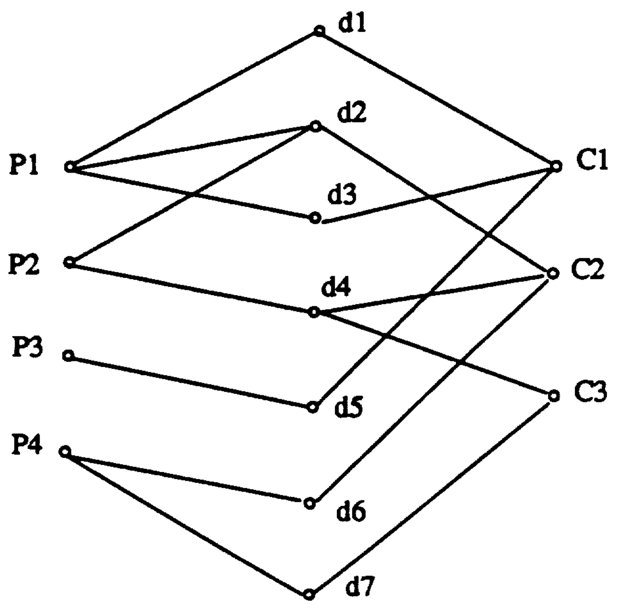


Figure 3.1. Graphical representation of the system in Example 3.1.

erroneous and at least one of the useful data elements evaluated by the check is erroneous. If none of the useful data elements evaluated by the check are erroneous, the check is not invalidated and it will detect an error in the pseudo-data element and hence the fault in the checking processor can be detected.

3.2.1. Detection and location of faults using the graph model

In this section, we describe the fault detection and location constraints derived in [23] using the graph model. To that end, we explain some terminologies used in that study.

The set of checks that may fail for an error pattern E^i is denoted by $FAIL(E^i)$. When E^i consists of a single data element, d_j, the set of checks in $FAIL(E^i)$ is guaranteed to fail. When E^i contains more than one element, the condition on the set of checks in

$FAIL(E^i)$ is that they "may fail" instead of being "guaranteed to fail." This is because it is quite possible that a check that is guaranteed to fail for an error in a single data element might become invalidated in the presence of other errors. However, if a check is not a member of $FAIL(E^i)$, it is guaranteed to pass. The set of checks that are invalidated by the presence of the error pattern E^i is denoted by $INVALID(E^i)$. Then a *generalized error table,* GET, which is a $2^n \times q$ array can be defined [23] such that $GET_{j,k} = 0$, 1, or X, (where X denotes an invalid entry) if for error pattern E^j present, check c_k is known to always pass, always fail, or have an unknown result.

In the following, we define two terms *masking* and *exposing* of faults in the context of error patterns produced by those faults. These terms are frequently used in upcoming discussions.

Definition 3.1. A fault pattern F^j is said to be *masked* by a fault pattern F^k if and only if there exist error patterns, $E^m \in ERROR(F^j)$ and $E^n \in ERROR(F^k)$, such that $FAIL(E^m) \subseteq INVALID(E^n)$. □

Definition 3.2. A fault in F^j is *exposed* if it is not masked by F^j. Suppose $f_b \in F^j$ such that it is exposed in F^j. This implies that for all error patterns $E^m \in ERROR(f_b)$ and $E^n \in ERROR(F^j)$, $FAIL(E^m) \not\subseteq INVALID(E^n)$. □

3.2.1.1. Conditions on fault detection

An algorithm has *t-fault-detectability* iff some check in C will definitely fail provided the number of faults present in the system, on which the algorithm is executed, does not exceed t. It was implicitly assumed that no check will fail if the system is fault-free. With these formulations, conditions are derived for t-fault-detectability [45].

THEOREM 3.1. An algorithm A executing on a computing system S has t-fault-detectability if and only if, for every non-zero $F^i \in F(t)$, it is implied that for all $E^j \in ERROR(F^i)$, $GET_k^j = 1$ for some $c_k \in C$.

PROOF: The proof of this theorem is given in [23].

This necessary and sufficient condition for fault detection is difficult to evaluate in practice. Instead, the concept of *closure* of a fault has been introduced [23], which is very similar to the closure of faults defined in [18]. Despite this concept, the algorithm for fault detection is based on the exhaustive enumeration of all error combinations and hence is exponential. However, it forms a basis for a condition for fault detection.

3.2.1.2. Conditions on fault location

An algorithm is said to have *t-fault-locatability* if and only if the application of the check set identifies precisely which faults are present, provided the number of faults does not exceed t. In order to evaluate the fault locatability of a system, the concept of row intersection has been used [23], similar to the row intersection operation (denoted by Π) defined in [18].

THEOREM 3.2. An algorithm has t-fault-locatability if and only if, for all unequal fault patterns, $F^i, F^j \in F(t)$, it is implied that for all $E^m \in ERROR(F^i)$, and for all $E^n \in ERROR(F^j)$

$GET_m \Pi GET_n = \varnothing$

PROOF: The proof of this theorem is given in [23].

It has been observed [23] that a system is *t−fault* locatable if in any fault pattern of cardinality k, min(k, 2t+1-k) faults are exposed for $k = 1, 2, \ldots \min(2t,n)$. Algorithms have been developed for determining the fault locatability of systems using this sufficient condition which again need the exhaustive enumeration of all error patterns. Based on these results, we derive better sufficiency conditions for *t−fault* locatability along with our second model.

3.2.2. Limitations of the graph-theoretic model

Here we summarize the drawbacks of the graph model. As discussed in the preceding sections, the analysis algorithms based on this model need exhaustive enumeration of all error patterns and hence are of exponential complexity. Since one pseudo-data element is introduced for every checking operation, that will effectively increase the number of data elements in the system which in turn means a larger exponent of complexity.

In the next section we propose a matrix-based model which does not have the above mentioned drawbacks. In order to incorporate the invalidation of checks done by faulty processors, we introduce one pseudo-data element per checking processor instead of one for each checking operation (note that a processor may perform more than one checking operation). The analysis algorithms are of linear complexity in the number of data elements, and polynomial in the number of processors.

3.3. An Improved Matrix-Based Model

In an improved model for multiple processor systems, the relationships between processors, data, and checks can be represented by three fundamental matrices, the PD (Processor-Data) matrix, the DC (Data-Check) matrix, and the PC (Processor-Check) matrix [46]. Unlike the graph-theoretic model described in the previous section, we do not make any assumptions regarding the fault secureness of the check evaluating processors in this model. Instead, the model is developed with the following general assumptions. Whenever a check evaluating processor becomes faulty, all of the checks done by that processor become invalid (Byzantine type faults are assumed here). If a processor is performing both useful computation and check evaluation, we identify two kinds of faults associated with it: (1) observable faults and (2) unobservable faults. For an observable fault, at least one of the data elements produced by the faulty processor will be erroneous, whereas for an unobservable fault all the useful computation results from the processor will be correct. In both the above cases, all of the check evaluations done by the faulty processor will be deemed to be invalid.

3.3.1. The model matrices

In the new model for multiple processor systems, the relationships between processors, data, and checks are represented by three fundamental matrices, the PD (Processor-Data) matrix, the DC (Data-Check) matrix, and the PC (Processor-Check) matrix. We define the following model matrices in terms of parameters N, the number of processors, n, the number of data elements, and q, the number of checking operations in the system.

DEFINITION 3.3. The PD matrix is an $N \times n$ matrix such that

$$PD_{ij} = \begin{cases} 1 & \text{if } d_j \in DATA(P_i) \\ 0 & otherwise \end{cases} \qquad \square$$

DEFINITION 3.4. The DC matrix is an $n \times q$ matrix such that

$$DC_{ij} = \begin{cases} 1 & \text{if } C_j \in CHECK(d_i) \\ 0 & otherwise \end{cases} \qquad \square$$

DEFINITION 3.5. The PC matrix is an $N \times q$ matrix which is the product of the PD and DC matrices. $\square$

It may be noted that so far in this model we have considered only actual data elements. Until now, there is no relationship established between a checking operation and the processor which performs that operation. (It may be noted that in the graph model this relationship was accounted for through pseudo-data elements.) However, we will incorporate this relationship between processors and checks performed by them in the next section by defining a new set of pseudo-data nodes.

Until now, there exists a correspondence between the system graph and the model matrices. If we split the tripartite graph into two bipartite graphs, a processor-data graph and a data-check graph, the PD and DC matrices are the adjacency matrices of those bipartite graphs, respectively. Now construct another bipartite graph having the set of processor nodes and the set of check nodes as its parts such that there is an edge from node P_i to node c_j if there is a path of length two between these two nodes in the original system graph. The PC matrix is the adjacency matrix of this new graph (can be a multigraph). However, the correspondence between the graph model and the matrix model will be lost once we introduce the concept of pseudo-data elements.

3.3.2. Physical significance of the model matrices

The physical significances of the PD and the DC matrices are clear from their definitions. In the PC matrix, PC_{ij} represents the number of data elements of P_i checked by check C_j. It can be seen that entries in the PD and DC matrices are either a 0 or a 1, whereas the PC matrix can have elements as large as n.

The importance of these matrices in the analysis of faults in the system will be revealed in the following discussion. Without loss of generality, we can use the same matrices for representing faults and errors in the system. The only difference is that in the PD and PC matrices, the row corresponding to P_i stands for a fault in processor P_i. Those elements of row P_i of the PD matrix will be 1 if the corresponding data elements are erroneous due to a fault in processor P_i.

$$PD_{ij} = \begin{cases} 1 & \text{if } d_j \textit{ is erroneous when } P_i \textit{ is faulty} \\ 0 & \textit{otherwise} \end{cases}$$

With this interpretation of matrix entries, it is easy to observe that each row in the fundamental PD matrix, defined earlier, represents a faulty processor whose output data elements are all wrong. The PD matrix will be different for different error combinations at the output. For coherence of terminology, the PD matrices resulting from various output error combination are called the *syndromes* of the original PD matrix as in Definition 3.3. Correspondingly, we will also have different syndromes of the PC matrix. The DC matrix will be independent of the output error combination and is determined only by the system designer and hence, has only one syndrome which is the DC matrix itself.

3.3.3. Check invalidation

In order to accommodate the *invalidation of checks* performed by the faulty processors, we introduce *pseudo–data* elements into the system model. These *pseudo–data* elements are conceptually similar to the *pseudo–data* nodes associated with the graph model, but are modeled and used differently. If a processor is performing one or more check evaluations, a *pseudo–data* element of infinite weight is attached to that processor. Later, every check done by that processor is assumed to be checking the correctness of its pseudo-data element also. If the pseudo-data element is erroneous, all of the checks done by that processor become invalid, since such a data element has infinite weight. Thus, check invalidation is translated into a problem of error detection at the output of a faulty processor.

Accordingly, the model matrices are extended as follows. Suppose m is the number of processors performing check evaluations.

DEFINITION 3.6. The PD matrix is an $N{\times}(n{+}m)$ matrix such that

$$PD_{ij} = \begin{cases} 1 & \text{if } d_j \in DATA(P_i) \\ \infty & \text{if } d_j \textit{ is the pseudo data element of } P_i \\ 0 & \textit{otherwise} \end{cases}$$ □

DEFINITION 3.7. The DC matrix is an $(n{+}m){\times}q$ matrix such that

$$DC_{ij} = \begin{cases} 1 & \text{if } C_j \in CHECK(d_i) \\ 1 & \text{if } C_j \textit{ is resident in } P_k \text{ and } d_i \textit{ is the pseudo data element of } P_k \\ 0 & \textit{otherwise} \end{cases}$$ □

The PC matrix is obtained by finding the product of the PD and the DC matrices.

Example 3.2. Let us consider the system shown in Figure 3.1. The check c_1 is performed by processor P_3 and the checks c_2 and c_3 are performed by P_4. The corresponding PD, DC, and PC matrices are

$$PD = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \infty & 0 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{bmatrix} \qquad DC = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$PC = PD \times DC = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ \infty & 0 & 0 \\ 0 & \infty & \infty \end{bmatrix}.$$

□

3.4. Conclusions

In this chapter we have presented a new matrix-based model for the analysis and design of fault-tolerant multiprocessor systems. The great complexity of the analysis algorithms based on the existing graph-theoretic model was the prime motivating factor in proposing the new model. How the reduction in complexity is achieved will be discussed in the next chapter. It should be noted that the matrix model is not the matrix equivalent of the graph model proposed in [23]. There are subtle differences in the formulations of the models. In the following, we summarize a comparison between the graph model and the matrix model.

3.4.1. Comparison between the graph model and the matrix model

As described earlier in the chapter, the graph model consists of a tripartite graph. Processors, data elements, and checks are the three parts in the graph. In the matrix model we also identify these three entities, whose relationships are represented as the PD, DC, and the PC matrices.

The main difference between the two models lies in the way check invalidation is handled when the processor performing that check is faulty. In the graph-theoretic model a pseudo-data node was defined along with each check in the system. This approach was borne out from the definition of *fault−secure* checks [12], in which checks are capable of indicating their correctness also. In the graph model these pseudo-data nodes are distinguished from the actual data nodes by their respective positions in the set of data nodes. The disadvantages of this approach are:

(1) The ordering of the data elements has to be preserved during the analysis; in other words, every data element (including the actual data and the pseudo-data) has its own identity. It is this constraint which causes an exponential complexity of the corresponding analysis algorithm as we shall see in the next chapter.

(2) Every time a new check is added in the system, a new pseudo-data element is also added. Therefore, the complexity of the analysis algorithm is exponential not only in the number of data elements but also in the number of checks, in an indirect way.

In the matrix-based model, whenever a processor is performing one or more checks, one pseudo-data element of infinite weight is added to the output data set of that processor (the actual data elements assume unit weight in the model). The pseudo-data elements and the actual data elements are distinguished from each other by their weights rather than by their positions. This permits a special grouping of actual data elements in a system while considering all the possible error patterns; data elements within a group do not have individual identity. Based on this grouping, we illustrate an *error collapsing technique*, which eventually results in much less complex analysis algorithms.

In the graph model, the system information is distributed and processed in two domains: the processor-data domain and the data-check domain. In the matrix model we introduced one more domain of operation, the processor-check domain. In fact, the PC matrix, which represents the processor-check domain, is our main work space. The PC domain is derived from the PD and DC domains, during which some information may be lost. However, most of the lost information happens to be unnecessary for the analysis as we shall see in the next chapter. Whenever necessary, we go back to the PD and DC domains to supplement the information to the PC domain. Again, selecting the PC domain as the main domain of analysis greatly simplifies the analysis procedure.

CHAPTER 4.

ANALYTICAL APPLICATIONS OF THE MATRIX-BASED MODEL

4.1. Introduction

In this chapter we describe the applications of the matrix-based model for the analysis of ABFT systems. Following the definitions given in the previous chapter, we develop algorithms for determining the *fault detectability* and *locatability* of ABFT systems. These algorithms are much less complex than the algorithms based on the graph-theoretic model. The new algorithm for the fault detectability analysis is of linear complexity in the number of data elements in the system, whereas the complexity of the locatability algorithm is quadratic in the number of data elements. The reduction in complexity is achieved by using: (1) a special *error collapsing* technique which allows the analysis of a system without having to enumerate all the possible error combinations; (2) simpler sufficiency conditions which are developed in this thesis. Even though these algorithms are developed particularly for the analysis of ABFT systems, they are applicable to the analysis of conventional fault-tolerant architectures such as *N-modular structures*.

We illustrate the applications of the algorithms by analyzing various fault-tolerant signal processing architectures. Finally we provide an alternative method for the invalidation of checks performed by faulty processors.

4.2. Fault Analysis of a System

As far as the fault detectability and locatability of a system are concerned, we have to consider only the observable faults, since the unobservable faults are not going to cause any error in the useful data elements. However, it is necessary to consider the detectability and locatability of observable faults in the presence of unobservable faults. For example, consider a fault pattern consisting of faults on three processors P_1, P_2, and P_3. Let the fault present in P_1 be an observable fault, and the faults in P_2 and P_3 be unobservable faults. Now, for the system to be 3–*fault detectable*, it is necessary that the observable fault in P_1 be detectable in the presence of unobservable faults in P_2 and P_3.

Therefore, if a fault is not observable, instead of assuming that that particular fault is not present, in our analysis we consider it as a detectable fault. In order to define the fault detectability and locatability of a system, we introduce the concept of *observability* of a fault pattern.

DEFINITION 4.1. A fault pattern is *observable* if and only if at least one of the individual faults present in it is observable. □

DEFINITION 4.2. A fault pattern is said to be *completely detectable* if it is either *unobservable* or it is detectable for all the possible output error combinations. □

In the following, we will use the terms *faults* and *fault patterns* interchangeably to mean either an individual fault or a set of faults depending on the situation.

DEFINITION 4.3. A fault-tolerant system has *t–fault detectability* if and only if some check C_i will definitely fail, provided the cardinality of any *observable* fault pattern does not exceed t.

4.3. Analysis for Fault Detectability

For the analysis we define matrices ^{r}PD and ^{r}PC which are derived from the PD and the PC matrices, respectively.

DEFINITION 4.4. The ^{r}PD matrix is defined as the matrix whose rows are formed by adding r different rows of matrix PD, for all possible different combinations of r rows, and then setting all nonzero elements, except the infinity elements, to 1. □

Note that a nonzero element greater than 1 results from the addition of rows of the PD matrix if the processors corresponding to those rows have common data elements. These nonzero elements are set to 1 in order to avoid duplication of the same data element.

DEFINITION 4.5. Matrix ^{r}PC is the product of ^{r}PD and the DC matrix. □

^{r}PC is an $\binom{n}{r} \times q$ matrix. In the fault analysis, each row of ^{r}PD and ^{r}PC will represent the situation in which r faults are present simultaneously. As a special case, it may be observed that ^{1}PC = PC.

DEFINITION 4.6. The row R of ^{r}PC is said to be completely detectable if and only if the fault represented by R is *completely detectable*. □

If R represents an observable fault, there should be at least one element in the row R which is less than or equal to the error detectability (h) of the check used, for all possible errors. If we enumerate all the possible error combinations, the algorithm to check the complete detectability of a row will be as complex as the previous ones. Instead, we use an error collapsing technique so that the algorithm converges much faster and needs less storage.

4.3.1. Algorithm to check whether R is completely detectable

The algorithm is outlined as ALGORITHM 1.

In the following discussion we will describe how the algorithm works. In the first step, if the entries of R are all zeros or infinity, then it is an unobservable fault and hence is a *completely detectable fault*. On the other hand, if some of the entries are zeros and the rest are greater than h, it means that the errors caused by the fault are not detectable, and hence the fault is not detectable. As mentioned before, in the case of analysis of faults in systems, the fundamental matrices PD and PC represent the situation in which all the output elements of a faulty processor are erroneous [46]. In the algorithm we start with a row R which is a combination of some rows of the PC matrix. Therefore, R represents a fault such that the output data elements of the processors associated with R are all erroneous. If at least one element of R is less than or equal to h and greater than zero (we call

---

ALGORITHM 1.

```
(1) If the elements of row $R$ are either zero or infinity, $R$ is completely detectable, stop.
    Otherwise, go to step 2.
(2) If there is no element in row $R$ which is less than or equal to $h$ and greater than 0, $R$
    is not completely detectable, stop. Otherwise go to step 3.
(3) Find all $j$ such that $0 < R_j \le h$.
(4) If $DC_{ij} = 1$ set ${}^{r}PD_{Ri} = 0$, where $i = 1, 2, \ldots q$. Do the same for all $j$ obtained from
    step 3.
(5) If the elements of the syndrome of $R$ are either zero or infinity, then $R$ is competely
    detectable, stop. Otherwise go to step 6.
(6) Find the new ${}^{r}$PC matrix by multiplying the new ${}^{r}$PD matrix obtained from step 3
    with the DC matrix and go to step 1.
```

---

such an entry a "valid entry") we can conclude that the fault is detectable by some checks provided all the data elements from the faulty processors are erroneous. This does not imply that the fault will be detected for all possible error combinations.

For example, consider a system which is graphically represented in Figure 4.1.

Here,

$DATA(P_1) = \{d_1, d_2\}$

$DATA(P_2) = \{d_3\}$

$CHECK(d_1) = CHECK(d_3) = \{C_1\}$

$CHECK(d_2) = \{C_2\}$

Then we have the fundamental matrices

$$PD = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad DC = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \quad PC = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

Obviously, the system is single fault detectable if h=1. In order to check whether the system is 2–*fault detectable* we compute ^2PC which is equal to [2, 1]. Since there is a 1 in this row, the fault is detectable when all the data elements (d_1, d_2, and d_3) are erroneous.

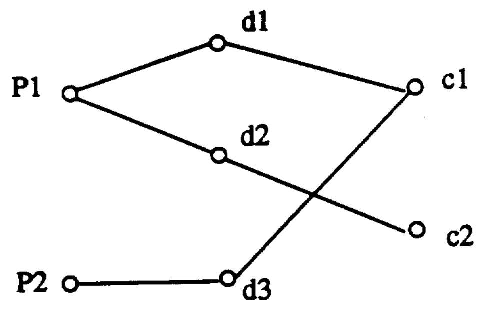


Figure 4.1. Graphical representation of an example system.

Now consider the situation in which the faulty processor P_1 produces an error in d_1 alone. A fault in P_2 will definitely cause an error in d_3. Thus, errors in d_1 and d_3 will invalidate the check C_1; at the same time, check C_2 will pass since the data element d_2 is not erroneous. As a result if P_1 and P_2 are faulty, the fault may not be detected, and hence the system is not 2–*fault detectable.*

This discrepancy is taken care of in Step 4 of Algorithm 1. The objective is to check whether the fault is detectable for all possible output error combinations. For that, we use a technique called *error collapsing*. All the elements of ^{r}PD which contributed to the valid entries of row R are set to zero. By doing this, effectively we are removing those errors which are detectable at the output. We may remove all those errors simultaneously, because if at least one of them were not removed, that would be detectable at the output and hence the fault is detectable.

The new ^{r}PC is calculated by multiplying the ^{r}PD matrix obtained after error collapsing with the DC matrix. This new matrix will be different from the old ^{r}PC in two ways. The new matrix will have zeros in the corresponding positions where the old ^{r}PC had valid entries. Some of the invalid entries in the old matrix might have become valid entries in the new matrix. This is because removal of errors may make some of the invalid checks valid.

This iteration is done as given in Algorithm 1 to check the complete detectability of row R. It may be noted that the same algorithm can be used for determining the fault detectability of systems having fault-secure check evaluation processors. In such a case, all the infinity values are set to zero and the analysis is done in the same way.

EXAMPLE 4.1. In this example, we present a simple instance of how *error collapsing* can help in reducing the complexity of analysis. In Figure 4.2, processor P produces three data elements d_1, d_2, and d_3. Data element d_i (i = 1, 2, 3) is checked by check c_i. Also, note that the check c_i checks data d_i only. We assume that the error detectability h of the checks is equal to 1.

In order to detect or locate a fault in processor P, we start the analysis by determining the detectability and locatability of the worst possible error; here we start with the case where all three data elements are erroneous. Since check c_i is checking d_i only, an error in d_i will always be detected by c_i irrespective of the status of the rest of the data in the system. Thus, we need check the detectability and locatability of only one error pattern (in which all the data elements are erroneous) instead of checking for all the possible (eight in this case) error combinations. □

In the following example, we illustrate the application of the *error collapsing* technique, for the analysis of a hypothetical fault-tolerant multiprocessor system.

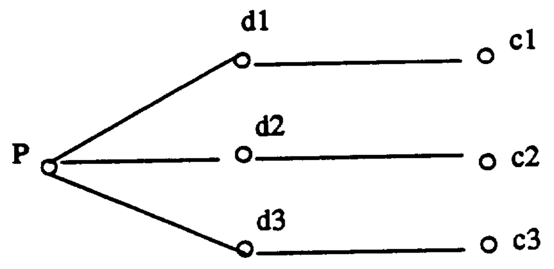


Figure 4.2. Example for *error collapsing*.

EXAMPLE 4.2. Consider a 4-processor fault-tolerant system (for simplicity of illustration, we assume that the checks will yield valid results even when the processors which perform those checking operations are faulty) whose PD and DC matrices are

$$PD = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad DC = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

The corresponding PC matrix is

$$PC = \begin{array}{c} \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \end{array} \begin{array}{l} R_1 \\ R_2 \\ R_3 \\ R_4 \end{array}.$$

Assuming that the error detectability of the checks $h = 1$, we consider complete detectability of rows R_1, and R_2. Since the second element of R_1 is a valid entry, we collapse the corresponding error $PD_{1,2}$ of the PD matrix. The resulting row syndrome of R_1 is [2 0 0] which has no valid entires at all, and hence is undetectable. Therefore, R_1 is not completely detectable. In the case of R_2, if we collapse the error $PD_{2,4}$ corresponding to the valid entry in R_2, the resulting row syndrome will be [0 1 0] which still has one valid entry. If we further collapse the error corresponding to that syndrome also, the resultant syndrome will be [0 0 0]. Then by Algorithm 1, R_2 is completely detectable. □

DEFINITION 4.7. The matrix ^{r}PC is said to be completely detectable if and only if all rows of ^{r}PC are completely detectable. □

THEOREM 4.1. A fault-tolerant system is *t–fault detectable* if and only if the matrices ^{i}PC, for $i = 1, 2, \ldots t$, are completely detectable.

PROOF:

Proof for the necessary condition, (by contradiction): If possible, let the system be *t–fault* detectable, and let ${}^{i}PC$ not be completely detectable for some $i \le t$. This will imply that there exists a fault pattern of cardinality $\le t$ which is not completely detectable. Therefore, the system is not *t–fault* detectable which is a contradiction.

Proof for the sufficiency condition: Complete detectability of ${}^{i}PC$ implies that every fault pattern represented by the rows of ${}^{i}PC$ are completely detectable. Therefore, the hypothesis of the theorem implies that every fault pattern of cardinality $\le t$ is detectable and hence the system is *t–fault* detectable. □

4.4. Analysis for Fault Locatability

Analyzing the system for its fault locatability is a much harder problem when compared to the problem of finding the fault detectability. This is because, in the case of locatability, we have to check not only whether some faults are detected, but also whether that fault is distinguishable from other faults.

DEFINITION 4.8. A system is said to have *t–fault locatability* if and only if the application of the check set identifies precisely which faults are present, provided the cardinality of any *observable* fault pattern does not exceed t. □

LEMMA 4.1. A necessary condition for *t–fault locatability* for $t \ge 1$ of a system is that $\Sigma_i PD_{ij} \le 1$ for all j.

PROOF: We may prove the lemma by contradiction. $\Sigma_i PD_{ij} \le 1$ implies that there can be at most one 1 in every column of the PD matrix which means $DATA(P_i) \cap DATA(P_j) = \phi$ for all $i \ne j$. If possible, let there be more than one 1 in a column, which implies that the data sets produced by certain processors are not disjoint. In that case, if an error is observed in the common data element or elements, it will not be possible to conclude which faulty processor produced that error. Therefore, the system is not single fault locatable. Therefore, the assumption that there is more than one 1 in a column is wrong, and hence the proof. □

In most of the existing multiprocessor systems, processors have nondisjoint output data sets so that the assumption $DATA(P_i) \cap DATA(P_j) = \phi$ is not valid. In those systems, locating a faulty processor will not be possible according to Lemma 4.1. However, processors whose data sets have nonempty intersections with other processors can be collapsed into *processor classes* [23] so that the *processor classes* will have disjoint data sets.

DEFINITION 4.9. A processor class π_i represents a maximal set of processors such that for each processor $p_j \in \pi_i$, there exists another processor $p_k \in \pi_i$, such that $DATA(p_j) \cap DATA(p_k) \neq \varnothing$. □

Any processor not belonging to any such processor class constitutes a class by itself. One may be able to locate a faulty *processor class* (a *processor class* is said to be faulty if at least one of the processors in the class is faulty) during the fault diagnosis of the system. The PD matrices for the *processor classes* are found by adding together the rows of the PD matrix corresponding to the processors in the *processor class* and by setting all nonzero elements to 1.

In the example given in Section 2.2 for the model matrices of a system, $DATA(P_1) \cap DATA(P_2) = \{d_2\}$. Here, P_1 and P_2 will form a *processor class*. Processors P_3 and P_4 form two different *processor classes*. Now the corresponding PD and PC matrices are

$$PD = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad PC = \begin{bmatrix} 2 & 2 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

For convenience, in the forthcoming discussion, we use the term *locatability* to mean *class locatability*.

DEFINITION 4.10. Rows R_1 and R_2 of matrix ^{k}PC are said to have $1-0$ *disagreement* if there is at least one valid element in row R_1 such that R_2 has a zero in the corresponding position. □

DEFINITION 4.11. Rows R_1 and R_2 of matrix ^{k}PC are said to have $0-1$ *disagreement* if there is at least one 0 in either row R_1 or R_2 such that the other row has a valid element in the corresponding position. □

EXAMPLE 4.3. Consider the PC matrix given below.

$$PC = \begin{array}{c} R_1 \\ R_2 \\ R_3 \end{array} \begin{bmatrix} 2 & 2 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

Here, R_1 and R_2 have $1-0$ *disagreement* whereas R_2 and R_1 have only $0-1$ *disagreement*. It can be seen that R_1 and R_3 have no disagreement at all. □

DEFINITION 4.12. If all pairs of rows of ^{k}PC have $0-1$ *disagreement*, then ^{k}PC is said to have a $0-1$ *disagreement*. □

DEFINITION 4.13. PC has $1-0$ *disagreement* with ^{k}PC if and only if every row R of PC has $1-0$ *disagreement* with all rows of ^{k}PC which do not contain R. □

It may be noted that a $1-0$ *disagreement* implies a $0-1$ *disagreement*, whereas a $0-1$ *disagreement* does not imply a $1-0$ *disagreement*. That is, $1-0$ *disagreement* is a stronger condition than a $0-1$ *disagreement*.

4.4.1. Physical significance of disagreement

When two rows of ^{r}PC have $0-1$ *disagreement*, that means that the faults corresponding to those rows are distinguishable provided the outputs from the processors involved in those faults are all erroneous. A $1-0$ *disagreement* between PC and ^{r}PC will imply that every individual fault is exposed (or not masked) (as defined in Chapter 3) in all fault patterns of cardinality $r+1$. This is because every row R in PC has a $1-0$ *disagreement* with all rows of ^{r}PC which do not contain R. In both of the above cases we need all data outputs from a faulty processor to be erroneous, which may not be the case all the time. Therefore, we define a stronger relation between rows, namely, *complete disagreement.*

DEFINITION 4.14. A disagreement ($0-1$, or $1-0$) between two rows is called a *complete disagreement,* if the disagreement exists for all possible error combinations caused by the faults associated with those rows. □

The disagreement defined in[31] was similar to the *0-1 disagreement* defined in this thesis. However, it must be noted that the disagreement used in [31] was defined in the set of error patterns rather than in the set of fault patterns.

In order to check for the *complete disagreement* between two rows we use an algorithm similar to the one used for finding the complete detectability. The procedure is outlined in Algorithm 2. Whenever there is a disagreement between rows, the valid entry or entries which caused the disagreement are set to zero by error collapsing as described in Algorithm 1. The algorithm always converges because removal of an error will never convert a 0 to a 1 or a higher value, whereas it may or may not decrease the values of nonzero entries. (Henceforth, we use the term *disagreement* to mean *complete disagreement.*) Now we prove some necessary and sufficient conditions for the t-fault locatability of a system, and develop an algorithm for the analysis.

From previous discussions in Chapter 3, we observe that whenever the cardinality of the fault pattern is $\leq t$, all individual faults should be exposed in order for the fault pattern to be locatable. When the cardinality is $2t - r$ for $0 < r < t$, a minimum of $r + 1$ faults should be exposed. In order to check whether the system is t-fault locatable, we have to consider all fault patterns of cardinality $\leq 2t$. We prove a simpler sufficient condition for *t–fault locatability* so that we need to consider only fault patterns of cardinality at most t.

THEOREM 4.2. A necessary and sufficient condition for *t–fault locatability* is that all individual faults are exposed in every fault pattern of cardinality $\leq t$, and all fault patterns of cardinality t are distinguishable from each other.

PROOF: We prove this theorem through a simple construction. We use rectangles to represent faults. The length of the rectangle corresponds to the cardinality of the fault

---

ALGORITHM 2.
Input to the algorithm are rows R_1 and R_2 whose complete disagreement is to be checked.

(1) Check whether R_1 and R_2 have a disagreement. If not, output *NO*, stop. Otherwise go to (2).

(2) Collapse errors and check whether the syndrome elements of either R_1 or R_2 are all zeros or infinity. If so, output *YES*, stop. Else go to step 1.

---

pattern. When two fault patterns have common individual faults, the rectangles overlap in their positions. Consider two faults F_1 and F_2 whose cardinalities are $\leq t$.

Case 1.

Let the cardinality of $F_1 = |F_1| = t$ and $F_2 \subset F_1$. The representation using rectangles is shown in Figure 4.3. By assumption, all individual faults in F_1 are exposed (since the cardinality is $\leq t$), which implies that there is a $0-1$ *disagreement* between regions A and B. But $F_2 = B$ which implies F_1 has a $0-1$ *disagreement* with F_2. That is, F_1 and F_2 are distinguishable.

Case 2.

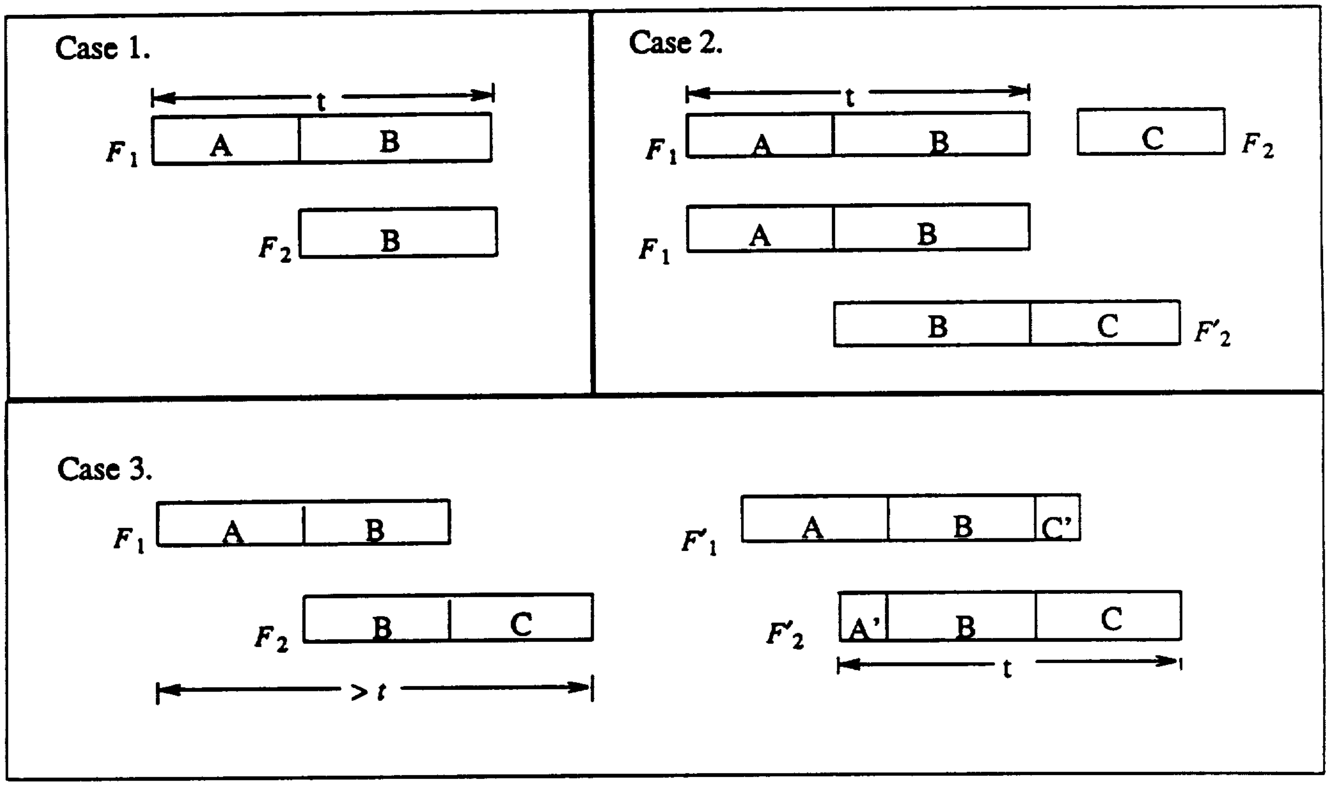


Figure 4.3. Fault patterns of cardinality $\leq t$.

Let $|F_1| = t$ and $|F_2| = r < t$. Also $F_1 \cap F_2 = \varnothing$. We augment F_2 with some faults contained in pattern F_1 such that the augmented F_2 (F'_2) has cardinality t. Now F_1 and F'_2 are distinguishable by the assumption of the theorem. That is, region A has a $0-1$ *disagreement* with region C. (Note that since region B is common to both F_1 and F'_2, it will not contribute to the distinguishability of the faults.) But $C = F_2$ and hence F_1 and F_2 are distinguishable.

Case 3.

Let $|F_1| < t$, $|F_2| < t$ and $F_1 \cap F_2 \neq \varnothing$, also $|F_1 \cup F_2| > t$. (This is the most general case.) In the figure we construct a fault F'_1 by augmenting F_1 with some faults (C') from region C such that $|F'_1| = t$. Similarly F'_2 is constructed by adding a portion of A (A') to F_2. Now F'_1 and F'_2 are distinguishable, which means regions $A - A'$ (part of region A which contains the faults which are not contained in A') and $C - C'$ have a $0-1$ *disagreement*. But $A - A' \subset F_1$, and $C - C' \subset F_2$. Therefore, F_1 and F_2 have a $0-1$ *disagreement* and are hence distinguishable.

Thus any two fault patterns of cardinality $\leq t$ are distinguishable and hence the sufficiency condition is proved. Proof for the necessary condition follows from the definition of *t-fault locatability*. □

The above results may be translated into the domain where we use the new model for the analysis of fault-tolerant systems.

THEOREM 4.3. A given fault-tolerant system is *t-fault locatable* if and only if matrices PC and ^{i}PC, for $i = 1, 2, \ldots (t-1)$, have $1-0$ *disagreement*, and ^{t}PC has $0-1$ *disagreement* with itself.

PROOF: The condition that PC has $1-0$ *disagreement* with ^{i}PC implies that all individual faults are exposed in fault patterns of cardinality $\leq t$. Since ^{t}PC has $0-1$ *disagreement* with itself, all fault patterns of cardinality t are distinguishable. Hence the system is *t–fault locatable* by Theorem 4.2. □

EXAMPLE 4.4. Now we will present a hypothetical system in order to illustrate the various concepts we developed in the preceding sections. Consider a computing system consisting of 5 processors each of which produces one data element each. Every processor performs useful computation as well as 4 checking operations on the data produced by the remaining 4 processors. The PC matrix for such a system is shown in Figure 4.4. In this example we assume that the processors involved in checking operations are not fault secure. Complete analysis of the system shows that it is 4–*fault* detectable and 2–*fault* locatable. It may be noted that this is the maximum detectability and locatability possible with a system having 5 processors.

$$PC = \begin{bmatrix} \infty & \infty & \infty & \infty & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & \infty & \infty & \infty & \infty & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & \infty & \infty & \infty & \infty & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & \infty & \infty & \infty & \infty & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \infty & \infty & \infty & \infty \end{bmatrix}$$

Figure 4.4. The PC matrix of the hypothetical system.

4.5. Complexity of the Algorithms

In the following, we provide a rigorous analysis of complexities of various algorithms proposed in preceding sections. Assume that $n = N \times d$, where n is the total number of data elements, N is the total number of processors, and d is the average number of data elements produced by each processor. Complexity of the algorithm based on the graph-theoretic model for determining the fault detectability (t faults) is exponential in the number of data elements n in the system. The algorithm has to check the detectability of all possible error combinations caused by every fault in $F(t)$.

Therefore, the complexity is

$$O\left(\sum_{i=1}^{i=t+1} \binom{N}{i} \times 2^{id}\right).$$

$$= O\left(\sum_{i=1}^{i=t+1} \binom{N}{i} \times 2^{\frac{in}{N}}\right)$$

$$= O\left(N^t \times 2^{\frac{tn}{N}}\right)$$

which is polynomial in N for a given value of $t \ll N$ (which is usually the case) and exponential in n.

Because of the error collapsing technique we use, the complexity of the algorithm based on our second model is linear in the number of data elements as shown in the following. The complexity of the detectability algorithm is $O\left(\sum_{i=1}^{i=t+1} \binom{N}{i} \times f(id)\right)$, where $f(id)$ is the number of steps taken in error collapsing which is bounded by $1 \le f(id) \le (id)$. More simplification will yield that the complexity

$$= O\left(N^t \times \frac{nt}{N}\right)$$

$= O\,(N^{t-1} \times nt\,)$,

which is a polynomial in N and a linear expression in n.

The complexity of the algorithms for fault locatability analysis is higher than the complexity of the detectability algorithms. In the graph theoretic model based algorithms [23] *l–fault locatability* of the system is determined by checking whether every individual fault is observable in the presence of fault patterns of cardinality ranging up to $2l$. Therefore, the complexity is $O\,(\sum_{i=1}^{i=2l} \begin{bmatrix} N \\ i \end{bmatrix} \times 2^{id} \times N \times 2^{d}\,) = O(N^{2l+1} \times 2^{\frac{(t+1)n}{N}}\,)$. Due to the new sufficient condition established in Theorem 4.2 and due to the error collapsing technique, the complexity of the algorithm based on the matrix model is only $O\,(\sum_{i=1}^{i=l} \begin{bmatrix} N \\ i \end{bmatrix} \times f\,(id) \times N \times f\,(d) + \begin{bmatrix} N \\ l \end{bmatrix}^2 \times f\,(ld)) = O\,(N^{l-1} \times l \times n^2 + N^{2l-1} \times ln)$.

Here the second term corresponds to the complexity involved in comparing faults of cardinality l among themselves. It may be noted that the complexity is a polynomial in N and a quadratic in n.

4.6. Examples for the Applications of the Model

In this section, we present a few carefully selected examples to illustrate the application of the model for the analysis of few realistic fault-tolerant architectures.

EXAMPLE 4.5. Consider matrix multiplication using checksums on a mesh connected processor array. The fault tolerance scheme has been proposed by Huang and Abraham [47]. We will briefly describe the system below.

Multiplication of two 3×3 matrices X and Y is done on a mesh connected processor array as shown in Figure 4.5(a). We assume that input data elements are broadcast on buses; the processors input the data from the buses. Under a processor failure, we assume that only the corresponding data element of the output matrix Z becomes erroneous. After the computation, the result Z resides in the local memories of the processors. Now the checking operations (six of them, three for rows and three for columns) are performed.

Thus, we have

$DATA\ (P_i) = d_i$ for $i = 1, 2, \ldots 9$.

$CHECK(d_1, d_2, d_3) = C_1$

$CHECK(d_4, d_5, d_6) = C_2$

$CHECK(d_7, d_8, d_9) = C_3$

$CHECK(d_1, d_4, d_7) = C_4$

$CHECK(d_2, d_5, d_8) = C_5$

$CHECK(d_3, d_6, d_9) = C_6$.

First, we do the analysis of the system assuming that the check evaluating processors, P_3, P_6, P_7, P_8 and P_9 are fault secure. In that case, the fundamental matrices of the system will be

$PD = I_9$, where I_9 is the identity matrix of order 9.

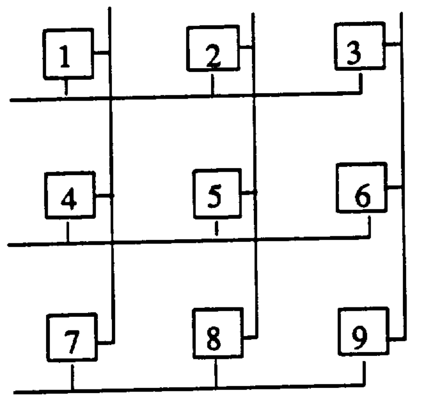
$$DC = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

$$PC = PD \times DC = DC, \text{ since } PD = I_9.$$

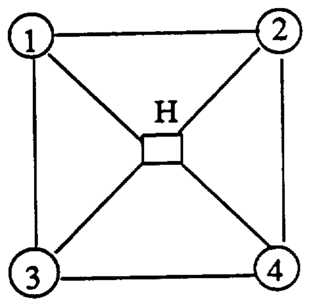
During the analysis of the system for *t–fault detectability*, we see that ^{r}PC is completely detectable for $r = 1, 2, 3$. In ^{4}PC the row R corresponding to the sum of rows P_1, P_2, P_4 and P_5 of PC is [2 2 2 2 0]. Since error detectability of checksum encoding is 1 (i.e., h=1) none of the elements in R is valid. Therefore, R and hence ^{4}PC is not completely detectable. Then, by Theorem 4.1, the system is 3–*fault detectable.* □

Next, we will consider the fault locating capability of the algorithm. Since the matrix PC has a $0-1$ *disagreement* with itself, the algorithm is 1–*fault locatable.* In the analysis procedure we observe that PC does not have $1-0$ *disagreement* with ^{2}PC. For example, row P_1 of PC is [1 0 0 1 0 0] and this does not have a $1-0$ *disagreement* with the row P_2P_4 (i.e., the sum of P_2 and P_4) of ^{2}PC which is equal to [1 1 0 1 1 0]. Hence the system is at most 2-fault locatable. As a next step we check whether all faults of cardinality 2 are distinguishable. For that, ^{2}PC should have $0-1$ *disagreement* with itself. One can observe that rows P_1P_5 = [1 1 0 1 1 0] and P_2P_4 = [1 1 0 1 1 0] of ^{2}PC do not have $0-1$ *disagreement.* Hence ^{2}PC does not have $0-1$ *disagreement* and the system is 1–*fault locatable* by Theorem 4.3.

Now we will analyze the same system with the assumption that the check evaluating processors are not fault secure. According to the definition of the fundamental matrices, we attach one *pseudo–data* element each to every check evaluating processor. It may be observed that the system is 0–*fault* detectable and 0–*fault* locatable. This is because, the data element d_9 produced by processor P_9 is checked only by processor P_9. If there occurs a fault in P_9, all the checks done by P_9 are considered to be invalid and hence the



(a) Mesh connected array



(b) 4-node hypercube

Figure 4.5. Processor arrays.

error in d_9 cannot be detected. On the other hand, from the description of the algorithm it may be noticed that d_9 is not a useful data element as far as the original matrix multiplication is concerned. Therefore, an error in d_9 may be disregarded during the analysis, which effectively means that the element d_9 can be taken off from the model. As far as check evaluation is concerned, processor P_9 checks the correctness of data elements d_3, d_6, d_7, and d_8 which are also not useful data elements. Thus, the processor P_9 is not doing any useful job in terms of computation or check evaluation. Therefore, we remove P_9 from the model. As mentioned before, since the actual data elements produced by other check evaluating processors are also not useful for the original matrix multiplication, they are also discarded. Therefore, the final model should be such that each check evaluating processor has only one pseudo-data element associated with it. Accordingly, the fundamental matrices are

$$PD = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \infty & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \infty & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \infty & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \infty \end{bmatrix}$$

$$DC = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The complete analysis shows that the system is 2–*fault* detectable and 1–*fault* locatable.

EXAMPLE 4.6. In this example we analyze an algorithm for fault-tolerant matrix multiplication using checksum encoding done on a hypercube. We consider partitioned matrix multiplication done on a 4-node hypercube as shown in Figure 4.5 (b). In the figure, the circles represent the hypercube nodes and the square represents the host processor. In the fault-tolerant algorithm suggested in [15], processor 1 checks the correctness of the data computed by processor 2 and sends a "pass" or "fail" signal to the host processor. At the same time processor 2 checks the data computed by processor 1 and sends a signal to the host. Similarly, processors 3 and 4 also check each other and notify the host of the result. □

Here, even though every processor P_i for $i = 1, 2, 3, 4$, produces data d_i, it is not necessary to include them in the PD matrix. This is because the check by the host processor is done only on the flag signals generated by the node processors. Let e_i be the flag signal generated by P_i. Then, from the description of the algorithm, we have

$CHECK(e_1, e_2) = C_1$

$CHECK(e_3, e_4) = C_2$.

Since both the checks are resident in the host processor, if the host processor fails, all checks performed in the system will become invalid, and in that case it is 0–*fault* detectable. However, we are interested in the fault tolerance of the hypercube nodes, provided the host processor does all the checks correctly. Now the PC matrix is

$$PC = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}.$$

It can be seen that the system is 1–*fault detectable* and 0–*fault locatable*. However, if the mutual checking processor pairs are changed in the fault-tolerant algorithm, that is if instead of 1, 2 and 3, 4, checking is done by pairs 1, 3 and 2, 4 and flag signals sent to the host processor, in effect we are adding two more checks given by

$CHECK(e_1, e_3) = C_3$

$CHECK(e_2, e_4) = C_4$.

The new PC matrix is

$$PC = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}.$$

Carrying out a similar analysis we found that the the algorithm is 3–*fault detectable* and 1–*fault locatable*. This example, was specifically chosen in order to illustrate the importance of selecting data elements for the PD matrix so that the analysis will be easier.

EXAMPLE 4.7. In this example we consider the Advanced Onboard Signal Processor (AOSP) architecture [48]. The AOSP is an architectural concept for an advanced signal processing computer that provides a fault-tolerant environment capable of supporting a wide range of signal processing applications. It is a loosely-coupled distributed multiprocessor system in which a large number of identical processors known as Array Computing Elements (ACEs) communicate both data and control information via packetized messages over networks of high-speed buses.

In order to achieve fault tolerance, one may incorporate some kind of system level fault diagnosis in the AOSP architecture. In this example we consider an AOSP with system level fault diagnosis. The encoding scheme to be used depends on the particular signal processing application for which AOSP is used. In the example we do not assume any particular computation or encoding scheme. The only assumption made is that the encoding scheme can detect one error (i.e., h=1).

The architecture of the AOSP is depicted in Figure 4.6. Due to the high density of the interconnection network, we have the luxury of having a fault-tolerant scheme in which any arbitrary processor can check the correctness of the computation done by any other processor. As an example we consider a scheme in which

$CHECK(d_8, d_4, d_3) = C_1$

$CHECK(d_8, d_6, d_1) = C_2$

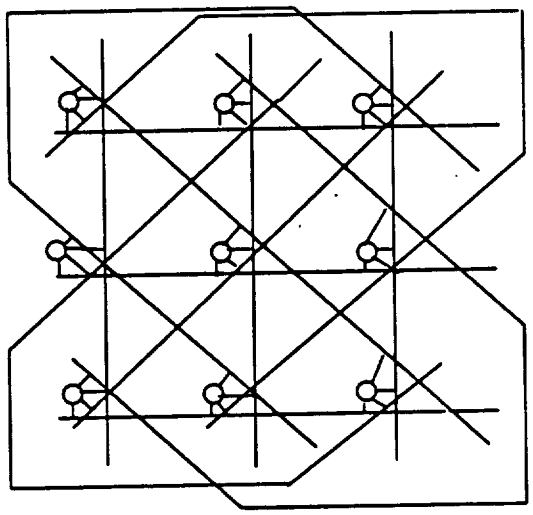


Figure 4.6. AOSP architecture.

$CHECK(d_4, d_2, d_9) = C_3$

$CHECK(d_1, d_5, d_9) = C_4$

$CHECK(d_3, d_5, d_7) = C_5$

$CHECK(d_3, d_5, d_9) = C_6$.

Suppose that the checks c_1 through c_6 are performed by processors P_1, P_2, P_3, P_4, P_6, and P_7, respectively. The analysis shows that the system is 1–*fault* detectable and 0–*fault* locatable. □

EXAMPLE 4.8. Consider an 8 node hypercube (3-cube) performing partitioned vector computations. Each node computes three partitions (subdivisions) of the same vector. After the first set of computations, the partial results are rotated in the clockwise direction in the lower dimension (involving four processors each) for further iteration as shown in Figure 4.7. Each computed part is check evaluated by three neighboring processors in their order (i.e., the first neighbor checking the first part, the second neighbor checking the second part and so on). It can be determined that the fault detectability of the system is 1, 2 and 5 for an error detectability of 1, 2 and 3, respectively.

4.7. An Alternative Approach to Check Invalidation

In this section we reconsider the problem of invalidation of checks, performed by faulty processors and develop an alternative method to handle the problem.

In this approach the analysis procedure is divided into two steps; (1) primary analysis, and (2) secondary analysis.

DEFINITION 4.15. *Home Processor* of a check is defined as the processor which performs that checking operation.

The primary analysis consists of analyzing the system with the assumption that a check will not be invalidated if its *home processor* is faulty. In the secondary step, some additional information related to the correspondence between processors and checking

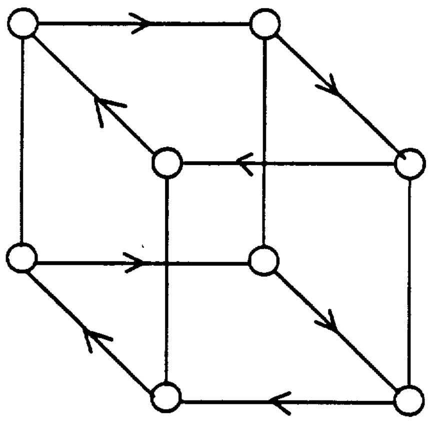


Figure 4.7. Data rotation in the hypercube.

operations performed by them is derived. With the help of the results obtained in the primary and secondary analyses, the actual fault tolerance capabilities of the system are determined. Even though this approach is more tedious than the one using *pseudo data elements*, it has the advantage that it will induce an easier design technique; first design the primary system and then decide the home processors for various checking operations using the properties we derive in the next section.

4.7.1. Secondary analysis

Before going into the details of the procedure, we define the following parameters associated with a fault-tolerant multiprocessor system.

DEFINITION 4.16. *Self-Tested Set* (*STS*) is defined as a set of processors such that at least for one particular possible output error combination of these processors, every valid check done on these processors is resident in that set itself.

EXAMPLE 4.9. Consider a system described as

$DATA(P_1) = \{d_1, d_2, d_3\}$

$DATA(P_2) = \{d_2, d_4\}$

$DATA(P_3) = \{d_5\}$

$DATA(P_4) = \{d_6\}$

$CHECK(d_1) = \{C_1\}$

$CHECK(d_2) = \{C_2\}$

$CHECK(d_3) = \{C_1\}$

$CHECK(d_4) = \{C_2, C_3\}$

$CHECK(d_5) = \{C_1\}$

$CHECK(d_6) = \{C_2, C_3\}$.

Now assume that check c_1 is resident in processor P_2, c_2 in P_3, and check c_3 in P_4. It can be observed that if processors P_2 and P_3 are faulty and if the corresponding error pattern is d_2, d_5, the valid check operations done on the output error pattern are c_1 and c_2. These two checks are resident among processors P_2 and P_3 and hence the set $\{P_2, P_3\}$ is an *STS*.

In further discussions, the cardinality of an *STS* is denoted by S.

DEFINITION 4.17. An *STS* is called a ***minimal*** *STS* if removal of at least one processor from that set will destroy the property of the *STS*.

DEFINITION 4.18. S_{min} is defined as the cardinality of the smallest minimal *STS* of the system.

Let f be a fault pattern involving processors $P_1, P_2, \ldots P_i$ and let $c_1, c_2, \ldots c_j$ be the checks which give valid output (that is, detect the fault) when all the data elements produced by the faulty processors are erroneous. Three cases may arise as described below.

Case 1.

The checks $c_1, c_2, \ldots c_j$ are resident in the processors of set f. In that case, set f is an *STS*.

Case 2.

Among the set of valid checks, some of them are resident in f and some of them are not resident in f. This does not guarantee that f is not an *STS*, since there may exist a particular error pattern for which f is an *STS*. In order to check that, enumerating all error combinations will be inefficient. Instead, we propose an error collapsing technique.

(Distinguish this error collapsing technique from the one described in Section 3.)

Case 3.

All the valid checks are resident outside the set *f*. This is a special instance of *Case* 2, where the number of checks resident in set *f* is equal to zero.

4.7.1.1. Algorithm to check whether *f* is an *STS*

The procedure is given in ALGORITHM 3.

In order to simplify the implementation of the algorithm for determining whether a given set of processors form an *STS*, we define one more model matrix called an *H* (home) matrix which gives the relationship between processors and checking operations resident in them.

DEFINITION 4.19. The *H* matrix is an $n \times l$ matrix such that

$$H_{ij} = \begin{cases} 1 & \text{if } C_j \textit{ is resident in } P_i \\ 0 & \textit{Otherwise} \end{cases}$$

ALGORITHM 3.

(1) Collapse errors which are checked by those checks which are not resident in *f*.

(2) If at least one processor in *f* is left with no output error at all, then *f* is not an *STS*. Otherwise go to step 3.

(3) Find the new set of valid check elements. If all of them are in *f* then the situation is equivalent to *Case* 1, and *f* is an *STS*. Otherwise go to step 1.

DEFINITION 4.20. The matrix ${}^{r}H$ is defined as the one whose rows are formed by adding r different rows of matrix H, for all possible different combinations of r rows.

It may be noted that the PC matrix and the H matrix will have the same dimensions, and hence ${}^{r}PC$ and ${}^{r}H$ will also have same dimensions.

DEFINITION 4.21. (Covering) A valid check is said to be covered with respect to row R of ${}^{r}PC$ if row R of ${}^{r}H$ has a *one* in the corresponding position.

DEFINITION 4.22. A row R of ${}^{r}PC$ is said to be *covered* if all the valid entries in that row are covered.

DEFINITION 4.23. A row R of ${}^{r}PC$ is said to be *completely covered* if row R is covered for at least one possible error syndrome.

The physical significance of covering is that if a check is covered with respect to R, the check operation is resident in the processor set R. If row R is covered, then all the valid check operations are resident in the processor set R itself when all the output data elements are erroneous. Complete covering implies that all the valid check operations are resident in set R itself for at least one possible output error combination.

LEMMA 4.2. A processor set f is an *STS* if and only if it is *completely covered.*

PROOF: Proof follows from the definitions of *STS* and *complete covering.* □

Now the previous algorithm to determine the *STS* nature of a processor set can be restated and implemented in terms of *covering*. The algorithm is called the *STS* ALGORITHM.

EXAMPLE 4.10. In Example 4.9, given to illustrate the property of *STS*, the PC and the H matrices are

STS **ALGORITHM**

To check whether f is an *STS*.

(1) Collapse errors which are not covered with respect to f.

(2) If at least one row of the PD matrix is zero, then f is not *completely covered*, and hence not an *STS*. Otherwise go to step 3.

(3) Check whether the new syndrome obtained after error collapsing is covered. If so, f is an *STS*; otherwise go to step 1.

$$PC = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \quad H = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Consider the row [1 2 1] of ^{2}PC (sum of 2nd and 3rd rows of PC), the corresponding row in ^{2}H is [1 1 0]. If $h = 1$, only the third check is uncovered. If we collapse the corresponding error, the new syndrome will be [1 1 0] which is covered. Therefore, the row [1 2 1] is *completely covered*, and hence, processors P_2 and P_3 form an *STS*.

THEOREM 4.4. If t is the fault detectability of a system obtained after primary analysis, any fault pattern of cardinality $\leq t$ is undetectable if and only if the set of processors involved in the fault is an *STS*.

PROOF:

Proof for the sufficiency condition follows from the definition of *STS*.

Proof for the necessary condition, (by contradiction): Let the fault pattern F be undetectable, at the same time F is not an STS. By the definition of an *STS*, this implies that set F

is such that for every output error pattern, there exists at least one valid check operation which is resident in a processor outside the set F and hence the fault is detectable. □

THEOREM 4.5. The actual fault detectability of the system $t_{act} = \min(t,\ S_{\min} - 1)$, where t is the value of fault detectability obtained after primary analysis.

PROOF: Proof of the theorem follows from Lemma 4.2. □

4.7.2. Analysis to determine actual locatability

The actual locatability l_{act} of a system can be determined by a similar type of analysis. Instead of *STS* we define another type of processor set called *Self-Locating Set* (*SLS*).

DEFINITION 4.24. *Self Locating Set* (*SLS*) is a set of processors for which at least one output error combination exists such that all the valid check operations, which distinguish faults in these processors from all other faults of cardinality less than or equal to the cardinality of the processor set, are resident in the given processor set itself.

In the following, we formulate an algorithm to determine whether the given set is an *SLS*.

Let $f = \{P_1, P_2, \ldots, P_r\}$. That is, $f \in {}^r\text{PC}$. Now to check whether f is an *SLS* we use the *SLS* ALGORITHM.

THEOREM 4.6. Actual fault locatability l_{act} of a system is

$$l_{act} = min\,(l,\ SL_{\min} - 1),$$

where l is the fault locatability obtained in the primary analysis and $S_{\min}$ is the cardinality of the smallest minimal *SLS*.

SLS ALGORITHM

For all rows P_is in the PC matrix which are also
elements of the set f

(1) Find all the checks which causes complete 1–0 *disagreement* with the row $f - P_i$ (ie., the set f excluding P_i) of ${}^{r-1}$PC.

(2) If all checks are covered with respect to f, then f is an *SLS*. Otherwise, go to step 3.

(3) Find all the checks which are not covered, and collapse corresponding errors. In any stage if row P_i of PC becomes zero, then f is not an *SLS*. Otherwise go to step 2.

PROOF: Proof of the theorem is similar to the proof of Theorem 4.4. □

EXAMPLE 4.11. In this example we present a complete analysis of a system using the secondary analysis we developed in this section. We consider the fault-tolerant AOSP architecture illustrated in Example 4.7. The check operations are distributed among the processors in such a way that the H matrix is

$$H = \begin{bmatrix} 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&1 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&1&0&0&0&0 \\ 0&0&0&0&1&0 \\ 1&0&0&0&0&0 \\ 0&0&1&1&0&0 \end{bmatrix}.$$

From the primary analysis of the system, using the fundamental matrices, we have already found that the system is 3–*fault detectable* and single fault locatable; i.e., $t = 3$ and $l = 1$.

Secondary analysis: It may be observed that none of the rows of PC are covered, whereas the row corresponding to the sum of rows P_8 and P_6 in ^{2}PC is [1 1 0 0 0 0] and is covered. (Note that the corresponding row in ^{2}H is also [1 1 0 0 0 0].) Thus, $S_{\min} = 2$ and by Theorem 4.4, actual fault detectability $t_{act} = min(3, 1) = 1$.

According to the check distributions, check c_1 is the one and only check which distinguishes the faults in processors P_6 and P_8. However, this check is covered by the 8th row of the H matrix and hence $SL_{\min} = 1$. The actual fault locatability $l_{act} = \min(1, 0) = 0$.

4.8. Further Extensions

Applications of the matrix-based model for the diagnosis of faults in a fault-tolerant system were further investigated by Vinnakota and Jha in [49].

The diagnosis problem is defined as, given a syndrome S, determine which fault produced that syndrome. The authors used the matrix-based model for locating and identifying a fault, once its presence is detected. They observe that those faults which do not have 1-0 disagreement with every row of $^{l-1}\text{PC}_s$ are not elements of the candidate fault pattern. This is borne out from the fact that for every locatable fault F, every processor in F is checked by at least one valid check that does not check any other processor in F. Based on this observation, an algorithm has been proposed. However, this algorithm tackles the problem in a roundabout fashion. Therefore, we suggest a straightforward algorithm for identifying the fault present.

DEFINITION 4.25. The matrix PC_S is obtained by deleting all the columns of the PC matrix, corresponding to the 0's in the syndrome S and then deleting all the resulting zero rows.

DEFINITION 4.26. PC_{S_r} is the matrix obtained by deleting the rows in PC_S which do not have a complete 1–0 disagreement with ${}^{1\text{-}1}PC_S$. □

4.8.1. Description of the diagnostic algorithm

Motivation behind performing up to step 3 is obvious from the definitions of the matrices PC_S and PC_{S_r}. In PC_{S_r} we have included only checks which are flagging a valid "error" output (that is the output is 1). Therefore, if the number of rows in PC_{S_r} is $\le l$, there cannot be any row representing a nonfaulty processor. If the number of rows in PC_{S_r} is $> l$, the size of the fault present cannot be larger than l by definition of *l–fault locatability*. However, we have to check whether the fault present is of cardinality $< l$. In the following discussion we find that that is also not possible.

If possible, let the fault pattern F be of size less than l. Again, by the definition of PC_{S_r}, F is checked by all the checks in PC_{S_r}. Then for any processor p in PC_{S_r} which is

---

DIAGNOSTIC ALGORITHM

(1) Compute PC_S for the given syndrome S.
(2) Compute ${}^{1\text{-}1}PC_S$.
(3) Compute PC_{S_r}.
(4) If the number of rows in PC_{S_r} is $\le l$, then the processors corresponding to the rows of PC_{S_r} represent the components of the required fault pattern. Otherwise go to step 5.
(5) Compute ${}^{1}PC_{S_r}$. The row which corresponds to the given syndrome represents the fault to be diagnosed.

---

fault-free, the set of checks that check that processor are checking the processors in the fault pattern F which are also fewer than l in number. This contradicts the conditions for *l–fault* locatability. Therefore, the only possibility is that the fault pattern that we are looking for is of cardinality l.

Now to determine exactly which fault pattern has occurred, we have to consider all the *l–combinations* of the faults represented by the rows of PC_{S_r}. Then, by checking the equality of the rows of ${}^{l}PC_{S_r}$ with the given syndrome, we can conclude which fault has been present in the system.

It should be noted that the diagnosis procedure described here needs an a priori knowledge of the fault locatability of the system; one can determine the locatability of the system using Algorithm 2 given earlier in this chapter. In fact, the diagnosis procedure could be integrated with the procedure for determining the fault locatability of the system.

4.9. Results and Conclusions

The concept of concurrent fault diagnosis involves the application of checking operations on the data generated by multiprocessor systems to obtain reliable results. We have proposed a new matrix-based model for analyzing the fault-detecting and -locating capability of such systems. A uniform framework was constructed in which faults in the processors performing useful computations can be treated along with faults in the processors evaluating the checks. The necessary and sufficient conditions for fault detection and location were derived. The algorithms based on these derived conditions are less complex than the existing algorithms because of the *error collapsing* technique we introduced and because of the simpler sufficiency conditions. The algorithms were used to determine the fault detectability and locatability of some realistic systems.

These algorithms have been implemented in *C* under UNIX on SUN workstations. The program takes the algorithm/system description as its input. The program analyzes the system for its fault detectability first, and then uses that result to evaluate the locatability. Some typical run times are given in Table 4.1 for some of the example systems discussed in the preceding sections. In the table, S_1 through S_4 are the systems described in Examples 4.6, 4.5, 4.7, and 4.8, respectively.

Table 4.1. Typical run times for the fault diagnosis program

Example	#P (=N)	#d (=n)	h	t	l	Runtime (sec)
S_1	4	4	1	3	1	3.1
S_2	8	8	1	2	1	6.0
S_3	9	9	1	1	0	3.6
S_3	9	9	2	5	1	118.3
S_4	8	24	3	2	0	8.9

#P is the number of processors
#d is the number of data elements

CHAPTER 5.

DESIGN OF ABFT SYSTEMS

5.1. Introduction

There are two ways to approach the problem of designing algorithm-based fault-tolerant systems: (1) given a non-fault-tolerant system, determine an efficient distribution of checks among the output data elements so that the system has the desired amount of fault tolerance; (2) given a fault-tolerant algorithm, synthesize an architecture so as to maximize quantities such as the fault detectability and locatability of the system. Both the approaches have their advantages and disadvantages. In the first approach, the fault-tolerant design is constrained by the fixed, non-fault-tolerant architecture. In the second approach, performance may be sacrificed in the process of achieving high fault tolerance.

Since most of the commercially available multiprocessors are built to maximize their performance, usually they do not carry any fault tolerance capabilities as such. According to the requirements of the application, it is up to the fault tolerance designer to make the system fault-tolerant. Therefore, in practice, the designer is forced to adopt the first approach. This is the philosophy followed by previous researchers also [50,51]. Since the first approach is immediately applicable to existing architectures, we also look at the problem from the first point of view. However, our methodology is different from the existing ones.

The design of fault-tolerant multiprocessors that have processors producing more than one data element by modifying the non-fault-tolerant architectures was considered to be an intractable problem [50]. In that study, in order to design a system with the required fault tolerance capabilities, first the cardinality of the largest error pattern generated by all possible faults is determined and the system is designed to detect and locate that number of errors. In this thesis, we propose a direct scheme to design such systems which eventually results in a smaller number of checks compared to the previous methods. The design procedure is illustrated with examples. A comparison between the existing schemes and the new scheme is done with respect to the number of checks required for each scheme.

5.2. Previous Work

Previous studies done by Banerjee and Abraham [50] and then by Rosenkrantz and Ravi [51] were geared towards computing the bounds on the number of checks required to be attached to a given non-fault-tolerant architecture in order to make it fault-tolerant in the desired amount. In the first study, bounds were derived for the number of checks required for the desired amount of fault detectability and locatability. The bounds for the detectability were later enhanced in the second study.

In both cases, bounds were developed through algorithmic procedures to construct such a system. The design of systems that have processors producing multiple data elements was considered to be an intractable problem. As a solution, they suggested an indirect approach. In order to design a system for *t–fault detectability*, first the size of the largest error pattern (let it be equal to s) for all fault patterns was determined and the system was designed for an error detectability of s. As an example, consider a non-fault-tolerant multiprocessor system consisting of 4 processors. Processor P_1 produces 3 data elements, P_2, 2 data elements and P_3 and P_4 produce one data element each. If the system is to be designed to be 2–*fault* detectable, first all the fault patterns of cardinality two are enumerated and their corresponding error patterns are determined. Then, the size of the largest error pattern is computed; in the example it is 5. (Note that the size of the largest error pattern caused by faults of cardinality $\leq t$ is always less than or equal to the size of the largest error pattern caused by faults of cardinality t. Therefore, one needs to consider only fault patterns of cardinality t in order to determine the size of the largest error pattern.) Now the system is designed to have an error detectability of 5.

The lower and upper bounds for the number of checks required were calculated in terms of g, h, and s. In the following subsection we give some sample bounds derived in [50] and [51].

5.2.1. A few sample bounds

As examples, we provide the bounds derived for 2–*error detectability* and 3–*error detectability*. It was shown in [51] that at least $2n/(g+1)$ checks are necessary to detect 2 errors. Rozenkrantz and Ravi showed that $\lceil 2n/(g+1) \rceil$ checks are sufficient for detecting 2 errors. For 3–*error detection* also $\lceil 2n/(g+1) \rceil$ is a trivial lower bound. The upper bound for this case derived in [50] was $qn/q+g-1$ where $q = \lceil (3g+1)/2 \rceil$. This bound was later improved in [51] to a higher value $\lceil (2n - \lfloor n/g \rfloor)/g \rceil + 1$.

5.2.2. Limitations

With the modern VLSI technology, individual nodes of multiprocessor systems are capable of having high computing powers. Every processor in the system may be computing a large volume of data which in turn means that the size of the error patterns produced by a faulty processor may be large. Since the designs are done for meeting the error detectability criterion, this will result in large complexities.

The methodology is efficient only when most of the *t–faults* are producing error patterns of cardinality s. On the contrary, if only a small number of fault patterns produce error patterns of cardinality s, the design procedure will be using larger number of checks unnecessarily. The approach lacks flexibility with respect to varying amount of computation performed by a processor. For example, if a system were designed for an error detectability of s and later one of the processors is assigned a bigger load of computing more data elements, in this method, the whole system has to be redesigned for the new error detectability (to maintain the original value of the fault detectability). Unfortunately, both these scenarios exist in real life. It was reported in [48] that in the AOSP architecture, every computing node in the structure can support a wide variety of signal processing computations and often, the amount of computation performed by various nodes is not the same. If we incorporate system level diagnosis in this case, inefficient designs will result as mentioned in the beginning of the paragraph.

In [50] and [51] the problem of minimizing the number of (g, h) checks was transformed into a problem of constructing a bipartite graph where the number of output nodes is minimized subject to the constraints of *t–fault* detection. Instead of using (g, h) checks, checks of type $(g', 1)$ were used where $g' = \lfloor g/h \rfloor$. After constructing a graph subject to the modified constraints, groups of h output nodes are merged together. The new merged graph satisfies the constraints of (g, h) checks. The limitation of this approach is that even though the merged graph preserves the fault detectability of the original graph, the locatability may not be preserved. In other words the design cannot handle detectability and locatability constraints simultaneously.

5.3. A New Approach for the Design of FTMP Systems

In this thesis, we propose a straightforward design procedure which needs a smaller number of checks than the previous techniques especially when the computation is nonuniformly distributed among the processing nodes. In our approach the system is designed directly for meeting the fault constraints rather than error constraints. Also the methodology can handle both detectability and locatability issues simultaneously. The use of the matrix-based model allows the use of simple vector space techniques to identify redundant checks. The flexibility involved in handling varying the amounts of computations performed by individual processor nodes is another advantage of our approach.

5.3.1. Problem definition

Using the matrix-based model parameters, we define the design problem as follows: Given the PD matrix, find a DC matrix so that the corresponding PC matrix has the required fault diagnosing capabilities.

Since PC = PD*DC, the design involves finding two variables (actually two matrices) from a single equation. Therefore, the solution is not unique. The selection of a particular solution should optimize the number of checks required, and the number of errors detectable and correctable by each check (assuming that the cost of each check increases proportionally to the number of errors it can detect and correct). Our approach to the problem consists of the following steps.

(1) Design a DC matrix for PD = I_m and $h = 1$, such that the system has *t-fault detectability* and *l-fault locatability*. Here I_m is the identity matrix of order m, where m is the number of processors in the system to be designed. We call this system the ***unit system*** of the actual fault-tolerant system to be designed.

(2) Modify the DC matrix of the unit system according to the given PD matrix in order to obtain the DC matrix of the actual system.

In the unit system, since the PD matrix is the unit matrix, every processor is producing only one data element. Therefore, the cardinality of the fault patterns will be the same as the cardinality of the resulting error patterns. As mentioned before, in such a situation, the techniques proposed in [50, 51] are efficient and can be used for the design of the unit system. Designs are already available for various values of fault detectabilities and locatabilities. These designs of the unit systems can be used as a template. Now the actual design consists of modifying these template designs to obtain the actual system.

DEFINITION 5.1. The *Product System* of a given non-fault-tolerant system and the corresponding fault-tolerant unit system is defined as the system obtained by connecting every data element affected by processor P_i in the non-fault-tolerant system to every check element in the unit system which checks the output of processor P_i. □

The construction of the product system is illustrated in the following example.

EXAMPLE 5.1. Let us consider a multiprocessor system with four processors. The first processor produces 4 data elements, the second one 2, and the third and the fourth processor produce only one data element each. A unit system is designed for 3–*fault detectability* and 1–*fault locatability*. Construction of the product system is given in Figure 5.1. □

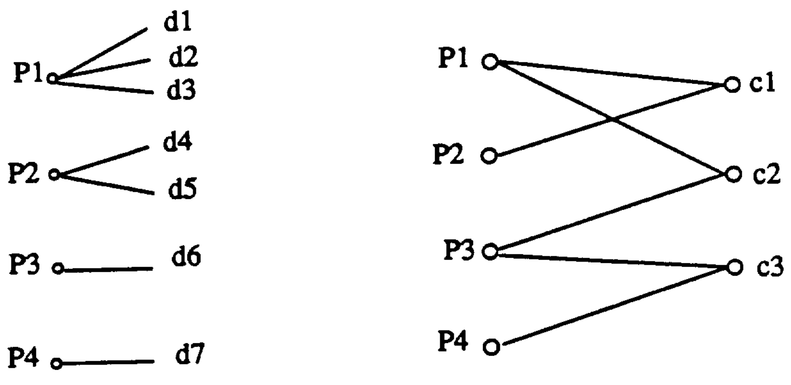
THEOREM 5.1. If the unit system is *t–fault detectable* and *l–fault locatable*, for t and $l > 0$, then the product system is *t–fault detectable* and *l–fault locatable* if and only if

$$h \geq \max\,[num\ (P_i)].$$

Here h is the error detectability of the checks in the product system and $num\ (P_i)$ is the number of data elements affected by processor P_i.

PROOF: Proof for the necessary condition, (by contradiction): Let $h < \max\,[num\ (P_i)]$, and the product system be *t–fault detectable* and *l–fault locatable*. Let processor P_j be such that $num\ (P_j) = \max\,[num\ (P_i)]$, where the maximum is computed for $i = 1, 2, 3, \ldots m$. If P_j fails in such a way that all the data elements produced by P_j become erroneous, then all the checks done on P_j in the product system will become invalid. Therefore, such a fault in P_j will not be detected and the system is 0–*fault detectable* which is a contradiction.

Proof for the sufficiency condition: Since the unit system is designed for $h = 1$, for any fault pattern of cardinality $\leq t$, there exists at least one check in the unit system which checks only one processor in that group of processors which are faulty. (In [31] this is referred to as 1-neighbor intersection property.) In the product system also, since

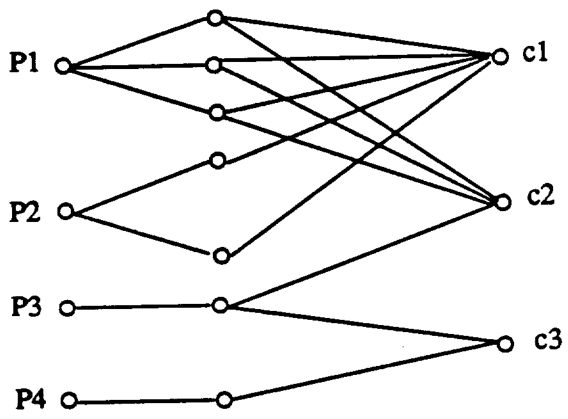


(a)

Non-fault-tolerant System

(b)

Unit System



(c)

Product System

Figure 5.1. Construction of a product system.

$h \geq \max\,[num\,(P_i)\,]$, this check will fail and hence the system has the same detectability and locatability as the unit system. Locatability of the product system can be argued in a similar fashion.

5.3.2. Construction of the actual system

Construction of the actual fault-tolerant system with given (g, h) checks is done by splitting each check in the product system into one or more checks such that every check in the resulting system has at most h data elements from each processor checked by those checks. Now the new system has the same detectability and locatability as the product system. This is because, whenever a check in the product system fails, at least one of the checks formed by splitting that check node will fail in the actual system. Actually, the detectability and locatability of the new system may be higher than the product system. However, we are interested only in the fact that the detectability and locatability of the final system are at least equal to that of the product system.

It should be noted that in the procedure described above, instead of combining h checks in the unit system to form a system having checks of error detectability h, we attach h data elements from every processor to that check. This approach will preserve the fault detectability and locatability of the unit system even after converting it into the final system. However, when most of the processors are producing only $<h$ data elements, the design may not be efficient. In those cases, the unit system itself may be designed by assuming that the checks have error detectability h. Correspondingly, while constructing the final system from the product system, the checks should be split in such a way that every check receives at most one data element from the processors which are being checked by that check.

Another point of interest is the assignment of checks to the processors, that is, which processor performs which check. Once the assignment is decided for the *unit system* the same assignment should be followed in the product system also. When checks in the product system (*parent* checks) are split into component checks, all the component checks are assigned to the same processor which was hosting the *parent* check in the product system.

In the following, we present the design procedure as an algorithm, in terms of the matrix-based model parameters. The resulting DC matrix (and the corresponding PC matrix) has the required fault diagnosing capabilities. The DC matrix can be further simplified by deleting some of the redundant columns as follows.

DEFINITION 5.2. Column C_i (i.e., check C_i) is said to be covered by one or more columns if and only if C_i can be written as a linear combination (with coefficient of multiplication equal to 1) of those columns.

DESIGN ALGORITHM

(1) Construct a DC matrix for the unit system (we call it the unit DC matrix), so that the unit system is *t-fault detectable* and *l-fault locatable.*

(2) The DC matrix for the product system (called the product DC matrix) is constructed by expanding the columns of the unit DC matrix vertically. The row corresponding to processor P_i in the unit DC matrix is replicated *num* (P_i) times.

(3) The DC matrix of the product system is partitioned into blocks of rows, such that the i^{th} block contains data elements produced by processor P_i.

(4) Each column in the product DC matrix is split into a minimum number of columns so that every column has at most h number of 1's in every block.

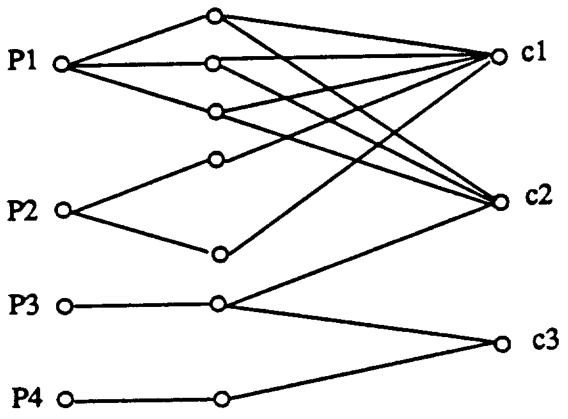
LEMMA 5.1. If C_i is covered by $C_{j1}, C_{j2}, \ldots C_{jk}$, then C_i is a redundant check and can be deleted.

PROOF: Since C_i can be obtained as a linear combination of $C_{j1}, C_{j2}, \ldots C_{jk}$, whenever C_i fails, at least one of the checks among $C_{j1}, C_{j2}, \ldots C_{jk}$ will fail. Therefore, if the system has checks $C_{j1}, C_{j2}, \ldots C_{jk}$, then check C_i is redundant and hence can be deleted. □

COROLLARY 5.1. Every column is covered by another identical column, if such a column exists.

Once the DC matrix is determined using the design algorithm, the DC matrix is further simplified by deleting all columns which are covered by some other columns. The procedure for modifying the product system to obtain the actual system is illustrated in the following example.

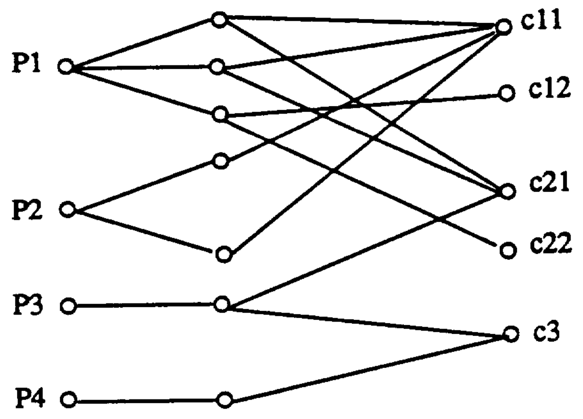
EXAMPLE 5.2. We consider the same multiprocessor system that we introduced in the previous example. We are supposed to design a fault-tolerant system consisting of all these processors such that the system will be 3–*fault detectable* and 1–*fault locatable*. In example 5.1, we have already seen how to construct the product system. The rest of the algorithmic procedure is shown in Figure 5.2. The graphical representations of the systems are shown along with their matrix representations. It can be observed in Figure 5.2 (b) that the DC matrix has two identical columns corresponding to checks C_{12} and C_{22}. Either one of those checks can be deleted from the system so that the final system has only 4 checks. □



$$\begin{array}{c|l}
\left[\begin{array}{c} 1\,1\,0 \\ 1\,1\,0 \\ 1\,1\,0 \\ \hdashline 1\,0\,0 \\ 1\,0\,0 \\ \hdashline 0\,1\,1 \\ \hdashline 0\,0\,1 \end{array}\right] & \begin{array}{l} \\ \text{Block 1} \\ \\ \text{Block 2} \\ \\ \text{Block 3} \\ \text{Block 4} \end{array}
\end{array}$$

(a)

Product system and the corresponding DC matrix



$$\left[\begin{array}{cc:cc:c}
1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 \\
\hdashline
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
\hdashline
0 & 0 & 1 & 0 & 0 \\
\hdashline
0 & 0 & 0 & 0 & 1
\end{array}\right]$$

(b)

Fault-tolerant System for h = 2 and its DC matrix

Figure 5.2. Design of the final system from the product system.

5.3.3. Comparison with previous schemes

In order to make a comparison between the newly proposed scheme and the existing ones, we consider the following example. The comparison is done with respect to the number of checks required in the design.

EXAMPLE 5.3. Suppose a system involving 500 processors has to be designed for 3–*fault detectability* and 1–*fault locatability*. The checks available are of type (5,1) (i.e., g = 5, and h = 1). It is also known that 10 of those 500 processors produce 2 data elements each.

Number of checks required as per the bounds given in [50] : 408

Number of checks required as per the bounds given in [51] : 363

Number of checks required for our scheme : 200 □

5.4. Conclusions

A systematic and straightforward design methodology is proposed for the design of FTMS where individual processors may produce multiple data elements. Our approach is to transform the non-fault-tolerant system directly to satisfy the fault detectability and locatability constraints to obtain a fault-tolerant system. The new scheme is more efficient than the previous schemes with respect to the number of checks used in the overall system. Examples are provided for the illustration of the design methodology and for the comparison of various schemes.

5.4.1. An alternative approach

For completeness of the thesis we briefly describe an alternative approach for designing a fault-tolerant system by mapping a fault-tolerant algorithm on a suitable multiprocessor architecture which will minimize the overhead [52]. To that end, a dependence graph-based approach has been suggested by Vinnakota and Jha in [52].

In the first stage of the design process, a particular encoding scheme is selected to meet the fault tolerance specifications. In the second stage, an optimal architecture to implement the scheme is chosen using dependence graphs.

Dependence graphs are graph-theoretic representations of algorithms [53]. After the first stage of the design, the encoded algorithm is represented as a dependence graph. This graph is then projected in several directions to obtain different realizations of architectures, among which the one with the optimal features is chosen. It was demonstrated that not all architectures are suitable for a particular ABFT scheme. In this study the authors claim that their approach is architecture independent. However, most of the cost-effective fault tolerance algorithms known until now are architecture specific. Therefore, the selection of a particular algorithm dictates the selection of the architecture also.

CHAPTER 6.

HIERARCHICAL DESIGN AND ANALYSIS

6.1. Introduction

The complexity of the detectability algorithm, based on the matrix model, is linear in the number of data elements, whereas the complexity of the locatability algorithm is quadratic in the number of data elements in the system. Even though these complexities are less than the complexities of previous algorithms [23], the computation may require a large amount of time and memory when the system has a large number of processors producing huge volumes of data. This motivates the development of a hierarchical approach to analysis which will reduce the complexity of the algorithms to a polynomial in the logarithm of the number of processors in the system.

A natural way to build large systems is to first build small units and then to construct bigger units from the small units in a hierarchical fashion. This principle has been followed in the design of most of the existing large multiprocessor systems. That is, a small unit is replicated many times with a systematic method of interconnection. For example, a two-dimensional mesh connected processor array may be considered as multiple replications of a linear array with corresponding elements of the copies connected in a linear fashion. It has been suggested that in such complex multicomputer structures, fault tolerance should also be handled in a hierarchical fashion [1]. However, even when the error detection, fault location and recovery are performed in a hierarchical fashion, analysis of these systems has conventionally been done without exploiting the hierarchy [23, 46, 18, 51].

In this chapter, we develop techniques to analyze fault-tolerant multiprocessor systems in a hierarchical fashion. The fault tolerance of the system at different levels of the hierarchy is determined separately and the overall fault tolerance capabilities are derived from those values. In order to exemplify such an approach, we first describe a type of hierarchy one may follow in order to build a large fault-tolerant multiprocessor system. Then, we develop an analytic technique that is based on a hierarchical description of the system using the matrix model mentioned earlier.

In the proposed hierarchical design, large fault-tolerant systems are constructed from smaller units (*basic units*) of known fault tolerance capabilities. Basic units (processors as well as checks) are replicated several times at the next level of the hierarchy and new checks are introduced. This procedure is repeated recursively through various levels of hierarchy. The ability to analyze different checks at different levels of hierarchy greatly simplifies the overall analysis of large systems, as we shall see in Section 6.3. We derive the relationship between the fault detectability (locatability) of the basic unit and the fault detectability (locatability) of systems hierarchically derived from the basic unit. In order to make the development of the theory simple, we first assume that the processors in the fault-tolerant systems under consideration produce only one data element each. However, the techniques developed in Chapter 5 may be used to extend the design to systems where individual processors produce multiple numbers of data elements.

The organization of the chapter is as follows. In Section 6.2 we develop the concept of independent and orthogonal checks. Section 6.3 deals with the hierarchical design and analysis of fault-tolerant systems. Our conclusions are stated in Section 6.4.

6.2. Independent and Orthogonal Checks

In this section, we develop certain properties of (g,h) checks which are described in Chapter 2. These properties are eventually used in the hierarchical analysis of systems.

DEFINITION 6.1. The *Domain* of a set of check S (denoted as $D(S)$) is defined as the set of processors that are checked by these checks.

$$D(S) = \{P_i \mid PC_{i,j} \neq \Phi \text{ ,for } all\ c_j \in S\}$$

where PC represents the PC matrix and Φ is the null set. □

DEFINITION 6.2. Sets of checks S_1 and S_2 are said to be *independent* if and only if $D(S_1) \cap D(S_2) = \Phi$. □

In Figure 6.1, A and B are the domains of sets of checks S_1 and S_2, respectively. Since $A \cap B = \Phi$, S_1 and S_2 are independent checks.

DEFINITION 6.3. $det(A)|_S$ is defined as the fault detectability of system A when it is checked by S. □

DEFINITION 6.4. $loc(A)|_S$ is defined as the fault locatability of system A when it is checked by S. □

LEMMA 6.1. If S_1 and S_2 are two independent sets of checks and A_1 and A_2 are their respective domains, then

$$det(A_1 \cup A_2)|_{S_1 \cup S_2} = \min\,[det(A_1)|_{S_1},\ det(A_2)|_{S_2}].$$

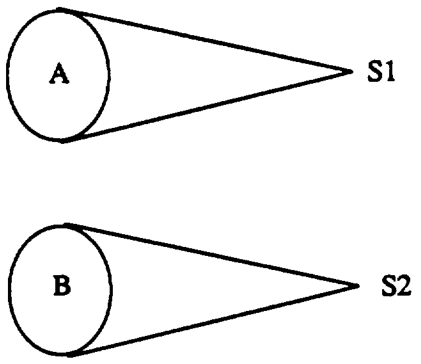


Figure 6.1. Independent checks.

PROOF: Follows from the definition of independence. □

LEMMA 6.2. If S_1 and S_2 have the same domain A, then

$$det(A)|_{S_1 \cup S_2} \geq \max\,[det(A)|_{S_1},\, det(A)|_{S_2}].$$

PROOF: Since S_1 and S_2 are applied on the same set of processors, a fault will be detected if either S_1 or S_2 detects it. Therefore, such an arrangement should be able to detect as many faults as either S_1 or S_2, whichever is larger. □

Similar results apply to the locatability of systems too. Now we find an upper limit for $det(A)|_{S_1 \cup S_2}$ and $loc(A)|_{S_1 \cup S_2}$. Intuitively, the upper limit is going to be dependent on the domains of individual checks in S_1 and S_2.

DEFINITION 6.5. Sets of checks S_1 and S_2 are said to be *orthogonal* to each other if: (1) any check in S_1 has at most one processor in common with any check in S_2; (2) for every check in S_1 there is at least one check in S_2 which shares a processor with the check in S_1. □

EXAMPLE 6.1. The set of the row checksums and the set of the column checksums applied to a mesh-connected processor array form two orthogonal sets of checks. □

LEMMA 6.3. If S_1 and S_2 are two sets of checks having the same domain A, then $det(A)|_{S_1 \cup S_2}$ is maximized when S_1 and S_2 are orthogonal to each other.

PROOF: Fault detectability of the system will be a maximum if individual checks in S_1 share a minimum number of processors with the checks in S_2. However, since it is necessary that every processor in A is checked by at least one check in S_1 and by at least one check in S_2, the minimum number of processors that they can share is one. Therefore, for the detectability to be a maximum, it is necessary that the checks are orthogonal to each other. □

THEOREM 6.1. If $det(A)|_{S_1} = t_1$, $det(A)|_{S_2} = t_2$, $loc(A)|_{S_1} = l_1$, and $loc(A)|_{S_2} = l_2$, where $t_1, t_2 \geq 1$, then

$$det(A)|_{S_1 \cup S_2} < (t_1 + 1)(t_2 + 1)$$

$$loc(A)|_{S_1 \cup S_2} < (l_1 + 1)(l_2 + 1).$$

PROOF: By the definition of detectability, the minimum size of a fault pattern which cannot be detected by S_1 is $(t_1 + 1)$. Similarly, the minimum size of the fault pattern which cannot be detected by S_2 is $(t_1 + 1)$. We want to maximize the size of the smallest fault pattern which cannot be detected by checks in both S_1 and S_2. From the previous lemma, the detectability is maximized when the checks are orthogonal. In this configuration one can observe that the maximum of the size of the smallest undetectable fault is equal to the product of the size of the smallest fault pattern undetectable by S_1 and the size of the smallest fault pattern undetectable by S_2. That is,

$$det(A)|_{S_1 \cup S_2} < (t_1 + 1)(t_2 + 1).$$

With a similar logic we arrive at the corresponding result for the locatability of the system. $\square$

In the following section, we show that this is a reachable bound. We describe the construction of a fault-tolerant system which achieves the maximum fault detectability and locatability.

6.3. The Hierarchical Approach

Our main objective is to show how checks can be treated at different levels of hierarchy during the analysis of a system. To that end we first describe a hierarchical approach for the design of fault-tolerant multiprocessor systems. One may seek various kinds of hierarchies to design a system. However, the particular hierarchy we suggest is motivated by two factors: (1) most of the existing multiprocessor systems are built using this type of hierarchy; for example, in the binary hypercube, an *n-dimensional* cube is constructed by connecting the corresponding processors in two $n-1$ *cubes*; (2) this will maximize the fault detectability and locatability of the overall system for a given fault detectability and locatability of the basic unit [54].

Before going into the details of the type of hierarchy we use in the design and analysis of systems, we will establish some properties related to the fault detectability/locatability of fault-tolerant systems and the *error detectability* of checks used in those systems. In this section, we assume that every processor produces only one data element and that the fault in a check-evaluating processor will not invalidate the checks performed by that processor. The presence of a fault is manifested as a single error. In other words, there is a one-to-one correspondence between faults and errors. Therefore, in ensuing discussions, we will use the terms *faults* and *errors* interchangeably.

DEFINITION 6.6. A fault-tolerant multiprocessor system is said to be *bounded* if and only if $t < N$ where t is the *fault detectability* of the system and N is the number of processors in the system. □

The concept of *boundedness* is relevant only if the checking operations do not become invalid even if the corresponding check processors fail. This may be achieved either by employing an external processor for the checking operation or by building the checking units inside the check processors to be *totally self–checking*. If this condition is not satisfied, trivially, the fault detectability of a system cannot exceed the number of check evaluating processors in the system and hence the system is always bounded.

EXAMPLE 6.2. Consider a mesh connected processor array in which processors are checked by column checks and row checks as shown in the Figure 6.2. Let the error detectability of the column/row checks be h=2. Using the algorithms given in the preceding section we find that the array in Figure 6.2 (a) has a fault detectability = 4, and the one in Figure 6.2 (b) has a fault detectability = 8. Therefore, system (a) is *not bounded* whereas system (b) is *bounded.* □

DEFINITION 6.7. A check is said to be *bounded* if it checks more than h data elements. □

LEMMA 6.4. A sufficient condition for a system to be *bounded* is that all the checks performed in the system are *bounded.*

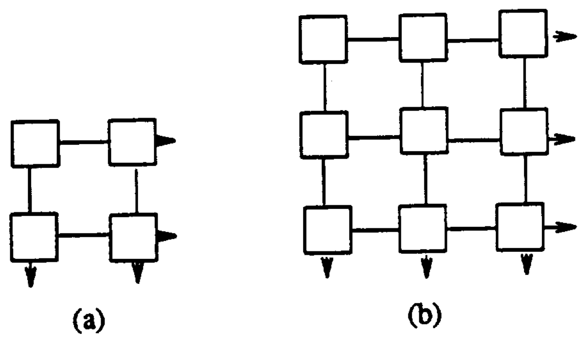


Figure 6.2. Examples for unbounded and bounded systems.

PROOF: Proof by contradiction: Suppose all the checks are *bounded* and the system is not *bounded*, which implies $N = t$. However, when all the processors are faulty, every check will be checking greater than h errors, and hence all of them will produce invalid results. Therefore, we cannot detect the simultaneous presence of N faults which is a contradiction to the hypothesis. □

It may be noticed that in Example 6.2, the system represented in Figure 6.2 (b) has all its checks *bounded* and hence the system is *bounded* as we had found out by other means.

LEMMA 6.5. If s_i, for $i = 1,2,...$, are subsystems of a given system S such that $s_i \subseteq S$, then S is bounded if and only if at least one subsystem s_j is bounded; then the fault detectability/locatability of S is less than or equal to the fault detectability/locatability of s_j.

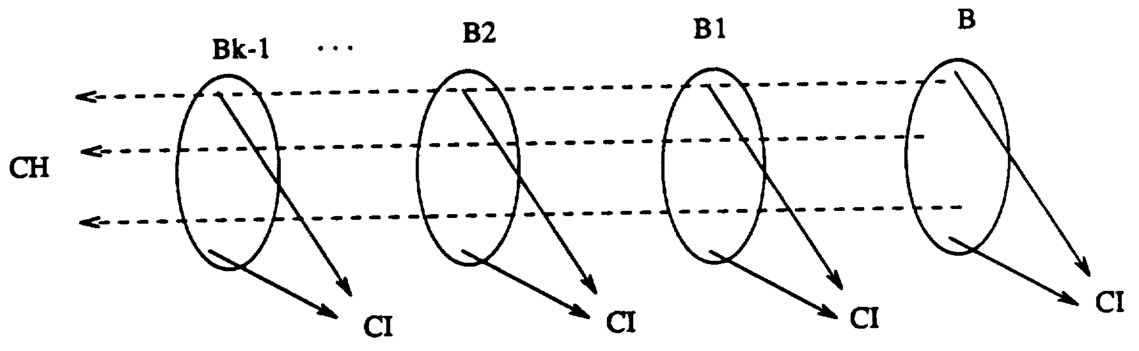
PROOF: The proof for the necessary condition is trivial, since by definition of s_i it could be the system S itself.

Proof of the sufficiency condition: Let s_j be a bounded subsystem. Then, irrespective of the rest of the system, the subsystem s_j will have a detectability, $t < |s_j|$ where $|s_j|$ denotes the number of processors in the subsystem. However, if the rest of the system has a detectability strictly less than t, then the overall detectability (T) of the system will be also strictly less than t. Therefore, $T \leq t < |S|$ and hence the proof. □

The motivation for Lemma 6.5 is to point out that if we add more processors to an already bounded system (note that there are no new checks added to the existing system during the expansion), the fault detectability of the overall system does not increase. We make use of this inference in the formulation of Theorem 6.2.

6.3.1. Construction of a hierarchical system

We now outline the procedure to build a hierarchical system from a basic unit. Let B be the given basic system with a known fault detectability and locatability. First we replicate copies of B (replication involves replication of the checks also). Let P_1, P_2, ..., P_r be the processors in B which are checked only by bounded checks. As a second step in the hierarchical expansion of the system, r new checks are introduced such that the first check performs the evaluation of processor P_1 in B and all its image processors in other copies of B, the second check evaluates processor P_2 and all its copies, and so on. The procedure is illustrated in Figure 6.3. We do not have to provide a check at the higher levels for those processors which are checked by at least one unbounded check because an unbounded check will always fail if the processor(s) checked by that check is (are) faulty regardless of the presence of any other fault.



CI - Internal check
CH - Check in the higher level

Figure 6.3. Hierarchical expansion of a basic system.

In the figure, B_1, B_2, ..., B_{k-1} are copies of the basic system B. CI represents the set of checks which are internal to every copy of B and CH represents the set of checks in the next higher level. In the following discussion, this kind of a construction will be referred to as *k-fold expansion of B in the next level of hierarchy*. Following a similar procedure, the expanded system may further be extended in the next higher level of the hierarchy.

It may be noted that the checks introduced at different levels of hierarchy may be of different types having different values of error detectability. In order to simplify the development of our theory, we assume that all the checks in the system are similar, and have the same error detectability. However, the value of g may be different. An important restriction we impose while expanding the system is that there is no data migration allowed between processors in different copies of the basic unit. In other words, faults in one copy of the basic unit will not affect other copies.

Now we derive the relationships between the fault detectability (locatability) of a basic unit and a system obtained by hierarchically expanding the basic unit.

THEOREM 6.2. If t and l are the fault detectability and locatability of a *bounded* basic system B, and S is a *k-fold ($k \le g$) d-level* hierarchical expansion of B, then the fault detectability (T_d) and locatability (L_d) of S are

$$T_d = \begin{cases} |B|.k^{d-1} & \text{for } 1 < k \le h \\ (t+1)(h+1)^{d-1} - 1 & \text{for } k > h \end{cases}$$

$$L_d = 2^{d-1}(l+1) - 1 \quad \text{for } k > 1$$

where $|B|$ represents the number of processors in the basic system.

PROOF: We prove the theorem by induction on the number of levels, d.

Proof for the detectability part:

Case 1. $1 < k \le h$

Basis: $d = 2$

Consider the lowest level and the next higher level of hierarchy (i.e., level 2). When $k \le h$, none of the checks in level 2 will become invalid for any combination of faults in B and the copies of B, since none of these checks are evaluating more than h data elements. On the other hand, at least one of these checks will fail for any combination of faults in the system. Therefore, the fault detectability is equal to the number of processors in the system which is equal to $|B| \,.\, k$.

Inductive Step:

Let the hypothesis be true for the number of levels up to $d-1$. Then

$$T_{d-1} = |B|\ k^{d-2}.$$

Now considering S_{d-1} as the basic unit and applying the basis case, we arrive at

$$T_d = |S_{d-1}|.k = (|B|.k^{d-2}).k = |B|.k^{d-1}.$$

Note that in this case the resulting system is *not bounded.*

Case 2. $k > h$.

Basis: d=2

Here, also, we first consider the basic unit B and its next level of hierarchy. We now prove that there exists at least one fault pattern of cardinality $(t+1)(h+1)$ which will not be detected in the 2–*level* system. Let there be identical fault patterns of cardinality $(t+1)$ occurring in $(h+1)$ copies of B. These faults will not be detected by checks inside the copies of B, since the fault detectability of B is equal to t. Every check at the second level is either checking processors which are not faulty or $(h+1)$ processors which are faulty. In either case, the checks may produce a "pass" output (since the error detectability of the checks is equal to h, faults of cardinality (h+1) may invalidate the checks). This means that the faults will not be detected in this level, also.

Now we prove that every fault pattern of cardinality less than or equal to $(t+1)(h+1) - 1$ will be detected. In order for such a fault pattern not to be detectable in the lower level, it is necessary that the copies of B having faulty processors should have more than t faults present in them. The fault is not detectable in the higher level only if the checks evaluating the faulty processors check more than h errors. If we distribute $(t+1)(h+1) - 1$ faults into $(h+1)$ copies such that every basic unit has at least $(t+1)$ faults, by the *pigeon hole* principle [55], at least one of the subunits will have $\leq t$ faults, and hence the fault is detectable in that copy. Conversely, if every subunit has $\geq (t+1)$ faults, at least one check in the next level will be checking $\leq h$ faults, and will detect those faults.

Therefore, the fault detectability of a 2–*level* expansion of B is $(t+1)(h+1) - 1$.

Inductive Step:

Let the hypothesis be true for the number of levels up to $d-1$. Then

$$T_{d-1} = (t+1)(h+1)^{d-2} - 1.$$

Now applying the basis case, we have

$$T_d = (T_{d-1} + 1)(h+1) - 1 = (t+1)(h+1)^{d-1} - 1.$$

Proof for the locatability part:

Basis: d=2

Here, also, we first consider the lowest level of hierarchy and the next level. First, we prove that there exists a fault pattern of cardinality $(2l+2)$ which will not be correctly located in a 2–*level* system. Consider two identical fault patterns of size $(l+1)$ occurring on two copies of B. Since the locatability of B is l, these fault patterns will not be located correctly within the copies of B (that is, the internal checks cannot locate the faults correctly). Even if all the checks in the next level detect faults, it may not be possible to locate the faults among the copies of B.

Now we prove that any fault pattern of cardinality $\leq 2l+1$ will always be correctly located in a 2–*level* system. Consider a fault pattern of size $(2l+1)$. If we distribute the individual faults in this pattern among various copies of B, by the pigeon hole principle, at most one copy of B will have $\geq (l+1)$ faults. Let us denote such a copy as B_i. Now the total number of individual faults distributed among all the other copies will be $\leq l$.

Therefore, any of those copies will have a fault pattern of cardinality $\le l$ and will be correctly located by checks within the copy. Now let us consider locating the fault within B_i. The maximum size of a fault which may occur in B_i is $2l + 1$ in which case none of the other copies will have any fault in them. Since $t \ge (2l + 1)$ [18], the fault in B_i will be detected by checks within B_i. The checks in the next level are checking only one fault each, and therefore, they can locate the faults within B_i. Hence the fault is locatable.

Now we consider a more general case in which the number of faults in B_i is $(l + r)$ where $r \ge 1$. The rest of the copies of B will have a total of $(l - r + 1)$ faults. Regardless of the way these faults are distributed among these copies, there will be at least $(2r - 1)$ number of faults in B_i such that there are no faults in any other copy of B in the corresponding positions (we refer to these faults as *unobscured* faults). These $(2r - 1)$ faults in B_i can be located by checks in the higher level. The remaining $(l - r + 1)$ faults can be uniquely located with the help of the syndrome generated by the internal checks of B_i since $(l - r + 1) \le l$. Therefore, any fault pattern of cardinality $\le (2l + 1)$ can be uniquely located in a 2-*level* system.

Inductive Step:

Let the hypothesis be true for the number of levels up to $d-1$. Then

$$L_{d-1} = 2^{d-2}(l+1) - 1.$$

Now applying the basis case, we arrive at

$$L_d = 2L_{d-1}+1 = 2^{d-1}(l+1) - 1.$$

It may be noted that the fault detectability of a system increases as the number of copies (k) in the same level increases, until k reaches a value equal to h. For $k > h$, the system is *bounded* in that level and the fault detectability attains a constant value as described in the beginning of this section. Locatability, however, is independent of the value of k. This is because of the generality of the definition of checks where we assume that an individual check can locate no faults, even though it can detect multiple faults.

In the following, we present two examples to illustrate the hierarchical construction of fault-tolerant systems.

EXAMPLE 6.3. As a first example, we consider a linear processor array as shown in Figure 6.4 (a). In the array, all the processors are evaluated by a check with error detectability, $h = 1$. Fault detectability of such a linear array is equal to 1 (i.e., t=1) and fault locatability is equal to 0 (l=0). Now we expand the system hierarchically to form a two dimensional mesh connected processor array as shown in Figure 6.4 (b). The newly added checks in the new level are shown by dotted lines.

By the previous theorem, the fault detectability of the mesh connected processor array is

$$T_2 = (t+1)(h+1)^{d-1} - 1 = 2 \times 2 - 1 = 3.$$

The fault locatability L is

$$L_2 = 2^{d-1}(l+1) - 1 = 2 \times (0+1) - 1 = 1.$$

These values conform to the values obtained from the analysis using the nonhierarchical algorithms presented in Section 6.2.

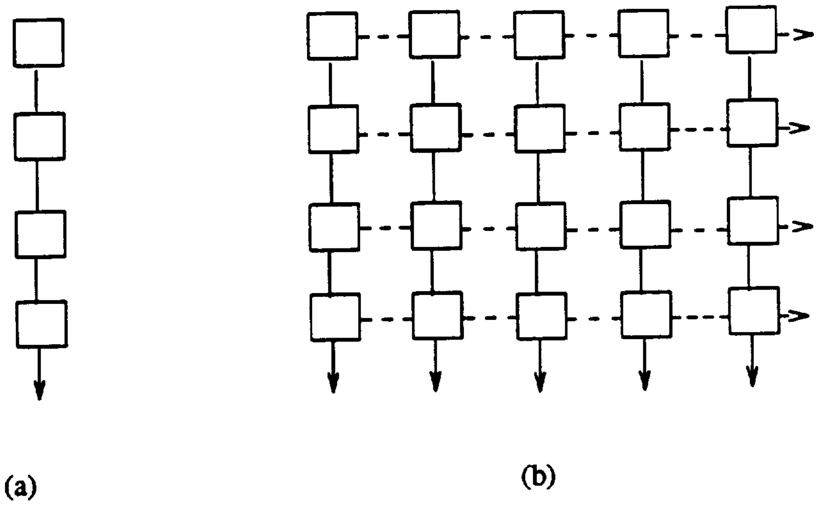


Figure 6.4. Hierarchical expansion of a linear array.

EXAMPLE 6.4. As a second example, we consider the hierarchical expansion of the Advanced Onboard Signal Processor (AOSP) architecture [48]. From the analysis of this system we find that the system is 3–*fault detectable* and *single fault locatable.*

Now let us consider a 4–*fold* 2–*level* expansion of AOSP (i.e., using AOSP as the basic system). The expansion scheme is illustrated in Figure 6.5. Every copy of AOSP is associated with six internal checks. Nine additional checks are added in the next level as shown in the figure by dotted lines. By Theorem 6.2, the fault detectability and locatability of this system are 7 and 3, respectively. □

6.3.2. The number of checks in the hierarchical system

In this section we compute the number of checks required in the hierarchical construction of a system in terms of the number of checks in the basic unit and the number of levels in the system. Here we assume that all the checks in the system (including the

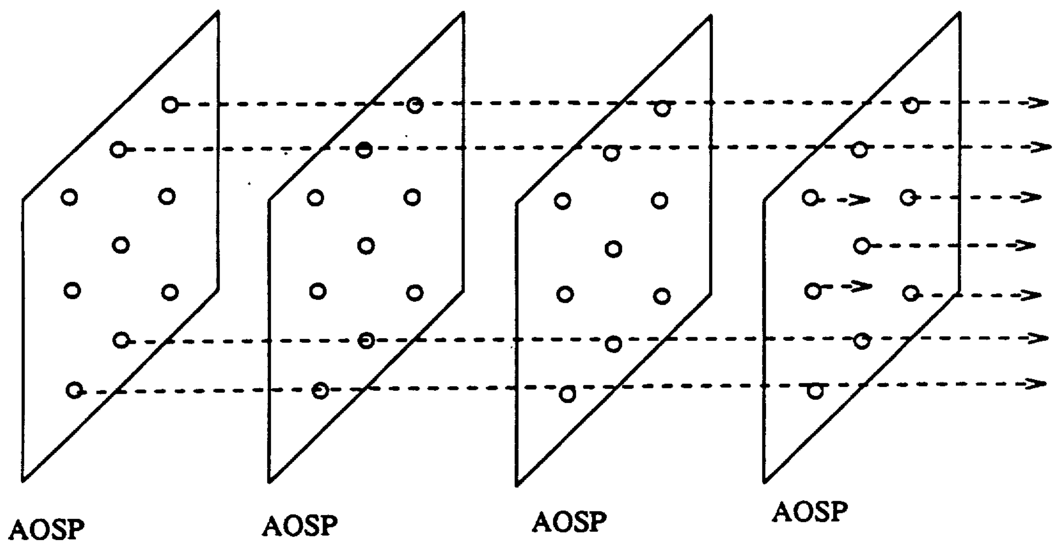


Figure 6.5. Hierarchical expansion of AOSP architecture.

checks in CI) are bounded.

THEOREM 6.3. The number of checks used in a hierarchical system built by a *k–fold*, *d–level* expansion of a basic unit is

$$H_d = r\,k^{d-1} + n\,(d-1)\,k^{d-2}.$$

where r is the number of checks in the basic unit and n is the number of processors in the basic unit.

PROOF: By construction, the number of checks in a hierarchical system satisfies the recursive equation

$$H_d = k\,H_{d-1} + N_{d-1}$$

where N_{d-1} is the number of checks in the $(d-1)$*–level* system. Since $N_d = n\,k^{d-1}$,

$$H_d = k\,H_{d-1} + n\,k^{d-2}.$$

Solving the recursive equation with boundary conditions, $H_1 = r$ and $H_2 = rk + n$ yields that

$$H_d \;=\; r\,k^{d-1} + n\,(d-1)\,k^{d-2}. \qquad \square$$

However, we observe that it is not necessary to have all the H_d checks. The observation is elaborated in the following lemmas.

LEMMA 6.6. There exists a 2–*level* system with detectability T_2 which requires only $H_2 - T_1$ checks.

PROOF: We shall prove that even if we remove any T_1 (note that $T_1 = t$) checks from the set of checks in the second level, the detectability of a 2–*level* system remains the same as T_2. Any detectable fault in the system should be detected either at the lower level (by *CI*s) or in the higher level. If the fault is detected in the lower level, removal of checks from the higher level is not going to affect the detectability of the fault. Therefore, we need to consider faults which are detectable only in the higher level. If such a fault occurs, some copies of the basic unit will have $\geq (t+1)$ number of faults whereas the rest of the copies will not have any faults at all. However, from Theorem 6.2 we know that there are at most h copies of the basic unit having $\geq (t+1)$ number of faults. In Figure 6.6, the large ellipses represent copies of the basic unit which have $\geq (t+1)$ faults. The shaded portions represent T_1 processors in every basic unit which are not checked in the next higher level. The corresponding checks which are removed from the system are denoted as set U. Since the size of the fault patterns present in the copies is $\geq (t+1)$, at least one faulty processor in that copy will be checked by a check C_h in the set $(CH - U)$. Since the number of such faults checked by every check in $(CH - U)$ is at most h, the fault will be detected. Thus, the detectability is unaffected despite the removal of T_1 checks from the original hierarchical system. □

A similar result exists related to the locatability of a system. However, a distinction has to be made between the problems of detectability and locatability here: in the case of detectability, we need help from the higher-level checks only when the fault is undetectable in all the copies of the basic unit whereas, in the case of locatability we need to use the higher-level checks whenever at least one copy of the basic unit has a fault pattern that is unlocatable with the help of the internal checks. Intuitively, we cannot remove as many checks in the case of locatability as in the case of detectability and still preserve the overall locatability of the system.

LEMMA 6.7. There exists a *2–level* system with locatability L_2 which requires only $H_2 - 1$ checks.

PROOF: From Theorem 6.2, we know that there is at most one copy of the basic unit which has $\geq (l + 1)$ number of faults, and we know that the purpose of the higher level checks is to locate correctly the ***unobscured*** faults. Therefore, we must ensure that the ***unobscured*** faults should not lie entirely inside the shaded region (that is, the set of

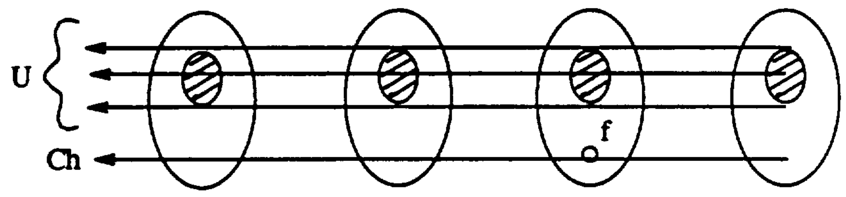


Figure 6.6. Unnecessary checks in the second level of hierarchy.

processors which had been checked by the checks in U). Since the minimum size of the set of *unobscured* faults is *one*, the maximum number of checks we can remove from CH, without altering the locatability of the system, is *one*. □

Now we generalize these results for a *d−level* system. Even though the saving in terms of the number of checks is small for a 2*−level* system, it will be shown that the overall saving may be significant for larger values of d.

THEOREM 6.4. There exists a *k−fold, d−level* system with detectability T_d using H_d - ${}^{\det}U_d$ number of checks, where

$${}^{\det}U_d = \frac{1}{(k-1)} [((t+1) \sum_{i=1}^{i=d-1} (k^{d-i} - 1)(h+1)^{i-1}) - \sum_{i=1}^{i=d-1} (k^{d-1} - 1)].$$

PROOF: From Lemma 6.6, we know that in a 2*−level* system, T_1 number of checks are unnecessary. In the hierarchical expansion, the second level systems will be considered as the new basic units and are replicated in the third level. Here, the overall saving will be $T_1 k + T_2$. If we recursively calculate the number of checks saved, we arrive at

$${}^{\det}U_d = \sum_{i=1}^{i=d-1} T_i (\sum_{j=0}^{j=d-i-1} k^j).$$

Now substituting the value for T_i as $(t+1)(h+1)^{i-1}$, we have

$${}^{\det}U_d = \frac{1}{(k-1)} [((t+1) \sum_{i=1}^{i=d-1} (k^{d-i} - 1)(h+1)^{i-1}) - \sum_{i=1}^{i=d-1} (k^{d-1} - 1)].$$ □

THEOREM 6.5. There exists a *k−fold, d−level* system with detectability T_d and locatability L_d using H_d - ${}^{\text{loc}}U_d$ number of checks, where

$${}^{\text{loc}}U_d = \sum_{i=1}^{i=d-1} (\sum_{j=0}^{j=i-2} k^j).$$

Proof: The proof is very similar to the previous theorem except that in every level we save only one check during expansion. In the second level we save one check, in the third level $(k + 1)$, and so on. In general the number of checks saved in level i is equal to $\sum_{j=0}^{j=i-2} k^j$. Summing all those values up to level d,

$$\mathrm{loc}_{U_d} = \sum_{i=1}^{i=d-1} \left(\sum_{j=0}^{j=i-2} k^j \right). \qquad \square$$

6.3.3. Hierarchical analysis of systems

The hierarchical principles derived in the preceding section can be translated into the domain of the fundamental matrices which constitute the matrix model. Replication of the basic unit is equivalent to a repetition of the PC matrix of the basic unit along the diagonal. The addition of checks in the new dimension is tantamount to adding identity matrices (one identity matrix per diagonal submatrix) to the expanded PC matrix. The matrix equivalent of the hierarchical expansion of a system is shown in Figure 6.7. Here PC represents the PC matrix of the basic unit and I is an identity matrix.

In order to analyze a given system hierarchically, we first arrange the rows and columns of the PC matrix in such a way that the final matrix is in the form shown in Figure 6.7. Now, the detectability and locatability of the basic PC matrix can be computed, from which the detectability and locatability of the entire system can be derived using the results in Theorem 6.2. Note that, typically, the size of the basic unit is considerably smaller than the size of the entire system.

However, in certain designs, it may be the case that the diagonal PC matrices will have the same number of rows, but a different number of columns, that is, during

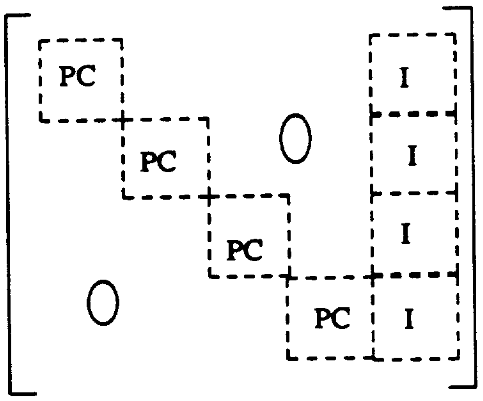


Figure 6.7. The PC matrix of a hierarchical system.

replication of the basic unit, all the internal checks (CI) were not replicated. In this case the fault detectability and the locatability of the images of the basic unit may be different. In such a case we cannot compute the actual values of fault detectability and locatability of the hierarchical system. However, we can calculate a lower bound on these figures.

COROLLARY 6.1. (Of Theorem 6.2.) If the diagonal PC matrices have different detectabilities and locatabilities, then the detectability and locatability of the hierarchical system are bounded by

$$T_d \geq (t_{\min} + 1)(h+1)^{d-1} - 1;$$

$$L_d \geq 2^{d-1}(l_{\min} + 1) - 1;$$

where $t_{\min}$ is the minimum value of detectability and $l_{\min}$ is the minimum value of locatability among the copies of basic units.

6.4. Conclusions

We developed the concept of independent and orthogonal checks depending upon the set of processors checked by the given sets of checks. Using *orthogonal checks*, hierarchical techniques for the design and analysis of large fault-tolerant multiprocessor systems were developed. We introduced the method to model different levels of checks, which greatly simplifies the analysis and design of systems. The relationships between the fault-diagnosing capabilities of basic systems and their hierarchical expansions were derived.

CHAPTER 7.

CONCLUSIONS

7.1. Summary of Results

In many critical applications of VLSI-based computer systems, it is important to have high performance as well as high reliability. High reliability has been achieved by the application of fault tolerance techniques. Since the fault tolerance techniques are dependent on the redundancy involved in the computations, such systems are either costly due to the hardware redundancy involved or they are unable to reach high performance levels due to the time redundancy. Therefore, the problem in hand is to investigate techniques by which a high degree of fault tolerance can be achieved without sacrificing too much performance. Algorithm-based fault tolerance (ABFT) has been proposed as a cost effective scheme to achieve fault tolerance in multiprocessor systems. These schemes use functional as well as system-level concurrent error detection for the fault diagnosis in a system.

This thesis has addressed the problem of modeling fault-tolerant systems using concurrent error detection schemes in general and those using ABFT schemes in particular. The major results in the thesis are recapitulated in the following.

In Chapter 2, we have given a general description of multiprocessor systems which have been selected for the application of ABFT systems. In order to exemplify the technique, we illustrated how fault-tolerant matrix multiplication can be performed on a mesh-connected processor array using checksum encoding techniques. Since most of the signal processing computations can be represented as matrix operations, it is desirable to have generalized encoding schemes for fault-tolerant matrix operations. In this chapter, we developed a general set of real-number codes for these computations. We proved that for every linear finite-field code, there exists a real-number code having similar error diagnosing capabilities as the finite-field code. Since most of the codes known until now fall in the set of finite-field codes, our new result has a far-reaching effect in the area of coding theory as it forms a bridge between finite-field codes and real-number codes.

A matrix-based model for ABFT systems is presented in Chapter 3. The model consists of three matrices: the PD (processor-data), the DC (data-check), and the PC (processor-check) matrix. The model used a broad interpretation of faults, errors, and checks. The problem of invalidation of a check, performed by a faulty processor, is efficiently handled by translating it into the problem of error detection at the output of the faulty processor. Based on the model, various necessary and sufficient conditions for the fault detectability and locatability of systems are derived. Using these constraints and sufficient conditions, algorithms were developed for the analysis of ABFT systems. These algorithms are much less complex than the previously available algorithms. A detailed discussion of the algorithms is given in Chapter 4.

Chapter 5 dealt with design of ABFT systems. We developed a systematic and straightforward methodology for the design of ABFT systems. The design requires a smaller number of checks when compared to the previous bounds, especially when the individual processors in the system are computing large volumes of data. Other advantages include the flexibility of the algorithm to accommodate varying amounts of computation performed by the computing nodes and the ability to handle detectability and locatability of the system simultaneously. The application of the matrix model helped in identifying the redundant checks using simple matrix operations.

In Chapter 6, we introduced a hierarchical approach for the analysis of fault-tolerant multiprocessor systems. Even though inclusion of checks at different levels of hierarchy has been practised in the past, the analysis of such systems was carried out on the basis of a nonhierarchical ("flat") description of the system. We proposed a hierarchical approach for the analysis of these systems. We treat the checks at different levels of hierarchy. The fault tolerance of the system at different levels is estimated separately and the overall fault tolerance is derived from those values. In order to illustrate the concept, we introduced a special type of hierarchy for the design of multiprocessor systems. This particular type was chosen since it is easily applicable to most of the commercially available multiprocessors. In addition, we observed that this particular type of hierarchy maximizes the fault detectability and locatability of the overall system for a given error-detecting capability of the individual checks.

7.2. Suggestions for Future Research

Even though ABFT techniques have been applied to most of the signal processing computations, the applicability of the technique in other computations and data manipulations has to be further investigated. As mentioned in Chapter 2, the ABFT techniques are application specific. However, it may be possible to identify groups of computations which can use similar encoding schemes to make the computation fault-tolerant. For instance, we have shown that there exists a general set of real-number codes applicable to various matrix operations such as multiplication, addition, transposition, and LU-decomposition. Similar generalization of codes for various other computations is desirable.

In the graph model as well as in the matrix model, it is assumed that all the data values checked by a *check processor* are available simultaneously at the input of the *check processor*. In fact, this is a general assumption made by researchers in coding theory. However, in system level diagnosis the availability of a particular data element at the input of a checking processor is dependent on: (1) the computing speed of the particular node which computes that data element; (2) the speed and band width of the communication channel between the computing node and the evaluation node; (3) the data traffic in the system. Therefore, it is desirable to include some timing features into the check evaluation process. In [56] the researchers use Petri Nets to study the timing behavior of fault-tolerant systems. The limitation of this approach was that even for systems having a small number of processors, it takes a large amount of time to verify the fault tolerance capabilities of the system. It will be interesting to study the possible extension of the matrix-based model to include time-dependent checks. We believe that with such a formulation, a faster evaluation of fault-tolerant systems will be possible.

Another suggestion is to extend the field of application of the matrix-based model. The advantage of the proposed model is that it is independent of the particular computational algorithm associated with the ABFT system. In order to model a system we need to know only the relationship between the various entities in the system. It is not difficult to model a fault-tolerant software system using this model. The difference between hardware and software modeling is that instead of each processor module in the hardware case, we will have a program module in the software system. In fact, the use of the model in the analysis and design of fault-tolerant software systems will be even more effective since the interaction between various nodes in the system is not limited by the physical interconnection between them.

The hierarchical approach developed in Chapter 6 deals with only one type of hierarchy. Even though this covers many of the commercially available fault-tolerant array processors, a generalization of the concept of hierarchy is desirable. In the proposed hierarchy, the checks at different levels are assumed to be *orthogonal* to each other. A general case may be derived by assuming less stringent relationships between the checks.

As mentioned in Chapter 5, there have been two approaches followed by ABFT system designers: (1) given a non-fault-tolerant system, determine an efficient distribution of checks among the output data elements so that the system has the desired amount of fault tolerance; (2) given a fault-tolerant algorithm, synthesize an architecture so as to maximize quantities such as the fault detectability and locatability of the system. In the first approach, the fault-tolerant design is constrained by the fixed, non-fault-tolerant architecture. Often, this may result in an inefficient design (as far as fault tolerance is concerned); however, it preserves the high performance of the original architecture. In the second approach, performance may be sacrificed in the process of achieving high fault tolerance. Therefore, a more efficient approach would be to synthesize fault-tolerant architectures directly from the original algorithms so that the architecture is optimal with respect to performance, diagnosability, and reconfigurability.

REFERENCES

[1] D. A. Rennels, "Fault-tolerant computing - concepts and examples," *IEEE Trans. Comput.*, vol. C-33, pp. 1116-1129, Dec. 1984.

[2] Ravishankar K. Iyer, Steven E. Butner, and Edward J. McCluskey, "A statistical failure/load relationship: Results of a multicomputer study," *IEEE Trans. Comput.*, vol. C-31, pp. 697-705, July 1982.

[3] T. Anderson and P. A. Lee, *Fault Tolerance - Principles and Practice* . New Jersey: Prentice Hall Inc., 1981.

[4] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies*. Princeton, NJ: Princeton University Press, pp. 43-99.

[5] T. B. Lewis, "Primary processor and data storage equipment for orbiting astronomical observatory," *IEEE Trans. Elect. Comput.*, vol. EC-12, pp. 677-686, Dec. 1963.

[6] J. G. Tryon, "Quadded logic," in *Redundancy Techniques for Computing Systems*. Washington, DC: Spartan Books, 1962.

[7] I. Koren, "A reconfigurable and fault-tolerant VLSI multiprocessor array," in *Proc. 8th Int. Symp. on Computer Architecture*, Minneapolis, Minnesota, pp. 425-442, May 1981.

[8] S. Y. Kuo and W. K. Fuchs, "Efficient spare allocation for reconfigurable arrays," *IEEE Design and Test*, pp. 24-31, Feb. 1987.

[9] M. Lowrie and W. K. Fuchs, "Reconfigurable tree architectures using sub-tree oriented fault tolerance," *IEEE Trans. Comput.*, vol. C-36, pp. 1172-1182, Oct. 1987.

[10] M. Sami and R. Stefanalli, "Reconfigurable architectures for VLSI processing arrays," *Proc. IEEE*, vol. 74, pp. 712-722, May 1986.

[11] P. Velardi and R. K. Iyer, "A study of software failures and recovery in the MVS operating system," *IEEE Trans. Comput.*, vol. C-33, June 1984.

[12] J. Wakerly, *Error-Detecting Codes, Self-Checking Circuits and Applications*. New York: Elsevier North Holland Inc., 1978.

[13] M. A. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital Systems*. Maryland: Comput. Sci. Press, 1976.

[14] K. H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. C-33, pp. 518-528, June 1984.

[15] P. Banerjee, J. T. Rahmeh, C. B. Stunkel, V. S. S. Nair, K. Roy, and J. A. Abraham, "Algorithm-based fault tolerance on a hypercube multiprocessor," *IEEE Trans. Comput.*, (to appear).

[16] J. Y. Jou and J. A. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures," *Proc. IEEE*, vol. 74, no.5, pp. 732-741, May 1986.

[17] F. P. Preparata, G. Metze, and R. T. Chien, "On the connection assignment problem of diagnosable systems," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 848-854, December 1967.

[18] J. D. Russel and C. R. Kime, "System fault diagnosis: Closure and diagnosability with repair," *IEEE Trans. Comput.*, vol. C-24, pp. 1078-1088, 1973.

[19] J. D. Russel and C. R. Kime, "System fault diagnosis: Masking, exposure, and diagnosability without repair," *IEEE Trans. Comput.*, vol. C-24, pp. 1155-1161, 1975.

[20] M. Adham and A. D. Friedman, "Digital system fault diagnosis," *J. Design Aut. and FaultTol. Comput.*, vol. 1, no.2, pp. 115-132, Feb. 1977.

[21] S. N. Maheswari and S. L. Hakimi, "On models for diagnosable systems and probabilistic fault diagnosis," *IEEE Trans. Comput.*, vol. C-25, pp. 228-236, Mar. 1976.

[22] H. Fujiwara and K. Kinoshita, "Some existence theorems for probabilistically diagnosable systems," *IEEE Trans. Comput.*, vol. C-27, no. 4, pp. 379-384, Apr. 1978.

[23] P. Banerjee and J. A. Abraham, "Concurrent fault diagnosis in multiple processor systems," in *Proc. 16th Int. Symp. Fault-Tolerant Comput.*, Vienna, Austria, pp. 298-303, 1986.

[24] R. Bisiani, A. Nowatzyk, and M. Ravishankar, "Coherent shared memory on a shared memory machine," in *Proc. Int. Conf. Parallel Processing*, vol. I, Chicago, Illinois, pp. 133-141, 1989.

[25] K. L. Wu and W. K. Fuchs, "Recoverable distributed shared virtual memory," *IEEE Trans. Comput.*, vol. 39, pp. 460-469, Apr. 1990.

[26] J. -C. Laprie, "Dependable computing and fault tolerance: Concepts and terminology," *Proc. Int. Symp. Fault-Tolerant Comput.*, pp. 2-11, June 1985.

[27] T. E. Mangir, "Sources of failures and yield improvement for VLSI and restructurable interconnects for RVLSI and WSI: Part II," *Proc. IEEE*, vol. 72, pp. 1687-1694, Dec. 1984.

[28] J. A. Abraham and W. K. Fuchs, "Fault and error models for VLSI," *Proc. IEEE*, vol. 74, no. 5, pp. 639-654, May 1986.

[29] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, vol. 27, pp. 228-234, Apr. 1980.

[30] K. A. Hua, "Design of systems with concurrent error detection using software redundancy," Ph.D. dissertation, Univ. of Illinois, Urbana, Illinois, 1987.

[31] P. Banerjee, "A Theory for algorithm-based fault tolerance in array processor systems," Ph.D. dissertation, Univ. of Illinois, Urbana, Illinois, 1985.

[32] V. S. S. Nair and J. A. Abraham, "Real number codes for fault-tolerant matrix operations on processor arrays," *IEEE Trans. Comput.*, pp. 426-435, Apr. 1990.

[33] F. T. Luk and H. Park, "Fault-tolerant matrix triangulation on systolic arrays," *IEEE Trans. Comput.*, vol. 37, pp. 1434-1438, 1988.

[34] J. Y. Jou and J. A. Abraham, "Fault-tolerant FFT networks," *IEEE Trans. Comput.*, vol. 37, pp. 548-561, May 1988.

[35] C. Y. Chen and J. A. Abraham, "Fault-tolerant systems for the computation of eigenvalues and singular values," *Proc. SPIE, Advanced Algorithms and Architectures for Signal Processing*, vol. 696, pp. 228-237, Aug. 1986.

[36] A. L. N. Reddy and P. Banerjee, "Algorithm-based fault detection for signal processing applications," *IEEE Trans. Comput.*, 1990, (to appear).

[37] C. Aykanat and F. Ozguner, "A conjugate gradient algorithm on a hypercube multiprocessor," *Proc. 17th Int. Symp. Fault-Tolerant Comput.*, pp. 204-209, 1987.

[38] V. S. S. Nair, "General linear codes for fault-tolerant matrix operations on processor arrays," M. S. thesis, Univ. of Illinois, Urbana, Illinois, Aug. 1988.

[39] V. S. S. Nair and J. A. Abraham, "General linear codes for fault-tolerant matrix operations on processor arrays," in *Proc. 18th Int. Symp. Fault-Tolerant Comput.*, Tokyo, Japan, pp. 180-185, June 1988.

[40] R. E. Blahut, *Theory and Practice of Error Control Codes*. Massachusetts: Addison Wesley, 1984.

[41] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*. Cambridge: MIT Press, 1981.

[42] B. Bose and T. R. N. Rao, "Theory of unidirectional error correcting/detecting codes," *IEEE Trans. Comput.*, vol. C-31, pp. 521-530, June 1982.

[43] C. W. Curtis, *Linear Algebra*. New York: Springer-Verlag, 1984.

[44] T. G. Marshall Jr., "Coding of real number sequences for error correction: A digital signal processing problem," *IEEE Journal on Selected Areas in Communication*, vol. SAC-2, no. 2, pp. 381-392, Mar. 1984.

[45] A. Costes, C. Landrault, and J. C. Lapnie, "Availability model for maintained systems featuring hardware failures and design faults," *IEEE Trans. Comput.*, vol. C-27, pp. 548-560, June 1978.

[46] V. S. S. Nair and J. A. Abraham, "A model for the analysis of fault-tolerant signal processing architectures," in *Proc. 32nd Int. Tech. Symp. SPIE*, San Diego, pp. 246-257, Aug. 1988.

[47] K. H. Huang and J. A. Abraham, "Low cost schemes for fault tolerance in matrix operations with processor arrays," in *Proc. 12th Int. Symp. Fault-Tolerant Comput.*, Santa Monica, California, June 21-24, 1982.

[48] J. R. Samson, Jr., and F. A. Horrigan, "The advanced onboard signal processor (AOSP) - A valid concept," *Proc. DARPA Strategic Space Symposium*, pp. 1-24, 1983.

[49] B. Vinnakota and N. K. Jha, "Diagnosability and diagnosis of algorithm-based fault-tolerant systems," in *Proc. 32nd Midwest Symp. Circuits and Systems*, Urbana, Illinois, Aug. 1989.

[50] P. Banerjee and J. A. Abraham, "Bounds on algorithm-based fault tolerance in multiple processor systems," *IEEE Trans. Comput.*, vol. C-35, pp. 296-306, Apr. 1986.

[51] D. J. Rosenkrantz and S. S. Ravi, "Improved bounds on algorithm-based fault tolerance," in *Proc. 26th Annual Allerton Conf. on Communication, Control, and Computing*, Monticello, Illinois, pp. 388-397, Sept. 1988.

[52] B. Vinnakota and N. K. Jha, "A dependence graph-based approach to the design of algorithm-based fault tolerant systems," in *Proc. 20 th Int. Symp. Fault-Tolerant Comput.*, Newcastle, England, 26 - 28th June, (to appear).

[53] S. Y. Kung, *VLSI Array Processors*. Englewoods Cliffs, NJ: Prentice Hall, 1988.

[54] V. S. S. Nair and J. A. Abraham, "Hierarchical analysis and design of fault-tolerant multiprocessor systems," *IEEE Trans. Comput.*, (under preparation).

[55] E. M. Reingold, J. Nievergelt, and N. Deo, in *Combinatorial Algorithms: Theory and Practice*. Englewoods-Cliffs, NJ: Prentice-Hall, 1977.

[56] J. Kljaich, Jr., B. T. Smith, and A. S. Wojcik, "Formal verification of fault tolerance using theorem-proving techniques," *IEEE Trans. Comput.*, vol. 38, pp. 366-376, Mar. 1989.

VITA

V. S. Sukumaran Nair was born in [redacted], on [redacted]. He received his B.Sc. Engg. degree in Electronics and Communication Engineering from the University of Kerala, India, in 1984. From 1984 to 1985, he was employed with the Indian Space Research Organization (ISRO) in Trivandrum. In 1986, he enrolled at the University of Illinois at Urbana-Champaign for his graduate studies. He received the M.S. degree in Electrical Engineering in 1988. While pursuing his M.S. and Ph.D. studies at the University of Illinois, he held a research assistantship in the Center for Reliable and High-Performance Computing at the Coordinated Science Laboratory from 1986 to 1990. His research interests include fault-tolerant computing, computer architecture, parallel processing, and VLSI. He is a student member of IEEE.