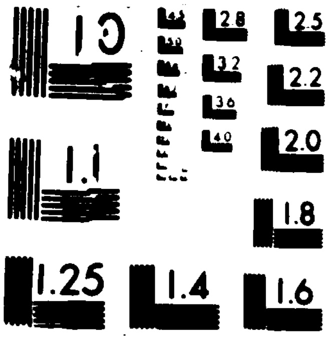
AD-A183 820   FAULT-TOLERANT MULTIPROCESSOR AND VLSI-BASED SYSTEMS   1/1
(U) MASSACHUSETTS UNIV AMHERST DEPT OF ELECTRICAL AND
COMPUTER ENGINEERING D K PRADHAN 15 MAR 87

UNCLASSIFIED   AFOSR-TR-87-0901 AFOSR-84-0052   F/G 12/7   NL

END
9-87
DTIC


1.0
4.5
5.0
5.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR	8b. OFFICE SYMBOL *(If applicable)* NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR 84-0052

8c. ADDRESS *(City, State and ZIP Code)*

Bldg 410
Bolling AFB DC 20332-6448

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	A3	

11. TITLE *(Include Security Classification)*

Fault-Tolerant ~~&~~ Multiprocessor & VLSI- Based Systems

12. PERSONAL AUTHOR(S)

D.K. Pradhan

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Yr., Mo., Day)*	15. PAGE COUNT
~~Research~~ Final	FROM 1984 TO 1987	March 15, 1987	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB. GR.	Fault-Tolerant Computing

AFOSR-TR- 87-0901

Fault-Tolerant Multiprocessor and VLSI-Based Systems

Final Report for the grant AFOSR 84-0052

Period covered: January 1984 - January 1987

Dhiraj K. Pradhan
Department of Electrical & Computer Engineering
University of Massachusetts
Amherst, MA 01003

Abstract

This report is a delineation of research results derived under the sponsorship of the AFOSR grant during January 1984 through January 1987. For the sake of conciseness, results already published in open literature are only abstracted. Other results that are yet to appear are described in more detail.

Table of Contents

1 Introduction

This final report for AFOSR 84–0052 provides a summary of various research activities carried out during three years of sponsorship.

The research proposed sought to focus on the following:

1.1

Study and development of certain fault-tolerant architectures that utilize the capabilities of the new IC technology was undertaken. Specifically, the research was aimed at network architectures, distinguished by a close interconnection of a large number of computing elements. Included is a subclass of specialized network architectures known as VLSI processor arrays. Besides *fault–tolerance–related* research for such arrays, also proposed was exploration of a new array architecture, developed for the express purpose of executing general algorithms on these arrays.

The precise research formulated – developing fault–tolerant multiprocessor network architectures – goes beyond earlier work. Here, the system interconnection structure, itself, was used as the primary design tool for achieving various and diverse objectives, including: low interconnection and layout complexities, dynamic reconfigurability, fault–tolerance through graceful degradation as well as self–diagnosability. Viability of the proposed research was demonstated in the proposal; new communication structures were introduced, along with concepts of admissability of multiple logical configurations, and algorithmic and detour routing that provide fault–tolerance and graceful degradation.

1.2

It was then proposed that the research be extended to the study of the design of certain VLSI processor arrays, particularly because this subject matter was so new, and only very limited work had yet been reported. Fundamental concepts were proposed identifying important relationships between various levels of redundancy and fault–tolerance. Also achieving testability and diagnosability internally, within the arrays, was sought.

1.3

A marked limitation of earlier VLSI arrays had been suitability only for very specialized and highly concurrent matrix–oriented computation. An entirely new way of utilizing these arrays was what was proposed here, which allow for execution of general algorithms, thereby making these arrays attractive for broader use.

This report is organized into three main sections. Section 2 highlights various results obtained on these above topics. Section 3 provides a complete list of publications and student support resulting from this research. Section 4 depicts several unresolved future research issues.

2 Review of Research Results

Below is delineated a summary of various research results derived under the grant. Those already published in open literature are only abstracted here; those yet to appear are described in some detail.

Fault-Tolerant Multiprocessor Networks

In [Prad85a], a class of link– and bus–oriented regular networks was presented. Significantly, these were shown to provide optimal/near optimal fault–tolerance. Various fault–tolerant properties of these networks were analyzed extensively, as well, compared against existing networks. What is novel is the capacity of these networks to be used to design any arbitrarily large networks, by using building blocks of any given number of connections per node. (Other fault–tolerant networks, like binary cube, suffer from the so–called "fan–out" problem, requiring that the number of connections per node be increased with the size of the network).

In a subsequent paper, [Prad85b], a dynamically restructurable fault–tolerant processor network architecture was presented. What is significant about these type of networks is that the inherent logical structure can be changed to fit the application in hand. Consequently, the proposed network admits efficient execution of a large class of algorithms. Even more importantly, these networks admit a measure of fault–tolerance because the faulty network exhibits all of the important properties of the fault–free network.

Of special note is a recommendation that this particular network [Prad85b] has significant potential for SDI application, from a recent study conducted by Control Data under the sponsorship of Rome Air Development Center [Appendix].

Several bus oriented fault–tolerant networks were reported on, as well, in [PrSc84].

Key considerations in the design of fault–tolerant multiprocessor systems are testing and diagnosis. A fault–tolerant system must also be testable with a high degree of confidence. Reliability of the system, itself, otherwise is compromised. Distributed self–diagnosis is a promising approach to testing/diagnosis problems. Here, by interrupting the computation, the processors are able to test certain of the other processors, determining if they are faulty. Interesting in this approach is its ability to be used in both acceptance testing and concurrent testing of multiprocessor systems. A new methodology was pursued, with the objective of minimizing testing overhead as well as of achieving greater test reliability, and/or more frequent testing. Several important results were published in [MePr85].

DeBruijn Multiprocessor Networks

Certain interesting work on networks able to solve a wide variety of problems evolved, described in some detail below.

Successfully grouped into various classes are computational problems. These are important classifications, include the *pipeline class*, *multiplex class*, the *NP-complete class*, the *ASCEND and DESCEND classes*, as well as the *decomposable searching class*.

Problems in the *pipeline class* can be efficiently solved in a pipe (known also as a linear array). Depending on the problem, data may flow in one direction or in both directions, simulataneously. Matrix-vector multiplication is a typical example of problems solvable with one-way pipeline algorithms. Band matrix-vector multiplication, recurrence evaluation and priority queues are problems that can be solved by two-way pipeline algorithms.

The *multiplex class* covers a range of problems characterized by: (1) Operation on N data operands to produce a single result; (2) Evaluation can be described by a tree. This category includes evaluation of general arithmetic expressions, polynomial evaluation, etc. The natural computation graph for this paradigm is a tree, whose nodes correspond to operations, and whose edges correspond to dataflow between operations. The CBT (complete binary tree) can be used to solve the problems belonging to this class. For another important class of problems, the *NP-complete class*, the CBT can efficiently implement exhaustive search algorithms, where time complexity still is exponential.

The *ASCEND and DESCEND classes* are comprised of highly parallel algorithms. The paradigm of the algorithms in this class is the iterative rendition of a divide-and-conquer scheme. The input and output are each a vector of $N(= 2^k)$ data items; "divide" refers to two subproblems of equal size; the "marry step" combines the results of two subproblems, executing a single operation on corresponding pairs of data items. Assume that input data $D_0, D_1, ...D_{N-1}$ are stored, respectively, in storage locations T[0], T[1], . . . ,T[N-1]. An algorithm in the DESCEND class performs a sequence of basic operations on pairs of data that are successively $2^{K-1}, 2^{k-2}, ..., 2^1, 2^0$ locations apart. In terms of the above divide-and-conquer model, the marry step involves pairs of 2^0 locations apart. In the dual class (the ASCEND class), the basic operations are performed on the data that are successively $2^0, 2^1, ..., 2^{k-1}$ locations apart; the marry step involves pairs of 2^{k-1} locations apart. These problems can be solved in the SE and the CCC.

Problems in the *decomposable searching class* can be described as illustrated. Preprocess a set F of N objects into a data structure D, such that certain kinds of queries about F can be answered quickly. A searching problem is decomposable if the response to a query Q, asking the relation of an object z to the set F, can be written as: $Q(z, F) = \delta q(z, f)$, for all f in F, where f is an element in F, δ is a binary operator which is associative, commutative and has an identity, and where q is the query asking the relation of the object z to the element f. The TM described solves this large class of *searching problems.*

Multiprocessor networks based on undirected binary de Bruijn graphs, able to solve *all* of the above mentioned classes of problems were presented in [PrSa87]. No other such network has, we believe, yet been identified. Another paper [PrSa85] presented a corollary of the above work, demonstrating that these networks can perform efficient sorting algorithms in various different input/output modes.

Also last year, in collaboration with researchers from the University of Wisconsin, we studied the problem of reconfiguration of interleaved memory in the presence of faults. These results will appear shortly in the 1987 IEEE/ACM Computer Architecture Symposium, and are described below in some detail, not reported on previously.

In a computer system that consists of a processing unit (CPU) connected to a memory system, the rate at which the CPU can process information is limited by the rate at which the memory can supply this information. Furthermore, the information-processing rate of the CPU is also limited by the bandwidth of the interface between the CPU and the memory. This is the well-known von Neumann bottleneck. Consequently, a decrease in the bandwidth of a memory system will directly affect the performance of the overall computer system.

There are two main approaches to attain a memory system with a high bandwidth. The first involves the use of a high-speed buffer or cache memory and the second involves the use of several memory banks connected in an interleaved fashion. Though the use of cache memories has become widespread, their utility is limited by their size. While cache memories are very effective for instructions and scalar data items, they have not proven to be effective for numeric processing machines that utilize large data structures (such as arrays). For such systems, in order to achieve a high-bandwidth memory system, one is forced to use interleaved banks of memory. Of course, the best effect is achieved by using a cache memory for instruction buffers (analogous to an instruction cache), large B and T register file (analogous to a cache for scalar data) and an interleaved memory for non-scalar data.

In an interleaved memory system that consists of N independent memory banks (or modules), by associating address latches and data latches with each bank, N different memory accesses can be carried out simultaneously. By doing so, the bandwidth of the memory system can be increased to N times the bandwidth of a single bank. A processing system that utilizes a cache memory for instructions and an interleaved

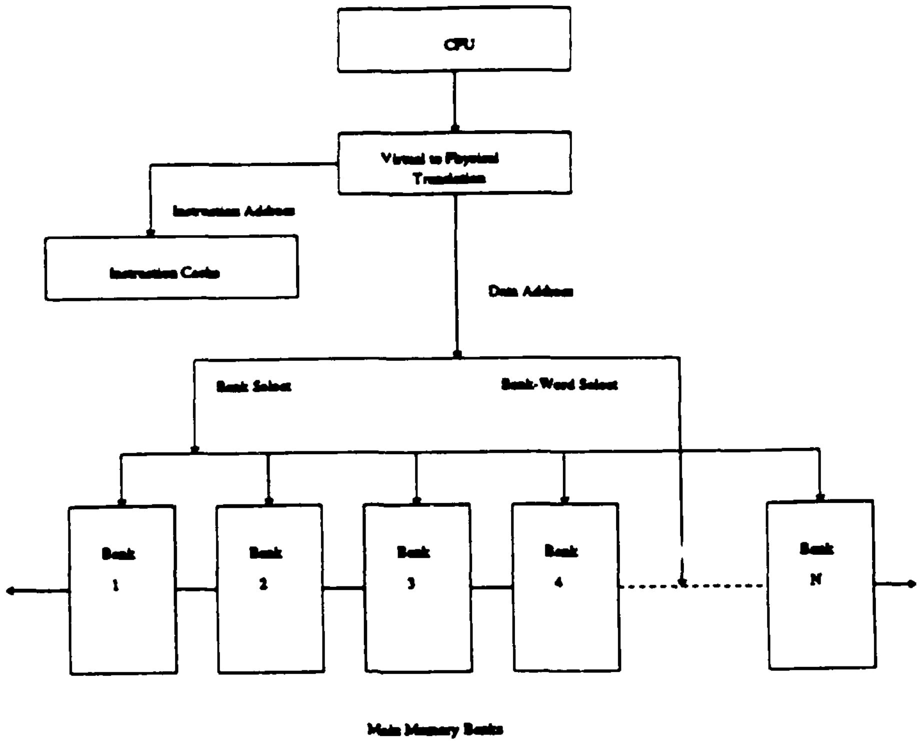


Figure 1: A Processor with an Interleaved Memory System

memory system for data is shown in Figure 1. The bandwidth of interleaved memories has been the subject of extensive study. Apart from the referencing behavior of programs, the main factor that influences the bandwidth of interleaved memory banks is the manner in which the addresses are distributed amongst the banks. Given the distribution of data amongst the memory banks, the appropriate address bits can be used to select the bank that contains the desired data item. Generally the number of banks, N that are used to build an interleaved memory is a power of 2, i.e., $N = 2^q$ where q is an integer. In such a system, q bits of the address suffice to select a bank and the remaining bits are used to select a word within a bank. If the q bits are the high–order bits of the address space, the scheme is a *high-order interleaving* scheme whereas a *low-order interleaving* scheme results if the low–order q bits are used to select the bank.

We should mention that an interleaving scheme is not restricted to using only a power of 2 number banks. Interleaving schemes that utilize a prime number of memory banks have been investigated and implemented. However, the utility of such a scheme in high performance machines is limited because of the complex logic that is needed to determine the appropriate bank/word from a given address.

In a high-order interleaved memory system, consecutive memory adddresses lie in the same bank. Therefore, if the memory is referenced sequentially, consecutive memory references access the same bank and no increase in bandwidth is obtained. In a low-order interleaved memory system, consecutive addresses lie in different banks. Now, if the memory is accessed sequentially, consecutive references will access different banks, thereby increasing the bandwidth of the memory. Since the memory referencing pattern for most programs is generally sequential (because of sequential instructions and array structures with a constant stride of unity), a low-order interleaved memory system generally has a higher bandwidth than a high-order interleaved memory system.

A low-order interleaving scheme has a major drawback – it is not modular, i.e., a failure in a single bank affects the entire address space. If no precautions are taken to handle such a situation, the bandwidth of the memory and consequently the performance of the processor could be degraded to an intolerable extent. In this paper, we study the organization of interleaved memories such that faults in the memory system degrade the performance in a graceful manner. We restrict our study to an interleaved memory system that starts out with a power of 2 number of banks and uses a low-order interleaving scheme. The ideas presented in [Pr et.al.87] can be extended to other interleaved memory schemes.

Faults in Interleaved Memories

Consider a memory that consists of several *groups* of interleaved memories with each group consisting of several banks. The number of banks in a group is a power of 2, say 2^r, and the banks within a group are fully interleaved. Thus, the banks within a group can be selected using a r–bit bank selection address field. Different groups can have a different number of banks in them. Thus group G_1 may consist of 4 banks while group G_2 may have only 1 bank. If the total number of banks in the memory system is 2^k where k is an integer, then there is only one group. This is the situation that exists in a conventional interleaved memory system without any faulty banks. Therefore, if each of the 2^q banks in the single group contains 2^p words, then the total addressable main memory of the system is 2^q (where $n = p + q$) words. Using a low- order interleaving scheme, bits $A_{q-1} \ldots A_0$ of the n–bit address $A_{n-1}\ A_{n-2}\ \ldots\ A_0$ (where A_{n-1} is the most significant bit) are used to select the bank and the remaining p bits, i.e., bits $A_{n-1}\ A_{n-2} \ldots A_q$ are used to select a word within a bank.

Consider what happens when a bank is deleted from a memory system that contains a single group of banks. This is exactly what happens when a fault in the memory system results into the loss of one complete bank. Therefore, our fault model is that a fault results in the loss of a complete bank of memory. We assume that a mechanism that detects the presence of a faulty bank exists. Such a fault–detection scheme is not the subject matter of this paper.. Our main thrust is to evaluate the loss in performance when a fault is reported and how the memory system might be organized so that the resulting degradation in performance is graceful.

If the memory system loses one bank, the number of banks in memory is reduced to $2^q - 1$ and the total addressable physical memory is reduced to $(2^q - 1)2^p$ words. The program must be stopped, correct information recovered from the backup store, the address translation mechanism informed about the faulty bank and program execution restored. However, since $2^q - 1$ is not a power of 2, the banks no longer form a single group and the system loses its capacity to interleave memory requests. Without interleaving, the bandwidth can be catastrophic to the performance of a high-speed CPU. What could we possibly do to salvage some of this lost memory performance? Two approaches follow.

The first approach involves the use of spare memory banks. After a faulty bank is detected, a spare bank can take its place. However, as more banks become faulty, the system will eventually run out of spare banks if the spare banks cannot be replaced. Once all the spare banks have been exhausted, another fault-tolerance scheme must come into play.

Reconfiguration of Non-faulty Banks

An alternative approach is to reconfigure the remaining non-faulty banks in order to salvage some of the lost performance. Such an approach could also be used if a system has spares but runs out of them eventually. The banks are reconfigured so that their bandwidth is improved. No doubt, a smaller physical memory will result in a larger probability of a page fault.

How might we organize the fault-free banks so that the performance is not degraded to an intolerable extent? A simple solution that could be used to salvage some of the lost bandwidth is to reduce the number of addressable banks to the nearest power of two, i.e., 2^{q-1} thereby achieving a maximum bandwidth of 2^{q-1} words per memory cycle. While the address translation and bank selection mechanism is quite straight-forward, $2^{q-1} - 1$ banks of fault-free physical memory are not addressable and therefore unutilized. With such a simple reconfiguration scheme, although the bandwidth may be high, it is likely to result into a high page fault rate compared to a scheme which uses all $2^q - 1$ fault free memory banks. A scheme that does not utilize all $2^q - 1$ memory banks may, therefore, perform poorly in spite of its high bandwidth. Hence, any scheme used to improve the memory performance must not only organize the fault-free banks in an interleaved manner to maximize the bandwidth, but it must also make sure that the available memory is being utilized to its fullest extent so that the performance degradation due to page faults is minimized.

Another important factor that must be kept in mind is the effect of the reconfiguration hardware on delay in the address and data path. One advantage of a low-order interleaving scheme is that the decoding logic needed to generate the bank select signals is very simple. Therefore, the delay in the address path between the processor and main memory is small. It is desirable to keep this delay small because any delay in this path will have a direct impact on the latency of each memory request. In the proposed reconfiguration, we pay special attention to this *critical path* performance. Thus, in addition to making the best use of available memory resources, we must also minimize the delay due to the additional hardware.

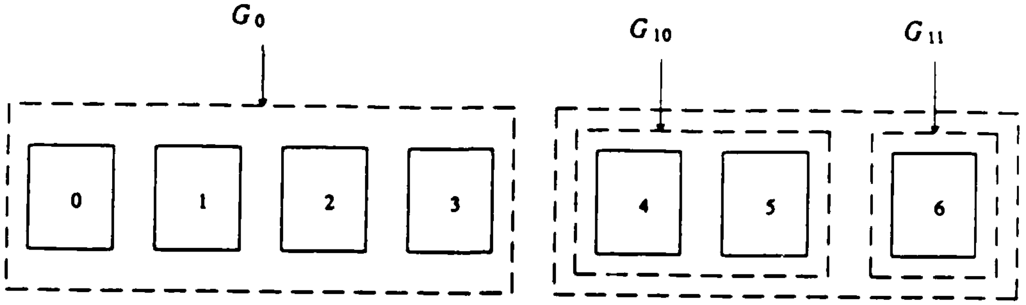


Figure 2: Partitioning Banks into Groups

The Reconfiguration Scheme

The proposed scheme reconfigures the remaining banks using a combination of high–order interleaving and low–order interleaving. All non–faulty banks are partitioned into sets. Thus, if $2^{q-1} - 1$ banks were available, they would be partitioned into q sets. These q sets form 2 subsets; subset $S_0(2^{q-2})$ and $S_1(2^{q-2} - 1)$. A set containing a power of 2 banks is called a *group*. Therefore, $S_0(2^{q-1})$ has one group $G_0(2^{q-1})$ that has 2^{q-1} banks and $S_1(2^{q-1} - 1)$ is made up of group $G_{10}(2^{q-2})$ which has 2^{q-2} banks and the subset $S_1(2^{q-2} - 1)$ which has $(2^{q-1} - 1)$ banks. This recursive partition stops when S_1 has only one bank. Clearly, the number of banks in each group is a power of 2, with unity being a special case. An example of the partitioning of 7 memory banks into groups is given in Figure 2. Banks within each group $G_i(2^k)$ are organized for low–order interleaving; high–order address bits are used to determmine the group. If there is only one group, e.g., in the fault–free case, no group selection needs to be done. The low–order q bits of the address select the bank and the high–order p bits of the address select the word within the bank. With one fault, the number of groups becomes q with the number of banks $= 2^q - 1$. Therefore, q bits suffice to uniquely identify $2^q - 1$ correct and one faulty bank. An address is decoded as follows: the most significant bit of the address, A_{n-1}, is used to select either group $G_0(2^{q-1})$ or subset $S_1(2^{q-1} - 1)$. If group $G_0(2^{q-1})$'s selected, then bits $A_{q-2}...A_0$ are used to select one of the 2^{q-1} banks within the group and bits $A_{n-2}...A_{q-1}$ are used to address the word within the bank. If $S_1(2^{q-1} - 1)$ is selected, then bit A_{n-2} of the address is used to select either $G_{10}(2^{q-2})$ (with bits $A_{q-3}...A_0$ used to select a bank within this group) or $S_1(2^{q-2})$ and so on. Note that this group identification scheme resembles the decoding scheme used to decode Huffman-encoded information. If there is only a single faulty bank, it is always indicated by a string of 1's in the q bits that are used to identify the bank number. Once the group number has been determined from the address, the appropriate p bits are used to select the work within the bank. The logic that decodes the address is now more complex than a simple decoder. We call this logic the *Address Transliterator (AT)*. Each memory address now passes through the AT before it is forwarded to the memory system (Figure 3). The design of the AT is discussed in detail in Section 4. The inputs to the AT are n-bit physical memory address and a 2^q-bit vector, the *Bank Status Indicator (BSI)*, that indicates the status of each bank. The output from the AT is the appropriate bank address and the address of the work within the bank.

Performance Evaluation

In order to evaluate the performance of the reconfigured memory system, we

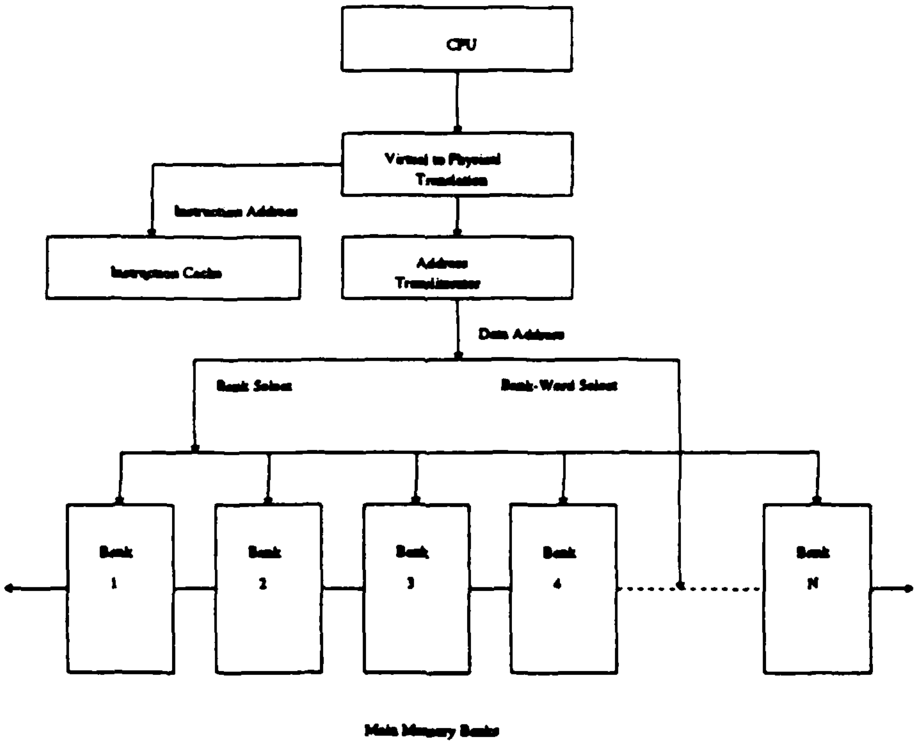


Figure 3: Interleaved Memory System with an Address Transliterator

carried out a trace-driven simulation of several programs on the VAX-11. A trace of instruction and data references were obtained for each program. Data from the trace files was fed into a program that simulates the interleaved memory system and determines the bandwidth. The simulation model used was similar to that used by other researchers. Memory references are divided into instruction references (put in the instruction queue) and data references (put in the data queue). During each memory cycle, requests in both queues are scanned alternately. The scanner stops admitting requests if a bank conflict occurs. We make the following assumptions; no interactions may occur between the instruction request stream and the data request stream, ii) no self-modifying code and iii) an additional data cycle is issued if the data queue is full.

The Performance Metric

Along with the bandwidth of the reconfigured memory system, the other important performance measure is the number of page faults. We combine these two measures into a single metric, P. The performance metric P is defined as:

- P=total data trace length (total data references/data bandwidth) x data pages allocated to a process + time to process a page fault x data pages allocated to a process x number of data page faults.

The metric P is then normalized with respect to P for the fault-free case.

Experiments and Results

We evaluated the reconfigured memory system for 4 different programs: i) *nroff*, which is a text formatter, ii) *compact*, which is a program that compresses a file using an adaptive Huffman encoding, iii) *boyer*, which is a theorem proving program and, iv) *tak*, which is an execution of the Takeuchi function. Initially, 16 banks of memory are present. The system uses demand paging with a page size of 2K bytes and a bus width of 4 bytes. The number of instruction and data references traced and the number of data pages used during a trace of the program is given in Table 1.

A program is allocated a fixed number of data pages (maximum of 32) for its use. A least recently used (LRU) replacement policy is used to replace a page when a page fault occurs and no free page frame is available. We assume that all instruction references are serviced by the instruction cache, i.e., only data requests to the interleaved memory. A loss of memory bank results in less data pages available for the program, but does not affect the instruction pages. Thus, the number of data pages allocated to a program is reduced when a bank is reported to be faulty. The

Trace	Trace Records		Data Pages
	Instruction	Data	Referenced
nroff	281513	178832	55
compact	233638	205298	23
boyer	217147	229871	216
tak	49814	54590	170

Table 1: Statistics for the Benchmark Programs

pages are distributed amongst the groups of the reconfigured memory in proportion to the number of banks in the group. Any page lies completely within a group. For example, if there are 2 groups consisting of 8 and 4 banks respectively, a process will place 67% of its data pages in the group of 8 banks and the remaining pages in the group of 4 banks. The time to process a page fault is 2000 memory cycles.

Using the above parameters, we calculated the value of P as the number of addressable memory banks reduced. The results are presented in Tables 2–5. In all cases, there is a significant increase in the page fault rate when fewer memory banks (pages) are available. The decrease in data bandwidth, however, is not very significant. The performance metric P degrades in a graceful manner as opposed to a sudden change if the number of addressable memory banks was reduced to 8 when a single bank became faulty.

Number	Data				P
of Banks	Bandwidth	Page Faults	Page Fault rate (%)	Pages	(normalized)
16	2.934366	58	0.032433	28	1.000000
15	2.744169	83	0.046412	26	1.213130
14	2.738286	136	0.076049	24	1.633969
13	2.740889	141	0.078845	22	1.541935
12	2.735506	173	0.096739	20	1.660630
11	2.663697	217	0.121343	18	1.820683
10	2.698165	300	0.167755	16	2.151699
9	2.675584	362	0.202425	14	2.234710
8	2.614926	1180	0.659837	12	5.881731

Table 2: Result for *nroff*

Number	Data				P
of Banks	Bandwidth	Page Faults	Page Fault rate (%)	Pages	(normalized)
16	2.906916	23	0.011203	14	1.000000
15	1.945492	51	0.024842	13	1.652334
14	1.959661	171	0.083294	12	3.283534
13	2.796937	467	0.227474	11	6.787020
12	2.698449	1048	0.510477	10	13.303313
11	2.585389	2334	1.136884	9	26.168747
10	2.648802	3986	1.941567	8	39.440582
9	2.573625	5382	2.621555	7	46.490295
8	2.546427	6497	3.164668	6	48.046799

Table 3: Result for *compact*

Number	Data				P
of Banks	Bandwidth	Page Faults	Page Fault rate (%)	Pages	(normalized)
16	3.283168	1465	0.637314	32	1.000000
15	2.808648	1696	0.737805	30	1.085571
14	3.109895	1962	0.853522	28	1.166053
13	3.042513	2328	1.012742	26	1.281456
12	3.142504	2754	1.198063	24	1.395280
11	2.957110	3209	1.396000	22	1.488599
10	2.997978	4454	1.937609	20	1.871798
9	2.223019	6696	2.912938	18	2.530375
8	2.997796	8895	3.869561	16	2.977763

Table 4: Result for *boyer*

Number	Data				P
of Banks	Bandwidth	Page Faults	Page Fault rate (%)	Pages	(normalized)
16	3.687766	605	1.108262	32	1.000000
15	2.936841	674	1.234658	30	1.046026
14	3.110719	759	1.390365	28	1.096997
13	2.535061	823	1.507602	26	1.106195
12	3.379349	895	1.639494	24	1.105987
11	3.261051	993	1.819015	22	1.124167
10	2.967439	1112	2.037004	20	1.144263
9	3.130943	1198	2.194541	18	1.108388
8	3.266726	1389	2.544422	16	1.140882

Table 5: Result for *tak*

3 Publications Supported by AFOSR 84–0052

[Prad86] Fault-tolerant Computing: Theory and Techniques, Vol. I and Vol. II, Prentice-Hall, Inc., May 1986. (Editor and Co–Author).

[PrKo87] "Modelling the Effect of Redundancy on Yield and Performance of VLSI Systems," *IEEE Transaction on Computers*, January 1987 (with I. Koren).

[PrKo86] "Yield and Performance Enhancement through Redundancy in VLSI and WSI Multiprocessor Systems," *Proceedings of IEEE*, (with I. Koren) May 1986.

[Prad85b] "Dynamically Restructurable Fault-tolerant Processor Network Architectures," *IEEE Transactions on Computers*, May 1985.

[PrKo85a] "Fault-tolerant Multilink Multibus Structures," *IEEE Transactions on Computers*, Vol. C-34, No. 1, January 1985.

[PrKo85a] "Introducing Redundancy into VLSI Designs for Yield and Performance Enhancement," *Proc. FTCS-15*, pp. 330–334, Ann Arbor, Michigan (with Israel Koren), June 1985.

[PrMe85] "Dynamic Testing Strategy for Distributed Systems," *Proc. FTCS-15*, Ann Arbor, Michigan, (with Fred Meyer), June 1985.

[PrSa85] "A Versatile Sorting Network," *Proc. 12th Annual Symposium on Computer Architecture*, (with M.R. Samatham), June 1985.

[PrSc84] "Fault-tolerant Multibus Architectures for Multiprocessors," *Proc. FTCS-14*, June 1984, Kissime, Florida, (with M.L. Schlumberger and Z. Hanquan) pp. 400-408.

[PrSa84] "A Multiprocessor Network Suitable for Single Chip VLSI Implementation," *Proc. 1984 IEEE 11th Annual Int. Symp. on Computer Architecture*, June 1984, pp. 328-337.

Publications Pending

[PrSa87] "The De Bruijn Multiprocessor Network: A Versatile Parallel Processing Network for VLSI," *IEEE Transactions on Computers*, (under revision).

[Pr er al. 87] "Design of Gracefully Degrading Interleaved Memory System", *Proc. IEEE/ACM Annual Symposium on Computer Architecture*, June 1987.

Graduate Students Supported Through AFOSR 84–0052

M.R. Samatham (Ph.D. Student)

Fred Meyer (Ph.D. Student)

John Petrovick (M.S. Student)

Najmi Jarawala (M.S./Ph.D. Student)

4 Research in Progress and Future Directions

4.1 Consensus with Dual Failure Modes

Consider a distributed system of processing elements (nodes) connected by a point-to-point network. One of the common problems is to maintain clocks (or some other concept of time) in approximate synchrony despite differences in the clock rates of individual processors and despite some faulty processors.

We consider a generalization of the clock synchronization problem called interactive consistency that involves reaching agreement on some value being sent by one of the processors, say s. Let $v(s)$ be the value that s wishes to transmit. Each processor decides on some value as having been sent by s.

Interactive consistency is defined as satisfying the following two requirements:

- **agreement** any two good processors decide that the same value was sent by s
- **sanity** if s is a good processor, then the value that any good processor decides was sent by s is, in fact, $v(s)$

The agreement requirement ensures that faulty processors cannot cause two good processors to "believe" different things. The sanity requirement ensures that consistency cannot be achieved by simply agreeing on a default value.

Figure 4 shows why it is impossible to reach agreement in the presence of one (malicious) fault when there are only three processors. The left illustration depicts the nature of messages sent when node i is transmitting the value 1 and node j is faulty. The right illustration depicts the nature of messages sent when node i is faulty and sends conflicting values to j and k. Node k is unable to distinguish between these two scenarios.

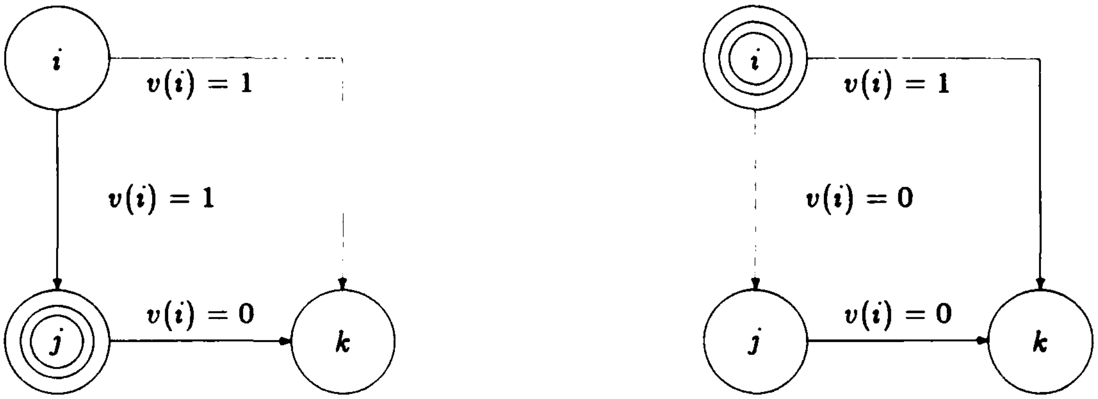


Figure 4: Agreement impossible with only three nodes

We consider systems where either or both of two types of faults may occur: benign or malicious. We have accomplished the following: (1) established a taxonomy of processor failure modes, (2) established a continuum between previous results on necessary and sufficient conditions for consensus algorithms to exist, and (3) developed more general algorithms to handle systems with two failure modes—achieving much improved reliability.

We have established a taxonomy that categorizes faults by their behavior. (See F. Meyer and D. Pradhan, "Consensus with dual failure modes," *Proc. 17th Fault-Tolerant Comput. Symp.*, for details). There are many possible failure modes. The three most thoroughly examined failure modes are: (1) Byzantine, (2) Authenticated, and (3) Dormant. Figures 5, 6, and 7 illustrate the sort of behavior that such faults may exhibit. The Dormant (Byzantine) failure mode has the most (least) constraints on its behavior.

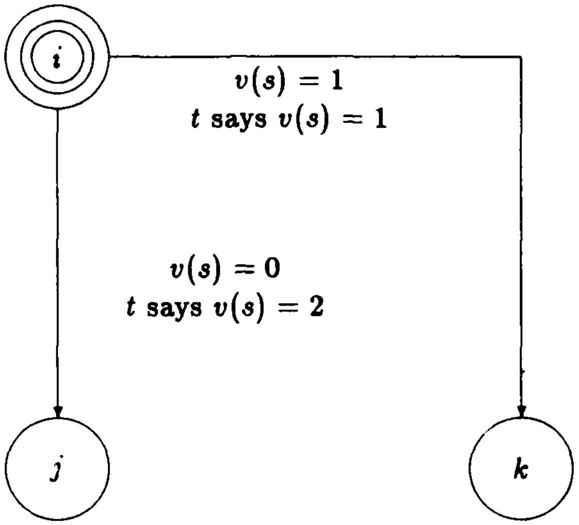


Figure 5: Byzantine fault exhibits arbitrary behavior

We have taken malicious failures to be Byzantine and benign failures to be Dormant. Previous researchers have developed algorithms that function in the presence of one of these two failure modes. F. Cristian, et al, "Atomic broadcast: From simple message diffusion to Byzantine agreement," *Proc. 15th Fault-Tolerant Comput. Symp.* gave a very simple algorithm that contends with any number of benign failures. But the algorithm almost always fails whenever a failed processor exhibits any behavior outside the constraints of a dormant failure (for instance, if it sends an incorrect message). D. Dolev, et al, "An efficient algorithm for Byzantine agreement without authentication," *Information and Control*, no. 52, gave a relatively efficient algorithm to contend with

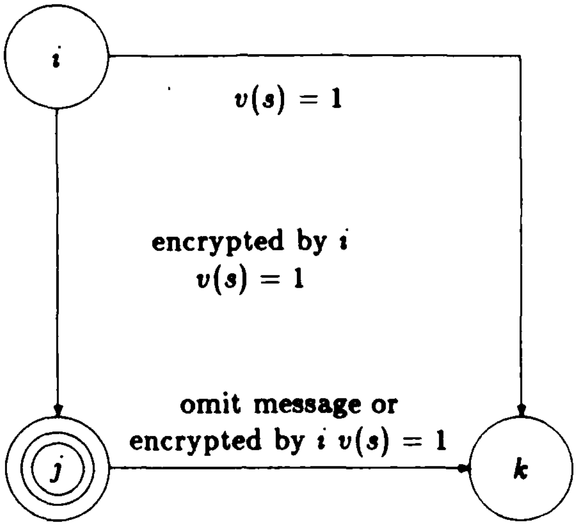


Figure 6: Authenticated fault is constrained by encrypted messages

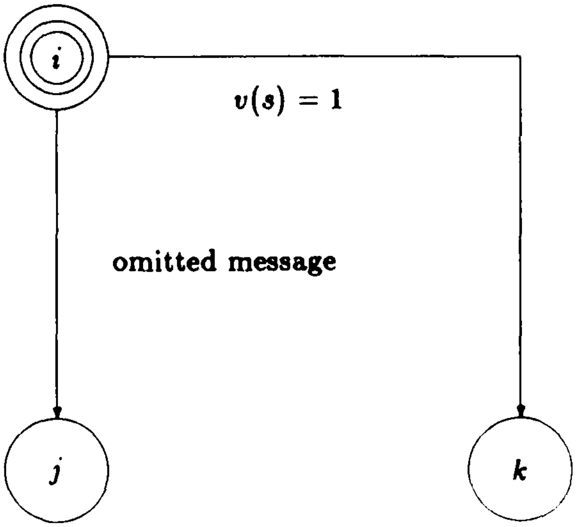


Figure 7: Dormant fault commits only faults of omission

a maximal number (about one-third of the processors) of malicious failures.

Figure 8 plots the reliabilities achieved by these two algorithms. The plot is for a 64-node system. The horizontal axis plots the probability a given fault is malicious for the range from zero to ten percent. The vertical axis shows the negative of the common log of the probability that the faults present exceed what the algorithm is rated to handle (so 2.0 equates to a probability of one percent). It might seem that for most systems, probability malicious is small. This plot shows that the system designer cannot casually assume that probability malicious is negligible. Of course, reliability decreases as probability malicious increases, even though the plot shows the reliability of the [Dolev, et al] algorithm increasing. We have kept the expectation of [the number of failures plus twice the number of malicious failures] constant so that the plot would be better constrained on the vertical axis. So the slope of the plot for the [Cristian, et al] algorithm is even steeper.

[Dolev, et al] suggested a way of moderately reducing the number of messages sent when few faults are expected. We have modified this algorithm to allow for a variable reduction in message complexity and to take advantage of the extra messages sent by using them to tolerate benign failures. As a result, we can tolerate a mixture of benign and malicious failures, thereby improving reliability. The reliability achieved, however, is slightly sensitive to the accuracy of the designer's estimate of probability malicious.

To overcome this sensitivity, we have developed another algorithm (called mixed-sum algorithm) that achieves still greater reliability and does not depend on probability

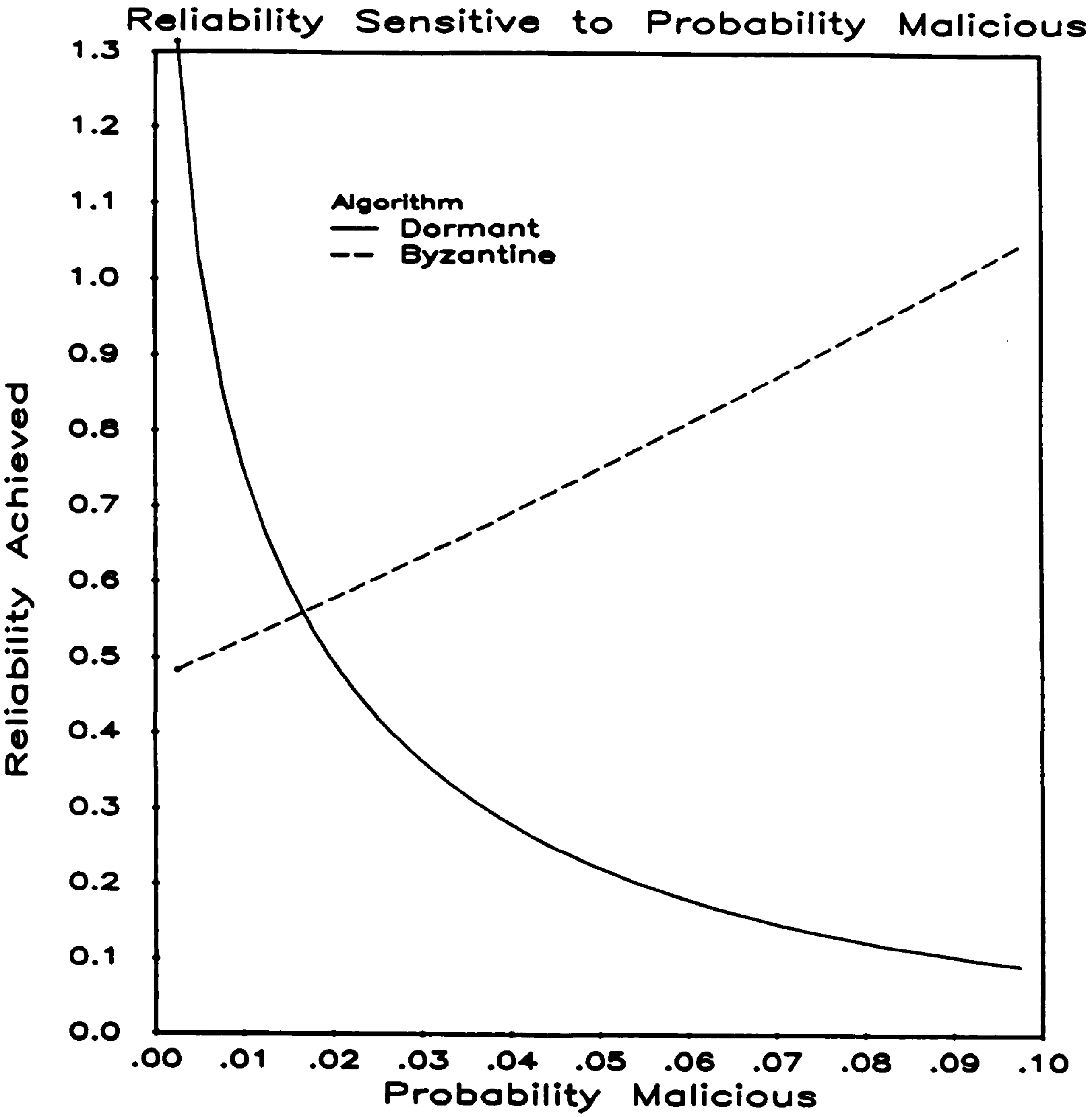


Figure 8: Reason for Considering Dual Failure Modes

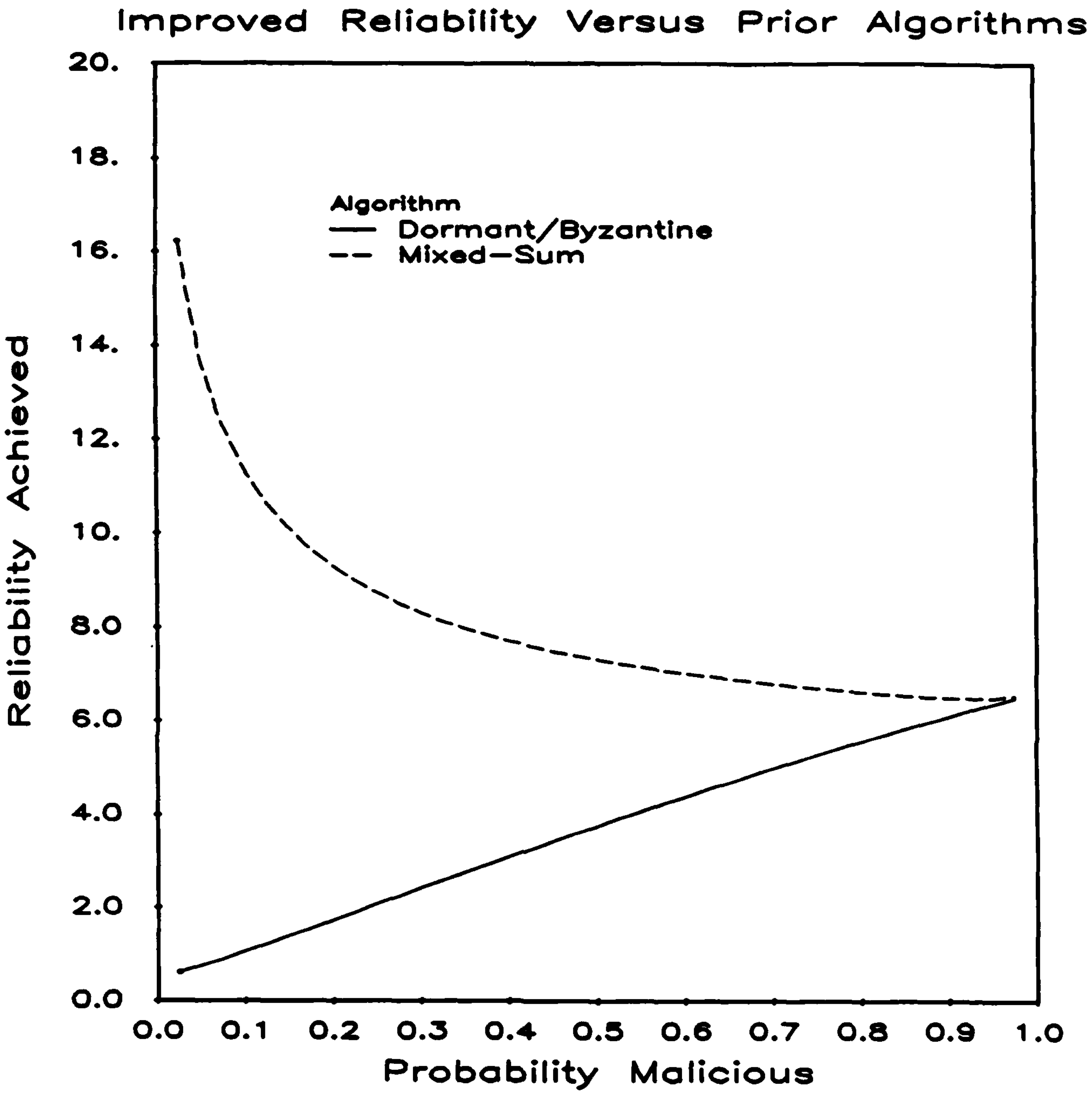


Figure 9: Potential Reliability Improvement

malicious. This algorithm is an adaptation of the algorithm given by L. Lamport, et al, "The Byzantine generals problem," *ACM Trans. Prog. Lang. & Sys.*, vol. 4, no. 3. Our algorithm achieves the provably maximal reliability under the dual failure mode model. Figure 9 plots the reliability achieved by (1) the better of the [Cristian, et al] or [Dolev, et al] algorithms against (2) our mixed-sum algorithm. Attention should be drawn to the significant improvement in the critical region (0 to 10 percent probability of malicious failure).

The [Lamport, et al] algorithm has a very large message complexity. While our mixed-sum algorithm shares this drawback, we are continuing this research to (1) develop a more efficient mixed-sum algorithm, (2) analyze other failure modes suggested by our taxonomy, and (3) consider bus networks (instead of point-to-point).

4.2 Methodologies for Designing Defect–Tolerant, Reliable, Testable VLSI Systems and Evaluating their Costs

This research aims at developing methodologies for designing VLSI systems which are: defect–tolerant, reliable, testable. Also it evaluates the penalty that has to be paid in terms of area, performance, yield, etc., to achieve such properties. Random Access Memories (RAMs) are to be investigated initially, because these devices are the highest density VLSI devices to be fabricated commercially. Though they are very difficult to test, their regularity permits innovation and experimentation and the results obtained can be extended to other, less regular systems. Finally, because of the low cost and wide usage of RAMs, any architectural improvements that result in increased yield or enhanced performance is likely to have considerable practical significance.

4.3 Results

4.3.1 TRAM Architecture

A new architecture for Multi-Megabit RAMs, the Tree Random Access Memory (TRAM) architecture, has been developed. Applying the principle of divide and conquer, the RAM is partitioned into modules, each appearing as the leaf node of a binary interconnect network. This network carries the address/data/control bus which permits the nodes to communicate with the outside world and with certain test logic embedded within the chip. Such an architecture is shown to be fault-tolerant, improving both yield and reliability. Also, it is easily partitionable, improving the probability of generating partially good products. Parallelism in testing, and partial self test results in a dramatic savings of testing time. Finally, unlike other fault tolerant/testability schemes, this approach promises *improved* performance in terms of lower access times, as well as reduction in the time required to refresh the device. These benefits are obtained at only a small increase in chip area. These results are obtained by detailed VLSI area/performance models that take into account implementation and technology dependencies.

4.3.2 On chip error control coding for yield and reliability enhancements in dynamic RAMs

Reduction in the DRAM cell size has increased its susceptibility to alpha particle radiation. On chip error detection and correction can provide operational fault tolerance against these soft errors. This research proposes and analyzes two new coding techniques for on chip ECC: the product code with *full* code word correction on each access and the odd-weight-column codes. Our proposed design differs from earlier designs in its implementation, with the potential for better performance as well as better reliability through smaller error latency. The area/performance costs of implementing these codes are analyzed for three RAM sizes - 1M, 4M and 16M - and for varying numbers of information bits - from 64 bits to 2K bits. The analysis shows that the area cost of implementing these codes is low ($\leq 10\%$) for large RAMs. For each of the RAM sizes, it also predicts the optimal number of information bits for both codes that will minimize area and performance cost. Overall, it is seen that the odd-weight-column codes have a lower area and performance cost. The analytical model used is quite general and can be used to analyze the cost/performance of other codes, as well as other fault/defect tolerant techniques.

4.4 Work in Progress

1. For the TRAM architecture, techniques to simplify restructuring the architecture–both during fabrication and in real time–are being explored. These would not only simplify the generation of partially good products, thus increasing the effec-

tive yield, but also permit graceful degradation in the event of real time failures. Detailed yield and reliability modelling is in progress.

2. A major implementation of the TRAM architecture is in progress. A 256K RAM, organized as 16 nodes of 16K bits each, is being designed in 1.25μm CMOS technology. This will be fabricated by MOSIS and tested.

3. Yield and reliability analysis is in progress for DRAMs using on chip error control coding. An interesting possibility that is being explored is the use of the hardware that is already present for generating the checkbits/syndrome, for aiding off line testing.

4. The TRAM architecture is being extended for Wafer Scale Memory Systems. Because of its hierarchical redundancy, easy restructurability and the H-tree bus structure that equalizes the access time to all nodes, this architecture is particularly suited for Wafer Scale implementation. However, the degree of redundancy, and the area, yield, performance, testability tradeoffs are very different for WSI and these are being explored.

5. The concepts developed for RAMs are being extended to other parallel computing structures.

Appendix

1. Copy of letter from Control Data to RADC

2. Copies of selected publications:

- D.K. Pradhan, "Fault–Tolerant Multiprocessor Link and Bus Network Architectures", *IEEE Transactions on Computers*, January 1985.

- F.J. Meyer and D.K. Pradhan, "Dynamic Testing Strategy for Distributed Systems", *Proc. FTCS-15*, June 1985

- I. Koren and D.K. Pradhan, "Yield and Performance Enhancement through Redundancy in VLSI and WSI Multiprocessor Systems", *IEEE Proceedings*, Vol. 74, May 1986.

CONTROL DATA

8800 Queen Avenue South
Mailing Address/Box 1305
Minneapolis, Minnesota 55440-1305

January 7, 1987

Prof. Dhiraj K. Pradhan
Department of Electrical
and Computer Engineering
University of Massachusetts
Amherst, MA 01003

Dear Prof. Pradhan:

Attached is a copy of the letter I wrote to Mr. Kaminski at RADC. Mr. Kaminski is the person you need to contact for a copy of the Tightly Coupled Network for VHSIC Architectures Final Report.

Sincerely,

James W Chapman

James W. Chapman
Control Data Corporation

/sh
Attachment

CONTROL DATA

8800 Queen Avenue South
Mailing Address/Box 1305
Minneapolis, Minnesota 55440-1305

December 16, 1986

RADC/COTC
Griffiss Air Force Base
Rome, New York 13441

ATTN: Robert Kaminski

Dear Mr. Kaminski:

As an author of the Tightly Coupled Network for VHSIC Architectures Final Report that was recently submitted by Control Data, I request that Professor D. K. Pradhan be provided a copy of the report. The Professor has intense interest in fault-tolerant network architectures, and some of the results of his research has been applied in the TCN design concept presented in that report.

He is engaged in DoD sponsored research in this area under Grant AFSOR 84-0052. His mailing address is:

Prof. Dhiraj K. Pradhan
Department of Electrical
and Computer Engineering
University of Massachusetts
Amherst, MA 01003

Sincerely,

James W Chapman

James W. Chapman
Control Data Corporation

/sh

Fault-Tolerant Multiprocessor Link and Bus Network Architectures

DHIRAJ K. PRADHAN, SENIOR MEMBER, IEEE

Abstract — **This paper presents a general class of regular networks which provide optimal (near-optimal) fault tolerance.**

The proposed networks compare favorably to other regular networks such as leaf-ringed binary trees and cube networks. In particular, the networks proposed possess certain advantages in that the number of connections per node is neither an arbitrarily fixed number (as in leaf-ringed trees) nor does it grow arbitrarily large with the size of the network (as in cube networks). This point has significant relevance to fault tolerance in that the degree of fault tolerance provided by the network can be varied according to the design specification. Also, the networks admit simple self-routing of messages and that routing is adaptable to faults.

Index Terms — **Algorithmic routing, circuit switching, connectivity, diameter of graphs, fault-tolerant communication network, multiple bus network, multiprocessor networks, packet switching, regular graphs, regular networks, shared-bus fault tolerance, shuffle-exchange graph.**

I. Introduction

RECENT developments in technology have made it possible to interconnect a large number of computing elements in order to form an integrated system. Various network architectures have been proposed that are suitable for both multiprocessors and VLSI systems [1]–[8], [11]–[14], [19]–[27].

The likelihood increases of one or more elements failing with the increasing number of elements in the system. Consequently, a key consideration in the design of such systems is their overall reliability and fault tolerance. The fault tolerance of a system can be defined in various ways. One measure that possesses relevance to a system which consists of a large number of homogeneous elements is the maximum number of elements which can become faulty without disconnecting the system. That is, the assumption is made that the system can perform in a degraded mode with loss of one or more components, as long as the system is fully connected. Also, more importantly, each element can still be capable of communicating with all of the other elements in the system with ease, in spite of the faults. In the context of communication delays, the performance degradation that is due to faults may be measured in terms of the increase in path lengths, and the associated increase in the routing overhead. So, it is not only important that the system remain fully connected, but also that the nodes be able to communicate with each other fairly easily — preferably with only minor modification to the original routing procedure. It is precisely in this framework that a class of new fault-tolerant architectures has been developed here.

The proposed class of networks is regular in that all of the system nodes (elements) possess the same number of connections per node. The proposed networks favorably compare to other regular networks such as the binary cube [1], [6], generalized hypercube networks [13], cube-connected cycles [2], leaf-ringed binary tree networks [3], and De Bruijn graph networks [6], [10], [14] as seen from Table I.

In general, the proposed networks possess the following attractive features.

1) Compared to other networks, the proposed networks possess certain advantages in terms of their number of connections per node. Specifically, networks such as the binary cube require that the number of connections per node increases with the number of nodes (whereas cube-connected cycles and binary tree networks use nodes that only have three connections per node). On the other hand, the proposed network of any arbitrarily large size can be built using nodes with any specified number of connections per node. For example, given nodes with 5 connections per node, one can build a network of 256 nodes ($r = 4, m = 4$), or 1024 nodes ($r = 4, m = 5$), or in general, any arbitrarily large 4^m node network.

2) The internode distances are *small*. The maximum internode distances are proportional to only the *logarithm* of the number of nodes; inversely, to the logarithm of the number of connections per node.

3) More importantly, the networks are capable of maximal or near-maximal fault tolerance.

4) Degradation that is due to an increase in the routing distances and communication overhead resulting from faults can be fairly low. A detour technique is presented that allows the network to degrade proportionately to the number of faults.

5) Also of interest is the fact that the network admits self-routing of messages, both when the network is fault free as well as when it is faulty. Self-routing refers to that ability to route messages from node to node by using information such as destination address tag bits contained within the message and where intermediate nodes perform no additional computation for routing. This is possible if the routing path can be determined algorithmically, without using routing tables and directories.

Manuscript received June 24, 1982; revised December 28, 1983 and August 2, 1984. This work was supported by the Air Force Office of Scientific Research under Grant AFOSR 84-0052.

The author is with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003.

0018-9340/85/0100-0033$01.00 © 1985 IEEE

TABLE I
DIFFERENT INTERCONNECTION NETWORKS

Network	Number of nodes	Number of connections/node	Routing distance	Fault tolerance
Leaf-ringed binary tree	$(2^m - 1)$	3	$(2m - 1)$	2
Cube-connected cycles	3.2^m	3	$m^2\ 2$	2
Binary cube	2^m	m	m	$(m - 1)$
Generalized hybercube	$\prod_{i=1}^{m} x_i$	$\sum_{i=1}^{m} x_i$	m	$\sum_{i=1}^{m} (x_i - 1)$
DeBruijn network	r^m	$2r$	m	$(2r - 3)$
Proposed network	r^m	r or $(r - 1)$	$(2m - 1)$	$(r - 1)$ or r

The next section presents the proposed network. Following this, Section III and Section IV develop the fault tolerance of the networks.

II. Proposed Network

First, given a graph structure one can formulate a link network and a bus network based on the graph structure as described below. Let $G = \langle V, E \rangle$ be an undirected graph where V is a set of n vertices represented as 0 through $(n - 1)$. The set E represents the edges denoted as (i, j) where i and j are two neighboring nodes connected by (i, j) in G. Let LA and BA be defined as two mappings of G into a link network and a bus network, as described below.

Let $LA(G) = \langle \mathrm{PE}, C \rangle$ where PE is a set of processing elements, represented as PE(0), PE(1), $\cdots$, PE$(n - 1)$, which corresponds to the set of vertices in G. Let C be the set of bidirectional communication links. There is a communication link $C(i, j)$ in C which connects PE(i) with PE(j) iff $(i, j) \epsilon E$.

Let $BA(G) = \langle B, \mathrm{PE} \rangle$ where B represents a set of buses defined as BUS(0), BUS(1), $\cdots$, BUS$(n - 1)$, corresponding to n vertices. The set PE represents the set of processing elements defined as PE(i, j) ϵ PE iff $(i, j) \epsilon E$. The processing element PE(i, j) is connected to buses BUS(i) and BUS(j) as shown in Fig. 1. Thus, the link architecture is obtained by using the interpretation that vertices denote processing elements and edges denote communication links. On the other hand, the bus architecture is obtained by using the interpretation that buses are shown as vertices and computing elements as edges in the graph. Thus, the number of processing elements in $BA(G)$ is equal to the number of edges in G. Each processing element is connected to two buses and each bus is connected to a subset of the processing elements. (The number of processing elements connected to BUS(i) is equal to the degree of node i in G.) This differs from the conventional multiple-bus design where all processing elements are connected to all buses. Since each bus is connected to a subset of processing elements, an inter-PE transfer may require several interbus transfers.

However, if bus load is equated to the number of connections per bus, then the $BA(G)$ network has a much smaller bus load when compared to an equivalent design which uses conventional shared buses. Therefore, a $BA(G)$ type bus network may have certain advantages over the conventional shared multiple-bus design when there are large numbers of processing elements to be connected. Also, it may be noted that one of the advantages of a $BA(G)$ network over $LA(G)$ network is that it can be easily extended by adding more PE's to buses as required and there are well developed bus protocols available.

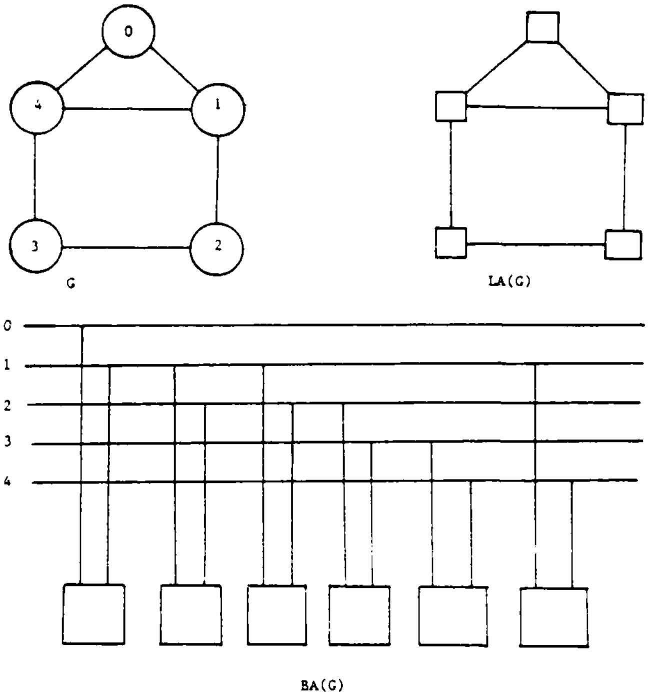


Fig. 1. Link and bus architecture.

This alternate multiprocessor multibus architecture can be quite attractive where

1) a processor may not have the hardware capabilities to allow its attachment to more than a certain number of buses, and

2) for reasons of reliability, the buses may be in physically different locations; hence, a processor may not be located next to every bus.

Various relationships between G and the corresponding $LA(G)$ and $BA(G)$ are described in Table II. In describing the FG networks below first the underlying graph structure FG is defined.

FG Network Design: The number of nodes in the graphs defined below is assumed to be equal to r^m. As seen later, the chosen values of r and m will determine the number of connections per PE (or BUS), the routing distance between PE's, and the degree of fault tolerance.

Here, the nodes are assumed to be numbered 0 through $(n - 1)$. Each node i has an m-tuple representation in radix-r; this will be denoted as $(i_{m-1}, \cdots, i_1, i_0)$.

Given $i, j, 0 \le i, j \le (n - 1)$, the following defines certain relationships denoted as g and h.

Here, it is assumed that $i = (i_{m-1}, \cdots, i_1, i_0)$ and $j = (j_{m-1}, \cdots, j_1, j_0)$ in radix-r.

Let $j = g(i)$, if $i_p = j_{p+1}$ for all p, $0 \le p \le (m - 2)$ and $i_{m-1} = j_0$. Thus, j is an end-around shift of i.

Let $j = h(i)$, if $i_p = j_p$ for all p, $1 \le p \le (m - 1)$.

(For $r = 2$ the mappings g and h correspond to shuttle and

TABLE II
CERTAIN EQUIVALENCES BETWEEN $BA(G)$ NETWORKS AND $LA(G)$ NETWORKS

$LA(G)$ link network	Equivalence in $BA(G)$ bus network
Number of processors	Number of buses
Number of links	Number of processors
Number of connections/processors	Number of connections/bus
Interprocessor transfer	Interbus transfer
Number of processor-processor transfers in a message path	Number of bus-bus transfers in a message path
Processor fault	Bus fault
Link fault	Processor fault

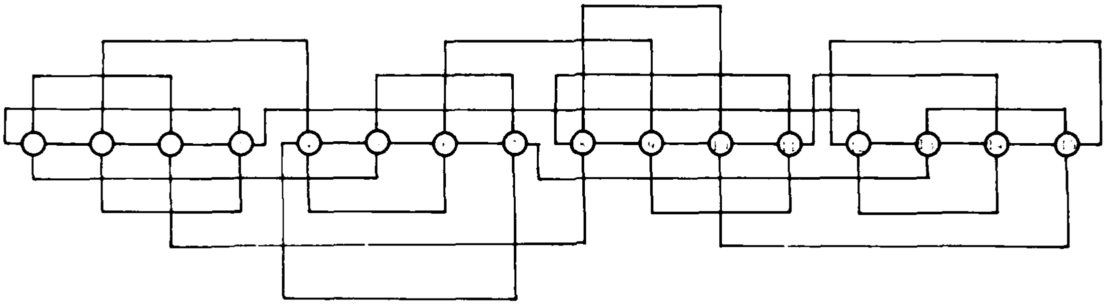


Fig. 2. $FG(4,2)$ network.

exchange mappings, respectively [8], [9], [18]).

Thus, $i = h(j)$ implies $j = h(i)$, i and j differ only in the last digit.

These graphs are constructed using a two-step approach. First, a skeletal graph SG is constructed, which is then augmented to obtain the fault-tolerant graph FG.

The skeletal graph denoted as $SG(r,m)$ is obtained by connecting every pair of nodes i and j that satisfy the relationship $i = g(j)$ or $i = h(j)$.

In the following, let $k = (r^m - 1)/(r - 1)$. Thus, $k = (1, 1, \cdots, 1)$ in radix-r.

$FG(r,m)$ Design, $m = 2$:

r = even: Construct an $SG(r,2)$ graph for the specific r; then augment the graph by adding $r/2$ links, defined as $(0,k)$, $(2k,3k)$, $\cdots$, $((r-2)k, (r-1)k)$.

Fig. 2 illustrates an $FG(4,2)$ network.

r = odd: Construct an $SG(r,2)$ graph for the specific r; then augment it by adding an extra node n which is then connected to nodes $0, k, 2k, \cdots, (r-1)k$ by adding r additional links: $(n,0), (n,k), \cdots, (n,(r-1)k)$.[1]

(The resulting graph when r = odd has $(n+1)$ nodes. This extra node can be used as a spare and as shown later, is useful for routing when faults occur in the system.)

$FG(r,m)$ Design, $m \geq 3$:

$$\text{Let } q = \begin{cases} (r^m - 1)/(r^2 - 1) & \text{for } m = \text{even,} \\ (r^{m+1} - 1)/(r^2 - 1) & \text{for } m = \text{odd.} \end{cases}$$

Thus, in radix r,

$$q = \begin{matrix} (0,1,0,1,\cdots,0,1) & \text{for } m = \text{even} \\ (1,0,1,0,\cdots,1,0,1) & \text{for } m = \text{odd.} \end{matrix}$$

$r = 2$: Construct an $SG(2,m)$ graph for the given m, then add links: $(0,k)$, $(k, k-q)$, and $(q,0)$. For odd m add an additional link $(k-q, q)$.

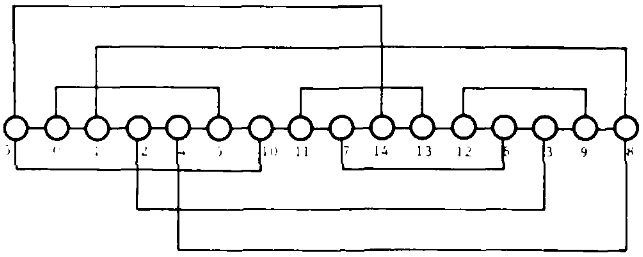


Fig. 3. $FG(2,4)$ network.

$FG(2,4)$ graph is illustrated in Fig. 3.

$r \geqq 3$: Construct an $SG(r,m)$ graph for the given r, m. Add r links, as defined below: $(0,k)$, $(k,2k)$, $\cdots$, $((r-2)k, (r-1)k)$, $((r-1)k, 0)$. If m = even, add additional $(r^2 - r)/2$ links defined by the following expression for all a, b, $a \neq b$ and $0 \leq a, b \leq (r-1)$:

$$(arq + bq, (r-1-a)rq + (r-1-b)q).$$

Fig. 4 illustrates $FG(3,3)$ graph.

The following basic properties of $FG(r,m)$ networks can be easily proven.

Theorem 1: $FG(r,m)$ is a regular network of degree r if $m = 2$.

Theorem 2: $FG(r,m)$ is a regular network of degree $(r+1)$ if $r \geq 3$ and $m \geq 3$.

Theorem 3: $FG(2,m)$, $m \geq 3$, m = even are regular networks of degree 3. For m = odd, all nodes are of degree 3 except nodes q and $(k-q)$, which have degree 5.

Thus, the $LA(FG)$ and $BA(FG)$ networks derived from FG will have the following characteristics. Each PE in $LA(FG)$ will have the same number of links connected to it—either r or $(r+1)$. Analogously, each bus in $BA(FG)$ will be connected to the same number of PE's—either r or $(r+1)$. Thus, from the point of view of I/O ports and inter-

[1] It may be noted that there cannot exist any degree r regular graph of exactly r^2 nodes since r = odd. Therefore, one must add an additional node.

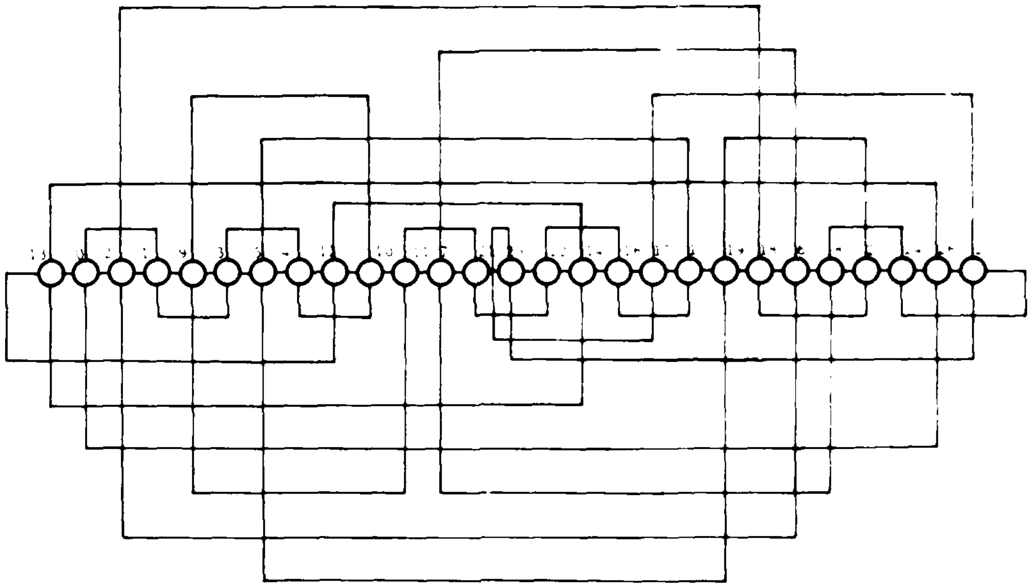


Fig. 4. $FG(3, 3)$ network

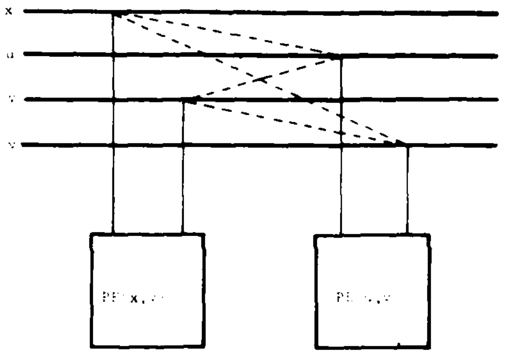


Fig. 5. Routing on bus architecture

connections, these networks can be considered regular. The similarities and differences are noted that exist between the link network and the bus network, are described in Table II.

The following shows that a path always exists of length, at most $(2m - 1)$, given any pair of nodes in $FG(r, m)$ networks.

First, it may be noted that in the link architecture $LA(FG)$, transmitting data from one PE to another PE may require one or more hops through other PE's in the path. Similarly, in the bus architectures $BA(FG)$, direct transfer is possible only when both of the processing elements are connected to the same bus; i.e., PE(x, y) can transfer to PE(u, v) if $x = u$ or v or if $y = u$ or v. In other cases, a transfer would require one or more interbus transfer through the connecting PE's. However, the difference between the bus architecture and the link architecture, insofar as formulating a routing path between a source and a destination PE is concerned, is that in the bus architecture various choices exist depending on the various combinations of the source and destination buses. For example, given PE(x, y) as the source and PE(u, v) as the destination, PE(x, y) can initiate the transmission on bus x or y, and PE(u, v) can receive it from bus u or v as illustrated in Fig. 5.

However, topologically, there are certain equivalences between paths in $LA(FG)$ and $BA(FG)$. Given a PE-to-PE path in the link architecture, there is an equivalent BUS-to-BUS path in the bus architecture. If an inter-PE transfer constitutes a single hop, both of these paths will have the same number of hops. On the other hand, given a PE-to-PE path in $BA(FG)$, there is an equivalent link to the link path in $LA(FG)$. Below, the routing in these networks is described in the context of the link network $LA(FG)$. The formulation below can be adapted for the bus architecture $BA(FG)$ as well.

Consider the following path from the source PE(s) to the destination PE(d). Let $s = (s_{m-1}, \cdots, s_1, s_0)$ and $d = (d_{m-1}, \cdots, d_1, d_0)$ in radix-r.

$$s = (s_{m-1}, \cdots, s_1, s_0)$$

$$(s_{m-2}, \cdots, s_1, s_0, s_{m-1})$$

$$(s_{m-2}, \cdots, s_1, s_0, d_{m-1})$$

$$(s_{m-3}, \cdots, s_0, d_{m-1}, s_{m-2})$$

$$(s_0, d_{m-1}, d_{m-2}, \cdots, d_1)$$

$$(d_{m-1}, \cdots, d_1, s_0)$$

$$d = (d_{m-1}, \cdots, d_1, d_0)$$

The above path will be herewith denoted as $\boldsymbol{pt}(s, \boldsymbol{d})$. This is used below to formulate a simple message routing procedure that routes the message from node to node using only the destination address information.

A message routing algorithm suitable for $LA(FG(r, m))$ is described here. It is assumed that each message carries m tag bits. These m bits denoted as T are initialized at the source PE, equal to d the destination address as shown in Fig. 6(a) and (b). The destination address d is also carried by the message separately.

Although the routing algorithms given below are for $LA(FG)$ networks, they can be used for the bus network $BA(FG)$ as well with some modifications. Each PE(x) in $LA(FG)$ corresponds to a node x in FG and vice versa. Also, every path from PE(x) to PE(y) in $LA(FG)$ has a corresponding path from node x to y in FG and vice versa. Therefore

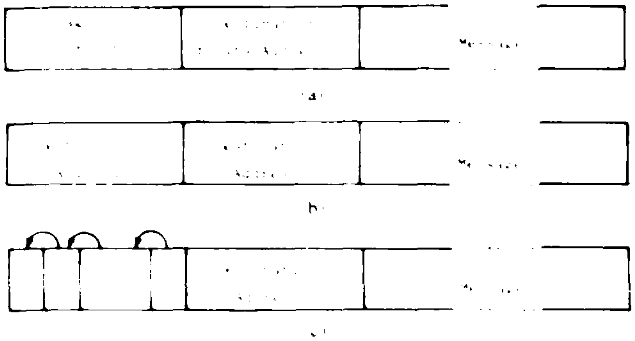


Fig. 6. (a) Message format. (b) Message at source node. (c) Shifting the tagbit at node x.

all the paths and routing actions for $LA(FG)$ will be described by using the graph FG. The remainder of the section will use node x to denote PE(x).

When a message arrives at node x, the following steps are executed to determine the next node in the path. Here, x_0 denotes the least significant bit of the binary number x.

Step 1: If $x = d$, then the message has reached the destination and is accepted. Otherwise, the message is forwarded to a neighbor of x by using the following steps.

Step 2: Compare x_0 (the least significant bit of x) to the leading bit of T. If they are equal, then go to Step 3, else forward the message to the neighbor of x, given as [illegible] in binary.

Step 3: Shift the tag field T left by one bit as shown in Fig. 6(c). Now if [illegible] then go to Step 2, else forward the message to the neighbor y of x, given as [illegible] in binary.

The following example illustrates the above routing steps further.

Example: Consider the 16-node $FG(2, 4)$ network. Let $s = 8$ and $d = 11$. The following table describes the path and the corresponding tag bits at different nodes.

Node	8	9	3	2	4	5	10	11
(in binary)	1000	1001	0011	0010	0100	0101	1010	1011
T at arrival		1011	0110	0110	1100	1100	1000	1000
T at departure	1011	0110	0110	1100	1100	1000	1000	

It can be easily seen that the above steps route the message along the path $pt(s, d)$ described earlier. The routing [illegible] algorithm and uses only local [illegible] the message.

III. [illegible] $FG(2, m)$ NETWORKS

[illegible] various [illegible] $FG(2, m)$ net[illegible] FG [illegible] networks for any [illegible] $FG(2, m)$ networks are de[illegible] single and [illegible] Section IV considers FG [illegible] based on [illegible] routing [illegible] trans[illegible] strategy [illegible] separately to [illegible]

In considering the fault tolerance of $LA(FG)$ and $BA(FG)$, the following may be noted regarding the effects of various faults from the point of view of communication and routing, as described in Table II. The effect of a faulty PE in the link architecture is equivalent to the effect of a faulty BUS in the corresponding bus architecture. Similarly, a faulty PE in the bus architecture has an equivalent effect on the routing, as a faulty link in the link architecture. The following discusses fault tolerance in the context of link architecture and this can fairly easily be extended to bus architecture.

Primarily, node failures are considered here since the effect of a link failure can be no worse than a node failure. (The paths affected by a link failure are a subset of the paths affected by the failure of one of the nodes connected by the link. Therefore, the routing and detour techniques can be adapted to link failures easily.)

The fault tolerance of a regular network of degree b can be, at most, equal to $(b - 1)$. This is because any node can always be disconnected from the network by removing the b nodes (links) that are connected to it. Accordingly, we will call a regular network an *optimal* fault-tolerant network if its fault tolerance is equal to $(b - 1)$, the maximum possible. Later, $FG(2, m)$ networks are shown to be optimally fault tolerant (m = even).

The case of single faults is considered separately first. An algorithm is developed below that routes messages from one node to another, in spite of any single faulty node or link.

Given a binary number x, let the weight of x, $(wt(x))$, represent the number of 1's in x.

Lemma 1: Any node x in the path $pt(t, 0)$ satisfies the relationship given below.

For $x \neq t$,

$$wt(x) \le wt(t), \quad \text{if } t_0 = 0$$

$$wt(x) < wt(t), \quad \text{if } t_0 = 1.$$

Proof: Consider any two consecutive nodes u and v in the path $pt(t, 0)$. Let $u = (u_{m-1}, \cdots, u_1, u_0)$ and $v = (v_{m-1}, \cdots, v_1, v_0)$ in binary, and let u precede v in the path. The following relationship between u and v must be satisfied: v is obtained from u by an end-around shift of u if the least-significant bit of u is 0, or by complementing the least-significant bit of u, if the least significant bit of u is 1. Thus, the number of 1's in v cannot exceed the number of 1's in u. Furthermore, if $u_0 = 1$, then the number of 1's in v is less than the number of 1's in u.

Hence, the Lemma. Q.E.D.

Lemma 2: Any node x in the path $pt(t, n - 1)$ satisfies the following.

For $x \neq t$,

$$wt(x) \ge wt(t), \quad \text{if } t_0 = 1$$

$$wt(x) > wt(t), \quad \text{if } t_0 = 0.$$

Proof: Proof is similar to Lemma 1. Q.E.D.

The proofs of the following Lemmas are also similar.

Lemma 3: Any node x in the path $pt(0, t)$ also satisfies Lemma 1.

Lemma 4: Any node x in the path $pt(n - 1, t)$ also satisfies Lemma 2.

The algorithm discussed below is based on the above observations and is useful for routing messages when a node becomes faulty. This also establishes an upper bound on the maximum path length, in the presence of a fault. The algorithm given is shown to be easily implementable.

Let f denote the faulty PE, PE(f).

Let s denote the source PE, PE(s).

Let d denote the destination PE, PE(d).

Routing with a Faulty PE: The following describes different paths from s to d in FG which correspond to different cases of f.

a) $wt(f) > wt(s)$ and $wt(f) > wt(d)$.

$$s \rightarrow \cdots \boldsymbol{pt}(s, 0) \cdots \rightarrow 0 \rightarrow \cdots \boldsymbol{pt}(0, d) \cdots \rightarrow d.$$

b) $wt(f) < wt(s)$ and $wt(f) < wt(d)$.

$$s \rightarrow \cdots \boldsymbol{pt}(s, n-1) \cdots \rightarrow (n-1) \rightarrow \cdots \boldsymbol{pt}(n-1, d) \cdots \rightarrow d.$$

c) $wt(f) > wt(s)$ and $wt(f) < wt(d)$.

$$s \rightarrow \cdots \boldsymbol{pt}(s, 0) \cdots \rightarrow 0 \rightarrow (n-1) \rightarrow \cdots \boldsymbol{pt}(n-1, d) \cdots \rightarrow d.$$

d) $wt(f) < wt(d)$ and $wt(f) > wt(d)$.

$$s \rightarrow \cdots \boldsymbol{pt}(s, n-1) \cdots \rightarrow (n-1) \rightarrow 0 \rightarrow \cdots \boldsymbol{pt}(0, d) \cdots \rightarrow d.$$

e) $wt(f) = wt(s)$ and $wt(f) > wt(d)$.

$$\text{If } s_0 = \begin{cases} 1 & \text{path as } a) \\ 0 & \text{path as } d). \end{cases}$$

f) $w(f) = wt(s)$ and $wt(f) < wt(d)$.

$$\text{If } s_0 = \begin{cases} 1 & \text{path as } c) \\ 0 & \text{path as } b). \end{cases}$$

g) $wt(f) = wt(d)$ and $wt(f) > wt(s)$.

$$\text{If } d_0 = \begin{cases} 1 & \text{path as } a) \\ 0 & \text{path as } c). \end{cases}$$

h) $wt(f) = wt(d)$ and $wt(f) < wt(s)$.

$$\text{If } d_0 = \begin{cases} 1 & \text{path as } d) \\ 0 & \text{path as } b). \end{cases}$$

i) $wt(f) = wt(s) = wt(d)$.

$$\text{If } \begin{cases} s_0 = 0, & d_0 = 0 \quad \text{path as } b) \\ s_0 = 0, & d_0 = 1 \quad \text{path as } d) \\ s_0 = 1, & d_0 = 0 \quad \text{path as } c) \\ s_0 = 1, & d_0 = 1 \quad \text{path as } a). \end{cases}$$

The following Theorem is an immediate consequence of the above routing steps.

Theorem 4: In the presence of any single fault, a message can be routed from any node to any other node using at most $(4m - 1)$ hops.

The earlier described routing procedure is easily adaptable when the network becomes faulty. As an example, assume node 0 is the faulty node. This will correspond to case b), which describes the following path from s to d: $s \rightarrow \cdots \boldsymbol{pt}(s, n-1) \cdots \rightarrow (n-1) \rightarrow \cdots \boldsymbol{pt}(n-1, d) \cdots \rightarrow d$.

This can be implemented as follows. The source s may initialize the tagfield T equal to $(n-1)$. This will route the message to the intermediate destination node $(n-1)$. The node $(n-1)$, upon recognizing that the destination address d is not equal to $(n-1)$, will replace T by d and then forward the message to the final destination d.

Next, we consider the case of double faults in $FG(2, m)$ networks $m \geq 3$. In Appendix A, a technique is exhibited that can route a message from a source node s to a destination node d, despite *two* faulty nodes. The following theorem is a direct consequence of this.

Theorem 5: In the presence of any two faults, a message can be routed from any node s to any node d in $FG(2, m)$, using at most $(4m + 2)$ hops.

Thus, a second fault may cause a small increase in the path length; $(4m + 2)$ versus $(4m - 1)$.

IV. Fault Tolerance of $FG(r, m)$ Networks

This section considers $FG(r, m)$ networks for all $r, m \geq 3$. Techniques are first formulated that construct detours around the faulty nodes. These detour techniques are applicable when the number of faults does not exceed $(r - 1)/2$. Following this, it is shown that these networks in general can also tolerate a much larger number of faults by showing how to construct paths from s to d when the number of faults is equal to $(r - 1)$.

The detour techniques shown below possess certain attractive features. These detours perform local alterations of paths which connect the two nodes that are adjacent to the faulty node(s). Hence, the faults can be made transparent to the global routing strategy. Other important aspects of these detours include the fact that they are of constant length, independent of m, thus, the size of the network. Furthermore, increases in path length that are due to the use of detours become directly proportional to the number of faults; thus, graceful degradation is made possible.

Definition: Let $F = \{f^1, f^2, \cdots, f^t\}$ represent the set of t faulty nodes.

Definition: Let the m-tuple $(f^i_{m-1}, f^i_{m-2}, \cdots, f^i_1, f^i_0)$ represent the faulty node f^i in radix-r.

Definition: Let e be a radix-r digit that does not belong to the set

$$\{f^1_{m-1}, f^2_{m-1}, \cdots, f^t_{m-1}, f^1_0, f^2_0, \cdots, f^t_0\}.$$

Thus, e is a digit that does *not* appear either in the least- or in the most-significant position of any of the faulty nodes. Since $2t < r$, there always exists such an e. Thus, any node that has e in the least or in the most significant position cannot be faulty. This will be quite useful later.

Here, in constructing these detours, it will be assumed that u, v, and w are three consecutive nodes in the path from s to d. The node v is assumed to be faulty with u and w being fault free. The detour therefore connects u with w without passing through any of the faulty nodes in F, including v. (The case when two or more consecutive nodes are faulty can be treated by successive applications of the given techniques.)

First, it may be seen that there are, altogether, three possible distinct relationships between u, v, and w in the type of path discussed earlier in the section. (This follows from the observation that if v is an h-neighbor of u, then w cannot also be an h-neighbor of v.) These relationships are described in the following:

$$R1: \; v = h(u) \quad \text{and} \quad w = g(v)$$

$$R2: \; v = g(u) \quad \text{and} \quad w = h(v)$$

$$R3: \; v = g(u) \quad \text{and} \quad w = g(v).$$

The following constructs detours that correspond to the above three cases. The detailed construction is described in Appendix B.

Case R1: Here, the original path contains the following sequence of nodes represented in radix-r as

$$u = (u_{m-1}, \cdots\cdots, u_1, u_0)$$

$$\text{faulty:} \quad v = (u_{m-1}, \cdots\cdots, u_1, c), \qquad c \neq u_0$$

$$w = (u_{m-2}, \cdots, u_1, c, u_{m-1}).$$

The following describes construction of an alternate path from u to w that does not pass through any of the nodes in fault set F (which includes the faulty node v):

$$u = (u_{m-1}, \cdots\cdots, u_1, u_0)$$

$$(u_{m-2}, \cdots, u_1, x, u_{m-1}) \qquad x \text{ as determined per Appendix B}$$

$$(u_{m-2}, \cdots\cdots, u_1, x, e)$$

$$(e, u_{m-2}, \cdots\cdots, u_1, x) \qquad e \text{ as defined earlier}$$

$$(e, u_{m-2}, \cdots\cdots, u_1, c)$$

$$(u_{m-2}, \cdots\cdots, u_1, c, e)$$

$$w = (u_{m-2}, \cdots, u_1, c, u_{m-1}).$$

The above detour is of length 7, and hence will result in a net increase of 5 in the path length.

Case R2: $u \rightarrow v = g(u) \rightarrow w = h(v)$. This corresponds to the following path segment:

$$u = (u_{m-1}, \cdots\cdots, u_1, u_0)$$

$$\text{faulty:} \quad v = (u_{m-2}, \cdots, u_1, u_0, u_{m-1})$$

$$w = (u_{m-2}, \cdots\cdots, u_1, u_0, c) \qquad c \neq u_{m-1}.$$

The following constructs an alternate path from u to w.

$$u = (u_{m-1}, \cdots\cdots, u_1, u_0)$$

$$(u_{m-1}, \cdots\cdots, u_1, x) \qquad x \text{ determined as per Appendix B}$$

$$(u_{m-2}, \cdots, u_1, x, u_{m-1})$$

$$(u_{m-2}, \cdots\cdots, u_1, x, e) \qquad e \text{ defined earlier}$$

$$(e, u_{m-2}, \cdots\cdots, u_1, x)$$

$$(e, u_{m-2}, \cdots\cdots, u_1, u_0)$$

$$(u_{m-2}, \cdots\cdots, u_1, u_0, e)$$

$$w = (u_{m-2}, \cdots\cdots, u_1, u_0, c).$$

TABLE III
FAULT-TOLERANT PROPERTIES OF $FG(r, m)$ NETWORKS

r	m	Fault tolerance t	Routing Distance with t faults
2	m	1	$(4m - 1)$
		2	$(4m + 2)$
$r > 3$	2	$(r - 1)$	12
$r \geqq 3$	$m \geqq 3$	$t \leqq (r - 1)/2$	$(2m - 1 + 11t)$
		$t > (r - 1)/2$	$(6m - 3)$

As in Case R1, the detour shown above is of length 7, and thus the path length will increase by at most 5.

Case R3: $u \rightarrow v = g(u) \rightarrow w = g(v)$. This corresponds to the following segment:

$$u = (u_{m-1}, \cdots\cdots\cdots, u_1, u_0)$$

$$\text{faulty:} \quad v = (u_{m-2}, \cdots\cdots, u_1, u_0, u_{m-1})$$

$$w = (u_{m-3}, \cdots, u_0, u_{m-1}, u_{m-2}).$$

One can construct a detour from u to w, as illustrated in the following:

$$u = (u_{m-1}, \cdots\cdots\cdots, u_1, u_0)$$

$$(u_{m-1}, \cdots\cdots\cdots, u_1, x) \qquad x \text{ and } y \text{ determined as per Appendix B}$$

$$(u_{m-2}, \cdots\cdots, u_1, x, u_{m-1})$$

$$(u_{m-2}, \cdots\cdots\cdots, u_1, x, y)$$

$$(u_{m-3}, \cdots, u_1, x, y, u_{m-1})$$

$$(u_{m-3}, \cdots\cdots, u_1, x, y, e) \qquad e \text{ defined earlier}$$

$$(e, u_{m-3}, \cdots\cdots, u_1, x, y)$$

$$(e, u_{m-3}, \cdots\cdots, u_1, x, e)$$

$$(e, e, u_{m-3}, \cdots\cdots, u_1, x)$$

$$(e, e, u_{m-3}, \cdots\cdots, u_1, u_0)$$

$$(e, u_{m-3}, \cdots\cdots\cdots, u_0, e)$$

$$(e, u_{m-3}, \cdots\cdots, u_0, u_{m-1})$$

$$(u_{m-3}, \cdots\cdots, u_0, u_{m-1}, e)$$

$$w = (u_{m-3}, \cdots, u_0, u_{m-1}, u_{m-2}).$$

The above detour is of length 13; thus, it will result in a net increase of 11 in the path length.

The above corresponds to the worst case increase in the path length. It may be noted that when two or more consecutive nodes are faulty, the above techniques can be applied iteratively, to construct a detour around the faulty nodes. As an example, consider the following path segment:

$$u \rightarrow v = h(u) \rightarrow w = g(v) \rightarrow z = g(w).$$

Assume here that both the nodes v and w are faulty. Thus, one needs to construct a detour from u to z. This can be constructed in two steps: first, by using Case R1, one can construct a detour of length 6 from u to $(u_{m-2}, \cdots, u_1, c, e)$.

Next, by applying Case R2, one can construct a detour of length 11 from $(u_{m-2}, \cdots, u_1, c, e)$ to $z = (u_{m-4}, \cdots, u_1, c, u_{m-1}, u_{m-2}, u_{m-3})$. Thus, the total detour length will be 17 for both the faulty nodes together. The following theorem is a direct consequence of these above discussions.

Theorem 6: A message can be routed from one node to another using at most $(2m - 1 + 11t)$ hops with t faults where $t \leq (r - 1)/2$.

The following exhibits the complete fault tolerance of $FG(r, m)$ networks for all $r, m, r \geq 3\ m \geq 2$. It is shown that in spite of any $(r - 1)$ faults, the network remains fully connected and the messages can still be routed easily from node to node.

Definition: Let N denote the set of all n nodes in $FG(r, m)$.

Definition: Let R_k denote the set of r nodes, $0, k, 2k, 3k, \cdots, (r-1)k$.

Definition: Let $N - R_k$ denote the set of $(n - r)$ nodes consisting of all nodes other than those appearing in the set R_k.

Fault Tolerance of $FG(r, m)$, $m = 2$, Networks: Since $m = 2$, any node i in the network can be represented in radix-r as (i_1, i_0). Now, consider the following r paths from s to the nodes in R_k:

$$\begin{aligned}
&s \text{ to } 0: && s = (s_1, s_0) \rightarrow (s_1, 0) \rightarrow (0, s_1) \rightarrow (0, 0) = 0\\
&s \text{ to } k: && s = (s_1, s_0) \rightarrow (s_1, 1) \rightarrow (1, s_1) \rightarrow (1, 1) = k\\
&s \text{ to } 2k: && s = (s_1, s_0) \rightarrow (s_1, 2) \rightarrow (2, s_1) \rightarrow (2, 2) = 2k\\
& && \vdots\\
&s \text{ to } (r-1)k: && s = (s_1, s_0) \rightarrow (s_1, r-1)\\
& && \quad \rightarrow (r-1, s_1) \rightarrow (r-1, r-1)\\
& && \quad = (r-1)k.
\end{aligned}$$

These above paths can be seen to be disjoint from the following observation.

Assume that there exists a node in the path from s to ik that also belongs to the path from s to jk where $i \neq j$. This would imply either that $(s_1, i) = (j, s_1)$, or that $(s_1, j) = (i, s_1)$. So, one has $s_1 = i = j$, a contradiction.

It may therefore be deduced that there always exists a node xk for which the path s to xk is fault free, in spite of any $(r - 1)$ faults.

Similarly, it can also be asserted that there always exists a node yk for which the path yk to d is fault free, in spite of any $(r - 1)$ faults. So, in order to establish that the fault tolerance of the network is $(r - 1)$, it will be sufficient to show that there always exist r disjoint paths between any xk and yk. The following shows techniques to construct such r paths from xk to yk.

(The paths are shown here using radix-r representation. Thus, $xk = (x, x)$ and $yk = (y, y)$ in radix-r.)

First, consider the following set of $(r - 1)$ paths denoted as path 1–path $(r - 1)$.

path-1: $(x, x) \rightarrow (x, y) \rightarrow (y, x) \rightarrow (y, y)$.

path-2–path $(r - 1)$: Let $w = 0, 1, \cdots, (r - 1)$ and $w \neq x, y$. There are exactly $(r - 2)$ distinct values of w and these define the following $(r - 2)$ paths:

$$(x, x) \rightarrow (x, w) \rightarrow (w, x) \rightarrow (w, y) \rightarrow (y, w) \rightarrow (y, y).$$

These $(r - 1)$ paths shown above are all disjoint. Paths 2 through $(r - 2)$ are disjoint because of the value of w, distinct for each path.

Path 1 is disjoint with paths 2 through $(r - 1)$ since $w \neq x$ and $w \neq y$.

Thus, it remains to be shown that there exists one additional path from xk to yk that is disjoint from the above $(r - 1)$ paths. The following constructs such a path, denoted as path-r for r = even and r = odd, separately.

r = even: First, it may be noted that as per the construction procedure, nodes xk and yk are directly connected to some nodes uk and vk (in R_k), respectively.

i) Let $u = y$ (thus, $v = x$). Hence, xk and yk are connected by the link (xk, yk). *Path r:* $xk \rightarrow yk$.

ii) Let $u \neq y$ (thus, $v \neq x$). Hence, xk and yk are not connected. *Path r:* $(x, x) \rightarrow (u, u) \rightarrow (u, v) \rightarrow (v, v) \rightarrow (y, y)$.

This above path is disjoint from the paths 1 through $(r - 1)$ shown above since $x \neq u$ and $y \neq v$.

r = odd: path r: $xk \rightarrow n \rightarrow yk$.

This path passes through node n which has not been used in any of the above paths.

Since $FG(r, 2)$ networks are of degree r, the following theorem is a direct consequence of the above discussions.

Theorem 7: $FG(r, 2)$ networks are optimally fault tolerant, and failure of any $(r - 1)$ components is tolerated.

Example 2: Consider the network shown in Fig. 2. Here $r = 4$. Hence, the network is 3-fault-tolerant. Let the nodes 0, 5, and 7 be faulty.

Given these nodes as faulty, one can construct the following path from any s to d, which will be fault free:

$$\begin{aligned}
(s_1, s_0) \rightarrow (s_1, 2) \rightarrow (2, s_1) \rightarrow (2, 2)&\\
\rightarrow (2, d_1) \rightarrow (d_1, 2) \rightarrow (d_1, d_0)&.
\end{aligned}$$

Thus, given $s = 4$ and $d = 15$, one has the following path:

$$4 \rightarrow 6 \rightarrow 9 \rightarrow 10 \rightarrow 11 \rightarrow 14 \rightarrow 15.$$

Fault Tolerance of $FG(r, m)$ Networks, $m \geq 3$: The following definitions and lemmas will be useful in constructing the paths.

Definition: Let $wt_i(x)$ represent the number of occurrences of i in radix-r representation of x.

For example, if $x = (0, 1, 1, 2, 1)$ in radix-r, $wt_0(x) = 1$, $wt_1(x) = 3$, $wt_2(x) = 1$, and $wt_3(x) = 0$.

Lemma 3: If $wt_i(x) \neq wt_i(y)$, for *any* i, $0 \leq i \leq (r - 1)$, then $x \neq yy$.

Let $\boldsymbol{pt}(i, j)$ represent the path from i to j, as defined in Section II.

The path $\boldsymbol{pt}(i, j)$ denotes a path of length at most $(2m - 1)$.

Lemma 4: Given any node x, $x \neq s$, in $\boldsymbol{pt}(s, ik)$, $ik \in R_k$, the following relationship holds:

$$\begin{aligned}
&wt_i(x) > wt_i(s) && \text{if } s_0 \neq i\\
&wt_i(x) \geq wt_i(s) && \text{if } s_0 = i\\
&wt_j(x) \leq wt_j(s) && \text{if } j \neq i.
\end{aligned}$$

Lemma 5: Two paths $\boldsymbol{pt}(s, ik)$ and $\boldsymbol{pt}(s, jk)$ are node disjoint if $i \neq j$.

Proof: Let $x \in \boldsymbol{pt}(s, ik)$ and $y \in \boldsymbol{pt}(s, jk)$ denote any two nodes in paths $\boldsymbol{pt}(s, ik)$ and $\boldsymbol{pt}(s, jk)$, respectively.

Case I: Let $s_0 = i$.

Thus, one has, from Lemma 4, $wt_j(y) > wt_i(s)$, but $wt_j(x) \leq wt_j(s)$. Hence, $x \neq y$.

Case II: Let $s_0 = j$.

Similar to Case I.

Case III: Let $s_0 \neq i$ and $s_0 \neq j$. Here, $wt_i(x) > wt_i(s)$, and $wt_i(y) < wt_i(s)$. Thus, $wt_i(x) \neq wt_i(y)$, and hence $x \neq y$. Q.E.D.

The following Lemmas are direct consequences of the above Lemma.

Lemma 6: Given any $(r - 1)$ faulty nodes, there exists at least one node xk where $xk \; \epsilon \; R_k$, such that the path $pt(s, xk)$ is fault free.

Lemma 7: Given any $(r - 1)$ faulty nodes, there exists at least one node yk where $uk \; \epsilon \; R_k$, such that the path $pt(yk, d)$ is fault free.

The following shows it is possible to travel from any s to any d, in spite of faults in any $(r - 1)$ nodes. This is illustrated by considering the following cases separately.

Case I: All of the faulty nodes belong to the subset R_k.

The subset R_k contains r nodes. Since the number of faulty nodes does not exceed $(r - 1)$, there must be at least one node xk which is fault free.

Consider the path $\boldsymbol{pt(s, xk)}$. This entire path must be fault free since all of the intermediate nodes in the path pass through nodes in $N - R_k$.

Similarly, the path $\boldsymbol{pt(xk, d)}$ must also be fault free. Hence, the following path from s to d must be fault free:

$$s \rightarrow \cdots \boldsymbol{pt}(s, xk) \cdots \rightarrow xk \rightarrow \cdots \boldsymbol{pt}(x, kd) \cdots \rightarrow d\,.$$

The length of the above path is, at most, equal to $(2m - 1) + (2m - 1) = (4m - 2)$.

Case II: All of the nodes in the subset R_k are fault free.

In this case, the faulty nodes are thus confined to the subset $N - R_k$.

As per Lemmas 4 and 5, one always has two fault-free paths of the type $\boldsymbol{pt}(s, xk)$ and $\boldsymbol{pt}(yk, d)$.

The following constructs a fault-free path from s to d.

If $x = y$, then follow the path given in Case I; else, if

$$x > y, \quad \text{and} \quad x - y \leq r/2 \quad \text{or}$$

$$x < y, \quad \text{and} \quad y - x > r/2 \quad \text{then:}$$

follow the path given below in (1); otherwise, follow the path given in (2)

$$s \rightarrow \cdots \boldsymbol{pt}(s, xk) \cdots \rightarrow xk \rightarrow (x - 1)k \rightarrow \cdots$$
$$\rightarrow (y + 1)k \rightarrow yk \rightarrow \cdots \boldsymbol{pt}(yk, d) \cdots \rightarrow d \qquad (1)$$

$$s \rightarrow \cdots \boldsymbol{pt}(s, xk) \cdots \rightarrow xk \rightarrow (x + 1)k \rightarrow \cdots$$
$$\rightarrow (y - 1)k \rightarrow yk \rightarrow \cdots \boldsymbol{pt}(yk, d) \cdots \rightarrow d\,. \qquad (2)$$

Here the maximum path length is equal to

$$(4m - 2) + r/2\,.$$

Case III: Nodes in both R_k as well as in $N - R_k$ are faulty.

As before, there always exist fault-free paths $pt(s, xk)$ and $pt(yk, d)$.

The following shows that there exist *at least r* disjoint paths between any xk and yk. Consider the following paths:

path-1: $xk \rightarrow (x + 1)k \rightarrow \cdots \rightarrow (y - 1)k \rightarrow yk$

path-2: $xk \rightarrow (x - 1)k \rightarrow \cdots \rightarrow (y + 1)k \rightarrow yk$

path-3 through r: for all w, $0 \leq w, \leq (r - 1)$ and $w \neq x$ and $w \neq y$,

consider the following $(r \cdot 2)$ paths:

$$
\begin{aligned}
xk &= (x, \cdots\cdots\cdots, x, x)\\
& (x, \cdots\cdots\cdots, x, w)\\
& (x, \cdots\cdots, x, w, x)\\
& (x, \cdots\cdots, x, w, y)\\
& (x, \cdots, x, w, y, x)\\
& (x, \cdots, x, w, y, y)\\
& (w, y, y, \cdots\cdots, y)\\
& (y, y, \cdots\cdots, y, w)\\
yk &= (y, y, \cdots\cdots\cdots, y)\,.
\end{aligned}
$$

Since $w \neq x$ and $w \neq y$, there are exactly $(r - 2)$ distinct values of w. Each one of these distinct values defines a path and in this set all of the $(r - 2)$ paths are disjoint since the value of w for each path is different. Now it may be seen all of the r paths shown above are disjoint. The intermediate nodes in path-1 and path-2 have only one digit in their representation, whereas the intermediate nodes paths 3–r have at least two digits in their representation. Therefore, there cannot be any intermediate nodes that are common to any two of the above r paths.

The maximum path length here is given as

$$(2m - 1) + (2m - 1) + (2m - 1) = (6m - 3)\,.$$

The following theorem is based on the above observation.

Theorem 8: Given any $FG(r, m)$ network, there exists a path from any node to any other node in spite of $(r - 1)$ faults of length at most $(6m - 3)$ and this path can be constructed algorithmically.

Example 3: Consider the $FG(4, 3)$ network which has 64 nodes and is of degree 5.

Let $s = 5$ and $d = 22$.

Here, $R_k = \{0, 21, 42, 63\}$. The following illustrates Case I and Case III.

Case I: Let nodes 0, 21, and 42 be faulty. Using the technique shown in Case I, one constructs the following path:

$$5 \rightarrow \cdots \boldsymbol{pt}(5, 63) \cdots \rightarrow 63 \rightarrow \cdots \boldsymbol{pt}(63, 22) \cdots \rightarrow 22$$

which is equal to

$$5 \rightarrow 7 \rightarrow 28 \rightarrow 31 \rightarrow 61 \rightarrow 63 \rightarrow 61 \rightarrow 55 \rightarrow 23 \rightarrow 22\,.$$

Case III: Let nodes 0, 21, and 23 be faulty. Using Case III, one has

$$5 \rightarrow \cdots \boldsymbol{pt}(5, 63) \cdots \rightarrow 63 \rightarrow 42$$
$$\rightarrow \cdots \boldsymbol{pt}(42, 22) \cdots \rightarrow 22$$

which is equal to

$$5 \rightarrow 7 \rightarrow 28 \rightarrow 31 \rightarrow 61 \rightarrow 63$$
$$\rightarrow 42 \rightarrow 41 \rightarrow 38 \rightarrow 37 \rightarrow 22\,.$$

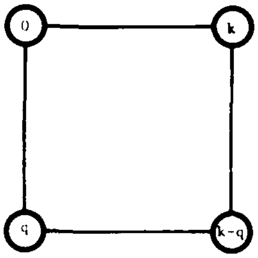


Fig. 7 Nodes in Q.

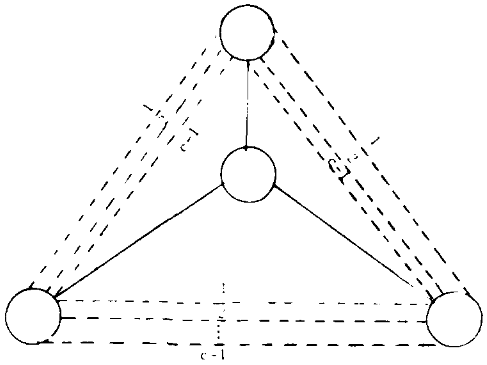


Fig. 8. Detour method of analyzing connectivity.

In the above, the fault tolerance of $FG(r, m)$ networks has been shown to be at least equal to $(r - 1)$. By using an analysis similar to that given for $FG(2, m)$ networks, it can also be shown that the fault tolerance of $FG(r, m)$ networks for m = even is equal to r, and thus is optimally fault tolerant. However, it remains to be seen whether $FG(r, m)$ networks for m = odd are also optimally fault tolerant.

V. Conclusion

This paper presents certain regular networks with optimal (near-optimal) fault tolerance. Various fault-tolerant properties of these networks are summarized in Table II. Further research of interest here is the VLSI layout [8], as well as internal testing and self-diagnosis [1], [4]-[6] of these networks. The binary $FG(2, m)$ network can be considered as a supergraph of the shuffle-exchange graphs [8]; this is known to possess efficient VLSI layouts [9], [18]. Also of interest is how best one can utilize the large degree of fault tolerance available in these networks for both yield enhancement as well as for fault tolerance.

It may also be noted that the graphs presented here may be of interest in the context of (d, k) graphs [4]. Specifically, the $FG(2, m)$ graphs with even m and $FG(r, 2)$ graphs provide optimum connectivity and small diameter for both the original graph and the subgraph obtained after deleting faulty nodes [19], [20]. Other recent work in the area [21]-[23] also have addressed these problems.

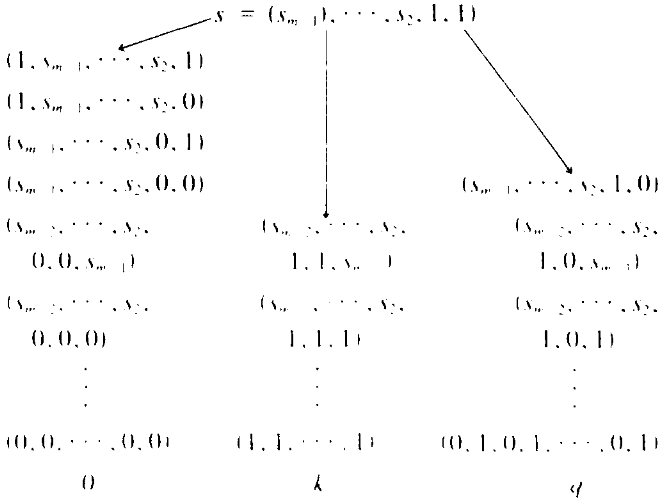
Finally, it may be added that the detour technique provides a tool for analyzing the connectivity [24] of arbitrary graph. One way to establish that the connectivity of a given graph is c is to prove that there exists at least c disjoint paths between every pair of nodes. But this requires a computation of $O(n^2)$ for an n node graph. However, an alternate approach would be to establish that there exist at least $(c - 1)$ independent detours around every node (connecting every pair of neighbors) as illustrated in Fig. 8. Such an analysis may be simpler at it would require a computational complexity of $O(n)$ for bounded degree graphs.

Appendix A

Let Q represent the set of four nodes $(0, k, q, k - q)$. These four nodes are interconnected, as shown in Fig. 7.

Let $N - Q$ represent the set of $(n - 4)$ nodes which consist of all nodes other than the nodes in Q.

To begin with, it may be seen that given any node s one can construct, from s, three disjoint paths to nodes 0, k, and q, below.


$s = (s_{m-1}, \cdots, s_2, 1, 1)$
$(1, s_{m-1}, \cdots, s_2, 1)$
$(1, s_{m-1}, \cdots, s_2, 0)$
$(s_{m-1}, \cdots, s_2, 0, 1)$
$(s_{m-1}, \cdots, s_2, 0, 0)$
$(s_{m-1}, \cdots, s_2, 1, 0)$
$(s_{m-2}, \cdots, s_2, 0, 0, s_{m-1})$
$(s_{m-2}, \cdots, s_2, 1, 1, s_{m-1})$
$(s_{m-2}, \cdots, s_2, 1, 0, s_{m-1})$
$(s_{m-2}, \cdots, s_2, 0, 0, 0)$
$(s_{m-2}, \cdots, s_2, 1, 1, 1)$
$(s_{m-2}, \cdots, s_2, 1, 0, 1)$
$(0, 0, \cdots, 0, 0)$
$(1, 1, \cdots, 1)$
$(0, 1, 0, 1, \cdots, 0, 1)$
0
k
q

$s_1 = s_2 = 1$ m = even

other cases are similar

In constructing these paths, it is assumed that m = even and $s = s_0 = 1$. (Proof of disjointedness can be readily derived.) Similar paths can be constructed for other combinations of (s_1, s_0) as well as when m = odd. Furthermore, it can be seen that there exist three paths from 0, k, and q to any destination node d where the paths are disjoint. The length of these disjoint paths can be seen to be at most equal to $2m$. These observations form the basis for the following path construction procedure, which constructs paths from s to d, given two faulty nodes in the network. This is described by considering various cases which correspond to different distributions of faulty nodes between Q and $N - Q$.

Case I: Both the faulty nodes are in $N - Q$.

This implies that all the four nodes in Q are fault free. Since there are three disjoint paths from s to nodes in Q, one can reach at least one of the nodes in Q with two faulty nodes in $N - Q$. Similarly, one can reach d from one of the nodes in Q. Since all of the nodes in Q are fault free, one can reach d from s through Q by using a path of length at most $2m + 2 + 2m = 4m + 2$ [there exists a path of length 2 between any pair of nodes (Fig. 7)].

Case II: Both the faulty nodes are in Q.

In this case, there must exist at least one node: 0, q, *or* k, through which a path can be constructed from s to d. For example, assume nodes 0 and q are faulty. Since the paths from s to 0, s to q, and s to dk are disjoint, one can reach k in spite of these two faults. Similarly, from k one can reach d. The total path length here will be equal to $4m$.

Case III: Faulty nodes in both Q and $N - Q$.

In this case, there is a node in Q and a node in $N - Q$, both of which are faulty. Thus, the three remaining nodes in Q are fault free and are connected. With a single faulty node in $N - Q$, one can reach at least two of the three remaining fault-free nodes. Similarly, d can be reached from at least two of these three fault-free nodes. Thus, there must exist one node 0, k, or q through which an entire fault-free path from s to d can be constructed. Thus, this path is of length at most $4m + 2$.

Appendix B
Construction of Detours

Case R1: Here, the original path contains the following sequence of nodes represented in radix-r as

$$u = (u_{m-1}, \cdots\cdots\cdots, u_1, u_0)$$

faulty $$v = (u_{m-1}, \cdots\cdots\cdots, u_1, c) \qquad c \neq u_0$$

$$w = (u_{m-2}, \cdots, u_1, c, u_{m-1})$$

First, it will be shown that there exists a *fault-free* path from u, of the type shown below, for some x. Then, this path will be used as part of the detour from u to w.

$$u = (u_{m-1}, \cdots\cdots\cdots, u_1, u_0)$$
$$(u_{m-1}, \cdots\cdots\cdots, u_1, x)$$
$$(u_{m-2}, \cdots, u_1, x, u_{m-1}).$$

In order to prove this, we will consider two separate cases. First, we consider the case $r = 3$, and next consider $r \geq 4$.

i) Let $r = 3$: Thus, one has $t = 1$; therefore, there is only one faulty node v. Consider the path

$$u \to (u_{m-1}, \cdots, u_1, e) \to (u_{m-2}, \cdots, u_1, e, u_{m-1}).$$

This path must be fault free because the node $(u_{m-1}, \cdots, u_1, e)$ cannot be the faulty node v since $e \neq c$, the least significant digit of v. The other node $(u_{m-2}, \cdots, u_1, e, u_{m-1})$ also cannot be faulty because then one has

$$(u_{m-2}, \cdots, u_1, e, u_{m-1}) = v = (u_{m-1}, \cdots, u_1, c).$$

This would in turn imply $c = e$, a contradiction of the definition of e.

ii) Let $r > 4$: Consider the following two subcases, the first corresponding to all of the t faulty nodes that have the same digit in the least significant position; that is, $f_0^1 = f_0^2 = \cdots = f_0^t = c$, the least significant digit of v. The second case corresponds to when the above is *not* the case.

a) Let $f_0^1 = f_0^2 = \cdots = f_0^t$. Consider all the h-neighbors of u. There are altogether $(r - 1)$ of these. No two of these nodes have the same digit in the least significant position. Since all of the faulty nodes have the same digit in the least significant position, only one of these h-neighbors of u can be faulty and this node is v. Thus, the remaining $(r - 2)$ h-neighbors must be fault free.

Without loss of generality, let us assume these nodes to be

$$(u_{m-1}, \cdots, u_1, u_0), (u_{m-1}, \cdots, u_1, 1), \cdots,$$
$$(u_{m-1}, \cdots, u_1, r - 3) \text{ in radix } r.$$

Consider the following $(r - 2)$ paths from u to the g-neighbors of these $(r - 2)$ nodes shown above:

Fault free At most t are faulty

$$\begin{tikzcd}[row sep=small]
 & (u_{m-1}, \cdots\cdots\cdots, u_1, 0) \arrow[r] & (u_{m-2}, \cdots\cdots\cdots, u_1, 0, u_{m-1}) \\
u \arrow[ur] \arrow[r] \arrow[ddr] & (u_{m-1}, \cdots\cdots\cdots, u_1, 1) \arrow[r] & (u_{m-2}, \cdots\cdots\cdots, u_1, 1, u_{m-1}) \\
 & \vdots & \vdots \\
 & (u_{m-1}, \cdots, u_1, r - 3) \arrow[r] & (u_{m-2}, \cdots, u_1, r - 3, u_{m-1}).
\end{tikzcd}$$

The $(r - 2)$ nodes shown in the right-hand side are seen to be all distinct. Therefore, at least $(r - 2 - t)$ of these are fault free. For $r \geq 4$, one has $(r - 2 - t) \geq 1$. There must be at least one path of the type $u \to (u_{m-1}, \cdots, u_1, x) \to (u_{m-2}, \cdots, u_1, x, u_{m-1})$ that is fault free.

For $r = 3$, one has $t = 1$, therefore, there is only one node that is faulty, the node v.

Now, consider the second case for which all of the t digits appearing in the least-significant position of the faulty nodes are not distinct. Thus, at most $(t - 1)$ distinct digits appear in the least-significant position. Consider the $(r - 1)$ h-neighbors of u. All of these nodes have distinct digits in the least-significant position. So, at most $(t - 2)$ of these h-neighbors can be faulty.

Consequently, there must be at least $(t + 1)$ of these h-neighbors that must be fault free since $(r - 1) \geq 2t$. As before, without loss of generality, we can assume these

$(t + 1)$ neighbors as $(u_{m-1}, \cdots, u_1, 0)$, $(u_{m-1}, \cdots, u_1, 1)$, $\cdots, (u_{m-1}, \cdots, u_1, t)$ in radix-r. Consider the following $(t + 1)$ g-neighbors of these nodes, through these nodes:

Fault free At most t are faulty

$$\begin{tikzcd}[row sep=small]
 & (u_{m-1}, \cdots, u_1, 0) \arrow[r] & (u_{m-2}, \cdots, u_1, 0, u_{m-1}) \\
u \arrow[ur] \arrow[r] \arrow[ddr] & (u_{m-1}, \cdots, u_1, 1) \arrow[r] & (u_{m-2}, \cdots, u_1, 1, u_{m-1}) \\
 & \vdots & \vdots \\
 & (u_{m-1}, \cdots, u_1, t) \arrow[r] & (u_{m-2}, \cdots, u_1, t, u_{m-1}).
\end{tikzcd}$$

Since the $(t + 1)$ nodes in the right-hand side are all distinct, there can be at most t of these that can be faulty. Thus, there must exist at least one path of the type $u \rightarrow (u_{m-1}, \cdots, u_1, x) \rightarrow (u_{m-2}, \cdots, u_1 x, u_{m-1})$ that is fault free.

Now consider the following path from u to w that uses the fault-free path, as constructed above.

b) Let $f_0^i \neq f_0^j$ for some $i \neq j$, $1 \leq, i, j \leq t$. Here, consider the following two separate subcases. First assume that the t-digits $f_0^1, f_0^2, \cdots, f_0^t$ appearing in the least significant position of the faulty nodes are all distinct.

One always has at least $(r - 1 - t)$ of the h-neighbors of u which are fault free. Let these fault-free nodes be represented as

$$(u_{m-1}, \cdots, u_1, 0), (u_{m-1}, \cdots, u_1, 1), \cdots,$$
$$(u_{m-1}, \cdots, u_1, r - 2 - t).$$

Now consider the following g-neighbors of these $(r - 1 - t)$ nodes. These may be represented as

$$(u_{m-2}, \cdots, u_1, 0, u_{m-1}), (u_{m-2}, \cdots, u_1, 1, u_{m-1}),$$
$$\cdots, (u_{m-1}, \cdots, u_1, r - 2 - t, u_{m-1}).$$

Note that these nodes are all distinct and have the same digit in the least-significant position. According to the hypothesis, no two faulty nodes have the same digit in the least-significant position. Thus, only one of the above nodes can be faulty the remaining $(r - 2 - t)$ of these must be fault free. For $r \geq 4$, one has $(r - 2 - t) \geq 1$. Consequently, in this case one has at least one path of the type shown below that is fault free.

$$u \rightarrow (u_{m-1}, \cdots, u_1, x) \rightarrow (u_{m-2}, \cdots, u_1, x, u_{m-1})$$

$$\begin{array}{lll}
u = & (u_{m-1}, \cdots\cdots, u_1, u_0) & \text{fault-free segment} \\
 & (u_{m-1}, \cdots\cdots, u_1, x) & \text{constructed as above} \\
 & (u_{m-2}, \cdots, u_1, x, u_{m-1}) & \\
 & (u_{m-2}, \cdots\cdots, u_1, x, e) & e \text{ appears in the least- or} \\
 & (e, u_{m-2}, \cdots\cdots, u_1, x) & \quad \text{most-significant position;} \\
 & (e, u_{m-2}, \cdots\cdots, u_1, c) & \quad \text{hence, the nodes are fault} \\
 & (m_{m-2}, \cdots\cdots, u_1, c, e) & \quad \text{free} \\
 & (m_{m-2}, \cdots, u_1, c, u_{m-1}). &
\end{array}$$

Case R2: $u \rightarrow v = g(u) \rightarrow w = h(v)$. This corresponds to the following path segment:

$$\begin{array}{rl}
 & u = (u_{m-1}, \cdots\cdots, u_1, u_0) \\
\text{faulty:} & v = (u_{m-2}, \cdots, u_1, u_0, u_{m-1}) \\
 & w = (u_{m-2}, \cdots\cdots, u_1, u_0, c) \qquad c \neq u_{m-1}.
\end{array}$$

Now, consider the $(r - 1)$ h-neighbors of u. All of these are distinct, and none of these is the faulty node v since this node cannot be both g and h-neighbor simultaneously. Beside v, there are $(t - 1)$ faulty nodes; thus, at most $(t - 1)$ of the h-neighbors of u can be faulty. This therefore implies that there are at least $(t + 1)$ h-neighbors of u that are fault free since $r - 1 \geq 2t$. So, using arguments similar to those given in Case R1 (when the least significant digits of faulty nodes are not distinct), one can assert that there exists a fault-free path of the type

$$u \rightarrow (u_{m-1}, \cdots, u_1, x) \rightarrow (u_{m-2}, \cdots, u_1, x, u_{m-1}).$$

Now, consider the following path from u to v that uses the above-described fault-free path:

$$\begin{array}{lll}
u = & (u_{m-1}, \cdots\cdots, u_1, u_0) & \\
 & (u_{m-1}, \cdots\cdots, u_1, x) & \\
 & (u_{m-2}, \cdots, u_1, x, u_{m-1}) & \\
 & (u_{m-2}, \cdots\cdots, u_1, x, e) & \\
 & (e, u_{m-2}, \cdots\cdots, u_1, x) & \text{nodes cannot be faulty, as} \\
 & (e, u_{m-2}, \cdots\cdots, u_1, u_0) & \quad e \text{ appears in the least- or} \\
 & (u_{m-2}, \cdots\cdots, u_1, u_0, e) & \quad \text{most-significant position.} \\
w = & (u_{m-2}, \cdots\cdots, u_1, u_0, c). &
\end{array}$$

Case R3: $u \rightarrow v = g(u) \rightarrow w = g(v)$. This corresponds to the following segment:

$$\begin{array}{rl}
 & u = (u_{m-1}, \cdots\cdots, u_1, u_0) \\
\text{faulty:} & v = (u_{m-2}, \cdots\cdots, u_1, u_0, u_{m-1}) \\
 & w = (u_{m-3}, \cdots, u_0, u_{m-1}, u_{m-2}).
\end{array}$$

As in Case R2, one can construct a fault-free path of the type

$$u \rightarrow (u_{m-1}, \cdots, u_1, x) \rightarrow (u_{m-2}, \cdots, u_1, x, u_{m-1}).$$

Consider the $(r - 1)$ h-neighbors of the node $(u_{m-2}, \cdots, u_1, x, u_{m-1})$. None of these nodes has u_{m-1} in the least-significant position; hence, none of these is the faulty node v. Since there are $(t - 1)$ faulty nodes besides v, at most $(t - 1)$ of these h-neighbors can be faulty. Thus, there must be at least $(t + 1)$ of these h-neighbors that are fault free. Without loss of generality, these nodes can be assumed to be $(u_{m-2}, \cdots, u_1, x, 0)$, $(u_{m-2}, \cdots, u_1, x, 1)$, $\cdots$, $(u_{m-2}, \cdots, u_1, x, t)$. Consider the g-neighbors of these nodes, as shown below:

$$(u_{m-3}, \cdots, u_1, x, 0, u_{m-2}), (u_{m-3}, \cdots, u_1, x, 1, u_{m-2}),$$
$$\cdots, (u_{m-2}, \cdots, u_1, x, t, u_{m-2}).$$

These $(t + 1)$ nodes are all distinct; hence, at least one of these must be fault free. Let this fault-free node be $(n_{m-1}, \cdots, u_1, x, y, u_{m-2})$. Thus, a fault-free path of the type shown below is easily constructed:

$$
\begin{aligned}
u &\rightarrow (u_{m-2}, \cdots, u_1, x) \rightarrow \\
&\rightarrow (u_{m-2}, \cdots, u_1, x, u_{m-1}) \rightarrow (u_{m-2}, \cdots, u, x, y) \\
&\rightarrow (u_{m-3}, \cdots, u_1, x, y, u_{m-2}) .
\end{aligned}
$$

Now, using this path, one can construct a detour from u to w, as illustrated in the following:

$$
\begin{array}{lll}
u = & (u_{m-1}, \cdots\cdots\cdots, u_1, u_0) & \\
 & (u_{m-1}, \cdots\cdots\cdots, u_1, x) & \\
 & (u_{m-2}, \cdots\cdots, u_1, x, u_{m-1}) & \\
 & (u_{m-2}, \cdots\cdots\cdots, u_1, x, y) & \\
 & (u_{m-3}, \cdots\cdots, u_1, x, y, u_{m-1}) & \\
 & (u_{m-3}, \cdots\cdots, u_1, x, y, e) & \\
 & (e, u_{m-3}, \cdots\cdots, u_1, x, y) & \\
 & (e, u_{m-3}, \cdots\cdots, u_1, x, e) & \\
 & (e, e, u_{m-3}, \cdots\cdots, u_1, x) & e \text{ appears in the most- or} \\
 & (e, e, u_{m-3}, \cdots\cdots, u_1, u_0) & \quad \text{least-significant} \\
 & (e, u_{m-3}, \cdots\cdots\cdots, u_0, e) & \quad \text{position; hence, these} \\
 & (e, u_{m-3}, \cdots\cdots, u_0, u_{m-1}) & \quad \text{nodes cannot be faulty} \\
 & (u_{m-3}, \cdots\cdots, u_0, u_{m-1}, e) & \\
w = & (u_{m-3}, \cdots, u_0, u_{m-1}, u_{m-2}) . &
\end{array}
$$

References

[1] J. R. Armstrong and F. G. Gray, "Fault diagnosis in Boolean *n*-cube array of microprocessors," *IEEE Trans. Comput.*, vol. C-30, pp. 590–596, Aug. 1981.

[2] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," in *Proc. 20th Annu. IEEE Symp. Found. Comput. Sci.*, 1979.

[3] E. Horowitz and A. Zorat, "The binary tree as an interconnection network applications to multiprocessor systems and VLSI," *IEEE Trans. Comput.*, vol. C-30, pp. 247–253, Apr. 1981.

[4] S. B. Akers, "On the construction of (d, k) graphs," *IEEE Trans. Comput.*, vol. C-19, 1965.

[5] I. Koren, "A reconfigurable and fault-tolerant VLSI multiprocessor array," in *Proc. 8th Annu. Symp. Comput. Arch.*, Minneapolis, MN, May 1981.

[6] J. Kuhl and S. Reddy, "Distributed fault-tolerance for large multiprocessor systems," in *Proc. 7th Annu. Symp. Comput. Arch.*, May 1980, pp. 23–30.

[7] D. K. Pradhan and S. M. Reddy, "A fault-tolerant communication architecture for distributed systems," *IEEE Trans. Comput.*, vol. C-31, Sept. 1982.

[8] H. S. Stone, "Parallel processing with perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153–161, Feb. 1971.

[9] F. T. Leighton, "Layouts for the shuffle-exchange graphs and lowerbound techniques for VLSI," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, 1981.

[10] M. L. Schlumberger, "DeBruijn communication networks," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1974.

[11] J. P. Hayes, "A graph model for fault-tolerant computing systems," *IEEE Trans. Comput.*, vol. C-25, pp. 876–884, Sept. 1976.

[12] C. L. Kwan and S. Toida, "Optimal fault-tolerant realizations of some classes of hierarchical tree systems," in *Proc. FTCS-11*, June 1981, pp. 176–178.

[13] L. Bhuyan and D. P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Comput.*, vol. C-30, Apr. 1984.

[14] A. H. Esfahanian and S. L. Hakimi, "Fault-tolerant routing in DeBruijn communication networks," *IEEE Trans. Comput.*, to be published.

[15] M. C. Pease, "The indirect binary *n*-cube microprocessor array," *IEEE Trans. Comput.*, vol. C-26, pp. 458–473, May 1977.

[16] M. Imase and M. Itoh, "Design to minimize a diameter on building block network," *IEEE Trans. Comput.*, vol. C-30, pp. 439–433, June 1981.

[17] M. T. Liu, "Distributed loop computer networks," *Advances Comput.*, vol. 17, pp. 163–221, 1978.

[18] C. D. Thompson, "A complexity theory for VLSI," Ph.D. dissertation, Carnegie-Mellon Univ., Pittsburgh, PA, Aug. 1980.

[19] J. Kuhl, "Fault diagnosis in computing networks," Ph.D. dissertation, Univ. Iowa, Iowa City, IA, 1980.

[20] D. K. Pradhan, "Interconnection topologies for fault-tolerant parallel and distributed architectures," in *Proc. 1981 Int. Conf. Parallel Processing*, Aug. 1981, pp. 238–242.

[21] W. E. Leland, "Density and reliability of interconnection topologies for multicomputers," Ph.D. dissertation, Univ. Wisconsin-Madison, Madison, WI, May 1982.

[22] V. P. Kumar and S. M. Reddy, "A class of fault-tolerant processor interconnections," in *Proc. 4th Int. Conf. Distrib. Comput. Syst.*, San Francisco, CA, May 1984, pp. 448–460.

[23] S. B. Akers and B. Krishnamurthy, "Group graphs as interconnection networks," in *Proc. FTCS-14*, Orlando, FL, June 1984, pp. 422–427.

[24] J. Bondy and U. Murthy, *Graph Theory with Applications*. New York: American Elsevier, 1976.

[25] R. A. Finkel and M. H. Solomon, "The lens interconnection strategy," *IEEE Trans. Comput.*, vol. C-30, pp. 960–965, Dec. 1981.

[26] M. Malek and E. Opper, "Multiple fault-diagnosis of SW-banyan networks," in *Proc. FTCS-13*, Milan, Italy, June 1983.

[27] D. K. Pradhan, Z. Hanquan, and M. L. Schlumberger, "Fault-tolerant multi-bus architectures for multiprocessors," in *Proc. FTCS-14*, Orlando, FL, June 1984, pp. 400–408.

Dhiraj K. Pradhan (S'70–M'72–SM'80) was born in India on December 1, 1948. He received the M.S. degree from Brown University, Providence, RI, in 1969, and the Ph.D. degree from the University of Iowa, Iowa City, in 1972.

He is currently a Professor in the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst. Previously he has held positions with Oakland University, Rochester, MI and the IBM Corporation, NY. He has been actively involved with research in fault-tolerant computing and parallel processing since 1972. He has presented several papers at fault-tolerant computing and parallel processing conferences. He has also published extensively in journals such as the IEEE TRANSACTIONS ON COMPUTERS and *Networks*. His research interests include fault-tolerant computing, computer architecture, graph theory, and flow networks.

Dr. Pradhan has edited the Special Issue on Fault-Tolerant Computing of IEEE COMPUTER (March 1980), and served as Session Chairman and Program Committee Member for various conferences. He is an Editor for the *Journal of VLSI and Digital Systems*. He is also the Editor of a forthcoming book entitled *Fault-tolerant Computing: Theory and Techniques* (Prentice-Hall).

DYNAMIC TESTING STRATEGY FOR DISTRIBUTED SYSTEMS

Fred J. Meyer
Dhiraj K. Pradhan

Reprinted from PROCEEDINGS OF THE FIFTEENTH ANNUAL INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING — FTCS 15, Ann Arbor, Michigan, June 19-21, 1985

DYNAMIC TESTING STRATEGY FOR DISTRIBUTED SYSTEMS

by

Fred J. Meyer and Dhiraj K. Pradhan

Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, Massachusetts 01003

Abstract: Testing and diagnosis is an important consideration in the implementation of fault-tolerant distributed systems. This paper treats fault diagnosis as two distinct processes: fault discovery and dissemination of diagnostic information. The corresponding testing overhead consists of periodic tests for fault discovery and further tests and/or message passing for dissemination of diagnostic information. Both homogeneous and nonhomogeneous systems in both synchronous and asynchronous environments are discussed.

Previous research derived precisely when a given set of tests in a homogeneous system can achieve a specified level of self-diagnosability. A new methodology is presented with the objective of minimizing the overhead associated with periodic testing. Minimizing periodic testing allows less testing overhead, greater test reliability, and/or more frequent testing. The method diagnoses up to t faults, where t is the fault-tolerance of the system (t is one less than the connectivity of the communication graph).

Index Terms: distributed computing system, fault diagnosis, fault discovery, fault tolerance, PMC models, self-diagnosable system, testing overhead.

I. INTRODUCTION AND MOTIVATION

Design of highly reliable systems has become increasingly important over the last two decades. It seems unrealistic, for instance, to attempt wafer scale integration without allowing for proper operation of the product despite some flawed components. Originally, research was devoted primarily to the efficient determination of tests for logic circuits. However, with the advent of LSI, system level diagnosis has become increasingly relevant.

Earlier work on system level diagnosis, by Preparata, Metze, and Chien [PrMe67], has provided the basis for much of the recent work. Hakimi and Amin [HaAm74] later have provided some fundamental results where a central observer diagnoses the system by interpreting the set of syndrome test results. More recently, interest has focused on diagnosis and reliability of distributed computing systems. Kuhl and Reddy [KuRe80] have developed a distributed diagnostic method in which the need was emphasized to account for the possibly malicious conduct of faulty components. Though research into central observer models continues (e.g. [DaMa84], researchers proposing [illegible] (e.g. [illegible]) have found it prudent to assume both the needs of a distributed environment and the potential for malicious faults (while research assuming dormant faulty components largely ceased).

Necessary and sufficient conditions on a static set of tests to achieve a level of diagnosability is established in [KuRe80]. They subsequently enhanced their method to be able to diagnose faulty communication links [KuRe81]. Hosseini pursued this research, investigating (nonhomogeneous) systems with a restricted set of allowable tests [Hoss83]; also that work sought to allow the incorporation into the system of rehabilitated components [Hoss83,HoKu84]. As did [PrMe67,KuRe80], we take testing to be singly-invalidating, singly-complete, symmetrically-invalidating, non-hardcore (SISN) [Kime85].

Most previous research has considered either: (1) the diagnosability of a system with a predetermined static testing graph [KuRe80] or (2) adaptive testing graphs (where one test is conducted at a time, its result determining the next test) [HaNa84]. Our approach is to determine a minimal testing graph (as measured by the number of edges) that may be applied to diagnose at least one fault. The distinction between our approach and earlier work is that the tests are neither conducted sequentially (as in adaptive methods), since the graph is known, nor is the graph static (as suggested by [KuRe80]). Instead, after a fault has been diagnosed, a new minimal graph is used to diagnose subsequent faults.

We adopt a graph-theoretic model of a distributed computing system, where graph G = (V,E). The vertices in V represent processors in the system; the edges in E represent communication links between processors. The edges in E are labelled (a,b), where a and b are labels for vertices in V. Let there be n nodes in G, n = |V|. The distance between two nodes is the minimum number of edges that must be traversed to travel between them. The diameter, K, of G is the maximum of the distances between all possible pairs of nodes. The f-fault diameter, K_f, is the maximum of the diameters of all graphs obtainable from G by removing any f nodes. The connectivity, κ, of G is the minimum number of nodes that must be removed in order to disconnect G [illegible] of arbitrary nodes. G can tolerate κ−1 node faults without risking disconnection.

Kuhl and Reddy [KuRe80] showed that a directed testing subgraph of G, with connectivity t, is

Research reported supported by grant AFOSR 84-0[illegible].

necessary and sufficient to diagnose t faults. They assumed that the testing graph is static. We consider what benefits may be enjoyed if the testing graph is modified as faults are discovered. Since each node must be tested, it is necessary for the testing graph to have at least n edges, whereas a t-connected graph has at least nt edges. Their strategy requires more than n tests (unless t = 1 and G is Hamiltonian [Deo 74]).

Decreasing the testing required from O[nt] tests to O[n] tests would improve the performance of the system. The savings could be distributed in any way desired amongst these three factors:

1) testing overhead. Some of the system time devoted to testing could be recovered for useful work.

2) test reliability. The fewer tests could be allotted more time--likely making them more thorough.

3) test frequency. The fewer tests could be conducted more frequently yielding a better average time between component failure and detection.

Diagnosis must be considered in both synchronous and asynchronous environments. A synchronous environment is usually achieved by message passing; the processing elements operate as though with a common clock. A synchronous environment enjoys the advantage of allowing the processors to conduct their tests simultaneously. This feature permits diagnosis by an analysis of the set of test results.

We adopt a more flexible strategy than [KuRe80]--once a node has been diagnosed faulty a new testing graph is used. We disseminate news of a fault in the same manner as [KuRe80]. After f faults have already been diagnosed, our procedure requires n-f tests on a periodic basis. Forming a proper testing graph requires the existence of a loop that includes at least t+1-f = n-f nodes. The existence of loops with at least 2(n-f) nodes is shown; our procedure, therefore, may always be applied.

For continually on-line systems, testing is conducted periodically and faults generally do not occur simultaneously. For these systems, testing overhead is reduced essentially by a factor of t compared to [KuRe80]. For some systems, it may be appropriate to shut down the system for testing. Faults are more likely to appear to occur simultaneously for such systems. The procedure we outline allows downtime to be proportional to the number of faults that have occurred since last testing, instead of being proportional to the number of faults tolerated.

Section 2 discusses the fault-diagnostic capabilities of a Hamiltonian circuit testing graph. In section 3 an algorithm is presented for diagnosis in an asynchronous environment. An algorithm for diagnosis in a synchronous environment is given in section 4. Section 5 summarizes our results.

II. HAMILTONIAN TESTING GRAPH

Preparata, Metze, and Chien [PrMe67] have considered the requirements for a test-and-replace strategy to succeed with a loop testing graph. Kavianpour and Friedman have given the optimal Strategy 3 [KaFr78], which interprets the test results of a Hamiltonian [Deo 74] testing graph. We review [PrMe67] here in the context of our dynamic testing graph strategy.

Consider a testing graph that consists of a circuit comprised of k nodes. What is desired are necessary and sufficient conditions on k to diagnose a fault in synchronous and asynchronous environments. If more than one fault is present, our approach requires only that at least one fault be discovered,because a new testing graph will then be employed, omitting that faulty node.

Theorem 2.1: In an asynchronous environment, a fault is guaranteed to be discovered in the presence of at most t faults iff k ≥ t+1.

Proof: In an asynchronous environment, the discoverer of a fault advises the network by broadcasting the information. Nodes in the network receiving such information accept the diagnosis iff they receive it from a node they subsequently test successfully. This test-and-broadcast strategy guarantees that the entire network will be informed of all faults discovered.

A fault is discovered iff the faulty node is tested by a nonfaulty node. If k ≤ t, then all the nodes of the loop could be faulty (=> necessity). If k ≥ t+1 then there is at least one nonfaulty node; further, since the testing graph is Hamiltonian, not all faulty nodes can be tested by other faulty nodes (=> sufficiency). □

Theorem 2.2: In a synchronous environment, a fault is guaranteed to be discovered in the presence of at most t faults iff k ≥ $(t+3)^2/4$.

Proof: See [MeCi85] or [PrMe67] for proof. The expression $(t+3)^2/4$ is equivalent to that found in [PrMe67], but in a simpler form. □

In this section, the requirements on the size of Hamiltonian circuits to be able to diagnose at least one fault, for both synchronous and asynchronous environments, have been reviewed. These results will be extended to the problem of distributed diagnosis in non-Hamiltonian graphs--asynchronous environments in Section 3, synchronous environments in Section 4.

III. ASYNCHRONOUS LOOP DIAGNOSIS

The previous section established the requirements for a Hamiltonian graph to be capable of discovering at least one fault if the testing graph is a Hamiltonian circuit. Graphs must also be considered that are not Hamiltonian or become non-Hamiltonian once faults occur. This section will first present an algorithm that accepts as input a graph, G, and a loop, L, and constructs a testing graph with only one loop (L). Then it is shown that the testing graph so constructed will discover at least one fault iff a graph formed only of L can discover a fault. Then follows a demonstration that every graph contains adequate loops; also an algorithm is offered that finds such a loop in the presence of faults.

Let G = (V,E) be a graph on n vertices with a subgraph, L, consisting of l vertices from a circuit of G and the edges comprising that circuit. An algorithm is presented that produces a testing graph, T = (V',E'), with the minimal n edges, and which includes only one circuit, L.

Algorithm TFORMER(G,L):

(1) Let V' = V. Let E' = { all directed edges in one direction of the circuit L }. Let N = { v ∈ V' : ∀ (a,b) ∈ E' b ≠ v }; i.e., N is the set of all nodes in V' not tested according to E'.

(2) V'-N is the set of nodes in V' not included in N. Let N' = { v ∈ N : ∃ a ∈ V'-N such that (a,v) ∈ E } and let a(v) be a function ∀ v ∈ N' returning an arbitrary node, a, for which, (a,v) ∈ E. a(v) will be the tester of v.

(3) Let E' ← E' ∪ { (a(v),v) : v ∈ N' }. Renew N = { v ∈ V' : ∀ (a,b) ∈ E' b ≠ v }.

(4) If N ≠ ∅ then go to (2) else let T = (V',E').

It is obvious that the only circuit in the testing graph thus formed is the one used as a basis for its construction. The algorithm will terminate unless N' = ∅ while N ≠ ∅. This, however, cannot occur, because it would require that no node in N be linked to any node in V'-N (which would indicate that the graph is disconnected).

Figure 3.1 illustrates the successive sets (E',N,N') in an application of TFORMER to a 5x5 end-around mesh, where the chosen circuit consists of the nodes around the border of the mesh. All edges of the mesh shown are in E'. Nodes in N-N' are indicated by "N" in their interior. Nodes in N' ⊆ N are indicated by "N'" in their interior. Now it is shown that TFORMER yields a testing graph that is adequate to diagnose a first fault with the flexible testing strategy presented here.

Theorem 3.1: In an asynchronous environment with test-and-broadcast dissemination of fault discovery, TFORMER(G,L) yields a testing graph that will diagnose at least one fault in the presence of up to t faults iff L consists of at least t+1 vertices.

Proof: With test-and-broadcast dissemination of fault discovery as in SELF? [KuRe80], it is necessary and sufficient for at least one faulty node to be tested by a nonfaulty node. Let i be an arbitrary node in V. Consider its unique sequence of predecessors in the testing graph. Since there are a finite number of nodes, this sequence must repeat. The loop embedded in this sequence must be L since L is the sole loop in T. Either (1) i is a node in L or (2) i is not a node in L.

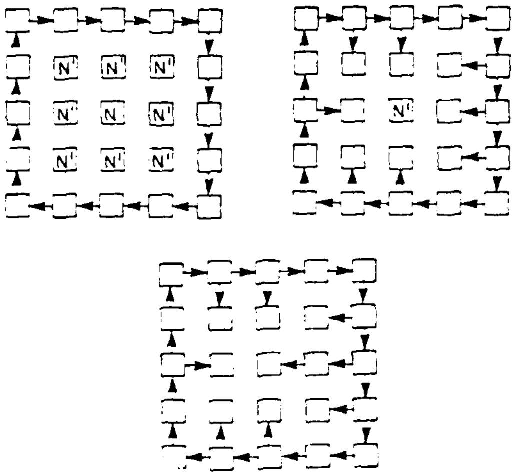


Figure 3.1: TFORMER Applied to a 5x5 End-round Mesh

(1) i is a node in L. No nodes in L are tested by nodes outside L. If there are no faults outside L, i can be diagnosed only if L meets the loop criterion established in the previous section. i can be diagnosed if L meets the loop criterion established in the previous section. i can be diagnosed, therefore, iff L consists of at least t+1 vertices.

(2) i is not a node in L. L must consist of at least t+1 vertices, otherwise [illegible] directly diagnosable. Let L consist of at least t+1 vertices. If there is a fault in L, then at least one fault will be discovered [illegible] a fault in L. If there are no faults in L, then the first positive test result [illegible] the path from L to i clearly defines a fault. If there are no positive test results from L to i, then i may be confidently diagnosed nonfaulty. In any instance, either i may be diagnosed or a fault is known elsewhere in T.

Corollary 3.1: A system [illegible] with minimal testing graph T [illegible] at least one fault in the presence of [illegible] iff ∀ L circuits ∈ T, [illegible] The [illegible]

Proof: [illegible]

[illegible]

Proof: [illegible]

f<c, so G' includes at least two adjacent nodes. Let a and b be any two adjacent nodes in G'. Let L consist of the two-node circuit including a and b. If f = c-1 then L includes 2(c-f) nodes. So we assume f < c-1. Since G' is formed by removing f vertices from G, which has connectivity, c, G' must have connectivity at least c-f. With f < c-1 and connectivity at least c-f, the connectivity of G' is at least two.

L includes n-f nodes unless there are nodes in G' outside L. Let v be an arbitrary node in G', but not in L. Since G' has connectivity at least c-f, there are c-f (node-disjoint) paths from v to distinct nodes in L [Dec 74]. Unless L includes at least 2(c-f) nodes, two of the nodes in L that v connects to must be linked. Let these nodes be y and z. Then the loop may be extended by replacing the edge between y and z with the paths between v and each of y and z. So the number of nodes in L may always be augmented until it includes at least 2(c-f) nodes or exhausts the n-f nodes in G'. □

Corollary 3.2: If G' is a graph formed by removing f vertices and their incident edges from G, where f < c, then ∀ i ∈ V there exists a circuit, L ⊆ G' on ℓ vertices such that ℓ ≥ min { n-f , 2(c-f) } and i ∈ L.

Proof: In the proof of Theorem 3.2, let i be one of the two adjacent nodes comprising the initial two-node circuit. Then i will be in the final loop arrived at. □

Since n-f ≥ c-f, 2(c-f) > c-f when f < c, and Theorem 3.1 requires a loop consisting of at least c-f nodes, Theorem 3.2 guarantees that an adequate loop always exists. An algorithm for finding a loop follows readily from the demonstration of Theorem 3.2. Such an algorithm is given below. It assumes a routing function, PATHFINDER(G,F,a,b), is available, which, given a graph, G, a set of faults, F, and two nonfaulty nodes, a and b, returns an ordered vector of nodes with b last and a first, describing a fault-free [illegible].

[illegible] [illegible]:

(1) Let i be an arbitrary node in L. Let [illegible] an arbitrary [illegible] f [illegible] in L. We describe [illegible], v_1, v_2, ..., $v_{|L|}$ [illegible] where [illegible], ..., [illegible] $v_{|L|}$ [illegible]

[illegible]

then go to [illegible].

[illegible] Let A = [illegible], ..., [illegible].

Let j be the least [illegible] that v_j, v_{j+1} [illegible] A. ∃ [illegible] that [illegible] = v_{j+1}. Let [illegible] be the last [illegible] the paths, PATH(G,F_i,v,v_j) and PATH(G,F[illegible]). ∃ c_x, c_r such that [illegible] is the [illegible] PATHFINDER(G,$F_{x'r}$,v,[illegible]).

(6) Define rev to be a function that accepts an ordered vector and returns a vector of the same length, but with the elements listed in reverse order. Define cdr to be a function that accepts an ordered vector and returns the vector without the first element. Augment L ← (v_1, ...

..., v_{j-1}, rev(cdr$^{c_x-1}$(PATHFINDER(G,F_x,v,[illegible]))),

cdr^{j_b}(PATHFINDER(G,F_z,v,v_{j+1})), v_{j+2}, ..., $v_{|L|}$)

(7) If |L| < [illegible] then go to [illegible].

Figure [illegible] illustrates an application of [illegible] LFINDER to the Moore graph with diameter three and [illegible]

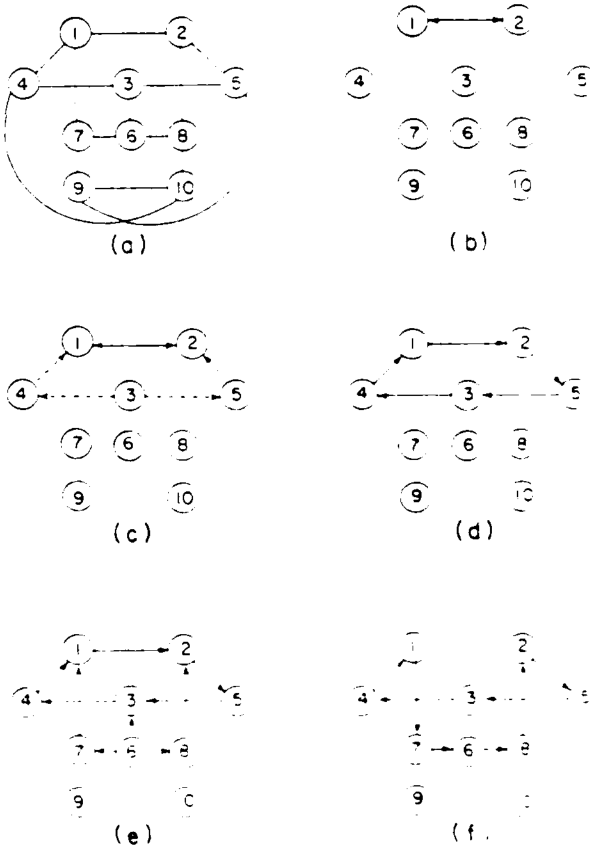


Figure [illegible] A[illegible] [illegible] M[illegible]
[illegible]

IV. SYN[illegible]N[illegible] [illegible]P DIAGNOSIS

[illegible] with [illegible] different [illegible] might [illegible] different faults; this will [illegible] agreement [illegible] the minimal testing graph [illegible] further faults. Byzantine agreement [illegible] when [illegible]. If a fault has occurred, [illegible] same testing [illegible] Byzantine agreement. The [illegible] sufficient that Byzantine agreement [illegible] testing message [illegible] will receive a copy of the [illegible] will [illegible] verify the signature of the [illegible].

The algorithm, [illegible], given in [illegible] [illegible] up to [illegible] faults [illegible] at least [illegible] tests [illegible] procedure that [illegible] faults and the conditions, [illegible] requires the minimal [illegible] tests. It [illegible] that this [illegible] iff [illegible] circuit [illegible] consists of more than [illegible]. The result indicates that the procedure [illegible] minimal tests when applied to all graphs with [illegible] at most five, provided Byzantine agreement [illegible] authentication is [illegible]. We relate the fault-free [illegible] of the graph [illegible] size of [illegible] providing a sufficiency condition for the [illegible] of adequate [illegible].

The testing graph employed here is the same as that in the [illegible] obtained by [illegible] algorithm [illegible]. We mention that [illegible] at least one fault in the presence [illegible] faults iff [illegible].

[illegible] Byzantine agreement [illegible] results, [illegible] yields [illegible] graph that will diagnose at least one fault in the presence of [illegible] faults iff [illegible] more than [illegible].

[illegible] fault iff [illegible]

condition will be derived pertaining to the diameter. Let K(G) denote the diameter of G. Let G_F denote the graph obtained by removing the nodes of F and their incident edges from G.

Theorem 4.1: If $2K > (c+1)^2/4$ then $\forall$ F such that $|F| < c-1$, G_F contains a loop with more than $(c-|F|+1)^2/4$ nodes.

Proof: See [MePr85] for proof. □

We see then that if $2K > (c+1)^2/4$ our synchronous loop diagnostic procedure may always be applied. We suggest there are stronger relationships between c, K, and the existence of suitable loops, L.

Further research could evolve around the following ingredients in a plan intent on synchronous diagnosis:

1) an efficient loop finding algorithm--even one that will not find an existing adequate loop if the time required would be exorbitant

2) an algorithm to form a testing graph from the loop

3) an efficient Byzantine agreement algorithm with authentication

and if the loop is adequate then

4a) a syndrome-interpreting algorithm, as indicated by the proof of Theorem 2.2

else, either apply asynchronous loop diagnosis for detection of the next fault, or, upon discovery of a fault, alert the system to apply more tests, as indicated by

4b) an algorithm to form a nonminimal testing graph and a corresponding fault-detection algorithm.

The alternative suggested in (4b) applies to those graphs where adequate loops for synchronous diagnosis are nonexistent (or difficult to find) and testing is so expensive that it must be minimized. A total of n periodic tests are sufficient for a fault to be discovered in an arbitrary graph, as has been shown in section 3. The asynchronous procedure requires n-f-2 additional tests to inform the remainder of the network of the fault. Fewer than n-f-2 additional tests may be sufficient to augment the syndrome so that it may clearly diagnose a fault. Byzantine agreement may be required in the message that alerts the network of a fault. What additional tests are necessary may depend on the tester and tested units that are in conflict. The additional tests may be [illegible], but it is necessary for the tests to allow the syndrome interpreter to succeed when the additional tests occur shortly after the original set of tests.

V. CONCLUSIONS

We have pursued a strategy of not utilizing the full capacity of the allowable testing graph in an effort to arrive at more efficient diagnosis. We estimate that the asynchronous algorithm of Kuhl and Reddy [KuRe80] will prove more efficacious than our synchronous approach. For asynchronous environments, we have ascertained that our approach is applicable to all homogeneous systems. In exchange for not being able to diagnose the disconnecting fault, we reduce substantially the testing overhead required.

We have not treated the diagnosis of edge faults. As in our attention to node faults, algorithms SELF3 [KuRe81] and modified SELF3 [Hoss82] require only that a fault be discovered. It is manifest that all edge faults (without an incident node fault) are discovered iff every edge in the communication graph appears in at least one direction in the testing graph. The implication that at least half the entire testing graph be used may be unacceptable for some systems. If a communication link may be tested more facilely than a processor, our diagnostic strategy may be augmented by performing the additional link tests separately.

For continually on-line systems, testing is conducted periodically and faults generally do not occur simultaneously. For those systems, testing overhead is reduced essentially by a factor of t. For some systems, it may be appropriate to shut down the system for testing. Faults are more likely to appear to occur simultaneously for such systems. The procedure we outline allows downtime to be proportional to the number of faults that have occurred since last testing, instead of being proportional to the number of faults tolerated.

Not all nodes need to execute LFINDER. In asynchronous envionments the discoverer of the fault discerned by the syndrome is known to be nonfaulty, so that node can be tasked with finding a new loop and the loop can be agreed upon by application of a Byzantine agreement algorithm. In synchronous environments the discoverer of a fault can execute LFINDER and broadcast the loop along with news of the fault. The connectivity of the graph of available tests guarantees that news of every fault will eventually be disseminated to all nonfaulty nodes. So inclusion of a list of known faulty nodes in a diagnostic broadcast will allow a node receiving new diagnostic information to know whether the node originally finding the received loop was yet aware of other faults in the network. If a discrepancy occurs, then a new loop can be found.

Systems have been considered under the assumption that each node can test all its neighbors and only its neighbors. Hosseini [Hoss82] has investigated networks where some tests either may not be possible or may be impractical. This may be due to passive devices, such as memories that do not have the capacity to test their neighbors. Also, some simple devices would [illegible] cost-effective [illegible] cost-effective if enhanced with testing capability. All potentially faulty components must be subject to detection.

In [Hoss82] necessary conditions are given for a nonhomogeneous graph. These are observations that each node must be subject to test and that each node must be capable of discerning interpretation of diagnostic information. Let $G^* = (V,E^*)$ be $G = (V,E)$ with the edges corresponding to the unallowed tests removed; the edges in E^* are directed.

From our discussions in sections 3 and 4, we can see that not all nodes need to test another node to ensure that a fault is discovered. The work of Hosseini determines how many faults may be diagnosed if the adopted testing graph includes all allowed tests. Consider a system, $G;G^*$, that is (t+1)-fault diagnosable according to [Hoss82]. We seek to minimize the number of tests that must be conducted periodically; once a fault has been discovered, the dissemination of the information is guaranteed by [Hoss82].

A more detailed discussion may be found in [MePr85]. For those nonhomogeneous systems that we can determine to be (t+1)-fault diagnosable by the sufficient conditions in [KuRe80] and in [Hoss82], we can employ TFORMER and an amalgamate of NON_HOMO1 and NON_HOMO2 [Hoss82] to arrive at a minimalistic diagnostic strategy.

Finally, the following suggests a topic for further research. We have derived testing graphs wherein each node is tested by only one neighbor. This minimizes the number of tests required. Not all nodes, though, need test a neighbor. Unless the graph is Hamiltonian [Deo 74], some node must test more than one other node. Depending on the type of testing engaged, it might or might not be convenient for some nodes to test many others. We have not established how to minimize the out incidence of the testing graph, but we conjecture that no node need test more than two neighbors periodically.

Conjecture 6.1: For an arbitrary homogeneous system, $G = (V,E)$, with connectivity, c, and for an arbitrary loop, $L \subseteq G$, such that $|L| \geq c$, $\exists\ T = (V',E') = (V,E')$ such that: $E' \subseteq E$, $\forall\ v \in V'\ |\{a \in V' : (a,v) \in E'\}| = 1$, $\forall\ v \in V'\ |\{a \in V' : (v,a) \in E'\}| \leq 2$, there exists only one loop, L, in T.

In conjunction with this, a corresponding efficient algorithm to produce such a testing graph, given an arbitrary adequate loop, would be necessary to make this a possibility of anything other than academic interest.

VII. BIBLIOGRAPHY

[DaMa84] Dahbura, A. and G. Masson, "An $O(n^{2.5})$ Fault Identification Algorithm for Diagnosable Systems," IEEE Trans. Computers, C-33, No. 6 (June 1984), 486-492.

[Deo 74] Deo, N., Graph Theory with Applications to Engineering and Computer Science, Englewood Cliffs, NJ: Prentice-Hall, 1974.

[Dole82] Dolev, D., "The Byzantine Generals Strike Again," Journal of Algorithms 3, 1982, 14-30.

[HaAm74] Hakimi, S. and A. Amin, "Characterization of Connection Assignment of Diagnosable Systems," IEEE Trans. Computers, C-23, No. 1 (January 1974), 86-88.

[HaNa84] Hakimi, S. and K. Nakajima, "On Adaptive System Diagnosis," IEEE Trans. Computers, C-33, No. 3 (March 1984), 234-240.

[Hoss82] Hosseini, S., "Fault Tolerance in Distributed Computing Systems and Databases," PhD dissertation, Department of Electrical and Computer Engineering, University of Iowa, 1982.

[HoKu84] Hosseini, S., J. Kuhl, and S. Reddy, "A Diagnosis Algorithm for Distributed Computing Systems with Dynamic Failure and Repair," IEEE Trans. Computers, C-33, No. 3 (March 1984), 223-233.

[KaFr79] Karunanithi, S. and A. Friedman, "Analysis of Digital Systems Using a New Measure of System Diagnosis," IEEE Trans. Computers, C-28, No. 2 (February 1979), 121-133.

[Kime85] Kime, C., "System Diagnosis," in Fault-Tolerant Computing: Theory and Techniques, Englewood Cliffs, NJ: Prentice-Hall, 1985.

[KuRe80] Kuhl, J. and S. Reddy, "Distributed Fault-Tolerance for Large Multi-processor Systems," Proc. Seventh Intl. Sym. Computer Architecture, 1980, 23-30.

[KuRe81] Kuhl, J. and S. Reddy, "Fault Diagnosis in Fully Distributed Systems," Digest of Papers - 11th Intl. Sym. on Fault-Tolerant Computing, 1981, 100-105.

[LaSh82] Lamport, L., R. Shostak, and M. Pease, "The Byzantine Generals Problem," ACM Trans. on Prog. Lang. & Systems, Vol. 4, No. 3 (July 1982), 382-401.

[MePr85] Meyer F., and D. Pradhan, "Dynamic Testing Strategy for Distributed Systems," Technical Report, UMass-Amherst, January 1986.

[PrRe82] Pradhan, D. and S. Reddy, "A Fault-Tolerant Communication Architecture for Distributed Systems," IEEE Trans. Computers, C-31, No. 9 (September 1982), 863-870.

[PrMe67] Preparata, F., G. Metze, and R. Chien, "On the Connection Assignment Problem of Diagnosable Systems," IEEE Trans. Electronic Computers, EC-16, No. 6 (December 1967), 848-854.

YIELD AND PERFORMANCE ENHANCEMENT THROUGH REDUNDANCY IN VLSI AND WSI MULTIPROCESSOR SYSTEMS

ISRAEL KOREN and DHIRAJ K. PRADHAN

Reprinted from IEEE Proceedings Vol 74. No 5. May 1986

Yield and Performance Enhancement Through Redundancy in VLSI and WSI Multiprocessor Systems

ISRAEL KOREN, MEMBER, IEEE, AND DHIRAJ K. PRADHAN, SENIOR MEMBER, IEEE

New challenges have been brought to fault-tolerant computing and processor architecture research because of developments in IC technology. One emerging area is development of architectures built by interconnecting a large number of processing elements on a single chip or wafer. Two important areas, related to such VLSI processor arrays, are the focus of this paper; they are fault-tolerance and yield improvement techniques.

Fault tolerance in these VLSI processor arrays is of real practical significance; it provides for much-needed reliability improvement. Therefore, we first describe the underlying concepts of fault tolerance at work in these multiprocessor systems. These precepts are useful to then present certain techniques that will incorporate fault tolerance integrally into the design. In the second part of the paper we discuss models that evaluate how yield enhancement and reliability improvement may be achieved by certain fault-tolerant techniques.

I. Introduction

The evolution of fifth-generation computers [44] makes it clear that traditional sequential computer architecture will soon see a striking departure, overtaken by newer architectures which use multiple processors as the state of the art. This particular thrust is enhanced by developments in IC technology [30], creating a widening gap between the technological advances and the architectural capabilities that can exploit these fully.

As a result, much recent research has focused on these new architectural innovations, especially those created by interconnecting multiple processing elements (PEs). One important class of such architectures is VLSI systems that interconnect a very large number of simple processing cells all on a single chip or wafer. Concerns about fault tolerance in VLSI based systems stem from the two key factors of reliability and yield enhancements. Low yield is a problem of increasing significance as circuit density grows. One solution suggests improvement of the manufacturing and testing processes, to minimize manufacturing faults. However, this approach is not only very costly, but also quite difficult to implement, with the increasing number of components that can be placed on one chip. However, incorporating redundancy for fault tolerance does provide a very practical solution to the low yield problem. This has been demonstrated in practice for high-density memory chips and should be extended to other types of VLSI circuits. In general, yield may be enhanced because the circuit can be accepted, in spite of some manufacturing defects, by means of restructuring, as opposed to having to discard the faulty chip. Achieving reliable operation also becomes increasingly difficult with the growing number of interconnected elements and hence, the increased likelihood that faults can occur.

In the design of such fault-tolerant systems, a major architectural consideration becomes the system interconnection. Consequently, one goal of this work is the study of sound fault-tolerant network architectures that can be well utilized in a wide range of VLSI-based systems. Also, of importance are the related problems of testing, diagnosis, and reconfiguration.

VLSI technology has many promising applications, including the design of special-purpose processors [7], for use as an interconnected array of processing cells on a single chip, as well as the design of supercomputers that use wafer-scale technology. These two factors, in conjunction, possess the potential for major innovations in computer architecture.

One principal aspect of such architectures is how fault tolerance can well be incorporated into such systems. Included here is the problem of the placement of redundant cells so as to achieve the elements of fault tolerance, yield enhancement, testability, and reconfigurability.

II. Fault Tolerance in VLSI and WSI

Two VLSI based areas in which important innovations are likely to occur are in the wafer-scale integrated architectures and in the single-chip/multiprocessing element ar-

Manuscript received September 28, 1984; revised August 20, 1985. This work was supported in part by AFOSR under Contract 84-005?.

I. Koren is with the Departments of Electrical Engineering and Computer Science, Technion—Israel Institute of Technology, Haifa [illegible], Israel.

D. K. Pradhan is with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, USA.

chitectures. The former has the potential for a major breakthrough with its ability to realize a complete multiprocessing system on a single wafer. This will eliminate the expensive steps required to dice the wafer into individual chips and bond their pads to external pins. In addition, internal connections between chips on the same wafer are more reliable and have a smaller propagation delay than external connections. The latter does make it possible to build a high-speed processor on a single chip, designed by interconnecting a large number of simple PEs. These architectures already have captured the imagination of several computer manufacturers and researchers alike.

As mentioned earlier, the motivation for incorporating fault tolerance (redundancy) is twofold: yield enhancement and reliability improvement. Both are achieved by restructuring the links so as to isolate the faulty element(s). Various link technologies are available now which allow such restructurability. Included among these are the laser-formed links, MOS links (tristate logic and transistors), fusible links, and so on.

Restructuring capability is either static or dynamic in type. Which type is selected depends on whether restructuring should be performed only once after manufacturing or an unlimited number of times, as may be required throughout the operational life.

The issue of fault tolerance in VLSI and WSI processing arrays has been the subject of recent studies, e.g., [8], [10], [18], [20], [26], [38], [40], [41]. In these publications, various schemes have been proposed that introduce fault tolerance into the architecture of processor arrays. Because fault tolerance is an involved subject, completely different schemes might be cost-effective in different situations and for different objective functions.

When evaluating a fault-tolerance strategy for multiprocessor systems we have to consider the following aspects:

a) types of failures to be handled and their probabilities of occurrence,
b) the costs associated with failure occurrences,
c) the applicable recovery methods,
d) the amount of additional hardware needed,
e) the system objective functions.

Fault tolerance strategies can be designed to deal with two distinct types of failures, namely, production defects and operational faults. In the current technology, a relatively large number of defects is expected when manufacturing a silicon wafer. Normally, all chips with production flaws are discarded leading to a low yield (expected percentage of good chips out of a wafer).

Operational faults (or just "faults") have, in comparison, a considerably lower probability of occurrence, the difference of which may be in orders of magnitude. Improvements in the solid-state technology and maturity of the fabrication processes have reduced the failure rate of a single component within a VLSI chip. However, the exponential increase in the component count per VLSI chip has more than offset the increase in reliability of a single component. Thus operational faults cannot be ignored although they have a substantially lower probability of occurrence compared to production defects. Consequently, a fault tolerance strategy that enables the system to continue processing, even in the presence of operational faults, can be beneficial.

The two types of failures, manufacturing defects and operational faults, also differ in the costs associated with them. Defects are tested for before the ICs are assembled into a system and, therefore, they contribute only to the production costs of the ICs. In contrast, faults occur after the system has been assembled and is already operational. Hence, their impact is on the system's operation and their damage might be substantial, especially in systems used for critical real-time applications. Clearly, a method which is cost-effective for handling defects is not necessarily cost-effective for handling operational faults, and *vice versa*.

For both types of failures in VLSI, a repair operation is impossible and the best one can do is to somehow avoid the use of the faulty part by restructuring the system. This implies that in the wafer (in the case of defects) or in the assembled system (in the case of faults) there are other operational parts which are either identical to the faulty one or that can fulfill the same tasks.

Restructuring can be static or dynamic. Static restructuring schemes are suitable only to avoid the use of parts with production flaws. Dynamic restructuring is required during the normal system operation, when faulty parts have to be restructured out of the system without human intervention. Such a dynamic strategy might be appropriate to handle defects as well. Static schemes tend to use comparatively less hardware but consume operator time, while dynamic schemes are controlled internally by the system and usually require extra circuitry.

Another aspect that has to be considered when evaluating the effectiveness of a given fault tolerance technique is the required hardware investment. The hardware added can be in the form of switching elements (e.g., [9], [38] and [37]) or redundancy in processors or communication links (e.g., [19], [26]). When carrying out such an analysis we have to take into account the following two parameters:

1) the relative hardware complexity of processors, communication links, and switching elements (if they exist),
2) the susceptibility to failures (manufacturing defects or operational faults) of all the above-mentioned elements.

Processing elements are traditionally considered the most important system resource; hence, achieving 100 percent utilization of them is many times attempted. For example, in [8], [38] and [37] switching elements are added between processors to assist in achieving this goal. In [ ] and [ ] connecting tracks are added on the wafer to be used in bypassing the defective PEs when connecting the fault-free ones. However, the silicon area that needs to be devoted to switching elements (e.g., switches capable of interconnecting 3 to 5 separate parallel buses [illegible] additional communication links [illegible] Such schemes might [illegible] substantially larger than [illegible] (e.g., [illegible]). Also [illegible] especially the longer interconnection [illegible] processors result in longer delays [illegible] the throughput of the system. [illegible] processors. The effect of this is to introduce extra stages in the pipeline thus increasing the latency of the pipeline without reducing its throughput.

In the above mentioned schemes, one of the underlying assumptions is that the extra circuitry (e.g., switching elements, communication links, or registers) are failure-free and only processors can fail. However, larger silicon areas devoted to those elements increase their susceptibility to defects or faults; as a result, the above-mentioned assumption might not be valid any more.

In VLSI, the silicon area devoted to a system element might be more important than its hardware complexity. Consequently 100-percent utilization of PEs is not necessarily the major objective, especially if this requires adding switches and/or communication links, which consume silicon real estate. In the new technology, processors will be the expendable components, as gates were in SSI or small logic networks in LSI.

This may justify different fault-tolerance schemes which do not attempt to achieve 100-percent utilization of the fault-free processors when the array is restructured to avoid the use of faulty ones [18]. Such schemes, which give up the use of some fault-free PEs upon restructuring, can be attractive for operational faults (which are few in number). Here, the lack of additional hardware (switches or links) allows a larger number of PEs to fit into the same chip area, thereby offsetting the penalty of giving up the use of fault-free PEs when restructuring.

The reported research in this area of fault-tolerant architectures, although a significant beginning, is limited in the following aspects:

a) Most of the proposed architectures have been developed on an *ad hoc* basis. No well-established criterion or framework yet exists for the formulation of these architectures.

b) As indicated above, redundancy can be used for both yield enhancement and reliability improvement. Recently, development of models to evaluate how can a given redundancy be shared to achieve the best combined improvement of yield and performance has begun [21] but more extensive work is still needed. Such models could also be used to compare and evaluate different architectures.

c) The testability and reconfigurability issues have seen very limited treatment. Algorithms for testing, diagnosis, and reconfiguration need to be developed

III A Taxonomy for Multiprocessor Architectures

Broadly, there are two types of interconnection architectures that are of interest to VLSI processor array implementation. The first type is the nearest neighbor interconnection which includes various mesh interconnections, as illustrated in Fig. 1. The second type we refer to here as algebraic graph networks which includes networks such as binary *n*-cube, cube-connected cycles, shuffle-exchange graph, shift-and-replace graph networks, and group graph networks. Examples of the latter are illustrated in Fig. 2. Like the mesh connection networks, these admit efficient execution of certain algorithms. Also algebraic structure of some of these networks can be exploited so as to realize assymptotically optimum VLSI layouts.

In order to represent uniformly different types of such

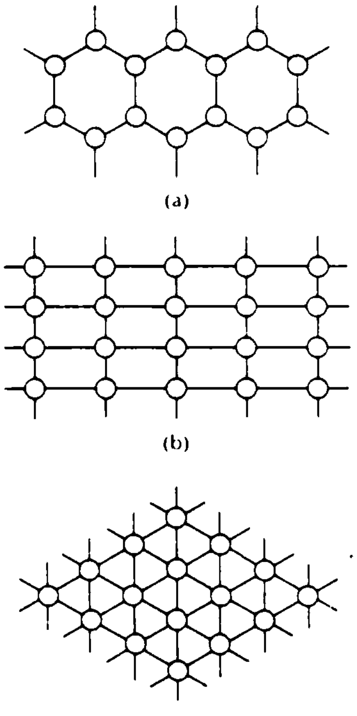
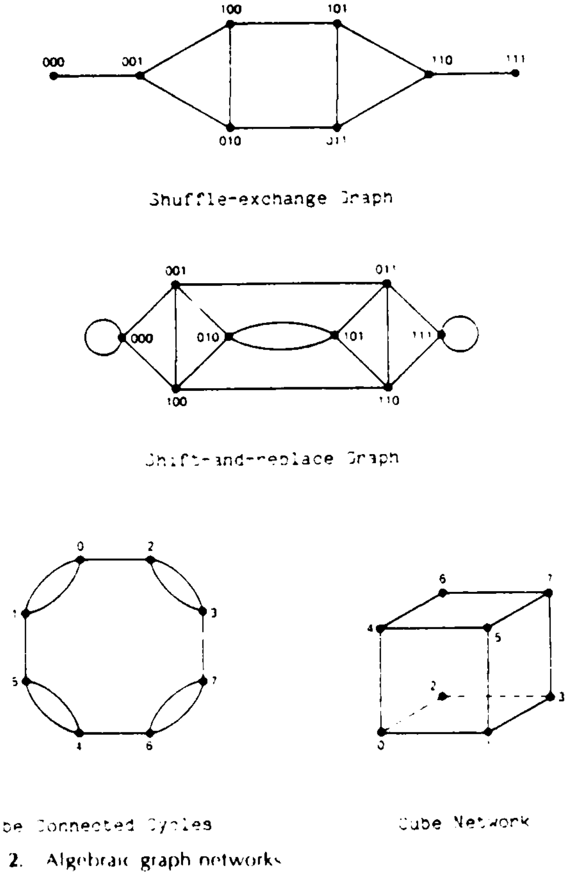


Fig. 1. Mesh connected arrays



Shuffle-exchange Graph

Shift-and-replace Graph

Cube Connected Cycles

Cube Network

Fig. 2. Algebraic graph networks

architectures, using different types of processing nodes (processors with internal switches and processors with external switches) and different types of switches (switches used for routing and switches used for fault detection and reconfiguration), we present the following taxonomy. Generally, there are two types of system nodes: nodes capable of only computation and nodes capable of both computing and switching for routing. In addition, there are two types of switches, the conventional switches, capable of only establishing connections, and fault-detecting switches, those that perform the function of both fault detection and reconfiguration. Different types of architectures are delineated in Fig. 3. The advantage, generally, in using external switches is that the computational space can be distinct from the communication space which, therefore, provides greater flexibility for emulation of a variety of communication geometries. The disadvantage of external switches, though, is that they require additional hardware support and occupy extra VLSI area.

Different types of architectures are illustrated in Fig. 3. First, Fig. 3(a) illustrates an architecture where the PEs perform internally all the switching necessary to establish connections. Fig. 3(b) represents an architecture where all the connections are established by using external switches. Such differences are best illustrated by using the following 5-tuple representation of networks. Let $N = \langle P, S, E_p, E_s, E_{ps} \rangle$ denote the network, where P represents the set of PEs, S denotes the set of switches, E_p denotes the set of direct processor–processor links, E_s denotes the set of direct switch–switch links, and E_{ps} denotes the set of processor–switch links. Different architectures can be conveniently categorized into the following four types, as shown below, where ϕ represents the null set:

Type 1:

$$\langle P, S = \phi, E_p, E_s = E_{ps} = \phi \rangle.$$

This denotes the type of architecture shown in Fig. 3(a). Here, the array contains only processing nodes with switches built in as an integral part of the processor. The mesh connections considered in [18] is an example of such an architecture.

Type 2:

$$\langle P, S, E_p = \phi, E_s, E_{ps} \rangle$$

This denotes the type of architecture shown in Fig. 3(b) where all of the configuration and communication functions are performed by switches that are external to the processor. The CHIP architecture proposed by Snyder [41] is an example of this type.

Type 3:

$$\langle P, S, E_p, E_s = \phi, E_{ps} \rangle.$$

Fig. 3(c) delineates such an architecture. Here, in addition to the external switches, each processor has an internal switch which sets up the connections between processors. The external switches are used to provide the function of fault detection through disagreement detection and subsequent switching out of the faulty processor, thus disconnecting it from the network.

Type 4:

$$\langle P, S, E_p, E_s, E_{ps} \rangle$$

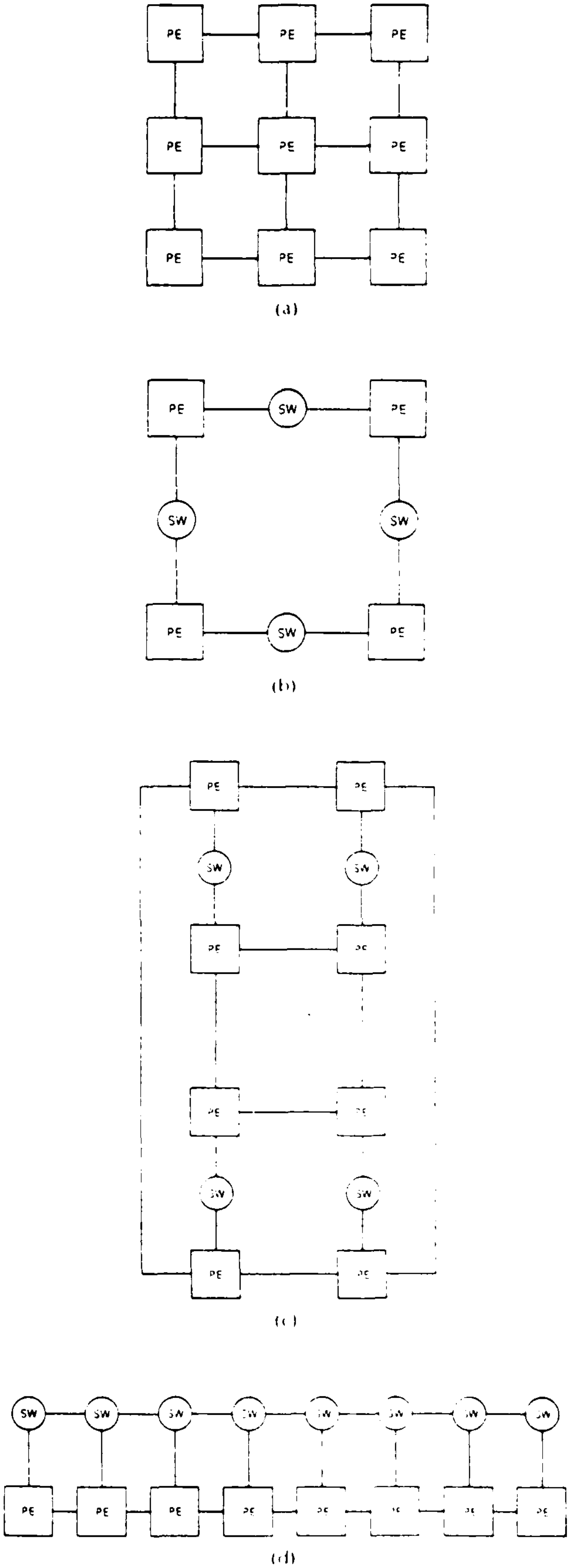


Fig. 3. (a) Type 1 architecture using internal switches. (b) Type 2 architecture using external switches. (c) Type 3 architecture. (d) Type 4 architecture.

This denotes a type of architecture where all of the different types of links are used. An example of such an architecture is illustrated in Fig. 3(d). Here a linear array of PEs is provided with external switch connections which can be configured in four ways, as shown in Fig. 4(a). The switches in such an architecture have a dual purpose. First, they can be used to provide multiple logical configurations such as binary tree in addition to the linear array; thus an application that requires both linear array and binary tree can use this architecture, as shown in Fig. 4(b). Secondly, the switches can be used to bypass the faulty elements, as shown in Fig. 4(c).

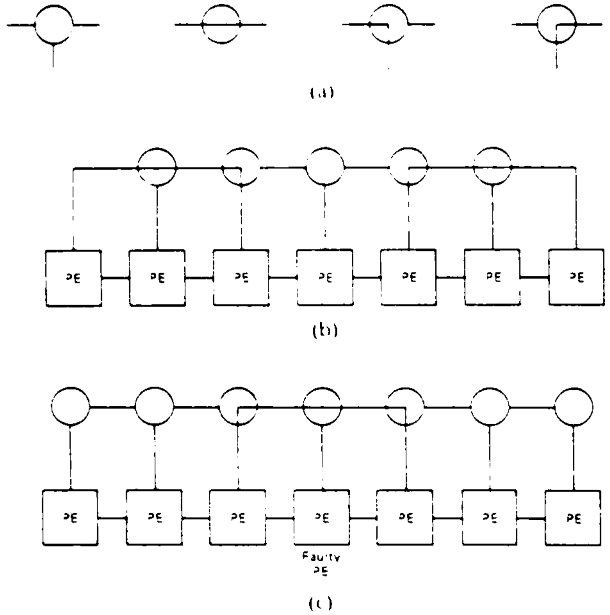


Fig. 4. (a) Different switch configurations. (b) Linear array and binary tree configurations. (c) Bypassing the faulty PE.

As we can see, these different categorizations encompass all of the different possible architectures that can be conceived. Therefore, the above taxonomy provides a convenient framework for both the analysis of different architectures and the conceptualization of new architectures.

There are two basic ways one can introduce fault tolerance into these arrays, the first approach would be to provide redundancy at each node so that the node can be reconfigured internally in the event of a fault. For example, consider a 9-node mesh connection shown in Fig. 5. If we assume that the interconnects are highly reliable, one way to design this array so that it will be fault-tolerant is to use two self-checking processors at each node, as shown in Fig. 6. The function of the external switch is to determine, in the event of a fault, which one of the two checkers is indicating errors and then to switch out the appropriate module.

However, if the interconnects cannot be assumed to be reliable, one has then to provide redundancy by designing an array larger than the maximum size required for the applications. For example, consider the 4 × 4 array shown in Fig. 7 which is designed to support various applications including the binary tree configuration shown in Fig. 8(a).

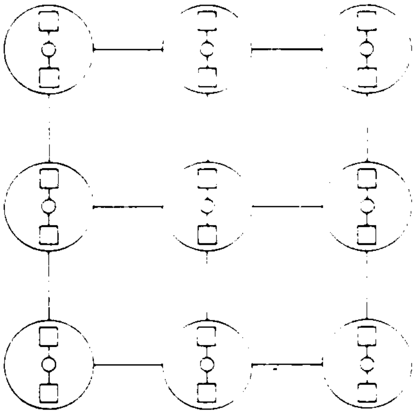


Fig. 5. A 9-node mesh connection.

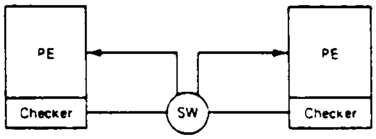


Fig. 6. Fault-tolerant node.

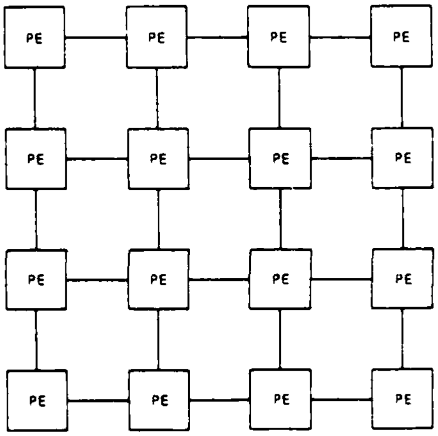


Fig. 7. A 4 × 4 mesh connection.

The mapping of the binary tree onto the array is depicted in Fig. 8(b). In this figure, the mapped nodes of the binary tree are shown, along with the inactive components, which are shown by dashed lines. Consider now that the active node 6 becomes faulty. It can be easily seen that the network can no longer admit the binary tree configuration, shown in Fig. 8(a). However, should it be possible to execute the same application on a reduced binary tree (perhaps with a degraded performance) such as the one shown in Fig. 9, the application can still be supported by the faulty array, as demonstrated below.

There are two different ways this can be achieved. First, the original 4 × 4 array can be restructured into a smaller 3 × 3 array, as shown in Fig. 10. This would require giving up the use of some processing nodes by turning them into connecting elements (CEs) [18]. Then, any application that can be executed on a 3 × 3 array can be executed on this new (logical) 3 × 3 array. The second approach would be to

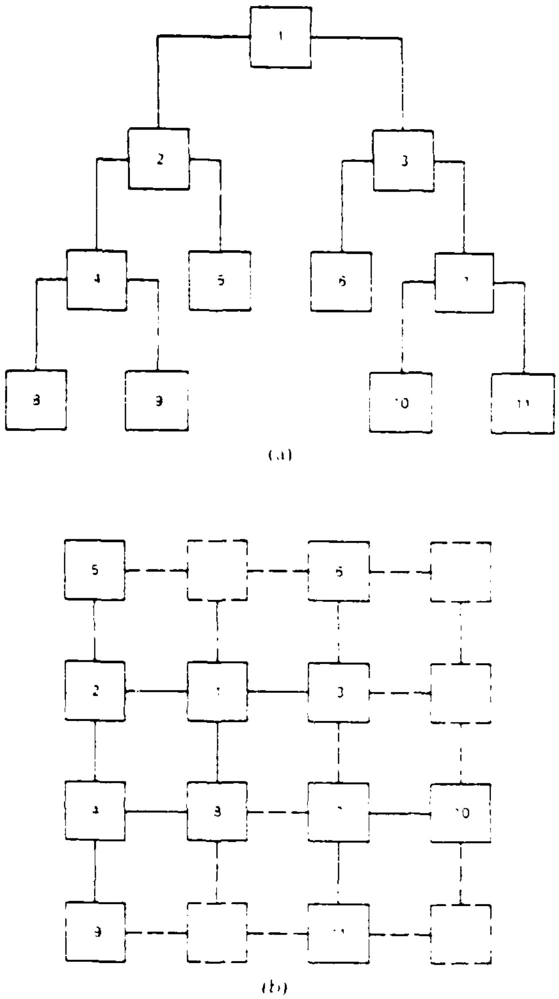


Fig. 8. (a) A binary tree configuration (b) Mapping of the binary tree onto the mesh

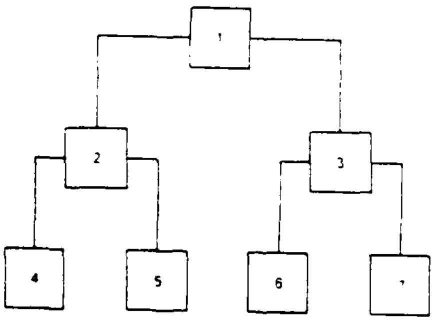


Fig. 9. Reduced binary tree

map directly the application configuration onto the faulty physical array However, the latter approach can be computationally complex [9]. Thus depending on whether or not such reduction is possible, the network may or may not be fault-tolerant, with respect to this application

Several important concepts emerge from the above discussion First, a node or link can assume several distinct states The following shows various possible states of the node:

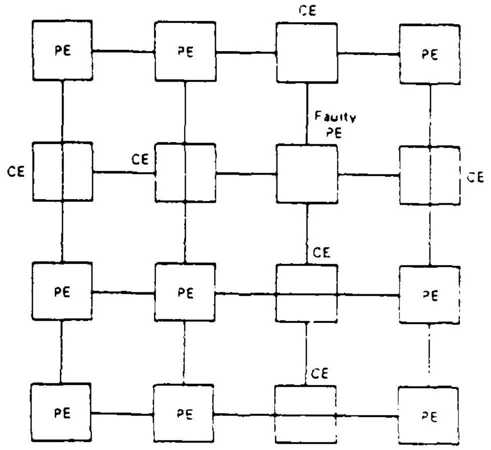
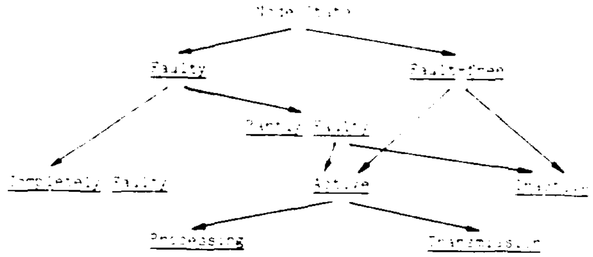
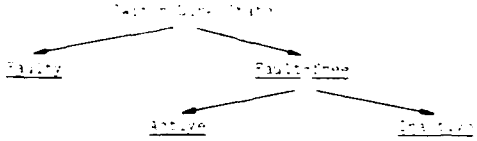


Fig. 10. Reduced 3 × 3 array

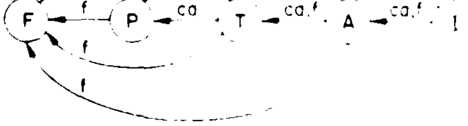


Here, the processing state of the node refers to that state in which the node is assigned to perform some useful computational task.

On the other hand, a node in the transmission state is assigned to perform only switching, so as to establish a path Thus a node in this state does not perform any computations, except those which may be required for routing, etc. For a link though, this distinction does not apply Accordingly, there are fewer states for a link, as shown below



The various possible state transitions are shown by the following directed graph Here, *F*, *P*, *T*, *A*, and *I* denote the faulty, processing, transmission, active, and inactive states, respectively The arc labels, *f* and *ca*, represent the transitions caused by fault, and change of application, respectively



Secondly, the various reconfiguration processes can be conceptualized through an abstraction of layers, formulated below.

Let the *physical layer* represent the topology which describes the interconnection structure, along with the status of the nodes and links in the physical array. A node/link in the physical layer can be either in the fault-free or faulty state.

Let an *application layer* represent that topology which is required to support a given application. Thus in this layer, all of the nodes are processing nodes; the links, active links.

Let the *logical layer* represent the topology which realizes a given application layer on a given physical layer. Thus a node in this layer is either in the processing state or in the transmission state. All of the links in the logical layer are in the active state.

For a given configuration, the above layers are related topologically, as shown in Fig. 11. The nodes in the application layer are a subset of the nodes in the corresponding logical layer and these are, in turn, a subset of the nodes in the physical layer.

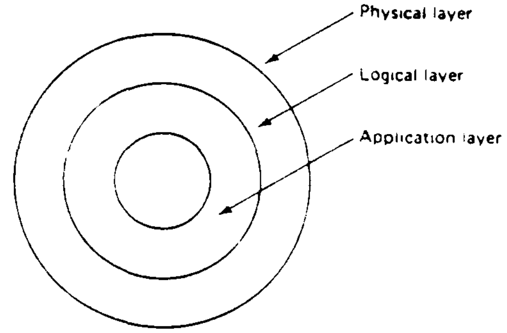


Fig. 11. Topological relationships.

The following defines a set of fundamental problems of practical importance:

Problem 1: Given an application layer (a set of application layers) and the physical array that admits these application(s), what is the minimum size (number of nodes, silicon area) of the physical layer that can admit the application(s) when t or fewer components fail?

Problem 2: Given the geometrical structure(s) of an application layer (set of application layers), how can a physical array be designed so that it can provide "efficient" fault-tolerant realization of the application(s)? The term efficient may be defined in terms of factors such as size of physical array, length of communication delay between adjacent application nodes, ease of testing and diagnosis, reconfigurability, etc.

The above problems need to be studied in the context of more general and flexible use of redundancy. For example, judicious use of node-level redundancy may offset the need for massive redundancy at the system level. Also broader use of switches as implied by Type 3 and Type 4 architectures may yield new system architectures—architectures that provide more efficient utilization of redundancy.

The above discussion is also applicable to the second type of networks, the algebraic networks. For example, consider the shift-and-replace graph networks proposed recently in [39] as a candidate for VLSI processor networks. Such an 8-node network is shown in Fig. 12(a). This network is capable of emulating various useful logical structures such as the linear array, binary tree, shuttle, and the shuffle-exchange communication structures, as shown in Fig. 12(b). More importantly, this algebraic network can emulate structures such as the linear array and binary tree, in spite of a fault. For example, consider the link connecting nodes 1 and 2 becoming faulty. In this case, the networks can still be restructured both as a linear array and as a binary tree, as shown in Fig. 13. Similarly, the network is also capable of emulating these structures in spite of any single-node failures.

It may also be noted that networks such as the binary n-cube and the cube-connected cycles provide some interesting fault-tolerant reconfiguration capabilities. For example, consider a 4-cube of 16 nodes, shown in Fig. 14(a). In the event of a fault, one can degrade this to a 3-cube of 8 nodes, as illustrated in Fig. 14(a). However, this would require giving up the use of seven good nodes. Alternatively, one can partition the 4-cube into 4 subnetworks of 2-cubes. Assuming that the problem can be divided into subproblems that can be executed on 2-cubes, one can use 3 of these, as shown in Fig. 14(b). This would necessitate giving up the use of only 3 good nodes. It is obvious that the fault tolerance of algebraic networks can be studied in the context of VLSI processor arrays.

IV. Testing and Reconfiguration Strategies

Central to the success of any fault-tolerance scheme is the formulation of effective testing and reconfiguration strategies. Basically, there are two different approaches to diagnosis and recovery: centralized and distributed. In a centralized procedure, one may assume an external unit which is responsible for initiating testing and reconfiguration. In a distributed procedure, the PEs themselves are responsible for performing periodic testing and reconfiguration.

The advantage of a centralized scheme is that no additional hardware and software support has to be provided within each PE to allow testing and reconfiguration. On the other hand, useful computation for the entire array has to be interrupted so that testing can be performed. Additionally, the complexity of the circuit and the limited access from the external unit may not allow a centralized procedure to be used. The advantage of distributed testing, on the other hand, is that since each processor can perform testing in an asynchronous mode, the testing can be interleaved with computation, thus not necessarily requiring a complete interruption of all useful computation. Moreover the distributed testing has the potential for better fault coverage because of the proximity of the testing unit and the unit under test.

From the above discussion it is apparent that a distributed procedure must strive to make the testing and reconfiguration task local to each node. This way the testing and reconfiguration can be made transparent to most of the

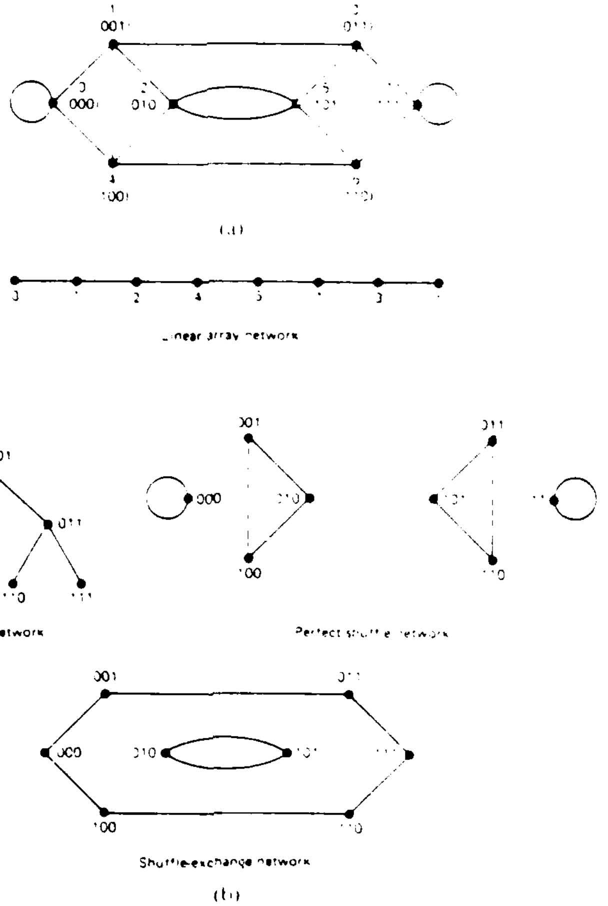


Fig. 12. (a) Shift-and-Replace graph. (b) Emulating logical structures on a Shift-and-Replace graph.

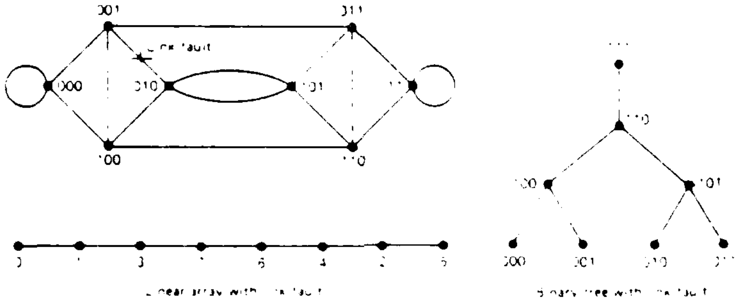
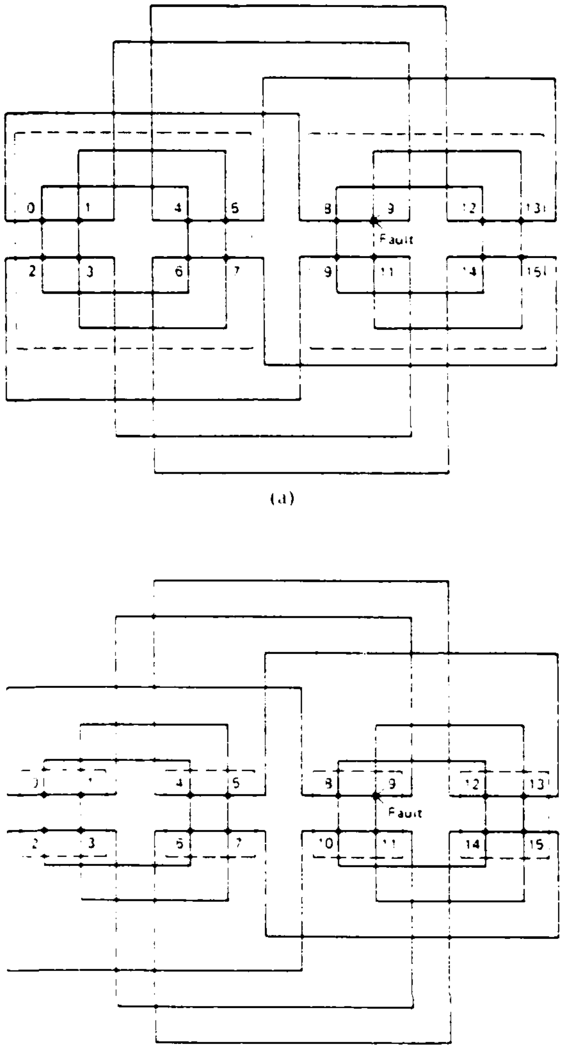


Fig. 13. Emulations in the presence of a faulty link.

network. However, performing these tasks locally requires extra hardware and software support at each node and a distributed procedure must try to minimize it. On the other hand, a centralized procedure must attempt to minimize the number of tests that will be required when no faults are present. Interruption of useful computation will be this way minimized.

In the following, we present an example for a distributed testing procedure in which every PE tests all its immediate neighbors. In this way, faulty PEs and faulty connections between PEs are detected by the adjacent PEs. The procedure first partitions all the PEs into m disjoint testing groups, $T_0, T_1, \cdots, T_{m-1}$. After this partitioning, there are m phases of testing, where at phase i ($0 \leq i \leq m-1$), the members of T_i test all their neighbors.

The partition is such that 1) every PE is surrounded by PEs of other groups, and 2) no PE has two neighbors belonging to the same group. These two properties guarantee that for every i, no two members of T_i will test each other, or try simultaneously to test a third PE. It can easily be shown that five (seven) groups are both necessary and sufficient for a partition with the above properties in the case of a square array [18] (hexagonal array [12]). The testing group numbers assigned to each PE in a square array and an hexagonal array



(a)

(b)

Fig. 14. (a) A binary 4-cube partitioned into two 3-cubes [illegible] node 9. (b) Partitioned binary 4-cube into four [illegible] faulty node 9.

[illegible] be calculated from its array indices (p, q) by $(p + [illegible])$ and $(p + 2q) \bmod 7$, respectively.

When all the m phases of the testing procedure have been completed, each and every PE knows the status ([illegible] not faulty) of all its immediate neighbors and the corresponding connecting links. There is no difference if the actual fault is in the neighboring PE proper, or in the [illegible] leading to it.

Moreover, the status of a faulty PE or link will be known [illegible] to its neighboring PEs. This locally stored information [illegible] sufficient for a distributed reconfiguration algorithm (e.g., [illegible] it will follow the testing procedure. Thus it may be [illegible] that the above distributed testing procedure does not [illegible] any passing of test results, as required in other, [illegible] general distributed diagnosis algorithms (e.g., [22]), [illegible] taking advantage of the regularity of the VLSI array.

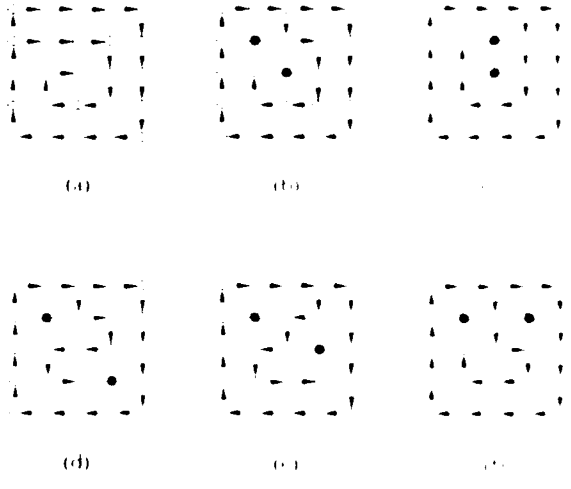
[illegible] be noted that the above algorithm will also work [illegible] simple comparison testing. In this type of testing, [illegible] no tests to be applied from one processor to the other. Simply what is required is that two neighboring processors i and j exchange the results of certain predetermined identical computation. In the event that there [illegible] mismatch, processor i can assume j is faulty and processor [illegible] can assume i is faulty.

In summary, a key feature of the above distributed testing procedure is that the testing and subsequent reconfiguration are transparent to all the nodes in the network except for those that are adjacent to the faulty node. The major disadvantage of distributed procedures [illegible] the extra hardware and software support that [illegible] to provide for testing and reconfiguration. [illegible] difficult to accomplish in processing arrays [illegible] small and simple PEs.

As discussed earlier, centralized testing [illegible] rupt all the computations in the array. [illegible] that the testing is done periodically, [illegible] number of tests and the testing time should [illegible] when there are no faults. The testing [illegible] proportionate to the number of faults [illegible] would require a minimum number of tests [illegible] of tests increasing with the number of [illegible] possible diagnosis strategy was suggested [illegible] testing very simple in the absence of [illegible] becomes progressively more time consuming [illegible] number of faults. Since most of the [illegible] present, the performance penalty [illegible] testing can be minimal. This is illustrated [illegible]

In Fig. 15 possible testing graphs for [illegible] mesh (the boundary nodes are also [illegible]. The darkened boxes represent nodes already diagnosed as being faulty. The edges with arrows [illegible] communication edges included in the testing graph. [illegible] point from the tester to the tested [illegible] [22] would require a graph with [illegible] to diagnose up to three faults. The strategy [illegible] never employs more than 25 periodic tests.

Fig. 15(a) indicates a possible initial testing graph [illegible] the end-around mesh is node symmetric [illegible] always be viewed as occurring in the [illegible] same testing graph may then be used [illegible]



(a)

(b)

(d)

(e)

Fig. 15. Various testing graphs for [illegible]

diagnosed. There must exist two adjacent fault-free rows (also columns) after no more than two faults have occurred. This ensures the graph may be covered with the faults restricted to the interior [illegible] with the border intact.

Fig. [illegible] illustrates five possible cases for the fault locations. In each instance the [illegible] a rectangular path. As [illegible] using the nodes [illegible] diagnosed [illegible] fault-free [illegible] the path [illegible] diagnosed.

[illegible] α [illegible] β [illegible]

[illegible]

defects which may occur during the manufacturing process and their distribution. The majority of fabrication defects can be classified as random spot defects [illegible] caused by minute particles deposited on the wafer. Hence each of them may affect only a single element (like a processor [illegible] of processors) [illegible]

[illegible]

may substitute [illegible] by a similar expression for [illegible] other defect distribution, e.g. Bose [illegible] The product [illegible]

[illegible]

expression [illegible] equivalent width [illegible] By comparing the [illegible]

[illegible]

be the initial state. Let a_i denote the probability of this event [2?]

$$a_i = \sum_{x=}^{\infty} Q_i^{N} \bullet \Pr\{X = x\} \qquad (8)$$

Using a_i we can calculate the yield as

$$Y = \sum a_i \qquad (9)$$

State [illegible] in Fig. [illegible] is a *terminal state* [illegible] (i.e., a state from which the only transition possible is to the system failure state F), where [illegible] is the largest number of faulty [illegible] that the system can tolerate if no redundant [illegible] were left when the system went into operation. [illegible]

[illegible] the system is in state ( ) at time t [illegible]

the system was initially in state ( )}

[illegible]

[illegible]

$x = 0$ then $R(t)$ is the reliability of the system if only perfect chips (with no defects) are accepted.

Similarly, we can define and calculate the computational availability $A(t)$ (the expected available computational capacity) and area utilization measure $U(t)$. The latter takes into account the additional area needed when fault tolerance is introduced into the system, and is defined in the following way

$$U(t) = \frac{\text{computational availability } A(t)}{\text{chip area increase } \gamma}$$

The expression for the above introduced computational availability measure is

$$A(t) = \sum c_i P_i(t) \qquad (\;)$$

where c_i is the computational capacity of the system in state (i) [ ], expressed, for example, in instructions per time unit. The computational capacity depends mainly on the number of processors available for computation in state ( ). This number is at most N [illegible] processors (where N [illegible] the number of processors in the fault-free system) and [illegible] determined by the reconfiguration strategy. In addition, [illegible] depends on the [illegible] system structure and application [illegible] processors are utilized [illegible] possible [illegible] of application.

[illegible] performance measures [illegible]

$$\int R(t)\,dt \qquad (\;)$$

[illegible]

Several problems related to testing and reconfiguration of these arrays have been described. Both the distributed and centralized modes of testing have been considered.

The last part of the paper is devoted to the presentation of analytical models for the evaluation of reliability and yield improvement through redundancy. The available redundancy on the chip or wafer is primarily limited by the size of the chip or wafer; hence it is imperative to find a method by which one can optimally share the available redundancy between yield enhancement and performance improvement. The models discussed can be used to study the effect of sharing available redundancy between these two somewhat competing requirements.

References and Bibliography

[illegible] Agrawal, [illegible] pp. [illegible] Apr. [illegible]

R. C. Aubusson and [illegible], "Wafer-scale integration: A fault-tolerant procedure," [illegible]

[illegible]

[illegible] Aug. [illegible]

[illegible] I. Koren and M. A. Breuer, "On area and yield considerations for fault-tolerant VLSI processor arrays," IEEE Trans. Comput., vol. C-[illegible], pp. [illegible], Jan. [illegible]

[illegible] I. Koren and D. K. Pradhan, "Introducing redundancy into VLSI designs for yield and performance enhancement," in Proc. [illegible]th Annu. Symp. on Fault-Tolerant Computing, pp. [illegible], June [illegible]

[illegible] J. C. Kuhl and S. M. Reddy, "Distributed fault-tolerance for large multiprocessor systems," in Proc. 7th Symp. on Computer Architecture, pp. [illegible], May [illegible]

[illegible] H. T. Kung, "The structure of parallel algorithms," in Advances in Computers, vol. [illegible], M. C. Yovits, Ed. New York: Academic Press, [illegible], pp. [illegible]

[illegible] "Why systolic arrays?" Computer, [illegible], pp. [illegible], Jan. [illegible]

[illegible] H. T. Kung and M. S. Lam, "Fault-tolerance and two-level pipelining in VLSI systolic arrays," in Proc. [illegible] Conf. on Advanced Research in VLSI (MIT), Cambridge, MA, Jan. [illegible], pp. [illegible]

[illegible] F. T. Leighton and C. E. Leiserson, "Wafer-scale integration of systolic arrays," IEEE Trans. Comput., vol. [illegible], pp. [illegible], May [illegible]

[illegible] A. D. [illegible], "Regular interconnection networks," Tech. Rep. CSD-[illegible], Dept. Comput. Sci., UCLA, [illegible]

[illegible] T. E. Mangir and A. Avizienis, [illegible] VLSI [illegible] IEEE Trans. [illegible]

[illegible] F. B. Manning, "An approach to highly integrated computer [illegible]" IEEE Trans. [illegible]

[illegible]

END

9-87

DTIC