Sādhanā, Vol. 11, Parts 1 & 2, October 1987, pp. 93–110. © Printed in India.

Fault tolerance in multiprocessor systems

NRIPENDRA N BISWAS and S SRINIVAS

Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore 560 012, India

Abstract. Multiprocessor systems which afford a high degree of parallelism are used in a variety of applications. The extremely stringent reliability requirement has made the provision of fault-tolerance an important aspect in the design of such systems. This paper presents a review of the various approaches towards tolerating hardware faults in multiprocessor systems. It emphasizes the basic concepts of fault tolerant design and the various problems to be taken care of by the designer. An indepth survey of the various models, techniques and methods for fault diagnosis is given. Further, we consider the strategies for fault-tolerance in specialized multiprocessor architectures which have the ability of dynamic reconfiguration and are suited to VLSI implementation. An analysis of the state-of-the-art is given which points out the major aspects of fault-tolerance in such architectures.

Keywords. Dynamic architecture; fault-tolerance; fault-tolerant computer architecture; multiprocessor systems; reconfiguration; system-level diagnosis; VLSI processor arrays.

1. Introduction

Fault-tolerant computing can be defined as the ability to execute specified algorithms correctly inspite of the presence of faults. The complexity of supersystems and the increasing use of such computer systems for critical applications have called for the consideration of fault-tolerance as one of the most important issues in the design of such systems.

Real-time computer systems impose the most stringent fault-tolerant requirements. A single faulty computation in such systems employed for computation-critical applications may result in the loss of human life or costly equipment. Moreover, the delay associated with fault recovery should be extremely small. Examples of real-time critical applications where fault-tolerant systems have to be utilized are in avionics computers for dynamically unstable aircraft, in spacecraft, traffic control, patient monitoring in hospitals and in anti-ballistic missile (ABM) defence applications. Other applications where fault-tolerant computers play a major role include long-life applications (e.g. unmanned spacecraft), applications requiring high availability (e.g. commercial services and telephone switching) and real-time signal processing.

From 1970 onwards, attention has been paid towards the technology of fault-tolerant computing. New theory and techniques for fault detection and error correction, fault modelling, analysis, synthesis, and architectures for fault-tolerant systems and their reliability evaluation are being developed; the ultimate aim is to design robust computers to meet the ever increasing fault-tolerance requirements in present day systems. One of the first operational computer with self-repair provisions was the JPL-STAR (Avizienis *et al* 1971). On the other side, the electronic switching system (ESS) was developed for high availability (Bell Labs 1977; Toy 1978). Two very advanced research machines were developed for commercial aircraft control under the sponsorship of NASA – the FTMP (fault-tolerant multiprocessor) (Hopkins *et al* 1978) and SIFT (software implemented fault-tolerance) (Wensley *et al* 1978). PLURIBUS (Katuski *et al* 1978), MICRONET (Wittie 1978) and TANDEM (Katzman 1978) are examples of other fault-tolerant computer systems developed for different applications.

The advent of VLSI (very large scale integration) technology has resulted in the development of multiprocessor systems consisting of an interconnection of autonomous processing elements. Such multiprocessor systems are superior to uniprocessor architectures for a variety of real-time applications since they support highly parallel computations. But at the same time, multiprocessor systems have added new dimensions to the problem of providing fault-tolerance. The large number of processors in the system increase the system's probability of failure. Correspondingly, complex mechanisms have to be incorporated for dealing with the effects of failures and providing greater system reliability. Further, the complexity of large multiprocessor systems dictate the need to develop complex fault models and methodologies for achieving the required level of fault-tolerance.

Two approaches for hardware fault-tolerance in multiprocessor systems can be outlined: (1) static redundancy techniques; (2) diagnosis techniques. Fault-tolerance through static redundancy is achieved by replicating the processors. Each processor takes the same input and feeds a voter, which votes the majority of the outputs as the output of the processors. Thus faults occurring in a certain number of processors are, in effect, masked by this method. The latter approach involves a systematic sequence in which tests are applied to locate the faulty processors with consequent isolation of the faulty processors and recovery of the processes. Depending upon the diagnosis method, the faulty processors may be replaced by spares in which case the system's throughput remains the same as before. On the other hand, spares may not be used and the system continues execution with performance degradation. These principles have been established by research over the past decade and several survey papers have appeared (Avizienis 1978; Rennels 1980, 1984; Friedman & Simoncini 1980; Avizienis & Kelly 1984; Siewiorek 1984; Kuhl & Reddy 1986) which either examine a specific aspect of fault-tolerance or give an overview of fault-tolerance techniques in a restricted class of systems. In this paper, we present a survey of the methods for obtaining fault-tolerance in multiprocessor systems. Fault-tolerance through diagnosis of faults has been an active area of research and new models, techniques and methods for diagnosis are being reported. We present a fairly in-depth and state-of-the-art survey of system-level diagnosis. Further, we also consider the various strategies for fault-tolerance in specialized multiprocessor architectures which have the ability of dynamic reconfiguration and also those which are suited for VLSI implementation.

The multiprocessor system under consideration consists of autonomous processor modules connected by an interconnection system (bus, direct links or switching networks). Each processor is equipped with a local memory which it can access by a local bus. The local memory of one processor module is accessible by other processor modules. A fault is assumed to manifest as a failure of the processors, while a fault in an interconnection facility is attributed to the failure of one or more processors which make use of that facility. Both distributed and centralized architectures are considered.

2. Static redundancy techniques

As mentioned earlier, in the context of multiprocessor systems, the utilization of extra hardware in a static redundancy scheme is at the processor level, with fault-tolerance being achieved by the replication of processors. A popular scheme used is the Triple Modular Redundancy (TMR) shown in figure 1. Here three identical processors take the same input and feed a common voter. The voter takes a majority vote to provide the correct output when a major number of processors are fault-free. It is seen that the TMR scheme can mask faults in any one of the processors.

The TMR scheme can be extended to the NMR (n-modular redundancy) scheme, where n identical processors feed a common voter. Since the voter has to take a majority decision, the number of processors n should be odd. In this case, faults in upto $(n-1)/2$ processors can be tolerated.

The main advantage of the TMR/NMR scheme is that it can mask errors instantaneously allowing programs to execute without interruption since there is no need for an error detection and fault recovery procedure. Hence it is used in systems employed for computation critical applications where even a small delay due to the occurrence of a fault can jeopardize the entire operation.

However, the scheme can be viewed as a rigid and expensive way of achieving fault-tolerance. The power consumption is considerable since all the redundant processors need to be powered. Further, the voter has to be designed to provide very high reliability, since the failure of the voter can cause system failure. To overcome this problem, a redundant voter scheme is sometimes adopted (Su & Hsieh 1982) where the voter is also replicated.

Many variations to the TMR/NMR scheme have been suggested to suit specific fault-tolerant requirements. One such scheme (Losq 1976) makes use of the

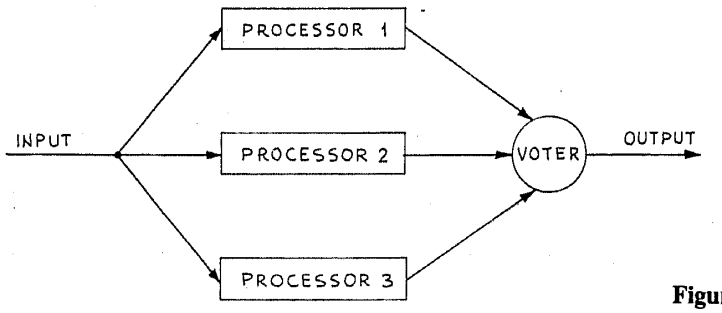


Figure 1. Triple modular redundancy.

threshold voter in place of the majority voter. The voter output is 1 only if the weighted sum of its inputs is equal to or greater than its threshold M. Thus upto M faulty processors can be tolerated. The *C.vmp* (Siewiorek *et al* 1978) utilizes the TMR scheme with bidirectional voters. Su & DuCasse (1980) present a scheme for tolerating multiple faults. In this method, a 5MR system will automatically reconfigure into a TMR system when two modules fail simultaneously, and thus it is more efficient than the ordinary 5MR scheme.

Another new technique is the (N, K) concept fault-tolerance (Krol 1986) which makes it possible to choose the ratio between memory and processor redundancy so as to minimize the total amount of hardware.

3. Diagnosis techniques

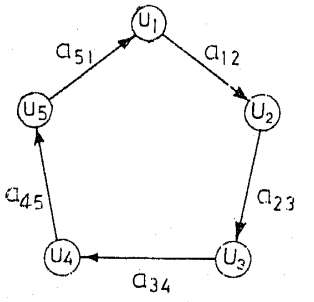
In contrast to the replication in hardware with consequent masking of faulty processors, another method of obtaining fault-tolerance in multiprocessor systems is by the automatic diagnosis of faulty units followed by system reconfiguration and a recovery of the processes, thus providing safe operation. This scheme removes the inflexibility of the static redundancy scheme inasmuch as it makes possible the repair or replacement of the faulty units or allows the system to work in a gracefully degraded fashion. But the trade-off for this advantage is the requirement of a technique for rapidly detecting and locating the faults. Almost all techniques for fault diagnosis consider the system to be partitioned into a number of subsystems, or units, and aim at unambiguously identifying malfunctioning subsystems upto a given multiplicity. What follows here is a brief survey of the theories, models and algorithms for system-level fault diagnosis. The techniques are presented with the view of any computer system in general and are applicable to multiprocessor systems, where our notion of each subsystem or unit refers to an autonomous processing element.

3.1 *System-level diagnosis*

Preparata *et al* (1967, PMC hereafter) proposed one of the first models for system diagnosis. The system is partitioned into a number of disjoint subsystems under the assumption that each subsystem or unit can be completely tested by some combination of other units. Each test so defined involves the controlled application of stimuli to the unit under test and the analysis of the ensuing responses resulting in the evaluation of the tested unit as being fault-free or faulty. The PMC model utilises a diagnostic graph in which the n units $(u_1, u_2, \ldots, u_n)$ of the system S are represented as nodes and the edges of the graph represent the connection assignment that assigns each unit to test a subset of other units. The outcome of a test in which u_i tests u_j is denoted by a_{ij}, where $a_{ij} = 1$ if unit u_i finds unit u_j faulty and $a_{ij} = 0$ otherwise. If u_i itself is faulty, a_{ij} is unreliable. Given the set of test outcomes $\{a_{ij}\}$, known as the syndrome, the problem is to identify all the faulty units in S. The PMC model gives the condition under which this is possible assuming that the system has atmost t faulty units. This has led to a measure called t-diagnosability. Further, all the t faulty units may be located under the application of a test set only once or under the application of the test set in a sequence of k steps, with some of the faulty modules being located and repaired at each step.

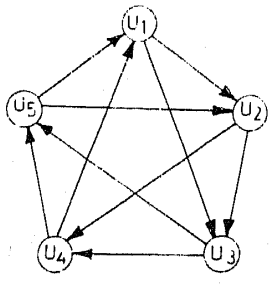
More specifically, a system is t-fault diagnosable without (with) repair if one test routine is sufficient to identify all (at least one) faulty units provided the number of such units does not exceed t. The t-fault diagnosability without repair (with repair) is also referred to as one-step diagnosability (sequential diagnosability). Preparata *et al* (1967) showed that if a system of n units is one-step t-fault diagnosable, then $n \geq 2t+1$, and each unit must be tested by at least t other units. For example, figure 2a shows a 1-fault diagnosable system. It can be verified that any single faulty unit can be located from the syndrome, but the presence of two faulty units makes the syndrome unreliable. The optimal connection assignment for a 2-fault diagnosable system is given in figure 2b. PMC also gave optimal assignments for sequential diagnosis procedures. Hakimi & Amin (1974) showed that the conditions of PMC are sufficient if no two units test each other in the system. They also give a necessary and sufficient condition for a general system (which does not have the above restriction) to be t-diagnosable. Based on the PMC model, researchers laid emphasis on three main problems of system diagnosis: (a) determination of necessary and sufficient conditions under which a system is t-fault diagnosable. In other words, this is the problem of synthesis – to determine the set of tests given a predetermined value of diagnosability; (b) determination of the diagnosability of the system given the set of tests – the problem of analysis, and (c) development of efficient algorithms for diagnosis.

In order to overcome the shortcomings of the PMC model and also to suit different environments, many generalizations were made in the graph model. Russell & Kime (1975a, b) formalize the model in terms of faults, tests and the relationships between them and represent the system of n units as $S = \{\mathcal{F}, \mathcal{T}, F, G\}$ where $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ is the set of n possible faults, $\mathcal{T} = \{t_1, t_2, \ldots, t_p\}$ is the set of p tests, $F = \{F^1, F^2, \ldots, F^{2^n}\}$ is the set of all fault patterns and G is a $2^n * p$ array called the Generalized fault table having $G_j^k = 0, 1$ or X, if for fault pattern F^K present, test t_j is known to always pass, always fail or has an unknown result. In contrast to the PMC model where each test completely checks exactly one unit [single unit per test (SUPT)] and is invalidated by exactly



FAULTY NODES	SYNDROME				
	a_{12}	a_{23}	a_{34}	a_{45}	a_{51}
U_1	X	0	0	0	1
U_1 U_2	X	X	0	0	1

(a)



(b)

Figure 2. Optimal connection assignment for (**a**) 1-fault diagnosable system (X represents unreliable output) and (**b**) 2-fault diagnosable system.

one unit [single invalidation per test (SIPT)], the Russell-Kime model relaxes these assumptions and allows for multiple units per test (MUPT) and multiple invalidations per test (MIPT). This removes the restriction placed by the PMC model on the communication paths being fault-free. (The abbreviations SUPT, SIPT etc. were introduced by Holt & Smith 1981.)

A simpler version of the PMC model, claimed to be more realistic, was introduced by Barsi *et al* (1976). The PMC model assumes that the test outcome is not predictable whenever the testing unit is faulty. This implies that if a faulty unit performs a test, a fault-free unit could be judged faulty or a faulty unit could be judged fault-free. This type of test invalidation is called symmetric invalidation. Barsi *et al* (1976) assumed that all invalidation takes the form of a correct unit being judged faulty, that is, if both the testing and the tested units are faulty, the test outcome is necessarily '1'. This is called asymmetric invalidation. In this model, the tested unit is unambiguously fault-free whenever the outcome is 0, and this leads to simpler diagnosis algorithms. The following table gives the test outcomes for both types of invalidations.

		Test outcome	
u_i	u_j	Symmetric invalidation	Asymmetric invalidation
Fault-free	Fault-free	0	0
Fault-free	Faulty	1	1
Faulty	Fault-free	0 or 1	0 or 1
Faulty	Faulty	0 or 1	1

Holt & Smith (1981) give the conditions for t-diagnosability with and without repair for systems with asymmetric invalidation. The other models for system diagnosis include the two-level model (McPherson & Kime 1979) which distinguishes the fault level at which testing is performed and the part level at which diagnosability is defined. Thus it removes the restriction that the level of replacable units be the same as the level of functional units. Kime (1979) defines a model which gives a mathematical interpretation and encompasses the previous models. A model in which propagation of faults is considered for diagnosis has been proposed recently (Huang & Chen 1986). McPherson & Kime (1984) analyse a model for fault diagnosis where immediate repair of faulty units is not assumed and thus diagnosis is performed in the presence of faults which have already been determined by previous tests. Maheshwari & Hakimi (1976) take into account the probabilistic nature of the occurrence of faults, thereby removing the assumption that all faults are equiprobable. They present necessary and sufficient conditions for a system to be probabilistically t-diagnosable.

For fault diagnosis with repair, Friedman (1975) proposes a new measure called t-out-of-s (t/s) diagnosability which assumes that some good units are also replaced. A system is t/s diagnosable if a set of $f \leq t$ faulty units can be located and repaired by replacing atmost s units. Chwa & Hakimi (1981) give a characterization of t/t diagnosable systems.

A number of efficient algorithms based on the above models and measures have been given: Meyer & Masson (1978) (one-step t-fault diagnosability with symmetric

invalidation); Smith (1979) and Butler (1981) (algorithms for *t/s* diagnosability); Ciompi & Simoncini (1979) (algorithm for *t*-fault diagnosability with repair) Meyer (1981) (algorithm for asymmetric invalidation) and Hayes (1976). Dahbura & Masson (1984) have exploited the graph theoretic properties of the graph model and have given an $O(n^{2.5})$ algorithm, which is the least complex as compared to other algorithms, for identifying faults in a *t*-diagnosable system. In an improvement of this work, they have identified a new class of systems, called self-implicating systems (Dahbura *et al* 1985). If a system is identified to be self-implicating, the diagnosis algorithm can be greatly simplified. Narasimhan & Nakajima (1986) give an algorithm for analysing the diagnosability of a system with asymmetric invalidation.

So far, the faults in the system were considered to be of the permanent type. Mallela & Masson (1978) studied the diagnosis capabilities of systems with intermittent faults. They show that in contrast to a permanent fault diagnosable system, there exists only a single type of intermittent fault diagnosable system – a *t*-intermittent fault diagnosable system both with and without repair must satisfy the same necessary and sufficient conditions. They have extended this work to include hybrid fault situations (Mallela & Masson 1980) which specifies bounded combinations of permanently faulty and intermittently faulty units in the system. Dahbura & Masson (1983) tackle the problem of intermittent faults and hybrid fault situations by a procedure called 'greedy diagnosis'. Intermittent faults are diagnosed by a comparison syndrome, which is obtained by assigning each job to two units and comparing the outcomes of the two units. For hybrid fault situations, the procedure aims at diagnosing units as faulty as soon as they satisfy certain conditions. But the diagnosis procedure may become complicated in certain cases.

3.1a *Adaptive diagnosis*: Nakajima (1981) proposed a new approach to system diagnosis called adaptive diagnosis. Instead of the normal procedure in which the test results are used to identify all the faulty units, this approach aims at identifying a fault-free unit first and then using this unit as a tester to identify all faulty units. Hakimi & Nakajima (1984) show that for systems with symmetric invalidation, a fault-free unit can be identified after the application of at the most $(2t-1)$ tests. This implies that at the most $(n-1)+(2t-1)$ tests are sufficient to identify all faulty units. An optimum adaptive algorithm has been presented for asymmetric invalidation also. The major limitation of adaptive system diagnosis is that every unit must be capable of testing every other unit. However, the number of tests required is reduced compared to conventional methods.

3.1b *Distributed diagnosis*: In most of the above methods, it was assumed that a central unit which forms the hardcore of the system executes the fault diagnosis algorithm and determines the faulty units from the set of test outcomes obtained from other units. But in large multiprocessor systems, a central unit may not be available for coordinating the fault diagnosing procedures. Even if a central facility is possible, this unit could pose a reliability bottleneck. This has led to the development of distributed diagnosis algorithms for such systems by which all the fault-free nodes can independently produce correct diagnoses of the condition of all the other units. Typically in such systems, diagnostic messages which contain information concerning test results are allowed to flow between nodes and such messages may reach non-neighbouring nodes by passing through one or more

intermediate nodes. A message passing through a faulty node may be altered or destroyed. This makes the diagnosis problem more complicated in distributed systems. Once all the fault-free nodes of the system produce correct diagnoses, they can stop interacting with the faulty nodes and thus such units are logically isolated from the system. This concept of distributed fault-tolerance was introduced by Kuhl & Reddy (1980). Diagnosis algorithms for distributed systems are given in Kuhl & Reddy (1981) and Hosseini *et al* (1984). The main feature of the algorithms is that a diagnostic message is passed in such a way as to ensure its reliability. Specifically, a node u_i will accept a diagnostic message from a neighbour u_j only if u_i is a tester of u_j and is certain that u_j is fault-free. In this way, valid diagnostic information flows backward along paths of the diagnostic graph. A diagnosability measure for distributed diagnosis is given (Hosseini *et al* 1984) and using this measure, sufficient conditions are given for a system employing the algorithm to achieve a given level of diagnosability.

Holt & Smith (1985) follow a different approach by relaxing the requirement that all good units be able to determine the location of all faults. Methods for diagnosis for repair and diagnosis for graceful degradation are considered. In the former case, identification of one faulty unit is sufficient and the diagnosis-repair cycle is repeated until the entire system is working. In the latter case, the goal is to identify some good sets of units that can remain in operation. The 'roving diagnosis' concept of Nair (1978) is useful for distributed diagnosis in which one portion of the system not performing computations at that time is utilised to diagnose another portion, while the remainder of the system continues normal operation. The processors which are diagnosed as fault-free in turn diagnose the other processors. Thus algorithm execution and system diagnosis can take place simultaneously.

3.2 *Recovery*

Following fault detection and diagnosis, the system undergoes a reconfiguration so that faulty processor nodes are purged out and replaced by spare nodes or the system continues to operate in a degraded mode. Recovery is the scheme for dealing with the damage caused by a fault (Kim 1979). All affected processes must be backed up or rolled back to a state which is fault-free. This scheme, known as backward error recovery, provides for recovery points (RP) for each process in the system. At each recovery point, all the necessary information about the current state of the process is saved. When applied to multiprocessor systems with many intercommunicating processes, the setting-up of proper recovery points poses a problem. To visualize this, consider the following example.

Figure 3 shows three communicating processes in a system. The dashed lines between processes indicate points of interprocess communication. The left brackets ([) represent the recovery points. If a fault was detected at point x in process A, only one recovery action needs to be done to back up to rA_3. If an error was detected at point y in process B, then process B has to backup to rB_3 and process A has to backup to rA_2, and not rA_3, since A communicates with B between rA_2 and rA_3. If an error at point z in process C is detected, then process C has to backup to rC_3 and process B to rB_2. This causes process A to backup to rA_1, and then process C has to backup to rC_1. Eventually, all three processes A, B and C have rolled back to their starting point. This phenomenon is called the domino effect of recovery.

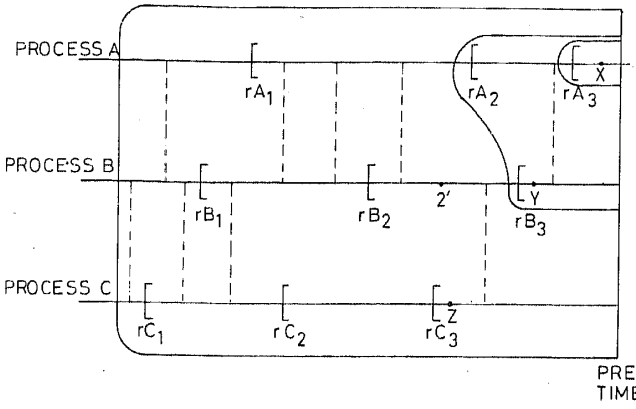


Figure 3. Recovery points assignment and the 'domino effect'.

The amount of roll-back for each of the situations depicted above is shown by the curved lines, called recovery lines. To prevent the domino effect, proper recovery points should be established. For example, if we had a recovery point at 2′ (before process B communicates with process C) then we can form a recovery line from rA_2 to rB_2' to rC_3. The selection of appropriate recovery points forms an important problem in multiprocessor systems with a large number of communicating processes.

A roll-back recovery mechanism using hardware recovery blocks has been given by Lee & Shin (1984). Each processor module consists of a number of state-save units controlled by a monitor switch. At regular intervals, each module saves its state and executes a diagnostic test. If the processor is fault-free, then the current state is considered as the recovery point for the next interval. Otherwise, the faulty processor is purged out and the associated process will roll back to one of the previously saved states. Analytical results indicate that proper partitioning and allocation of tasks are necessary to reduce the probability of multistep roll-back and the domino effect.

4. Reconfigurable architectures

The ability of a multiprocessor system to reconfigure dynamically in order to purge out the faulty nodes is one of the aspects to be considered while designing the system. In addition to enhancing fault-tolerance capabilities, reconfiguration can also be used to restructure the system to suit the specific task being executed. This type of functional reconfiguration can increase the throughput of the system by matching the architecture to the algorithm. Many reconfigurable multiprocessor/multicomputer architectures have been proposed and implemented, some of which have both functional and fault-tolerant reconfiguration capabilities (Kartashev & Kartashev 1980; Snyder 1982; Pradhan 1985b; Rucinski & Pokoski 1986), while some architectures are designed mainly to handle fault-tolerant reconfiguration (Negrini *et al* 1986; Raghavendra *et al* 1984; Pradhan & Reddy 1982; Pradhan 1985a; Clarke & Nikolaou 1982). We discuss important methods adopted to obtain reconfiguration in some of the proposed architectures. Reconfiguration aspects pertaining to VLSI array architectures will be discussed in § 5.

The reconfiguration methodology adopted for fault-tolerance may serve two purposes. In one case, the reconfiguration may be able to disconnect the faulty

processors and simultaneously bring in spare processors so that the system will continue to work with the same throughput. In the other case, spares may not be used and the system may be reconfigured so that faulty nodes are effectively removed and the connectivity of the system is not lost. Here the system will continue to work with a degraded performance. Certain reconfiguration strategies may adopt a combination of the above two types, with spares being used to replace critical processors (which have a high probability of failure), the remaining processors being designed for graceful degradation.

Two types of reconfiguration may be outlined: one is the logical reconfiguration, where no switching mechanism is employed and the processors are connected by direct links. The internode communication is established by logically routing the information so that the faulty processors are avoided. The second type is the hardware or physical reconfiguration, which makes use of a switching network to establish different connections.

Any reconfiguration method should satisfy the following requirements. First, the time for reconfiguration should be minimal. Second, the bit size of the routing code required to route the information to various nodes under faulty conditions should also be minimal. Third, fast internode communication should be possible. This implies that the switching network used for reconfiguration should not introduce a large delay. Another important measure of the effectiveness of the methodology is the number of faulty nodes that can be reconfigured out of the system without losing the system's connectivity.

Pradhan & Reddy (1982) and Pradhan (1985a) propose reconfigurable fault-tolerant multiprocessor network architectures. The context of fault-tolerance considered here is that of direct link networks designed for performance degradation with logical reconfiguration. The networks are established by algebraic properties and exploiting the algebraic structure of the network yields optimality in terms of routing distance with faults, number of connections per node and the number of faulty nodes that could be tolerated. For example (Pradhan & Reddy 1982), for a network with $n = r^m$ nodes, any two nodes i and j are connected

if $\quad i_\omega = j_{\omega-1},\ 1 \leqslant w \leqslant m-1,$

or $\quad i_\omega = j_{\omega+1},\ 0 \leqslant w \leqslant m-2,$

where $(i_{m-1} \ldots i_1 i_0)$ and $(j_{m-1} \ldots j_1 j_0)$ are the radix-r representations of i and j respectively. This network has $nr-(r^2+r)/2$ data links and can tolerate upto $(r-1)$ faulty nodes. The architectures adopt self-diagnosis for use in a distributed environment. Pradhan (1985b) has proposed a similar architecture but with the added advantage of supporting functional reconfiguration.

Raghavendra *et al* (1984) consider fault-tolerant reconfigurations in binary tree architectures. The scheme utilises one spare node per tree level and a number of redundant links which are connected by means of decoupling networks. Hardware reconfiguration is performed by setting switches in the decoupling networks by a host computer. One faulty node per level can be tolerated by this method. Another scheme for fault-tolerance with performance degradation is also given. In this case, the neighbour of a faulty node acts as a spare and makes use of redundant links for communication with the children of the faulty node. Hassan & Agarwal (1986) suggest a modular approach for fault-tolerant binary trees which uses redundant blocks. Each block consists of four nodes connected in such a manner that if any

one node goes faulty, the remaining three nodes can be restructured to form the binary subtree. This scheme removes some of the drawbacks of the method of Raghavendra *et al* (1984) by having localized switching control, less redundant links and higher reliability.

4.1 *Dynamic architectures*

An interesting class of reconfigurable multiprocessor/multicomputer parallel architectures, called dynamic computer architectures (Kartashev & Kartashev 1980), is now under development. These architectures can be reconfigured to give variable width computers so that dynamic adaptation to varying degrees of instruction and data parallelism can be achieved. Another attribute of the dynamic architecture is its capability to function as a multicomputer/multiprocessor network characterized by different topological configurations among its computers. The high degree of parallelism and adaptability afforded by the dynamic architectures makes it suitable for real-time applications, like radar signal processing (Davis *et al* 1982).

One of the widely used computing structures of the dynamic architecture class is the reconfigurable binary tree. Kartashev & Kartashev (1981) have developed an efficient reconfiguration technique for a binary tree structure organized using the Dynamic Computer Group where each tree node is an autonomous computer element (CE) consisting of a processor element (PE), memory element (ME) and I/O element (GE). With this technique, a binary tree structure with K nodes is established by providing an n-bit reconfiguration constant ($n = \log_2 K$) called bias to all the tree nodes. For this purpose, each tree node is provided with an n-bit shift register called shift register with variable bias (SRVB) (see figure 4) which stores the position code of the node. When a bias B is given, each node N generates the position code of its successor node N^* by the following operation:

$$N^* = 1[N] \oplus B, \tag{1}$$

where $1[N]$ represents a one-bit noncircular left shift of N and $\oplus$ represents mod 2 (EX-OR) addition. Thus N establishes connection with N^*. For example, figure 5a shows a configuration of a binary tree of eight nodes $(0, 1, \ldots, 7)$ when it receives bias $B = 001$. By changing the bias to $B = 010$, we get a new tree configuration as shown in figure 5b. With an n-bit bias, it is possible to generate 2^n different trees by this method.

The fast reconfiguration technique and the availability of 2^n configurations can serve as a powerful tool for enhancing the fault-tolerance of a binary tree with

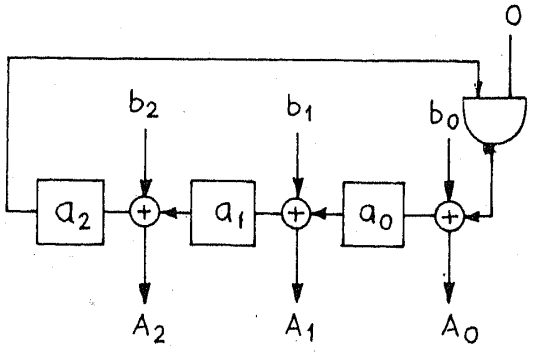


Figure 4. A 3-bit shift register with variable bias. $a_2a_1a_0$ is the position code of the node. $A_2A_1A_0$ is the position code of the successor node obtained by application of bias $b_2b_1b_0$.

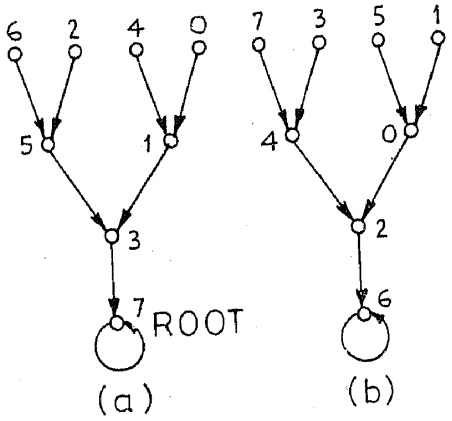


Figure 5. Two configurations of a 8-node reconfigurable binary tree structure; (**a**) for bias $B = 001$, and (**b**) for bias $B = 010$

multiple faults. Kartashev & Kartashev (1983) have suggested fast reconfiguration techniques by which all the faulty nodes in the tree can be purged out and the binary tree continues to work in a gracefully degraded fashion. Consider the 4-level binary tree with bias 14 and root 10 as shown in figure 6a. Suppose, during the course of the system operation, nodes 0,2,4,6 and 8 are found faulty. Then, by applying bias 3, the tree can be configured as shown in figure 6b, in which all the faulty nodes have been purged into the leaves positions. Now the binary tree can continue to work as a 3-level tree so that the connectivity of the fault free nodes is not lost. This type of gracefully degraded tree (GDT) is called 1-truncated GDT.

Consider a second example (figure 7a) in which we have faulty nodes in both the leaf and nonleaf positions. Now the tree can be configured as shown in figure 7b, in which all the faulty nodes have been purged out into a 2-level end subtree. This type of GDT in which the faulty nodes form an i-level end subtree is called an i-truncated GDT. Kartashev & Kartashev (1983) have shown that finding the bias for a 1-truncated GDT can be done by a single mod-2 addition (one clock period). For the case of the i-truncated GDT, the bias can be found by a sequence of $(i-1)$ mod 2 additions. Once the bias has been found, reconfiguration can be performed by (1) during the time of a single clock period.

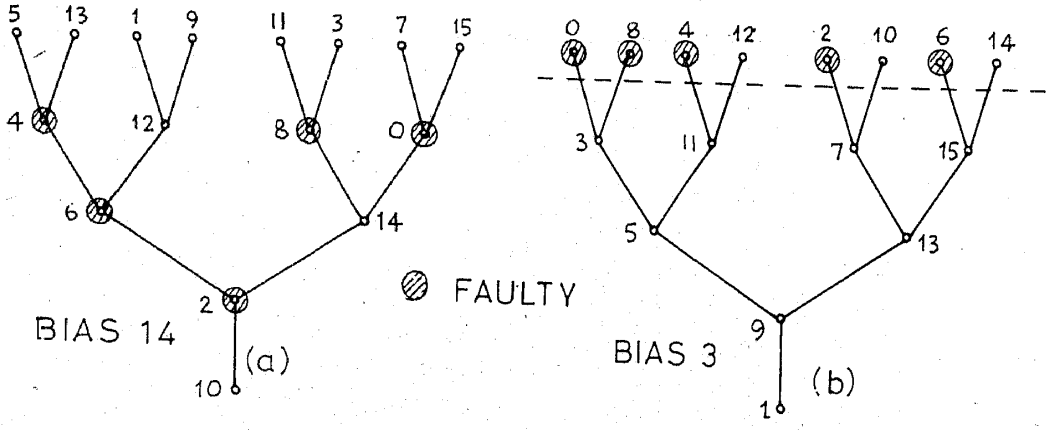


Figure 6. (**a**) A 4-level binary tree with faulty nodes. (**b**) 1-truncated GDT.

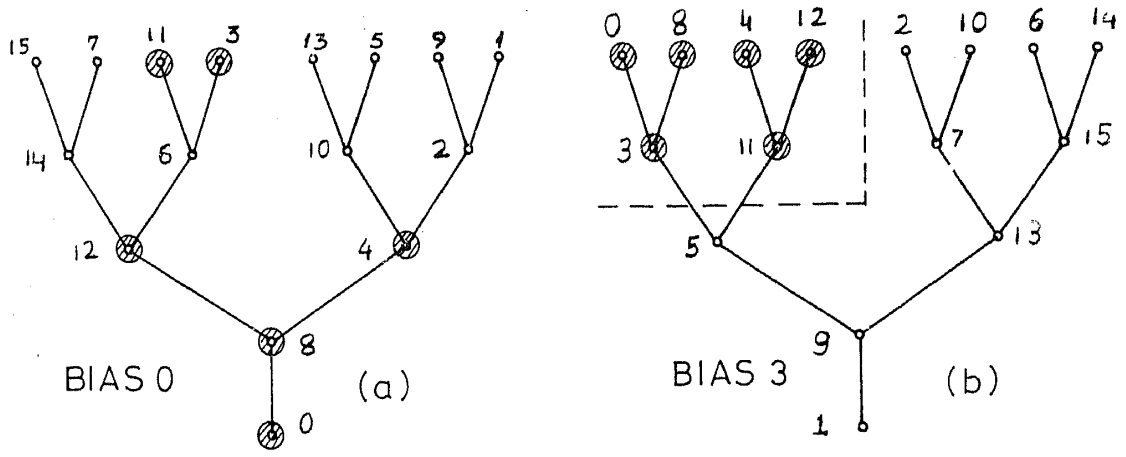


Figure 7. (a) A 4-level tree with faulty leaves and non-leaves. (b) 2-truncated GDT with a 2-level end subtree.

5. Fault-tolerant VLSI processor arrays

Fault-tolerance considerations in VLSI/WSI multiprocessor systems have received a lot of attention lately. Processor arrays which afford high parallelism are well-suited to VLSI or WSI implementation because of the regularity of their architecture and the locality of their interconnection structure. When a processor array is implemented on a single chip or wafer, the provision of fault-tolerance poses many extra problems not encountered in multichip or non wafer-scale architectures. First, the chip area should be utilised very efficiently. It has been shown (Mead & Conway 1980) that the probability of finding a fault-free circuit on chip decreases exponentially with the chip area. Hence excessive increase in chip area due to the introduction of fault-tolerance circuits may actually decrease the reliability of the system rather than enhancing it. Another consequence of an increase in area is a possible reduction in wafer yield (Koren & Breuer 1984). Hence the fault-tolerance circuits should be simple and regular and should occupy less area but, at the same time, should support a variety of fault-tolerance algorithms. The importance of maintaining a small chip area is evidenced by the fact that many fault-tolerance models (for e.g., Rosenberg 1985) measure the suitability of the design in terms of the area occupied. Second, ordinary fault models do not suffice in the VLSI environment. A physical defect, which may have occurred at production time, may render a large block of logic as faulty. Hence the fault model should be able to take care of such cluster distribution of faults also. Third, faulty processors on chip cannot be repaired or replaced. The alternative is to utilise spare processors and dynamically reconfigure the array to bring in the spares and purge out the faulty modules or allow for graceful degradation of the system. In the case of reconfiguration, the locality of the interconnections should be maintained and simple routing techniques should be adopted. Further, each processor should have self-testing circuits and it should be able to transmit its state (as faulty or fault-free) to its neighbouring processors by a single bit code.

Negrini *et al* (1986) propose a number of reconfiguration algorithms for two-dimensional VLSI processor arrays which vary in terms of the probability of

survival to a given number of processor faults and the complexity of the reconfiguration-controlling circuits. To understand the basic principles involved in such algorithms, consider the 5*5 VLSI processor array shown in figure 8a. In addition to the 16 processors active under fault-free conditions, it has an extra row and an extra column of processors. In the event of multiple faults occurring in the array (figure 8b), the reconfiguration algorithm restructures the array into the fault-free array with the faulty cells by-passed. Variations to the straightforward restructuring in the above example include the "fixed fault stealing" and "variable fault stealing" algorithms (Sami & Stefanelli 1986) which are more complex but show an increased probability of tolerance to faults. Algorithms to deal with cluster distribution of faults is given in Negrini & Stefanelli (1985). Rucinski & Pokoski (1986) propose a reconfigurable architecture for executing systolic algorithms. The grid of processors is restructured to tailor the architecture to the algorithm being executed. The upper layer of processors monitors the structure and enables the system to reorganize itself in case of faults.

Koren (1981) gives distributed algorithms for structuring arrays and trees on a grid of processors in the presence of faults. In particular, he addresses the problem of embedding a binary tree on the grid under faulty conditions. In this method, all processors in the row and the column of a faulty processor are configured as connecting elements thus isolating the faulty processor. Though the structuring algorithm is relatively simple, the technique results in many fault-free processors acting as connecting elements, and thus they are underutilized.

Many fault-tolerant array architectures (Manning 1977; Fussell & Varman 1982), in addition to the one discussed above, have internal switching mechanisms inside each processor and the processors perform all the switching necessary to establish connections. Snyder (1982) suggested the CHiP (Configurable, Highly Parallel) computer in which the switches are segregated from the PE. The CHiP architecture consists of a collection of PE, a switching lattice and a controller. Each switch contains memory which stores several configuration settings which enable it to

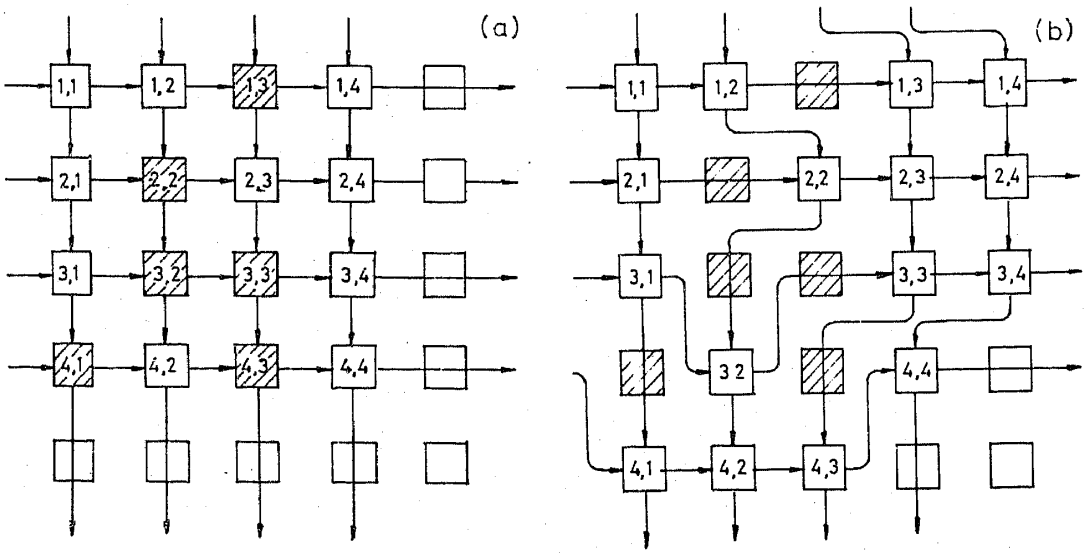


Figure 8. (a) A VLSI array with faulty processors (shaded cells are faulty). (b) Reconfiguration of (a) to the fault-free array by utilization of spare processors.

establish connections among its incident data paths. The controller loads the switch memory with the configuration codes. In the event of a faulty processor being detected, a configuration code is broadcast to route around the faulty processor. This scheme utilises the fault-free processors adequately but the reconfiguration algorithms are complex.

The Diogenes strategy (Rosenberg 1983) is a new approach for realizing testable fault-tolerant arrays with 100% utilization of fault-free processors. In this method, the processors are laid out in a line, with global busses running above the line. The connection of each processor to the bus is through switches which are controlled by control lines. For example, in the case of a linear array, only one control line ($GOOD_i$) is sufficient (see figure 9) for each processor. $GOOD_i = 1$ if the processor is fault-free, and 0 otherwise. Thus, when the array of processors is scanned, only those processors with $GOOD_i = 1$ get connected while others are just by-passed. Thus it combines the advantages of having external switches and also simple and fast dynamic reconfiguration. But the global busses have to be fault-free and may thus pose a reliability bottleneck. Rosenberg (1983) also gives layouts for a binary tree, a pyramid and a rectangular grid. All these layouts aim at linearizing the topology to utilize the principle of the Diogenes strategy.

So far, fault-tolerance with the utilization of spares has been considered. An alternative scheme is to allow for graceful degradation. Fortes & Raghavendra (1985) suggest schemes for graceful degradation of processor arrays wherein both the processor array and the algorithm in execution are simultaneously reconfigured. They have shown that any algorithm executable in a processor array can be reorganized to suit the reconfiguration properties and executed in the degraded array.

A new method for fault-tolerance in a mesh-connected processor array is the algorithm-based fault-tolerance (Huang & Abraham 1984; Bannerjee & Abraham 1986). In contrast to the reconfiguration techniques, this method aims at obtaining reliable results from computations by on-line detection and correction of faults. The algorithm is redesigned to execute encoded data and produce encoded results which can be used for fault detection. Both permanent and transient faults can be tolerated but the method is not applicable to a general computational environment. To visualize the basic principles involved in such a scheme, consider a matrix multiplication operation performed on a multiple processor system. Suppose

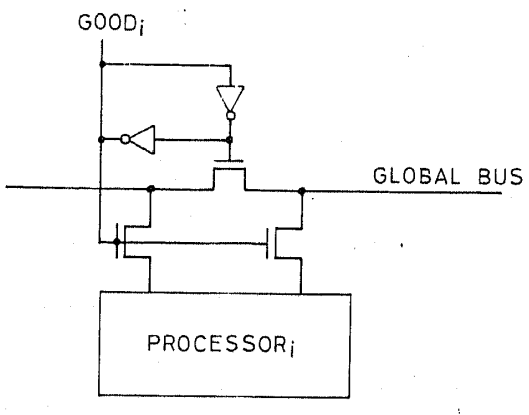


Figure 9. The processor layout in the Diogenes approach.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \text{ and } C = A*B = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}.$$

We form augmented matrices

$$A' = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}, \text{ and } B' = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix}.$$

A' has an additional row which contains the column checksum (that is, $a_{31} = a_{11}+a_{21}$; $a_{32} = a_{12}+a_{22}$) and B' has an additional column which contains the row checksum (that is, $b_{13} = b_{11}+b_{12}$; $b_{23} = b_{21}+b_{22}$). Now

$$A'*B' = C' = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}.$$

It can be verified that $c_{13} = c_{11}+c_{12}$; $c_{23} = c_{21}+c_{22}$; $c_{31} = c_{11}+c_{21}$; $c_{32} = c_{12}+c_{22}$ and $c_{33} = c_{13}+c_{23}$. In other words, the matrix multiplication operation has preserved the checksum property. The multiplication is executed on a processor array as shown in figure 10 and the results of the computation are stored in the corresponding processors (processor P_{ij} stores the result c_{ij}). Now the row checksum and column checksum are calculated and compared with the result obtained in the checksum row and the checksum column. If any single processor P_{ij} is faulty and has given an erroneous result, it will result in the checksum in the ith row and the jth column disagreeing with the calculated value. Thus the faulty processor can be located at the intersection of the ith row and the jth column and the result can be corrected (by adding the difference of the correct checksum and the obtained checksum to c_{ij}). It has been shown that the checksum property is preserved for other matrix operations like scalar product, addition, LU decomposition and transpose. Algorithm-based fault-tolerance is also being investigated for other applications like the solution of Laplace equations. The overhead required in terms of the hardware redundancy and time for checking consistency is small compared to other schemes.

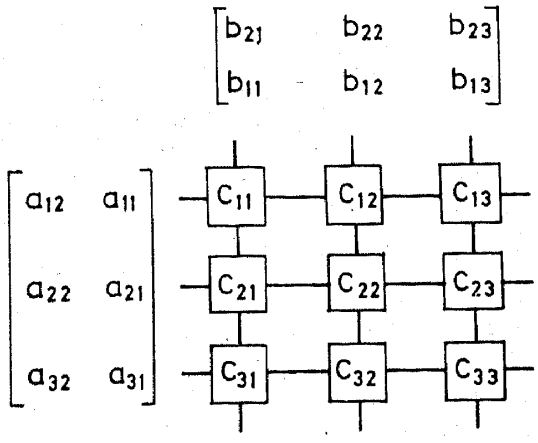


Figure 10. Checksum matrix multiplication in a mesh-connected processor array.

6. Conclusions

Though a number of techniques for achieving fault-tolerance have been and are being developed, fault-tolerant technology has continued to pose many challenges to researchers. The increasing complexity of present day computer systems and the very high reliability requirements of the applications for which they are employed have resulted in a diversified approach towards fault-tolerance. There is as yet no comprehensive and universal method for fault-tolerance in multiprocessor systems. Future research should aim at designing a multiprocessor architecture which adapts to computational and reliability requirements by exercising both functional and fault-tolerant reconfiguration. In addition, the architecture should be suitable to VLSI implementation.

References

Avizienis A 1978 *Proc. IEEE* 66: 1109–1125

Avizienis A, Gilley G C, Mathur F P, Rennels D A, Rohr J A, Rubin D K 1971 *IEEE Trans. Comput.* C-20: 1312–1321

Avizienis A, Kelly J P J 1984 *Computer* 17: 67–80

Banerjee P, Abraham J A 1986 *IEEE Trans. Comput.* C-35: 296–306

Barsi F, Grandoni F, Maestrini P 1976 *IEEE Trans. Comput.* C-25: 585–593

Bell Labs 1977 *Bell Syst. Tech. J.* 56: 1015–1331

Butler J T 1981 *IEEE Trans. Comput.* C-130: 590–596

Chwa K, Hakimi S L 1981 *IEEE Trans. Comput.* C-30: 414–422

Ciompi P, Simoncini L 1979 *IEEE Trans. Comput.* C-28: 362–365

Clarke E M, Nikolaou C N 1982 *IEEE Trans. Comput.* C-31: 771–784

Dahbura A T, Masson G M 1983a *IEEE Trans. Comput.* C-32: 777–782

Dahbura A T, Masson G M 1983b *IEEE Trans. Comput.* C-32: 953–957

Dahbura A T, Masson G M 1984 *IEEE Trans. Comput.* C-33: 486–492

Dahbura A T, Masson G M, Yang C 1985 *IEEE Trans. Comput.* C-34: 718–723

Davis C, Kartashev S P, Kartashev S I 1982 *Proc. 1982 AFIPS Conf.* (Montvale, NJ: AFIPS Press) 51: 167–185

Fortes J A B, Raghavendra C S 1985 *IEEE Trans. Comput.* C-34: 1033–1044

Friedman A D 1975 *Proc. 1975 Int. Symp. Fault-tolerant Computing* (Silver Spring, MD: IEEE Comput. Soc. Press) pp. 167–169

Friedman A D, Simoncini L 1980 *IEEE Comput.* 13: 47–53

Fussell D, Varman P 1982 *Proc. 9th Int. Symp. Comput. Architecture* (Silver Spring, MD: IEEE Comput. Soc. Press)

Hakimi S L, Amin A T 1974 *IEEE Trans. Comput.* C-23: 86–88

Hakimi S L, Nakajima K 1984 *IEEE Trans. Comput.* C-33: 234–240

Hassan A S M, Agarwal V K 1986 *IEEE Trans. Comput.* C-35: 356–361

Hayes J P 1976 *IEEE Trans. Comput.* C-25: 875–884

Holt C S, Smith J E 1981 *IEEE Trans. Comput.* C-30: 679–690

Holt C S, Smith J E 1985 *IEEE Trans. Comput.* C-34: 19–32

Hopkins A L, Smith T B, Lala J H 1978 *Proc. IEEE* 66: 1221–1239

Hossieni S H, Kuhl J G, Reddy S M 1984 *IEEE Trans. Comput.* C-33: 223–233

Huang K H, Abraham J A 1984 *IEEE Trans. Comput.* C-33: 518–528

Huang K H, Chen T 1986 *IEEE Trans. Comput.* C-35: 1082–1086

Kartashev S I, Kartashev S P 1980 *IEEE Trans. Comput.* C-29: 1114–1132

Kartashev S P, Kartashev S I 1981 *Proc. 1981 Int. Conf. on parallel processing* (Silver Spring, MD: IEEE Press) pp. 131–141

Kartashev S P, Kartashev S I 1983 *Proc. 1983 AFIPS Conf.* (Montvale, NJ: AFIPS Press) 53: 595–610

Katuski D, Elsam E S, Mann W F, Roberts E S, Robinson J G, Skowrolski F S, Wolf E W 1978 *Proc. IEEE* 66: 1146–1159

Katzman J A 1982 in *Computer structures: principles and examples* (eds) D P Siewiorek, C G Bell, A Newell (New York: McGraw Hill)
Kim K H 1979 *Proc. 1st Int. Conf. on Distributed Computer Systems* (Silver Spring, MD: IEEE Comput. Soc. Press) pp. 284–295
Kime C R 1979 *IEEE Trans. Comput.* C-28: 754–767
Krol T 1986 *IEEE Trans. Comput.* C-35: 339–349
Koren I 1981 *Proc. 8th Annual Symp. on Computer Architecture* (Silver Spring, MD: IEEE Comput. Soc. Press) pp. 425–442
Koren I, Breuer M A 1984 *IEEE Trans. Comput.* C-33: 21–27
Kuhl J G, Reddy S M 1980 *Proc. Seventh Annual Symp. on Computer Architecture* (Silver Spring, MD: IEEE Comput. Soc. Press) pp. 23–30
Kuhl J G, Reddy S M 1981 *Proc. 11th Int. Symp. on Fault-Tolerant Computing* (Silver Spring, MD: IEEE Comput. Soc. Press) pp. 100–105
Kuhl J G, Reddy S M 1986 *Computer* 19: 56–67
Lee Y, Shin K G 1984 *IEEE Trans. Comput.* C-33: 113–124
Losq T 1976 *IEEE Trans. Comput.* C-25: 569–578
Maheshwari S N, Hakimi S L 1976 *IEEE Trans. Comput.* C-25: 228–236
Mallela S, Masson G M 1978 *IEEE Trans. Comput.* C-27: 560–566
Mallela S, Masson G M 1980 *IEEE Trans. Comput.* C-29: 461–470
Manning F B 1977 *IEEE Trans. Comput.* C-26: 536–552
McPherson J A, Kime C R 1979 *IEEE Trans. Comput.* C-27: 16–27
McPherson J A, Kime C R 1984 *IEEE Trans. Comput.* C-33: 943–947
Mead C, Conway L 1980 *Introduction to VLSI systems* (Reading, MA: Addison-Wesley)
Meyer G G L 1981 *IEEE Trans. Comput.* C-30: 81–83
Meyer G G L, Masson G M 1978 *IEEE Trans. Comput.* C-27: 1059–1063
Nair R 1978 Diagnosis, self-diagnosis and roving diagnosis in distributed digital systems, TR-823, Coord. Sci. Lab. Univ. Illinois, Urbana
Nakajima K 1981 *Proc. 19th Annu. Allerton Conf. Commun. Contrib. and Comput.* (New York: IEEE Press) pp. 697–706
Narasimhan J, Nakajima K 1986 *IEEE Trans. Comput.* C-35: 1004–1008
Negrini R, Sami M, Stefanelli R 1986 *Computer* 19: 78–87
Negrini R, Stefanelli R 1985 *Proc. Int. Conf. Circuits and Systems* (New York: IEEE Press) pp. 190–196
Pradhan D K 1985a *IEEE Trans. Comput.* C-34: 33–45
Pradhan D K 1985b *IEEE Trans. Comput.* C-34: 437–447
Pradhan D K, Reddy S M 1982 *IEEE Trans. Comput.* C-31: 863–870
Preparata F, Metze G, Chien R 1967 *IEEE Trans. Electron. Comput.* EC-16: 848–854
Raghavendra C S, Avizienis A, Ercegovac M D 1984 *IEEE Trans. Comput.* C-33: 568–572
Rennels D A 1980 *IEEE Comput.* 13: 55–65
Rennels D A 1984 *IEEE Trans. Comput.* C-33: 1116–1129
Rosenberg A L 1983 *IEEE Trans. Comput.* C-32: 902–910
Rosenberg A L 1985 *IEEE Trans. Comput.* C-34: 578–584
Rucinski A, Pokoski J L 1986 *Proc. 1986 Int. Conf. Distributed Comput. Systems* (Silver Spring, MD: IEEE Comput. Soc. Press) pp. 175–182
Russell J, Kime C R 1975a *IEEE Trans. Comput.* C-24: 1078–1089
Russell J, Kime C R 1975b *IEEE Trans. Comput.* C-24: 1155–1161
Sami M, Stefanelli R 1986 *Proc. IEEE* 74: 712–722
Siewiorek D P 1984 *Computer* 17: 9–18
Siewiorek D P, Kini V, Mashburn H, McConnel S, Tsao M 1978 *Proc. IEEE* 66: 1178–1220
Smith J E 1979 *IEEE Trans. Comput.* C-28: 374–378
Snyder L 1982 *IEEE Comput.* 15: 47–56
Su S Y H, Du Casse E 1980 *IEEE Trans. Comput.* C-29: 254–257
Su S Y H, Hsieh Yu-I 1982 in *Designing and programming modern computers and systems* (eds) Kartashev S P, Kartashev S I (Englewood Cliffs, NJ: Prentice Hall) vol. 1
Toy W N 1978 *Proc. IEEE* 66: 1221–1239
Wensley J H, Lamport L, Goldberg J, Green M W, Levitt K H, Smith P M M, Shostak R E, Weinstock C B 1978 *Proc. IEEE* 66: 1240–1255
Wittie L 1978 *Simulation* 31(11): 145–153