PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

Multiprocessor Systems For High Performance High Reliability Applications

John P Hayes, John F Meyer, Kang G Shin

John P Hayes, John F Meyer, Kang G Shin, "Multiprocessor Systems For High Performance High Reliability Applications," Proc. SPIE 0880, High Speed Computing, (20 April 1988); doi: 10.1117/12.944045

SPIE.

Event: 1988 Los Angeles Symposium: O-E/LASE '88, 1988, Los Angeles, CA, United States

Downloaded From: https://www.spiedigitallibrary.org/conference-proceedings-of-spie on 27 May 2019 Terms of Use: https://www.spiedigitallibrary.org/terms-of-use

Multiprocessor systems for high performance high reliability applications *

John P. Hayes, John F. Meyer and Kang G. Shin

Dept. of Electrical Engineering and Computer Science, University of Michigan
Ann Arbor, Michigan, 48109-2122

ABSTRACT

The paper describes results of recent research on the design, testing, and evaluation of high performance, high reliabilty mutiprocessor systems at the University of Michigan. This work is aimed at the fundamental problem of realizing systems which perform extremely well and, moreover, are able to sustain satisfactory performance in the presence of faults. In our view, such work, if it is to be successful, must deal simultaneously with a variety of related issues. The presentation supports this position and, thus, covers a relatively broad range of topics. More detailed descriptions of these results are available in most instances and are referenced as part the discussion.

1 INTRODUCTION

Demands for high performance, highly reliable multiprocessor systems are increasing in a variety of application domains. Moreover, the role of such systems is often mission-critical in the sense that application objectives cannot be met if computer performance-reliability requirements are not satisfied. The design problem to be overcome is the realization of both high performance (very high processing rates, very low response times) and high reliability (fault tolerance, degradable performance) in a manner that optimizes the coexistence of these competing attributes. In the case of mission-critical systems, means of validating such systems with respect to specified performance-reliability requirements is likewise an important problem that calls for innovative methods and tools for both theoretical and experimental validation.

The discussion that follows describes aspects of recent work on these basic problems at the University of Michigan. With regard to high performance, the design techniques being explored involve architecture and algorithm considerations that permit highly parallel computation with fast I/O and interprocessor communication. To permit simultaneous achievement of high reliability, our approach is the development of techniques for error detection, fault location, and system reconfiguration that minimize performance degradation. On the testing side, both external and internal (self-testing) methods are being investigated in an effort to overcome existing problems of inadequate fault coverage, excessive testing time, and high hardware overhead. In the evaluation area, we seek improved methods and tools for evaluating a system's performance, dependability (reliability, availability), and, in general, its ability to perform in the presence of faults (performability). Evaluation studies are also an integral part of our research on design and testing techniques.

2 DESIGN

2.1 Hypercube Multiprocessors

Much of our work concerning design issues has focused on hypercube multiprocessors and related architectures [1,2,3,4,5,6,7]. A particular advantage of hypercubes is that they can support multiple users, each of which can be assigned an independent subcube of processors by the operating system. In such an environment, especially one where requests for cubes of various dimensions arrive very frequently, it is important to make judicious allocations of subcubes, so that the hypercube does not become badly fragmented. AXIS, the host operating system of the NCUBE hypercube series, for example, supports such a multiuser environment [1]. AXIS maintains a bit-vector for allocating the hypercube nodes (processors), where a 0 (1) in position i means the node with address i is free (allocated). When a k-dimensional cube is requested, AXIS searches in the bit-vector for the first integer m such that all nodes with addresses from $m \times 2^k$ to $(m+1) \times 2^k$ are free; it then allocates the k-cube composed of these nodes. This "linear" strategy recognizes only a fraction of the total number of free subcubes and does not attempt to pack the hypercube tightly.

Another allocation scheme that is well known in memory allocation and can also be used for subcube allocation is the ***buddy strategy*** [8]. We have examined properties of this strategy and have compared them with those of a new strategy using Gray Codes, referred to simply as the *GC Strategy* [4]. In this study, the buddy strategy is proved to be statically optimal in the sense that only minimal subcubes are used by the strategy to accommodate each sequence of incoming requests when processor relinquishment is not considered, i.e., static allocation. However, in the case when processor relinquishment is taken into account, the buddy strategy is shown to be poor in recognizing or detecting the availability of subcubes in the hypercube. Due to the special structure of the hypercube, the availability of some subcubes cannot be detected by the buddy strategy, and the processor utilization is thus degraded. The ability of detecting the availability of subcubes is termed as subcube recognition ability. The GC strategy was proposed to remedy the processor under-utilization problem of the buddy strategy.

In [4], performances of both the buddy and GC strategies are compared. It is shown that the GC method is also optimal for the static allocation problem. Furthermore, it is proved that subcube recognition ability is enhanced significantly by the GC strategy; the number of recognizable subcubes using the GC strategy is twice that of the buddy strategy. Note that there are many different GCs for a hypercube, each of which is associated with a set of recognizable subcubes. An allocation strategy using more than one GC can usually recognize a greater number of available subcubes (and thus improve processor utilization) than can a strategy using only one GC. The relationship between the GCs employed and their subcube recognition ability has also been derived and used to determine an allocation strategy with multiple GCs.

Although the GC strategy recognizes more subcubes than the buddy method, it does not significantly reduce the fragmentation

*This work was supported by the Office of Naval Research under Contract No. N00014-85-K-0531

Downloaded From: https://www.spiedigitallibrary.org/conference-proceedings-of-spie on 27 May 2019
Terms of Use: https://www.spiedigitallibrary.org/terms-of-use

of the hypercube. In the latter regard, we have developed new allocation and coalescing algorithms that have the goal of minimizing fragmentation [5]. The approach used here is based on identifying a class of disjoint subcubes called maximal set of subcubes (MSS), which are useful in making allocations that result in a tightly packed hypercube. The problems of allocating subcubes and of forming MSS's have been formulated as decision problems, and shown to be NP-hard. We have developed optimal algorithms for allocating subcubes and for forming an MSS. To form an MSS requires generation of the largest maximal set of disjoint prime cubes of a particular dimension, and generation of a new set of prime cubes for the free nodes that do not compose this largest maximal set. Heuristic procedures have been devised to reduce the cost of the first of these tasks. For the second task, a data structure called the consensus graph is used as a means of quickly forming the new set of prime cubes from the current set of prime cubes.

In this allocation scheme, the free nodes of the hypercube are maintained in the form of a free list of disjoint cubes, which serves as an approximation to the current MSS. When cubes are released it is important that they be coalesced with the subcubes in the free list, so as to form the biggest possible subcube. We have constructed a heuristic algorithm for coalescing released subcubes in this manner. This algorithm, coupled with a simple allocation scheme, gives a performance improvement in hit ratio of up to 55% over the buddy strategy. Though the worst-case complexity of this algorithm is exponential in the dimension of the hypercube, simulation results show that it is at most two to four times slower than the buddy method.

We have also carried out simulation studies comparing three different allocation and coalescing strategies under various distributions of subcube request dimensions and allocation times (times for which a subcube remains allocated before being released) for a Poisson arrival of requests. These results show that the algorithms based on MSS identification provide a marked improvement in performance over previously proposed hypercube allocation schemes.

With their relatively large numbers of links and nodes, hypercube computers are good candidates for constructing large fault-tolerant multiprocessor systems. To make a hypercube fault-tolerant requires a basic interprocessor communication capability that functions in the presence of faults. Most of the message-passing algorithms proposed to date for hypercubes computers break down when nodes or links fail [9]. We have designed a new scheme for near-optimal routing and broadcasting algorithms based on the systematic avoidance of certain problem nodes [3].

We first observe that an optimal routing strategy which finds a minimal "safe" path for each message, i.e., a shortest path containing no faulty nodes, can be easily achieved when each node stores complete (global) information about the faults present in the system and performs an exhaustive search of the available safe paths. However, the potentially high storage and computational costs involved make this approach unattractive. We consider instead providing each node with local information about the faults in the system. We have shown that with this approach, low-cost routing algorithms can be designed to tolerate node faults that meet a specified separation criterion.

For certain complex fault patterns, the foregoing algorithms may cause excessively long routing delays. To reduce these delays, we have introduced the concept of unsafe nodes. There are nodes in the hypercube Q_n that can cause excessive routing delay, and so are avoided during message passing. A non-faulty node is defined to be unsafe if it satisfies either of the following conditions: (1) it has more than one faulty neighbor; or (2) it has at least two faulty or unsafe neighbors. A distributed algorithm with communication complexity $O(n^3)$ has been devised to identify all unsafe nodes in Q_n. An efficient algorithm can then be given to route messages in a faulty Q_n containing no more than $\lceil n/2 \rceil$ faulty nodes. The maximum length of routing paths used by this algorithm is $p+2$, where p is the minimum safe distance from the source to the destination. If the routing path length is exactly $p+2$, then the source and destination nodes form a subcube containing only unsafe or faulty nodes; we call this an unsafe subcube.

We have obtained simulation results which show that the average percentage of the unsafe nodes present in the system is less than 15% if the number of faulty nodes is no greater than $\lceil n/2 \rceil$, for $n > 5$. This indicates that the percentage of messages which experience the worst-case $(p+2)$-hop routing delay is fairly small. We have further shown that broadcasting can be achieved under the same fault conditions with $n+1$ steps, which is only one more step than required when no faults are present.

2.2 Networks

In addition to hypercube multiprocessors, we have also been examining local area network architectures where the hosts (nodes) are computers (possibly hypercubes) which communicate via shared buses. As is the case in the work cited above, we seek designs which are able to perform well in the presence of faults. The use of multiple buses between hosts is a natural consideration in this regard. Moreover, a relatively general class of host-bus connection alternatives can be formulated via the mathematical notion of a ***balanced incomplete block design*** (BIBD) [10,11]. This is an arrangement of v distinct objects into b blocks such that each block contains exactly k objects, each object occurs in exactly r different blocks, and every pair of distinct objects a_i, a_j occurs together in exactly λ blocks.

A specific BIBD is thus determined by an instantiation of the 5-tuple (v, b, r, k, λ). These five parameters are not independent, however, due to the following basic relations.

$$bk = vr$$

$$r(k-1) = \lambda(v-1)$$

Given the values of three of the parameters, the other two are implied by the above relations. Accordingly, a BIBD is often specified by the values of v, k, and λ. The objects assigned to each block are described by its ***incidence matrix*** $A = [a_{i,j}]$, where if the objects of the BIBD are $a_i, \ldots, a_v$ and the blocks are $B_1, \ldots, B_b$ then

$$a_{ij} = \begin{cases} 1 & a_i \in B_j \\ 0 & a_i \notin B_j \end{cases}$$

Given some BIBD, its corresponding ***BIBD network*** is defined by taking the hosts of the network to be the objects of the design and by associating a bus with each block. A host is connected to a bus if and only if it is in the block corresponding to that bus. A (v, b, r, k, λ)-BIBD thus determines a network with v hosts and b buses. Each host is connected to r of the buses, each bus has k hosts connected to it, and each pair of hosts share λ buses. When $\lambda > 1$, the multiple channels between hosts can be used to increase bandwidth and provide a degradable communication structure in the presence of faults.

The class of BIBD networks is therefore quite general and includes a number of popular interconnection schemes. For example, a v-host, single-bus network such as Ethernet corresponds to a BIBD with pa-

Downloaded From: https://www.spiedigitallibrary.org/conference-proceedings-of-spie on 27 May 2019
Terms of Use: https://www.spiedigitallibrary.org/terms-of-use

rameter values $k = v$ and $b = r = \lambda = 1$. A multiple bus version of this architecture with M parallel buses, referred to in [12] as an *M-LAN*, is a BIBD network with $k = v$ and $b = r = \lambda = M$. A fully interconnected architecture with v hosts and a dedicated bus between each pair of hosts is also BIBD-determined; in this case $b = \binom{v}{2}$, $r = v - 1$, $k = 2$, and $\lambda = 1$. A hypercube, on the other hand, is not a BIBD network since the number of host pairs sharing a common bus can be either 0 or 1.

The use of BIBDs in a computer network design context is relatively new [13], although earlier work has dealt with special subclasses of BIBDs. For example, the kind of projective planes considered in [14] are equivalent to symmetric BIBDs (i.e., $v = b$; see [10]) with $\lambda = 1$. Our use of this formalism is aimed at obtaining designs which, under specified workload and fault assumptions and with respect to designated measures of performability, perform well in the presence of faults. Initial work in this regard [15] has concerned recursive methods of designing fault-tolerant BIBD networks from smaller networks which need not be fault-tolerant. An example of such a network, based on an (8,14,7,4,3)-BIBD, is shown in Fig. 1. Using stochastic activity network models, the performability of this network has been compared with that of M-LANs (see above). In each of two distinct application scenarios, the design in question performed better in the presence of faults than did M-LANs of comparable complexity.

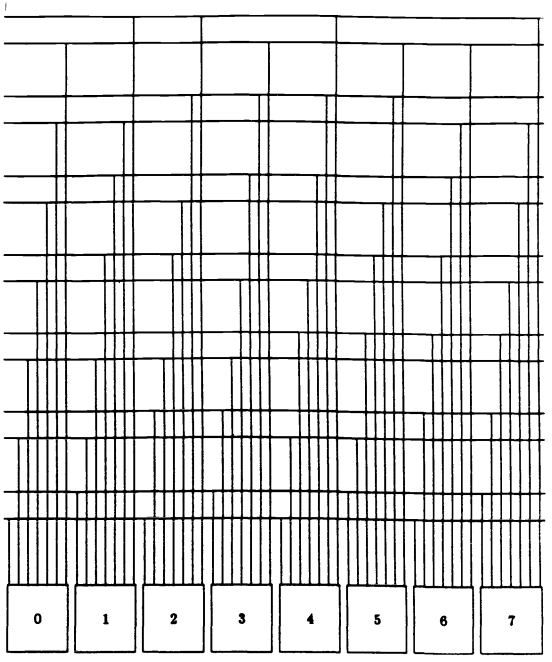

Fig. 1: An (8,14,7,4,3) - BIBD Network.

3 ERROR PROPAGATION

In any computing system, it is practically impossible to install a perfect detection mechanism with which all types of errors can always be detected before they propagate to other parts of the system. Thus, upon detection of an error, it is difficult to tell whether the error is induced by a fault that occurred in the same part of the system where the error is detected or it is the propagation of an error induced by a fault in some other part of the system.

To clarify the terminology used here, an *error* is defined as an incorrect state of the system which could be an incorrect data, an incorrect control signal, or an abnormal system behavior, and a *fault* is the source of an error, e.g., a broken wire, an electrical short, or a bug in a program. The effects of error propagation on fault location, reconfiguration and error recovery are significant, because of the uncertainty as to which components are really faulty and/or erroneous[16]. Most approaches reported in the literature circumvent the problem of error propagation by assuming a perfect coverage in detecting errors. However, such an assumption is unrealistic and, often, unacceptable for real systems, since even a near-perfect detection mechanism is very difficult to obtain without entailing excessive amount of resources or performance degradation. It is therefore necessary to consider the error propagation problem in the design and analysis of fault-tolerant systems. As a first step to meet such a need, we propose a general error propagation model and examine its power and limitations.

Error propagation was first recognized and utilized in testing combinational circuits. The D-algorithm is an example of using the error propagation property of logic gates to generate test patterns for combinational circuits [17]. Recently, error propagation properties at the logic gate level [18] and the transistor level [19] have been derived for the design of totally self-checking integrated circuits. In [20], error propagation at the register level has been considered for the design of a strongly fault secure processor. Zielinski [21] has proposed a model of error propagation among communicating processes in a distributed computer system and used it to express an error recovery method with the recovery block scheme.

All the previous error propagation models are deterministic in nature, because they are based on a specific fault/error model or the system is assumed to have some restricted, predictable behavior. However, there is, in practice, very little *a priori* information on the behavior of faults and errors, and inter-component communications may take place in an arbitrary fashion. So, error propagation cannot in general be modeled deterministically. We have proposed a stochastic model where the primary parameters — error propagation times between all pairs of system components — are random variables. We shall also comment on how the distributions of error propagation times can be determined systematically and efficiently.

We focus here only on the modeling of error propagation in a multi-module computing system. Application of the error propagation model for fault location, damage assessment and error recovery will be addressed in other papers [22].

Consider a computing system composed of multiple components, each of which is called a *module*. A module represents any well-defined component of the system; it could be a hardware or software unit or even a combination of hardware or software. Each module is a self-contained entity with input/output from/to others. A module is said to be *faulty* if a fault has occurred in the module and is said to have been *contaminated* if it contains one or more errors. For each module in the system, the *faulty moment* is the time instant a fault occurs within the module, and the *contaminating moment* is the time instant the first error occurs due to either the manifestation of a fault within the module or the propagation of error(s) from other module(s). For a module, the interval between the faulty moment and the contaminating moment is called the *fault latency* of the module.

Error propagation among the modules can best be described by a digraph, denoted by $D = (V, E)$, where N is the number of modules in the system, $V = \{v_1, \ldots, v_N\}$ is the set of nodes and $E = \{e_{ij} : 1 \leq i, j \leq N\}$ is the set of directed edges. Each node in D represents a module in the system, and the directed edge e_{ij} represents the

Downloaded From: https://www.spiedigitallibrary.org/conference-proceedings-of-spie on 27 May 2019
Terms of Use: https://www.spiedigitallibrary.org/terms-of-use

communication link from v_i to v_j. Typical means of communication between software modules are message passing or shared memory. Hardware modules can communicate via control and data signals. If there is no communication link from v_i to v_j, then E will not contain e_{ij}, or e_{ij} is a null edge. A *propagation path* from v_i to v_k, written as $(v_i, \ldots, v_k)$, is a directed path in D where all nodes in the path are *distinct* so that an error may propagate from v_i to v_k through the path. It is meaningless to consider the case of error propagation into a module which has already been contaminated; this is the very reason why the nodes in a propagation path are distinct. Errors may propagate from v_i to v_k if there exists at least one propagation path from v_i to v_k.

The *error propagation time* from v_i to v_j, denoted by X_{ij}, is defined as the time interval between the contaminating moment of v_i and that of v_j. The density function and cumulative distribution function of X_{ij} are denoted by $g_{ij}(\cdot)$ and $G_{ij}(\cdot)$, respectively. Clearly, X_{ij} holds a physical meaning only when $X_{ij} \geq 0$. Thus, the definition of X_{ij} is usually made under the assumption that v_i is the only faulty module or the first module to be contaminated in the system. If the system is assumed to have at most one fault at a given time, the faulty module will be the first module to be contaminated.

Many useful pieces of information can be derived from the g_{ij}'s and G_{ij}'s. For example, they can be used for evaluating the rollback recovery block scheme for concurrent cooperating processes[23].

Although error propagation times contain complete information on the behavior of error propagation and can be directly measured experimentally, there are several drawbacks as follows:

- It is very costly to measure error propagation times for all pairs of modules. For a system with N modules, $N(N-1)$ error propagation times must be measured experimentally regardless of the number of communication links in the system.
- The distribution of any error propagation time is fixed under a specific fault model. However, should a new fault model be needed, all error propagation times must be measured again under the new fault model, since the distributions change with the fault model.
- The error propagation times over different paths are dependent on one another whenever they have some path segments in common. Also, the dependencies among the error propagation times are very difficult to experimentally measure but necessary to compute their joint distribution. A useful joint distribution for example, is

  $$\text{Prob}[X_{i1} \leq x_1, \cdots, X_{iN} \leq x_N]$$

  which characterizes the spread of error(s) in the system from a faulty module v_i.

To overcome the above drawbacks, a *direct propagation time* is defined as follows. If e_{ij} is a non-null edge in E, then the *direct propagation time*, denoted by B_{ij}, is the time for an error to propagate from v_i to v_j via e_{ij}. The density function and the cumulative distribution function of B_{ij}, denoted by p_{ij} and P_{ij}, respectively, are called the *direct propagation functions* of e_{ij}.

The differences between an error propagation time and a direct propagation time lie in that (1) the latter is associated with a directed edge while the former is defined for every ordered pair of modules, and (2) the latter accounts for error propagation through a particular edge while the former is the minimum propagation time over all propagation paths between the given pair of modules. We showed in [24] how to systematically and efficiently compute error propagation times from direct propagation times. Since direct propagation times are defined for the communication links in the system, without loss of generality, one can assume that they are independent of one another and their distributions will not change with the underlying fault models. Moreover, this assumption greatly reduces the experimental cost to measure propagation times. For example, a five-node-eight-edge system would require 20 measurements of error propagation times for each fault model, but would require only 8 measurements of direct propagation times for all fault models.

From the direct propagation functions, another useful function called the *error containment function*, $EC_i(t)$, is defined for a module v_i as the probability that errors have not propagated from v_i to other modules up to time t (measuring from the v_i's contaminating moment). For example, if v_i has outgoing communications only with v_j, v_k, and v_m, then $EC_i(t)$ can be calculated as

$$\begin{aligned} EC_i(t) &= \text{Prob}[B_{ij} > t, B_{ik} > t, B_{im} > t] \\ &= (1 - P_{ij}(t))\,(1 - P_{ik}(t))\,(1 - P_{im}(t)) \end{aligned}$$

because B_{ij}, B_{ik}, and B_{im} are independent of one another.

4 SELF-TESTING VLSI CIRCUITS

Despite the substantial research effort devoted to digital system testing over the past two decades, it continues to be one of the most difficult and costly problems facing industry today. Rapid developments in VLSI technology, as well as the increasing use of microprocessors, have rendered many standard testing techniques obsolete. The problems of testing VLSI circuits are mainly due to the large number of possible failures, and the difficulty of obtaining access to internal circuits. They result in excessive testing time, costly test generation and application procedures, and a lower than desirable percentage of detected faults (fault coverage). Self-testing or built-in testing attempts to place the main testing processes, including test pattern generation and response verification, in the system under test, and is an important way of improving the testability of VLSI circuits [25].

Self-testing techniques can be grouped under two broad headings: concurrent methods where testing is performed concurrently with normal operation, and non-concurrent methods where testing is done off-line in a special test mode of operation. Concurrent self-testing is represented by the use of error-detecting codes. The fault coverage of error-detecting codes is less than 100%, and hard to measure. Moreover, such codes are applicable only to a few types of computer components. Non-concurrent self-testing, which is the focus of our research, has the advantage of allowing high levels of fault coverage to be achieved, possibly at the expense of high circuit overhead and slow detection of faults.

It has long been recognized that iterative circuits composed of identical subcircuits (cells) with regular interconnection structures are easily testable. For instance, an n-bit ripple-carry adder composed of n full-adder cells, can be tested for all faults within a single cell using just 8 test patterns. These input patterns exhaustively test each 3-input cell, while ensuring that the test results are propagated to the observable outputs. The property of being able to detect all single-cell faults with a constant number of test patterns independent of the array size n is termed C-testability. A related testability property that is especially suited to selftesting is I-testability which requires all single-cell faults in an n-cell array to be detectable by a set of tests (I-tests) that produce identical responses from all cells.

Downloaded From: https://www.spiedigitallibrary.org/conference-proceedings-of-spie on 27 May 2019
Terms of Use: https://www.spiedigitallibrary.org/terms-of-use

An n-cell array is CI-testable if it is both C- and I-testable General iterative arrays, including the ripple-carry adder, can readily be modified to make them I-testable or CI-testable [26].

We have recently developed a novel approach to nonconcurrent self-testing design which attempts to encompass a broad range of computer components, from arithmetic-logic units to main memories [27,28]. It is based on the observation that VLSI circuits exhibit some degree of regularity, e.g., due to the use of a small set of cells as building blocks and the need for simple interconnections. Most large circuits contain enough irregularities to make them difficult to test. However, it may be possible to "regularize" them by introducing small amounts of special control logic to reconfigure them as iterative arrays during testing. The reconfigured circuits then have the appearance of replicated circuits, but with a much lower overhead in extra circuitry. Their array-like structure permits the CI-testing concepts mentioned above to be applied, yielding high fault coverage with relatively short testing times.

The proposed design methodology is applied to a given moderately irregular circuit C in several steps illustrated in Fig. 2. First C is partitioned into a set of similar subcircuits (partitions) π_1, π_2, ..., π_n which may correspond to cells or bit slices in original design. Next the π_i's are made approximately identical (regularization) by systematically adding redundant data and control logic, so that the final circuit (Fig. 2) is essentially an iterative logic array. The added control inputs permit the redundancy, and hence the regularity, to be disabled during normal operation, and to be enabled only during testing. Other input control circuits are introduced to allow a common set of test patterns to be applied to all π_i's simultaneously. Thus the size of the test set and the corresponding test time are determined by the partition size rather than the overall circuit size.

The main design problem in the foregoing process is the regularization step, which varies from trivial in the case of pure iterative circuits like a ripple-carry adder, to intractable in the case of randomly-structured logic circuits. There appears, however to be a fairly large class of ***nearly regular*** iterative circuits which can usefully be regularized via heuristic techniques [27]. We illustrate this for the case of a dynamic random-access memory (DRAM) which is a key component in computer design.

DRAM's are composed of very large regular arrays of storage cells, with a small amount of control logic. The latter includes on-chip refreshing logic to prevent the loss of data due to the stored-charge leakage effects that are characteristic of dynamic memories. Although they are quite regular, large RAM chips are normally quite hard to test. For example, the traditional RAM test procedure GALPAT may require several days to test a 1-Mbit RAM. Moreover, the fault coverage of such heuristic procedures drops sharply when complex failure modes like pattern sensitivity are taken into account.

A typical DRAM comprises addressing logic, data storage arrays, control and timing logic, and data input/output circuits, as illustrated by the unshaded part of the design of Fig. 3 [29]. To minimize the layout area, the storage cells, address decoders, and sense amplifiers are organized into several large two-dimensional iterative arrays. The remaining peripheral circuits may be irregular, but occupy only a small fraction of the chip area. The partitioning procedure discussed above applies only to the regular circuits; the rest of the DRAM can be regularized by duplication. A partition is therefore taken to be a set of one or more storage-cell arrays with their common address decoding logic and sense amplifiers. In the representative DRAM shown in Fig 3, there are two identical partitions, each containing a pair of storage subarrays and a sense amplifier array.

A standard one-transistor design is assumed for the DRAM storage cell. The test patterns for the cell are derived from analysis of its major failure modes, such as neighborhood interference, bit-line imbalance, and transmission-line effects [30]. The tests for an entire partition are obtained by systematically superimposing the test pattern sequences required by the individual cells. Many cells in the partitions may be tested concurrently, in order to minimize the overall testing time for the DRAM. The tests for each partition require the following sequence of operations: (1) write a background data pattern for the set of cells S_τ under test; (2) apply an excitation sequence of write and read operations to S_τ; (3) read and verify the contents of S_τ; (4) modify the background data patterns for testing another set of cells $S_{\tau+1}$. These four operations must be performed repeatedly until all the cells of the partition have been tested.

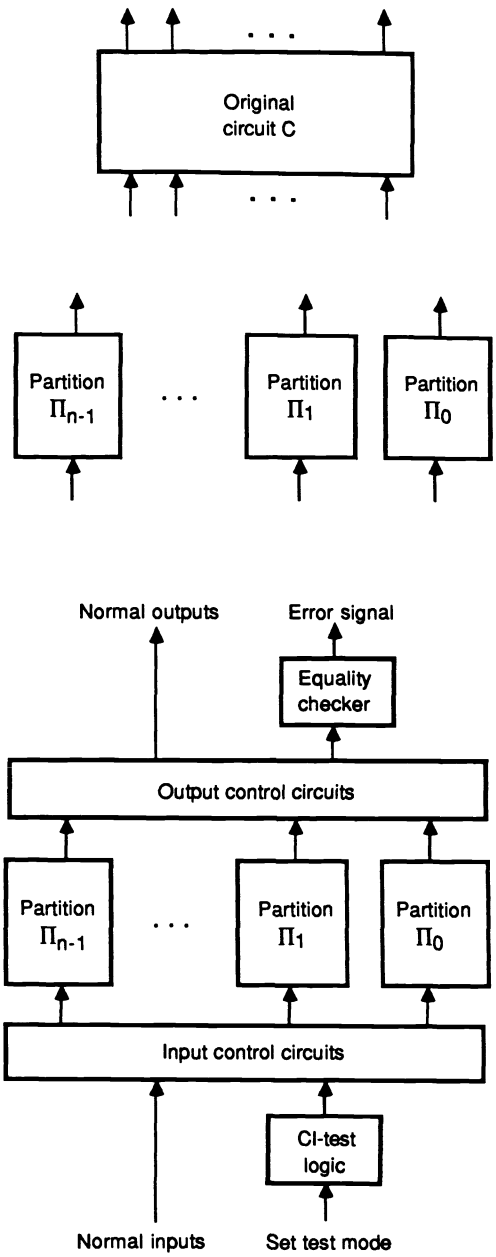

Fig. 2: A self-testing bit-sliced system based on CI-testability

Downloaded From: https://www.spiedigitallibrary.org/conference-proceedings-of-spie on 27 May 2019
Terms of Use: https://www.spiedigitallibrary.org/terms-of-use

The foregoing write, read and modify steps can be implemented by a single on-chip shift operation, which employs the original refresh logic and a slightly modified sense amplifier array. When a word line in a $p \times q$-bit storage array is activated by a row address, the signals at the q-cell row are transferred to bit lines and sensed by q sense amplifiers. In normal operation, the contents of the q sense amplifiers are then returned to the original cells. In the modified design, however, the sensed data is sent to, and stored in, the adjacent cells of the selected row during testing. By saving the data shifted out from the rightmost sense amplifier and returning it to the leftmost cell in the next shifting operation, the entire $p \times q$-bit cell array becomes a pq-bit circular shift register. A circular shift produces a new test pattern for each partition. $2q + 1$ shift operations suffice to produce the test patterns necessary to test all the cells in the partition.

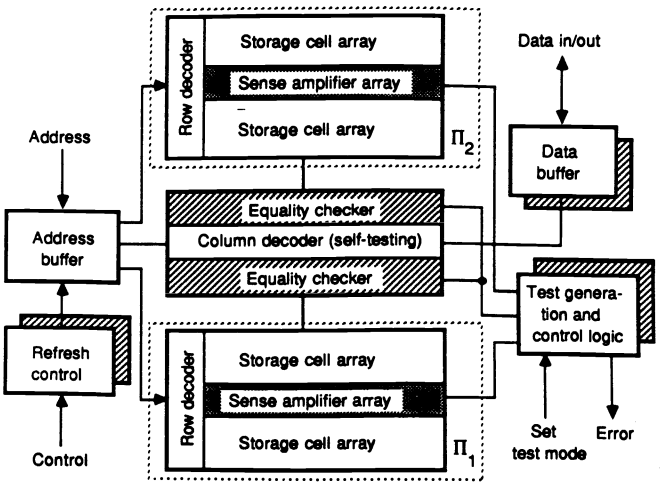

Fig. 3: A self-testing dynamic RAM chip.

As in the general case of Fig. 3, the DRAM partitions are tested in parallel, and a duplicated equality checker is used to compare the sense amplifier outputs from the two cell partitions; this comparison is done during the shift operation, and two entire rows are checked at a time. This significantly reduces the testing time. In the 64-Kbit DRAM case, for example, $q = 256$, so 256 cells are tested simultaneously per partition. Conventional external testing methods can only test one DRAM cell at a time. The peripheral circuits, test generator, the refresh and timing control logic, and the data buffers are made self-testing by duplication. Alternatively, parity checking may be used to detect errors in the peripheral circuits.

An important feature of our self-testing DRAM design is that the testing time need not increase with the memory size, if the number of partitions rather than the partition size in increased. The circuit overhead, which is indicated by shading in Fig. 3, is mainly limited to the sense amplifier modifications and the equality checkers. Clearly this overhead decreases as the number of cells increases. We expect about 5% circuit overhead in a 1-Mbit DRAM, which is lower than other self-testing methods, such as the more than 20% overhead found in [31]. Since our test patterns are derived from the major physical failure modes, higher fault coverage than standard test procedures such as GALPAT or error-detecting codes can be expected.

5 EVALUATION

Much of our effort in the evaluation area concerns the various techniques we are exploring to improve multiprocessor system performability, including specific aspects of both performance and reliability. Indeed, evaluation studies, of either a model-based or experimental nature, have accompanied most of our design-oriented investigations. Such studies have also addressed techniques developed by others, e.g., in examining the the influence of workload on transient fault recovery in random access memories [32], we evaluated a self-exercising, self-checking memory design proposed by Rennels and Chau [33].

Another significant part of this effort concerns evaluation methods and tools, per se, which are suited to the performability evaluation of distributed real-time systems. Such methods/tools can assist comparative evaluations of design alternatives (as in the specific studies cited above) or, when performability requirements are specified, can serve as a means of system validation. In this regard, we are continuing to explore the utility of stochastic activity network (SAN) models [34,35,36] and a SAN-based performability evaluation tool called METASAN† [37]. We are also concerned with modeling methods [38] which permit more definitive evaluations of how workload affects dependability and, more generally, performability.

In the case of dependability, prior studies of workload influence have, for the most part, dealt with specific systems and have been based on measurement or simulation data. Although studies such as this will continue to be useful, there is need to gain a more fundamental understanding of how workload, through its interaction with faults, relates to system failure. Our approach in the latter regard is analytical and is based on a general type of stochastic model referred to as a *Markov renewal process* (see [39], for example). In the current version of the model [38], the faults considered are transient (i.e., temporary external faults) which, in turn, result in dormant, soft internal faults. Such faults can accumulate in the system and be activated by computational demands or internal exercising. An error occurs when a dormant fault in the system is activated. Depending on the severity of the faults and the source of activation, a soft fault can be corrected, remain in the system, or cause an error that corresponds to system failure, i.e., the system fails to perform as required.

The workload we consider consists of arrivals of various types of tasks which have different processing requirements. The relationship between workload and the system is such that for each type of task, say type i, there corresponds a threshold value m_i. If the total number of (dormant) faults in the system does not exceed m_i, an arrival of a task of type i activates and corrects any existing faults in the system (via fault tolerance mechanisms provided by the system) and brings the system back to the fault-free state. On the other hand, if the total number of faults in the system exceeds m_i, the service request of a type i task cannot be met; accordingly, an arrival of such a task causes the system to fail. This threshold value m_i is the *fault margin* associated with type i tasks. The set of fault margins characterizes the fault tolerance of the system.

In many cases fault margins will be small integers indicating limited fault tolerance with respect to the processing of such tasks. In certain cases, however, it is possible for a fault margin to be infinite, i.e., a task of this type is impervious to an arbitrary number of faults. For example, a write operation in a memory system corrects all existing dormant faults at the accessed location and hence, under the above assumptions, can never cause a system failure. Workload in such systems may play opposing roles depending on the number of accumulated dormant faults. On the one hand, it can favor

† METASAN is a Trademark of the Industrial Technology Institute.

Downloaded From: https://www.spiedigitallibrary.org/conference-proceedings-of-spie on 27 May 2019
Terms of Use: https://www.spiedigitallibrary.org/terms-of-use

fault tolerance by detecting dormant faults and triggering correction mechanisms. On the other hand, when faults have accumulated in the system to an extent that exceeds the fault margin of some task type, then an arrival of such a task causes the system to fail.

Under the above assumptions, states of the system can be represented by the set $E = \{0, 1, ..., m\}$, where m is determined by the set of fault margin values $\{m_i\}$, i.e., $m = \bar{m} + 2$, where $\bar{m} = \max \{m_i \mid m_i < \infty\}$. Thus an input task type having the largest finite fault margin determines the size of the state space. For $i = 0, 1, \ldots, m-2$, state i means there are i faults the system; state $m-1$ means there are at least $m-1$ faults in the system; state m corresponds to system failure. State transitions are caused by "events" where, in the current version of the model, an event is either an arrival of a task or an occurrence of a transient fault. The resulting model is a Markov renewal process. Moreover, the state behavior of this process is influenced by both workload (task arrivals) and faults in a manner that can be determined analytically.

In particular, if dependability is the issue and the variable in question is T_f, the "time to system failure", then one can derive a general expression for the Laplace-Stieltjes transformation of the probability distribution function (PDF) of T_f. Moreover, in more special cases, e.g., when interarrival times between input tasks and between faults are exponentially distributed, we've been able to obtain closed-form, time-domain solutions of the PDF of T_f. Formulations of the mean and variance of T_f have likewise been derived and, for certain instances of the model, we've shown that, to a close first approximation, T_f is exponentially distributed. This result is significant since, as demonstrated in [38], it permits larger systems, comprised of subsystems that fail in this manner, to be evaluated without having to deal with an exceedingly large state space. Possible extensions of the basic renewal process model are currently under investigation; these, in turn, should yield additional results of the type described above.

References

[1] J.P. Hayes, T.N. Mudge, Q.F. Stout, S. Colley and J. Palmer, "A microprocessor-based hypercube computer," *IEEE Micro,* vol. 6, no. 5, pp 6-17, October 1986.

[2] M.S. Chen and K.G. Shin, "Embedding of interacting task modules into a hypercube multiprocessor," in *Hypercube Multiprocessors 1987,* pp. 122-129, SIAM, Philadelphia, 1987.

[3] T.C. Lee and J.P. Hayes, "Routing and broadcasting in faulty hypercube computers," *Third Conf. on Hypercube Concurrent Computers and Applications,* 1988, to appear.

[4] M.S. Chen, and K.G. Shin, " Processor allocation in an N-cube multiprocessor using Gray codes," in *IEEE Trans. on Comput.,* vol. C-36, no. 12, pp. 1396-1407, December 1987.

[5] S. Dutt and J.P. Hayes, "On allocating subcubes in a hypercube multiprocessor," *Third Conf. on Hypercube Concurrent Computers and Applications,* 1988,W to appear.

[6] F. Harary, J. Hayes and H.-J. Wu, "A survey of the theory of hypercube graphs," *Computer and Math. with Applications,* 1988, to appear.

[7] T.N. Mudge, J.P. Hayes, and D.C. Winsor, "Multiple bus architectures," *Computer,* vol. 20, no. 6, pp. 42-48, June 1987.

[8] K.C. Knowlton, "A fast storage allocator," in *Commun. of ACM*, vol. 8, no. 10, pp. 623-625, October 1965.

[9] Y. Saad and M.H. Schultz, "Data communication in hypercubes," Tech. Rept. YALEU/DC5/RR-389, Department of Computer Science, Yale University, June 1985.

[10] M. Hall, Jr., *Combinatorial Theory*, Blaisdell, 1967.

[11] R. Mathon and A. Rosa, "Tables of parameters of BIBDs with $r \leq 41$ including existence, enumeration, and resolvability results," *Annals of Discrete Mathematics*, vol. 26, pp. 275-308, 1985.

[12] M. Ajmone Marsan, "Multichannel local area networks," in *Proc. COMPCON Fall '82*, Washington, D.C., September 20-22, 1982, pp. 493-502.

[13] J. Opatrny, N. Srinivasan, and V. S. Alagar, "Construction of large fault-tolerant communication network models," in *Proc. 16th Int'l Symp. on Fault-Tolerant Comput.*, Vienna, Austria, July 1986, pp. 110-116.

[14] M. D. Mickunas, "Using projective geometry to design bus connection networks," in *Proc. Workshop on Interconnection Networks*, West Lafayette, IN, April 21-22, 1980, pp. 47-55.

[15] B.E. Aupperle and J.F. Meyer, "Fault-tolerant BIBD networks," Comput. Res. Lab., Univ. of Michigan, Ann Arbor, MI, Tech. Rep. CRL-TR-14-87, December 1987 (submitted to *18th Int'l Symp. on Fault-Tolerant Comput.*).

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

[17] J.P. Roth, W.G. Bouricius, and P.R. Schneider, "Programmed algorithms to compute tests to detect between failures in logic circuits," in *IEEE Trans. on Electron. Comput.*, vol. EC-16, no. 10, pp. 567-580, October 1967.

[18] J.E. Smith and G. Metze, "The design of totally self-checking combinatorial circuits," in *Proc. Int'l Symp. on Fault-Tolerant Comp.,* June 1977.

[19] M. Nicolaidis and B. Courtois, "Design of self-checking systems based on analytical fault hypotheses," Research Report RR-353, IMAG, March 1983.

[20] T. Nanya and T. Kawamura, "Error secure/propagation concept and its application to the design of strongly fault secure processors," in *Proc. Int'l Symp. on Fault-Tolerant Comp.,* pp. 19-21, June 1985.

[21] K. Zielinski, "Model of error propagation in systems of communicating processes," in *Sci. Comput. Program.,* vol. 6, no. 2, pp 191-205, March 1986.

[22] T.-H. Lin and K.G. Shin, "Location of faulty module in a computing system," submitted to *IEEE Trans. on Comput.*

[23] K.G. Shin and Y.H. Lee, "Evaluation of error recovery blocks used for cooperating processes," *IEEE Trans. Soft. Eng.,* vol. SE-10, no. 6, pp. 692-700, November 1984.

[24] K.G. Shin and T.-H. Lin, "Modeling and measurement of error propagation in a multi-module computing system," to appear in *IEEE Trans. on Comput.*, vol. C-37, no. 9, September 1988.

[25] E.J. McCluskey, "Built-in self-test techniques," *IEEE Design and Test,* vol. 2, no. 2, pp. 21-28, April 1985.

Downloaded From: https://www.spiedigitallibrary.org/conference-proceedings-of-spie on 27 May 2019
Terms of Use: https://www.spiedigitallibrary.org/terms-of-use

[26] T. Sridhar and J.P. Hayes, "Design of easily testable bit-sliced systems," *IEEE Trans. Circuits and Systems,* vol. CAS-28, pp 1046-1058, Nov. 1981.

[27] Y. You, "Self-testing VLSI circuits," Ph.D. Dissertation, Dept. of Electrical Engineering and Computer Science, University of Michigan, 1986.

[28] Y. You and J.P. Hayes, "A built-in testing approach for regular VLSI circuits," *Proc. Int'l Conf. on Circuits and Systems,* (ISCAS 85), Kyoto, pp. 1309-1312, June 1985.

[29] T.C. Lo *et al.*, "A 64K FET dynamic random access memory: Design consideration and description," *IBM J. Res. Develop.,* vol. 21, pp. 318-327, May 1980.

[30] Y.You and J.P. Hayes, "A self-testing dynamic RAM chip," in *IEEE Jour. Solid-State Circuits,* vol. SC-20, pp. 428-435, February 1985.

[31] J. Yamada *et al.*, "A submicron VLSI memory with a 4b-at-a-time built-in ECC circuit," *Proc. Int'l Solid State Circuit Conf.,* pp. 104-105 and p. 325, February 1984.

[32] J.F. Meyer and L. Wei, "Influence of workload on error recovery in random access memories," to appear in *IEEE Trans. on Comput.*, vol. C-37, no. 4, April 1988.

[33] D. Rennels and S. Chau, "A self-exercising, self-checking memory design," in *Digest 16th Int'l Symp. on Fault-Tolerant Computing* , Vienna, Austria, July 1986, pp. 358-363.

[34] A. Movaghar and J. F. Meyer, "Performability modeling with stochastic activity networks," in *Proc. Real-Time Systems Symposium*, Austin, TX, December 4-6 1984, pp. 215–224.

[35] J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic activity networks: Structure, behavior, and application," in *Proc. Int'l Workshop on Timed Petri Nets*, Torino Italy, July 1-3 1985, pp. 106-115.

[36] W. H. Sanders and J. F. Meyer, "Performability evaluation of distributed systems using stochastic activity networks," in *Proc. Int'l. Workshop on Petri Nets and Performance Models*, Madison, WI, August 24-26 1987, pp. 111-120.

[37] W. H. Sanders and J. F. Meyer, "METASAN: A performability evaluation tool based on stochastic activity networks," in *Proc. ACM-IEEE Fall Joint Computer Conference*, Dallas, TX, November 2-6 1986, pp. 807–816.

[38] J.F. Meyer and L. Wei, "Analysis of workload influence on dependability," Comput. Res. Lab, Univ. of Michigan, Ann Arbor, MI, Tech. Rep. CRL-TR-15-87, December 1987 (submitted to *18th Int'l Symp. on Fault-Tolerant Comput.*).

[39] E. Cinlar, *Introduction to Stochastic Processes*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1975.

Downloaded From: https://www.spiedigitallibrary.org/conference-proceedings-of-spie on 27 May 2019
Terms of Use: https://www.spiedigitallibrary.org/terms-of-use