ONR Grant N00014-91-J-1404
Final Report for

Synthesis Techniques and Analysis Tools for On-Chip Fault-Tolerance

Dhiraj K. Pradhan
Computer Science Department
Texas A&M University
College Station, TX 77843

Contents

This report summarizes significant research performed under ONR grant N00014-91-J-1404 on a broad range of issues related to fault-tolerant design both at the chip and system level.

First discussed is an integrated approach, currently under development for integrating concurrent checking with BIST. The goal here is to generate synthesis tools to develop low-cost fault-tolerant VLSI chip design tools that are both easy to test as well as being robust against operational errors. Section 2 reviews REACT, a tool currently under development for fault-tolerant architecture characterization. Finally, Section 3 presents on-going research on the development of novel fault-tolerant architectures.

1 Fault-tolerant VLSI: An Integrated Approach

The traditional focus of VLSI design has been performance and area. Increasingly integral to the design methodology are newer concerns aimed at reducing both production and life-cycle costs through enhanced testability and field diagnosability.

Also system availability is becoming a key feature of both fault-tolerant and non fault-tolerant systems. Crucial to achieving high availability is the ability to rapidly perform field diagnosis. Achieving this requires both the ability to perform concurrent as well as periodic checking.

However, in the context of VLSI design, concurrent checking and periodic testing have been treated in isolation. Concurrent checking has been the primary focus of those system designers concerned with system availability and serviceability. On the other hand, techniques such as BIST (self-test) have been the focus of VLSI test engineers concerned with product quality.

Error detecting and correcting codes which form the basic framework for the design of concurrent checking methodology have recently been shown by the authors to provide a comprehensive mathematical framework for analysis and synthesis of a large variety of BIST methodology, as well. This important new linkage can provide a unified framework for the development of an integrated design methodology.

The importance of BIST is two-fold: (i) reducing the cost of testing and (ii) identification of faulty field replaceable units for repair and reconfiguration. Similarly, the goal of concurrent checking is also two-fold: providing high reliability through error containment as well as high availability through rapid identification of the faulty field replaceable unit. Consequently, there is significant overlap between the objectives of BIST and concurrent checking. This can be exploited to the mutual advantage of BIST and concurrent checker designs.

Although many present day VLSI design methodologies have integrated DFT techniques, the design and placement of concurrent checkers as well as BIST support logic is still carried out in an ad-hoc manner. There are many obvious drawbacks to this which stem from the following two factors.

Firstly, the designs of concurrent checkers and BIST hardware, in and of themselves, without consideration of any potential interactions, are obviously not tailored to efficient sharing of the available silicon area. The other important drawback of treating these two aspects independently is that the overlapping information is not allowed to be used to realize potential performance improvements.

Demonstrated clearly in this research is that the BIST design can greatly benefit from the use of available concurrent checkers and vice-versa[11]. Our main goal is, therefore, to develop theory and design techniques to unify these design methodologies, taking advantage of potential interactions. Specifically, following are some of the major advantages in developing this integrated framework.

- First, it may be seen that a key objective of BIST is to detect and identify the faulty subcircuit. So, also, the concurrent checkers provide information about the location of faults through detection of on-line errors. Therefore, there is a natural overlap in the information from BIST hardware and concurrent checkers. Consequently, if these two are designed for sharing this information, it can be expected that the overall effectiveness can be increased with reduced hardware overhead.
- The effectiveness of a concurrent checker is highly dependent on minimizing the probability of undetected error. Similarly, the effectiveness of the BIST compressor is highly

dependent on the aliasing probability. Using algebraic coding theory, the authors recently were able to establish that there are simple relationships between the probability of undetected error and aliasing probability. These relationships naturally provide the basis for exploring the precise benefits of an integrated design methodology. For example, understanding of error models for effective BIST compression can also lead to effective concurrent checking.

- The BIST and concurrent checkers can be designed to complement each other. For example, the periodic self-test, using BIST, can be aimed at precisely only those faults that escape the concurrent checkers. This could not only simplify the BIST hardware but also increase the effectiveness of the overall fault diagnosis strategy.

- This integrated approach would lead to a better understanding of the trade-offs involved between concurrent checking and self-testing. For example, in a noisy environment, both error correction and detection may be essential. This may, in turn, require significant VLSI complexity for the concurrent checker. However, as shown here in this research, if concurrent checkers are used in conjunction with BIST, one can significantly enhance the fault coverage. Thus in the integrated environment, the overhead of implementing concurrent checking can be compensated (partially) by a potential decrease in VLSI area required for BIST.

- Effective fault isolation strategy is critical to implementation of cost effective Field Replaceable Unit (FRU) replacement policy. It can be seen that the concurrent checkers and the BIST hardware both provide information about the location of errors in the system. Hence, if the field diagnosis procedures are formulated under an integrated design environment, then the information from concurrent checkers can be utilized by the BIST and vice versa, in order to formulate an efficient strategy.

Previous approaches include a self-verification scheme and the UBIST scheme. Our approach provides an integrated coding theory based framework for studying the dual use of concurrent checkers for operational fault-detection and BIST[11]. Specifically, proposed is a self-test scheme combining parity (code) checking and MISR compression. In the proposed

scheme, the circuit augmented with parity (code) predictors is not required to be fault-secure. Most of the extra circuitry added for self-test is the parity prediction circuitry. This is very useful during the normal circuit operation, providing the main advantage of the proposed scheme. Fault-escape probability in the proposed scheme is first studied for various error models. The general framework being developed by us is being used to compute aliasing probability for the scheme. The effectiveness of the proposed scheme on Read Only Memories, compared with the effectiveness of the ODM scheme for ROMs is under study. Parity predictors are being synthesized for benchmark circuits, in order to evaluate the area overhead that is due to parity predictors. Also, the synthesized circuits will be used to obtain error models; these help compute fault-escape probability for the proposed scheme.

2 The Reliable Architecture Characterization Tool

The Reliable Architecture Characterization Tool (REACT) is a software testbed which performs automated life testing of many user-defined multiprocessor systems through simulated fault-injection [7, 10]. This involves emulating the high-level hardware and software components of a given system while concurrently injecting bit-level faults and errors into it. During a single *simulation run*, the code conducts a certain number of experiments or *trials* in which an initially fault-free system is operated until it fails or reaches a specified censoring time. The exact number of trials required is determined by the desired confidence intervals about the system dependability attribute being investigated. Extensive instrumentation has been included in the program in order to collect data from each trial which is later aggregated over the entire simulation run in order to generate the outputs. Graphs of reliability and availability, a comprehensive failure mode report and various statistical measurements are provided as output by the software. REACT consists of 8000 lines of C running under UNIX and completes a "typical" simulation run in less than 10 hours on a dedicated DECstation 5000/120.

REACT can analyze a class of architectures in which any number of processor and memory modules may be specified and each can be designated as initially active or a hot or cold standby spare. *Groups* of processors or memories may also be defined in which all

modules operate redundantly. The error control logic may be built from various combinations of components commonly found in fault-tolerant designs. Custom error control logic circuitry may also be specified by the user. Processors are simulated at the functional-level whereas a logical-level description is used for the memory modules and error control logic. Logic values 0 and 1 are not differentiated in the system model: only error-free and erroneous states exist for each bit. Memory depth is variable and a 16-bit word width for memory and all data paths has currently been implemented. Other word sizes may be realized with minor modifications in the code.

A synthetic workload is assumed in which processors continually perform *computation cycles* consisting of an instruction fetch, a possible operand read, a computation and a possible result write. Real code and data are not used by REACT, but errors are allowed to propagate throughout the system as if the application program was actually being executed. Behavior of the application workload is specified by a mean instruction execution rate, the probabilities of performing a data read and write per instruction plus a locality of reference model. Values for the mean number of data accesses made during the execution of an instruction may be obtained either through trace analysis or directly from the measurement of operational hardware. It is assumed that all memory references access one whole word. Which memory locations are accessed during a computation cycle are determined via the locality of reference model. The testbed implements a model based on Bradford-Zipf distributions which suggests that α % of all accesses go to β % of the memory under the condition $\alpha + \beta = 1$. Reference addresses are assumed to be uniformly distributed inside and outside of the locality and no attempt is made to separate code from data in memory with the model.

The fault/error model employed by REACT accounts for permanent, intermittent and transient faults in the processors plus permanent and transient faults in the memories as well as the error control logic. Faults with a Weibull distribution (of which the exponential distribution is a subset) for their inter-arrival times are injected into these modules only at the beginning of a computation cycle. Faults are assumed to always cause immediate errors, so their fault (but not error) latency is 0. Correlated failures are presently not considered.

Processor fault effects are assumed to be completely characterized by the rate at

which errors appear on its memory bus. Three types of errors exist: transients lasting only one computation cycle, intermittents with a Weibull distributed duration and permanents which have an effect in every computation cycle. Errors may affect either addresses, (write) data or both addresses and data simultaneously. An erroneous address is assumed to access a random memory location while erroneous data take a random value. In addition, erroneous processor reads generate output errors in the same computation cycle.

Memory faults are divided among the bit-array and addressing-logic regions of a memory module. The fraction of faults which fall into each of these regions may be approximated by their relative chip areas. Bit-array faults are assumed to affect a single random bit in a word at a random address while a random location is referenced during an addressing-logic fault. A transient bit-array fault may be overwritten (changing it from the erroneous to error-free state) at any time, but a permanent can never be overwritten. Addressing-logic transients last one computation cycle and permanents will cause the memory module to endlessly access random words. An access to a random address reads or writes a value with randomly corrupted bits, representing the difference between the bit values of the word that was accessed and the word that should have been accessed. Finally, faults within one of the error control logic components are assumed to affect a single random bit either permanently or for one computation cycle in the case of transients.

2.1 Reliability Analysis of Unidirectional Voting TMR Systems

Computer systems used in aircraft and reactor control often require critically high reliability for moderately short mission times. Triple-modular redundant (TMR) hardware has been employed in many of these ultrahigh reliability applications. The three redundant processors of a TMR system concurrently execute identical tasks while the triplicated memories contain the same code and data. Majority voting is used to mask erroneous module outputs. The voter (V) is usually inserted into the redundant system buses between the processors (P) and memories (M). Bit-wise voting is typically performed on data, address and control lines during both read and write accesses to memory. Such a system will be referred to as *bidirectional voting* (BDV) TMR.

Voting has a substantial performance penalty associated with it. This degradation can be attributed to two specific delays [6]. The **propagation delay** of signals through the voter logic is the more obvious contributor to increased memory access times. Less apparent is the **synchronization delay** incurred when clock skew requires modules to wait for a lagging signal before performing a vote. This penalty becomes even greater if a module fails in such a way that it does not respond, forcing a timeout period to be suffered on each memory reference. TMR systems used in hard real-time applications may not be able to tolerate the ensuing drop in throughput after this type of failure.

It is possible to significantly reduce the performance degradation of a BDV system by **voting only on one type of memory access**, either reads or writes. These *unidirectional voting* systems are expected to have lower reliability than the bidirectional design since a smaller fraction of errors will be masked, possibly allowing them to propagate and corrupt the state of non-faulty modules.

Because the voter may be by-passed on either memory read or write accesses to achieve higher performance, two different unidirectional voting systems exist. The *Read-Only Voting* (ROV) TMR system removes the voting delays from the bus cycle on writes. Processor generated read addresses and memory outputs are voted upon and a single voted value is distributed to all three processors. Processor outputs are written straight into the corresponding memories without any error masking. The ROV TMR system therefore allows processor errors to propagate into the memories while all single errors from memory will be contained by the voter.

The dual of the ROV system is the *Write-Only Voting* (WOV) TMR system which eliminates the delay associated with voting on read accesses. It performs a vote only at the outputs of the processors and writes a single voted value into all three memories at a voted address. No masking of data or addressing errors takes place during reads, so erroneous memory outputs may propagate directly into the associated processors. Voting terminates any single processor error before it reaches the memories.

Both unidirectional voting TMR systems can realize better performance than the traditional bidirectional voting system. However, **WOV should have better performance than ROV** because it suffers the delays of voting less often since reads generally occur

much more frequently than writes. In terms of fault-tolerance, one might **expect ROV to provide higher reliability than WOV** for similar reasons. When processors and memories experience faults at the same rate, the percentage of potentially fatal errors that will get masked will be larger with the ROV system. In addition, memory often has a higher fault rate than processors so the percentage of errors masked will be even greater when the voter is placed at the output of the less reliable component.

Two parametric analyses of the bidirectional and unidirectional voting TMR systems were carried out with REACT [8, 9]. The following observations were made:

- the tradeoff of reliability for performance made by the unidirectional voting systems becomes more effective as the difference between processor and memory module failure rates increases
- near ideal tradeoffs can be attained for some failure rate combinations, particularly when memory is more likely to fail than the processors
- the analytical model traditionally used to predict the reliability of TMR designs is indicative of some of the differences between the bidirectional and unidirectional voting systems, but is not always accurate
- reliability of the ROV system is generally better than the WOV system, except when processor failure rates are high relative to the memory failure rates
- system failure is caused by propagation of errors more often in the WOV system than in the ROV system
- workload has limited effects on reliability when memory error latency is low

Results demonstrated that in many cases, acceptably little reliability was sacrificed by the unidirectional voting TMR systems for a potentially large increase in performance.

3 Novel Fault Tolerant Architecture Development

3.1 Roll-forward Checkpointing Schemes

A fault-tolerant multiprocessor environment wherein each task is executed simultaneously on two processing modules is considered. A pool of a small number of nondedicated spares or processing modules with spare processing capacity is assumed available (see Figure 1). Duplex fault-tolerant architectures that require no rollback for most faults are proposed.

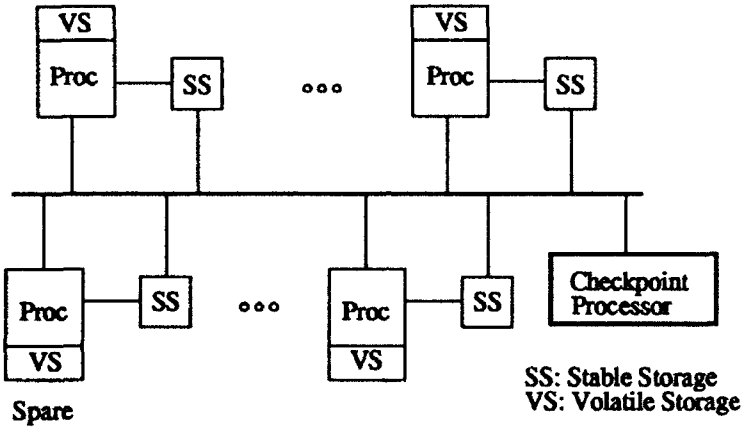
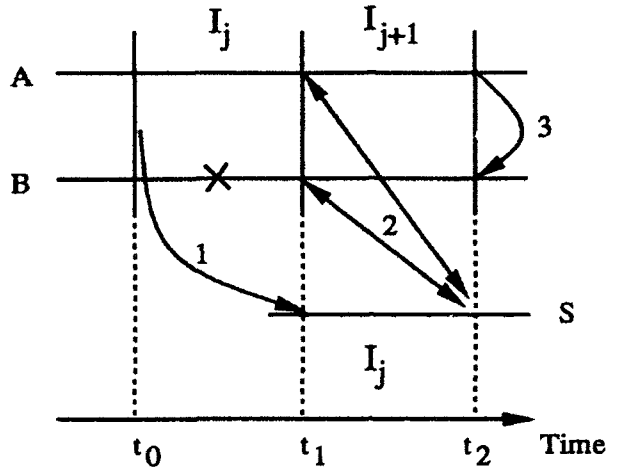


Figure 1: System architecture for roll-forward checkpointing schemes

In the proposed schemes, at each checkpoint the state of the two modules executing the task is compared for detection of faults. If a fault is detected, instead of usual rollback, the following mechanism is used for identification of the faulty processing module [13, 14, 16]. The good state of the previous checkpoint is loaded into a spare module. The checkpoint interval in which the failure is detected is then "retried" on the spare module. Concurrently, the task continues execution on both processing modules in the duplex system. At the next checkpoint the state of the spare is compared with the state of the two processing modules at the previous checkpoint where disagreement occurred. This allows for the identification of the faulty module (see Figure 2). Once the faulty module is identified, the state of the faulty module is made consistent with the state of the fault-free module in the duplex system and the spare is released to the pool.



1 : Copy state to the spare
2 : Compare state of the spare with the state of A and B
3 : Copy state from A to B
X A fault

Figure 2: Roll-forward checkpointing scheme

These schemes are termed as Roll-Forward Checkpointing Schemes (RFCS). The proposed RFCS schemes provide a mechanism for identifying the faulty processing module and recovering it, in most cases, without the overhead of rollback. It is demonstrated that the proposed schemes have potential performance advantages over conventional duplex system with rollback.

Specifically, the advantage of the proposed schemes is that they achieve a lower average execution time with a lower variance as compared to the rollback schemes. This is crucial for real-time systems with hard deadlines as lower variance enhances the predictability of the task completion time.

3.2 Reliable Memory Design

The use of a hybrid memory structure consisting of both highly reliable and normal memory can further support persistent and recoverable memory [1]. Hybrid algorithms that manage the writable memory and read-only memory separately are proposed. The traditional

measures of virtual memory algorithms (i.e., lifetime and space-time) have been extended to account for the dual nature of the policies. Several properties of the policies have been explored. It has been shown that the knee of a hybrid lifetime curve produces a near minimum space-time product as with the existing algorithms. Hybrid policies are more controllable with respect to highly reliable memory because they can constrain the amount of writable memory and gain performance by using additional read-only memory. The lifetime measure for the hybrid policies under constrained writable memory, when compared at equal amounts of highly reliable memory, is better than the single policy algorithm at a small cost of additional read-only memory. Furthermore, even at an unconstrained amount of writable memory, the hybrid policy produces approximately equal performance while the writable memory can be completely fixed in size. Theoretical results are also derived for a property which indicates the optimal performance for a hybrid reference stream based on two individual streams.

The ability to accurately predict the reliability of a system is very important. Two novel techniques have been developed which focus on dynamic aspects of memory [2, 3, 4, 5]. The first focuses on the memory reference patterns of a particular program while the second looks at memory behavior due to memory management actions.

The first novel technique evaluates the probability of correct execution of a program based on the program's memory access behavior. The approach is an analytical study using an existing model which characterizes an address trace with four parameters. Three cases are developed based on the storage allocation policy (i.e., pre-allocated, dynamically allocated, or constrained in allocation). The models are able to compare the traditional view that is taken in standard memory reliability analysis to that of a real world environment where a program uses a varying fraction of the memory at different instances. Using these models, it is shown that the reliability may be significantly better than the apparent reliability when the program behavior was not considered. It provides one explanation for the cause of unobserved faults along with an analytical basis for determining the extent of faults not being observed. Possibly the most important application of these models is to analytically quantify the observed phenomenon that failure rates increase with increased workload. A new explanation has been proposed for this phenomenon based on the notion that programs

often have storage allocated which will never be referenced again and cannot cause a failure. Assuming a constant fault rate over increased workloads, the model shows that there could be a significant increase in observed failures. The model was validated with actual program traces and shown to be very accurate. Finally, several techniques have been shown for extracting the fractal parameters of a program trace.

The second novel technique for reliability analysis uses the memory space allocated to more accurately calculate the reliability. This can be used to understand the relationship between the amount of memory allocated and the reliability. This effect has been quantified based on the relative cost of a fault. Distinct effects have been measured depending on the relative speed of the paging device. For small reload times it is found that a decrease in the memory partition size leads to an increase in reliability at the cost of additional instruction overhead. For extremely long reload times it is found that larger amounts of memory lead to increased reliability. There also exists a middle reload time where the optimal reliability corresponds to the optimal space-time performance. Other aspects of virtual memory algorithms such as small pages and different paging algorithms were studied. Furthermore, the methodology was applied to study the reliability of cache memories which have the characteristic of very small reload delays. The results show that the reliability improvement factor can change by several orders of magnitude based on the cache size. For small memory sizes it was found that a very small number of page durations contribute to a majority of the total unreliability. Two techniques have been suggested to remove these long durations, which then lead to even greater improvements in the reliability. One is an algorithm called *selective scrubbing* to break the long durations, which could either be implemented in software or hardware. A second technique showed that the addition of very small amounts of highly reliable memory can also lead to significant reliability improvements.

3.3 Safe Modular Redundant Systems

Dependability considerations warrant that in addition to reliability, a dependable system must have a high level of safety. Therefore, there is a need to ensure operation which is both error-free under adverse conditions, as well as safe under severely adverse conditions.

We have analyzed a technique for implementing systems requiring high reliability and

safety [15]. These systems, named n-Safe modular redundant (nSMR) systems, achieve high reliability and safety using module replication and redundancy in module output.

An nSMR system consists of n identical modules and an arbiter. The arbiter uses outputs of all the n modules to decide the nSMR system output. Reliability and safety of the system are a function of the arbitration strategy used. When reliability is the only criterion, an optimal arbitration strategy that maximizes the reliability can be designed. With reliability and safety both of concern, usually no single arbitration strategy is optimal. We have presented an implementation of *maximal* arbitration strategies which achieve different maximal reliability and safety combinations. Maximal arbitration strategies are such that no arbitration strategy has better reliability and safety, compared to a maximal strategy.

The effect of increasing redundancy on the achievable reliability and safety has been analyzed for systems with and without redundant module outputs. Detailed results on binary SMR systems using binary arbiters have also been obtained. The results of this chapter are summarized below.

- It is shown that for modules without output redundancy, no arbitration strategy exists for $(n+1)$SMR which achieves better reliability and safety compared to certain arbitration strategies for nSMR. Further, given any arbitration strategy for nSMR, there always exists an arbitration strategy for $(n+2)$SMR that achieves higher reliability and safety.

- It is shown that if modules have output redundancy, given an arbitration strategy for nSMR, one can always find an arbitration strategy for $(n+1)$SMR that achieves better reliability and safety.

- A detailed analysis of binary nSMR systems with single bit output is presented. Whether binary $(n+1)$SMR dominates binary nSMR is shown to be dependent on the relation between the likelihood of a detected error (p_d) and the likelihood of an undetected error (p_u) in a binary module's output. It is shown that when $p_d = p_u$, binary $(n+1)$SMR does not dominate any of the plurality strategies for binary nSMR. Also, exact expressions for the reliability and safety of the maximal strategies for such systems have been presented.

- Design of a family of threshold-based maximal arbitration strategies which achieve different reliability and safety is presented. Design of a class of arbitration strategies easier to implement as compared to the threshold-based arbitration strategies is also presented. These arbitration strategies are obtained by generalizing the plurality strategies.

4 Papers Under ONR Grant and References

1. N. S. Bowen, *Fault-tolerant aspects of memory design.* PhD thesis, University of Massachusetts-Amherst, February 1992.

2. N. S. Bowen and D. K. Pradhan, "Program fault tolerance based on memory access behavior," in *21rst Symp. on Fault-Tolerant Computing*, pp. 426–433, IEEE, June 1991.

3. N. S. Bowen and D. K. Pradhan, "Effect of memory management on reliability," Tech. Rep. TR-91-CSE-3, University of Massachusetts, Feb. 1991.

4. N. S. Bowen and D. K. Pradhan, "Reliability aspects of memory management policies," Tech. Rep. TR-91-CSE-16, University of Massachusetts, July 1991.

5. N. S. Bowen and D. K. Pradhan, "Issues in fault tolerant memory management," Tech. Rep. TR-91-CSE-20, University of Massachusetts, Aug. 1991.

6. J. A. Clark and D. K. Pradhan, "Unidirectional voting TMR systems," Tech. Rep. TR-91-CSE-6, University of Massachusetts, Apr. 1991.

7. J. A. Clark and D. K. Pradhan, "REACT - the reliable architecture characterization tool," Tech. Rep. TR-92-CSE-22, University of Massachusetts, June 1992.

8. J. A. Clark and D. K. Pradhan, "Reliability analysis of unidirectional voting TMR systems through simulated fault-injection," in *Digest of Papers for the 1992 Workshop on Fault-Tolerant Parallel and Distributed Systems*, pp. 72–81, IEEE, July 1992.

9. J. A. Clark and D. K. Pradhan, "Reliability analysis of unidirectional voting TMR systems through simulated fault-injection," Tech. Rep. TR-92-CSE-9, University of Massachusetts, Mar. 1992.

10. J. A. Clark and D. K. Pradhan, "REACT - a synthesis and evaluation tool for fault-tolerant multiprocessor architectures." to appear in the *Annual Reliability and Maintainability Symposium*, Jan. 1993.

11. Gupta, S. K. and Pradhan, D. K., "Can concurrent checkers help BIST?", to appear in ITC 1992.

12. B. K. Kar and D. K. Pradhan, Patent application filed. "A New Implementation Scheme of Rank Order/Stack Filters".

13. D. K. Pradhan and N. H. Vaidya, "Roll-forward checkpointing scheme: Concurrent retry with nondedicated spares," in *IEEE Workshop on Fault Tolerant Parallel and Distributed Systems*, July 1992.

14. D. K. Pradhan and N. H. Vaidya, "New roll-forward checkpointing schemes for modular redundant systems," in *Hardware and Software Fault Tolerance in Parallel Computing Systems* (D. R. Avresky, ed.), England: Ellis Horwood, 1992.

15. N. H. Vaidya and D. K. Pradhan, "Voting in fault-tolerant systems: Reliability and safety issues," Tech. Rep. TR-91-CSE-7, ECE Department, Univ. of Massachusetts, June 1991. Also accepted for publication in IEEE Transactions on Computers.

16. N. H. Vaidya, *Low-Cost Schemes for Fault Tolerance.* PhD thesis, University of Massachusetts-Amherst, August 1992.