AD-A279 896

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

2. REPORT DATE

3. REPORT TYPE AND DATES COVERED
FINAL/01 SEP 92 TO 31 AUG 93

4. TITLE AND SUBTITLE
FAULT-TOLERANCE IN DISTRIBUTED & MULTIPROCESSOR REAL-TIME SYSTEMS

5. FUNDING NUMBERS
2304/FS
F49620-92-J-0383

6. AUTHOR(S)
DR. PRADHAN

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
TEXAS ENGINEERING EXPERIMENT CENTER
308 WERC
COLLEGE STATION, TX 3124

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR- 94 0330

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
110 DUNCAN AVE, SUITE B115
BOLLING AFB DC 20332-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-92-J-0373

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
New schemes for fault-tolerance in multiprocessor and distributed systems have been developed in the following areas:
* We have investigated a number of fault tolerance schemes to evaluate performance, reliability, and availability trade-offs. Fault tolereance schemes are being developed for various fault models (fail-stop model, fail-slow model, and arbitrary failure model) and application areas (applications that are to provide results at the end of computation and applications that are long-running but should also provide results during computation).
* In the area of software-implemented fault tolerance, we are studying approaches for providing user transparent mechanisms for fault tolerance to design and implement a software library to which the user can link existing application software to achieve the desired level of fault tolerance.
* We are developing a new tool (Reliable Architecture Characterization Tool--REACT) for evaluating the reliability and availability of distributed multiprocessor systems using various fault tolerance techniques. This tool will facilitate evaluation of the fault tolerance schemes that we develop.

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR(SAME AS REPORT)

Final Report for AFOSR Grant #F49620-92-J-0383DEF

Prepared by:

Dr. Dhiraj K. Pradhan
H.R. Bright Building
Department of Computer Science
Texas A&M University
College Station, TX 77843

Submitted to:

AFOSR/PKA
110 Duncan Avenue, Suite B115
Bolling AFB, D.C. 20332-0001

Department of Computer Science
Texas A&M University

301 Harvey R. Bright Bldg • College Station, Texas 77843-3112

Contents

1 Introduction

This report summarizes research carried out under AFOSR grant #F49620-92-J-0383DEF. Under AFOSR sponsorship, new schemes for fault-tolerance in multiprocessor and distributed systems have been developed as described below:

- Design and implementation of fault tolerance schemes for multiprocessor and distributed systems. We have investigated a number of fault tolerances schemes to evaluate the performance, reliability and availability trade-offs. Fault tolerance schemes will be developed for various fault models and application areas. The fault models may be divided into three classes: (i) fail-stop model, (ii) fail-slow model, and (iii) arbitrary failure model. The applications are divided into two types: (i) long-running applications (such as distributed simulations, weather-forecasting, etc.) which are expected to provide results at the end of computation. (ii) applications that are long-running but are also expected to provide results often during the computation. The requirements of these two application areas are somewhat different, requiring different fault tolerance techniques.

  The goal of our research has been to design unified approaches to deal with various fault models and experimentally evaluate the performance of the proposed fault tolerance mechanisms.

- Software-implemented fault tolerance for multiprocessor systems such as nCUBE and MasPar. We are studying approaches for providing user-transparent mechanisms for fault tolerance. The goal here is to design and implement a software library. The user can link the existing application software to this library and achieve the desired level of fault tolerance.

- Design and development of a new tool for evaluating the reliability and availability of distributed and multiprocessor systems using various fault tolerance techniques. Such a tool will facilitate evaluation of the fault tolerance schemes that we propose to develop.

2 Research Progress in Detail

This section discusses the above three thrust areas, and also presents our preliminary work in each of the areas.

2.1 Fault Tolerance in Multiprocessor and Distributed Systems

Design and implementation of fault tolerance schemes for multiprocessor and distributed systems is a thrust area of this continuing research. We are investigating a number of fault tolerance schemes for multiprocessor and distributed systems to evaluate the performance, reliability and availability trade-offs. The fault tolerance mechanism used in such systems must be chosen based on a number of criterion. The criterion that we consider important are as follows.

- Reliability and availability requirements. These requirements have a serious impact on the level of redundancy required. These requirements may be specified probabilistically (e.g. availability of 99.2%) or deterministically (e.g. tolerate up to two simultaneous failures). High reliability and availability requirements typically require higher redundancy.

- The fault model. The fault models that are applicable to most real-life systems are:
  (i) fail-stop model. This is the most frequently studied fault model. Here it is assumed that a faulty processor detects its own failure and stops functioning immediately. Although easy to understand, realization of this fault model results in significant hardware overhead.
  (ii) fail-slow model. This model is weaker than the fail-stop model. Here, it is assumed that a faulty processor detects its failure within a certain time after the fault occurs.
  (iii) arbitrary failure model. Sometimes this is called the Byzantine fault model. Here, no assumption is made on the behavior of a faulty processor. This fault model is the easiest to realize. However, achieving fault tolerance is much more difficult than the other two fault models.
  We propose to investigate fault tolerance schemes for all the three models.

- The application. Two types of applications are of particular interest: (i) long-running applications which are expected to provide results at the end of computation (e.g. distributed simulations, weather-forecasting, etc.) (ii) applications that are long-running but are also expected to provide results often during the computation. The requirements of these two application areas are somewhat different, requiring different fault tolerance techniques.

The goal of our research is to provide fault tolerance approaches to match various reliability and application requirements, and experimentally evaluate the performance of the proposed fault tolerance mechanisms. We propose to develop a testbed to implement a wide range of fault tolerance schemes for multiprocessor and distributed environments. An important objective here is to provide a common basis for experimental evaluation and comparison of various schemes. To illustrate our goals, we now present a fault tolerance scheme for the following scenario.

- **Reliability goal:** Tolerance of a single failure.
- **Fault model:** Arbitrary or Byzantine fault model.
- **Application:** Long-running application providing results at the very end.

This fault tolerance scheme has been implemented for the purpose of evaluating the performance overhead of fault tolerance. We discuss its implementation and present some measurements.

The main features of the proposed checkpointing and recovery scheme [11] are:
- Process duplication,
- Fault identification using retry on a fault-free processor,
- No output delays due to checkpointing or message logging.

Sender-based message logging was put forth in [8], to minimize the cost of achieving fault tolerance by avoiding logging of inter-process messages on a stable storage. There, fault tolerance is achieved by logging at the sender, the message as well as its Receive Sequence Number (RSN). Receive sequence number of a message indicates when the message was received relative to the other messages received by that receiver. In this scheme, the receiver informs the sender the RSN of each received message, the sender acknowledging receipt of the RSNs. A receiver process cannot commit any output until it receives the acknowledgements for the RSNs of *all* messages consumed before the output was produced.

Byzantine failure model makes process duplication necessary to achieve single fault detection. The messages are logged by the sender process in its volatile storage and RSNs are logged by the receiver process in its volatile storage. Thus, each processor maintains a send log and a receive log for each process scheduled on that processor. When a processor fails, undetected errors may be introduced into both volatile logs and the volatile state of a process executing on the faulty processor. A failure is detected when the messages sent by replicas of a process mismatch, or when the state of the replicas mismatches at a checkpoint. Although a failure is detected by such a mismatch, still, the faulty processor cannot be identified.

Our scheme can be extended to systems where a processor executes multiple processes, however, in the following each processor is assumed to be executing a single process. Therefore, the following uses the terms *process* and *processor* interchangeably.

System Model

The system consists of multiple processors, as shown in Figure 1. Each processor consists of a CPU and volatile memory. Each process has two replicas. The replicas of a process are scheduled on *different* processors to prevent simultaneous failure of both the replicas. For example, Figure 1 shows replicas of two processes, P and Q. (The two replicas of a process are identified by subscripts 1 and 2).

Processes communicate only through messages (no shared memory). Each process may send messages to other processes, as well as consume messages sent by other processes. All processes are *deterministic*; if the two process replicas start in an identical state and consume the same set of messages in an identical order, then the replicas will end up in an identical state. A logical clock is associated with each replica. The logical clock may simply count the number of messages sent and consumed by a process replica by incrementing the clock just before sending or consuming a message. The logical clock may be made *faster* by also incrementing it at other points in the code. Logical clocks of both the replicas are incremented at the same logical points during execution. Each process checkpoints periodically. Both the replicas checkpoint at the same logical point in the execution, achieved using the logical clock. Different processes checkpoint independently; no coordination is assumed.

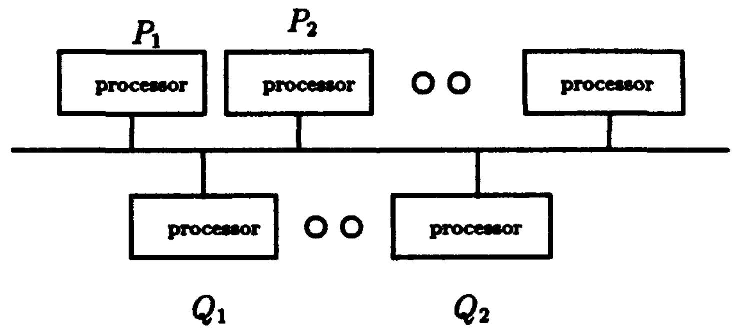

Figure 1: System model

Message Passing Mechanism

Each fault-free replica of a process must consume identical messages in the same order. This can be achieved using message authentication and time-outs. The message passing mechanism ensures that either both fault-free replicas of a process receive a message, or neither does. Also, it is ensured that both the replicas receive the messages in the same order. Discussion of the message passing protocol is omitted here. A message may be consumed only when two identical copies of the message are received from the two sender replicas. The RSN of the message is determined by when the second copy of the message is received.

Each replica of a message's sender retains in a volatile *send log* a copy of the message. This copy includes the Sender Sequence Number (SSN) of the message which indicates the position of the message in the stream of outgoing messages. The received message also includes the SSN. Each replica of the receiver process retains in its volatile *receive log* an entry containing the message, its RSN and its SSN. These logs need not be saved on the stable storage. If necessary, to free the memory, the logs may be saved (asynchronously) on a local disk or the stable storage.

A message in the send or receive log is discarded when (i) sender process has checkpointed after sending this message *and* (ii) the receiver process has checkpointed after consuming this message. The following discusses the two basic steps: (i) fault detection, followed by (ii) fault identification and recovery.

Fault Detection

A process is said to be faulty if one of its replica has failed. As shown below, the proposed scheme ensures that the fault-free replica of a faulty process detects the failure before its next checkpoint is taken. Let P be the faulty process with replicas P_1 and P_2. Also, let P_1 be the faulty replica. When the fault-free replica P_2 detects the failure, it broadcasts a message to all the processes that process P has failed. Note that even if P_2 broadcasts a message that P_1 has failed, there is no reason for the other processes to assume that replica P_2 is fault-free (because P_2 itself may, in fact, be faulty). When the other processes receive any such message, the fault identification procedure (presented later) is initiated. Note that a process is considered to be faulty only when one of its replicas broadcasts the message that it has failed. Now, the four different ways in which the fault-free replica may detect a failure are discussed.

(a) Replica P_2 will detect the failure of P_1 if P_1 does not correctly participate in the message passing and agreement protocol. On the other hand, if P_1 executes the agreement protocol correctly, then P_2 detects a failure in P_1 by one of the following three ways:

(b) Both the replicas of a process checkpoint periodically at the same logical time. All the volatile state is included in the checkpoint; the send and receive logs, however, are *not* saved as a part of the checkpoint. Each replica saves its state on a stable storage and compares the state with that of its replica. The comparison may be performed using signatures [4]. If the two states do not match, then a failure of one of the replicas is detected. In our example, if the volatile state of P_1 is corrupted due to the failure, then P_2 will detect the failure when it tries to take the next checkpoint. If the state of the two replicas of P matches, then before the checkpoint can be considered valid, the checkpoint of process P must be "approved" by each process that has received a message from P since P's previous checkpoint. A fault-free process Q_i "approves" the checkpoint taken by P only if the two replicas of P did not send mismatching messages to Q_i. A checkpoint is not valid until it is approved by all relevant processes. A receiver process, Q_i, may inform the sender process its "disapproval" any time after it receives the mismatching messages. There are two ways this may occur.

(c) Replica P_1 sends message M_1 to Q_1 and Q_2 and replica P_2 sends message M_2. Both Q_1 and Q_2 detect the message mismatch and send "disapproval" to P_1 and P_2. When P_2 receives the disapprovals from both Q_1 and Q_2, it concludes that process P is faulty. In this case, Q_1 and Q_2 cannot determine whether P_1 or P_2 is faulty.

(d) A situation similar to (c) arises if P_1 sends two different messages to Q_1 and Q_2. As message passing uses authentication, both Q_1 and Q_2 detect that P_1 has sent them different messages, and they both send their disapprovals to P_1 and P_2. As before, when P_2 receives the disapprovals from both Q_1 and Q_2, it concludes that process P is faulty. Actually, in this case, Q_1 and Q_2 both know that P_1 has failed. Therefore, Q_1 and Q_2 can broadcast this information to all the processes and the fault identification procedure described later is not required.

Thus, a failure in P_1 is detected by P_2 by one of the above four means. When a process receives a message from P_2 that "process P is faulty", the message is interpreted to mean that "one of the two replicas of process P is faulty" and both the processors executing the replicas of P are considered suspect.

When the fault in P is detected, the replicas of process P are forced to save their state on the stable storage at the next increment of their logical clock and stop executing. Let the logical time at which replica P_1 thus saves its state be t_1 and the logical time at which P_2 thus saves its state be t_2. Note that if the failure is detected, as discussed in (b) above, then $t_1 = t_2$; otherwise t_1 and t_2 may be different. The previous checkpoint of process P is also retained on the stable storage.

Fault Identification and Recovery

The faulty processor is identified using "retry" of the computation of the faulty process on a fault-free processor, say processor R. Due to the single undetected fault assumption, all the processors that are not suspect can be considered fault-free.

When a process replica fails, any of the following may be corrupted: send log, receive log, messages sent to other processes and the volatile state. The following procedure detects all these errors.

To determine which processor executing replicas of P is faulty, first assume that P_1 is fault-free. The computation performed by P_1 since its previous checkpoint is retried on R. For this purpose, first the previous checkpoint taken by process P is loaded on R. Also, the volatile send and receive logs maintained by P_1 are sent to R. Then, R executes process P, starting from the previous checkpoint. The messages are consumed by R in the same order as indicated by the receive log of P_1. The receive log of P_1 includes for each received message the identification of its sender and the SSN. Using the sender ID and the SSN in the receive log of P_1, R requests the sender to resend the corresponding message. When the message is received, it is compared with the copy of the message in P_1's receive log. A mismatch indicates that P_1 is faulty. Also, if the original assumption that P_1 is fault-free is true, then when the computation is retried on R, R must send exactly the same messages as those sent earlier by P_1, if any. If the messages sent by P_1 and by R (during retry) do not match, then P_1 must be faulty and thus P_2 is identified to be fault-free.

If all the messages sent by R and P_1 match, then the state of R at logical time t_1 is compared with the state of P_1 at t_1. Recollect that P_1 saved its state at logical time t_1 after process P was detected to be faulty. If the comparison results in a mismatch, P_1 must be faulty and P_2 is considered fault-free. If the comparison results in a match, P_1 must be fault-free and P_2 is identified as faulty.

Note that if a process replica has failed, at least one of the following must be corrupted: send log, receive log, messages sent to other processes and the volatile state. The above procedure detects errors in each of these. Therefore, the fault identification procedure correctly identifies the faulty processor.

Once the fault-free replica of the faulty process is identified, the state of the fault-free replica can be copied to another processor, thereby creating two consistent replicas (both in correct state) of process P. Thus, the system recovers from the single processor failure.

Experimental Testbed and Evaluation

An experimental system has been developed to measure the performance degradation caused by the above fault tolerance mechanism. A software layer has been developed that implements the algorithm presented above. The software is developed in C language and measurements are carried on a network of SPARC workstations. The software layer and measurement methodology is described below.

The implementation can broadly be classified into the following four modules:

1. *Initiation*: Given the number of processes, and the host names, this module is responsible to open communication channels. For testing purposes we have used a complete network connection. The communication protocol used is TCP.

2. *User Processes*: For measurement purposes, user processes periodically send messages, receive messages, and checkpoint. The rate at which these operations are performed can be controlled by input parameters. The time complexity of the user process is dependent on the number of messages sent by this process.

3. *Message Reception and processing*: This module performs the Byzantine message agreement protocol required for the scheme.

4. *Checkpoint*: This module has two components: (i) *_chkpnt*, and (ii) *_compare*. *_chkpnt* component checkpoints the process. The *chk_freq* determines when the process has to take its checkpoint. For measurement purposes, the checkpoint component dumps *chk_data_size* bytes to the disk (stable storage). The *chk_freq* and *chk_data_size* is varied to change the checkpoint frequency and the size respectively. After dumping its state, if its replica has also taken the checkpoint, checkpoint comparison takes place, else, it continues with its process. This is achieved using the *_compare* component. If the previous checkpoint has not yet been compared the process enters the *_poll* mode. The *chk_poll_freq* variable determines as to how frequently the process should poll the disk for its replica's checkpoint. For all the measurements, we have assumed *chk_poll_freq* to be half of the *chk_freq* value.

Experimental Results

To measure the overhead imposed by the fault tolerance scheme, we measured two quantities: (i) total execution time required to complete the task without fault tolerance, and (ii) total execution time with the fault tolerance mechanism. These experiments were performed to determine the execution overhead of the fault tolerant scheme during normal execution (without failures).

The parameters varied during the different runs were the checkpoint size, checkpoint frequency, and the length of execution time. The execution time was assumed to be proportional to the number of messages sent. Thus by varying the number of messages to be sent by a process, we got different lengths of execution time.

The measurements of the execution times for the various checkpoint sizes and checkpoint frequencies is shown in Table 1. The execution time for the process without incorporating fault tolerance is also shown. The Size column of the table refers to the checkpoint size, CI columns shows the execution times for the various checkpoint interval sizes. CI=2 means that a checkpoint was taken every 2 messages sent. The Without column shows the execution time for the process in the absence of fault tolerance.

We did some independent measurements of the time taken due to checkpointing and comparison. Let T_{chk} be the time taken to checkpoint and perform the checkpoint comparison. Average value of T_{chk} was found to be 0.85 secs, for a checkpoint size of 100 Kbytes.

From Table 1, we can estimate the overhead due to checkpointing, checkpoint comparison, and the Byzantine agreement. Let T_{total} be the total overhead, N be the number

Table 1: Execution times

Num of Msgs	Size (bytes)	CI=2 (secs)	CI=5 (secs)	CI=20 (secs)	Without (secs)
20	5K	7	7	-	6
	10K	9	7	-	
	50K	10	8	-	
	100K	13	10	-	
100	5K	33	26	15	13
	10K	40	29	17	
	50K	57	30	18	
	100K	64	34	19	
200	5K	65	45	31	28
	10K	72	48	33	
	50K	117	54	35	
	100K	125	69	39	

of checkpoints, and T_{byz} be the overhead due to the Byzantine agreement protocol. Table 2 lists some calculations for checkpoint size of 100 Kbytes.

Table 2: Checkpoint Overhead

Num of Msgs	CI=20 (secs)	Without (secs)	T_{total} (secs)	N	T_{chk} (secs)	$N * T_{chk}$ (secs)	T_{byz} (secs)
100	19	13	6	4	0.85	3.4	2.6
200	39	28	11	9	0.85	7.65	3.35

We propose to perform similar experiments using different fault tolerance schemes that we will develop.

2.2 Software Implemented Fault Tolerance in Parallel Computers

Over the past decade the push for higher throughput from existing technology has forced parallel computing systems into the mainstream. To date, hundreds of parallel systems representing millions of invested dollars are currently in use. Table 3 , taken from a recent

Company	Number sold to date	1991 Sales in Millions
Intel	> 325	$90
Meiko Scientific	> 425	25
nCUBE	> 300	18
Parsytec GmbH	100	8
Thinking Machines	90	85
TOTAL	> 1,240	226

Table 3: Sales of Parallel Computers

article in the IEEE Spectrum [12], shows that over one thousand parallel machines have been sold for a total of more than two hundred million dollars. However, in spite of their popularity these systems are largely unsuitable for applications demanding high reliability or prolonged availability due to high failure rates.

To some it may come as a surprise that parallel systems suffer from high failure rates due to the common assumption that such systems are inherently reliable. The following quotation, taken from a recent publication by Harper and Lala [6], points out the error in this logic.

> The assertion is often made that parallel processors are intrinsically reliable, fault tolerant, and reconfigurable due to their multiplicity of processing resources. In fact, the only intrinsic attribute guaranteed by multiple processors is a higher total failure rate.

Thus, it is apparent that parallel systems are in desperate need of fault-tolerant features if they are to provide the same level of dependable service as their uniprocessor ancestors. In all practicality, this need eventually becomes a requirement as parallel systems grow in scale.

Potential Solutions

One solution is to provide some level of redundancy within the hardware. Although advances are being made in the realm of fault-tolerant parallel computing architectures, it will be some time before they are commercially available. Meanwhile, sales of existing parallel systems are growing along with the expansion of their application libraries. When the new architectures finally do hit the market the necessary additional hardware and increased complexity will significantly compound their cost. Therefore, like the necessities that brought parallel computing systems into the forefront, there is the need for cost effective fault-tolerance from existing parallel systems. Software implemented fault-tolerance (SIFT) is one very promising solution. Currently, however, there are two problems plaguing SIFT: there are no easy ways to utilize existing approaches and no easy ways to experiment with new approaches.

There have been numerous approaches proposed for the provision of hardware fault-tolerance within software, such as recovery blocks, duplex, TMR, and checkpointing schemes. However, in order to utilize these schemes programmers must explicitly incorporate them into applications. For parallel programmers this compounds the already daunting task of parallel programming. In addition, little help comes in the form of special programming languages because designers of most parallel languages have largely ignored the issue of fault-tolerance in favor of high performance. Explicit implementation of fault- tolerance schemes within existing applications requires explicit modification, which is a very difficult and costly undertaking. Thus, requiring that parallel programmers explicitly handle fault-tolerance makes its incorporation into new and existing applications intolerably difficult and expensive.

Because of the difficulty realizing SIFT approaches, experimentation and testing of new and existing approaches is severely hampered. Clearly, in order to qualify a new or existing SIFT approach as effective, it must be implemented and tested. Presently, researchers are required to spend countless hours programming new SIFT approaches into specific applications in order to evaluate them. This slows down the research process and prolonging the achievement of inexpensive, effective fault-tolerance on existing parallel systems.

All these difficulties can be overcome by implementing fault-tolerance into an application independent, user-transparent, modular software layer. By handling all of the requisite duplication, comparison, recovery, and synchronization chores necessary for fault-tolerance operation, such a layer could make it possible to execute new and existing applications reliably without explicit programmer intervention. Likewise, the layer could be made flexible enough to allow easy modification of the SIFT approach used, thus making it amenable to experimentation with new approaches.

Research Goal

The main goal of this on-going research is to develop the aforementioned software layer for the purpose of providing dependable operation at minimal cost on numerous existing parallel computing systems, as well as the provision of a flexible framework within which to explore new fault-tolerance approaches.

The following sections present the various aspects of the proposed research. The preliminary requirements of the SIFT layer are discussed.

Preliminary Requirements

If a software implementation is to succeed, it must follow complete and precise design requirements. The following is a collection of the more general preliminary requirements presently established for the SIFT layer.

(1) It should be developed on top of the existing system software. The purpose for this requirement is to prevent the layer from becoming too hardware dependent, which would limit portability and reduce maintainability.

(2) It should provide the user the capability to choose which fault-tolerant approach is to be used. This requirement has a two-fold purpose. First of all, it would allow the user to control the level of fault tolerance according to the importance of the application being executed. Secondly, it would provide researchers the ability to test and compare the characteristics of various fault-tolerant approaches simply and effectively.

(3) It should be independent of the interconnection topology of the target parallel computer. The purpose of this requirement is the same as that for (1).

(4) It should not be application dependent. The purpose for this requirement is to guarantee that the programmer need not handle the fault-tolerance explicitly.

(5) It should allow existing programs to be run without modification. This requirement ensures that the software layer will be transparent to the user.

(6) It should be based on a set of primary functions collectively referred to as the SIFT-kernel. This requirement implies that the layer be divided into two primary modules: the SIFT-functional layer and the SIFT-kernel. The SIFT-kernel should contain routines that provide a uniform, hardware independent interface to the SIFT-functional layer. The SIFT-functional layer should contain only the functions that are specific to the SIFT approach being used and should not bypass the SIFT-kernel to get to the underlying system. This serves two main purposes. First, it increases portability and maintenance by limiting hardware specific modifications to the SIFT-kernel only. Secondly, it requires that the layer be constructed in a modular design so that new fault-tolerant approaches can be easily constructed from available function primitives.

(7) It should be amenable to formal representation for the purpose of formal verification. This requires that the program designers develop a clean implementation that can be formally represented by means of a set of assertions for the purpose of revealing implementation and design faults utilizing function-deterministic tests. This is a very important requirement since the level of dependability provided to the application level is only as good as the dependability of the SIFT layer. Thus, it is imperative that the layer be thoroughly tested for intrinsic faults.

(8) It should permit fault injection for the purpose of dependability validation. This requirement ensures that there is a means for validating the software layer.

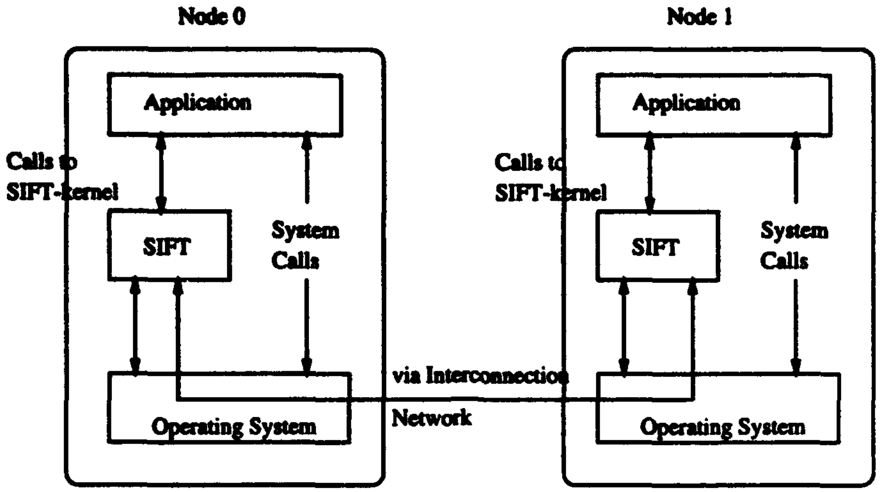

Figure 2: Example of the SIFT Layer Utilizing A Duplex Approach

From the preliminary requirements, it is possible to visualize the relationship and interaction between a user application, the proposed SIFT layer, and the operating system. For example, Figure 2 shows two nodes of a multiprocessing computer running a user application on top of the proposed SIFT layer. In this example, the SIFT layer is configured to use the duplex approach.

As is apparent from the figure, the application program has direct access to the operating system for system calls not pertinent to fault tolerance (such as the acquisition of a file handle) but calls that do require special attention (such as the spawning of a new process) are intercepted by the SIFT layer and handled transparently. When it is necessary to synchronize the duplexed applications or make a comparison, the peer SIFT layers can communicate over the interconnection network using ordinary system calls unbeknownst to the duplexed applications.

Example SIFT Approaches

As was stressed earlier, one objective of the proposed SIFT layer is that it be flexible enough to allow the implementation and testing of limitless new and existing approaches to fault-tolerance. Some examples of the possible SIFT approaches that could be implemented and tested within the proposed SIFT layer are duplex, TMR, New Roll-Forward checkpointing [10], and Skew Roll-Forward checkpointing.

Example Implementation of the SIFT Layer on the nCUBE

The nCUBE has been chosen as the target machine for the initial development of the SIFT layer because Texas A&M is currently in possession of a 64-node nCUBE 2. The following is a brief overview of relevant characteristics of the nCUBE.

The nCUBE computer is a distributed memory, message passing multiprocessor. Its processing nodes are connected via a hypercube interconnection network. Every node within the network shares the same system clock, but each executes instructions from its own memory independently from the rest. However, it is possible to synchronize programs running on separate nodes through an exchange of messages.

The nCUBE is not a stand-alone computer, but requires a host to act as a user interface. The host communicates with the nCUBE via a program on a special Platform Interface Board called VORTEX. Each node within the nCUBE runs its own copy of a UNIX-like operating system called VERTEX. This operating system provides a program the capability to start a new process, duplicate a current process, suspend a process, and restart a process.

Programs can be loaded onto the nodes from three different environments: a shell on the host computer, a program running on the host, and a program running on a node. Parallel programs written for the nCUBE typically consist of a collection of program elements that execute on separate processors, each accomplishing their own portion of the overall task. These program elements can also communicate with each other over the interconnection network.

The VERTEX operating system contains a number of basic commands and features that together form a sturdy platform for the SIFT- kernel. Notable among these are commands that do the following: (i) allow a process on one node to load and execute a process on another node, (ii) synchronize processes on separate nodes using messages, (iii) exactly duplicate a running process, and (iv) suspend and restart a process. In addition, the nCUBE system software includes an interactive source and symbolic debugger that allows programmers to examine variables, data, call stacks, procedure arguments, message queues, and registers, thus providing the means for software testing. Thus it is evident that the VERTEX operating system provides some functionality necessary for the realization of the SIFT-kernel layer.

2.3 The Reliable Architecture Characterization Tool

Synthesizing an architecture and evaluating its dependability are perhaps the two greatest challenges which the designers of fault-tolerant computing systems must face. Many highly dependable systems being realized today utilize one of several proven configurations consisting of redundant processors and memories interconnected with some form of logic to detect, correct or mask errors and remove failed modules. Systems making use of N-modular redundancy, duplication and comparison, standby sparing or system-wide coding are familiar members of this class of fault-tolerant multiprocessor architectures. The reliability and availability metrics of these systems have typically been validated either through combinatorial approaches such as fault-trees and reliability block diagrams, or through Markov modeling. The procedure for the synthesis and evaluation of a fault-tolerant architecture is frequently iterative in nature, terminating when the particular system design has been optimized to meet specifications for dependability, performance, cost, size, weight and power consumption. It is therefore essential that automated methods of analyzing this family of reliable architectures be at hand to assist in the design procedure.

A REliable Architecture Characterization Tool (REACT) is currently being developed to meet this need for a generalized simulation tool which can analyze the high-level dependability metrics of a variety of fault-tolerant computer designs [2]. Incorporating detailed system, workload and fault/error models into the integrated framework of a testbed, this software can be more accurate and easier to use than many tools based on analytical approaches. Because it facilitates precise estimation of reliability and availability early in the concept and design phase of system development, this tool will potentially enable the engineer to synthesize an architecture which better matches specifications than possible with the more traditional analytical techniques.

We are currently extending REACT to aid in reliability and availability evaluation of multiprocessor and distributed systems.

Features of REACT

REACT is a software testbed that performs automated life testing of many user-defined multiprocessor architectures through simulated fault-injection. 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 measured. The censoring time dictates the maximum operational lifetime of interest for the given system. Those trials in which the system remains functional beyond the censoring time are terminated, thereby shortening the run-time of the simulation without affecting the measurements of interest. Extensive instrumentation has been included in the program to collect data from each trial, which is later aggregated over the entire simulation run in order to generate the outputs. Graphs of reliability or availability, a comprehensive failure mode report and various statistical measurements are provided as output by REACT. The software now consists of approximately 10000 lines of C running under UNIX and completes a "typical" simulation run in a few hours on an engineering workstation.

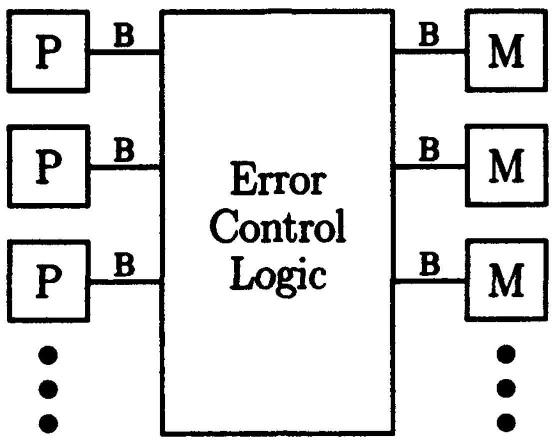

Figure 3: Class of Architectures REACT Can Analyze

System Model

Presently, REACT can analyze the class of architectures consisting of one or more processor modules (P) interconnected via buses (B) to one or more memory modules (M) through a block of error-control logic, as pictured in Figure 3. Any number of processors and memories may be specified and each can be designated as initially active or a hot or cold standby spare. Homogeneous *groups* of processors or memories may be defined in which all modules operate redundantly. A group of processors execute the same workload in lock-step synchronization; memories in a group have identical contents and are accessed simultaneously.

The error-control logic may be built from various combinations of components often found in fault-tolerant designs such as voters, comparators, switches and error detecting/correcting codes. Custom error-control logic circuitry may also be specified by the user. This flexibility allows the system model to represent a variety of multiprocessor designs utilizing multiple levels of passive, active or hybrid redundancy, or coding to achieve fault-tolerance.

A functional-level abstraction is used in modeling the operation of both processor and memory modules. The state of a processor is defined by the values driven on its data and address buses and is determined by inputs it receives over the bus. The state of a memory is defined by the contents of its bit-array, with the functionality of its addressing-logic being simulated on every access. In order to reduce some of the unnecessary complexity in the system model, logic values 0 and 1 are not differentiated: only "error-free" and "erroneous" states exist for each bit. Memory depth is variable and word width for memory and all data paths may be changed with minor modifications in the code.

Workload Model

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. REACT is an event-driven simulator, since only those computation cycles in which errors both propagate and change the erroneous state of the system need to be simulated.

Because several years of system operation may be simulated during a single trial, average workload characteristics are utilized. 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. By definition, one memory read to fetch an instruction is made every computation cycle. 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 have proven to be representative of memory access behavior [1]. This locality model suggests that $\alpha \times 100\%$ of all accesses go to $\beta \times 100\%$ of the memory under the condition $\alpha + \beta = 1$. Heising first reported what was deemed an "80/20 Rule" when parameter values $\alpha = 0.8$ and $\beta = 0.2$ were observed to hold for many commercial applications [7]. Reference addresses are assumed to be uniformly distributed inside and outside of the locality; no attempt is made to separate code from data in memory with the model.

Fault and Error Model

The fault and error model employed by REACT accounts for permanent, intermittent and transient faults in the processors plus permanent and transient faults in the memories and the error-control logic. Faults with inter-arrival times that are sampled from a Weibull distribution (of which the exponential distribution is a subset) are injected into these modules only at the beginning of a computation cycle. Repair times for failed modules have a log-normal distribution after a fixed logistics delay.

The exact behavior of a processor in the presence of faults can only be determined with a complex architectural model for that particular processor. In order to preserve generality of the testbed, detailed knowledge of the processor architecture is not mandatory. Instead, it is assumed that processor fault effects are 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 addresses, (write) data, or both addresses and data simultaneously. An erroneous address is assumed to access a random memory location while erroneous data take on a random value. In addition, erroneous processor reads are assumed to generate output errors in the same computation cycle. Several fault-injection experiments on actual processors have obtained results which support this functional processor abstraction by providing measurements for its parameters [3, 5, 9].

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, corresponding to the difference in bit values between 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).

Extensions of REACT for Multiprocessor and Distributed Systems

REACT is a very useful tool for evaluating a fault tolerant architecture that can be modeled using Figure 3. Such an architecture is suitable to implement a single node in a distributed or multiprocessor system. The architecture used for a node determines which fault model may be used for that node (i.e. fail-stop, fail-slow, or Byzantine). Presently, REACT is useful to evaluate a single such node. We propose to extend REACT to allow evaluation of a multiprocessor or distributed system consisting of multiple such nodes. We propose that the extended REACT will take into account the fault tolerance scheme used by an application executing on the multiple nodes. Thus, reliability and availability will not only be function of the architecture of each individual node, but also depend on the fault tolerance mechanism used by the application.

3 Synopsis of Future Research

The thrust of our future research is on issues related to fault tolerant multiprocessor and distributed systems.

3.1 Fault Tolerance Schemes for Multiprocessor and Distributed Systems

Design and implementation of fault tolerance schemes for multiprocessor and distributed systems is a thrust area of this proposal. We propose to investigate a number of fault tolerance schemes for multiprocessor and distributed systems to evaluate the performance, reliability and availability trade-offs. The fault tolerance mechanism used in such systems must be chosen based on a number of criterion. The criterion that we consider important are as follows.

- Reliability and availability requirements. These requirements have a serious impact on the level of redundancy required. These requirements may be specified probabilistically (e.g. availability of 99.2%) or deterministically (e.g. tolerate up to two simultaneous *failures*).
- The fault model. The fault models that are applicable to most real-life systems are: (i) fail-stop model. (ii) fail-slow model. (iii) arbitrary or Byzantine failure model. We propose to investigate fault tolerance schemes for all the three models.
- The application. Two types of applications are of particular interest: (i) long-running applications which are expected to provide results at the end of computation (e.g. distributed simulations, weather-forecasting, etc.) (ii) applications that are long-running but are also expected to provide results often during the computation. The requirements of these two application areas are somewhat different, requiring different fault tolerance techniques.

A goal of the proposed research is to provide fault tolerance approaches to match various reliability and application requirements, and experimentally evaluate the performance of the proposed fault tolerance mechanisms. We propose to develop a testbed to implement a wide range of fault tolerance schemes for multiprocessor and distributed environments. A goal being to to provide a common basis for experimental evaluation and comparison of various schemes.

3.2 Software-Implemented Fault Tolerance for Multiprocessors

A major goal of the proposed research is to develop a software layer for the purpose of providing dependable operation at minimal cost on numerous existing parallel computing systems (such as nCUBE and MASSPAR), as well as the provision of a flexible framework within which to explore new fault-tolerance approaches. We propose to provide user-transparent Software-Implemented Fault Tolerance (SIFT) for hardware failures. We propose to develop a SIFT layer that will be located between the operating system and the user application. The following is a collection of requirements that will be satisfied by the SIFT layer.

(1) The SIFT layer should reside on top of the existing system software.

(2) It should allow the user to choose the fault-tolerant approach to be used.

(3) It should be independent of the interconnection topology of the target parallel computer.

(4) It should be application independent.

(5) It should allow existing programs to be run without modification.

(6) It should be based on a set of primary functions collectively referred to as the SIFT-kernel.

(7) It should be amenable to formal representation for the purpose of formal verification.

(8) It should permit fault injection for the purpose of dependability validation.

Due to the availability of the nCUBE machine at Texas A&M University, we are developing the SIFT-layer on nCUBE. This layer will also be able to be ported to other multiprocessors.

3.3 Analysis Tool for Multiprocessor and Distributed Systems

The different fault tolerance mechanisms proposed to be developed during the course of this research need to be evaluated to determine the level of reliability and availability achieved using those mechanisms. We have already developed a tool, REACT, which is very useful for evaluating most fault tolerant architectures used to implement a single node in a multiprocessor or distributed system. The architecture used for a node determines which fault model may be used for that node (i.e. fail-stop, fail-slow, or Byzantine). Presently, REACT is useful to evaluate a single such node.

We are currently in the process of extending REACT to allow evaluation of a multiprocessor or distributed system consisting of multiple such nodes. The extended REACT takes into account the fault tolerance scheme used by an application executing on the multiple nodes. Thus, reliability and availability will not only be function of the architecture of each individual node, but also depend on the fault tolerance mechanism used by the application.

In summary, this tool is useful in evaluating the reliability achieved by the fault tolerance schemes that we will propose.

References

[1] R. B. Bunt, J. M. Murphy, and S. Majumdar, "A measure of program locality and its application," in *Proceedings of the 1984 Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 28–40, ACM, August 1984.

[2] J. A. Clark and D. K. Pradhan, "REACT: A synthesis and evaluation tool for fault-tolerant multiprocessor architectures," in *Proceedings of the 1993 Annual Reliability and Maintainability Symposium*, pp. 428–435, IEEE, January 1993.

[3] P. Duba and R. K. Iyer, "Transient fault behavior in a microprocessor, a case study," in *Proceedings of the 1988 International Conference on Computer Design*, pp. 272–276, IEEE, October 1988.

[4] W. K. Fuchs, K.-L. Wu, and J. A. Abraham, "Low-cost comparison and diagnosis of large remotely located files," in *Fifth Symp. Reliab. in Distr. Soft. & Database Sys.*, pp. 67–73, January 1986.

[5] U. Gunneflo, J. Karlsson, and J. Torin, "Evaluation of error detection schemes using fault injection by heavy-ion radiation," in *Digest of papers: The* 19^{th} *Int. Symp. Fault-Tolerant Comp.*, pp. 340–347, IEEE, June 1989.

[6] R. E. Harper and J. H. Lala, "Fault-tolerant parallel processor," *Journal of Guidance, Control, and Dynamics*, vol. 14, no. 3, pp. 554–563, 1990.

[7] W. P. Heising, "Note on random addressing techniques," *IBM Systems Journal*, vol. 2, pp. 112–116, June 1963.

[8] D. B. Johnson and W. Zwaenepoel, "Sender-based message logging," in *Digest of papers: The* 17^{th} *Int. Symp. Fault-Tolerant Comp.*, pp. 14–19, June 1987.

[9] D. Lomelino and R. K. Iyer, "Error propagation in a digital avionic processor, a simulation-based study," in *Proceedings of the 7th International Real Time Systems Symposium*, pp. 218–225, IEEE, December 1986.

[10] 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.

[11] N. H. Vaidya and D. K. Pradhan, "A fault tolerance scheme for a system of duplicated communicating processes," in *IEEE Workshop on Fault Tolerant Parallel and Distributed Systems*, pp. 98–104, July 1992.

[12] G. Zorpette, "The power of parallelism," *IEEE Spectrum*, pp. 28–33, September 1990.

4 Publications Supported by AFOSR grant #F49620-92-J-0383DEF

In Journals

1. "Issues in Fault Tolerant Memory Management", (with N. Bowen), *IEEE Transactions on Computers*, to appear.

2. "A Survey of Fault-Injection Experimentation for Validating Computer System Dependability", (with J. Clark), *IEEE Computer Magazine*, to appear.

3. "Can Concurrent Checkers Help BIST?", (with S. Gupta), *IEEE Transactions on Computers*, to appear.

4. "Degradable Byzantine Agreement," (with N. Vaidya), *IEEE Transactions on Computers*, to appear.

5. "Safe System Level Diagnosis", (with N. Vaidya), *IEEE Transactions on Computers*, to appear.

6. "Communication Structures in Fault-Tolerant Distributed Systems," (with Fred J. Meyer), *NETWORKS*, Vol. 23, pp. 379-389, 1993.

7. "Yield Optimization of Redundant Multimegabit RAM's Using the Center-Satellite Model," (with D. Das Sharma and F. Meyer), *IEEE Transactions on VLSI Systems*, to appear.

8. "The Hyper-deBruijn Networks: Scalable Versatile Architecutre," (with E. Ganesan), *Transactions on Parallel and Distributed Systems*, Vol. 4, No. 9, pp. 962-978, September 1993.

9. "The Effect of Memory-Management Policies on System Reliability", (with N. Bowen), *IEEE Transactions on Reliability*,Vol. 42, No. 3, pp. 375 - 383, September 1993.

10. "Processor and Memory Based Checkpoint and Rollback Recovery," (with N.S. Bowen), *COMPUTER*, pp. 22-31, February 1993.

11. "Recursive Learning: A Precise Implication Procedure and its' application to Test Pattern Generation in Digital Circuits" (with W. Kunz), *IEEE Transactions on Computer-Aided Design*, to appear.

12. "A Hybrid Memory Structure and Algorithms for Improved Fault Tolerance" (with N.S. Bowen), *IEEE Transactions on Computers*, to appear.

13. "Modeling Live and Dead Lines in Cache Memory Systems" (with D. Thiebaut and A. Mendelson), *IEEE Transactions on Computers*, Vol. 42, No. 1, pp. 1-14, January 1993.

14. "A New Algorithm for Rank-Order Filtering and Sorting" (with Barun Kar), *IEEE Transactions on ASSP*, Vol. 41, No. 8, pp. 2688-2694, August 1993.

15. "Virtual Checkpoints: Architecture and Performance" (with N.S. Bowen), *IEEE Transactions on Computers*, Vol. 41, pp. 516-525, May 1992.

16. "Accelerated Dynamic Learning for Test Pattern Geeration in Combinational Circuits" (with W. Kunz), *IEEE Transactions on Computer-Aided Design*, Vol. 12., No. 5, pp. 684-694, May 1993.

17. "Survey of Checkpoint and Rollback Recovery Techniques", (with N. Bowen), *Computer*, Vol. 26, No. 2, pp. 22-31, February 1993.

18. "Fault-Tolerant Design Strategies for High Reliability and Safety" (with N. Vaidya), *IEEE Transactions on Computers*, Vol. 42, No. 10, October 1993.

19. "A New Class of Bit and Byte Error Control Codes" (with N. Vaidya), *IEEE Transactions on Infomation Theory*, Sept. 1992

In Conference Proceedings

1. "Job Scheduling in Mesh Multicomputers",(with D. Das Sharma), accepted in *1994 International Conference on Parallel Processing.*

2. "Subcube Level Time-Sharing in Hypercube Multicomputers",(with D. Das Sharma and G. D. Holland), accepted in *1994 International Conference on Parallel Processing.*

3. "Synthesis of Initializable Asynchronous Circuits", (with S Chakradhar, S. Banerjee and R. Roy), *International Conference on VLSI Design*, Calcutta, India, December 1993.

4. "Recovery in Distributed Mobile Environments" (with P. Krishna and N.H. Vaidya), *IEEE Workshop on Advances in Parallel and Distributed Systems*, October 1993.

5. "A Method to Derive Compact Test Sets for Path Delay Faults in Combinational Circuits" (with J. Saxena), *1993 International Conference on Computer-Design*, Cambridge, Massachusetts, pp. 518-522, October 4-6, 1993.

6. "Design for Testability of Asynchronous Sequential Circuits", (with J. Saxena), *International Test Conference*, Baltimore, October 17-21, 1993.

7. "Fast and Efficient Strategies for Cubic and Non-Cubic Allocation in Hypercube Multiprocessors", (with D. Das Sharma), *1993 International Conference on Parallel Processing*, Chicago, August 1993.

8. "A Synthesis and Evaluation Tool for Fault-Tolerant Multiprocessor Architectures" (with J. Clark), *Annual Reliability and Maintainability Symposium*, pp. 428-435, January 1993.

9. "Buffer Assignment for Date Driven Architectures," (with M. Chatterjee), *International Conference on Computer Aided Design '93*, November 1993.

10. "A Fast and Efficent Strategy for Submesh Allocation in Mesh-Connected Parallel Computers, (with D. Das Sharma), *5th IEEE Symposium on Parallel and Distributed Processing*, December 1993.

11. "Optimal Broadcasting in de Bruijn Networks and Hyper-de Bruijn Networks" (with E. Ganesan), *International Parallel Processing Symposium*, April 1993.

12. "Degradable Agreement in the Presence of Byzantine Faults" (with N. Vaidya), *13th International Conference on Distributed Computing Systems*, Pittsburgh, Pennsylvania, May 1993.

5 Vitae of the Principal Investigator

CURRICULUM VITAE

Dhiraj K. Pradhan
H. R. Bright Building
Department of Computer Science
Texas A&M University
College Station, TX 77843
Tel: (409) 862-2438
Fax: (409) 847-8578
Email: pradhan@cs.tamu.edu

Positions—Academic

1992 – present	COE Endowed Chair Professor, Department of Computer Science, Texas A&M University, College Station, Texas.
1983 – 1992	Professor and Coordinator of Computer Systems Engineering, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, Massachusetts.
1978 – 1982	Associate Professor, School of Engineering, Oakland University, Rochester, Michigan.
1979	Research Associate Professor, Stanford University, Stanford, California.
1973 – 1978	Associate Professor, Department of Computer Science, University of Regina, Regina, Canada. (1973–1976, Assistant Professor).

Positions—Industrial

1972 – 1973	Staff Engineer, IBM, Systems Development Laboratory, Poughkeepsie, New York.

Honors

1990	*Humboldt Distinguished Senior Scientist Award,* Germany
1989	*Fellow,* Japan Society for Promotion of Science
1988	*Fellow,* IEEE, "For contributions to techniques and theory of designing fault-tolerant circuits and systems"

Education

1972, Ph.D. (Electrical Engineering), University of Iowa
Iowa City, Iowa.

1970, M.S. (Electrical Engineering), Brown University,
Providence, Rhode Island.

Personal

Born on December 1, 1948, Married, Five Children, U.S. Citizen

Professional Activities

1994 –	"Professor Arun Kumar Choudhury Best Paper Award, The Seventh International Conference on VLSI Design, Calcutta, India, January 5-8, 1994.
1993 –	*Program Chair,* 10th and 11th IEEE VLSI Test Symposium
1992 –	*Advisory Committee,* IEEE Technical Committee on Parallel Processing
1992 –	*Conference Chair,* 22nd International Symposium on Fault-Tolerant Computing Boston, Massachusetts
1991 –	*Editor, IEEE Transactions on Computers*
1990 – 1993	*ACM Lecturer*
1990 – 1993	*IEEE Distinguished Visitor,* Computer Society
1990 –	*Editor, IEEE Computer Society Press*
1990 –	*Keynote Speaker,* International Symposium on Fault-Tolerant Systems and Diagnostics, Varna, Bulgaria
1989 –	*Associate Editor, Journal of Circuits, Systems and Computers* World Scientific Publishing Co., New Jersey
1988 –	*Editor, Journal of Electronic Testing, Theory and Applications* Kluwer Academic Publishers, Boston
1987	*Co-Chairperson,* IEEE Workshop on Fault-Tolerant Distributed and Parallel Systems, San Diego, California
1986	*Guest Editor, IEEE Transactions on Computers,* Special Issue on Fault-Tolerant Computing, April 1986
1986 – 1988	*Editor, Advances in VLSI Systems,* Computer Science Press, Maryland
1982 – 1985	*IEEE Distinguished Visitor,* Computer Society

1982 –	Consultant to Mitre, CDC, IBM, AT&T, DEC and Data General
1981 – 1988	*Editor, Journal of VLSI and Digital Systems,* Computer Science Press, Maryland
1980	*Guest Editor,* Special Issue on Fault-Tolerant Computing, *IEEE Computer,* March 1980
1980 – 93	*Member of Program Committee* for Fault-Tolerant Computing Symposium, Computer Architecture Conference and other conferences Chaired sessions and organized panel discussions at various international conferences

Grants

1973 – present	Multiple grants from NSF, AFOSR, ONR, SRC, Bendix, IBM and NRC (Canada); supported continuously, $50,000–$200,000 per year.

Research Supervision

1978 – present	Several Ph.D. Students, placed in IBM, AT&T as well as leading universities
1977 –	Research Associates: K.L. Kodandapani T. Nanya K. Matsui I. Koren D. Avresky F. Meyer J. Jain

Patents

"Easily Testable High Speed Architecture for Large RAMs", U.S. Patent No. 4,833,677, May 23, 1989.

"DeBruijn Graph Based VLSI Viterbi Decoder", Application No. 904341, June 18, 1992.

List of Publications

Text Book

Fault-tolerant Computing: Theory and Techniques (Editor and Co-Author), Vol. I and Vol. II, Prentice-Hall, Inc., May 1986 (Second Edition to appear 1993). *VLSI Testing and Design for Testabilita,* (with W. Kunz and E.J. McCluskey, under preparation.

Book Chapters

Communication Structures in Fault-Tolerant Distributed Systems, (with F.J. Meyer), in *Hardware and Software Fault-Tolerance in Parallel Computing Systems,* Ellis Horwood, Simon & Schuster International Group, Chichester, England, D. Avresky, Editor, August 1992.

New Roll-Forward Checkpointing Schemes for Modular Checkpointing Schemes for Modular Redundant Systems, (with N. Vaidya), in *Hardware and Software Fault-Tolerance in Parallel Computing Systems,* Ellis Horwood, Simon & Schuster International Group, Chichester, England, D. Avresky, Editor, August 1992.

Computer Reliability, (with J. Stiffler) in *Encyclopedia of Microcomputers,* Vol. 4, Marcel Dekker, Inc., New York, 1990.

In Journals

1. "Issues in Fault Tolerant Memory Management", (with N. Bowen), *IEEE Transactions on Computers*, to appear.
2. "A Survey of Fault-Injection Experimentation for Validating Computer System Dependability", (with J. Clark), *IEEE Computer Magazine*, to appear.
3. "Can Concurrent Checkers Help BIST?", (with S. Gupta), *IEEE Transactions on Computers*, to appear.
4. "Degradable Byzantine Agreement," (with N. Vaidya), *IEEE Transactions on Computers*, to appear.
5. "Safe System Level Diagnosis", (with N. Vaidya), *IEEE Transactions on Computers*, to appear.
6. "Communication Structures in Fault-Tolerant Distributed Systems," (with Fred J. Meyer), *NETWORKS*, Vol. 23, pp. 379-389, 1993.
7. "Yield Optimization of Redundant Multimegabit RAM's Using the Center-Satellite Model," (with D. Das Sharma and F. Meyer), *IEEE Transactions on VLSI Systems*, to appear.
8. "The Hyper-deBruijn Networks: Scalable Versatile Architecutre," (with E. Ganesan), *Transactions on Parallel and Distributed Systems*, Vol. 4, No. 9, pp. 962-978, September 1993.
9. "The Effect of Memory-Management Policies on System Reliability", (with N. Bowen), *IEEE Transactions on Reliability*,Vol. 42, No. 3, pp. 375 - 383, September 1993.

10. "Processor and Memory Based Checkpoint and Rollback Recovery," (with N.S. Bowen), *COMPUTER*, pp. 22-31, February 1993.

11. "Recursive Learning: A Precise Implication Procedure and its' application to Test Pattern Generation in Digital Circuits" (with W. Kunz), *IEEE Transactions on Computer-Aided Design*, to appear.

12. "A Hybrid Memory Structure and Algorithms for Improved Fault Tolerance" (with N.S. Bowen), *IEEE Transactions on Computers*, to appear.

13. "Modeling Live and Dead Lines in Cache Memory Systems" (with D. Thiebaut and A. Mendelson), *IEEE Transactions on Computers*, Vol. 42, No. 1, pp. 1-14, January 1993.

14. "A New Algorithm for Rank-Order Filtering and Sorting" (with Barun Kar), *IEEE Transactions on ASSP*, Vol. 41, No. 8, pp. 2688-2694, August 1993.

15. "Virtual Checkpoints: Architecture and Performance" (with N.S. Bowen), *IEEE Transactions on Computers*, Vol. 41, pp. 516-525, May 1992.

16. "Accelerated Dynamic Learning for Test Pattern Geeration in Combinational Circuits" (with W. Kunz), *IEEE Transactions on Computer-Aided Design*, Vol. 12., No. 5, pp. 684-694, May 1993.

17. "Survey of Checkpoint and Rollback Recovery Techniques", (with N. Bowen), *Computer*, Vol. 26, No. 2, pp. 22-31, February 1993.

18. "Fault-Tolerant Design Strategies for High Reliability and Safety" (with N. Vaidya), *IEEE Transactions on Computers*, Vol. 42, No. 10, October 1993.

19. "A New Class of Bit and Byte Error Control Codes" (with N. Vaidya), *IEEE Transactions on Infomation Theory*, Sept. 1992

20. "Yield Optimization in Large RAMs with Hierarchical Redundancy" (with K.N. Ganapathy and A.D. Singh), *IEEE Journal of Solid State,* Vol. 26, No. 9, pp. 1259-1264, September 1991.

21. "A New Framework for Designing and Analyzing BIST Techniques and Zero Aliasing Compression" (with S.K. Gupta), *IEEE Transactions on Computers*, Vol. 40, No. 6, pp. 743–763, June 1991.

22. "Consensus with Dual Mode Failures" (with F.J. Meyer), *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 2, pp. 214-222, April 1991.

23. "Error Correcting Codes in Fault-Tolerant Computers" (with E. Fujiwara), *Computer*, Vol. 23, No. 7, pp. 63–72, July 1990.

24. "Aliasing Probability for a Multiple Input Signature Analyzer and a New Compression Technique" (with S. Gupta and M. Karpovsky), *IEEE Transactions on Computers*, Vol. 39, pp. 586–591, April 1990.

25. "Organization and Analysis of Gracefully-Degrading Inter-leaved Memory Systems" (with K. Saluja, G. Sohi and K. Cheung), *IEEE Transactions on Computers*, Vol. 39, No. 1, pp. 63–71, January 1990.

26. "Modeling Defect Spatial Distribution" (with F.J. Meyer), *IEEE Transactions on Computers*, Vol. 38, No. 4, pp. 538–546, April 1989.

27. "The DeBruijn Multiprocessor Networks: A Versatile Parallel Processing Network for VLSI" (with M. Samatham), *IEEE Transactions on Computers*, Vol. 38, No. 4, pp. 567–581, April 1989.

28. "Dynamic Testing Strategy for Distributed System" (with F.J. Meyer), *IEEE Transactions on Computers*, Vol. 38, No. 3, pp. 356–365, March 1989.

29. "TRAM: A Design Methodology for High Performance Testable Large RAMs" (with N. Jarwala), *IEEE Transactions on Computers*, Vol. C-37, No. 10, pp. 1235–1250, October 1988.

30. "Designing Interconnection Buses in VLSI and WSI for Maximum Yield and Minimum Delay" (with I. Koren and Z. Koren), *IEEE Journal of Solid State Circuits*, Vol. 23, pp. 859–866, June 1988.

31. "Flip Trees: A Fault-Tolerant Network with Wide Containers" (with F.J. Meyer), *IEEE Transactions on Computers*, Vol. 37, No. 4, pp. 472–478, April 1988.

32. "Modeling the Effect of Redundancy on Yield and Performance of VLSI Systems" (with I. Koren), *IEEE Transaction on Computers*, Vol. C-36, No. 3, pp. 344–355, April 1987.

33. "Yield and Performance Enhancement through Redundancy in VLSI and WSI Multiprocessor Systems" (with I. Koren), *IEEE Proceedings*, Vol. 74, No. 5, pp. 699–711, May 1986.

34. "Dynamically Restructurable Fault-tolerant Processor Network Architectures", *IEEE Transactions on Computers*, Vol. C-34, No. 5, pp. 434–447, May 1985.

35. "Fault-tolerant Multiprocessor Structures", *IEEE Transactions on Computers*, Vol. C-34, No. 1, pp. 33–45, January 1985.

36. "Synthesis of Directed Multi-Commodity Flow Problems" (with A. Itai), *Networks*, Vol. 14, pp. 213–224, 1984.

37. "Sequential Network Design Using Extra Inputs for Fault Detection", *IEEE Transactions on Computers*, Vol. C-32, No. 3, pp. 319–323, March 1983.

38. "A Fault-Tolerant Distributed Processor Communication Architecture" (with S. Reddy), *IEEE Transactions on Computers*, Vol. C-31, No. 9, pp. 863–870, September 1982.

39. "A Class of Unidirectional Error Correcting Codes", *IEEE Transactions on Computers*, Special Issue on Fault-Tolerant Computing, Vol. C-32, No. 6, pp. 564–568, June 1982.

40. "A Uniform Representation of Permutation Networks Used in Memory-Processor Interconnection" (with K.L. Kodandapani), *IEEE Transactions on Computers*, Special Issue on Parallel Processing, Vol. C-29, No. 9, pp. 777–791, September 1980.

41. "A New Class of Error Correcting-Detecting Codes for Fault-Tolerant Computer Applications", *IEEE Transactions on Computers*, Special Issue on Fault–Tolerant Computing, Vol. C-29, No. 6, pp. 471–481, June 1980.

42. "Error-Correcting Codes and Self-Checking Circuits" (with J.J. Stiffler), *IEEE Computer*, Special Issue on Fault-Tolerant Computing, Vol. 13, No. 3, pp. 27–38, March 1980.

43. "Undetectability of Bridging Faults and Validity of Stuck-at Fault Test Sets" (with K.L. Kodandapani), *IEEE Transactions on Computers*, Vol. C-29, No. 1, p. 55–59, January 1980.

44. "Fault-Tolerant Asynchronous Networks Using Read-Only Memories", *IEEE Transactions on Computers*, Vol. C–27, No. 7, pp. 674–679, July 1978.

45. "Fault Secure Asynchronous Networks", *IEEE Transactions on Computers*, Vol. C–27, No. 5, pp. 396–404, May 1978.

46. "A Theory of Galois Switching Functions", *IEEE Transactions on Computers*, Vol. C–27, No. 3, pp. 239–249, March 1978.

47. "Universal Test Sets for Multiple Fault Detection in AND-EXOR Arrays", *IEEE Transaction on Computers*, Vol. C–27, No. 2, pp. 181–187, February 1978.

48. "Store Address Generator with Built-In Fault Detection Capabilities" (with M.Y. Hsiao & A.M. Patel), *IEEE Transactions on Computers*, Vol. C–26, No. 11, pp. 1144–1147, November 1977.

49. "A Graph-Structural Approach for the Generalization of Data Management Systems", *Information Sciences*, American Elesevier Publishing Company, Inc., pp. 1–17, March 1977.

50. "Techniques to Construct (2,1) Separating Systems from Linear Codes" (with S.M. Reddy), *IEEE Transactions on Computers*, Vol. C–25, No. 9, pp. 945–949, September 1976.

51. "Reed-Muller Canonic Forms for Multivalued Functions" (with A.M. Patel), *IEEE Transactions on Computers*, Vol. C–24, No. 2, pp. 206–220, February 1975.

52. "Fault-Tolerant Carry Save Adders", *IEEE Transactions on Computers*, Vol. C–23, No. 11, pp. 1320–1322, November 1974.

53. "Design of Two-Level Fault-Tolerant Networks" (with S.M. Reddy), *IEEE Transactions on Computers*, Vol. C–23, No. 1, pp. 41–48, June 1974.

54. "Fault-Tolerant Asynchronous Networks" (with S.M. Reddy), *IEEE Transactions on Computers*, Vol. C–22, No. 7, pp. 662–669, July 1973.

55. "Error Correcting Techniques for Logic Processors" (with S.M. Reddy), *IEEE Transactions on Computers*, Vol. C–21, No. 12, pp. 1331–1335, December 1972.

In Conference Proceedings

1. "Synthesis of Initializable Asynchronous Circuits", (with S Chakradhar, S. Banerjee and R. Roy), *International Conference on VLSI Design*, Calcutta, India, December 1993.

2. "Recovery in Distributed Mobile Environments" (with P. Krishna and N.H. Vaidya), *IEEE Workshop on Advances in Parallel and Distributed Systems*, October 1993.

3. "A Method to Derive Compact Test Sets for Path Delay Faults in Combinational Circuits" (with J. Saxena), *1993 International Conference on Computer-Design*, Cambridge, Massachusetts, pp. 518-522, October 4-6, 1993.

4. "Design for Testability of Asynchronous Sequential Circuits", (with J. Saxena), *International Test Conference*, Baltimore, October 17-21, 1993.

5. "Fast and Effi ient Strategies for Cubic and Non-Cubic Allocation in Hypercube Multiprocessors", (with D. Das Sharma), *1993 International Conference on Parallel Processing*, Chicago, August 1993.

6. "A Synthesis and Evaluation Tool for Fault-Tolerant Multiprocessor Architectures" (with J. Clark), *Annual Reliability and Maintainability Symposium*, pp. 428-435, January 1993.

7. "Buffer Assignment for Date Driven Architectures," (with M. Chatterjee), *International Conference on Computer Aided Design '93*, November 1993.

8. "A Fast and Efficent Strategy for Submesh Allocation in Mesh-Connected Parallel Computers, (with D. Das Sharma), *5th IEEE Symposium on Parallel and Distributed Processing*, December 1993.

9. "Optimal Broadcasting in de Bruijn Networks and Hyper-de Bruijn Networks" (with E. Ganesan), *International Parallel Processing Symposium*, April 1993.

10. "Degradable Agreement in the Presence of Byzantine Faults" (with N. Vaidya), *13th International Conference on Distributed Computing Systems*, Pittsburgh, Pennsylvania, May 1993.

11. "A Novel Approach for Subcube Allocation in Hypercube Multiprocessor", (with D. Das Sharma), *IEEE Symposium on Parallel and Distributed Processing*, pp. 336 - 345, Dallas, Texas, 1992.

12. "A Fault Tolerance Scheme for a System of Duplicated Communicating Processes" (with N. Vaidya), *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, Amherst, Massachusetts, July 1992.

13. "Roll-Forward Checkpointing Scheme: Concurrent Retry with Nondedicated Spares" (with N. Vaidya), *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, Amherst, Massachusetts, July 1992.

14. "A Design for Testability Scheme to Reduce Test Application Time in Full Scan" (with Jayashree Saxena), *10th IEEE VLSI Symposium,* Atlantic City, New Jersey, April 1992.

15. "Signature Analysis under a Delay Fault Model" (with Jayashree Saxena), European Conference on Design Automation, pp. 285–290, Brussels, Belgium, March 1992.

16. "A Hierarchical Directory Scheme for Large-Scale Cache Coherent Multiprocessors" (with Y.-C. Maa and D. Thiebaut), 6th International Parallel Processing Symposium, pp. 43–46, Beverly Hills, California, March 1992.

17. "Yield Optimization of Redundant Multimegabit RAM's using the Center-Satellite Model" (with D. Dassharma), IEEE International Conference on Wafer Scale Integration, San Francisco, California, January 1992.

18. "A Virtual Memory Translation Mechanism to Support Checkpoint and Rollback Recovery" (with N.S. Bowen), *Supercomputing '91*, pp. 890–899, November 1991.

19. "Two Economical Directory Schemes for Large-Scale Cache Coherent Multiprocessors" (with Y.-C. Maa and D. Thiebaut), *ACM SIGARCH Computer Architecture News,* pp. 10–18, September 1991.

20. "Technique for Virtual Memory Architecture to Support Checkpoint and Rollback Recovery" (with N.S. Bowen), *IBM Technical Disclosure Bulletin,* Vol. 34, pp. 451–457, September 1991.

21. "High Level Synthesis of Data Driven ASICs" (with B. Patel), *Proc. Fourth Annual IEEE International ASIC Conference & Exhibit — ASIC'91",* Rochester, NY, September 1991.

22. "Program Fault-tolerance Based on Memory Access Behavior" (with N. S. Bowen), *Proc. 1991 International Symposium on Fault-Tolerant Computers,* pp. 426–433, Montreal, Canada, June 1991.

23. "System Level Diagnosis: Combining Detection and Location"(with N. H. Vaidya), *Proc. 1991 International Symposium on Fault-Tolerant Computing,* pp. 488–495, Montreal, Canada, June 1991.

24. "A Methodology for Partial Scan Design" (with S. Nori and J. Swaminathan), *Proc. Second European Test Conference,* pp. 263–271, Munich, Germany, April 1991.

25. "Weight/Space Bounded Error Control", *Proc. 1990 International Conference on Information Theory and Its Applications*, pp. 31–34, Honolulu, Hawaii, November 1990.

26. "Application Specific VLSI Architectures Based on De Bruijn Graphs", *Application Specific Array Processors,* IEEE Computer Society Publications, pp. 628–640, November 1990.

27. "Modeling of Live Lines and Tree Sharing in Multi-Code Memory Systems", *International Conference on Parallel Processing*, Vol. I, pp. 326–330, August 1990.

28. "Zero Aliasing Compression", *Proc. 1990 International Symposium on Fault-Tolerant Computing*, Newcastle, U.K., pp. 254–263, July 1990.

29. "On Implementing Improved Access Control Protocol for Shared Data Systems" (with A. Mendelson and A.D. Singh), *Proc. of 1st Annual IEEE Symposium on Parallel and Distributed Computing*, Dallas, TX, pp. 389–396, May 1989.

30. "Yield Modeling and Optimization of Large Redundant RAMs" (with A.D. Singh and K. Ganapathy), *International Conference on Wafer Scale Integration*, San Francisco, CA, pp. 273–287, January 1989.

31. "RTRAM: Reconfigurable and Testable Multi-bit RAM Design", *International Test Conference*, Washington, DC, pp. 263–278, September 1988.

32. "A New Framework for Designing and Analyzing BIST Techniques: Computation of Exact Aliasing Probability", *International Test Conference*, Washington, DC, pp. 329–340, September 1988.

33. "An Easily Testable Architecture for Multimegabit RAMs" (with N. Jarwala), *Proc. of International Test Conference*, pp. 750-758, Washington, September 1987.

34. "Consensus with Dual Failure Modes" (with F.J. Meyer), *Proc. FTCS-17*, Pittsburgh, pp. 48–54, July 1987.

35. "Cost Analysis of OnChip Error Control Coding for Fault-Tolerant Dynamic RAMs" (with N. Jarwala), *Proc. FTCS-17*, Pittsburgh, pp. 278–283, July 1987.

36. "Organization and Analysis of Gracefully-Degrading Interleaved Memory Systems" (with K. Cheung, G. Sohi, K. Saluja), *Proc. 14th International Symposium on Computer Architecture*, pp. 224-231, Pittsburgh, June 1987.

37. "Wafer-Scale Integration of Multiprocessor Systems" (with I. Koren and Z. Koren), *Proc. of HICSS-20 Hawaii International Conference on System Sciences*, pp. 13–20, January 1987.

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

39. "Dynamic Testing Strategy for Distributed Systems" (with F.J. Meyer), *Proc. FTCS-15*, pp. 84-90, Ann Arbor, Michigan, June 1985.

40. "A Versatile Sorting Network" (with M.R. Samatham), *Proc. 12th Annual Symposium on Computer Architecture*, pp. 360–367, June 1985.

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

42. "A Multiprocessor Network Suitable for Single Chip VLSI Implementation", *Proc. 1984 IEEE 11th Annual International Symposium on Computer Architecture*, pp. 328–337, June 1984.

43. "Fault-Tolerant Network Architectures for Multiprocessors and VLSI Based Systems", *Proc. FTCS-13*, Milan, Italy, pp. 436–441, June 1983.

44. "On a Class of Multiprocessor Network Architectures", *Proc. of International Conference on Distributed Processing*, Miami, Florida, pp. 302–311, October 1982 (also reprinted in *Interconnection Networks for Parallel and Distributed Processing,* edited by C. Wu and T. Feng, August 1984).

45. "Testing for Delay Faults in a PLA" (with K. Son), *Proc. International Conference on Circuits and Computers*, pp. 346–349, September 1982.

46. "Interconnections Topologies for Fault-Tolerant Parallel and Distributed Architectures", *Proc. of 10th International Conference on Parallel Processing*, pp. 238–242, August 1981.

47. "Fault-Diagnosis of Parallel Processor Interconnection Networks" (with K.M. Falavarajani), *Proc. Eleventh Annual International Symposium on Fault-Tolerant Computing*, pp. 209–212, June 1981.

48. "A Fault-Tolerant Communication Architecture for Distributed Systems", *Proc. Eleventh International Conference on Parallel Processing*, pp. 214–220, June 1981.

49. "A Solution to Load-Balancing and Fault Recovery in Distributed Systems" (with K. Matsui), *Symposium on Reliability in Distributed Software and Database Systems*, pp. 89–94, July 1981.

50. "Completely Self–Checking Checkers" (with K. Son), *Digest of 1981 Test Conference*, pp. 231–237, October 1981.

51. "A Fault–Diagnosis Technique for Closed Flow Networks", *Proc. of 1980 Symposium on Fault–Tolerant Computing*, pp. 302-304, Kyoto, Japan, October 1980.

52. "Effect of Undetectable Faults on Testing PLAs" (with K. Son), *Digest of 1980 Test Conference*, pp. 359–367, November 1980.

53. "An Easily Testable Design of PLAs" (with K. Son), *Cherry Hill Test Conference*, Philadelphia, Pennsylvania, November 1980 (reprinted in IEEE Tutorial on VLSI Testing, edited by Rex Rice, 1981).

54. "A Generalization of Shuffle-Exchange Networks", *Proc. of Fourteenth Annual Conference on Information Sciences and Systems*, Princeton, New Jersey, March 1980.

55. "A Framework for the Study of Permutations and Applications to Memory Processor Interconnection Networks" (with K.L. Kodandapani), *Proc. 1979 International Conference on Parallel Processing*, pp. 148–158, August 1979.

56. "Shift Registers Designed for On-Line Fault Detection", *Proc. of 1978 International Symposium on Fault Tolerant Computing*, Toulouse, France, pp. 173–178, June 1978.

57. "A Synthesis Algorithm of Directed Two-Commodity Networks", *1978 IEEE International Symposium on Circuits and Systems*, New York, pp. 93–98, May 1978.

58. "Error Control Techniques for Array Processors", *1977 International Symposium on Information and Theory*, Ithaca, New York, October 1977.

59. "On Undetectability of Bridging Faults" (with K.L. Kodandapani), *Proceedings of 1977 International Symposium on Fault-Tolerant Computing*, p. 192, Los Angeles, California, June 1977.

60. "Fault-Tolerant Fail-Safe Logic Networks" (with S.M. Reddy), *Proceedings on IEEE Compcon*, pp. 363–366, March 1977.

61. "Further Results on m-RMC Forms" (with K.L. Kodandapani), *Proceedings of 1976 International Symposium on Multivalued Logic*, Logan, Utah, pp. 88–93, May 1976.

62. "A Graph Structural Approach to Data Management Systems" (with L.C. Chang), *Proc. Ninth Hawaii International Conference on System Sciences*, Western Periodicals, pp. 254–258, January 1976.

63. "Fault-Tolerant Asynchronous Networks Using (2,1)-Type Assignments", *Digest of Fifth International Symposium on Fault-Tolerant Computing*, Paris, France, June 1975.

64. "Construction on Error Correcting Codes with Run-Length Limited Properties", presented in *1974 International Symposium on Information and Theory*, Notre Dame, Indiana, November 1974.

65. "Synthesis of Arithmetic and Logic Processors by using Nonbinary Codes" (with L.C. Chang), Digest of Papers, *Fourth International Symposium on Fault–Tolerant Computing*, IEEE Computer Society Publications, pp. 4–22, June 1974.

66. "A Multi-Valued Switching Algebra Based on Finite Field", *Proc. 1974 International Symposium on Multiple Valued Logic*, IEEE Computer Society Publications, Vol. 3, pp. 95–113, May 1974.

67. "On Fault Diagnosis of Sequential Machines", *Proc. VI Hawaii Conference on System Sciences*, Western Periodicals, January 1973.

68. "A Design Technique for Synthesis of Fault-Tolerant Adders" (with S.M. Reddy), *Digest of Papers of 1972 International Symposium on Fault-Tolerant Computing*, IEEE Computer Society Publications, pp. 20–25, June 1972.