University of Massachusetts - Amherst
ScholarWorks@UMass Amherst

Masters Theses 1896 - February 2014 | Dissertations and Theses

2009

A Hardware Framework for Yield and Reliability Enhancement in Chip Multiprocessors

Abhisek Pan
University of Massachusetts - Amherst, abhisekpan@gmail.com

Follow this and additional works at: http://scholarworks.umass.edu/theses

Pan, Abhisek, "A Hardware Framework for Yield and Reliability Enhancement in Chip Multiprocessors" (2009). *Masters Theses 1896 - February 2014*. Paper 327.
http://scholarworks.umass.edu/theses/327

This Open Access is brought to you for free and open access by the Dissertations and Theses at ScholarWorks@UMass Amherst. It has been accepted for inclusion in Masters Theses 1896 - February 2014 by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.

A HARDWARE FRAMEWORK FOR YIELD AND RELIABILITY ENHANCEMENT IN CHIP MULTIPROCESSORS

A Thesis Presented

by

ABHISEK PAN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

SEPTEMBER 2009

Electrical and Computer Engineering

© Copyright by Abhisek Pan 2009

All Rights Reserved

A HARDWARE FRAMEWORK FOR YIELD AND RELIABILITY ENHANCEMENT IN CHIP MULTIPROCESSORS

A Thesis Presented

by

ABHISEK PAN

Approved as to style and content by:

_____
Sandip Kundu, Chair

_____
C. Mani Krishna, Member

_____
Russell G. Tessier, Member

_____
C. V. Hollot, Department Head
Electrical & Computer Engineering

DEDICATION

To my parents.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Prof. Sandip Kundu for his support and guidance throughout this project. I also wish to thank the following: my committee members, Prof. C. Mani Krishna and Prof. Russell Tessier for their valuable criticism and suggestions; Prof. Csaba Andras Moritz for his guidance during the initial part of the project; Mr. Omer Khan for his insightful inputs; and the Office of Information Technologies for helping me with the formatting of the document. Finally I would like to thank the Semiconductor Research Corporation for sustaining me financially during the period when this work was done.

ABSTRACT

A HARDWARE FRAMEWORK FOR YIELD AND RELIABILITY ENHANCEMENT IN CHIP MULTIPROCESSORS

SEPTEMBER 2009

ABHISEK PAN, B.E.E., JADAVPUR UNIVERSITY

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Sandip Kundu

Device reliability and manufacturability have emerged as dominant concerns in end-of-road CMOS devices. Today an increasing number of hardware failures are attributed to device reliability problems that cause partial system failure or shutdown. Also maintaining an acceptable manufacturing yield is seen as challenge because of smaller feature sizes, process variation, and reduced headroom for burn-in tests. In this project we investigate a hardware-based scheme for improving yield and reliability of a homogeneous chip multiprocessor (CMP). The proposed solution involves a hardware framework that enables us to utilize the redundancies inherent in a multi-core system to keep the system operational in face of partial failures due to hard faults (faults due to manufacturing defects or permanent faults developed during system lifetime). A micro-architectural modification allows a faulty core in a multiprocessor system to use another core as a coprocessor to service any instruction that the former cannot execute correctly by itself. This service is accessed to improve yield and reliability, but at the cost of some loss of performance. In order to quantify this loss we have used a cycle-accurate architectural simulator to simulate the performance of dual-core and quad-core systems with one or more cores sustaining partial failure. Simulation studies indicate that when a

large and sparingly-used unit such as a floating point unit fails in a core, even for a floating point intensive benchmark, we can continue to run the faulty core with as little as 10% performance impact and minimal area overhead. Incorporating this recovery mechanism entails some modifications in the microprocessor micro-architecture. The modifications are also described here through a simplified model of a superscalar processor.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS v

ABSTRACT vi

LIST OF TABLES x

LIST OF FIGURES xi

CHAPTER

1. INTRODUCTION 1

2. BACKGROUND: YIELD AND RELIABILITY CHALLENGES 2

   Yield 2
   Reliability 4

3. RELATED WORK 6

   Software-based approach 9
   Hardware-based approach 9

4. PROPOSED APPROACH 11

   Inter-Core Queue 11
   Faulty Core 15
   Helper Core 16
   Overhead 18

5. ARCHITECTURAL PRE-REQUISITES 20

   Manufacturing-time Detection and Diagnosis 20
   Online Detection and Diagnosis 21
   De-configuration 23

6. SIMULATION FRAMEWORK 24

   Processor Configuration 24
   Workloads 25

7. RESULTS AND ANALYSIS 27

Dual Core Results 27
Quad-Core Results 32

8. LIMITATIONS 37

9. CONCLUSION 38

BIBLIOGRAPHY 39

LIST OF TABLES

Table Page

1. CMP Configuration …….. 24

2. Workloads for Dual-Core System ……..... 26

3. Workloads for Quad-Core System ……... 28

4. Faulty Units for 2 Simultaneous Faulty Cores ……. 28

5. Failure Configurations for Two Faulty Cores …….. 34

LIST OF FIGURES

Figure Page

1. Inter-Core Queue: A Logical View …….. 11

2. Inter-Core Queue for Dual-core System …….. 14

3. Modifications in the Faulty Core ……. 16

4. Modifications in the Helper Core …….....17

5. Data-Flow with Remote Execution …….. 18

6. Intra-cycle Logic Dependence ……..21

7. Relative Performance Variation with ICQ Depth (Idling Interval = 5 cycles) …….31

8. Relative Performance Variation with Idling Interval (Queue depth = 10) ……... 31

9. Relative Performance Variation with ICQ Depth: Both Cores Faulty ……. 31

10. Performance variation with IC queue depth: Core 1 Faulty ……... 35

11. Performance variation with IC queue depth: Core 3 Faulty…… 35

12. Performance variation with IC queue depth: Core 3 and Core 4 Faulty …… 35

13. Performance variation with IC queue depth: Core 1 and Core 3 Faulty …… 36

14. Performance variation with IC queue depth: Core 1,2,3 and Core 2,3,4 Faulty …... 36

CHAPTER 1

INTRODUCTION

Ensuring reliable operation during the entire service-period is essential for computing systems, especially for those deployed in safety-critical applications and in remote and otherwise hazardous locations. However today's deep submicron systems are becoming increasingly vulnerable to premature system failures due to unreliable hardware substrate and environmental stress. Also increased manifestation of random, systematic, and parametric yield loss mechanisms are making it difficult for the chip manufacturers to maintain an acceptable manufacturing yield rate. Hence designing adaptive systems, which can be fit for deployment and can continue to function reliably even with faulty components, can go a long way in sustaining the present growth-rate of device-count and clock-frequency in face of critical yield and reliability problems.

In this project, we propose and investigate a low-overhead hardware framework for a multi-core system that serves to improve the yield and reliability of the system. The framework enables the system to remain in operation, albeit in a degraded performance mode, even when one or more of the cores sustain partial failure, by exploiting the inherent redundancy already in the system. The proposed scheme involves hardware-assisted communication between the cores to share functional units across them. This sharing is exposed at an instruction-level granularity. The hardware and power overhead for the proposed scheme is minimal. Besides improving long-term reliability through graceful degradation, the scheme can also achieve reasonable yield enhancement at quite acceptable performance degradation, if we consider the fact that most systems contain sparsely used functional units with large on-chip area.

CHAPTER 2

BACKGROUND: YIELD AND RELIABILITY CHALLENGES

Relentless advancement in process technology during the last four decades has led to processor designs with progressively higher transistor count and increased clock frequency. Introduced in 1971, Intel's first microprocessor built with silicon-gate MOS technology, the Intel 4004, had about 2300 transistors and ran at 108 KHz. Today the 45nm Penryn family of processors can pack in 400 to 800 million transistors and reach clock speeds up to 3GHz [1]. It is widely believed that CMP architectures will allow a clear path to ITRS technology scaling projections of 100 billion transistors per chip by year 2020 [7]. However, sustaining this explosive device-count growth on a chip is going to be difficult due to critical yield and reliability problems [2].

Yield

The factors that contribute towards the loss of manufacturing yield in integrated circuits are broadly referred to as *yield loss mechanisms* (YLM). Traditionally yield loss mechanisms are classified as follows [2]:

Random YLM: This constitutes of random particulate and contaminant induced defects that may result in open and short circuits, and these faults are studied through statistical models.

Systematic YLM: These effects are primarily functions of specific layout patterns and manifest in faults that show strong spatial or temporal correlation, eg. Mask misalignments, optical proximity effects like line-end shortening and line-width differences, via /via stack failure as function of interconnect length and so on.

Parametric YLM: These are caused when otherwise functional chips fail to meet the acceptable performance specifications, and are the results of intra and inter-die variations in device electrical characteristics.

Random yield is usually modeled through Poisson distribution considering the defects to be independent, where the yield is given by:

$$Y \propto e^{-D_0 \times A \times KR} \quad (1)$$

where D_0 is defect density in defect/area, A is area of chip and KR (kill ratio) is the fraction of total area that can be affected by the defects [3]. Although random defects are being controlled through use of clean-room facilities, shrinking feature size means increase in kill ratio and packing multiple cores in a die means increase in area, thus potentially leading to a reduction in yield. Also shrinking feature sizes and magnified process variation effects are exposing the limitations of traditional resolution enhancement techniques (RET) and restrictive design rules (RDR) in reducing systematic and parametric yield loss [4]. The traditional accelerated life tests (burn-ins) used to filter out defective chips at their infancy are also losing their effectiveness due to reduced headroom for stress testing [5]. Consequently, users of semiconductor chips may experience higher infant mortality problems. Infant mortality problems are aggravated further due to new aging defect mechanisms such as NBTI [6]. Yield recovery in field has been proposed as a potential solution to these problems [7]. This has been described in ITRS as: "Relaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce costs of manufacturing, verification, and test. Such a paradigm shift is likely forced in any case by technology scaling, which leads to more transient and permanent failures of signals, logic values, devices, and interconnects." [7]

Reliability

Another area of concern in foreseeable future is system reliability. Current design practice for processors (except high-end mainframes and some safety-critical systems) is to assume that the underlying fabric of transistors and interconnects will always operate correctly during the product lifetime. However continuous push for smaller devices and interconnects has moved the technology closer to a point of unreliability where such design paradigm is not valid [9]-[11]. For example, at 90nm technology Negative Bias Temperature Instability (NBTI), where a PMOS device degrades continuously with voltage and temperature stress, had become a major reliability concern, [12]-[13]. At 45nm, the problem has exacerbated due to lower threshold voltage, and Positive Bias Temperature Instability (PBTI) for NMOS devices has been added to the list [12]. Impact of other device-failure mechanisms like time dependent dielectric breakdown (TDDB) are also increasing because of extremely high on-chip temperatures, high current densities and thinner gate oxides [14]-[16]. Copper electro-migration, stress voiding, and electrical breakdown of low-k ILD have compromised the reliability of interconnects [7]. Erratic bit errors in SRAM and SEU (single event upset) based soft errors are on the rise. These problems, cumulatively referred to as PVT issues (process corner, voltage and temperature), are expected to worsen in future nano-CMOS technology [17].

Along with device imperfections, designs aimed at maximizing performance and area efficiency will contribute towards loss of reliability. High packing density will allow the fabrication of complex heterogeneous monolithic processing engines, but the potential of undetected design errors will increase proportionally [9]. Aggressive adaptive

voltage and frequency management of modern processors are expected to compound such errors as well [8][18].

These reliability defects cannot be pre-screened during manufacture. The majority of these defects will appear under specific voltage, temperature, frequency and workload conditions. Hence, design paradigm of the future need to concentrate on building systems with imperfect transistors and interconnects; systems that will continue to function in spite of deteriorating components and multiple field failures [19]-[20]. A well-known approach towards building such systems involves the use of redundancy inherent in modern processors [21]-[33].

CHAPTER 3

RELATED WORK

Processor systems today have multiple units with same functionality in order to exploit instruction and thread-level parallelism, and actually require only a critical subset of their hardware in working condition to remain functional. In addition, higher device density and lower cost-per-transistor allow us to include power-gated redundant structures in processors which can be swapped in for units which fail in operation. Because of the redundancies, manufacturers can avoid throwing away chips unless they have critical functionality-disrupting defects, thereby improving yield. The spare structures are also used to extend the lifetime of a processor beyond that of the baseline architecture by keeping it functional, possibly in a degraded mode, even in presence of failed sub-units. The existing solutions consider redundancies at granularities from system to intra-processor levels. However sharing of hardware across multiple cores on a chip has remained a relatively less explored area.

The idea of incorporating redundancy in microprocessors for yield and defect tolerance is well entrenched. The 16-bit HYETI microprocessor is an early example of such a system, which had circuits for most of its functional units replicated 16 times in a bit-sliced design for optimal redundancy [33]. Redundancy for fault-tolerance can be incorporated in different levels of granularity. Commercial high-availability multi-processor systems like the IBM p690 have been designed to exploit redundancy at chip and module level [25]. These systems can map detected failures to individual CPUs and achieve system recovery by de-configuring the faulty processor during runtime or boot-up. The HP *NonStop*© *Advanced Architecture* uses a massively parallel cluster of dual or

triple modular redundant processors connected through system area networks to provide very high levels of fault-tolerance and availability for customer applications [26]. Aggarwal *et al* proposed a CMP architecture supporting isolation and de-configuration of groups of cores in order to provide fault-containment and availability [27]. More recently, researchers have proposed ideas to use redundancy at finer granularities in order to achieve more efficient use of redundant hardware [23]-[24].

Within a processor chip, we can identify three broad areas each of which need to be provided with different fault-tolerance methods:

- Large memory structures (Caches, TLB, Register files)
- Small memory structures (Reorder buffer, Issue queue, Load-Store buffer)
- The data-path and control logic

Large storage components and logic arrays are provided with error-correcting codes, redundant rows, columns and sub-arrays for effective yield and reliability improvement [32], [35]. The overhead of such mechanisms can sometimes be prohibitive for small-area arrays and queues. However Shivakumar *et al* [23] and Powell *et al* [28] have shown through experiments that there is minimal performance impact if the decode queues, reservation stations and reorder buffers lose several entries due to hard faults. The defective entries can be identified and de-configured by using a valid bit for each entry and appropriately modifying the decoder and counter logic [23] [35]. Hence smaller storage structures can be effectively protected through some adding some redundant entries and de-configuring the defective entries.

However structural irregularity and testability issues in logic and control units render them unsuitable for such partial and cost-effective redundancy [21]. Hence several

micro-architectural techniques have been proposed to handle this issue. In multi-core processors the simplest way is shut down a faulty core entirely. A more efficient alternative is to replicate entire functional units in order to achieve better yield or fault-tolerance. In this regard Shivakumar *et al* explored the possibility of using multiple execution units already present within a processor to improve manufacturing yield at the cost of performance degradation [23]. In [24] somewhat similar ideas of exploiting existing redundancies or building idle 'spare' units within a processor to improve lifetime reliability were introduced. However the use of in-core redundant execution unit is limited to units that have small area and power requirements. Powell *et al* considers the effectiveness of execution unit redundancy for a typical x86 core [28]. They classify instructions into three classes based on whether they could be executed on redundant resources:

- Class A: instructions that cannot be executed with redundant units (example -branch; integer and fp load-stores, fp add, mult, and divide; integer mult and divide)
- Class B and C: instructions that can be executed by more than one structures (example- int ALU; shift; int shuffle; simd shift and shuffle)

The non-redundant execution structures are found to occupy almost three fourth of the total execution area. Hence a core containing a faulty non-redundant functional unit would lose its ISA compliance and could not be salvaged through existing in-core redundancy or cold spares unless we are prepared to incur considerable power and hardware overhead. On the other hand, in multi-core systems, one or more of the other healthy cores already possess copies of the same functional units. Hence it is only natural

that we employ the units in other cores to execute the instructions that the damaged core is unable to serve. Such resource sharing across cores can be achieved through hardware or software.

Software-based approach

Reference [22] analyzes the benefits of sharing resources across partially damaged cores for yield enhancement. It proposes software-controlled thread swapping across cores to avoid faulty units. Along with the inherent performance degradation due to presence of faulty units, this scheme suffers from additional performance penalties due to repeated core hopping and context switching overheads (saving process state, cache/TLB misses, and branch mis

predictions).

Hardware-based approach

Reference [28] introduces the idea of thread migration or swapping through hardware. The scheme involves hardware-controlled migration of the process state between cores through an on-die SRAM. This scheme can be applied with minimal hardware modifications in the processors that already provide this capability of storing process states in an on-die SRAM for power savings [29]. This scheme also suffers from the same performance drawbacks as the previous software controlled thread-swapping scheme. The authors also investigate a hybrid approach comprising of micro-architectural intra-core redundancy and thread migration. Romanescu *et al* proposed the *core cannibalization architecture* (CCA) for multi-core processors where inter-core resource sharing is done at the granularity of pipeline stages [30]. The scheme adds considerable complexity to processor design and verification and the authors applied the scheme for

simple in-order cores only. Sharing pipeline stages form a neighboring core increases the cycle time for the defective core as well. The CASH (CMP And SMT Hybrid) architecture also advocates sharing of sparsely used functional units across several cores as a way to save area and reduce hardware complexity of individual SMT superscalar cores [34].

In this project, we consider hardware-controlled resource sharing across cores. A core containing a faulty functional unit is unable to serve instructions that require the use of that unit. In order to remain operational the faulty core uses the appropriate functional unit of its neighboring core. To this end we propose the use of a centralized Inter-Core Queue (ICQ) as an interface between cores in order to enable resource sharing among them [Figure 1]. The idea is developed in detail in the next section.

CHAPTER 4

PROPOSED APPROACH

A simplified version of the homogeneous dual-core CMP described in [31] is presented to illustrate the proposed scheme. The idea can be easily extended for a quad-core or many-core system since we are looking at the interaction of two cores (one faulty and one helper) at a time. Each core is assumed to be a superscalar out-of-order execution machine with private L1 data and instruction caches, and shared L2 cache. We assume a Symmetric Multiprocessor (SMP) paradigm for this proposed scheme. Basic modification involves incorporation of an Inter-Core queue (ICQ). The architecture of the ICQ and the structural and behavioral modifications required in the cores are described below.

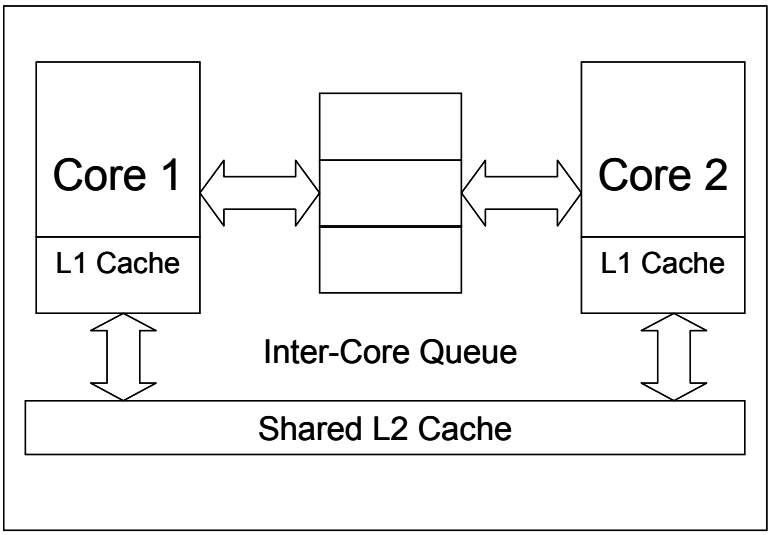

Figure 1: Inter-Core Queue: A Logical View

Inter-Core Queue

Inter-Core Queue forms the interface for data-flow between a faulty and helper core in this proposed scheme [Figure 1]. It acts as a temporary storage for instructions

that are to be transferred across cores. The ICQ maintains ordering of instructions using FIFO order. Each entry in the ICQ has data fields for the instruction including the opcode, source and destination operand values, as well as some control bits to manage the control flow. We define the following control bits for proper communication between the ICQ and the cores accessing the queue. The details of the faulty and helper core are described in the later sections.

- Valid: Identifies whether the entry is suitable for use or faulty
- Emergency: When set, identifies that the instruction has been in the queue long enough and needs to be served as early as possible
- Executed: When set, it indicates that instruction has completed execution at the helper core and the result is available in the ICQ
- Exception Info: These bits contain the exception information specific to the instruction. The helper core is responsible for exception detection, but the faulty core handles in during the retirement stage.

In addition, in order to unambiguously identify the source of each instruction, each entry needs to have bits identifying the source core, in which the instruction was originally issued. Also if multiple choices are available for the helper core, there has to be destination core identification.

The reset valid bit signifies that the slot is ready to be used. When a faulty core schedules an instruction to the tail of the ICQ, the source and destination core fields, and the execution bits are updated. We propose a push-pull scheme for scheduling the head of the ICQ to a helper core. When an instruction is ready in the ICQ to be serviced by the helper core, a bit in the decode unit of the helper core is set to convey the information.

Subsequently, at each cycle, after its native instructions are scheduled, the decode unit looks for an empty issue slot to schedule this instruction. If the decode unit finds a slot it pulls the instruction from the ICQ to its pipeline. On the other hand, if the helper core does not pull the ICQ, the emergency bit is set after a specified maximum wait period called idling interval. A hardware counter is used to count the number of cycles the instruction spends in the ICQ, and the emergency bit is set after the idling threshold is crossed. Once the emergency bit for this instruction is flagged in the ICQ, the instruction is given higher priority than the native instructions, and is pushed into the helper core pipeline before any additional native instructions are processed. Once the helper core is ready to retire this instruction, the ICQ is updated with the result and the Executed bit is set. Any exception detected during the execution of this instruction by the helper core is also updated in the ICQ. In our study we insert a maximum of one instruction per cycle.

Two important design parameters involving the ICQ are the depth per core and the idling interval. The depth per core refers to the number of instructions from each core that simultaneously resides in the queue. Increasing the depth is expected to improve performance of the faulty core for workloads that have clusters of instructions requiring the use of the faulty resource. However the area overhead will also increase. The idling interval, on the other hand, is expected to have a bearing on the performance of the helper core. Higher values for the interval would mean less frequent force-through from the ICQ, possibly leading to lower performance degradation for the helper core. The advantage is expected to be more pronounced if the helper core also faces a strong demand for the shared functional units from its native instructions.

The ICQ forms a critical component of the scheme. Hence it has to be protected by redundancies. Fortunately the regular structure of buffers means they can be provided effective protection with low hardware overhead [35].

Implementation-wise, the ICQ can be implemented through a SRAM array with pointers for head and tail of the queue. For a dual-core system, in order to allow each of the cores to read from and write into the queue every cycle, we need to have two physical queues - queue 1 for instruction migration from core 1 to core 2 and queue 2 for migration in the reverse direction [Figure 2]. For a quad-core system we need four physical buffers, one for each core to send instructions to. Also a selector logic block is required which can map instructions in the faulty core to available helper cores and dispatch them to the helper cores in every cycle.

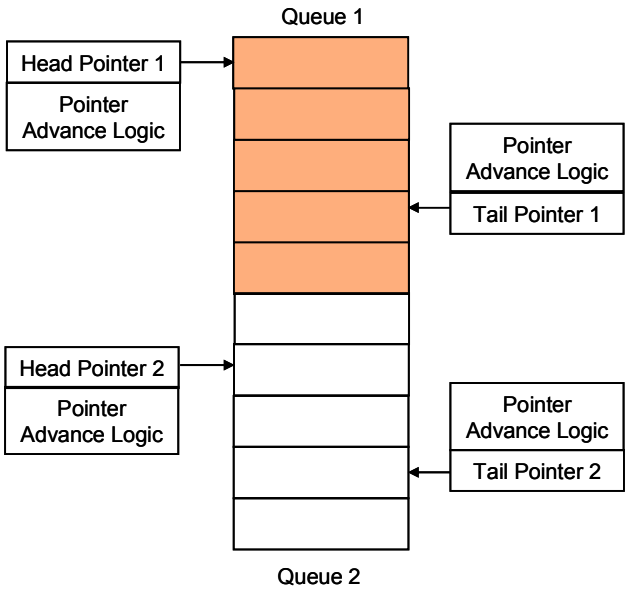

Figure 2: Inter-Core Queue for Dual-core System

Faulty Core

The path of an instruction requiring a faulty unit is illustrated in Figure 3. We consider a simplified Tomasulo data-flow model for a speculative out-of-order pipeline. Functional unit 1 is considered to be faulty. During the decode stage of the pipeline, a parallel lookup of the hardware fault table identifies whether an instruction requires the use of a faulty unit or not. When an instruction requires the use of a faulty unit, a flag, called the migration bit, is set in the in the control store entry for that instruction to ensure proper control flow in the subsequent pipeline stages. Then the faulty instruction is allowed to flow through the pipeline. The schedule unit dispatches the instruction to the reservation station (RS) from the fetch queue when it finds an empty reservation station and an empty slot in the Reorder Buffer (ROB). The capability of the faulty functional unit to write back to the Common Data Bus (CDB) is disabled in order to prevent data-corruption on the CDB. When the instruction reaches the head of the ROB, all dependencies are resolved and operands are available. Usually, the instruction is now executed and ready to be retired. However, if the migration bit is set, the instruction is scheduled to the IC queue if an empty slot is available. Otherwise, the instruction waits in the ROB for an IC queue slot. After the instruction is sent to the IC queue, the IC queue is polled for results and exception information. When the instruction is marked executed in the IC queue it is de-allocated from the IC queue and updated in the ROB. Now the result would be broadcasted into the CDB from the ROB head, following the normal mode of operation. Any instruction waiting for the result of this instruction would not get the value from the functional unit output but from the ROB head. Also the commit unit handles the instruction commit to the architectural state. Exceptions are handled during

this stage, depending upon the exception information received from the IC queue. The use of the migration bit in the control-store information and the exception bit in the IC queue preserves the speculative and precise interrupt behavior of the processor. An alternative mode of execution would be to send the instruction to the IC queue as soon as

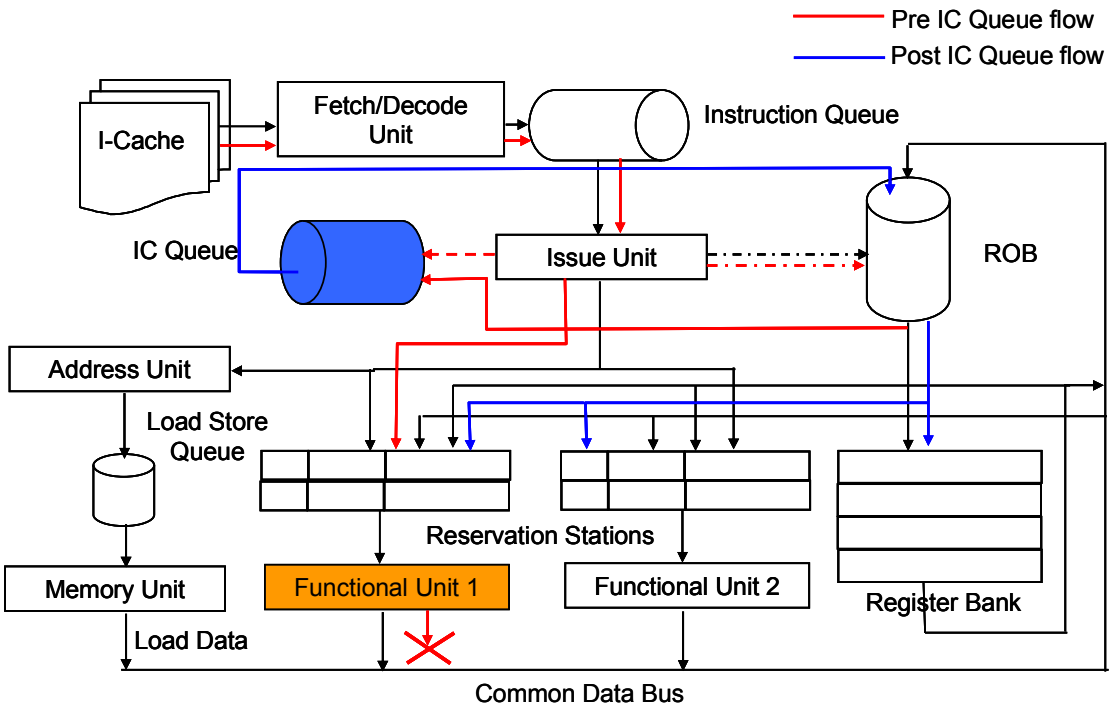

Figure 3: Modifications in the Faulty Core

all the source operands are available. However such a scheme would cause speculative instructions to be sent to the IC queue and executed in the helper core. Sending instructions from the head of ROB prevents the use of the helper core for speculative and potentially futile instructions.

Helper Core

An instruction is either pushed or pulled by the helper core as described before. In either case a control store entry is created for proper control flow for this instruction in the subsequent stages of the pipeline. After empty reservation stations and ROB entry is

found, the instruction is dispatched by the issue unit, and the operands are pulled from the ICQ. We note that the operands can also be pulled by the execution units, wherever the critical path is mitigated. Once the instruction completes execution, the results and any exception detected during the instruction execution in the helper core are written back to the ICQ. The executed-bit in the ICQ for this instruction is also set. The reservation station and ROB entries are freed. The flow is illustrated in Figure 3. An important consideration here is that the result once computed by the functional unit will be broadcasted to the CDB. So any other instruction waiting in any reservation station

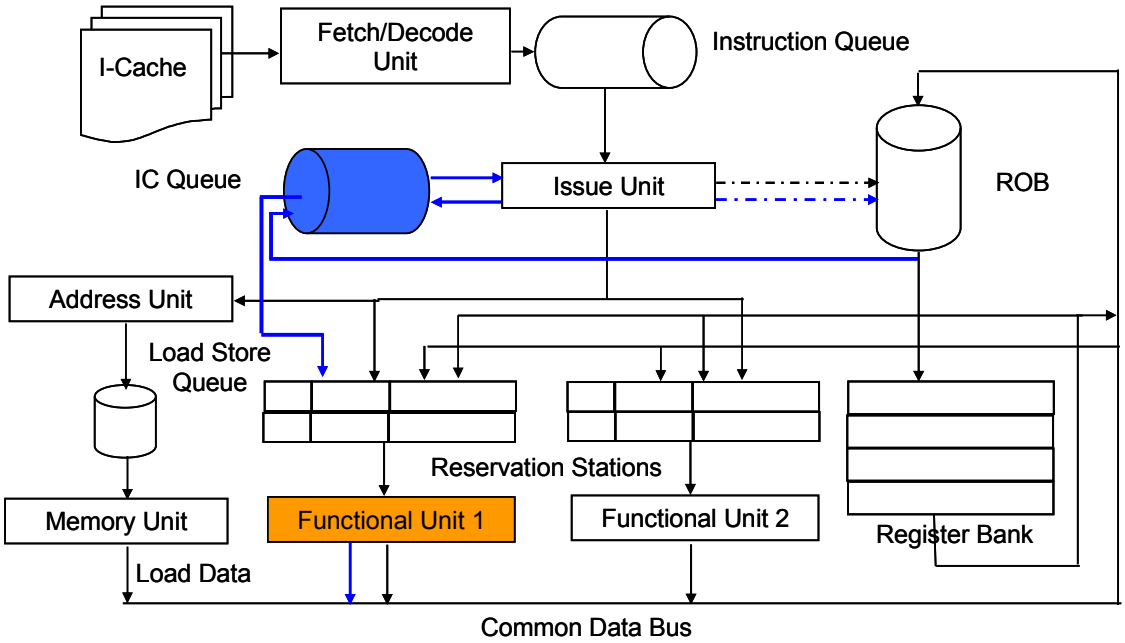

Figure 4: Modifications in the Helper Core

should not interpret this result as a native result. Assuming that results are tagged with ROB entry number, we need to add an additional bit to the tag in order to identify the result as native or foreign. This composite tag would avoid any data corruption in the helper core.

The data flow of the instructions through the faulty core and the helper core is shown in Figure 5. A single ICQ can be used for transferring instructions from each of the two cores to the other one as required. The change in the data-flow is constrained within the pipeline, and introduces no data consistency problems in the architectural state of the system.

Overhead

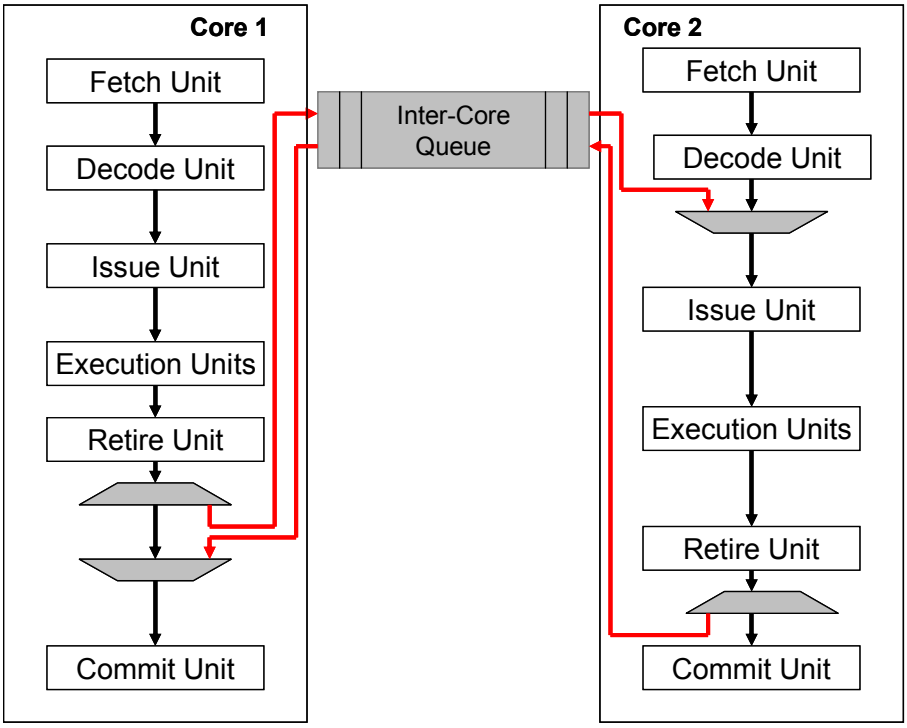

Figure 5: Data-flow with Remote Execution

This scheme certainly entails some overhead in terms of area and complexity. The additional hardware for incorporating this scheme involves the following:

- ICQ and the buses connecting the ICQ to each core,
- Hardware Counters for calculating the idling interval
- A FSM controller to control data flow through the IC cores

- Extra complexity in the control and synchronization logic of the cores to control the migration of instructions,
- A couple of bits in the control-store entry for an instruction, and
- Hardware fault-map and associated wires to read and write the map.

Compared to the area of a dual-core multi-processor, the area overhead is quite low. The ICQ needs to be placed symmetrically between two cores to ensure equal in-flight time for instructions between the queue and the cores. This places an additional constraint on the layout of the chip. The controller design complexity is also increased, which will require some extra design and test effort.

CHAPTER 5

ARCHITECTURAL PRE-REQUISITES

Any fault-tolerance scheme comprises of two parts: fault detection and isolation, and recovery. This project deals with fault recovery, and is independent of the underlying detection or isolation technique. However in order to make use of the proposed micro-architectural modification for yield and reliability enhancement, the processors must be able to execute the following functions correctly:

- Detection of hard faults,
- Diagnosis of the faulty unit, and
- De-configuration of the faulty unit.

Incorporating such fault awareness requires non-trivial modifications to the system hardware. An existing scheme that can be used for this purpose is the hard-fault detection and diagnosis framework described in [36], involving a low-cost hardware checker [38] and saturating counters. This section provides a brief outline of the methodology in [36] for the sake of completeness. The work here has no contributions towards this end.

Manufacturing-time Detection and Diagnosis

In order to use the scheme for yield enhancement, faults need to be detected and diagnosed to individual functional units. Well-known testing and design for testability (DFT) techniques are employed in detect the presence of faults in manufactured processors [39]. However these techniques usually isolate the faults to core-level granularity. Schuchman and Vijaykumar outline a detection and isolation methodology using common testing techniques which can be used to isolate faults to micro-

architectural blocks [40]. The authors define intra-cycle logic independence (ICI) as the condition necessary to enable conventional ATPG based scan-testing methods to isolate faults to micro-architectural blocks. ICI condition means that if a piece of combinational circuit bounded by latches can be decomposed into blocks such that there is no communication between the blocks within one cycle, any scan-detectable fault can be

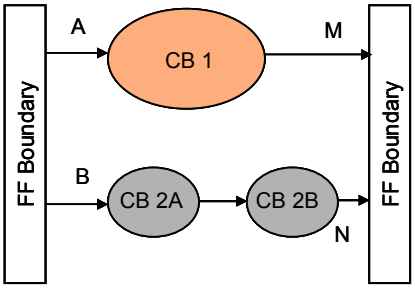

Figure 6: Intra-cycle Logic Independence

unambiguously mapped into any one of those blocks. Considering figure 6, we see that nodes A, B, M, and N are all observable in scan-based testing. Since there is no communication between combinational block 1(CB 1) and the combinational block 2 (made up of CB 2A and CB 2B), ICI is satisfied. Accordingly any fault detected at M can be uniquely mapped to CB 1 and any fault detected at N can be mapped to CB 2. However since blocks 2A and 2B communicate within one cycle, ICI is violated and we cannot unambiguously determine whether the fault observed at N developed in CB 2A or in CB 2B. The authors also propose DFT techniques to make combinational circuits ICI compliant.

Online Detection and Diagnosis

Online de-configuration of faulty units for reliability enhancement requires detection and diagnosis of faults. Such detections of hard faults can be performed through chip-level redundant multi-threading [37], online detection frameworks such as Dynamic

Implementation Verification Architecture (DIVA) [38], or through periodic health check involving built-in self-test (BIST).

In chip-level redundant multi-threading, identical threads are executed on separate processor cores, with the trailing thread receiving load values and line prediction outcomes from the leading thread. A store comparator is used alongside the store queue to compare store outcomes from the leading and trailing threads before writing them to the data cache.

The DIVA dynamic verification technique uses low-cost checkers at the commit stage of the pipeline to re-execute and verify the instructions coming out of the main superscalar pipeline in program order. The checkers are simplified in-order approximate computation units, with low overheads and higher fault-resilience. A mismatch in the results indicates hardware failure in the main pipeline, but no information regarding the nature and diagnosis of the fault is available.

Once a hard fault is detected, the location of the fault needs to be identified. In order to identify and diagnose faults, sub-structures in the cores that we wish to isolate and de-configure are classified as field de-configurable units (FDUs). Additional bits in the instructions are used to track FDU usage by an instruction from decode to commit stage. If an instruction result is found to be erroneous, the faulty FDU in use is recorded by incrementing a saturating counter corresponding to each and every FDU used by the instruction. If the fault-count for an FDU rises beyond a threshold within a pre-specified time interval, the fault in that unit is considered to be permanent [36]. Experimental results indicate that most hard faults can be suitably detected and diagnosed within a few thousand instructions after the faults develop.

Another alternative method proposed by Shyam *et al* involves the use of online distributed BIST checks performed periodically during idle intervals for processor components [5]. Each component is tested with high quality test vectors stored in an on-chip ROM, and the results are checked through simplified on-chip checkers.

We note that although low-cost checkers are devised for most of the components, effective online detection mechanisms for floating point units are still very difficult to design. References [41] and [42] provide instances of low-overhead reliable floating point units.

De-configuration

There are several ways to de-configure a faulty unit [36]; the one suitable for our scheme involves maintaining a hardware fault-table of the FDUs. There has to be one entry for each FDU, containing its operational health information. For many-core systems, the table can be extended to a fault-map, mapping the helper cores to be accessed for each FDU. This table will be updated online depending upon the entries in the saturating counters. For yield enhancement purposes, the fault-table can be initialized offline during pre-shipment testing to de-configure any faulty FDU.

CHAPTER 6

SIMULATION FRAMEWORK

For simulation studies, we used the SESC architectural microprocessor simulator. It is an event-driven cycle-level simulator built on MINT, a MIPS processor emulator [43]. The simulator was suitably modified to model dual-core and quad-core chip multiprocessors running multi-programming workloads. The SESC framework supports chip multiprocessing. This made it possible for me to implement process scheduling for multi-programmed workloads on the CMP models.

Processor Configuration

Table 1: CMP Configuration

Individual Cores	
Fetch Width	4
Issue Width	4
Retire Width	4
FP FU Latency	ALU:1, Mult:6, Div:12
Integer FU Latency	ALU:1, Mult:4, Div:12
# Ld St Units	2
# FP Units	1 each (ALU, Mult, Div)
# Integer Units	2 each (ALU, Mult, Div)
ICQ Access Latency	2 cycles
Memory Configuration	
L1 I Cache (private)	64 Kb, 4-way, WB
L1 D Cache (private)	64 Kb, 4-way, WB
L2 (shared)	8 Mb, 8-way, WB
Technology Parameters	
Technology	90 nm
V_{dd}	1.2 volts
Frequency	3 GHz

In this project we model 90-nm 32-bit symmetric dual-core and quad-core processors. Each core is a four-way speculative out-of- order superscalar running at 3GHz frequency. Relevant system parameters for each core are summarized in Table 1.

Dual-Core Modeling

For experiments on a dual-core system, we model one or both cores as being damaged permanently. Since we are concentrating on high-area, high-latency and low-utilization units in this study, we model one of the floating-point ALU, multiplier, and divider units as the faulty unit in each damaged core. Identical units in both cores are not treated as faulty simultaneously. The recovery scheme fails for such a pathological case.

Quad-Core Modeling

In quad-core simulation, we model one, two, or three cores as being damaged simultaneously. Target damaged units are floating-point ALU and/or divider units. For simulation we model a centralized queue.

Workloads

Dual-Core Workload Mix

For any simulation run, we combine two benchmarks to form a multi-programmed workload, and then spawn the threads separately on two cores. The benchmarks used are classified according to the proportion of floating point instructions contained in them. SPEC2000 benchmarks equake and gcc are picked with low floating-point instruction count, and flops and fbench with high floating-point instruction count. We combine these to form an appropriate mix that is interesting for the analysis, as shown in Table 2. These combinations form a representative set of the workloads that the cores can face with respect to floating point intensity. For each workload, we set each of the FP ALU, multiplier and divider units as faulty and measure the performance loss in the degraded system compared to a fault-free system.

Quad-Core Workload Mix

Here we combine eight different benchmarks programs to form three four-threaded multi-programmed workloads, and then spawn the threads separately on four cores. SPEC2000 benchmarks equake, gcc, mcf, and ammp are picked with low floating-point instruction count, art is picked with moderate floating-point intensity, and flops and fbench with high floating point instruction count. Combinations of these benchmarks form a representative set of workloads for this study (Table 3). For each workload, we set each of the FP ALU and divider units as faulty and measure the performance loss in the degraded system compared to a fault-free system. The helper cores are chosen in a round-robin fashion.

We vary the two design parameters, the ICQ depth and the maximum idling interval and analyze their impact on the performance loss. We also record the percentage of instructions that go through the ICQ and exceed the maximum idling interval. We run one billion instructions across the cores after fast-forwarding the initial two billion instructions in each core. The performance of each core is measured based on number of instructions issued per cycle (issued IPC). Hence an instruction issued in a faulty core and served by a helper core will be counted in the IPC of faulty core. The IPC for the helper core reflects its performance in executing its native thread only.

Table 2: Workloads for Dual-Core System

Workload		**FP instruction intensity**	
Faulty Core	Helper Core	Faulty Core	Helper Core
equake	gcc	Low (0.3%)	Low (0.0%)
flops	gcc	High (27.5%)	Low (0.0%)
flops	fbench	High (27.5%)	High (18.5%)

CHAPTER 7

RESULTS AND ANALYSIS

When a module in a core becomes faulty, a functional neighboring core helps with the execution. This may lead to performance degradation for both the cores involved in such interaction. We report this performance degradation in the faulty and helper cores with respect to the fault-free IPC of the individual cores. In the figures that follow, simulation results for selected workloads are shown, illustrating the performance degradation.

Dual Core Results

Figure 7 and 8 are used to illustrate the performance degradation when any one of the cores is faulty. The Y-axis shows the relative performance of each core compared to fault-free situation when no neighborly help is sought. The performance varies with depth of ICQ, type of the faulty unit considered, and nature of the workload. For example, if floating-point unit is defective, it is more likely to impact performance of a floating-point intensive program. The X-axis in Figure 7 represents the faulty unit type and the depth of ICQ. The depth of the ICQ was varied from 2 to 20 entries per core, keeping the idling interval constant at 5 cycles. The X-axis in Figure 8 represents the idling interval after which an instruction forces its way through. The idling interval was varied from 2 to 10 cycles for constant ICQ depth of 10. In both cases, the results were more-or-less consistent for the static parameter (idling interval or ICQ depth), so we only show results for a single constant variable. The performance of an infinite depth ICQ was also studied. However, it was found that the performance improvement obtained from increasing the depth tends to saturate at a value around 20. Hence we report result up to depth 20 only.

Figure 9 on the other hand, shows results when both cores have different faulty units, so that both the cores have to utilize the other core simultaneously and the flow of instructions through the ICQ occurs both ways. Here the performance improvement with ICQ depth saturated at a depth of 40 instructions in the worst case. The X-axis represents the depth of ICQ for the various combinations of faulty units in both cores, and the Y-axis denotes the relative performance of the cores. Table 4 contains the various combinations of faulty units used in simulation.

Table 3: Workloads for Quad-Core System

Workload	FP instruction intensity			
eaff	equake (3.5)	art (5)	flops (27.5)	fbench (18.5)
mgff	mcf (0.0)	gcc (0.3)	flops (27.5)	fbench (18.5)
mgaa	mcf (0.0)	gcc (0.3)	art (5)	ammp (0.0)

Table 4: Faulty Units for 2 Simultaneous Faulty Cores

Core 1	Core 2
FP-ALU	FP-Multiplier
FP-Multiplier	FP-ALU
FP-ALU	FP-Divider
FP-Divider	FP-ALU

Workload equake-gcc

For this workload, for a faulty FP-ALU, less than 1% of the fetched instructions are switched from the faulty core to the helper core, while the helper core has no floating-point instructions of its own. The idling interval has consistently shown to have no impact on performance. This is of particular interest when the helper core is running a critical thread region and would incur a wait period to service remote instructions. As expected for this workload, system performance is similar in presence of a single faulty core or two simultaneous faulty cores.

Workload flops-gcc

Here the ICQ Depth is found to be quite dominant in terms of performance impact. For faulty FP-ALU unit, the faulty core used a helper for about 14% of the issued instructions. Varying the ICQ depth from 2 to 20, the faulty core performance loss improved from 75 to 12%. Similar results are seen for a faulty FP-Multiplier unit that has approximately 12% of the issued instructions sent to helper core for execution (67 to 11%). In case when the FP-Divider is not working, about 2% of the instructions are sent to the helper core. The worst and best case degradations are 30% and 10% respectively. There is no impact of idling interval on the faulty-core performance. The monotonic improvement in performance of the faulty core with increase in ICQ depth can be seen in Figure 3.

In this workload, the helper core had no native floating-point instructions. Hence there was no contention for the floating-point execution units. The base-case IPC for the 4-way helper core is only around 0.95, which means the schedule and issue units are also utilized only partially, primarily due to the lack of instruction-level parallelism in its native thread. Hence these units have enough free resources available to serve any foreign instruction that is injected. Almost all switched instructions were served within the idling interval and very few had to be forced through the helper core. Instead the helper core actually observes 2-5% performance improvement. This apparent oddity is due to the nature of the simulator. The simulator actually stops execution when the sum of fetched instructions in both cores equals the specified number. Hence while the faulty core incurred more dead cycles due to extra latency of executing faulty instructions and

executed lesser instructions, the helper core fetched and executed more instructions, thus changing its native workload profile slightly.

When both the cores have faulty units (Figure 9), core 1 running the floating point intensive benchmark flops sees marked performance improvement with increase of the ICQ depth, and the improvement saturates at a depth of 30. The recovery is better for a faulty fp-divider than for an fp-ALU because of lower demand on the divider unit. Core 2, which executes the low-intensity benchmark gcc recovers the performance loss almost entirely at a depth of around 10.

Workload flops-fbench

This mix of floating-point intensive applications represents the worst-case combination that the system can face since both faulty and helper cores have significant floating-point load. Although the percentage of instructions switched remains same as the previous case, the best-case degradation achieved goes down to from 12 to 16% for faulty FP-ALU unit and from 11 to 15% for FP-Multiplier unit. Results for the FP-Divider were similar to the previous case (see Figure 7). When both the cores have faulty units, there can be a permanent performance degradation of around 10% in both the cores in the worst case, and the improvement saturates at a higher ICQ depth of 40. Since there was no significant performance improvement with variations in the maximum idling interval, the results for varying the idling interval for two simultaneous faulty cores were not shown here.

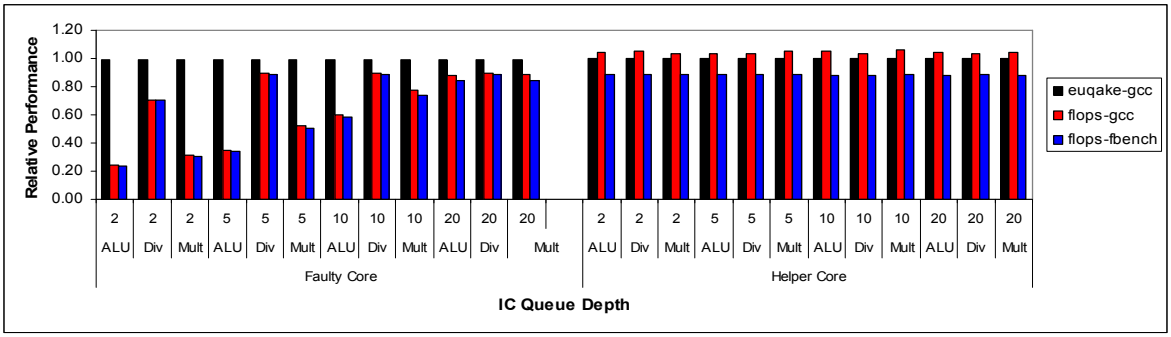

Figure 7: Relative Performance Variation with ICQ Depth (Idling Interval = 5 cycles)

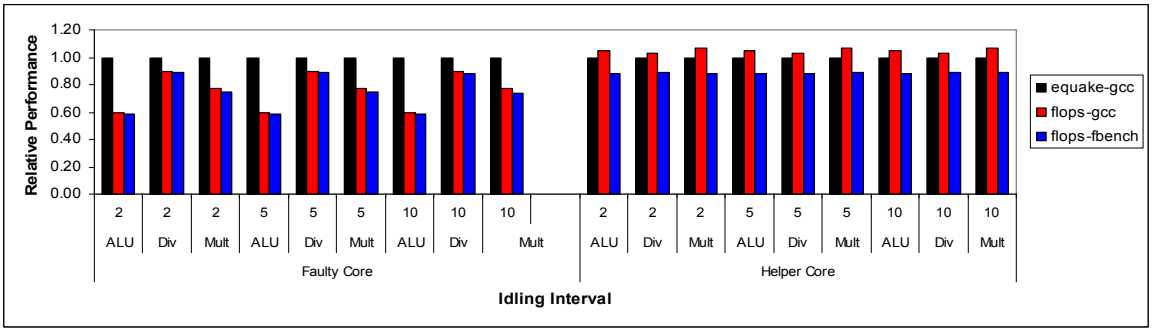

Figure 8: Relative Performance Variation with Idling Interval (Queue depth = 10)

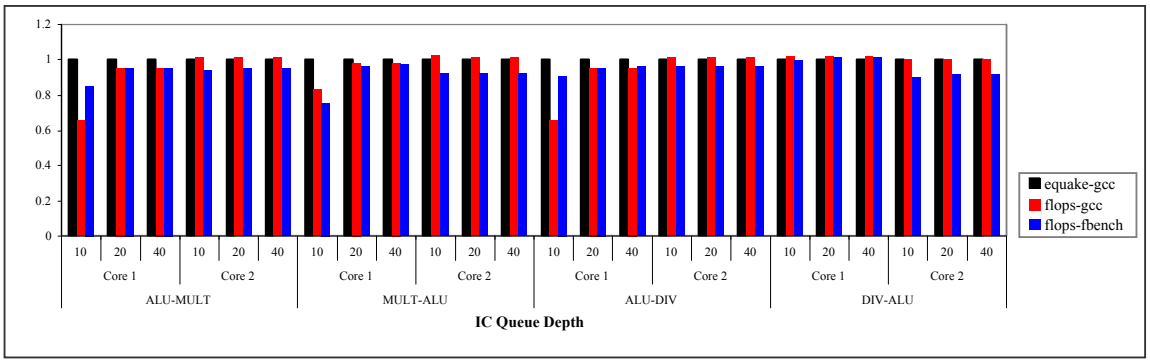

Figure 9: Relative Performance Variation with ICQ Depth: Both Cores Faulty (Idling Interval = 5 cycles)

Quad-Core Results

Single Faulty Core

For this configuration, we show the results for faults in cores 1 and 3 in figures 10 and 11 respectively. Across the workloads, core 1 faces the least floating-point intensity and core 3 sees the most.

In Figure 10, we see that for mgaa and mgff workloads, there is no performance degradation in the system at all, since the IC queue is not pressed into service. For eaff, there is moderate floating-point activity in core 1, hence for a faulty FP-ALU, there is a loss of 3% in the faulty core for a queue depth of 4. However a depth of 8 is sufficient to recover the loss. About 2% of the fetched instructions are switched from core 1 to the other cores, while the helper cores show no performance loss in executing their native threads. Very few instructions survive in the IC queue up to the maximum idling interval, indicating that the helper cores had enough space to accommodate the foreign instructions without sacrificing their native IPC. For a faulty FP-Divider unit, less than 1% of instructions are switched and there is no appreciable loss in the system even for a queue depth of 4.

In figure 11, for mgaa and a faulty FP-ALU unit, there is about 5% drop in performance in core 3 for a depth of 4, which improves to 1% for a depth of eight. All other cores are unaffected. For mgff and eaff, the faulty core suffers about 60% degradation at depth of 4, but recovers sufficiently with increase in the depth reaching almost full performance at a depth of 32. Also, a small drop of 1-2% can be seen in the helper cores that have native floating-point instructions to run, representing mild contention for resources. Varying the idling interval has no effect on reducing this drop.

Hence for a workload of low FP intensity, which is the case for a significant proportion of programs running in commercial and personal space, the scheme enables us to salvage a faulty chip without much performance degradation. The idling interval is again consistently shown to have no impact on performance.

Two faulty cores

Here we investigate three different configurations as shown in Table 5. Case 1 represents the worst case possible, when both the fp-intensive cores are faulty (Figure 12). For mgaa, the performance is similar to the case with only the 3rd core faulty, because the 4th core does not have any appreciable fp-intensity anyway. For the rest, IC queue depth is found to be quite dominant in terms of performance impact on the faulty cores. For faulty FP-ALU unit, core 3 used helper cores for about 18% of the instructions, and core 4 did the same for about 7%. In mgff, by varying the IC queue depth from 8 to 48, the performance loss for core 3 improved from 40 to less than 1%. Performance loss of core4 improved from 90 to 4%. For eaff, the corresponding numbers were 50 to 1% and 60 to 3% respectively. Only mild to negligible drop was observed in the helper cores' performance (1 and 2) because of lack of native floating-point intensity. For cases 2 (Figure 13) and 3, the results show a similar trend. More the floating- point intensity in a faulty-core, larger is the impact of increasing the queue depth. The performance loss generally saturates within 5% for deep-enough queues. The helper cores do not show any significant degradation in these configurations.

Three Faulty Cores

We report the results of two configurations in figures 14 - faulty FP-ALU units in cores 1, 2, 3 and cores 2, 3, 4 respectively. In the first configuration, for mgff, core 3 suffers the most significant degradation, but the loss saturates to 1% at a depth of 32.

Table 5: Failure Configurations for Two Faulty Cores

Configuration	Cores				Nature of the Mix of Faulty Cores
	# 1	# 2	# 3	# 4	
1			X	X	2 high fp-intensive cores
2	X		X		1 high and 1 low fp-intensity
3	X	X			2 low fp-intensity cores

There is a 3-5% loss in the only helper core (core 4) due to the bottleneck – the only floating-point unit in the system– and cannot be avoided by varying any design parameter. For eaff, the trend is similar with core 3 saturating at 5% loss at depth 32, and core 4 suffering a steady 3% loss. In Figure 13, for mgff workload, core 4 saturates at a loss of 5% and core 3 recovers completely. There is no appreciable loss in the helper core (core 1) since it has no native floating-point instructions. For eaff, faulty cores 2 and 4 suffer a loss of 2-3% whereas for faulty core 3 the loss is negligible. The helper core (core 1) does not see any degradation. Hence we see that the loss in the faulty cores is generally bounded within 5% for a queue size not exceeding 50. The helper cores also do not show degradation exceeding 10%. The Variations in idling interval have minimal impact, as the cores are wide enough to accept instruction from a faulty core without much interference to their native instructions.

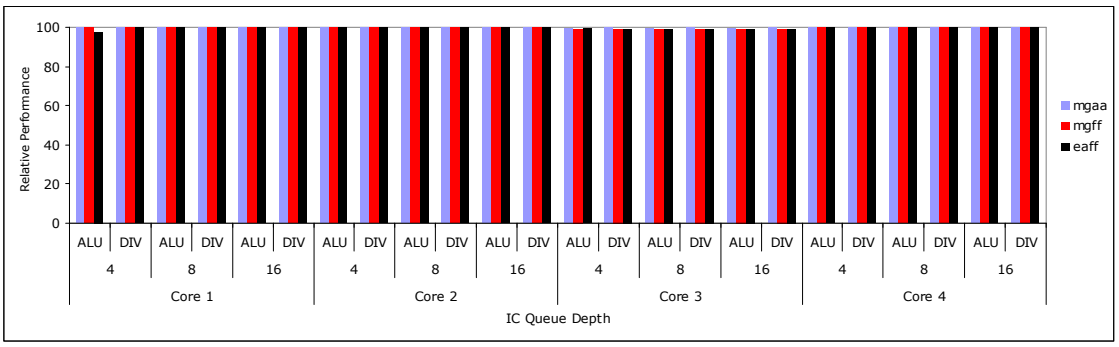

Figure 10: Performance variation with IC queue depth: Core 1 Faulty

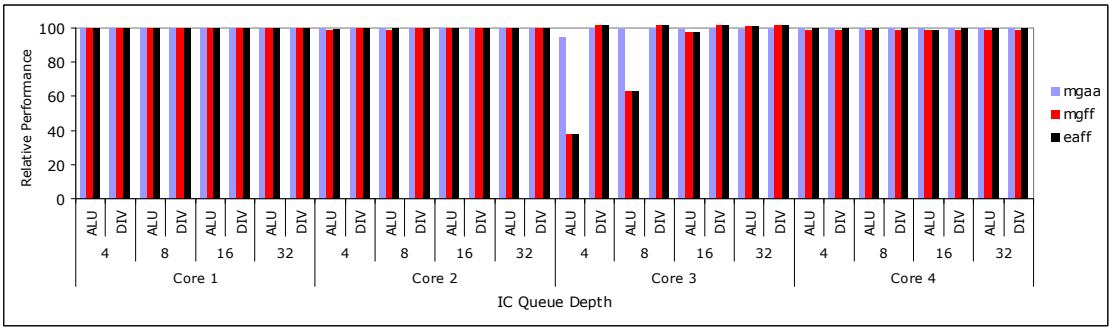

Figure 11: Performance variation with IC queue depth: Core 3 Faulty

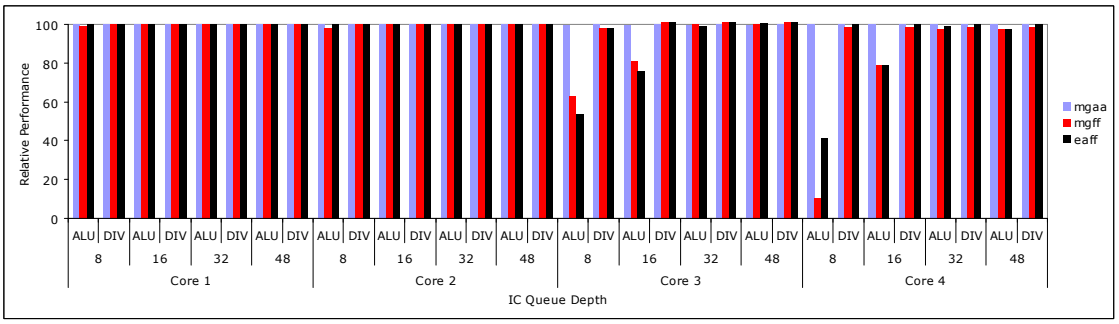

Figure 12: Performance variation with IC queue depth: Core 3 and Core 4 Faulty

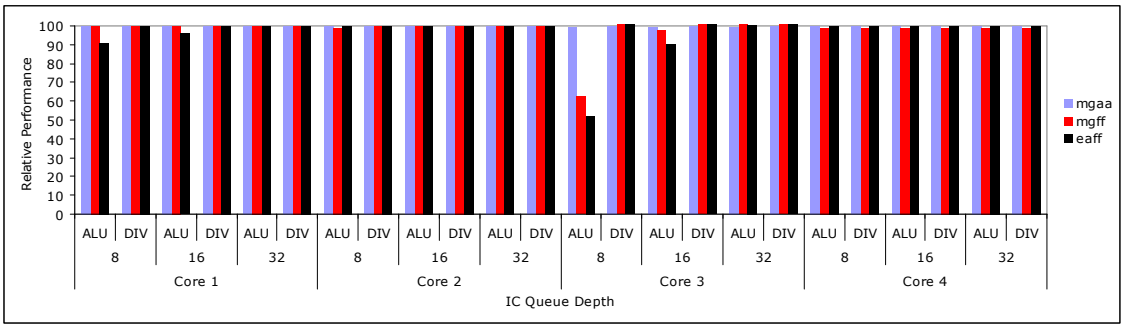

Figure 13: Performance variation with IC queue depth: Core 1 and Core 3 Faulty

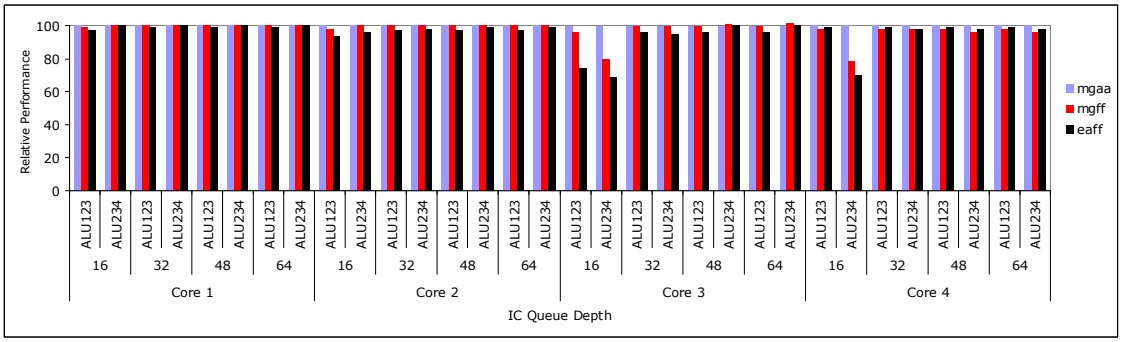

Figure 14: Performance variation with IC queue depth: Core 1,2,3 and Core 2,3,4 Faulty

CHAPTER 8

LIMITATIONS

The scheme investigated here is particularly suitable for large sparingly-used units inside a core. In order to cover the entire processor area, the structures used to execute frequently occurring instructions like integer ALU operations, load-store, and branch instructions should be protected through redundancy. In hardware-based approach, this can be done through in-core spare functional units and redundant entries in the storage buffers. The software controlled thread swapping provides a viable alternative to utilize the inter-core redundancy. However the overheads for thread swapping would be amortized when the swapping occurs only infrequently, which means the program profile would be such that the offending instructions occur infrequently and in clusters. However when the instructions requiring the use of faulty units occur more regularly, the hardware controlled instruction migration would be more efficient.

CHAPTER 9

CONCLUSION

Multi-core processors have inherent redundancy in them. In this work, a micro-architectural technique was proposed to exploit such redundancy for salvaging yield and improving reliability. The central idea was to implement an inter-core queue to seek execution help from functioning neighboring cores. The resulting design changes are minimal and impose insignificant cost in terms of area and power. Simulation shows that significant yield recovery is possible with only 10-15% performance degradation in the worst case. The proposed scheme is useful for high-area high-latency instructions that are executed sparingly. The proposed scheme by itself is not sufficient to provide fault-tolerance to the entire processor area. However along with memory protection techniques and in-core redundant units this scheme can be effective in improving yield and reliability for chip multi-processors. For many-core processors, we can provide effective coverage through a modular framework of clusters of four cores connected through the IC queue.

BIBLIOGRAPHY

[1] Introducing the 45nm Next-Generation Intel® Core™ Microarchitecture. *Intel White Paper, 2007.*

[2] Guardiani, C., Bertoletti, M., Dragone, N., Malcotti, M., and McNamara, P. 2005. An effective DFM strategy requires accurate process and IP pre-characterization. In *Proceedings of the 42nd Annual Conference on Design Automation, 2005.*

[3] Maly, W. and Deszczka, J. Yield estimation model for VLSI artwork evaluation. In *Electronic Letters,* vol 19, pp 226-227, March 1983.

[4] Shepard, K.L., Maynard, D.N. Variability and yield improvement: rules, models, and characterization. *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on* , vol., no., pp.834-835, 5-9 Nov. 2006

[5] Shyam, S., Constantinides, K., Phadke, S., Bertacco, V., and Austin, T. 2006. Ultra low-cost defect protection for microprocessor pipelines. *SIGPLAN* Not. 41, 11 (Nov. 2006), 73-82.

[6] Mitra S., Agarwal M. Circuit failure prediction to overcome scaled CMOS reliability challenges. *International Test Conference,* 2007.

[7] http://www.itrs.net/Links/2007ITRS/Home2007.htm

[8] C. He, et al. A reconfiguration-based defect-tolerant design paradigm for nanotechnologies. *IEEE Design & Test of Computers,* pp. 316-326, July-Aug. 2005.

[9] Borkar S. Y. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro*,Vol.25,Issue.6,pp.10-16,Nov.-Dec. 2005.

[10] Carulli J. M. and Anderson T.J. Test Connections – Tying Application to Process. In *Proc. Intl. Test Conf.*, pp. 679-686, 2005.

[11] Van Horn J. Towards Achieving Relentless Reliability Gains in a Server Marketplace of Teraflops, Laptops, Kilowatts, & "Cost, Cost, Cost" ... (Making Peace between a Black Art and the Bottom Line). In *Proc. Intl. Test Conf.*, pp. 671-678, 2005.

[12] Zafar, S. A Model for Negative Bias Temperature Instability in Oxide and High-K pFETs. *Integrated Circuit Design and Technology, 2007. IEEE International Conference on,* pp.1-5, May 30 2007-June 1 2007.

[13] Denais M., Huard V., Parthasarathy C., Ribes G., Perrier F., Revil N., and Bravaix A. Interface Trap Generation and Hole Trapping Under NBTI and PBTI in Advanced CMOS Technology With a 2-nm Gate Oxide. *IEEE Transactions on Device And Materials Reliability,* vol. 4, no. 4, December 2004.

[14] Wu E. Y., Abadeer W. W., Han L.-K., Lo S.-H., and Hueckel G. Challenges for accurate reliability projection in the ultrathin oxide regime. in *Proc. IRPS,* p. 57, 1999.

[15] Stathis J.H. Reliability limits for the gate insulator in CMOS technology. *IBM Journal of Research and Development* 46 (2/3), pp. 265–286, 2002.

[16] Srinivasan, J., Adve, S.V., Bose, P., Rivers, J.A. The impact of technology scaling on lifetime reliability. *Dependable Systems and Networks, 2004 International Conference on,* pp. 177-186, June-July 2004.

[17] Groseneken G., Degraeve R., Kaczer B., and Rousel P. Recent Trends in Reliability Assessment of Advanced CMOS Technology. In *Proceedings of IEEE 2005 International Microelectronics Test Structure,* vol. 18, April 2005.

[18] Choi K., Soma R., and Pedram M. Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-off based on the Ratio of Off-chip Access to On-chip Computation Times. *Proc. of Design Automation and Test in Europe*, Feb. 2004.

[19] He C., Jacome M. F., and G. de Veciana. A reconfiguration-based defect-tolerant design paradigm for nanotechnologies. *IEEE Design & Test of Computers,* pp. 316-326, July-Aug.2005.

[20] Wang T., Qi Z., and Moritz C. A. Opportunities and challenges in application-tuned circuits and architectures based on nanodevices. In *Proc. International Conference on Computing Frontiers,* pp. 503-511, Apr. 2004.

[21] Koren I. and Koren Z. Defect Tolerant VLSI Circuits: Techniques and Yield Analysis. In *Proc. of the IEEE,* Vol.86, pp.1817-1836, Sept. 1998.

[22] Joseph R. Exploring Salvage Techniques for Multi-core Architectures. *HPCRI-2005 Workshop in Conjunction With HPCA-2005,* February 2006.

[23] Shivakumar P., Keckler, S.W., Moore, C.R., Burger, D. Exploiting microarchitectural redundancy for defect tolerance. *Computer Design, 2003. Proceedings. 21st International Conference on,* vol., no., pp. 481-488, 13-15 Oct. 2003.

[24] Srinivasan, J., Adve, S.V., Pradip Bose, Rivers, J.A. Exploiting structural duplication for lifetime reliability enhancement. *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on,* pp. 520-531, 4-8 June 2005.

[25] Bossen D. C., Kitamorn A., Reick K.F., and Floyd M.S. Fault-tolerant design of the IBM pSeries 690 system using POWER4 processor technology. *IBM Journal Of Research and Development,* vol. 46, no. 1, January 2002.

[26] Bernick, D., Bruckert, B., Vigna, P. D., Garcia, D., Jardine, R., Klecka, J., and Smullen, J. 2005. NonStop® Advanced Architecture. In *Proceedings of the 2005 international Conference on Dependable Systems and Networks,* , pp.12-21, June 28 - July 01, 2005.

[27] Aggarwal, N., Ranganathan, P., Jouppi, N. P., and Smith, J. E. 2007. Configurable isolation: building high availability systems with commodity multi-core processors. *SIGARCH Comput. Archit. News* 35, 2 (Jun. 2007), pp.470-481.

[28] Powell, M. D., Biswas, A., Gupta, S., and Mukherjee, S. S. 2009. Architectural core salvaging in a multi-core processor for hard-error tolerance. *SIGARCH Comput. Archit. News* 37, 3 (Jun. 2009), pp. 93-104.

[29] Intel Corporation. Intel Core 2 Duo Processor and Intel Core 2 Extreme Processor on 45-nm Process for Platforms Based on Mobile Intel 965 Express Chipset Family. htttp://download.intel.com/design/mobile/datashts/31891401.pdf, Jan. 2008.

[30] Romanescu, B. F. and Sorin, D. J. 2008. Core cannibalization architecture: improving lifetime chip performance for multicore processors in the presence of hard faults. In *Proceedings of the 17th international Conference on Parallel Architectures and Compilation Techniques (Toronto, Ontario, Canada, October 25 - 29, 2008). PACT '08*

[31] Tendler J. M., Dodson J. S., Fields J. S. Jr., Le H., and Sinharoy B. POWER4 System Microarchitecture. *IBM Journal Of Research and Development,* vol. 46, no. 1, January 2002.

[32] Stapper, C. H. 1993. Improved Yield Models for Fault-Tolerant Memory Chips. *IEEE Trans. Comput.,* vol. 42, no. 7, pp. 872-881, July 1993.

[33] Leveugle, R., Koren, Z., Koren, I., Saucier, G., Wehn, N. The Hyeti defect tolerant microprocessor: a practical experiment and its cost-effectiveness analysis. *Computers, IEEE Transactions on,* vol.43, no.12, pp.1398-1406, Dec 1994.

[34] Dolbeau R., Seznec A. CASH: Revisiting Hardware Sharing in Single-Chip Parallel Processors. *J. Instruction-Level Parallelism* vol.6, 2004.

[35] Bower, F.A.; Shealy, P.G.; Ozev, S.; Sorin, D.J. Tolerating hard faults in microprocessor array structures. *Dependable Systems and Networks, 2004 International Conference on,* pp. 51-60, June-July 2004.

[36] Bower, F.A., Sorin, D.J., Ozev, S. A mechanism for online diagnosis of hard faults in microprocessors. *Microarchitecture, 2005. MICRO-38. Proceedings. 38th Annual IEEE/ACM International Symposium on,* pp. 12-16 Nov. 2005.

[37] Mukherjee, S. S., Kontz, M., and Reinhardt, S. K. 2002. Detailed design and evaluation of redundant multithreading alternatives. In *Proceedings of the 29th Annual international Symposium on Computer Architecture, Anchorage, Alaska, May 25 - 29, 2002*.

[38] Austin T. M. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. *Proc. of the 32nd Annual International Symposium on Microarchitecture,* pp.196-207, Nov. 1999.

[39] Bushnell, M. and Agrawal, V.D. Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits. *Kluwer Academic,* 2000.

[40] Schuchman, E., Vijaykumar, T.N. Rescue: a microarchitecture for testability and defect tolerance. *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on,* vol., no., pp. 160-171, 4-8 June 2005.

[41] Jien-Chung Lo. Reliable floating-point arithmetic algorithms for error-coded operands. *Computers, IEEE Transactions on,* vol.43, no.4, pp.400-412, Apr 1994.

[42] Shekarian, S.M.H., Ejlali, A., Miremadi, S.G. A Low Power Error Detection Technique for Floating-Point Units in Embedded Applications. *Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on,* vol.1, no., pp.199-205, 17-20 Dec. 2008.

[43] R Renau J., et al. SESC Simulator, January 2005. http://sesc.sourceforge.net.