To my mother and father,
and my fiancé Zongyu
ACKNOWLEDGMENTS

Without the support of many, I would not have successfully journeyed through this beginning. My first, and most earnest, acknowledgment must go to my advisor, Dr. Tao Li for his encouragement, advice, mentoring, and research financial support. His tough and incisive questions, unending drive and dedication to the field of computer architecture never cease to amaze me. Not only was he readily available for me, as he so generously is for all of his students, but he always read and responded to the drafts of my work more quickly than I could have hoped. In every sense, none of this work would have been possible without him.

I would like to gratefully acknowledge my co-advisor, Dr. José A. B. Fortes for his advice and support. Jose has been instrumental in ensuring my academic, professional, financial, and moral well being ever since. His patience and encouragement always inspire me to explore and challenge further. Many thanks also go to committee members Dr. Renato J Figueiredo and Dr. Jih-Kwon Peir for their valuable comments, productive suggestions, and the time for reading the draft of my thesis.

I would like to thank the students (past and current) at Intelligent Design and Efficient Architectures Laboratory (IDEAL) and Advanced Computing and Information Systems (ACIS) Laboratory. I want especially to thank (in alphabetical order) Chang-Burm Cho, Clay Hughes, James Michael Poe II, and Wangyuan Zhang for helping me setting up the simulation environments used in this research, sharing their technical wisdom and research ideas, and providing valuable comments on drafts of my paper submissions and useful feedback at practice talks. I also thank Prapaporn Rattanatamrong, David Wolinsky, Liping Zhu, Ming Zhao, Jing Xu, Yuchu Tong, Jiangyan Xu, Jian Zhang, Girish Venkatasubramanian, Pierre Tony St. Juste, Selvi Kadirvel, Priya Chaman Bhat, Erin taylor, Mauricio Tsugawa, Andréa Matsunaga, and Yonggang Liu for their friendships and their collective encouragement to finish this dissertation.
A penultimate thank-you goes to my friends (Qiong Zhou, Haoyang Zhuang, Jian Sun) for their help and support throughout the completion of this degree.

I express my gratitude to the High Performance Computing (HPC) Center at University of Florida for providing me the simulation resources. Thanks to Dina, Cathy, Shannon, Julie, and other administrative assistants who has worked in Computer Engineering in the past years. During the course of my research, I have submitted several papers to peer-reviewed conferences. The anonymous reviewers have provided valuable insights, pointers to literature, and criticisms that I have used to make my research stronger.

My final, and most heartfelt, acknowledgement must go to my wonderful parents Yafei Wang and Zhongyang Fu, my fiancé Zongyu Zhai, and my family (my grandfather, uncles and aunts), for their love, support, and encouragements. They are always there when I need them most, they deserve far more credit than I can ever give them. This dissertation is dedicated to them.
A Hierarchical Design of NoC That Mitigates the Effects of NBTI and PV ..........................184
Intra-Router Level NoC PV and NBTI Optimization .................................................................184
  NBTI and PV optimization for NoC combinational-logic structures ..........................185
  NBTI and PV optimization for NoC storage cell based structures ..........................190
Inter-Router Level NoC PV and NBTI Optimization .........................................................193
Evaluation .............................................................................................................................196
  Experimental Methodologies .........................................................................................196
  Effectiveness of the VA_SA_M1 ..............................................................................196
  Effectiveness of VC_M2 .........................................................................................198
  Effectiveness of IR_M3 .........................................................................................199
Real Workload Results ..................................................................................................200
Related Work ....................................................................................................................201

9 CONCLUSIONS AND FUTURE WORKS...........................................................................220
Conclusions ...........................................................................................................................220
Future Works .......................................................................................................................224

LIST OF REFERENCES ..........................................................................................................226

BIOGRAPHICAL SKETCH ....................................................................................................239
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>A comparison of different architectural level reliability analysis tools</td>
<td>38</td>
</tr>
<tr>
<td>2-2</td>
<td>Trivial instructions (in Alpha ISA) identified by Sim-SODA</td>
<td>38</td>
</tr>
<tr>
<td>2-3</td>
<td>Simulated machine configuration</td>
<td>39</td>
</tr>
<tr>
<td>2-4</td>
<td>SPEC 2000 INT and BioInfoMark benchmarks (The data set name is integrated with the benchmark name)</td>
<td>39</td>
</tr>
<tr>
<td>3-1</td>
<td>Variation in run-time microarchitecture vulnerability</td>
<td>53</td>
</tr>
<tr>
<td>3-2</td>
<td>Events used by PMC-15 and PMC-5 schemes</td>
<td>53</td>
</tr>
<tr>
<td>3-3</td>
<td>Number of phases identified by the K-means clustering algorithm</td>
<td>54</td>
</tr>
<tr>
<td>3-4</td>
<td>AVF phase classification error on different machine configurations</td>
<td>54</td>
</tr>
<tr>
<td>4-1</td>
<td>Accuracy of using PC to identify ACE instructions (committed instruction only)</td>
<td>83</td>
</tr>
<tr>
<td>4-2</td>
<td>Simulated machine configuration</td>
<td>83</td>
</tr>
<tr>
<td>4-3</td>
<td>The studied SMT workloads</td>
<td>84</td>
</tr>
<tr>
<td>4-4</td>
<td>A summary of the proposed techniques</td>
<td>84</td>
</tr>
<tr>
<td>5-1</td>
<td>The studied SMT workloads</td>
<td>112</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>ACE and un-ACE lifetime partitions due to the different orders of un-ACE reads (marked as read*) and ACE reads</td>
<td>40</td>
</tr>
<tr>
<td>2-2</td>
<td>Instruction level vulnerability profile of the studied benchmarks</td>
<td>40</td>
</tr>
<tr>
<td>2-3</td>
<td>AVF of instruction window</td>
<td>40</td>
</tr>
<tr>
<td>2-4</td>
<td>AVF of instruction window and wake-up table</td>
<td>41</td>
</tr>
<tr>
<td>2-5</td>
<td>AVF of ROB</td>
<td>41</td>
</tr>
<tr>
<td>2-6</td>
<td>AVF of register files</td>
<td>41</td>
</tr>
<tr>
<td>2-7</td>
<td>AVF of function unit</td>
<td>42</td>
</tr>
<tr>
<td>2-8</td>
<td>Data array AVF of L1 data cache (DL1), data TLB (DTLB), victim buffer (VBuf), load queue (LQ) and store queue (SQ)</td>
<td>42</td>
</tr>
<tr>
<td>2-9</td>
<td>Tag array AVF of L1 data cache (DL1), data TLB (DTLB), victim buffer (VBuf), load queue (LQ) and store queue (SQ)</td>
<td>42</td>
</tr>
<tr>
<td>3-1</td>
<td>Run-time instruction window AVF profile on benchmark gcc and parser</td>
<td>55</td>
</tr>
<tr>
<td>3-2</td>
<td>The correlations between AVFs and a simple performance metric. A) Instruction windows. B) Reorder buffer. C) Function unit. D) Wakeup table.</td>
<td>56</td>
</tr>
<tr>
<td>3-3</td>
<td>COVs yielded by different phase classification schemes. A) Instruction windows. B) Reorder buffer. C) Function unit. D) Wakeup table.</td>
<td>57</td>
</tr>
<tr>
<td>3-4</td>
<td>COVs yielded by hybrid schemes. A) Instruction windows. B) Reorder buffer. C) Function unit. D) Wakeup table.</td>
<td>58</td>
</tr>
<tr>
<td>4-1</td>
<td>Microarchitecture soft-error vulnerability profile on SMT processor</td>
<td>85</td>
</tr>
<tr>
<td>4-2</td>
<td>The histograms of ready queue length and ACE instruction percentage on a 96-entry IQ. SMT processor issue width = 8, the experimented workloads: 4-context CPU (bzip, eon, gcc, perl, bmk)</td>
<td>85</td>
</tr>
<tr>
<td>4-3</td>
<td>Dynamic IQ resource allocation based on IPC, ready queue length and total IQ size</td>
<td>85</td>
</tr>
<tr>
<td>4-4</td>
<td>L2-cache-miss sensitive IQ resource allocation</td>
<td>86</td>
</tr>
</tbody>
</table>
4-5 Reliability and performance with ICOUNT fetch policy. A) Normalized IQ AVF. B) Normalized throughput IPC.

4-6 Reliability and performance using different fetch policies. A) Normalized IQ AVF. B) Normalized IPC.

4-7 A) IQ AVF contributed by waiting instructions and ready instructions. B) Profiles of the quantity of ready instructions and waiting instructions in a 96 entries IQ. C) Residency cycles of ready instructions and waiting instructions in a 96 entries IQ. The ICOUNT fetch policy is used.

4-8 ORBIT with prediction.

4-9 Predict the load instruction completion time.

4-10 Readiness prediction example on PredictALL and Predict_non_load.

4-11 Reliability and performance results when using ICOUNT fetch policy. A) Normalized IQ AVF. B) Normalized throughput IPC. C) Normalized harmonic IPC.

4-12 A Comparison with different fetch policies. A) IQ AVF. B) Throughput IPC. C) Harmonic IPC.

4-13 The Impact of proposed techniques on core and other microarchitecture structures’ vulnerability (DALL => DelayALL, PDACE => Predict_DelayACE).

4-14 (A) IQ AVF contributed by waiting instructions and ready instructions. B) Profiles of the quantity of waiting instructions in the 96 entries IQ with Predict_DelayACE. C) Profiles of residency cycles of waiting instructions in the 96 entries IQ with Predict_DelayACE. The ICOUNT fetch policy is used.

5-1 Soft error robust SRAM (rSRAM) Cell (6T+2C). The rSRAM cell is built from a standard 6 transistor high density SRAM cell above which two stacked Metal-Insulator-Metal (MIM) capacitors are symmetrically added. The embedded capacitors increase the critical charge required to flip the cell logic state and lead to a much lower SER. The common node of the two capacitors is biased at VDD/2.

5-2 The control flow of instruction dispatch in the proposed IQ using hybrid radiation hardening techniques.

5-3 An overview of radiation hardened IQ design using hybrid techniques.

5-4 The wakeup logic of the RIQ.

5-5 The designs alternative of the hybrid radiation hardened IQ. A) First design. B) Second design.

5-6 The correlation between ROB AVF and L2 cache miss.
A comparison of normalized IQ SER, throughput and harmonic IPCs. A) Normalized IQ SER. B) Normalized throughput IPC. C) Normalized harmonic IPC.


ROB SER reduction and processor power overhead with L2_miss trigger and enhanced trigger. A) ROB SER reduction. B) Power overhead.


The aggregate effect of the proposed two techniques.

Multi-level quad-tree partitioning to model system variation.

Standard 6T SRAM schematics with current source inserted at Q.

V(Q) and V(QB) of SRAM (in 45nm processing technology) under a particle strike. A) Without PV. B) In the presence of PV.

Critical charge variation map for a chip (data are presented in units of fC).

Q_{crit_entry} of each entry in the register file (80 entries, each entry contains 64 bits).

Entry- and bit-level Q_{crit} distribution in the register file.

A) IQ AVF in baseline case and VL-RF technique on gcc over a period of 375ms. B) The # of issue pipe stalls and IQ AVF difference between VL-RF and baseline case.

Critical charge vs. delta threshold voltage in range of [-0.3v, 0.3v] in 6T SRAM.

Entry-BVM architectural overview.

BB in each phase.

Structure-BVM Pseudo-code.

Normalized IQ SER and IPC*f_{pv} yielded by VL related techniques. A) Normalized IQ SER. B) Normalized IPC*f_{pv}.

Normalize VPP on VL related techniques.

Normalize IQ SER with BB related techniques.

Normalized IPC*f_{pv} with BB related techniques.

Normalized VPP with BB related techniques.
6-17 $AVF_{pv} \cdot IPC \cdot f_{pv}$ plot for intervals on *equake* (up) and *swim* (down) (Intervals within the drawn area are close to the mean value) ..................................................150

6-18 A) Normalize IQ SER under Entry-BVM + Structure-BVM. B) Normalized VPP under Entry-BVM + Structure-BVM ..........................................................151

7-1 Different guardband settings for tolerating NBTI ..........................................................174

7-2 The limitation of simply combining NBTI and PV mitigation techniques ....................174

7-3 2-Read port register files with detailed read port design .............................................174

7-4 Cycle time under NBTI and PV effects. A) Baseline case without optimization. B) O1 ..................................................................................................................175

7-5 Pseudo code for port switching in O1 ..........................................................................175

7-6 Examples of PS in O1 ..................................................................................................176

7-7 An example of 2-to-4 decoder .....................................................................................176

7-8 O2 circuit design ........................................................................................................177

7-9 $V_{th}$ (in mV) variation map for a cache ......................................................................177

7-10 Fundamental idea of O3 in the 4-way L1 cache ..........................................................178

7-11 The effectiveness of O1 in RF. A) Normalized CPI. B) Normalized NBTI guardband. C) Normalized $NBTI&PV_{efficiency}$ ..........................................................179

7-12 The effectiveness of O2 in IntALU. A) Normalized $NBTI&PV_{guardband}$. B) Normalized $NBTI&PV_{efficiency}$ ..............................................................180

7-13 The effectiveness of O3 in L1 cache. A) Normalized CPI. B) Normalized $NBTI&PV_{efficiency}$ ..........................................................181

7-14 NBTI&PV_{efficiency} with various granularities .......................................................181

8-1 Two-stage adaptive router microarchitecture .............................................................203

8-2 Circuit design of the two-stage adaptive router [28]. A) Circuit design of the two-stage adaptive router. B) Zoom-in view at VA logic ...........................................204

8-3 Circuit design in the first step of the VA stage (one VC entry is shown) .......................205

8-4 The implementation of VA_M1 ..................................................................................205

8-5 The guardband and power benefit of applying AVS+ during the lifetime. A) Guardband benefit. B) Power benefit .................................................................206
8-6 $V_{th}$ (in mV) variation map for a router .................................................................206

8-7 The implementation of VC_M2 ..............................................................................207

8-8 The implementation of IR_M3 ..................................................................................208

8-9 The effectiveness of VA_SA_M1 on NBTI&PV_overhead and NBTI_guardband. A) Uniform random traffic. B) Bit-complement traffic. C) Transpose traffic. D) Tornado traffic .................................................................209

8-10 Sensitivity analysis on threshold (URT with 0.3flit/node/cycle injection rate) ..........211

8-11 The effectiveness of VC_M2 on network latency and NBTI&PV_overhead. A) Uniform random traffic. B) Bit-complement traffic. C) Transpose traffic. D) Tornado traffic .................................................................212

8-12 The Effectiveness of IR_M3 on Network latency and NBTI_guardband. A) Uniform random traffic. B) Bit-complement traffic. C) Transpose traffic. D) Tornado traffic .................................................................214

8-13 The effectiveness of IR_M3 on NBTI&PV_overhead. A) Uniform random traffic. B) Bit-complement traffic. C) Transpose traffic. D) Tornado traffic .................................................................216

Abstract of Dissertation Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

CHARACTERIZING, MODELING AND MITIGATING MICROARCHITECTURE VULNERABILITY AND VARIABILITY IN LIGHT OF SMALL-SCALE PROCESSING TECHNOLOGY

By

Xin Fu

December 2009

Chair: Tao Li
Cochair: José A. B. Fortes
Major: Electrical and Computer Engineering

The rapidly increased soft error rate (SER) due to the scaled processing technology is one of the critical reliability concerns in current processor design. In order to characterize and mitigate the microarchitecture soft-error vulnerability in modern superscalar and multithreaded processors, I developed Sim-SODA (Software Dependability Analysis), a unified framework for estimating microprocessor reliability in the presence of soft errors at the architectural level. By using Sim-SODA, I observed that a single performance metric is not a good indicator for program vulnerability; on the other hand, a combination of several performance metrics can well predict the architecture-level soft-error vulnerability. Based on the observation that issue queue (IQ) is a reliability hot-spot on Simultaneous Multithreaded (SMT) processors, I proposed VISA (Vulnerable InStruction Aware) Issue and ORBIT (Operand Readiness Based InsTruction) dispatch to improve the IQ reliability. I further combined the circuit and microarchitecture techniques in soft error robustness on SMT processors to leverage the advantage of the two levels’ techniques while overcoming the disadvantage of both. Results show that my proposed techniques have strong ability in improve IQ reliability with negligible performance penalty.
As one of the nano-scale design challenges, process variation (PV) significantly affects chip performance and power. I characterized the microarchitecture soft error vulnerability in the presence of PV, and proposed two techniques that work at fine-grain and coarse-grain levels to mitigate the impact of PV mitigation techniques on reliability and maintain optimal vulnerability, performance, and power trade-offs. Negative Body Temperature Instability (NBTI) has become another important reliability concern as processing technology scaled down. Observing that PV has both positive and negative effects on circuits, I took advantage of the positive effects in NBTI tolerant microarchitecture design to efficiently mitigate the detrimental impact of PV and NBTI simultaneously. The trend towards multi-/many- core design has made network-on-chip (NoC) a crucial hardware component of future microprocessors. I proposed several techniques that hierarchically mitigate the PV and NBTI effect on NoC while leveraging their benign interplay.
CHAPTER 1
INTRODUCTION

Recently, VLSI technology continues to provide increasing numbers of transistors and clock speeds to allow computer architects to build powerful microprocessors and computer systems. On the other hand, the continuously scaled processing technology at nano-scale exacerbates processor reliability and variability issues, which lead to high error rates in processors, greatly affect the processor lifetime, and result in substantial performance loss and power overhead. High availability and reliability are essential for any computer system. Now, it is crucial for computer architects to quantifying and mitigating the deleterious vulnerability and variability factors. This chapter (1) introduces three critical reliability and variability issues: soft errors, negative biased temperature instability, and process variations; (2) describe the necessity for mitigating those vulnerability and variability factors; (2) presents the objectives and contributions of this dissertation.

Processor Vulnerability and Variability

With the advance of semiconductor processing technologies, several critical reliability issues become the major causes of processor failures. For instances, soft error rate increases significantly as transistor size becomes smaller; Negative Bias Temperature Instability (NBTI), a wear-out mechanism, has become the considerable reliability concern. Moreover, it becomes harder to control sub-wavelength lithography and channel doping which results in Process Variation (PV), those variations cause a large amount of frequency loss, leakage power consumption and also affect the chip lifetime.

Soft Errors

Soft errors [2, 27, 28], also referred to as transient faults, or Single Event Upsets (SEU)—can cause single or multiple bit flips in memory, flip-flops and registers, and finally lead to a
wrong computation even in perfectly working circuit. As opposed to hard errors, soft errors are not due to physical damage but rather are caused by unpredictable, temporary and environmental conditions. By the virtue of their nature, soft errors are out of the scope of testing and verification, and are difficult to pinpoint on the field.

Soft errors are caused by high-energy (more than 1 MeV) neutrons from cosmic radiation, alpha particles emitted by trace uranium and thorium impurities in packaging materials and low-energy cosmic neutron interactions with the isotope boron-10 \((^{10}\text{B})\) in IC materials used to form insulator layers in manufacturing. When the particles pass through the semiconductor devices, electron-hole pairs are generated. The source and diffusion nodes will collect the charges, a sufficient amount of accumulated charges will invert the state of the logic device and cause a bit flip. The error will not be restored until the device is reset or rewritten.

Safety-critical applications are severely threatened by soft errors. For instance, the electronics must operate correctly in high latitude and harsh environment conditions in aero spatial and military area; in automotive and transportation, the failure of antilock brake system caused by soft errors can even endanger human lives; the attack of soft errors in banking systems can results in a substantial monetary losses.

Soft errors are not a new problem, they became well known during 1970s’ with the introduction of RAM. They originally cause a simple problem in applications in high risk environment. However, as the semiconductor technology scales down to the nano-scale, it results in the increasing complexity of microprocessors, the shrinking of transistor size, the lowing of supply voltage and the increasing number of gates on the same die. Therefore, soft-error rates of future generations of processors are projected to increase significantly.
**Negative Biased Temperature Instability**

Negative bias temperature instability (NBTI) is a considerable reliability concern for sub-micrometer CMOS technologies. NBTI occurs in PMOS devices when the gate-source voltage is negative \( V_{gs} = -V_{dd} \). NBTI increases the threshold voltage \( V_{th} \) and reduces the drive current \( I_{dsat} \), which causes degradation in circuit speed and requires a minimum voltage \( V_{min} \) increase in storage cells to keep the content. Eventually, this will lead to failures in logic circuits and storage structures due to timing violations or \( V_{min} \) limitations. The NBTI effect in PMOS transistors, which stems from an electro-mechanical reaction involving the electric field, holes, Si-H bonds, and temperature, is not a recently discovered wear-out mechanism. It was originally observed in the early phases of CMOS development (almost 40 years ago), but was not considered important because of the low electric fields under normal operating conditions. However, technology scaling has resulted in the convergence of several factors (e.g. the introduction of nitrided oxides, the increase in gate oxide fields, and operating temperature), which have made NBTI the most critical reliability concern for deep sub-micrometer transistors [98, 99, 100]. For example, it has been observed that NBTI can increase \( V_{th} \) by as much as 50mV for devices operating at 1.2V or below [99] and the circuit performance degradation may extend up to 20% in 10 years [100].

**Process Variation**

Process variation (PV), the divergence of transistor process parameters from their nominal specifications, results in variability in circuit performance/power and has become a major challenge in the design and fabrication of future microprocessors [101, 102, 103, 104]. For example, chip frequency can be degraded by as much as 30% in 45nm process technology due to process variation [102] and a 20x increase in leakage power consumption is reported in [101]. PV is caused by the difficulty in controlling the sub-wavelength lithography and channel doping.
as process technology scales. Process variation consists of die-to-die (D2D) and within-die (WID) variations. Die-to-die variation consists of parameter fluctuations across dies and wafers, whereas within-die variation refers to variations of design parameters within a single die. As technology scales, within-die variation, which is the primary focus of this study, has become more significant and is a growing threat to future microprocessor design [101, 102].

Contributions

This dissertation makes several contributions to model, characterize, and mitigate microarchitecture soft error vulnerability in microprocessors and multi-threaded processors; characterize and mitigate the soft error vulnerability in the presence of process variation; build the NBTI-tolerant microarchitecture design under the impact of process variation; and further improve the reliability in network-on-chips. The summary of the contributions is listed below:

1. In order to improve the soft error robustness, architects need a framework to understand the structures’ vulnerability and evaluate the effectiveness of their proposed vulnerability mitigation techniques. I developed Sim-SODA (SOftware Dependability Analysis). Sim-SODA, built on top of the Sim-Alpha tool sets, is the first unified simulation framework for high-performance microprocessor reliability estimation in the presence of soft errors. Sim-SODA is public available. It has been used by more than 90 universities and research labs (e.g. Cornell University, Louisianan State University, and Lawrence Livermore National Lab) for their research projects.

2. Characterizing and predicting program phase behavior from a reliability perspective is crucial in order to apply dynamic fault-tolerant mechanisms and to optimize performance/reliability trade-offs. I found that a single performance metric (e.g. IPC, cache miss) is not a good indicator for program vulnerability. The vulnerabilities of the structures are correlated with program code-structure and run-time events to identify vulnerability phase behavior. I observed that in general, tracking runtime performance metrics performs better than tracking program control flow in vulnerability phase classification.

3. The issue queue (IQ) is a key microarchitecture structure for exploiting instruction-level and thread-level parallelism in dynamically scheduled SMT processors. However, exploiting more parallelism yields high susceptibility to transient faults on a conventional IQ. With the rapidly increasing soft error rates, the IQ is likely to be a reliability hot-spot on SMT processors. I proposed reliability-aware instruction scheduling (VISA) and resource allocation to reduce the quantity and residency cycle of vulnerable instructions in the IQ, therefore, optimize the IQ reliability to soft errors. Moreover, I observed that IQ soft-error vulnerability is largely affected by instructions waiting for their source operands.
I explored operand-readiness-based instruction dispatch (ORBIT) to minimize the number of waiting instructions in IQ and reduce the IQ vulnerability with negligible performance degradation. Furthermore, I extended ORBIT with prediction methods that can anticipate the readiness of source operands ahead of time to achieve more attractive reliability/performance trade-offs.

4. Techniques for structures’ reliability improvement in SMT processors exist on both circuit and microarchitecture levels, but there are relatively few studies that cost-effectively integrate them together. I bridged the gap by proposing combined circuit and microarchitecture techniques to leverage their advantage while overcoming the disadvantages. The combined techniques achieve significant reliability improvement in multithreaded environment.

5. As transistor process technology approaches the nanometer scale, process variation (PV) significantly affects chip performance and power. However, the impact of process variation on soft error vulnerability is not well studied. I characterized the microarchitecture soft error vulnerability in the presence of PV, and proposed two techniques that work at fine grain (entry-based) and coarse grain (structure-based) levels to mitigate the deleterious impact of PV mitigation techniques on reliability and maintain optimal vulnerability, performance, and power trade-offs.

6. Negative bias temperature instability (NBTI), which increases the delay and reduces the lifetime of PMOS transistors, is becoming a growing reliability concern for sub-micrometer CMOS technologies. Process variation (PV) can exacerbate the PMOS transistor wear-out problem and further reduce the reliable lifetime of microprocessors. Observing that PV also has positive effect, I proposed to take advantage of the PV positive effect in NBTI tolerant microarchitecture design which effectively mitigate the detrimental impact of PV and NBTI simultaneously, while achieving good trade-offs among chip performance, power, lifetime, and area overhead.

7. The trend towards multi-/many-core design has made network-on-chip (NoC) a crucial hardware component of future microprocessors. Meanwhile, process variation (PV) and negative bias temperature instability (NBTI) increasingly affect hardware reliability and lifetime. I proposed novel techniques that can hierarchically mitigate the PV and NBTI effect on NoC while leveraging their benign interplay. The low-level mechanisms improve PV and NBTI efficiency of key components (e.g. virtual channels, switch arbiters) that are in the critical paths of the pipelined router microarchitecture. The high-level mechanisms leverage NBTI degradation and PV information from multiple routers to intelligently route the packets, delivering optimized performance-power-reliability efficiency across the NoC substrate.
Dissertation Organization

The rest of the dissertation is organized as follows: Chapter 2 describes the development of the unified simulation framework to estimate microarchitecture processor vulnerability in the presence of soft errors. Chapter 3 characterizes the microarchitecture soft-error vulnerability phase behavior. Chapter 4 proposes two architecture level techniques to mitigate the soft-error vulnerability of issue queue in the simultaneous multithread architectures. Chapter 5 combines the circuit-level and microarchitecture-level techniques to efficiently reduce the soft error rate of microarchitecture structures in simultaneous multithreaded architectures. Chapter 6 characterizes the impact of process variation on soft error vulnerability, improves processor reliability and variability, and meanwhile, achieves the optimum trade-offs among performance, reliability, and power. Chapter 7 takes advantage of the positive effects in NBTI tolerant microarchitecture design to efficiently mitigate the detrimental impact of process variation and NBTI simultaneously. Chapter 8 targets on the reliability enhancement in network-on-chip, and proposes several techniques to hierarchically mitigate the process variations and NBTI effects on network-on-chip. Chapter 9 concludes the dissertations and suggests future opportunities.
CHAPTER 2
SIM-SODA: SOFTWARE DEPENDABILITY ANALYSIS

Sim-SODA estimates the dependability of hardware components in a high-performance, out-of-order superscalar microprocessor using the computation methods introduced in [4, 22]. Compared with previous studies [4, 19, 22], Sim-SODA provides a unified infrastructure to study the reliability of all major units of a high-performance microprocessor with a single run, proposes a fine-grained reliability analysis to improve the accuracy of the reliability estimation, and also proposes a hybrid method that can be used to accurately estimate the vulnerability of complex structures. While previous architectural reliability analysis tools were built on proprietary performance models [13, 24], Sim-SODA uses an open source, publicly available simulator Sim-Alpha [11, 12], which makes porting the reliability analysis framework described in this dissertation to other popular simulator tool suites (such as Simplescalar[6] and M5 [3]) relatively easy.

Related Work

There has been prior work on dependability modeling at a high level. For example, hardware RTL models have been used in the past to estimate processor reliability [25, 27]. The RTL models contain all of the detailed information about the microprocessors. Nevertheless, the simulation slowdown of RTL models is too expensive for architecture studies, in which the tradeoffs between many hardware configurations need to be considered. Moreover, these models are generally not available during the architectural exploration phase of a microprocessor design. The Architectural Vulnerability Factor (AVF) analysis methods proposed by Mukherjee et al used a performance model to generate reliability estimates. In [4, 22] the vulnerability of hardware structures (e.g. issue queue, execution unit, TLB and caches) of an Itanium2-like IA64 processor was studied. In [1], Asadi et al estimated the vulnerability of L1 cache through the
residency time of critical words in the cache. In [19], Li and Adve developed SoftArch, an architecture level tool for modeling and analyzing soft errors. The SoftArch framework estimates reliability using a probabilistic model of the error generation and propagation process in a processor. As a complementary approach to AVF computation, statistic fault injection has been used in several studies [5, 10, 25, 27] to evaluate architectural reliability. To obtain statistic significance, a large number of experiments need to be performed on an investigated hardware component. Table 2-1 summarizes the features of several architectural reliability estimation tools from the perspectives of methodology, modeled hardware structures and the availability of baseline models.

**Microarchitecture Soft-Error Vulnerability Estimation**

Sim-SODA estimates microprocessor reliability using the Architectural Vulnerability Factor (AVF) computing methods introduced in [4, 22]. In this section, I briefly review the concept and the computation of AVF. Since not all soft errors can cause erroneous program execution, the probability that a fault in a hardware structure will cause an externally visible error in the final output of a program is referred to as the architectural vulnerability factor (AVF) of that hardware structure. A hardware structure’s error rate is the product of its raw error rate, mainly determined by device and circuit design technology, and the AVF. The key to calculating the AVF is to determine which bits affect the final system output and which do not. In [22], a subset of processor state bits required for architecturally correct execution (ACE) are called ACE bits. Hence, the AVF of a hardware structure in a given cycle is the percentage of the ACE bits in that structure. The AVF of a hardware structure during program execution is the average AVF at any point in time.
The Sim-SODA Reliability Estimation Framework

Overview

I have developed Sim-SODA, an architectural framework to estimate the reliability of programs running on high-performance, out-of-order microprocessors. To track the residence time of ACE bits in various structures, I instrumented Sim-alpha, an open source, validated cycle-accurate performance simulator for Alpha 21264. In the Sim-SODA framework, I classify each dynamic instruction of a program's execution based on whether the instruction’s output affects the outcome of that program. Since instructions executed along a mispredicted path will not be committed, and do not affect AVF, I only considered committed instructions. I consider an instruction an ACE instruction if its results might affect the final program output, and an instruction un-ACE if its results definitely will not affect the program output. Bits in an ACE instruction are ACE, but an un-ACE instruction contains both ACE and un-ACE bits (details are explained in [22]). Additionally, I classify one type of un-ACE instruction dynamically dead if its results are not used subsequently (more detailed classification of un-ACE instructions will be introduced in section 5.1). In Sim-SODA, I implemented the post-commit analysis window proposed in [22] to determine if the instruction is dynamically dead or if there are any bits that are logically masked. Through cycle-level simulation, both microarchitecture and architecture states are classified into ACE/un-ACE bits and their residency and resource usage counts are generated. This information is then used to estimate the reliability of various hardware structures.

Fine-Grained Reliability Estimation

Instruction window

In high performance processors, the instruction window is used to support dynamic scheduling and out-of-order execution. In [22], the instruction window is treated as a bulk structure. Sim-SODA provides fine-grained reliability analysis for the instruction window. When
an instruction completes its execution, its destination register ID is broadcasted to all the
instructions in the window to inform all dependent instructions of the availability of the result.
Each entry compares the broadcasted register ID with its own source register ID. If there is a
match, the source operand is latched and the dependent instruction may be ready to execute. This
register ID broadcast and associated comparison is called instruction wake-up. A soft error that
results in an incorrect match between a broadcasted physical register ID and a corrupted tag may
cause instructions waiting for that operand to be issued pre-maturely. A single bit error in the tag
array that results in a mismatch where there should have been a hit can prevent ready instructions
from being issued, causing a deadlock in issuing instructions. Therefore, the wake-up table is
vulnerable to soft error strikes. The Sim-SODA framework estimates the vulnerability of both
the instruction window and the wake-up table.

When a new instruction is allocated in the instruction window, the wake-up table records
the renamed physical register IDs of instructions on which that instruction depends. There are
two fields in each wake-up table entry to hold the renamed register IDs for the two source
operands of an instruction. A field in the wake-up table entry becomes invalid once the source
operand is ready. The operations on the wake-up table include “fill”, “read” and “invalidate”;
therefore, its non-overlapping lifetime can be partitioned into fill-to-read, read-to-read and
invalidate-to-fill periods. Note that there is no read-to-invalidate component because the last read
between fill and invalidate will cause a match between the stored register ID and the broadcasted
register ID. Once there is a match, the field in the wake-up table will become invalid
immediately. In other words, the lifetime of the read-to-invalidate component in the wake-up
table is always zero. Therefore, I combine fill-to-read and read-to-read components together, and
attribute the invalidate-to-fill component as un-ACE.
**Trivial instruction**

In [22], Mukherjee et al identified logical masking instructions as a source of un-ACE bits. An operand and its bits are logically masked and can be attributed to un-ACE bits if the operand does not influence the result of an instruction execution. In their study, Mukherjee et al considered three types of logical masking instructions: compare instructions prior to a branch, bitwise logical operations and 32-bit operations in a 64-bit architecture. In this study, I identified further logical masking bits. I found that the bits used to encode the specifiers of source registers which hold logically masked values are un-ACE bits. This is because a corrupted register specifier may cause the processor to fetch the wrong data from a different register. Nevertheless, the computation result will not be altered because of the logical masking effect.

Additionally, I extend logical masking instructions to trivial instructions [29] in this study. Trivial instructions are those computations whose output can be determined without performing the computation, so they cover all the un-ACE bits that logical masking instructions can identify. In this study, I further classified the trivial instructions into the following three categories. The first type of trivial instructions has two source registers. For these trivial instructions, a soft error is tolerant when it strikes a register whose contribution to the computation result is masked by the second register. For example, in a multiplication instruction, if one of the source registers is equal to zero, a soft error that hits the other register would not affect the result. Therefore, the bits held by that source register are un-ACE bits. Additionally, the bits used for encoding the other source register specifier within the same instruction are also un-ACE bits. The second type of trivial instructions contains an immediate value and only one source register. The bits in the source registers can be considered un-ACE when the immediate value masks the instructions contribution to the computation results. Similarly, bits in the immediate value can be considered un-ACE when the source register doesn’t affect the computation result. The source register
specifier bits in that instruction become un-ACE if the value held in that register is un-ACE. Note that in both types of trivial instructions, only one of the source operands can be considered un-ACE at a time, since a soft error hit to the operand which provides the masking function will skew the computation results. The last type of trivial instructions is specific to XOR and EQV operations (see Table 2-2). For these two operations, when the first and second register specifiers are identical, bits in that register are un-ACE. Table 2-2 summarizes the trivial instructions identified by the Sim-SODA framework. The particular source operand value that trivializes the operation is also listed.

Integer adds and subtractions are not included in Table 2-2. This is because their computation results depend on both operands. A bit change in either operand may result in incorrect computation output. Integer divisions are not listed in Table 2-2 because a division operation is implemented through multiplication instructions in Alpha instruction set [8, 9, 16]. Note that trivial instructions can be dynamically dead instructions also. If an instruction is both a trivial instruction and a dynamically dead instruction, I attribute it to the dynamically dead instructions because more un-ACE bits can be derived from that type of instruction. In other words, trivial instructions analyzed by the Sim-SODA framework are special ACE instructions that have un-ACE bits in the instructions and their registers.

Reliability Estimation of Unexplored Structures

Hybrid AVF computation for register files

The register files hold the architectural state of program execution. Previous studies [19, 25, 27] show that the register files are highly vulnerable to soft errors. In these studies, the reliability analysis of register files was performed by injecting faults statistically or modeling error propagation. The AVF calculation for register files has not been addressed.
When comparing register files with the data cache, I summarize some common characteristics between the two. The register files are similar to the data array in a data cache: both are used to keep values for instruction execution. Activities occurring during the lifetime of a bit in the register files also include “idle”, “fill”, “read”, “write” and “evict”. Therefore, if I follow the methodology to compute the AVF for address-based structures [4], I can also classify the register files’ lifetimes into non-overlapping ACE or un-ACE periods. For example, idle, read-to-write and write-to-write are un-ACE, while fill-to-read and write-to-read are ACE. However, the calculation of register files’ AVF in this way can be very conservative. This is because unlike the data cache, registers are heavily utilized and write and read operations occur very frequently.

To obtain a more realistic estimate of register files’ AVF, I analyzed each ACE lifetime component to discover which operations on the bits convert ACE lifetime to un-ACE lifetime. I found that not every read affects the final program output. For example, if there is a read caused by a dynamically dead instruction, the final output will not change even if the data is incorrect. Written data is also un-ACE when the write is caused by a dynamically dead instruction. Additionally, as described in section 3.2.2, when a trivial instruction has a read operation on bits of the register files, the read data is un-ACE if it doesn’t affect the result of the instruction. Since I combine ACE/un-ACE lifetime analysis with an instruction analysis window (which is used to detect dynamically dead instructions and trivial instructions, see section 4), I call the method a hybrid AVF computation scheme.

A write to the register file is usually followed by more than one read. Two types of reads can occur. The first is caused by ACE instructions and I call it an ACE read. The second is caused by dynamically dead or trivial instructions and I call it an un-ACE read. Whether to
convert ACE lifetime to un-ACE lifetime depends on the order un-ACE reads take place among all the reads after a write. I identified three cases based on the order of un-ACE read operations. First, as Figure 2-1 (A) shows, un-ACE reads (marked as read*) occur closely after the write activity. The second case is shown in Figure 2-1 (B). There are one or more ACE reads before and after un-ACE reads. In the third case, as illustrated by Figure 2-1 (C), there are no more ACE reads but another write or evict follows un-ACE reads. In Figure 2-1 (A) and (B), un-ACE reads can not be converted from ACE time into un-ACE time since they are followed by ACE reads which can not bear any soft error. In Figure 2-1 (C) if I follow the methodology applied in address-based structure AVF computation [4], the lifetime component between the last ACE read and the last un-ACE read should be identified as ACE, however, I can convert it into un-ACE since there are no more ACE reads following un-ACE reads. To calculate the un-ACE lifetime, I identify the last ACE read after the write and then attribute the remaining lifetime between it and the next write or evict to un-ACE.

Similar to the data cache array, edge effects also arise in register files’ AVF computation. For example, if the simulation ends at a point after a write to the register files completes, I can not determine whether the period between that write and the ending point is ACE lifetime or un-ACE lifetime. Therefore, I have to count the above period as unknown. Since COOLDOWN mechanism [4] has a remarkable impact on reducing the unknown portion of a data cache array’s AVF, I applied the COOLDOWN strategy to compute register files’ AVF.

The granularity at which I maintain the lifetime information can have a significant impact on register files’ AVF. I can’t set the granularity to 64-bit; since not all of the instructions defined in the Alpha Instruction set [8, 9, 16] consume the whole 64-bit data word (a register in Alpha 21264 processor is 64-bit). They read or write 32-bit data occasionally, which means the
other 32-bits in that register are idle at that time. The methodology used to compute the AVF of address-based structures [4] partitions a tag-based structure into two parts: data array and tag array. Since register files are specified by their identifiers instead of tag array, there is no false positive or false negative match in the register files’ AVF computation. Because the case in which read data from register files is less than 32-bit occurs infrequently, it is unnecessary to perform per-byte analysis, or even more detailed per-bit analysis such as the one I used to analyze the logical masking effect. In this study, I maintain granularity for register files’ AVF analysis as 32-bit.

**ROB AVF computation**

In an out-of-order execution microprocessor, the reorder buffer (ROB) stores all uncommitted instructions. To effectively exploit ILP, modern processors use large ROB, implying its AVF can greatly affect the AVF of the entire chip. I have developed an ROB AVF model for the Sim-SODA framework.

Data in each ROB entry is allocated for an on-the-fly instruction. A ROB entry includes instruction number, register specifiers and operands. If an instruction is un-ACE, program output will not be affected, so bits in that entry are un-ACE bits. If an instruction is a trivial instruction, some bits in that entry are un-ACE. In the Sim-SODA framework, I use an instruction analysis window and trivial instruction classification to identify these scenarios in the ROB.

**Victim buffer’s AVF computation**

The Sim-SODA framework models the AVF of a victim buffer. Whenever there is a cache miss in the L1 cache, the replaced block will be evicted to the victim buffer. Since the evicted block may be recalled by the program again, any single bit error in it can cause incorrect program output. The victim buffer is also an address-based structure, and only has “fill”, “read”, “evict” and “end” activities. I classified no-overlapping ACE, un-ACE and unknown lifetime
components on it. For example, fill-to-read, read-to-read are ACE; read-to-evict, fill-to-evict, evict-to-fill, evict-to-end are un-ACE; and fill-to-end, read-to-end are unknown. I used COOLDOWN and hamming distance one mechanisms introduced in [4] to accurately compute AVF.

The above AVF models can be integrated into a range of architectural simulators to provide reliability estimates. To implement these AVF models into a unified, timing accurate framework, I have instrumented the Sim-Alpha architectural simulator. I chose Sim-Alpha because previous work [11, 12] has shown that Sim-Alpha can accurately model an Alpha 21264 processor, and in [11, 12], the authors showed that Sim-Alpha is much more accurate than Simplescalar for modeling real hardware. I have extended the simulator with a post-commit instruction analysis window (with a size of 40,000 instructions) which supports the identification of dynamically dead and trivial instructions. The Sim-SODA framework also includes AVF models for cache, TLB, and load/store queue. I used synthesized micro-benchmarks with known characteristics to validate the Sim-SODA framework. The dynamically dead and trivial instructions and the ACE and un-ACE time reported by Sim-SODA match my expectation in the micro-benchmarks.

**Experimental Setup and Results**

**Experimental Setup**

Using the Sim-SODA framework, I performed a detailed reliability analysis of an Alpha-21264-like microprocessor running a wide range of applications. Table 2-3 summarizes the simulated machine configuration. The workloads I used in this study include 12 programs from SPEC 2000 INT and 6 programs from BioInfoMark [20]. I didn’t include SPEC 2000 FP benchmarks because Sim-Alpha does not model floating point pipeline execution accurately [11]. To reduce the simulation time while still maintaining representative program behavior, I
obtained the number of instructions to skip using SimPoint analysis [26] and run each SimPoint for 50 million instructions. Table 2-4 lists the skipped instructions and the input data set for each benchmark. The numbers I present in this dissertation are results for the first SimPoint of each benchmark. I abbreviate the input names as follows: bzip2-source is bzip2-s, gcc-166 is gcc-1, eon-rushmeier is eon-r, gzip-graphic is gzip-g, parser-dict is parser-d, perlbmlk-splitmail is perlbmlk-s, vpr-route is vpr-r, clustalw-ureaplasm is clustalw-u, dnapenny-ribosomal is dnapenny-r, glimmer-bacteria is glimmer-b, hmmr-SWISS-PORT is hmmr-S, predator-eukaryote is predator-e, promlk-17 species is promlk-1.

Experimental Results

Program vulnerability profile at instruction level

Figure 2-2 shows an instruction level vulnerability profile of the studied benchmarks. On average, 69% and 73% of the committed instructions are ACE instructions for SPEC 2000 integer and BioInfoMark suites respectively. The un-ACE instructions include NOPs, prefetch and dynamically dead instructions. As suggested in [22], dynamically dead instructions can be classified into two types: (1) first-level dynamically dead (FDD) if their computation results are simply not read by any other instructions, or (2) transitively dynamically dead (TDD) if their results are only consumed by FDD or other TDD instructions. The “Unknown” refers to those instructions whose destination registers’ lifetimes can not be determined by the instruction analysis window.

As shown in Figure 2-2, NOPs and FDD instructions (FDD_reg and FDD_mem) dominate un-ACE instructions In this study, I found 10% NOPs in the SPEC2000 integer suite. This is the same as the NOPs fraction Fahs et al [14] reported in SPEC2000 integer using the Alpha instruction set. TDD instructions (TDD_reg and TDD_mem) contribute a negligible fraction (e.g. less than 1%) of un-ACE instructions. Similar to the results reported in [22], my study shows that
the fraction of FDD_reg (11% in SPEC and 9% in BioInfoMark) instructions is normally higher than that of FDD_mem (7% in both benchmark suites) instructions. The fraction of TDD and FDD instructions reported by Sim-SODA framework is 17% in SPEC2000 suite and it is close to that reported in [14] (14% FDD and TDD instructions tracked via register and memory).

**AVF of major microarchitecture structures**

Figure 2-3 shows the AVF of the instruction window. I further decompose ACE bits stored in the instruction window based on their instruction types. As can be seen, the dominant portion of ACE bits in the instruction window comes from ACE instructions. For prefetch and NOP instructions, only instruction opcodes are ACE bits [22]. In this work, I count all the opcode and destination register specifier bits of FDD and TDD instructions as ACE bits; all other instruction bits are un-ACE bits [22].

Figure 2-3 shows that the instruction window’s AVF ranges from 26% (gzip) to 62% (gap) in SPEC 2000 and from 44% (dnapenny) to 84% (clustalw) in BioInfoMark respectively. On average, the AVF of the instruction window is 42% and 52% in SPEC 2000 and BioInfoMark.

Biological multiple sequence alignment benchmark clustalw has the highest AVF. This is because clustalw has the highest ACE instruction fraction (e.g. 90% as shown in Figure 2-2). Instruction residency time in the instruction window also affects the AVF result, so the benchmark promlk yields a higher AVF than hmer even though its ACE instruction fraction is lower than hmer (58% vs. 85% respectively).

Figure 2-4 shows the AVF of the instruction window, the wake-up table and the aggregated results (i.e. considering the instruction window and the wake-up table as a single structure). I can see the AVF of the wake-up table is much lower than that of the instruction window. This is because an instruction’s ACE time in the wake-up table is always shorter than the instruction’s residency time in the instruction window. An instruction may still need to wait
for a function unit by staying in the instruction window after all of its source operands are ready. As shown in Figure 2-4, the aggregated results do not reduce significantly (4%-10% in SPEC 2000 and 6%-9% in BioInfoMark). This is because the number of bits contained in the wake-up table is less than that in the instruction window.

Figure 2-5 shows the AVF of the reorder buffer. Interestingly, the ROB’s AVF is significantly lower than that of the instruction window. This is due to the following effect. The Alpha-21264 processor has separate integer and floating point instruction windows. The integer instruction window has 20 entries. The ROB is used to hold all types of instructions and the size of the ROB is 80 entries. Because of the lack of floating point operations, the fraction of idle bits in the ROB is much higher than that in the instruction window.

Sim-SODA models both the high and low 32 bit of the physical registers. In this dissertation, I report the average AVF of the entire 64-bit registers. As shown in Figure 2-6, hybrid AVF computation can reduce register files’ AVF on many workloads (e.g. 9% on crafty, 10% on promlk and 13% on predator). On average, hybrid AVF calculation reduces register files’ AVF by 4% and 5% on SPEC 2000 and BioInfoMark respectively.

I assume each function unit has about 50% control latches and 50% datapath latches, and the datapath within it has a width of 64 bits. The AVF numbers shown in Figure 2-7 are the average statistics of all four function units. I apply trivial instruction analysis to each function unit to further attribute un-ACE bits to different instructions. The semantics of trivial instructions implies that at least one input value to the function unit can be un-ACE. There are other instructions that only produce 32-bit outputs. In that case, the upper 32-bits in the output data path become idle. As is shown, function unit AVF bits are mainly caused by the operands of ACE instructions as well as the output of those instructions. Because the majority of instructions
have two input operands and one output operand, the input bits contribute more ACE bits in the function units than the output bits do.

The level 1 data cache, data TLB, victim buffer and load/store queues are address-based structures. I applied lifetime analysis to both tag and data arrays of these structures and classified lifetime into ACE, un-ACE and unknown components. The Sim-SODA framework uses bit level analysis for tag array and byte level analysis for data array. I implemented the COOLDOWN mechanism to reduce the unknown fraction since edge effect can be significant in these structures [4]. To avoid false positive and false negative matches in the tag array, I have also implemented the hamming-distance-one analysis method [4] in Sim-SODA.

Figure 2-8 and 2-9 show the data array and the tag array AVF for L1 data cache (DL1), data TLB (DTLB), victim buffer (VBuf), load queue (LQ) and store queue (SQ). As can be seen, the L1 data cache tag array’s AVF is higher than the data array’s AVF. This is because the L1 data cache in the Alpha 21264 [17, 18] is a write-back cache and the cache tag must be correct at eviction time. Therefore, all bits of the tag are ACE from the time that there is any write activity that occurs in that entry until it is evicted. The same scenario happens in load and store queues. In contrast, the victim buffer tag array’s AVF is lower than the data array’s AVF. This is because it is a write-through cache and the ACE time in the tag array is much lower.
### Table 2-1. A comparison of different architectural level reliability analysis tools

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Methodology</td>
<td>AVF</td>
<td>Statistic fault injection</td>
<td>Probabilistic model of error generation and propagation</td>
<td>AVF, AVF for address-based structures and hybrid AVF computing</td>
</tr>
<tr>
<td>Hardware structures modeled</td>
<td>Issue queue, function unit, Data cache and TLB, store buffer</td>
<td>Pipeline and its control states</td>
<td>Instruction buffer, decode unit, register file, functional unit, TLB, issue queue</td>
<td>Issue queue, register file, function unit, cache, TLB, ROB, load/store queue, victim buffer</td>
</tr>
<tr>
<td>Baseline models and availability</td>
<td>Asim, Intel’s proprietary tool for modeling Itanium 2-like processor</td>
<td>Verilog model of an Alpha processor</td>
<td>Turandot, available on request</td>
<td>Sim-Alpha, publicly available</td>
</tr>
<tr>
<td>Comment</td>
<td>Complex hardware such as issue queue is modeled as bulk structure</td>
<td>A subset of Alpha ISA is modeled. Caches are not modeled. RTL model is not usually available at early design stage</td>
<td>Memory hierarchy is not modeled. Complex hardware such as issue queue is modeled as bulk structure</td>
<td>Fine-grained AVF models for complex structures. Covers more hardware structures</td>
</tr>
</tbody>
</table>

### Table 2-2. Trivial instructions (in Alpha ISA) identified by Sim-SODA

<table>
<thead>
<tr>
<th>Type</th>
<th>Operation</th>
<th>Triviality Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>MULL/MULQ/MULH: A * B</td>
<td>A=0 or B=0</td>
</tr>
<tr>
<td></td>
<td>AND: A &amp; B</td>
<td>A=0 or B=0</td>
</tr>
<tr>
<td></td>
<td>BIS: A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>BIC: A &amp; ~ B</td>
<td>A=0 or B=1</td>
</tr>
<tr>
<td></td>
<td>ORNOT: A</td>
<td>~ B</td>
</tr>
<tr>
<td>II</td>
<td>MULLI/MULQI/MULH/: A * IMM</td>
<td>A=0 or IMM=0</td>
</tr>
<tr>
<td></td>
<td>ANDI: A &amp; IMM</td>
<td>A=0 or IMM=0</td>
</tr>
<tr>
<td></td>
<td>BISI: A</td>
<td>IMM</td>
</tr>
<tr>
<td></td>
<td>BICI: A &amp; ~ IMM</td>
<td>A=0 or IMM=1</td>
</tr>
<tr>
<td></td>
<td>ORNOTI: A</td>
<td>~ IMM</td>
</tr>
<tr>
<td>III</td>
<td>XOR: A ^ B</td>
<td>A=B</td>
</tr>
<tr>
<td></td>
<td>EQV: A ^ ~B</td>
<td>A=B</td>
</tr>
</tbody>
</table>
Table 2-3. Simulated machine configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline depth</td>
<td>7</td>
</tr>
<tr>
<td>Integer ALUs/multi</td>
<td>4/4</td>
</tr>
<tr>
<td>Integer ALU/multi latency</td>
<td>1/7</td>
</tr>
<tr>
<td>Fetch/slot/map/issue/commit width</td>
<td>4/4/4/4/11 instructions per cycle</td>
</tr>
<tr>
<td>Issue queue size</td>
<td>20</td>
</tr>
<tr>
<td>Reorder buffer size</td>
<td>80</td>
</tr>
<tr>
<td>Register file size</td>
<td>80</td>
</tr>
<tr>
<td>Load/store queue size</td>
<td>32</td>
</tr>
<tr>
<td>Branch predictor</td>
<td>Hybrid, 4K global + 2-level 1K local+ 4K choice</td>
</tr>
<tr>
<td>Return address stack</td>
<td>32-entry</td>
</tr>
<tr>
<td>Branch misprediction penalty</td>
<td>7 cycles</td>
</tr>
<tr>
<td>L1 instruction cache</td>
<td>64KB instruction/64KB data, 2-way, 64B line, 1-cycle latency</td>
</tr>
<tr>
<td>L1 data cache</td>
<td>64KB instruction/64KB data, 2-way, 64B line, 3-cycle latency</td>
</tr>
<tr>
<td>L2 cache</td>
<td>2048KB, direct mapped, 64B line, 7-cycle latency</td>
</tr>
<tr>
<td>TLB size</td>
<td>128-entry ITLB/128-entry DTLB, fully-associative</td>
</tr>
<tr>
<td>MSHR entries</td>
<td>8/cache</td>
</tr>
<tr>
<td>Prefetch MSHR</td>
<td>entries 2/cache</td>
</tr>
<tr>
<td>Victim buffer</td>
<td>8 entries, 1-cycle hit latency</td>
</tr>
</tbody>
</table>

Table 2-4. SPEC 2000 INT and BioInfoMark benchmarks (The data set name is integrated with the benchmark name)

<table>
<thead>
<tr>
<th>SPEC 2000 INT</th>
<th>Instructions Fast Forwarded</th>
<th>BioInfoMark</th>
<th>Instruction Fast Forwarded</th>
</tr>
</thead>
<tbody>
<tr>
<td>bzip2-s</td>
<td>64 M</td>
<td>clustalw-u</td>
<td>20,400 M</td>
</tr>
<tr>
<td>gcc-1</td>
<td>30 M</td>
<td>dnapenny-r</td>
<td>140 M</td>
</tr>
<tr>
<td>crafty</td>
<td>123 M</td>
<td>glimmer-b</td>
<td>20 M</td>
</tr>
<tr>
<td>eon-r</td>
<td>216 M</td>
<td>hmmmer-S</td>
<td>27,200 M</td>
</tr>
<tr>
<td>gap</td>
<td>88 M</td>
<td>predator-e</td>
<td>25,900 M</td>
</tr>
<tr>
<td>gzip-g</td>
<td>1 M</td>
<td>promlk-1</td>
<td>320 M</td>
</tr>
<tr>
<td>mcf</td>
<td>143 M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parser-d</td>
<td>1,771 M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>perlblmk-s</td>
<td>1 M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>twolf</td>
<td>312 M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vortex-3</td>
<td>47 M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vpr-r</td>
<td>3 M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 2-1. ACE and un-ACE lifetime partitions due to the different orders of un-ACE reads (marked as read*) and ACE reads.

Figure 2-2. Instruction level vulnerability profile of the studied benchmarks

Figure 2-3. AVF of instruction window
Figure 2-4. AVF of instruction window and wake-up table

Figure 2-5. AVF of ROB

Figure 2-6. AVF of register files
Figure 2-7. AVF of function unit

Figure 2-8. Data array AVF of L1 data cache (DL1), data TLB (DTLB), victim buffer (VBuf), load queue (LQ) and store queue (SQ)

Figure 2-9. Tag array AVF of L1 data cache (DL1), data TLB (DTLB), victim buffer (VBuf), load queue (LQ) and store queue (SQ)
CHAPTER 3
SOFT ERROR VULNERABILITY PHASE CHARACTERIZATION

Experimental results from Sim-SODA show that the microarchitecture components manifest significant time varying behavior in their run-time reliability characterization. To identify appropriate reliability estimators, I used various performance metrics and examined their correlations with vulnerability factors at the microarchitecture level. I found that in general, using a simple performance metric is insufficient to predict program vulnerability behavior. I then explored code structure based and run-time event based phase analysis techniques and compared their reliability phase prediction abilities.

For my reliability-oriented phase characterization experiments, I simulated an Alpha-21264-like 4-way dynamically scheduled microprocessor with a two level cache hierarchy. The baseline microarchitecture model is detailed in Table 2-3. I used 10 SPEC2000 integer benchmarks and obtained AVF and performance characteristics using Sim-SODA. To perform this study within a reasonable amount of time, I used the MinneSPEC [59] reduced input datasets. However, bzip2 and gzip still require a significant amount of time to finish simulation. Therefore, I could not include their results in this dissertation draft. In order to collect enough interval information for accurate AVF phase behavior classification, I chose each benchmark’s interval size based on the total dynamic instructions executed. For instance, gcc and parser have 100,000 instructions in each interval, while perlbmk has an interval size of 200,000 instructions.

Characterizing Run-Time Microarchitecture Vulnerability to Soft Errors

Time Varying Behavior of Microarchitecture Vulnerability

Figure 3-1 shows the run-time instruction window AVF profile on the benchmarks gcc and parser. Each point of the plots represents the AVF of an interval with a size of 100,000 instructions. (I don’t show all the 10 benchmarks’ AVF profiles because of space limitations). As
can be seen, during program execution, the instruction window’s vulnerability shows significant variation between intervals, and some intervals also show similar AVF behavior regardless of temporal adjacency. Table 3-1 lists AVF statistics in terms of mean and Coefficient of Variation (COV) for all studied structures and all simulated benchmarks. The COV has been widely used in evaluation of phase analysis techniques. It is the standard deviation divided by the mean. In other words, COV measures standard deviation as a percentage of the mean. Table 3-1 shows that although both the instruction window and the ROB are used to support out-of-order execution, they show significant discrepancy in their AVFs: the ROB’s AVF is lower than the instruction window’s AVF on a majority of studied benchmarks. This can be explained as follows: The Alpha 21264 processor has separate integer and floating point instruction windows [56]. The integer instruction window has 20 entries. The ROB is used to hold all types of instructions and the size of the ROB is 80 entries. Due to the lack of floating point operations, the fraction of idle bits in the ROB is much higher than that in the instruction window. Table 3-1 also shows that the COVs of the run-time AVFs on mcf, gap, and vortex are much higher than those on the remaining benchmarks. This indicates that these programs contain complex vulnerability phases which may be challenging for accurate phase classification.

How Correlated Is AVF to Performance Metrics

If a hardware structure’s AVF shows strong correlations with a simple performance metric, its AVF can be predicted using that performance metric across different benchmarks. To determine how correlated vulnerability is to performance metrics, I calculated the correlation coefficients between the run-time hardware structure’s AVF and each of the simple performance metrics using statistics that were gathered from each interval during program execution. I chose widely used performance metrics such as IPC, branch prediction rate, L1 data and instruction miss rates and L2 cache miss rate.
Figure 3-2 shows there are fuzzy correlations between the AVFs and the different performance metrics. For example, for the instruction window, IPC shows strong correlation with AVF on the benchmarks *perlbmk, vortex, mcf* and *crafty*, while yielding near zero correlation coefficients on *twolf* and *eon*. Intuitively, high IPC reduces the ACE bits residency time in a microarchitecture structure, resulting in a reduction of the AVF. On the other hand, the high ILP (usually manifested by high IPC) in a program can cause the microprocessor to aggressively bring more instructions into the pipeline, increasing the total number of ACE bits in a microarchitecture structure. Interestingly, I observed that the same performance metric (e.g. IPC) can exhibit both positive and negative correlations with the AVFs of different structures running on the same benchmark. For example, on *perlbmk*, the correlation coefficient between IPC and the ROB’s AVF is -0.98, while the correlation coefficients between IPC and other structure’s AVFs are all around 0.99. As mentioned in Chapter 2.2, ACE bits residency time plays an important role in determining AVF. For the same instruction, its residency time in different structures can vary significantly. For example, when a processor executes a low ILP code segment, few instructions can graduate. This can increase the ROB’s AVF if those instructions contain a significant amount of ACE bits. Note that the low IPC doesn’t necessarily mean a long residency time of instructions in the instruction window, the functional units and the wakeup table. The instructions may have been completed a long time before their final graduation from the ROB because of the out-of-order execution and in-order commitment. Overall, the results shown in Figure 3-2 suggest that I simply cannot use one performance metric to indicate hardware reliability behavior.

**Program Reliability Phase Classification**

Various phase detection techniques have been proposed in the past. In [26, 37, 41, 60, 62], phases are classified by examining the program’s control flow behavior via the features of basic
block, working set, program counter, call graphs and control branch counters. This allows phase tracking to be independent of the underlying architecture configuration. In [39, 40], performance characteristics of the applications are used to determine phases. These methods detect phase changes without using knowledge of program code structure; however, the identified phases may not be consistent across different architectures. There are also previous studies on comparing and evaluating different phase detection techniques. In [38], Dhodapkar and Smith showed that basic block vectors perform better than instruction working set and control branch counters in tracking performance oriented phases. The studies reported in [41, 43] revealed the correlations between application’s phase behavior with code signatures. Isci and Martonosi recently conducted a study [40] where they compared different phase classification techniques in tracking run-time power related phases. Although the above methods have been used in performance and power characterization, their effectiveness in classifying program reliability-oriented phases remains unknown. In this section, I examine program vulnerability phase detection using two popular techniques: the basic block vector based and the performance event counter based schemes. A complete comparison of all of the phase analysis techniques exceeds the scope of this dissertation.

**Basic Block Vectors vs. Performance Monitoring Counters**

A basic block is a single-entry, single-exit section of code with no internal control flow. Therefore, sequences of basic blocks can be used to represent control flow behavior of applications. In [26], Sherwood and Calder proposed the use of Basic Block Vectors (BBV) as a metric to capture a program’s phase behavior. They first determine all of the unique basic blocks in a program, create a BBV to represent the number of times each unique basic block is executed in an interval, and then weight each basic block so that instructions are represented equally regardless of the size of the basic block. In this dissertation, I quantify the effectiveness of basic
block vectors in capturing AVF phase behavior across several different microarchitecture structures.

In [39], Isci and Martonosi showed that hardware performance monitoring counters (PMC) can be exploited to efficiently capture program power behavior. To examine an approach that uses performance characteristics in AVF phase analysis, I collected a set of performance monitor counters (PMC) to build a PMC vector for each execution interval. I then used the PMC vectors as a metric to classify AVF phases. In my experiments, I instrumented the Sim-SODA framework with a set of 18 performance counters and dumped the statistics of these counters for each execution interval. I then ran an exhaustive search on the 18 counters to determine the 15 counters (shown as PMC-15 in Table 3-2) that were able to characterize the AVF phases most accurately. I chose the size of the counter vector to be 15 because BBV is commonly reduced to 15 dimensions after random projection [26]. To investigate whether the AVF phases are predictable using a small set of counters, I further reduced the PMC vector to 5 dimensions (shown as PMC-5 in Table 3-2). The 5 counters were chosen based on their importance in performance characterization and their generality and availability across different hardware platforms.

The BBVs of each benchmark were generated using the SimPoint tool set [26] and the PMC vectors were dumped for every interval during Sim-SODA simulation. I then ran the k-means clustering algorithm [61] to compare the similarity of all the intervals and to group them into phases. Table 3-3 lists the number of phases identified by the k-means clustering algorithm. In this study, I also examined the use of hybrid information that combined both BBV and PMC vectors to characterize reliability phases. More detailed information about the hybrid schemes can be found in Chapter 3.2.2. As shown in Table 3-3, the BBV generates a slightly larger
number of phases than the PMC-15. The PMC-15 scheme detects more phases than the PMC-5 on a majority of the studied benchmarks.

I use the COV to compare different phase classification methods. After classifying a program’s intervals into phases, I examined each phase and computed the average AVF of all of the intervals within the phase. I then calculated the standard deviation of the AVF for each phase, and I divided the standard deviation by the average to get the COV. I calculate an overall COV metric for all phases by taking the COV of each phase, weighting it by the percentage of execution that the phase accounts for, and then summing up the weighted COV. Better phase classification will exhibit lower COV. In the extreme case, if all of the intervals in the same phase have exactly the same AVF, then the COV will be zero.

Figure 3-3 shows that both BBV and PMC methods can capture program reliability phase behavior on all of the studied hardware structures. As compared to the COV baseline case without phase classification, the BBV based techniques achieve significantly higher accuracies in identifying reliability phases. On the average, the BBV based scheme reduces the COVs of AVF by 6x, 5x, 6x, and 4x on the issue queue, ROB, function unit and wakeup table respectively. Figure 3-3 further shows that the PMC-15 scheme can achieve even higher accuracies in characterizing program reliability behavior. For example, PMC-15 yields lower COVs than BBV on 8 out of the 10 studied benchmarks for all of the examined structures. On benchmarks eon and vpr, BBV performs better than PMC-15 because of the weak correlations between performance metrics and vulnerability shown in Figure 3-2. Overall, the PMC-15 scheme leads to an average COV of 3.5%, 4.5%, 4.3% and 5.7% on the issue queue, reorder buffer, function unit and wakeup table, while BBVs achieve COVs of 4.9%, 5.8%, 5.4% and 6% on the four studied microarchitecture structures respectively.
As I expected, Figure 3-3 shows that the COV of AVF for PMC-5 is higher than that for PMC-15 in most cases. This is because PMC-15 provides additional information that can improve the accuracy in phase clustering. The only exception is vortex shown in Figure 3-3 (a). From Figure 3-2 (a), one can see that on benchmark vortex, several performance metrics (e.g., IPC, DL1miss_rate and L2miss_rate) already exhibit strong correlation with the instruction window’s AVF. When more performance metrics are included, it is possible that additional noise is also introduced into the clustering. However, this situation does not happen frequently as I observed only one exception among all the cases that I analyzed. On the average, reducing the PMC dimensionality from 15 to 5 increases the COV by 1.3%, 0.8%, 1.5% and 2.1% on the 4 studied microarchitecture structures respectively. In practice, gathering information for 5 counters is much easier than collecting statistics for 15 counters. This suggests that the PMC-5 scheme can be used as a good approximation to the more expensive PMC-15 scheme. Compared with the BBV technique, the PMC-5 scheme yields better AVF prediction for the ROB but exhibits worse accuracy for the wakeup table. For the instruction window and function unit, both schemes show similar performance in AVF classification.

In summary, my experiments show that both BBV and PMC phase analysis have significant benefit in characterizing power behavior. I found that in general, the PMC-15 phase analysis performs better than the BBV based approach.

**The Effectiveness of Hybrid Schemes**

I also explored two hybrid phase detection approaches that combine information used by both BBV and PMC schemes. In my first experiment, I used random projection to obtain a BBV with 10 dimensions and then concatenated the BBV with a PMC-5 vector of the same program execution interval to form a hybrid vector, i.e. Hybrid(10+5). In my second experiment, I appended the PMC-5 vector to the original BBV to form a 20 dimension hybrid vector, i.e.
Hybrid (15+5). I then invoked the k-means clustering algorithm on the two hybrid vectors. I compared the effectiveness of hybrid schemes with the default BBV and a BBV with a 20 dimensions of data.

Figure 3-4 shows the COVs on Hybrid(10+5), BBV, Hybrid(15+5) and BBV-20 schemes. As can be seen, simply concatenating BBV and PMC information does not show significant benefit in reliability phase detections. In fact, Hybrid(10+5) show slightly worse performance than BBV on a majority of benchmarks. Similar observation is held when Hybrid(15+5) and BBV-20 are compared. One possible reason is that when the PMC vector and the BBV are randomly merged together, their information can interfere with each other. Such interference can reduce the quality of AVF phase classification. Although I only examine two combinations of PMC and BBV in this dissertation, I think it may be possible to gain some benefit from other combinations that can take the advantages of both schemes while avoiding the interference between each other. Further exploration of effective hybrid phase detection schemes is one aspect of my future work.

**Sensitivity Analysis**

Phase analysis techniques which use architecture dependent characteristics (e.g. PMC) may be sensitive to the machine configuration. To evaluate the robustness of the prior studied phase classification techniques in the context of AVF phase classification, I examine the applicability of using PMC, BBV, and hybrid techniques to detect program reliability phases across different architectures. As my baseline model is a 4-way issue Alpha-21264-like superscalar microprocessor, to generate different machine configurations, I varied the bandwidth in each pipeline stage, modified the latency, size and associativity of the L2 cache, and resized the branch predictor, load store queue and register files. I created two additional configurations to model 8-issue and 16-issue processors. The 8-issue machine has 120 physical registers, a
20KB hybrid branch predictor, a 64-entry load/store queue, and an 8MB 8-way set associative L2 cache. I also increased the branch misprediction penalty to 14 cycles on the 8-issue processor. I used a similar approach to further scale up resources for the 16-issue machine. Using these new configurations, I ran the Sim-SODA framework on all of the benchmarks. The collected data were then processed using the k-means clustering algorithm to obtain the phase classification results (shown in Figure 3-5).

Figure 3-5 shows that the COVs of reliability phase classification yielded on the 8-issue architecture are very close to those on the 16-issue architecture regardless of phase characterization techniques. This indicates that the various phase classification schemes I examined in chapter 3.2.1 and 3.2.2 show similar performance in reliability phase detection across different architecture configurations. Interestingly, I observed that the COV on the 8 and 16-issue machines is higher than that on the 4-issue machine. Intuitively, as the processor issue width increases, programs show larger variation in different characteristics because of the reduced resource constraint. The increased standard deviation can result in an increment in COV. However, if the configuration of the hardware structures (i.e. instruction window, ROB, function unit and wakeup table) is fixed, the impact of the increased issue width will be limited by a threshold. The structures’ AVF will not change significantly when the issue width surpasses that threshold. This explains the noticeable increase in COV when the machine configuration is varied from the 4-issue to the 8-issue and the diminishing variation in COV when the issue width is further increased to 16. Table 3-4 shows AVF phase classification error on different machine configurations. Due to space limitations, I report the average statistics of the 10 studied benchmarks. Compared with COV, AVF errors are slightly changed across different configurations. This is because the positive and negative errors may cancel with each other,
resulting in an overall reduction in that metric. The results shown in Table 3-4 indicate that various techniques are still able to track the aggregated AVF behavior on the more aggressive 8 and 16-issue machines despite of the increased COV in phase classification.
Table 3-1. Variation in run-time microarchitecture vulnerability

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Inst. Window</th>
<th>Reorder Buffer</th>
<th>Function Unit</th>
<th>Wakeup Table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MEAN</td>
<td>COV</td>
<td>MEAN</td>
<td>COV</td>
</tr>
<tr>
<td>mcf</td>
<td>23</td>
<td>86%</td>
<td>23</td>
<td>28%</td>
</tr>
<tr>
<td>eon</td>
<td>37</td>
<td>7%</td>
<td>17</td>
<td>7%</td>
</tr>
<tr>
<td>gcc</td>
<td>27</td>
<td>11%</td>
<td>10</td>
<td>15%</td>
</tr>
<tr>
<td>gap</td>
<td>24</td>
<td>96%</td>
<td>40</td>
<td>43%</td>
</tr>
<tr>
<td>parser</td>
<td>32</td>
<td>16%</td>
<td>13</td>
<td>13%</td>
</tr>
<tr>
<td>perlbmk</td>
<td>29</td>
<td>9%</td>
<td>12</td>
<td>47%</td>
</tr>
<tr>
<td>twolf</td>
<td>40</td>
<td>13%</td>
<td>15</td>
<td>14%</td>
</tr>
<tr>
<td>vortex</td>
<td>26</td>
<td>20%</td>
<td>13</td>
<td>54%</td>
</tr>
<tr>
<td>vpr</td>
<td>40</td>
<td>9%</td>
<td>15</td>
<td>20%</td>
</tr>
<tr>
<td>crafty</td>
<td>29</td>
<td>25%</td>
<td>10</td>
<td>37%</td>
</tr>
<tr>
<td>AVG</td>
<td>31</td>
<td>29%</td>
<td>17</td>
<td>28%</td>
</tr>
</tbody>
</table>

Table 3-2. Events used by PMC-15 and PMC-5 schemes

<table>
<thead>
<tr>
<th>Events</th>
<th>PMC-15</th>
<th>PMC-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Loads</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Number of Stores</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Number of Branches</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Total Instructions</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Total Cycles</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Correctly Predicted Branches</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Pipeline Flushes due to Branch</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Pipeline Flush due to other</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Victim Buffer Misses</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Integer Register File Reads</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Integer Register File Writes</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>L1 Data Cache Misses</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>L1 Instruction Cache Misses</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>L2 Cache Misses</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Data TLB Misses</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Squashed Instructions</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Idle Entry in IQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Idle Entry in ROB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3-3. Number of phases identified by the K-means clustering algorithm

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>BBV</th>
<th>PMC-15</th>
<th>PMC-5</th>
<th>Hybrid(5+10)</th>
<th>Hybrid(5+15)</th>
<th>BBV-20</th>
</tr>
</thead>
<tbody>
<tr>
<td>crafty</td>
<td>35</td>
<td>28</td>
<td>26</td>
<td>33</td>
<td>32</td>
<td>34</td>
</tr>
<tr>
<td>eon</td>
<td>37</td>
<td>25</td>
<td>28</td>
<td>26</td>
<td>34</td>
<td>29</td>
</tr>
<tr>
<td>gap</td>
<td>28</td>
<td>25</td>
<td>23</td>
<td>24</td>
<td>22</td>
<td>24</td>
</tr>
<tr>
<td>gcc</td>
<td>30</td>
<td>26</td>
<td>25</td>
<td>29</td>
<td>28</td>
<td>33</td>
</tr>
<tr>
<td>mcf</td>
<td>25</td>
<td>27</td>
<td>22</td>
<td>22</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>parser</td>
<td>33</td>
<td>26</td>
<td>22</td>
<td>25</td>
<td>30</td>
<td>25</td>
</tr>
<tr>
<td>perlblrk</td>
<td>11</td>
<td>21</td>
<td>20</td>
<td>11</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>twolf</td>
<td>26</td>
<td>29</td>
<td>28</td>
<td>26</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>vortex</td>
<td>31</td>
<td>27</td>
<td>22</td>
<td>26</td>
<td>27</td>
<td>25</td>
</tr>
<tr>
<td>vpr</td>
<td>29</td>
<td>26</td>
<td>22</td>
<td>20</td>
<td>29</td>
<td>30</td>
</tr>
</tbody>
</table>

### Table 3-4. AVF phase classification error on different machine configurations

<table>
<thead>
<tr>
<th></th>
<th>Instruction Window</th>
<th>Reorder Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4 issue</td>
<td>8 issue</td>
</tr>
<tr>
<td>PMC-5</td>
<td>0.70%</td>
<td>1.59%</td>
</tr>
<tr>
<td>PMC-15</td>
<td>2.06%</td>
<td>2.32%</td>
</tr>
<tr>
<td>BBV</td>
<td>3.21%</td>
<td>3.19%</td>
</tr>
<tr>
<td>BBV-20</td>
<td>2.74%</td>
<td>1.57%</td>
</tr>
<tr>
<td>Hybrid(10+5)</td>
<td>1.28%</td>
<td>1.66%</td>
</tr>
<tr>
<td>Hybrid(15+5)</td>
<td>1.48%</td>
<td>1.16%</td>
</tr>
</tbody>
</table>

### Table 3-4. Continued

<table>
<thead>
<tr>
<th></th>
<th>Function Unit</th>
<th>Wakeup Table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4 issue</td>
<td>8 issue</td>
</tr>
<tr>
<td>PMC-5</td>
<td>1.04%</td>
<td>1.38%</td>
</tr>
<tr>
<td>PMC-15</td>
<td>3.85%</td>
<td>2.50%</td>
</tr>
<tr>
<td>BBV</td>
<td>3.67%</td>
<td>3.63%</td>
</tr>
<tr>
<td>BBV-20</td>
<td>3.69%</td>
<td>2.40%</td>
</tr>
<tr>
<td>Hybrid(10+5)</td>
<td>0.83%</td>
<td>1.29%</td>
</tr>
<tr>
<td>Hybrid(15+5)</td>
<td>1.68%</td>
<td>0.88%</td>
</tr>
</tbody>
</table>
Figure 3-1. Run-time instruction window AVF profile on benchmark gcc and parser.
Figure 3-2. The correlations between AVFs and a simple performance metric. A) Instruction windows. B) Reorder buffer. C) Function unit. D) Wakeup table.
Figure 3-3. COVs yielded by different phase classification schemes. A) Instruction windows. B) Reorder buffer. C) Function unit. D) Wakeup table.
Figure 3-4. COVs yielded by hybrid schemes. A) Instruction windows. B) Reorder buffer. C) Function unit. D) Wakeup table.
Figure 3-5. The COVs of reliability phase classification on the 8-issue and 16-issue processors. 
CHAPTER 4
ISSUE QUEUE RELIABILITY OPTIMIZATION

Using a reliability-aware architecture simulator, I characterize soft error vulnerability of several key microarchitecture structures in a SMT processor. Results are shown in the form of the architecture vulnerability factor (AVF) [22] of a hardware structure which estimates the probability that a transient fault in that hardware structure will produce incorrect program results. Details of my simulation framework, machine configuration, evaluated workloads and metrics are presented in Section 4.1.3. Figure 4-1 indicates that among the microarchitecture structures studied, the IQ exhibits the highest vulnerability. This indicates that the IQ is likely to be a reliability hot-spot in SMT processors fabricated using advanced technology nodes. In this chapter, I explore several optimizations to mitigate IQ soft error vulnerability on SMT architectures. I focus my study on the IQ which shows the highest vulnerability, and it is a necessary first step towards protecting entire chip with multiple components to which similar principles might also be applicable.

Vulnerable InStruction Aware (VISA) Issue

Vulnerable InStruction Aware (VISA) Issue

Since ACE instructions are the major contributor of ACE bits, the IQ soft error vulnerability can be mitigated by reducing the residency cycles and quantity of ACE instructions in the IQ. To reduce ACE instruction residency cycles in the IQ, I propose Vulnerable InStruction Aware (VISA) issue that gives the ACE instructions higher priority than the un-ACE instructions. Therefore, once there is a ready ACE instruction, it can bypass all the ready-to-execute un-ACE instructions. If there are several ready ACE instructions, they will be issued in the program order. Note that the ready un-ACE instructions cannot be issued until all the ready-to-execute ACE instructions have been issued. If the number of ready ACE instructions is less
than the number of available issue slots, the ready un-ACE instructions can also be issued in their program order.

Workload IQ utilization characteristics (e.g. the average number ready instructions per cycle and the fraction of ACE instructions in ready instructions) can significantly affect the efficiency of applying VISA issue for IQ vulnerability reduction. For example, if the average number of ready instructions per cycle is smaller than the issue bandwidth, it is unnecessary to use the VISA issue policy since all ready ACE and un-ACE instructions can be issued within the same cycle. If the average fraction of ACE instructions per cycle is small (or there is no ACE instruction in the ready state at all), the benefit of applying VISA issue policy will be marginal.

To answer the above questions, I set up experiments to characterize a SMT processor’s IQ utilization from both performance and reliability perspectives. In each cycle, I count the number of instructions stayed in the ready queue (a collection of IQ entries that holds ready-to-execute instructions) and break them down into ACE and un-ACE instructions. I then plot the histograms of ready queue size with the corresponding ACE instruction percentage. Figure 4-2 shows results for the 4-context CPU workload. The Y-axis on the left represents the probabilistic-based ready queue length distribution and the Y-axis on the right represents the corresponding ACE instruction percentage.

As can be seen, the maximal ready queue length is 73. This indicates that on SMT processors, exploiting TLP increases the number of ready-to-execute instructions in the IQ. Interestingly, a hill is shown in the ready queue length distribution and its peak value is around 26. Moreover, only 10% of execution cycles have a ready queue length less than 9. Since the issue width of the SMT processor is 8, there are abundant ready-to-execute instructions that can be chosen by the issue logic. The ACE instruction percentage plot shows that on average, 60% of
the ready instructions are ACE instructions. The fraction of ACE instructions becomes higher when the ready queue is short. The above observations indicate that there are good opportunities to reduce IQ vulnerability by giving ACE instructions a higher issue priority.

The VISA-based issue policy heavily relies on the capability of identifying the ACE-ness of each instruction in the IQ. However, in general, a retired instruction cannot be classified as ACE or un-ACE until a large amount of its following instructions have graduated. In [22], a post-graduate instruction analysis window with a size of 40,000 instructions is used for vulnerability classification. To perform the just-in-time vulnerability identification at per-instruction level, I propose to perform instruction vulnerability characterization offline and extend ISA (Alpha ISA is applied in my work) to encode the 1-bit ACE-ness tag (e.g. ACE or un-ACE). When an instruction is decoded, its ACE-ness tag will be checked to determine its vulnerability. The offline profiling statically classifies each PC as ACE or un-ACE. A PC is classified as ACE if any of its dynamic instances is identified as an ACE instruction. Instructions executed along the mispredicted paths are not used for this classification. By doing this, I make my classification independent of branch predictor implementation and the non-deterministic inter-thread branch aliasing. The profiling method is conservative since it can predict some finally squashed instructions as ACE. Moreover, the same instruction is not always ACE or un-ACE during the entire program execution. For example, an instruction within a loop may be un-ACE in the first several iterations, but becomes ACE at the last iteration if only the last iteration’s computation result is consumed by other instructions, and vise versa. I refer false-positive to the case that un-ACE instructions are incorrectly identified as ACE, whereas false-negative is the opposite case. As described above, using instruction PC to identify vulnerable instructions can avoid false-negative, namely, no ACE instruction is mispredicted. However, my
method can cause false-positive. Table 4-1 shows the accuracy of using instruction PC to identify ACE instructions from committed instructions across different SPEC CPU2000 benchmarks. As can be seen, the identification accuracy in most applications is around 98% and the average accuracy is 94%. This indicates the false-positive matches happen infrequently. My PC-based ACE-ness classification can incorrectly predict a number of squashed instructions as ACE if their PCs are tagged as ACE, however, the prediction still achieves good accuracy (83% on average) when squashed instructions are considered.

Exploring VISA Issue Based Optimizations

The vulnerability-aware instruction scheduling shows the potential to reduce the residency cycles of vulnerable bits in the IQ. In this section, I further apply dynamic resource allocation to the IQ to prevent excessive vulnerable bits from entering the IQ. Combining with vulnerability-aware instruction scheduling, the proposed schemes achieve significant IQ reliability enhancement by effectively reducing the number of vulnerable bits and their residency in the IQ.

Optimization 1: dynamic IQ resource allocation

Using VISA-based issue reduces ACE instructions residency cycles but it also increases the overall IQ utilization. Because ACE instructions generally exhibit longer data dependence chains than unACE instructions, more ILP can be exploited by issuing ACE instructions earlier. As a result, more instructions in ROB can be dispatched into IQ. From the reliability perspective, as the number of instructions in IQ increases which results in more ACE bits moving to the IQ, the IQ becomes more vulnerable to soft error strikes. If IQ resource allocation can be dynamically controlled to maintain performance while minimizing soft error vulnerability, a better performance/reliability tradeoff can be achieved. In this section, I explore reliability-aware IQ resource allocation to control the quantity of ACE bits that the processor can bring into the
IQ. Incorporated with VISA issue policy, the proposed techniques can effectively mitigate IQ vulnerability to soft error.

When an instruction is in the dispatch stage, the processor checks whether there is a free entry in the IQ. If the IQ is fully occupied, the instruction has to stay in the ROB to wait for an available entry. To reduce IQ vulnerability, I setup a threshold to cap IQ utilization. The instruction dispatch logic compares current IQ utilization with this threshold. If the current IQ utilization is higher than the threshold, the processor will not allocate new IQ entries for instructions even there are idle entries in the IQ. As a result, the processor can not dispatch any instruction to the IQ until the IQ utilization drop below the threshold. To identify the optimal threshold, a large number of off-line simulations need to be performed for each SMT workload. In this dissertation, I propose an on-line mechanism that learns and adapts the IQ utilization threshold dynamically. I sample workload execution characteristics within fixed-size intervals and then use the gathered statistics to setup IQ resource utilization appropriately for the next execution interval. Figure 4-3 shows the dynamic IQ resource allocation algorithm.

In Figure 4-3, RQL is the ready queue length and IQ_SIZE is the total size of the IQ. As can be seen, I adapt IQ resource allocation strategies using workload IPC. This is because IQ utilization can highly correlate with IPC: high-IPC workloads require more IQ entries to exploit ILP. Since processor performance is sensitive to the size of ready queue, I incorporate it in my decision making policies. To give vulnerability reduction a priority, I use static caps which are proportional to the total size of the IQ. Since the maximal commit bandwidth of the studied SMT processor is 8, I partition the IPC into 4 non-overlapping regions (my experimental results show that 4 regions outperform other number of regions) and set up the ratios using the low and high IPCs of each region. Alternatively, these ratios can be dynamically setup using the actual IPC. I
experiment with dynamic ratio setup using linear models that correlates with IPC. Simulation results show that both static and dynamic ratios show similar efficiency. I use static ratios in this dissertation due to their simplicity. The interval size is another important parameter in my design. If it is too large, the technique will not be adaptive enough to the resource requirements. If it is too small, the technique will be too sensitive to the changes of workload behavior. After experimenting with various interval sizes, I choose an interval size of 10K instructions in my design.

**Optimization 2: handling L2 cache misses**

As the number of L2 cache miss increases, the ready queue becomes shorter and there will be more instructions sitting in the waiting queue until the cache misses are solved. Due to the clogged IQ, processors exhibit low IPC performance. After the cache misses are solved, the ready queue size and IPC increases rapidly. However, as shown in Figure 4-3 when the IPC and the ready queue length are both low, the number of allocated IQ entries will be small. This will limit the number of instructions that can be dispatched to the waiting queue. After L2 misses are solved, the number of instructions that become ready-to-execute will be much less than the scenario where no IQ resource control is performed. Therefore, the optimization shown in Figure 4-3 will result in noticeable performance degradation if there are frequent L2 cache misses.

Recall that my goal is to mitigate IQ soft error vulnerability with negligible performance penalty. To achieve this, I propose a L2-cache-miss sensitive IQ resource allocation strategy. As Figure 4-4 shows, when the L2 cache miss frequency is below a threshold ($T_{cache\_miss}$), the dynamic IQ resource allocation mechanism described in Figure 4-3 is used to perform reliability optimization. On the other hand, when the L2 cache miss frequency exceeds the threshold, FLUSH fetch policy [31] is used for vulnerability mitigation. On SMT processors, FLUSH stalls threads that cause L2 cache misses by flushing their instructions from the pipelines. The de-
allocated pipeline resource can be efficiently used by the non-offending threads to boost their performance. Note that the cache miss threshold $T_{\text{cache\_miss}}$ is an important parameter in my design. Using different SMT workload mixes, I performed a sensitivity analysis and choose 16 as the L2 cache miss threshold.

**Experimental Setup**

To evaluate the reliability and performance impact of the proposed techniques, I use a reliability-aware SMT simulation framework developed in [63]. It is built on a heavily modified and extended M-Sim simulator [33] which models a detailed, execution driven simultaneous multithreading processor. I use the computed AVF as a metric to quantify hardware soft error susceptibility. In my experiments, although ACE-ness is classified at instruction-level, the AVF computation is performed at bit-level. Table 4-2 lists microarchitecture parameters of the SMT processor I considered in this study. The ICOUNT [35], which assign the highest priority to the thread that has the fewest in-flight instructions is used as the default fetch policy. In this work, I further examine the efficiency of the proposed IQ reliability optimizations using a set of advanced fetch policies such as STALL [31], DG [42], PDG [42] and FLUSH [31].

The SMT workloads in my experiments are comprised of SPEC CPU 2000 integer and floating point benchmarks. I create a set of SMT workloads with individual thread characteristics ranging from computation intensive to memory access intensive (see Table 4-3). The CPU and MEM workloads consist of programs all from the computation intensive and memory intensive workloads respectively. Half of the programs in a SMT workload with mixed behavior (MIX) are selected from the CPU intensive group and the rest are selected from the MEM intensive group. The results I presented in this dissertation are average statistics of each benchmark category. I use the Simpoint tool [26] to pick the most representative simulation point for each benchmark and each benchmark is fast-forwarded to its representative point before detailed
multithreaded simulation takes place. The simulations are terminated once the total number of simulated instructions 400 million.

**Evaluation**

In this section, I evaluate the efficiency of VISA-based issue and optimizations across various SMT workload mixes. Figure 4-5 presents the average IQ AVF and the throughput IPC yielded on each type of SMT workloads (CPU, MIX and MEM). ICOUNT is used as the default fetch policy. The IQ AVFs and throughput IPCs are normalized to the baseline case without any optimization.

As I expected, by issuing ACE instructions first, the IQ AVF is reduced moderately (5% on average), and throughput IPC is close to the baseline case (1% improvement on average). This is due to the higher IQ utilization. The IQ AVF is further reduced by applying dynamic IQ resource allocation (VISA+opt1). Using CPU workloads as an example, IQ AVF is reduced by about 34% while maintaining the same IPC. This suggests that VISA+opt1 can be used to effectively control the IQ utilization and more aggressively reduce vulnerability without performance penalty on computation intensive workloads. Conversely, on MIX and MEM workloads, VISA+opt1 reduces both throughput IPC and IQ AVF noticeably. This indicates that this scheme is not well suited to handling memory intensive workloads which introduce resource contention more frequently. The results become more promising, however, if I use the number of L2 cache misses to trigger FLUSH. When I apply VISA+opt2, the throughput IPC is improved 1% than the baseline case on the average, with a 48% reduction in IQ AVF. The IQ AVF reduction on MIX and MEM workloads (56%) is higher than that on CPU workloads (33%) because the baseline IQ AVF is lower on CPU workloads which encounter fewer resource clogs that extend ACE instruction residency. On MEM workloads, VISA+opt2 yields slightly lower IPC than the baseline case. This is caused by the FLUSH fetch policy. FLUSH continues to fetch
for at least one thread even if all other threads are stalled and their corresponding pipeline resources have been de-allocated. Thus, the active thread’s instructions will occupy the entire pipeline. On MEM workloads, the performance of the active threads can not be improved much by increasing the pipeline resources due to their inherently low ILPs. Worse, the IQ can be occupied by active threads with even lower IPCs than the stalled threads. Interestingly, the normalized IPC yielded by VISA+opt2 on MIX workloads is higher than that on the baseline case. Due to the mix of computation intensive and memory intensive programs, FLUSH is triggered less frequently. Further, when FLUSH is triggered, the probability that the active threads will be low IPC is less than that on the MEM workloads. If the active threads are computation intensive, the throughput IPC will increases. Therefore, the normalized IPC on MIX workloads shows less predictable behavior.

VISA-based issue and optimizations can be integrated into any SMT fetch policy. Next I show the results achieved by using FLUSH, STALL, DG and PDG as the default fetch policy. Figures 4-6 (a) and (b) show the average IQ AVF and IPC in each workload category when the above fetch policies are used. The results are normalized to the baseline cases of the corresponding fetch policies. As is shown in the figures, even when advanced fetch policies are used, my approaches are still able to provide an impressive IQ AVF reduction of 36% with only a 1% performance penalty on the average. On MIX and MEM workloads, the IQ AVF reduction is less significant using the FLUSH policy than when using the other fetch policies. This is because the FLUSH baseline case is already proficient at handling resource congestion and its IQ AVF is already much lower than the baseline cases of the other fetch policies. On CPU workloads, however, the differences in the advanced fetch policies do not affect the IQ AVF reduction since there are less cache misses and thus the advanced fetch polices have less
opportunity to take effect. When VISA+opt2 is employed, IPC increases on MIX workloads but
decreases on MEM workloads when comparing DG and PDG baseline cases. The trend is similar
to that of using ICOUNT as the default fetch policy (see Figure 4-5 (b)).

**Operand Readiness Based Instruction Dispatch (ORBIT)**

**IQ Soft-Error Mitigation Through Instruction Dispatch**

In a dynamic-issue, out-of-order execution microprocessor, an instruction dispatched from
the reorder buffer (ROB) will stay in the IQ until all of its source operands are ready and the
appropriate functional unit is available. An instruction IQ residency time can be broken down
into cycles during which the instruction is waiting for its source operands and cycles during
which the instruction is ready to execute but is waiting for an available function unit. An
instruction in the IQ can be classified as either a waiting instruction or a ready instruction,
depending on the readiness of its source operands. Both waiting instructions and ready
instructions affect the IQ soft-error susceptibility. Figure 4-7 (a) shows the IQ AVF contributed
by waiting instructions and ready instructions across three types of workloads (shown as Table 4-
3) on the studied SMT processor (shown as Table 4-2). As IQ AVF is determined by the number
of vulnerable instructions per cycle and instruction residency cycles in IQ, Figure 4-7 (b) and (c)
depict the quantity and residency cycles of waiting instructions and ready instructions in the IQ.
Since ACE instructions are the major source of ACE bits, Figure 4-7 (b) and (c) also profile the
number of waiting and ready ACE instructions and their residency cycles.

As Figure 4-7 (a) shows, on an average, waiting instructions contribute to 86% of the total
IQ AVF. Figure 4-7 (b) and (c) help to explain the high AVF contribution from waiting
instructions. As can be seen, waiting instruction residency time in the IQ ranges from 10 to 48
cycles, whereas ready instructions usually spend 1.5 cycles in the IQ on average. This suggests
that an instruction can spend a significant fraction (91% on average) of its IQ residency cycles
waiting for source operands that are being produced by other instructions. Previous studies [33] have also observed that instructions usually spend most of their IQ residency cycles waiting for their operands to be ready. Nevertheless, no attempt has been made to exploit operand readiness in reliability optimizations. Figure 4-7 shows that, at every cycle, the number (61 on average) of waiting instructions also overwhelms that (9 on average) of ready instructions. Especially on MEM workloads that exhibit higher cache miss rates, most instructions are congested in the IQ waiting for ready operands - both the quantity and residency cycles of waiting instructions greatly surpass those of ready instructions. As a result, waiting instructions contribute to 98% of the total IQ AVF. Furthermore, as Figure 4-7 (b) and (c) show, waiting ACE instructions also play a much more important role than ready ACE instructions in determining the IQ AVF due to their higher quantity and longer IQ residency. In short, in order to mitigate IQ AVF, I should focus on the waiting instructions. IQ residency cycles can be minimized if instructions are dispatched from the ROB with ready operands; meanwhile, the number of waiting instructions is also reduced because when instructions are dispatched they are ready-to-execute directly. Therefore, dispatching instructions only when their operands are ready can effectively control both the quantity and residency of waiting instruction in the IQ and reduce IQ AVF significantly. Note that using operand readiness for instruction dispatch does not adversely affect other structures’ (e.g. ROB) AVF. For example, even though the instructions’ dispatch is delayed, their residency time in the ROB is the same since they have to remain in the ROB until graduation. If the ready instructions can be dispatched to the IQ in a timely fashion for execution, the performance impact will be negligible.

To effectively reduce instruction waiting cycles in the IQ, I propose a scheme (DelayALL) which delays the dispatch for instructions with at least one non-ready operand. Since physical
registers are allocated at the register renaming stage, upon instruction dispatch the operands availability of an instruction can be obtained by checking the ready state of the corresponding registers. The DelayALL scheme blocks an instruction dispatch if this check returns at least one non-ready status. Younger instructions can still be dispatched if their source operands are ready and this does not affect the correctness of program execution since instructions are still committed in order. The primary goal of ORBIT is to reduce the instruction waiting cycles in the IQ. Compared with un-ACE instructions, ACE instructions are the major source of ACE-bits since bits in an ACE instruction are ACE while an un-ACE instruction only contains a small portion of ACE-bits (e.g. opcode). Based on this observation, I propose applying operand readiness based instruction dispatch only to ACE instructions (DelayACE). Compared with DelayALL, the DelayACE can achieve better performance since it does not block the dispatch of un-ACE instructions (31% instructions [30] in SPEC CPU2000 workloads are un-ACE) which are not critical to reliability.

**Combine ORBIT with Prediction**

The schemes I propose in Section 4.2.1 can degrade performance since they delay the dispatch of instructions (ACE instructions in DelayACE) until their operands are ready. Alternatively, I can dispatch instructions slightly before their operands are ready. By doing so, instructions will become ready-to-execute soon after entering the IQ. Putting more instructions into the IQ one cycle ahead of time can help improve performance by effectively exploiting ILP. In this subsection, I propose a scheme called PredictALL, which allows instructions with non-ready operands to be dispatched just before the predicted operand ready cycle.

The efficiency of the PredictALL scheme relies on several key design parameters. First, the ready time of instruction source operands should be predicted. Otherwise, instructions can not be dispatched ahead of their ready time. Second, once the ready time is predicted, the scheme
should be able to dispatch instructions ahead of time while minimizing their residency cycles in the IQ. In this dissertation, I opt for dispatching instructions only one cycle before they become ready to execute because this is the required latency to place instructions from the ROB to the IQ. By doing this, instructions will be immediately available for issuing and execution at the time they enter the IQ. Finally, since predicting an operand’s ready time usually involves forecasting the completion time of the instructions which produce the source operands, it is crucial to decide when such a prediction should be made. The prediction of instruction completion time can be made at several pipeline stages, such as fetch, renaming, dispatch and issue. Predictions are less accurate when made earlier since there are more unpredictable variations (e.g. resource conflicts) between the prediction time and the actual completion time. In this study, I predict an operand’s ready time in the dispatch stage. Note that if an instruction’s predicted operand readiness time is longer than the actual readiness time, ORBIT will dispatch the instruction based on its actual readiness time.

Figure 4-8 illustrates the overall architecture of ORBIT with prediction. As can be seen, to implement PredictALL, I need a timer for each physical register to count down the remaining cycles during which a source operand will not be available. The timer decreases its value by one in each cycle and it returns ready-to-dispatch when it reaches one since instructions are dispatched one cycle before they become ready to execute. The timer is initialized when the corresponding register is allocated in renaming stage and will not count down until it is set. When instructions are in the ROB, their dispatch will be blocked if their prediction is not ready, so each ROB entry will add two fields: the source register ID and the register ready bit. There is a dedicated bus between the ROB and the timers, when a timer is reduced to one, it will send the signal to the ROB and write the corresponding ready bits in each ROB entry as ready. When an
instruction is considered for dispatch, the ready bits of the source registers in the ROB entry are checked. The instruction is blocked if either of the ready bits is set as not ready. When an instruction is permitted to dispatch, its predicted completion time will be recorded into the timer associated with its destination register. The timer is set to the sum of dispatch-to-issue delay, issue-to-execute delay and function unit latency (which can be obtained from an instruction’s opcode).

Due to cache misses, accurate prediction of the memory reference instructions’ completion time is challenging. Since store instructions have no data consumers, I do not predict their completion time. Therefore I only need to predict operand readiness for load instructions. The load latency, depending on cache hit/miss at different levels, varies from zero cycles (e.g. a hit in load store queue) to hundreds of cycles (e.g. a L2 cache miss). This information can not be accurately determined until the effective address is calculated in the execution stage and the cache access is performed. In order to obtain the load latency for instructions’ pre-schedule in the IQ, researchers [44, 48] have proposed several complicated predictors to predict the effective address and cache hit/miss at the instruction fetch stage. To implement those predictors, multi-level last value tables, prediction engine and complex operations on the predictors are required. Note that in my proposed techniques, predicting a cache miss is not required since forecasting load instruction completion time can be deferred to one cycle ahead of the dispatch of load dependent instructions. As a result, my prediction techniques do not require the last value table [44] and the cache miss prediction engine [48]. Figure 4-9 illustrates the load instruction completion time prediction in detail. When a load instruction is dispatched, I temporarily predict that its destination register will be ready after a long latency which is equal to the largest number of cycles the timer can be set to. The predicted completion time is updated after the load
instruction’s effective address is calculated and the cache access is performed. Note that bus competition is not taken into consideration for prediction simplicity, and my load instruction complete time prediction does not consider scenarios such as TLB misses and page faults. In these cases, the faulty instruction will be terminated and re-executed after the exception is handled.

In this study, I also examine an alternative design called Predict_non_load, which skips completion time prediction for all load instructions. As a result, instructions that directly depend on loads can not be dispatched to the IQ ahead of time, but prediction resumes for the following dependent instructions (except loads). Figure 4-10 illustrates the difference between Predict_non_load and PredictALL with a code segment example, and the data dependence among instructions is also presented. Note that node 9 performs a store operation and its completion time is not predicted in either of the designs, and other instructions’ readiness is predicted in PredictALL. In Predict_non_load, however, the readiness of instruction 3 and 6 (in grey color) are not forecasted since they are loads. Correspondingly, instruction 4 and 7 (in shade) are not dispatched until they are ready-to-execute due to the direct data dependence on instruction 3 and 6. Additionally, readiness prediction on instruction 5 and 10 are resumed in the Predict_non_load scheme since they are indirectly dependent on loads 3 and 6. Since un-ACE instructions contribute fewer ACE-bits than ACE instructions, I further extend the PredictALL and Predict_non_load to PredictALL_DelayACE and Predict_non_load_DelayACE. Similar to DelayACE, both PredictALL_DelayACE and Predict_non_load_DelayACE don’t apply the operand readiness based dispatch to un-ACE instructions.

To improve the performance of SMT processors, Joseph et al. propose 2OP_BLOCK [33], which blocks instructions with two non-ready operands and their corresponding threads in the
dispatch stage until one of the sources becomes ready. The goal of 2OP_BLOCK is to improve IQ utilization by suspending threads with long latency instructions and allocating more IQ entries to threads exhibiting high ILP. 2OP_BLOCK can reduce IQ AVF since instructions with long waiting cycles are blocked at the dispatch stage. 2OP_BLOCK resumes instruction dispatch once one source operand of the instruction is ready. Therefore, the instruction still spends IQ residency cycles waiting for other source operands. Another difference between 2OP_BLOCK and my techniques is that by using out-of-order dispatch, I block not-ready instructions but don’t stall the corresponding thread. I implement 2OP_BLOCK and examine its efficiency on soft-error vulnerability mitigation. In addition, I explore 2OP_BLOCKACE which applies 2OP_BLOCK to only ACE instructions. Table 4-4 summarizes the eight IQ soft-error vulnerability mitigation schemes that I examine in this dissertation.

Evaluation

The machine configuration is shown in Table 4-2. And the simulation workloads are shown in Table 4-3.

Reliability and performance impact

Figure 4-11 (a)-(c) shows IQ AVF, throughput IPC and harmonic IPC yielded on the proposed techniques. The results are normalized to a baseline case without optimization. As Figure 4-11 (a) shows, on average, the DelayALL scheme reduces IQ AVF by about 86%. On MEM workloads where a high frequency of L2 cache misses cause more long latency instructions, the IQ AVF is reduced by 96%. This suggests that the long residency cycles of these instructions are effectively reduced by ORBIT. Figure 4-11 (b) and (c) show that DelayALL decreases throughput and harmonic IPC by 6% and 10% respectively. DelayACE achieves an 83% IQ AVF reduction which is slightly smaller than that on DelayALL since DelayACE does not block the dispatch of un-ACE instructions and un-ACE instruction residency cycles also
contribute to the IQ AVF as un-ACE instructions still contain a small number of ACE bits. On average, DelayACE results in better performance by showing a 5% reduction in both throughput and harmonic IPC.

PredictALL yields higher performance than both DelayALL and DelayACE. On average, it decreases throughput IPC by 4% and harmonic IPC by 4%. The IQ AVF reduction is 84%, which is smaller than that on DelayALL since statistically each instruction could reside for one more cycle in the IQ. Ideally, PredictALL should have no performance penalty since instructions are always permitted to dispatch one cycle before they are ready. However, the maximum number of dispatched instructions can not exceed the processor dispatch bandwidth. In a scenario that multiple predicted ready to execute instructions compete for limited dispatch bandwidth, dispatch congestion occurs and prevents the dispatch of these instructions on time. This will eventually affect the number of instructions that the processor can issue. The performance penalty of PredictALL is more noticeable on MEM workloads due to burst increased pressure on the dispatch bandwidth caused by L2 misses. Compared with PredictALL, on average across all of the workloads, PredictACE further improves performance by showing 1% throughput IPC and 3% harmonic IPC reduction. The IQ AVF reduction on PredictACE is 79%. Figure 4-11 (b) and (c) show compared to PredictALL, Predict_non_load_DelayALL yields lower throughput and harmonic IPC. In Predict_non_load_DelayALL, the dispatch of direct consumers of load instructions are delayed until their operands are available as the completion time of load instructions is not predicted. As a result, Predict_non_load_DelayALL also shows lower IQ AVF. A similar observation holds when comparing Predict_non_load_DelayACE with PredictACE.
Figure 4-11 also shows the IQ AVF and performance yielded on 2OP_BLOCKALL and 2OP_BLOCKACE. The 2OP_BLOCK techniques increase performance by 2% and reduce IQ AVF by 65%. Note that [33] reports a higher performance improvement (9% and 5% increase in throughput IPC and harmonic IPC) on 2OP_BLOCK on an IQ with a size of 64. The modeled SMT processor in my study has as a 96-entry IQ. As discussed in [33], the benefit of 2OP_BLOCK reduces with increased IQ size. On CPU and MIX workloads, PredictACE and Predict_non_load_delayACE show similar performance characteristics as 2OP_BLOCK while PredictACE and Predict_non_load_delayACE exhibit superior capability in mitigating IQ AVF (e.g. on MIX workloads, IQ AVF is further decreased by 18% on PredictACE and Predict_non_load_delayACE).

A comparison with different fetch policies

To improve performance in modern SMT processors, various fetch policies have been proposed in the past. For example, STALL [31] blocks instruction fetch from offending threads when experiencing a L2 cache miss. As an extension of STALL, FLUSH [31] not only stalls those threads but also squashes instructions from them. DG [42] and PDG [42] respond to cache misses by assigning a lower priority to offending threads. These fetch policies are built on the ICOUNT scheme. In [63], the impact of SMT fetch policies on microarchitecture soft-error vulnerability was analyzed and the study shows that compared with ICOUNT, the advanced fetch policies exhibit a superior capability of soft-error mitigation. In this subsection, I compare the reliability and performance characteristics of the proposed techniques implemented with ICOUNT with those on advanced fetch policies.

Figure 4-12 shows the results on IQ AVF (a), throughput IPC (b) and harmonic IPC (c). Due to space limitations, I show results using the worst case (DelayALL), two best cases (Predict_DelayACE and Predict_no_load_DelayACE), and the average statistics across all 8
schemes. As Figure 4-12 (a) depicts, the advanced fetch polices improve IQ reliability to some extent, especially FLUSH which reduces IQ AVF significantly on MEM workloads due to the frequently triggered flush operation. Compared to advanced policies, ORBIT schemes achieve a greater IQ AVF reduction. Note that all of these fetch policies aim to improve IQ throughput by disallowing instructions to occupy an IQ entry for too long. However, instructions are permitted to dispatch with not ready operands and cycles spent waiting on source operands are unavoidable. Throughput IPC comparison (Figure 4-12.b) shows that FLUSH and STALL can boost the overall performance of SMT execution while my designs (e.g. Predict_DelayACE and Predict_no_load_DelayACE) yield negligible performance degradation. Nevertheless, FLUSH and STALL suffer the fairness problem as shown in Figure 4-12 (c) since they blindly enforce flush/stall operations on all of the instructions from the offending threads and are biased to high ILP threads. Contrary to this, ORBIT techniques only yield a negligible loss in harmonic IPC. Although DG and PDG yield higher performance than ORBIT schemes, the much higher IQ vulnerability reduction gained from ORBIT schemes outweigh the slight performance difference.

**Reliability impact on the entire processor core**

As can be seen, the proposed techniques exhibit strong ability in improving IQ reliability, it is interesting and important to evaluate their impact on the entire processor core and other core structures. Figure 4-13 shows AVF of the processor core and the major microarchitecture structures (such as reorder buffer, register file, load store queue, function units and DTLB) under the techniques across the three types of workloads. Results are normalized to the baseline case without any optimization. Due to the space limitation, I present the results of the worst case (DelayALL) and the best case (Predict_DelayACE) on performance degradation, and the averaged case through all the proposed techniques. As Figure 4-13 shows, on average, my techniques reduce core AVF by 10%. The impact of the techniques on other microarchitecture
structures is trivial except ROB (AVF reduces 22%). Because my techniques delay instructions’
dispatch in each thread, the default fetch policy (ICOUNT) will fetch fewer instructions when
the thread has a number of delayed instructions in the pipeline. As a result, it prevents vulnerable
bits which have long residency cycles from moving into the thread’s private ROB. On the other
hand, ICOUNT brings more vulnerable bits into ROBs of other threads with fewer instructions in
the pipeline, however, their residency time in those ROBs is short. And the overall ROB AVF
still decreases.

Discussion

Recall that my proposed techniques in the dissertation are motivated by Figure 4-7 which
reveals waiting instructions contribute 86% of IQ AVF, and the large quantity and long IQ
residency cycles of waiting instructions are the two factors which result in a highly vulnerable
IQ. In this subsection, I revisit the IQ AVF and two characteristics of waiting instructions with
Predict_DelayACE on the three types of workloads. The results are shown in Figure 4-14 (a)-(c).
Dotted lines represent the reduction obtained from Predict_DelayACE compared with the
baseline case. Characteristics of waiting ACE instructions are also presented in Figure 4-14 (b)
and (c). Due to page limitations, I focus the discussion on Predict_DelayACE, and the other
techniques show similar behavior. As Figure 4-14 shows, both the quantity and residency cycles
of waiting instructions on Predict_DelayACE decrease dramatically. This is also the case on
waiting ACE instructions. On average, the number of waiting instructions per cycle drops from
61 to 8, and instruction residency cycles also reduce from 23 to 2. Therefore, the IQ AVF
reduces significantly. The quantity and residency cycle reductions on waiting instruction show
the efficiency of my techniques in IQ reliability improvement by preventing the dispatch of not-
ready instructions. Note that my techniques do not enforce limitations on ready instructions, the
quantity and residency cycles of ready instructions decrease slightly which is not shown in Figure 4-14.

**Related Work**

In the past, there has been extensive research on instruction scheduling. To avoid the large issue queue size which degrades the clock cycle time, Lebeck et al. [51] proposed the waiting instruction buffer to hold instructions dependent on a long latency operation and reinsert them into a small size IQ when the operation completes. Various forms of dependence-based IQ design were proposed in [52, 53, 54, 55, 57], they generally sorted instructions following data-flow order in the partitioned IQ based on dependence chains, instructions thereby is issued in-order which reduces the complexity of issue logic and improves the clock frequency for large IQ window size. In [58], checkpoint processing and recovery microarchitecture is proposed to implement large instruction window processor without large cycle-critical buffers. Srinivasan et al. [64] found out critical load instructions by building a critical table. Tune et al. [65] and Fields et al. [66] independently proposed the prediction of critical path instructions which provides the opportunity of optimizing processor performance. Seng et al. [67] proposed to issue critical instruction in-order and slot them to fast function units to reduce power consumption. In my work, instruction is prioritized and scheduled based on their reliability criticality.

As power consumption has become an important consideration in processor design, researchers have also studied low power IQ design. Ponomarev et al. [68] proposed to dynamically adjust the IQ size based on the interval sampling of its occupancy. Folegnani et al. [69] partitioned issue queue into blocks, and disabled them while the IPC monitoring mechanism reports they have no contribution to IPC. Jones et al. [70] saved power via software assistance which dynamically resizes IQ based on compiler analysis. In [71], Brooks et al. explored the tradeoffs between several mechanisms for responding to periods of thermal trauma. With
appropriate dynamic thermal management, the CPU can be designed for a much lower maximum power rating with minimal performance impact for typical applications. In [72], a novel issue queue is proposed to exploit the varying dynamic scheduling requirement of basic blocks to lower the power dissipation and complexity of the dynamic issue hardware. In [73], a circuit design of an issue queue is presented for a superscalar processor that leverages transmission gate insertion to provide dynamic low-cost configurability of size and speed. Karkhanis et al. [74] proposed to save energy by delivering instructions just-in-time, it inhibits instruction fetching when there are a certain number of in-flight instructions in the pipeline. Buyuktosunoglu et al. [75] combined fetch gating and issue queue adaptation to match the instruction window size with the application ILP characteristics and achieved energy saving. In my study, the dynamic resource allocation in IQ is dependent on several important reliability-relevant feedback metrics (e.g. RQL) which are not applied in previous studies, and the methodology to implement the resource allocation is different from prior work. Additionally, DVM works on SMT architectures with several novel futures including wq_ratio adaptation, reliability-aware instruction dispatching and on-line AVF estimation.

There is a growing amount of work aimed at characterizing soft error behavior at the microarchitecture level. Walcott et al. [32] used a set of processor metrics to predict structure AVF, which is then applied to trigger redundant multithreading implementation for structure’s reliability maintenance. Soundararajan et al. [76] proposed dispatching stall and partial redundant thread to control the ROB AVF under certain threshold at cycle level. My work is unique in its joint consideration of performance and reliability of IQ design on multithread environment without using redundant execution.
Joseph et al. [33] proposed 2OP_BLOCK to block instructions with 2 non-ready operands and the corresponding thread at dispatch stage to improve the performance, but its impact on reliability is unknown. My work is unique in its joint consideration of performance and reliability of IQ design on SMT architectures. In [77, 78, 79], instructions’ completion time is predicted for instructions’ pre-schedule in instruction window and therefore, improve the performance. To achieve the target, prediction has to be done at an early pipeline stage (e.g. decode and renaming stage), and as a result, these mechanisms rely on implementing complicated prediction tables to maintain the required prediction accuracy. While in my techniques, a much simpler predictor was implemented since I only need to perform prediction during the dispatch stage.
Table 4-1. Accuracy of using PC to identify ACE instructions (committed instruction only)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Accuracy</th>
<th>Benchmark</th>
<th>Accuracy</th>
<th>Benchmark</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>applu</td>
<td>99.8%</td>
<td>galgel</td>
<td>98.8%</td>
<td>mgird</td>
<td>99.9%</td>
</tr>
<tr>
<td>bzip</td>
<td>87.8%</td>
<td>gap</td>
<td>95.9%</td>
<td>perlbmk</td>
<td>99.9%</td>
</tr>
<tr>
<td>crafty</td>
<td>89.4%</td>
<td>gcc</td>
<td>96.5%</td>
<td>swim</td>
<td>99.8%</td>
</tr>
<tr>
<td>eon</td>
<td>87.6%</td>
<td>lucas</td>
<td>99.2%</td>
<td>twolf</td>
<td>95.8%</td>
</tr>
<tr>
<td>equake</td>
<td>99.1%</td>
<td>mcf</td>
<td>96.1%</td>
<td>vpr</td>
<td>81.8%</td>
</tr>
<tr>
<td>facerec</td>
<td>93.7%</td>
<td>mesa</td>
<td>74.9%</td>
<td>wupwise</td>
<td>97.5%</td>
</tr>
<tr>
<td>AVG</td>
<td>93.7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4-2. Simulated machine configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Width</td>
<td>8-wide fetch/issue/commit</td>
</tr>
<tr>
<td>Baseline Fetch Policy</td>
<td>ICOUNT</td>
</tr>
<tr>
<td>Issue Queue</td>
<td>96</td>
</tr>
<tr>
<td>ITLB</td>
<td>128 entries, 4-way, 200 cycle miss</td>
</tr>
<tr>
<td>Branch Predictor</td>
<td>2K entries Gshare, 10-bit global history per thread</td>
</tr>
<tr>
<td>BTB</td>
<td>2K entries, 4-way</td>
</tr>
<tr>
<td>Return Address Stack</td>
<td>32 entries RAS per thread</td>
</tr>
<tr>
<td>L1 Instruction Cache</td>
<td>32K, 2-way, 32 Byte/line, 2 ports, 1 cycle access</td>
</tr>
<tr>
<td>ROB Size</td>
<td>96 entries per thread</td>
</tr>
<tr>
<td>Load/Store Queue</td>
<td>48 entries per thread</td>
</tr>
<tr>
<td>Integer ALU</td>
<td>8 I-ALU, 4 I-MUL/DIV, 4 Load/Store</td>
</tr>
<tr>
<td>FP ALU</td>
<td>8 FP-ALU, 4FP-MUL/DIV/SQRT</td>
</tr>
<tr>
<td>DTLB</td>
<td>256 entries, 4-way, 200 cycle miss</td>
</tr>
<tr>
<td>L1 Data Cache</td>
<td>64KB, 4-way, 64 Byte/line, 2 ports, 1 cycle access</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>unified 2MB, 4-way, 128 Byte/line, 12 cycle access</td>
</tr>
<tr>
<td>Memory Access</td>
<td>64 bit wide, 200 cycles access latency</td>
</tr>
</tbody>
</table>
Table 4-3. The studied SMT workloads

<table>
<thead>
<tr>
<th>Thread Type</th>
<th>Group A</th>
<th>Group B</th>
<th>Group C</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>bzip2, eon, gcc, perlbmk</td>
<td>gap, facerec, crafty, mesa</td>
<td>gcc, perlbmk, facerec, crafty</td>
</tr>
<tr>
<td>MIX</td>
<td>gcc, mcf, vpr, perlbmk</td>
<td>mcf, mesa, crafty, equake</td>
<td>vpr, facerec, swim, gap</td>
</tr>
<tr>
<td>MEM</td>
<td>mcf, equake, vpr, swim</td>
<td>lucas, galgel, mcf, vpr</td>
<td>equake, swim, twolf, galgel</td>
</tr>
</tbody>
</table>

Table 4-4. A summary of the proposed techniques

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DelayALL</td>
<td>Delay dispatch of all the non-ready-to-execute instructions</td>
</tr>
<tr>
<td>DelayACE</td>
<td>Delay dispatch of ACE non-ready-to-execute instructions, ignore un-ACE instructions</td>
</tr>
<tr>
<td>Predict_DelayALL</td>
<td>Delay all the instructions’ dispatch till they are predicted to be ready in the next cycle</td>
</tr>
<tr>
<td>Predict_DelayACE</td>
<td>Delay ACE instructions’ dispatch till they are predicted to be ready in the next cycle, ignore un-ACE instructions</td>
</tr>
<tr>
<td>Predict_non_load_DelayALL</td>
<td>Similar to Predict_DelayALL, but load instructions completion time is not predicted</td>
</tr>
<tr>
<td>Predict_non_load_DelayACE</td>
<td>Similar to Predict_DelayACE, but load instructions completion time is not predicted</td>
</tr>
<tr>
<td>2OP_BLOCKALL</td>
<td>Block all the instructions with two non-ready operands and their corresponding threads until one source is ready</td>
</tr>
<tr>
<td>2OP_BLOCKACE</td>
<td>Block ACE instructions with two non-ready operands and their corresponding threads until one source is ready, ignore un-ACE instructions</td>
</tr>
</tbody>
</table>
Figure 4-1. Microarchitecture soft-error vulnerability profile on SMT processor

Figure 4-2. The histograms of ready queue length and ACE instruction percentage on a 96-entry IQ. SMT processor issue width = 8, the experimented workloads: 4-context CPU (bzip,eon,gcc,perlbmk)

\[
0 \leq IPC \leq 2, \quad IQL = \min \left\{ \frac{RQL + \frac{1}{6} \cdot IQ\_SIZE}{1}, \frac{1}{3} \cdot IQ\_SIZE \right\} \\
2 < IPC \leq 4, \quad IQL = \min \left\{ \frac{RQL + \frac{1}{3} \cdot IQ\_SIZE}{1}, \frac{1}{2} \cdot IQ\_SIZE \right\} \\
4 < IPC \leq 6, \quad IQL = \min \left\{ \frac{RQL + \frac{1}{2} \cdot IQ\_SIZE}{1}, \frac{2}{3} \cdot IQ\_SIZE \right\} \\
6 < IPC \leq 8, \quad IQL = \min \left\{ \frac{RQL + \frac{2}{3} \cdot IQ\_SIZE}{1}, IQ\_SIZE \right\}
\]

Figure 4-3. Dynamic IQ resource allocation based on IPC, ready queue length and total IQ size
$$L_2\_cache\_miss \leq T_{cache\_miss} \begin{cases} 0 \leq IPC \leq 2, \quad IQL = \min \left( RQL + \frac{1}{6} * IQ\_SIZE, \frac{1}{3} * IQ\_SIZE \right) \\ 2 < IPC \leq 4, \quad IQL = \min \left( RQL + \frac{1}{3} * IQ\_SIZE, \frac{1}{2} * IQ\_SIZE \right) \\ 4 < IPC \leq 6, \quad IQL = \min \left( RQL + \frac{1}{2} * IQ\_SIZE, \frac{2}{3} * IQ\_SIZE \right) \\ 6 < IPC \leq 8, \quad IQL = \min \left( RQL + \frac{2}{3} * IQ\_SIZE, IQ\_SIZE \right) \end{cases}$$

$$L_2\_cache\_miss > T_{cache\_miss}, \text{ enable}\_\text{FLUSH}\_\text{policy}$$

Figure 4-4. L2-cache-miss sensitive IQ resource allocation

Figure 4-5. Reliability and performance with ICOUNT fetch policy. A) Normalized IQ AVF. B) Normalized throughput IPC.
Figure 4-6. Reliability and performance using different fetch policies. A) Normalized IQ AVF. B) Normalized IPC
Figure 4-7. A) IQ AVF contributed by waiting instructions and ready instructions. B) Profiles of the quantity of ready instructions and waiting instructions in a 96 entries IQ. C) Residency cycles of ready instructions and waiting instructions in a 96 entries IQ. The ICOUNT fetch policy is used.
Figure 4-8. ORBIT with prediction

Figure 4-9. Predict the load instruction completion time
0: addl r2, r5, 3
1: addl r8, r4, 5
2: subl r9, r8, 1
3: ldq_u r10, 0(r2)
4: mult r16, r2, 10
5: subl r12, r16, r8
6: ldwu r11, 0(r9)
7: addl r15, r11, 6
8: addl r3, r7, 8
9: stb r3, 0(r15)
10: mult r6, r15, 2
11: beq r6, r12, L3
12: subl r20, r21, r26
13: addl r25, r20, 8
14: mult r22, r20, 5

Figure 4-10. Readiness prediction example on PredictALL and Predict_non_load
Figure 4-11. Reliability and performance results when using ICOUNT fetch policy. A) Normalized IQ AVF. B) Normalized throughput IPC. C) Normalized harmonic IPC.
Figure 4-12. A Comparison with different fetch policies. A) IQ AVF. B) Throughput IPC. C) Harmonic IPC
Figure 4-13. The Impact of proposed techniques on core and other microarchitecture structures’ vulnerability (DALL => DelayALL, PDACE => Predict_DelayACE)
Figure 4-14. (A) IQ AVF contributed by waiting instructions and ready instructions. B) Profiles of the quantity of waiting instructions in the 96 entries IQ with Predict_DelayACE. C) Profiles of residency cycles of waiting instructions in the 96 entries IQ with Predict_DelayACE. The ICOUNT fetch policy is used.
The circuit-level SER can be expressed by the following empirical model [80]:

\[
SER \propto F \cdot (A_{d,p} + A_{d,n}) \cdot K \cdot e^{-\frac{Q_{\text{crit}}}{Q_s}}. 
\]  

(5-1)

Where \( F \) is the total neutron flux within the whole energy spectrum, \( A_{d,p} \) and \( A_{d,n} \) are the p-type and n-type drain diffusion areas which are sensitive to particle strikes, \( K \) is a technology-independent fitting parameter, \( Q_{\text{crit}} \) is the critical charge, and \( Q_s \) is the charge collection efficiency of the device. A soft error occurs if the collected charge \( Q \) exceeds the critical charge \( Q_{\text{crit}} \) of a circuit node. For a given technology and circuit node, \( Q_{\text{crit}} \) depends on supply voltage \( V_{DD} \), the effective capacitance of the drain nodes \( C \) and the charge collection waveform. The critical charge \( Q_{\text{crit}} \) of a six transistor SRAM cell is a function (shown as Equation 5-2) of \( V_{DD} \), the threshold voltage \( V_T \) and the effective time constant \( T \) of the collection waveform. In Equation 5-2, the time dependence of current transients is given by \( T_0 \), which depends strongly on the strike location and activated mobility models. Equation 5-1 and 5-2 show that SER increases exponentially with reduction in \( Q_{\text{crit}} \) and \( Q_{\text{crit}} \) is proportional to the effective capacitance of the node and the supply voltage. Hence, the SER is exponentially dependent on \( C \) and \( V_{DD} \).

\[
Q_{\text{crit}} (V_{DD}, T) = C \cdot (V_{DD} + (V_{DD} - V_T) \cdot \frac{T}{T_0})
\]  

(5-2)

Soft Error Robust SRAM (rSRAM)

Equation 5-2 suggests that the minimum amount of charges required to flip the SRAM cell logic state is proportional to the internal node capacitances. Therefore, increasing the effective
capacitances will reduce the SER of a storage node. In [81], the soft error robust SRAM (rSRAM) cell (see Figure 5-1) is built by symmetrically adding two stacked capacitors to a standard six transistor high density SRAM cell. Both area penalty and manufacturing cost of the rSRAM can be mitigated by adding the two capacitors in the vertical dimension (i.e. between the polysilicon and the Metal 1 levels) and manufactured with a standard embedded DRAM process flow. Accelerated alpha and neutron tests have demonstrated that the rSRAM devices are alpha immune and almost insensitive to neutrons [82].

The rSRAM cell symmetry and the transistor sizing remain strictly identical to the standard SRAM. Further comparison between rSRAM and standard SRAM shows that they both have similar power consumption, leakage and area. However, there are trade-offs between robustness and timing performance. Compared with the standard SRAM, both the read current and the static noise margin of rSRAM are unchanged, whereas the intrinsic write operation of the rSRAM is slowed down proportionally to the extra loads on the two internal nodes. The normalized SER rates for the rSRAM as a function of the added capacitor value were studied in [82] using Monte Carlo simulations. For example, to achieve the desired SER rates on rSRAM fabricated using 90nm CMOS technology, a capacitor with value of 12fF needs to be added to each node which degrades the memory cell write timing performance by a factor three in typical conditions. For very high capacitor values, the write might become even slower than the read, leading to significant cycle time penalty. Such disadvantage limits the applicability of using the rSRAM to harden hardware structures that resident in the critical path of the processor pipeline.

**Voltage Scaling for SRAM Soft Error Robustness**

Equation 5-2 shows that $Q_{crit}$ has a linear relation with the supply voltage. Transistors with high supply voltage exhibit strong immunity to soft errors since the particle energy threshold required to cause soft errors is increased. Therefore, scaling up supply voltage can provide
immunity to soft errors. In this dissertation, I used dual-$V_{DD}$ [83], a technique that is originally proposed for power saving, to enhance hardware SER robustness. However, scaling up voltage will increase dynamic and leakage power consumption. For example, dynamic power of the circuit is proportional to the square of the supply voltage. Therefore, it is important to appropriately scale up supply voltages such that the power savings can be balanced with concerns of reliability. This dissertation proposes methods that can selectively adjust the supply voltage and achieve attractive trade-offs between power and reliability.

Differing from circuit level radiation hardening methods, microarchitecture level soft error vulnerability mitigation techniques exploit program characteristics to achieve application-oriented reliability optimization. In general, these techniques can reduce soft error failure rate but does not guarantee convergence to the high reliability design goal.

**Combined Circuit and Microarchitecture Techniques**

**Radiation Hardening IQ Design Using Hybrid Techniques**

As described in Chapter 4, microarchitecture level techniques such as operand readiness based instruction (ORBIT) dispatch can effectively reduce the IQ AVF despite that they provide no protection to soft errors. However, this technique results in performance penalty. This is because instructions are selected for dispatching solely based on their operand readiness instead of their criticality to performance. Note that the increased program runtime will increase processors’ overall transient fault susceptibility since soft errors now have more opportunities to strike the chips. Therefore, microarchitecture soft error mitigation techniques should cause minimal performance overhead. Due to the superior soft error robustness of the rSRAM cell, it can be used to implement IQ, a SRAM based structure. However, the using of rSRAM increases write latency, which implies that an IQ entirely implemented with the rSRAM will suffer noticeable performance degradation.
To leverage the advantage of circuit and microarchitecture level soft error tolerant techniques while overcoming the disadvantage of both, I propose an IQ consists of a part implemented using the standard SRAM cells (NIQ) and a part implemented using the radiation hardened rSRAM technologies (RIQ). The operands ready instructions are dispatched into NIQ while other not-ready but performance critical instructions are dispatched into RIQ. By decreasing both quantity and residency cycles of instructions’ vulnerable bits in a hardware structure, the operand readiness based dispatch can effectively mitigate soft error vulnerability of NIQ where no error protection is provided. The filtering out of performance critical instructions from the delayed dispatch alleviates performance penalty. Meanwhile, the write latency of the rSRAM based RIQ can be efficiently hidden since instructions dispatched to the RIQ normally will not be immediately ready for issuing. The rSRAM technique, which provides great soft error immunity, successfully protects those instructions from soft error strikes during their RIQ residency period. Therefore, compared with methods that exclusively rely on circuit or microarchitecture solution, the hybrid schemes can achieve more desirable trade-offs between reliability and performance.

Figure 5-2 presents the control flow of instruction dispatch in the proposed IQ design that uses hybrid radiation hardening techniques. When instructions in ROB are scheduled for dispatch, the dispatch logic only places ready-to-execute instructions into the NIQ. By doing so, the quantity and residency cycles of instructions in the NIQ are significantly reduced and the corresponding IQ SER decreases. The performance criticality of other not-ready-to-execute instructions is examined and critical instructions are dispatched to the RIQ without delay. Therefore, only non-critical instructions are delayed at the dispatch stage.
In this study, I investigate hybrid schemes that can achieve attractive reliability and performance tradeoffs without significantly increasing the hardware cost. I assume that the NIQ and RIQ have the total size equal to that of the original IQ, and they share the same amount of dispatch bandwidth as in the original design. Figure 5-3 (a) provides an overview of the architecture support for the proposed ideas. The detailed circuit design on supporting RIQ wakeup will be discussed in Section 5.2.2. In order to obtain their operands readiness when the instructions are sitting in the ROB, a multi-banked, multi-ported array is built to record the register files’ readiness state. The bit array is updated during write back stage. The ROB can be logically partitioned into several segments to allow parallel accesses to the multiple banks of the array which hold the same copies of information. A simple AND gate is added in each ROB entry to determine the readiness of a instruction. Note that in my scheme, younger instructions can still be dispatched if their source operands are ready and this does not affect the correctness of program execution since instructions are still committed in order.

Figure 5-3 (b) illustrates how the critical instructions can be identified. In this dissertation, I define the performance critical instructions as branch instructions and the instructions with long dependence chain in ROB. I use critical tables proposed in [64] to quantify an instruction’s criticality. Each thread’s ROB is associated with a critical table and each ROB entry has a corresponding critical table entry to represent the data dependences of other instructions on this instruction. Each critical table entry is a vector having one bit per ROB entry, a certain bit of the vector is set as “1” if its corresponding ROB entry is direct or indirect data dependent on the current ROB entry. The sum of each bit in the $i_{th}$ critical table entry represents the length of the $i_{th}$ instruction’s data dependence chain which, in other words, describes its performance criticality. The critical table is updated at decode and renaming stages. As shown in Figure 5-
3(b), assume that an instruction is placed into the $j_{th}$ ROB entry, and it directly consumes the computation results of instructions in the $m_{th}$ and $n_{th}$ ROB entries. Then the $j_{th}$ column in the critical table is updated as the bitwise OR result of the $m_{th}$ and $n_{th}$ columns, meanwhile, the $m_{th}$ and $n_{th}$ entry of the $j_{th}$ column are also marked as one. Therefore, the bit with “1” in the $j_{th}$ column demonstrates on which instructions the newly inserted instruction is data dependent. As the instruction’s criticality is available in critical table, a criticality threshold is set to classify the instructions into critical instructions and non-critical instructions. Instructions with higher criticality than the threshold are recognized as critical instruction, and vise versa. Branch instructions are always identified as critical. Note that the criticality threshold affects the required RIQ size and correspondingly, the performance and reliability of the proposed techniques. A detailed analysis can be found in Section 5.3.2.

The RIQ Design and Alternative

An conventional IQ entry consists of several fields: 1) payload area (such as the opcode, destination register address, function units type and so on); 2) left and right tags of the two source registers, and each tag is coupled with a CAM (content-addressable memory) for register number comparison; 3) left and right source ready bits, used to record the availability of the source registers; 4) and another ready bit to present the instruction’s readiness, which is the logic AND result of the two source ready bits. When an instruction completes its execution, it destination resister identifier is sent to the tag buses and broadcasted through all IQ entries. The CAM in each IQ entry figures out whether there is a match between the instruction’s source register number and identifier in the tag buses, and the corresponding source ready bit is set to “1” if a match occurs. In the case that both source ready bits are set to “1”, the instruction is ready, and ready bit will raise the issue request signal to the selection logic.
In my hybrid IQ, the wakeup logic of NIQ is identical to that of the conventional IQ. Care must be taken for the RIQ design due to the extra write latency to the rSRAM cells. Figure 5-4 describes the detailed circuit design on each field of the RIQ entry. Since instructions dispatched into the RIQ usually are not ready-to-execute, the latency caused by initial write operations to the RIQ entry can be overlapped with the instructions’ waiting-for-ready period. As a result, the rSRAM is used to build the payload area and tags in each RIQ entry. However, the write latency delays the update of the ready bits and prevents the instructions from being issued on time. In another word, the selection and issue stages of the pipeline will be postponed. To avoid the negative performance impact of the rSRAM, I implement the three ready bits per IQ entry using standard SRAM-based cells and use ECC to protect their integrity. The overhead of ECC is small due to the overall small quantity of the ready bits in the IQ.

Another important design consideration for RIQ entry is the CAM which is composed of storage cell (SRAM) and comparison circuit (XOR gates), the rSRAM techniques can also be used to implement robust CAM without any area penalty. [81] proposed to extend rSRAM technique into CAM (i.e. rCAM). The rCAM has the similar characteristic as rSRAM, namely, it also suffers from the write latency, but read time is unchanged. In this study, I also consider rCAM implementation for a fault-free RIQ. Since the data (source register number) is written to CAM storage cell once the instruction is dispatched into RIQ and stay there until the instruction is issued, the write latency in rCAM is overlapped with that on writing instruction information into the RIQ payload and tags. Therefore, rCAM doesn’t introduce extra performance delay in RIQ. However, it is possible that the instruction misses the register number broadcasting while its information is being written into the rCAM. In order to timely update the instruction’s source
ready bits, as shown in Figure 5-3 (a), the register ready bits array will be checked once the write operation completes.

The using of rCAM can be avoided in wakeup-free version of the RIQ design. In [85, 86], instruction reinsertion and selective replay were proposed for performance enhancement. I propose two designs in Figure 5-5. The first approach (shown as Figure 5-5.a) is to reinsert the instructions from RIQ back to the NIQ when they will be ready soon. The reinserted instructions take the dispatch bandwidth and have higher priority than other normally dispatched instructions. If the NIQ is full, the reinsert operation has to be stalled. Since RIQ is wakeup free, instructions can not be selected and issued directly from the RIQ. In order to fully explore the SER robustness of the RIQ, instructions in it should not be reinserted until they become ready in the following cycle. To satisfy this requirement, prediction on the ready time of the source register has to be taken, which involves complicated hardware designs [86, 87], such as the load address predictor, the cache miss/hit predictor and the timing table which is used to record the predicted ready time. An alternative approach (Figure 5-5.b) to the above design is to directly issue instructions from RIQ when they are predicted to be ready-to-execute. Accurate wakeup prediction is required in the case, which also introduces several complicated predictors mentioned above. Since the predictor can not be 100% accurate, the mis-prediction has to be handled: instructions’ source register ready states have to be checked before they are really issued into function units. If they are not ready, instructions have to be replayed (reinsert into IQ), and the information in the predictor has to be updated also. As can be seen, the two wakeup free designs depend on complicated register ready time prediction which results in additional hardware implementations and operations. In my study, the ECC protected SRAM based ready bits and rCAM are selected.
Using Dual-$V_{DD}$ to Improve ROB Reliability

ROB is another important microarchitecture structure in SMT processors. As introduced in Section 5.1, supplying high $V_{DD}$ to CMOS circuit can improve hardware structure’s raw soft error rate. However, high $V_{DD}$ should be judiciously applied since the dynamic power consumption is quadratic to supply voltage. In this dissertation, I explore using microarchitecture level soft error vulnerability characteristics and runtime events to enable and disable high $V_{DD}$, which can achieve attractive trade-offs between reliability and power. Recall that the overall soft error rate of a microarchitecture structure is determined by FIT rate per bit and AVF at microarchitecture level. In the case that different $V_{DD}$ varies FIT per cycle, Equation 5-3 can be rewritten as:

$$\text{SER} = \frac{\text{FIT}_{\text{normal}} \cdot \sum_{\text{FIT}} \text{#ACE bits per cycle} + \text{FIT}_{\text{enhanced}} \cdot \sum_{\text{FIT}} \text{#ACE bits per cycle}}{\#B \cdot T_{\text{execution cycles}}}$$

where $\text{FIT}_{\text{normal}}$ represents the FIT with normal $V_{DD}$, while $\text{FIT}_{\text{enhanced}}$ represents the FIT with high $V_{DD}$. Correspondingly, $T_{\text{normal FIT}}$ and $T_{\text{enhanced FIT}}$ depict the period of $\text{FIT}_{\text{normal}}$ and $\text{FIT}_{\text{enhanced}}$ respectively. As can be seen from Equation 5-3, when the number of ACE bits in the structure is small during $T_{\text{enhanced FIT}}$, the SER reduction gained via reducing FIT (i.e. increasing $V_{DD}$) is substantially discounted. Take an extreme case for example, when there is no ACE bit, I can not gain any benefit from increasing $V_{DD}$ since all the errors are masked at microarchitecture level. On the other hand, when all the bits in the structure are ACE (e.g. no error can be masked), the benefit can be totally exploited.

In order to effectively improve ROB reliability and control the extra power consumption, I propose to trigger high $V_{DD}$ when ROB shows high vulnerability at microarchitecture level and switch $V_{DD}$ back to normal when the vulnerability drops below a threshold. Due to the circuit-level complexity concerns, I limit my scheme to two supply voltages, and that supply voltage
transition is called dual- \( V_{DD} \) technique [83]. A DC-CD converter can continuously adjust the supply voltage, unfortunately, the converter requires a long time for voltage ramping [88] and it is not suitable for high performance SMT processors. I choose to use two different power supply lines for the quick \( V_{DD} \) switching, and a pair of PMOS transistors is inserted to handle the voltage transition. Li et al. [88] and Usami et al. [89] proved that the energy and area overhead from the two-supply-power-network is negligible. In this dissertation, I make simplistic assumption that varying supply voltage in CMOS doesn’t cause power and performance overhead. The clock frequency maintains the same while dual- \( V_{DD} \) is applied since the transistor can operate with normal frequency when the \( V_{DD} \) switches to high voltage. In [90], Burd et al. showed that CMOS can continuously operate when the voltage switch is limited in a certain amount per nano second. In other words, the voltage transition can not be completed immediately. Therefore, when triggering high \( V_{DD} \), the structure’s high vulnerability period should be long enough to cover the transition cycles. Figure 5-6 shows the relation between L2 miss and ROB AVF over a period of 5000 cycles on benchmark \( vpr \) execution. Note that the right Y-axis just simply describes the occurrence of L2 miss, and “1” represents that L2 miss exists at that cycle. As can be seen, the ROB AVF jumps high when L2 miss occurs, and drops down after it is solved. Because upon an L2 cache miss, the pipeline usually ends up stalling and waiting for data, instructions can fill up the ROB quickly and the congestion will not be solved until L2 cache miss is handled. Furthermore, since high utilization in ROB results in high quantity of vulnerable bits, the ROB AVF usually exhibits a strong correlation to L2 cache miss. In SMT processors, L2 cache miss latency often lasts for hundreds of cycles which can cover the \( V_{DD} \) transition cycles, therefore, L2 cache miss is a good trigger for \( V_{DD} \) switching.
Experimental Setup and Evaluation Results

Experimental Setup

To evaluate the reliability and performance impact of the proposed techniques, I use the same framework used in Chapter 4. In addition, I ported Wattch power model [91] into my simulation framework for power evaluation. The baseline machine configuration is shown in Table 4-2. The IQ is a shared structure while the ROB is private to each thread. I use ICOUNT as the baseline fetch policy. I assume 90nm CMOS technology. In [81], the relation between added capacitor value, write time and SER for 90nm standard rSRAM was studied. The results show that the write time is linearly related to the capacitor value, and SER reduced dramatically while capacitor value increases from 0 fF to 12 fF, however, it varies slightly as the increase of capacitor value when the value is larger than 12 fF. Therefore, in my experiments, I select the capacitor value as 12 fF, correspondingly, the write time in rSRAM is as three times as the standard SRAM. In my experiment, I assume the normal $V_{DD}$ is 1.2 V and high $V_{DD}$ is set as 2.4 V. The enhanced SER_{SRAM} is computed using Eq.1 and 2. I assume the voltage can transit 0.05V/ns, so the transition time lasts for 20 cycles in the simulated 1GHz processor.

The SMT workloads in my experiments are comprised of SPEC CPU 2000 integer and floating point benchmarks. I create a set of SMT workloads with individual thread characteristics ranging from computation intensive to memory access intensive (see Table 5-1). The overall SER capturing vulnerability on both circuit and microarchitecture levels is used as a baseline metric to estimate how susceptible a microarchitecture structure is to soft-error strikes. I use throughput IPC, which qualifies the throughput improvement, and harmonic mean of weighted IPC.
Evaluation

Effectiveness of rSRAM based IQ design

I compare my hybrid scheme with several existing techniques (e.g. 2OP_BLOCK [33] and ORBIT) which exhibit good capability in achieving IQ reliability enhancement. A comparison is also performed with the design that uses rSRAM to implement the entire IQ. Additionally, [63] showed that among the several advanced fetch policies in SMT processors, FLUSH can effectively reduce IQ vulnerability. I also compare my technique with the baseline SMT processors that use FLUSH fetch police. In the hybrid scheme, I set critical threshold as 2 with RIQ size of 24, and the threshold increases as high as the ROB size during L2 miss. A detail sensitivity analysis is presented in Section 5.3.2.

Figure 5-7 (a) - (c) presents the overall IQ soft error rate, throughput IPC and harmonic IPC yielded by various techniques across three SMT workload categories. The results are normalized to the baseline case without any optimization technique. Note that rSRAM-based IQ has zero soft error rate when normalized, its SER is not presented in Figure 5-7 (a). As can be seen in Figure 5-7 (a), on average, my hybrid scheme exhibits strong SER robustness which reduces IQ SER 80% with only 0.3% throughput IPC and 1% harmonic IPC reduction through all the workloads. The IQ SER reduction is more noticeable on MEM workloads, because low IPC workloads have less ready-to-execute instructions and RIQ is fully utilized to protect the ACE bits in those instructions. ORBIT obtains similar IQ SER reduction as my design since they have common property that only ready-to-execute instructions can be dispatched into unprotected IQ. The 2OP_BLOCK scheme, which blocks instructions with 2 non-ready operands and the corresponding thread at dispatch stage but still allows the dispatching of unready instructions to unprotected IQ, gains 20% less SER reduction compared with the hybrid scheme. Moreover, my design outperforms FLUSH policy by 58% in reliability improvement. On the
performance perspective, as Figure 5-7 (b) and (c) show, the hybrid scheme surpasses other techniques on both throughput and fairness performance, and the performance difference is more noticeable in MIX and MEM workloads. As I expected, the rSRAM based IQ suffers significantly performance penalty (20% degradation on both throughput and harmonic IPC), and the performance degradation can be as worse as 35%.

**Sensitivity analysis on criticality threshold and RIQ size**

In SMT environment, a L2 miss can cause congestion in the corresponding thread’s ROB. As a result, the computed instruction criticality using the critical table can easily surpass the pre-set criticality threshold. Nevertheless, most instructions are data dependent on the load miss instruction and can not become ready-to-execute until the L2 cache miss is solved. Their entrance to the RIQ, however, results in RIQ resource congestion and prevents the dispatching of critical instructions from other high performance threads. In my study, in order to avoid the RIQ congestion and improve the overall throughput, each thread is assigned with a pre-set critical threshold and the threshold is adjusted to a high value (for example, equal to the RIQ size) when the thread is handling L2 cache miss.

Both criticality threshold and RIQ size can control the dispatching of instructions into RIQ and affect the effectiveness of my hybrid scheme. In this dissertation, I perform a sensitivity analysis to understand the impact and interaction of these two factors. As can be seen, the two factors interact each other, when criticality threshold is high, a large RIQ is not necessary; on the other hand, a small RIQ requires a high criticality threshold. In my study, I start the analysis from the fixed criticality threshold of two, because instructions with less than two consumers are likely to be dynamically dead instructions whose computation result will not affect the program final output, therefore, they are not performance critical. The fixed criticality threshold is combined with various RIQ size ranging from 8 to 64. By doing this, I can quickly figure out the
optimal RIQ size required to satisfy the lowest criticality threshold. Note that RIQ size cannot be extended to extraordinary large or small, because with the fixed total IQ size, an extra large RIQ size corresponds to an extremely small NIQ size which has difficulty in holding all the ready-to-execute instructions and delays their dispatching. On the other hand, the benefit from dispatching not-ready critical instructions to RIQ is disappeared with an extremely small RIQ. Figure 5-8 (a)-(c) presents the normalized throughput IPC, harmonic IPC and IQ SER to the baseline case on various RIQ sizes. As can be seen, IQ SER reduces as the RIQ size increases, because the unprotected NIQ size is reduced and less vulnerable bits are exposed to soft error strikes. However, the increase of RIQ size results in deleterious performance impact due to the thirst for NIQ to hold ready-to-execute instructions. As shown in Figure 5-8, RIQ size with 24 generates the closest performance to the baseline case in all the three workload categories and it satisfies my target on maintaining application performance while improving IQ reliability. After the RIQ size is fixed at 24 for the lowest criticality threshold, another set of experiments can be performed to search for an appropriate criticality threshold. However, higher criticality threshold requires smaller RIQ which results in higher IQ vulnerability, it is not suitable to my target even though the performance can be improved. In my dissertation, the 24 entry RIQ with pre-set criticality threshold equal to two is used in the experiments.

**Effectiveness of dual-$V_{DD}$ in ROB SER robustness**

In this subsection, the efficiency of applying Dual-$V_{DD}$ for ROB SER enhancement is examined. Black bars in Figure 5-9 (a) and (b) show the reduced ROB SER and the power overhead of the processor core after the proposed technique applied to the three types of workloads. As can be seen, on average, ROB SER reduces 50% by consuming extra 6% core power. And in MEM workloads which encounter a large number of L2 misses, my technique gains 67% ROB SER reduction. In most architecture design, 6% power overhead is larger than
the acceptable boundary; therefore, the using L2 miss as a trigger has to be improved. Notice that the number of vulnerable bits in the ROB is not always positively proportional to the ROB utilization, which suggests that L2 miss does not always imply a large number of ACE bits in the ROB. In this dissertation, I propose an enhanced trigger which takes the quantity of the ACE bits in ROB into account. The trigger performs as follows: when a L2 miss occurs, the number of ACE bits in ROB per cycle is countered and averaged in the following 20 cycles, and the high $V_{DD}$ will not be supplied if there are not enough ACE bits, saying, lower than a vulnerability threshold. After the L2 miss is solved, the $V_{DD}$ is switched back to normal. Since on-line, accurate ACE bits identification is difficult, in my study, I approximate the number of ACE bits at the instruction level. The basic idea is: the longer dependence chain the instruction has, the higher possibility its computation result affects program final output. Consequently, I assume the bits in instructions with high criticality (e.g. criticality > 16) are ACE. The information stored in the critical table can be used for ACE-ness estimation. Note that the pre-defined vulnerability threshold affects both the ROB SER reduction and the power overhead. Care must be taken when choosing a pre-defined vulnerability threshold, as setting this value too high can result in limited ROB reliability improvement, and setting it too low can result in minimal control over the power consumption. In this study, I vary the vulnerability threshold in my experiments dependant upon the size of the ROB (within a range of $1/2$*ROB_size to $5/6$*ROB_size). To evaluate the effectiveness of various thresholds, I propose a metric, SER_reduction/power_overhead, which describes the tradeoff between reliability and power. A higher SER_reduction/power_overhead value indicates a better tradeoff. Figures 5-10 (a) - (c) present the ROB SER reduction, power overhead and SER_reduction/power_overhead across various vulnerability thresholds and three workload categories. As expected, both the ROB SER
reduction and power overhead increase as the threshold decreases because high \( V_{DD} \) is triggered more frequently. However, this is not the case for SER_reduction/power_overhead. When the threshold is set to 64, as shown in Figure 5-10 (a) and (b), SER_reduction/power_overhead attains its maximum value on CPU and MIX workloads. Therefore, a vulnerability threshold of 64 is selected for my study. The white bars in Figure 5-9 present the results yielded by the enhanced trigger, and on average, ROB SER reduces 35% with only a 3.5% power overhead.

**Putting them together**

Figure 5-7 and 5-9 show that both the hybrid radiation hardened IQ and the Dual-Vdd based ROB exhibit strong SER robustness while yielding a negligible performance and power overhead. I also apply the two techniques simultaneously and evaluate their aggregate effect on the entire processor core SER. The impact of the two proposed techniques on the vulnerability of other primary structures, such as register files, load store queue, DTLB and function units, is also examined. The normalized SER results (to the baseline case where no optimization is applied) are shown in Figure 5-11. As can be seen, on average, the core SER substantially decreases by 23% while other structures’ SERs are slightly affected by my techniques. Furthermore, the load store queue vulnerability is also reduced by 15%. I exclude a discussion of the performance penalty and power overhead for the aggregated technique as they have already been discussed in previous sections.

**Related Work**

Various methodologies exist to model processor vulnerability to soft error. In the past, duplicated coarse-grained structures such as functional units, processor cores or hardware contexts have been used to detect and tolerate transient faults [21]. However, those approaches result in significant overhead in performance, area and power. [92] proposed to perform redundant execution only during low ILP and L2 misses in order to achieve high error coverage.
with low performance loss. In [93], SlicK is introduced to avoid the redundancy on results predictable instructions. Wang et al. [94] showed that soft errors produce observable symptoms, and used these to trigger roll back execution. Soft error tolerance techniques also exist at the circuit level. [80] established a relation between the atmospheric neutron soft error rate and technology feature size. [95] proposed a soft error detection circuit based on abnormal switching currents. Choudhury et al. [7] proposed the use of gate size and $V_{DD}$ as design parameters to realize SER robustness circuit design. [96] selectively resized the most vulnerable gates to improve the combination logic circuit single-event upset (SEU) robustness. In [97], Srinivasan et al. applied Asymmetric SRAM optimized for storing zero (ASRAM-0) to reduce structures’ SER based on the observation that most of the configuration bit stream is composed of zero. My approach is unique in that I explore designs using combined circuit and microarchitecture level SER robustness techniques.
Table 5-1. The studied SMT workloads

<table>
<thead>
<tr>
<th>Thread Type</th>
<th>Group</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Group A</td>
<td><code>bzip2, facerec, gap, wupwise</code>,</td>
</tr>
<tr>
<td></td>
<td>Group B</td>
<td><code>crafty, fma3d, mesa, perlbench</code></td>
</tr>
<tr>
<td></td>
<td>Group C</td>
<td><code>eon, gcc, wupwise, mesa</code></td>
</tr>
<tr>
<td>MIX</td>
<td>Group A</td>
<td><code>crafty, gap, lucas, swim</code></td>
</tr>
<tr>
<td></td>
<td>Group B</td>
<td><code>mcf, mesa, twolf, wupwise</code></td>
</tr>
<tr>
<td></td>
<td>Group C</td>
<td><code>equake, facerec, perlbench, vpr</code></td>
</tr>
<tr>
<td>MEM</td>
<td>Group A</td>
<td><code>applu, galgel, twolf, vpr</code></td>
</tr>
<tr>
<td></td>
<td>Group B</td>
<td><code>ammp, equake, lucas, twolf</code></td>
</tr>
<tr>
<td></td>
<td>Group C</td>
<td><code>lucas, mcf, mgrid, swim</code></td>
</tr>
</tbody>
</table>
Figure 5-1. Soft error robust SRAM (rSRAM) Cell (6T+2C). The rSRAM cell is built from a standard 6 transistor high density SRAM cell above which two stacked Metal-Insulator-Metal (MIM) capacitors are symmetrically added. The embedded capacitors increase the critical charge required to flip the cell logic state and lead to a much lower SER. The common node of the two capacitors is biased at $V_{DD}/2$.

Figure 5-2. The control flow of instruction dispatch in the proposed IQ using hybrid radiation hardening techniques.
Figure 5-3. An overview of radiation hardened IQ design using hybrid techniques

Figure 5-4. The wakeup logic of the RIQ
Figure 5-5. The designs alternative of the hybrid radiation hardened IQ. A) First design. B) Second design.

Figure 5-6. The correlation between ROB AVF and L2 cache miss
Figure 5-7. A comparison of normalized IQ SER, throughput and harmonic IPCs. A) Normalized IQ SER. B) Normalized throughput IPC. C) Normalized harmonic IPC.
Figure 5-8. Criticality threshold analysis. A) CPU combination workloads. B) MIX combination workloads. C) MEM combination workloads.
Figure 5-9. ROB SER reduction and processor power overhead with L2_miss trigger and enhanced trigger. A) ROB SER reduction. B) Power overhead.
Figure 5-10. Vulnerability threshold analysis. A) CPU combination workloads. B) MIX combination workloads. C) MEM combination workloads.
Figure 5-11. The aggregate effect of the proposed two techniques
The continued scaling of CMOS process technology to the nanometer scale has resulted in process variation (PV) that lead to significant variability in chip performance and power [101, 102, 104, 107, 141]. Existing work on analyzing and mitigating the impact of process variation on microprocessor design has been largely focused on performance and power domains [118, 119]. Since the soft error rate is inherently related to the device parameters (e.g. gate length \(L\) and threshold voltage \(V_{th}\)) [116, 117, 139], as technology advances, the impact of process variation should be taken into consideration by architects in reliability evaluation. Furthermore, techniques aimed at mitigating the process variation should also consider their reliability impact. Nevertheless, at the microarchitecture level, there has been no prior work on (1) characterizing the impact of parameter variation on soft error robustness, and (2) optimizing soft error reliability in light of process variation.

In this chapter, I characterize the impact of process variation on the processor microarchitecture soft error vulnerability. I find that although the critical charge (a widely used metric to evaluate the raw soft error rate (SER)) varies significantly under process variation at the bit-level, it exhibits small variation at the entry-level of microarchitecture structures. I examine the impact of two recently proposed process variation tolerant techniques (e.g. Variable Latency Register Files (VL-RF) [118] and Dynamic Fine Grain Body Biasing (DFGBB) [119]) on microarchitecture soft error reliability and find that both techniques increase the SER. I propose reliability-aware process variation tolerant mechanisms that are capable of achieving an optimal vulnerability-frequency-leakage operating point.
Experimental Methodology

This section presents a model of process variation and describes the methodology I use to estimate the impact of PV on soft error robustness. In addition, the microarchitecture-level soft error reliability computation method is introduced.

Process Variation Modeling

Process variation is a combination of random effects (e.g. random dopant fluctuations) and systematic effects (e.g. lithographic lens aberrations). Random variation refers to random fluctuations in parameters from die to die and device to device. Systematic variation, on the other hand, refers to the layout-dependant variation through which nearby devices share similar parameters. Devices exhibit spatial correlations within a given die. Die-to-Die (D2D) variation mainly exhibits as a random variation, whereas Within Die (WID) variation is composed of both random and systematic variation. I model variations on $L$ and $V_{th}$ since they are the two major process variation sources [102]. $L$ and $V_{th}$ in each device can be represented as follows:

$$ L = L_{nom} + \Delta L_{D2D} + \Delta L_{WID} \quad (6-1) $$

$$ V_{th} = V_{th, nom} + \Delta V_{th, D2D} + \Delta V_{th, WID} \quad (6-2) $$

where $L_{nom}$ and $V_{th, nom}$ are the nominal value of gate length and threshold voltage respectively. $\Delta L_{D2D}$ and $\Delta V_{th, D2D}$ represent the D2D variations. Devices in a single die share the same $\Delta L_{D2D}$ and $\Delta V_{th, D2D}$ which are generally constant offsets. $\Delta L_{WID}$ and $\Delta V_{th, WID}$ represent the WID variation which can be further expressed as the additive effect of systematic and random variations. I focus the PV modeling on within-die variation since die-to-die variation can be modeled as an offset value to all the devices within the chip.

To model the random effects of WID variation, I generate random variables that follow a normal distribution. To model systematic variations, I use the multi-level quad-tree partitioning
method proposed in [121], which has been widely used in prior work [105, 118, 122]. Figure 6-1 illustrates this method. A chip is covered by several layers and each layer has a number of quadrants. Each quadrant is assigned a random variable (following a normal distribution) and all devices in the same quadrant share the same value. The systematic variation of each device within the chip is equal to the sum of the random variables of the quadrants, through all the layers, to which it belongs. Nearby devices share more quadrants than far away devices. For example, as shown in Figure 6-1, devices in quadrant (2,1) and (2,2) share the random variables in quadrant (0,1) and (1,1), but devices in quadrant (2,1) and (2,16) only share quadrant (0,1). This approach effectively captures the spatial correlation between devices. I chose an area of 32 6T SRAM cells as the granularity of the smallest quadrant, which is sufficient to capture systematic variation. The WID variation follows a normal distribution (random variables are generated through Monte-Carlo simulation) with a standard deviation \( \sigma = \sqrt{\sigma_{\text{rand}}^2 + \sigma_{\text{sys}}^2} \), where \( \sigma_{\text{rand}} \) and \( \sigma_{\text{sys}} \) represent the standard deviation for random and systematic variation respectively. In this study, I simulate processors developed using 45nm process technology and assume \( \sigma/\mu = 12\% \) and \( \sigma_{\text{rand}} = \sigma_{\text{sys}} = \sigma/\sqrt{2} \) based on variability projections from [123]. I use the Alpha 21264 as our baseline machine. The layout is scaled down to 45nm from an Alpha 21264 chip floor-plan and 400 chips are generated for statistical analysis. Predictive Technology Models [124], the evolution of previous Berkeley Predictive Technology Models (BPTM), are used to provide the basic device parameters for HSPICE simulations.

**Impact on Circuit-Level Soft Error Vulnerability**

As Equation 5-1 shows, the SER increases exponentially with reduction in \( Q_{\text{crit}} \). In this study, we use the variability of critical charge under process variation to estimate the raw SER variation at the circuit level.
There are several types of storage cells (e.g. SRAM, Transmission Gate Flip-Flop, and dynamic latch) and the critical charge in each type is different. As an example, I present the analysis on a standard 6T SRAM in this subsection. Figure 6-2 shows the schematic. When the word line is not asserted, transistors M5 and M6 disconnect the cell from the bit lines and the two cross-coupled inverters formed by M1~M4 reinforce one another, maintaining a stable state in the cell. There are only two stable states (each state corresponds to store ‘0’ or ‘1’) in the SRAM: Q=0 and QB=1 or Q=1 and QB=0. When energetic particles hit Q (assuming Q=1) and discharge it, M2 begins conducting and charges QB. The state of the cell will be flipped after Q drops below \(V_{DD}/2\), and a soft error occurs. The estimation of critical charge can be expressed as follows:

\[
Q_{crit} = \int_0^{T_f} I_d dt
\]

(6-3)

where \(T_f\) is the flipping time and \(I_d\) is the drain current induced by the particle. Device parameters (e.g. \(L\), \(V_{th}\)) impact both \(T_f\) and \(I_d\), suggesting their effect on \(Q_{crit}\). I insert a current source (shown in Equation 6-4) into node Q to model the particle strike:

\[
I_{in}(t) = \frac{Q}{\tau_a - \tau_\beta} (e^{-t/\tau_a} - e^{-t/\tau_\beta})
\]

(6-4)

where \(Q\) is the charge deposited as a result of particle strike, \(\tau_a\) is the collection time constant of the junction, and \(\tau_\beta\) is the ion-track establishment time constant. Figure 6-3 depicts the impact of PV on SRAM \(Q_{crit}\). This figure plots two sets of our experimental results showing how voltages change when a SRAM with process variation (shown in Figure 6-3(a)) and without process variation (shown in Figure 6-3(b)) are attacked by the same particle at 100 picoseconds. \(\{V(Q), V(QB)\}\) refers to the voltages of Q and QB without PV and \(\{V(Q_{pv}), V(QB_{pv})\}\) refers to the voltages of Q and QB with PV. As can be seen, V(Q) and V(QB) maintain the initial state
and tolerate the strike while $V(Q_{pv})$ and $V(QB_{pv})$ reverse quickly after the strike, leading to a failure. Note that PV also has positive effect on critical charge since parameters can vary up and down around the mean value.

I model the critical charge variability of different types of storage cells and combinational logic (e.g. chain of 6 inverters, 2-input NAND gate) using the SER- $Q_{crit}$ model and HSPICE simulations similar to the one described above. The $L$ and $V_{th}$ of transistors are established using random variables from Monte-Carlo simulation. Figure 6-4 shows the chip-wide critical charge variation map obtained from our experiments (data are presented in units of fC). The results, which exhibit an average variation of 18.9% across the simulated chips, show that there is large variation in the critical charge under PV. Since SER is exponentially related to critical charge, one can expect an even larger variation in the SER in the presence of PV.

**Microarchitecture-Level Soft Error Vulnerability Analysis**

A key observation on the effect of soft error at the microarchitecture level is that a SEU may not affect a processor state that is required for correct program execution. The overall microarchitecture structure’s soft error rate without the impact of PV, as given in Equation 6-5 [126], is determined by two factors: the FIT rate (Failures in Time, which is the raw SER at the circuit level), mainly determined by circuit design and process technology, and the architecture vulnerability factor (AVF)

$$\text{SER} = \text{FIT} \cdot \text{AVF}$$ (6-5)

A hardware structure’s AVF refers to the probability that a transient fault within that hardware structure will result in incorrect program results. Therefore, the AVF, which can be used as a metric to estimate how vulnerable a hardware structure is to soft error during program execution, is determined by the processor state bits required for architecturally correct execution.
(ACE) [22]. At cycle level, the AVF of a hardware structure is the percentage of bits that are ACE within that structure. At program level, the AVF of a hardware structure is derived by averaging cycle-level AVF of the structure across program execution [22], as shown in Equation 6-6:

$$AVF = \frac{\sum_{\text{cycles}} \# \text{ACE bits per cycle}}{\#B \cdot T_{\text{cycles}}}$$

(6-6)

where $\#B$ is the number of bits in the structure. As can be seen, AVF is primarily determined by the quantity of ACE bits per cycle and their residency time within the structure.

Note that Equation 6-6 assumes that the frequency is constant. However, the frequency may vary throughout execution due to PV mitigation techniques (e.g. DFGBB boosts chip frequency [119]). To quantify the architecture vulnerability factor in light of process variation, I propose a PV-aware vulnerability metric $AVF_{pv}$, which is expressed as follows:

$$AVF_{pv} = \frac{\sum_{\text{cycles}} \# \text{ACE bits per cycle}}{\#B \cdot \sum_{\text{cycles}} \frac{1}{f_{pv}}}$$

(6-7)

where $f_{pv}$ is the frequency under the influence of PV mitigation techniques. Since the FIT/bit (determined by the critical charge) can have large variations due to PV, it is not appropriate to use a random bit’s FIT to compute the entire microarchitecture structure’s FIT (Note that without PV, the FIT of a structure is the product of FIT/bit and the number of bits in the structure) and obtain the SER. An analysis cell should be introduced, which can be as small as a single bit or as large as the entire structure. Each bit within a cell is assumed to share the same FIT/bit and the overall structure’s SER under PV ($SER_{pv}$) can be expressed as:

$$SER_{pv} = \sum_{\#\text{of cells}} FIT_{pv\_cell} \cdot AVF_{pv\_cell}$$

(6-8)
where $FIT_{pv\_cell}$ and $AVF_{pv\_cell}$ represent FIT and $AVF_{pv}$ of the finest analysis cell in the structure. Note that even though we assume each bit’s FIT within an analysis cell is identical, we still compute $AVF_{pv}$ at the bit level.

**Microarchitecture-Level Soft Error Vulnerability Characterization Under Process Variations**

In this section, I perform an extensive study to estimate FIT variation across the analysis cells. In addition, the impact of PV mitigation techniques on microarchitecture-level soft error vulnerability is presented.

**FIT Variation Across the Analysis Cells**

In the previous section, I introduce a method to quantify critical charge variation at the bit level. At the microarchitecture level, data within a structure are usually accessed on a per-entry basis. Therefore, a characterization of the critical charge variation at the entry level ($Q_{crit\_entry}$) can provide further insight for exploring reliability-aware PV optimizations. In this section, I present a $Q_{crit\_entry}$ variation analysis for the register file and the same characterization procedure has been applied to other microarchitecture structures. For a given microarchitecture structure, the analysis cell is set to be equal to the size of one entry of that microarchitecture structure. In this study, I opt to use the minimum critical charge as the $Q_{crit\_entry}$. Since a low critical charge increases the FIT rate, by choosing the minimum critical charge I base my estimation on the bit that is the most critical in determining the upper bound SER of an entry within the microarchitecture structure. I obtain a $Q_{crit\_entry}$ distribution for each chip’s register file and present data from the chip with the largest $Q_{crit\_entry}$ variation.

Figure 6-5 shows each entry’s $Q_{crit\_entry}$ within the 80-entry register file and a zoom-in view of the entry with the minimum $Q_{crit\_entry}$ by displaying each bit’s critical charge within that entry. Figure 6-6 plots both the entry-level and bit-level critical charge distribution within the
register file. The standard deviation of the entry-level critical charge (3.5%) is much smaller than that at the bit level (25.9%). This is because a large portion of bit-level critical charge variation is smoothed out at the entry level (shown in Figure 5). I can conclude that $Q_{\text{crit, entry}}$ only changes slightly within the structure under PV.

**Microarchitecture Soft Error Vulnerability Under Process Variation Mitigation Techniques**

In this section, I evaluate the impact of PV mitigation techniques on microarchitecture soft error vulnerability. I evaluate the effect of variable-latency register file and dynamic fine-grained body biasing techniques.

**The effect of variable-latency register file (VL-RF)**

The multi-ported Register File (RF) is a critical component in determining both the frequency and IPC of a processor. Delay in the register file is dominated by the SRAM access time. Frequency loss within the register file due to process variation can be reduced by applying variable-latency (VL) techniques. In [118], for each read port in the register file, entries are partitioned into fast and slow entries based on the SRAM read delay. Read operations can complete within one cycle in fast entries, but take two cycles in slow entries. These slow entries are not accounted for during register file frequency calibration. n% VL-RF defines the RF frequency based on the slowest read time of the fastest n% RF entries for each read port. The frequency is pre-defined by testing the read ports in each RF entry. In VL-RF, it is possible that a RF entry will have both slow and fast read ports. When a slow port is assigned to a read operation, one cycle stall is encountered in the pipeline that the port belongs to, and consequently, the issue width for the next cycle shrinks and IPC is reduced.

[118] evaluated the effect of VL-RF on performance. In this paper, we perform a complementary evaluation of its impact on reliability. The issue queue (IQ) is a good starting
point for such an investigation since shrinking the issue width directly affects the instruction
issue within the IQ. It is important to note that since the bandwidths of other stages (e.g. fetch,
renaming, etc) are not affected when a slow read port is selected, only the number of instructions
exiting the IQ is reduced and the number of instructions entering the IQ remains the same. As a
result, the IQ holds a larger number of instructions and the quantity of ACE bits within the IQ
increases correspondingly. Moreover, the slow read operation on RF will cause back-to-back and
dependent instructions to wait for an extra cycle, extending their residency cycles within the IQ.
Compared to the baseline case that includes PV but not fast/slow RF entries, IQ AVF increases
since both the quantity and residency time of ACE bits are increased. Figure 6-7(a) compares the
IQ AVF of the baseline case to that of using the VL-RF technique on the benchmark gcc over a
period of 375ms at 1ms granularity. Figure 6-7(b) plots the IQ AVF differences between the two
and the average number of issue pipe stalls during the interval. As can be seen in Figure 6-7(a),
the IQ AVF in VL-RF is substantially higher than that in the baseline case. As the number of
issue pipeline stalls climbs to a high level, there is a corresponding increase in the IQ AVF
difference between VL-RF and the baseline case (shown in Figure 6-7(b)). This illustrates the
degradation in soft error robustness that is introduced with VL-RF, and suggests that this
degradation is largely a result of the pipeline stalls.

In order to mitigate the IPC loss due to VL-RF, [118] proposed a port switching technique
that switches from slow ports to fast ports when reading from the RF. This technique
compensates for IPC loss by avoiding a large portion of reads from slow ports. This produces
fewer pipeline stalls, and the high IQ AVF caused by VL-RF is also mitigated. However, port
switching cannot eliminate the use of slow ports and therefore the IQ AVF is still higher than the
baseline case.
The effect of dynamic fine-grained body biasing (DFGBB)

Body Biasing (BB) applies a voltage between the source or drain and substrate to adjust the threshold voltage. Forward body biasing (FBB) decreases the $V_{th}$, decreasing the delay of the transistors - but making them leakier. Contrarily, reverse body biasing (RBB) increases the $V_{th}$, creating less leaky but slower transistors. The impact of body biasing on storage cells’ SER has been studied in the past. [138] found that FBB has the ability to improve a flip-flop’s soft error robustness by 35% and RBB degrades the SER by 36% in 130nm process technology. To evaluate the effect of body biasing on an SRAM’s SER in 45nm technology, I compute the critical charge using HSPICE simulations by biasing $V_{th}$ in the range of [-0.3v, 0.3v] in 6T SRAM with a resolution of 0.032v. Results are presented in Figure 6-8 (delta threshold voltage is equal to the subtraction of $V_{th}$ after BB has been applied from the original $V_{th}$. FBB corresponds to positive $\Delta V_{th}$ since it reduces $V_{th}$ and RBB corresponds to negative $\Delta V_{th}$ as it increases $V_{th}$). As can be seen, there is a linear relationship between critical charge and $V_{th}$, and critical charge reduces as $V_{th}$ increases. Compared to the baseline case without $V_{th}$ biasing, the critical charge increases 44% with a forward body biasing of 0.3v, and decreases 33% when a reverse body biasing of 0.3v is applied. Since the SER is exponential to the critical charge, the impact of the varying $V_{th}$ on the SER will be further amplified.

To mitigate the impact of PV on performance under a pre-defined power budget, Static Fine-Grained Body Biasing (SFGBB) is applied during the chip calibration time. It partitions the chip into cells, and statically sets each cell’s body biasing value under the worst case (e.g. the cell is fully loaded in the worst case temperature). Recently, Teodorescu and Torrellas proposed Dynamic Fine-Grain Body Biasing (DFGBB) scheme [119], which allows bias voltages to adapt to dynamic conditions. Take a cell which is initially slow for example, SFGBB applies forward body biasing at the calibration time. With DFGBB, each cell’s body biasing value is set to be the
SFGBB value when it is first powered-up. DFGBB dynamically reduces the forward body biasing or even apply reverse body biasing to save the leakage power when the cell is under-loaded. Conversely, when the cell is fully loaded, the temperature increases which results in a frequency loss, DFGBB increases forward body biasing value to improve the frequency. As can be seen, DFGBB is able to reduce the power overhead by adjusting the body biasing value of the cell to the dynamically changing workload. DFGBB, however, ignores the reliability domain characteristics during body biasing adaptations and results in reliability degradation. For example, when an L2 cache miss occurs and the pipeline stalls, cells such as the IQ and ROB buffer large numbers of instructions, leading to a high AVF. In this scenario, DFGBB will reduce the forward body biasing or even apply reverse body biasing, and as a result, degrade the soft error robustness because of the associated decrease in the critical charge. On the other hand, when the structure has a high throughput and the instructions’ residency time is short, the AVF is low. In this situation, DFGBB will increase the forward body biasing that is applied since the temperature is increased. This forward body biasing reduces the FIT (affected by critical charge) but the SER improvement is limited due to the already low AVF. To summarize, DFGBB does not effectively capitalize on opportunities to reduce a structure’s vulnerability.

Reliability-Aware Process Variation Mitigation

As discussed above, previous PV mitigation techniques degrade microarchitecture soft error robustness. In this section, I propose reliability-aware PV tolerant schemes that are capable of (1) mitigating microarchitecture soft error vulnerability and (2) achieving an optimal trade-off between performance, reliability and power in the presence of PV.

Entry-Level Granularity Vulnerability Mitigation

I propose a technique that operates at a fine granularity to reduce IQ vulnerability based on VL-RF with port switching (VL-RF+PS). I call the technique Entry Based Vulnerability
Mitigation (Entry-BVM). Note that port switching is not an essential requirement for my technique, but my method is adaptable to VL-RF. My goal is to mitigate IQ AVF without losing the performance improvement obtained by VL-RF+PS. Since high IQ AVF is a result of both a large quantity of ACE bits and the long residency cycles of ACE bits in the IQ, to reduce IQ AVF both of these factors should be considered.

As described in Chapter 4, assigning ACE-ready instructions a higher issue priority than un-ACE instructions can reduce the residency time of ACE bits because these instructions will be removed from the IQ more quickly, reducing the number of resident ACE bits. Greater reliability benefit is gained when the issue width decreases due to slow RF reads. As the issue pipe becomes more competitive, granting an issue request to an ACE instruction can reduce its probability of waiting extra cycles within the IQ.

As described in [22], an instruction cannot be characterized as ACE or un-ACE until its dependence chain has been determined. This can be difficult to accomplish at run time, but there are several methods available to characterize ACE instructions on the fly. A method that uses a SlicK table is proposed in [93] to identify ACE instructions by taking advantage of the time lag between the leading and trailing threads. This method, however, is not adaptable to my design. In [64], a critical table was proposed to quantify an instruction’s criticality. The rationale of this approach is that performance critical instructions usually have a large number of dependent instructions and their computation results are likely to be propagated to the program final output. Consequently, there is a large overlap between ACE and performance critical instructions. In this study, I apply the criticality-based approach to dynamically identify vulnerable instructions: instructions with high criticality are assumed to be ACE instructions. Note that I still perform rigorous AVF computation [22] in the technique evaluation.
With VL-RF+PS, a large number of ACE bits in the IQ are created by the imbalanced bandwidth between the issue stage and the other pipeline stages. In [118], a stall signal is generated and sent to the issue stage when a slow RF read occurs. In Entry-BVM, this signal is also sent to other pipeline stages since the input/output imbalance should be avoided in all pipeline stages if possible. For example, if the dispatch stage were the only stage to receive the stall signal, the decode buffer would have to tolerate the imbalance and the vulnerability would migrate from the IQ to the decode buffer. The only exception is the commit stage since reducing the commit width degrades the IPC. In Entry-BVM, RF entries with a larger number of fast ports are assigned a higher renaming priority, and by doing this, slow RF entries are avoided. This has a positive effect on IPC and helps to compensate for the performance loss caused by issue width reduction. Similar to VL-RF+PS, the per-port speed information for registers is loaded from a ROM which records all of the pre-collected port speed information for each RF entry. Therefore, it is straightforward to obtain the number of fast ports within the registers. To implement this approach, two bits describing the fast port quantity in the 4-port RF entry are added into the free list macro. These bits are read when selecting free physical registers for register renaming.

Figure 6-9 presents an architectural overview of Entry-BVM. Floating point structures are omitted due to space limitations, but are similar to the integer structures. Note that Entry-BVM is based upon VL-RF+PS. The detailed implementation of VL-RF+PS, such as the slow and fast RF entry pre-partitioning, VL-RF frequency pre-definition, and the circuit support (e.g. latches, MUX) for port switching, is described in [118]. In the renaming stage, registers with a large number of fast ports will be given higher priority when selecting from the free physical register pool. The critical table is added to implement dynamic vulnerability identification. The table is updated when an instruction is renamed in order to record up-to-date dependence chain
information. For each instruction within the IQ, the table provides the number of dependent instructions in the pipeline. A larger number represents a higher performance criticality. Upon table lookup, instruction criticality is provided to the IQ. This is done in parallel with the instruction wakeup stage and does not introduce any delay into the pipeline. Instructions are recognized as ACE (“1”) or un-ACE (“0”) if their criticality is higher or lower than a given threshold. The request signal accompanied by the vulnerability (“1” or “0”) information is sent to the selection logic when the instruction becomes ready. The selection logic grants the request signal, taking into consideration the vulnerability information. In the register read stage, the stall signal will be sent to other stages, and their bandwidth will be reduced by one per stall signal at that cycle. Since variable-latency techniques can be applied to other microarchitecture structures, Entry-BVM is also adaptable to other structures.

**Structure-Level Granularity Vulnerability Mitigation**

In order to fully exploit soft error robustness improvement opportunities while achieving optimal trade-offs between vulnerability, performance and power, I propose coarse-grain, Structure Based Vulnerability Mitigation (Structure-BVM) techniques which use two metrics - \( AVF_{pv} \) and \( IPC*fpv \) - to guide dynamic body biasing adaptations. Although IPC is a widely used performance metric, it is not suitable for measuring performance across chips with variable frequencies. In my scheme, \( IPC*fpv \) is used for performance evaluation. Similar to [119], Structure-BVM is implemented based on SFGBB where the body biasing value is statically set for the worst-case execution environment. The maximum forward body biasing applied to Structure-BVM is the body biasing value determined by SFGBB since the power consumption cannot exceed the upper bound power overhead.

The AVF of a microarchitecture structure exhibits significant variation at runtime [30, 32] and it is possible to achieve a considerable SER improvement by reducing the raw FIT when
AVF is high. Since both $AVF_{pv}$ and $IPC*fpv$ are considered in Structure-BVM and do not always exhibit a strong correlation (i.e. high IPC does not always imply a high AVF, and vice versa), I propose the categorization of program execution into four types of phases based on $AVF_{pv}$ and $IPC*fpv$; namely high $AVF_{pv} + high \ IPC*fpv$, high $AVF_{pv} + low \ IPC*fpv$, low $AVF_{pv} + high \ IPC*fpv$ and low $AVF_{pv} + low \ IPC*fpv$. The boundary of a phase is the mean value of $AVF_{pv}$ and $IPC*fpv$, which equally partitions the two combined parameters into four sets. Consequently, the number of times I apply FBB and RBB will be roughly equal, avoiding the substantial power overhead of over using FBB or the performance loss of over using RBB. The above two-dimensional partition allows me to explore desirable trade-offs across multiple domains (e.g. reliability, performance and power). For instance, a high $AVF_{pv}$ phase will require the application of forward body biasing to gain the highest degree of soft error robustness; and a higher $AVF_{pv}$ corresponds to a higher forward body biasing value. On the other hand, for a low $AVF_{pv}$ phase, I can apply reverse body biasing to save leakage power with negligible effect on $SER_{pv}$; similarly a lower $AVF_{pv}$ corresponds to a higher reverse body biasing value. However, overall performance may be reduced significantly when reverse body biasing is applied during a phase of low $AVF_{pv} + high \ IPC*fpv$. Because of this, reverse body biasing will not be applied if the total performance cannot be maintained at a level comparable to the baseline case with process variation without any optimization. While in a phase of low $AVF_{pv} + low \ IPC*fpv$, the use of reverse body biasing will be less constrained since its negative effect on performance is small. Figure 6-10 illustrates how forward body biasing and reverse body biasing are applied in each phase in the proposed scheme.

I present the pseudo code of our technique in Figure 6-11. Body biasing is adjusted at an interval granularity with a length of 1ms. At the beginning of each interval, the $AVF_{pv}$ and
$IPC*f_{pv}$ of the last interval are computed (the temperature effect on $f_{pv}$ is also considered in our study). Their maximum, mean and minimum values are updated correspondingly. The performance effect due to body biasing is evaluated, which consists of the product of the last interval’s IPC and the frequency difference with and without body biasing applied within the interval. The overall performance impact (perf_gain) is updated correspondingly. The $AVF_{pv}$ and $IPC*f_{pv}$ of the current interval are predicted to be the same as last interval, and the body biasing value for the current interval is determined by $AVF_{pv}$. The total performance effect and $IPC*f_{pv}$ are used to check whether RBB should be applied. Since the body biasing value is discrete, I partition $[mean\ AVF_{pv}, maximum\ AVF_{pv}]$ into smaller, equal ranges. The number of ranges is determined by FBB_steps, which is the total number of FBB steps, and each range corresponds to a FBB value. Similarly, $[minimum\ AVF_{pv}, mean\ AVF_{pv}]$ is partitioned and linked to RBB values. These ranges can be partitioned statically to avoid the overhead of the maximum, mean, and minimum $AVF_{pv}$ updates and dynamic range computations. Since $AVF_{pv}$ can only vary from 0% to 100%, the $AVF_{pv}$ range can be pre-determined using information such as the number of body biasing steps that can be applied. Static range partitioning, however, does lose the ability to keep a balanced utilization between FBB and RBB (see Evaluation Section for a detailed evaluation).

As can be seen, Structure-BVM requires online AVF estimation for the structure. The critical table introduced in Entry-BVM for ACE instruction identification is not applied in Structure-BVM, because every cell that requires AVF computation must access the table. This would introduce a large number of read ports and wires into the table, increasing the power and area overhead. [140] proposed online AVF estimation by performing fault injections during program execution and observing whether the errors result in failures. This requires extra
hardware support and results in an area overhead. In this study, I use the methodology proposed in [76], which computes the number of ACE bits by observing the number of instructions read and written into the cell on every cycle, therefore, performs just-in-time vulnerability estimation. The dynamic vulnerability estimation is used to guide body biasing adaptation within the cell and accurate AVF computation is still performed for evaluation purposes. Fine-grain cells within the chip are generally partitioned at the microarchitecture structure level [108] and [119] demonstrates the feasibility of applying body biasing techniques at a per structure level. Note that FGBBB is applied independently to each structure, and therefore my technique does not simply migrate vulnerability from one structure to another. To implement body biasing techniques, a bias generator is attached to the structure. It generates bidirectional but equivalent body biasing values for PMOS and NMOS transistors separately, because the same body biasing value has an opposite effect on the two types of transistors. The detailed circuit modifications can be found in [119].

**Evaluation**

In this section, I describe the experimental methodology and evaluate the two proposed reliability-aware PV mitigation techniques. Next, I examine the aggregated effects of the two proposed techniques. Finally, I discuss the area and power overhead.

**Experimental Methodology**

To evaluate reliability at the microarchitecture level, I use the reliability-aware simulation framework Sim-SODA [131]. In addition, I port Wattch [91] into my simulation framework for dynamic power evaluation. The leakage power is estimated using HotLeakage [134]. The power results are scaled based on technology projections from ITRS [135]. The HotSpot tool [136] is used to estimate chip temperature. I use a default Alpha 21264 machine configuration with 20-entry INT and 15-entry FP IQs, an 80-entry ROB, an 80-entry INT and 72-entry FP register file.
with 4-rd/2-wr ports, a 32KB L1 D-cache, a 32KB L1 I-cache, and a 2MB L2 cache. The processor pipeline bandwidth is set to four. I use SPEC CPU 2000 integer and floating point benchmarks in our evaluation. The Simpoint tool [26] is used to identify the most representative simulation interval for each benchmark. After that, each benchmark is fast-forwarded to its representative interval before detailed simulation takes place. I simulate 500 million instructions for each benchmark and present the average result across 400 chips. 80% VL-RF is chosen in [118] because it is built on 65nm technology, which has smaller process variation than 45nm technology. For the Entry-BVM technique, we implement 70% VL-RF and obtain a 10% frequency increase compared to the chip without VL-RF. A larger portion of slow RF entries has to be discounted for in order to gain a significant frequency improvement in 45nm. I set the body biasing range as [-0.3v, 0.3v], with a resolution of 32mv. The dynamic FIT computation is based on the linear interpolation shown in Figure 6-8.

Besides mitigating microarchitecture vulnerability in the presence of process variation, my techniques also target achieving the optimal trade-off among reliability, performance and power. I propose a metric Vulnerability-Power/Performance (VPP), which can be expressed as

$$\frac{SER_{pv} \cdot power}{IPC \cdot f_{pv}}$$

, to evaluate the cross domain efficiency of the proposed techniques. A better technique will achieve a lower VPP.

Effectiveness of Entry-BVM

We compare our Entry-BVM scheme to VL-RF and VL-RF+PS techniques. Figure 6-12 (a)-(b) shows the IQ SER and performance ($IPC*f_{pv}$) across the studied benchmarks. Results are normalized to the baseline case without any optimization under the impact of process variation. As expected, VL-RF reduces IQ reliability significantly. For example, the IQ SER increases by 58% on gcc. As can be seen, VL-RF does not introduce a severe IQ reliability degradation on
some benchmarks (e.g. equake, mcf). This is because there are a large number of L2 cache
misses in those benchmarks which limit the quantity of ready instructions in the IQ for issue.
There are still enough pipes for instructions’ issue even though the number of pipes decreases
because of VL-RF. Consequently, the IQ AVF is slightly affected by VL-RF. As shown in
Figure 6-12 (a), on average, VL-RF degrades the IQ SER by 18% compared to the baseline case.
VL-RF+PS mitigates the negative effect of VL-RF to some degree, but still increases IQ SER by
8%. Entry-BVM exhibits a strong ability to improve IQ soft error robustness. It not only
mitigates the vulnerability increase caused by VL-RF techniques, but further reduces IQ SER by
24%. As Figure 6-12 (b) shows, the $IPC \times f_{pv}$ of VL-RF+PS and Entry-BVM is almost equal, thus
Entry-BVM is able to maintain the same performance improvement as VL-RF+PS. Figure 6-13
shows the trade-off metric VPP across the various techniques. The results are normalized to the
baseline case. Entry-BVM produces a much lower VPP than other approaches. On average, the
VPP of Entry-BVM is 50% and 28% lower than that of VL-RF and VL-RF+PS respectively.
Since Entry-BVM controls the number of input/output instructions to each microarchitecture
structure in the entire pipeline via broadcasting the stall signal, it does not migrate the
vulnerability from the IQ to other microarchitecture structures.

**Effectiveness of Structure-BVM**

I first compare Structure-BVM schemes (using both dynamic and static $AVF_{pv}$ range
partitioning) with DFGBB [119] on reliability. Figure 6-14 presents the IQ SER across the
various techniques. Results are normalized to the baseline case. Because DFGBBB does not
effectively take advantage of FBB to improve soft error robustness, it does not decrease but
instead increases the IQ SER by an average of 8% across the studied benchmarks. Compared to
the baseline case, the static $AVF_{pv}$ range partition slightly affects IQ vulnerability because it lacks
the ability to adapt to the dynamic $AVF_{pv}$ behavior and choose an optimal BB value. Structure-
BVM with dynamic $AVF_{pv}$ range partitioning shows a strong ability to reduce IQ vulnerability. As a result, the IQ SER is 30% and 20% lower than that on DFGBB and the baseline case respectively. The normalized performance ($IPC*fpv$) results are shown in Figure 6-15. As can be seen, DFGBB suffers a considerable performance loss even though it maintains the frequency, while Structure-BVM with dynamic $AVF_{pv}$ range partition maintains or improves performance by 22%. This is because the per-interval performance constraint does not incur any negative effect from biasing the threshold voltage. Consequently, performance usually increases. Note that while Structure-BVM has an intrinsic power constraint, both DFGBB and Structure-BVM have the ability to reduce the power overhead introduced by SFGBB. One can expect a greater power saving on DFGBB than on Structure-BVM, since Structure-BVM trades off a certain amount of power reduction in improving reliability and performance. Instead of analyzing the power consumption of these techniques, it is more interesting to evaluate the trade-off among the three domains. I present the normalized VPP in Figure 6-16. On average, the VPP of DFGBB is 40% higher than Structure-BVM with dynamic $AVF_{pv}$ range.

As shown in Figure 6-16, the benefit gained from Structure-BVM varies substantially across benchmarks. For example, a large IQ SER reduction is obtained on equake, while the IQ SER on swim is only slightly changed when Structure-BVM is applied. With Structure-BVM, intervals which have close-to-mean $AVF_{pv}$ will be assigned a close-to-zero BB value. If most of an application’s intervals have an $AVF_{pv}$ close to the mean value, the effectiveness of Structure-BVM is reduced. I select one representative chip and show the $AVF_{pv}$ - $IPC*fpv$ plot in Figure 6-17 for intervals on equake and swim respectively. Dotted lines mark the boundaries of the four phases, and an area is drawn with its center (with mean $AVF_{pv}$ and mean $IPC*fpv$) at the
intersection of the boundaries. As is shown, a large number of intervals in *swim* are covered by the area whereas few intervals in *equake* are included within the area.

**Combining Entry-BVM and Structure-BVM**

As the results have shown, both Entry-BVM and Structure-BVM exhibit a strong ability to improve a structure’s soft error robustness. Since they focus on difference levels, they can be combined to further reduce the vulnerability and push the trade-off among the three studied domains closer to an optimal point. Figure 6-18 (a) and (b) show the normalized IQ SER and VPP under the aggregate effect of Entry-BVM + Structure-BVM across the benchmarks. Compared to the baseline case with process variation, the combined technique reduces IQ SER substantially by 40% while improving VPP by 46%.

**Overhead of Entry-BVM and Structure-BVM**

It was found in [118] that VL-RF+PS adds a 2% area overhead and less than 5% power overhead. Entry-BVM adds a small amount of logic in register renaming and instruction selection on VL-RF+PS. A $n \cdot n$ bit table is added for dynamic vulnerability identification, where $n$ is the number of ROB entries. I estimate the extra area overhead introduced by Entry-BVM to be less than 1% and with negligible power overhead. Previous studies have also estimated the overhead of body biasing techniques [108]. Generally, FGBB introduces a 2% area overhead and a 1% power overhead. The timing overhead of FGBB is negligible [119]. Structure-BVM updates several parameters (e.g. $AVF_{ps}$, $IPC*fpv$) at the end of each interval, however the processor does not need to stop for these parameter updates. The previous BB can be applied until its re-evaluation is completed, so Structure-BVM introduces no timing overhead. The extra power consumed by the updates is small. There are few circuits and buffers added by Structure-BVM and the estimated extra area overhead is less than 1%.
Related Work

There have been many studies on characterizing the impact of PV on performance and power. Borkar et al. [101] investigated the impact of PV on both circuit and microarchitecture levels. Bowman et al. [102] introduced maximum clock frequency (FMAX) to model the chip frequency under PV. Orshansky et al. [103] derived a model to estimate the performance degradation for a given circuit and process parameters in the presence of PV. Agarwal et al. [105] presented a statistical timing analysis technique that accounts for WID variations with spatial correlation. Statistical modeling and analysis of leakage power under PV were performed in [104, 106, 107]. There are limited works studying the impact of PV on SER. Ding et al. [110] investigated the impact of $L$ and $V_{th}$ variations on critical charge separately. Their work was performed at the circuit level and is limited to single storage cells with single parameter variation. Our study extends the SER variation characterization to the microarchitecture level and considers the combinational effect of $L$ and $V_{th}$ variations.

Various PV mitigation techniques have been proposed in the past. Body biasing is widely applied to mitigate both D2D and WID variation [102] or achieve an optimal tradeoff between frequency and power [108]. [109] developed power optimization techniques considering the PV effect based on the statistical analysis. [115] showed that PV can slow down the issue queue read/write operations and hurt the performance, it proposed instruction steering, operand- and port-switching to improve the performance. Tiwari et al. [111] proposed Recycle, which applies cycle time stealing to microarchitecture pipeline stages for frequency improvement. Liang et al. [112] proposed a memory architecture based on novel 3T1D DRAM to tolerate PV. [113] applied two fine-grained tuning techniques- voltage interpolation and variable latency to reduce the frequency variation between chips, between cores on one chip and between units within one core. [14] used linear programming to find the best voltage and frequency levels for each core of
the CMP in the presence of PV. However, these PV optimization schemes largely ignore microarchitecture soft error reliability. In this paper, I proposed novel techniques to mitigate the negative PV effect considering reliability, performance and power factors.
Figure 6-1. Multi-level quad-tree partitioning to model system variation

Figure 6-2. Standard 6T SRAM schematics with current source inserted at Q

Figure 6-3. V(Q) and V(QB) of SRAM (in 45nm processing technology) under a particle strike. 
A) Without PV. B) In the presence of PV.
Figure 6-4. Critical charge variation map for a chip (data are presented in units of fC).

Figure 6-5. $Q_{\text{crit, entry}}$ of each entry in the register file (80 entries, each entry contains 64 bits).

Figure 6-6. Entry- and bit- level $Q_{\text{crit}}$ distribution in the register file.
Figure 6-7 A). IQ AVF in baseline case and VL-RF technique on gcc over a period of 375ms. B) The # of issue pipe stalls and IQ AVF difference between VL-RF and baseline case.

Figure 6-8. Critical charge vs. delta threshold voltage in range of [-0.3v, 0.3v] in 6T SRAM
Figure 6-9. Entry-BVM architectural overview

Figure 6-10. BB in each phase
1. Every 1ms
2. {
3.   AVFpv and IPC$f_pv$ computing();
4.   Max, Min and Mean values updating();
5.   Perf_gain computing();
6.   if AVFpv belongs to [Mean_AVFpv, Max_AVFpv]
7.   then
8.     {AVF_step = (Max_AVFpv - Mean_AVFpv)/FBB_steps;
9.        BB = Max_FBB*(AVFpv-Mean_AVFpv)/AVF_step;
10.   } else
11.     {AVF_step = (Mean_AVFpv - Min_AVFpv)/RBB_steps;
12.        BB = Max_RBB*(Mean_AVFpv-AVFpv)/AVF_step;
13.        if IPC$f_pv$ > Mean_IPC$fpv$ and Perf_gain < 0
14.          then BB = 0
15.     }
16. }

Figure 6-11. Structure-BVM Pseudo-code

Figure 6-12. Normalized IQ SER and $IPC*fpv$ yielded by VL related techniques. A) Normalized IQ SER. B) Normalized $IPC*fpv$
Figure 6-13. Normalize VPP on VL related techniques

Figure 6-14. Normalize IQ SER with BB related techniques

Figure 6-15. Normalized $IPC^*f_{pv}$ with BB related techniques
Figure 6-16. Normalized VPP with BB related techniques

Figure 6-17. $AVF_{pv}$ - $IPC*fpv$ plot for intervals on *equake* (up) and *swim* (down) (Intervals within the drawn area are close to the mean value)
Figure 6-18. A) Normalize IQ SER under Entry-BVM + Structure-BVM. B) Normalized VPP under Entry-BVM + Structure-BVM
CHAPTER 7
NBTI TOLERANT MICROARCHITECTURE DESIGN IN THE PRESENCE OF PROCESS VARIATION

The technology scaling has resulted in the convergence of several factors (e.g. the introduction of nitrided oxides, the increase in gate oxide fields, and operating temperature), which have made Negative Bias Temperature Instability (NBTI) [100] a critical reliability threat for deep sub-micrometer CMOS technologies. NBTI increases the PMOS transistor threshold voltage (Vth) and reduces the drive current, causing degradation in circuit speed and requires an increase of the minimal voltage in storage cells to keep the content. The PMOS degradation (i.e. wear-out) problem due to NBTI is aggravated in the presence of process variation. Under the impact of PV, circuit operating frequency decreases significantly after the chip is fabricated (frequency is determined by the slowest critical path). The NBTI effect further exacerbates circuit performance degradation during chip operation due to increased $V_{th}$. Consequently, the decreasing circuit operating frequency is a cumulative effect of both PV and NBTI. Current PV-tolerant mechanisms largely ignore the NBTI wear-out problem. On the other hand, existing NBTI-tolerant techniques lack the ability to address the deleterious impact of PV. As a result, the chip can still suffer a significant frequency loss and increased power overhead even though the NBTI-tolerant mechanisms are applied. In the upcoming nano-/atom- scale transistor design era, microarchitecture design techniques which can effectively address the combined PV and NBTI effect are greatly needed.

In this chapter, I show that simply combining PV mitigation techniques with NBTI recovery mechanisms cannot efficiently address the aggregated effect. Observing that process variation has both positive and negative effects on circuits, I take advantage of the positive effects in NBTI-tolerant design. I propose three microarchitecture NBTI reliability enhancements in the presence of process variation which mitigate the detrimental impact of PV and NBTI
simultaneously, while achieving attractive trade-offs among chip performance, power, lifetime, and area overhead. I show that the proposed techniques can be applied to a wide range of microarchitecture structures, leading to significant reliability and performance improvements at the chip level.

**Background**

In this section, I illustrate the effect of NBTI on PMOS transistors and describe mechanisms to recover NBTI degradation. The interaction between NBTI and PV is discussed as well.

**Negative Bias Temperature Instability (NBTI)**

NBTI is the result of interface trap generation in the silicon/oxide interface of PMOS transistors. When the PMOS transistor is under negative voltage, the silicon-hydrogen bonds at the silicon/oxide interface can easily break and generate interface traps ($N_{IT}$). $N_{IT}$ captures electrons flowing from the source to the drain and increases the PMOS threshold voltage. As a result, the transistor becomes slower and can cause failures when the delay exceeds timing specifications. NBTI leads to failures in the storage cell as well. Higher $V_{th}$ requires a higher $V_{min}$ to keep the content and $V_{min}$ in the cell may not be able to satisfy this requirement due to limited power budget. Note that PBTI (Positive Bias Temperature Instability) also occurs in NMOS transistors. However, its impact is negligible compared to the NBTI effect in PMOS transistors [84]. NBTI degradation can be recovered when the positive voltage is set at the gate of PMOS transistors. It helps to heal the interface traps generated, which partially recovers $V_{th}$. Thus, a PMOS experiences the period of either stress mode (gate is set as “0”) or recovery mode (gate is set as “1”) during its lifetime. The NBTI degradation is partially recovered once the stress is moved. Therefore, minimizing the period during which negative voltage is applied at the gate of PMOS can reduce the NBTI effect. Other methods, such as resizing PMOS or reducing the
operating voltage, can also be applied to mitigate NBTI degradation [120, 126]. As discussed in [127], considering performance, power, and area overhead introduced, reducing the amount of time PMOS under stress outperforms other NBTI mitigation methods.

To mitigate NBTI degradation in combinational logic units, [127] proposed the use of special vectors as input into the units when they are idle; avoiding the aggressive stress on a specific PMOS. As a result, PMOS transistors in the units degrade evenly and their lifetime is extended since lifetime is determined by the most degraded PMOS. In storage cell (e.g. 6T SRAM) based structures (e.g. register file and cache), there is always one PMOS under stress and another under recovery. Therefore, the best NBTI degradation scenario is to degrade the two PMOS in the SRAM evenly. Storing “0” and “1” 50% of the time can achieve balanced NBTI degradation. To achieve this goal, [127] observed that on average, a register file entry is free (time between the release and the next write operation) around 50% of the time and proposed to invert the register file entry while in the free state. In addition, [127] proposed to invalidate and store the sampled inverted values into 50% of the L1 cache lines during the entire lifetime to statistically degrade the two PMOS in each SRAM bit evenly.

Guardbanding, as a conservative approach and a last resort, can be used to tolerate NBTI degradation. Guardbanding reduces the processor frequency or increases the minimal voltage to defend against the expected degradation in logic circuits or storage structures during the targeted microprocessor lifetime. For instance, in [128], 20% of the cycle time is reserved to combat NBTI degradation. Mitigating NBTI degradation can reduce the necessity of guardbanding, leading to improvements in frequency and power savings. However, NBTI mitigation techniques can cause performance penalties and power overhead, making it a poor choice if the overhead outweighs that of guardbanding. $NBTI_{efficiency}$ (shown as Eq.1) is proposed in [127] to evaluate
the efficiency of NBTI tolerant schemes. It quantifies the trade-off among performance (Delay),
power and area overhead (TDP), and lifetime (the amount of required NBTIguardband). The
Delay and TDP obtained by the technique will be normalized to the case without NBTI and PV
effects. As can be seen, lower $NBTI_{efficiency}$ implies an improved approach and the optimum
technique will achieve a $NBTI_{efficiency}$ of 1 since both the Delay and TDP will be 1, and the
$NBTI_{guardband}$ is equal to zero.

\[ NBTI_{efficiency} = (\text{Delay} \cdot (1 + NBTI_{guardband}))^3 \cdot \text{TDP} \] (7-1)

**The Interplay Between NBTI and PV**

As described earlier, both NBTI and PV affect PMOS $V_{th}$. Therefore, guardbanding should
consider the potential $V_{th}$ increase contributed by all factors. Only targeting on NBTI (or PV)
underestimates the guardband requirement and results in a shorter lifetime. This is because the
frequency loss and power overhead caused by PV (or NBTI) is not counted. On the other hand,
simply adding a NBTI guardband to the PV guardband will overestimate the actual guardband
investment since doing so conservatively assumes the worst case scenario and ignores the benign
impact of PV on NBTI, which helps reduce the guardband. The excessive guardband causes
unnecessary frequency loss and power overhead.

Since parameters vary around their nominal design specification, PV can have both
positive and negative effects on transistor characteristics: it either decreases $V_{th} (-\Delta V_{th})$ or
increases $V_{th} (+\Delta V_{th})$. NBTI degradation only increases $V_{th}$, but the amount of increase on PMOS
$V_{th}$ varies significantly due to the different stress period. NBTI impact can be generally described
as either high $V_{th}$ increase ($high \_\Delta V_{th}$) or low $V_{th}$ increase ($low \_\Delta V_{th}$). We can classify the
aggregated effect of PV and NBTI into four categories: $-\Delta V_{th}$ & $high \_\Delta V_{th}$, $-\Delta V_{th}$ & $low \_\Delta V_{th}$,
$+\Delta V_{th}$ & $high \_\Delta V_{th}$ and $+\Delta V_{th}$ & $low \_\Delta V_{th}$. The guardband will be as high as the sum of NTBI
and PV guardbands if \( +\Delta V_{th} \) \& \( \text{high} - \Delta V_{th} \) dominates. Note that NBTI is a temporal effect, its impact on \( V_{th} \) dynamically changes across runtime during the lifetime, depending on the fraction of time its gate is set as “0”. The \( \text{high} - \Delta V_{th} \) shift can be compensated by PMOS with \( -\Delta V_{th} \) with low performance penalty and power overhead. Therefore, the total guardband can be reduced to the max(\( -\Delta V_{th} \) \& \( \text{high} - \Delta V_{th} \), \( +\Delta V_{th} \) \& \( \text{low} - \Delta V_{th} \)) and a large amount of frequency and power savings is reclaimed. In an ideal scenario, where all positive effects of PV are exploited to mitigate the NBTI degradation, guardband will decrease to as low as the PV guardband. Figure 7-1 illustrates the difference between the conservatively estimated guardband with the optimized one which considers the interaction between NBTI and PV. The difference can be as large as 36% based on our evaluation.

As discussed above, to achieve an optimized NBTI+PV guardband setting, it is important to consider the interaction between NBTI and PV. However, to my knowledge, existing NBTI and PV tolerant mechanisms [118, 119, 127] address the two factors individually and separately. In this study, I propose several cost-effective PV-aware NBTI tolerant methodologies.

**Process Variation Aware NBTI Tolerant Microarchitecture**

In this Section, I argue that simply putting NBTI and PV tolerant techniques together can only reduce the total guardband requirement to a limited extent. Moreover, even though it can maximally reduce the guardband in some cases, it results in a large performance penalty. To efficiently reduce the total guardbanding while minimizing the negative impact on performance and power, I propose a set of PV-aware NBTI tolerant techniques for different types of microarchitecture structures that can exploit the positive interaction between NBTI and PV.
Motivation

In order to reduce the required NBTI and PV guardbands, one can apply NBTI tolerant and PV mitigation techniques together. This will mitigate the NBTI degradation and the deleterious PV effect independently. Take a multi-ported register file (RF) as an example. It is comprised of combinational logic circuits (decoders, wordlines, bitlines, and output amplifiers) and storage cells (SRAM based RF entries). The NBTI mitigation techniques that target logic circuits and storage cells can be applied to reduce NBTI guardband. The NBTI guardband of the entire RF is determined by the highest NBTI guardband of the two parts. Meanwhile, the VL+PS (i.e. variable latency and port switching) scheme can be applied to the RF to reduce the frequency loss caused by PV and to minimize the PV guardband. However, as my evaluation results show in the Evaluation Section, simply putting the NBTI and PV mitigation techniques together only reduces the PV guardband and even has a negative effect on NBTI guardband because the PV mitigation technique exacerbates the NBTI degradation. The reason is that this method largely ignores the interplay between NBTI and PV and loses the opportunity to reduce the total guardband further. Since the ultimate goal of NBTI mitigation techniques is the same for different microarchitecture structures, one can expect that similar scenarios occur in other structures (e.g. issue queue, functional units). Figure 7-2 illustrates the limitation of the simple NBTI+PV mitigation technique.

Note that with a considerable performance and power overhead, it is still possible for the simple combined approach to reduce the total guardbands by a significant margin. However, as shown in Equation 7-1, guardband is not the only factor that determines the efficiency of the proposed techniques. The trade-off between reliability and performance/power should also be considered. The interaction between NBTI and PV provides the opportunity to minimize the performance penalty or power overhead without degrading the guardband enhancement obtained
by the combined technique. To summarize, simply combining NBTI with PV mitigation techniques lacks the capability to exploit the positive interaction between NBTI and PV which is beneficial to achieve either a lower guardband or less performance penalty and power overhead.

**PV-Aware NBTI Mitigation for Multi-Ported Based Microarchitecture Structures**

In this Section, I present the proposed techniques in light of register file (RF) design since the RF is a representative multi-ported microarchitecture structure. In a multi-ported RF, the RF delay is dominated by the read access time since write access time is not as delay critical as read access time [129]. In this study, I focus on RF read access and leave write access as future work. Figure 7-3 presents the 2-read port RF with detailed read port design. Only one bit cell is shown in this Figure due to space limitations. As it shows, a read port includes two wordline (the inverter) and two bitline transistors. The read access time consists of the wordline charge delay and the bitline discharge delay. Variation of the four transistors will cause a difference in the read access time of each read port. It will further affect the RF frequency, which is determined by the slowest read access time. Therefore, the effect of PV and NBTI on the read port should be accounted for by guardband estimation.

When a read port is selected to perform the read operation (e.g. read port A in Figure 7-3), the decoder will trigger the wordline associated with that port. This causes a negative voltage to be set at the PMOS gate in the inverter and triggers the NBTI degradation. On the other hand, if the port is not selected (e.g. port B in Figure 7-3), the positive voltage is set at the PMOS gate, putting that PMOS under the recovery mode. As can be seen, the port is under stress mode whenever it is enabled for read operation. Therefore, reducing the port utilization can help mitigate NBTI degradation.

Based on the above observation, I propose microarchitecture optimization 1 (O1) which assigns higher utilization to the ports with shorter read access times. By doing so, the ports with
longer read access times suffer much less NBTI degradation since their utilization decreases. As can be seen, O1 leverages the interaction between NBTI and PV by migrating more NBTI degradation to the ports with low $V_{th}$ (due to PV). Therefore, it minimizes the case of $+\Delta V_{th}$ & high $-\Delta V_{th}$ and efficiently reduces the NBTI guardband requirement. Since VL has been proved as an efficient PV mitigation method [118], I use VL technique in O1 to reduce the PV guardband.

The read ports are partitioned into fast/slow ports. In 45nm processing technology, the fastest 60% to 80% of ports can be classified as fast ports and correspondingly, the slowest port in the slow ports requires 1.16 to 1.22 cycle time to complete a read access [118]. Since they are assigned two cycles for the read operation, at least 78% of the cycle time can be used to tolerate the extra delay caused by NBTI degradation. Therefore, aggressively using the slow ports will not affect the VL frequency nor, as a consequence, the required guardband. Note that the access time also varies among fast ports and there is a fraction of fast ports with short access times which allow them to be continuously utilized (their PMOS are under the stress mode) without contributing to the NBTI guardband. I define them as absolute fast ports (AFPs). The remaining fast ports are called possible fast ports (PFPs) because the NBTI degradation on them likely leads to a time violation and contributes to the NBTI guardband. I estimated the read port speed of each RF entry across 400 chips under the impact of PV and observed that on average the fastest 36% read ports in a chip can be classified as AFP since they are at least 15% faster than the VL cycle time. One may notice that even using AFP we may still eventually fail to meet the time specification since NBTI degradation can cause as much as 20% frequency loss during the targeted lifetime period [127]. The PFP still needs to be used in case there is no available AFP. Meanwhile, using PFP lowers the threshold for AFP classification and increases the fraction of
ports that can be included in the AFP category. As a result, the overall guardband requirement should consider the wear-out of both PFP and AFP and is determined by the maximum of the two. Migrating RF port utilization from PFP to AFP and slow ports can greatly reduce the guardband requirement. To better understand the proposed technique, I present cycle time variation under the impact of NBTI and PV in Figure 7-4. Figure 7-4 (a) shows the baseline case and the optimized scenario is shown in Figure 7-4(b). In both cases, the read ports are arranged based on their access delay. In the baseline case, the initial cycle time is determined by the longest port delay due to the PV. Generally, NBTI degrades the ports evenly and the final cycle time is an accumulated effect of the worst case in PV and NBTI. On the other hand, with O1, the initial cycle time is greatly improved by VL; the read ports are partitioned into AFP, PFP and slow ports based on their delay and only PFP are vulnerable to NBTI effects. Moreover, NBTI degrades ports unevenly based on their category under the control of O1. Therefore, the cycle time is efficiently reduced compared to the baseline case. The description above mainly focuses on the combinational circuits in RF since it is crucial to the RF frequency. The inversion method proposed in [127] is applied to the SRAM based RF entries for NBTI recovery.

To implement O1, a key issue is the port utilization assignment. In the proposed scheme, PS is applied to switch from PFP to either AFP or slow ports whenever possible, occurring once the instruction is dispatched into the issue queue (IQ). Since instructions have to stay in the IQ for wakeup and selection, the port information checking and switching can be performed simultaneously without affecting the performance. When the IPC is low, switching from PFP to slow port occurs. The amount of required issue bandwidth is usually low during the low IPC phase and pipe stalls caused by the slow port will cause few issue stalls in the following cycles and hence the impact on performance is small. Intuitively, to avoid the large number of pipe
stalls, one needs to limit the number of instructions using slow ports for RF reading. I found that it is unnecessary to do so since there are only about 20% slow ports, the probability that all instructions will be issued on the same cycle, causing pipeline stalls, is low. When the IPC is high, O1 checks the possibility of switching from PFP to AFP. If it cannot be performed and the use of slow port is unavoidable, O1 will try to use a slow port for the other operand read. Because a pipe stall will occur, the performance impact is the same no matter if only one or both of the read ports are slow. However, the NBTI effect is different when one PFP and one slow port are used compared with two slow ports being used. Figure 7-5 shows the pseudo code of PS in O1. The IPC is updated every 100 cycles and an IPC of 1 is used as a threshold between high and low performance phases. Figure 7-6 shows an example of port switching in O1. The port information is attached to each register file entry and the operand in each instruction is originally assigned a read port. The detailed operations are shown when a PS occurs for a given instruction. The implementation of port information profiling and reading, and the hardware support for port switching can be found in [118]. As discussed in [118], VL+PS results in 2% area overhead, O1 introduces extra 1% area overhead to record the port information.

Note that each read port is assigned to a decoder for the port activation. The port is linked to a specific decode line in the decoder. Since the read critical path delay includes the decode delay [118] as well, the NBTI effect caused by port utilization on the decoder cannot be ignored. For illustration, we consider the 2-to-4 decoder in Figure 7-7. The decode line contains an inverter, a NOR gate, and a NAND gate which also have PMOS transistors. In order to understand the input of each gate for NBTI degradation analysis, a truth table is included in Figure 7-7. An output of “0” in D0–D3 causes the port connected to the decode line to be activated for a read operation. In addition, the detailed circuit of NOR and NAND gates are
presented to illustrate each PMOS transistor’s stress or recovery mode depending on the two inputs. I show an example where both of the inputs to the gate are “0”. As can be seen, the input “0” stresses the PMOS gate and the input “1” will recover the PMOS. As the truth table shows, when a port is activated, its corresponding decode line will have two “0” inputs in the NOR gate and two “1” inputs in the NAND gate. Correspondingly, the two PMOS transistors in the NOR gate are under stress mode while those in the NAND gate are under recovery mode. When a port is deactivated, there are three input combinations to the NOR gate, which result in either one of the PMOS being under recovery or two of them being under recovery. Additionally, the two PMOS transistors in NAND are under stress mode. Approaches such as resizing transistors [120] can be used to tolerate the NBTI degradation on the inverter, which is not private to a specific decode line. Generally, half of the PMOS transistors in the decode line are under stress mode and the remaining are under recovery mode whenever the port connected to the line is enabled or disabled. In another words, O1 does not affect the amount of NBTI degradation stressed on the decode line. The idea of inserting input vectors [127] when the decoder is idle is used to recover NBTI degradation, solving the uneven degradation problem in the decoder line.

**PV-Aware NBTI Mitigation for Combinational Blocks**

In this section, I propose PV-aware NBTI tolerant schemes that target microprocessor combinational blocks. I illustrate the design on the functional units.

As described in Background Section, the NBTI recovery in a functional unit can be performed whenever the functional unit is idle. A longer idle time provides more opportunity for NBTI recovery [127], resulting in reduced NBTI guardband. In high performance 64-bit microprocessors, many operand values in the applications do not require the full 64-bit width. These operands are referred to as narrow-width values. When there is an instruction whose operands are narrow-width values, the instruction requires an add operation and the two values
only occupy 16 bits. 1/4 of the 64-bit functional unit will be devoted to the instruction’s computation and the remaining 3/4 of the unit can stay in idle mode, providing opportunities for NBTI recovery. As can be seen, narrow-width values can help exploit idle time within a functional unit for NBTI recovery. Previous studies show that there are a large number of narrow-width operations in general purpose applications. For example, in SPEC 2000 INT benchmarks, about 50% of the instructions contain operands no wider than 16 bits. In our study, a 64-bit functional unit is partitioned into four segments with granularities of 16 bits. Each segment can complete 16-bit executions independently. For normal-width values, which are wider than 16 bits, all four segments are involved in computation.

In order to achieve high performance, the combinational blocks in functional units are either pipelined or parallelized. Take the carry look-ahead adder (CLA) as an example. Instead of waiting for the carry to ripple through all the previous stages to find its proper value, as in a ripple carry adder (RCA), the CLA calculates the dependence of each carry-out bit on the first carry-in bit, and parallelizes the carry-out bit computation. Therefore, the add operation in CLA is much faster than in RCA. The frequency of CLA is determined by the longest carry-out bit computation. The disadvantage of CLA is the rapidly increasing complexity as the number of bits increases. A multi-level CLA is proposed to create a larger adder. The frequency of a multi-level CLA is determined by the carry-out computation delay across all the levels. For instance, a 64-bit adder can be built upon 4 parallelized 16-bit CLAs, which match the segment partition introduced above. The 64-bit CLA (partitioned as 4 segments) delay is dominated by the carry-out computation delay in the 16-bit CLAs. The case is similar for other pipelined or parallelized units. As can be seen, the functional units’ frequency is highly related to the critical path delay in each pipelined stage or parallelized block, which is the partitioned segment in our study.
Due to the effect of PV, the critical path delay varies in each segment. The narrow-width operations should not be assigned randomly to the segment without considering the interaction between NBTI and PV. For example, the benefit of narrow-width operations for NBTI guardband reduction will be nullified if the operation is always performed on the segment with the longest delay, which results in more $+\Delta V_{th}$ & $high\_\Delta V_{th}$ cases. Even though other segments achieve high NBTI mitigation, it is equivalent to the case without narrow-width detection since the guardband is determined by the worst-case delay. In this study, I propose optimization technique 2 (O2) which steers the narrow-width operation to the fastest segment. In general, a functional unit is more resilient to PV than RF because its critical path is longer than that in RF and the delay difference among the segments is usually smaller than 20% [118]. This differs from the AFP in RF since an absolute fast segment is usually nonexistent. The initial fastest segment will become the bottleneck for the guardband reduction if it keeps being utilized. An online detection of the aggregated effect of NBTI and PV is required to guide migration of the narrow-width operations to the current fastest segment. IDDQ, which describes the standby leakage current in the circuit, can be applied to detect the effect. IDDQ is originally used for testing manufacturing faults [132]. The IDDQ values can demonstrate the underlying parameter variations [133]. Recently, [142] discovered that IDDQ can be applied in NBTI degradation detection as well because the leakage current decreases exponentially as $V_{th}$ increases in transistors. Therefore, IDDQ has the capability to capture both the static and dynamic variations in $V_{th}$. In this study, the segment with the highest IDDQ is the fastest one and is selected for the narrow-width value operation.

Figure 7-8 shows the hardware implementation to support O2. The narrow-width value detection occurs after the result is computed. The 48 most significant bits are checked, in
parallel, to determine if they are all 1’s (one-detector) or 0’s (zero-detector) – indicating that the operand is a narrow-width value. One bit, the narrow width record bit (NWR), is added into the RF entry to record whether the value is narrow width. When the two operands are read-out from the RF and written into the latch, the NWR is checked to determine whether a narrow-width operation can be performed. If it is narrow width, the highest 48 bits will not be latched and will be written directly into the result lines. For each operand, 4 MUXs are added between the latch and the four segments and are used to select an input value to the segment between the NBTI recovery patterns (shown as the special input in Figure 7-8) and the real value (shown as A or B in Figure 7-8). Therefore, a total of 8 MUXs are used for the two operands. If this is a narrow-width operation, 4 copies of the 16-bit value (a total of 8 copies for the two operands) will be sent to the MUXs. Otherwise, the normal value is used as the input. It is possible that the 16-bit operation causes an overflow. In this case, 4 carry-out lines are added in the output. In O2, the IDDQ testing is performed in each segment periodically, the testing current is sent to the 4-input comparator. The comparison output will determine which segment should be selected for the narrow-width operation and its two inputs will be the 16-bit values. Other segments will be inserted with the recovery vector. Another 4 MUXs are added at the output of the comparator before the comparison result is sent to those 8 MUXs for input selection. Because the comparison output should be masked if the current operation is not narrow-width, all of the input should be the real value instead of the NBTI recovery vector. The signal “select the real value” will be multiplexed with the comparison output and the signal “narrow-width operation” determines which signal will be sent out to the 8 MUXs. Similarly, the signal is sent to the output of each segment to decide whose computation result is valid for launching into the result line. The circuit of IDDQ testing, which mirrors the circuit IDDQ (“in” in Figure 7-8) to $M_n$ through
M_{in} is also shown. The analog voltage signal (“out” in Figure 7-8) reflects the changes in circuit IDDQ. Note that the IDDQ testing and comparator are not in the critical path and they do not introduce any extra delay in the cycle time. As shown in Figure 7-8, O2 only introduces the MUX and zero detection into the critical path, considering the comparatively long execution path in the functional unit, their effects to the cycle time is negligible. Moreover, their area and power overhead is around 1%.

**PV-Aware NBTI Mitigation for Storage Cell Based Structures**

In this section, the PV-ware NBTI mitigation technique is proposed for cache, which is the representative storage cell-based structures. PV exhibits both random and systematic effects. Due to systematic effects, transistors share similar parameters with other (e.g. nearby) transistors. These transistor groups define an area in which the transistors exhibit similar behavior. Since the parameter variation between two transistors is larger as their adjacency distance increases [143], transistors which are far away from this area will exhibit different behavior. If they share another parameter with transistors around them, those transistors can be classified into another area.

Figure 7-9 shows the $V_{th}$ variation map for a cache. As can be seen, the $V_{th}$ variation in cache is not entirely random. Since the cache occupies a large portion of the chip area, transistor $V_{th}$ can be generally high/low in some areas of the cache. Areas with similar $V_{th}$ can be easily found. For other structures, such as RF and functional units, which occupy a small area of the chip, the critical path variation is mainly caused by the random effect since the similar systematic effect performs across the entire structure. Therefore, although the critical paths in the structure are very close, they still vary in the path delay.

It is well known that body biasing (BB) is an efficient method for PV mitigation. However, it must be applied at the structure level and a finer granularity is not achievable with BB technology [119]. Usually, a cache is assigned one BB generator and a uniform voltage biasing is
applied in all areas, whether they have high or low $V_{th}$. The amount of BB applied is determined by the worst case across the entire cache. [127] proposes a NBTI recovery mechanism for cache structures by invalidating 50% of the cache lines and uses them to store the inverted values. However, keeping half of the cache invalidated increases the cache miss rate and degrades performance, especially on applications that have high cache utilization. When combining the BB technology [119] with the NBTI recovery approach [127], the guardband is reduced significantly. Note that areas with low initial $V_{th}$ can tolerate more NBTI degradation and, as long as the final $V_{th}$ does not exceed that in the areas with high initial $V_{th}$ due to PV, the strict cache line inversion percentage (e.g. 50%) can be appropriately relaxed in those areas. Doing so reduced the number of invalidated cache lines, which decreases the cache miss rate and performance loss, leading to an improvement in the technique efficiency to NBTI and PV mitigation in terms of performance, power, and chip lifetime.

Based on the above observation, I propose O3 to take advantage of the systematic effect of PV in guardband reduction while maintaining performance. I apply adaptive body biasing (ABB) in O3 to mitigate the PV effect. First, O3 partitions the cache into several areas according to the similarity of transistors’ $V_{th}$. Each area has its individual inversion percentage (areas with lower $V_{th}$ will be assigned a lower inversion percentage, corresponding to a smaller number of invalidated and inverted cache lines). The percentage is estimated based on the difference between the highest $V_{th}$ in the cache and that in the area. Similar to the proposal in [127], the valid/state bits are used to indicate whether the cache line is valid and non-inverted, or invalid and inverted. A counter is used in each area to count the number of inverted cache lines. Once it is below the pre-defined threshold, one LRU cache line is invalidated and written with the
inverted value. Since different cache ways are implemented close to one another, the PV exhibits a stronger systematic effect in the horizontal direction than in the vertical direction [144].

The cache area is partitioned at the set level. However, the partition granularity should be considered. If it is too small, there are fewer cache lines being chosen from the area for inversion and it becomes more difficult to match the required the inversion percentages to a concrete inversion number. In addition, a large number of counters are required for the inversion percentage control, which causes a higher area overhead. On the other hand, if the granularity is too large, the systematic effect cannot be efficiently exploited. For example, when the granularity is set as the entire cache, O3 will be the same as combining BB with the technique proposed in [127]. I perform the sensitivity analysis in Evaluation Section and choose the granularity as 8 sets. Figure 7-10 describes the idea of O3 in the 4-way L1 cache. The cache line with gray color represents the invalidated and inverted lines.

Evaluation

In this Section, I evaluate the three techniques proposed in this Chapter.

In this study, I model the dynamic NBTI degradation in $V_{th}$ by applying the reaction-diffusion (RD) model proposed in [145], the PMOS stress and recovery cycles are obtained via the microarchitectural simulator, and the signal possibility is computed and inserted into the model to determine the shift in $V_{th}$ due to NBTI. I used the same architecture level evaluation methodology described in Chapter 6 for the techniques evaluation. Since both NBTI and PV effects are addressed in our study, I extend the $NBTI_{efficiency}$ metric to $NBTI&PV_{efficiency}$ (Equation 7-2), which quantifies the technique efficiency to both NBTI and PV. Correspondingly, the NBTI+PV guardband is named as $NBTI&PV_{guardband}$.

$$NBTI & PV_{efficiency} = (Delay \cdot (1 + NBTI & PV_{guardband})^3 \cdot TDP$$ (7-2)
Effectiveness of O1

O1 is compared with the baseline case without any optimization. I also compare the technique combining 70% VL with port switching (PS) and the NBTI mitigation technique, which inserts a special input vector (SIV) in the idle time (defined as VL+PS+SIV). Figure 7-11 (a)-(c) presents the CPI, NBTI guardband, and NBTI&PV_efficiency of the three cases in RF. The CPI and NBTI guardband are normalized to the baseline case. The TDP of VL+PS+SIV and O1 is 1.02 and 1.03 respectively due to the area overhead. As shown in Figure 7-11 (a), CPI increases in both of the NBTI&PV mitigation techniques because the use of slow read ports cannot be eliminated. When they are selected for RF read operation, pipe stalls occur and degrade the performance. However, the performance penalty is negligible in some applications (e.g. equake, mcf) because they are running in low IPC phases most of the time and the pipe stalls are tolerated by the low bandwidth requirement. One may notice that O1 increases the CPI by 2% compared to VL+PS+SIV. This happens because slow ports are intentionally chosen for read operations when the IPC is low in order to reduce the PFP utilization. When the IPC information obtained from the last phase generates an incorrect prediction, a slow port is selected by mistake, which causes performance loss. Even though O1 slightly increases the CPI, it gains a significant NBTI guardband reduction. As Figure 7-11 (b) shows, on average, O1 reduces NBTI guardband by 35% and 36% compared to the baseline case and VL+PS+SIV, respectively. Interestingly, the VL+PS+SIV exacerbates the NBTI degradation compared to the baseline case because fast ports are used aggressively in VL+PS+SIV and they must accept the utilization migrating from the slow ports. Meanwhile, the SIV does not help reduce the NBTI degradation in read ports since the port switches to the recovery mode automatically when it is free, additionally, the positive effect caused by SIV on the decoder line is not noticeable enough to combat the negative effect. I forgo a presentation of NBTI&PV guardband, which is equal to the
sum of NBTI and PV guardband. In the baseline case, on average across all the simulated chips, the PV guardband is set to be 0.3, when applying VL technique, improving the frequency by 20% and the PV guardband reduces to 0.1. Figure 7-11 (c) proves that O1 reduces NBTI&PV\_efficiency greatly. It reduces the efficiency as high as 1.00 compared to the baseline case, which implies it improves the efficiency 100% since the best technique has the efficiency of 1 (no PV and NBTI effect). Moreover, it exhibits much stronger ability than VL+PS+SIV in solving NBTI and PV because it achieves 30% improvement in NBTI&PV\_efficiency.

**Effectiveness of O2**

O2 is compared with the baseline case, the NBTI mitigation technique SIV, the technique which applies SIV and takes narrow-width operation into consideration (defined as SIV+NW). Since the VL technique [118] is orthogonal to the above methodologies, the discussion on their combination to VL is skipped. Figure 7-12 (a)-(b) presents the NBTI&PV\_guardband, which is normalized to the baseline case, and the efficiency of the four cases in Integer ALU. CPI is not shown in the figure since it has a negligible effect on performance. The TDP in SIV+NW and O2 is 1.01 and 1 in SIV and the baseline case. I show the results of IntALU because most of the narrow-width operations are integer arithmetic and logic operations. It is not fair to judge the efficiency of the techniques in functional units (e.g. FPU) with few narrow-width operations. As Figure 7-12 shows, compared to the baseline case, on average across all the benchmarks, SIV reduces the guardband by 28%. It gains less reduction than that reported in [127] (63%) because I focus on the IntALU, which performs both arithmetic and logic operations and has less idle time than the adder studied in [127]. O2 exhibits much stronger capability in guardband reduction, which are 55% and 59% in INT and FP benchmarks respectively and, as a result, improves the efficiency by 73% and 76% in the two benchmark categories. Compared to SIV+NW, which blindly assigns the narrow-width operations inside the unit, O2 decreases the
guardband 15% and 12% in INT and FP benchmarks. This contributes to 18% and 13% efficiency improvement compared with SIV+NW.

**Effectiveness of O3**

Figure 7-13 (a)-(b) shows the normalized CPI and NBTI&PV_{efficiency} generated by the baseline case, the technique applying ABB with cache line inversion (CLI) (define as ABB+CLI), and O3. Since the NBTI and PV problem can be easily solved in the L2 cache by implementing periodical inversion [146], I focus the study on L1 data cache. Note that HotLeakage is used to evaluate the power overhead caused by ABB. As can be seen, ABB+CLI has negligible CPI impact on some benchmarks (e.g. *lucas, mcf*) because of frequent L2 cache misses: a L1 miss latency caused by the cache line inversion will be covered by the L2 miss which occurs simultaneously. However, it degrades the performance significantly on benchmarks with low L2 cache miss rates. O3 solves this problem since it efficiently utilizes the L1 resources. For example, O3 improves the performance by 19% in *eon* and 8% in *mesa*. As shown in Figure 7-13 (a), O3 obtains similar CPI results as the baseline case. It improves the NBTI&PV efficiency 13% compared to ABB+CLI. Figure 7-14 describes the NBTI&PV_{efficiency} obtained by O3 as the granularity varies from a single set to the entire cache. I perform the analysis on benchmarks (e.g. *eon, vpr*) which are sensitive to ABB+CLI technique. As expected, the performance loss is high when the granularity is extremely small or large. An 8-set granularity achieves the best efficiency, it is chosen in the O3 implementation but requires an extra cache line and 16 counters, which results in 1% additional area overhead.

**NBTI&PV Efficiency Regarding to the Entire Chip**

In order to evaluate the effectiveness of the three proposed techniques on the entire chip, I compute the *NBTI&PV_{efficiency}* of the processor following the equations proposed in [127]; based on each structure’s Delay, *NBTI&PV_{guardband}* and TDP generated by our techniques.
On average, I obtain an efficiency of 2.20 for the entire chip. In the baseline case without any optimization, the chip NBTI&PV_efficiency goes up to 3.375. As can be seen, the proposed techniques improve the efficiency by 117%. The effectiveness of simply combining PV and NBTI mitigation techniques is evaluated for the comparison, its NBTI&PV_efficiency is 2.41, and my technique outperforms this technique by 21%.

**Related Work**

There have been several studies on NBTI modeling and mitigation at both the circuit and microarchitectural levels. The Reaction-diffusion (R-D) model has been widely used to model the NBTI degradation and recovery effect [147, 148]. [145] recently considered temperature variation in the NBTI model. The impact of NBTI on the performance of combinational circuits is investigated in [149], which shows that NBTI degradation is sensitive to the input patterns and the stress time. In addition, the NBTI effect on SRAM array is modeled and studied in [150], where it is shown that the read stability degrades due to NBTI and that the degradation is exacerbated in the presence of PV. To mitigate combinational circuit aging under NBTI, adaptive body biasing (ABB) is applied in NBTI resilient circuits [151]. [157] proposes to identify the critical gates that are most important for timing degradation and protects them from NBTI. To improve the storage cell reliability under NBTI, [152] proposes a new memory cell design consisting of a number of NAND gates instead of inverters to reduce the average degradation on each PMOS. Periodic inversion [146] is proposed to flip the contents of all cells periodically, keeping the balance between “0” and “1” in the cell and is an efficient way to mitigate NBTI in storage cells, but the extra flipping delay in the critical path causes 10% frequency loss. [153] improves the cache reliability under NBTI. It proposes proactive use of microarchitectural redundancy, in which the two components operate either in active mode or in recovery mode, periodically transitioning between the two modes according to a recovery
schedule. The combined effect of PV and NBTI has been modeled and analyzed in [154, 155].
Moreover, [158] proposes online PV and NBTI detection in logic circuits and applies ABB to
tolerate the $V_{th}$ variations. [156] proposed a technique called “Razor” to tune the supply voltage
by monitoring the error rate caused by PV and NBTI during circuit operation, thereby
eliminating the need for voltage margins. “Razor” mainly targets combinational logics. In our
study, we target the mitigation of NBTI and PV effect in both combinational circuits and storage
cell based structures with desirable trade-offs among performance, reliability, and power. To my
knowledge, this is the first work taking advantage of the interplay between PV and NBTI to
efficiently address the variation problem caused by NBTI and PV.
Consider the interaction between NBTI and PV

- **Optimized guardband**
- **Conservative guardband**

Figure 7-1. Different guardband settings for tolerating NBTI

Consider the interaction between NBTI and PV

- **Optimized guardband**
- **Conservative guardband**

Figure 7-2. The limitation of simply combining NBTI and PV mitigation techniques

Figure 7-3. 2-Read port register files with detailed read port design
Figure 7-4. Cycle time under NBTI and PV effects. A) Baseline case without optimization. B) O1.

1. Every cycle
2. {
3.   IPC update every 100 cycles();
4.   IF (last interval IPC <=1) THEN
5.     {
6.       switch from PFP to slow ports;
7.     }
8. ELSE
9.   {
10.    IF (AFP is available for switch) THEN
11.      {
12.         switch from PFP or slow ports to AFP;
13.      }
14.    ELSE IF (slow ports is unavoidable) THEN
15.      {
16.         switch from PFP to slow ports;
17.      }
18.    ELSE
19.       no port switching;
20.  }
21. }
1. ADD R5, R1, R3
2. AND R7, R4, R5
3. SUB R6, R2, R6

1. If(last_interval_IPC<=1)
   PS(R1, R3);
   /*switch from PFP to slow ports*/
   else
   no PS;
2. PS(R4, R5);
   /*switch from PFP to AFP
   when AFP is vailable for switch*/
3. PS(R2, R6);
   /*even it is in high performance
   phase, switch from PFP to slow ports
   when slow port is unavoidable*/

Figure 7-6. Examples of PS in O1

Figure 7-7. An example of 2-to-4 decoder
Figure 7-8. O2 circuit design

Figure 7-9. $V_{th}$ (in mV) variation map for a cache
Figure 7-10. Fundamental idea of O3 in the 4-way L1 cache
Figure 7-11. The effectiveness of O1 in RF. A) Normalized CPI. B) Normalized NBTI guardband. C) Normalized \textit{NBTI&PV\_efficiency}. 

179
Figure 7-12. The effectiveness of O2 in IntALU. A) Normalized $NBTI&PV_{guardband}$. B) Normalized $NBTI&PV_{efficiency}$. 
Figure 7-13. The effectiveness of O3 in L1 cache. A) Normalized CPI. B) Normalized NBTI&PV_efficiency.

Figure 7-14. NBTI&PV_efficiency with various granularities
The trend towards multi-/many-core processor design has made a scalable and high-bandwidth on-chip communication fabric that connects these cores critically important. The packet-switched network-on-chip (NoC) [159] is emerging as the pervasive design paradigm for multi-core communication fabrics. With the continuous down-scaling of CMOS processing technologies, reliability and variability (e.g. process variation (PV), negative biased temperature instability (NBTI)) is becoming a primary target in NoC design [160]. Although PV and NBTI can be addressed at the device- or circuit- levels, such solutions are costly in terms of area and power and exhibit poor scalability. Several architecture and system level techniques [111-115, 118, 127, 153, 161, 162] have been proposed to mitigate the NBTI and PV effects on processor operations and achieve a lower guardband. As a result, the frequency loss and power consumption reserved for the guardband can be reduced. These techniques focus on processor cores and memory hierarchy and largely ignoring the emerging NoC architectures whose design is significantly different from processor architectures. Ignoring the reliability of the NoC can turn it into a potential reliability bottleneck of multi-/many-core architectures. In NoC architectures, ultra-low latency designs are desired since shared-memory workloads are highly sensitive to the interconnect latency [163]. In addition, power management is also critical in a NoC, the literature [164, 165, 166] reports the interconnect power consumption at approximately 30% to 40% of total chip power consumption. Since NBTI and PV affect both NoC delay and power, it is imperative to address these challenges at the NoC architecture design stage to ensure its efficiency as the underlying CMOS fabrication technologies continue to scale [167].

As introduced in Chapter 7, PMOS transistor wear-out caused by NBTI is aggravated in the presence of process variation. The guardband considering both NBTI and PV will be an
additive result from the guardbands for the two effects respectively. In the upcoming nano-scale transistor design era, NoC design techniques that can effectively address the combined PV and NBTI effect are needed. In this chapter, I target NoC architectures and propose novel circuit, router microarchitecture, and inter-router reliability enhancements in the presence of process variation and NBTI, while achieving attractive trade-offs among NoC performance, power, guardbands for reliability, and area overhead. Instead of using a flat/centralized method (as [161]), I propose three techniques to hierarchically mitigate PV and NBTI effects in NoC at both inter-router (high) and intra-router (low) levels.

**Packet-Switched NoC Router Microarchitecture**

Peh and Dally [168] proposed the canonical NoC virtual channel router microarchitecture. The router is input-queued and has P-input and P-output ports, while P is usually set as five. Four of them connect to the four cardinal directions (North, East, West, and South) and one connects to the local processing element. The router microarchitecture is composed of the following key elements: virtual channel (VC) FIFOs, routing computation unit, VC allocation logic, crossbar allocation logic, and the crossbar itself. There are four stages in the router pipeline. When a flit enters the router through one of the input ports, it is stored in the VC buffer that has been reserved by the upstream node. If the flit is a header (the first flit of a new packet), it proceeds to the route computation (RC) stage. The routing computation unit determines the output port for this new packet. The following pipeline stage is VC allocation (VA), during which the VC allocation logic attempts to assign a free VC in the next hop to the header flit. In the following cycle, if VC allocation is successful, the flit enters the switch allocation (XA) stage, during which it competes with other flits from the router for the output port. The data and tail flits belonging to the same packet as the header flit can skip the RC and VA stages and proceed to XA. Once the link between the input port and its output port is built up, the flit enters the
crossbar traversal (XB) stage and is sent to the next hop. To reduce router latency and improve performance, researchers have proposed a two-stage router pipeline, which incorporates route look-ahead [169] and path speculation [168]. The first technique performs routing one hop in advance. The second technique speculates that the waiting packet will successfully obtain the output VC from the VC allocation logic and parallelizes the switch allocation and virtual channel allocation stages. If both allocation requests are granted, the latency of switch arbitration is absorbed. The router pipeline can be further reduced to a single stage [170] by performing additional speculation, but will incur more mis-speculations at high loads and cause one-cycle penalty.

Prior studies [171, 172] have also proposed adaptive routing for NoC design. Compared with dimension-order routing (DOR), the adaptive routing schemes can achieve better fault tolerance and congestion avoidance capability. In this study, we consider a two-stage adaptive router microarchitecture similar to that proposed by Kim et al. [172]. I opt to use this design since both reliability and performance are considered in NoC design. Figure 8-1 shows the baseline adaptive router microarchitecture and pipeline. Adaptive routing requires extra logic to collect congestion statistics, which is used to pre-select the preferred output port (the port with the least congestion) for each packet one cycle in advance [172].

A Hierarchical Design of NoC That Mitigates the Effects of NBTI and PV

In this section, I propose a hierarchical design of NoC that efficiently mitigates the impact of NBTI and PV while leveraging the benign interaction between them. The hierarchical design performs at both intra-router (low) and inter-router (high) levels.

Intra-Router Level NoC PV and NBTI Optimization

The key components of the pipelined router can be classified into combinational logic structures (e.g. VC allocation logic, switch allocation unit) and storage-cell structures (e.g.
virtual channels). My intra-router NBTI and PV optimization schemes target both types of structures.

**NBTI and PV optimization for NoC combinational-logic structures**

Prior studies [168, 173] have shown that, when compared to other router pipeline stages (e.g. XB and LT), the delay in VA and XA stages largely determines the frequency of canonical router microarchitecture. I thus focus our study on virtual channel allocation logic and switch allocation unit as the representative combinational-logic structure in a pipelined router.

Figure 8-2(a) illustrates the detailed circuit design of the virtual channels and Figure 8-2(b) shows a zoom-in view of the VA logic of the two-stage adaptive router [172]. A packet in a minimally routed 2D mesh can only proceed to two of the four quadrants (i.e. NE, NW, SE, and SW). The RC units determine the quadrant to which the packet should travel based on its destination direction, and the pre-selection function selects one output port for the quadrant depending on the congestion information. The VA and XA are then performed for the selected output port. In order to support adaptive routing, the VCs are partitioned into four sets. Each VC set is assigned to one quadrant and is used to collect the flits routed to this quadrant. There are three groups of VCs within each set to accept flits from the local processing element (PE) as well as from the two other directions that are not included in this quadrant. For example, the VC set assigned to NW quadrant can only accept flits from PE and E and S directions. Note that a packet whose destination is the local PE will be injected into the PE directly without going through the VA, XA, and XB stages. In the adaptive router, the VA stage operates in two steps: the first step assigns a free output VC (VC at the downstream nodes) to each request from VCs in the four sets. It requires one arbiter per VC in the sets and a total number of $12v$ arbiters is needed (i.e. there are four sets and each set has three groups of VC, $v$ stands for the number of virtual channels in a group). The second step produces a winner for each output VC among all
the competing VCs in the sets. Correspondingly, one arbiter is required for each output VC. The total number of arbiters required for the second step is \(8v\). Since VA logic consists of multiple arbiters in parallel, the VA delay is sensitive to the PV effect \([173]\). Therefore, mitigating the NBTI and PV effects in the VA stage will allow a lower guardband and directly boost the router frequency.

As shown in Figure 8-2 (b), each VC entry has a private arbiter in the first VA step. Since there are multiple parallel arbiters in the first VA step, they exhibit various delays due to the PV effect and the slowest critical path under the impact of PV determines the delay of the first VA step. As a NBTI recovery method, the special input values \([127]\) can be inserted into the arbiter when there is no request signal sending out from the associated VC entry (i.e. the arbiter is idle). However, it is possible that the slowest arbiter is frequently utilized, losing the opportunity for NBTI recovery. This will lead to a longer VA delay since the NBTI degradation on the slowest arbiter is not efficiently recovered even when the NBTI optimization technique is applied. In this study, I propose NBTI and PV mitigation technique 1 at the VA stage (VA_M1) to assign a higher utilization to the faster arbiters and insert the special input values to idle arbiters. By doing so, the slowest arbiters obtain more idle time to perform the NBTI recovery. As can be seen, the fast arbiters absorb more NBTI degradation while protecting the slowest arbiters from the impact of NBTI at the most degree. Therefore, the guardband decreases. Due to its overhead, the adaptive body biasing (ABB) technique \([19]\) is not suitable for mitigating PV effect at fine granularity. In the study, I apply the ABB technique in a chip-wide manner to tolerate the PV effect across the entire chip.

As mentioned earlier, the first VA step selects a free output VC entry from a specific VC group at the downstream node. Since arbiter utilization depends on the requests from the
associated VC, instead of blocking the requests to the arbiter I prefer to avoid using the VC even when it is free. If there is no flit stored in the VC, no request will be sent out. In other words, the arbiter gets the opportunity to recover from NBTI degradation if its associated VC is idle. Figure 8-3 shows the circuit of the first VA step (only one VC entry is shown). As can be seen, when a flit at a local VC sends an output VC allocation request to its private arbiter, the request will be fanned out to $\nu$ AND gates ($\nu$ stands for the number of virtual channels in a group) and ANDed with each VC status (i.e. idle: “1”, or occupied: “0”) at the downstream node. The arbiter only accepts the fan-out request signals for downstream VCs whose status is idle. At the downstream node, when the VC associated with the slowest arbiter is free and there is more than one free VC in the group, VA_M1 marks it as occupied before a credit describing its status is sent back to the upstream node. Consequently, other free VCs in the same group that link to faster arbiters will be used to store the flit. In the case where there is only one free VC, which is exactly the one linked to the slowest arbiter, VA_M1 marks it as free in order to maintain the performance. Since VCs are not fully occupied during most of the router service period, VA_M1 has substantial opportunities to migrate the NBTI degradation from the slowest arbiter to faster ones. In addition, the arbiters use a least-recently-served priority scheme [170], every faster arbiter receives even NBTI degradation migrated from the slowest one. Therefore, the NBTI degradation will not accumulate on a specific fast arbiter, which can eventually become the new bottleneck for the guardband reduction.

Figure 8-4 presents the implementation of VA_M1. An IDDQ detector is attached to each arbiter to perform an online detection of the aggregated NBTI and PV effect on it. In this study, I assume the critical paths within each arbiter have similar delay due to the systematic effect of PV. Therefore, one detector per arbiter can well describe the NBTI and PV effect in the arbiter and
the arbiter with the lowest IDDQ value is the slowest one. The detector operates periodically and sends the testing current to a \( \nu \)-input NMOS analog voltage comparator [174] that determines which VC should be marked as occupied. The comparison result is written into the VC status table (a hardware component existing in the typical flow-control-based router), which reflects the availability of each VC in the router. Note that the IDDQ detector and comparator perform concurrently with the VA stage and the backward credit representing a VC status does not need to wait for the latest VC status before being sent out. Therefore, no extra delay is introduced into the router pipeline. When inserting the special input values into the arbiter, invalid grant signals will be generated. In order to block them from entering the second VA step, an invalid port number is sent to the DEMUX at the arbiter output. Note that VA includes two-step arbitration, VA_M1 will lose its efficiency if the arbitration delay variation in the first step is not critical to the VA delay. Upon successful operation in the first step, the grant signal is fanned out to the arbiters in the second step. Therefore, the delay variation in the first step cannot be tolerated by the second step – the slow arbiters in the first step will affect the entire VA delay. Moreover, VA_M1 mitigates the NBTI degradation within the first step and it does not introduce any extra NBTI effect to the second step. In order to mitigate the NBTI and PV effects in the second VA step, VA_M1 inserts the special input values when the arbiter is idle.

The switch allocation unit in the XA stage, which is another combinational-logic structure, is sensitive to the effect of PV and NBTI as well [173]. Since every flit of the packet has to experience the XA stage while only head flit requires virtual channel allocation, switch allocation unit is frequently used and suffering more NBTI degradation than virtual channel arbiters. Therefore, only focusing on the PV and NBTI mitigation at VA stage will lead to another frequency bottleneck at XA stage. As a widely used mechanism to dynamically tune the
circuit delay, Adaptive Voltage Scaling (AVS) [162] is applied to the switch allocation units in our study. By increasing the supply voltage of the combinational-logic blocks, the critical path delay can be effectively reduced. However, a higher supply voltage results in larger power consumption (higher temperature as well), which also increases transistor aging-rate (i.e. the transistors will have shorter lifetime). Generally, the XA delay is shorter than VA delay at the beginning of the service time when PV is the major effect on circuit delay [168, 173]. In this study, I propose to trigger the higher supply voltage (AVS+) when XA delay becomes the limitation to the router frequency (Ideally, it is the time that XA delay is longer than the VA delay at the end of the lifetime after VA_M1 is applied). By doing that, not only the power consumption can be significantly reduced, but also the negative effect on aging-rate is efficiently alleviated (applying AVS+ towards the end of the lifetime has little effect on aging-rate [162]). Figure 8-5 (a)-(b) show the guardband and power benefit of applying AVS+ during the lifetime. As Figure 8-5(a) shows, after integrating the AVS+ technique with VA_M1, the guardband significantly decreases compared to the case in which only VA_M1 is applied. In addition, the power overhead is reduced substantially when AVS+ is enabled later at the service time instead of at the very beginning. The combined VA_M1 and AVS+ technique is named as VA_SA_M1.

Since the AVS+ technique at XA stage is implemented simultaneously with VA_M1, one cannot recognize the final effectiveness of VA_M1 on VA delay during the service time and use it to determine the appropriate trigger time for AVS+. In order to explore a simple design, I enable AVS+ when the XA delay is longer than a pre-set threshold delay. Note that the threshold is supposed to be shorter than the final VA delay, and less power is consumed when it is closer to that ultimate delay. I set a conservative threshold based on the sensitivity analysis in the Evaluation section. Since a conservative threshold is applied, instead of monitoring each arbiter's
delay in switch allocation unit to obtain the accurate XA delay and resulting in a large area overhead, I apply one IDDQ detector to the whole unit for XA delay estimation. The IDDQ output is compared with the threshold for AVS+ activation. Note that the IDDQ detection and comparison occurs simultaneously with the router pipeline and no extra delay is introduced. To reduce the implementation complexity, two different power lines to supply the normal (1.0V) and higher (1.2V) voltage are used respectively at the chip level, and a pair of PMOS transistors is inserted at the switch allocation unit in each router to handle the voltage switching. Note that the AVS+ trigger time at the switch allocation unit per router varies according to the PV and NBTI effects on each individual router. As can be seen, the three-transistor based IDDQ detectors, the 2-to-1 MUXs, and the comparators are the major extra overhead induced by VA_SA_M1. The gate-level estimation shows that VA_SA_M1 causes around 3% area overhead to the virtual channel and switch allocation logics.

In addition, as shown in Figure 8-2, the crossbar delay is dominated by the wire delay [168]. Wires are immune to the NBTI degradation and techniques reducing the wire delay are out of the scope of this paper.

**NBTI and PV optimization for NoC storage cell based structures**

In this section, I present an optimization technique for VC buffers, which are the representative storage-cell structures in NoC. Recall that an efficient NBTI degradation mitigation approach for the storage-cell structure is to keep 50% of the bits storing the sampled inverted value. To apply this technique in our work, the inversion at flit-size granularity is performed, which is large enough to statistically summarize the NBTI degradation in all of the VCs. To implement this technique, only half of the VC capacity can be used to store the flits and the performance loss is significant on workloads that exhibit heavy traffic. Due to the systematic component of process variation (i.e. the spatial correction effect of WID), transistors usually
share similar behavior with nearby transistors. I synthesized and generated the layout of a prototype router at 45nm processing technology. The router layout is similar to those reported in previous work [163, 175]. The methodology to model the systematic and random variations on $V_{th}$ can be found in Chapter 6. Figure 8-6 presents the $V_{th}$ variation map for the router. Note that it is based on the physical layout of the router instead of the conceptual router microarchitecture shown in Figure 8-2. As shown, the distance between the transistors in the same VC set (shown by the rectangular) is much shorter than that between the transistors from different VC sets. As a result, the $V_{th}$ of transistors within each input port exhibit similar characteristics allowing VCs from the same port to be grouped into one area. Transistors from the same area are characterized by one uniform $V_{th}$, which is determined by the worst-case $V_{th}$ (i.e. the highest $V_{th}$, because it affects the minimal voltage has to be applied) in that area. Therefore, there are several areas in the router (one area per set) with various representative $V_{th}$ values. I observe that areas with low $V_{th}$ can tolerate more NBTI degradation and, as long as the final $V_{th}$ does not exceed that in the area with the highest $V_{th}$, the strict inversion percentage (i.e. 50%) can be appropriately relaxed in those areas.

The second mitigation technique proposed in this Chapter targets VC buffers (VC_M2) and assigns lower inversion percentage to areas with lower $V_{th}$. By doing so, there are more VC buffers to hold flits and mitigate the performance loss, achieving a good trade-off among reliability, performance, and power. Based on my NBTI modeling, I characterize the relation between inversion percentage and the corresponding increase of $V_{th}$. Using the above information and the statistics of the highest $V_{th}$ and the $V_{th}$ of a given area, one can compute out the inversion percentage for that area during the calibration time. Figure 8-7 (a) shows one VC set when VC_M2 is applied. In VC_M2, a recover flit is defined as the flit with inverted values, and it
does not contain valid data. To implement VC_M2, two extra buffers are added in the set. One stores the sampled inverted value. The other one is called the status buffer and records the status ("0": free, "1": occupied) of all the flit-size buffers in the VCs: if one flit-size buffer is written by a flit or a recover flit, its corresponding status bit will be set to 1. On the contrary, when the buffer is released, the status bit is reset to 0. Note that at the XA stage, the flit will not send out a traversal request if there is no free buffer in the allocated output VC. In general, a credit returns to the upstream node when there is at least one free buffer in the allocated VC at the current node. Note that the credit describes the availability of buffers in a specific VC – it is different from the VC status credit (discussed in VA_M1 techniques) which shows the availability of a VC. In VC_M2, when sending back the credit representing the buffer availability, the buffer holding the recover flit should also be marked as occupied. However, the VC status is still considered as free if the VC only contains recover flits. Since the recover flit does not belong to any packet, allocating a recover flit-occupied VC to a new packet will not cause the mix of packets into single VC, which is not allowed in the flit based flow control [168]. VC_M2 mainly focuses on the credit of buffer availability and it does not affect the credit of VC availability. Therefore, it is orthogonal to VA_M1.

As introduced in Section 2.1, in a typical router, each VC is a FIFO-based structure [168, 171] and every flit in the VC will move from the tail to the head and finally enter the crossbar. In VC_M2, the recover flit follows the same policy. When it arrives at the head of the FIFO (as shown in Figure 8-7 (a)), it will not be read out, but overwritten by the following flit, therefore, no request to the arbiters in the first VA step. As can be seen, even though a VC is re-defined as “occupied” in VA_M1 due to its link to the slowest arbiter, the buffers in this VC still can be used to hold recover flits and its private arbiter becomes idle during the period for NBTI.
recovery. In each set, there is a threshold value to describe the number of inverted flit-size buffers corresponding to the required inversion percentage. Meanwhile, a counter (IBCNT) is attached to the set to track the inversion number. Once the IBCNT is below the threshold (e.g. the recover flit arrives the head of the FIFO and is overwritten), VC_M2 will read the status buffer, and pick a free flit-size buffer to store the inverted value and maintain the inversion percentage. When the recover flit is overwritten, a buffer at the tail of the FIFO will be released simultaneously. Therefore, there is always at least one free buffer for inversion (shown in Figure 8-7 (a) to (b)). As can be seen, VC_M2 performs simultaneously with router pipeline stages and it does not increase the router latency. When implementing VC_M2 at each set of virtual channels, it introduces a status buffer with the bit size equal to the number of buffers in the set, a flit-size buffer to keep the inverted value, a 5-bit buffer for inversion threshold, a 5-bit counter for IBCNT, and simple combinational-logics. In summary, the area overhead caused by VC_M2 to the virtual channels is around 3%.

**Inter-Router Level NoC PV and NBTI Optimization**

In the NOC architecture, routers experience WID variations allowing them to support different frequencies. In this work, I assume NoCs with a single frequency domain, which is determined by the slowest router, and apply the chip-wide adaptive body biasing technique [162] to mitigate the PV effect. The routers under the PV positive effect (e.g. faster routers) can be assigned higher utilization (e.g. handling a larger number of packets) while exhibiting less NBTI degradation. By doing so, the routers under negative PV effect will process less packets, which limits the impact of NBTI. Since the NBTI degradation migrates from the slower routers to faster ones, the guardband for the chip frequency significantly decreases. Note that network congestion status should be taken into consideration when mitigating the NBTI effect at the inter-router level. For instance, to migrate the NBTI effect on slower routers, it is possible that the packets
are all routed to the faster routers. Consequently, the faster routers quickly become congested, resulting in longer network latency. Prior work [171] proposes regional-congestion-aware routing to balance the traffic load in the NoC and improve the performance. However, the NBTI and PV effects are not considered.

The third technique proposed in this paper to mitigate NBTI and PV effects achieves both reliability and congestion aware routing at the inter-router level (IR_M3). In IR_M3, each router (1) collects its NBTI, PV and congestion status and produces an aggregated statistic; (2) computes reliability and performance efficiency metrics by considering the aggregated statistics from both local and remote routers; (3) determines whether the aggregated metric or the local congestion metric should be used for the port selection. In the case when the packet stays in the VC for a long time and keeps sending the output VC request to the VA logic, the router suffers more NBTI degradation. Therefore, if the current router is the slowest one, instead of trying to route its packet to a faster but possibly more congested node, I prefer to quickly send it to a less congested node which can reduce the VA utilization in the slowest router; (4) selects a preferred output port for the packets based on the computed metrics to achieve a good trade-off between reliability and performance; and (5) propagates its aggregated statistic to its neighbor routers.

Figure 8-8 shows the implementation of IR_M3. The aggregation and propagation modules are added into the adaptive router to perform steps (1), (2) and (5). In addition, the pre-selection unit takes the computed metrics to perform step (4). Note that the above-mentioned steps perform simultaneously with the router pipeline and they do not cause any extra delay to the flit traversal.

As mentioned earlier, IDDQ can quantify both NBTI and PV effects. Since the delay in VA and XA stage mainly determines the router frequency, in each router, I reuse those IDDQ detectors deployed in VA_SA_M1 and the lowest IDDQ current is used as the reliability
estimation. A high IDDQ value represents small NBTI and PV effects. As suggested in [171], the combination of free virtual channels and crossbar demands (vc_xb) is used for congestion estimation in our study. A high vc_xb value indicates low congestion. In order to route the packets to an optimal path, the collected estimations needs to be integrated as a single statistic. Note that the remote statistics are obtained from all network directions and an aggregated statistic is produced for each network direction. It consists of two stages: first, a weighting function combines the reliability and congestion estimations together to produce an aggregated statistic for the local router. If there is no preference between reliability and performance, one can simply choose a 50-50 weight. If reliability has a higher priority, one can grant it a higher weight, and vice versa. Furthermore, one can dynamically change the weighting function based on the packet characteristics. For example, if there are a large number of redundant data in a packet, the content of the packet can be represented by a small number of bits. The VC buffers storing this packet can be used to store the inverted value for NBTI mitigation. Therefore, reliability statistic will not be considered in its next hop routing since the packet will not exacerbate the reliability of the downstream router and a router with high NBTI degradation can still accept it. In that case, a 100-0 weighting function can be applied between performance and reliability. A sensitivity analysis is performed to explore various weighting functions in the Evaluation section. In the second stage, the router assigns equal weight to the local and remote aggregated statistics to compute the combined reliability and performance metrics. In the pre-selection function, the metrics from the two directions of each quadrant are compared and the neighboring node with higher value (i.e. representing a better reliability and performance tradeoff) is chosen as the next hop. The propagation is performed on each direction as well – a more detailed discussion can be found in [171]. The aggregation/propagation module mainly
consists of two adders. Our gate-level estimation shows that the extra area overhead caused by IR_M3 is around 2% to each router.

**Evaluation**

**Experimental Methodologies**

In this study, I use Garnet [176], which is a detailed cycle-accurate NoC simulator, and extend it to support the two-stage adaptive routing. I use the integrated Orion power model [177] to track the dynamic and leakage power of the NoC. All simulations are performed for a 25-node (5x5) mesh network. I restrict our evaluation to a 2D mesh NoC, but the general principles presented here could be applied to other NoC topologies as well. Each VC group has 4 virtual channels. Each VC holds four 128-bit flits. I evaluate the proposed techniques using a set of representative synthetic traffic patterns (i.e. uniform random, transpose, bit-complement and tornado). In addition, I use traffic traces from real-world workloads such as SPLASH-2 [178], SPEComp [179] and Specjbb 2005 [180] in techniques’ evaluation. For synthetic traffic simulation, I modified the Garnet simulator to inject packets during a period of 1 million cycles (including 100K warm-up cycles). Both one-flit and five-flit long packets are injected. Please also refer the Evaluation section of Chapter 6 and Chapter 7 for the NBTI and process variation modeling, and the circuit, architecture level evaluation methodology.

**Effectiveness of the VA_SA_M1**

As shown in [127], inserting the special input values (SIV) into a combinational logic unit during its idle periods shows a strong capability in NBTI mitigation. I compare the effectiveness of VA_SA_M1 with that of SIV in both virtual channel allocation and switch allocation logics. I incorporate adaptive body biasing in SIV scheme for the purpose of fair comparison. Note that the VA_SA_M1 shares the same PV guardband as SIV. Figure 8-9 (a)-(d) shows \( NBTI\_guardband \) and \( NBTI&PV\_overhead \) (In this dissertation, I use \( NBTI&PV\_efficiency \) and...
achieved by VA_SA_M1 and SIV when using the four synthetic traffic patterns with different packet injection rates. Since there are multiple routers on the chip and VA_SA_M1 performs at intra-router level, the analysis is focused on routers that suffer the most severe NBTI degradation. Note that VA_SA_M1 does not block any packet during the VA stage so it does not influence the number of cycles that a packet needs to traverse through the router pipeline. Therefore, the network latency is not affected by our proposed techniques. The area and power overhead of VA_SA_M1, which affects TDP is incorporated in the NBTI&PV_overhead calculation.

Figure 8-9 shows that VA_SA_M1 provides strong NBTI mitigation under heavy traffic loads. For example, it achieves 47% NBTI_guardband reduction compared to SIV in uniform random traffic when the injection rate is 0.1. When there are light traffic loads (e.g. 0.02 flits/node/cycle), VA_SA_M1 gains less NBTI_guardband reduction. Due to the lower arbiter utilization, the long idle period already provides a good opportunity for SIV to achieve NBTI recovery. Therefore, there is a limited room for VA_SA_M1 to further improve it. On the other hand, when the network loads are extremely heavy (e.g. 0.42 flits/node/cycle in uniform random traffic), all the arbiters are busy and the possibility for VA_SA_M1 to migrate the utilization is low. Therefore, the guardband reduction becomes smaller. In general, VA_SA_M1 reduces the NBTI&PV_overhead by 10% compared to SIV. One may notice that VA_SA_M1 shows smaller NBTI mitigation improvement for tornado traffic. This is because in tornado traffic, packets are only sent along X direction, the VCs and their local arbiters at the east and west network directions are highly utilized. As a result, VA_SA_M1 does not have enough opportunities to hide-the NBTI degradations via the faster arbiters at the X dimension. Note that the
**NBTI_guardband** is determined by the maximum $V_{th}$. Even though VA_SA_M1 efficiently mitigates the NBTL effect in the Y dimension, its benefit does not show up in the final results.

Figure 8-10 shows the sensitivity analysis on the threshold guardband used for enabling AVS+. The uniform random traffic with 0.3 flit/node/cycle injection rate is used in this example. As it shows, when the threshold exceeds a certain value (17%), both **NBTI_guardband** and **NBTI&PV_overhead** are high and largely hold constant. Because the threshold is higher than the final VA delay under VA_M1, AVS+ is not triggered and the **NBTI_guardband** is mainly affected by XA delay. Meanwhile, a high **NBTI_guardband** leads to a high **NBTI&PV_overhead**. When the threshold guardband is lower, VA_SA_M1 obtains considerable **NBTI_guardband** reduction with even lower **NBTI&PV_overhead**. However, when it drops below 12%, VA_SA_M1 cannot obtain more benefit in reducing the **NBTI_guardband** and **NBTI&PV_overhead** starts to increase again. This is because the extremely low threshold causes a large amount of power overhead due to AVS+; but the **NBTI_guardband** is limited by the VA delay. In this study, I conservatively set the threshold guardband as 11%.

**Effectiveness of VC_M2**

VC_M2 is compared with the 50%_inversion scheme which fixes the inversion percentage as 50% for NBTL mitigation. Note that adaptive body biasing technique is trigged in 50%_inversion as well. Figure 8-11 (a)-(d) presents the network latency and **NBTI&PV_overhead** of VC_M2 and 50%_inversion for the different traffic patterns. Note that both techniques target the same maximum $V_{th}$. Therefore, they achieve the same guardband. The TDP of VC_M2 and 50%_inversion is 1.03. As can be seen, when there is heavy traffic, VC_M2 significantly improves the network latency and reduces the **NBTI&PV_overhead** compared to 50%_inversion. Take uniform random traffic as an example, compared to 50%_inversion, VC_M2 can absorb 24% more NoC load before becoming saturated (an NoC is saturated when
the network latency is three times the zero-load latency) and this also results in 164X

NBTI&PV_overhead reduction. VC_M2 obtains similar latency as 50%_inversion when the
traffic is light. As most of the VC buffers are free, inverting 50% of the free buffers will not
introduce a significant performance penalty.

**Effectiveness of IR_M3**

IR_M3 is compared with two other schemes: RCA (Regional Congestion Aware routing)
[171], which is an adaptive routing scheme that gives the performance metric 100% weight and
therefore does not consider the NBTI and PV effect, and AR_reliability, which only takes the
reliability statistics into consideration when routing packets. Adaptive body biasing technique is
included in both RCA and AR_reliability. Figure 8-12 shows the NBTI&PV guardband and
network latency of these three techniques across different traffic patterns. Note that SIV is
applied to all the three schemes. As Figure 8-12 shows, in most cases, IR_M3 achieves a much
lower guardband (e.g. 50%) compared to RCA. The benefit becomes smaller when the traffic is
heavy due to the high router utilization. On the other hand, IR_M3 significantly improves the
network throughput (around 19%) when compared to AR_reliability. Figure 8-13 shows the
NBTI&PV overhead, which considers trade-offs among performance, reliability, and power. As
can be seen, IR_M3 outperforms both RCA and AR_reliability. In general, it reduces the
NBTI&PV overhead by 14% and 30% when compared to the two schemes, respectively. Figure
8-13 shows that when the packet injection rate increases, the NBTI&PV overhead in
AR_reliability first increases to an extremely high value (due to its higher network latency
compared to the baseline case without PV and NBTI effect), and then drops down (because the
network latency in the baseline case increases to infinite as well). Different from AR_reliability,
the NBTI&PV overhead in IR_M3 and RCA first drops before the variations, because both of
them improve the network throughput compared to the baseline case. For example, under the
same injection rate (e.g. 0.29 flits/node/cycle in transpose traffic), the network latency in the baseline case becomes infinite while it still stays low in the two schemes. IR_M3, RCA, and AR_reliability merge to the same NBTI&PV_overhead when the injection rate is around 0.45 flits/node/cycle since the network is saturated in all these cases.

Since the proposed intra-router and inter-router techniques are orthogonal to each other, I propose to combine them together to gain additional benefits in NBTI mitigation. The combined techniques cause total 4% area overhead to each router. I compare our combined scheme with the technique (i.e. SIV+50%_inversion+RCA) that uses SIV, 50%_inversion and RCA together, adaptive body biasing is incorporated as well. Compared with SIV+50%_inversion+RCA, my combined technique can reduce the NBTI&PV_guardband and NBTI&PV_overhead by as much as 70% and 41% respectively while improving network throughput by 5%. Note that the network latency has already been improved by using SIV+50%_inversion+RCA due to the use of congestion aware adaptive routing to balance the buffer utilization in routers.

Real Workload Results

Figure 8-14 (a)-(c) shows the NBTI&PV_guardband, network latency, and NBTI&PV_overhead of the real workloads when the proposed schemes are applied. The results are normalized to the technique of SIV+50%_inversion+RCA. As it shows, my techniques are able to improve the NBTI recovery on benchmarks with high traffic load. For example, on water-spatial which exhibits high traffic, when compared with SIV+50%_inversion+RCA, the combined scheme can reduce the NBTI&PV_guardband and network latency by 59% and 4% respectively, leading to 33% reduction in NBTI&PV_overhead. Similarly, on benchmarks with relatively high traffic (e.g. water-nsquared, equake, fma3d, and mgrid), the combined technique also improves the NBTI&PV_guardband, network latency and NBTI&PV_overhead by 48%, 2% and 20%, respectively. The improvement on medium-load benchmarks (e.g. ocean, radix, barnes
and fft) is 30% in NBTI&PV_guardband, 1% in network latency, and 12% in NBTI&PV_overhead. The combined technique achieves smaller improvements in light-load benchmarks such as raytrace, because there are fewer uses of VA arbiters and less congestion in routers. The NBTI&PV_guardband reduction is 5% with 0.5% and 2% improvement in network latency and NBTI&PV_overhead, respectively.

**Related Work**

There is a vast body of previous work on improving performance and power efficiency of NoC design [165, 166, 181-184]. These studies all assume that the underlying transistors share the same characteristics. In [164], Li and Peh observed that NoC design choices are very much influenced by the effects of process variation. Their study focuses on analyzing process variation early in the design flow while our work proposes PV mitigation techniques suitable for runtime NoC operation. In [125], Ogras and Marculescu studied the use of NoC consisting of multiple voltage-frequency islands to cope with parameter variation problems. To tolerate the effect of process variations on NoC, [86] proposed using self-correcting links that automatically detect delay variations and compensate them, [173] proposed VA compaction and SA folding mechanisms to increase the NoC immunity to PV effects. The above studies exclusively focus on PV while my work targets both PV and NBTI, which inherently interplay with each other.

In the past, various PV mitigation techniques have been proposed [111-115]. Those PV mitigation schemes largely focus on processor core architectures while our work proposes both intra-router and inter-router PV mitigation techniques suitable for the state-of-the-art router microarchitecture and the distributed, large-scale NoC substrates. I expect that my work will open new opportunities for robust nano-scale NoC design. The combined effect of PV and NBTI has been modeled and analyzed in [154, 155,161]. In this work, we focus on leveraging NoC microarchitecture characteristics for the purpose of NBTI and PV mitigation. For example, in
flow-control-based NoC, credit of VC availability is sent back from the downstream router to its upstream router. In our two low-level techniques, we take advantage of backward credit to intelligently steer the VC allocation (VA_M1) and control the required inversion percentage (VC_M2) in the downstream node. Compared to Razor [156], in this study, I target the mitigation of NBTI and PV effect in both combinational circuits and storage-cell structures with desirable trade-offs among performance and reliability. Recently, Tiwari and Torrellas proposed aging driven application scheduling [162], which can hide the effect of aging by intelligently mapping workloads to cores with different frequency (caused by process variation). To the best of my knowledge, there has been no prior work on addressing NBTI effect in NoCs.
Figure 8-1. Two-stage adaptive router microarchitecture
Figure 8-2. Circuit design of the two-stage adaptive router [28]. A) Circuit design of the two-stage adaptive router. B) Zoom-in view at VA logic.
Figure 8-3. Circuit design in the first step of the VA stage (one VC entry is shown)

Figure 8-4. The implementation of VA_M1
Figure 8-5. The guardband and power benefit of applying AVS+ during the lifetime. A) Guardband benefit. B) Power benefit

Figure 8-6. $V_{th}$ (in mV) variation map for a router
Figure 8-7. The implementation of VC_M2
Figure 8-8. The implementation of IR_M3
Figure 8-9. The effectiveness of VA_SA_M1 on $NBTI_{guardband}$ and $NBTI_{PV\_overhead}$. A) Uniform random traffic. B) Bit-complement traffic. C) Transpose traffic. D) Tornado traffic.
Figure 8-9. Continued
Figure 8-10. Sensitivity analysis on threshold (URT with 0.3 flit/node/cycle injection rate)
Figure 8-11. The effectiveness of VC_M2 on network latency and NBTI&PV_overhead. A) Uniform random traffic. B) Bit-complement traffic. C) Transpose traffic. D) Tornado traffic.
Figure 8-11. Continued
Figure 8-12. The Effectiveness of IR_M3 on Network latency and $NBTI \_guardband$. A) Uniform random traffic. B) Bit-complement traffic. C) Transpose traffic. D) Tornado traffic.
Figure 8-12. Continued
Figure 8-13. The effectiveness of IR_M3 on $NBTI&PV_{overhead}$. A) Uniform random traffic. B) Bit-complement traffic. C) Transpose traffic. D) Tornado traffic.
Figure 8-13. Continued
Figure 8-14. The effectiveness of the combined techniques (VA_SA_M1+VC_M2+IR_M3) on real workloads. A) Normalized NBTI&PV_guardband. B) Normalized network latency. C) Normalized NBTI&PV_overhead.
Figure 8-14. Continued.
Conclusions

Semiconductor transient faults have become an increasing challenge for reliable software execution. To explore cost-effective fault tolerant mechanisms for dependable execution of the next generation of software, researchers clearly need to analyze program vulnerability to soft errors at a high level and at an early design stage. I have developed Sim-SODA, a unified framework to estimate software vulnerability to transit faults at the architectural level. The foundations for the vulnerability modeling infrastructure are parameterized AVF models of microarchitecture structures present in modern high-performance microprocessors. Compared with previously proposed tools, Sim-SODA provides fine-grained AVF models and covers more hardware structures.

Phase analysis is becoming increasingly important for optimizing the efficiency of next generation computer systems. As semiconductor transient faults become an increasing threat to reliable software execution, it is imperative to design workload dependent fault tolerant mechanisms for future microprocessors which contain billions of transistors. Observing reliability-oriented program phase behavior is the first step leading to dynamic fault tolerant systems which can be tuned to meet the workload reliability requirement. I studied the run-time reliability characteristics of four important microarchitecture structures. Is investigated the applicability and effectiveness of different phase identification methods in classifying program reliability-oriented phases. I found that both code-structure based and run-time event based phase analysis techniques perform well in detecting reliability oriented phases. In general, the performance counter scheme outperforms the control flow based scheme on a majority of benchmarks. I also found that program reliability phases can be cost-effectively identified using
five performance counters that are generally available across different hardware platforms. This suggests that on-line reliability phase detection and prediction hardware with low complexity can be built on a small set of performance counters.

Since IQ exhibits high vulnerability in SMT environment, I have presented novel microarchitecture techniques designed to reduce the IQ vulnerability to soft errors on SMT processors. The key observation is that the number of vulnerable instructions that are ready to execute on SMT processors is much higher than that on superscalar processors. The IQ vulnerability to soft error can be reduced by assigning vulnerable instructions a higher issue priority. This has the effect of reducing the number of ACE-bit resident cycles and thus the vulnerability of the IQ. I further apply reliability-aware dynamic resource allocation to the IQ to prevent excessive vulnerable bits from entering the IQ. Results from the implementation and evaluation of the proposed techniques show 42% reliability improvement in the IQ with 1% performance improvement. Based on the observation that instructions’ long waiting-for-ready time significantly increases IQ soft-error vulnerability, I have further developed six novel microarchitecture techniques to improve the instruction queue reliability and meanwhile, maintain processor performance. I use instruction operand readiness prediction to reduce the performance penalty of ORBIT schemes. The proposed Predict_DelayACE scheme reduces IQ vulnerability by 79% with 1% throughput IPC and 3% harmonic IPC degradation on average across all three types of workloads. The ORBIT schemes outperform the advanced SMT fetch policies when reliability, throughput and fairness are all considered. Results also prove that the IQ vulnerability does not migrate to other structures. In this study, I focus on the IQ, however I believe our technique could be extended to other microarchitecture structures.
Improved reliability design and control over design overhead can be achieved through the use of hybrid approaches that allow simultaneous tradeoffs between circuit and architecture techniques. In this dissertation, SER robustness techniques at the circuit and microarchitecture levels are effectively combined. They benefit each other and are able to obtain a greater reliability improvement with negligible performance loss and power overhead. To my knowledge, this is the first work to efficiently bridge the gap between circuit and microarchitecture level fault tolerance techniques. I first proposed hybrid radiation hardened IQ design, which dispatches performance critical but not-ready instructions into well-protected RIQ, and uses un-protected NIQ to only hold operand ready instructions. The hybrid design takes the advantage of RIQ in SER robustness to achieve the goal of not only protecting critical instructions but also providing a time of issue guarantee. Results show the technique achieves 80% IQ SER reduction with only 0.3% throughput IPC and 1% harmonic IPC penalty. The hybrid scheme also outperforms other existing techniques (e.g. 2OP_BLOCK, ORBIT and FLUSH fetch policy). To improve ROB reliability, I propose the use of architectural characteristics to trigger Dual-$V_{DD}$, and switch the normal $V_{DD}$ to a higher level when the ROB exhibits high Vulnerability during a L2 miss. This technique obtains 35% ROB SER robustness with 3.5% power overhead. I also put the two techniques together and evaluate their aggregate affect on the entire core. On average, the core SER reduces 23%, and as a byproduct, LSQ achieves 15% SER reduction.

Process variation is becoming the major challenge that needs to be addressed as process technology continues to scale down. Most PV related studies focus only on performance and power domains, and little work has been done to link PV to soft error vulnerability. However, an efficient PV mitigation technique should have the ability to take reliability, performance, and
power into account. I present the first study to optimize the reliability of PV mitigation techniques while considering all three domains. I characterize the critical charge variation at the bit and entry level in the presence of PV. I propose Entry-BVM and Structure-BVM to improve microarchitecture soft error reliability and its trade-offs with performance and power in light of PV. Simulation results show that Entry-BVM and Structure-BVM can reduce IQ SER by 24% and 20% and achieve a trade-offs improvement of 27% and 26% respectively. The combinatorial effect of the two techniques can achieve even higher IQ SER reduction (i.e. 40%) with 46% trade-offs improvement.

NBTI is another crucial reliability concern in nanometer technology. It degrades PMOS transistors by increasing their $V_{th}$, which leads to failures in both logic circuits and storage cells. Meanwhile, process variations (PV), which result in a static parameter variation (e.g. $L$ and $V_{th}$) in transistors, exacerbate the reliability problem in current high performance processors. Methodologies to mitigate both PV and NBTI effects are highly desired. In this study, I observe that techniques leveraging the positive interaction between PV and NBTI can obtain attractive trade-offs among performance, reliability, and power. I propose three microarchitecture optimizations to efficiently take advantage of the positive interplay between NBTI and PV to mitigate NBTI effect in the presence of PV. My techniques are flexible and can be applied to most of the microarchitecture structures. My experimental results show that the aggregated effect of the proposed methods has the ability to improve the chip NBTI&PV efficiency by 117% compared to the baseline case without any optimization, and by 21% compared to the technique which simply combines NBTI and PV mitigation methods.

NoC is becoming the imperative communication fabric for emerging multi-/many- core processors. Existing NoC designs largely assume reliable processing technologies and uniform
transistor characteristics. As CMOS fabrication technologies approach the nano- and atom-
scales, process variation can significantly degrade the performance and reliability of these
designs. This problem is further compounded by the NBTI effect, which wears-out transistors
and reduces their lifetime. Therefore, it is unwise to ignore the impact of PV and NBTI in NoC
architecture design. In this study, I propose novel techniques to mitigate the impact of PV and
NBTI on NoC. Experimental results show that my intra-router techniques (i.e. VA_SA_M1 and
VC_M2) reduce guardband by 47% while improving network throughput by 24%. My inter-
router optimization scheme (i.e. IR_M3) results in 50% guardband reduction and 19% network
latency improvement. To the best of our knowledge, this is the first work that optimizes both PV
and NBTI effect on the emerging NoC architecture design.

**Future Works**

In the near future, I am interested in extending my Ph.D. research work to design network-
on-chip architectures in many-/multi-core processors considering the interactions among OS,
architecture, and circuit levels to improve performance, reliability and power on emerging
application execution. For example, observing that routers in NoC exhibit different speed due to
the PV effect, and faster routers can tolerate more delay caused by NBTI degradation, I intend to
leverage OS in scheduling the applications with heavy traffic to the area including a large
number of faster routers; on the other hand, applications with light traffic are assigned to the area
with slow routers. By doing that, we can achieve an optimal trade-offs among performance,
reliability and power.

In addition, the one-to-many (multicasting) traffic degrades the performance significantly
in current network-on-chip architectures, it is crucial to explore an improved NoC architecture
which provides low latency, high throughput, and high reliability to multicasting traffics. I intend
to dynamically build up and update the multicasting trees to avoid the traffic congestion, and
mitigate the impact of process variation and NBTI degradation in each router. Moreover, OS can help to reduce the number of hops required in each multicasting tree by assigning the specific cores which should be involved into an application. Furthermore, in order to further improve the latency and throughput in NoC architectures, 3D technologies and the nanophotonic devices have been introduced to network-on-chip, I intend to look at the challenges and opportunities in those novel techniques for NoC design.

My long-term research will focus on exploring the reliable and high efficient many-/multi-core processor architecture design. Semiconductor roadmaps forecast that ten years from now, processor chips will integrate 128 billion transistors in 11nm technology. This will likely translate into hundreds and even thousands of cores into one chip. It greatly helps to boost the performance, but will keep exacerbating the reliability issues. For example, some cores may fail infancy due to the impact of process variation, some cores may fail during the required lifetime due to the severe NBTI degradation, moreover, some cores may not be able to execute the program correctly due to its high vulnerability to soft errors. I intend to develop techniques across OS, architecture and circuit levels to build robust many-/multi-core processors. I intend to explore techniques to support systems that automatically analyze heterogeneous workloads, dynamically reconfigure the application components based on the current reliability/performance/power characteristics of the cores assigned to the application. At circuit level, vulnerability mitigation techniques (e.g. dynamic voltage scaling, adaptive body biasing) can be efficiently combined with the architecture and OS level techniques to achieve the optimal trade-offs among reliability, performance and power in future many-/multi-core architecture design.
LIST OF REFERENCES


[63] Wangyuan Zhang, Xin Fu, Tao Li, and José Fortes, “An Analysis of Microarchitecture Vulnerability to Soft Errors on Simultaneous Multithreaded Architectures,” in Proc. ISPASS, April 2007


BIOGRAPHICAL SKETCH

Xin Fu was born in Xiangxiang, China, as the daughter of Zhongyang Fu and Yafei Wang. After completing her high school education at the First Middle School in Xiangxiang, China, she entered the Department of Computer Science and Technology, Central South University in Changsha, China in September 1999. She received the degree of Bachelor of Engineering in computer science and engineering from Central South University in July 2003. From September 2003 to July 2004, she joined the graduate program for computer science and engineering at the Department of Computer Science and Technology, Changsha, China. In September 2004, she entered the Ph.D. program in computer engineering at The University of Florida. She is a student member of The Institute of Electrical and Electronics Engineers (IEEE), The Institute of Electrical and Electronics Engineers (IEEE) Computer Society, Association for Computing Machinery (ACM), and Association for Computing Machinery Special Interest Group on Computer Architecture. (ACM SIGARCH).