The Dissertation Committee for Arun Arvind Nair
certifies that this is the approved version of the following dissertation:

Efficient Modeling of Soft Error Vulnerability in Microprocessors

Committee:

Lizy Kurian John, Supervisor

Lieven Eeckhout

Mattan Erez

Nur Touba

Earl E. Swartzlander, Jr.

Michael D. Bryant

Efficient Modeling of Soft Error Vulnerability in Microprocessors

by

Arun Arvind Nair, B.E.; M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2012

To my parents: Arvind and Premlata Nair.

Acknowledgments

I am grateful to my advisor Prof. Lizy John for her guidance and support throughout my tenure here at UT Austin. I am grateful for the flexibility and resources she has provided me in order to make my research possible, and her availability to discuss issues and problems with research.

I have had the privilege of collaborating with Prof. Lieven Eeckhout, who greatly helped to shape this dissertation. I am immensely grateful to him for his guidance and encouragement, and the many hours of proof-reading and correcting my conference paper submissions.

I thank Dr. Stijn Eyerman for his guidance while developing the mechanistic model. He provided me his implementation of interval analysis profilers, which enabled me to concentrate on developing the AVF model. I am deeply indebted to Arijit Biswas for taking time out to explain the various issues regarding AVF evaluation, and for reading my papers and providing detailed comments. I am thankful to Shubu Mukherjee for the initial discussions and inputs that lead to the work presented in this dissertation.

I am grateful to the members of my committee (in alphabetical order): Prof. Michael Bryant, Prof. Mattan Erez, Prof. Earl Swartzlander, and Prof. Nur Touba. Their feedback has been invaluable towards improving the quality of my research. Prof. Herb Krasner was kind enough to employ me

as a Teaching Assistant for various courses that he has taught. I had the pleasure of working as a TA with Prof. Vijay Garg and Prof. Miryung Kim, and I thank them for their support. I am also grateful to Prof. Vijay Reddi for his advice and help towards my job search.

I also thank my labmates at the Laboratory of Computer Architecture (LCA) (in alphabetical order): Aashish Phansalkar, Ciji Isen, Deepak Panwar, Dimitris Kaseridis, Faisal Iqbal, Jian Chen, Jeff Stucheli, Jungho Jo, Karthik Ganesan, Lloyd Bircher, Nidhi Nayyar, Umar Farooq and YoungTaek Kim, for comments, or feedback on my research, proposal and defense practice talks. I enjoyed my collaboration and conversations with Jian on each other's work. Discussions with Faisal over coffee and/or lunch, on research, cricket, culture, or pretty much anything else were always fun. I have also enjoyed the company of Dimitris and Ciji and particularly appreciate their efforts in making me feel home when I first joined the group. I am grateful to Karthik for his help with the SNAP GA framework, and the simpoints that he generated for SPEC CPU2006.

I am grateful to a number of people in my family who made this possible: to my parents, Arvind and Premlata Nair, for their support, love, and encouragement; to my sister Sonal, brother-in-law Dhaval, cousins Anand and Amrita, and Ami for their support. And not the least, to my fiancèe Sunita, for all the love, encouragement, support, and patience.

Last but not the least, I am indebted to the taxpayers of the states of Texas and California for affording me an excellent education.

Efficient Modeling of Soft Error Vulnerability in Microprocessors

Publication No. _____

Arun Arvind Nair, Ph.D.
The University of Texas at Austin, 2012

Supervisor: Lizy Kurian John

Reliability has emerged as a first class design concern, as a result of an exponential increase in the number of transistors on the chip, and lowering of operating and threshold voltages with each new process generation. Radiation-induced transient faults are a significant source of soft errors in current and future process generations. Techniques to mitigate their effect come at a significant cost of area, power, performance, and design effort. Architectural Vulnerability Factor (AVF) modeling has been proposed to easily estimate the processor's soft error rates, and to enable the designers to make appropriate cost/reliability trade-offs early in the design cycle. Using cycle-accurate microarchitectural or logic gate-level simulations, AVF modeling captures the masking effect of program execution on the visibility of soft errors at the output. AVF modeling is used to identify structures in the processor that have the highest contribution to the overall Soft Error Rate (SER) while running typical workloads, and used to guide the design of SER mitigation mechanisms.

The precise mechanisms of interaction between the workload and the microarchitecture that together determine the overall AVF is not well studied in literature, beyond qualitative analyses. Consequently, there is no known methodology for ensuring that the workload suite used for AVF modeling offers sufficient SER coverage. Additionally, owing to the lack of an intuitive model, AVF modeling is reliant on detailed microarchitectural simulations for understanding the impact of scaling processor structures, or design space exploration studies. Microarchitectural simulations are time-consuming, and do not easily provide insight into the mechanisms of interactions between the workload and the microarchitecture to determine AVF, beyond aggregate statistics.

These aforementioned challenges are addressed in this dissertation by developing two methodologies. First, beginning with a systematic analysis of the factors affecting the occupancy of corruptible state in a processor, a methodology is developed that generates a synthetic workload for a given microarchitecture such that the SER is maximized. As it is impossible for every bit in the processor to simultaneously contain corruptible state, the worst-case realizable SER while running a workload is less than the sum of their circuit-level fault rates. The knowledge of the worst-case SER enables efficient design trade-offs by allowing the architect to validate the coverage of the workload suite and select an appropriate design point, and to identify structures that may potentially have high contribution to SER. The methodology induces 1.4× higher SER in the core as compared to the highest SER induced by SPEC CPU2006 and MiBench programs.

Second, a first-order analytical model is proposed, which is developed from the first principles of out-of-order superscalar execution that models the AVF induced by a workload in microarchitectural structures, using inexpensive profiling. The central component of this model is a methodology to estimate the occupancy of correct-path state in various structures in the core. Owing to its construction, the model provides fundamental insight into the precise mechanism of interaction between the workload and the microarchitecture to determine AVF. The model is used to cheaply perform sizing studies for structures in the core, design space exploration, and workload characterization for AVF. The model is used to quantitatively explain results that may appear counter-intuitive from aggregate performance metrics. The Mean Absolute Error in determining AVF of a 4-wide out-of-order superscalar processor using model is less than 7% for each structure, and the Normalized Mean Square Error for determining overall SER is 9.0%, as compared to cycle-accurate microarchitectural simulation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Shrinking process geometries have enabled an exponential increase in the number of transistors fabricated on a chip, with each successive process generation. While this process has enabled lower operating voltages and higher frequencies, it has also made reliability of hardware an increasingly important design criterion. Radiation induced faults are a significant source of transient faults in hardware, and the situation is expected to worsen with smaller feature sizes [1–4].

Transient faults, unlike permanent or hard faults, are temporary in nature, and disappear once the factor causing the fault disappears. Radiation-induced faults are transient faults that occur as a result of strikes by energetic particles such as neutrons from cosmic radiation, and alpha particles from radioactive trace elements in the packaging materials. A particle of sufficient energy displaces electrons in the substrate in sufficient number to flip the state of the storage element, or logic gate. As transistor threshold voltages and operating voltages reduce, less energy is required to cause a bit flip, making soft errors a more acute problem in future process generations.

High-end microprocessors have protected large SRAM arrays such as

the last-level caches against transient faults with Single-bit Error Detection, Double-bit Error Correction (SECDED) codes. Increasingly, however, it is becoming necessary to protect latches and flip-flops in the core as well. For example, Fijutsu's 130nm SPARC64 design [5] protects 80% of the latches with parity bits. IBM POWER7 [6] additionally correct single event upsets by enabling re-execution of faulting instructions. Such soft-error mitigation strategies incur significant overheads in terms of performance, power, area and design effort, necessitating judicious application of these techniques.

Mitigating Soft Errors: Soft errors can be mitigated through changes in process technology, circuits, microarchitecture, or architecture [7]. Mukherjee et al. [7] note that while Silicon On Insulator (SOI) technology reduces the the radiation induced fault rate at the transistor level, it does not completely eliminate the problem. Circuit-level techniques such as radiation hardened circuits, or increased transistor sizing can be employed to reduce the susceptibility of individual circuits. However, they incur a significant area and power penalty. Microarchitecture-level solutions such as parity-based error detection, and correction or recovery can be utilized. For example, the IBM POWER 7 has features to detect soft errors, and re-execute faulting operations [6]. These techniques protect against faults that are detected early enough such that re-execution is possible. Naturally, there is a significant overhead for implementing error detection and recovery circuitry. At a higher level, redundant execution techniques, such as Triple Modular Redundancy (TMR) can be

used, in which three pipelines execute the same program in lockstep, and the output is compared at every cycle. In the event that one processor experiences a fault, a voting mechanism picks the majority value as the correct one.

Due to the significant overhead in implementing SER mitigation schemes, it is necessary to balance the need for SER mitigation with performance, power, area and design effort targets. Two methodologies to address this issue are presented in this dissertation. A methodology to estimate the worst-case SER observable while running a realizable workload is developed. This methodology enables the designer to make informed decisions regarding efficient design for worst-case SER. Furthermore, a first-order analytical model to estimate SER is developed. This enables the architect to perform design space exploration, parametric studies, and workload characterization for soft error vulnerability.

1.1 Modeling the Vulnerability to Soft Errors

Most usage scenarios do not need the high level of error protection afforded by techniques such as TMR. In such usage scenarios, the additional overhead of guaranteeing error-free operation may be undesirable. For example, typical servers or consumer devices need to meet a reliability target for which strategies such as TMR are excessive, and too expensive. For such usage scenarios, designers would prefer to meet their reliability objectives efficiently, by protecting a small number of structures in the processor, such that the overhead of the soft-error mitigation mechanisms is minimized. The designers thus need a methodology to identify the structures that require protection

such that the Soft Error Rate (SER) is brought within specifications.

Prior research [2, 8] has shown that masking effects of program behavior have a significant impact on the visibility of faults to the user. Wang et al. [8] report that nearly 85% of all the transient faults in the core are masked by program execution. Different programs stress the microarchitecture differently, thereby exposing or masking faults at different rates. Furthermore, certain structures, such as the branch predictor affect only performance, and have no impact on correctness: a fault in the branch predictor's history counter values may affect performance, but not correctness. This suggests that it is unnecessary to protect every single bit in the processor to bring the SER within specifications. Therefore, it becomes necessary to model the impact of program execution on the visibility of the faults, in order to protect the structures in the processor that are the highest contributors to the overall SER, such that the overall reliability target is met.

This masking effect of program execution is captured using *Architectural Vulnerability Factor* (AVF) modeling, which expresses the probability that a fault occurring in a particular structure will manifest itself as an error in the program output. Architects use AVF modeling to determine the masking effect of a set of workloads. The architect may decide to add sufficient SER mitigation mechanisms such that the observable SER while running a typical workload is brought within specifications. AVF modeling thus enables the architect to efficiently select SER mitigation schemes such that the target SER is met.

Statistical Fault Injection (SFI) can be used to compute the AVF of a structure. Using a Register Transfer Level (RTL) or logic gate-level models of the processor executing a workload, single faults are injected into a structure at random instants in time. The average fraction of such injected faults that manifest as program errors would then be measured to estimate the masking effect of the workload on the faults in that structure. However, RTL simulations are time-consuming, and SFI requires a large number of runs with randomly injected faults, for statistical significance. Furthermore, RTL models are not available during the early design planning and thus have limited use in guiding microarchitectural decisions for mitigating soft errors.

In order to move AVF modeling earlier in the design cycle, Mukherjee, et al. [2] propose *Architecturally Correct Execution* (ACE) analysis to provide a conservative estimate of AVF. ACE analysis only requires a single execution of the workload on a microarchitecture-level performance simulator, which is orders of magnitude faster than an RTL model. ACE analysis estimates AVF by capturing the fraction of bits per cycle in a hardware structure that contain correctness-critical state. ACE analysis is thus a very useful tool in estimating the efficacy of architecture-level SER mitigation schemes during the early design phase. It enables the architect to estimate the power, performance, area and reliability trade-offs of design decisions much earlier in the design cycle.

1.2 Motivation

The observable SER of a workload is strongly dependent on the microarchitecture, workload, and underlying circuit-level fault rates. Different programs stress microarchitectural structures differently, and hence a change in the microarchitecture, or underlying circuit-level fault rates alters their observed SER by different proportions. There has been limited work on analyzing the fundamental program characteristics affecting the masking effect of program execution. The complex interaction between the microarchitecture and workload imply that simple performance metrics, such as IPC or cache miss rates do not correlate with AVF. AVF estimation is reliant on detailed microarchitectural simulations, which report aggregate metrics, but do not uncover the fundamental factors affecting AVF. A large number of time-consuming simulations are required to derive some insight on the effect of parametric or design changes on AVF. Whereas black-box statistical or machine-learning based methodologies have been proposed [9–11], they rely on detailed microarchitectural simulations, and provide no insight into the precise mechanisms influencing AVF. In the following discussion, a motivation for better understanding the effect of program execution on AVF is presented.

1.2.1 Mitigating the Effect of a Biased Workload Suite

There is no known methodology for selecting workloads for AVF benchmarking that are demonstrably representative of the entire population of user workloads. A workload suite that offers adequate coverage on one microar-

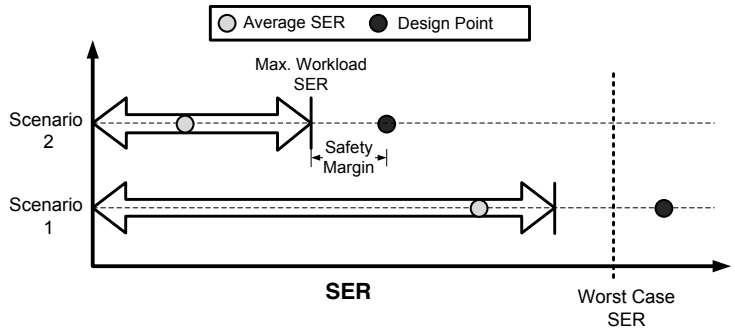


Figure 1.1: Choice of the design-point, under different SER coverage scenarios

chitecture and circuit-level fault-rate does not neccessarily do so when either factor is changed. Therefore, any design decision made based on the AVF induced by a small set of workloads is potentially biased either towards over-design, or under-design. This is a common statistical sampling problem: how does one verify whether the mean of the sample is equal to the mean of the population? This is made worse by the fact that workloads for AVF evaluation are not picked through random sampling, but a result of specific choices, and are hence likely to be biased. Furthermore, AVF simulation requires detailed, microarchitecture-level simulations, which are computationally intensive. In order to avoid the prohibitive expense of running the entire workload on a detailed performance simulator, the designer would typically run short traces of these workloads, potentially increasing the sampling bias.

Such biases in the workload suite may potentially lead to over-design or under-design for SER mitigation. An overdesigned processor has a higher power, performance and area penalty, whereas an under-designed processor

may fail to meet its reliability goal. To protect against potential under-design due to a workload suite biased towards low SER, architects add a guard band, or safety margin [12]. There is no known methodology for estimating the guard band. This guard band is often selected based on designer intuition. It may be very expensive to correct underdesign for SER later in the design cycle, and it is therefore important to model AVF as accurately as possible during the early design stages. Figure 1.1 represents two scenarios, assuming a design for the highest observed SER (other design points are possible, such as average, median, or a percentile of the workload suite). The arrows depict the range of SER observed while running the workload suite. Scenario 1 represents the case in which the workload suite has sufficient SER coverage, and an additional guard band is unnecessary. Automatic addition of a guard band will push the design point beyond the maximum attainable SER, leading to over-design. On the other hand, scenario 2 represents a case in which the workload suite requires a significant safety margin to guard against potential under-design, due to the significant gap in its SER coverage, relative to the worst-case observable SER. The knowledge of the worst-case observable SER thus allows the architect to pick the appropriate guard band, and to validate the SER coverage of the workload suite.

Estimating this worst-case observable SER is non-trivial. It is not possible for every bit in the processor to contain correctness-critical state simultaneously, due to complex interactions between the various structures in the processor. For example, instructions in the rerder buffer will be issued into the

load queue, store queue, branch units, or arithmetic units, based on their type. An increase in instructions of one type will result in a proportionate reduction in state in the other units. Thus, simply adding the circuit-level failure rates of all bits in the core would result in gross overestimation. Furthermore, the circuit-level fault rates of each structure may be different. The same amount of correctness-critical state in a structure with a higher fault rate is likely to expose more errors than a structure with a low fault rate. There is therefore a need for a systematic methodology to develop a workload that maximizes the visibility of soft errors for a given microarchitecture and circuit-level fault rates.

1.2.2 Design-time AVF modeling

Detailed microarchitectural simulations have become the mainstay of computer architecture research and development, due to their relative accuracy and flexibility. Architects typically use detailed microarchitectural simulations to study the effect of microarchitectural or parametric changes on AVF, power, area and performance, in order to determine the best trade-off between these objectives. However, such simulations can be time-consuming when performed over a large number of workloads, for a large number of instructions, and over a large number of microarchitectural configurations. Such simulations provide aggregate metrics, and it is difficult to draw inferences on the precise impact of the workload and various microarchitectural parameters on AVF and performance using these metrics.

Additionally, it is time consuming to perform design space exploration or parametric sweeps for various microarchitectural parameters using detailed simulation. The simulator masks the precise microarchitectural mechanisms that influences the aggregate metrics, providing little insight into their underlying causes. For example, it is difficult to estimate the efficacy of an SER mitigation scheme proposed in literature [13], involving the enabling soft-error mitigation mechanisms in the shadow of a last-level data cache miss, without implementing this mechanism in the simulator.

Analytical design space methodologies can complement cycle-accurate simulations for performing experiments, such as parametric sweeps or design space exploration. The computational simplicity of an analytical model allows the architect to use these models to cheaply explore the design space, and eliminate infeasible design points, and guide detailed simulations. A well-constructed analytical model also provides invaluable insight into the mechanisms influencing AVF and performance, allowing the designer to make informed design choices. Owing to its simplicity, an analytical model also allows the architect to study a greater number of workloads, over a larger number of instructions than may be possible using detailed simulations.

1.2.3 Characterizing Workloads for their Impact on AVF

The precise mechanism of the masking effect of program execution on the visibility of soft-errors has not been sufficiently studied, making workload characterization for AVF a significant challenge. Fu et al. [14] report a "fuzzy

relationship" between AVF and simple performance metrics. Other statistical or machine-learning based models [9–11] rely on detailed microarchitectural simulations and fail to provide insight on the fundamental interaction between the software and hardware that together determine AVF. Given a workload and aggregate metrics obtained using microarchitectural simulations, it is only possible to make very qualitative predictions on its influence on AVF of a structure. Oftentimes, the interaction between various microarchitectural events produces results that run counter-intuitive to these qualitative predictions.

The ability to characterize workloads for their influence on AVF allows the architect to identify workloads that are likely to induce high or low AVF in a particular structure. This, in turn, enables the architect to validate the heterogeneity of the workload suite being used to evaluate the Soft Error Rate (SER). A workload suite with sufficient heterogeneity is essential for enabling the architect to make correct design decisions for reliability and performance.

1.3 Thesis Statement

The microarchitectural events triggered by the execution of the workload influence the architectural vulnerability factor of the structures in a given microarchitecture. It is feasible to develop an automated methodology that generates a synthetic workload to exercise these microarchitectural events such that the Soft Error Rate is maximized. It is also feasible to efficiently model the architectural vulnerability factor by modeling the impact of these microarchitectural events on the occupancy of correctness-critical program state in these

structures.

1.4 Contributions

The challenges outlined in Section 1.2 are addressed through the following contributions:

- Starting with a detailed study of the impact of microarchitecture-dependent workload characteristics on the occupancy of corruptible state in the core, an automated methodology for developing a workload that uncovers the worst-case Soft Error Rate (SER) of a given microarchitecture is developed. First, a systematic methodology to develop a code generator which takes a set of parameters, or knobs, to produce a synthetic workload is demonstrated. The knobs control the microarchitecture-dependent workload characteristics that influence the occupancy of corruptible state in microarchitectural structures is proposed. Second, the code generator is interfaced to an iterative, feedback-driven optimization loop utilizing a genetic algorithm. Upon the convergence of the genetic algorithm, the optimized workload, called an AVF stressmark, estimates the worst-case soft error rate.

- It is demonstrated that the stressmark achieves 1.4× higher SER in the core, 2.5× higher SER in the data L1 cache and TLB, and 1.5× higher SER in L2 cache as compared to the highest SER induced by SPEC CPU2006 and MiBench programs for a processor similar to the

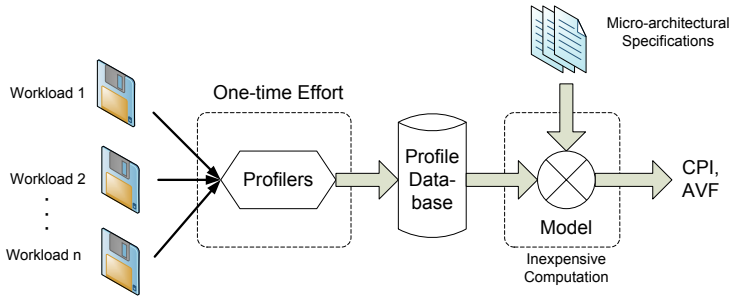


Figure 1.2: Utilizing the model to perform design space exploration.

Alpha 21264. The flexibility of this methodology across different microarchitectures, and underlying fault rates, is demonstrated. It is also demonstrated that naïve estimates of the worst-case, such as adding the raw circuit-level fault rates of all structures, or adding the highest SER induced by any workload in the workload suite, on a per-structure basis, to estimate the worst-case observable SER will lead to significant errors, and consequently, potential overdesign or underdesign.

- Starting with a comprehensive study of the impact of microarchitectural events on the occupancy of correct-path state in microarchitectural structures, a mechanistic modeling methodology that predicts the occupancy of correctness-critical state in the microarchitecture, using inexpensive profiling is developed. The modeling methodology is developed from first principles of out-of-order processor execution, to provide insight into the precise mechanisms that influence the SER of the microarchitecture executing a given workload. Figure 1.2 presents the general overview of the modeling methodology. Workloads are profiled to capture important

metrics required by the model, which is a one-time effort. Multiple microarchitectures can then be modeled using the data from a single profile. The Mean Absolute Error for estimating the AVF for a structure of a 4-wide out-of-order microarchitecture is less than 7%, and the Normalized Root Mean Square Error is 9.0%.

- It is demonstrated that this mechanistic model can be used to cheaply perform design space exploration studies, and evaluate the efficacy of soft error mitigation mechanisms. The model can be used to study the impact of design changes on AVF and performance. Due to its construction, it is able to provide novel insight into the interaction between the workload and the microarchitecture that together determine the AVF of a structure.

- It is demonstrated that the mechanistic model can be used to perform workload characterization for AVF. Using the mechanistic model, the architect can study a greater number of workloads, for a longer period of time than might be possible using detailed simulations. The model can also be used to identify workloads that would induce high or low AVF in a structure, thereby enabling the architect to validate the heterogeneity of the workload suite.

Acronym or Abbreviation	Expansion
ROB	Reorder buffer
IQ	Issue queue
LQ	Load queue
SQ	Store queue
RF	Register file
ARF	Architected register file
PRF	Physical register file
FU	Functional unit
SB	Store buffer
IB	Instruction buffer
TVF	Timing Vulnerability Factor (Section 2.5)
AVF	Architectural Vulnerability Factor (Section 2.6)
SER	Soft Error Rate
ACE	Architecturally Correct Execution (Section 2.7)
FIT	Failure In Time (Section 2.1)
MTTF	Mean Time to Failure (Section 2.1)
MTBF	Mean Time Between Failure (Section 2.1)
MTTR	Mean Time to Repair (Section 2.1)
SEU	Single Event Upset
SFI	Statistical Fault Injection (Section 1.1, 2.7.4)
IPC	Instructions Per Cycle
CPI	Cycles Per Instructions

Table 1.1: List of Acronyms or Abbreviations used in this Dissertation.

1.5 List of Acronyms and Abbreviations

A list of acronyms and abbreviations used in this work is presented in Table 1.1, to aid easy reference. Additionally, the expansion of each acronym or abbreviation is repeated in each chapter of this dissertation, to aid clarity.

1.6 Organization

Chapter 2 provides a background for the causes of soft errors in modern microarchitectures, and methodologies to estimate the soft error rate of the

processor while running a workload. A background on the first-order mechanistic modeling of performance of a processor, and other work related to the objectives of this dissertation can also found in Chapter 2. Chapter 3 outlines the simulators, workloads, and evaluation methodology used in this dissertation. Chapter 4 discusses the methodology to generate an AVF stressmark, its evaluation, and application towards design for mitigating soft errors. Chapter 5 elaborates on the mechanistic modeling methodology for AVF, its evaluation, and applications towards design for soft error mitigation. Finally, Chapter 6 summarizes the key results and insights presented in the dissertation, and proposes the directions for future research.

Chapter 2

Background

Radiation-induced transient faults, also known as Single Event Upsets (SEU), occur as a consequence of strikes from energetic sub-atomic particles, such as alpha particles and neutrons. Alpha particles are emitted as a result of radioactive decay of contaminants in the packaging materials. Neutrons that reach the earth's surface typically arise as a result of the interaction between cosmic radiation and the earth's upper atmosphere. Particles with sufficient energy may strike silicon to release electron-hole pairs, which may then be swept into the diffusion region of the transistor in sufficient number so as to register an incorrect value. The lower the threshold and operating voltages, the fewer such electron-hole pairs are necessary to cause a fault. This implies that less energetic sub-atomic particles may cause faults as operating voltages are lowered in future process generations. Soft errors due to neutron bombardment tends to increase with altitude. For example, Denver, CO, situated at 5000 feet above sea level experiences nearly 5× more cosmic radiation flux than New York City, at sea level [15]. Leadville, CO, located at over 10,000 feet above sea level experiences nearly 13× more cosmic ray flux than New York City [15].

Soft errors have long been a problem in space and avionics applications, due to the higher rate of bombardment from cosmic radiation. High-availability and high-reliability mainframes have devoted resources for protection from soft errors at ground level. However, as operating voltages are lowered, as a result of shrinking device geometries, and an acute need to reduce the power consumption of the chip, soft errors are becoming an issue for commodity servers as well. A fault that passes out to the program output without being detected by hardware is termed as a Silent Data Corruption (SDC). If the hardware is equipped to detect and correct this fault, the SDC is eliminated. However, if the system is unable to correct the fault, it may raise an exception, or crash. This is termed as Detected Unrecoverable Error (DUE). High-reliability and high-avaliability systems aim to minimize SDC, and reduce the DUE errors.

2.1 Metrics for Reliability

Mean Time to Failure (MTTF) or *Mean Time Between Failures* (MTBF) is often used to capture the reliability of a system, and is typically measured in terms of the number of failures per year. MTBF is defined as the sum of the MTTF and *Mean Time To Repair* (MTTR). In other words, MTBF is the sum of the system uptime, and downtime. IBM Server group specifies an MTBF for the entire system of 1000 years for SDC, 25 years for system software crashes, and 10 years for application crashes [2, 12]. Each system will have multiple CPUs, and other components, that may fail for many reasons

in addition to SEUs. *Failure in Time* (FIT) is often used to express the SER of a structure, and is a reciprocal of the Mean Time to Failure. Specifically, FIT captures the number of faults occurring in 10^9 hours. Thus, 1 FIT is equivalent to an MTTF of 10^9 hours.

2.2 Incidence of Soft Errors in Real Systems

There have been numerous studies on the incidence of soft errors at or near ground level for SRAMs and DRAMs [3, 15–17]. In 2000, Sun Microsystems reported that the error protection scheme implemented for their SRAM chips on their UltraSPARC II-based servers was insufficient to handle soft errors [18]. In 2005, Hewlett-Packard reported that their 2048-CPU ASC-Q supercomputer installation at Los Alamos National Laboratory, located 7000 feet above sea level, had frequent crashes due to soft errors in its parity-protected board-level cache tag array [18, 19]. With the lowering of operating voltages and more state on chip as a result of complex pipelines, a greater number of latches, flip-flops, and logic are vulnerable to particle strikes, making the processor pipeline vulnerable to soft errors [2, 12, 20–22].

When a large number of such processors are used, such as large datacenters, or supercomputer installations, the combined effect of such faults becomes especially acute. Owing to their regular structure, SRAMs and DRAMs are easier to protect with error detection and correction codes. However, the processor pipeline, or core, has unstructured logic, latches, and flip-flops, which are difficult to protect thus, without incurring significant power, performance,

and area penalties. Fortunately, not all faults occurring in the core necessarily result in soft errors due to masking at the circuit-level, microarchitecture-level, or program-level. Masking of faults may occur due to program characteristics, underutilization of structures in the core, or due to structures that do not contribute to program correctness, such as the branch predictor.

2.3 Modeling Intrinsic SER

This section provides a brief overview of the raw SER estimation methodologies for a bit or circuit element. The first step is to determine the critical charge (Q_{crit}) for the transistors in the circuit. Q_{crit} is defined as the minimum charge produced by an incedent neutron or alpha particle, necessary to flip the state of the transistor. Q_{crit} can be determined from simulation, such as SPICE models, or empirically, by exposing a circuit to elevated levels of neutron and alpha particle bombardment. One of the models used to compute the circuit level fault rate is the Hazucha and Svensson model [23], and is expressed as

$$Circuit\_SER \propto Flux \times Area \times e^{-\frac{Q_{crit}}{Q_{coll}}}$$

where Flux, Area, and Q_{coll} refer to the neutron or alpha particle flux, exposed circuit area, and collection efficiency, respectively. Q_{coll} is defined as the fraction of charge generated by a particle strike that was collected by the transistors and is determined emperically. Q_{crit}, Q_{coll} and Area generally decrease with each new process generation, at different rates for combinatorial, latch and SRAM circuits. Sivakumar et al. [20] find that combinatorial circuits

will see a significant increase in SER, latches will experience some increase in SER, and SRAMs will experience negligible increase in SER with each process generation.

2.4 Masking Effect of the Circuit on SER

Combinatorial logic gates result in three kinds of masking [18, 20, 24]. *Logical masking* occurs if the particle strike affects a portion of the circuit that has no bearing on the output of the gate. For example, a bit flip on one input node of an AND gate, when the other input node is set to 0, is logically masked. *Electrical masking* occurs as a result of attenuation of the pulse created by the particle strike as it propagates through the combinatorial logic, such that it is supressed before it reaches the output of the circuit. *Latch window masking* occurs if the Single Event Upset (SEU) pulse reaches a latch outside its setup and hold time. These factors can be estimated using a simulator, such as SPICE [20].

Timing Vulnerability Factor (TVF), also referred to as timing derating [12], captures the fraction of time for which a circuit is vulnerable to SEUs [18].

Sivakumar et al. [20] note that technology scaling rapidly reduces the size and Q_{crit} of logic gates, as compared to SRAM cells, making them more vulnerable in future process generations. Additionally, electrical masking in logic circuits will decrease in future generations, resulting in increased SER due to the core. Furthermore, in high frequency designs, the ratio between setup +

hold time of latches and flip-flops as compared to the clock period is not small, resulting in a larger TVF [12]. Consequently, Nguyen and Yagil [12] predict that the SER contribution due to static combinatorial circuits will become comparable to that of latches in future process technologies. Sivakumar et al. predicted an exponential increase in the SER due to combinatorial circuits, under the assumption of aggressive processor pipelining with each process generation. Power constraints have meant that very deeply pipelined designs such as the Intel Pentium 4 have fallen into disuse, hence the exact rate of increase is unclear, and is not as much of an issue as predicted. Nevertheless, the pipelines in current processors are generally deeper than those dating back to the late 1990's or early 2000's, with the notable exception of the Intel Pentium 4.

2.5 Masking Effect of Program Execution on SER

The execution of a program on the microarchitecture also determines what fraction of SEUs manifest as errors at the output. Program execution is ultimately the cause of logical masking at the circuit level. Program execution also determines the utilization of structures, and whether a computation affected by an SEU is necessary for correctness of the program. A fault in a storage element at a time when it is not being used, such as an unused entry in the ROB, will not propagate to the output, and is therefore masked. A fault in the unused 32 bits in a 64-bit register holding a 32-bit value will be masked. The former case is considered a time component of masking, wheareas the

latter case is a space component of masking of an active entry [12]. Modern microprocessors contain a significant amount of state devoted to structures that do not affect correctness of the program, but only enhance performance. For example, a fault in an entry of the branch predictor's history table or the Branch Target Buffer may lead to an incorrect control path being fetched. However, this would be detected when the branch is executed, and corrected by discarding the wrong-path instructions, and fetching from the correct path.

This speculative execution is a common feature in modern microprocessors, and creates masking effects of its own. For example, the branch predictor may predict the branch to be taken, whereas the correct control flow of the workload requires it not to be taken. Upon the detection of this "misprediction", the instructions fetched from the wrong path (i.e., the taken path) must be discarded. Any fault affecting the instructions in the wrong path will never be committed to the program output, and is hence masked.

Instructions in the workload may also result in masking. For example, NOPs do not affect the result of computation. Therefore, other than the bits that identify the NOP (corresponding to its opcode), a fault in a NOP entry will be masked. Compilers also introduce instructions to enhance performance that may be dynamically dead. For example, a load instruction may be hoisted past a conditional branch to hide some of its latency, but may be used only along one branch path. If the other branch path is taken, the loaded value would be dynamically dead. Similarly, a function from a dynamically linked library may return a value that is never read, and hence dead. Butts and Sohi

[25] show that 3-16% of dynamic instructions are dead. Instructions whose values are not consumed are called *First Level Dynamically Dead* (FDD). *Transitively Dynamically Dead* (TDD) instruction values are used only by FDD instructions, or other TDD instructions. This is equally applicable to memory operations: stores whose values are subsequently overwritten before they are read are dynamically dead. As the values of these instructions do not affect the output of the program, their correctness is not critical.

Logical masking could occur at the program level. For example, an operation such as `R3 = R1 AND 0x00FF` only retains the lower 8 bits of `R1` and clears the rest, implying that a fault in the higher order bits is masked. Some Instruction Set Architectures (ISA) support predication. Predication attempts to reduce the branch misprediction penalty of hard-to-predict branches by converting a control dependence into a data dependence. Predication makes the execution of an instruction dependent on (or predicated on) the status of a predicate register, set by the branch condition evaluation instruction when it evaluates to true. Instructions for which the predicate register evaluates to false are discarded, thereby masking faults in their results.

2.6 Architectural Vulnerability Factor

Architectural Vulnerability Factor (AVF) expresses the probability that a user-visible error will occur, given a Single Event Upset (SEU) in a bit or storage element [2]. AVF is analogous to derating, or logic derating [12], error cross-section and residency [26], which are terms used depending on the level

of analysis, or institution. This dissertation will consistently use the term AVF. AVF is a property of a hardware bit in a structure, and can be used to compute its soft error rate as follows:

$$SER_{bit} = AVF_{bit} \times TVF_{bit} \times intrinsic\ fault\ rate_{bit} \tag{2.1}$$

where TVF refers to the Timing Vulnerability Factor (see Section 2.4). The intrinsic fault rate is estimated using experimental, or simulation methodologies, such as those outlined in Section 2.3.

The SER of each bit in a chip is aggregated to compute the overall SER for the chip, while running the workload. This method to estimate SER is therefore also called AVF+SoFR, where SoFR stands for Sum of Failure Rates.

2.7 ACE Analysis

AVF expresses the probability that a radiation-induced fault in a bit in a hardware structure will be observable at the output. The computation of AVF requires the determination of whether the corruption of a value contained in the bit will affect the correctness of the program. The greater the fraction of time for which the bit-cell in hardware holds critical values, the lesser is the masking of faults affecting the bit-cell. Although it is possible to determine AVF of a bit in a structure using Statistical Fault Injection (SFI) on a gate-level, or Register Transfer Level (RTL) model, this methodology requires a large number of simulations for statistical significance, and is there-

fore time-consuming. Additionally, RTL models are unavailable during early design planning. Mukherjee et al. [2] propose ACE analysis in order to allow conservative AVF estimation during the early design phase. ACE analysis requires only a single execution of a workload on a microarchitectural model, which is significantly faster than SFI.

2.7.1 Architecturally Correct Execution (ACE) Bits

Mukherjee et al. term bit values — induced by a workload in a structure — whose correctness is essential for the correctness of the program, as *Architecturally Correct Execution* bits or ACE bits [2]. An ACE bit is one whose correctness is required for the correctness of the program. A bit could be either microarchitecturally or architecturally ACE. A microarchitectural ACE bit is not architecturally visible, but its correctness is nevertheless required. For example, the head and tail pointers of the ROB or IQ are microarchitecturally ACE, as any corruption in their contents would lead to incorrect execution of the program, even though the programming model is oblivious to their existence. On the other hand, architecturally ACE bits are directly visible to the programmer, and any corruption in their state may result in incorrect execution. These include corruption in data resident in the ROB, issue queue, register files and caches.

Conversely, Mukherjee et al. term bits that are not critical to program correctness as *un-ACE*. Microarchitecturally un-ACE bits could result from bits that represent unused or invalid state, bits discarded as a result of mis-

speculation, or bits in predictor structures. Architecturally un-ACE bits are a direct result of the instructions in the binary. Examples of these include NOPs, software pre-fetches, predicated false instructions, and dynamically dead instructions. Other than the few bits that are critical for these instructions to be decoded correctly, the bits for both these instructions are un-ACE.

Whether a bit is ACE or not in cache structures depends on the nature of reads and writes to that structure. For example, a store to a location that has been affected by an upset will overwrite the corrupted data; the location, during the period of time leading up to the write, is therefore un-ACE. On the other hand, a load from a location affected by a fault will bring in corrupted data into the processor. Assuming that this load itself is ACE, this location in the cache is also ACE. Biswas et al. [27] introduce the concept of lifetime analysis to determine the ACE-ness of a cache structure. Assuming a writeback cache, a cache-line is ACE between *Fill* $\Rightarrow$ *Read*, *Read* $\Rightarrow$ *Read*, *Write* $\Rightarrow$ *Read* and *Write* $\Rightarrow$ *Evict*.

For Content Addressible Memory (CAM) arrays, assuming a single bit upset model, a corrupted entry could be mistaken for another, if they differ in only one bit position, or Hamming distance of one. Therefore, a per-bit lifetime analysis is performed only on such bits.

2.7.2 Computing AVF using ACE Analysis

Mukherjee et al. formally define AVF of a structure of size N bits, as follows:

$$AVF_{structure} = \frac{1}{N} \times \sum_{i=0}^{N} \left(\frac{ACE\ cycles\ for\ bit\ i}{Total\ Cycles} \right) \tag{2.2}$$

AVF of a structure may thus be expressed as the average number of ACE bits per cycle divided by the total number of bits in the structure. The AVF is multiplied by the circuit-level fault rate (i.e., $TVF \times instrinsic\ fault\ rate$) to estimate the SER of the structure. The SER of each structure in the chip is added, to compute the overall SER of the chip.

ACE analysis is intended to provide a conservative estimate of AVF during early design planning. Thus, it assumes a bit is ACE unless it can be proven otherwise. Similarly, it will make the most conservative assumption regarding the circuit-level masking of soft errors. Additionally, it assumes that the soft error rate is low enough such that the probability of multiple radiation strikes affecting the same bit value or instruction is negligibly small, when used to estimate SER. This is generally true in terrestrial applications, but may overestimate SER under elevated levels of radiation, such as accelerated bombardment of the chip in radiation chambers. Nevertheless, it provides a good estimate of the occupancy of corruptible state in the processor.

It should be mentioned that classical reliability analysis has been applied to computational reliability, at a circuit level. Assuming that errors are exponentially distributed with a constant failure rate λ, the reliability of that

component over a time period of t is modeled as $R(t) = e^{-\lambda t}$. While this model may help estimate the fault rate of each bit, it is unclear whether the distribution holds after the masking effect of logic and program execution is considered [22, 28]. Therefore, using a methodology called SoftArch, Li et al. [22] use the exponential failure rate model to estimate AVF, in a manner similar to ACE analysis. The key difference is that ACE analysis is a point of strike model, directly measuring AVF of the bit based on whether the bit value will eventually affect the output, whereas SoftArch propagates the probability of a fault until it reaches the output through a store operation to memory and then computes the Mean Time to Failure (MTTF) based on the aggregate probabilities of faults of each operation that led to the output. AVF of a structure is then computed using the MTTF.

2.7.3 Limitations of ACE Analysis

ACE Analysis has limitations, owing to its conservatism in computing SER, lack of detail in the the microarchitectural model, and the sum-of-failure rate methodology under certain circumstances. These are outlined below:

- *Practical Memory Limitations:* In theory, ACE Analysis should track instruction dependencies across registers, memory and even I/O devices, such as the disk. In practice, such tracking would require massive amounts of memory, and is therefore impractical. Therefore, a scope is defined, and any instruction that crosses this scope is considered to be ACE. For example, if the scope is defined as the CPU-memory interface,

then all stores to memory are ACE unless demonstrated otherwise. If the scope is extended to include the memory, such that all operations crossing the memory-I/O device interface are ACE, it is possible that instructions and bits that were earlier considered ACE would be un-ACE. In this dissertation, the scope is defined to be the CPU-memory interface, using a methodology similar to that used by Mukherjee et al. [2]. Memory and register dependencies are tracked over a sliding window of 50,000 instructions to limit the memory requirement of ACE analysis. Each bit is assumed to be ACE unless it can be proven otherwise. Doubtless, this introduces conservatism to the SER estimated using ACE Analysis. This limitation is shared with other performance-model based methodologies such as SoftArch [22].

- *Conservatism due to the Microarchitectural Model:* Wang et al. [29] show that AVFs computed using a less detailed performance model may overestimate the resultant SER by two or three times, as compared to Statistical Fault Injection (SFI). As a rebuttal, Biswas et al. [30] show that this detail is easily added, minimizing the overestimation of SER. Additionally, ACE analyis approximates program behavior in the presence of faults; faults that change the behavior of the processor are not modelled. Wang et al. [29] claim that another potential reason for the overestimation of SER using ACE Analysis may be due to so-called "Y branches". In earlier work, Wang et al. [31] report the existence of branches that may be forced down the wrong path due to a transient

fault, but would eventually reconverge with the correct path without affecting the correctness of the output. They refer to such branches as "Y branches". ACE analysis conservatively assumes all branches to be ACE, and will execute correct path instructions. Thus, it does not account for such behavior. It is expected, however, that such branches are relatively infrequently observed [18, 30], but can be accounted for using additional analysis. This limitation is shared with other methodologies, such as SoftArch [22].

- *Sum of Failure Rates:* Using SoftArch, Li et al. [28] argue that ACE analysis will overestimate the SER in systems with an extremely large number of components (such as tens of thousands of processors), or at extremely high flux density of particle bombardment (such as in a radiation chamber), or workloads with extremely long phases that are extremely different from one another. They acknowledge that for typical applications, ACE analysis provides is accurate. The potential cause of this discrepency is due to the occurrance of multi-bit faults. A single entry in a structure may be hit multiple times over its lifetime under elevated levels of radiation, and ACE analysis is unable to account for temporal multi-bit faults (a limitation shared with SoftArch). Sum of Failure Rates assumes that the faults in each structure are independent of those in other structures. It is possible, however, that faults in multiple different structures combine to produce only one error. Due to the propagated fault model used in SoftArch, it is able to account for

the joint probability of multiple faults affecting a single output value, whereas Sum of Failure Rates cannot. Nevertheless, for practical purposes, the conservativeness of ACE analysis and sum of failure rates may be sufficient.

The conservatism of ACE analysis in determining SER is still reasonable if the designer is interested in bounding SER. ACE analysis is utilized in this dissertation to estimate the occupancy of corruptible state in the core, and not to measure the actual SER. For the former purpose, ACE analysis is accurate. SoftArch, or any other methodology may be used instead of ACE analysis to achieve the same objectives.

2.7.4 Limitations of SFI

It is also pertinent to note the limitations of more direct methods such as Statistical Fault Injection using RTL models. SFI overcomes some of the limitations of ACE analysis, or SoftArch, due to its low-level detail, but introduces some of its own. Due to the onerously long runtimes (orders of magnitude slower than performance models), and the need to perform multiple simulations for statistical significance, a workload is simulated only for 1,000 – 10,000 cycles [18]. It often takes much longer for a fault to propagate to the program output, so that can be confirmed that it will not be masked. In contrast, microarchitectural models are run for billions of cycles, providing a better view of this masking effect. Additionally, SFI will run two copies of the RTL simulation: one fault-free, and the other into which faults are injected,

and the output is compared to detect the incidence of a fault. At the end of this relatively short simulation, all mismatches in state, whether architectural or microarchitectural must be conservatively treated as if they are errors (i.e., ACE). Some of these errors may be masked due to future operations in the workload, but practical limitations obviate the long simulation time required to establish this fact. SFI may thus be reasonable to conservatively compute the AVF of structures such as pipeline latches, in which the faults quickly be propagated to the registers, or get masked. For structures such as caches or register files in which a faulty value can remain for a long time before manifesting as an error, or being masked, SFI may result in pessimistic results. As noted earlier, RTL models are available only after the design has been finalized, and implemented, making it a poor choice for early design planning. The cost of changing the architecture at this stage to improve reliability may be excessively high.

2.8 Mechanistic Modeling of CPU Performance

Karkhanis and Smith [32] devise a first-order analytical model for estimating the performance of an out-of-order processor, which is refined by Eyerman et al. [33] using *interval analysis*. Eyerman et al. [33] report a 7% error in estimating Cycles Per Instruction (CPI) on a 4-wide machine. Interval analysis models the program execution as an ideal, miss-free execution, interrupted by miss events that would disrupt the dispatch of instructions, as

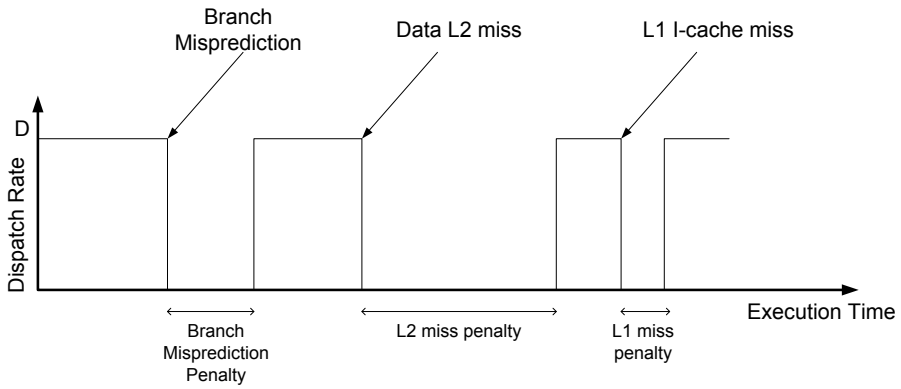


Figure 2.1: Interval Analysis for Modeling Performance.

illustrated in Figure 2.1. In the absence of any miss events, the processor is able to dispatch instructions at the maximum dispatch rate D[1]. The stall-free or ideal execution represents the case in which the processor has no stalls due to miss events. This dissertation uses the term instruction window to refer to the instructions in flight, or the ROB. The following discussion is intended to provide the reader with a basic understanding of Interval Analysis.

Each miss event interrupts the dispatch of instructions until it resolves. An interval begins with the resolution of an earlier miss event, and ends just before the resolution of the next miss event, penalizing the performance of the processor in proportion to its miss penalty. As an out-of-order processor can extract Memory-Level Parallelism (MLP), and nearly all of the latency of the overlapped miss is hidden behind that of the non-overlapped data L2/TLB miss, it is sufficient to count only the non-overlapped data L2 and TLB miss

[1]Note that D may be less than the peak designed dispatch width if the program lacks sufficient inherent Instruction Level Parallelism (ILP).

cycles towards estimating performance, for a given instruction window size. Shorter latency data cache miss events that cannot individually stall the processor, such as a data L1 cache miss that hits in the L2 cache can be modeled in the same manner as arithmetic instructions. The branch misprediction penalty for an out-of-order processor is modeled as the sum of the front-end pipeline depth, and the branch resolution penalty. The branch resolution penalty is the number of cycles between the mispredicted branch entering the instruction window, and the misprediction being detected. The following section explains each event or interval in detail.

2.8.1 Steady State, or Ideal Execution

Given an instruction window of size W, the total number of cycles taken to execute all instructions in the instruction window is a function of the latency of executing the critical dependence path. The average critical dependence path length $K(W)$ for a given program being executed by a processor with instruction window size W is modeled as $K(W) = \frac{1}{\alpha}W^{1/\beta}$ [32, 34] where α and β are constants that are determined by fitting the relationship between $K(W)$ and W to a power curve. This analysis is performed assuming that all instructions have unit latency. Therefore, given an average instruction latency l, the critical path would require $l.K(W)$ cycles. Using Little's law, the ideal Instruction Level Parallelism (ILP), or Instructions Per Cycle (IPC) (i.e., $I(W)$) that can be extracted from the program given an instruction window of size

W is presented in Equation 2.3 [32, 34].

$$I(W) = \frac{W}{l.K(W)} = \frac{\alpha}{l}.W^{(1-1/\beta)} \tag{2.3}$$

Setting β to 2 in Equation 2.3, the available ILP for a given workload has a quadratic relationship with the instruction window size [34]. Earlier studies by Riseman and Fisher [35] emperically obtain this approximately quadratic relationship. For the SPEC CPU2006 workloads in this dissertation, β varies between 1.24 and 2.40. The larger the value of β, the shorter the critical dependency path $K(W)$, and greater the available ILP.

The workload is profiled using a functional simulator, over a range of instruction window sizes W to determine the corresponding critical dependence path length $K(W)$. The profiler sweeps over the entire length of the workload using a sliding window of size W_{max}, simultaneously recording statistics for all instruction window sizes, from 1 to W_{max}. This profiling is a one-time effort for each workload. This relationship between $K(W)$ and W over a range of instruction window sizes is fitted to a power curve to obtain α and β.

2.8.2 Non-Overlapped Long-Latency Data Cache Misses

Non-overlapped last-level cache misses or data Translation Lookaside Buffer (TLB) misses can stall the dispatch of intstructions until they are resolved. Each non-overlapped miss blocks the dispatch of instructions for a period of time equal to its miss penalty. In order to determine the non-overlapped last-level data cache or TLB misses, the workload is profiled using a cache sim-

ulator, over a range of instruction window sizes, in a single pass, using a sliding window similar to that for estimating $K(W)$. The number of non-overlapped data L2 and TLB misses are recorded for each window size. Thus, the profile only needs to be rerun if the cache hierarchy is changed. Fortunately, the number of practical cache hierarchies possible is limited.

2.8.3 Branch Misprediction Penalty

Branch misprediction penalty can be considered as the cost of opportunity of an incorrectly predicted branch. All instructions fetched after the mispredicted branch are incorrect, and hence the pipeline is flushed after the branch is detected, and the pipeline is refilled from the correct path. Thus, the branch misprediction penalty is equal to the sum of the branch resolution time and the pipeline refill penalty. Pipeline refill penalty is equal to the front-end pipeline depth. The branch resolution time is the amount of time between the branch entering the instruction window, and the misprediction being detected. Karkhanis and Smith [32] show that, assuming an earliest-first issue policy, the mispredicted branch is among the last correct-path instruction to be executed, and that the impact of issual of wrong-path instructions on the branch resolution time is minimal. Karkhanis and Smith estimate the branch resolution time as a leaky bucket algorithm[2] in which correct-path instructions

[2]The leaky-bucket algorithm is so called, because it parallels the behavior of a bucket full of water with a small hole at the bottom. The pressure at the hole is maximum when the bucket is full, and hence the volume of water leaving the hole is large. As the bucket drains, the pressure at the hole decreases, and the volume of flow out of the hole also decreases.

are continually being drained from the pipeline until the misprediction is detected, triggering a pipeline flush. For a designed dispatch width D, setting $I(W) = D$ in Equation 2.3, the number of instructions in flight during ideal execution is as follows:

$$W(D) = \left(\frac{l.D}{\alpha}\right)^{\frac{\beta}{\beta-1}} \tag{2.4}$$

At clock cycle $t = 0$, when the mispredicted branch instruction enters the instruction window, there are $W_{t=0} = W(D)$ instructions in flight during ideal execution. During each subsequent cycle, D instructions are issued, reducing the number of unissued correct-path instructions in flight, and thus $W_{t=1} = W_{t=0} - D$. The issued instructions will quickly be retired (under the assumption of ideal, or miss-free execution). $W_{t=1}$ is substituted into Equation 2.3 to determine $I(W_{t=1})$. If $I(W_{t=1}) \geq D$, then D more instructions are issued in the subsequent cycle, and the process repeats. Note that due to the earliest-first issue policy, correct-path instructions are prioritized to be issued over wrong-path instructions. At some stage, the number of correct-path instructions in flight becomes less than the minimum number necessary to support the designed issue/dispatch width. Therefore, applying the formula $W_{t=n+1} = W_{t=n} - min(D, I(W_{t=n}))$ iteratively until only one correct-path instruction remains in the pipeline, the branch misprediction penalty is modeled using this leaky-bucket algorithm. Eyerman et al. [33] refine this modeling by considering the clustering of mispredicted branches. A branch misprediction that occurs immediately after another branch misprediction will have fewer instructions in flight when it enters the instruction window, and hence lower

Abbreviated Event Name	Event Definition
ideal	Steady-state execution, in the absence of instruction-cache misses, branch mispredictions or long-latency data cache misses
brMp	Branch Mispredictions
IL1Miss	L1 I-cache misses that hit in L2
IL2Miss	I-cache misses that also miss in L2
ITLBMiss	ITLB misses
DL2Miss	Non-overlapped data L2 cache miss
DTLBMiss	Non-overlapped data TLB miss

Table 2.1: Definition of Events for Interval Analysis

misprediction penalty.

Branch misprediction statistics can be computed using a simple branch predictor profiler. In practice, there only are a limited number of reasonable branch predictor configurations possible.

2.8.4 Instruction Cache and TLB misses

Instruction cache (I-cache) misses and TLB misses interrupt the dispatch of instructions. As the time required to drain the front-end pipeline is roughly equal to the time required to refill it after the I-cache miss resolves, the miss penalty of an I-cache or I-TLB miss is equal to its latency. I-cache and I-TLB miss count can be determined through simple cache simulation.

2.8.5 Estimating Cycles Per Instruction

The total number of cycles for executing a program is modeled as $C_{total} = C_{ideal} + C_{IL1Miss} + C_{ITLBMiss} + C_{brMp} + C_{DL2Miss} + C_{DTLBMiss}$. The

expansions of the abbreviated event names in the subscript can be found in Table 2.1. Miss events that would not interrupt dispatch, such as data cache hits, are modeled similar to arithmetic instructions. The model assumes a balanced microarchitecture design; specifically, that the processor would not frequently stall in the absence of miss events, while running typical workloads. Karkhanis and Smith [32], and Eyerman et al. [33] demonstrate that it is sufficient to model these intervals as being independent of one another, with little loss in accuracy. This key simplifying assumption does not hold true for occupancy. For example, a mispredicted branch that is dependent on an L2 data cache miss significantly reduces the occupancy of correct-path bits in the shadow of the L2 miss, and is non-trivial to estimate using the existing interval analysis model or aggregate metrics.

A single profile can be used to perform parametric studies on ROB size, issue width, and the latencies of instructions, caches, TLBs and main memory. If the cache hierarchy or the branch predictor is changed, the corresponding profiler would need to be rerun. As noted earlier, there are only a limited number of practical cache hierarchies and branch predictors, and once the database is populated, all estimations of Cycles Per Instruction (CPI) are nearly instantaneous.

2.9 Related Work

This section contrasts prior attempts at estimating the worst-case SER due to program execution, with the AVF stressmark generation methodology

presented in this dissertation. Prior work on modeling the AVF of a microarchitecture are also presented, and contrasted with the mechanistic model presented in this dissertation.

2.9.1 Estimating the Worst-Case Observable SER

There has been some prior work on attempting to increase the visibility of radiation induced faults at the program output. Kellington et al. [36] and Sanda et al. [37] study the soft-error tolerance of the IBM POWER6 processor under a radiation beam. They use a proprietary validation software called *Architectural Verification Program* (AVP) which injects random instructions into the core, and detects errors on the fly. They report that AVP injects roughly 20% un-ACE bits, and mainly exercises the core and not the caches. Due to the proprietary set-up of AVP, the extent to which it exercises the core is unkown, but it is reasonable to expect that completely random injection of instructions, even if they were all ACE, would likely not maximize the corruptible state resident in the processor.The precise factors affecting occupancy of state in the processor to estimate the maximum occupancy of corruptible state is developed and presented in this dissertation. It is therefore extremely unlikely that AVP would chance upon the exact combination of factors that maximize the visibility of soft errors. Even with the incorporation of a machine learning methodology, a systematic approach, such as the one presented in this dissertation, would be superior in terms of convergence time, and confidence in the result. Circuit level techniques have been proposed, such as work

by Sanyal et al [38–40]. However, this does not consider the masking effect of program execution, and cannot be used during the early design stage.

Joshi et al. [41], Polfliet et al. [42], and Ganesan et al. [43, 44] utilize genetic algorithms to develop stressmarks for power and thermal stressmarks. Their methodology cannot be directly used for AVF, since it has no means of capturing ACE and core or cache occupancy. Furthermore, they rely on microarchitecture independent program characteristics, which are not not useful, since AVF is strongly dependent on microarchitecture. The AVF stressmark methodology creates a code generator that is expressly designed for generating an AVF stressmark, by working from first principles. Consequently, most of the knobs used, and the nature of the code generator, are significantly different.

Other methodologies to estimate AVF, such as SoftArch [22], or *Program Vulnerability Factor* (PVF) [45] and *Hardware Vulnerability Factor* (HVF) [46], may be utilized instead of ACE Analysis, to compute the Soft Error Rate (SER) of a given microarchitecture. These methodologies by themselves cannot be used to estimate the worst-case SER, as they are themselves dependent on workloads, just as is the case with ACE analysis.

2.9.2 Analytical Modeling of AVF and SER

Mukherjee et al. [2] use Little's Law as a high-level technique to estimate occupancy of state in the structure; however, this methodology still requires detailed simulation to extract the Instructions Per Cycle (IPC) and the average latency of each correct-path instruction in each structure. Com-

puting the latter from profiling is non-trivial for an out-of-order processor due to overlapping of some execution latencies, and dependence on the latencies of other instructions in that structure. Furthermore, it fails to provide insight into the fundamental factors affecting the occupancy of correct-path state beyond aggregate metrics.

As AVF represents the combined effect of the workload and its interaction with the hardware, Sridharan and Kaeli [45] attempt to decouple the software component of AVF from the hardware component through a microarchitecture-independent metric called *Program Vulnerability Factor* (PVF). PVF has been shown to model the AVF of the Architected Register File using inexpensive profiling. However, for estimating the AVF of other structures, their methodology relies on the estimation of *Hardware Vulnerability Factor* (HVF) [46], which in turn requires detailed simulation, and thus provides less insight than a well constructed mechanistic model. Sridharan and Kaeli have shown that HVF correlates with occupancy of structures such as the ROB, and hence it is expected that the mechanistic modeling methodology presented herein can be used to model the HVF of the applicable structures.

Fu et al. [14] report a "fuzzy relationship" between AVF and simple performance metrics. Therefore, black-box statistical models for AVF that utilize multiple microarchitectural metrics have been proposed by Walcott et al. [9] and Duan et al. [10] for dynamic prediction of AVF. These models use metrics such as average occupancy, and cumulative latencies of instructions in various structures as inputs to the statistical model. However, these metrics

are not available without detailed simulation. Cho et al. [11] utilize a neural-network based methodology for design-space exploration, and use it to model AVF of the Issue Queue. As each workload is associated with its own neural network model, training it would potentially require a significant amount of detailed simulations. All these models combine the software and hardware component of AVF, and do not uncover the fundamental mechanisms influencing AVF, thereby providing less insight than the approach presented in this dissertation. As the methodology presented herein derives the factors affecting AVF from first principles that explicity models this fuzzy relationship, it enables the architect to identify the precise cause of high or low AVF in a particular structure, and characterize workloads for AVF.

Chapter 3

Methodology

This chapter provides an overview of the simulators, workloads and evaluation methodology used in this dissertation.

3.1 Simulators

Two simulators that simulate microarchitectures using the Alpha ISA are used to produce the data presented herein. SimSoda [47] simulator performs ACE analysis and is built on top of the SimAlpha [48] simulator. SimAlpha models the Alpha 21264 microarchitecture in great detail, and has been validated for integer microbenchmarks against an Alpha 21264 processor. However, it lacks flexibility and models features that are unique to the microarchitecture of the Alpha 21264. Therefore, SimpleScalar [49] is used when a more generic and flexible microarchitecture model is required. SimpleScalar, however, models the ROB, IQ and RF in a single structure called the *Register Update Unit* (RUU). It also implements a unified *Load and Store Queue* (LSQ). As modern microprocessors do not use a unified RUU, instead preferring a separate ROB, IQ and register file, the simulator needs modification to be representative of current microarchitectures. Additionally, many microar-

chitectures implement a separate load queue and store queue. The results presented in this disseration using SimpleScalar are generated using a modified version of the simulator that implements separate ROB, IQ, RF, LQ and SQ.

3.2 ACE Analysis

In order to compute the AVF of a structure using ACE analysis on a performance simulator using Equation 2.2, the performance simulator must provide the sum of residence cycles for ACE bits in that structure, the total number of elapsed cycles of execution, and the total number of bits in the structure. Performance models provide the total number of elapsed cycles, and the number of bits in the structure is known at design time. Therefore, ACE analysis only requires the additional task of computing the residence cycles of ACE bits in the structure.

In order to determine the residency of ACE bits in the structure, the performance simulator counts the number of cycles for which an instruction is resident in the structure. The instruction may then be committed eventually, or quashed as a result of a misprediction. If the instruction is committed, it is put into a post-commit analysis window that tracks whether the instruction is dynamically dead, or is logically masked. This post-commit analysis window may be thousands of instructions in size; a window size of 50,000 instructions is assumed. Mukherjee [18] states that a window of a few thousand instructions is sufficient to capture most of the dynamically dead instructions, and logical

masking.

The post-commit analysis window maintains a list of instructions to be analyzed, along with their corresponding per-structure residence cycle counts, and a table specifying producers and consumers of each instruction within the analysis window. Instructions are inserted into the analysis window at commit-time in program order, and instructions that are older than the size of the analysis window are removed. When a dynamically dead instruction is found, all instructions that exclusively depend on dynamically dead instructions are also marked dynamically dead. Instructions that are removed from the analysis window are then checked for ACE-ness; if they were not dynamically dead or masked, the residence cycle count for each structure is added to the ACE cycle counter corresponding to that structure. Thus, the methodology assumes that each instruction is ACE unless it can be proven otherwise. Additional detail may also be added to ACE analysis at the bit level granularity, for increased accuracy. For example, branch instructions do not require a destination register specifier, and thus, the fields in the ROB that correspond to the destination register specifier are not ACE. As discussed earlier in Chapter 2, addition of such detail reduces the overestimation of AVF [30].

3.3 Genetic Algorithm

Genetic Algorithms (GA) are global optimization heuristics used to find optimal solutions to complex problems. Genetic algorithms mimic biological systems in nature. Using the principles of natural selection, biological systems

seek to improve the quality of their gene pool. Genetic algorithms similarly evolve an optimal solution from a set of random values, or initial seed values. Each set of values is called an individual, and each individual is composed of multiple "genes". Each gene influences an aspect, or part, of the overall solution. A string of genes are collecively referred to as a chromosome for the individual. A set of individuals together constitute a generation of the population. The individuals in the generation are evaluated for fitness; for example, if the objective is to maximize a function, then individuals that produce higher values for the function are more fit. Fitter individuals are given a higher probability to propagate their genes. Conversely, unfit individuals are less likely to be selected for breeding, and will likely die out. A whole new population of solutions is thus generated by mating the highly fit individuals of the current generation with each other.

Just like in biological systems, individuals in each generation are "reproduced" or "cross-bred" by combining parts of the genes of the two individuals to produce a new individual. This process is referred to as *crossover*. Two individual chromosomes are cut at a random location, producing head and tail chromosomes. The tails of the two individuals are swapped to produce two new individuals, each containing some genes from each parent. Crossover is not applied to every pair of individuals selected for mating; rather, a random choice is made to perform reproduction, with a probability of between 0.6 and 1 [50]. If a crossover is not performed, the offspring are simply duplicates of their parents, giving each individual a chance of propagating their genes

without any crossover. Additionally, to avoid being stuck in local maxima or minima, the genetic algorithm also introduces "mutation", which involves the introduction of random changes to genes in an individual. The probability of mutation is selected to be less than 0.05, to avoid excessively random variations in population, and allow for a gradual evolution of a solution through reproduction (i.e., crossover). As the population matures, the average fitness of the population begins to approach the most fit individual found thus far. A gene is said to have converged when 95% of the population shares the same values. A population converges when all the genes have also converged. To further avoid being stuck in local maxima or minima, the GA may introduce cataclysmic events. During a cataclysm, the best solution in the population is selected, and placed in a completely new population of randomly seeded individuals, and the process is restarted.

Genetic Algorithms (GA) have been shown to successfully deal with a wide range of problem areas, that are particularly difficult to solve using other methods [50]. GA's are not guaranteed to produce the absolute global optimum solution to a given problem. Nevertheless, they are good at finding satisfactorily good solutions, acceptably quickly with little intervention. Genetic algorithms to maximize the occupancy of state in the core are utilized. This involves complex tuning of program characteristics that cannot be expressed in a manner that is amenable to mathematical optimization techniques, making the GA an ideal candidate for such optimizations. This dissertation uses the IBM SNAP Genetic Algorithm, obtained under NDA for university research.

3.4 Evaluation Methodology

The methodologies presented in this dissertation are evaluated using SPEC [51] CPU2006 benchmark suite, and MiBench [52] benchmark suite. SPEC CPU2006 is an industry standard benchmark suite for comparing the performance of high performance processors, and has a large memory footprint. MiBench is a benchmark suite used to evaluate embedded system processors and has a small memory footprint. Consequently, SPEC CPU2006 workloads have dynamic instruction counts of trillions of instructions, whereas MiBench workloads have dynamic instruction counts of millions of instructions. The working set, and typically, the memory footprint, of MiBench workloads fits in the last-level cache of high performance processors, which is not the case for many SPEC CPU2006 workloads.

Owing to the large dynamic instruction count of SPEC CPU2006 workloads, it is impractical to run the entire workload on a performance simulator. Therefore, it is necessary to be able to run representative traces of the workload. The SimPoint [53] methodology was devised to address the issue of identifying representative traces. The workload is broken into equal intervals of execution. Each interval is profiled to identify constituent basic blocks. A basic block is a region of execution of code that has exactly one control flow entry and exit. Simpoint profiling identifies the basic blocks in each chunk of execution, to produce a basic block vector. Using machine learning, SimPoints methodology clusters these basic block vectors based on their similarity to one another, and picks a representative interval from each cluster. Clusters are

assigned weights based on the number of intervals contained in each. Each representative interval is run on a performance simulator, and the relevant statistics are collected. A weighted average of the relevant statistics provides an accurate estimation of the same statistics obtained using a complete run of the workload. A single representative interval may also be picked using this methodology; this is called Single SimPoints. Single SimPoints methodology is used to evaluate SPEC CPU2006 workloads. The interval size is chosen to be 100M instructions, as proposed in the original SimPoint work [53].

Chapter 4

An Automated Methodology for Bounding Microprocessor Vulnerability to Soft Errors

In this Section, an automated methodology for bounding microprocessor vulnerability to soft errors is presented. Starting from the first principles of superscalar execution, a set of microarchitecture-dependent characteristics that maximize the occupancy of state in the major structures of the processor is identified. A code generator that manipulates these characteristics based on its inputs, or "knobs" is developed, and interfaced with a machine learning algorithm in a closed-loop feedback process. Upon the convergence of the machine learning algorithm, the workload generated by the code generator is shown to induce significantly higher Soft Error Rate (SER) than the highest SPEC CPU2006 or MiBench workload.

The significant contributions of this work are as follows:

1. A flexible and automated methodology to generate an AVF stressmark is developed. This AVF stressmark is designed to approach the maximum observable SER for a given microarchitecture.

2. The deficiencies in current methodologies for the estimation of the observable worst-case SER are highlighted. Also highlighted are the poten-

tial pitfalls of soft-error reliability design without the knowledge of the observable worst-case SER. The knowledge of the observable worst-case SER enables designers to quantify design trade-offs such that their SER design objectives can be met efficiently.

4.1 Issues affecting SER benchmarking

Prior research [2, 8] has shown that masking effects of program behavior have a significant impact on the visibility of faults to the user. *Architected Vulnerability Factor* (AVF) modeling, which quantifies this masking effect, enables architects to determine the highest per-structure SER observed while running typical workloads. The observable SER of a workload is strongly dependent on the microarchitecture and underlying circuit-level fault rates. Different programs stress microarchitectural structures differently, and hence a change in microarchitecture or underlying fault rate alters their observed SER by different proportions. A workload suite that offers adequate coverage on one microarchitecture and circuit-level fault-rate does not neccessarily do so when either factor is changed.

There is no known methodology to ensure that the benchmark suite covers the entire range of observable SER, from zero to the worst-case observable SER. Therefore, architects run a large number of programs in the hope that sufficient coverage is achieved. Architects choose the SER design objective appropriate for the usage environment, such as design for the average workload-induced SER, or for the highest workload-induced SER. A safety

margin is added to determine the design point, to cover for the possibility of inadequate SER coverage and representativeness of the workload suite [12]. The choice of this safety margin is largely based on designer intuition, and it is difficult to know whether it is adequate. Figure 1.1 represents two workload scenarios with different SER coverages. The arrows represent the range of SER observed while running programs in the workload suites. The workload suite in scenario 1 has good SER coverage, whereas the workload suite in scenario 2 does not. Suppose that the architect is designing for the highest workload-induced SER. As shown in Figure 1.1, the addition of the safety margin to the highest workload-induced SER pushes the design point well beyond the worst-case observable SER, leading to over-design. On the other hand, the safety margin for scenario 2 is insufficient to cover for the worst-case. In the absence of a methodology for determining the worst-case SER, it is impossible to know whether these safety margins are excessive, or inadequate. On similar lines, the architect may choose to design for the average-case workload, or some percentile of the workload suite. Consider Scenario 1 which has a relatively high average SER. An aggressive safety margin over the average case in Scenario 1 may push the design point close to, or beyond the worst-case SER, leading to over-design. On the other hand, an aggressive safety margin is required in Scenario 2 to cover for its lack of adequate SER coverage. The knowledge of the worst-case SER allows thus the architect to rationalize about the amount of the safety margin necessary, and define the design point relative to the worst-case SER and the design objective. The knowledge of the worst-case

SER also indicates whether the workload suite needs additional benchmarks to make up for its lack of SER coverage. It is expected that designing for the worst-case SER will increase in significance in future technologies, due to elevated levels of SER as a result of aggressive lowering of operating voltages to reduce power consumption.

4.2 Difficulties in determining the worst-case SER

It is impossible for every bit in the processor to simultaneously have 100% AVF while running a program: structures in processors are typically over-designed to handle bursty program behavior, and have interdependencies such that all of them cannot contain useful program state simultaneously. This suggests that the overall worst-case SER calculated by adding up circuit-level fault rates of individual circuits, without considering the masking effect of program behavior would lead to an overly pessimistic design. For similar reasons, it would be incorrect to estimate the worst-case by adding up the highest per-structure SER observed using AVF modeling.

Therefore, there is a need to determine the highest observable SER in a holistic manner. A workload that exposes this highest observable SER is referred to as a stressmark, drawing an analogy with power or thermal stressmarks (also called viruses), which are designed to maximize power and temperature of the processor, respectively. As every gate in the circuit cannot be toggling simultaneously, the power or thermal virus focusses on instructions that maximize overall power dissipation or temperature.

Parameter	Baseline
Integer ALUs	4, 1 cycle latency, 64 bit wide
Integer Multiplier	1, 7 cycle latency, 64 bit wide
Fetch/slot/map/issue/commit	4/4/4/4/4 per cycle
Integer Issue Queue	20 entries, 32 bits/entry
ROB	80 entries, 76 bits/entry
Integer rename register file	80, 64 bits/register
LQ/SQ	32 entries each, 128 bits/entry
Branch Predictor	Hybrid, 4K global, 2 level 1K local 4K choice
Branch Misprediction Penalty	7 cycles
L1 I-cache	64kB, 2-way, 64B line, 1 cycle latency
L1 D cache	64kB, 2-way, 64B line, 3 cycle latency
DTLB	256 entry, fully associative, 8kB page
L2 cache	1MB, direct mapped, 7 cycle latency

Table 4.1: Baseline Configuration of Processor.

A comprehensive methodology that simultaneously increases the AVF of multiple structures in the processor such that the observable SER approaches the maximum is developed herein. The search space for such a program is large and complex. Starting from first-principles, a set of microarchitecture dependent factors that affect the occupancy of useful state in the processor is derived, and used to develop a code generator that defines a feasible search space. A Genetic Algorithm (GA) to explore this search space and generate the stressmark is then used. When the GA has converged, the resulting workload will induce an SER that approaches the maximum observable SER.

Interdependence of the AVF of Processor Structures

Occupancy, and hence AVF of structures in an OoO processor are not completely independent of one another. This interdependence also ensures that all bits in the processor cannot be ACE simultaneously.

Consider the Alpha 21264 whose configuration is outlined in Table 4.1. Every instruction in the ReOrder Buffer (ROB) must exist in either the Issue Queue (IQ), Load Queue (LQ) or Store Queue (SQ), or have been executed in the Function Units (FU). However, the total number of entries in the integer IQ, LQ and SQ alone is more than the size of the ROB, implying that the ROB, IQ, LQ, SQ and FU cannot simultaneously have 100% AVF.

The number of rename registers in use depends on the number of instructions in flight, and hence the occupancy of the ROB. Unlike architected registers, rename registers cannot hold ACE data all the time. Many rename registers hold values that are quickly consumed, and not read again. The process of retiring, releasing, re-assigning and writing to a rename register file takes multiple cycles, and hence AVF of the physical register file is never 100%. Additionally, stores and branch instructions do not write ACE data to a rename register.

The interdependence in occupancy also implies that assuming that the instantaneous occupancy of ROB and IQ, and the Instruction mix (I-mix) are known, the occupancy/utilization of LQ, SQ, FU and rename RF can be bounded, thereby bounding AVF. Additional information about the proportion

of ACE instructions in each type (load, store, arithmetic) allows a tighter bound on AVF.

FU utilization is maximum when the processor can issue arithmetic instructions at maximum bandwidth. However, LQ and SQ occupancy will be lower, since the instruction mix has fewer loads and stores.

The Alpha 21264 also allows only two memory instructions to issue per cycle, restricting the rate at which they can be filled with ACE bits.

It is clear from the above example that simply adding the circuit-level fault rates of individual structures, or the highest per-structure SER, to calculate worst-case SER would be incorrect. This is generally true of any other microarchitecture as well. There is therefore a need for a methodology that addresses the issue of quantifying the observable worst-case SER.

4.3 Design of the Code Generator

In this section, the methodology used to build a code generator for AVF stressmarks is described. This code generator must be provided with knobs to control various parameters. The knobs are used to interface the code generator with a Genetic Algorithm (GA) tool, which then controls the characteristics of the output program. Figure 4.1 outlines the framework for stressmark creation. In the first step, the Genetic Algorithm produces a set of knob values, that is used by the code generator to create a candidate stressmark. In the next step, this candidate stressmark is compiled and run on a simulator for measuring

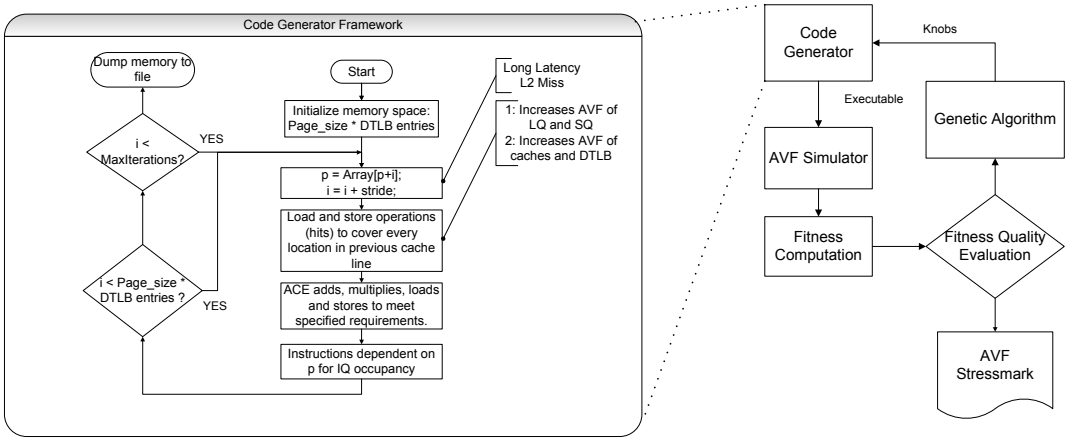


Figure 4.1: Methodology for creation of an AVF stressmark.

AVF. In the following step, the output of the simulator is evaluated by a fitness function, which evaluates whether the output has converged. The result is fed back into the GA, and the above steps will be repeated until convergence is achieved, or a maximum number of runs are reached. Additionally, the code generator must ensure that every instruction is ACE so that entries in the core and cache are also ACE. In order to define the knobs for the code generator, the microarchitecture-dependent program characteristics that affect occupancy in cores and caches are studied.

Microarchitectural structures are classified into Queueing Structures (QS) and storage structures. For queueing structures such as the IQ, LQ, SQ, ROB, and FU, the AVF is proportional to occupancy, if the proportion of ACE bits in the program is kept fixed. This correlation between AVF and occupancy has been utilized to predict AVF of such structures [9, 10].

For storage structures, overall occupancy does not necessarily correlate to AVF, since data in cache lines may switch between being ACE and un-ACE, depending on access patterns. For caches, AVF is influenced by the working set size [27] and coverage of cache locations.

4.3.1 AVF due to Microarchitecture-Dependent Behavior

The factors that affect the overall occupancy of queuing structures, and liveness of caches are also to be considered while designing the code generator. For the discussion below, assume that the instruction stream has a constant proportion of ACE bits. This analysis is used to determine the factors that are required to be controlled in a code generator that increases AVF in a core.

4.3.1.1 Long-Latency Operations

A long-latency operation, such as an L2 or DTLB miss, double-precision divide, or square root may cause the processor to eventually stall (if their latencies are not overlapped with another long-latency operation). Consider the example of an L2 miss. Typically, in the shadow of an L2 miss, the ROB fills up completely, and all FU activity ceases eventually. The IQ contains instructions dependent on the L2 miss. The LQ data array corresponding to an issued load contains ACE bits only after the data has been brought from the memory hierarchy; until then, only the tag array holds ACE bits.

4.3.1.2 ILP and instruction latency

Low ILP and/or higher instruction latency increases the occupancy of the IQ. Higher instruction latency increases the occupancy of ROB, LQ and SQ, provided that the IQ is not full. Since FUs have fixed latencies, the only way to increase occupancy is through maximum IPC (high bandwidth, per Little's law).

4.3.1.3 Instruction Mix

Instructions in the ROB are distributed among the FUs, LQ and SQ, and an increase in one type of instruction will cause an increase the average occupancy of its corresponding unit, and a proportionate decrease in the occupancy of the others. The size of operands used also affects the ACE-ness of entries in the load queue, store queue and register file. For instance, a 32-bit store instruction on a 64-bit machine would have the other 32-bits as un-ACE, thereby lowering its AVF [2]. As the LQ and SQ typically contain more bits than function units, programs that have a greater proportion of loads and stores will have more corruptible state in the processor, all else being equal.

4.3.1.4 Front-End Misses

I-cache misses, I-TLB misses and fetch inefficiency reduce AVF of all structures by reducing the supply of useful instructions. In the case of a branch misprediction, all instructions fetched along the wrong path are un-ACE, and the subsequent pipeline flush reduces the occupancy of the queues.

4.3.1.5 Cache Coverage and Working Set

The AVF of a cache depends on the number of cache lines that contain ACE data, and the duration for which the lines are ACE [27]. A high number of accesses to a few cache lines will give a high hit rate, but low AVF. On the other hand, a high miss rate could also result in high AVF, if the evicted lines, and the filled lines replacing them are ACE. The working set may also be fragmented due to the cache line; a strided access pattern may not use every memory location in the cache line, and hence only a part of the line will contain ACE bits.

Additionally, the compiler introduces un-ACE instructions such as NOPs for alignment of loops to cache line boundaries, prefetches to reduce L2 miss penalty, and dynamically dead instructions. AVF is sensitive to the compiler used, and aggressiveness of compilation options. For an AVF stressmark, all un-ACE instructions should be eliminated.

It may be clear that occupancy, and hence the vulnerability to soft errors is super-linear in the number of ACE instructions in flight. Intuitively, any program that does not have a high proportion of branch mispredictions, has a high proportion of loads and stores, and a high miss rate in the cache would have high occupancy. The above insights are used to derive a code generator.

4.3.2 Design of the Code Generator

The knobs required for the code generator are derived using the insights outlined in Section 4.3.1. The code generator must allocate a large enough memory region such that every line in the data caches and DTLB are covered. High AVF of caches is ensured by performing ACE loads and stores such that every cache line is 100% ACE (other strategies are possible). Simultaneously, high DTLB AVF is ensured by requiring the loads and stores to cover every line in the DTLB without evictions (read to evict is un-ACE). A code generator based on the framework outlined in Figure 4.1 is implemented. The code generator must be provided with the size of the ROB, and the caches, of the particular microarchitecture.

Using a strided load in the inner loop, that will miss in the L2 cache, and is dependent on itself (pointer chasing) avoids any Memory-Level Parallelism for the L2 misses. Ideally, having the size of the inner loop equal to the size of the ROB minimizes the number of L2 misses in the ROB, while also maximizing the number of instructions in the shadow of the L2 miss. As the loop gets larger than the ROB size, fewer instructions occur in the shadow of the L2 miss. The code generator is allowed to determine the size of the loop, but its maximum size is restricted to 1.2× the size of the ROB. Separately, another code generator framework is implemented in which the L2 miss is converted into an L2 hit, keeping the rest of the requirements the same. This models the case of L2 miss-free behavior. The code generator then fills up the inner loop (see Figure 4.1) with ACE instructions as specified using param-

eterizable knobs derived from the characteristics summarized under section 4.3.1, below:

1. *I-mix*: The fraction of loads, stores and arithmetic instructions are specified using this knob. This determines the occupancy of LQ, SQ and FU respectively.

2. *Dependency distance*: This knob controls the number of instructions between two dependent instructions and affects placement of instructions. Dependency distance has been used as a microarchitecture-independent metric for ILP [54, 55]. The code generator interleaves dependence chains to meet this requirement.

3. *Fraction of Long-Latency Arithmetic:* This knob controls the mix of long-latency and short-latency arithmetic instructions. This affects the average latency of each instruction and hence the issue rate.

4. *Average Dependence Chain Length:* This controls the average length of the instruction chain dependent on a load, leading up to a store. This knob affects the ILP. This is implemented by having a knob that specifies the fraction of arithmetic instructions that are to be transitively dependent on loads. These instructions are distributed uniformly over all loads, and chain loads to available stores.

5. *Register Usage:* This knob affects the proportion of Reg-Reg vs. immediate instructions, and hence determines the number of register values

that are ACE.

6. *Instructions Dependent on L2 Miss:* This knob controls the number of instructions occupying the IQ in the shadow of the L2 miss.

7. *Random seed:* This knob is passed to a random number generator that randomizes the placement of long-latency vs. short latency instructions in the code. This is used to discover the best code schedule.

8. *Code Generator Switch:* This switches between the code generators with and without L2 misses.

Every value that is loaded or produced must transitively produce a value that is stored to memory, to ensure 100% ACE-ness of instructions and data. Also, stored results must not overwritten before they are read. The code generator produces code in C, with embedded Alpha assembly instructions. Assembly instructions are used to precisely control the output of some of the above knobs.

Unique Requirements of the Code Generator: The requirement of 100% ACE instructions, and increasing susceptible state in the processor are two factors that distinguish this effort from typical functional verification, testing methodologies or power viruses. Functional verification or testing emphasizes on bug or defect coverage without any regard to ACE-ness or susceptible state resident in the processor. Therefore, functional verification tools may not

achieve as high AVF as the AVF stressmark methodology, or may require an unreasonably large number of random runs (if not directed) to achieve such high AVF. There is no correlation between power and state resident in the core. For example, long-latency stalls increase AVF, but provide opportunities to reduce core power using clock and/or power gating. Power dissipation is typically maximized when the processor is able to issue multiple arithmetic instructions at full bandwidth, but this typically implies that the occupancy of other queues are less than 100%. Furthermore, un-ACE instructions consume power but do not contribute to AVF. Thus, power viruses are unlikely to be high AVF workloads, by design. Deriving the properties that affect AVF from first principles allows the architect to restrict the search space by disallowing infeasible solutions, and to allow a quick generation of a high-AVF stressmark.

4.4 Framework for the Generation of the AVF Stressmark

The search space for an AVF stressmark, despite the pruning performed while creating the code generator, remains complex. As seen in the discussion in Section 4.2, the task of creating the optimal instruction schedule that satisfies the constraints of a microarchitecture, while simultaneously increasing SER is non-trivial. Therefore, utilizing a a Genetic Algorithm (GA) automates the exploration of the search space defined above. A Genetic Algorithm is evolutionary machine learning methodology which is often used to find "approximately optimal" solutions to complex optimization problems. The GA

initially starts from a set of random solutions. For each solution, a fitness value is computed, and the best results form the baseline for future generations. The GA applies mutation, crossover and migration to these solutions, to generate a new solution. Mutation involves random changes to the solution, crossover involves swapping parts of existing solutions to create offspring generations whereas migration involves changing the population of the solution. When the solutions in a generation converge, the GA introduces a cataclysmic event, to completely change the population of the best known solution and avoid being stuck in a local maxima or minima. The GA continues with the process of creating new generations until no further improvement is reported.

The use of a machine learning algorithm such as GA reduces the dependence on a designer's intimate knowledge of the microarchitecture while creating the stressmark. The IBM SNAP genetic algorithm framework, obtained under NDA for university research, is used to create the knob values for the code generator, as outlined in Figure 4.1. The output of the code generator is compiled and run on the AVF simulator (outlined below). The results are used to calculate the fitness metric (SER), which is fed back to the GA, to create future generations.

4.5 Evaluation Methodology

The methodology is evaluated on a modified version of SimSoda [47], which computes AVF using the ACE analysis methodology proposed by Mukherjee et al. [2] and Biswas et al. [27]. Simsoda is based on SimAlpha [48], which

models an Alpha 21264 (EV6) in great detail. SimAlpha models the Integer IQ and Floating Point IQ as separate structures. The experiments presented herein concentrate on the integer pipeline, for parity with SPEC CPU2006 integer results. The methodology, however, is general enough to be trivially extended to include the FP pipeline. As presented in Figure 4.1, the GA generates knobs that are provided as inputs to the code generator. The code generator produces the corresponding output, and is run on the SimSoda simulator.

The Genetic Algorithm (GA) runs for 50 generations, with 50 individuals per generation (a total of 2,500 runs), and the best result is picked as the stressmark. The stressmark is executed for 100M instructions. The stressmark is compared with 11 CPU2006 Integer Workloads and 10 CPU2006 FP workloads. The remaining workloads in the SPEC CPU2006 suite did not compile successfully due to compiler issues. A single simulation point of length 100M instructions is identified, using the SimPoint methodology [53], and used for a detailed simulation at this simulation point. The stressmark results are also compared with 12 MiBench [52] programs, for diversity of workloads in the workload suite. The stressmark and all the benchmarks were compiled using gcc version 4.1 with the -O2 flag. The probability of mutation is set as 0.05 and a crossover rate as 0.73 in the GA, based on recommended ranges from literature, such as Grefenstette [56], and Srinivas and Patnaik [57]. The choice of the settings primarily affect the rate of convergence of the genetic algorithm.

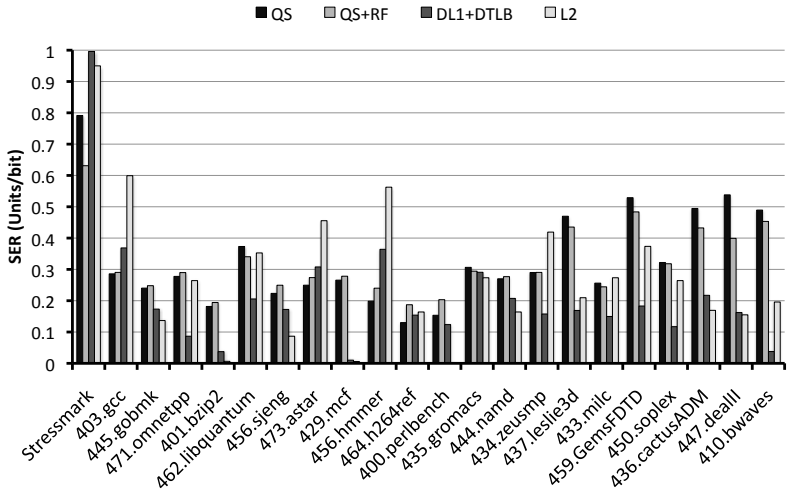


Figure 4.2: Comparison between the overall SER induced by the Stressmark and CPU2006 workloads on the core and caches for the Baseline Configuration.

4.6 Results

Figure 4.2 and Figure 4.3 represent the overall SER of the architecture specified in Table 4.1, which is called the Baseline Configuration. It is assumed that the circuit-level fault rate of the underlying circuits is 1 unit/bit. This is an arbitrary unit, since only the relative magnitude is of importance for the methodology. The SER of *Queuing Structures* (QS), *Queuing Structures and the Register File* (QS+RF), DL1+DTLB, and L2 are presented separately, as caches have significantly more bits than the core, and would dominate all SER computation. The SER values reported are normalized by dividing them by the total number of bits in that class of structure, in the interest of clarity. For example, the SER computed for the queuing structures is divided by the

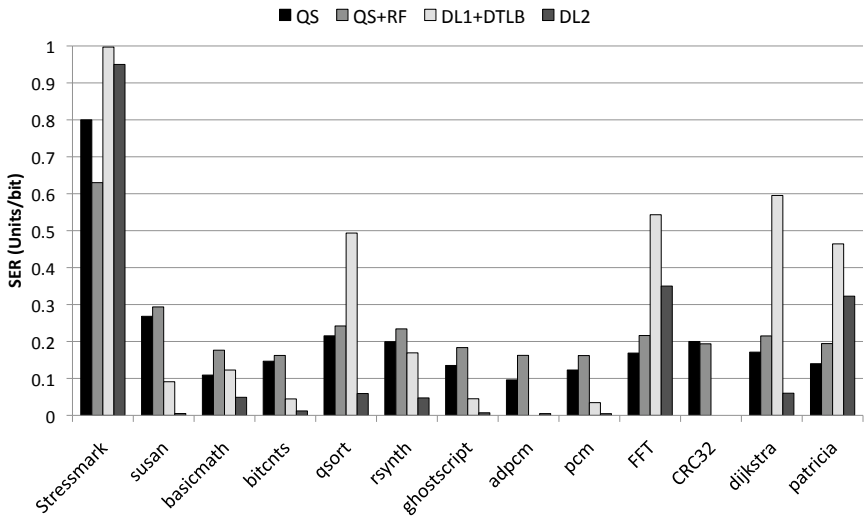


Figure 4.3: Comparison between the overall SER induced by the Stressmark and MiBench workloads on the core and caches for the Baseline Processor Configuration.

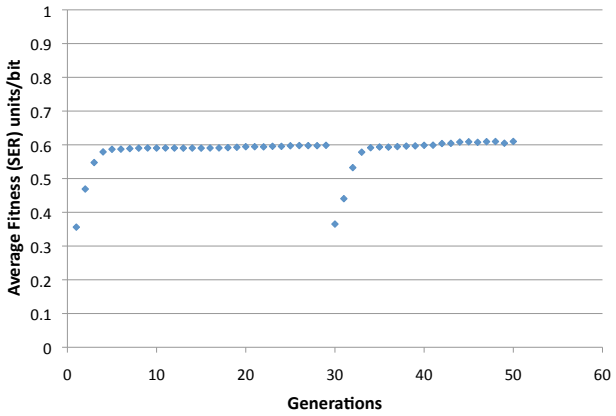
total number of bits in them.

Analysis

Figure 4.4(a) shows the final parameters generated by the GA as the optimal solution. The generated code utilizes every architected register, thereby maintaining high ACE in the Architected RF, by utilizing the appropriate number of reg-reg instructions. The GA selects short dependence chains to control ILP and hence occupancy of IQ, and a loop size almost equal to the size of the ROB. Figure 4.4(b) shows the convergence of Fitness Function for each generation, averaged over the 50 individuals per generation. The abrupt drop in the Average Fitness Function at generation 30 is due to a cataclysm triggered by SNAP as a result of convergence of solutions. The best solution

Parameter	Value
Loop Size	81
No. of loads	29
No. of stores	28
No. of Independent Arithmetic Instructions	5
No. of instructions dependent on L2 miss	7
Avg. Dependence Chain Length	2.14
Depencency Distance	6
Fraction of Long Latency Arithmetic	0.8
Fraction of Reg-Reg arithmetic instructions	0.93

(a) Knob settings of final GA solution



(b) Convergence of GA

Figure 4.4: Stressmark generated by the Genetic Algorithm for the Baseline Processor Configuration.

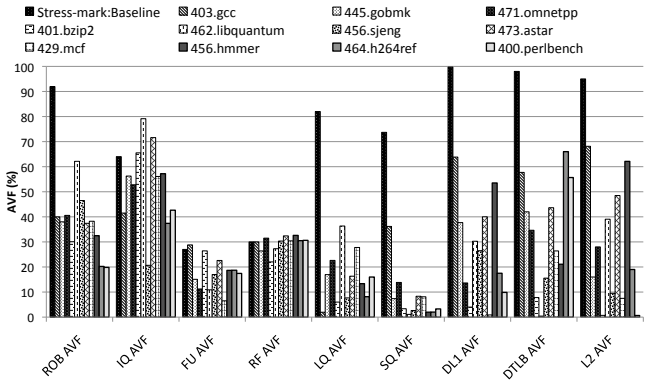
from generation 29 is moved into a new population of random mutations, and the process is repeated. At the end of 50 generations (2,500 runs), the GA has converged. Six individuals are run in parallel to speed up the process. The overall execution time for creating this stressmark is roughly 48 hours.

The stressmark induces an SER of 0.797 units/bit, 0.997 units/bit and 0.931 units/bit in queues, DL1+DTLB and L2, respectively. Workload *403.gcc* induces the highest overall SER (core+cache) of all the workloads in Figure 4.2 and Figure 4.3. Compared to this, the stressmark induces over 2× higher SER in QS+RF, and DL1+DTLB, and around 1.5× higher SER in L2. The AVF of individual structures while running the benchmarks is also captured at a granularity of 50,000 instructions. It is found that the stressmark is significantly higher than the AVF of such short traces as well. However, using

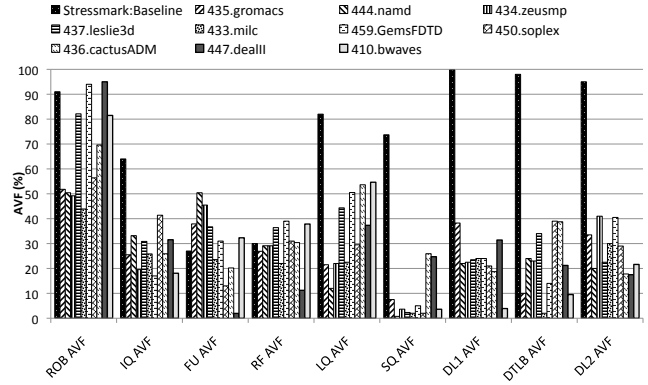
short traces may not give sufficient number of instructions for accurate ACE analysis, and may be pessimistic.

The Alpha 21264 has a separate 2-issue FP pipeline, in addition to the 4-issue integer pipeline. As FP programs are able to issue more instructions than integer programs, the SER of queuing structures in SPEC CPU2006 FP workloads is relatively high, compared to SPEC CPU2006 integer workloads. The stressmark has much higher vulnerable bits than *459.GemsFDTD* or *434.zeusmp*, in the core or caches. The SER induced by MiBench workloads is low.

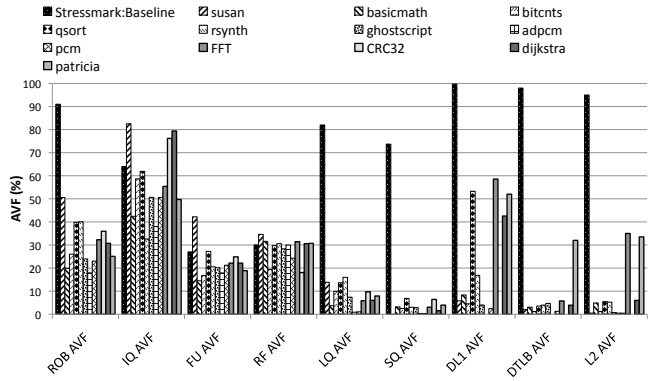
The highest instantaneous SER in the core would occur when the 80 entries in the ROB are distributed as 32 entries in each of the LQ and SQ, and 16 in the IQ. At this instant, AVF of FU would be 0%. The instantaneous worst-case occupancy for queuing structures, in the shadow of an L2 miss, is 0.899 units/bit (as compared to 0.797 units/bit for the stressmark). As RF AVF depends on the duration between production and consumption, it is difficult to estimate its AVF this way. Any processor making forward progress will have decreased occupancy just after the blocking L2 miss retires, and the ROB filling up completely in the shadow of the next L2 miss. Constraints such as the restriction on the number of loads and stores per cycle, and the load latency, and data dependencies required to maintain ACE-ness affect overall occupancy of a real program. Therefore, it is clear that the stressmark achieves AVF that is close to the theoretical and unsustainable maximum. It is impos-



(a) AVF of SPEC CPU2006 Integer Workloads

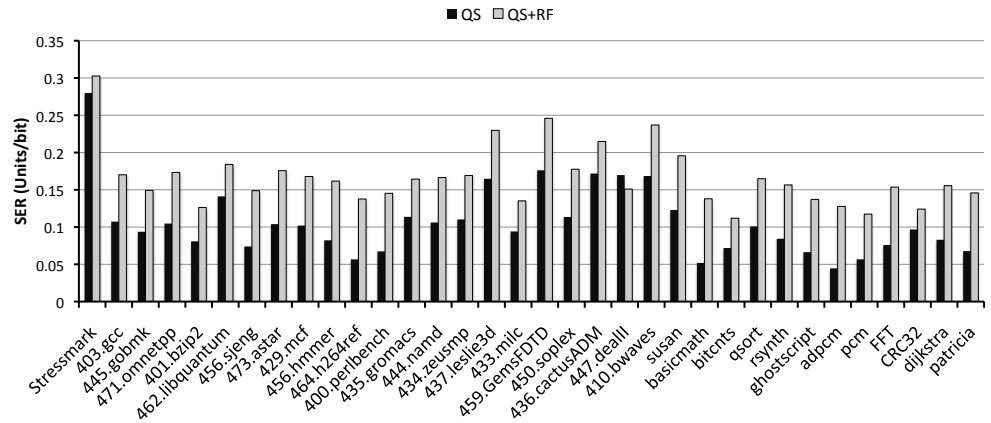


(b) AVF of SPEC CPU2006 FP Workloads

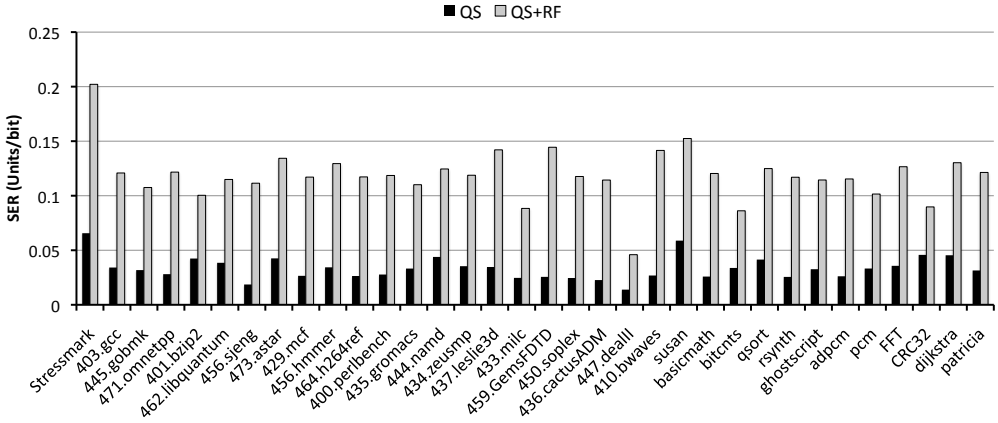


(c) AVF of MiBench Workloads

Figure 4.5: AVF of queuing and storage structures for SPEC CPU2006 and MiBench workloads on the Baseline Processor Configuration.



(a) SER for workloads on Configuration with Radiation Hardened Circuitry (RHC)



(b) SER for workloads on Configuration with Error Detection and Recovery (EDR)

Figure 4.6: SER induced on Processor Configurations RHC and EDR, by workloads from SPEC CPU2006 and MiBench.

sible to positively prove that the stressmark induces the absolute, sustainable maximum SER (a problem shared with power and thermal viruses). It is for this reason that the ability of the GA to optimize for such a complex solution space is leveraged. The convergence of the GA, and low difference between an idealized, "back of the envelope" calculation of instantaneous maximum SER and the stressmark-induced SER provides confidence that the SER induced by the stressmark is very near the maximum.

Figures 4.5 presents the AVF of SPEC CPU2006 and MiBench benchmarks on the baseline configuration, on individual structures. In contrast with SPEC CPU2006, the AVF stressmark for this microarchitecture achieves much higher AVF on all the structures, with the exception of FUs and in some cases, RF.

4.6.1 Stressmark generation for different circuit-level fault rates

The task of manually generating a stressmark when the circuit-level fault rates are not the same is even more challenging. For the GA, however, it is only a matter of changing the fitness function to reflect the new values. In this section, stressmarks are generated for two configurations for the same microarchitecture in Table 4.1 but different underlying fault rates outlined in Figure 4.7(a). Unchanged fault rates in DL1, DTLB and L2 are assumed. Consider the case in which the ROB, LQ and SQ are protected using Radiation-Hardened Circuitry (RHC), and a case in which these structures are protected using Error Detection and Recovery (EDR). Circuit-level fault

rates of structures are not publicly available, so the failure rates assumed are arbitrary. These assumed failure rates are still useful for demonstrating the effectiveness of the AVF stressmark methodology.

Configuration RHC: In the case of Config RHC, the IQ and RF are more vulnerable than the ROB, LQ and SQ. The methodology compensates by trading off some AVF in the less vulnerable units, to drive up the AVF of IQ and RF, and hence overall SER. The GA thus attempts to find a point where all trade-offs put together maximize the fitness function. This comparison is presented in Figure 4.7(d). Comparing Figure 4.7(b) to Figure 4.4(a), it is seen that the GA chooses fewer loads and stores, very short dependency distance and longer average dependence chain length. This reduces ILP and increases the occupancy of the IQ. Since this setting uses more arithmetic instructions, the fraction of reg-reg instructions required to use all architected registers is reduced. The GA selects an instruction schedule such that the overall SER for this new configuration approaches the maximum. Figure 4.6(a) presents the SER of the core of SPEC CPU2006 and MiBench programs. The AVF stressmark induces a significantly higher SER than any SPEC CPU2006 or MiBench programs.

Configuration EDR: As the AVF of the ROB, LQ and SQ are zero, the observable SER in the shadow of an L2 miss is relatively low. The GA therefore switches to the L2 miss-free case. Loads and stores are still required, due to the

Structure	Circuit-level Fault Rate (Units/bit)	
	RHC	EDR
ROB	0.25	0
IQ	1	1
FU	1	1
RF	1	1
LQ Tag	0.4	0
LQ Data	0.4	0
SQ Tag	0.35	0
SQ Data	0.35	0

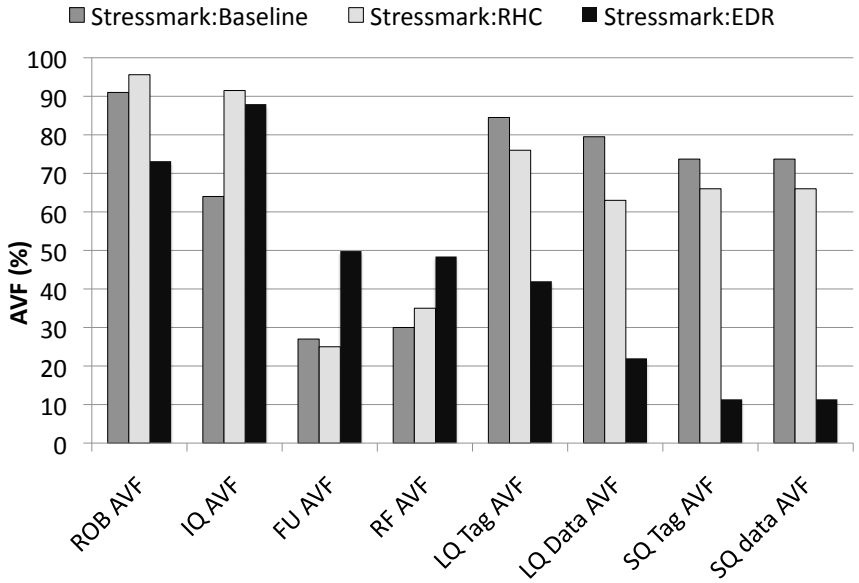
(a) Intrinsic fault rate of structures

Parameter	Value
Loop Size	74
No. of loads	20
No. of stores	20
No. of Independent Arithmetic Instructions	11
No. of instructions dependent on L2 miss	4
Avg. Dependence Chain Length	2.7
Depencency Distance	1
Fraction of Long Latency Arithmetic	0.7
Fraction of Reg-Reg arithmetic instructions	0.52

(b) Knob Settings for Config RHC

Parameter	Value
Loop Size	54
No. of loads	2
No. of stores	6
No. of Independent Arithmetic Instructions	5
No. of instructions dependent on L2 hit	15
Avg. Dependence Chain Length	6.5
Depencency Distance	1
Fraction of Long Latency Arithmetic	0.9
Fraction of Reg-Reg arithmetic instructions	0.4

(c) Knob Settings for Config EDR



(d) AVF of queuing structures

Figure 4.7: Results of AVF Stressmark Methodology on different circuit-level fault rates.

need for maintaining AVF of the caches. As there are no long-latency stalls, IPC is higher and hence FU AVF is higher. The turn-around time between releasing and re-assigning a renamed register is significantly decreased and hence RF AVF is higher. Simultaneously, the AVF of the IQ is also increased through longer dependence chains. Figure 4.7(d) represents the results of running Configuration EDR. As expected, AVF of FU and RF are driven high, at the cost of LQ and SQ occupancy. Figure 4.6(b) presents the SER induced by the workload suite on Configuration EDR. In this case too, the SER induced by the AVF stressmark exceeds that of any other program in the workload suite. It is thus demonstrated that the code-generator and GA methodology is flexible enough to adapt to such that overall error rate is increased.

4.6.2 Stressmark generation for a different microarchitecture

For completeness, a stressmark (Stressmark:LargeROB) for a 4-issue OoO processor with a larger IQ, ROB and rename register file in the core, and a larger DTLB and L2 cache and latency (Configuration:LargeROB), outlined in Table 4.2, is created. Figure 4.8 details the overall SER of Configurations Baseline and LargeROB, in which all structures are assumed to have the same circuit-level fault rate of 1 unit/bit. Further, assume that the sizes of each entry in the queuing structures of LargeROB are the same as the Baseline Configuration. In order to increase the AVF of the relatively larger IQ, the GA picks a shorter dependency distance, and much more instructions depen-

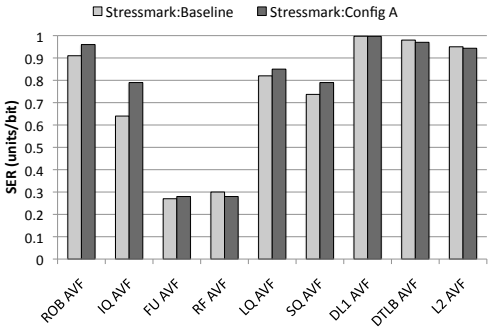
Parameter	Configuration:LargeROB
Integer ALUs	4, 1 cycle latency
Integer Multiplier	4, 7 cycle latency
Fetch/slot/map/issue/commit	4/4/4/4/4 per cycle
Issue queue	32 entries
ROB	96 entries
Integer rename register file	96
LQ/SQ size	32 entries
Branch Predictor	Hybrid, 4K global, 2 level 1K local, 4K choice
L1 I cache	64kB, 2-way, 64B line, 1 cycle latency
L1 D cache	64kB, 4-way, 64B line, 3 cycle latency
D TLB	512 entry, fully associative
L2 cache	2MB, 8 way, 12 cycle latency

Table 4.2: Alternate configuration for evaluating the stressmark creation methodology

dent on the L2 miss. The RF AVF is relatively lower, because the size of the architected register stays the same, but the number of rename registers increases. Thus, the methodology is flexible enough to automatically adapt to different microarchitectures.

4.7 Implications of the AVF Stressmark Methodology on Design

The stressmark methodology can be used by architects to evaluate the impact of design choices for reducing SER of their design. This discussion shall be restricted only to the core (Queueing structures + Register File), as



(a) AVF of queuing structures

Parameter	Value
Loop Size	91
No. of loads	29
No. of stores	29
No. of Independent Arithmetic Instructions	5
No. of instructions dependent on L2 miss	14
Avg. Dependence Chain Length	2.14
Depencency Distance	1
Fraction of Long Latency Arithmetic	0.6
Fraction of Reg-Reg arithmetic instructions	0.96

(b) Knobs for final GA solution

Figure 4.8: AVF of queueing and storage structures for Configuration:LargeROB.

the methodology clearly induces very high AVF on the caches. The overall SER induced in the core by the stressmark for the Baseline, RHC and EDR configurations is presented in Table 4.3. Using this information, the architect can study the area, power and performance penalty of the SER-mitigation techniques under consideration, and make appropriate trade-offs. A significant advantage of this technique is its adaptiveness. When the circuit-level fault rate of one or more structures are reduced, the framework automatically stresses other structures such that the overall SER approaches the maximum. The architect can thus pick candidate structures for protection from soft errors, that demonstrably have a significant impact on the overall SER.

4.7.1 Comparison with Other Possible Methodologies

In the absence of an AVF stressmark, it is impossible to know whether the set of 33 workloads offers sufficient AVF coverage. Table 4.3 shows the

worst SER observed on the set of workloads. The stressmark induces increased SER of 37%, 29% and 33% over the highest SER-inducing programs for the Baseline Configuration, Configuration RHC, and Configuration EDR, respectively. Clearly, a safety margin that does not account for this lack of SER coverage may result in under-design. Conversely, an aggressive safety margin could result in over-design.

Table 4.3 also presents the worst-case SER estimated by picking the highest SER on a per-structure basis, and adding them together, which is referred to as "Sum of highest per-structure SER". This methodology results in an error of 8%, 3%, and 17%, relative to the stressmark, for the Baseline Configuration, Configuration RHC, and Configuration EDR, respectively. For the selected workloads, the stressmark induces higher AVF. This is not necessarily the case, as one could write programs that drive individual structures to maximum AVF. This methodology produces variable results, and is fundamentally unsound. The worst-case SERs calculated by adding the raw circuit-level SER for individual circuits would be 1 unit/bit for the Baseline, 0.59 units/bit for Configuration RHC and 0.39 units/bit for configuration EDR. This is an overestimation, and will lead to an extremely pessimistic design. This, in turn, will impact performance, power and design effort of the processor.

4.7.2 Utilizing the Stressmark Methodology

The knowledge of the observable worst-case SER allows an architect to evaluate the robustness of the SER evaluation workload suite in use. In the

Configuration	Stressmark (units/bit)	Best Individual Program SER (units/bit)	Sum of Highest per-structure SER (units/bit)
Baseline	0.63	0.46 (447.dealII)	0.58
RHC	0.31	0.24 (459.gemsFDTD)	0.3
EDR	0.2	0.15 (susan)	0.17

Table 4.3: Comparison of worst-case SER estimation methodologies in the Core using SPEC CPU2006 and MiBench

case of the workloads considered in this section, the worst-case SER induced by an individual program in the workload suite of 33 programs is significantly less than the stressmark. This suggests that, at least for the microarchitecture under consideration, SPEC CPU2006 and MiBench may not be varied enough. The workload suite is lacking in programs that occupy the upper end of SER range. The stressmark reveals "SER bottlenecks" in the processor, and can be used to identify programs that may target these structures to induce high AVF. This, however, motivates the need for a rigourous methodology for selecting workloads for that achieve sufficient AVF/SER coverage, and are representative of user workloads, for SER evaluation.

Extending the Stressmark to Include Other Structures The code generator is currently designed to target the parts of the processor that contain the most state, and hence the highest sources of SER. However, the methodology is general enough to be extended to other structures, with or without modification. For example, fetch and decode queues are always maintained at 100% AVF as the stressmark never incurs any branch mispredictions. FP

instructions can be trivially incorporated into the same framework as integer instructions. However, if all these large structures are protected with error detection and recovery, the SER bottleneck will shift to other parts of the microarchitecture. This will potentially require the design of a different code generator, that stresses these smaller structures. A study similar to the one presented herein will be required, that identifies microarchitecture-dependent characteristics, and utilizes these to create a code generator. Restricting the search space of the GA is important to allow it to converge in a reasonable amount of time.

4.8 AVF Stressmark Generation for In-order Pipelines

The task of generating AVF stressmarks for in-order pipelines is generally simpler than that for out-of-order machines. As instructions in an in-order machine execute in the same order as in the program binary, many large and complex structures that contain state, such as the ROB and the load and store queues are eliminated. For this work, we consider the instruction buffer, store buffer, functional units, and architected register file, as these contain the largest amount of state for an inorder machine. Table 4.4 represents an inorder superscalar machine with an designed issue width of 2 instructions per cycle.

The *instruction buffer* (IB) holds decoded instructions until they are ready to be issued. The issued instructions execute in the function units, and are subsequently retired. The *store buffer* (SB) contains stores that have been retired, but are waiting to be written out to the caches or memory. The Alpha

Parameter	Configuration
Issue Width	2, in-order
Integer ALU	2 ALUs, 1 cycle latency
Integer Multiplier	2, 7 cycle latency
Instruction Buffer	16 entries, 32 bits per entry
Store Buffer	8 entries, 128 bits per entry
Architected Register File	32 entries, 64 bits per entry
L1 Instruction and Data Cache	32kB, 4 way set associative, 1 cycle latency
L2 Cache	1MB, 8 way set associative, 9 cycle latency
TLB	512 entry, fully associative
Branch Predictor	4k gshare, 4k BTB

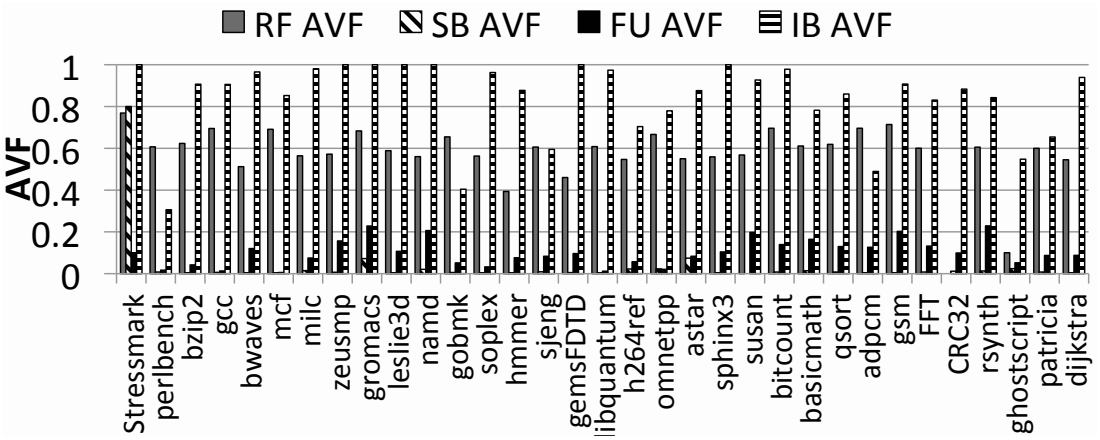
Table 4.4: In-order Configuration.

ISA defines 32 general-purpose integer registers, which are contained in the *Architected Register File* (ARF). The in-order machine is modeled using Simplescalar simulator [49], modified to capture the AVF of each of the structures under consideration.

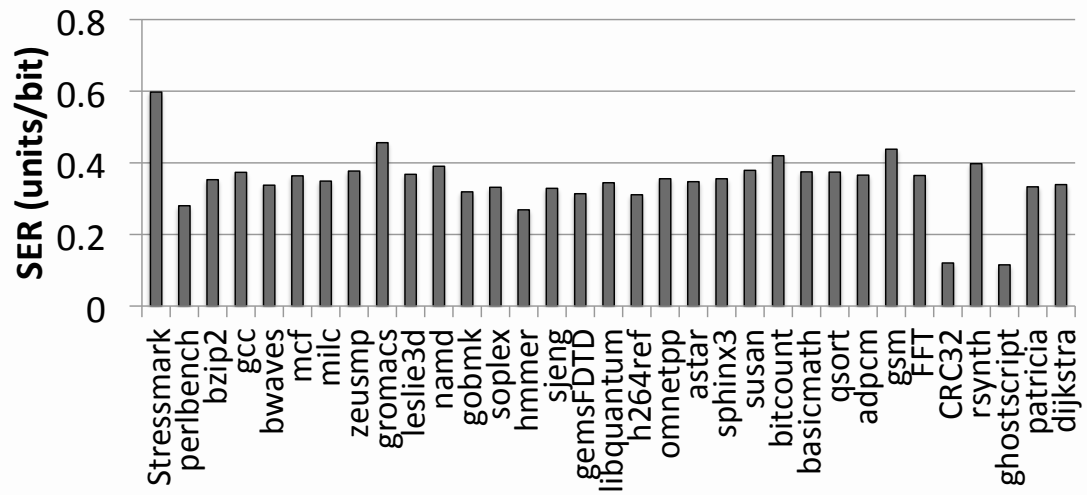
The same code generator framework used for generation of stressmarks for out-of-order machines is used. However, only a subset of these knobs have an impact on the AVF of the aforementioned structures in an in-order machine. Additionally, the strategy of stalling on L2 misses will increase the occupancy of the IB, but not the SB or FUs, as was the case in an out-of-order machine. In an in-order machine, stalling on an L2 miss will result in very low utilization of the SB and the FUs. It is possible to prune the search space by eliminating such redundancies or obviously poor strategies. However, for the data presented herein, the code generator for the out-of-order machine is run as-is.

Figure 4.9 presents the results of the AVF stressmark generated for the

in-order configuration outlined in Table 4.4, assuming an underlying circuit-level fault rate of 1 unit/bit. The stressmark induces higher AVF in every individual structure than any SPEC CPU2006 or MiBench workload. The stressmark increases the AVF of the ARF, SB, IB, and FU to increase the overall SER. Each register is un-ACE between a read and subsequent write. As the GA uses arithmetic instructions to increase the AVF of the FUs, some writes to the ARF are inevitable, resulting in an AVF of 0.77 in the ARF. In this architecture, the number of bits in the function units are relatively significant in comparison to other structures in the core, and is hence a larger trade-off between RF AVF and FU AVF. The stressmark induces 30% higher SER in the in-order core than the highest SPEC CPU2006 or MiBench workload (*gromacs*). The methodology does not change significantly for multithreaded in-order machines. An additional thread would be necessary to populate the ARF of the other hardware context. This hardware thread can then stall, allowing the stressmark presented herein to maximize the utilization of the shared structures. Although multithreading two workloads may potentially increase the AVF of the FUs, this will result in a reduction in the AVF of the register file, as the register is un-ACE for the duration between the last read, and write. Additionally, they also reduce the effective issue width for the other thread. Therefore, there is only a need to maximize the AVF of the register files, which can be easily done. Thus, creation of a multithreaded in-order stressmark is a relatively straightforward extension of the methodology



(a) AVF induced by SPEC CPU2006 and MiBench workloads compared to the Stressmark



(b) SER observed while running SPEC CPU2006 and MiBench workloads compared to the Stressmark

Figure 4.9: AVF of the Core for the In-order Configuration

Knob	Value
Loop Size	35
No of Loads	1
No of Stores	7
No of dependent arithmetic	1
No of independent arithmetic	22
No of instructions dependent on L2 hit	1
Dependency Distance	7
Fraction of long-latency instructions	0.0
Fraction of Reg-Reg instructions	0.73

Table 4.5: Knob Settings for the In-order Stressmark

presented herein.

4.9 Discussion

The AVF stressmark methodology presented herein is limited by the lack of low-level detail in an academic simulator. In the absence of low-level microarchitectural and circuit detail, ACE analysis is done conservatively. As low-level modeling information is added to the AVF estimation, the conservatism in the estimation of AVF is significantly reduced.

Furthermore, structures such as control circuits, much of the datapath, register alias tables, branch misprediction recovery checkpoints, decoders, etc. have not been modeled in this work, owing to the relatively small amount of state contained in them as compared to the structures considered in this work, or the lack of low-level detail necessary to model these structures, or both. As

noted earlier, it is impossible for all structures to contain state simulateneously, making sum of intrisic fault rates a poor metric for estimating the worst-case observable SER. The stressmark methodology provides an effective way of determining the worst-case observable SER.

SER estimation using AVF modeling also depends on accurate estimation of intrinsic fault rates. As AVF is multiplied by the intrinsic fault rates to estimate soft error rates, errors in estimating intrisic fault rates will be amplified. It is therefore necessary to compute the intrinsic fault rate accurately.

4.10 Conclusions

In this section, the lack of a methodology for evaluating the highest observable SER is highlighted. It is demonstrated that methodologies that ignore the interactions between structures within a processor may incur significant errors while estimating the highest SER under program influence. Therefore an automated and flexible methodology, derived from a comprehensive study of interactions between structures in an OoO processor is proposed, that generates an stressmark that approaches the maximum SER observable while running a program. It is demonstrated that the methodology can enable architects to make quantifiable decisions regarding the effect of various SER mitigation mechanisms on overall highest SER. This knowledge enables the architect to make better informed trade-offs between performance, power, area and SER reliability. It is shown that the stressmark achieves 1.4×, 2.5×, and 1.5× higher SER in core, DL1+DTLB and L2 respectively, as compared

to the highest SER induced by SPEC CPU2006 and MiBench programs for a 4-wide out-of-order architecture similar to the Alpha 21264.

Chapter 5

Mechanistic Modeling for Architectural Vulnerability Factor

In this chapter, a first-order mechanistic model for Architectural Vulnerability Factor (AVF) is presented. Derived from the first principles of superscalar processor execution, the mechanistic model is designed to provide insight into the precise interaction between the microarchitecture and the workload that together influence AVF. Using a set of inexpensive profiles, the model estimates the AVF of the ROB, IQ, LQ, SQ and FU with a Mean Absolute Error of less than 7%. The model is used to perform design space exploration and parametric sweeps, and to characterize workloads for their impact on AVF. Figure 1.2 provides an overview of the modeling methodology. The workloads are profiled once, and the statistics thus collected may be used to estimate the performance and AVF of multiple microarchitectures nearly instantaneously.

Owing to its construction, the model presented herein can provide insight into the fundamental factors influencing the AVF of a structure, beyond what is obtained using detailed simulation, or "black-box" models such as statistical or machine-learning based models [9–11]. Detailed, microarchitectural simulations typically present the aggregate masking effect of the ACE bits

due the workload, and the impact of microarchitectural events triggered by a workload on the occupancy of these ACE bits in a structure. It is difficult to infer the exact factors affecting AVF from aggregate metrics [14]. Statistical or machine-learning based modeling also do not easily quantify the fundamental interactions between the workload and microarchitecture, making it difficult to derive insight into the factors affecting AVF. The methodology presented herein models the workloads influence on microarchitectural events, and on the number of ACE bits induced in the structure, providing greater insight into the factors affecting AVF of a design.

Eyerman et al. [33] refer to analytical modeling methodologies that capture the fundamental mechanisms of operation of the processor as *mechanistic models*, to contrast them with other black-box modeling alternatives. The same terminology will be used in this dissertation.

The following are the unique contributions derived from the work presented herein:

- A novel first-order analytical model for AVF, designed from first principles to capture the impact of microarchitectural events on the AVF of major out-of-order processor structures is presented. The key novelty of this modeling effort over prior mechanistic models for performance [32, 33] is that it captures the interaction between different events occurring in a processor, and estimates the average occupancy of correct-path state, with low error. This enables the architect to derive unique insight

into the factors affecting the AVF of a structure, not available using aggregate metrics or black-box models.

- As the model requires inexpensive profiling, it can be used to perform design space exploration studies nearly instantaneously. The model is used to study the effect of scaling the ROB, issue width and memory latency on AVF, which provides valuable insight into the effect of microarchitecture and workload interactions on AVF.

- The model is used for workload characterization for AVF. It quantitatively explains why some high-IPC workloads induce high AVF in CPU structures whereas others do not, and why not all workloads with a large number of last-level data cache misses induce high AVF. The methodology can be used to identify high AVF workloads.

5.1 Modeling AVF using Interval Analysis

The central idea behind this modeling methodology is to model the occupancy of correct-path state in the core. Recall from Section 2.5 that the mispredicted, or wrong-path state in the processor will be un-ACE. By derating the occupancy of correct-path state in a structure by the proportion of ACE bits induced in it by the workload, the AVF of the structure can be estimated.

The modeling methodology is inspired by Interval Analysis, which was presented in Section 2.8. Similar to Interval Analysis for performance, the oc-

cupancy of correct-path state in the core is modeled as an ideal, uninterrupted occupancy, punctuated by miss events that alter this occupancy depending on their behavior. By computing the average occupancy, weighted by the number of cycles spent in each interval, the overall average occupancy of correct-path state can be estimated. A key departure from the Interval Analysis methodology for performance is its assumption that miss events are independent of one another, in the first order [32, 33]. This assumption does not hold true for occupancy. Therefore, modeling of correct-path occupancy necessitates the modeling of interactions between various miss events, and their collective influence on occupancy.

The following discussion describes the methodology for estimating the occupancy of correct-path state, of the ROB, LQ, SQ, IQ, and FU, which contain the largest amount of corruptible state in the core. The exclusion of wrong-path instructions from the occupancy estimations enables the easily computation AVF by derating this occupancy by the fraction of bits introduced into a structure that were ACE. Un-ACE instructions are identified through profiling and this information is used to determine the number of ACE bits injected in to each structure while running the workload. The separation of the program's influence on the number of ACE bits induced in a structure, and the residency of these ACE bits in the structure enables the architect to gain deeper insight into the interaction of events, and their contribution to overall AVF. As with any analytical modeling methodology, the design objective in this modeling methodology is to balance accuracy with simplicity

of formulation, the ability to provide quantitative insight, and ease of collecting necessary program characteristics.

The following section describes the estimation the occupancy of eventually committed state in the ROB using Interval Analysis. The ROB occupancy governs the occupancy of LQ, SQ, and FU, and can be used to estimate their occupancy/utilization. As the IQ can issue instructions out-of-order, its occupancy is estimated independently in Section 5.3. AVF can be estimated by derating the occupancy of the structures with the average fraction of ACE bits in these structures.

5.2 Modeling the AVF of the ROB

The occupancy of the ROB is modeled using interval analysis as having a steady-state, or ideal value in the absence of miss events. Each miss event would alter occupancy of correct-path instructions, depending on its behavior. Averaging the occupancy during the steady-state execution and miss events, weighted by the cycles spent in each interval gives the overall average occupancy. The ramp-up and ramp-down curves for occupancy are linearized, with slopes equal to the steady-state dispatch rate, in the interest of simplicity. The effect of each miss event on occupancy are studied independently of one another, in Sections 5.2.1 through 5.2.3. Subsequently, an analysis of the impact of interaction between miss events is presented in Section 5.2.5.

5.2.1 Modeling steady-state occupancy

As seen in Section 2.8, given an instruction window of size W, the total number of cycles taken to execute all instructions in the instruction window is a function of the latency of executing the critical path. The average critical path length $K(W)$ for a given program is modeled as $K(W) = \frac{1}{\alpha}W^{1/\beta}$ [32, 34] where α and β are constants that are determined by fitting the relationship between $K(W)$ and W to a power curve. This analysis is performed assuming that all instructions have unit latency. Therefore, given an average instruction latency l, the critical path would require $l \cdot K(W)$ cycles. Using Little's law, the ideal IPC ($I(W)$) that can be extracted from the program given an instruction window of size W is presented in Equation 5.1 [32, 34]. For a processor with a designed dispatch width D, setting $I(W) = D$, and rearranging the terms in Equation 5.1 gives us the steady-state ROB occupancy, O_{ideal}^{ROB}, or $W(D)$ necessary to sustain the peak dispatch rate.

$$I(W) = \frac{W}{l.K(W)} = \frac{\alpha}{l} \cdot W^{(1-1/\beta)} \tag{5.1}$$

$$\therefore O_{ideal}^{ROB} = W(D) = \left(\frac{l \cdot D}{\alpha}\right)^{\frac{\beta}{\beta-1}} \tag{5.2}$$

If the ideal IPC of the program is less than the designed dispatch width, the program requires a much larger instruction window to extract the necessary ILP. In this case, the occupancy of the ROB will saturate to 100%. As noted by Karkhanis and Smith [32], a processor that has a balanced design for a

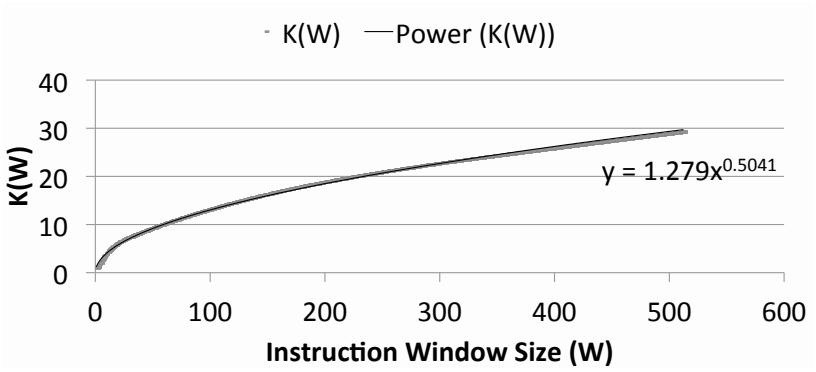
Workload	α	β
perlbench	1.43	1.51
bzip2	0.78	1.98
gcc	1.50	1.61
bwaves	2.16	1.34
mcf	1.74	1.60
milc	0.78	2.40
zeusmp	0.88	2.11
gromacs	1.70	1.70
leslie3d	1.02	1.81
namd	1.07	1.76
gobmk	1.00	1.84
soplex	0.98	1.80
hmmer	1.09	1.82
sjeng	0.8	1.73
gemsFDTD	1.32	2.12
libquantum	2.45	1.24
h264ref	1.95	1.57
omnetpp	1.36	1.58
astar	0.69	1.93
sphinx3	1.26	1.69

Table 5.1: Values of α and β for SPEC CPU2006 workloads.

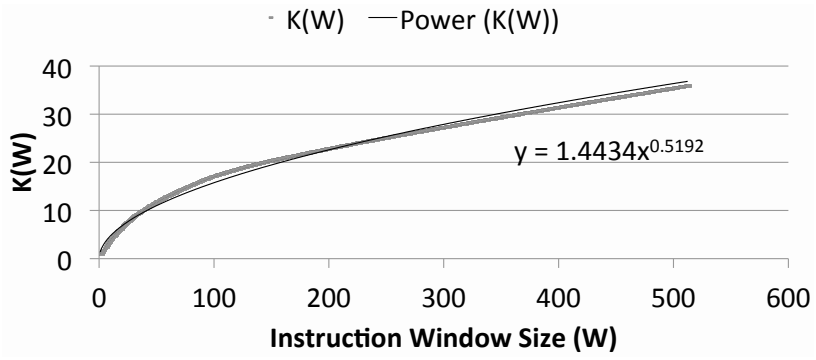
typical workload will not frequently stall due to a full IQ. Therefore, only the typical case is considered for modeling.

Table 5.1 lists the values of α and β for workloads in the SPEC CPU2006 suite. Note that $\beta > 1$ for any real workload. Values of $\beta < 1$ are meaningless, as they either imply that the critical path in the instruction window $K(W)$ is potentially longer than the instruction window size W ($0 < \beta < 1$), or that the critical path length reduces as the instruction window size increases ($\beta < 0$). In the case of SPEC CPU2006, β ranges between 1.24 (*libquantum*) and 2.40 (*milc*). Lower values of β indicate less ILP, whereas higher values indicate more ILP. Consequently, $\frac{\beta}{\beta-1}$ in Equation 5.2 asymptotically approaches ∞ as $\beta \to 1$, and approaches 1 for large values of β. This implies that workloads with low ILP need a larger instruction window to be able to issue the same number of instructions per cycle as compared to a workload with high ILP.

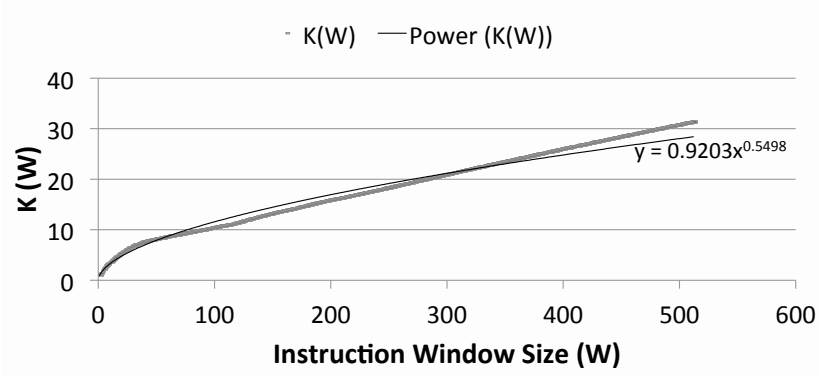
Figure 5.1 presents the relationship between K(W) and W for a range of instruction window sizes up to 512 entries. The power curve fit for determining α and β is depicted by the solid black curve, with the corresponding equation for the power curve. For workloads such as *bzip2* (Figure 5.1(a)), the fit is nearly exact. For other workloads such as *astar* (Figure 5.1(b)), the fit diverges somewhat at the higher end of the curve. These two cases are typical of most workloads. For yet other workloads such as *hmmer* (Figure 5.1(c)), the I-W curve is slightly irregular, leading to a relatively sub-optimal fit. Although the general trend is approximately that of a power curve, the curve itself has



(a) I-W curve for *bzip2*

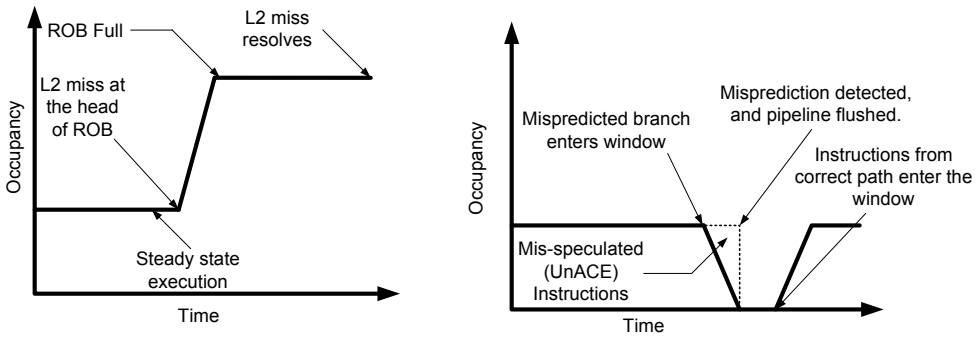


(b) I-W curve for *astar*



(c) I-W curve for *hmmer*

Figure 5.1: I-W characteristics for sample SPEC CPU2006 workloads.



(a) Occupancy of the ROB in the shadow of an L2 miss

(b) Occupancy of the ROB during a branch misprediction

Figure 5.2: Modeling the Occupancy of the ROB Using Interval Analysis.

changing behavior in different intervals, leading to divergence with the fitted power curve at various points along it. This may be problematic for workloads that have few miss events, making the accuracy of the I-W curve fit a critical factor. This will be discussed in detail in Section 5.7.

It is recommended that a value of W that is much larger than the range of instruction window sizes of interest be selected for doing curve fitting. This avoids any error from the divergence at the extremes.

5.2.2 Modeling Occupancy in the Shadow of Long-Latency Data Cache Misses

As shown in Figure 5.2(a), a non-overlapped data L2 miss (or a TLB miss for a hardware-managed TLB) reaches the head of the ROB, blocking the retirement of subsequent instructions. The processor continues to dispatch instructions until the ROB fills up completely. Thus, the occupancy in

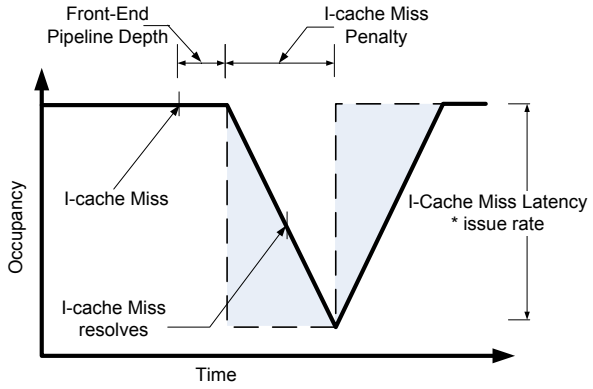


Figure 5.3: Modeling the Occupancy During an I-cache Miss.

the shadow of a non-overlapped L2 miss is $O^{ROB}_{DL2Miss} = W$. When the data eventually returns from main memory, the L2 miss completes, and the processor is now able to retire instructions. At this point, the assumption about interval analysis requires that the occupancy of the ROB returns to steady-state. However, this need not be the case: the occupancy of the ROB can remain at nearly 100% if the processor is capable of dispatching and retiring instructions at the same rate. In Section 5.2.5, a procedure for accounting for this interaction is explained.

5.2.3 Modeling Occupancy During Front-End Misses

Modeling Occupancy During an L1 I-cache Miss: The occupancy of the ROB during an L1 I-cache miss depends on the hit latency of the L2 cache, as shown in Figure 5.3, and therefore requires special modeling. When an L1 I-cache miss occurs, the processor is initially able to dispatch instructions until the front-end pipeline drains. Subsequently, the occupancy of the ROB

decreases by a rate determined by the ideal IPC (see Equation 5.1), as depicted by the solid line. Once the I-cache miss resolves and the front-end pipeline is refilled, occupancy of the ROB starts increasing at the rate of the ideal IPC (Equation 5.1). Linearizing ramp-up and ramp-down, the shaded areas under the ramp-up and ramp-down are equal, allowing the model to approximate occupancy as depicted by the dotted line. As depicted in Figure 5.3, the occupancy of state during an I-cache miss reduces in proportion to the designed dispatch or retirement rate D (assuming that they are equal), and the latency of the I-cache miss. Thus, $O^{ROB}_{IL1Miss} = O^{ROB}_{ideal} - lat_{L2} \cdot D$, where lat_{L2} cycles is the hit latency of the L2 cache. This allows us to model changes in occupancy as steps, greatly simplifying computation, and is used to model other miss events as well.

The occupancy during other front-end misses such as L1 I-cache misses, L2 instruction misses, and I-TLB misses can be modeled on similar lines. As the latencies of L2 instruction misses and I-TLB misses are relatively large, the occupancy of the ROB goes down to zero.

Modeling Occupancy During a Branch Misprediction: Figure 5.2(b) illustrates the effect of a branch misprediction on the occupancy of the ROB. The solid line depicts the occupancy of correct-path instructions in the ROB. All instructions fetched after the mispredicted branch are eventually discarded, and hence un-ACE. As correct-path instructions are retired, and instructions from the mispredicted path continue to be fetched, the occupancy of ACE

state decreases. The overall occupancy, as indicated using the dotted line remains at the steady-state value, until the branch misprediction is detected and the pipeline is flushed. Karkhanis and Smith [32] show that assuming an oldest-first issue policy, the mispredicted branch is among the last correct-path instructions to be executed in the instruction window. Simultaneously, retirement of instructions drains the ROB of correct-path state, resulting in low ACE occupancy by the time the misprediction is detected, and the pipeline is flushed. Thus, $O_{brMp}^{ROB} \approx 0$. After the front-end pipeline refills and dispatch resumes, the occupancy of the ROB eventually returns to the steady-state value.

5.2.4 Computing Occupancy of Correct-Path Instructions in the ROB

The interval analysis model for performance enables the estimation of the number of cycles spent during each execution interval. The model presented herein enables the estimation of the occupancy of correct-path instructions the ROB during these intervals. Thus, average occupancy of the ROB, computed over the execution of the program is as follows:

$$O_{total}^{ROB} = \frac{1}{C_{total}} \times (O_{ideal}^{ROB} \cdot C_{ideal} + O_{DL2Miss}^{ROB} \cdot C_{DL2Miss} + O_{IL1Miss}^{ROB} \cdot C_{IL1Miss} + O_{brMp}^{ROB} \cdot C_{brMp} + O_{ITLBMiss}^{ROB} \cdot C_{ITLBMiss} + O_{DTLBMiss}^{ROB} \cdot C_{DTLBMiss}) \tag{5.3}$$

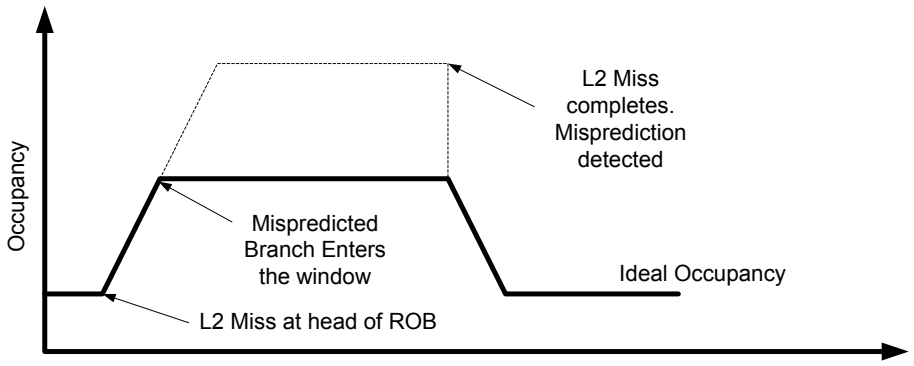
In essence, Equation 5.3 is the average occupancy, weighted by the number of cycles spent in each interval.

5.2.5 Modeling the Effect of Interactions Between Miss Events

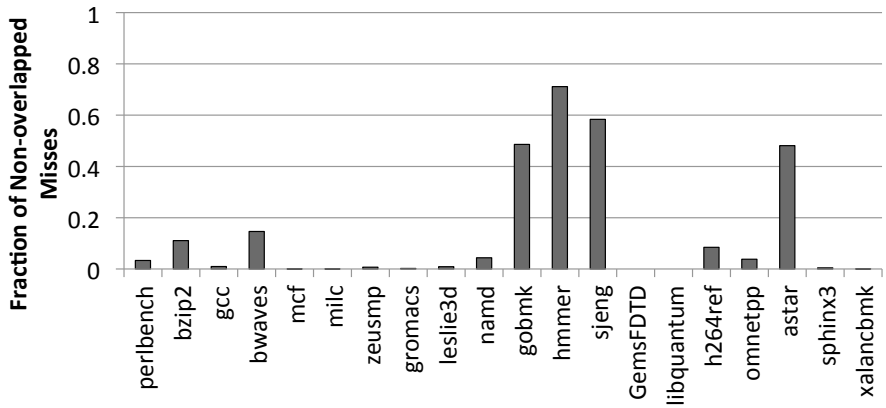
The following discussion explores the cases in which multiple miss-events may interact with each other, and whether they have a significant impact on the accuracy of the model.

Dependent Branch Mispredictions in the Shadow of a Long-Latency Data Miss: Consider the case in which a branch is dependent on a long-latency data L2/TLB miss, and occurs within the same instruction window. If such a branch is mispredicted, all instructions in the ROB fetched after the branch instruction are un-ACE. As the branch will not resolve until the cache miss completes, the occupancy of correct-path state in the shadow of this L2 miss is not 100%, as shown in Figure 5.4(a). Programs such as *perlbench, gcc, mcf* and *astar* have a significant number of such interactions. Branch mispredictions that are independent of long-latency data cache misses will resolve quickly enough such that their interaction has little effect on occupancy, and are hence ignored. This assumption is tested in Section 5.7.2.

This interaction between long-latency data misses and dependent branch mispredictions is captured by computing the number of instances in which a non-overlapped data L2 or TLB miss has a dependent mispredicted branch in its instruction window ($N_{dep}(W)$), and the average number of instructions between the *earliest* dependent mispredicted branch and the non-overlapped miss at the head of the ROB ($len_{DL2,Br}(W)$, $len_{DTLB,Br}(W)$). Note that the



(a) Mispredicted Branch dependent on an L2 miss.



(b) Quantifying the number of data L2/TLB misses followed by long intervals, before the occurence of a front-end miss.

Figure 5.4: Modeling the Effects of Interactions Between Miss Events on Occupancy.

mispredicted branch only needs to be dependent on *any* data L2 or TLB cache miss in the instruction window. This computation can be added to the existing profiler for non-overlapped data cache misses with little overhead. It does, however, require that information on mispredicted branches from the branch profiler be made available to the non-overlapped data cache miss profiler. Thus, $N_{dep}(W)$ data L2 misses have only $len_{DL2,Br}(W)$ correct-path instructions in their shadow, whereas the remaining data L2 misses have W correct-path instructions in their shadow. Assuming $N_{DL2Miss}(W)$ non-overlapped misses, the term $O^{ROB}_{DL2Miss} \cdot C_{DL2Miss}$ in Equation 5.3 is expressed as $lat_{DL2Miss} \cdot (len_{DL2,Br}(W) \cdot N_{dep}(W) + W \cdot (N_{DL2Miss}(W) - N_{dep}(W)))$.

Interaction of Data cache and Instruction Cache Misses: The model presented herein is impacted by two types of interactions between data-cache and instruction cache miss events. The first case occurs when an L2 instruction cache or ITLB miss occurs in the shadow of a non-overlapped DL2 or DTLB miss, resulting in less than 100% occupancy of the ROB. This case is very rare; only *perlbench* is significantly impacted by this, due to its higher proportion of ITLB misses. A procedure similar to the aforementioned case of dependent branch instructions is followed to estimate the impact of these interactions. L1 I-cache misses in the shadow of a non-overlapped L2/DTLB miss will resolve quickly, and hence their interaction has negligible effect on average occupancy.

A second case is when the duration between a long-latency data cache miss and front-end miss is too long. As described in Section 5.2.2, a simplifying

assumption that the occupancy of the ROB returns to steady state relatively quickly after a long-latency data cache miss retires, as a result of subsequent front-end misses causing the ROB to drain, may not be entirely accurate. Therefore, the fraction of non-overlapped DL2 and DTLB misses that are separated from a front-end miss by *at least* $2W$ instructions are measured. This interval length is chosen to be large enough to eliminate misses that occur in the shadow of the L2 and DTLB miss. Furthermore, a significant number of misses occur within $2W$ of the non-overlapped L2/TLB miss, which have negligible impact on accuracy. Thus, only the length of long intervals is captured, and the impact of very short intervals on the average is filtered out. Figure 5.4(b) outlines the average number of non-overlapped misses that are followed by intervals of greater length than $2W$ instructions, for the cache and branch predictor configuration outlined for the *wide-issue machine*, in Table 5.2. As seen in Figure 5.4(b), with the exception of *hmmer, gobmk, sjeng*, and *astar*, this situation occurs infrequently. The average length of such sequences for these workloads range between 300 and 450 instructions. Thus, the fraction of non-overlapped L2 misses in Figure 5.4(b) will experience a subsequent region of ideal execution in which occupancy is W, which is included in the calculations for the model.

A similar experiment performed to capture long intervals between two consecutive data L2/TLB misses, results in the conclusion that they are much fewer in number, and affect only workloads dominated by these misses. Therefore, they have negligible impact on average occupancy of these workloads due

to the dominance of L2/TLB misses on overall occupancy.

Clustered Front-End Misses: With the exception of the L1 I-cache miss, the ROB is completely drained after a front-end miss event. As these misses are independent of each other, their impact on occupancy is separable, in the first-order. Due to the relatively low latency of an L1 I-cache miss that hits in the L2, the ROB may not be completely drained before the miss resolves. Thus, two I-cache misses relatively close to each other may drive occupancy lower than if they occurred separately. Similarly, an I-cache miss quickly following a branch misprediction may experience lower occupancy. However, the latency of each I-cache miss is relatively short, and therefore, an large number of such events are necessary to produce a meaningful impact on overall occupancy. For the workloads under consideration, the number of I-cache misses are not significant enough in number to warrant modeling of their interactions. The relative infrequency of such events, and the relatively low impact on average occupancy implies that their impact is negligible. This assumption will be evaluated in Section 5.7.2.

5.3 Modeling of the AVF of the IQ

The occupancy of the Issue Queue requires separate modeling due to the fact that instructions can issue out-of-order. The model assumes an oldest-first issue policy. Occupancy of correct-path instructions during front-end misses is modeled in exactly the same way as the ROB. Steady-state occupancy, and

occupancy in the shadow of a long-latency data cache or TLB miss needs to be modeled differently, as outlined below.

Steady-state IQ occupancy: Let $A(W)$ be the average number of instructions in a chain of dependent instructions in the instruction window. $A(W)$ is obtained as a by-product of the critical-path profiling necessary to determine $K(W)$. The average latency of each instruction in the IQ is $l \cdot A(W)$. Using Little's law, $O^{IQ}_{ideal} = l \cdot A(W) \cdot min(D, I(W))$ [58].

Occupancy in the Shadow of a Long-Latency Data Miss: When issue of instructions ceases in the shadow of an L2/TLB miss, the IQ contains only the instructions dependent on such misses. The average number of instructions dependent on the L2 and DTLB misses in the instruction window is measured, to determine the average occupancy during such miss events. This profiling can be added to the existing profiler for determining non-overlapped data-cache misses, and incurs no overhead. This profiling also captures the effect of interactions between data L2/TLB misses and front-end misses, similar to the procedure outlined in Section 5.2.5.

5.4 Modeling the AVF of LQ, SQ, and FU

The occupancy of the Load Queue, Store Queue and Function Units can be derived from the occupancy of the ROB, and the instruction mix (I-mix). Additionally, by classifying un-ACE instructions according to the I-mix, the

occupancy of each of these units is derated to estimate AVF. FU utilization can be estimated using Little's Law, as the latency of each arithmetic instruction and the issue rate are known.

Loads and stores enter the LQ and SQ after they are issued, and remain there until they are retired. As seen in Section 5.3, the average dispatch-to-issue latency for an instruction is $l \cdot A(W)$ cycles. Thus, the LQ and SQ occupancy can be estimated as the fraction of loads and stores in the ROB, adjusted for the average dispatch-to-issue latency of the loads/stores in the instruction stream. Thus, the occupancy-cycle product in the ideal case for SQ is expressed as $(N_{stores}/N_{total}) \cdot O_{ideal}^{ROB} \cdot C_{ideal} - l \cdot A(W) \cdot N_{stores}$, where N_{stores} and N_{total} are the number of stores, and total number of instructions respectively. All other occupancy-cycle products from Equation 5.3 are multiplied by the fraction of stores to estimate O_{total}^{SQ}. The occupancy estimation is further improved by also including the number of loads and stores in the shadow of a non-overlapped data L2/TLB in the calculations.

5.5 Assumptions of the Model

The model assumes that for a processor with designed issue or dispatch width D, in the absence of any long-latency miss events, the processor is able to dispatch instructions at peak dispatch width, and is not constrained for resources (functional units, load and store queue entries, MSHRs, etc) while running typical workloads. An implicit assumption of the model is that the processor must not be underprovisioned. In prior work, Eyerman et al. [33]

assume a balanced processor, and define it as one in which "for a given dispatch width D, the ROB (window size) and other resources such as the issue buffer(s), load/store buffers, rename registers, MSHRs, functional units, write buffers, etc., are of sufficient size to achieve sustained processor performance of D instructions per cycle in the absence of miss events. Furthermore, for a given balanced design, reducing the size of any one of the resources will reduce sustained performance below D instructions per cycle". In this dissertation, rather than emphasizing the assumption of a balanced design, a configuration that is not underprovisioned is assumed.

It is assumed in this dissertation that the sizes of the load and store queues are large enough so as to extract all the available MLP from the instructions in the instruction window. If the size of the load and store queue constrains the extraction of MLP from the instruction window, the load and store queue sizes need to be considered while estimating the number of non-overlapped data cache misses.

All scaling studies and design space evaluation studies presented herein assume that the processor is not underprovisioned.

5.6 Evaluation

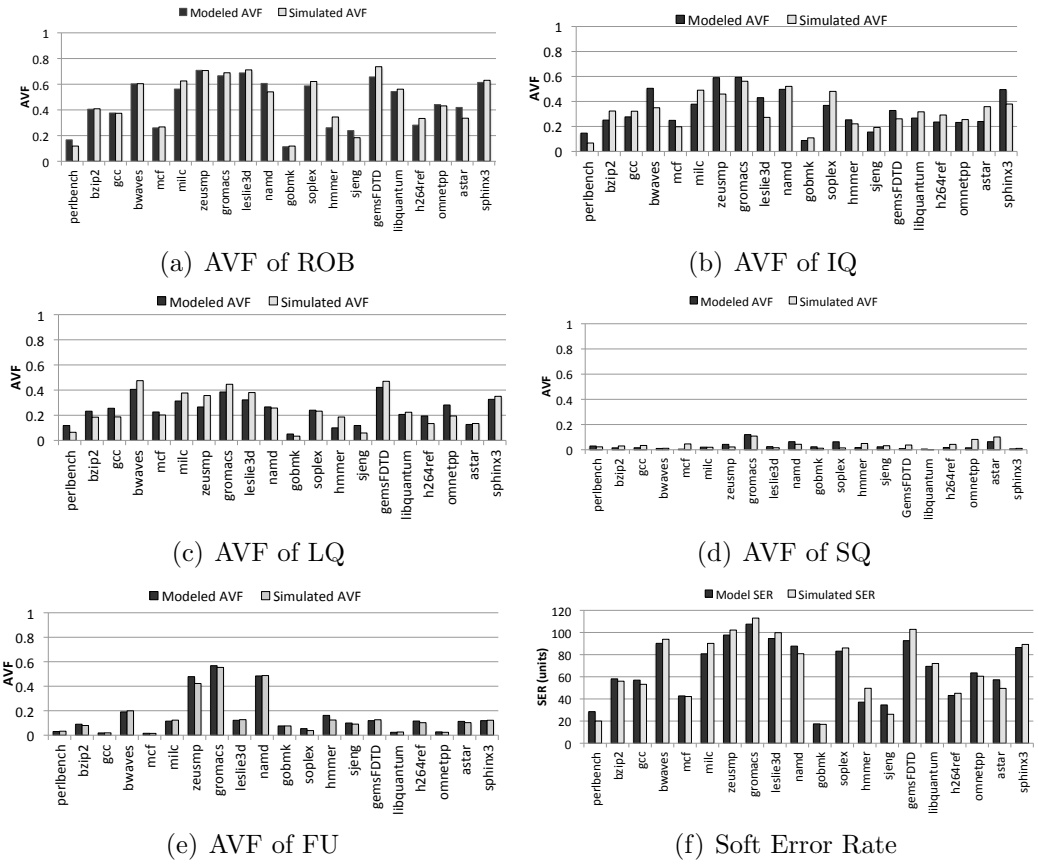
SimpleScalar [49] is used to evaluate the accuracy of the mechanistic model presented herein. ACE analysis is implemented on a modified version of SimpleScalar. Bit-wise ACE analysis is performed, as opposed to an occupancy

Parameter	Wide Issue Machine	Narrow Issue Machine
ROB	128 entries, 76 bits per entry	64 entries, 76 bits per entry
Issue Queue	64 entries, 32 bits per entry	32 entries, 32 bits per entry
LQ	64 entries, 80 bits per entry	32 entries, 80 bits per entry
SQ	64 entries, 144 bits per entry	32 entries, 144 bits per entry
Branch Predictor	Combined, 4K bimodal, 4K gshare, 4K choice, 4K BTB	Combined, 4K bimodal, 4K gshare, 4K choice, 4K BTB
Front-end pipeline depth	7	5
Fetch/dispatch/ issue/execute/commit	4/4/4/4/4 per cycle	2/2/2/2 per cycle
L1 I-cache	32kB, 4-way set associative	32kB, 4-way set associative
L1 D cache	32kB, 4-way set associative	32kB, 4-way set associative
L2 cache	1MB, 8-way set associative	1MB, 8-way set associative
DL1/L2 latency	2/9 cycles	2/9 cycles
DTLB and ITLB	512 entry, fully associative	512 entry, fully associative
Memory Latency	300 cycles	300 cycles
TLB Miss Latency	75 cycles	75 cycles

Table 5.2: Processor Configurations

heuristic. Simplescalar models a unified instruction window, that combines the ROB and the IQ into a single unit. Therefore, Simplescalar is modified to have a separate IQ. Further, the LQ and SQ are modeled separately, as opposed to the unified Load-Store Queue (LSQ) in SimpleScalar.

The accuracy of the first-order mechanistic model is evaluated using 20 SPEC CPU2006 workloads (compiler issues prevented the remaining workloads for Alpha from being successfully compiled) using gcc v4.1 compiled with -O2 flag. The profilers and the detailed simulator on single simulation points of length 100 million instructions, identified using the SimPoint methodology [53]. The two configurations evaluated in this discussion are presented in Table 5.2. *Wide-issue machine* represents an 4-wide issue out-of-order superscalar, whereas *Narrow-issue machine* represents a 2-wide issue out-of-order superscalar.



(a) AVF of ROB

(b) AVF of IQ

(c) AVF of LQ

(d) AVF of SQ

(e) AVF of FU

(f) Soft Error Rate

Figure 5.5: Modeling the AVF of the Wide-Issue Machine.

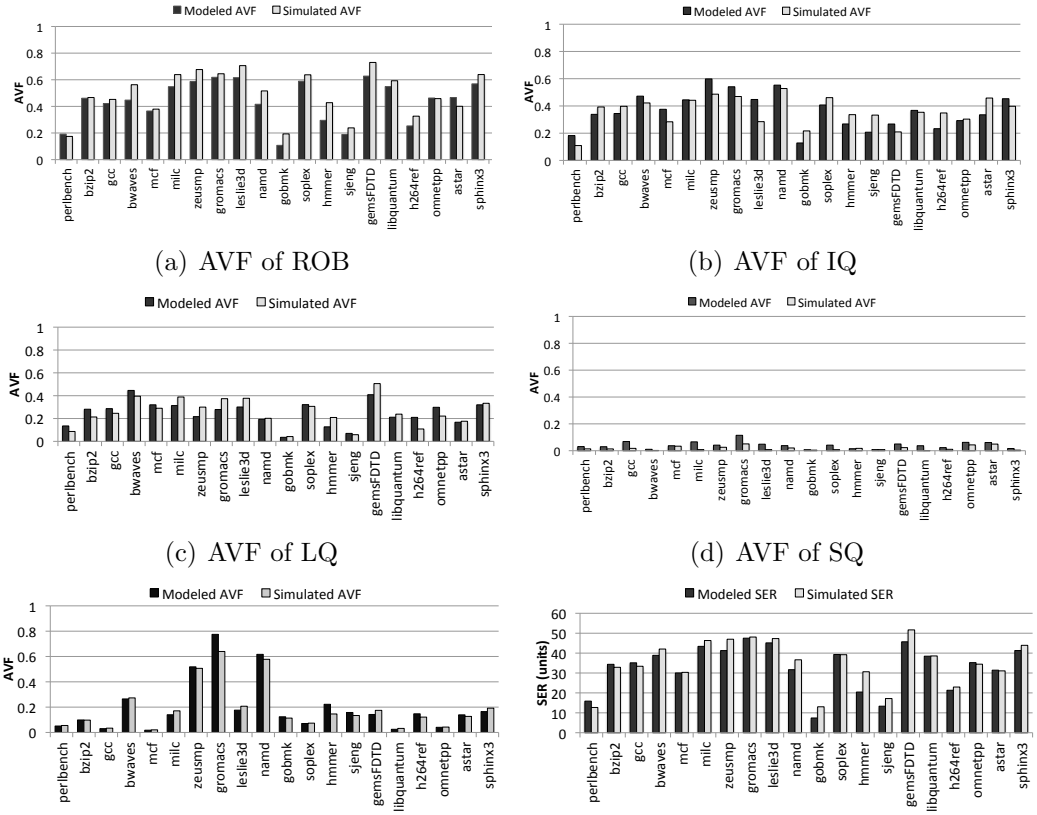
5.7 Results

The AVF of the ROB, IQ, LQ, SQ, and FU is presented in Figure 5.5. The overall SER for these structures is also computed, assuming an arbitrary circuit-level fault rate of 0.01 units/bit, due to the unavailability of real data. However, it allows us to compute the relative error in the SER estimated by the model.

Figure 5.5(a) presents a comparison between the modeled and simulated AVF of the ROB. The mean absolute error in estimating AVF is 0.03, with a maximum error of 0.08 for *hmmer*. As AVF is normalized to the number of bits in the structure (see Equation 2.2), it has a tendency to amplify small errors in small structures. Therefore, for a sense of proportion, the absolute error in estimating SER in terms of the circuit-level fault-rate of each entry in the corresponding structure is expressed. Thus, the mean absolute error in the ROB is equivalent to the fault-rate of 3.8 entries, and the worst-case error is 10.2 entries. In the interest of brevity, in the following discussion, when the absolute SER error is expressed as n entries, it stands to mean "equivalent to the circuit-level fault-rate of n entries in the corresponding structure".

The average absolute IQ AVF error is 0.07 (4.5 IQ entries), with a maximum error of 0.155 for *bwaves*, (9.9 entries) in the IQ. The average absolute LQ and SQ errors are 0.045, and 0.02, respectively, which translates to an error of 2.8 entries, and 1.3 entries, respectively. The maximum errors are 0.09 (*zeusmp*) and 0.06 (*omnetpp*), which translates to the fault-rate of 5.7 entries, and 3.84 entries in the LQ, and SQ, respectively. The mean absolute error for the FUs is 0.01, with a maximum error of 0.05 for *zeusmp*.

Figure 5.5(f) presents the combined SER for the ROB, IQ, LQ, SQ and FU. Root Mean Square Error (RMSE) is typically used to compute the accuracy of a model, and is computed as $\sqrt{\frac{1}{N}\sum_{i=0}^{N}(m_i - a_i)^2}$, where m_i, a_i and N represent the modeled value, actual value, and total number of workloads



(a) AVF of ROB

(b) AVF of IQ

(c) AVF of LQ

(d) AVF of SQ

(e) AVF of FU

(f) Soft Error Rate

Figure 5.6: Modeling the AVF of the Narrow-Issue Machine

respectively. RMSE places higher weights on larger deviations, due to the squaring of errors. Normalized RMSE (NRMSE) is computed by dividing the RMSE by the arithmetic mean of the actual values. The NRMSE for the mechanistic model on the wide-issue machine is 9.0%.

Figure 5.6 presents the AVF and SER for the narrow-issue machine outlined in Table 5.2. The mean absolute error in AVF for the ROB is 0.06 (3.84 entries) with a maximum absolute error of 0.13 (8.3 entries) for *hmmer*. The mean absolute error in estimating the AVF of the IQ is 0.067 (2.14 entries) with a max error of 0.16 (5.12 entries) for *leslie3d*. The mean absolute error for the LQ is 0.047 (1.5 entries) with a maximum error of 0.1 (3.2 entries) for *gemsFDTD*. The average absolute error for the SQ is 0.02 (0.64 entries) with a maximum error of 0.07 (2.24 entries) for *milc*. The average FU AVF error is 0.02, with a maximum of 0.13 for *gromacs*. The SER using the model and simulation is presented in Figure 5.6(f). The NRMSE for the narrow-issue configuration is 10.3% as compared to detailed ACE analysis.

For completeness, the model is also used to estimate the AVF while running the stressmark for the baseline configuration presented in Section 4 on the wide-issue machine. The model estimates the AVF of the ROB with an absolute error of 0.04. Thus, the model estimates AVF for the wide-issue machine with reasonable accuracy. However, the error in estimating the LQ AVF is significantly high. This is because of the atypical nature of the stressmark: address of all loads are dependent on addresses of other last-level cache misses (pointer chasing). The model assumes that loads will issue as

soon as their dependences resolve, and that the dependence is equal to the average steady-state dependence chain latency $l \cdot A(W)$ (see Section 5.4). In the case of the stressmark, all loads are dependent on other last-level cache misses, making their dependence chain latency significantly larger than the average case. This factor can be modeled with additional profiling to detect pointer chasing. However, this is an atypical behavior. It is recommended that the profiling detect cases in which such pointer-chasing occurs, and model it if it occurs frequently.

5.7.1 Potential Sources of Error

The proportion of ACE bits injected in a structure by the program is multiplied with the average occupancy to compute AVF, under the assumption that the proportion of ACE bits induced by the workload remains roughly constant during each interval. This is a reasonable assumption over the simulation points used. Over larger execution lengths, a conservative approach would be to estimate AVF over smaller execution lengths, and combine the results to determine overall AVF. This does not significantly increase the profiling time or AVF estimation time, but may require additional storage.

For workloads such as *hmmer* that incur very few miss events, and are such that the relationship between W and $K(W)$ does not exactly fit a power curve, this approximation may induce errors in modeling the ROB occupancy (See Section 5.2.1). Despite this, the absolute error for *hmmer* is not very large, and the equation is accurate for other workloads.

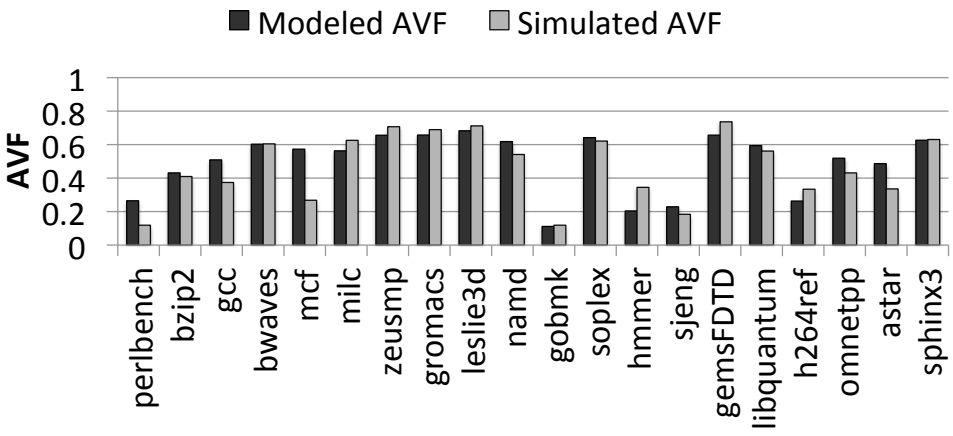


Figure 5.7: Impact of Ignoring the Interaction between Miss Events.

The out-of-order issue of instructions from the IQ causes errors in the estimation of AVF. For example, NOP instructions leave the IQ almost immediately, but are included in the computation of $A(W)$, and the average number of ACE bits induced by the instruction stream. Capturing these effects would require the combination of profiling and ACE analysis. This approach is specifically avoided so that the model can provide insight into the architectural and microarchitectural contributors to AVF, and avoid re-running of profiling and ACE-analysis on microarchitectural changes such as to the fields in each IQ entry, or using a different cache hierarchy.

5.7.2 Impact of Interaction between Miss Events

As noted in earlier discussions, the interaction between data cache and TLB misses with front end events only affects the accuracy in estimating AVF of a subset of workloads. Figure 5.7 illustrates the AVF of the ROB for the

wide-issue machine, computed by assuming that each miss event is independent of all the others. When compared with the ROB AVF error presented in Figure 5.5(a), it is observed that the impact of such interactions is negligible for a majority of workloads. However, workloads *perlbench, gcc, mcf* and *astar* have increased error. This is especially true of *perlbench* and *mcf*, in which a significant fraction of data L2 and TLB misses also have dependent mispredicted branches, or instruction TLB misses, in their shadow. Workloads such as *hmmer* also see an increase in error when the long intervals between a data L2 or DTLB miss and front-end miss is not handled. The mean absolute error in this case is 0.075 (as opposed to 0.03 when interactions are considered), with the maximum error of 0.3 for *mcf* (as opposed to 0.08 when interactions are considered). This graph demonstrates that ignoring these interactions will induce significant error in the estimation of such workloads.

It is argued in Section 5.2.5 that consecutive, or clustered I-cache misses have an insignificant impact on the estimation of correct path state due to their infrequency and low latency. Table 5.3 presents the contribution of *all* I-cache misses that hit in the L2 cache towards overall CPI. In no case do I-cache misses have an influence of more than 6.41% on overall performance, and for most workloads, it is less than 0.5%. Recall from Equation 5.3 that the occupancy during each event is weighted by its contribution to the total number of execution cycles to determine overall occupancy. A significant fraction of these I-cache misses will not be clustered, resulting in a low inaccuracy if clustering is ignored. For workloads that have a significant number of I-cache misses,

Workload	Total Performance Penalty due to I-cache misses (%)	Total Performance Penalty due to Independent Mispredictions in the Shadow of Data Misses (%)
perlbench	6.41	0.33
bzip2	0.00	0.00
gcc	0.01	0.30
bwaves	0.00	2.31
mcf	0.00	0.02
milc	0.00	0.00
zeusmp	0.00	2.04
gromacs	0.00	0.0
leslie3d	0.01	1.03
namd	0.00	1.33
gobmk	6.12	0.67
soplex	0.00	0.78
hmmer	0.00	1.28
sjeng	1.76	0.77
gemsFDTD	0.00	0.1
libquantum	0.00	0.9
h264ref	0.46	0.5
omnetpp	0.37	1.05
astar	0.00	0.0
sphinx3	0.01	1.23

Table 5.3: Contribution of I-cache misses, and branch mispredictions in the shadow of long latency data cache misses, to overall CPI for the wide-issue machine

it may be necessary to model such clustering. The need for this will be indicated by the number of cycles lost as a result of I-cache misses. In this case, the I-cache miss immediately following an earlier I-cache miss will experience lower occupancy which can be computed in a manner similar to an individual I-cache miss.

Table 5.3 also outlines the contribution of branch mispredictions that are in the shadow of the data L2/TLB miss, and independent of long latency data misses to overall modeled CPI. The interval analysis model for CPI does not account for such interactions, as the error in ignoring them is negligible [32, 33]. As argued in Section 5.2.5, it is reasonable to assume that these independent branch mispredictions in the shadow of long-latency cache misses resolve quickly enough such that their overall impact on the average occupancy is negligible, and hence can be ignored. In other words, for the workloads under study, it is reasonable to model these branch mispredictions as if occurring outside the shadow of the blocking long-latency data L2/TLB miss.

5.8 Applications of the Model

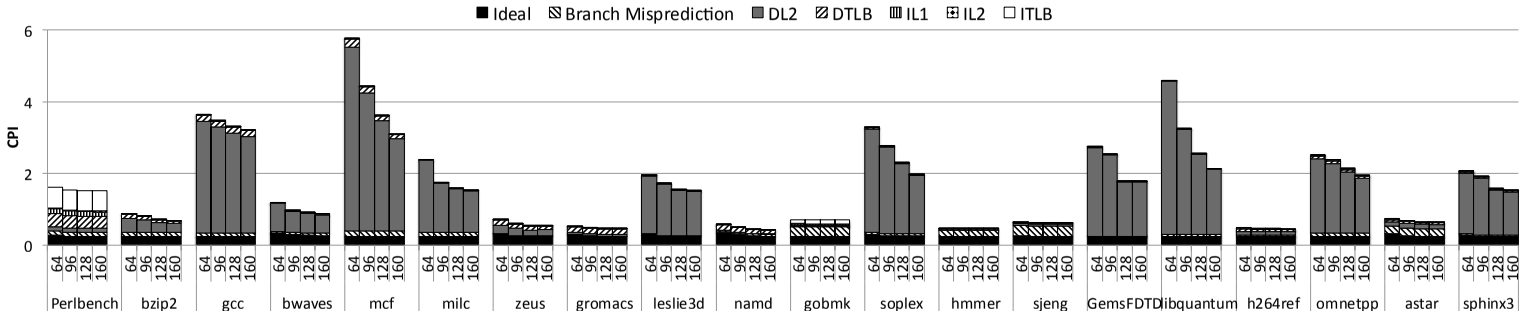
The analytical model can be used to study performance vs. AVF tradeoffs of SER mitigation techniques, the impact of sizing of structures on AVF and performance, compiler optimizations on AVF, different cache sizes and latencies, different branch predictors, etc. In this section, a small subset of these design choices are explored. Specifically, the impact of scaling the ROB, memory latency, and issue width are analyzed.

5.8.1 The Impact of Scaling Microarchitectural Parameters

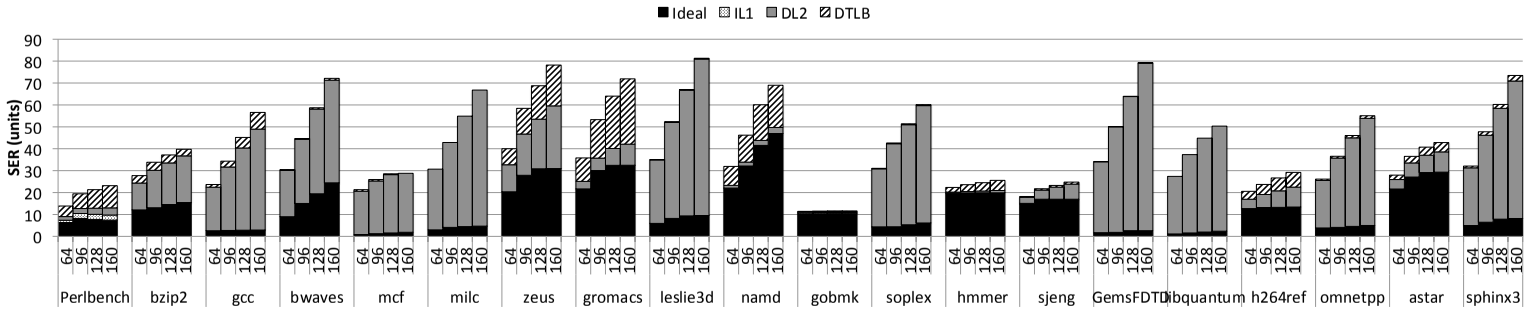
5.8.1.1 Impact of Scaling the ROB on AVF and performance

Sizing studies for AVF and performance are interesting because they allow the architect to determine the trade-off between altering the size of a structure on performance and AVF. For example, it may be reasonable to reduce the ROB size by a small amount provided that it has negligible impact on performance, but significantly reduces SER. Using the model, an architect can instantaneously determine the impact of scaling a structure on AVF and CPI. This section studies the impact of sizing the ROB on AVF and CPI on the wide-issue machine presented in Table 5.2, assuming a circuit-level fault rate of 0.01 units/bit. It is assumed that the IQ, LQ, SQ, FU, and other structures are scaled to maintain the same proportion with the ROB as for the wide-issue machine, so that the processor remains balanced.

Figure 5.8 illustrates the impact of scaling the size of the ROB from 64 to 160 entries, on the wide-issue machine. The trend in SER due to increase in ROB size has two general mechanisms. Workloads for which the ROB is not large enough to be able to sustain an ideal IPC of four will see an increase in the contribution from ideal execution until this is satisfied. Workloads with MLP will be able to exploit it, resulting in fewer stalls due to data L2/TLB misses. However, for larger ROB sizes, the occupancy of instructions in the shadow of these data L2/TLB misses increases as well, resulting in an overall increase in SER. The following discussion presents a few examples for, and



(a) Effect of scaling the ROB size on CPI



(b) Effect of scaling the ROB size on its SER

Figure 5.8: Effect of Scaling ROB Size on its CPI and SER

exceptions to, these mechanisms.

Workloads such as *gobmk* do not see significant change in their CPI or SER due to a large enough ROB, and little available MLP. On the other hand, workloads such as *namd* have a long critical dependency path, which results in high values for $\beta/(\beta-1)$ and l/α (Section 5.2). Consequently, from Equation 5.2, *namd* induces high SER for all ROB sizes despite its low CPI.

For workloads such as *libquantum*, the increase in ROB size provides increased MLP, resulting in lower CPI, but also a greater SER in the shadow of the L2/DTLB miss. *Libquantum* is able to exploit more MLP than *gemsFDTD* resulting in a greater rate of reduction of CPI, and a lesser rate of increase of SER. *Bwaves* and *zeus* experience an increase in SER due to both mechanisms.

Perlbench and *mcf* represent two important exceptions to this general trend. Despite both workloads having a significant number of data L2/TLB misses, *mcf* experiences a significant number of branch mispredictions dependent on data L2 misses, and *perlbench* experiences I-TLB misses, in the shadow of data L2/TLB misses. Consequently, scaling of the ROB size has little impact on SER, as the occupancy of state per cycle does not change significantly. *Mcf* experiences reduction in CPI due to MLP, but the occupancy of ACE state during such misses is still limited by the dependent mispredicted branches.

The scaling study allows the architect to make the appropriate tradeoffs between performance and SER, and understand the factors affecting the

scaling of workloads. For example, the 128-entry ROB provides a speedup of 1.098 (harmonic mean) and increases the average SER by 18% over the 96-entry ROB.

5.8.1.2 Sensititivity of AVF to Memory Latency

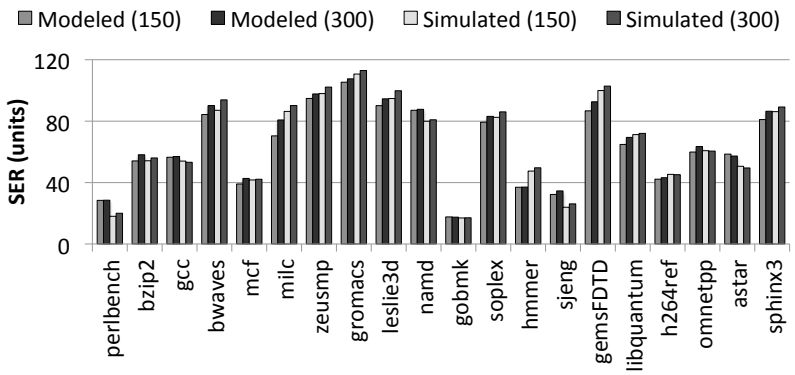
The impact on AVF of changing memory latency, to provide insight into the influence of memory bandwidth contention in CMPs, or *Dynamic Voltage and Frequency Scaling* (DVFS) is an interesting study, because it provides insight into the effect of these commonplace techniques on AVF. Figure 5.9 presents the overall SER for a memory latency of 150 cycles, and 300 cycles, obtained using the model, and from detailed simulation, assuming a constant circuit-level fault rate[1] of 0.01 units/bit. The memory latency used by the model and simulation is enclosed in parentheses. From the formula for AVF (see Equation 2.2), it can intuitively be seen that reduction in memory latency reduces the total number of cycles of ACE bit residency, and the total number of execution cycles. Consequently, the change in AVF in Figure 5.9 is sub-linear, and thus, less sensitive to memory latency when compared with CPI. AVF typically decreases with a decrease in memory latency, although it is mathematically possible for it to increase as well, as seen with *astar*. The comparison with simulation also serves to validate the model. The average change in AVF as predicted by the model is 3.25 units, as compared to 2.22

[1] Although the circuit-level fault rate will significantly increase at low voltages, a constant value provides insight into the change due to AVF

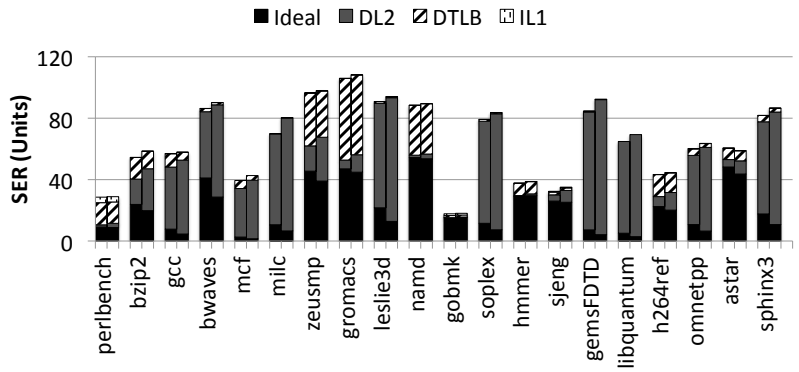
units from simulation. The model faithfully captures the trend for change in SER, with low error. Figure 5.9(b) illustrates the fraction of SER attributable to each event, for a memory latency of 150, and 300 cycles. Although the overall AVF remains nearly the same, the contribution of AVF in the shadow of an L2 miss reduces significantly, for workloads that are dominated by L2 cache misses. Conversely, the relative contribution from ideal execution and DTLB miss increases. The workload experiences fewer cycles from ACE bits in the shadow of an L2 miss, but also fewer execution cycles overall, resulting in a reduction in contribution from the L2 miss, and a greater relative contribution from other misses towards overall SER.

5.8.2 Design Space Exploration

The model can be used to compare different microarchitectures for their impact on performance and AVF/SER. Figure 5.10 presents the CPI and SER of the wide-issue and narrow-issue machine outlined in Table 5.2. The SER is computed for the ROB, LQ, SQ, IQ and FU, and is broken down into its contributing events, so as to provide better insight. On average, there is an 81% increase in SER, and an average speedup of 1.35 (harmonic mean) going from the narrow-issue to the wide-issue configuration. This is attributable to an increase in ROB size and dispatch width. Unlike scaling the ROB size (Section 5.8.1), increasing the issue-width typically increases the SER across all workloads. From Equation 5.2, a larger instruction window is required to sustain a larger dispatch and issue width. For the 20 SPEC CPU2006 work-



(a) Sensitivity of SER to memory latency

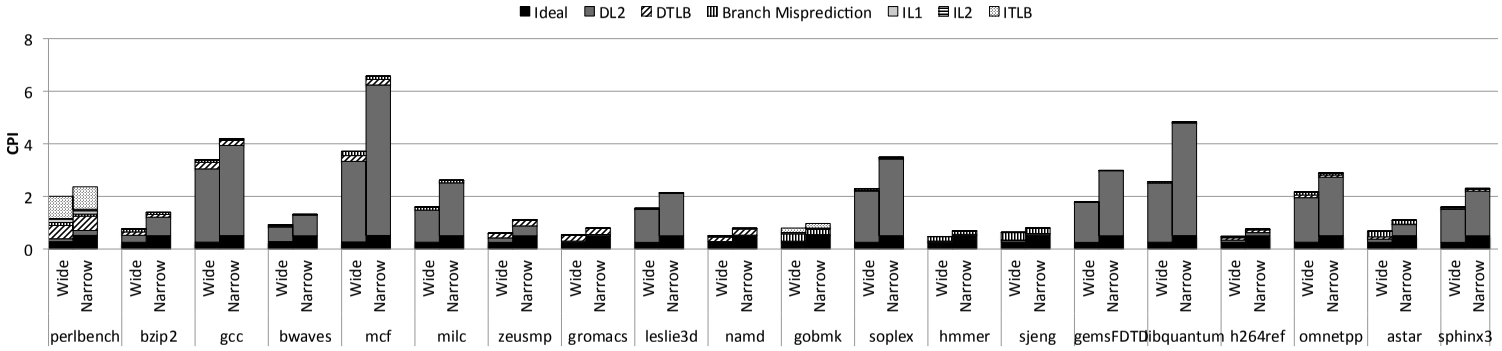


(b) Impact of scaling memory latency from 150 cycles (left) to 300 cycles (right)

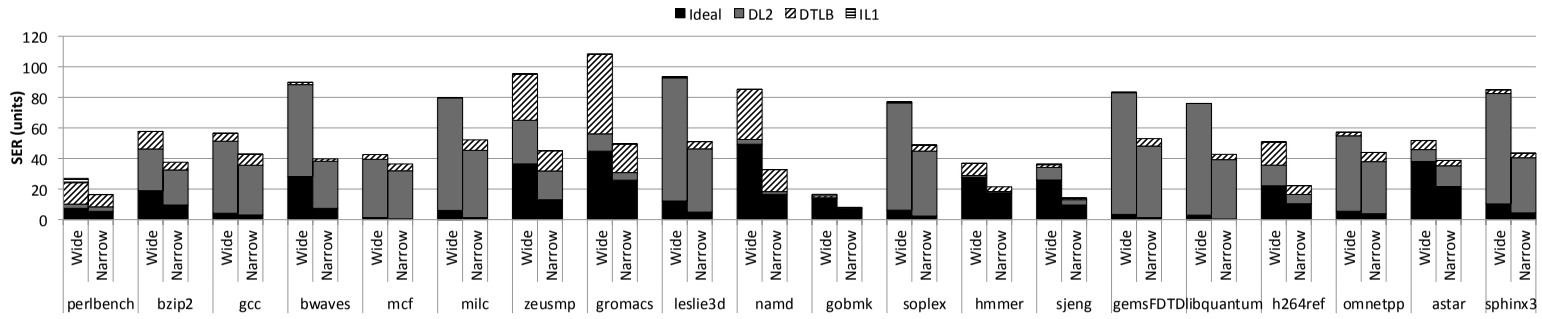
Figure 5.9: Sensitivity of AVF to Memory Latency.

loads considered in the experiments presented herein, β is between 1.24 and 2.39, resulting in a super-linear increase in the ideal occupancy. Although branch resolution time increases with dispatch width, it is reasonable to expect that SER would generally increase with dispatch width, on a balanced design. As noted in Section 5.8.1, *namd* and *bwaves* have long critical paths $K(W)$. The workloads have sufficient ILP for the narrow-issue machine, but not the wide-issue machine, resulting in maximum occupancy of state during ideal execution for the wide-issue case. Additionally, *bwaves* also experiences an increase in SER due to data L2 misses. The SER for *bwaves* and *namd* increases by a factor of 2.26, and 2.6 respectively. On the other hand, *mcf* is unaffected by increase in issue-width or ROB size, due to the large number of dependent mispredicted branches in the shadow of its data L2 misses.

To understand the implications on multi-core design, the SER for a homogeneous Chip Multiprocessor (CMP) using multiple wide-issue and narrow issue under the same area budget are compared. Using the McPAT simulator [59], it is estimated that on a 32nm process, the wide-issue machine (core+cache) has a 65% higher area than the narrow-issue machine. Given that the wide-issue machine has on average 81% higher SER for the ROB, LQ, SQ, IQ, a wide-issue multi-core CMP would be, on average, more vulnerable for these structures, for the same area. Of course, not all structures have been modeled herein (although the larger ones are covered), or the impact of shared resources in the memory hierarchy considered, such as memory bandwidth, of the CMPs. Nevertheless, as the ROB occupancy governs occupancy



(a) CPI stacks for the wide and narrow-issue machine



(b) SER contribution of microarchitectural events for the wide and narrow-issue machine

Figure 5.10: Comparison of CPI and SER of the wide and narrow-issue machines.

of state in most other structures in the core, this result gives some insight on the SER of core structures in the CMPs, for the configurations under consideration. It must also be emphasized that the result presented is only applicable to the microarchitectures under consideration. A different microarchitectural configuration for the wide-issue, narrow-issue, or both configurations may yield different conclusions.

The model can also be used to provide insight into the efficacy of soft error mitigation schemes. Gomaa et al. [13] propose an opportunistic mechanism, called Partial Explicit Redundancy (PER), of enabling Redundant Multi-Threading (RMT) [21] during low IPC events, such as L2/TLB miss, and disabling it during high-IPC intervals, to minimize the performance loss. RMT employs a lagging thread that re-executes the operations of the leading thread and detects faults by comparing the output. Load values and branch outcomes are forwarded by the leading thread so that the lagging thread does not incur any miss penalties, and always runs in the ideal mode. Gomaa et al. [13] report that PER reduces the AVF of the Issue Queue in their microarchitecture by an average of 57%. Using detailed simulation for a specific microarchitecture, Sridharan et al. [60] investigate this effect for the ROB, LQ, SQ, and IQ, and report that nearly 60% of vulnerability occurs in the shadow of a long-stall instruction, most of which are data L2 cache misses. Both studies were performed using workloads from SPEC CPU2000.

Under an optimistic assumption of no performance loss using the opportunistic scheme, the components of SER in Figure 5.10(b) corresponding

to data L2 and TLB misses would disappear. Whereas this scheme generally reduces the AVF of most workloads significantly (resulting in an SER reduction of 66% for the wide-issue machine), *namd* would still have high AVF. Furthermore, *namd* has a high IPC of 3.8 during ideal execution such that enabling RMT would result in roughly doubling the contribution of the ideal interval towards overall CPI (Figure 5.10(a)). Of course, these results are microarchitecture and workload specific. For example, there is an average SER reduction of 60% as computed using the model, when the memory latency of the wide-issue machine is reduced to 150 cycles, as illustrated in Figure 5.9(b). The results obtained using the model are similar to earlier work, and allows architects to estimate the efficacy of such a scheme in the first order, for their microarchitecture and workloads.

5.9 Workload Characterization for AVF

It is difficult to draw inferences on the effect of a workload on the AVF of a structure using aggregate metrics, beyond a qualitative analysis. Aggregate metrics such as cache miss rates or branch misprediction rates provide hints, but as observed in earlier sections, there may be exceptions to such general intuitions of occupancy of state. Given a microarchitecture such as the wide-issue machine, the model enables an architect to identify *namd* as a high-IPC workload inducing high AVF in multiple structures, and *gobmk* as a comparably high-IPC workload that induces very low AVF in the same set of structures. The model provides an explanation as to why workloads such as

mcf with more non-overlapped data cache misses than *bwaves* induces much lower AVF. The model uncovers the complex relationship between various microarchitectural events that combine to induce AVF in a structure. The model provides the break-down of the contribution of each microarchitectural event towards AVF, thereby enabling an intuitive understanding of their influence.

Fu et al. [14] report a "fuzzy relationship" between AVF and simple performance metrics. Therefore, black-box statistical models for AVF that utilize multiple microarchitectural metrics have been proposed by Walcott et al. [9] and Duan et al. [10] for dynamic prediction of AVF. These models use metrics such as average occupancy, and cumulative latencies of instructions in various structures as inputs to the statistical model, which are not available without detailed simualation. Cho et al. [11] utilize a neural-network based methodology for design space exploration, and use it to model AVF of the IQ. As each workload is associated with its own neural network model, training it would potentially require a significant amount of detailed simulations. All these models combine the software and hardware component of AVF, and do not uncover the fundamental mechanisms influencing AVF, thereby providing less insight than the approach presented herein. As the model is constructed from the factors affecting AVF from first principles and explicity models this fuzzy relationship, an architect can identify the precise cause of high or low AVF in a particular structure, and characterize workloads for AVF.

The model enables the architect to study a greater number of workloads and over longer intervals of execution than may be feasible using detailed

simulation, and within the bounds of error of the model, to identify workloads or regions of execution in the workload that induce high AVF in particular structures, enabling better workload characterization for AVF.

5.10 Conclusion

In this work, a first-order mechanistic model for AVF is developed. The model is derived from first-principles of out-of-order execution to provide quantifiable insight into the factors affecting the AVF of structures, and requires only inexpensive profiling. It is shown that this methodology has low mean absolute error of less than 7%, for the ROB, LQ, SQ, IQ and FU.

Additionally, the model quantifies the impact of each microarchitectural event on AVF and SER. The model can be used to nearly instantaneously perform studies on the impact of scaling the ROB size on its AVF and performance, sensitivity of AVF to changes in memory latency, design space exploration comparing the SER of different microarchitectures, and workload characterization. This work enables the architect to identify workloads that would induce high AVF in CPU structures, and characterize workloads for AVF.

Chapter 6

Conclusion

This chapter summarizes the conclusions of the work presented in this dissertation, and outlines future research directions to aid better understanding of the impact of workload execution on the visibility of soft errors, and steps that need to be taken to mitigate them.

6.1 Summary and Conclusions

The problem of soft errors due to sub-atomic particle strikes is becoming significant in current and future process generations, due to an exponential increase in the number of transistors per chip, and the steady lowering of operating and threshold voltages with each generation. Mitigation of soft errors comes with a significant penalty of power, performance, area and design effort, necessitating their judicious application. A good understanding of the impact of workloads on the visibility of soft errors at the program output is critical for efficient design of soft error mitigation schemes. Poor workload characterization may lead to overdesigned, or underdesigned microarchitectures. However, workload characterization for AVF, and selection of a sufficiently heterogeneous workload suite for AVF are an open problem.

Two methodologies to enable the efficient design of soft error mitigation schemes are presented herein. In Chapter 4, an automated methodology for bounding the worst-case soft error rate for a microprocessor running a realizable workload was presented. The knowledge of this worst-case allows the architect to determine whether the workload suite in use has sufficient coverage for SER, and whether an additional guard band is necessary to make up for any lack of coverage, while avoiding potential overdesign or underdesign. Naïve estimations of this worst-case will lead to pessimistic designs. It is demonstrated that this workload achieves 1.4× higher SER in the core, 2.5× higher SER in the data L1 cache and TLB, and 1.5× higher SER in L2 cache as compared to the highest SER induced by SPEC CPU2006 and MiBench programs for a processor similar to the Alpha 21264. The methodology is also demonstrated to be flexible across different microarchitectures. A description of the stressmark methodology presented herein has been published at the forty-third International Symposium on Microarchitecture (MICRO-43) [61].

Chapter 5 presents a first-order mechanistic model to estimate the AVF of any structure in the core whose AVF correlates with its utilization. Derived from the first principles of out-of-order CPU execution, this model is designed to provide insight into the complex interaction between the workload and the microarchitecture that together influence AVF. It is shown that this methodology has a mean absolute error in AVF estimation of less than 7%. The mechanistic model is used to cheaply perform design space exploration and parametric variation studies. The model may be used in conjunction with

cycle accurate simulation by eliminating infeasible design points. More significantly, the construction of this model allows the architect to derive insight into the precise mechanism affecting the AVF of a structure. Owing to its construction, the model can be used for workload characterization for AVF, which is not possible using black-box statistical or machine learning models, or using aggregate metrics reported using cycle-accurate simulation. At the time of writing, this work has been selected to be published at the thirty-nineth International Symposium on Computer Architecture (ISCA-39) [62].

6.2 Future Research Directions

The methodologies presented herein can be extended, or adapted to further improve AVF modeling methodologies. Some of these potential research directions are presented below.

6.2.1 AVF Stressmark for Multicore Machines

The AVF stressmark can be extended for multicore machines, in order to estimate the overall SER of such a chip. In particular, multicore machines have shared resources, such as on-chip interconnect, shared last-level caches, memory controllers, etc. which additionally need to be modeled. The multicore AVF stressmark provides the architect with the worst-case SER possible while running a multithreaded or multiprogrammed workload, and enables design for the same. Ganesan et al. [44] use machine learning to develop a methodology for multi-core power virus generation. They model the data

sharing patterns between various workloads to exercise the interconnect and the memory hierarchy such that power is maximized. A similar methodology can be developed, that leverages this data sharing pattern between cores to maximize SER. Ganesan et al. [44] find that exercising the interconnect and memory hierarchy often results in a reduction in power consumption for the core. Similarly, if the interconnect and memory controllers hold more state than structures in the core, the GA will target these shared structures while sacrificing some AVF/SER in the core itself to maximize overall SER.

6.2.2 Online Estimation of SER

The mechanistic model presented herein may be implemented in hardware to estimate AVF of individual structures, or overall SER. This data may be presented to the OS or software through performance counters, and the OS or software may take remedial action, reducing the amount of hardware intervention. Currently, AVF information is not available to software, and hence software cannot take remedial action. Dynamic prediction of AVF has been proposed in prior work by Walcott et al. [9], Duan et al. [10], and Sridharan and Kaeli [46] as a means of enabling Redundant Multithreading (RMT) only when the SER vulnerability is high. Disabling RMT during periods of low vulnerability avoids the performance penalty of RMT during those periods. The methodologies proposed by Walcott et al. and Duan et al. are regression-based models, and neither provides a hardware implementation for the same. Sridharan and Kaeli [46] propose H-Box hardware to predict AVF when used in

conjunction with Program Vulnerability Factor data obtained from offline profiling, but it requires extensive amount of hardware per structure to estimate AVF. It may be possible to reduce the amount of hardware necessary using the mechanistic model. Eyerman et al. [63, 64] leverage the mechanistic model for CPI to develop CPI stacks for single-threaded, and multi-threaded CPUs. It may be possible to enhance this mechanism to capture the statistics necessary for estimating AVF. A significant challenge with online AVF estimation is ACE analysis; specifically, the detection of dynamically dead instructions without significant investment in hardware. Butts and Sohi [25] report that 3-16% instructions are dynamically dead. Even if instructions are dynamically dead, they are not completely immune to producing soft errors; faults in the target register/address specifiers, or a change in opcode may still result in an error. Consequently, it may not be a significant overestimation to treat dynamically dead instructions as ACE, and achieve a slightly conservative bound on SER.

6.2.3 Estimating per-thread AVF or Resource Sharing in SMT

Simultaneous Multithreaded Processors (SMT) employ multiple hardware contexts to allow sharing of resources within the chip. It is generally difficult to estimate the impact of this resource sharing on AVF. Eyerman and Eeckhout [33] use interval analysis to generate per-thread CPI stacks in SMT processors. On similar lines, it may be possible to estimate the impact of SMT on AVF of the processor. As the mechanistic model proposed herein estimates occupancy, it may also be used to estimate the utilization of shared resources

in the SMT by each hardware context. Although Chen et al. [65] use the interval analysis model for CPI to dynamically partition shared resources, it cannot estimate the utilization of each structure. The knowledge of the utilization of each structure while running a set of workloads enables the architect to perform sizing studies for the structure.

Bibliography

[1] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, pp. 305–316, Sept. 2005.

[2] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," in *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, pp. 29–40, 2003.

[3] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, B. Chin, M. Nicewicz, C. A. Russell, W. Y. Wang, L. B. Freeman, P. Hosier, L. E. LaFave, J. L. Walsh, J. M. Orro, G. J. Unger, J. M. Ross, T. J. O'Gorman, B. Messina, T. D. Sullivan, A. J. Sykes, H. Yourke, T. A. Enger, V. Tolat, T. S. Scott, A. H. Taber, R. J. Sussman, W. A. Klein, and C. W. Wahaus, "IBM experiments in soft fails in computer electronics (1978 – 1994)," *IBM Journal of Research and Development*, vol. 40, pp. 3 –18, January 1996.

[4] S. Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[5] H. Ando, Y. Yoshida, A. Inoue, I. Sugiyama, T. Asakawa, K. Morita, T. Muta, T. Motokurumada, S. Okada, H. Yamashita, Y. Satsukawa, A. Konmoto, R. Yamashita, and H. Sugiyama, "A 1.3-GHz fifth-generation SPARC64 microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 11, pp. 1896 – 1905, 2003.

[6] R. Kalla, B. Sinharoy, W. Starke, and M. Floyd, "POWER7: IBM's Next-Generation Server Processor," *IEEE Micro*, vol. 30, pp. 7 –15, March-April 2010.

[7] S. Mukherjee, J. Emer, and S. Reinhardt, "The soft error problem: an architectural perspective," in *11th International Symposium on High-Performance Computer Architecture, 2005, HPCA-11*, pp. 243 – 247, February 2005.

[8] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," in *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, pp. 61–70, 2004.

[9] K. R. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic prediction of architectural vulnerability from microarchitectural state," in *ISCA '07: Proceedings of the 34th Annual International Symposium on Computer Architecture*, pp. 516–527, ACM, 2007.

[10] L. Duan, B. Li, and L. Peng, "Versatile prediction and fast estimation of Architectural Vulnerability Factor from processor performance metrics,"

in *HPCA 2009: IEEE 15th International Symposium on High Performance Computer Architecture, 2009*, pp. 129–140, February 2009.

[11] C.-B. Cho, W. Zhang, and T. Li, "Informed microarchitecture design space exploration using workload dynamics," in *40th Annual IEEE/ACM International Symposium on Microarchitecture, 2007, MICRO 2007*, pp. 274 –285, December 2007.

[12] H. Nguyen and Y. Yagil, "A systematic approach to SER estimation and solutions," in *41st Annual IEEE International Reliability Physics Symposium Proceedings, 2003.*, pp. 60 – 70, March-April 2003.

[13] M. Gomaa and T. Vijaykumar, "Opportunistic transient-fault detection," in *Proceedings of 32nd International Symposium on Computer Architecture, 2005, ISCA '05*, pp. 172–183, June 2005.

[14] X. Fu, J. Poe, T. Li, and J. Fortes, "Characterizing Microarchitecture Soft Error Vulnerability Phase Behavior," in *14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006.*, pp. 147 – 155, September 2006.

[15] J. Ziegler and H. Puchner, *SER–history, trends and challenges: a guide for designing with memory ICs*. `http://www.cypress.com/?rID=14793`, Cypress, 2004.

[16] T. May and M. Woods, "Alpha-particle-induced soft errors in dynamic memories," *IEEE Transactions on Electron Devices*, vol. 26, pp. 2 – 9, January 1979.

[17] E. Normand, "Single event upset at ground level," *IEEE Transactions on Nuclear Science*, vol. 43, pp. 2742 –2750, Dec 1996.

[18] S. Mukherjee, *Architecture Design For Soft Errors*. Burlington, MA, USA: Morgan Kauffman Publishers, 2008.

[19] S. Michalak, K. Harris, N. Hengartner, B. Takala, and S. Wender, "Predicting the number of fatal soft errors in Los Alamos National Laboratory's ASC Q supercomputer," *IEEE Transactions on Device and Materials Reliability*, vol. 5, pp. 329 – 335, Sept. 2005.

[20] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proceedings of the International Conference on Dependable Systems and Networks, DSN 2002*, pp. 389–398, 2002.

[21] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, pp. 25–36, 2000.

[22] X. Li, S. Adve, P. Bose, and J. Rivers, "SoftArch: An Architecture Level Tool for Modeling and Analyzing Soft Errors," in *DSN '05: Proceedings of*

the 2005 International Conference on Dependable Systems and Networks, pp. 496–505, 2005.

[23] P. Hazucha and C. Svensson, "Impact of CMOS technology scaling on the atmospheric neutron soft error rate," *IEEE Transactions on Nuclear Science*, vol. 47, pp. 2586 –2594, Dec. 2000.

[24] P. Liden, P. Dahlgren, R. Johansson, and J. Karlsson, "On latching probability of particle induced transients in combinational networks," in *Digest of Papers, Twenty-Fourth International Symposium on Fault-Tolerant Computing, 1994. FTCS-24*, pp. 340 –349, June 1994.

[25] J. A. Butts and G. Sohi, "Dynamic dead-instruction detection and elimination," in *ASPLOS-X: Proceedings of the 10th international conference on Architectural Support for Programming Languages and Operating Systems*, pp. 199–210, 2002.

[26] K. Zick and J. Hayes, "High-level vulnerability over space and time to insidious soft errors," in *IEEE International High Level Design Validation and Test Workshop, 2008. HLDVT '08.*, pp. 161 –168, November 2008.

[27] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," in *ISCA '05: Proceedings of 32nd International Symposium on Computer Architecture*, pp. 532–543, June 2005.

[28] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumptions," in *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 266–275, 2007.

[29] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ace analysis reliability estimates using fault-injection," in *Proceedings of the 34th annual International Symposium on Computer Architecture*, ISCA '07, (New York, NY, USA), pp. 460–469, ACM, 2007.

[30] A. Biswas, P. Racunas, J. Emer, and S. Mukherjee, "Computing Accurate AVFs using ACE Analysis on Performance Models: A Rebuttal," *IEEE Computer Architecture Letters*, vol. 7, pp. 21–24, January 2008.

[31] N. Wang, M. Fertig, and S. Patel, "Y-branches: When you come to a fork in the road, take it," in *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, PACT '03, (Washington, DC, USA), pp. 56–67, IEEE Computer Society, 2003.

[32] T. Karkhanis and J. Smith, "A first-order superscalar processor model," in *Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004*, pp. 338–349, June 2004.

[33] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A mechanistic performance model for superscalar out-of-order processors," *ACM Transactions on Computer Systems*, vol. 27, pp. 3:1–3:37, May 2009.

[34] P. Michaud, A. Seznec, and S. Jourdan, "Exploring Instruction-Fetch Bandwidth Requirement in Wide-Issue Superscalar Processors," in *PACT '99: Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques*, pp. 2–10, 1999.

[35] E. M. Riseman and C. C. Foster, "The Inhibition of Potential Parallelism by Conditional Jumps," *IEEE Transactions on Computers*, vol. 21, pp. 1405–1411, December 1972.

[36] J. W. Kellington, R. McBeth, P. Sanda, and R. N. Kalla, "IBM POWER6 Processor Soft Error Tolearance Analysis Using Proton Irradiation," in *SELSE 07: Third workshop on System Effects of Logic Soft Errors,* `http://www.selse.org`, 2007.

[37] P. N. Sanda, J. W. Kellington, P. Kudva, R. Kalla, R. B. McBeth, J. Ackaret, R. Lockwood, J. Schumann, and C. R. Jones, "Soft-error resilience of the IBM POWER6 processor," *IBM Journal of Research and Development*, vol. 52, no. 3, pp. 275–284, 2008.

[38] A. Sanyal, K. Ganeshpure, and S. Kundu, "On Accelerating Soft-Error Detection by Targeted Pattern Generation," in *8th International Symposium on Quality Electronic Design, 2007. ISQED '07.* , pp. 723–728, March 2007.

[39] A. Sanyal, K. Ganeshpure, and S. Kundu, "Accelerating Soft Error Rate Testing Through Pattern Selection," in *13th IEEE International On-Line Testing Symposium, 2007. IOLTS 07*, pp. 191–193, July 2007.

[40] A. Sanyal, K. Ganeshpure, and S. Kundu, "An Improved Soft-Error Rate Measurement Technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, pp. 596–600, April 2009.

[41] A. M. Joshi, L. Eeckhout, L. K. John, and C. Isen, "Automated microprocessor stressmark generation," in *Tenth International Symposium on High Performance Computer Architecture (HPCA)*, pp. 229–239, 2008.

[42] S. Polfliet, F. Ryckbosch, and L. Eeckhout, "Automated full-system power characterization," *Micro, IEEE*, vol. 31, no. 3, pp. 46 –59, 2011.

[43] K. Ganesan, J. Jo, W. L. Bircher, D. Kaseridis, Z. Yu, and L. K. John, "System-level max power (SYMPO): a systematic approach for escalating system-level power consumption using synthetic benchmarks," in *Proceedings of the 19th international conference on Parallel Architectures and Compilation Techniques*, PACT '10, pp. 19–28, 2010.

[44] K. Ganesan and L. K. John, "MAximum Multicore POwer (MAMPO): An automatic multithreaded synthetic power virus generation framework for multicore systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, (New York, NY, USA), pp. 53:1–53:12, ACM, 2011.

[45] V. Sridharan and D. Kaeli, "Eliminating microarchitectural dependency from Architectural Vulnerability," in *IEEE 15th International Symposium on High Performance Computer Architecture, HPCA 2009*, pp. 117–128, February 2009.

[46] V. Sridharan and D. R. Kaeli, "Using hardware vulnerability factors to enhance AVF analysis," in *ISCA '10: Proceedings of the 37th annual International Symposium on Computer Architecture*, pp. 461–472, ACM, 2010.

[47] X. Fu, T. Li, and J. Fortes, "Sim-SODA: A framework for microarchitecture reliability analysis," in *Proceedings of the Workshop on Modeling, Benchmarking and Simulation (Held in conjunction with International Symposium on Computer Architecture)*, 2006.

[48] R. Desikan, D. Burger, S. W. Keckler, and T. Austin, "Sim-alpha: A Validated, Execution-Driven Alpha 21264 Simulator," in *Tech report TR-01-23, The University of Texas at Austin*, 2001.

[49] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *SIGARCH Computer Architecture News*, vol. 25, pp. 13–25, June 1997.

[50] D. Beasley, D. R. Bull, and R. R. Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals, University Computing, `http://ralph.cs.cf.ac.uk/papers/GAs/ga_overview1.pdf`," 1993.

[51] SPEC, "Standard performance evaluation corporation, `http:\\www.spec.org`."

[52] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A free, commercially representative embedded benchmark

suite," in *IEEE International Workshop on Workload Characterization, 2001. WWC-4. 2001*, pp. 3 – 14, Feb. 2001.

[53] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *ASPLOS-X: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 45–57, 2002.

[54] P. K. Dubey, G. B. Adams III, and M. J. Flynn, "Instruction window size trade-offs and characterization of program parallelism," *IEEE Transactions on Computers*, vol. 43, pp. 431 –442, April 1994.

[55] D. Noonburg and J. P. Shen, "A Framework for Statistical Modeling of Superscalar Processor Performance," in *Proceedings of International Symposium on High Performance Computer Architecture (HPCA)*, pp. 298–309, 1997.

[56] J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 16, pp. 122 –128, January 1986.

[57] M. Srinivas and L. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, pp. 656 –667, April 1994.

[58] T. S. Karkhanis and J. E. Smith, "Automated design of application specific superscalar processors: an analytical approach," in *Proceedings of*

the 34th annual international symposium on Computer architecture, ISCA '07, pp. 402–411, 2007.

[59] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pp. 469–480, ACM, 2009.

[60] V. Sridharan, D. Kaeli, and A. Biswas, "Reliability in the Shadow of Long-Stall Instructions," in *SELSE 07: Third workshop on System Effects of Logic Soft Errors,* `http://www.selse.org`, 2007.

[61] A. A. Nair, L. K. John, and L. Eeckhout, "AVF Stressmark: Towards an Automated Methodology for Bounding the Worst-Case Vulnerability to Soft Errors," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, (Washington, DC, USA), pp. 125–136, IEEE Computer Society, 2010.

[62] A. A. Nair, S. Eyerman, L. Eeckhout, and L. K. John, "A First-Order Mechanistic Model for Architectural Vulnerability Factor," in *The Proceedings of the 39th annual IEEE/ACM International Symposium on Computer Architecture*, ISCA '12, IEEE, 2012.

[63] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A Performance Counter Architecture for Computing Accurate CPI components," in *Proceedings of the 12th International Conference on Architectural Support for*

Programming Languages and Operating Systems, (New York, NY, USA), pp. 175–184, ACM, 2006.

[64] S. Eyerman and L. Eeckhout, "Per-thread cycle accounting in SMT processors," in *Proceedings of the 14th international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '09, pp. 133–144, 2009.

[65] J. Chen and L. K. John, "Predictive coordination of multiple on-chip resources for chip multiprocessors," in *Proceedings of the international conference on Supercomputing*, ICS '11, pp. 192–201, 2011.