A First-Order Mechanistic Model for Architectural Vulnerability Factor

Arun Arvind Nair† Stijn Eyerman‡ Lieven Eeckhout‡ Lizy Kurian John†
†The University of Texas at Austin ‡Ghent University, Belgium
{*nair,ljohn*}@*ece.utexas.edu* {*leeckhou,seyerman*}@*elis.ugent.be*

Abstract

Soft error reliability has become a first-order design criterion for modern microprocessors. Architectural Vulnerability Factor (AVF) modeling is often used to capture the probability that a radiation-induced fault in a hardware structure will manifest as an error at the program output. AVF estimation requires detailed microarchitectural simulations which are time-consuming and typically present aggregate metrics. Moreover, it requires a large number of simulations to derive insight into the impact of microarchitectural events on AVF. In this work we present a first-order mechanistic analytical model for computing AVF by estimating the occupancy of correct-path state in important microarchitecture structures through inexpensive profiling. We show that the model estimates the AVF for the reorder buffer, issue queue, load and store queue, and functional units in a 4-wide issue machine with a mean absolute error of less than 0.07. The model is constructed from the first principles of out-of-order processor execution in order to provide novel insight into the interaction of the workload with the microarchitecture to determine AVF. We demonstrate that the model can be used to perform design space explorations to understand trade-offs between soft error rate and performance, to study the impact of scaling of microarchitectural structures on AVF and performance, and to characterize workloads for AVF.

1 Introduction

The mitigation of radiation-induced soft errors has emerged as a key design challenge in current and future process technologies, as a result of increasing transistor densities and lowering of operating voltages [1–4]. However, a significant fraction of radiation-induced faults do not affect the correctness of program execution [5]. Therefore, *Architectural Vulnerability Factor* (AVF) modeling is utilized to quantify this masking effect of program execution, enabling designers to devise efficient soft error mitigation schemes. AVF captures the probability that a fault in a structure will manifest as an error in the program output, and can be modeled using *Architecturally Correct Execution* (ACE) analysis [2] or SoftArch [6] using microarchitectural simulators. Whereas AVF modeling using ACE analysis allows the architect to estimate *Soft Error Rate* (SER) during early de-

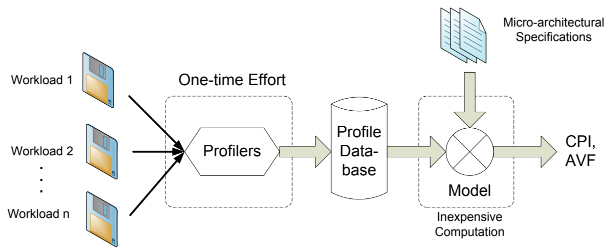


Figure 1. Utilizing the model to perform design space exploration.

sign planning, it requires a large number of time-consuming microarchitectural simulations to be able to derive insight into the relationship between a microarchitectural change and AVF. Additionally, these simulations typically present the combined 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. Statistical or machine-learning based modeling [7–10] also do not easily quantify the fundamental interactions between the workload and the microarchitecture, making it difficult to derive insight into the factors affecting AVF.

In this work, we develop a modeling methodology to analytically obtain the AVF of a structure, in the first order, using statistics collected from relatively inexpensive profiling. Our model allows the architect to quantitatively understand the factors affecting AVF and guide higher level design decisions. The core idea behind this methodology is to divide program execution into intervals [11]. Each interval is an independent region of execution delimited by miss events that disrupt the dispatch of instructions. We model the occupancy of state that will eventually be committed by the processor in each interval. The occupancy of correct-path state during each interval is averaged, weighted by the time spent in each interval. The AVF of a structure is then estimated by derating this occupancy with the average proportion of un-ACE bits induced in it by the workload. Additionally, our model is deliberately constructed to capture the interaction of the various miss events, and their combined impact on occupancy. This allows us to derive novel quantitative insights into the workload's influence on AVF of a structure, that may not be obvious from the aggregate metrics such as cache miss rates. We use the same terminology as Eyerman et al. [11] to refer to such "whitebox" models as *mechanistic* models, in contrast to "black-box" statistical or machine-learning based analytical models. Figure 1 presents the general overview of our modeling methodology. Workloads are profiled to capture important metrics required by the model, which is often a one-time effort. Multiple microarchitectures can then be modeled using the data from a single profile.

Our methodology can be used to model the AVF of any structure whose AVF correlates with its utilization, and whose utilization is influenced by program execution. In this paper, we demonstrate the methodology for estimating the AVF of the reorder buffer (ROB), issue queue (IQ), load and store queues (LQ, SQ), and functional units (FUs). We show that the mean absolute error in estimating AVF for each of these structures is less than 0.07, for a 4-wide out-of-order machine, as compared to ACE analysis using a cycle-accurate microarchitectural simulator.

The key contributions of this work are as follows:

- We present 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. The key novelty of this modeling effort over prior mechanistic models for performance [11, 12] is that it captures the interaction between different events occurring in a processor, and estimates AVF 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 prior work using black-box models [7–10].
- As the model requires inexpensive profiling, it can be used to perform design space exploration studies nearly instantaneously. In this work, we demonstrate how the model can be 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.
- We demonstrate how the model can be used for workload characterization for AVF. As the model quantitatively expresses the exact mechanisms influencing AVF, it can be used to identify high or low AVF inducing workloads. We also demonstrate why aggregate metrics such as IPC or cache miss rates do not correlate with the AVF of a structure.

The remainder of this paper is organized as follows: Section 2 outlines AVF modeling using ACE analysis, and the interval analysis methodology for modeling performance. We outline our modeling methodology and associated tradeoffs in Section 3. We compare the AVF as predicted by the model, to the AVF computed using detailed simulation in Section 4. Finally, we utilize the model to derive insights into the effect of microarchitectural changes on AVF in Section 5.

2 Background

ACE Analysis: In order to compute AVF, Mukherjee et al. [2] introduce the concept of *Architecturally Correct Execution* (ACE) bits. An ACE bit is one whose correctness is required for the correctness of the program. A bit could be either microarchitecturally or architecturally ACE. Bits that are not critical to program correctness are termed *un-ACE*. Microarchitecturally un-ACE bits include bits due to unused or invalid state, bits discarded as a result of misspeculation, or bits in predictor structures. Architecturally un-ACE bits are a direct result of the instructions in the binary, such as NOPs, software pre-fetches, predicated false instructions, and dynamically dead instructions.

Mukherjee et al. formally define the AVF of a structure of size N bits, as $AVF_{structure} = \frac{1}{N} \times \sum_{i=0}^{N} \left(\frac{ACE\ cycles\ for\ bit\ i}{Total\ Cycles} \right)$. Thus, AVF of a structure is expressed as the average number of ACE bits per cycle divided by the total number of bits in the structure. AVF is used to derate the intrinsic fault rate of the structure to estimate its *Soft Error Rate* (SER). The derated fault rates for all structures are added up to estimate the overall SER of the processor.

Interval Analysis: Our first-order mechanistic model for AVF is inspired by earlier work for estimating Cycles per Instruction (CPI) for out-of-order superscalar architectures, proposed by Karkhanis and Smith [12], and refined by Eyerman et al. [11] using *interval analysis*. In the interest of brevity, we only present the basic ideas here. Interval analysis models the program execution as an ideal, miss-free execution, interrupted by miss events that disrupt the dispatch of instructions. In the absence of any miss events, the processor is able to dispatch instructions at the maximum dispatch rate D[1]. Each miss event interrupts the dispatch of instructions until it resolves. 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 cycles towards estimating performance, for a given instruction window size. We use the term instruction window to refer to the instructions in flight, or the ROB. 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, which is the number of cycles between the mispredicted branch entering the instruction window and the misprediction being detected.

Therefore, the total number of cycles for executing a program is modeled as the sum of the cycles spent in each interval. Miss events that would not interrupt dispatch, such as data cache hits, are modeled similar to arithmetic instruc-

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

tions. The model assumes a microarchitecture design such that the processor would not frequently stall in the absence of miss events, while running typical workloads. Karkhanis and Smith [12], and Eyerman et al. [11] 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 estimating occupancy and AVF. For example, a mispredicted branch that is dependent on a load miss in the L2 cache significantly reduces the occupancy of correct-path (ACE) bits in the shadow of the L2 miss, and is non-trivial to estimate using the existing interval model or aggregate metrics. It is therefore necessary for our AVF model to account for such interactions.

The profilers for the model are implemented as sliding windows, and collect statistics for a range of instruction window sizes [11]. Thus, ROB size, issue width, front-end pipeline depth and the latencies of instructions, caches, TLBs, and main memory can be changed without requiring additional profiling. If the cache hierarchy or the branch predictor is changed, the corresponding profiler would need to be rerun. Our model retains the same flexibility to allow easy design space exploration.

3 Modeling AVF using Interval Analysis

The unique requirements of our model are to capture the occupancy of state in a given structure in the core, while discarding the occupancy of wrong-path instructions, and faithfully modeling the complex interaction between front-end and data cache/TLB misses that affect AVF.

In this section, we describe the methodology for estimating the occupancy of correct-path state of the ROB, LQ, SQ, IQ, and FUs, which contain the largest amount of corruptible state in the core. We then estimate AVF by derating this occupancy by the fraction of bits introduced into a structure that were un-ACE. We identify un-ACE instructions through profiling and use this information to determine the number of ACE bits injected into 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, we seek to balance accuracy with simplicity of formulation, the ability to provide quantitative insight, and ease of collecting necessary program characteristics.

Our modeling methodology must account for the effect of interaction between miss events – ignored in interval analysis for CPI – on the occupancy of correct-path state. The data necessary to achieve this is easily added to the profiling necessary for interval analysis, incurring negligible overhead.

We estimate the occupancy of eventually committed state in the ROB in Section 3.1. The ROB occupancy governs the occupancy of LQ, SQ and FUs. As the IQ can issue instructions out-of-order, its occupancy is estimated independently in Section 3.2.

3.1 Modeling the AVF of the ROB

In this section, we study the effect of each miss event on the occupancy of the ROB independently of one another, and analyze the impact of interaction between miss events. Figure 2 illustrates a simplified view of modeling occupancy using interval analysis. We model the occupancy of the ROB 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. We linearize the ramp-up and ramp-down curves for occupancy, with slopes equal to the dispatch rate, in the interest of simplicity. It is assumed for the purposes of this work that the designed dispatch and retire widths for the processor are equal.

3.1.1 Occupancy of Correct-Path Instructions

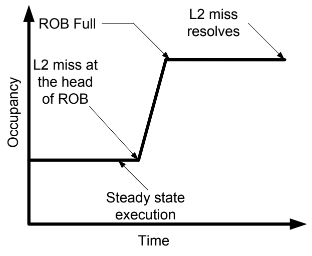
The interval model for performance allows us to estimate the number of cycles spent during each execution interval. Our model allows us to estimate the occupancy of correct-path instructions in the ROB during these intervals. Thus, average occupancy of the ROB, computed over the execution of the program is:

$$\begin{aligned} O^{ROB}_{avg} = \frac{1}{C_{total}} \cdot (&O^{ROB}_{ideal} \cdot C_{ideal} + O^{ROB}_{DL2Miss} \cdot C_{DL2Miss} \\ &+ O^{ROB}_{IL1Miss} \cdot C_{IL1Miss} + O^{ROB}_{ITLBMiss} \cdot C_{ITLBMiss} \\ &+ O^{ROB}_{brMp} \cdot C_{brMp} + O^{ROB}_{DTLBMiss} \cdot C_{DTLBMiss}) \end{aligned} \quad (1)$$

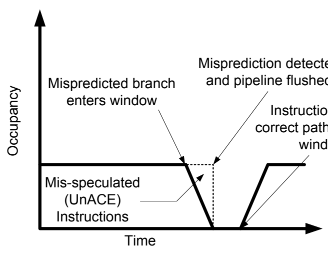
In the above equation, C refers to the total number of cycles, and O refers to the occupancy of state during each type of miss event. In essence, Equation 1 is the average occupancy, weighted by the number of cycles spent in each interval. In the following sections, we describe how the occupancy of state during each miss event is computed.

3.1.2 Modeling Steady-State Occupancy

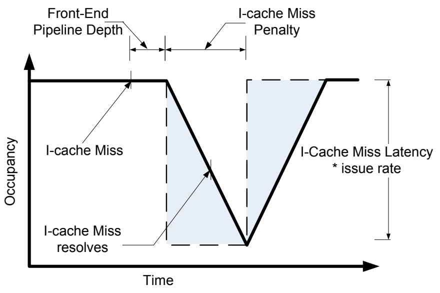
Given an instruction window of size W, the total number of cycles taken to execute all instructions in the instruction window is a function of the latency of executing the critical path. The average critical path length $K(W)$ for a given program is modeled as $K(W) = \frac{1}{\alpha}W^{1/\beta}$ [12, 13] 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 or steady-state IPC ($I(W)$) that



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


Misprediction detected, and pipeline flushed.
Mispredicted branch enters window
Instructions from correct path enter the window
Occupancy
Mis-speculated (UnACE) Instructions
Time

(b) Occupancy of the ROB during a branch misprediction.



(c) Modeling Occupancy During an I-cache Miss.

Figure 2. Modeling Occupancy of the ROB Using Interval Analysis.

can be extracted from the program given an instruction window of size W is presented in Equation 2 below [12, 13]. For a processor with a designed dispatch width D, setting $I(W) = D$, and rearranging the terms in Equation 2 gives us the steady-state ROB occupancy, O^{ROB}_{ideal}, or $W(D)$ necessary to sustain the peak dispatch rate.

$$I(W) = \frac{W}{l \cdot K(W)} = \frac{\alpha}{l}.W^{(1-1/\beta)} \qquad (2)$$

$$\therefore O^{ROB}_{ideal} = W(D) = \left(\frac{l \cdot D}{\alpha}\right)^{\frac{\beta}{\beta-1}} \qquad (3)$$

If the steady-state 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 [12], a processor that has a balanced design for a typical workload will not frequently stall due to a full IQ. Therefore, we consider only the typical case for our modeling. An implicit assumption of Equation 2 is that the product of the longest dependence chain length and average latency ($l \cdot K(W)$) is approximately equal to the latency of executing the critical dependence path. This approximation may induce errors for pathalogical workloads with few miss events. However, the separation of critical path and latency profiling allows us to easily change instruction or cache latencies without re-profiling the workload.

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

As shown in Figure 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 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. However, the occupancy of the ROB need not return to steady-state after the L2 miss completes; it can remain at nearly 100% if the processor is capable of dispatching and retiring instructions at the same rate. In Section 3.1.5, we explain the procedure for accounting for this interaction.

3.1.4 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 2(c), 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 2), 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 2). Linearizing ramp-up and ramp-down[2], the shaded areas under the ramp-up and ramp-down are equal, allowing us to approximate occupancy as depicted by the dotted line. As depicted in Figure 2(c), the reduction in correct-path state during an I-cache miss is $lat_{L2} \cdot D$, where lat_{L2} cycles is the hit latency of the L2 cache, and D is the steady-state dispatch or retirement rate. Thus, $O^{ROB}_{IL1Miss} = O^{ROB}_{ideal} - lat_{L2} \cdot D$. 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 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

[2]Although it is possible to compute the exact ramp-up and ramp-down curves using Equation 2 and 3, the error due to linearization is negligible.

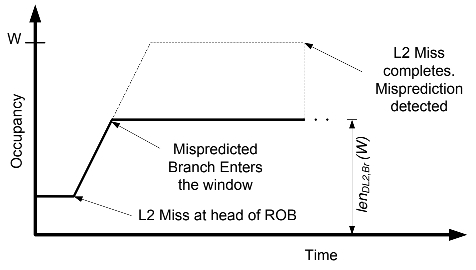
state decreases. The overall occupancy, as indicated using the dotted line remains at the steady-state value, until the branch misprediction is detected and the pipeline is flushed. Karkhanis and Smith [12] show that assuming an oldest-first issue policy, the mispredicted branch is among the last correct-path 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.

3.1.5 Modeling Interactions Between Miss Events.

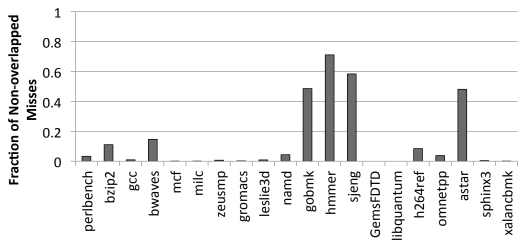
Dependent Branch Mispredictions in the Shadow of a Long-Latency Load Miss: Consider the case in which a branch is dependent on a long-latency load miss and occurs within the same instruction window. If such a branch is mispredicted, all instructions in the ROB fetched after the branch instruction are 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 less than 100%, as shown in Figure 3(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 such that their interaction has a negligible effect on occupancy.

We capture this interaction 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 ($N_{dep}(W)$). The average number of instructions between the *earliest* dependent mispredicted branch and the non-overlapped miss at the head of the ROB ($len_{DL2,Br}(W)$, $len_{DTLB,Br}(W)$) is also captured to estimate the occupancy of correct-path state in the shadow of the L2 miss. Note that the 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. For a total number of non-overlapped L2 misses $N_{DL2Misses}(W)$, we express the term $O_{DL2Miss}^{ROB} \cdot C_{DL2Miss}$ in Equation 1 as $lat_{DL2Miss} \cdot (len_{DL2,Br}(W) \cdot N_{dep}(W) + W \cdot (N_{DL2Miss}(W) - N_{dep}(W)))$.

Interaction of Data and Instruction Cache Misses: Our model is impacted by two types of interactions between data cache and instruction cache miss events. The first case occurs when an instruction miss in the L2 cache or ITLB occurs in the shadow of a non-overlapped DL2 or DTLB miss, resulting in less than 100% occupancy of the ROB. We find


W
Occupancy
L2 Miss completes. Misprediction detected
Mispredicted Branch Enters the window
L2 Miss at head of ROB
$len_{DL2Br}(W)$
Time

(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 3. Modeling the Effects of Interactions Between Miss Events on Occupancy.

that this case is very rare; only *perlbench* is significantly impacted due to its higher proportion of ITLB misses. We follow a procedure similar to the aforementioned case of dependent branch instructions. 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 occurs when the duration between a long-latency data cache miss and front-end miss is too long. As described in Section 3.1.3, we made 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. We therefore measure the fraction of non-overlapped DL2 and DTLB misses that are separated from a front-end miss by *at least* $2W$ instructions. 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, we only capture the length of long intervals, and filter out the impact of very short intervals on the average. Figure 3(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 1. As seen in Figure 3(b), with the exception of *hmmer, gobmk, sjeng*, and *astar*, this situation occurs infrequently. The average length of such sequences for these workloads ranges between 300 and 450 instructions. Thus, the fraction of non-overlapped L2 misses in

Parameter	Wide Issue Machine	Narrow Issue Machine
ROB	128 entries, 76 bits/entry	64 entries, 76 bits/entry
Issue Queue	64 entries, 32 bits/entry	32 entries, 32 bits/entry
LQ	64 entries, 80 bits/entry	32 entries each, 80 bits/entry
SQ	64 entries, 144 bits/entry	32 entries each, 144 bits/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 1. Processor Configurations.

Figure 3(b) will experience a subsequent region of ideal execution in which occupancy is W, which is included in our calculations for the model. We perform a similar experiment to capture long intervals between two consecutive data L2/TLB misses, but find that they are infrequent, and affect only workloads dominated by these misses. Although included in our modeling, they have negligible impact on average occupancy 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 one another, their impact on occupancy is separable, in the first order. The impact of consecutive I-cache misses occurring very close in time to each other is ignored due to their relative infrequency, and their low impact on average occupancy.

3.2 Modeling the AVF of the IQ

The occupancy of the IQ requires separate modeling as instructions can issue out-of-order. We assume an oldest-first issue policy for our model. Occupancy of correct-path instructions during front-end misses is modeled in a manner similar to that of the ROB. Ideal occupancy, and occupancy in the shadow of a long-latency data cache or TLB miss is 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, $O^{IQ}_{ideal} = l \cdot A(W) \cdot min(D, I(W))$ [14].

Occupancy in the Shadow of a Long-Latency Load Miss: When issue of instructions ceases in the shadow of an L2/TLB load miss, the IQ contains only the instructions dependent on such misses. We measure the average number of instructions dependent on the L2 and DTLB misses in the instruction window 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 with no overhead. We also capture the effect of interactions between data L2/TLB misses and front-end misses, similar to the procedure outlined in Section 3.1.5.

3.3 Modeling the AVF of LQ, SQ and FU

The occupancy of the LQ, SQ and FUs 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.

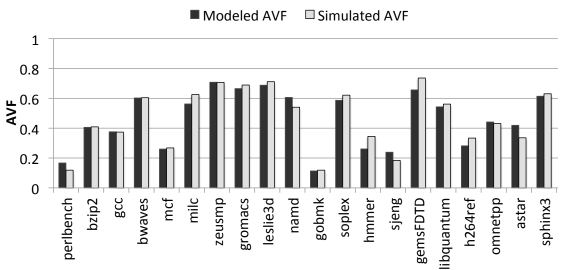
Loads and stores enter the LQ and SQ after they are issued, and remain there until they are retired. As we have seen in Section 3.2, the average dispatch-to-issue latency for an instruction is $l \cdot A(W)$ cycles. Thus, the LQ and SQ occupancy can be estimated as the fraction of loads and stores in the ROB, adjusted for the average dispatch-to-issue latency of the loads/stores in the instruction stream. Thus, we compute the occupancy-cycle product in the ideal case for SQ as $(N_{stores}/N_{total}) \cdot O^{ROB}_{ideal} \cdot C_{ideal} - l \cdot A(W) \cdot N_{stores}$, where N_{stores} and N_{total} are the number of stores, and total number of instructions, respectively. This assumption may be violated if a majority of the stores are dependent on loads in the same instruction window that miss in the L2 cache or DTLB. This is infrequent in typical workloads. However, it can be modeled if necessary using the IQ occupancy estimation methodology during a load miss in the L2/TLB, and capturing the number of stores in the instruction window that are dependent on the load miss. All other occupancy-cycle products from Equation 1 are multiplied by the fraction of stores to estimate O^{SQ}_{total}. We improve the occupancy estimation by capturing the number of loads and stores in the shadow of a non-overlapped data L2/TLB.

4 Evaluation of the Model

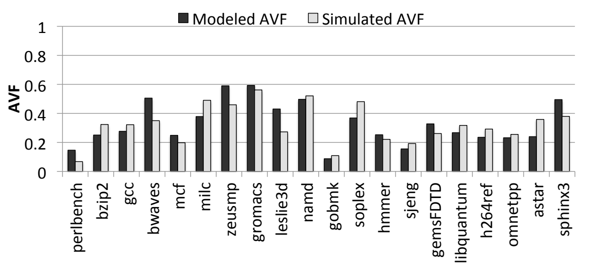
We implement ACE analysis on a modified version of SimpleScalar [15]. We implement detailed, bit-wise ACE analysis in which each entry in the microarchitectural structure of interest has ACE bits fields based on its opcode. For example, stores or branch instructions do not need a result register, and thus the corresponding fields in their ROB entries are un-ACE. We also implement a separate IQ, LQ and SQ in SimpleScalar. We evaluate the accuracy of our model using 20 SPEC CPU2006 workloads (we were unable to compile the remaining workloads for Alpha) using gcc v4.1 compiled with the -O2 flag. We run the profilers and the detailed simulator on single simulation points of length 100 million instructions, identified using the SimPoint methodology [16]. The two configurations evaluated in this Section are presented in Table 1. *Wide-issue machine* and *Narrow-issue machine* represent a 4-wide and 2-wide issue out-of-order superscalar, respectively.

Results

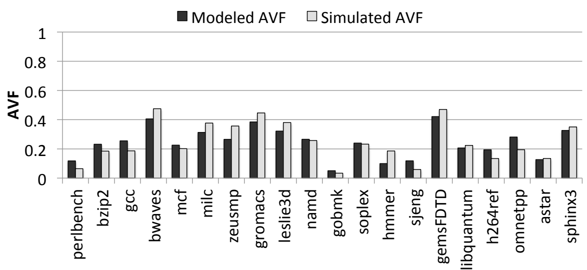
The AVF of the ROB, IQ, LQ, SQ and FUs computed using the model and microarchitectural simulation is pre-



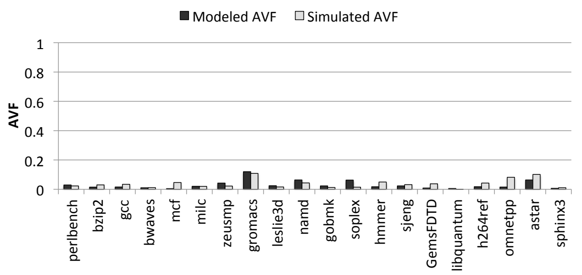
(a) AVF of ROB.



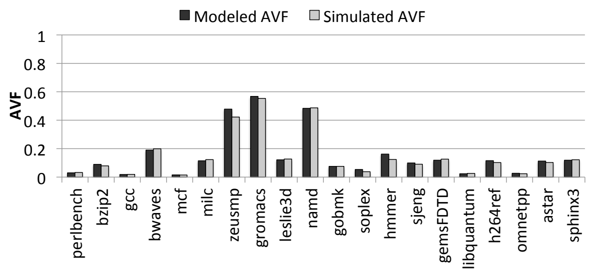
(b) AVF of IQ.



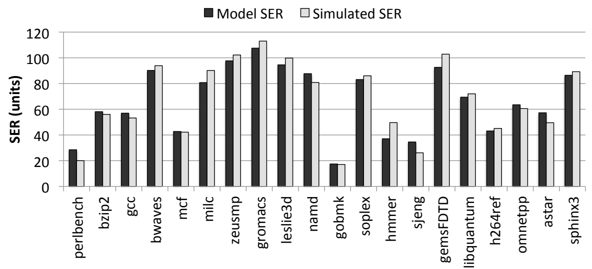
(c) AVF of LQ.



(d) AVF of SQ.



(e) AVF of FU.



(f) Soft Error Rate.

Figure 4. AVF of the Wide-Issue Machine.

sented in Figure 4 and 5 for the wide-issue and narrow-issue machine, respectively. We compute the overall SER for these structures assuming an arbitrary intrinsic fault rate of 0.01 units/bit.

As presented in Table 2, the mean absolute error (MAE), and the maximum absolute error is no larger than 0.07, and 0.16 respectively. As AVF is normalized to the number of bits in the structure (Section 2), it amplifies errors in small structures. For a sense of proportion of the error in computing SER, we express the absolute error in estimating SER in terms of the intrinsic fault rate of each entry in the corresponding structure. In the interest of brevity, in the following discussion, when we express absolute SER error as n entries, we mean "equivalent to the intrinsic fault rate of n entries in the corresponding structure".

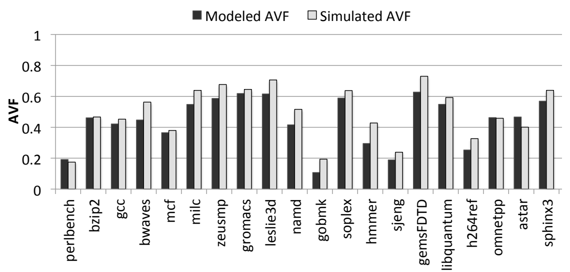
The MAE for estimating SER of the ROB, IQ, LQ, and SQ for the wide-issue machine is 3.8, 4.5, 2.8, and 1.3 entries, respectively. The maximum absolute error for estimating the SER of these structures is 10.2, 9.9, 5.7, and 3.8 entries, respectively. Similarly, the MAE for estimating the SER of the aforementioned structures of the narrow-issue machine is 3.8, 2.1, 1.5, and 0.64 entries, respectively. The

	Wide-issue Machine		Narrow-issue Machine	
	MAE	Max. Abs. Error	MAE	Max. Abs. Error
ROB	0.03	0.08 *(hmmer)*	0.06	0.13 *(hmmer)*
IQ	0.07	0.16 *(bwaves)*	0.07	0.16 *(leslie3d)*
LQ	0.05	0.09 *(zeusmp)*	0.05	0.1 *(gemsFDTD)*
SQ	0.02	0.06 *(omnetpp)*	0.02	0.07 *(milc)*
FU	0.01	0.05*(zeusmp)*	0.02	0.13 *(gromacs)*

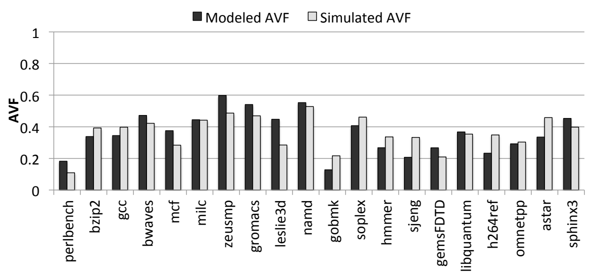
Table 2. Error in Estimating AVF.

maximum absolute error for these structures is 8.3, 5.12, 3.2, and 2.24 entries, respectively.

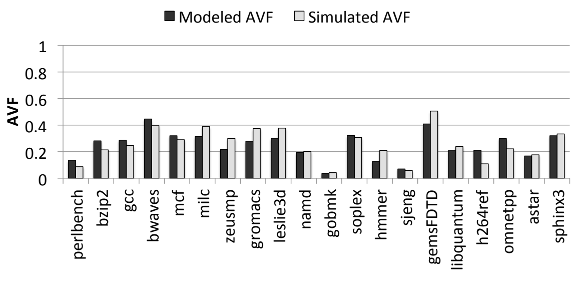
Figures 4(f) and 5(f) present the combined SER for the ROB, IQ, LQ, SQ and FU. Root Mean Square Error (RMSE) is typically used to compute the accuracy of a model, and is computed as $\sqrt{\frac{1}{N}\sum_{i=0}^{N}(m_i - a_i)^2}$, where m_i, a_i and N represent the modeled value, actual value, and total number of workloads, 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 our model on the wide-issue and narrow-issue machine is 9.0%, and 10.3%, respectively.



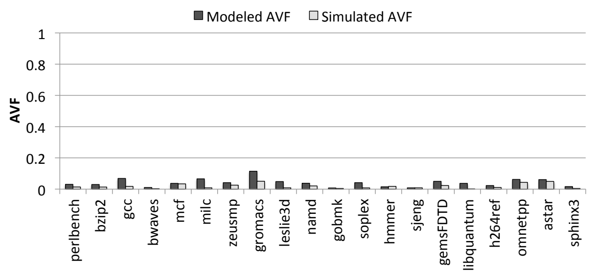
(a) AVF of ROB.



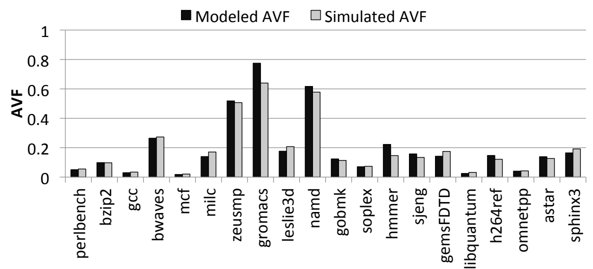
(b) AVF of IQ.



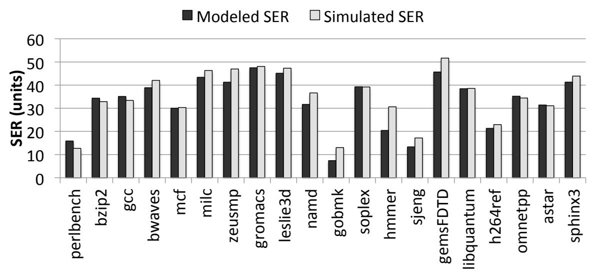
(c) AVF of LQ.



(d) AVF of SQ.



(e) AVF of FU.



(f) Soft Error Rate.

Figure 5. AVF of the Narrow-Issue Machine.

Potential Sources of Error: We multiply the proportion of ACE bits injected in a structure by the program, 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. We find that this is reasonable 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.

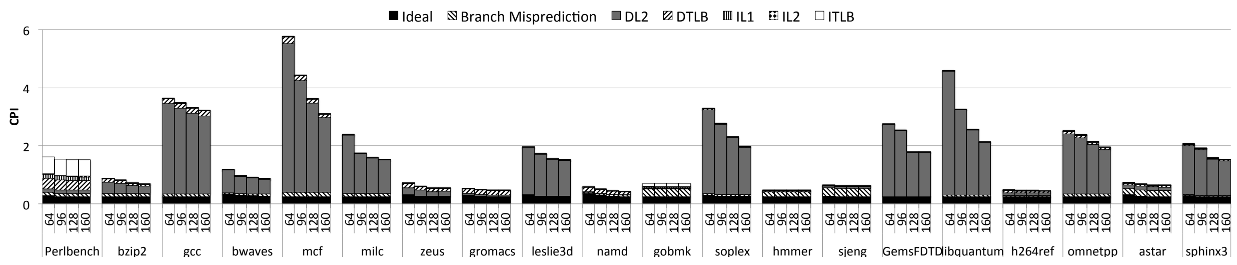
The out-of-order issue of instructions from the IQ causes errors in the estimation of AVF. For example, NOP instructions leave the IQ almost immediately, but are included in the computation of $A(W)$, and the average number of ACE bits induced by the instruction stream. Capturing these effects would require the combination of profiling and ACE analysis. We avoid this approach so that we can gain 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 Applications of the Model

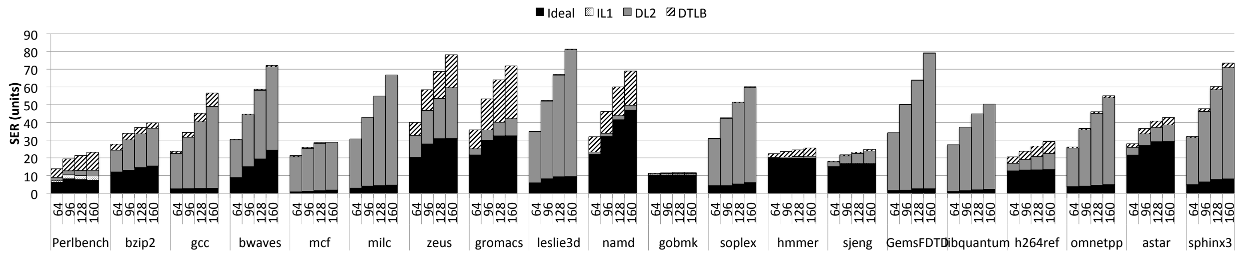
The analytical model can be used to study performance vs. reliability tradeoffs of SER mitigation techniques, the impact of sizing of microarchitectural structures on AVF and performance, compiler optimizations on AVF, different cache sizes and latencies, different branch predictors, etc. In this section, we study a small subset of these potential applications of the model. Specifically, we use the model to explore performance and AVF sensitivity to microarchitecture structure sizing, to drive design space exploration, and to characterize workloads for AVF.

5.1 Impact of Scaling Microarchitectural Parameters

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



(a) Effect of scaling the ROB size on CPI.



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

Figure 6. Effect of Scaling ROB Size on CPI and SER.

size by a small amount provided that it has negligible impact on performance, but significantly reduces SER. Using our model, we can instantaneously determine the impact of scaling a structure on AVF and CPI. In this section, we study the impact of sizing the ROB on AVF and CPI on the wide-issue machine presented in Table 1, assuming an intrinsic fault rate of 0.01 units/bit. We assume that the IQ, LQ, SQ, and other structures are appropriately scaled with the ROB size such that the processor is not unbalanced or constrained.

Figure 6 illustrates the impact of scaling the size of the ROB from 64 to 160 entries, on the wide-issue machine. Note that DL2 and IL2 refer to data and instruction cache misses in the (unified) L2 cache. 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. We present a few examples for, and exceptions to, these mechanisms below.

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 $\frac{\beta}{\beta-1}$ and l/α (Section 3.1). Consequently, from Equation 3, *namd* induces high SER for all ROB sizes despite its low CPI.

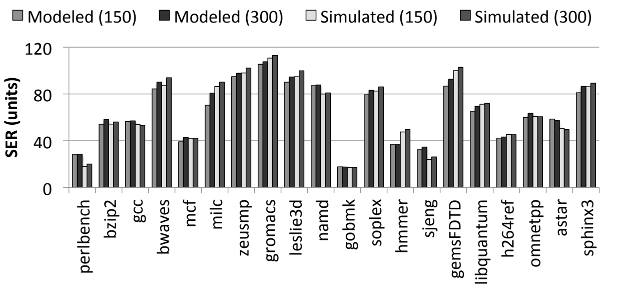
For workloads such as *libquantum*, the increase in ROB size provides increased MLP, resulting in lower CPI, but also a greater SER in the shadow of the L2/DTLB miss. *Libquantum* is able to exploit more MLP than *gemsFDTD* resulting in a greater rate of reduction of CPI, and a lesser rate of increase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

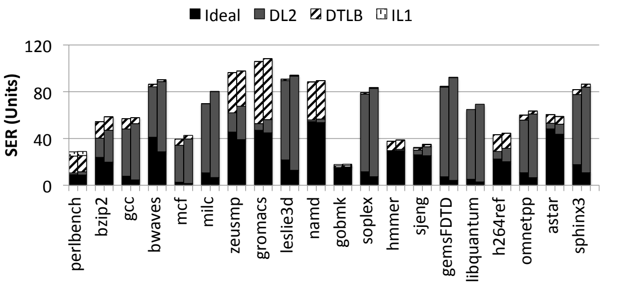
The scaling study allows the architect to make the appropriate trade-offs between performance and SER, and understand the factors affecting the scaling of workloads. For example, the 128-entry ROB provides a speedup of 1.098 (harmonic mean) and increases the average SER by 18% over the 96-entry ROB.

Sensititivity of AVF to Memory Latency: We study 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). Figure 7 presents the overall SER for a memory latency of 150 cycles, and 300 cycles, obtained using our model, and from detailed simulation, assuming a constant intrinsic fault rate[3] of 0.01 units/bit. The memory latency used by

[3]Although the intrinsic fault rate will significantly increase at low voltages, a constant value allows us to highlight the change due to AVF.



(a) Sensitivity of SER to memory latency.



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

Figure 7. Sensitivity of SER to Memory Latency.

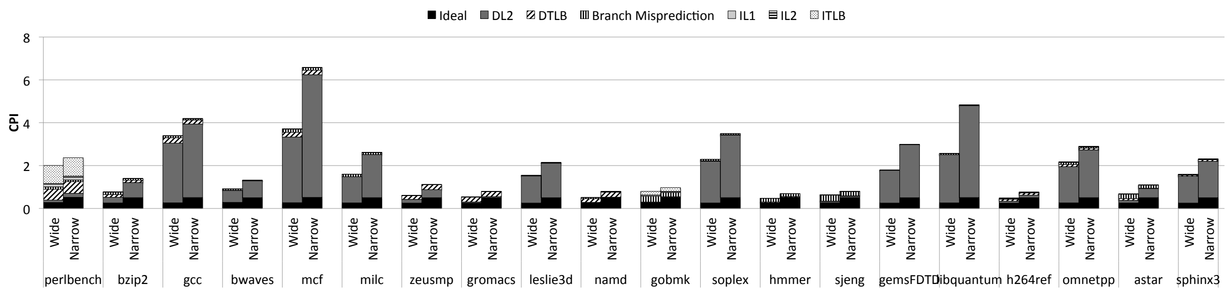
the model and simulation is enclosed in parentheses. From the formula for AVF (Section 2), we observe that reduction in memory latency reduces the total number of cycles of ACE bit residency, and the total number of execution cycles. Consequently, the change in AVF in Figure 7 is sub-linear, and thus, less sensitive to memory latency when compared with CPI. AVF typically decreases with a decrease in memory latency, although it may increase, as seen with *astar*. The comparison with simulation also serves to validate our model. The average change in AVF as predicted by the model is 3.25 units, as compared to 2.22 units from simulation. The model faithfully captures the trend for change in SER. Figure 7(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 significantly reduces for workloads dominated by L2 cache misses. Conversely, the relative contribution from ideal execution and DTLB miss increases. The workload completes faster due to lower memory latency, and consequently, the contribution of steady-state and DTLB misses to the overall occupancy increases.

5.2 Design Space Exploration

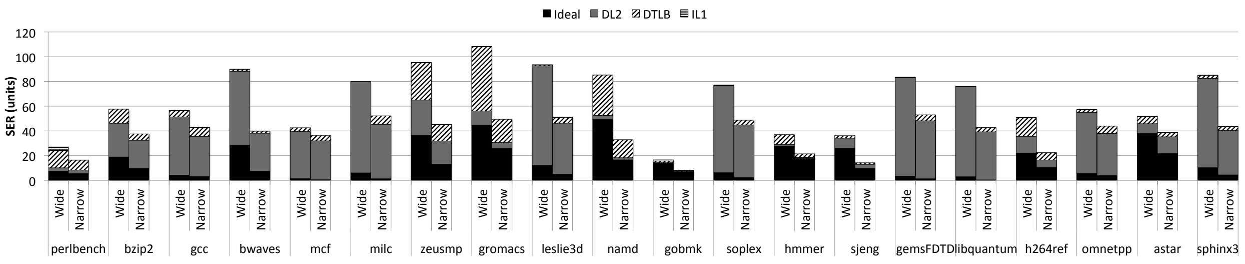
The model can be used to compare different microarchitectures for their impact on performance and AVF/SER. Figure 8 presents the CPI and SER of the wide-issue and narrow-issue machine outlined in Table 1. The SER is computed for the ROB, LQ, SQ, IQ and FUs, and is broken down into its contributing events, 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 5.1), increasing the dispatch width typically increases the SER across all workloads. From Equation 3, a larger instruction window is required to sustain a larger dispatch width. For our workloads, 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 5.1, *namd* and *bwaves* have a long critical path $K(W)$. These 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.

The model can also be used to provide insight into the efficacy of soft-error mitigation schemes. Gomaa et al. [17] propose an opportunistic mechanism, called Partial Explicit Redundancy (PER), of enabling Redundant Multi-Threading (RMT) [18] 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. Using detailed simulation for a specific microarchitecture running SPEC CPU2000 workloads, Sridharan et al. [19] investigate the efficacy of PER 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.

Under an optimistic assumption of no performance loss using the opportunistic scheme, the components of SER in Figure 8(b) corresponding to data L2 and TLB misses would disappear. Whereas this scheme generally reduces the AVF of most workloads significantly (we compute an SER reduction of 66% for the wide-issue machine), *namd* would still have high AVF. Furthermore, given that it incurs few miss events (Figure 8(a)), the performance of *namd* will be significantly impacted if RMT is enabled. Of course, these results are microarchitecture and workload specific. For example, we compute that PER results in an average SER reduction of 60% when the memory latency of the wide-issue machine is reduced to 150 cycles, as illustrated



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



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

Figure 8. Comparison of CPI and SER of the wide- and narrow-issue machines.

in Figure 7(b). We reach conclusions similar to earlier work. Thus, the model enables architects to estimate the efficacy of such a scheme in the first order, for their microarchitecture and workloads.

5.3 Workload Characterization for AVF

It is difficult to infer 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 noted in earlier sections, there may be exceptions to our intuitions of occupancy of state. Given a microarchitecture such as the wide-issue machine, the model enables us to identify *namd* as a high-IPC workload inducing high AVF, and *gobmk* as a comparably high-IPC workload that induces very low AVF in multiple structures. Our model uncovers the complex relationship between various microarchitectural events that combine to induce AVF in a structure, thereby enabling an intuitive understanding of their influence. 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 phases in the workload that induce high AVF in particular structures, enabling better workload characterization for AVF.

6 Related Work

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. Computing 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 provides limited 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 [20] 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) [21], which in turn requires detailed simulation, and thus provides less insight than our model. Sridharan and Kaeli have shown that HVF correlates with occupancy of structures such as the ROB, and hence we expect that our modeling methodology can be used to model HVF of the applicable structures.

Fu et al. [7] 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. [10] and Duan et al. [8] 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 simulation. Cho et al. [9] 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 our approach. As we derive the factors affecting AVF from first principles, we can identify the precise cause of high or low AVF in a particular structure, and characterize workloads for AVF.

7 Conclusion

In this work, we developed a first-order mechanistic model for AVF, derived from first principles of out-of-order processor execution, to provide quantifiable insight into the factors affecting the AVF of structures. The modeling methodology requires inexpensive profiling, and computes AVF with mean absolute error of less than 0.07, for the ROB, LQ, SQ, IQ and FU. Additionally, the model quantifies the impact of each microarchitectural event on AVF and SER. We have demonstrated that the model can be used for understanding how microarchitecture affects AVF. The model was used to perform design space exploration, to evaluate the impact of parametric changes, and to perform workload characterization for AVF. By modeling the complex relationship between various miss events that affect the occupancy of state in the processor, we are able to quantitatively explain the lack of correlation between AVF and aggregate metrics observed in earlier work. Finally, this work enables the architect to identify workloads that would induce high AVF in microarchitectural structures.

8 Acknowledgment

We are grateful to Arijit Biswas (Intel) for his guidance. We thank Jian Chen, Faisal Iqbal, Mattan Erez, and all reviewers for their comments and suggestions. Lizy John and Arun Nair are partially supported by NSF grant 1117895 and AMD. Stijn Eyerman is a postdoctoral fellow with the Fund for Scientific Research - Flanders (FWO). Lieven Eeckhout is supported by the FWO projects G.0255.08 and G.0179.10, and the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC Grant agreement no. 259295. Any opinions, findings, conclusions, or recommendations do not necessarily reflect the views of these funding agencies.

References

[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 *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, pp. 29–40, Dec. 2003.

[3] S. Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," *IEEE Micro*, vol. 25, pp. 10–16, nov.-dec. 2005.

[4] 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*, pp. 389–398, 2002.

[5] 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 *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, pp. 61–70, June-July 2004.

[6] X. Li, S. Adve, P. Bose, and J. Rivers, "SoftArch: An Architecture Level Tool for Modeling and Analyzing Soft Errors," in *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, pp. 496–505, June-July 2005.

[7] 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*, pp. 147 – 155, September 2006.

[8] L. Duan, B. Li, and L. Peng, "Versatile prediction and fast estimation of Architectural Vulnerability Factor from processor performance metrics," in *IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 129–140, February 2009.

[9] 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*, pp. 274 –285, December 2007.

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

[11] 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.

[12] 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.

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

[14] 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*, pp. 402–411, June 2007.

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

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

[17] M. Gomaa and T. Vijaykumar, "Opportunistic transient-fault detection," in *Proceedings of 32nd International Symposium on Computer Architecture*, pp. 172–183, June 2005.

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

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

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

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