AVF Stressmark: Towards an Automated Methodology for Bounding the Worst-case Vulnerability to Soft Errors

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

***Abstract*—Soft error reliability is increasingly becoming a first-order design concern for microprocessors, as a result of higher transistor counts, shrinking device geometries and lowering of operating voltages. It is important for designers to be able to validate whether the Soft Error Rate (SER) targets of their design have been met, and help end users select the processor best suited to their reliability goals. The knowledge of the observable worst-case SER allows designers to select their design point, and bound the worst-case vulnerability at that design point. We highlight the lack of a methodology for evaluation of the overall observable worst-case SER. Hence, there is a clear need for a so called stressmark that can demonstrably approach the observable worst-case SER. The worst-case thus obtained can be used to identify reliability bottlenecks, validate safety margins used for reliability design and identify inadequacies in benchmark suites used to evaluate SER. Starting from a comprehensive study about how microarchitecture-dependent program characteristics affect soft errors, we derive the insights needed to develop an automated and flexible methodology for generating a stressmark that approaches the maximum SER of an out-of-order processor. We demonstrate how our methodology enables architects to quantify the impact of SER-mitigation mechanisms on the worst-case SER of the processor. The stressmark achieves 1.4× higher SER in the core, 2.5× higher SER in DL1 and DTLB, and 1.5× higher SER in L2 as compared to the highest SER induced by SPEC CPU2006 and MiBench programs.**

I. 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], [2]. Borkar [3] states that the soft-error failure rate at 16 nm is expected to be 100 times higher than at 180 nm.

Additional hardware such as radiation-hardened circuitry or error detection/recovery mechanisms may be required in order to meet the target SER requirements. This additional hardware has implications on area, power and performance, and hence the design-point needs to be carefully selected, to avoid over-designing or under-designing the processor. In this work, we develop a methodology that can be used by architects to estimate the observable worst-case SER on the microarchitecture. The knowledge of this observable worst-case SER allows architects to pick the appropriate design point from an SER perspective for their microarchitecture, and also validate the SER coverage of their workload suite. Our methodology adapts to different microarchitectures and underlying circuit-level fault-rates in an automated fashion in order to produce an AVF stressmark, such that the observable SER will approach the maximum.

Issues affecting SER benchmarking: Prior research [2], [4] 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-rates 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 select benchmarks such that they cover 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. The choice of this safety margin is largely based on designer intuition, and it is difficult to know whether it is adequate. Figure 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-

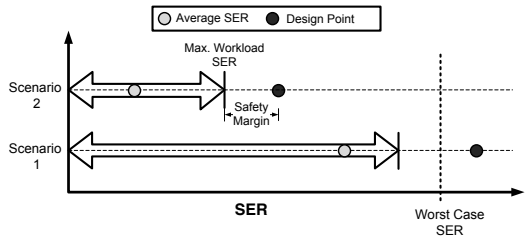


Figure 1. Choice of the design point, under different SER coverage scenarios

induced SER. As shown in Figure 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 indequate. On similar lines, the architect may choose to design for the average-case workload. 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.

Difficulties in determining the worst-case SER: We note that 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. For example, the branch recovery checkpoint structure is not accessed if the program does not experience branch mispredictions. On the other hand, branch mispredictions reduce the AVF of structures in the rest of core. 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 calculated using AVF modeling.

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

We propose a comprehensive methodology that simultaneously increases the AVF of multiple structures in the processor such that the observable SER approaches the maximum. The search space for such a program is large and complex. Starting from first principles, we derive a set of microarchitecture dependent factors that affect the occupancy of useful state in the processor, and use these insights to develop a code generator that defines a feasible search space. We then use a Genetic Algorithm (GA) to explore this search space to to generate the stressmark.

The significant contributions of this work are as follows:

1) We develop a flexible and automated methodology to generate an AVF stressmark. This AVF stressmark is designed to approach the maximum observable SER for a given microarchitecture.
2) We highlight deficiencies in current methodologies for the estimation of the observable worst-case SER. We also highlight the potential 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.

Paper Outline: The remainder of this paper is organized as follows: Section II provides a background on ACE analysis. Section III highlights the interdependence of occupancy and hence AVF of structures in an OoO processor, which ensures that the AVF of all structures cannot be 100% simultaneously. This observation motivates our research to create a stressmark that induces the observable worst-case SER. We outline the methodology for creating a code generator for the AVF stressmark in Section IV, derived from a comprehensive study on microarchitecture-dependent characteristics that influence occupancy of structures. Section V outlines the framework and evaluation methodology for generating an AVF stressmark using a Genetic Algorithm (GA). We discuss results in Section VI. Section VII includes a discussion on how architects can use our methodology to identify reliability bottlenecks, and quantitatively measure the impact of their SER mitigation mechanisms on the worst-case SER.

II. Background

Mukherjee et al. [2] define Architectural Vulnerability Factor (AVF) as a measure of the probability that a radiation-induced fault in a structure will be visible in the program output. Mukherjee et al. formally define AVF of

Table I
BASELINE CONFIGURATION OF PROCESSOR

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

a structure of size N bits, as: $AVF_{structure} = \frac{1}{N} \times (\sum_{i=0}^{N}(\frac{ACE\ cycles\ for\ bit\ i}{Total\ Cycles})_i)$. AVF is the derating factor on the raw fault rate of the underlying circuit, and captures the masking effect of program execution on soft errors. This derated failure rate is added for all structures on the chip, to derive its overall SER.

Conversely, Mukherjee et al. [2] term bits that are not critical to program correctness as *un-ACE*. These include bits such as those in unused or invalid state, bits discarded as a result of mis-speculation, or bits in predictor structures, and bits corresponding to instructions such as NOPs, software prefetches, predicated false instructions, and dynamically dead instructions. Butts et al. [5] note that 3 - 16% of instructions are dynamically dead. Since the values of these instructions do not affect the output of the program, their correctness is not critical.

Whether a bit is ACE or not in on-chip cache structures depends on the nature of reads and writes to that structure. Biswas et al. [6] 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 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.

In this work, we use ACE analysis to measure the AVF and overall SER of the processor. We define worst-case SER as the highest SER calculated using ACE analysis (AVF+ Sum of Failure Rates), assuming a fixed circuit-level fault rate, i.e., all changes in SER are due to changes in AVF alone.

III. INTERDEPENDENCE OF 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. We use the example of the Alpha 21264 microarchitecture to illustrate this point. However, the idea is true of any other microarchitecture.

Consider the Alpha 21264 whose configuration is outlined in Table I. 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, we see that 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 up, and hence the period of time for which the LQ and SQ can have high AVF, in the shadow of an L2 miss.

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. We therefore need a methodology that addresses the issue of quantifying the observable worst-case SER.

IV. CODE GENERATOR FOR THE AVF STRESSMARK

In this section, we describe the methodology used to build a code generator for AVF stressmarks. 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 2 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 AVF. In the next step, the output of the simulator is evaluated by a fitness function, which

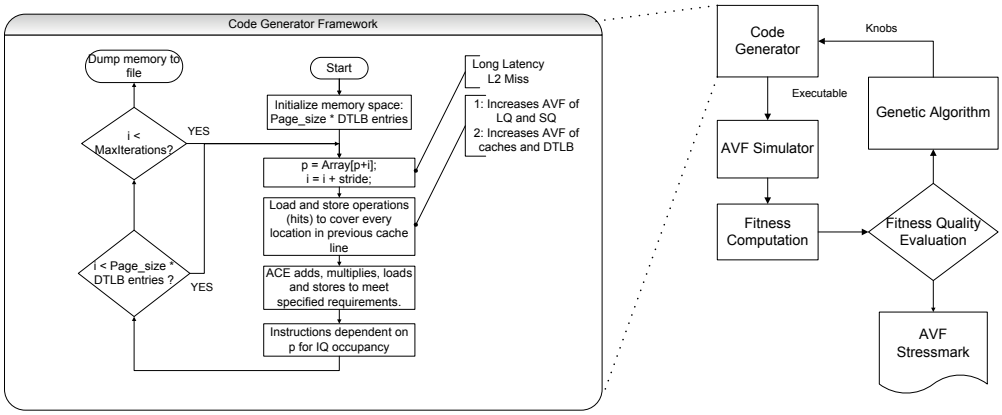


Figure 2. Methodology for creation of an AVF stressmark.

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, we study the microarchitecture-dependent program characteristics that affect occupancy in cores and caches.

We classify microarchitectural structures into Queueing Structures (QS) and storage structures. For queueing structures such as the IQ, LQ, SQ, ROB and FU, 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 [7], [8]. 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 [6] and coverage of cache locations.

A. AVF due to Microarchitecture-Dependent Behavior

We now look at the factors that affect the overall occupancy of queuing structures, and liveness of caches. For the discussion below, we assume that the instruction stream has a constant proportion of ACE bits. We use this analysis to determine the factors that are required to be controlled in a code generator that increases AVF in a core.

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. 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.

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).

3) Instruction Mix: As noted in Section III, dynamic instructions get 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]. Since 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) 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.

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 [6]. 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, we should eliminate all un-ACE instructions.

We thus note that occupancy, and hence AVF 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. We use the above insights to derive a code generator.

B. Design of the Code Generator

We use the insights outlined in Section IV-A to derive the knobs for the code generator. 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). We implement a code generator based on the framework outlined in Figure 2. The code generator must be provided with the size of the ROB, and the caches, of the particular microarchitecture. We utilize a strided load in the inner loop, that will miss in the L2 cache, and is dependent on itself (pointer chasing). This avoids any Memory-Level Parallelism for the L2 misses. Ideally, we expect that 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. We allow the code generator to determine the size of the loop, but restrict its maximum size to $1.2\times$ the size of the ROB. We separately implement another code generator framework 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 2) with ACE instructions as specified using parameterizable knobs derived from the characteristics summarized under section IV-A, below:

1) *I-mix*: We specify the fraction of loads, stores and arithmetic instructions. 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 [9], [10]. 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. We implement this 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. We also ensure that stored results are 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 our 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 our 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. By deriving the properties that affect AVF from first principles, we restrict the search space by disallowing infeasible solutions, and allow a quick generation of a high-AVF stressmark.

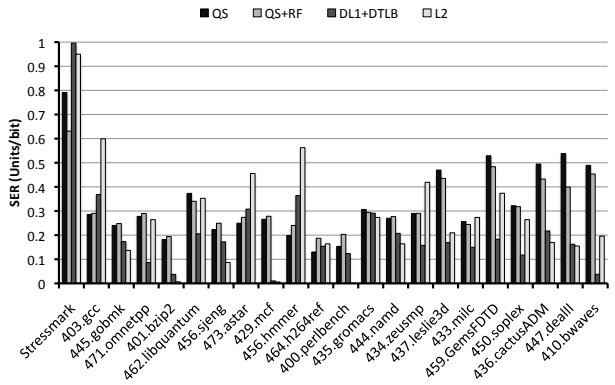


Figure 3. Comparison between the overall SER induced by the Stressmark and CPU2006 workloads on the core and caches for the Baseline Configuration

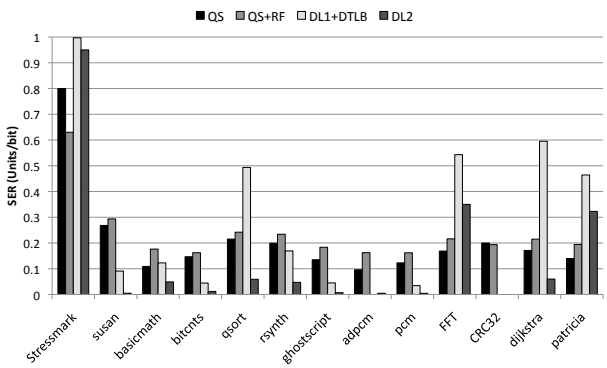


Figure 4. Comparison between the overall SER induced by the Stressmark and MiBench workloads on the core and caches for the Baseline Processor Configuration.

V. 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 our discussion in Section III, the task of creating the optimal instruction schedule that satisfies the constraints of a microarchitecture, while simultaneously increasing SER is non-trivial. We therefore explore the search space defined above using a Genetic Algorithm (GA). GA is an 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. We use IBM SNAP genetic algorithm framework, obtained under NDA for university research, to create the knob values for the code generator, as outlined in Figure 2. The output of the code generator is compiled and run on our 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.

Evaluation Methodology: We evaluate our methodology on our modified version of SimSoda [11], which computes AVF using the ACE analysis methodology proposed by Mukherjee et al. [2] and Biswas et al. [6]. Simsoda is based on SimAlpha [12], which models an Alpha 21264 (EV6) in great detail. SimAlpha models the Integer IQ and Floating Point IQ as separate structures. Our experiments concentrate on the integer pipeline, for parity with SPEC CPU2006 integer results. Our methodology, however, is general enough to be trivially extended to include the FP pipeline. In Figure 2, the GA generates knobs that are provided as inputs to the code generator. The code generator produces the corresponding output, and run on the SimSoda simulator.

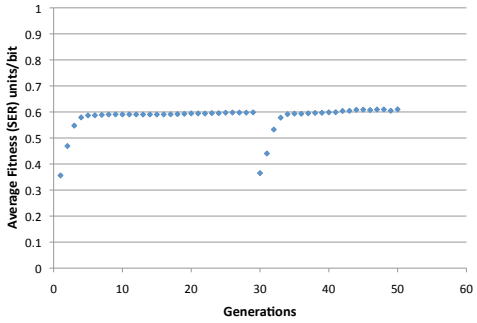
We allow the Genetic Algorithm (GA) to run for 50 generations, with 50 individuals per generation (a total of 2,500 runs), and the best result is is picked as the stressmark. The stressmark is executed for 100M instructions. We compare the stressmark with 11 CPU2006 Integer Workloads and 10 CPU2006 FP workloads. We were unable to compile the entire CPU2006 suite due to compiler issues. We identify a single simulation point of length 100M instructions using the SimPoint methodology [13], and perform a detailed simulation at this simulation point. We also compare our stressmark results with 12 MiBench [14] programs, for diversity of workloads in our workload suite. The stressmark and all the benchmarks were compiled using gcc version 4.1 with the -O2 flag. We assign the probability of mutation as 0.05 and a crossover rate of 0.73 in the GA, based on recommended ranges from literature, such as Grefenstette [15], and Srinivas and Patnaik [16].

VI. Results

Figure 3 and Figure 4 represent the overall SER of the architecture specified in Table I, which we call the Baseline Configuration. We assume that the raw fault rate of the underlying circuits is 1 unit/bit. This is an arbitrary unit, since only the relative magnitude is of importance for our

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

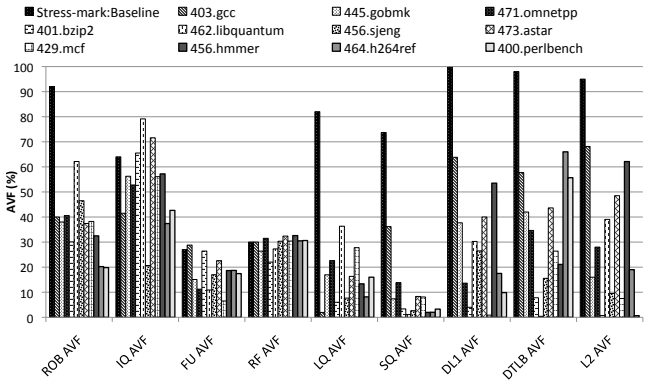


(b) Convergence of GA

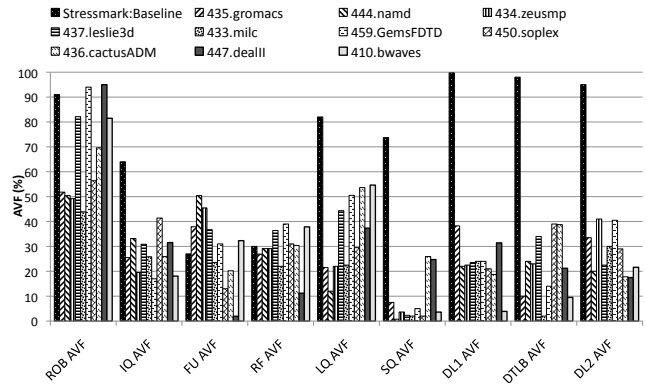
Figure 5. Stressmark generated by the Genetic Algorithm for the Baseline Processor Configuration.

methodology. We present the SER of Queuing Structures (QS), Queuing Structures and the Register File (QS+RF), DL1+DTLB, and L2 separately, since caches have significantly more bits than the core, and would dominate all SER computation. We normalize the SER values reported by dividing them by the total number of bits in that class of structure, in the interest of clarity. For example, we divide the SER computed for the Queuing Structures by the total number of bits in them.

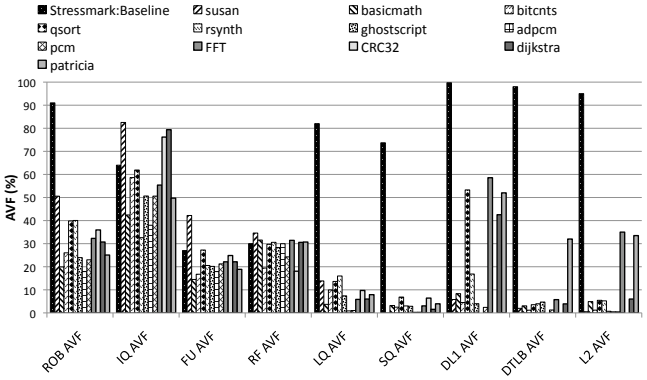
Analysis: Figure 5(a) shows the final parameters generated by the GA as the optimal solution. The generated code utilizes every architected register, thereby maintaining high ACE in the Architected RF, by utilizing the appropriate number of reg-reg instructions. The GA selects short dependence chains to control ILP and hence occupancy of IQ, and a loop size almost equal to the size of the ROB. Figure 5(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 from generation 29 is moved into a new population of random mutations, and the process is repeated. We see that, at the end of 50 generations (2,500 runs), the GA has converged. We execute 6 runs in parallel to speed up the process. The overall execution time for creating this stressmark is roughly 48 hours.



(a) AVF of SPEC CPU2006 Integer Workloads



(b) AVF of SPEC CPU2006 FP Workloads

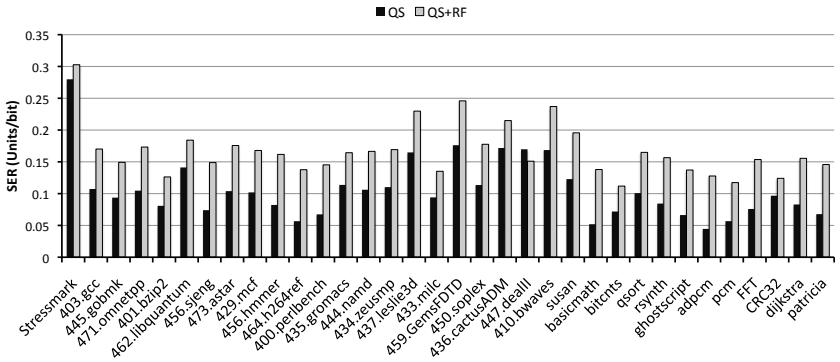


(c) AVF of MiBench Workloads

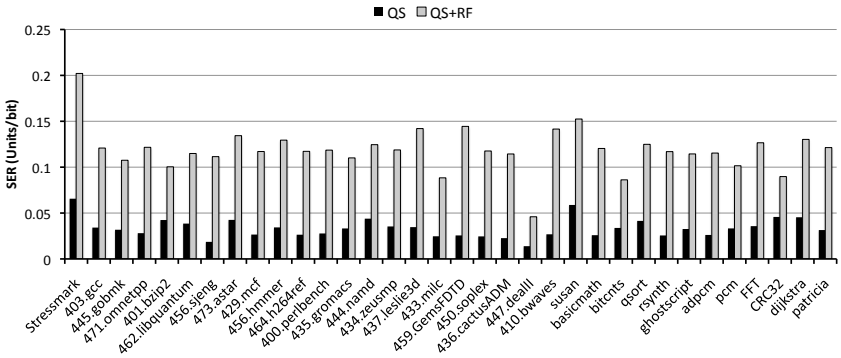
Figure 6. AVF of queuing and storage structures for SPEC CPU2006 and MiBench workloads on the Baseline Processor Configuration.

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* has the highest overall AVF (core+cache) of all the workloads in Figure 3 and Figure 4. Compared to this, we have over 2× higher SER in QS+RF, and DL1+DTLB, and around 1.5× higher SER in L2.

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, we note that the SER of queuing structures in



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



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

Figure 7. SER induced on Processor Configurations RHC and EDR, by workloads from SPEC CPU2006 and MiBench.

SPEC CPU2006 FP workloads is relatively high, compared to SPEC CPU2006 integer workloads. Our 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%. We calculate the instantaneous worst-case occupancy for queuing structures, in the shadow of an L2 miss as 0.899 units/bit (as compared to 0.797 units/bit for the stressmark). Since 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. We thus find that our stressmark achieves AVF that is close to the theoretical and unsustainable maximum. It is impossible to positively prove that our stressmark induces the absolute, sustainable maximum SER (A problem shared with power and thermal viruses). It is for this reason that we leverage the ability of the GA to optimize for such a complex solution space. 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 gives us confidence that the SER induced by the stressmark is very near the maximum.

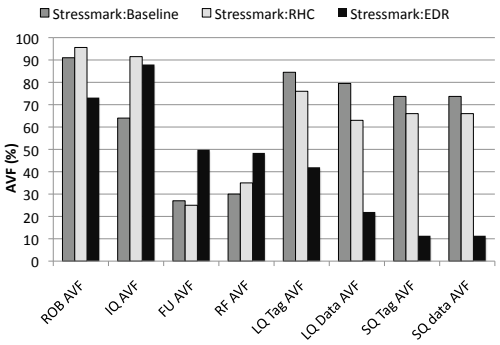
Figures 6 presents the AVF of SPEC CPU2006 and MiBench benchmarks on our baseline configuration, on individual structures. In contrast with SPEC CPU2006, our AVF stressmark for this microarchitecture achieves much higher AVF on all the structures, with the exception of FUs and in some cases, RF.

A. 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, we generate stressmarks for two configurations for the same microarchitecture in Table I but different underlying fault-rates outlined in Figure 8(a). We assume unchanged fault rates in DL1, DTLB and L2. We 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

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



(b) AVF of queuing 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

(c) 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

(d) Knob Setting for Config EDR

Figure 8. Results of AVF Stressmark Methodology on different circuit-level fault rates.

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 our methodology.

Configuration RHC: In the case of Config RHC, the IQ and RF are more vulnerable than the ROB, LQ and SQ. Our 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 8(b). Comparing Figure 8(c) to Figure 5(a), we see 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 7(a) presents the SER of the core of SPEC CPU2006 and MiBench programs. We find the AVF stressmark induces a significantly higher SER than any SPEC CPU2006 or MiBench programs.

Configuration EDR: Since 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, since we need to maintain AVF of the caches. Since 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 8(b) 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 7(b) presents the SER induced by our 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. We thus demonstrate the the code-generator and GA methodology is flexible enough to adapt to such that overall error rate is increased.

B. Stressmark generation for a different microarchitecture

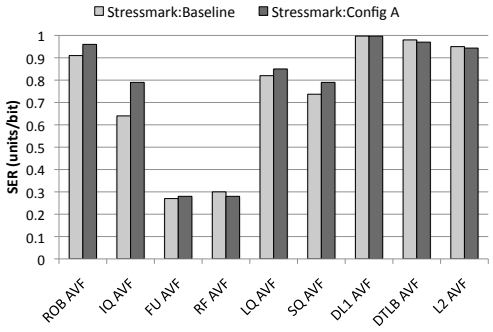
For completeness, we create Stressmark:Config A for a 4-issue OoO processor with a larger IQ, ROB and rename register file in the core, and a larger DTLB and L2 cache and latency (Configuration A), outlined in Table II. Figure 9 details the overall SER of Configurations Baseline and A, in which all structures are assumed to have the same circuit-level fault rate of 1 unit/bit. Further, we assume that the sizes of each entry in the Queuing Structures of Config A 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 dependent 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. We thus demonstrate that our methodology is flexible enough to automatically adapt to different microarchitectures.

VII. 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. We will restrict the following discussion to the core (Queueing structures + Register File), since we clearly achieve 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 III. 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 our technique is its adaptiveness. When the circuit-level fault rate of one or more structures are reduced,

Table II
ALTERNATE CONFIGURATION FOR EVALUATING THE STRESSMARK CREATION METHODOLOGY

Parameter	Configuration A
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



(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 9. AVF of queueing and storage structures for Configuration A.

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.

Comparison with Other Possible Methodologies: In the absence of an AVF stressmark, it is impossible to know whether our set of 33 workloads offers sufficient AVF coverage. Table III shows the worst SER observed on our set of workloads. Our 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 III
COMPARISON OF WORST-CASE SER ESTIMATION METHODOLOGIES IN THE CORE USING SPEC CPU2006 AND MIBENCH

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 III 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, our stressmark induces higher AVF. This is not necessarily the case, since one could write programs that drive individual structures to 100% 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 over-estimation, and will lead to an extremely pessimistic design. This, in turn, will impact performance, power and design effort of the processor.

Utilizing the Stressmark Methodology: The knowledge of the observable worst-case SER allows us to evaluate the robustness of the SER evaluation workload suite in use. In our case, the worst-case SER induced by an individual program in our 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: Our 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, our 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 pro-

tected 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 ours 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. We believe that our work is a significant first step towards defining a methodology for SER benchmarking and evaluation.

VIII. Related Work

There has been some prior work on attempting to increase the visibility of radiation induced faults at the program output. Kellington et al. [17] and Sanda et al. [18] 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, we are unaware of the extent to which it exercises the core, but we expect that completely random injection of instructions, even if they were all ACE, would likely not maximize the corruptable state resident in the processor. It would be unlikely for it to arrive at the characteristics outlined in our work, randomly. A machine learning technique can be incorporated, but we expect that our solution still helps the GA converge much faster and with more confidence in the result. Circuit level techniques have been proposed, such as work by Sanyal et al [19], [20], [21]. However, this does not consider the masking effect of program execution, and cannot be used during the early design stage.

Recent work by Sridharan et al. [22] introduces the concept of *Hardware Vulnerability Factor* (HVF), which can be used to bound the AVF on a structure while running a program. They show that HVF correlates strongly with occupancy, although not exactly equivalent. Whereas HVF may be used to estimate the highest possible SER while running a workload suite, it still cannot be used to determine the worst-case observable SER due to its dependence on workloads, and their coverage. It thus shares the same limitation of ACE analysis, for determining the observable worst-case SER. Sridharan et al. [23] use a microarchitecture independent metric *Program Vulnerability Factor* (PVF) to study the AVF of register files. They use PVF to study the effect of compiler flags on vulnerability of the Architected Register File, and report inconsistent effects of the compiler on AVF of the register file. Sridharan et al [24] study the potential benefit of techniques that rely on mitigating AVF in the shadow of long-latency stalls, such as those presented in [25], [26], and report that almost 60% of the AVF of queueing structures is accounted for in the shadow of a long latency miss.

Joshi et al. [27] and Ganesan et al. [28] 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. Our 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.

Although we use ACE analysis to develop our stressmark, other methodologies such as SoftArch [29] can be used. However, SoftArch cannot be used to determine the worst-case SER without our methodology, and is not a substitute for this work. Li et al. [30] report that ACE analysis (AVF+Sum of Failure Rates) may not accurately estimate MTTF in the presence of a large number of processors or very high fault rates and long running workloads. This does not affect our work, since our objective is to increase the overall susceptible state in the processor, and not measure actual MTTF. ACE analysis is still a useful tool in guaging the susceptibility of a processor to soft-errors.

IX. Conclusion

In this work, we highlight the lack of a deterministic methodology for evaluating the highest observable SER. We demonstrate that methodologies that ignore the interactions between structures within a processor may incur significant errors while estimating the highest SER under program influence. We therefore propose an automated and flexible methodology, derived from a comprehensive study of interactions between structures in an OoO processor, that generates an stressmark that approaches the maximum SER observable while running a program. We demonstrate how our methodology can enable architects to make quantifiable decisions regarding the effect of various SER mitigation mechanisms on overall highest SER. This knowledge enables better informed trade-offs between performance, power, area and SER reliability. 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.

Acknowledgment

We thank Arijit Biswas for his invaluable comments and discussions towards improving the paper. We also thank Shubhendu S. Mukherjee for the discussions that led to this work. We also acknowledge Xin Fu and Tao Li for providing the SimSoda simulator, and all the reviewers of this work for their comments and suggestions. We are grateful to Jason

F. Cantin for facilitating our use of the IBM SNAP GA framework. Lizy John's research is supported by National Science Foundation under grant number 0702694. Lieven Eeckhout is supported in part through the FWO projects G.0232.06, G.0255.08, and G.0179.10, and the UGent-BOF projects 01J14407 and 01Z04109. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, 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*. Washington, DC, USA: IEEE Computer Society, 2003, p. 29.

[3] 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.

[4] 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*. Washington, DC, USA: IEEE Computer Society, 2004, p. 61.

[5] 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*. New York, NY, USA: ACM, 2002, pp. 199–210.

[6] 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, 2005,* , June 2005, pp. 532–543.

[7] 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,* , Feb. 2009, pp. 129–140.

[8] 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*. New York, NY, USA: ACM, 2007, pp. 516–527.

[9] P. Dubey, I. Adams, G.B., and M. Flynn, "Instruction window size trade-offs and characterization of program parallelism," *IEEE Transactions on Computers*, vol. 43, no. 4, pp. 431 – 442, apr 1994.

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

[11] 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.

[12] 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.

[13] 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*. New York, NY, USA: ACM, 2002, pp. 45–57.

[14] 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*, 2 2001, pp. 3 – 14.

[15] J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 16, no. 1, pp. 122 –128, jan. 1986.

[16] M. Srinivas and L. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 4, pp. 656 –667, apr. 1994.

[17] 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*, 2007.

[18] 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 J. Res. Dev.*, vol. 52, no. 3, pp. 275–284, 2008.

[19] 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.* , March 2007, pp. 723–728.

[20] 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*, July 2007, pp. 191–193.

[21] 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, no. 4, pp. 596–600, April 2009.

[22] 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*. New York, NY, USA: ACM, 2010, pp. 461–472.

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

[24] 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*, 2007.

[25] C. T. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt, "Reducing the Soft-Error Rate of a High-Performance Microprocessor," *IEEE Micro*, vol. 24, no. 6, pp. 30–37, 2004.

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

[27] 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)*, 2008, pp. 229–239.

[28] K. Ganesan, J. Jo, L. W. 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 *Nineteenth International Conference on International Conference on Parallel Architectures and Compilation Techniques (PACT), Vienna, Austria*, 2010.

[29] 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*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 496–505.

[30] 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*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 266–275.