Workload-driven selective hardening of control state elements in modern microprocessors

Michail Maniatakos
EE Department
Yale University
michail.maniatakos@yale.edu

Yiorgos Makris
EE & CS Departments
Yale University
yiorgos.makris@yale.edu

***Abstract*—We present a method for selective hardening of control state elements against soft errors in modern microprocessors. In order to effectively allocate resources, our method seeks to rank the control state elements based on their susceptibility, taking into account the high degree of architectural masking inherent in modern microprocessors. The novelty of our method lies in the way this ranking is computed. Unlike methods that compute the architectural vulnerability of registers based on high-level simulations on performance models, our method operates at the Register Transfer (RT-) Level and is, therefore, more accurate. In contrast to previous RT-Level methods, however, it does not rely on extensive transient fault injection campaigns and lengthy executions of workloads to completion, which may make such analysis prohibitive. Instead, it monitors the behavior of key global microprocessor signals in response to a progressive stuck-at fault injection method during partial workload execution. Experimentation with the Scheduler module of an Alpha-like microprocessor corroborates that our method generates a near-optimal ranking, yet is several orders of magnitude faster.**

I. Introduction

The increasing threat of soft errors in nanometer technologies has resulted in a plethora of design solutions for protecting latches from Single Event Upsets (SEUs) [1], [2], as well as combinational logic from Single Event Transients (SETs) [3], [4]. Despite the demonstrated effectiveness of these solutions, applying them blindly across an entire design incurs prohibitive cost. As a result, various methods for assessing the susceptibility of individual latches or logic gates have also been proposed [5], [6], in order to support partial hardening approaches [7], [8], [9], [10], [11]. Susceptibility evaluation and the corresponding ranking of latches or logic gates typically takes into account a number of factors, including electrical device characteristics, timing issues, as well the actual logic function implemented. These factors reflect the circuit-level and gate-level reasons that may prevent an SEU or an SET from causing a soft error in a circuit. However, they are unable to capture error masking causes at higher levels and, therefore, they prove rather insufficient when applied to modern microprocessors.

Modern microprocessors exhibit a high degree of architectural-level and application-level masking, resulting in many errors being suppressed or having a low probability of affecting the workloads that are typically executed. Indeed, the multitude of functional units and stages in the deeply-pipelined superscalar microprocessors, along with advanced architectural features such as dynamic scheduling and speculative execution, imply that rather complex conditions need to be satisfied in order for an error in the control logic to affect the architectural state of the microprocessor or the outcome of an application. In an effort to capture these additional masking factors, vulnerability analysis methods have been developed specifically for microprocessors [12], [13], [14], [15]. These methods typically employ simulation of actual workload using an architectural performance model or an RT-Level model of the microprocessor and aim to assess the probability that a transient error in a state element will affect workload execution. As we discuss in the next section, however, the use of performance models limits the accuracy of the vulnerability analysis, while the use of RT-Level models requires prohibitive simulation time.

In this paper, we propose a new method for ranking control state elements in modern microprocessors, which maintains the accuracy of RT-Level simulations yet requires several orders of magnitude less simulation time. The proposed method leverages a strong correlation between discrepancies in global microprocessor signals and the probability that an error will affect program execution, in order to provide an accurate ranking of the state elements. The remainder of the paper is structured as follows. Ranking of state elements based on previous vulnerability analysis methods is discussed in Section II. Ranking based on the proposed global signal monitoring method is presented in Section III. The infrastructure employed to perform a comparative evaluation of the two alternatives is described in Section IV and the results are presented in Section V.

II. Ranking based on Architectural Vulnerability Factor (AVF)

The notion of Architectural Vulnerability Factor (AVF) has been extensively used in the past to rank state elements based on their criticality to program execution correctness. AVF expresses the probability of a bit-flip resulting in a visible system error. Previously proposed methods for performing AVF analysis employ either performance models [12] or RT-Level models [13], [14], [15] of a microprocessor. As we explain below, the former enable fast AVF estimation but suffer in terms of accuracy, while the latter offer far more accurate results but require prohibitive simulation times.

978-1-4244-6650-4/10/$26.00 ©2010 IEEE

The performance model-based method described in [12] introduces the concept of Architecturally Correct Execution (ACE) and defines ACE bits as those that can cause corruption in the final output of the program. In contrast, un-ACE bits are those which under no conditions may produce a discrepancy in the final program outcome (i.e. branch predictor bits). ACE analysis is performed and evaluated on an IA-64 performance simulator, using which the authors can generate deterministic AVF estimates for the ACE bits and rank the corresponding state elements based on their criticality. For this purpose, workloads are simulated to completion and the impact of faults in these state elements is analyzed in a single simulator pass, which is performed rapidly. The major drawback of ACE analysis and AVF estimation using the architectural performance model, however, is the lack of detail about the actual hardware structures of the microprocessor. Therefore, the analysis is only performed for the modeled components, which in the case of [12] includes components that affect the performance of a microprocessor. This results in significant loss of accuracy in AVF estimation. Furthermore, extensive manual effort is required to identify the conditions that classify a bit as ACE or un-ACE.

The methods described in [13], [14], [15] resolve the AVF estimation accuracy problem by performing statistical fault injection in the RT-Level model, which reflects the actual hardware structures of the microprocessor. This accuracy, however, comes at a cost: RT-Level simulations are far slower than performance models and fault simulation tools are not readily available at this level. Furthermore, a rigorous transient fault injection campaign requires excessive simulation times in order to provide statistically significant results, especially since AVF computation requires that the workload is executed to completion. In [13], the authors provide a qualitative comparison of the AVF estimation accuracy of their extensive RT-Level simulations to the ACE analysis presented in [12]. Their findings conclude that ACE analysis overestimates soft error vulnerability by about 3.5x and that this discrepancy stems from the model's lack of hardware detail and the single-pass simulation methodology.

III. Ranking based on Global Signal Vulnerability (GSV)

In order to generate an accurate control state element ranking without extensive simulations, we present an alternative method based on monitoring of global microprocessor signals. Specifically, instead of estimating the probability that transient errors in a state element will affect the outcome of a program, we assess the vulnerability of global microprocessor signals to stuck-at faults in these state elements. The underlying conjecture is that there exists a strong correlation between these two metrics. For example, it is expected that an error that causes a discrepancy in a memory access signal will eventually lead to an incorrect program execution outcome. Admittedly, there are cases where such discrepancies will be masked by the application itself, hence the Global Signal Vulnerability (GSV) may not exactly reflect the AVF. Nevertheless, since the proposed method observes *global* signals, such application-level masking of discrepancies on these signals is minimal, as we experimentally corroborate in Section V. Similar to [13], in order to obtain an accurate ranking of the control state elements, we estimate GSV on an RT-Level model of the microprocessor. However, unlike [13] which simulates each possible transient error (or a sample thereof) in a state element, our method only requires one simulation for each stuck-at fault in this state element. As a result, the simulation time required to rank the state elements based on their GSV is several orders of magnitude faster.

A simplified flow diagram of our method is presented in Fig. 1. First, the list of key global signals as well as the list of state elements to be injected are parsed. Examples of key global signals include TLB misses, stalls or register file access signals. For every state element in the list, two microprocessor model replicas are warmed-up for some number of instructions. Afterwards, a stuck-at fault in the specified state element is injected in one of the two models while they execute in parallel. In every clock cycle, the specified global signals are checked for discrepancies. As long as no discrepancies are identified, the simulation proceeds until the clock cycle limit. If a discrepancy is identified, the signal name is stored, the faulty model is reset by transferring the correct state from the golden machine and simulation resumes. In this way, several discrepancies in global signals may be identified in a single fault simulation pass. *The GSV of a state element is defined based on the number of such discrepancies.* The higher the GSV of a state element, the more vulnerable the microprocessor is to transient errors in this state element and, by extension, the higher the probability of an incorrect program outcome. Ranking of control state elements is, subsequently, performed based on their GSV.

Evidently, as compared to the AVF-based ranking of [13], some accuracy loss is expected due to the following reasons:

1) Discrepancies identified at global signals may be masked by the application.
2) GSV may reflect discrepancies that only occur if a state element is stuck-at a value for a multi-cycle period, but would not occur due to a single-cycle transient error.
3) The period between the time of fault injection and the time of discrepancy appearance may underestimate the number of transient errors that would affect the application. For example, if a fault is injected at time $t = 0$ and a discrepancy at a global signal appears at time $t = 100$, any number between 1 and 100 transient errors could potentially result in an incorrect application outcome.

The major advantage of the described method, however, is that ranking is performed using a single simulation pass for each control state element, far less than the sample of 10,000 injections per state element performed in [13]. Furthermore, simulations are not executed to workload completion but rather only to a user-defined end point, thereby speeding up the process.

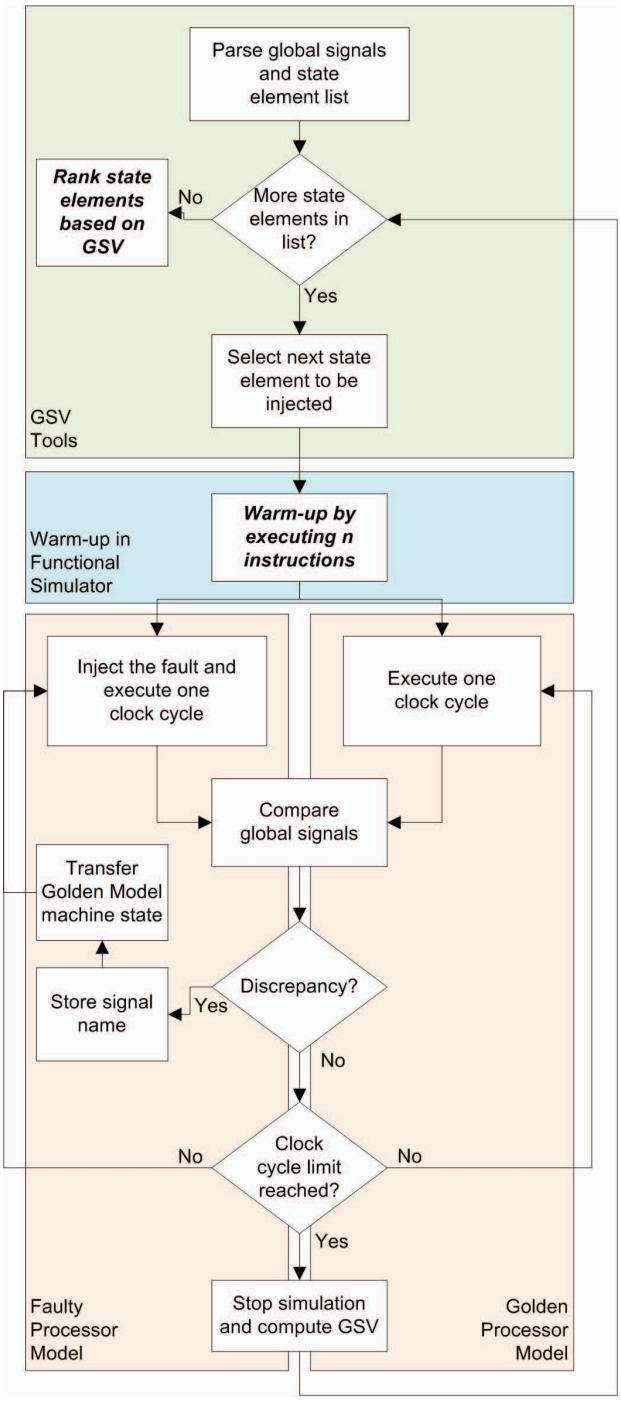


Fig. 1. Flow diagram of GSV-based ranking method

IV. Study Infrastructure

To evaluate the effectiveness of the proposed ranking method, we use a complex, high performance microprocessor model named IVM (Illinois Verilog Model) [16]. IVM is a Verilog model resembling an Alpha 21264 microprocessor and implementing a subset of its instruction set. IVM features a 12-stage pipeline with up to 132 instructions in flight and many modern high performance features, such as out-of-order execution, hybrid branch prediction, dynamic scheduling, speculative execution and memory dependence prediction. The high performance features implemented require a large number of control state elements, which are the main target of this study.

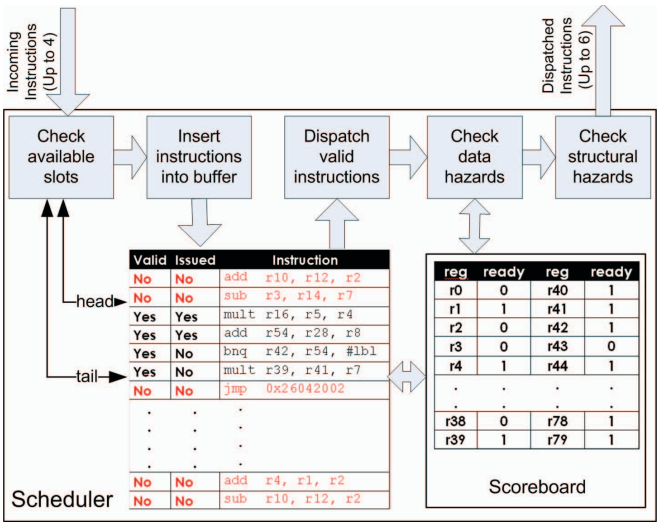


Fig. 2. Instruction scheduler diagram

Specifically, we analyze the instruction scheduler, which is a key control module of the out-of-order logic. The scheduler occupies approximately 16% of the microprocessor area, excluding the caches and the fetch unit, and contains 5,664 control state elements. A diagram of its operation is shown in Fig. 2. The scheduler receives up to 4 instructions from the rename unit, maintaining a buffer of up to 32 instructions. The scheduler dispatches up to 6 instructions (one for each of the six functional units in IVM) depending on the availability of instructions and any data or structural hazards. Its control fields include the program counter, valid and issued bits, functional unit designator, instruction tag, source/destination register pointers etc., fields that are utilized in many control units of the microprocessor. Given its complicated operation and rich set of control fields, the instruction scheduler constitutes an excellent candidate for our analysis.

Since the IVM model implements only a subset of the Alpha ISA, a functional simulator [17] is also used in conjunction with the RT-Level model. The functional simulator can execute a number of instructions before transferring the state to the RT-Level model and can also execute a workload to its completion after the state is transferred back from the RT-Level model. Fault injection is performed during RT-Level simulation by mutating the microprocessor model. More information about the infrastructure as well as the fault injection technique can be found in [18].

V. Experimental Results

In this section, we experimentally compare the ranked lists generated based on AVF and GSV in terms of accuracy and simulation time, and we discuss the results.

TABLE I
STATISTICS FROM EXECUTING UTILIZED BENCHMARKS FOR 10,000 CLOCK CYCLES

SPEC Benchmark	Instructions retired	Conditional branches	Cache accesses	Stalls due to not enough free registers	Stalls due to full memory unit	Application Masking
Training set						
bzip	7015	577	4037	121	24	92%
mcf	2548	91	3823	0	4144	94%
cc	1932	227	2933	4	711	89%
Evaluation set						
perlbmk	1392	122	2589	1	479	95%
crafty	9765	1765	5265	0	0	95%
eon	3359	296	3592	0	352	97%

A. Experimental Setup

A training set of three different benchmarks, namely *bzip*, *mcf* and *cc* is used to generate the two lists and an evaluation set of three more benchmarks, *perlbmk*, *crafty* and *eon* is used for comparing them. The chosen benchmarks represent a variety of typical workload, such as memory intensive, heavy branch usage or high instruction throughput applications. Table I lists some statistics for the workloads utilized. Consistent with the results shown in [13], all these benchmarks exhibit very high application-level masking. All 5,664 control state elements of the instruction scheduler of the IVM microprocessor are used in this study; all simulations are performed using the infrastructure described in section IV on two servers with two Quad-Core processors each.

1) Generating AVF-based ranking: In order to rank the control state elements based on AVF, we repeat the fault injection campaign described in [13]. Specifically, 6,000 transient errors are injected in each of the 5,664 control state elements uniformly at random over a time period of 10,000 clock cycles. The IVM model is warmed-up for 50,000 clock cycles using the functional simulator, the 10,000 clock cycles are subsequently executed at the RT-Level, and then the benchmark continues execution until completion using the functional simulator. The AVF of each control state element is computed as the ratio of incorrect over all 6,000 fault-injected executions. This process is repeated for each of the three different benchmarks in the training set and the final ranked list of control state elements is generated based on the average AVF.

2) Generating GSV-based ranking: Table II lists the global signals which we use for generating the GSV-based ranking of state elements in the IVM scheduler. This selection includes memory access signals (i.e. register file, l1cache, etc.) as well as global error flag signals (i.e. stall, tlbmiss, etc.). Using these signals, the single-pass stuck-at fault injection method described in section III is applied for each of the 5,664 control state elements, both for stuck-at-0 and stuck-at-1 faults. Again, 50,000 clock cycles are used for warm-up in the functional simulator, followed by the progressive stuck-at fault injection during the execution of 10,000 clock cycles at the RT-Level. Note that in the GSV-based ranking method the benchmarks do not need to be executed to completion; hence, simulation stops at this point and the GSV of each state element is computed as the number of times a discrepancy in one of the global microprocessor signals occurs during the fault-injected RT-Level execution of the 10,000 clock cycles. This process is repeated for each of the three different benchmarks in the training set and the final ranked list of control state elements is generated based on the average GSV.

TABLE II
GLOBAL SIGNALS MONITORED

Signal Name	Description
PC_next	Control flow violation
itlbmiss	Invalid instruction pointer
Stall	Global stall signal
val_out	Register file corruption
data[1-2]_out, addr[1-2]_out	Off-chip memory access discrepancy
dtlbmiss	Invalid memory access

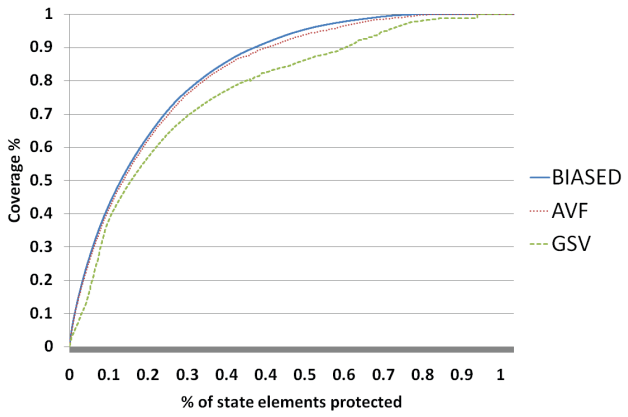
B. Results

We now compare and assess the accuracy of the two ranked lists, as well as the simulation time required to generate them.

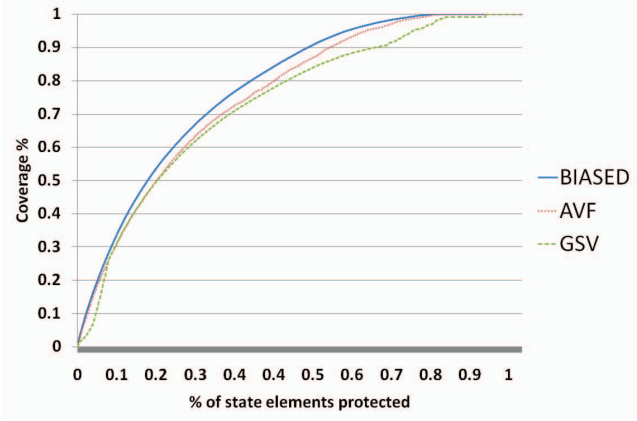
1) Positional comparison of ranked lists: The average positional difference of a state element in the two ranked lists is 24. Given that the lists consist of 5,664 elements, this amounts to 0.42%, implying that the two ranked lists are very similar.

2) Coverage comparison of ranked lists: Fig. 3 reports the coverage achieved by each of the two ranked lists for each of the three benchmarks in the training set and each of the three benchmarks in the evaluation set. The Y-axis represents the percentage of transient errors that are suppressed by protecting the corresponding percentage of the 5,664 state elements shown in the X-axis. As a point of reference, a third curve (i.e. biased) is also plotted for each benchmark, reflecting an optimal state element ranking with regards to this particular benchmark only. As may be observed, both the AVF-based and the GSV-based ranked lists achieve near-optimal coverage and the difference between them is very small. The average difference between the coverage achieved by the AVF-based and the GSV-based ranked lists over any percentage of protected state elements is 1.4%.

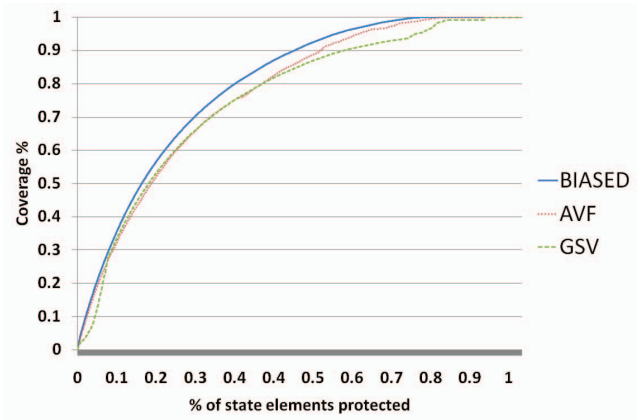
3) Simulation times: The main advantage of the presented technique lies in the simulation times required to generate the near-optimal ranking of control state elements. Table III contrasts the time required for calculating the AVF-based and GSV-based ranked lists. The GSV-based method achieves a 1,215x speed-up over the AVF-based method. This difference in simulation time is expected since the GSV-based method



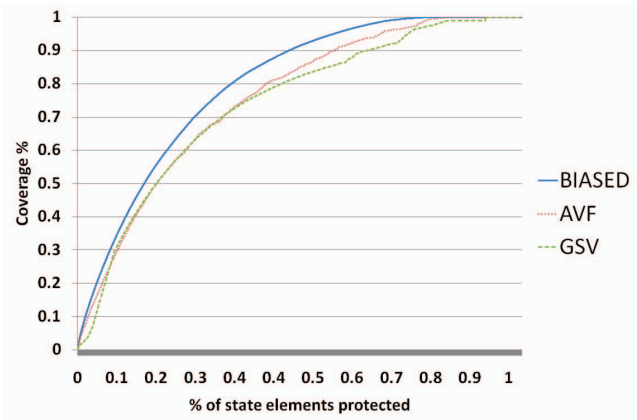
(a) bzip



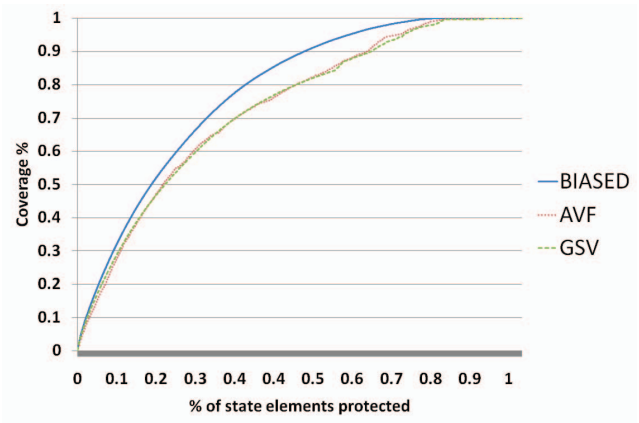
(b) mcf



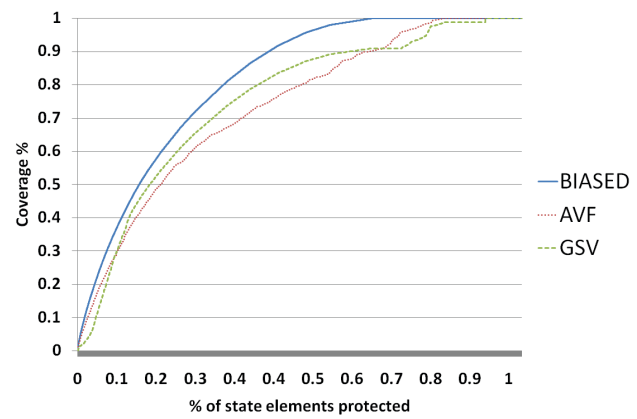
(c) cc



(d) perlbmk



(e) crafty



(f) eon

Fig. 3. Coverage comparison of AVF-based and GSV-based ranked lists

collects ranking data for a state element in 6 passes (i.e. two passes, one for stuck-at-0 and one for stuck-at-1 faults, for each of the three benchmarks in the training set), while the AVF-based method performs statistical transient error injection and requires 18,000 passes (i.e. 6,000 transient errors for each of the three benchmarks in the training set). Furthermore, benchmarks in the GSV-based method are not run to completion, thus saving additional time. However, each stuck-at fault simulation pass in the GSV-based method is slower than a transient error injection pass in the AVF-based method due to the overhead of clearing the faulty machine state and resuming the simulation. Nevertheless, the overall speedup exceeds three orders of magnitude while the ranking accuracy is minimally impacted.

TABLE III
RANKING SPEED-UP

SPEC Benchmark	CPU hours AVF[1]	GSV	GSV speedup
bzip	6,173	5.42	1,138x
mcf	8,850	6.51	1,359x
cc	6,490	5.79	1,120x
Total	**21,513**	**17.7**	**1,215x**

[1] 10,000 clock cycles run in the RT-Level model and the rest in the functional simulator.

C. Discussion

The simulation times needed to generate a ranked list of control state elements based on their criticality to workload execution using RT-Level estimation of the AVF metric [13] is prohibitive for a full microprocessor model. As can be observed in Table III, ranking the control state elements of only the instruction scheduler while utilizing a limited set of 3 benchmarks required 2 months of extensive simulations on 16 microprocessor cores, rendering any extensive study infeasible. Thus, the speed-up achieved by the presented GSV-based ranking method is essential in order to enable a designer to perform full-scale analysis of microprocessor control state elements within a reasonable time.

The results of this analysis can be used to guide an efficient allocation of resources, aiming to maximize reliability improvement given a certain budget. For example, if a designer is allowed to allocate 10% of the area for protecting the most important control state elements, the presented method can provide a rapid, yet still accurate ranking of the control state elements based on their criticality to instruction execution correctness.

Besides the absolute ranking of state elements, such analysis results can further provide insight to the designer with regards to vulnerable portions of the modules. Browsing through the ranked lists for the IVM scheduler, we notice storage elements that are obvious candidates and are expected to cause problems when unprotected, such as the `issue_head` or `issue_tail` pointers. However, among the top ranked state elements we also find unexpected ones which are not straightforward high-risk candidates, such as a set of registers called `temp_full`, which temporarily holds portions of the instruction to be issued. Using this information, the designer may choose to refine the design taking into account the criticality of the various control state elements.

VI. CONCLUSION

This study reveals a strong correlation between the probability that transient errors in a control state element will affect the outcome of a program and the vulnerability of select global microprocessor signals to stuck-at faults in this state element. This correlation is leveraged by the method described herein in order to quickly and accurately rank control state elements based on their criticality to application execution correctness. Experimental results demonstrate that the ranked state element list obtained by the proposed method is very similar to the one obtained using the AVF metric, yet the time needed to generate it is three orders of magnitude faster.

REFERENCES

[1] L.R. Rockett Jr, "An SEU-hardened CMOS data latch design," *IEEE Transactions on Nuclear Science*, vol. 35, no. 6, pp. 1682–1687, 1988.

[2] M. Zhang, S. Mitra, T.M. Mak, N. Seifert, N.J. Wang, Q. Shi, K.S. Kim, N.R. Shanbhag, and S.J. Patel, "Sequential element design with built-in soft error resilience," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 14, no. 12, pp. 1368–1378, 2006.

[3] Q. Zhou and K. Mohanram, "Transistor sizing for radiation hardening," in *International Reliability Physics Symposium*, 2004, pp. 310–315.

[4] N. Miskov-Zivanov and D. Marculescu, "MARS-C: modeling and reduction of soft errors in combinational circuits," in *Design Automation Conference*, 2006, pp. 767–772.

[5] C. Zhao, X. Bai, and S. Dey, "A scalable soft spot analysis methodology for compound noise effects in nano-meter circuits," in *Design Automation Conference*, 2004, pp. 894–899.

[6] M. Zhang and N.R. Shanbhag, "Soft-error-rate-analysis (SERA) methodology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2140–2155, 2006.

[7] R. Garg, N. Jayakumar, S.P. Khatri, and G. Choi, "A design approach for radiation-hard digital electronics," in *Design Automation Conference*, 2006, pp. 773–778.

[8] Q. Zhou and K. Mohanram, "Gate sizing to radiation harden combinational logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 1, pp. 155–166, 2006.

[9] S. Almukhaizim, Y. Makris, Y. Yang, and A. Veneris, "Seamless Intergration of SER in Rewiring-based Design Space Exploration," in *International Test Conference*, 2006, vol. 2, pp. 29.3.1–29.3.9.

[10] C.G. Zoellin, H.J. Wunderlich, I. Polian, and B. Becker, "Selective Hardening in Early Design Steps," in *European Test Symposium*, 2008, pp. 185–190.

[11] S. Krishnaswamy, S.M. Plaza, I.L. Markov, and J.P. Hayes, "Signature-based SER analysis and Design of Logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 74–86, 2009.

[12] 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 *International Symposium on Microarchitecture*, 2003, pp. 29–40.

[13] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ACE analysis reliability estimates using fault-injection," *SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 460–469, 2007.

[14] E.W Czeck and D.P Siewiorek, "Effects of transient gate-level faults on program behavior," in *International Symposium on Fault-Tolerant Computing*, 1990, pp. 236–243.

[15] K. Seongwoo and A.K. Somani, "Soft error sensitivity characterization for microprocessor dependability enhancement strategy," in *International Conference on Dependable Systems and Networks*, 2002, pp. 416–425.

[16] 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 *International Conference on Dependable Systems and Networks*, 2004, pp. 61–70.

[17] D. Burger, T. M. Austin, and S. Bennett, "Evaluating future microprocessors: The simplescalar tool set," Tech. Rep. CS-TR-1996-1308, 1996.

[18] N. Karimi, M. Maniatakos, Y. Makris, and A. Jas, "On the correlation between controller faults and instruction-level errors in modern microprocessors," in *International Test Conference*, 2008, pp. 24.1.1–24.1.10.