Global Signal Vulnerability (GSV) Analysis for Selective State Element Hardening in Modern Microprocessors

Michail Maniatakos, *Student Member*, *IEEE*, Chandrasekharan (Chandra) Tirumurti, *Member*, *IEEE*, Rajesh Galivanche, *Senior Member*, *IEEE*, and Yiorgos Makris, *Senior Member*, *IEEE*

Abstract—Global Signal Vulnerability (GSV) analysis is a novel method for assessing the susceptibility of modern microprocessor state elements to failures in the field of operation. In order to effectively allocate design for reliability resources, GSV analysis takes into account the high degree of architectural masking exhibited in modern microprocessors and ranks state elements accordingly. The novelty of this method lies in the way this ranking is computed. GSV analysis operates either at the Register Transfer (RT-) or at the Gate-Level, offering increased accuracy in contrast to methods which compute the architectural vulnerability of registers through high-level simulations on performance models. Moreover, it does not rely on extensive Statistical Fault Injection (SFI) campaigns and lengthy executions of workloads to completion in RT- or Gate-Level designs, which would 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 and Reorder Buffer modules of an Alpha-like microprocessor and a modern Intel microprocessor corroborates that GSV analysis generates a near-optimal ranking, yet is several orders of magnitude faster than existing RT- or Gate-Level approaches.

Index Terms—GSV, AVF, reliability, vulnerability analysis, modern microprocessor, control logic, workload.

1 Introduction

Aggressive technology advancements and continuous shrinking of modern process geometries enable multiple sources of errors to jeopardize the reliability of modern microprocessor designs. Among them, soft (or transient) errors due to neutron particles from cosmic rays and alpha particles from packaging material [1], [2] have resurfaced as a key point of interest. Reports of numerous incidents in major server lines and high availability computer systems [3] highlight the extent of the problem. Besides soft errors, uncertainties caused by process, voltage and temperature variations have also become a serious concern, forcing designers to use pessimistic design margins. This, in turn, elucidates the conflicting objectives of reliable operation and high performance, since the latter is typically achieved in modern microprocessors through aggressive voltage scaling and power manipulation.

To address these increasing threats, a plethora of design solutions for protecting latches from Single Event Upsets (SEUs), as well as combinational logic from Single Event Transients (SETs) have been proposed. Solutions can be generally applicable, such as duplication or triplication [4], latch hardening techniques [5], [6] or transistor sizing for reliability techniques [7], [8]. Alternatively, solutions that are customized to specifically exploit modern microprocessor features have also been proposed: checkpointing and symptom-based error detection and recovery were introduced in [9], functional inherent codes to exploit the inherent redundancy present in modern microprocessors were utilized in [10], a small, in-order checker was used to detect errors in a high-performance microprocessor in [11], while flushing as an efficient technique to protect against errors was suggested in [12].

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 [13], [14], in order to support partial hardening approaches [15], [16], [17], [18]. 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 as 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.

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 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

- *M. Maniatakos is with the Department of Electrical Engineering, Yale University, 10 Hillhouse Ave. #505, New Haven, CT 06520-8267. E-mail: michail.maniatakos@yale.edu.*
- *C. Tirumurti and R. Galivanche are with the Validation and Test Solutions Group, Intel Corporation, Santa Clara, CA 95050. E-mail: {chandra.tirumurti, rajesh.galivanche}@intel.com.*
- *Y. Makris is with the Department of Electrical Engineering, The University of Texas at Dallas, Richardson, TX 75080. E-mail: yiorgos.makris@utdallas.edu.*

Manuscript received 28 Feb. 2011; revised 11 Aug. 2011; accepted 13 Aug. 2011; published online 31 Aug. 2011.
Recommended for acceptance by S. Shukla.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-02-0133.
Digital Object Identifier no. 10.1109/TC.2011.172.

0018-9340/12/$31.00 © 2012 IEEE Published by the IEEE Computer Society

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 [19], [20], [21], [22]. These methods typically employ simulation of actual workload using a 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 vulnerability analysis, while the use of RT-Level models requires prohibitive simulation time.

In this paper, we propose a new method for ranking state elements in modern microprocessors. The proposed method maintains the accuracy of RT- and Gate-Level simulations, yet requires several orders of magnitude less simulation time. To this end, it 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 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 2. Ranking based on the proposed global signal monitoring method is introduced in Section 3. The infrastructure employed to perform a comparative evaluation of the two alternatives is described in Section 4. The setup used for the experiments is discussed in Section 5, and the results are presented in Section 6.

2 Ranking Based on Architectural Vulnerability Factor (AVF)

The notion of Architectural Vulnerability Factor 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. Visible system errors include operating system crashes, incorrect program outputs, microprocessor stalls, etc. Not all faults in microprocessor structures affect the outcome of an application; a fault might be suppressed by several factors, such as electrical, logical, and latch-window masking. Furthermore, architectural and application-level masking might also prevent a fault from affecting the user. For example, an error in the branch prediction can lead to an incorrect prediction with no visible impact to the user. Thus, the branch predictor has an AVF of 0. On the other hand, an error affecting the address where data are stored might appear to the user as an illegal address exception or corrupt some files on the hard disk. Arguing that a bit has an AVF factor of 0.80 implies that, on average, 8 out of 10 bit-flips on that particular bit will cause a visible system error. AVF is extremely useful in evaluating the reliability of a design and calculating whether a given architecture meets the Mean Time Between Failures (MTBF) and Failures in Time (FIT) specifications required. For example, to meet an MTBF target of 1,000 years (114 FIT), given that all bits exhibit an AVF factor of 100 percent, one concludes that 80 percent of these bits must be protected [19].

Accurate calculation of AVF is not trivial and requires lengthy simulations. Previously proposed methods for performing AVF analysis employ either performance models [19] or RT-Level models [20], [21], [22] 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.

The performance model-based method described in [19] introduces the concept of Architecturally Correct Execution (ACE) and defines ACE bits as those that can cause corruption in the final output of a 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, analysis is only performed for the modeled components, which in the case of [19] 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 [20], [21], [22] 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 [20], the authors provide a qualitative comparison of the AVF estimation accuracy of their extensive RT-Level simulations to the ACE analysis presented in [19]. Their findings conclude that ACE analysis overestimates soft error vulnerability by about 3.5× and that this discrepancy stems from the model's lack of hardware detail and the single-pass simulation method. In [23], the authors of [20] counter argued that added detail to ACE analysis can lead to tighter AVF bounds. However, more detail slows down simulation, defeating the purpose of fast performance simulators. Thus, the limited details available at the performance model remain the main contributor to AVF overestimation.

3 Ranking Based on Global Signal Vulnerability (GSV)

In order to generate an accurate state element ranking without extensive simulations, we present an alternative method based on monitoring of global microprocessor signals, called Global Signal Vulnerability analysis. Specifically, instead of estimating the probability that transient errors in a state element will affect the outcome of a program (i.e., the AVF), we assess the vulnerability of global microprocessor signals to stuck-at faults in this state element.

Specifically, *the GSV of a state element is defined as the number of discrepancies that appear on predefined global signals during execution of representative workload in the presence of stuck-at faults in this state element.* Every time a discrepancy appears at a global signal, the GSV of the fault-injected state element is incremented and the fault simulation resumes after reinstating the correct state. Intuitively, the higher the GSV of a state element, the more vulnerable the microprocessor is to errors in this state element and, by extension, the higher the probability of an incorrect program outcome.

Algorithm 1 presents the details of the GSV calculation method. The input sets of the GSV algorithm are:

1. The set of latches to be analyzed and ranked; this set can include any number of latches from any hardware structure in the microprocessor.
2. The set of workload portions used for ranking; long traces and large variety of workloads provide more accurate results.
3. The set of signals to be monitored for discrepancies; these signals can be generic, such as off-chip memory access signals and any signal in the chip periphery, or more specific, such as TLB misses and specific stall signals indicating severe malfunctions.

After the designer defines the inputs of the algorithm, the microprocessor is warmed up either by simulating the workload from the beginning until cycle W_{start} or by restoring a checkpointed state of the microprocessor at the same cycle. Afterward, the chosen workload is executed for the duration defined by $[W_{start}, W_{end}]$. Then, for every latch, stuck-at-0 and stuck-at-1 faults are injected; for every discrepancy that appears in any of the signals in the $GSVList$, the GSV of the latch increases by 1 and the faulty microprocessor state is replaced by the correct one. Ranking of state elements is, subsequently, performed based on their GSV.

Algorithm 1. GSV algorithm

Algorithm 1: GSV algorithm

```
1  Assume T is the set of latches to be ranked;
2  Assume Workload_[start,end] is the set of workloads;
3  Assume GSVList is the set of monitored signals;
4  foreach Latch in T do
5  |  Initialize GSV_latch = 0 ;
6  end
7  foreach W in Workload do
8  |  Warm-up microprocessor until cycle W_start ;
9  |  foreach FaultType in [stuck-at-0, stuck-at-1] do
10 |  |  foreach Latch in T do
11 |  |  |  for c = W_start; c < W_end; c++ do
12 |  |  |  |  Inject FaultType in Latch and fault
   |  |  |  |  simulate;
13 |  |  |  |  if Discrepancy appears in GSVList then
14 |  |  |  |  |  GSV_latch += 1;
15 |  |  |  |  |  Clear faulty state;
16 |  |  |  |  end
17 |  |  |  end
18 |  |  end
19 |  end
20 end
```

The underlying conjecture is that there exists a strong correlation between GSV and AVF. Despite this correlation, differences compared to AVF-based ranking are expected due to the following reasons:

- Discrepancies identified at global signals may be eventually masked by the application. Examples include data written in L1 caches that are never used by the application.
- GSV may reflect discrepancies that only occur if a state element is stuck-at a value for a multicycle period, but would not occur due to a single-cycle transient error. For example, if a transient error alters the ready bit of an instruction to be issued, then this instruction will be issued at the next cycle and the error will be masked. If the same bit is stuck-at a value for more cycles, this instruction will never be issued and the microprocessor will stall.
- 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.

Nevertheless, since the proposed method observes *global* signals, the impact of these differences on the accuracy of the ranked list is minimal, as we experimentally corroborate in Section 6. Similar to [20], in order to obtain an accurate ranking of the state elements, we estimate GSV on an RT-Level model of the microprocessor. However, instead of simulating all possible transient errors in a state element or a sample thereof (e.g., 10,000 injections per state element were performed in [20]), 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*.

4 Study Infrastructure

To evaluate the effectiveness of the proposed ranking method, we use two different complex, high-performance microprocessor models: Illinois Verilog Model (IVM) [24], which is an open-source Alpha 21264 microprocessor and a modern Intel microprocessor, which implements the P6 architecture [25]. Both models implement many high-performance features, requiring a large number of state elements, which are the main target of this study.

4.1 Microprocessor Models

IVM. 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. A block diagram of the microprocessor is presented in Fig. 1.

Since the IVM model implements only a subset of the Alpha ISA, a functional simulator [26] is also used in

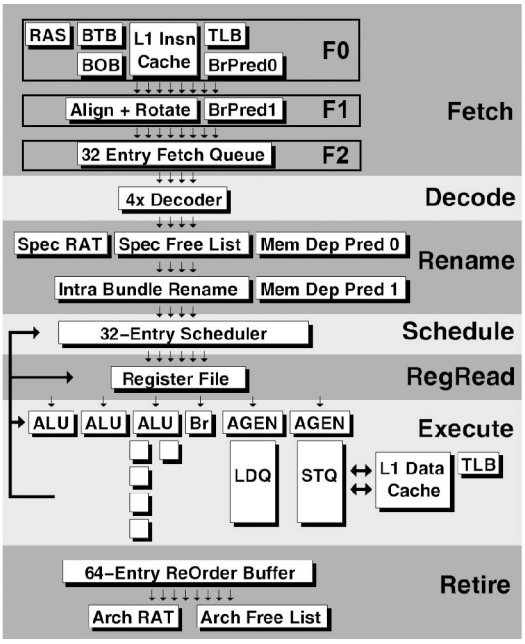


Fig. 1. IVM architecture [24].

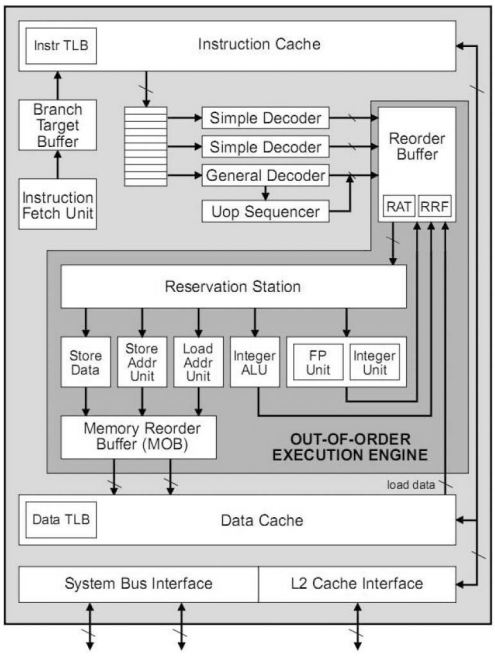


Fig. 2. P6 architecture [25].

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 [27].

Intel P6. In order to evaluate the effectiveness of GSV in commercial microprocessors, we performed extensive experiments on a contemporary Intel microprocessor implementing the P6 architecture, whose features include out-of-order execution, register renaming, superpipelining, and speculative execution.

Fig. 2 shows the basic P6 architecture. Instructions flow from the instruction cache to the decoders, where they enter the out-of-order cluster and are stored in the Reservation Station (RS). The Reorder Buffer (ROB) guarantees in-order retirement after out-of-order instruction execution. The Memory Reorder Buffer (MOB) interacts with the data cache in order to fetch the required information.

4.2 GSV Infrastructure

In order to implement the algorithm presented in Section 3, we built an infrastructure around the two microprocessor models. A simplified flow diagram of our method is presented in Fig. 3. First, the list of key global signals and the list of state elements to be injected are parsed. Examples of key global signals include TLB misses, stalls, and register file access signals. For every state element in the list, two microprocessor model replicas are warmed up for a number of instructions. Next, 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 discrepancy is identified, the simulation proceeds until a prespecified 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 the simulation resumes. In this way, several discrepancies in global signals may be identified in a single fault simulation pass.

Due to certain limitations of the designs and the tools, some parts of the infrastructure (gray boxes in Fig. 3) were implemented differently for the two microprocessor models. Specifically, warm up in IVM is performed with the use of a functional simulator (since IVM does not implement the entire Alpha ISA). Intel P6 is warmed up by loading a checkpointed state of the whole microprocessor on each iteration. As for the golden model state transfer, clearing the faulty state in IVM involves deleting the faulty machine and creating a new replica of the golden one. For the Intel microprocessor, the closest checkpoint is loaded and a fault free simulation is performed until the desired clock cycle.

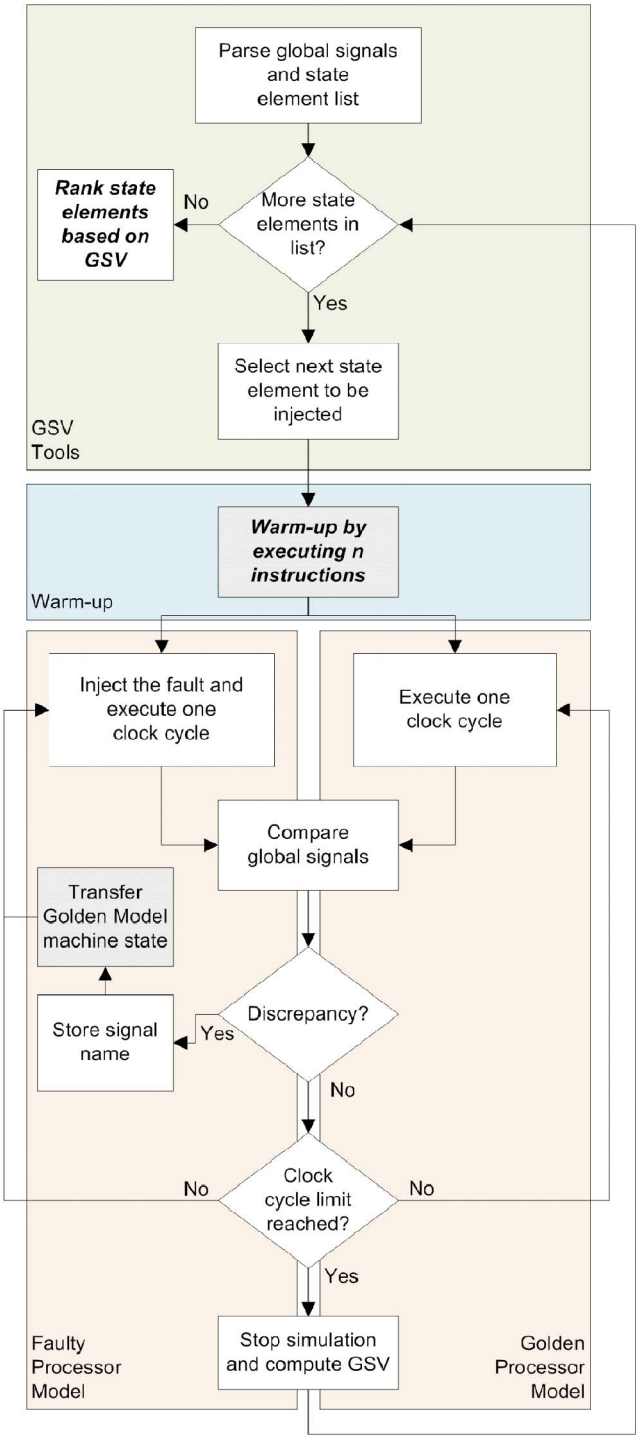
5 Experimental Setup

This section describes the details of the performed analysis. Simulations on IVM were performed using two servers with two Quad-Core processors each, while simulations on P6 were performed using Intel's in-house tools and infrastructure.

5.1 Modules

Two modules were selected for GSV evalution: The instruction scheduler (named reservation station in P6) and the ReOrder Buffer. Both modules are vital parts of the out-of-order execution of the two microprocessors; given their complicated operation and rich set of control fields, both modules constitute excellent candidates for our analysis.

Scheduler. The scheduler receives several instructions from the rename unit, maintaining a buffer of up to 128 instructions. It dispatches instructions to each functional



unit depending on the availability of instructions and the presence of 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. The scheduler in IVM consists of 5,664 latches.[1]

ROB. The reorder buffer ensures correct execution order within the out-of-order cluster. It assigns an identification tag to each instruction and validates proper commitment. All finished instructions are stored in a buffer and their results commit to the Real Register File (RRF) after all preceding instructions have committed. The ROB in IVM consists of 12,176 latches.

5.2 Workload

A training set of three different benchmarks, namely *bzip*, *mcf*, and *cc* is used to generate AVF-based and GSV-based 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 1 lists some statistics for the workloads utilized. Consistent with the results shown in [20], all these benchmarks exhibit very high masking factors.

5.3 Generating Ranked Lists

Generating AVF-based ranking. In order to rank the state elements based on AVF, we repeat the fault injection campaign described in [20]. Specifically, 6,000 transient errors are injected to all state elements uniformly at random over a time period of 10,000 clock cycles. As shown in [28], the vast majority of the faults appear within that time period. The AVF of each 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 state elements is generated based on the average AVF.

Generating GSV-based ranking. Table 2 lists the global signals which we use for generating the GSV-based ranking of state elements in IVM. Similar signals were selected for the analysis on P6. This selection includes memory access signals (i.e., register file, l1cache, etc.) as well as global error flag signals (i.e., stall, tlb miss, etc.). Using these signals, the single-pass stuck-at fault injection method described in Section 3 is applied to each state element, both for stuck-at-0 and stuck-at-1 faults. Note that in the GSV-based ranking method the benchmarks do not need to be executed to completion; hence, simulation stops at the end of the selected workload portion 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 execution. This process is repeated for each of the three different benchmarks in the training set and the final ranked list of state elements is generated based on the average GSV.

6 Experimental Results

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

6.1 Positional Comparison of Ranked Lists

The first set of results compares the position difference of the latches in the two lists generated by AVF and GSV analysis. Table 3 shows that the average difference for the

TABLE 1
Statistics from Executing Utilized Benchmarks for 10,000 Clock Cycles

SPEC Benchmark	Instructions retired	Conditional branches	Cache accesses	Stalls due to insufficient free registers	Stalls due to full memory unit	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%

three benchmarks used for evaluation is less than 56 positions, for both microprocessors and both modules. Given that in IVM the scheduler consists of 5,664 elements and the ROB consists of 12,176 elements, this average positional difference amounts in less than 1 percent for both modules, implying that the two ranked lists are very similar. Results for P6 are consistent with this observation.

6.2 Coverage Comparison of Ranked Lists

Figs. 4 and 5 report 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, for both modules and microprocessor models. The Y-axis represents the percentage of transient errors that are suppressed by protecting the corresponding percentage of the state elements shown in the X-axis, for each benchmark shown in Z-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. Indeed, 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 percent for the Scheduler of IVM (Fig. 4a), 1.6 percent for the ROB of IVM (Fig. 4b), 2.4 percent for the Scheduler of P6 (Fig. 5a), and 0.9 percent for the ROB of P6 (Fig. 5b). *Thus, selecting the GSV over the AVF list in order to protect a fixed amount of state elements results in insignificant coverage loss.*

Concerning the coverage of the lists vis-a-vis the biased list, AVF and GSV perform comparably as shown in Table 4. The numbers shown in Table 4 are the average among the two microprocessors and the two modules. On average, the coverage difference between the biased list and the AVF-based list is 5.3 percent, while the same figure between the biased list and the GSV-based list is 5.6 percent. *Thus, both AVF-based and GSV-based lists can provide accurate measurements about the vulnerability of hardware structures.*

6.3 Simulation Times

The main advantage of the presented technique lies in the simulation times required to generate the near-optimal ranking of state elements. Tables 5 and 6 contrast the time required for calculating the AVF-based and GSV-based ranked lists. The GSV-based method achieves a $1{,}215\times$ speedup over the AVF-based method for IVM, while the corresponding speedup for the Intel microprocessor is $1{,}017\times$. This difference in simulation time is expected since the GSV-based method collects ranking data for a state element in six 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). Apparently, if a larger sample was used for statistical fault injection the speedup would be even greater. 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 reinstating the fault-free machine state and resuming the simulation. *Nevertheless, the overall speedup exceeds three orders of magnitude while the ranking accuracy is minimally impacted.*

6.4 Discussion

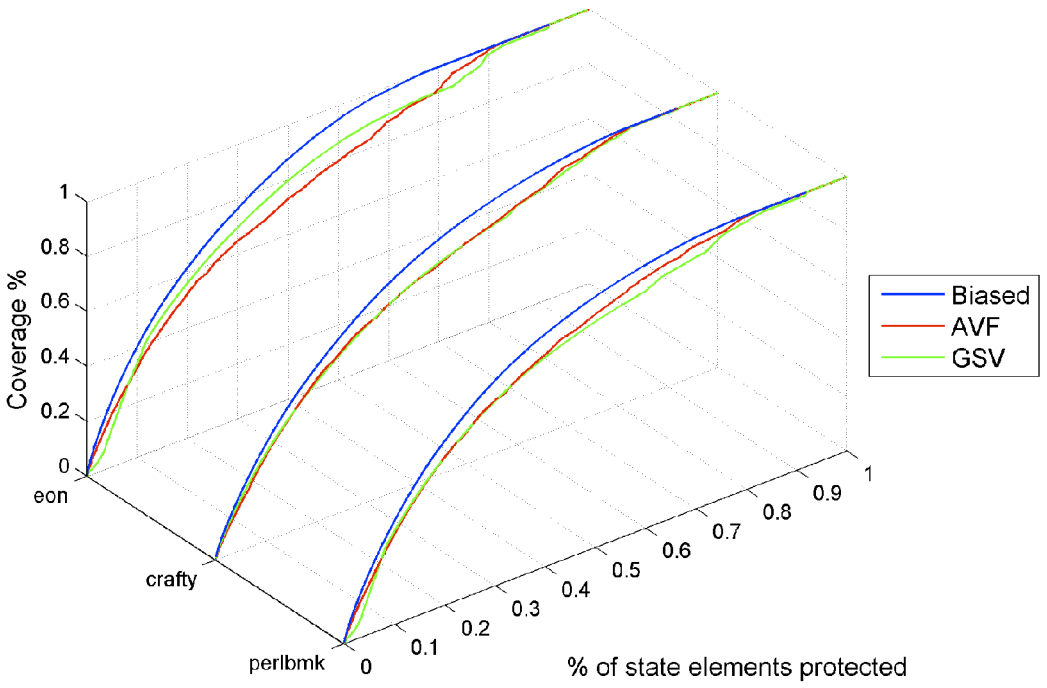
The simulation times needed to generate a ranked list of state elements based on their criticality to workload execution

TABLE 2
Global Signals Monitored in IVM

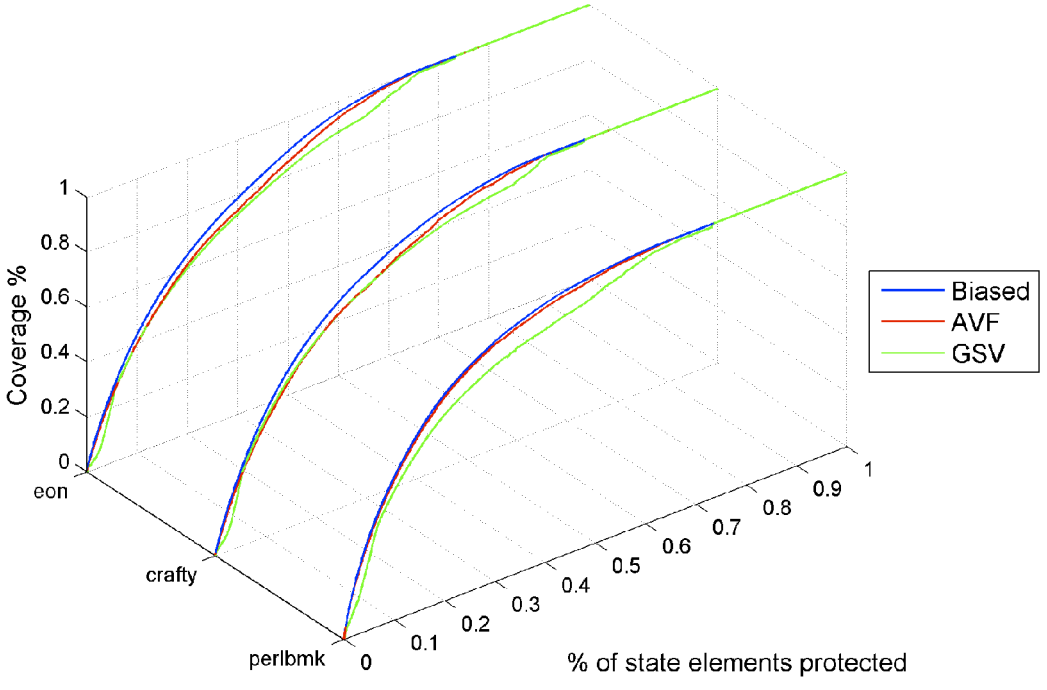
Signal Name	Description
PC_next	Control flow violation
itlbmiss	Invalid instruction pointer
Stall	Global stall signal
val_out	Register file corruption
data[1-2]_out, addr[1-2]_out	Off-chip memory access discrepancy
dtlbmiss	Invalid memory access

TABLE 3
Average Position Difference in Ranked Lists

SPEC Benchmark	Average Position Difference			
	Alpha 21264		Intel P6	
	Scheduler	ROB	Scheduler	ROB
perlbmk	21	66	41	32
crafty	14	38	66	21
eon	38	41	62	47
Average	24	48	56	33



(a) IVM Scheduler

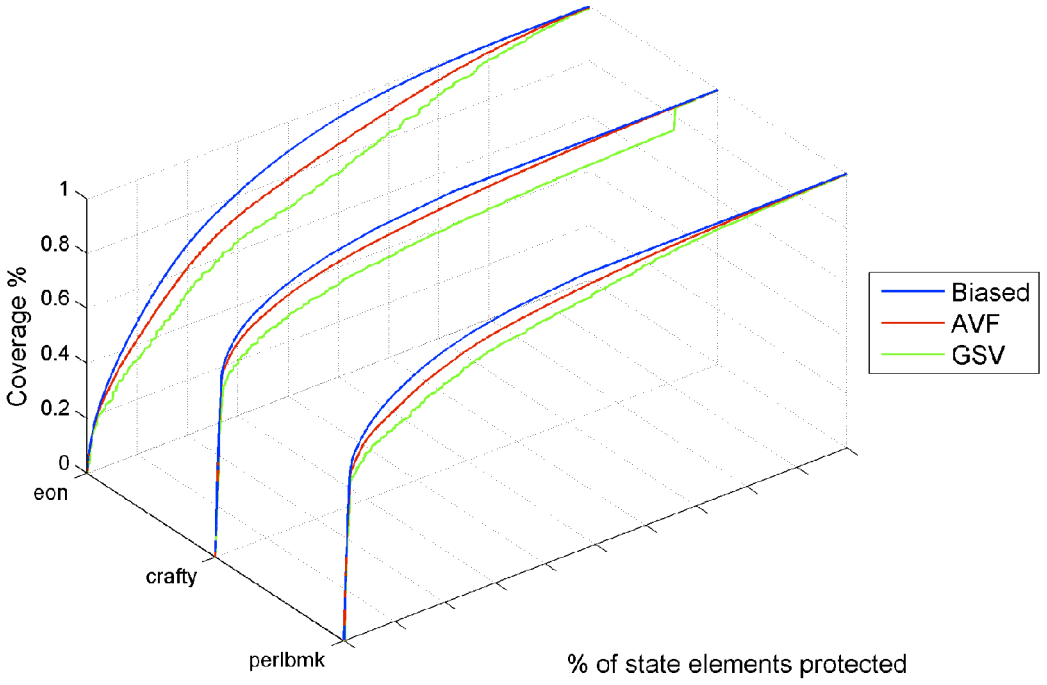


(b) IVM ROB

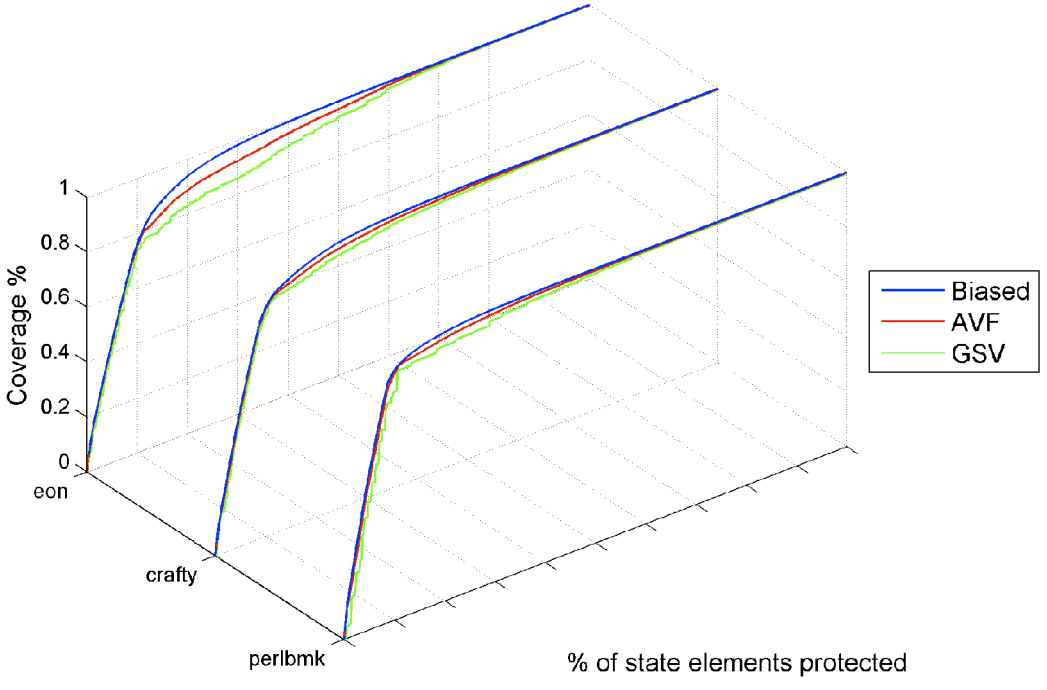
Fig. 4. Coverage comparison of AVF-based and GSV-based ranked lists in IVM.

using RT-Level estimation of the AVF metric [20] are prohibitive for a complete microprocessor model. As can be observed in Table 5, ranking the state elements of only the instruction scheduler while utilizing a limited set of three benchmarks required two months of extensive simulations on 16 microprocessor cores, rendering any extensive study infeasible. Thus, the speedup achieved by the presented GSV-based ranking method is essential in order to enable a designer to perform full-scale vulnerability analysis of microprocessor 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 specified budget. For example, if a designer is allowed to allocate 10 percent of the area for protecting the most important state elements, the presented method can provide a rapid, yet still very



(a) P6 Scheduler



(b) P6 ROB

Fig. 5. Coverage comparison of AVF-based and GSV-based ranked lists in P6.

accurate ranking of the 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 state elements.

TABLE 4
Coverage and Ranking Difference Compared to Biased

SPEC Bench-mark	Coverage difference compared to biased AVF Max	AVF Avg	GSV Max	GSV Avg	Average ranking position difference compared to biased AVF	GSV
perlbmk	8.4%	4.2%	9.6%	5.7%	61	78
crafty	9.2%	4.7%	9.3%	4.9%	65	68
eon	11.3%	7.0%	12.7%	7.1%	91	81
Average	9.6%	5.3%	10.5%	5.6%	72	75

TABLE 5
Ranking Speedup for IVM

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.

TABLE 6
Ranking Speedup for P6

SPEC Benchmark	CPU hours AVF	GSV	GSV speedup
bzip	1,121	1.21	923x
mcf	1,802	1.62	1,106x
cc	1,326	1.35	981x
Total	**4,249**	**4.18**	**1,017x**

7 Conclusion

This study reveals a strong correlation between the probability that transient errors in a 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 state elements based on their criticality to application execution correctness. Experimental results on both an open-source Alpha microprocessor and a commercial Intel microprocessor demonstrate that the ranked state element list obtained by GSV is equally accurate to the one obtained using the AVF metric, yet the time needed to generate it is three orders of magnitude faster.

Acknowledgments

This work was supported by a generous gift from Intel Corporation The first author performed part of this research during a summer internship with Intel Corporation in Santa Clara, CA. Preliminary versions of parts of the results reported herein were presented at the 2010 VLSI Test Symposium [29].

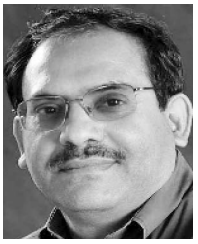
References

[1] E. Normand, "Single Event Upset at Ground Level," *IEEE Trans. Nuclear Science,* vol. 43, no. 6, pp. 2742-2750, Dec. 1996.
[2] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability," *IEEE Micro,* vol. 23, no. 4, pp. 14-19, July/Aug. 2003.
[3] R.C. Baumann, "Soft Errors in Commercial Semiconductor Technology: Overview and Scaling Trends," *IEEE Reliability Physics Tutorial Notes, Reliability Fundamentals,* 2002.
[4] E. Touloupis, J.A. Flint, V.A. Chouliaras, and D.D. Ward, "Study of the Effects of SEU-Induced Faults on a Pipeline Protected Microprocessor," *IEEE Trans. Computers,* vol. 56, no. 12, pp. 1585-1596, Dec. 2007.
[5] L.R. Rockett Jr, "An SEU-Hardened CMOS Data Latch Design," *IEEE Trans. Nuclear Science,* vol. 35, no. 6, pp. 1682-1687, Dec. 1988.
[6] 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 Trans. Very Large Scale Integration Systems,* vol. 14, no. 12, pp. 1368-1378, Dec. 2006.
[7] Q. Zhou and K. Mohanram, "Transistor Sizing for Radiation Hardening," *Proc. Int'l Reliability Physics Symp.,* pp. 310-315, 2004.
[8] N. Miskov-Zivanov and D. Marculescu, "MARS-C: Modeling and Reduction of Soft Errors in Combinational Circuits," *Proc. Design Automation Conf.,* pp. 767-772, 2006.
[9] N.J. Wang and S.J. Patel, "Restore: Symptom Based Soft Error Detection in Microprocessors," *Proc. Int'l Conf. Dependable Systems and Networks,* pp. 30-39, 2005.
[10] C. Metra, D. Rossi, M. Omana, A. Jas, and R. Galivanche, "Function-Inherent Code Checking: A New Low Cost On-Line Testing Approach for High Performance Microprocessor Control Logic," *Proc. IEEE European Test Symp.,* pp. 171-176, 2008.
[11] T.M. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," *Proc. ACM/IEEE Int'l Symp. Microarchitecture,* pp. 196-207, 1999.
[12] S.S. Mukherjee, M. Kontz, and S.K. Reinhardt, "Detailed Design and Evaluation of Redundant Multithreading Alternatives," *Proc. Int'l Symp. Computer Architecture,* pp. 99-110, 2002.
[13] C. Zhao, X. Bai, and S. Dey, "A Scalable Soft Spot Analysis Methodology for Compound Noise Effects in Nano-Meter Circuits," *Proc. Design Automation Conf.,* pp. 894-899, 2004.
[14] M. Zhang and N.R. Shanbhag, "Soft-Error-Rate-Analysis (SERA) Methodology," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 25, no. 10, pp. 2140-2155, Oct. 2006.
[15] R. Garg, N. Jayakumar, S.P. Khatri, and G. Choi, "A Design Approach for Radiation-Hard Digital Electronics," *Proc. Design Automation Conf.,* pp. 773-778, 2006.
[16] S. Almukhaizim, Y. Makris, Y. Yang, and A. Veneris, "Seamless Integration of SER in Rewiring-Based Design Space Exploration," *Proc. IEEE Int'l Test Conf.,* pp. 29.3.1-29.3.9, 2006.
[17] C.G. Zoellin, H.J. Wunderlich, I. Polian, and B. Becker, "Selective Hardening in Early Design Steps," *Proc. European Test Symp.,* pp. 185-190, 2008.
[18] S. Krishnaswamy, S.M. Plaza, I.L. Markov, and J.P. Hayes, "Signature-Based SER Analysis and Design of Logic Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 28, no. 1, pp. 74-86, Jan. 2009.
[19] 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," *Proc. Int'l Symp. Microarchitecture,* pp. 29-40, 2003.
[20] N.J. Wang, A. Mahesri, and S.J. Patel, "Examining ACE Analysis Reliability Estimates Using Fault-Injection," *ACM SIGARCH Computer Architecture News,* vol. 35, no. 2, pp. 460-469, 2007.
[21] E.W Czeck and D.P Siewiorek, "Effects of Transient Gate-Level Faults on Program Behavior," *Proc. Int'l Symp. Fault-Tolerant Computing,* pp. 236-243, 1990.
[22] K. Seongwoo and A.K. Somani, "Soft Error Sensitivity Characterization for Microprocessor Dependability Enhancement Strategy," *Proc. Int'l Conf. Dependable Systems and Networks,* pp. 416-425, 2002.
[23] A. Biswas, P. Racunas, J. Emer, and S. Mukherjee, "Computing Accurate AVFs Using ACE Analysis on Performance Models: A Rebuttal," *IEEE Computer Architecture Letters,* vol. 7, no. 1, pp. 21-24, Jan. 2008.
[24] N.J. Wang, J. Quek, T.M. Rafacz, and S.J. Patel, "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," *Proc. Int'l Conf. Dependable Systems and Networks,* pp. 61-70, 2004.
[25] L. Gwennap, "Intel's P6 Uses Decoupled Superscalar Design," *Microprocessor Report,* vol. 9, no. 2, pp. 9-15, 1995.
[26] D. Burger, T.M. Austin, and S. Bennett, "Evaluating Future Microprocessors: The Simplescalar Tool Set," Technical Report CS-TR-1996-1308, Univ. of Wisconsin, Madison, 1996.

[27] M. Maniatakos, N. Karimi, C. Tirumurti, A. Jas, and Y. Makris, "Instruction-Level Impact Analysis of Low-Level Faults in a Modern Microprocessor Controller," *IEEE Trans. Computers*, vol. 60, no. 9, pp. 1260-1273, Sept. 2011.
[28] M. Maniatakos, C. Tirumurti, A. Jas, and Y. Makris, "AVF Analysis Acceleration via Hierarchical Fault Pruning," *Proc. European Test Symp. (ETS)*, pp. 87-92, 2011.
[29] M. Maniatakos and Y. Makris, "Workload-Driven Selective Hardening of Control State Elements in Modern Microprocessors," *Proc. VLSI Test Symp.*, pp. 159-164, 2010.

Michail Maniatakos received the BS and MS degrees in computer science and embedded systems from the University of Pireaus, Greece, in 2006 and 2007, respectively, as well as the MS degree in electrical engineering from Yale University, New Haven, CT, in 2008, where he is currently working toward the PhD degree. His current research interests include test and reliability of modern microprocessors and computer architecture. He is a student member of the IEEE.

Chandrasekharan (Chandra) Tirumurti is a research scientist with the Validation and Test Solutions group at Intel Corporation based in Santa Clara, California. His current focus is on strategic manufacturing test initiatives for mainstream CPUs. An alumnus of Indian Institute of Technology, Kharagpur, India, he has wide experience in many areas of CAD and design, including simulation, data path synthesis, defect-oriented testing and fault tolerance. He has published several papers in the areas of test and fault tolerance. He mentors funded research and SRC projects actively for Intel and is an avid cricketer. He is a member of the IEEE.



Rajesh Galivanche received the MS degree in electrical and computer engineering from the University of Iowa. He is a senior principal engineer in the Technology and Manufacturing Group at Intel. As the architect for DFT and Test Technology, he sets the strategy for research and development of design-for-test and manufacturing test technologies for Intel Core-based microprocessor and atom-based consumer SoC products. He also chaired the Intel wide task force on logic fault tolerance in Intel products. In these roles, he works closely with both the academia and the EDA industry in advancing the state-of-the-art in test and fault tolerant systems. He has published several papers in IEEE conference proceedings, three patents issued, and two patent applications pending. He served as keynote speaker in many workshops in manufacturing test and online testing related workshops. He served on the program committees of IEEE VLSI Test Symposium, and IEEE International Test Conference European Test Symposium in the past. He is a senior member of the IEEE.

Yiorgos Makris received the Diploma of computer engineering and informatics from the University of Patras, Greece, in 1995, and the MS and PhD degrees in computer science and engineering from the University of California, San Diego, in 1997 and 2001, respectively. He then spent more than 10 years as a faculty of electrical engineering and of computer science at Yale University and he is currently an associate professor of electrical engineering at The University of Texas at Dallas, where he leads the Trusted and Reliable Architectures (TRELA) Research Group. His current research interests include soft-error mitigation in digital circuits, machine learning-based testing of analog/RF circuits, mitigation of hardware Trojans, as well as test and reliability of asynchronous circuits. He serves on the organizing and program committees of many conferences in the areas of test and reliability and is the program chair for the 2012 Test Technology Education Program (TTEP) of the IEEE Test Technology Technical Council (TTTC). He is a senior member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.