

# Single-Threaded Mode AVF Prediction During Redundant Execution

Blake C. Sutton and Sudhanva Gurumurthi  
Department of Computer Science  
University of Virginia  
Charlottesville, VA 22904  
{bcs8d, gurumurthi}@cs.virginia.edu

**Abstract**—Transient faults can lead to serious errors in execution. Providing protection for the processor core against these faults requires redundant execution, which leads to a performance loss. However, not all bit flips have equal impact on the processor. The Architectural Vulnerability Factor (AVF) quantifies when a soft error is likely to alter the final output and when it has little impact due to the effects of masking. Thus, redundancy is only important during periods of high AVF. Although calculating the AVF typically requires post-execution analysis of the microarchitectural behavior of a program, recent work has shown it can be estimated online. However, redundant execution changes the bits that flow through the processor, exposing bottlenecks that single-threaded execution may not display and slowing overall execution to an unpredictable degree. This variability complicates estimation of the single-threaded AVF during redundant execution, making it difficult to decide when protection is unnecessary due to low vulnerability. To leverage these low AVF periods without leaving the processor vulnerable to transient faults, we need a way to track the single-threaded AVF even when protection is enabled. Our solution is to investigate the predictability of the single-threaded AVF during redundant execution and develop predictors for the underlying AVF of three processor structures. We then evaluate these predictors in a partial RMT implementation using intelligent toggling with a sample reliability policy.

## I. INTRODUCTION

Transient faults caused by particle strikes are a serious reliability problem for current and future generations of chips. Upcoming processors are increasingly vulnerable to these unpredictable bit flips, which in turn can lead to silent data corruption (SDC) and other failures. When these faults occur, they are expensive to correct, and in some cases, catastrophic, but mechanisms to reduce faults and increase reliability all incur some performance cost.

Although some applications require a guaranteed, absolute level of protection from faults, for most consumer-market applications striking a balance between performance and reliability is the goal. A reliability solution is not attractive if the performance cost is greater than the amount saved by preventing some number of faults. In addition, the ability to tune the strength of fault protection mechanisms after design time (divorcing policy from mechanism) is desirable to save the expense of buying a new system when reliability needs or application fault-tolerance requirements shift.

The Architectural Vulnerability Factor [1] (AVF) is one way to direct the reliability aspect of this tradeoff. The AVF is a metric which accounts for the presence of error masking in modern processors and instruction streams - in other words, the chance that an occurring fault will actually cause a deviation from Architecturally Correct Execution (ACE). This metric is workload-dependent and has been shown to vary significantly over time and application phases [2] [3]. As a result, the potential exists to take advantage of periods of low vulnerability to improve performance or power without sacrificing overall reliability. For example, fault protection can be reduced or disabled completely whenever the AVF drops below a certain threshold.

Although the AVF is a useful metric for reliability tradeoffs, it is too time-consuming to precisely calculate online. Previous work has been done to estimate the AVF online by tracking the propagation of emulated error bits [4], or using linear regression to predict the AVF and drive a reliability solution which selectively toggles Redundant Multithreading (RMT) [3]. However, both approaches have a limitation - once the processor has toggled RMT and is running a redundant thread of execution, there is no way to gauge the underlying vulnerability in order to return to single-threaded mode. Disabling RMT arbitrarily risks either temporary exposure to faults while the AVF is newly estimated or wasted performance from leaving RMT on unnecessarily.

Ideally, we would like to accurately estimate the AVF with and without redundancy to make the most informed decisions. However, redundant execution complicates AVF estimation. During RMT, the amount of work remains the same while the total load on the processor increases, causing the processor to exhibit an unpredictable degree of performance degradation compared to normal execution of a workload. As a result, current AVF estimation methods must be modified in order to provide accurate prediction of the AVF during redundant execution.

In this paper, we show that it is possible to track the underlying AVF of three processor structures during RMT with models adapted for the impact of the second, redundant thread. To develop these models, we paired the single-threaded AVF of each processor structure calculated through offline simulation with a set of microarchitectural metrics gathered from running the same simulations under RMT. We then conducted a regression analysis across the space of metrics

to produce three different predictors for each structure. We measure predictor accuracy by comparing against four workloads excluded from the analysis. Finally, we present a partial RMT implementation driven by our predictors which allows more flexible and efficient tradeoffs between performance and reliability.

## II. RELATED WORK

Due to the impact of performance-enhancing instructions, speculative execution, and general code structure, not every soft error is guaranteed to lead to an observable fault which alters the end result of a computation. The Architectural Vulnerability Factor (AVF) is one method of capturing the effect of this masking on overall reliability [1]. It is a time-varying, workload-dependent quantity that is essentially the percentage of cycles that a given structure contains Architecturally Correct Execution (ACE) bits - bits which impact the final execution output. There have been several approaches to estimating the AVF in simulation offline through analysis of instruction vulnerabilities or fault injection [1] [5]. Recent work has estimated the AVF online by simulating fault injection and studying the error propagation through the instruction stream [4]. There has also been work to predict the AVF online by using regression analysis from simulation data to correlate the AVF to a set of microarchitectural metrics and form an online predictor [3]. The correlation between individual microarchitectural metrics and vulnerability phase behavior has also been studied [2]. However, this work was focused on predicting reliability phases, rather than estimating the actual value of the AVF at a given time. None of these approaches to AVF estimation consider the impact of RMT on the original AVF calculation, or how to track the single-threaded AVF over time while protection is enabled.

Several forms of partial RMT have already been proposed, aiming to mitigate the cost of RMT with regard to power [6] or performance [7] [8]. However, these approaches do not factor the effects of masking into their protection model and thus still incur equal cost at times when the AVF is low as when it is high. Our form of partial RMT seeks to exploit those effects for performance savings.

## III. AVF BEHAVIOR AND RMT

AVF calculation is only valid in single-threaded mode, since the time redundancy introduced by RMT is designed to protect only against single faults. RMT changes the observable performance of the processor - Mukherjee et al. showed that on average there is a 30% performance degradation, but this number varies greatly even across a single benchmark [9]. However, since the second thread consists completely of previously executed instructions, it should be possible to capture the differences in processor behavior introduced by RMT. Previous RMT-related research has exploited the common properties between the redundant threads to boost performance [10], [9], [8].

Our approach is to use statistical analysis to explore the relationship between the AVF and various performance metrics gathered from the processor, as in previous work [3] [2]. This

methodology is appealing because it can pinpoint where to focus in a large space of metrics and can produce a practical way to model the AVF. Specifically, we focus on relating the AVF to the measurable performance of the processor during redundant execution. To accomplish this, we calculated the AVF over regular windows of instructions, then collected microarchitectural performance metrics separately in RMT mode. We also collected the same set of metrics during single-threaded execution, to serve as a basis for comparison and provide insight into the commonalities between both modes. With this data, we then conducted separate statistical analyses between the AVF and our collected metrics for RMT and single-threaded execution.

### A. Data Collection

We chose three buffered processor structures to study - the Register Update Unit (RUU), the Issue Queue (ISQ), and the Load Store Queue (LSQ). The AVFs of all of these structures vary greatly and range from very low to very high depending on the workload, making them interesting candidates for AVF prediction. Since the simulation was done offline, we tracked 202 microarchitectural metrics, including several which are thread-specific during RMT. To determine if these thread-specific metrics were necessary for prediction, we also added the totaled versions of each partitioned metric to our set.

We calculated the AVF values in simulation and collected performance metrics over fixed windows of 10000 instructions. For each workload, we ran two separate simulations - one in single-threaded mode and one with RMT enabled for the full interval. Since our data is aggregated by instructions instead of cycles, the performance penalty of RMT has no impact on the window placement for metrics collected or AVF calculation. This is an important issue, since RMT's impact on performance is highly variable.

Our simulator is SimpleScalar 3.0, modified to support RMT and calculate the AVF. Specifically, the type of RMT we use is Simultaneous Redundant Threading (SRT), an RMT variant which leverages the Simultaneous Multithreading (SMT) microarchitecture and features several performance enhancements [10]. As input to our simulations, we took multiple Simpoints from all 26 benchmarks of the SPEC2000 suite. We simulate an eight-wide issue SRT-based processor with 64 KB level-1 instruction- and data-caches, each of which are 4-way set-associative with a 32-byte line size, and a 512 KB unified level-2 cache that is also 4-way set associative with a 64-byte line size.

### B. Regression Analysis

To study AVF predictability in RMT mode, we conducted a regression analysis between the single-threaded AVF and our two sets of microarchitectural metrics (single-threaded and redundantly-threaded). We included the single-threaded metrics in our analysis to explore the possibility that a common metric set exists to predict the AVF in both modes. This would allow reduced implementation cost for the monitoring architecture, since the same metrics for each structure would be tracked with or without redundancy. To avoid over-fitting

the data, we excluded four workloads from the regression analysis (apsi, galgel, mcf, and twolf), in order to use them to evaluate the predictors.

First, we studied the impact of window size by upsampling gradually from 10,000 up to 10 million instructions. We use intervals of 1 million instructions in the remainder of our analysis, as it offers a compromise between the high amounts of noise apparent at low window sizes and the decreased responsiveness to program behavior at very high window sizes.

Next, we used least-squares regression individually against each of our 202 microarchitectural metrics and calculated correlation coefficients. We then selected the metric (or metrics, for those with very similar correlations) with the highest correlation for use in a single-variable predictor. This initial analysis showed that although some metrics stood out with clear correlations to the AVF, many seemed to have very similar correlations for no intuitive reason. Studying subgroups of the training set separately indicated that some metrics correlated strongly to specific workloads while correlating very weakly to others, leading to overall correlations which did not reflect the usefulness of a metric in the predictor. To narrow down the set under analysis, we excluded those metrics most specific to microarchitectural details - for example, the cache hierarchy and branch predictor metrics.

With the culled set of metrics, we developed more accurate predictors by adding terms with multiple linear regression. To do this, we selected the metric with the single highest correlation as the first term of the equation, then continued to use multiple linear regression individually against each other metric to determine the combined correlation coefficient for each model. When given a choice between multiple metrics with similar correlations, we preferred those with some intuitive relationship to the AVF, such as the occupancy of the fetch queue. For all structures except the LSQ, we stopped the multiple regression process at 3 terms. In most cases a plot of correlation coefficients against the number of metrics revealed a clear “knee” after which the gain in correlation from additional metrics was greatly decreased. In the case of the LSQ, the initial correlation was low enough that using 4 terms provided significantly greater accuracy over 3.

We developed two multivariable models per structure for the single-threaded and RMT modes: the most accurate model for each mode and a model using common metrics for both modes. To evaluate our predictors, we considered the overall correlation coefficient - the percentage of variation in the AVF explainable by the metrics used. To observe the accuracy of our predictors when applied to previously unseen data, we calculated the Root Mean Square (RMS) error between the predicted and actual AVF for the 4 benchmarks excluded from analysis.

#### IV. RESULTS

In this section, we discuss our models and evaluate their accuracy. We also compare the RMT predictors to the accuracy of the single-threaded predictors and discuss the tradeoffs involved in using two models with common metrics for single-threaded and RMT prediction.

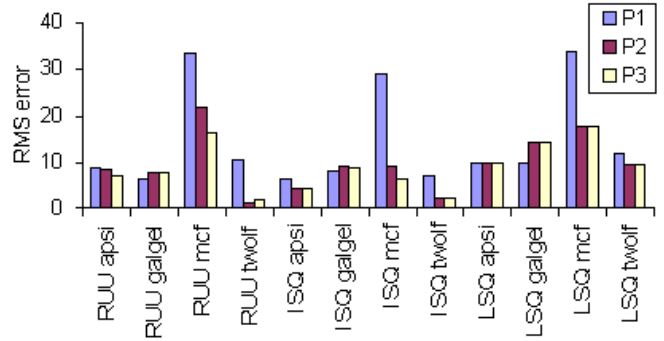


Fig. 1: RMS error for the three predictors of the AVF in single-threaded mode.

As detailed in the previous section, we developed 3 predictors each for the AVF of the RUU, ISQ, and LSQ. These models and a description of the variables tracked are shown in Tables 1 and 2. The first model, P1, consists only of the single most highly correlated metric for each structure, with an average  $R^2$  value of across all structures of 0.65. Our second model, P2, is the result of multiple linear regression of multiple metrics, where an effort was made to find the common set of metrics that is most accurate in both the single-threaded and RMT mode predictor, and has an average  $R^2$  value of 0.77. Finally, P3 is the most accurate model, which does not guarantee that the same metrics are used in each model, and has an average  $R^2$  value of 0.79. The  $R^2$  value for a predictor measures the amount of variation in the AVF explainable by our metrics - thus, even our simplest model, P1, is relatively sensitive to the AVF variation even while running a redundant thread.

Figure 1 shows the RMS error of the RMT mode predictors for apsi, galgel, mcf, and twolf, the benchmarks excluded from regression analysis. Figure 2 shows the same calculation for the single-threaded models. As in previous work [3], the overall correlation and accuracy of prediction from the single-threaded metrics is relatively good. However, the RMT mode generally shows a higher RMS error and lower overall correlation. This is due to the addition of the redundant thread, which introduces bottlenecks and other irregularities for some workloads that did not exist in the single-threaded execution. We are currently studying this behavior more carefully to understand the reasons behind it. One consideration is that the memory hierarchy of the SRT processor is outside the Sphere of Replication (SOR), and thus unaffected by performance slowdowns from RMT. Since in SRT the LVQ buffers values that the leading thread reads from memory to the redundant thread, if an application’s performance is driven (even temporarily) by memory delays it is likely to have an impact that is not captured by our analysis. Thus memory latency has the potential to completely hide the performance cost of the redundant thread.

It is important to note that although P3 tends to be more accurate than the common model P2, and both are usually more accurate than P1, there is no guarantee of a lower RMS error for any single predictor across all benchmarks

Var.	Description
$R_o$	RUU occupancy
$L_o$	LSQ occupancy
$I$	Total instructions executed (speculative and committed)
$I_1$	Total instructions executed by trailing thread
$B$	Total branches executed (speculative and committed)
$L$	Total loads executed (speculative and committed)
$I_o$	IFQ occupancy, thread 0
$ASC$	Average number of slip cycles

Pred.	Struct.	Equation - for Single-Threaded Mode	$R^2$
$P_1$	RUU	$-2.35 + (.68R_o)$	0.7955
	ISQ	$-2.08 + (1.37L_o)$	0.85
	LSQ	$4.35 + (1.17L_o)$	0.57
$P_2$	RUU	$22.59 + (0.57R_o) - (8.20e^{-5}B) - (2.60e^{-5}L)$	0.85
	ISQ	$23.19 + (1.23L_o) - (4.11e^{-5}B) - (1.52e^{-5}I)$	0.90
	LSQ	$1.42 + (3.00I_o) + (0.05IPB) - (3.00CPI)$	0.68
$P_3$	RUU	$39.12 + (0.62R_o) - (1.04e^{-5}L - (2.97e^{-5}I)$	0.83
	ISQ	$29.47 - (1.29L_o) - (2.59e^{-5}I)$	0.89
	LSQ	$-.83 + (0.30R_o) + (0.043IPB) - (2.73CPI) + (0.51L_o)$	0.75

Pred.	Struct.	Equation - for RMT Mode	$R^2$
$P_1$	RUU	$7.18 + (5.31I_o)$	0.66
	ISQ	$0.70 + (2.63L_o)$	0.74
	LSQ	$6.20 + (2.98L_o)$	0.53
$P_2$	RUU	$7.86 + (1.06R_o) + (7.55e^{-6}B) - (4.77e^{-5}L)$	0.71
	ISQ	$-23.31 + (2.69L_o) - (7.42e^{-5}B) + (1.84e^{-5}I)$	0.90
	LSQ	$4.67 + (4.05I_o) + (0.05IPB) - (3.10CPI)$	0.69
$P_3$	RUU	$33.73 + (1.28R_o) + (1.10e^{-4}I_1) - (7.16e^{-5}I)$	0.79
	ISQ	$35.95 + (2.69L_o) - (7.42e^{-5}I_1) + (1.84e^{-5}I)$	0.86
	LSQ	$11.48 + (0.47I_o) + (3.65I_o) + (0.05IPB) + (16.73e^{-6}CPI) + (0.16ASC)$	0.71

Table 1: Description of the variables used in Table 2.

Table 2: Predictors for single-threaded and RMT mode from our regression analysis for RUU, LSQ, and ISQ.  $P_1$  is our single-variable linear predictor model,  $P_2$  is our model sharing all terms with RMT mode model, and  $P_3$  is our hybrid model sharing all terms but one.

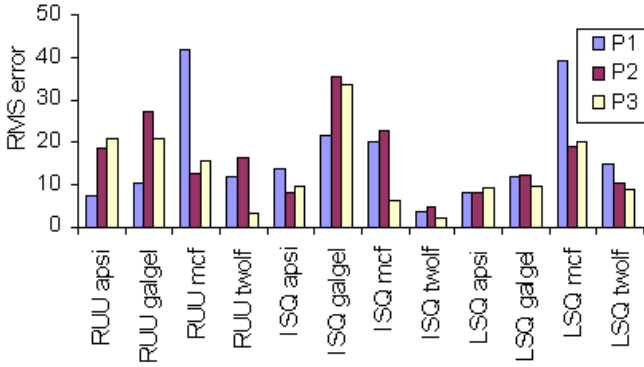


Fig. 2: RMS error for the three predictors of the AVF from RMT mode.

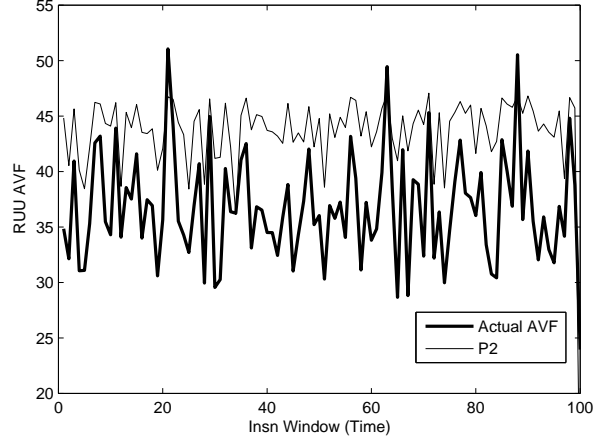


Fig. 3: Variation in the single-threaded AVF and its corresponding prediction during RMT.

and structures. For example, the mcf benchmark seems to be particularly problematic to predict in both single-threaded and RMT mode, showing high error levels for both the LSQ and RUU. This indicates that there is some aspect of application behavior that is not captured by the metrics used in the predictors. For example, we find that the average miss-rate in the L1 data-cache across all the Simpoints for mcf is 31.3% which is significantly higher than the miss-rate for the other benchmarks. As mentioned previously, since the L1 data-cache lies outside the Sphere of Replication and we do not use microarchitectural metrics related to the memory hierarchy to build the predictors, our methodology needs to be calibrated to target workloads such as mcf to predict AVF with greater accuracy.

Figure 3 shows another view of predictor accuracy, where the predicted AVF in RMT mode is plotted against the calculated AVF for the first Simpoint of mcf. Although the prediction is never guaranteed to be lower or higher than the

actual value, it is clear that the prediction closely follows the variation of the actual AVF, indicating a relative level of accuracy and responsiveness.

## V. APPLICATION: INTELLIGENT RMT TOGGING

To show the utility of RMT prediction, we present a partial RMT implementation which allows a more flexible compromise between performance and reliability during program execution. RMT toggling provides performance savings by only enabling protection from faults for certain periods, which are determined by a user-specified reliability policy. The toggling is driven by the AVF predictors from the previous section in single-threaded and RMT mode, which could be calculated using a weighted sum from a set of hardware performance counters.

We evaluate the benefit of RMT prediction to toggling in terms of performance and reliability by comparing against three cases: the case where RMT is never turned on, the case where RMT is always turned on, and a periodic toggling implementation which tracks the single-threaded AVF and toggles off automatically after 10 million cycles. For toggling, we chose to enable RMT whenever the maximum of the AVFs of the three structures exceeds a given threshold, and disable it once the collective AVF is below the same threshold. Of course, the space of possible policies is rich and our choice is only one example.

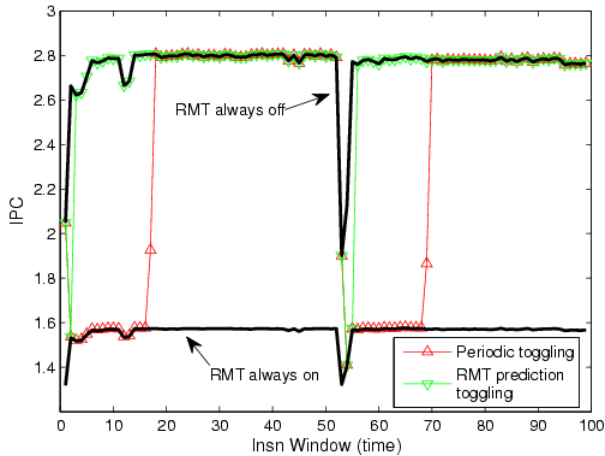


Fig. 4: Comparison of IPC over time for twolf. The two curves representing no RMT and all RMT bracket the toggling curves. The AVF threshold is 25.

To illustrate the benefits of predicting AVFs during RMT mode, Figure 4 shows the IPC of the twolf benchmark over time for all four choices, with an AVF threshold of 25 for toggling. Both toggling implementations are bracketed by the maximum IPC (no RMT) and the minimum IPC (complete RMT). However, RMT prediction shows better overall performance than periodic toggling, since the latter must remain on for at least 10 million cycles, while the RMT prediction-based toggling can make more fine-grained decisions. For this example, RMT prediction leads to an overall IPC savings of 13% compared to periodic toggling, and 54% compared to full RMT.

RMT prediction-based toggling also provides better protection than periodic toggling. Since periodic toggling uses no estimate of the underlying AVF, after RMT is toggled off the processor can run unprotected for one data collection period before the single-threaded predictor can re-enable protection. RMT prediction-based toggling remedies this issue. Figure 5 demonstrates vulnerable periods which are eliminated by RMT prediction-based toggling. Here, the AVF threshold is set to 15 to highlight the toggling behavior if the AVF curve never falls below the threshold. The points when the AVF is zero represent periods when RMT is enabled, since all of the instructions executed are protected by redundancy, and the spikes above the threshold represent points where the processor has mistakenly disabled RMT. Thus, with periodic toggling the processor is vulnerable at 5 points and violates the

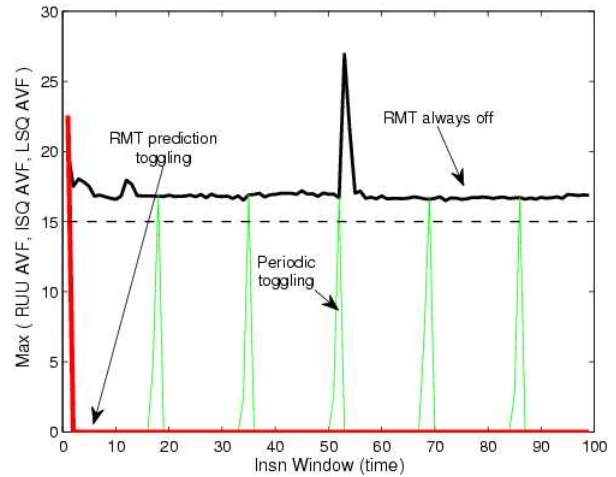


Fig. 5: Vulnerability comparison for twolf: completely disabled RMT, periodic toggling, and RMT prediction toggling. In this figure the AVF threshold is 15 (see dashed line).

reliability policy, while with RMT prediction-based toggling the processor is never exposed.

## VI. CONCLUSIONS

In this paper, we address an important open problem in runtime AVF prediction – estimating the single-threaded mode AVF when the processor is operating in redundant mode. Knowing the single-threaded mode AVF is important in formulating policies that make informed decisions about when to disable redundant execution so that one could optimize for performance or power while still meeting reliability goals. We show that it is possible to design regression-based predictors that use microarchitectural metrics measured during redundant execution to accurately estimate the runtime variations in the AVF if the processor were running in single-threaded mode. We present and evaluate three such predictors. We finally demonstrate how we could use these predictors with a partial RMT scheme that intelligently toggles redundant execution based on AVF variations in the single-threaded mode.

## VII. ACKNOWLEDGEMENTS

This research has been supported in part by NSF CAREER Award 0643925 and gifts from Intel.

## REFERENCES

- [1] 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*. IEEE Computer Society, 2003, p. 29.
- [2] X. Fu, J. Poe, T. Li, and J. A. B. Fortes, "Characterizing microarchitecture soft error vulnerability phase behavior," in *MASCOTS '06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*. IEEE Computer Society, 2006, pp. 147–155.
- [3] 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*. ACM, 2007, pp. 516–527.
- [4] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "Online estimation of architectural vulnerability factor for soft errors," *SIGARCH Comput. Archit. News*, vol. 36, no. 3, pp. 341–352, 2008.

- [5] 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*. IEEE Computer Society, 2005, pp. 496–505.
- [6] M. W. Rashid, E. J. Tan, M. C. Huang, and D. H. Albonese, "Exploiting coarse-grain verification parallelism for power-efficient fault tolerance," in *PACT '05: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*. IEEE Computer Society, 2005, pp. 315–328.
- [7] S. Kumar and A. Aggarwal, "Reducing resource redundancy for concurrent error detection techniques in high performance microprocessors," *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pp. 212–221, Feb. 2006.
- [8] A. Parashar, S. Gurumurthi, and A. Sivasubramaniam, "A complexity-effective approach to alu bandwidth enhancement for instruction-level temporal redundancy," *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pp. 376–386, June 2004.
- [9] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt, "Detailed design and evaluation of redundant multithreading alternatives," in *ISCA '02: Proceedings of the 29th annual international symposium on Computer architecture*. IEEE Computer Society, 2002, pp. 99–110.
- [10] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*. ACM, 2000, pp. 25–36.