Dynamic Prediction of Architectural Vulnerability from Microarchitectural State

Kristen R. Walcott
walcott@cs.virginia.edu

Greg Humphreys
humper@cs.virginia.edu

Sudhanva Gurumurthi
gurumurthi@cs.virginia.edu

Department of Computer Science
University of Virginia
Charlottesville, VA 22904

Abstract

Transient faults due to particle strikes are a key challenge in microprocessor design. Driven by exponentially increasing transistor counts, per-chip faults are a growing burden. To protect against soft errors, redundancy techniques such as redundant multithreading (RMT) are often used. However, these techniques assume that the probability that a structural fault will result in a soft error (i.e., the Architectural Vulnerability Factor (AVF)) is 100 percent, unnecessarily draining processor resources. Due to the high cost of redundancy, there have been efforts to throttle RMT at runtime. To date, these methods have not incorporated an AVF model and therefore tend to be ad hoc. Unfortunately, computing the AVF of complex microprocessor structures (e.g., the ISQ) can be quite involved.

To provide probabilistic guarantees about fault tolerance, we have created a rigorous characterization of AVF behavior that can be easily implemented in hardware. We experimentally demonstrate AVF variability within and across the SPEC2000 benchmarks and identify strong correlations between structural AVF values and a small set of processor metrics. Using these simple indicators as predictors, we create a proof-of-concept RMT implementation that demonstrates that AVF prediction can be used to maintain a low fault tolerance level without significant performance impact.

Categories and Subject Descriptors:
C.1.0 [Processor Architectures]: General

General Terms: Reliability.

Keywords: Architecture Vulnerability Factor, Redundant Multithreading, Performance, Reliability, Microarchitecture.

1. INTRODUCTION

Transient faults have emerged as a key challenge in microprocessor design. Lowered supply voltages and increasing clock frequencies make processors especially vulnerable to transient faults from external charged particles, such as high-energy neutrons from terrestrial cosmic rays or alpha particles from chip packaging [35]. These faults can lead to spurious bit flips, which may propagate to the architected state and lead to incorrect program execution. Although traditionally considered a problem only for large memory structures (e.g., caches and main memory), transient faults are becoming a serious problem even for the latches and combinational logic within the processor core [28].

Techniques for protecting against faults in memory structures include the use of parity and error-correcting codes [4]. These techniques fit well within the overall design of a memory system but are poorly suited for use within the processor core due to their area, performance, and power overheads. Instead, processor core protection is generally provided via some form of redundancy. A number of redundancy techniques have been proposed to detect [15, 22, 25, 30] and recover from [8, 33] transient faults. These schemes, which are collectively known as *Redundant Multithreading (RMT)* techniques, leverage the hardware thread contexts provided by a simultaneous multithreading (SMT) or multi-core processor. In RMT, two identical program threads are generated in the microarchitecture and then are allowed to execute independently for some time frame. The outputs of the threads are compared to verify correctness. Although these schemes are effective in providing protection against transient faults, they impose significant performance overheads. Mukherjee et al. showed that, on average, performance is degraded by 30% on single-threaded workloads and 32% on multithreaded workloads [15].

RMT provides fault detection throughout the entire lifetime of a process, so no transient errors will go undetected. However, this level of protection is not needed at all points in a process's lifecycle. Although the error rate of any one processor structure is a function of several unchanging quantities such as the underlying process technology, circuit design style, and the number of bits comprising the structure, it is also dependent on an application-dependent factor known as the *Architectural Vulnerability Factor* (AVF) [16]. The AVF is a time-varying quantity that measures the current probability that an internal fault in the structure would result in an externally visible error. This means that if the AVF of a structure is low, redundancy may not be required to meet an application's fault-tolerance requirements. Using RMT in such a scenario is therefore unnecessary and wastes valuable processor resources.

Counteracting this performance degradation has been the topic of much recent research [9, 13, 18, 19, 21, 31]. How-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISCA'07, June 9–13, 2007, San Diego, California, USA.
Copyright 2007 ACM 978-1-59593-706-3/07/0006 ...$5.00.

ever, none of these techniques takes AVF into consideration or gives any other probabilistic guarantees about fault tolerance, even though error protection is required only at those times when a structure is vulnerable. When designing a microprocessor, it would be prudent to attempt to determine dynamically which bits affect the final system output and which do not. For example, a bit flip in a branch predictor table may lead to a misprediction, but the execution would still be semantically correct. This suggests that the processor should track the AVF of its key structures and incorporate those measurements into a dynamic model of RMT necessity. Unfortunately, tracking the instantaneous AVF is extremely costly, so AVF calculations are performed only in postmortem analysis, not on the fly.

In this paper, we create a rigorous statistical characterization of AVF values that can be used to estimate vulnerability at any point in program execution. We also show that there is considerable variation in AVF across different applications for various processor structures, justifying the need for such an analysis. Our vulnerability estimator can be represented as a low-dimensional function over commonly tracked performance metrics and could therefore be computed easily in current hardware with little modification.

In summary, the principal contributions of this paper are:

- A demonstration that the AVF of multiple key structures within a processor shows significant runtime variations for the benchmarks of the SPEC2000 suite, both between separate benchmarks and also over time within a single benchmark. Because of this variability, the need for a dynamic characterization that can be evaluated efficiently at any point in program execution is established.

- A rigorous characterization of AVF behavior based on a thorough statistical analysis. Our analysis identifies the key measurable indicators of the AVF at any instant and determines the correlations between these performance metrics and AVF values. To further confirm the usefulness of this predictive model, we describe a proof-of-concept RMT implementation that enables redundancy only when the vulnerability of multiple key processor structures rises above a threshold. This approach helps mitigate the redundancy overhead when vulnerability is low. For the `twolf` benchmark in the SPEC2000 suite, we achieve IPC improvements from 14% to 55% depending on the level of redundancy being guaranteed.

- An *a posteriori* analysis of our AVF predictor using benchmarks not included in our training set serves to quantitatively validate our statistical approach. We perform an RMS error analysis of three proposed prediction approaches, and also show that all predictors can capture even subtle time-varying AVF behavior. Our AVF predictors achieve correlation coefficients in excess of 0.93, meaning that over 93% of the variation in the AVF measurements can be explained by the predictor. Our methodology can be easily adapted to any new processor with different characteristics or additional processor structures.

The remainder of this paper is organized as follows. In Section 2, we discuss background and related work regarding architectural vulnerability factors and redundant multithreading. We then analyze trends in dynamic AVF behavior in Section 3. We use this data in Section 4 to develop a formalization describing instantaneous AVF behavior and to quantitatively demonstrate the success of our approach. In Section 5, we demonstrate that our predictors can be used to adaptively enable and disable redundant multithreading based on the current processor vulnerability, thereby achieving significant speedups. Finally, in Section 6, we conclude and suggest directions for future research in this area.

2. BACKGROUND AND RELATED WORK

Microprocessors, especially those used for mission critical applications with stringent fault tolerance requirements, can be rendered useless without adequate protection from soft errors. Care must be taken at all stages of processor design to evaluate the appropriate amount of protection necessary for the processor's target application. In this section, we review relevant work in transient fault tolerance techniques and the *Architectural Vulnerability Factor* (AVF), a quantitative measure of the likelihood that a soft error will cause incorrect execution.

2.1 The Architectural Vulnerability Factor

The transient or *Soft Error Rate* (SER) of a chip is recursively defined as the summation of the SER of each microarchitectural structure (SER_i), given by the equation [16]:

$$SER_i = (\#B)\left(\frac{\text{FIT}}{\text{Bit}}\right)(\text{AVF})(\text{TVF}).$$

In this equation, $\#B$ is the number of bits in the structure. The intrinsic *Failures in Time* (FIT) Rate, $\frac{\text{FIT}}{\text{Bit}}$, is determined by two main factors: (i) the neutron or alpha particle flux and (ii) properties of the circuit, primarily the critical charge. The *Timing Vulnerability Factor* (TVF) encapsulates the fraction of each cycle that a bit is vulnerable. As in prior work [16], we assume that the TVF is factored into the raw device FIT rate.

The final term in the equation is the AVF, the probability that a fault in a structure will result in an architecturally visible incorrect execution [2, 16]. The AVF of a bit is the fraction of time that it is in a state of *Architecturally Correct Execution* (ACE) [16]. In an ACE state, a change to the bit's value would affect the program's final outcome, whereas a change to a non-ACE bit would not propagate through program execution. For example, a flipped bit within a dynamically dead instruction, a branch predictor table, or a wrong-path instruction would not lead to architecturally incorrect execution. Hence, the bit is in a non-ACE state. As non-ACE bits will lower the AVF of the processor, any intervals that cannot be proven to be non-ACE are assumed to be ACE for the purpose of providing a conservative upper bound on the AVF.

Little's Law gives an estimate of a structure's AVF as:

$$\text{AVF} = \frac{(\text{B}_{\text{ACE}})\,(\text{L}_{\text{ACE}})}{\#B}, \tag{1}$$

where B_{ACE} is the average bandwidth of the ACE bits into the structure and L_{ACE} is the average residence time of ACE bits in the structure [16]. This formalization provides a good estimate, but because not all transient faults lead to architecturally incorrect execution, it also requires classifying all bits at all points of execution as ACE or non-ACE

and then tracking all ACE bit bandwidth and residency values. Clearly, calculating this estimate is not an easy task even in simulation, and it is clearly impractical to track in actual hardware.

2.2 Redundancy Techniques

Because AVF estimation is so difficult, existing techniques to provide protection against transient faults in microprocessors generally assume that the AVF is always 100%. Space and time redundancy are then used to guard against faults.

In hardware replication, multiple identical components are operated in lockstep to ensure that, both components always produce identical outputs when given the same inputs [34]. These ideas have enjoyed commercial success; high-end servers such as the HP NonStop Architecture [1] and the IBM G5 [29] have used space redundancy techniques. However, because system resources must be statically partitioned to allow for redundancy, full replication of all components significantly increases cost [22].

A less expensive technique used in fault-tolerant systems that combines space and time redundancy is Redundant Multithreading (RMT). RMT leverages the multiple execution contexts provided by SMT and multi-core processors to implement redundant execution. First, multiple threads of a program are created at the *input replication* point and are allowed to execute independently. The duplicate, or trailing, thread runs with a temporal lag (as with a time-redundant approach) and it may use different functional units, occupy different issue queue slots, etc., than the original thread, thereby employing a certain amount of space redundancy as well. At the *output comparison* point, a voting operation detects any error before any process state is modified. Thus, fault tolerant execution is provided between the input replication and output comparison points. This protected region is referred to as the *Sphere of Replication* (SoR) [22]. The structures outside the SoR are assumed to be protected via other means.

While performing the same function as replicated hardware components, RMT processors are preferred because (i) they may require less hardware when space redundancy is not critical, and (ii) they potentially provide better performance by dynamically partitioning resources [22]. Thus, redundant multithreading has become a popular transient fault tolerance technique [15, 22, 25, 30]. Most proposals advocate the creation of redundant threads in the microarchitecture, but there is also work on software [23] and combined hardware/software approaches [24] to implement RMT.

One significant drawback to RMT is that creating multiple threads for a single program leads to greater contention for shared processor resources, thereby degrading single-thread performance and increasing power consumption. A number of solutions have been proposed to mitigate these problems. To reduce performance degradation, efforts have been made to exploit value locality to non-speculatively remove instructions from redundant threads [9, 18], to speculatively remove instructions [19, 21], and to share processor resources more efficiently between the threads [13, 31]. There has also been work that investigates reducing the power overhead of RMT [14, 20]. While these techniques have been effective in reducing power consumption and increasing performance, each still builds on the fundamental underlying assumption that the microprocessor is vulnerable 100% of the time.

2.3 Bridging the Gap

Because AVF estimation is so difficult at runtime, even the aforementioned "partial" RMT techniques waste valuable resources by providing redundancy when vulnerability is low or does not exist. One of the primary goals of this research is to find a way to remedy this waste by allowing RMT implementations to model and predict AVF at runtime. Little's Law (Equation 1) suggests that a practical runtime AVF measurement might be possible because there exists some relationship between structure utilization and AVF. In this paper, we compare measured runtime AVF behavior to combinations of other microprocessor performance metrics in order to determine a rigorous, empirical characterization of dynamic AVF behavior.

Very recently, Fu et al. investigated the nature of dynamic AVF behavior [7]. In this paper, they demonstrate convincingly that microarchitecture structures exhibit significant variability over time in their reliability characteristics. Using this knowledge, they attempt to find a correlation between AVF and a host of individual time-varying performance metrics, with the objective of classifying program reliability phase behavior. Their approach is an important first step towards efficient fault tolerance in modern processors. The processor metrics that they chose to focus on were limited in number and exhibited inconsistent correlation to AVF across benchmarks. Because of their limited set of metrics, they concluded that a single-variable correlation does not exist and abandoned a predictive approach. Our work reexamines this choice by exploring multivariate statistical relationships between AVF and a wide variety of easily measurable time-varying performance metrics. Our use of multiple simultaneous metrics allows us to infer AVF values independent of phase behavior. We are able to predict structural AVF with very high accuracy by using an empirical, data-driven, nonlinear model of multiple variables.

3. CHARACTERIZATION OF DYNAMIC AVF BEHAVIOR

In this section, we show the results of experiments that demonstrate the time-varying AVF behavior that we seek to model and exploit in our research. We establish that the AVF changes significantly over time, justifying our search for an AVF predictor. Also, we visualize a few of the trends between microarchitectural measurements and AVF values. This simple examination provides evidence that correlations exist and motivates a more rigorous analysis of the trends.

3.1 Experimental Setup

All of our experiments were conducted using the SimpleScalar 3.0 simulator [3], appropriately modified to compute the AVF and implement redundant multithreading. In particular, we compute the AVFs of the Issue Queue (ISQ), the combined Reorder Buffer and Physical Register File (RUU), and the Load/Store Queue (LSQ).

The design space for RMT implementations is quite large. For example, implementations will differ depending on which pipeline stages perform input/output replication, whether both fault detection and recovery are required, or whether all redundant threads need to be identical. The baseline RMT scheme that we use in this paper is based on the Simultaneous and Redundantly Threaded (SRT) design [22], proposed for transient fault detection in SMT processors. The

Fetch/Decode/Issue/Commit Width	8
Fetch-Queue Size	16
Branch-Predictor Type	Bimodal Table-size of 2K entries
RAS Size	64
BTB Size	2K-entry 4-way
Branch-Misprediction Latency	7 cycles
RUU Size	128
LSQ Size	64
Integer ALUs	6 (1-cycle latency)
Integer Multipliers/Dividers	2 (3,20)
FP ALUs	4 (2)
FP Mult./Div./Sqrt.	2
L1 Cache Ports	4
L1 D-Cache	64KB, 4-way with 32B line-size (2)
L1 I-Cache	64KB, 4-way with 32B line-size (2)
L2 Unified Cache	512 KB, 4-way with 64B line-size (12)
I-TLB	128 entries 4-way set-associative
D-TLB	256 entries 4-way set-associative
TLB Miss-Latency	30 cycles
Main Memory Latency	200 cycles

Table 1: Processor configuration setup for our experiments. Numbers in parentheses are latencies.

input replication and output comparison points for SRT are the L1 I-cache and D-cache interfaces respectively. SRT uses additional microarchitectural structures within the processor core to create a temporal lag between redundant thread pairs (for performance and time redundancy purposes) and also to ensure precise interrupts, correct handling of cached loads and uncached I/O.

For the evaluations, we use all 26 benchmarks from the SPEC2000 suite [32]. Each benchmark is run for multiple 100-million instruction SimPoints [27] based on the data given on the SimPoint website [17]. These simulation points, which are independent from the underlying architecture, are representative of the full program's execution when varying the architecture parameters. Within each SimPoint interval, we checkpoint the entire microarchitectural state every 4 million instructions and calculate the AVF values for the RUU, LSQ, and ISQ on that state. After computing the AVFs, the simulation resumes from the checkpoint. Therefore, for each simulated SimPoint, we obtain a total of 25 "snapshots" of the AVFs and other performance related information such as IPC, branch prediction statistics, and various aggregate queue occupancies. The precise processor configuration used in this study is given in Table 1.

3.2 Time-varying Microarchitectural Behavior in SPEC2000

Figure 1 shows runtime behavior over the first two SimPoints of three benchmarks from the SPEC2000 suite. These data represent only a fraction of all data gathered for this research; other benchmarks and SimPoints behave similarly and are omitted here for brevity and clarity. The first row of graphs shows the IPC trends, the second shows the RUU occupancy statistics, and the next three rows give the variation in the ISQ, RUU, and LSQ AVFs respectively. In the following, unless specifically stated otherwise, we will restrict our discussion to the first SimPoint.

3.2.1 IPC and AVF Variation

As an example performance metric, we will consider IPC trends. IPC is often used as a predictor of other time-varying metrics [5], so one might assume that IPC and AVF are closely linked. Intuitively, high IPC would reduce an ACE bit's residence time in a given microarchitecture structure, thus reducing the structure's AVF.

Within the first Simpoint interval, we are able to observe significant variation in program behavior, as desired. The variation is large for `bzip2`, whose IPC value varies in the range $[1.5, 2.0]$, and is slightly less pronounced for `perlbmk`. The IPC for `art` is relatively constant and is very low for both SimPoints. These IPC trends are indicative of the variations in the throughput of instructions through the pipeline. As a result of the throughput variations, the bits associated with these instructions should also show variations in their bandwidth and residence time characteristics for all microarchitectural structures that they occupy, thereby having an impact on those structures' AVF. The net impact of these microarchitectural variations can be seen in the AVF graphs in Figure 1.

As expected, the AVF graphs also exhibit large temporal variation. For example, for `bzip2`, the AVFs of the RUU and LSQ vary from nearly 31–67% and 20–60% respectively. Even within a single benchmark, the extent to which the AVF varies can differ significantly from one structure to another. For example, when we look at the profiles for the ISQ and RUU AVF for `bzip2` (Figures 1(h) and (k)), we see that the AVF of the RUU changes by a much larger magnitude than that of the ISQ. Similarly, the range of LSQ AVF values in `art` (Figure 1(o)) is much higher (48-59%) compared to the other two AVFs, which show less than 3% variation.

3.2.2 Prediction with Little's Law

AVF calculation by means of Little's Law as presented in Equation 1 takes into account the residency, L_{ACE}, and bandwith, B_{ACE}, of ACE bits. In our example benchmarks, we find that the average residence time of instructions in the LSQ for the second SimPoint are 18, 33, and 44 cycles for `perlbmk`, `bzip2`, and `art` respectively. Correspondingly, we find that the AVF of the LSQ for the `perlbmk` benchmark varies over a lower numerical range (14-37%, as shown in Figure 1(m)), compared to `art` (48-60%). However, if we

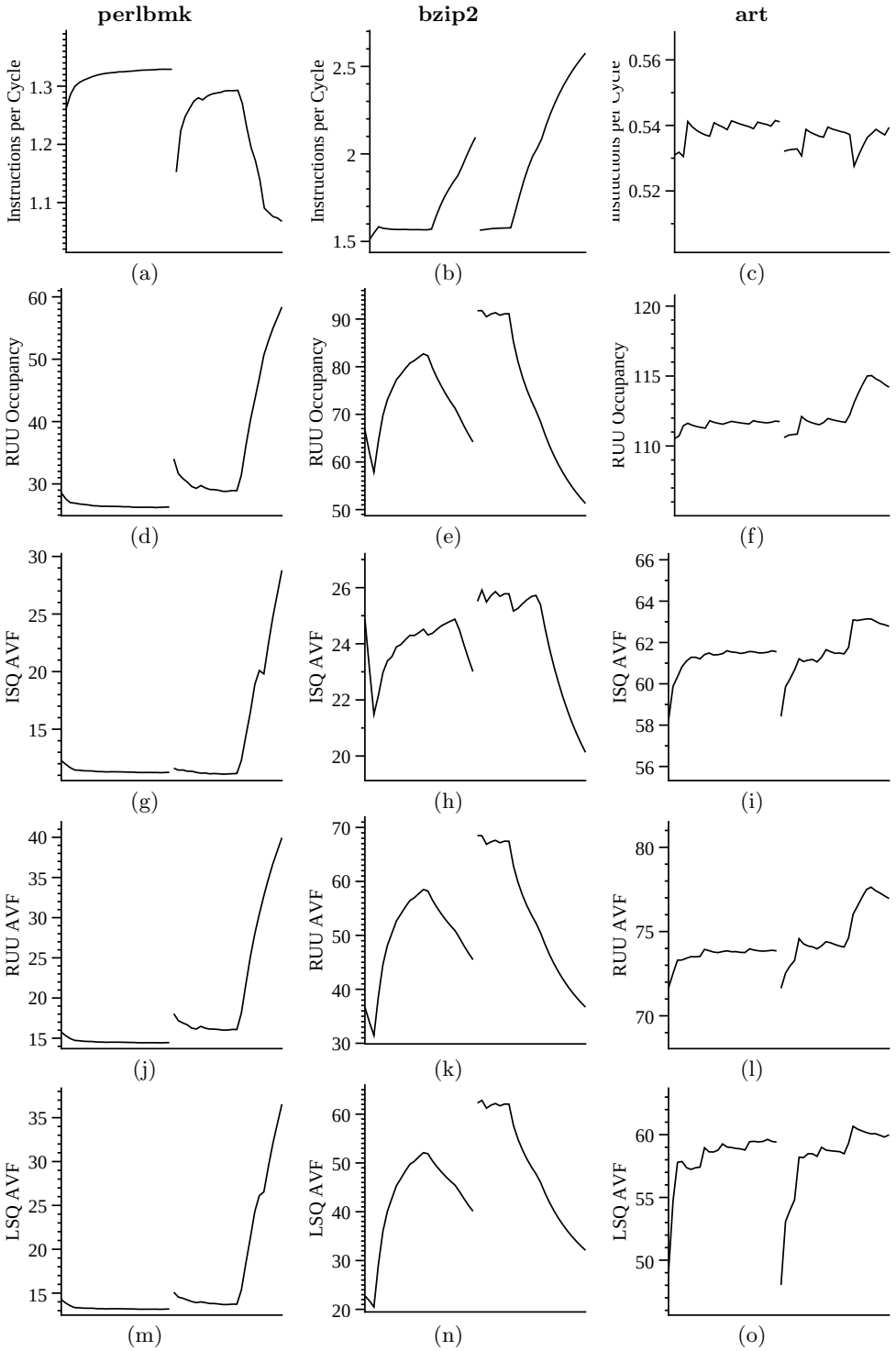

Figure 1: Time-Varying Metrics and AVF Behavior. All the graphs present data for the first two SimPoints of 100 million instructions of three different SPEC2000 benchmarks (the graphs are discontinuous between SimPoints). The x axes all denote time; values are gathered at each of 50 checkpoints. The key observation is that while AVF can clearly vary significantly over time, its relationship to IPC and RUU occupancy can actually invert from benchmark to benchmark, implying the need for a more sophisticated prediction mechanism.

look at the LSQ AVF graphs for `bzip2` and `art`, we can see that values for the former benchmark are higher, although the residence time of instructions in the LSQ are lower. One reason for this inversion is that the IPC of `bzip2` is significantly higher than that of `art`, so the correspondingly larger B_{ACE} term can overwhelm the L_{ACE} value, leading to a higher AVF.

While the benchmark data fits the AVF characterization described by Little's Law, calculating and properly weighing the the residency and bandwidth of every bit is extremely difficult. At a high level, there are many complex microarchitectural interactions that combine to produce the AVF value. Therefore, one of the main goals of this paper is to use empirically gathered data to abstract away this complexity and to develop a data-driven approach to accurately infer the AVF at runtime.

3.2.3 Detecting Correlations

One way by which we can abstract microarchitectural complexity is through correlation detection. By visual inspection alone, we can see that relationships may exist between AVF behavior and microarchitecture metrics. For example, in `art`, the AVF curves bear a strong resemblance to the IPC profile. However, this relationship is not consistent across all benchmarks. Although the IPC curves for all three benchmarks appear similar, the AVF curves for `perlbmk` and `bzip2` show an inverted characteristic, while the ones for `art` do not. On the other hand, the shape of the curves representing RUU occupancy also match those of the AVF values, presenting RUU occupancy as a strong candidate for correlation.

These trends suggest that an easily measurable metric, such as IPC or RUU occupancy could be used to calculate the AVF values. However, discerning the precise nature of this relationship requires analyzing a much larger space of processor events that could potentially affect the AVF.

3.2.4 Our Approach

It is clear that there are temporal variations in the AVF behavior of a benchmark, even within a single SimPoint phase. Therefore, it would be beneficial to design redundant multithreading schemes that can adapt to this variation. Furthermore, a qualitative inspection of the data suggests that a relationship exists between AVF and measurable events in the processor, but a thorough analysis must be performed in order to create a model that will hold across all workloads. In our simulation infrastructure, there are 160 events/variables that could possibly correlate to the AVF. Navigating this large space of variables to identify the subset of events that must be tracked to accurately measure AVF over time is a challenging problem. Moreover, it is important that any online AVF monitoring mechanism be as lightweight as possible, both to avoid itself becoming a bottleneck and also to ensure that its predictions can keep pace with the flow of instructions through the pipeline. In other words, any hardware AVF monitoring mechanism must be able to provide an accurate estimate of the AVF while tracking only a small number of events, so that both the performance overhead and implementation complexity are low. In Section 4, we use regression analysis to identify the relevant subset of processor events, and construct and evaluate a practical predictor for the runtime AVF.

4. AVF BEHAVIOR ANALYSIS

The AVF and performance graphs (Figure 1) suggest that a correlation may exist between AVF values and other common performance metrics. Such a correlation would allow for the creation of a prediction function for estimating instantaneous AVF values. Ideally, such a predictor would be a low-dimensional function that could be evaluated cheaply in the hardware. To determine the strength of correlations between the variables and the corresponding prediction functions, we performed an extensive statistical analysis on the data obtained from executing twenty-two of the twenty-six SPEC2000 benchmarks. The four benchmarks were removed to avoid training our predictor to the test data. These benchmarks, two floating point (`apsi` and `galgel`) and two integer (`mcf` and `twolf`), were used for experimental analysis and verification. At each checkpoint during execution, 160 architectural variables were tracked along with AVF values for the RUU, LSQ, and ISQ, as described in Section 3. In this section, we provide the details of our statistical methodologies and show the resulting predictors.

4.1 Principal Component Analysis

Principal Component Analysis (PCA) has been used successfully in the computer architecture community to reduce the dimensionality of datasets and to tease out the important latent structure of correlated measurements (e.g., [6, 10, 11]). We first experimented with the use of PCA to reduce the number of microarchitectural components needed to create an accurate AVF-prediction function while retaining as much of the variation in the dataset as possible. PCA gives the optimal linear transformation for projecting data into a new coordinate system; by retaining lower-order principal components and ignoring higher-order ones, the dimensionality of the dataset can be reduced without removing the characteristics of the dataset that contribute the most to its variance [12].

Formally, we let X be a data matrix where each row corresponds to a performance metric and each column contains measurements of a checkpoint. Then the principal components are determined by finding the *Singular Value Decomposition* of X:

$$X = W\Sigma V^T.$$

The important quantities in this expression are V, a matrix whose columns form an orthogonal set of basis vectors spanning the checkpoint measurements, and Σ, a diagonal matrix whose entries indicate the strength of correlation between the original dataset and the corresponding basis vector.

After performing PCA, we identified 69 principal components (this number is not equal to the full 160 variables due to the numerical rank of our measurement matrix). The first four components had substantially higher singular values, as shown in Figure 2. Performing a linear regression using just these four linear components created a fit with an R^2 value of 0.61 for the RUU, 0.47 for the ISQ, and 0.55 for the LSQ. An R^2 value of 0.61 for the RUU simply means that sixty-one percent of the variation in the RUU measurements can be explained by the target components.

Unfortunately, despite the reasonable data fit, PCA is a poor choice for this application. Although applying PCA can substantially reduce the dimensionality of the dataset, the spanning eigenvectors are themselves linear combinations of the original 160 performance variables, and there-

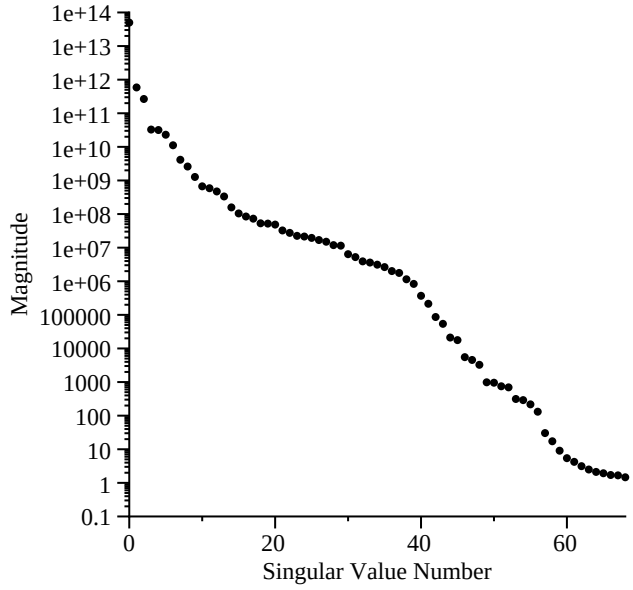

Figure 2: Singular values for the matrix of all performance measurements (columns) over each benchmark of SPEC2000 (rows). This log-plot is a direct visualization of the diagonal of the matrix Σ, where the ith entry indicates the degree to which the ith eigenvector accounts for the variance in the dataset. As is typical of datasets that are fundamentally lower-dimensional but embedded in a higher dimensional space, the magnitude of the singular values decreases extremely rapidly, suggesting that the data can be projected onto a low-dimensional subspace with little loss of fidelity.

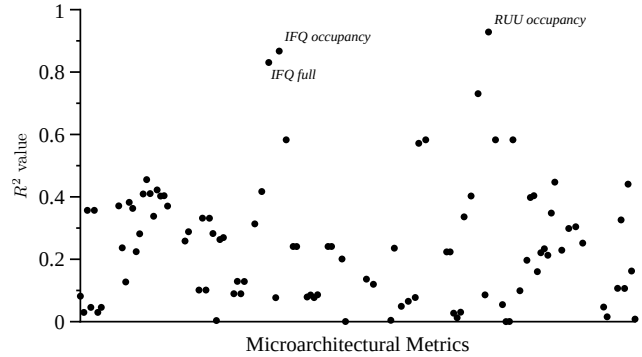

Figure 3: RUU AVF coefficients of variation for each of our 160 microarchitectural metrics when considered in isolation. Notice that while the majority of the data are only weakly correlated to RUU AVF, three datapoints (labeled above) clearly stand out with coefficients > 0.8, indicating a very strong correlation.

fore would require tracking all variables at runtime. Our goal is more ambitious; we want to track only a few variables and estimate AVF from those alone. Thus, PCA does not provide the low-dimensional function that we desire.

Instead, we investigated the possibility of a regression-based approach by separately considering the linear correlation between the AVF of each structure and each of our 160 microarchitectural metrics. The results of this experiment for the RUU are shown in Figure 3. That plot clearly shows that there are a very small number of variables which have high correlation with the RUU AVF value (results for ISQ and LSQ show qualitatively similar results and are omitted here). We were therefore encouraged to explore a multivariate regression to find a practical predictor.

4.2 Practical AVF Prediction

A comprehensive regression analysis reveals that we can track only a few variables to estimate the AVF values of the RUU, ISQ, and LSQ with great accuracy. Linear functions of a small number of performance variables could easily be calculated in hardware, and our analysis reveals that substantial precision can be achieved with only two to five variables. A nonlinear fit can also be used in order to obtain even higher statistical confidence in the predicted AVF values, but of course the resulting equations would be more costly to compute (for our experiments we looked only at quadratic fits).

Regression techniques enable us to calculate a function that will predict the value of a dependent variable from a collection of predictor variables. Formally, let $f_1, f_2, \ldots f_k$ be the predictor variables and y be a response variable. For example, the predictor variables might be IPC, cache miss rate, and branch misprediction rate, and the response variable may be the RUU AVF value. We then find the optimal representation of y as a weighted combination of basis functions b_i, where each b_i depends only on the values of a subset of the predictor variables.

The classical linear regression model for y uses the simplest basis set possible: the first order monomials. Specifically, it yields a set of weights β, one for each predictor variable f, and an error term ϵ:

$$y_i = \beta_0 + \beta_1 f_{1i} + \beta_2 f_{2i} + \ldots + \beta_k f_{ki} + \epsilon_i.$$

Here we explicitly use the fact that our variables are discretely sampled, so y_i is the ith response (e.g., AVF value), f_{ji} is the jth basis function (e.g., IPC) evaluated at the ith observation, and ϵ_i is the ith statistical error.

Regression analysis finds the optimal values of the coefficients $\beta_1, \ldots, \beta_k$ so as to minimize the sum of squared errors $\sum_i e_i^2$. It also yields a "coefficient of variation" (written R^2), which measures how well the generated function fits the observed data (larger R^2 values indicate a better fit, with $R^2 = 1$ taken to mean that the observations are collinear).

4.2.1 Linear Procedure

Here we describe the design of our linear predictor for the RUU AVF. Predictors for ISQ and LSQ were created in an analogous manner. Our goal is to select a small subset of the 160 tracked microarchitectural metrics that will give the highest possible correlation for any subset of that size. We begin by including the single variable with the highest correlation (for RUU AVF, this is the RUU occupancy with an R^2 of 0.93). We then consider all remaining 159 variables in turn, and for each we compute the linear regression of the two-variable expression involving the RUU occupancy and that variable. Once we have found the best two-variable approximation, this process repeats, adding one variable at a time until all variables have been exhausted. This approach of adding variables is mathematically justified due to the linear nature of the eventual fit.

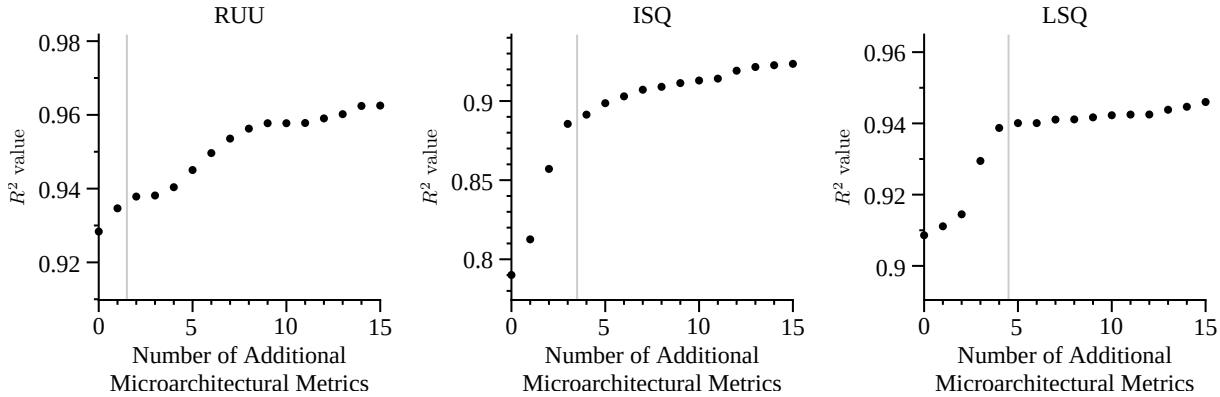

Figure 4: Coefficients of variation as more metrics are included in our predictor equations. For ISQ and LSQ, there is a clear knee in the curve at 4 and 5 variables, respectively, so we take that number of terms. The RUU fit has very high correlation even with a small number of terms, so its slope is very slight. We therefore include only two terms for that predictor.

Var.	Description
R_c	RUU count
R_l	RUU latency
R_o	RUU occupancy
S_{BW}	Total instructions executed (speculative and committed)
L_c	LSQ count
L_l	LSQ latency
L_o	LSQ occupancy
SC	Total number of "slip" cycles (from issue to retirement)
ASC	Average number of slip cycles
I_o	IFQ occupancy

Table 2: Description of notation used in Table 3.

Pred.	Struct.	Equation	R^2
P_1	RUU	$-2.27 + (0.71R_o) + (0.017 * \text{IPB})$	0.93
	ISQ	$(0.20R_l) + (0.55R_o) - (27.38 * \text{IPC}) + (2.73S_{BW})$	0.74
	LSQ	$-1.07 - (5.03e^{-9}L_c) + (1.35L_o) - (5.65e^{-9}R_c) + (0.01 * \text{IPB}) + (5.77e^{-9}SC)$	0.94
P_2	RUU	$1.14 + (0.001R_l) - (1.47e^{-6}R_l^2) + (0.10 * \text{IPC}) + (0.73R_l * \text{IPC}) - (0.09 * \text{IPC}^2)$	0.96
	ISQ	$-25.48 - (0.02ASC) + (1.41e^{-6}ASC^2) + (16.42 * \text{IPC}) + (0.51ASC * \text{IPC}) - (3.47 * \text{IPC}^2)$	0.81
	LSQ	$-1.49 + (0.03L_l) - (0.00003L_l^2) + (2.92 * \text{IPC}) + (1.41L_l * \text{IPC}) - (0.80 * \text{IPC}^2)$	0.94
P_3	RUU	$-3.06 + (0.73R_o)$	0.92
	ISQ	$-0.90 - (4.83I_o) + (0.74L_o) + (0.81R_o)$	0.69
	LSQ	$-1.23 + (1.41L_o)$	0.90

Table 3: Predictors obtained by our regression analysis for RUU, LSQ, and ISQ. P_1 is our multi-variable linear predictor model, P_2 is our quadratic model, and P_3 is our single-variable linear predictor model.

This procedure imposes an ordering on our set of microarchitectural metrics. It is important to note that this ordering is *not* the same as the ordering that would result by simply sorting the metrics based on their individual correlation with the AVF, because that ordering cannot take into account any dependencies between variables (this phenomenon is explored in more detail in Section 4.3). Figure 4 shows the results of this incremental subset construction for the RUU, ISQ, and LSQ. For the ISQ and LSQ, there is a clear knee in the curve at subsets including four and five variables, respectively. We therefore select those subsets for our predictor, because adding more variables would provide little additional correlation for the added expense of more predictor terms. In the case of RUU AVF prediction, the RUU occupancy metric is already so highly correlated with the AVF that a simple two-term approximation provides an excellent result of $R^2 = 0.935$. Because the total range of correlation coefficients is so small, no clear knee is present and we use a simple two-term approximation.

4.2.2 Quadratic Procedure

We have also experimented with a nonlinear regression resulting in a quadratic fit. Unlike the above procedure, we are no longer justified in adding terms one at a time, due to the nonlinear nature of the eventual model. Furthermore, because quadratic functions are more expensive to evaluate in hardware, and because our linear models already behave excellently with only a very small number of variables, we opted to limit the number of terms in our quadratic fits to two. We therefore consider all $\binom{160}{2} = 12720$ possible pairs of metrics, and for each pair (x, y) we perform a nonlinear least-squares fit using the basis $\{1, x, y, xy, x^2, y^2\}$.

4.3 Predictor Usage and Results

Here we show the precise prediction coefficients and metric subsets selected by the procedures described above. Specifically, we describe predictor P_1, the multi-variable linear predictor described in Section 4.2.1 and predictor P_2, the quadratic predictor from Section 4.2.2. In addition, we show a linear predictor P_3 whose terms are those variables with the highest individual correlation to the AVF. As described

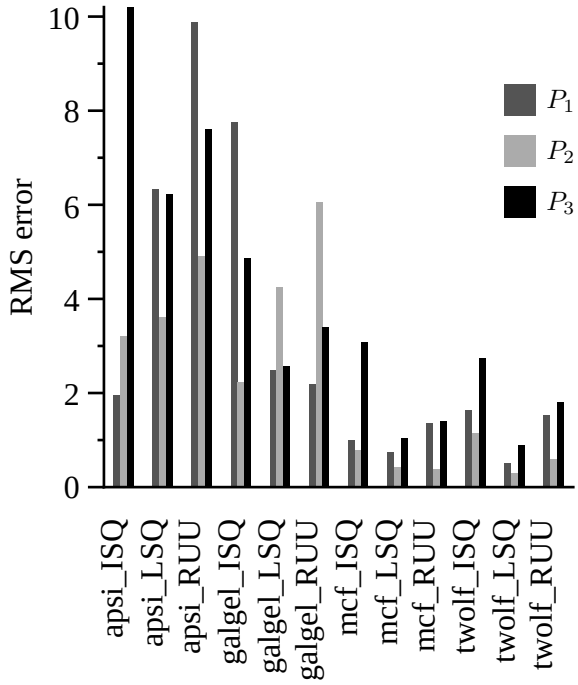

Figure 5: Root-mean squared error for all configurations of our test benchmarks. Three predictors are used for RUU, LSQ, and ISQ AVF estimation over four benchmarks. These benchmarks were not used in predictor training. Although no one predictor consistently performs best, the multivariable linear predictor (P_1) seems to provide an excellent compromise between cost of implementation and accuracy.

above, P_3 ignores dependencies between variables, and is presented here for comparison purposes only. Table 2 gives the notation for the individual microarchitectural metrics selected for these predictors, and Table 3 shows the actual equations and the obtained R^2 correlation coefficients.

Figure 5 shows the root-mean squared error when using each of the three predictors for each component, over four benchmarks. Specifically, we use the four SPEC2000 benchmarks that we omitted when performing our regression analysis (`apsi`, `galgel`, `mcf`, and `twolf`) to ensure that our results were not unfairly biased. These results show that P_1 is almost always better than P_3, as expected. However, the fact that P_1 is not *always* better underscores the importance of performing regression analysis on a representative workload for the processor's target application.

We can also see that the quadratic predictor P_2 frequently achieves lower error than the linear P_1. However, this is not always the case, and P_2 will likely be more expensive to evaluate in actual hardware. Our conclusion again is that in a production system it would be necessary to perform careful experiments on typical workloads to determine the ideal predictor to use.

Figure 6 is a different way of comparing the success of the three proposed prediction schemes. We compare the predicted versus measured RUU AVF of the `galgel` benchmark over time. These experiments on other components or benchmarks yield qualitatively similar results and are therefore omitted. The multivariable linear predictor P_1 performs best on this test. Furthermore, the behavior of the measured AVF for this configuration is very noisy, but all three predictors capture this quality faithfully. This is a testament to the accuracy of our AVF characterization.

5. CASE STUDY: AN AVF-AWARE RMT IMPLEMENTATION

In this section, we describe one possible use for an accurate online AVF predictor. In order to demonstrate the utility of our prediction scheme, we augmented our simulation infrastructure to use the predicted vulnerability values to enable RMT only when the total vulnerability of the LSQ, ISQ, and RUU was above a user-supplied threshold. Once enabled, RMT is then left enabled for a fixed number of cycles. We stress here that the details of this particular strategy for enabling/disabling RMT, as well as the particular settings of our user-defined parameters (critical AVF threshold and duration of RMT enabling) are not meant to be prescriptive. In practice, the proper settings of these parameters (and indeed the RMT enabling scheme itself) would likely either be customized to the particular architecture and application or provided as a knob to the operating system. Our goal here is merely to demonstrate that an AVF predictor *can* be used to enhance performance while still providing safety when needed, rather than to suggest precisely *how* such a feature should be implemented. Therefore, we do not present a thorough analysis of the design space of AVF thresholds, but rather demonstrate that the values we have chosen work well for our particular application.

5.1 RMT Toggling Approach

Our simulation begins with redundancy disabled. Every two million cycles, we evaluate our AVF predictor for the LSQ, ISQ, and RUU components, and calculate the total AVF of the processor as the sum of these three quantities. Whenever the total AVF rises above a vulnerability threshold, we effectively generate a processor interrupt, flushing the entire pipeline. The trailing thread and all of its associated data structures are then enabled, and execution resumes. Again we emphasize that our choices for frequency of vulnerability checking and threshold are not meant to be prescriptive; we merely found them to work well for our applications, and processor designers should carefully consider this design space for their own needs.

Once RMT is enabled, we need a criterion for disabling it again. Ideally, we would track what we call the processor's *virtual AVF*: the current vulnerability of the processor if the trailing thread were disabled. Unfortunately, this quantity is almost impossible to measure, let alone predict, because AVF depends on time-varying quantities that are unknown while the processor is in RMT mode. Furthermore, it would not make sense to measure or predict the AVF of the processor while RMT is enabled; recall that AVF is the likelihood that a soft error would result in incorrect execution, but RMT makes this probability effectively zero. Therefore, we disable RMT after a fixed number of cycles (ten million in our setup). The processor then begins predicting AVF again, as before.

Note that this scheme could potentially disable RMT at a time when the virtual AVF was high, since we do not estimate or measure virtual AVF. In such a case, RMT will be re-enabled quickly, but the processor would be left vul-

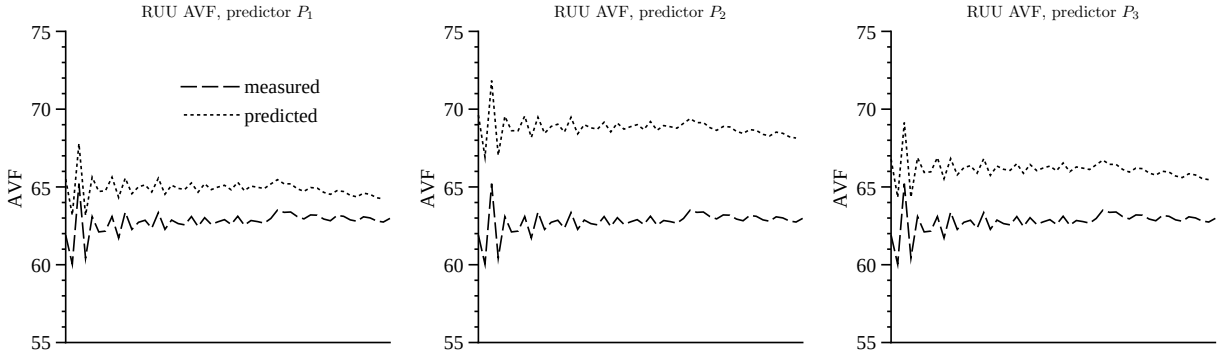

Figure 6: Measured versus predicted RUU AVF results for the `galgel` benchmark. The P_1 predictor achieves substantially less RMS error than the other two. In addition, note that all three predictors are able to faithfully capture the noisy time-varying behavior of AVF, confirming that our prediction model is extremely accurate.

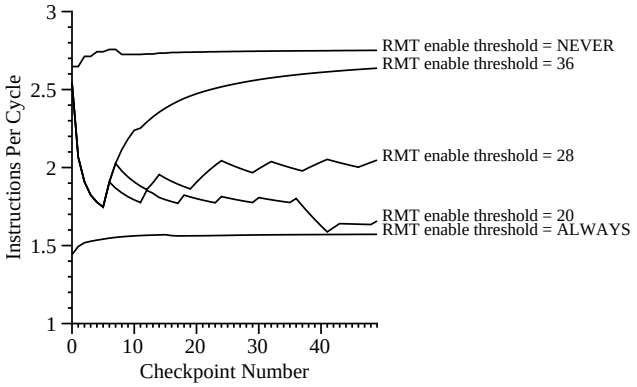

Figure 7: IPC over time for `twolf`. Five configurations are shown: no redundancy, full redundancy, and three AVF-aware partial redundancy runs. Each AVF-aware run has a different threshold for enabling redundancy. Partial redundancy provides a significant performance boost over full redundancy, as expected.

nerable during this time. This tradeoff between occasional vulnerability and resource utilization would need to be evaluated with respect to the fault-tolerance needs of a processor's target applications. As power consumption and parallel performance become more and more important even for mission critical applications, we believe that this tradeoff can be made reasonably while still providing standard probabilistic fault-tolerance guarantees.

5.2 Performance results for `twolf`

We implemented the above RMT toggling approach in our processor simulator and ran the entire SPEC2000 benchmark suite in this mode. Because this case study is meant primarily as illustrative of the power of AVF prediction, here we select the `twolf` benchmark as representative of the type of results we achieved across the entire benchmark suite.

We first ran `twolf` twice, once with SRT always enabled, and once with it always disabled. In these configurations, the benchmark achieved an average IPC of 1.56 and 2.74, respectively. These values provide a lower and upper bound on the performance of our RMT toggling approach. We tested several different AVF thresholds; the results of these experiments are shown in Figure 7. With the highest threshold (36), `twolf` achieves an average IPC of 2.41, a performance boost of 55% compared to full redundancy. Of course, a high threshold provides the least redundancy. At the other extreme, a low threshold (20) results in an average IPC of 1.78, a 14% performance increase with excellent vulnerability protection. Inspecting the graph reveals that RMT toggling lies between the two boundary curves, as expected. Note that the subtle oscillatory behavior of the middle IPC curve reveals the locations of our RMT toggle decisions.

6. CONCLUSION AND FUTURE WORK

We have shown that the temporal variation in vulnerability can be modeled with high accuracy using an inexpensive quadratic or linear model in only a few easily-tracked processor performance variables. Such a predictor is an important step forward for fault-tolerant processor design because it allows resources to be reallocated at runtime to where they would be most useful. The key insight is that the expensive calculations needed to determine AVF can be performed in a one-time *offline* analysis of detailed simulation traces, and the results of these calculations can be expressed as an inexpensive *online* predictor, yielding a practical new approach for AVF-aware RMT-enabled processor design.

Currently, we are performing a more rigorous exploration of the design space of partial RMT implementations. While the results in Section 5.2 demonstrate a significant performance improvement, those experiments were intended mainly to verify our hypothesis that an AVF-aware RMT approach could provide performance gains without sacrificing fault-tolerance. We believe that additional performance improvements could be achieved with a more sophisticated toggling scheme, such as adaptive AVF thresholds and non-constant RMT window sizes.

With the ability to dynamically predict the AVF, and therefore adapt the RMT at runtime, comes the opportunity to trade off performance, power, and fault-tolerance under operating system control. We intend to explore the space of possibilities offered by this technique. For example, entertainment and multimedia applications tend to be more tolerant of transient faults [26]. The operating system should therefore be able to adapt its level of fault tolerance

to the currently running application in order to save power and increase performance.

Finally, armed with our knowledge of which processor structures are most highly correlated with vulnerability, we would like to explore a symbiotic hardware/software approach where a model of AVF is incorporated into a machine code generation model to lower the runtime AVF induced by a given program. We will investigate static compilation techniques to achieve this goal, as well as the more exotic approach of using hardware feedback to guide the operation of a dynamic binary translation system. We believe that the power of this kind of dynamic hardware/software symbiosis has not yet been fully appreciated, and it could have a major impact on how robust processors and operating systems are co-designed in the future.

7. ACKNOWLEDGEMENTS

This research was supported by a grant from Intel Corporation. Kristen Walcott is funded via a NSF Graduate Research Fellowship. We would like to thank the FACT Group for their valuable feedback. Also, we would like to thank the anonymous reviewers for their comments.

8. REFERENCES

[1] D. Bernick and et al. NonStop®Advanced Architecture. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 12–21, June 2005.

[2] A. Biswas, P. Racunas, R. Cheveresan, J. S. Emer, S. S. Mukherjee, and R. Rangan. Computing architectural vulnerability factors for address-based structures. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 532–543, 2005.

[3] D. Burger and T. Austin. The SimpleScalar Toolset, Version 3.0. http://www.simplescalar.com.

[4] C.L. Chen and M.Y. Hsiao. Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review. *IBM Journal of Research and Development*, 28(2):124–134, March 1984.

[5] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and predicting program behavior and its variability. In *PACT '03: Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, page 220, Washington, DC, USA, 2003. IEEE Computer Society.

[6] L. Eeckhout, H. Vandierendonck, and K. D. Bosschere. Workload design: Selecting representative program-input pairs. In *PACT '02: Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, pages 83–94, Washington, DC, USA, 2002. IEEE Computer Society.

[7] X. Fu, J. Poe, T. Li, and J. Fortes. Characterizing Microarchitecture Soft Error Vulnerability Phase Behavior. In *Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, September 2006.

[8] M. Gomaa, C. Scarbrough, T. Vijaykumar, and I. Pomeranz. Transient-Fault Recovery for Chip Multiprocessors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 98–109, June 2003.

[9] M. A. Gomaa and T. N. Vijaykumar. Opportunistic transient-fault detection. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 172–183, 2005.

[10] D. Grunwald, A. Klauser, S. Manne, and A. R. Pleszkun. Confidence estimation for speculation control. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 122–131, 1998.

[11] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. D. Bosschere. Performance prediction based on inherent program similarity. In *PACT '06: Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, pages 114–122, New York, NY, USA, 2006. ACM Press.

[12] I. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[13] S. Kumar and A. Aggarwal. Reduced Resource Redundancy for Concurrent Error Detection Techniques in High Performance Microprocessors. In *Proceedings of the International Conference on High Performance Computer Architecture (HPCA)*, pages 212–221, February 2006.

[14] N. Madan and R. Balasubramonian. A First-Order Analysis of Power Overheads of Redundant Multi-Threading. In *Proceedings of the Workshop on the System Effects of Logic Soft Errors (SELSE)*, April 2006.

[15] S. Mukherjee, M. Kontz, and S. Reinhardt. Detailed Design and Evaluation of Redundant Multithreading Alternatives. In *International Symposium on Computer Architecture (ISCA)*, pages 99–110, May 2002.

[16] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 29–40, December 2003.

[17] Multiple SimPoints. http://www.cse.ucsd.edu/∼calder/simpoint/multiple-standard-simpoints.htm.

[18] A. Parashar, S. Gurumurthi, and A. Sivasubramaniam. A Complexity-Effective Approach to ALU Bandwidth Enhancement for Instruction-Level Temporal Redundancy. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 376–386, June 2004.

[19] A. Parashar, S. Gurumurthi, and A. Sivasubramaniam. SlicK: Slice-based Locality Exploitation for Efficient Redundant Multithreading. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 95–105, October 2006.

[20] M. Rashid, E. Tan, M. Huang, and D. Albonesi. Exploiting Coarse-Grained Verification Parallelism for Power-Efficient Fault Tolerance. In *Proceedings of the*

International Conference on Parallel Architectures and Compilation Techniques (PACT), pages 315–325, September 2005.

[21] V. Reddy, S. Parthasarathy, and E. Rotenberg. Understanding Prediction-Based Partial Redundant Threading for Low-Overhead, High-Coverage Fault Tolerance. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 83–94, October 2006.

[22] S. Reinhardt and S. Mukherjee. Transient Fault Detection via Simultaneous Multithreading. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 25–36, June 2000.

[23] G. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. August. SWIFT: Software Implemented Fault Tolerance. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, March 2005.

[24] G. Reis, J. Chang, N. Vachharajani, R. Rangan, D. August, and S. Mukherjee. Design and Evaluation of Hybrid Fault-Detection Systems. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2005.

[25] E. Rotenberg. AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors. In *Proceedings of the International Symposium on Fault-Tolerant Computing (FTCS)*, pages 84–91, June 1999.

[26] J. Sheaffer, D. Luebke, and K. Skadron. The visual vulnerability spectrum: Characterizing architectural vulnerability for graphics hardware. In *Proceedings of the 2006 Graphics Hardware Workshop*, 2006.

[27] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 2002.

[28] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi. Modeling the Effect of Technology Trends on Soft Error Rate of Combinational Logic. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, June 2002.

[29] T. J. Slegel, R. M. A. III, M. A. Check, B. C. Giamei, B. W. Krumm, C. A. Krygowski, W. H. Li, J. S. Liptay, J. D. MacDougall, T. J. McPherson, J. A. Navarro, E. M. Schwarz, K. Shum, and C. F. Webb. Ibm's s/390 g5 microprocessor design. *IEEE Micro*, 19(2):12–23, 1999.

[30] J. Smolens, B. Gold, J. Kim, B. Falsafi, J. Hoe, and A. Nowatzyk. Fingerprinting: Bounding Soft-Error Detection Latency and Bandwidth. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 224–234, October 2004.

[31] J. Smolens, J. Kim, J. Hoe, and B. Falsafi. Efficient Resource Sharing in Concurrent Error Detecting Superscalar Microarchitectures. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 257–268, December 2004.

[32] SPEC CPU2000. http://www.spec.org/cpu2000/.

[33] T. Vijaykumar, I. Pomeranz, and K. Cheng. Transient-Fault Recovery via Simultaneous Multithreading. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 87–98, May 2002.

[34] A. Wood. Data integrity concepts, features, and technology. White Paper, Tandem Division, Compaq Computer Corporation.

[35] J. Zeigler. Terrestrial Cosmic Rays. *IBM Journal of Research and Development*, 40(1):19–39, January 1996.