Versatile Prediction and Fast Estimation of Architectural Vulnerability Factor from Processor Performance Metrics

Lide Duan, Bin Li and Lu Peng
Louisiana State University, Baton Rouge, LA 70803
{lduan1, bli, lpeng}@lsu.edu

Abstract

The shrinking processor feature size, lower threshold voltage and increasing clock frequency make modern processors highly vulnerable to transient faults. Architectural Vulnerability Factor (AVF) reflects the possibility that a transient fault eventually causes a visible error in the program output, and it indicates a system's susceptibility to transient faults. Therefore, the awareness of the AVF especially at early design stage is greatly helpful to achieve a trade-off between system performance and reliability. However, tracking the AVF during program execution is extremely costly, which makes accurate AVF prediction extraordinarily attractive to computer architects.

In this paper, we propose to use Boosted Regression Trees, a nonparametric tree-based predictive modeling scheme, to identify the correlation across workloads, execution phases and processor configurations between a key processor structure's AVF and various performance metrics. The proposed method not only makes an accurate prediction but quantitatively illustrates individual performance variable's importance to the AVF. Moreover, to reduce the prediction complexity, we also utilize a technique named Patient Rule Induction Method to extract some simple selecting rules on important metrics. Applying these rules during run time can fast identify execution intervals with a relatively high AVF.

1. Introduction

The electronic noise, which usually comes from large power supplies, strong radiations, or high-energy particle strikes [28], may invert the state of a logic device when the resulted charge has been accumulated to a sufficient amount. The introduced logic fault is termed as a soft error or a transient fault [16]. The shrinking trend in processor feature size, particularly the exponential growth rate of on-chip transistors, along with lower supply voltage and increasing clock frequency make modern processors extremely vulnerable to transient faults. Fortunately, not all such faults eventually affect the final program outcome. For example, a bit flip in an empty Reorder Buffer entry will not cause any effect in the program execution. Based on this observation, Mukherjee et al. [16] defined a structure's Architectural Vulnerability Factor (AVF) as the probability that a transient fault in that structure will finally produce a visible error in the output of a program. At any point of time, a structure's AVF can be derived via counting all the important bits that are required for Architecturally Correct Execution (ACE) in the structure, and dividing them by the total number of bits of the structure. Using the ACE analysis method, many publications (e.g. [16][9][10]) have reported a large masking effect of transient faults at the architectural level, that is, a key processor structure usually shows an AVF below 40%, but with a large variation over time.

The AVF values provide computer architects with an indicator, or actually an upper bound, of the system's susceptibility to transient faults. Dynamically tracking the AVF would be greatly helpful to achieve a trade-off between system performance and reliability. However, tracking a processor structure's vulnerability during program execution is extremely costly. In [16][9], the authors implemented a post-commit analysis window which tracks the most recent 40K committed instructions to determine the exact type of each instruction, and then used this information backward to estimate the reliability of various hardware structures. In other words, the AVF calculations were not performed on the fly.

Under this limitation, some mechanisms were introduced to predict, instead of measuring, the instantaneous AVF values at any point of program execution. By observing a fuzzy correlation between the hardware AVF and some common performance metrics such as IPC, branch prediction rate, cache miss rate, etc, Fu et al. [10] concluded that a simple performance metric was not a good indicator to the program reliability behavior. Walcott et al. [23] reexamined the correlation by extending the variable set to 160 easily-measured time-varying processor metrics. They adopted a multivariate regression-based statistical model using 22 workloads as a training set to extract a quantitative relationship between the AVF and a small subset of the variables, and then applied the obtained predictor to another 4 workloads. By demonstrating a very accurate prediction of the reliability behaviors, their work convincingly proved the

existence of a correlation between the AVF and various processor performance metrics. However, they restricted their model training/test within one configuration, and only focused on the first SimPoints [20] of SPEC2000 suite. It is not clear that the predictor obtained from one set of phases (i.e. the first SimPoints) will give accurate estimation for another set of phases (e.g. the second SimPoints), and most likely the model developed under one configuration would not work for other configurations, which significantly narrows its applicability.

In this paper, by employing Boosted Regression Trees (BRTs), a nonparametric tree-based predictive modeling scheme, we propose a *versatile* method which accurately predicts the AVF across different workloads, execution phases and processor configurations. Initially, a statistical model is trained with the first SimPoints measured from a set of workloads under a BRT-based algorithm, and then is tested by other workloads that are not included in the training set. The testing results show that the prediction is very accurate. Within the same configuration, the trained model also succeeds in predicting the vulnerabilities of the second SimPoints of all the workloads. We then extend our model by adding the configuration parameters to the training variable set, and demonstrate a very high accuracy in predicting the AVF variations under different configurations. Finally, to make our method easier to be used in practice, we propose a *fast* estimation approach which utilizes a Patient Rule Induction Method (PRIM) to extract some simple selecting "IF-ELSE" rules on important performance metrics. These rules can be used to monitor the performance variables during a program execution, and then to efficiently identify vulnerable intervals experiencing high AVF values.

In summary, the main contributions of this paper are the followings:

- ***Versatile AVF Predictions:*** our proposed method accurately predicts the AVF across different workloads, execution phases and processor configurations.
- ***Model Interpretation:*** the proposed model can quantify performance metrics' importance and the AVF's dependence on these variables. This provides computer architects with a scientific view on the processor AVF variation.
- ***Fast AVF Estimations:*** the selecting "IF-ELSE" rules generated from the PRIM-based algorithm greatly reduce the prediction complexity. This enables computer architects to efficiently identify highly vulnerable execution intervals during a program's run time.

The remainder of this paper is organized as follows. Section 2 introduces the statistical methods and their specific algorithms used in this paper. Section 3 describes our experimental setup. In Section 4, we first illustrate the influence of a variable on the vulnerabilities, and then demonstrate our model's applicability across workloads, phases, and configurations. In Section 5, we use PRIM-based scheme to fast estimate AVF online by generating some simple rules on a small set of performance variables. Section 6 lists the related work, and we finally draw the conclusions in Section 7.

2. Background and Methodology

We propose to use a nonparametric tree-based predictive modeling method, named Boosted Regression Trees, to predict the architectural vulnerability from the processor performance metrics. BRT is capable of identifying a few important features from a large number of performance variables and accurately capturing the correlation between a processor structure's AVF and these selected features. Although the fitted BRT model consists of an ensemble of (hundreds to thousands of) regression trees, it can be summarized, interpreted and visualized similarly to conventional regression models through measuring of relative variable importance and partial dependence functions. For a fast AVF prediction, we employ another scheme, i.e. Patient Rule Induction Method (PRIM), which is able to identify the "highly vulnerable" intervals based on a few interpretable "IF-ELSE" rules.

2.1 Boosted Regression Trees

Boosted regression trees, originally proposed by Friedman [7], is an ensemble technique that aims to improve the performance of a single model by fitting many models and combining them for prediction. BRT employs two algorithms: "regression trees" from Classification And Regression Tree (CART) [4] and "boosting" which builds and combines a collection of models, i.e. trees.

CART is a binary recursive partitioning algorithm and provides an alternative to traditional parametric models for regression problems. The term "binary" implies that CART first splits the space into two regions, and models the response by a constant for each region. Then the optimal variable and the split-point are chosen to achieve the best fit again on one or both of these regions. Thus, each node can be split into two child nodes, in which case the original node is called a parent node. The term "recursive" refers to the fact that the binary partitioning process can be applied over and over again. Thus, each parent node can give rise to two child nodes and, in turn, each of these child nodes may themselves be split to generate additional children. Although CART represents information in a way that is intuitive and easy to be visualized, it is not usually as accurate as its competitors.

Boosting is one of the recent enhancements to tree-based methods that have met with considerable success in prediction accuracy. In boosting, models such as re-

gression trees are fitted iteratively to the training data, using appropriate methods to gradually increase emphasis on observations modeled poorly by the existing collection of trees.

The detailed BRT algorithm used in our paper is described in Algorithm 1. We consider a problem with n observations $\{y_i, \mathbf{x}_i\}$, $i=1,2,...,n$, where $\mathbf{x}_i$ is a p-dimensional input vector (i.e. the performance variables) and y_i is the response (i.e. the AVF).

1. Initialize $\hat{f}_0(\mathbf{x}_i) = \bar{y}$, where $\bar{y}$ is the average for $\{y_i\}$.
2. Repeat for $m = 1,2,...,M$:
   a) Compute the current residuals $r_{im} = y_i - \hat{f}_{m-1}(\mathbf{x}_i),\ i = 1,...,n.$
   b) Partition the input space into H disjoint regions $\{R_{hm}\}_{h=1}^{H}$ based on $\{r_{im}, \mathbf{x}_i\}_{i=1}^{n}$.
   c) For each region, compute the constant fit $\gamma_{hm} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{hm}} (r_{im} - \gamma)^2$.
   d) Update the fitted model $\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \nu \times \gamma_{hm} I(\mathbf{x} \in R_{hm})$.
3. End algorithm.

Algorithm 1. BRT-based algorithm used in this paper.

Note that in Step 2.d, $I(\bullet)$ is an indicator function which returns 1 if its argument is satisfied, otherwise 0, and ν which controls the learning rate of the procedure is the "shrinkage" parameter between 0 and 1. Empirical results (see e.g. [7]) have shown that small values of ν always lead to better generalization errors. In this study, we fixed ν at 0.01.

From a user's point of view, BRT has the following advantages. First, BRT is inherently nonparametric and can handle mixed-type of input variables naturally. Not like other parametric models, BRT doesn't need to make any assumptions regarding the underlying distribution of the values for the input variables. For example, BRT can make researchers to avoid determining whether variables are normally distributed, and making transformations if they are not. Second, tree is adept at capturing complex-structured behaviors. In other words, complex interactions among predictors are routinely and automatically handled with relatively few inputs required from the analyst. This is in contrast to some other multivariate nonlinear modeling methods, in which extensive inputs from the analyst, analysis of interim results, and subsequent modifications of the method are required. Third, tree is insensitive to outliers. It is unaffected by monotone transformations and different scales of measurement among inputs.

2.2 Interpretation and Visualization from BRT

Even producing a model with hundreds to thousands of trees, BRT does not have to be treated like a black box. A BRT model can be summarized, interpreted and visualized similarly to conventional regression models. This includes identifying parameters that are most influential in contributing to the response's variation, and visualizing the nature of dependence of the fitted model on these important parameters.

The relative variable importance measures are based on the number of times a variable is selected for splitting, weighted by the squared improvement to the model as a result of each split, and then average over all trees. The relative influence is scaled so that the sum adds to 100%, with a higher number indicating a stronger influence on the response.

Visualization of fitted functions in a BRT model can be easily achieved through a partial dependence function, which shows the effect of a subset of variables on the response after accounting for the average effects of all other variables in the model. Given any subset $\mathbf{x}_\mathbf{s}$ of the input variables indexed by $\mathbf{s} \subset \{1,\cdots,p\}$. The partial dependence of $f(\mathbf{x})$ is defined as

$$F_s(\mathbf{x}_\mathbf{s}) = E_{\mathbf{x}_{\setminus \mathbf{s}}}[f(\mathbf{x})],$$

where $E_{\mathbf{x}_{\setminus \mathbf{s}}}[\cdot]$ means expectation over the joint distribution of all the input variables with index not in $\mathbf{S}$. In practice, partial dependence can be estimated from the training data by

$$\hat{F}_\mathbf{s}(\mathbf{x}_\mathbf{s}) = (1/n)\sum_{i=1}^{n} \hat{f}(\mathbf{x}_\mathbf{s}, \mathbf{x}_{i\setminus \mathbf{s}}),$$

where $\{\mathbf{x}_{i\setminus \mathbf{s}}\}_1^n$ are the data values of $\mathbf{x}_{\setminus \mathbf{s}}$.

2.3 Patient Rule Induction Method

The objective of PRIM, which was originally proposed by Friedman and Fisher [8], is to find a set of subregions in the input space such as performance measures with relatively high values for the output (the AVF in this paper). The subregion (or "box") is described in an interpretable form involving simple "rules" taking the form $B = \bigcap_{j=1}^{p} (x_j \in s_j)$. For continuous variables, the subset s_j are represented by contiguous subintervals $s_j = [b_j^-, b_j^+]$. Thus, the projection of a box B on the subspace of real valued inputs is a hyper-rectangle. The box construction strategy of PRIM consists of two

Pipeline stages	8
Fetch/slot/map/issue/commit width	4/4/4/4/11
Fetch/slot queue size	4/4
Issue queue size	20
Reorder buffer size	80
Load/store queue size	32/32
Integer register file size	41 (1-cycle read latency)
Integer ALUs/multipliers	4/4 (latency: 1/7)
Branch predictor	Hybrid (local: 1K+1K; global: 4K; choice: 4K)
L1 instruction cache	64KB (64B block, 2-way, 1-cycle latency)
L1 data cache	64KB (64B block, 2-way, 3-cycle latency)
L2 cache	2MB (64B block, 1-way, 7-cycle latency)
ITLB/DTLB	Each: 128 entries, fully-associative
Victim buffer	8 entries, 1-cycle latency

Table 1. The Alpha-21264-like machine configuration (our baseline setting)

phases: (1) patient successive top-down peeling process; (2) bottom-up recursive pasting process.

The top-down peeling begins with the box B that covers all the data. At each iteration, a small subbox b within the current box B is removed, which yields the largest output mean value with the next box B-b. For each real valued variable, the two eligible subboxes b_{j-} and b_{j+} border its respective lower and upper boundaries of the current box B $b_{j-} = \{\mathbf{x} \mid x_j < x_{j\alpha}\}$ and $b_{j+} = \{\mathbf{x} \mid x_j > x_{j(1-\alpha)}\}$. Here $x_{j\alpha}$ is the α-quantile of the x_j-values for data within the current box. The peeling procedure stops when the support of the current box B is below a chosen threshold β. Hence, in this study, α is the proportion of intervals removed in each peeling process while β is the approximate proportion of intervals identified as high AVF regions. We fixed α at 0.05 and β at 0.1 in this study.

The pasting algorithm is the inverse of the peeling procedure. Starting with the peeling solution, the current box B is iteratively enlarged by pasting onto it a small subbox that maximizes the output mean in the new (larger) box. The bottom-up pasting is iteratively applied, successively enlarging the current box, until the addition of the next subbox causes the output mean to decrease.

3. Experimental Setup

We use *Sim-SODA* [9], a unified simulation framework that models software reliability of different microarchitecture structures in a microprocessor system, to measure the AVFs and a large set of performance metrics. *Sim-SODA* was developed based on *Sim-alpha* [6] which has been validated as an accurate Alpha 21264 simulator, and has been incorporated microarchitecture level AVF calculation methods for key processor structures. In this work, we use *Sim-SODA* to dump the time-varying AVF values for Integer Issue Queue (IQ) and Reorder Buffer (ROB). We believe that these two structures produce significant impact on the processor vulnerability. Without losing generality, our methods can be also used for other processor components.

Table 1 shows the Alpha-21264-like baseline machine configuration which will remain unchanged in Section 4.1. In Section 4.2, several key parameters will be tuned to generate 15 different configurations. For the experiments, all the benchmarks except one from the SPEC CINT 2000 suite are evaluated. The only exception is *gzip* whose simulation cannot be finished in a reasonable time in *Sim-SODA*. The floating point benchmarks of SPEC 2000 suite are not included in our experiments because *Sim-alpha* cannot accurately model Alpha 21264 floating point pipeline (thus *Sim-SODA* does not support AVF measurements for FP workloads). In order to perform a sufficient model training/test, we provide each benchmark with different inputs if possible, and the total 19 workloads are listed in Table 2 in which the training set includes the white columns and the test set consists of the gray columns. Note that the training and test sets are disjoint.

Each workload is run for two 100-Million Instruction SimPoints [1]. Table 2 also gives the number of instructions (unit: 100M) fast-forwarded to reach the SimPoints that we are interested in. In this paper, we term each SimPoint (i.e. the execution of 100M instructions) as a "phase", and each 500K instructions within a SimPoint as an "interval". In other words, for each workload, we simulate 2 phases, each containing 200 intervals. The granularity of dumping the AVFs and performance metrics is "interval", that is, the system records the AVF values (of IQ and ROB) and the values of 217 performance variables after the execution of every interval. We don't list all of them due to the page limit; instead, the following subsections will analyze the most important ones. Table 3 explains the abbreviation of variable names.

Benchmark	Phase 1	Phase 2	Benchmark	Phase 1	Phase 2	Benchmark	Phase 1	Phase 2
bzip2.source	4	104	*mcf*	1	37	*crafty*	114	252
eon.cook	78	187	*parser*	173	309	*gcc.200*	101	137
eon.kajiya	389	410	*perlbmk.makerand*	0	5	*gcc.integrate*	1	11
eon.rushmeier	210	213	*twolf*	2	122	*vortex.lendian3*	97	311
gap	83	239	*vortex.lendian1*	78	127			
gcc.166	0	20	*vortex.lendian2*	164	422			
gcc.expr	8	24	*vpr.route*	2	265			
gcc.scilab	38	112						

Table 2. Workloads and SimPoints used in our experiments

Abbreviation	Example	Meaning
xxx_count	load_q_writes_count	# writes to load queue in current interval
xxx_cumulative_count	ready_q_cumulative_count	the cumulative # ready queue entries in all cycles of current interval
xxx_average_count	rob_average_count	rob_cumulative_count / # cycles of current interval
xxx_cumulative_latency	fu_cumulative_latency	the cumulative # cycles that the committed instructions of current interval stayed in functional unit
xxx_occupant_rate	issue_q_occupant_rate	issue_q_average_count / issue_q_size

Table 3. Explanation of variable names

4. Versatile AVF Prediction

Generally, we believe that the AVF value of a key processor structure is a complex function of a large set of processor performance metrics. The exact form of the function may vary in different execution stages or different configurations. Nevertheless, our proposed method (i.e. BRT) is capable of identifying important features from a large set of performance variables and accurately predicting the vulnerabilities across workloads, execution phases, and different configurations. We show the AVF prediction in this section.

4.1 Prediction within the Same Processor Configuration

This subsection discusses the model training and test under our baseline setting (Table 1) to demonstrate that BRT accurately predicts the vulnerabilities of other workloads and future execution phases. Specifically, 15 phase files (workloads in the white columns in Table 2) are used to train a BRT model, which is then applied to other 4+19 phase files (phase 1 of 4 workloads and phase 2 of all workloads, as shown in the gray columns in Table 2).

We first apply Algorithm 1 (described in Section 2.1) using all 217 performance variables. Recall that, in each iteration, some variables are selected in Step 2.b of Algorithm 1 as features to partition the input space into H disjoint regions. We term variable importance as the average number of times (weighted by the contribution to the squared improvement made by the corresponding variable) a variable is selected in this step. The 10 most influential variables are listed in Figure 1. Note that the values shown have been scaled to a sum of 100%, with a higher percentage indicating a stronger influence on the AVF. As can be seen, the number of valid entries (*cumulative_count*, *average_coun*t) and the cumulative latency that the committed instructions spent in the structure significantly contribute to the vulnerability of the structure. In addition, states of some other microarchitecture components (e.g. Ready Queue, Load/Store Queue, Register File, etc) also show strong effects on the vulnerabilities of the IQ and ROB structures.

After identifying the 10 most important performance metrics, we refit the BRT model by only using the 10 selected variables. The prediction results of the workloads in the test set are shown in Figure 2. Note that in this paper all the AVF values are shown in a range of 0 to 100. As can be seen, for the 4 workloads (phase 1) on the left, the mean absolute errors (MAEs) for IQ and ROB AVFs are 0.93 and 0.55, respectively, validating the ability of our model to accurately predict the AVF variation on different workloads. Furthermore, the MAEs for the second phases of all 19 workloads are almost all below 4 with only two exceptions *mcf* and *vpr* whose IQ errors reach about 8. The small average MAEs (2.23 for IQ and 1.16 for ROB) of the phase 2 files indicate that the cross-phase correlation between the vulnerability and performance metrics can be captured by our model. Besides, the overall R^2 on the test set is 0.737 for IQ AVF, and 0.950 for ROB AVF, meaning that 73.7% of the variation in the IQ AVF and 95% of the variation in the ROB AVF can be captured by our predicted curves. On the other hand, Figure 3 depicts the overall empirical cumulative density function (CDF) for the absolute error values on the test set. We see that over 90% of the intervals are predicted be-

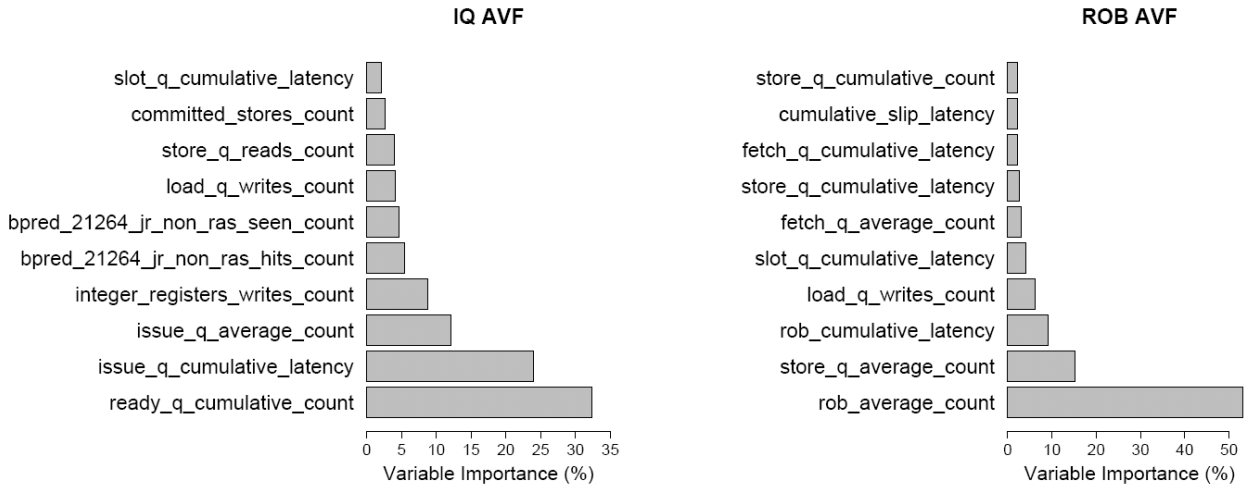

Figure 1. Relative variable importance (within the same configuration).

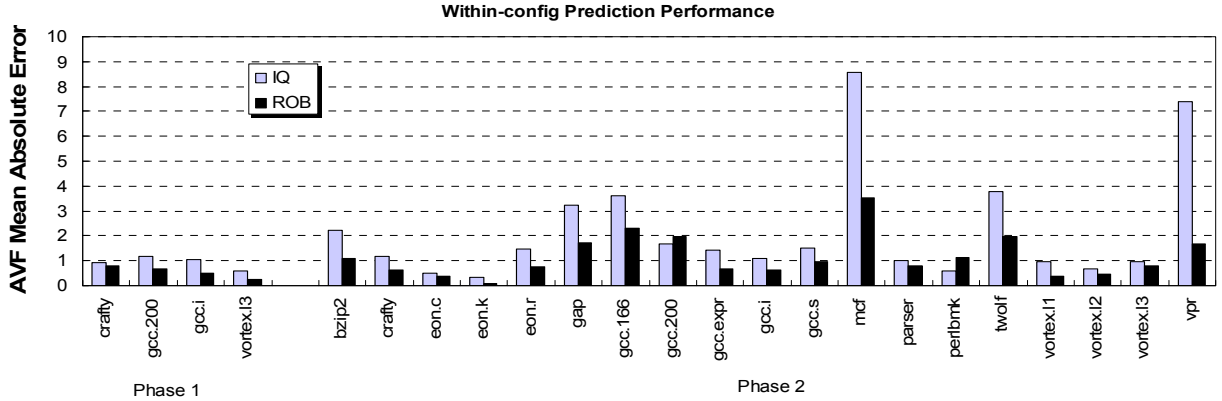

Figure 2. Prediction results on different workloads (4 phase 1 on the left) and future phases (19 phase 2 on the right).

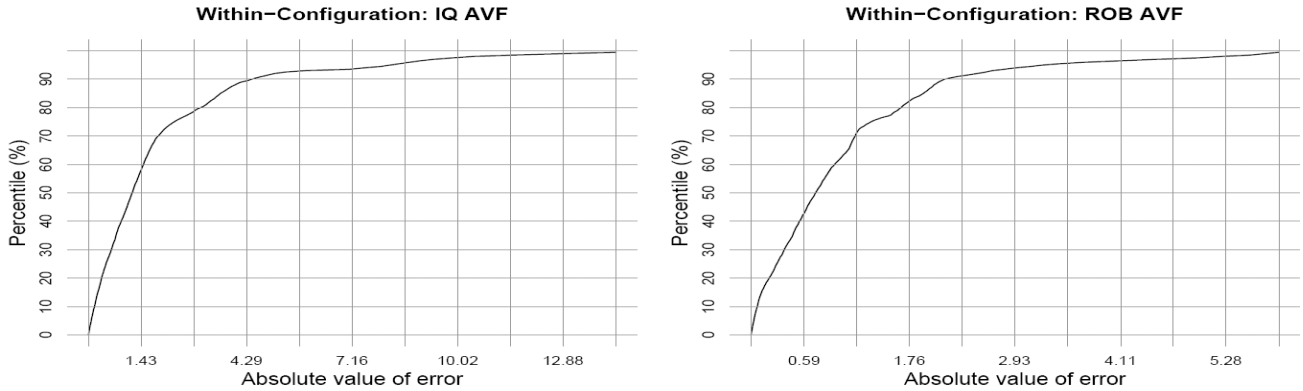

Figure 3. Empirical CDF on absolute errors in the within-configuration study

low absolute errors of 4.5 and 2.2 for IQ and ROB AVFs, respectively.

4.2 Prediction across Different Processor Configurations

The previous subsection demonstrates a correlation between the AVFs and a small set of performance met-

	Fetch/slot/map /issue widths	Commit width	Issue queue size	Reorder buffer size	Simulated workloads (2 phases for each)
cfg1	4	11	20	80	*mcf, vpr*
cfg2	4	11	40	40	*eon.cook, gap*
cfg3	4	11	30	60	*crafty, perlbmk.makerand*
cfg4	4	11	40	80	*eon.cook, eon.rushmeier*
cfg5	2	7	20	80	*eon.kajiya, gap*
cfg6	2	7	40	40	*gcc.166, gcc.200*
cfg7	2	7	20	40	*gcc.expr, gcc.integrate*
cfg8	2	7	40	80	*gcc.scilab, mcf*
cfg9	1	3	20	80	*parser, perlbmk.makerand*
cfg10	1	3	40	40	*twolf, vortex.lendian1*
cfg11	1	3	20	40	*vortex.lendian2, vortex.lendian3*
cfg12	1	3	40	80	*vpr.route, bzip2.source*
cfg13	4	11	20	40	*bzip2.source, crafty*
cfg14	2	7	30	60	*eon.kajiya, twolf*
cfg15	1	3	30	60	*gcc.expr, vortex.lendian1*

Table 4. Configurations used in Section 4.2. The training set contains the 48 phase files of *cfg1* to *cfg12* (white), and the test set includes the 12 phase files of *cfg13* to *cfg15* (gray).

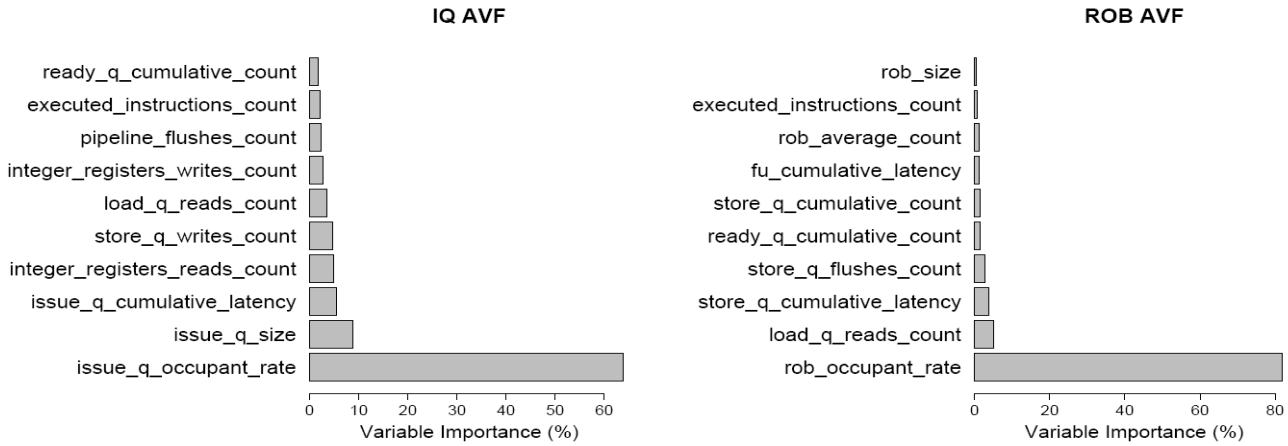

Figure 4. Relative variable importance (across different configurations).

rics across workloads and phases but within a specified processor configuration. In this subsection, we further extend our methodology to address the cross-configuration situation. Specifically, we tune the 7 parameters (4 of them are the same) listed in Table 4 to generate 15 different configurations because these parameters are believed to be dominant in producing the vulnerabilities of IQ and ROB. Note that *cfg1* is the baseline setting described in Table 1. We still employ the BRT methodology to perform the prediction in this case. However, in order to characterize the changing of configuration, we also include the tuned parameters in the performance metrics set as additional variables. Two randomly selected workloads, each also containing two phases, are simulated under each configuration. The training set consists of the phases under *cfg1* to *cfg12* (48 phase files in total) while the test set is composed of the other 3 configurations (12 phase files shown in gray).

Similar to the within-configuration study, we first apply BRT using all 217+7 input variables, and select the most important 10 features. After that, we refit the BRT model with the 10 metrics. The relative variable influences for this case are quantified in Figure 4. We see that the variable importance distribution (percentage and ranking) shown in Figure 4 is quite different from those depicted in Figure 1. This happens due to the multicollinearity problem in multiple regression models [17]. When many correlated input variables exist, the estimate of variable coefficients and their importance can be unstable since the effect from one variable may be disguised by its correlated variable(s). For example,

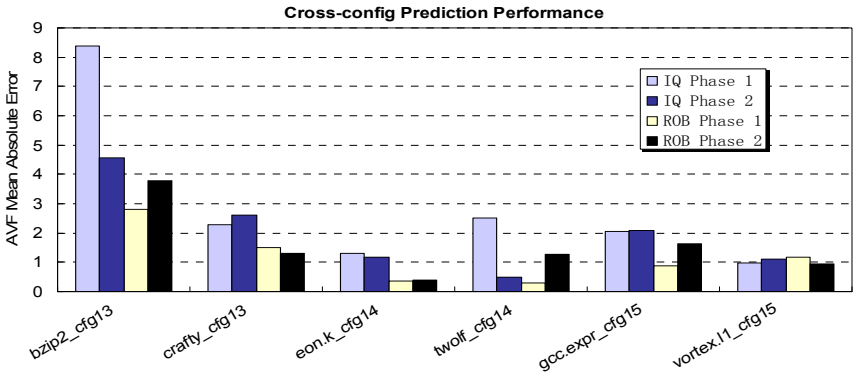

Figure 5. Prediction results on different configurations

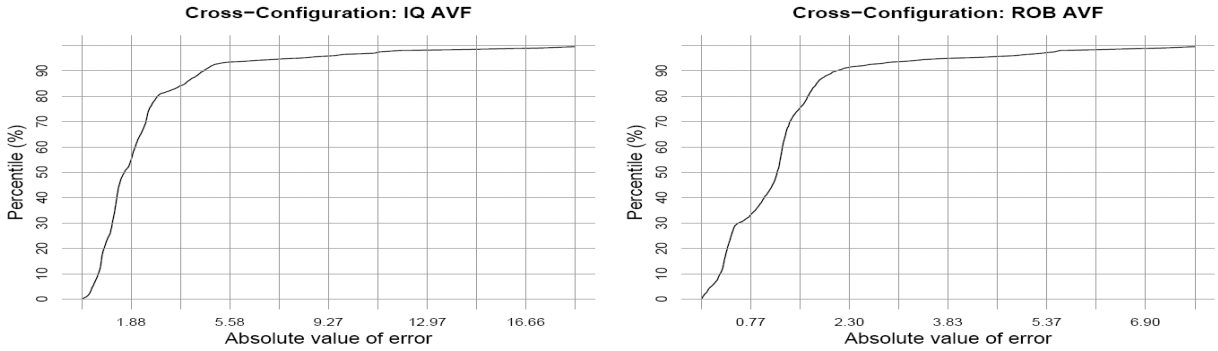

Figure 6. Empirical CDF on absolute errors in the cross-configuration study.

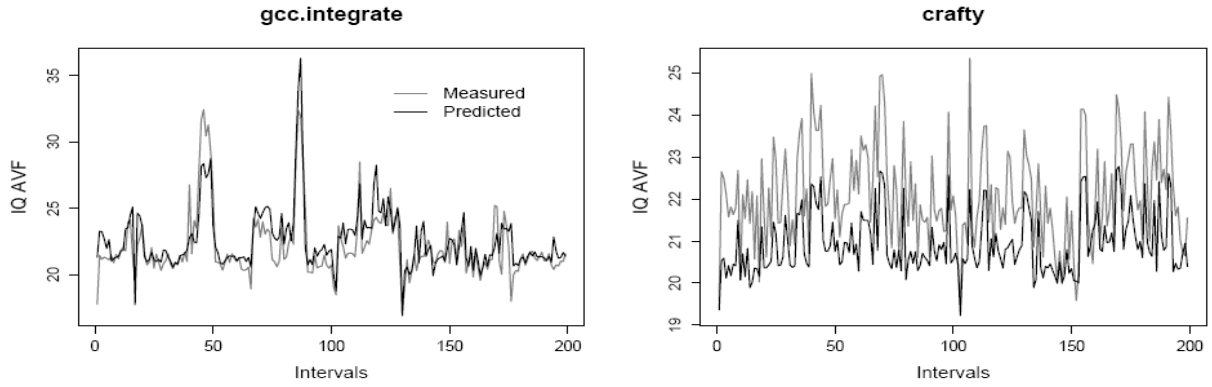

Figure 7. Measured and predicted IQ AVF curves for *gcc.integrate* and *crafty* in the within-configuration study.

rob_average_count and *rob_occupant_rate* are two highly corrected variables, but show completely different influences to the response in Figure 1 and 4.

As for the prediction performance illustrated in Figure 5, only one workload (*bzip2* under *cfg13*) is predicted with mean absolute errors of AVFs above 3, and the other 5 workloads under test show very high prediction accuracies in both IQ and ROB AVFs. Specifically, the overall MAEs of all the 6 workloads in the test set are 2.91 for IQ AVF (Phase 1), 2.0 for IQ AVF (Phase 2), 1.17 for ROB AVF (Phase 1), and 1.56 for ROB AVF (Phase 2). Note that the 6 workloads are simulated under 3 different configurations which also differ from the configurations used in the training set. Hence, the accurate prediction results validate that our model is capable of predicting vulnerability behaviors across configurations. The overall R^2 on the test set turns out to be 0.874 and 0.906 for IQ and ROB AVFs. In addition, Figure 6 shows that over 90% of the intervals are pre-

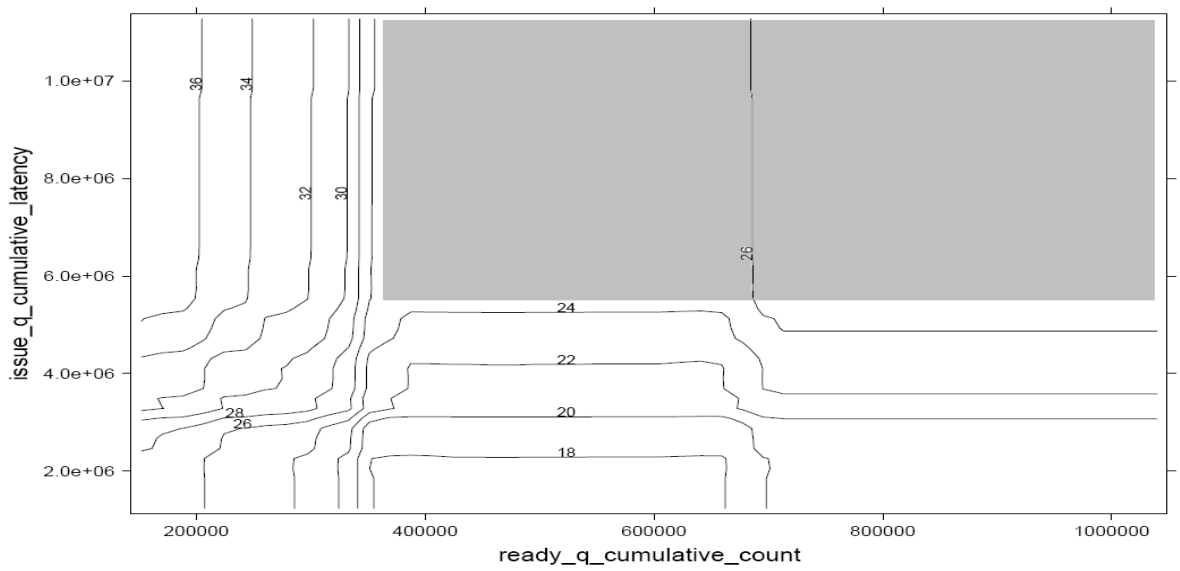

Figure 8. Partial dependence of the IQ AVF on the two most important variables in the within-configuration study.

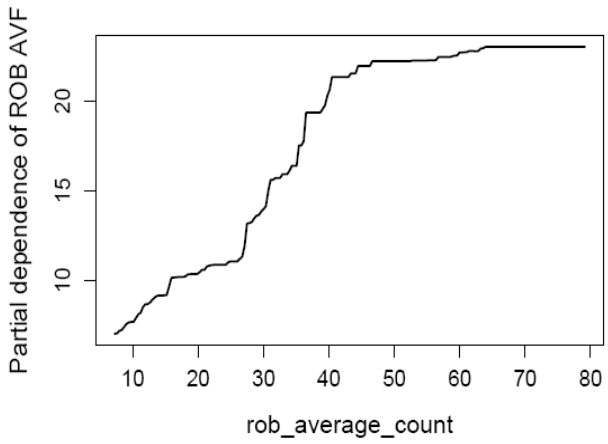

Figure 9. Partial dependence of the ROB AVF on the most important variable in the within-configuration study.

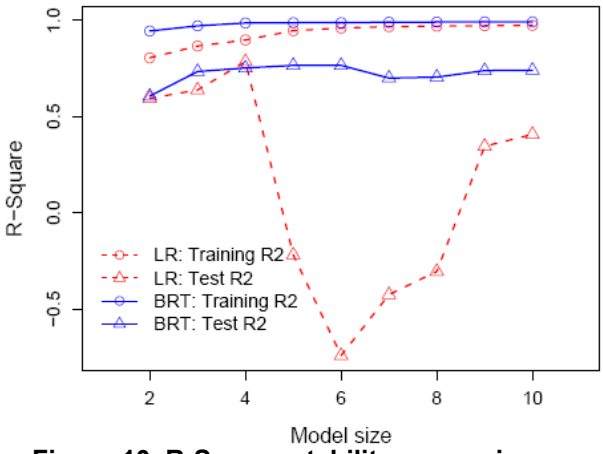

Figure 10. R-Square stability comparison between the BRT model and the Linear Regression model in [23].

dicted below absolute errors of 4.6 and 2.1 for IQ and ROB AVFs, respectively.

4.3 AVF Behavior Analysis and Model Interpretations and Comparison

Figure 7 provides another approach to compare the predicted and measured IQ AVF curves for two workloads *gcc.integrate* and crafty which are randomly selected as an example. Although the measured AVF behavior shows extremely strong variation over time, our prediction method is able to faithfully capture this behavior, and therefore confirms the high accuracy of BRT-based prediction.

Additionally, one can refer to Figure 8 and 9 for the partial dependence plots of the AVFs on the most important variables. As described in Section 2.2, Partial Dependence Function summarizes the effect of a subset of variables on the response (i.e. the AVF) after accounting for the average effect of other variables in the model. Therefore, partial dependence of the AVF provides computer architects with visible interactions between important performance metrics, and also implies the vulnerability trends and bottlenecks.

Specifically, Figure 8 illustrates that how the two most important variables contribute to the IQ AVF in the within-configuration study. This figure shows two hills in which variations of the parameters results in significant changes of the AVF and one plateau in which the AVF is insensitive to the variables' changes. Obviously, when the *issue_q_cumulative_latency* (the Y

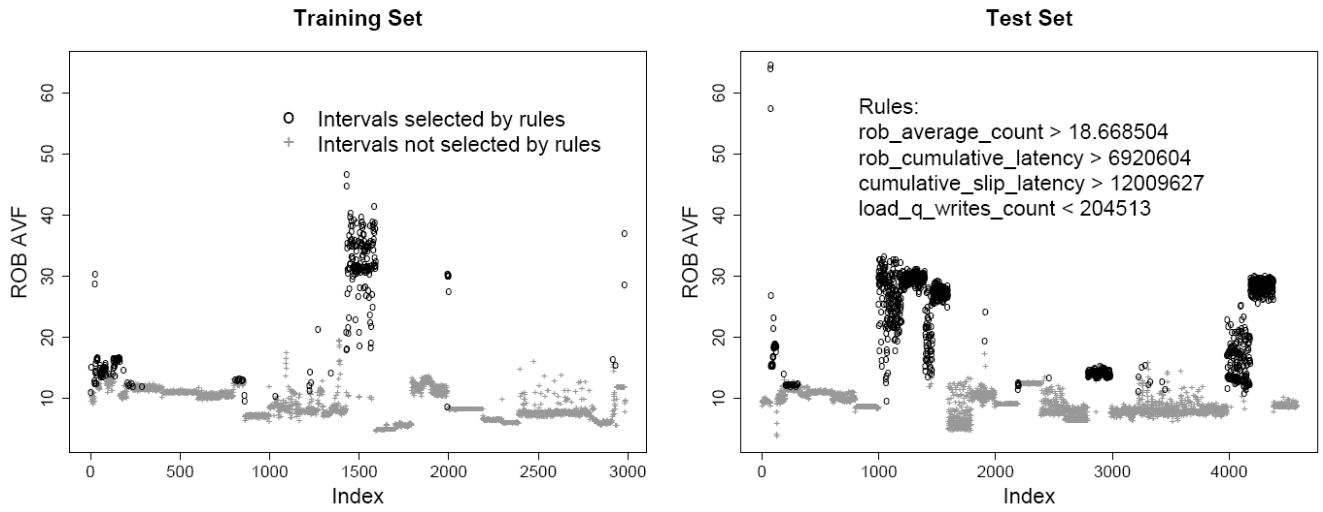

Figure 11. Fast Estimation of the ROB AVF in the within-configuration study.

axis) is less than 5.8e+06, increasing this latency boosts the AVF from 18 to 24. In addition, when the *ready_q_cumulative_count* (the X axis) is less than 350K, decreasing the cumulative Ready Queue count leads to a further AVF increasing from 26 to 36. A larger cumulative IQ latency means that the ACE instructions were kept for a longer time in IQ. A lower Ready Queue count indicates a worse congestion in IQ where the issued instructions wait for their operands. Both of the two cases contribute to a higher vulnerability of the IQ. The gray area in this figure represents a plateau where variations of the two metrics rarely affect the AVF and other processor variables should be considered to reduce the AVF in this situation. In [12], the authors also used a nonparametric model and contour maps to analyze the roughness and bottlenecks of processor design topologies.

The proposed model can also quantify the AVF's partial dependence to one very important variable. For example, the contribution of the ROB average count to the ROB AVF is shown in Figure 9. We can observe that the increasing of the ROB average count results in the increasing of the ROB AVF. This can be easily explained as the proportion of the valid ROB entries approximately characterizes the vulnerability of the ROB. The vulnerability saturates at around 23 when the ROB average count exceeds 48, in which case the ROB average count is no longer a driving factor to the AVF and other variables should be considered.

We also make a quantitative comparison between our suggested BRT method and classical linear regression approach. For linear regression, we followed Walcott et al.'s approach in [23]. Figure 10 illustrates the comparison result. Here we only consider the IQ AVF prediction for the within-configuration case. It shows the R-squares on the training and test sets with different number of variables included in the model in the linear regression approach and our BRT method. We see that although the R-squares increase monotonically on the training set for both linear regression and BRT, the test R-squares do not. Particularly in linear regression, the test R-square goes below zero when five to eight variables are included in the model. Therefore, the test R-square in BRT is much more stable than linear regression. This comparison demonstrates the robustness of our BRT model.

5. Fast AVF Estimation

In practice, a simpler AVF prediction mechanism is easier to be adopted. In order to reduce the model complexity, we further propose to use a PRIM-based technique described in Section 2.3 to summarize some simple and interpretable "IF-ELSE" rules that can be applied on the important performance variables selected in Section 4 during run time to quickly identify the intervals with high AVF values. Due to the page limit, we demonstrate the effectiveness of this method by only illustrating the ROB AVF prediction results within the baseline configuration. Other AVF predictions can also be applied.

The results of fast ROB AVF estimation in the within-configuration study are shown in Figure 11. We intend to find the top ~10% of the intervals in terms of the vulnerability level. Note that we denote a high vulnerable interval as a black "o" while an interval with a low vulnerability as a gray "+" in this figure. The training and test sets are the same as those in Section 4.1, that is, the training set shown in the left part of Figure 11 contains 3,000 intervals (white columns in Table 2) while the test set contains 4,600 intervals from the

benchmarks and phases listed in gray columns of Table 2. The rules extracted from the training data can be described as:

```
IF      ((rob_average_count > 18.668504)
   AND (rob_cumulative_latency > 6920604)
   AND (cumulative_slip_latency > 12009627)
   AND (load_q_writes_count < 204513))
THEN {
        The interval is declared to have a high
        ROB AVF value
}
```

One can refer to Table 3 for the explanation of variable names. The only one here that was not listed in Table 3 indicates the cumulative latency that the committed instructions spent in passing the whole pipeline. From the testing results shown in the right part of Figure 11, we can see that applying these simple rules to the test set makes an accurate AVF estimation, i.e. the AVF of current interval is high or not. The derived rules can be explained in an architectural way. Again, the valid ROB entries and the cumulative latency to go through it perform the estimation in the first place. Longer cumulative slip latency reflects a lower instruction processing speed of the whole pipeline, and infrequent writes to the Load Queue also make the vulnerable instructions stay long in the pipeline. Hence, all the identified rules show strong significance in estimating the architectural vulnerability.

6. Related Work

[15] compared the advantages and disadvantages of three different RMT techniques: (1) Lockstepping, a cycle-by-cycle synchronization that has long been used on commercial fault-tolerant systems; (2) Simultaneous and Redundantly Threading (SRT), which was first discussed in [18], utilizes the dynamic resource sharing from SMT processors to reduce performance degradation due to redundancy; and (3) Chip-Level Redundant Threading (CRT), which extends SRT to CMP environment, explores significant performance benefit on multithreaded workloads. Vijaykumar et al. [22] and Gomaa et al. [11] proposed the recovery schemes for SRT and CRT, respectively.

The concept of AVF was originally termed in [16], and Biswas et al. [2] extended it to address-based processor structures. There are two main approaches to calculate the AVF values: ACE analysis and Statistical Fault Injection (SFI). The former provides a (tight, if the underlying system is appropriately modeled [3]) lower bound on the reliability level of various processor structures, and has been adopted in many research works on performance models. Fu et al. [10] quantitatively characterized vulnerability phase behavior of four microarchitecture structures based on a system framework proposed in [9], which is also the simulator used in this paper. Zhang et al. [27] performed a similar analysis on SMT architectures. Soundararajan et al. [21] described a simple infrastructure to estimate an upper bound of the ROB AVF, and also proposed two mechanisms (Dispatch Throttling and Selective Redundancy) to bound the vulnerability to any limit.

Alternatively, SFI randomly (or statistically) injects into program execution a set of faults, each being independently analyzed and determined to see a visible error of the outcome. The AVF is the ratio of the number of trials that eventually raise an error to the total number of trials performed. Wang et al. [25] implemented a latch-accurate Verilog model to simulate an Alpha processor while Li et al. [14] incorporated a similar probabilistic model of error generation and propagation into an architecture-level tool. Wang et al. [24] compared ACE analysis to their fault-injection IVM, and claimed that ACE analysis was highly conservative by identifying two sources of its conservatism (lack of system detail and single-pass simulation). However, a recent publication [3] refuted their claim by stating that a small amount of additional details can result in a much tighter AVF bound and quantifying the small effect of Y-bits on system simulation.

Besides [10][23], some other works also addressed the problem of AVF prediction at run time. Cho et al. [5] examined workload dynamics in a design space of microarchitecture configurations. For each workload, they trained a set of neural networks with series of wavelet coefficients decomposed from AVF behaviors under different configurations, predicted the wavelet coefficients of any other configuration, and reconstructed the AVF curve (of the target configuration) from the predicted coefficients. Their work is completely different from ours in this paper because they required a separate (or different) set of neural networks for each workload while our model has been demonstrated to be validated across workloads, phases and configurations. Very recently, Li et al. [13] developed an algorithm to estimate processor structures' vulnerability online using a modified error injection and propagation scheme from their previous work [14]. Their method does not need any offline simulation (except some experimental experience to determine key parameters), but requires hardware modification of the processor to support error propagation and detection rules.

7. Conclusions

In this paper, we have proposed to use Boosted Regression Trees, a nonparametric tree-based predictive modeling scheme, to identify the correlation (across different workloads, execution phases, and processor configurations) between a key processor structure's

AVF and various performance metrics. Experimental results showed that our model can accurately predict the AVF in the above situations. In addition, the proposed model provides valid interpretation tools for computer architects to quantify important variables and the AVF's dependence on them. Finally, to reduce the prediction complexity, we also utilize another technique named Patient Rule Induction Method to extract some simple selecting rules to monitor a few important metrics, which can be used to quickly identify the execution intervals with a relatively high AVF.

Acknowledgement

This work is supported in part by the Louisiana Board of Regents grant LEQSF (2006-09)-RD-A-10 and the Louisiana State University. Anonymous referees provide helpful comments.

References

[1] 100 Million Interval Size Multiple Simulation Points. http://www.cse.ucsd.edu/~calder/simpoint/points/standard/spec2000- multiple-std-100M.html

[2] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. Mukherjee, and R. Rangan. Computing Architectural Vulnerability Factors for Address-Based Structures. In *International Symposium on Computer Architecture (ISCA)* 2005.

[3] A. Biswas, P. Racunas, J. Emer, and S. Mukherjee. Computing Accurate AVFs using ACE Analysis on Performance Models: a Rebuttal. In *Computer Architecture Letters* Vol. 7, 2008.

[4] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth International Group, Belmont, California, 1984.

[5] C. Cho, W. Zhang, and T. Li. Informed Microarchitecture Design Space Exploration using Workload Dynamics. In *International Symposium on Microarchitecture (MICRO)* 2007.

[6] R. Deskan, D. Burger, S. Keckler, and T. Austin. Sim-alpha: A Validated, Execution-Driven Alpha 21264 Simulator. Tech Report TR-01-23, The University of Texas at Austin, 2001.

[7] J. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. In *The Annuals of Statistics*, 29, 1189-1232, 2001.

[8] J. Friedman and N. Fisher. Bump Hunting in High-dimensional Data. In *Statistics and Computing*, 9, 123-143, 1999.

[9] X. Fu, T. Li, and J. Fortes. Sim-SODA: A Unified Framework for Architectural Level Software Reliability Analysis. In *Workshop on Modeling, Benchmarking and Simulation* 2006.

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

[11] M. Gomaa, C. Scarbrough, T. Vijaykumar, and I. Pomeranz. Transient-Fault Recovery for Chip Multiprocessors. In *ISCA* 2003.

[12] B. Lee and D. Brooks, Roughness of Microarchitectural Design Topologies and its Implications for Optimization. In *International Symposium on High-Performance Computer Architecture (HPCA)* 2008.

[13] X. Li, S. Adve, P. Bose, and J. Rivers. Online Estimation of Architectural Vulnerability Factor for Soft Errors. In *ISCA* 2008.

[14] X. Li, S. Adve, P. Bose, and J. Rivers. SoftArch: An Architecture-Level Tool for Modeling and Analyzing Soft Errors. In *International Conference on Dependable Systems and Networks (DSN)* 2005.

[15] S. Mukherjee, M. Kontz, and S. Reinhardt. Detailed Design and Evaluation of Redundant Multithreading Alternatives. In *ISCA* 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 *MICRO* 2003.

[17] J. Neter, and et al. *Applied Linear Statistical Models*, McGraw-Hill/Irwin; 4th edition, Feb. 1996.

[18] S. Reinhardt and S. Mukherjee. Transient Fault Detection via Simultaneous Multithreading. In *ISCA* 2000.

[19] G. Reis, J. Chang, N. Vachharajani, R. Rangan, D. August, and S. Mukherjee. Design and Evaluation of Hybrid Fault-Detection Systems. In *ISCA* 2005.

[20] T. Sherwood and et al. Automatically Characterizing Large Scale Program Behavior. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* 2002.

[21] N. Soundararajan, A. Parashar, and A. Sivasubramaniam. Mechanisms for Bounding Vulnerabilities of Processor Structures. In *ISCA* 2007.

[22] T. Vijaykumar, I. Pomeranz, and K. Cheng. Transient-Fault Recovery Using Simultaneous Multithreading. In *ISCA* 2002.

[23] K. Walcott, G. Humphreys, and S. Gurumurthi. Dynamic Prediction of Architectural Vulnerability from Microarchitectural State. In *ISCA* 2007.

[24] N. Wang, A. Mahesri, and S. Patel. Examining ACE Analysis Reliability Estimates Using Fault-Injection. In *ISCA* 2007.

[25] N. Wang, J. Quek, T. Rafacz, and S. Patel. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. In *DSN* 2004.

[26] C. Weaver, J. Emer, S. Mukherjee, and S. Reinhardt. Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor. In *ISCA* 2004.

[27] W. Zhang and et al. An Analysis of Microarchitecture Vulnerability to Soft Errors on Simultaneous Multithreaded Architectures. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)* 2007.

[28] J. Ziegler and et al. IBM Experiments in Soft Fails in Computer Electronics (1978-1994). IBM Journal of Research and Development, Volume 40, Number 1, 1996.