Efficient Microarchitectural Vulnerabilities Prediction Using Boosted Regression Trees and Patient Rule Inductions

Bin Li, Lide Duan, and Lu Peng, *Member*, *IEEE Computer Society*

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 (BRT), 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 also quantitatively illustrates individual performance variable's importance to the AVF. A quantitative comparison between our model and conventional linear regression is performed in terms of model stability, showing that our model is more stable when the model size varies. Moreover, to reduce the prediction complexity, we also utilize a technique named Patient Rule Induction Method (PRIM) to extract some simple selecting rules on important metrics. Applying these rules during runtime can fast identify execution intervals with a relatively high AVF. A case study that enables PRIM-based ROB redundancy has been performed to demonstrate a possible application of the trained PRIM rules.

Index Terms—Hardware reliability, modeling and prediction, modeling of computer architecture.

1 Introduction

The electronic noise, which usually comes from large power supplies, strong radiations, or high-energy particle strikes [32], 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 [19]. 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, Li et al. [17] defined a structure's Architectural Vulnerability Factor (AVF) as the probability that a transient fault in the structure finally produces 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., [19], [12], [13]) 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 percent, 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 [19], [12], 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 this paper, we utilize a similar scheme to calculate the AVF in the simulator. Besides the analysis window itself, a large amount of other structures need to be implemented in the window to maintain the dependencies among the instructions. The AVF simulation results are therefore delayed by the size of the window, and the simulation is significantly slowed down due to the operations performed in the window. Based on our simulation, the simulation time is increased to be at least 10 times larger than that without AVF measurements. These simulation overheads, which will result in hardware overheads as well if the AVF calculation is implemented in real processors, motivate us to predict, instead of measuring, the instantaneous AVF values. Moreover, the online prediction scheme bridges the gap (caused by the post-commit window) between completing performance simulation and calculating the AVF. In

- *B. Li is with the Department of Experimental Statistics, Louisiana State University, Baton Rouge, LA 70803. E-mail: bli@lsu.edu.*
- *L. Duan and L. Peng are with the Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803. E-mail: {lduan1, lpeng}@lsu.edu.*

Manuscript received 1 Nov. 2008; revised 3 Mar. 2009; accepted 25 Mar. 2009; published online 11 Sept. 2009.
Recommended for acceptance by C. Bolchini and D. Sciuto.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-2008-11-0543.
Digital Object Identifier no. 10.1109/TC.2009.140.

other words, with AVF prediction, we are able to quantify the AVF during program runtime, instead of waiting until 40K more instructions to be committed. This advantage further enables the processor to respond earlier to vulnerable situations.

Some mechanisms were already introduced to predict the AVFs 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, Fu et al. [13] concluded that a simple performance metric was not a good indicator to the program reliability behavior. Walcott et al. [27] 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 [23] 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 (BRT), 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 is then 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 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 following:

- **Versatile AVF prediction:** 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.
- **Model comparison:** We also compare the model stability between BRT and linear regression. We found that BRT is more stable when the model size varies.
- **Fast AVF estimation:** 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 runtime.
- **A case study:** PRIM-based ROB Redundancy. This case study demonstrates a possible application of PRIM-based fast AVF estimation. It effectively reduces the ROB AVF to a very low level with negligible performance degradation.

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. We also compare the stability of our proposed method with the conventional linear regression models. In Section 5, we use PRIM-based scheme to fast estimate the 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. Appendix A illustrates the usefulness of our method by an ROB reliability throttling technique.

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 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 "high-vulnerable" intervals based on a few interpretable "IF-ELSE" rules.

2.1 Boosted Regression Trees

Boosted regression trees, originally proposed by Friedman [10], 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, i.e., "regression trees" from Classification And Regression Tree (CART) [5], and "boosting," which builds and combines a collection of models (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

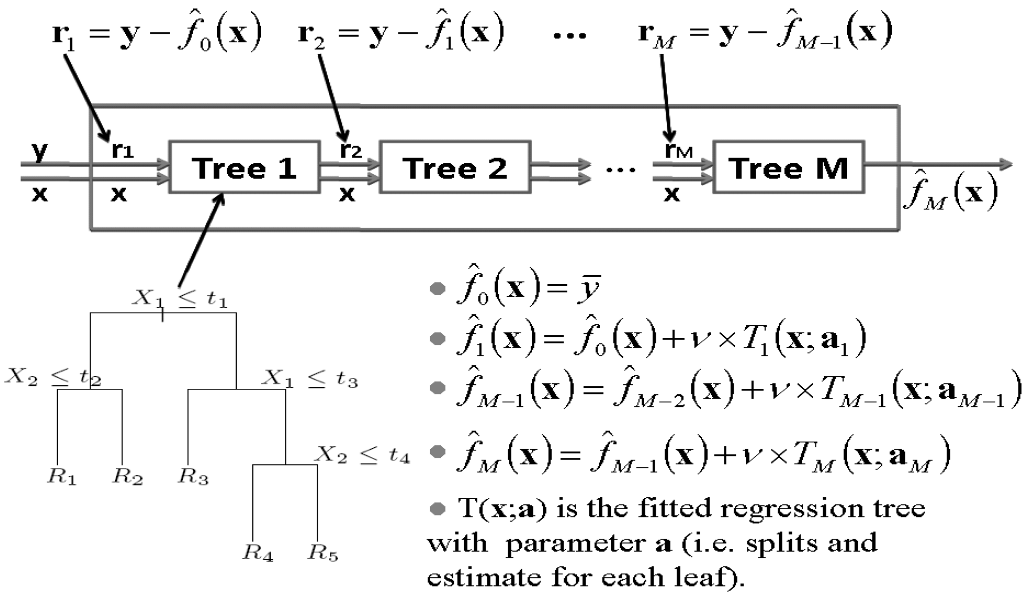

Fig. 1. Illustration of the BRT algorithm in 2D input space.

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 regression 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 $\{y_i, x_i\}, i = 1, 2, \ldots, n,$ where x_i is a p-dimensional input vector (i.e., the performance variables) and y_i is the response (i.e., the AVF).

Algorithm 1. BRT-based algorithm used in this paper
1. Initialize $\hat{f}_0(\mathbf{x}_i) = \bar{y},$ where $\bar{y}$ is the average for $\{y_i\}$.
2. Repeat for $m = 1, 2, \ldots, M$:
   a) Compute the current residuals
      $r_{im} = y_i - \hat{f}_{m-1}(\mathbf{x}_i), i = 1, \ldots, n.$
   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
      $\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \nu \times \sum_h \gamma_{hm} I(\mathbf{x} \in R_{hm})$
3. End algorithm.

Note that in d) of Step 2, $I(\bullet)$ is an indicator function, which returns 1 (otherwise 0) if its argument is satisfied. ν is a parameter between 0 and 1, controlling the learning rate of the procedure. Empirical results (e.g., [10]) have shown that smaller values of ν always lead to better generalization errors. In this study, we fixed ν at 0.01. For simplicity, Fig. 1 illustrates the BRT algorithm when the input space has only two dimensions. Basically, a regression tree is constructed by recursively partitioning the input space in each of the M iterations, and the prediction function is updated by the parameters calculated in the leaves of the tree.

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. Unlike 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 percent, with a higher number indicating a stronger influence on the response.

TABLE 1
The Alpha-21264-Like Machine Configuration (Our Baseline Setting)

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

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 $\mathbf{X}_\mathrm{s}$ of the input variables indexed by $\mathrm{s} \subset \{1, \ldots, p\}$. The partial dependence of $f(\mathbf{x})$ is defined as

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

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

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

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

2.3 Patient Rule Induction Method

The objective of PRIM, which was originally proposed by Friedman and Fisher [11], 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 $B = \cap_{j=1}^{p} (x_j \in s_j)$. For continuous variables, the subsets 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 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 $\mathrm{bj}-$ and $\mathrm{bj}+$ 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* [12], 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* [8] that has been validated as an accurate Alpha 21264 simulator, and has been incorporated with 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 also be 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

TABLE 2
Workloads and SimPoints Used in Our Experiments

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						

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 two 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 the variables due to the page limit; instead, the following sections will analyze the most important ones. Table 3 explains the abbreviation 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 section 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 b) of Step 2 in Algorithm 1 as important 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 Fig. 2. Note that the values shown have been scaled to a sum of 100 percent, with a higher percentage indicating a stronger influence on the AVF. As can be seen, the number of valid entries (*cumulative_count*, *average_count*) 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) also strongly affects 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

TABLE 3
Explanation of Variable Names

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

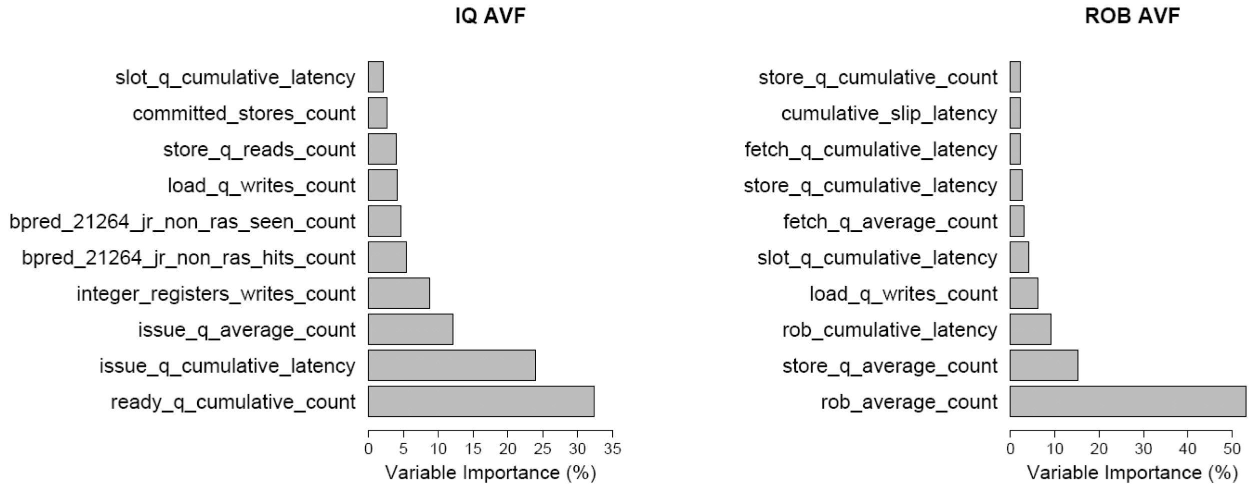

Fig. 2. Relative variable importance (within the same configuration).

variables. This effectively reduces the number of dimensions of the input space from 217 to only 10, thus significantly reducing the model complexity. The prediction results of the workloads in the test set are shown in Fig. 3. Note that in this paper, all the AVF values are shown in the range of 0-100 (rather than 0-1). For the four 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.

For comparison purpose, we also present the relative error for each phase in Fig. 4. As can be seen, most workloads in the test set are predicted with a relative error lower than 10 percent. Only *mcf* is an outlier that suffers from significant relative errors. From the simulation result, we found that its data L1 (DL1) cache miss rate is only 0.0095 in the first SimPoint, while this miss rate is 0.2877 in the second SimPoint. Therefore, the training data from *mcf* does not represent its typical memory-intensive behaviors. We believe that this is the reason why *mcf* shows high relative errors in the prediction.

Generally, we believe that relative error is not a good metric to report the predictive performance in this work. If we intended to look at the average AVF value throughout the entire phase (instead of a variation consisting of many intervals), relative error might be a good choice. However, our work addresses the instantaneous AVF curve that consists of hundreds of measured points within a phase, and some of them are very close to zero. The relative error could reach a very high value even by a small absolute error. For example, if the true AVF measure is 1 and the predicted AVF is 2, although the absolute error is only 1, the relative error is 100 percent. This kind of "outliers," though only a few, strongly affects the average of relative error, but does not reasonably reflect the predictive power of the model. Therefore, we will mainly focus on analyzing the mean absolute error of the AVF in this paper.

Empirical Cumulative Density Function (CDF) is another way to report the prediction performance. From Fig. 5, we see that over 90 percent of the intervals are predicted below absolute errors of 4.5 and 2.2 for the IQ and ROB AVFs, respectively.

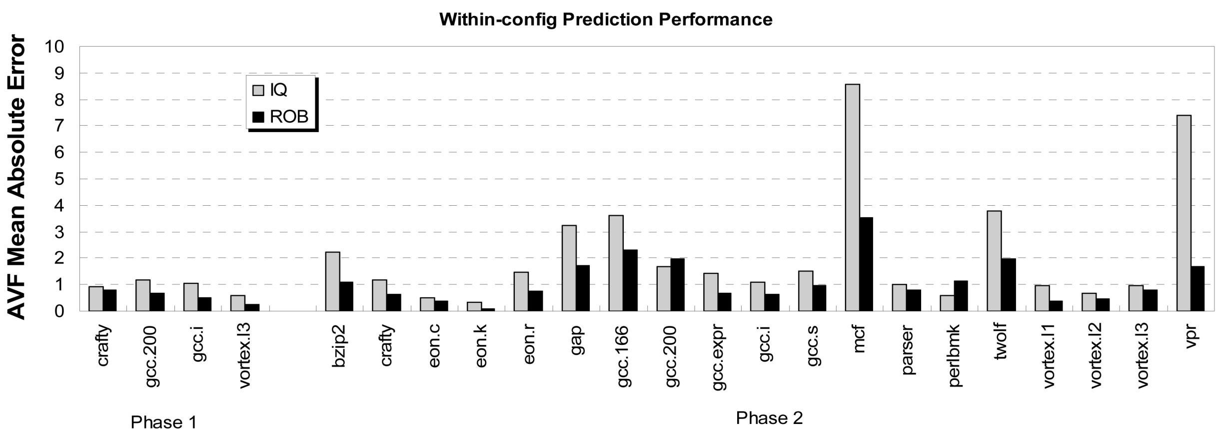

Fig. 3. Prediction results on different workloads (4 phase 1 on the left) and future phases (19 phase 2 on the right).

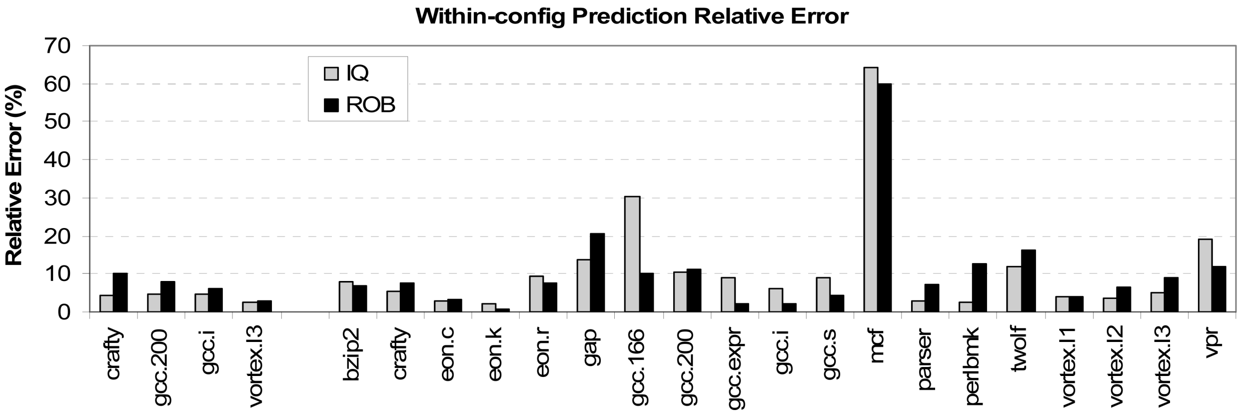

Fig. 4. Prediction relative error.

4.2 Prediction across Different Processor Configurations

The previous section demonstrates a correlation between the AVFs and a small set of performance metrics across workloads and phases but within a specified processor configuration. In this section, we further extend our methodology to address the cross-configuration situation. Applying the well-trained cross-configuration predictive model on different existing configurations enables calculating the AVF without implementing the various hardware required to calculate the AVF in each configuration. As mentioned in Section 1, an analysis window is implemented to store 40K instructions after their commit stage. In order to detect the instruction type which will be used in calculating the AVF, we also need to maintain the instruction dependencies in the window by implementing a large amount of additional structures. Besides, a lot of state bits and control logics are expected in the AVF calculation. Therefore, if we have a cross-configuration predictive model, we can accurately quantify the AVF behaviors on various hardware platforms without actually implementing the AVF calculation designs in hardware, thereby saving large hardware overheads.

Specifically, we tune the four parameters 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 change in 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 three configurations (12 phase files).

Similar to the within-configuration study, we first apply BRT using all $217 + 4$ 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 Fig. 6. Interestingly, the structure's occupant rate becomes the most important variable to its AVF. We also observe that two configuration parameters ($issue\_q\_size, rob\_size$) appear in the lists, indicating that chaning configuration does have some effect in producing the AVF. As can be seen, the variable importance distribution (percentage and ranking) shown in Fig. 6 is quite different from those depicted in Fig. 2. This happens due to the multicollinearity problem in multiple regression models [20]. 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

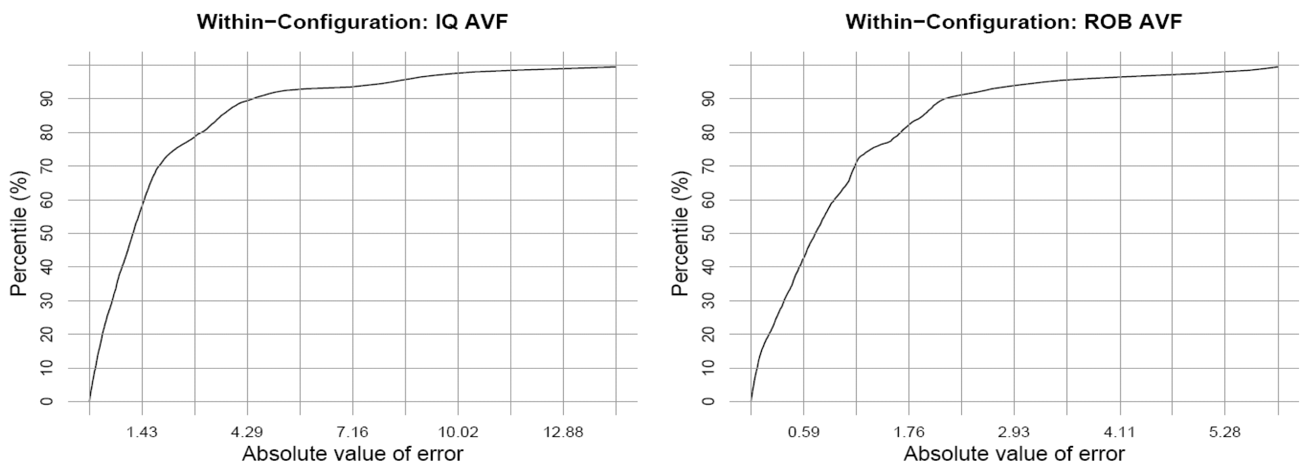

Fig. 5. Empirical CDF on absolute errors in the within-configuration study.

TABLE 4
Configurations Used in Section 4.2

	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*

The training set contains the 48 phase files of cfg1 to cfg12, and the test set (shown in gray) includes the 12 phase files of cfg13 to cfg15.

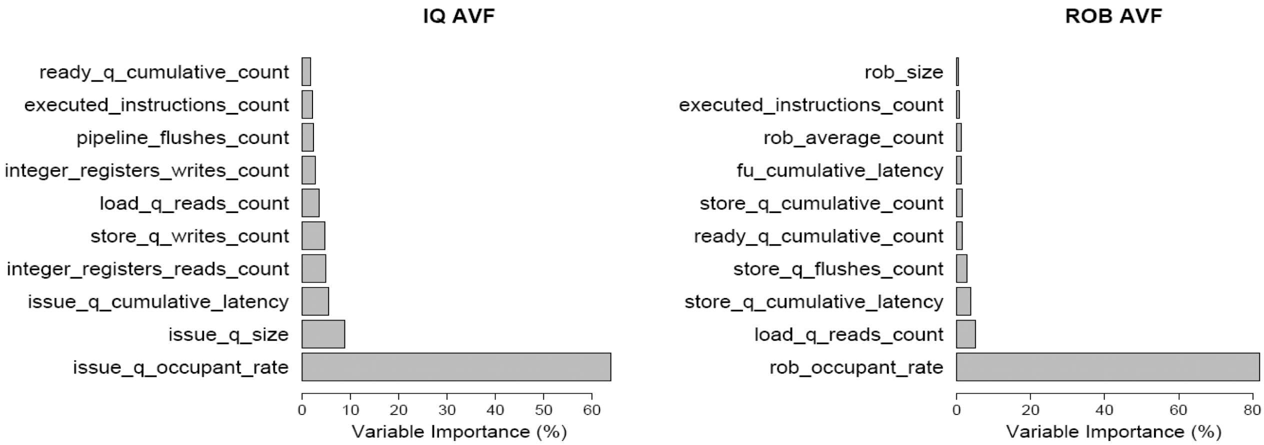

Fig. 6. Relative variable importance (across different configurations).

disguised by its correlated variable(s). For example, *rob_average_count* and *rob_occupant_rate* are two highly corrected variables, but show completely different influences to the response in Figs. 2 and 6.

As for the prediction performance illustrated in Fig. 7, only one workload (*bzip2* under *cfg13*) is predicted with mean absolute errors of AVFs above 3, and the other five workloads in test show very high prediction accuracies in both IQ and ROB AVFs. Specifically, the overall MAEs of all the six 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 six workloads are simulated under three different configurations which also differ from the configurations in the training set. Hence, the accurate prediction results validate that our model is capable of predicting vulnerability behaviors across configurations. In addition, Fig. 8 shows that over 90 percent of the intervals are predicted below absolute errors of 4.6 and 2.1 for IQ and ROB AVFs, respectively.

4.3 AVF Behavior Analysis and Model Interpretation

Fig. 9 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.

In addition, one can refer to Figs. 10 and 11 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, Fig. 10 illustrates how the two most important variables contribute to the IQ AVF in the within-configuration study. The data are plotted into a contour map, which shows two hills in which variation of the parameters results in significant changes of the AVF, and one plateau in which the AVF is insensitive to the variables' changes. As can be seen, when the *issue_q_cumulative_latency* (the Y-axis) is less than $5.8\mathrm{e}+06$, increasing this latency boosts the AVF from 18 to 24. In addition, when the *ready_q_cumulative_count* (the

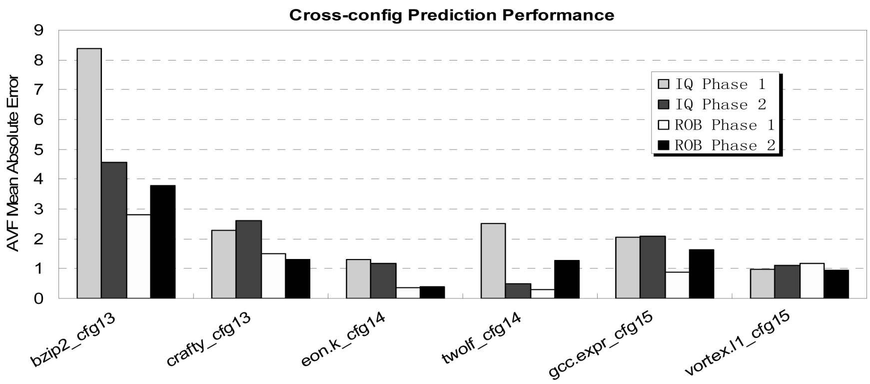

Fig. 7. Prediction results on different configurations.

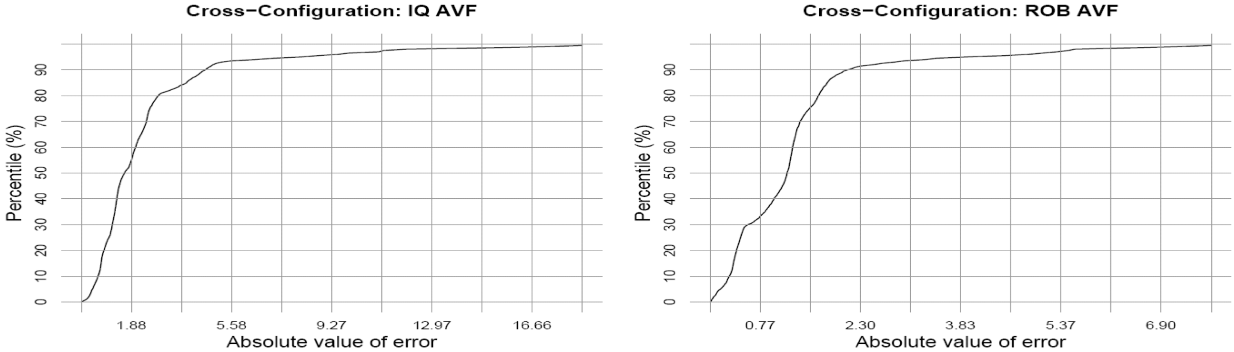

Fig. 8. Empirical CDF on absolute errors in the cross-configuration study.

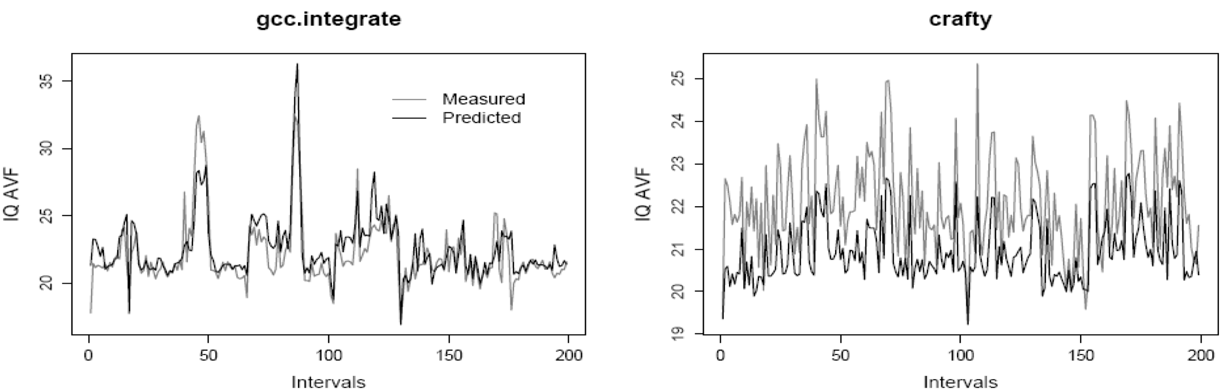

Fig. 9. Measured and predicted IQ AVF curves for *gcc.integrate* and *crafty* in the within-configuration study.

X-axis) is less than 350K, decreasing the cumulative Ready Queue count leads to a further AVF increase from 26 to 36. A larger cumulative IQ latency means that the ACE instructions were kept for a longer time in the IQ. A lower Ready Queue count indicates a worse congestion in the IQ where the issued instructions wait for their operands to be ready. 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. In this case, other processor variables should be considered. In [15], 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. The contribution of the ROB average count to the ROB AVF is shown in Fig. 11. We can observe that the increase of the ROB average count results in the increase 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.

4.4 A Comparison between BRT and Linear Regression

In this section, we 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 [27]. Their proposed practical linear procedure started from including the single variable with the largest correlation into the model. Then by considering all the remaining variables (omitting the selected one) in turn, they

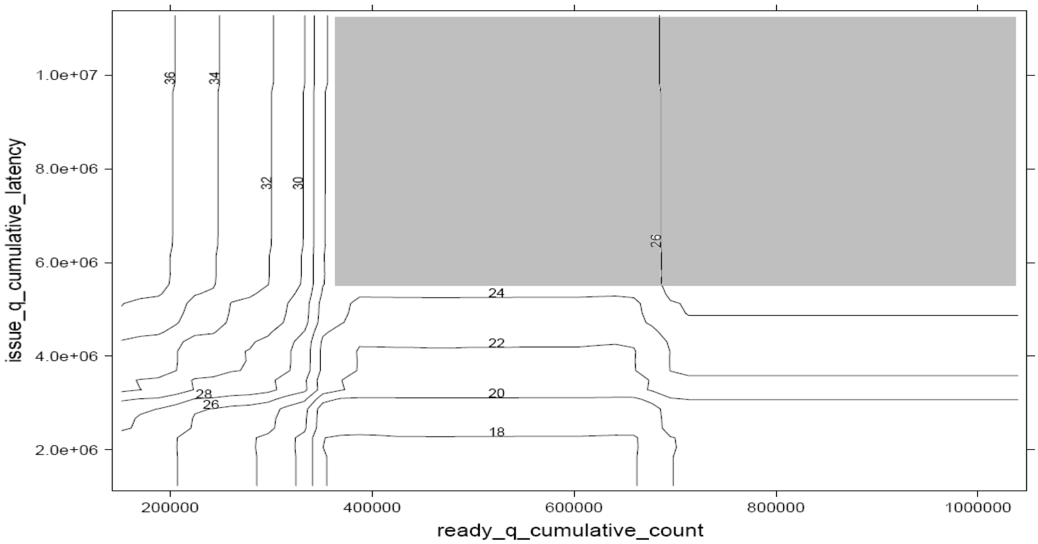

Fig. 10. Partial dependence of the IQ AVF on the two most important variables in the within-configuration study.

selected the one that achieves the best bivariate linear regression approximation involving the previously selected variables. The procedure continues until all variables are included in the model.

Fig. 12 illustrates the comparison. Here, we only consider the IQ AVF prediction for the within-configuration case. The left panel 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 more stable than linear regression. This is also exemplified in the right panel of Fig. 12, which shows the coefficient solution paths along the model size in linear regression. Notice that for "X3," its estimated coefficient begins with a negative value at model size 3, shrinks towards zero at model size 4 & 5, changes to positive at model size 6 & 7, and goes below zero again for model size 8-10.

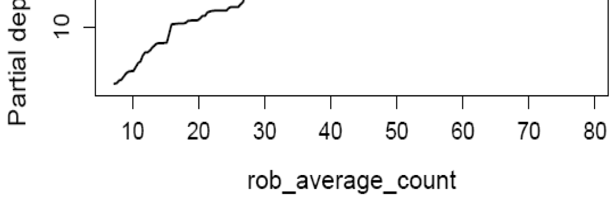

Fig. 11. Partial dependence of the ROB AVF on the most important variable in the within-configuration study.

The instability issue in model fitting and estimation has been well studied in [6], where it pointed out that neural nets, classification and regression trees, and subset selection in linear regression were unstable. Instability refers to the situation that a small change in input data set (i.e., include/exclude one workload or phase) or model setting (such as model size) can result in a large change in the fitted model (i.e., estimate of model coefficient and fitted value). As demonstrated previously, linear regression suffers from such instability issues. One common approach to alleviate instability problem is model averaging. As we mentioned earlier, regression trees are the building blocks for BRT. Although they are highly greedy and instable optimization methods, BRT averages a large number of regression trees with a tiny equal weight $v(v = 0.01)$ on each tree. This stabilizes the estimate from BRT and achieves better prediction performance than a single regression tree. Fig. 13 shows the R-square curves for training and test sets with ν equals to 0.01 and 1 in the BRT model. The benefit of using small value of ν is evident. With smaller value of ν, the test R-square curve reaches a higher value and stays there for many iterations. In other words, our BRT model has successfully stabilized the prediction.

In addition, BRT is a more flexible modeling scheme than linear regression. For example, it considers nonlinear relationship between the AVF measure and input variables, and also addresses complex interactions among input variables. If we consider the nonlinearity and interactions in the linear regression model, we have to specify each term carefully and do the model checking before we got the final model. Our method also has model interpretation (see Section 4.3) which helps us visually analyze the vulnerability trends and bottlenecks. This cannot be done by a linear regression 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

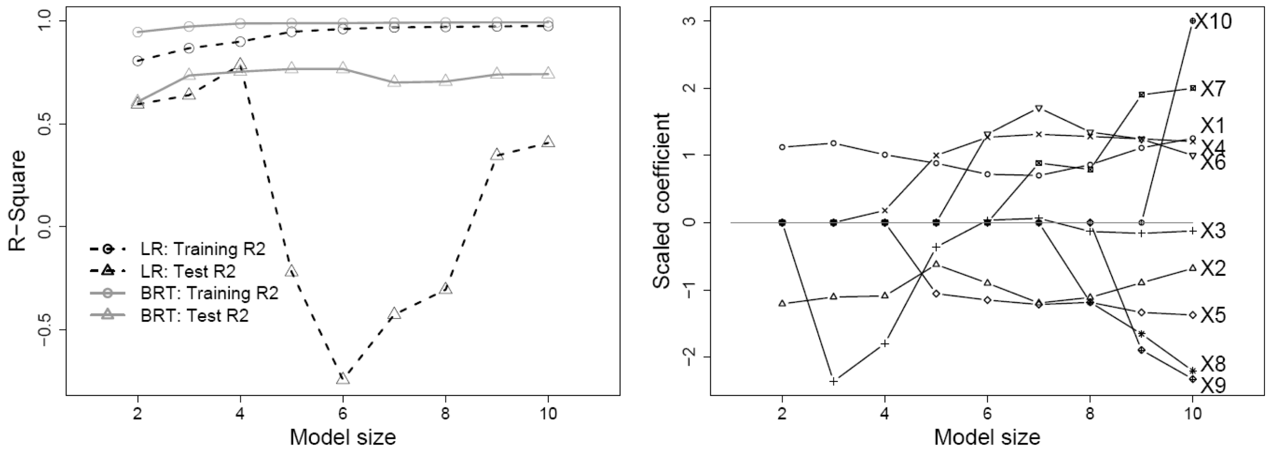

Fig. 12. R-squares on training and test sets over model size in linear regression and BRT methods (Left), and the scaled coefficients for the first 10 selected variables in linear regression (Right).

Section 2.3 to summarize some simple and interpretable "IF-ELSE" rules that can be applied on some important performance variables during runtime 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, but other AVF predictions can also be applied.

The results of fast ROB AVF estimation in the within-configuration study are shown in Fig. 14. We intend to find the top ∼10 percent 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 Fig. 14 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 Fig. 14, 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 from architectural wise: 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

Mukherjee et al. [18] compared the advantages and disadvantages of three different RMT techniques: 1)

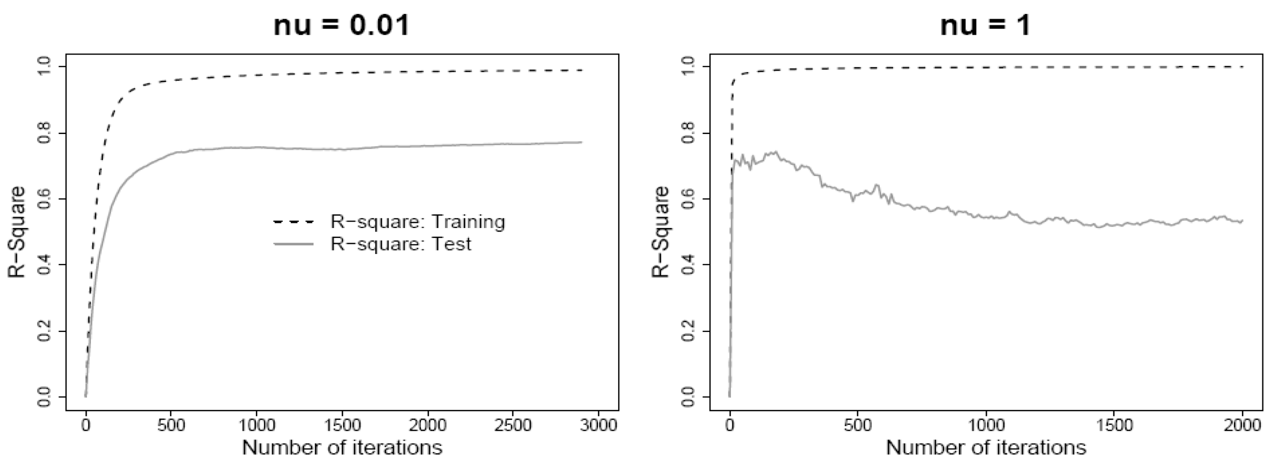

Fig. 13. R-square curves for the training and test sets with different learning rate ν in BRT.

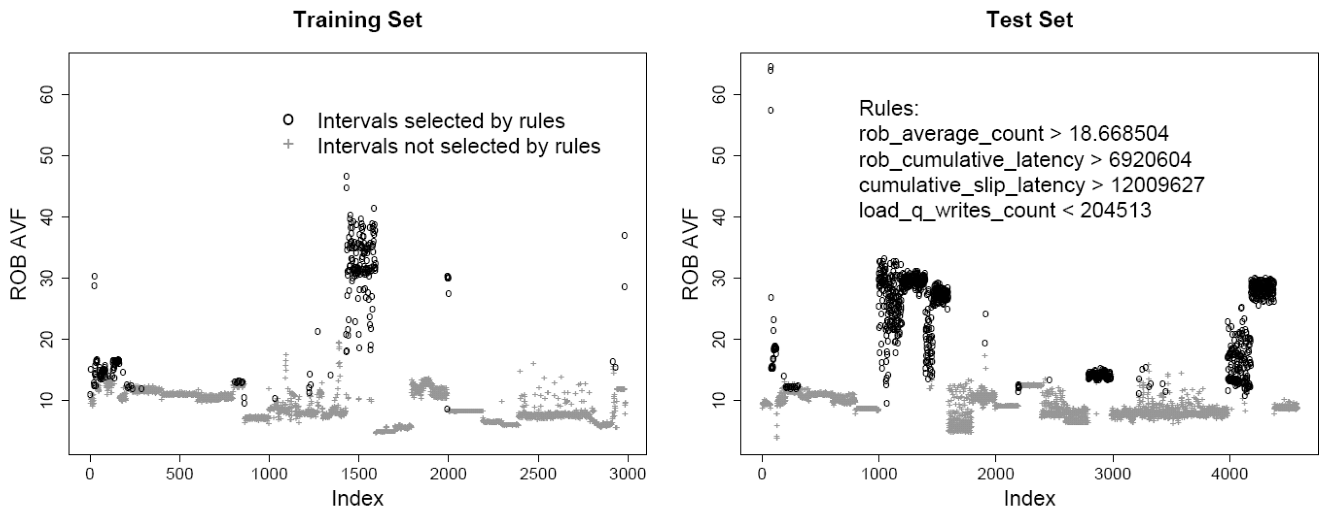

Fig. 14. Fast estimation of the ROB AVF in the within-configuration study.

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 [21], 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. [26] and Gomaa et al. [14] proposed the recovery schemes for SRT and CRT, respectively. Prior to these schemes, Rotenberg [23] and Austin [2] at first proposed different transient fault detection architectures.

The concept of AVF was originally termed in [19], and Austin [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 [4]) lower bound on the reliability level of various processor structures, and has been adopted in many research works on performance models. Fu et al. [13] quantitatively characterized vulnerability phase behavior of four microarchitecture structures based on a system framework proposed in [12], which is also the simulator used in this paper. Zhang et al. [31] performed a similar analysis on SMT architectures. Soundararajan et al. [25] described a simple infrastructure to estimate an upper bound of the ROB AVF, and also proposed two mechanisms (Dispatch Throttling and Selective Redundancy) to restrict 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. [29] implemented a latch-accurate Verilog model to simulate an Alpha processor, while Li et al. [17] incorporated a similar probabilistic model of error generation and propagation into an architecture-level tool. Wang et al. [28] 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 [4] 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 [13], [27], some other works also addressed the problem of AVF prediction at runtime. Cho et al. [7] 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. [16] developed an algorithm to estimate processor structures' vulnerability online using a modified error injection and propagation scheme from their previous work [17]. 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. In [9], we proposed the versatile AVF prediction method across different workloads, execution phases, and processor configurations, and fast estimation to identify intervals with high AVF values. This paper extends [9] by adding a stability comparison between the proposed BRT method and the conventional linear regression model and an application of the usage of the PRIM-based AVF estimation.

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. A quantitative comparison between the BRT model and linear regression demonstrates that our scheme is more stable and has many advantages. 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. The case study performed in the paper also justifies the applicability of our fast AVF estimation scheme.

Appendix A
Case Study: PRIM-Based ROB Redundancy

This section describes a possible application of the PRIM-based scheme introduced in Section 5 to efficiently bind the ROB AVF value under a system-affordable low level. While most intervals shown in Fig. 14 demonstrate a very good invulnerability to transient faults in the ROB, some of them (around 10 percent in the entire workload space) do have a relatively high ROB AVF that may result in a failure in system reliability requirements. However, as can be seen in the figure, our PRIM-based model is able to effectively identify such vulnerable intervals, in which we can essentially partition the ROB into two identical halves that execute the same instruction flows, generating necessary redundancy to detect possible ROB transient faults. Similar studies can be found in [27], [25], but ours does not bound the AVF below some prespecified threshold; instead, we systematically analyze the vulnerability distribution in the entire workload space, and generate some simple rules that select the intervals of interest, i.e., the vulnerable intervals showing a high AVF. As a proper extension of Section 5, this section mainly focuses on reducing the ROB AVF in the within-configuration study, but the approach can be easily applied to the cross-configuration situation or other structures, e.g., the IQ.

A.1 PRIM-Based ROB Redundancy Approach

The basic idea of our approach is that whenever the program execution finishes an interval whose measured performance metrics conform to the trained PRIM rules (i.e., the interval is declared to be vulnerable in the ROB), we partition the ROB into two identical halves, both running the same instruction flows from the next interval. This is defined to be in a *redundant mode*, during which the ROB AVF is effectively reduced to zero due to the generated redundancy. After a fixed number of intervals, the ROB will switch back to a *normal mode* which recombines the two parts into the original buffer, and start checking the metrics again. In the following detailed algorithm, $redundancy\_flag$ records how many intervals remain in the redundant mode to reach the next normal mode interval.

Algorithm 2. PRIM-based ROB redundancy.

```
At the beginning of each interval:
  IF  ((redundancy_flag = 0) AND
      (ROB in the redundant mode))
  THEN {
      ROB switches to the normal mode
      }
At the end of each interval:
    IF  ((redundancy_flag = 0) AND
        (current measured metrics conform to the PRIM
        rules))
    THEN {
        ROB switches to the redundant mode
        set redundancy_flag to n
        }
    IF (redundancy_flag > 0)
    THEN {
        decrement redundancy_flag by 1
        }
```

To switch the ROB from the normal mode to the redundant mode, we need to properly process the valid instructions that still remain in the ROB such that the execution correctness is maintained. This can be done via *Fetch Throttling*, which is similar to *Dispatch Throttling* described in [25]. When a vulnerable interval is detected, we temporarily stop fetching instructions until the whole pipeline is drained and the ROB finishes the mode switching. After that the pipeline resumes instruction fetching, and every instruction will be placed in both ROB partitions to generate the redundancy, which endows the ROB with the ability to detect possible transient faults.

When the ROB is in the redundant mode, the effective ROB size reduces to a half of the original size and the PRIM rules that were derived in the normal mode become improper. Therefore, we do not follow those selecting rules under the redundant mode, but simply switch the ROB back to the normal mode after n intervals. In this study, we fix n at 10. Note that this scheme may potentially disable the ROB redundancy even when its AVF of current interval turns out to be high (but will re-enable the redundancy immediately after current interval if this happens), resulting in some "outliers" escaping from our bounding effect. This is considered to be negligible since such outliers rarely present in our 200-interval workload executions.

A.2 Performance Degradation Analysis

Our PRIM-based ROB redundancy may suffer from performance degradation mainly due to two reasons. One is that the effective ROB size is reduced to half in the intervals under the redundant mode, and the other is that we need to throttle fetching instructions to drain the pipeline when the normal-to-redundant mode switching happens. We analyze the performance degradation in this section to demonstrate that our redundancy scheme sacrifices only a negligible amount of performance to provide significant increase in reliability.

The average IPC degradation after applying our ROB redundancy scheme in within-configuration study is 1.9 percent for the 23 workloads in the test set shown in Table 2. Actually, there are only 3 of them showing a

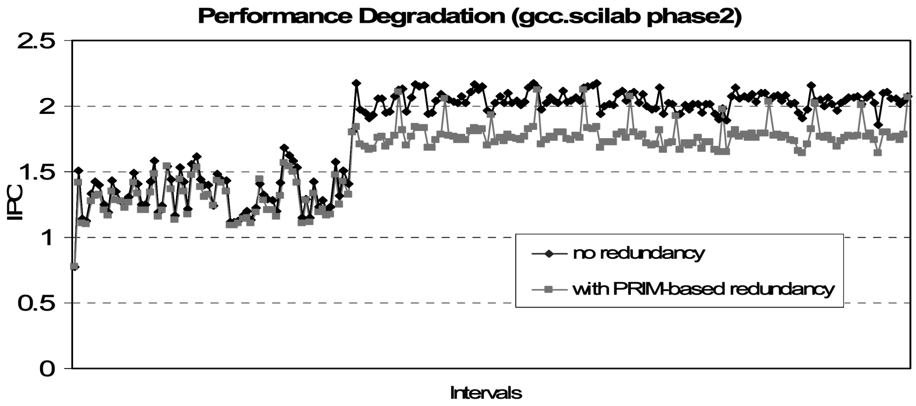

Fig. 15. IPC variation for *gcc.scilab* in phase 2 of both schemes: without ROB redundancy and with PRIM-based ROB redundancy.

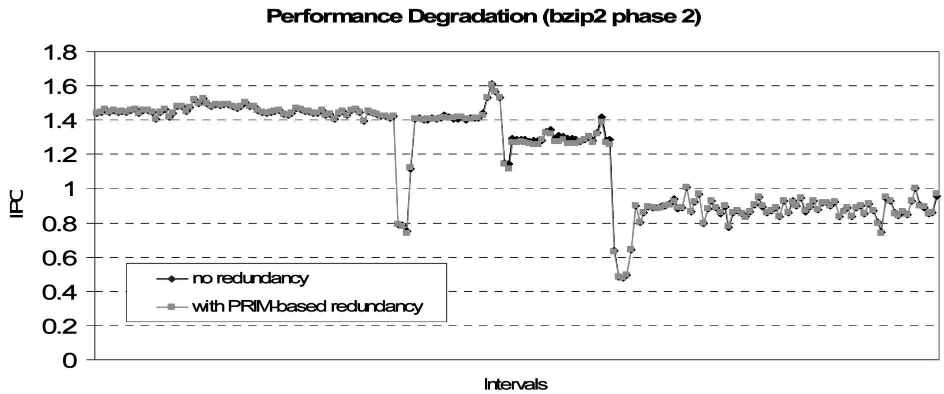

Fig. 16. IPC variation for *bzip* in phase 2 of both schemes: without ROB redundancy and with PRIM-based ROB redundancy.

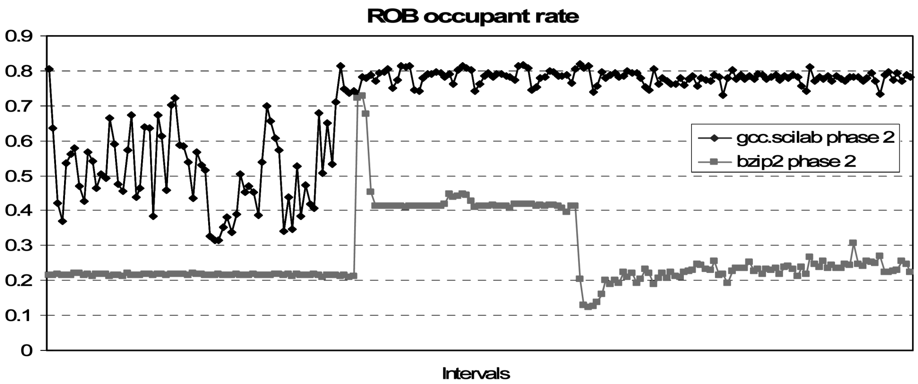

Fig. 17. ROB occupant rate for *gcc.scilab* in phase 2 and *bzip2* in phase 2.

relatively significant IPC decrease, while the rest 20 retain their performance very well. For analysis purpose, we only present two typical examples here: one is from the former group that suffers from performance degradation, and the other is from the latter ones that retain the IPCs.

gcc.scilab (phase 2), whose IPC decreases by 9.3 percent when enabling the redundancy scheme, is one of the three workloads that suffer from a noticeable IPC degradation. Fig. 15 shows its IPC variation before and after applying PRIM-based ROB redundancy. An apparent two-step curve can be seen from the figure: the IPC retains well in the first step but nearly decreases by a constant amount in the second step. One can refer to Fig. 17 for an easy explanation: the ROB occupant rate (for *gcc.scilab*) hangs around 50 percent in the first step but reaches 80 percent in the second step, where the system would be short of the ROB entries if the ROB size is cut to half. In contrast, *bzip2* (phase 2) only presents a 0.2 percent decrease in the IPC, whose variation is depicted in Fig. 16. Note that the two curves are highly overlapped with only a few points in the middle showing slight difference. Fig. 17 also gives the ROB occupant rate for bzip2, and it's not surprised to see a well-retained IPC in this workload since its ROB occupancy persistently stays below 50 percent.

ACKNOWLEDGMENTS

This work is supported in part by the Louisiana Board of Regents grant LEQSF (2006-09)-RD-A-10, NSF (2009)-PFUND-136, 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, 2009.

[2] T. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," *Proc. Int'l Symp. Microarchitecture (MICRO)*, 1999.

[3] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. Mukherjee, and R. Rangan, "Computing Architectural Vulnerability Factors for Address-Based Structures," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2005.

[4] A. Biswas, P. Racunas, J. Emer, and S. Mukherjee, "Computing Accurate AVFs Using ACE Analysis on Performance Models: A Rebuttal," *Proc. Computer Architecture Letters*, vol. 7, 2008.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees.* Wadsworth Int'l Group, 1984.

[6] L. Breiman, "Heuristics of Instability and Stabilization in Model Selection," *Annals of Statistics*, vol. 24, pp. 2350-2383, 1996.

[7] C. Cho, W. Zhang, and T. Li, "Informed Microarchitecture Design Space Exploration Using Workload Dynamics," *Proc. Int'l Symp. Microarchitecture (MICRO)*, 2007.

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

[9] L. Duan, B. Li, and L. Peng, "Versatile Prediction and Fast Estimation of Architectural Vulnerability Factor from Processor Performance Metrics," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA)*, 2009.

[10] J. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *The Annals of Statistics*, vol. 29, pp. 1189-1232, 2001.

[11] J. Friedman and N. Fisher, "Bump Hunting in High-dimensional Data," *Statistics and Computing*, vol. 9, pp. 123-143, 1999.

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

[13] X. Fu, J. Poe, T. Li, and J. Fortes, "Characterizing Microarchitecture Soft Error Vulnerability Phase Behavior," *Proc. Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems (MASCOTS)*, 2006.

[14] M. Gomaa, C. Scarbrough, T. Vijaykumar, and I. Pomeranz, "Transient-Fault Recovery for Chip Multiprocessors," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2003.

[15] B. Lee and D. Brooks, "Roughness of Microarchitectural Design Topologies and Its Implications for Optimization," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA)*, 2008.

[16] X. Li, S. Adve, P. Bose, and J. Rivers, "Online Estimation of Architectural Vulnerability Factor for Soft Errors," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2008.

[17] X. Li, S. Adve, P. Bose, and J. Rivers, "SoftArch: An Architecture-Level Tool for Modeling and Analyzing Soft Errors," *Proc. Int'l Conf. Dependable Systems and Networks (DSN)*, 2005.

[18] S. Mukherjee, M. Kontz, and S. Reinhardt, "Detailed Design and Evaluation of Redundant Multithreading Alternatives," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2002.

[19] 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," *Proc. Int'l Symp. Microarchitecture (MICRO)*, 2003.

[20] J. Neter et al., *Applied Linear Statistical Models*, fourth ed., McGraw-Hill/Irwin, Feb. 1996.

[21] S. Reinhardt and S. Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2000.

[22] G. Reis, J. Chang, N. Vachharajani, R. Rangan, D. August, and S. Mukherjee, "Design and Evaluation of Hybrid Fault-Detection Systems," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2005.

[23] E. Rotenberg, "AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessor," *Proc. Fault-Tolerant Computing Systems (FTCS)*, 1999.

[24] T. Sherwood et al., "Automatically Characterizing Large Scale Program Behavior," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.

[25] N. Soundararajan, A. Parashar, and A. Sivasubramaniam, "Mechanisms for Bounding Vulnerabilities of Processor Structures," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2007.

[26] T. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient-Fault Recovery Using Simultaneous Multithreading," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2002.

[27] K. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic Prediction of Architectural Vulnerability from Microarchitectural State," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2007.

[28] N. Wang, A. Mahesri, and S. Patel, "Examining ACE Analysis Reliability Estimates Using Fault-Injection," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2007.

[29] N. Wang, J. Quek, T. Rafacz, and S. Patel, "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," *Proc. Int'l Conf. Dependable Systems and Networks (DSN)*, 2004.

[30] C. Weaver, J. Emer, S. Mukherjee, and S. Reinhardt, "Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2004.

[31] W. Zhang et al. "An Analysis of Microarchitecture Vulnerability to Soft Errors on Simultaneous Multithreaded Architectures," *Proc. Int'l Symp. Performance Analysis of Systems and Software (ISPASS)*, 2007.

[32] J. Ziegler et al. "IBM Experiments in Soft Fails in Computer Electronics (1978-1994)," *IBM J. Research and Development*, vol. 40, no. 1, pp. 3-18, 1996.

Bin Li received the bachelor's degree in biophysics from Fudan University, China. He obtained the master's degree in biometrics in August 2002, and the PhD degree in statistics in August 2006 from the Ohio State University. He joined the Experimental Statistics Department at Louisiana State University as an assistant professor in August 2006. His research interests include statistical learning & data mining, statistical modeling on massive and complex data, and Bayesian statistics. He received the Ransom Marian Whitney Research Award in 2006 and a Student Paper Competition Award from ASA on Bayesian Statistical Science in 2005. He is a member of the Institute of Mathematical Statistics (IMS) and American Statistical Association (ASA).

Lide Duan received the bachelor's degree in computer science and engineering from Shanghai Jiao Tong University, China, in June 2006. He is currently a PhD student in the Department of Electrical and Computer Engineering, Louisiana State University. His research interests include processor reliability characterization, high-performance and reliable processor design.

Lu Peng received the bachelor's and master's degrees in computer science and engineering from Shanghai Jiaotong University, China. He obtained the PhD degree in computer engineering from the University of Florida in Gainesville in April 2005. He joined the Electrical and Computer Engineering Department at Louisiana State University as an assistant professor in August 2005. His research focus is on memory hierarchy system, reliability, power efficiency and other issues in CPU design. He also has interests in network processor. He received an ORAU Ralph E. Powe Junior Faculty Enhancement Award in 2007 and a Best Paper Award from IEEE International Conference on Computer Design in 2001. He is a member of the ACM and the IEEE Computer Society.