IEICE TRANS. INF. & SYST., VOL.E95–D, NO.1 JANUARY 2012

PAPER Special Section on Trust, Security and Privacy in Computing and Communication Systems

Accurate and Simplified Prediction of L2 Cache Vulnerability for Cost-Efficient Soft Error Protection

Yu CHENG[†a)], **Anguo MA**[†], ***Student Members***, ***and*** **Minxuan ZHANG**[†], ***Nonmember***

SUMMARY Soft errors caused by energetic particle strikes in on-chip cache memories have become a critical challenge for microprocessor design. Architectural vulnerability factor (AVF), which is defined as the probability that a transient fault in the structure would result in a visible error in the final output of a program, has been widely employed for accurate soft error rate estimation. Recent studies have found that designing soft error protection techniques with the awareness of AVF is greatly helpful to achieve a tradeoff between performance and reliability for several structures (i.e., issue queue, reorder buffer). Considering large on-chip L2 cache, redundancy-based protection techniques (such as ECC) have been widely employed for L2 cache data integrity with high costs. Protecting caches without accurate knowledge of the vulnerability characteristics may lead to the over-protection, thus incurring high overheads. Therefore, designing AVF-aware protection techniques would be attractive for designers to achieve a cost-efficient protection for caches, especially at early design stage. In this paper, we propose an improved AVF estimation framework for conducing comprehensive characterization of dynamic behavior and predictability of L2 cache vulnerability. We propose to employ Bayesian Additive Regression Trees (BART) method to accurately model the variation of L2 cache AVF and to quantitatively explain the important effects of several key performance metrics on L2 cache AVF. Then we employ bump hunting technique to extract some simple selecting rules based on several key performance metrics for a simplified and fast estimation of L2 cache AVF. Using the simplified L2 cache AVF estimator, we develop an AVF-aware ECC technique as an example to demonstrate the cost-efficient advantages of the AVF prediction based dynamic fault tolerant techniques. Experimental results show that compared with traditional full ECC technique, AVF-aware ECC technique reduces the L2 cache access latency by 16.5% and saves power consumption by 11.4% for SPEC2K benchmarks averagely.

key words: *architectural vulnerability factor (AVF), AVF modeling and prediction, AVF-aware protection, cost-efficient, L2 cache*

1. Introduction

Cache has been a major design point for microprocessors, and cache data integrity has been an important design concern for microprocessors [1]–[6]. Errors in cache can easily propagate through the system and lead to data integrity issues [5], [6]. Soft errors, cause by external radiation events, are the largest contributors to the vulnerability of cache, and the situation is expected to worsen with aggressive technology scaling [7]–[9]. Moreover, researches have found that soft error rates of cache are projected to increase linearly with cache sizes [10], [11]. A larger cache provides improved performance, but comes at the expense of increased vulnerability to soft errors. Inside a processor, since L2 cache is the largest component and its size is become larger, L2 cache is the most vulnerable on-chip component and suffers from increasing vulnerability to soft errors.

Redundancy-based techniques, such as parity and Error Correction Codes (ECC), are the most commonly used fault tolerant techniques for cache. And ECC has been widely used to protect L2 cache in modern commercial microprocessors [12]–[18]. Although ECC maintains a desired reliability goal for L2 cache, it incurs significant area, performance and power costs. Researches have found that ECC has incurred the "one process generation penalty" (approximately 2X scaling in area, speed and power) [19]. Protecting L2 cache in a cost-effective way is increasingly attractive for designers.

A variety of techniques have been proposed to reduce the costs of ECC [20]–[24]. However, all these techniques assumed that the probability a soft error would result in an erroneous program outcome (i.e., AVF [9]) is 100%, and therefore provide full ECC protection throughout the entire execution lifetime of programs. However, this level of full ECC protection may be not necessary at all execution points, since AVF of several structures has been demonstrated to vary significantly during the lifetime of programs, and many execution points exhibit extremely low level of vulnerability [25], [26]. For this reason, protecting structures without accurate knowledge of their dynamic vulnerability characteristics may lead to the over-protection and incur significant costs. Hence, it is necessary to develop novel cost-effective fault tolerant techniques with the awareness of vulnerability characteristics.

Recently, researchers have investigated the dynamic AVF behavior of several important structures, and have created AVF predictors based on several key performance metrics. Furthermore, they have incorporated the AVF predictors into the dynamic fault tolerant system design, so as to meet the reliability goal with minimum performance penalty [26]–[28]. To our knowledge, they only focused on a few given micro-architecture structures, e.g., issue queue (IQ), reorder buffer (ROB) and register file, not including caches.

The focus of this work is on developing fast and accurate AVF prediction for delay and energy efficient L2 cache design. Firstly, we develop an improved cache AVF estimation framework for better quantitative measurement of cache vulnerability to soft errors. Then we characterize the dynamic AVF behavior of L2 cache and investigate the correlations between the L2 cache AVF and several

Manuscript received March 16, 2011.

[†]The authors are with the National Laboratory for Parallel and Distributed Processing, School of Computer, National University of Defense Technology, China.

a) E-mail: y.cheng@nudt.edu.cn

DOI: 10.1587/transinf.E95.D.56

key performance metrics. Furthermore, by using Bayesian Additive Regression Trees (BART) method [29] and bump hunting technique [30], we develop an accurate and fast L2 cache AVF prediction mechanism across different execution phases of SPEC2K benchmarks. This facilitates the implementation of the AVF-aware dynamic fault tolerant technique which is based on accurate L2 cache AVF prediction.

In summary, the main contributions of this paper are as follows:

1. We propose a novel fine-grained lifetime analysis and post-committed analysis combined method to improve the accuracy of cache AVF estimation. We integrate the method into a general simulator and develop an improved AVF estimation framework, SS-SERA (Soft Error Reliability Analysis based on SimpleScalar). By using the AVFs computed by SS-SERA, we perform a rigorous characterization of dynamic behavior and predictability of L2 cache AVF across different execution phases of SPEC2K programs. Our analysis results suggest that it is inadequate to use simple predictive models based on several performance metrics for accurate L2 cache AVF prediction.

2. We propose to use BART method for accurate L2 cache AVF prediction across different execution phases of SEPC2K programs with different train/test splits. Our results show that BART not only makes an accurate AVF prediction, but also quantifies the dependence of AVF on various performance metrics. In addition, we conduct a comprehensive comparison between BART and other competitive predictive methods (i.e., the linear regression model and boosted regression trees (BRT)) to quantitatively validate the superiority and robustness of BART in L2 cache AVF prediction across different test/train splits and different model sizes.

3. We further employ bump hunting technique to obtain a simplified and fast estimation of L2 cache AVF, thus enabling a feasible online L2 cache vulnerability monitor to identify the execution intervals of high vulnerability. To further confirm the utility of the AVF predictor, we present an AVF-aware ECC technique that provides ECC protection only for the points of high L2 cache AVF. Experimental results show that compared with traditional full ECC technique, AVF-aware ECC technique achieves an average reduction of L2 cache access latency and power consumption by 16.5% and 11.4% for SPEC2K programs respectively.

Throughout this paper, we assume L2 cache adopts a write-back policy. This is because for bandwidth reasons, write-through policy is not a good choice for L2 cache, and most current L2 caches employs a write-back policy [12]–[14]. The remainder of this paper is organized as follows. Section 2 describes the improved cache AVF computing method and the SS-SERA framework. Section 3 describes experimental setup and analyzes the dynamic characteristics of L2 cache. Section 4 introduces the BART method and performs an accurate L2 cache AVF prediction using BART. Section 5 proposes the simplified and fast L2 cache AVF estimation and the case study of AVF-aware fault tolerant technique. Section 6 discusses the related work. Finally, we conclude in Sect. 7.

2. The SS-SERA Framework

In order to quantify cache vulnerability accurately, we propose a novel fine-grained lifetime analysis and post-committed analysis combined method to improve the accuracy of cache AVF estimation. Then we integrate the improved AVF computing method into a popular simulator (i.e., SimpleScalar [31]), and develop an architectural level soft error reliability analysis framework SS-SERA (Soft Error Reliability Analysis based on SimpleScalar) for more general and accurate estimation of cache AVF. In this section, we describe the improved cache AVF computing method used in SS-SERA.

2.1 Cache AVF Computation Equation

In SS-SERA, we first assume that all bits are ACE bits, and then identify the un-ACE bits of structures as many as possible. This method was also adopted in [9], [32]. For cache structures, lifetime analysis [33] method is employed to divide a bit's lifetime into ACE, un-ACE and unknown components according to the activities occurring during the lifetime of the bit (i.e., "idle", "fill", "read", "write", "evict"). For example, for a write-back data cache, fill-to-read period of a bit is identified as ACE, because any fault occurred on the bit during this period would cause erroneous data read.

The granularity at which we maintain the lifetime analysis can have a big impact on data array's AVF. Empirically, we adopt a fine-grained lifetime analysis method and maintain the lifetime information on a per-byte basis for cache data array. Therefore, AVF of a data array is the fraction of all bytes' lifetime during which the bytes are in the ACE state, computed by Eq. (1).

$$AVF = \frac{\sum_{i=1}^{N_{bytes}} ACE_i}{N_{bytes} * total\_exec\_cycles} \quad (1)$$

N_{bytes} is the total number of bytes of a data array. ACE_i represents the total residency cycles of byte i in ACE state.

2.2 Improved Cache AVF Computing Method

In SERA, AVFs of cache data arrays are computed using Eq. (1). We propose a novel fine-grained lifetime analysis and post-committed analysis combined method to identify more un-ACE bytes of cache data arrays.

First of all, since the basic access unit of data cache is byte, bytes in the same cache line may be in different state. For example, some bytes in a dirty cache line may be clean. However, during the eviction of a dirty cache line, all bytes in the dirty cache line are treated equally to be written back to the next cache level, this would induce an overestimation of cache AVF. Considering two bytes in the same cache line (i.e., A and B), if only A is written into, then the fill-to-evict period of B becomes ACE due to the inevitable written back

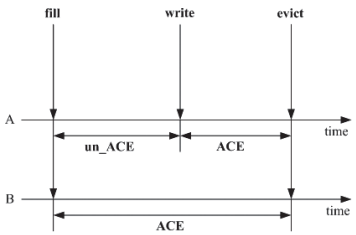


Fig. 1 Potential ACE condition.

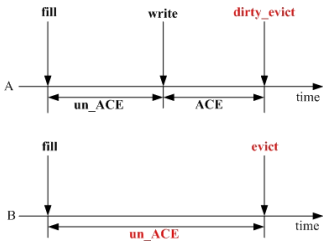


Fig. 2 Eliminating potential ACE.

of the entire cache line to the next cache level, shown as Fig. 1.

In SERA, we propose a novel fine-grained lifetime analysis method to identify more un-ACE bytes. In detail, we add "dirty" bits associated with different portions of a cache line, and identify the modified bytes according to their dirty bits values. When a byte is modified, its "dirty" bit is set to 1. Consequently, we distinguish the eviction of a byte into "dirty_evict" and "evict". "dirty_evict" represents the eviction of modified bytes, and "evict" denotes the eviction of clean bytes. If the "dirty" bit of a byte is set to 0, eviction of the clean byte would not result in a written back event. Then the above fill-to-evict period of B in Fig. 1 is always viewed as un-ACE, shown as Fig. 2.

Secondly, since there are un-ACE instructions which contain un-ACE operands, not every read/write would affect the final program output. We employ post-committed analysis [9] to identify instructions which generate un-ACE reads/writes, and then combine lifetime analysis with post-committed analysis to identify the un-ACE periods of cache lifetime due to the un-ACE reads/writes. Note: periods between any activities and write (e.g., read-to-write) are always un-ACE, so whether writes are ACE or un-ACE could not affect the accuracy of cache AVF computation. Hence, we only consider the effects of un-ACE reads in cache AVF estimation. Figure 3 shows different ACE and un-ACE lifetime divisions due to different orders of un-ACE reads occurring during the cache lifetime.

We can see that since write-to-dirty_evict is always ACE (write is the last write), any reads (no matter ACE or un-ACE) between this period are ACE, shown as Fig. 3 (a) and (b). As Fig. 3 (c) and (d) shows, the last ACE read after write is ACE, and the remaining un-ACE reads/writes to the next write are un-ACE. The period between the last ACE read and the last un-ACE read (shown in red color in Fig. 3) should be identified as ACE. However, using our improved

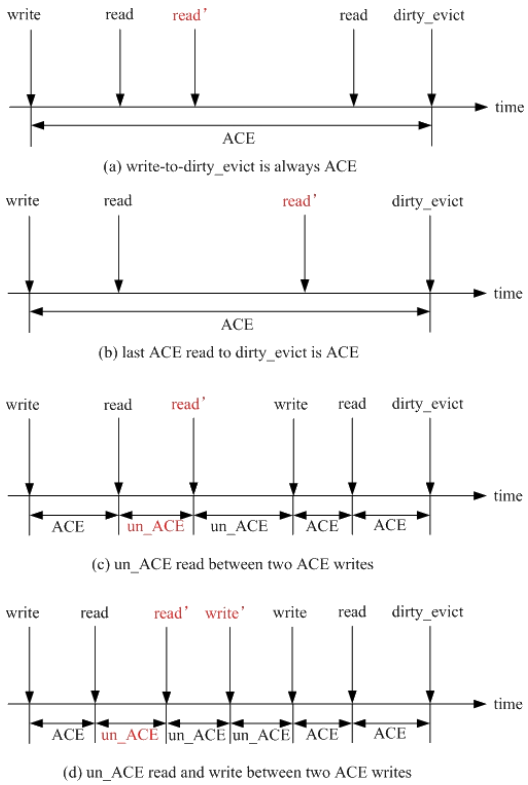


Fig. 3 Different ACE and un-ACE lifetime divisions due to different orders of un-ACE reads. read' and write' are un-ACE read and un-ACE write respectively.

AVF analysis method, this period is classified as an un-ACE component.

3. Characterization of Dynamic L2 Cache AVF Behavior

3.1 Experimental Setup

All of the experiments are conducted with SS-SERA. The baseline configuration of SS-SERA is listed in Table 1. SPEC2K INT and FP benchmarks compiled for the Alpha ISA are evaluated, and each of them is run for multiple 100-Million instruction SimPoints [34]. To simulate a SimPoint, we should fast forward $Num_{fastforward}$ instructions, calculated using Eq. (2):

$$Num_{fastforward} = (SN_{SimPoint} - 1) * Size_{SimPoint} \quad (2)$$

$SN_{SimPoint}$ is the serial number of the SimPoint in a program, $Size_{SimPoint}$ is the size of the SimPoint (i.e., 100 M). For example, for SimPoint1827, simulation of the SimPoint starts just after fast forwarding $1826 * 100$ M instructions from the start of program.

Each SimPoint is partitioned into 25 intervals of 4 million instructions. We have proposed to employ checkpoint mechanism and cooldown technique in SS-SERA to guarantee the correctness and to improve the accuracy of cache AVF computation of small intervals. The detail of

Table 1 Baseline configuration.

Parameters	Values
Fetch/Decode/Issue/Commit width	8
Fetch queue size	16
Reorder buffer size	128
Load/store queue size	64
Integer ALUs/multipliers	6/2
FP ALUs/ multipliers	4/2
L1 D-cache	64 KB, 4-way, 32B line-size
L1 I-cache	64 KB, 4-way, 32B line-size
Unified L2 cache	512 KB, 4-way, 64B line-size
ITLB	128 entries, 4-way set-associative
DTLB	256 entries, 4-way set-associative
TLB miss-latency	30 cycles
Main memory latency	200 cycles

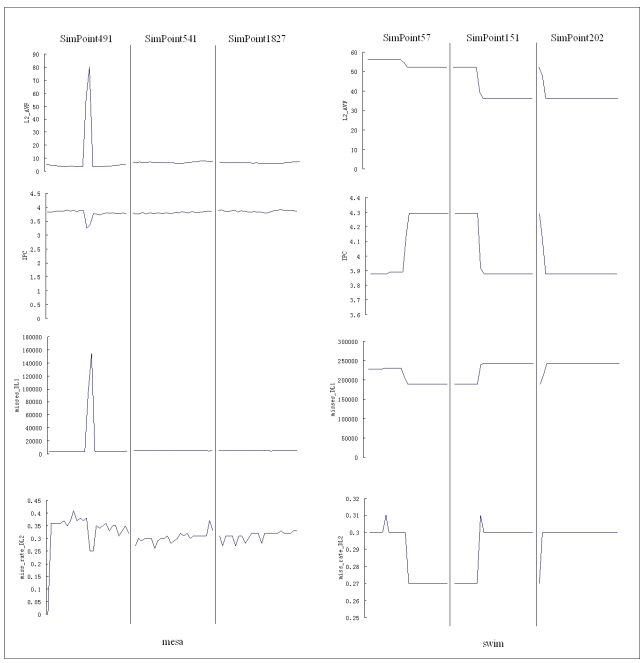


Fig. 4 Profile of time-varying L2 cache AVF and several performance metrics.

our schemes for estimating cache AVFs of small intervals is discussed in another paper. For L2 cache AVF estimation of small intervals, we have reduced the unknown components of L2 cache AVF computation to be less than 1% using appropriate cooldown period sizes. Besides, for SPEC2K benchmarks, the unknown components are shown to be mostly un-ACE, with little increase for AVF [33]. We believe L2 cache AVF computation with unknown component less than 1% is accurate enough to represent the vulnerability of L2 cache.

3.2 Time-Varying Behavior of L2 Cache

Figure 4 shows the profile of runtime L2 cache AVF and several key performance metrics across several SimPoints of two benchmarks (i.e., mesa and swim). Other SimPoints and benchmarks exhibit similar time-varying behavior and their results are omitted for brevity. Each 100M-sized SimPoints contains 25 points of plots which represent the L2 cache AVF and performance information of 4M-sized intervals.

Table 2 Variance of L2 cache AVF.

benchmarks	WCoV	benchmarks	WCoV
gzip_graphic	0.14	vortex_one	0.19
gzip_log	0.16	vortex_two	0.33
gzip_program	0.29	vortex_three	0.09
gzip_random	0.19	crafty	0.09
gzip_source	0.29	ammp	0.79
vpr_route	0.21	applu	0.88
gcc_166	0.16	apsi	0.62
gcc_200	0.28	art_110	0.09
gcc_expr	0.23	art_470	0.17
gcc_integrate	0.3	equake	0.72
gcc_scilab	0.33	facerec	0.23
mcf	0.46	fma3d	0.72
parser	0.47	galgel	0.33
perlbmk_diffmail	0.17	lucas	0.02
perlbmk_makerand	0.23	mgrid	0.15
perlbmk_perfect	0.036	sixtrack	0.32
perlbmk_splitmail	0.41	swim	0.23
gap	0.2	wupwise	0.60
		mesa	0.64

3.2.1 L2 Cache AVF Variation

Firstly, experiment results show that L2 cache AVF exhibits different varying characteristics across different execution phases of the same program. Take mesa for example, L2 cache AVF of SimPoint491 varies in a much larger range (3.55%-80.11%) than SimPoint541 (6.01%-7.87%).

Secondly, we can see that L2 cache AVF also exhibits different degrees of variations across different execution phases of different programs. For instance, variation of L2 cache AVF is large for SimPoint151 of swim (36.18%-52.26%), but is much small for SimPoint1827 of mesa (5.69%-7.47%).

Table 2 lists the variance of L2 cache AVF for SEPC2K programs. Variance of L2 cache AVF is represented by Weighted Coefficient of Variation (WCoV). Coefficient of Variation (CoV) is the standard deviation divided by the mean. Since each SimPoint of a program has an associated weight, WCoV of a program is the sum of the weighted CoV of each SimPoint, computed as Eq. (3). Lower WCoV indicates that execution points of a program exhibit more similar AVF behavior.

$$WCoV_{program\text{“}x\text{”}} = \sum_{s_i \in program\text{“}x\text{”}} CoV_{s_i} * weight_{s_i} \qquad (3)$$

Here, $CoV_{s_i} = sd_{s_i}/mean_{s_i}$. s_i represents one of the representative SimPoints of program “x”. sd_{s_i} is the standard deviation of L2 cache AVF for s_i, and $mean_{s_i}$ is the average L2 cache AVF of s_i.

3.2.2 Correlations between L2 Cache AVF and Performance Metrics

Although L2 cache AVF exhibits significant variation dur-

ing program execution lifetime, there still exist fuzzy correlations between L2 cache AVF and several important performance metrics.

As the most frequently used metric for time-varying performance and vulnerability predictions [25], [26], [35], IPC exhibits a fuzzy relationship with L2 cache AVF. Figure 4 shows that IPC has either positive or negative effects on L2 cache AVF for different SimPoints. The curves of IPC bear a resemblance to the profile of L2 cache AVF on SimPoint541/1827 of mesa and SimPoint151/202 of swim, whereas change in an opposite direction to the L2 cache AVF profile on SimPoint491 of mesa and SimPoint57 of mesa. We can see that residence time of the ACE bytes and total execution cycles of a program in Eq. (1) are both affected by IPC, indicating that IPC definitely has significant influence on L2 cache AVF. Whether IPC shows positive or negative effects on L2 cache AVF, it further depends on the memory access characteristics of programs.

Except for IPC, the profile of misses_DL1 accords with the profile of L2 cache AVF for mesa, but is not completely identical with the profile of L2 cache AVF for swim. We also notice that the curves of miss_rate_DL2 change in an opposite direction to the L2 cache AVF on SimPoint151/202 of swim, yet bear a weak similarity to the profile of L2 cache AVF on SimPoint57 of swim. For mesa, the profile of miss_rate_DL2 changes irregularly.

In summary, we conclude that it is inadequate to use only two or three performance metrics to track L2 cache AVF accurately, thus motivating us to characterize L2 cache AVF behavior within a larger space of performance metrics. Consequently, it is necessary to employ a powerful predictive method to model the variations of L2 cache AVF and to predict L2 cache AVF accurately.

4. L2 Cache AVF Prediction

Several predictive modeling approaches have been developed in the past few years [29], [36]–[39]. Sum-of-trees based predictive method, which is an ensemble technique, fits a large number of tree models and combines them for prediction. Compared to other models, such as a single regression tree model, the sum-of-trees model is more flexible and adaptive. Random forests [38], boosting regression trees (BRT) [39], and Bayesian additive regression trees (BART) [29] are typical sum-of-trees based methods, which use different patterns of tree models fitting.

In our study, we propose to use BART method for accurate L2 cache AVF prediction. Furthermore, the basic features of BART, including model-free variable selection and partial dependence functions, are used to achieve a better interpretation and visualization of BART model. We use these features of BART to better understand the relationships between L2 cache AVF and various performance metrics.

4.1 BART Predictive Model

BART consists of three essential parts: the sum-of-trees model, the regulation prior and the backfitting MCMC (Markov Chain Monte Carlo) algorithm. As a representative sum-of-trees based predictive method, each of the trees in BART explains a small part of the overall model. A regulation prior shrinks each tree to be small and simple, preventing the effect of individual tree from being overly influential. Besides, BART uses Bayesian backfitting MCMC algorithm to iteratively sample from the posterior distribution, in order to achieve a convergent fitting quickly. The induced samples obtained during the backfitting procedure can provide a variety of inferential quantities of interest, such as partial dependence functions and variable selection. We explain the construction of BART model and its basic features in this section.

4.1.1 Fitting of BART Model

Considering a fundamental predictive problem in which an dependent variable Y needs to be predicted from a p-dimensional vector of input variables $X = (x_1, x_p)$:

$$Y = f(X) + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2) \tag{4}$$

For BART, $f(X)$ in Eq. (4) is approximated by a summation of regression trees, computed by Eq. (5):

$$f(X) = \beta_0 + \sum_{j=1}^{m} g_j(X) \tag{5}$$

g_j denotes a regression tree for BART, and m is the number of trees.

Let T denote a regression tree with a set of interior and terminal nodes. Each interior node is associated with a binary decision rule. Suppose the number of terminal nodes is B, and each terminal node of T is associated with a parameter value $\mu_b (b = 1, \ldots, B)$. Each input variable $X = (x_1, , x_p)$ is associated with one of the B terminal nodes of regression tree T, and assigned with the μ_b value of the terminal node.

Thus, g_j in Eq. (5) can be presented as $g_j(X; T_j, M_j)$, and BART predictive model can be transformed into:

$$f(X) = \beta_0 + \sum_{j=1}^{m} g_j(X; T_j, M_j) + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2) \tag{6}$$

Here, $M_j = \{\mu_{j1}, \mu_{j2}, \ldots, \mu_{jB}\}$.

The flexibility and adaptability of the BART model are determined by the number of trees and the complexity of each individual tree jointly. Therefore, a regulation prior is imposed on the parameters of BART model, i.e., β_0, μ_b, the variance σ^2 of Gaussian noise ε, and m. The prior is specified conservatively in order to keep the individual tree effects from being overly influential. As a result, the size of each tree is small, and trees are turned into "weak learners".

To facilitate the use of BART, the prior is specified by several interpretable hyperparameters (i.e.,(υ, q, k, m)). These hyperparameters can either be regulated via cross-validation, or be set to the defaults (3, 0.90, 2, 200). In our study, BART-default is employed.

Furthermore, to fit the sum-of-trees model, BART uses the Bayesian backfitting MCMC to iteratively samples from the posterior distribution $P(\{T_j, \mu_j\}_{j=1}^m, \beta_0, \sigma|y)$. Using MCMC, a sequence of draws of $((T_1, \mu_1), \ldots, (T_m, \mu_m))$ is generated to converge to the posterior distribution. At each iteration, both the tree structures and associated parameters are updated. Iterations are repeated until satisfactory convergence is obtained.

Based on the sum-of-trees model, regulation prior and MCMC algorithm, the final BART model is constructed and fitted suitably. Given an input value $X'(x_1, \ldots, x_p)$, BART predict the response value Y' by an average of draws of all sampled trees.

4.1.2 Partial Dependence Function

Partial dependence function [39] reveals the marginal effect of a subset of variables on the response. Divide the p-dimensional variable $X(x_1, \ldots, x_p)$ into two parts: x_s (variables of interest) and x_c (complement of x_s), the partial function dependence is defined as:

$$f(x_s) = \frac{1}{n} \sum_{i=1}^{n} f(x_s, x_{ic}) \tag{7}$$

x_{ic} is the i^{th} observation value of x_c, and n is the total number of observations.

In BART, the backfitting MCMC algorithm generates a sequence of draws of functions $f_1^*, \ldots, f_K^*$, which is regarded as an approximate of the "true" predictive function $f(X)$. For each draw, $f(x_s)$ in Eq. (7) is computed using Eq. (8):

$$f_k^*(x_s) = \frac{1}{n} \sum_{i=1}^{n} f_k^*(x_s, x_{ic}), \quad k \in \{1, \ldots, K\} \tag{8}$$

Then, the average of $f_1^*(x_s), \ldots, f_K^*(x_s)$ yields an estimate of $f(x_s)$, computed as Eq. (9):

$$f(x_s) = \frac{1}{K} \sum_{k=1}^{K} f_k^*(x_s) \tag{9}$$

4.1.3 Predictor Variable Selection

By observing the variable usage frequency in a sequence of draws of functions $f_1^*, \ldots, f_K^*$, BART can also be used to select the most influential variables for explaining the variation of response variable Y. Considering the i^{th} component of X, the number of times that the variable is selected for splitting (denoted as z_{ik}) is obtained from each function f_k^*. Then weighting z_{ik} by the number of input data points present in the node, we can get the weighted usage frequency of the variable in each sampled function f_k^* (denoted as z_{ik}^*). Finally, the average weighted usage frequency of the variable is computed using Eq. (10):

$$v_i = \frac{1}{K} \sum_{k=1}^{K} z_{ik}^* \tag{10}$$

The variable with larger v_i indicates better prediction for the response variable. Such variable selection approach is model-free because it is not based on the usual assumption of an encompassing parametric model [29].

4.2 BART-Based L2 Cache AVF Prediction Results

In this section, we perform a robust and accurate L2 cache AVF prediction using BART method, and we quantify the influences of various performance metrics on L2 cache AVF using two basic features of BART (i.e., partial dependence functions, and model-free variable selection). Then we make a comparison between BART and other competing methods (i.e., linear regression method and BRT), thus demonstrating the robustness and accuracy of BART in L2 cache AVF prediction.

4.2.1 Robust and Accurate Prediction of L2 Cache AVF

Our data set contains the performance metrics and L2 cache AVF for 7200 intervals of 288 SimPoints of SPEC2K benchmarks. In order to validate the robustness and accuracy of BART in L2 cache AVF prediction, we firstly create 18 independent train/test splits by randomly selecting 5/6 of the data set as a train set and the remaining 1/6 as a test set. Thus, the train set contains the data of 6000 intervals of 240 SimPoints, and the test set contains the data of 1200 intervals of 48 SimPoints. Then we train the BART model on the train set, and apply the model to predict L2 cache AVF for the test set and obtain the predictive RMSE of the 18 train/test splits respectively. RMSE is computed using Eq. (11). We consider relative RMSE (RRMSE), which is RMSE divided by the minimum RMSE, to facilitate the comparisons among different train/test splits.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2} \tag{11}$$

n is the number of intervals in test set, y and $\hat{y}$ are the true and the predictive response variables of each interval respectively.

Figure 5 shows that RRMSE ranges in $(1, 1.49)$, indicating that all the RMSE values are close to each other, further revealing the robustness of BART with different train/test splits.

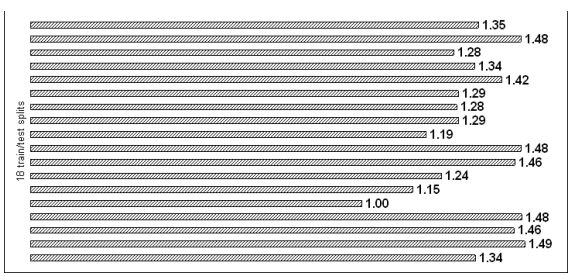


Fig. 5 RRMSE values of 18 train/test splits.

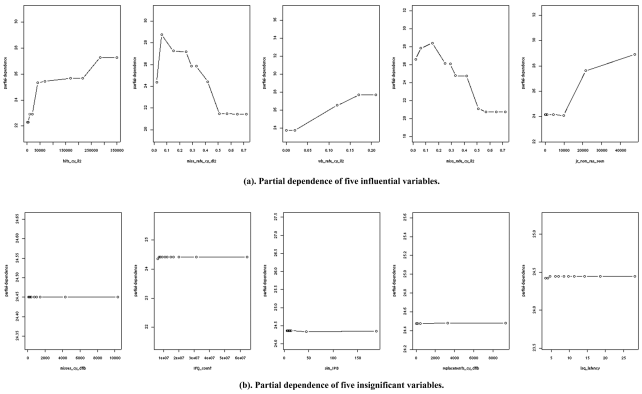


Fig. 6 Partial dependence.

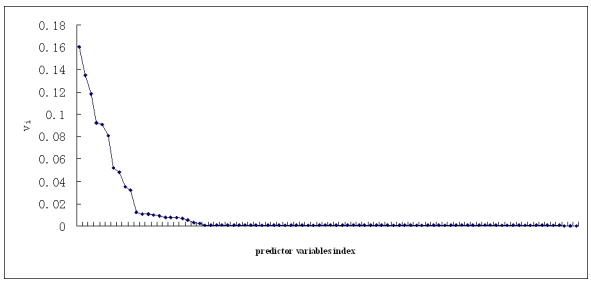


Fig. 7 Usage percentage of variables.

We choose the train/test split of the minimum RMSE for the following illustrations of BART features, so as to understand the dependence of various performance metrics on L2 cache AVF.

Partial dependence function (explained in Sect. 4.1.2) can be used to obtain the marginal effects of a subset of variables on the response variable. We show the partial dependence plots for five influential variables in Fig. 6 (a), and their importance to L2 cache AVF is testified by the nonzero marginal effects of them. By comparison, Fig. 6 (b) shows the insignificant effects of other five variables on L2 cache AVF, which is reflected by the zero marginal effects of them.

Besides, BART can be used to select the most influential variables for explaining the variation of response variable (i.e., model-free variable selection described in Sect. 4.1.3). Figure 7 shows the average weighted usage frequency (i.e., v_i in Eq. (10)) for all the predictor variables. We can identify top 10 variables which show strong influences on L2 cache AVF.

Although BART works well for L2 cache AVF prediction, variables used in BART fitting are overmuch. To facilitate the use of L2 cache AVF predictor, the predictor must be a low-dimensional function. In other words, the number of variables used to fit BART model should be controlled. Therefore, BART model is refitted using the 10 most influential variables based on the best train/test split above. Figure 8 shows the refitted BART inferences.

Figure 8 shows the sequence of σ draws over the iterations. The draws of σ nicely wander around the value $\sigma = 2$,

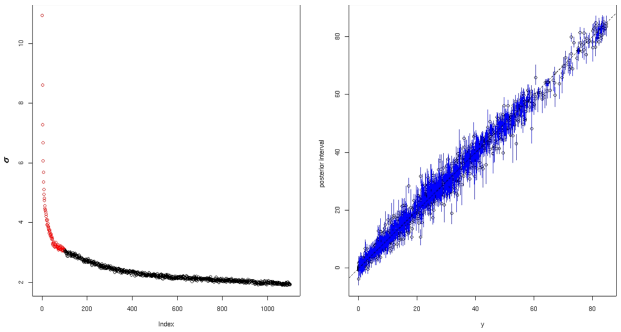


Fig. 8 Refitted BART inferences.

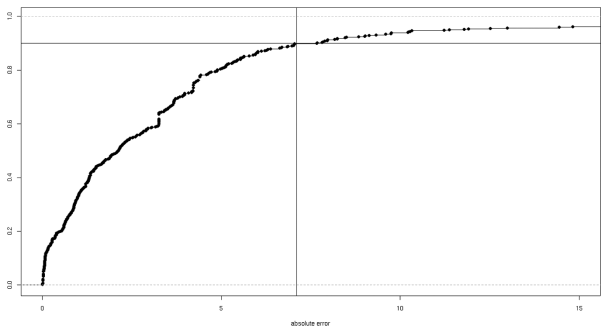


Fig. 9 Cumulative density function for the absolute error values on the test set.

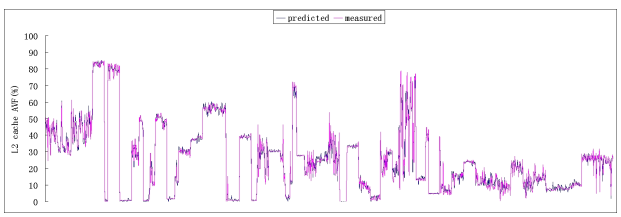


Fig. 10 Measured and predicted L2 cache AVF profiles on the test set.

implying that BART model is fitted well. Figure 8 also plots posterior mean estimate $f(\hat{x})$ against the true response value y. Vertical lines indicate that the 90% posterior intervals for the response value. We can see that most values correlate well with the true response values, and the intervals tend to cover the true response values.

Besides, we show the overall cumulative density function for the absolute error values on the test set using the refitted BART model in Fig. 9. We find that 90% of data points are predicted with absolute errors less than 7.1.

Moreover, in order to visualize the results of BART-based L2 cache AVF prediction better, we compare the measured and predicted L2 cache AVF profiles on the test set, shown in Fig. 10. We can see that the refitted BART predictive model faithfully detects the time-varying behavior of actually measured L2 cache AVF, and predicts L2 cache AVF with high fidelity.

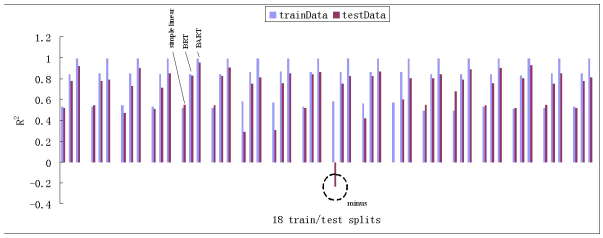


Fig. 11 R^2 of different models on 18 train/test splits.

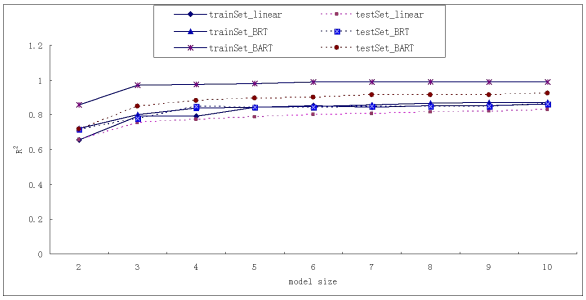


Fig. 12 R^2 of different models over different model sizes.

4.2.2 Comparison of L2 Cache AVF Predictive Methods

We conduct a comparison between BART and other competing methods, including the linear regression method and BRT which have been employed for architectural vulnerability prediction of several microarchitectures [26], [27]. We use Multiple R-Squared (denoted as R^2) which is the square of correlation coefficient to explain the predictive results. R^2 is no more than 1, computed by Eq. (12). Larger R^2 represents a better predictive fit, indicating that the corresponding performance metric correlates well with L2 cache AVF.

$$R^2 = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2 - \sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{12}$$

In our study, y_i is the response value (i.e. L2 cache AVF) of each interval, $\bar{y}$ is the average L2 cache AVF of all the intervals in a program, and $\hat{y}$ is the predicted L2 cache AVF of each interval.

Firstly, we compare the three predictive models across different test/train splits. Figure 11 shows R^2 of different models on 18 different train/test splits. Each train/test split corresponds to three items, each of which represents R^2 on train set and test set respectively. From left to right, the three items indicate R^2 of the simple linear, BRT and BART models. On average, R^2 of the simple linear, BRT and BART models on the train set is 53.5%, 84.9%, and 99%, and R^2 of the three models on the test set is 43.3%, 76.9%, and 86.4% respectively. BART achieves higher R^2 than the simple linear and BRT models on both train set and test set. Note: R^2 of the simple linear model could sometimes be minus, indicating that simple linear model is unstable for L2 cache AVF prediction.

Secondly, using the best train/test split obtained in Sect. 4.2.1, we compare these three predictive models over different model sizes, shown in Fig. 12. Normally, R^2 of three models on the train set and test set increase monotonously when the model size is no more than 4. When the model size continues to increase, R^2 of three models on both the train set and test set gradually achieves a stable value. Compared with the simple linear and BRT models, BART obtains the best R^2 on both train set and test set.

Thus, the above comparisons demonstrate the robustness and superiority of BART model across different test/train splits and different model sizes.

5. Online L2 Cache AVF Estimation

In this section, we create a simplified and fast L2 cache AVF predictor. Then we integrate the AVF predictor into ECC schemes and develop an AVF-aware ECC technique. We analyze the costs of the AVF-aware ECC technique, thus demonstrating the prospect of AVF prediction based dynamic fault tolerant techniques.

5.1 Simplified and Fast L2 Cache AVF Predictor

In order to integrate the AVF prediction into AVF-aware dynamic fault tolerant management schemes, we consider reducing the complexity of L2 cache AVF prediction to get an effective online AVF estimator. Although BART method is proved to perform well for L2 cache AVF prediction, the dimensionality of the predictive function is high, even the refitted BART function which relies on only 10 variables is not suitable for online AVF prediction. So we further employ bump hunting technique [30] to obtain the simplified L2 cache AVF predictor. The goal of bump hunting is to partition the feature space into box-shaped regions and to seek boxes with a high average of the response variable. And bump hunting technique has been successfully used for fast estimation of AVF of IQ and ROB [27].

In our study, we set the high AVF threshold at 40%. That is to say, if L2 cache AVF is no less than 40%, L2 cache is regarded to exhibit high vulnerability during this interval. Note: in practice, setting of high AVF threshold would be customized to special reliability/power/performance demands of processor design. In this paper, we try to explain the feasibility of creating a simplified and fast online cache AVF predictor, so we just set the high AVF threshold at 40%. We apply bump hunting technique to our data set, and then extract five simple selecting rules on several key performance metrics to identify intervals of high L2 cache AVF (as shown in Fig. 13)

We apply the selecting rules on our test set, and the result is illustrated in Fig. 14. As Fig. 14 shows, intervals of high L2 cache AVF could be effectively identified using the above selecting rules, thus proving that the simple selecting rules work well for fast and accurate L2 cache AVF estimation.

If
{ miss_rate_DL2 <0.69
&&miss_rate_IL2 <0.954
&&hits_IL2>197781
&&ruu_occupancy >22.688
&&page_count>1245.25
}
{ L2 cache is regarded to exhibit high vulnerability during this interval. }

Note: *miss_rate_DL2* and *miss_rate_IL2* are the miss rate of level2 instruction and data cache. *hits_IL2* is the total number of hits of level2 instruction cache. *ruu_occupancy* is the average RUU occupancy. *page_count* is total number of pages allocated.

Fig. 13 Selecting rules.

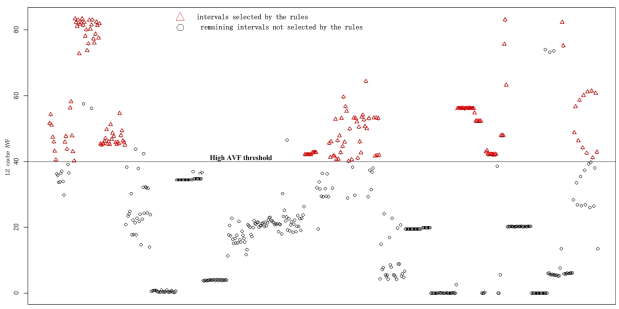


Fig. 14 Fast estimation of L2 cache AVF on the test set.

5.2 An Example of Predictor Usage

In order to explain the usage of the online AVF predictor and to demonstrate the utility of AVF prediction based dynamic fault tolerant mechanisms, we integrate the simplified L2 cache AVF predictor into the ECC technique. Then we develop an AVF-aware ECC technique which provides ECC protection only for the execution points of high L2 cache AVF.

In our experiment, cache is initially simulated with ECC disabled. Every two million cycles, we get the instantaneous L2 cache AVF value using our simplified AVF predictor (shown in Fig. 13). If L2 cache AVF is above the threshold (i.e., 40%), ECC protection is enabled for the following simulation intervals. Once ECC protection is enabled, we need a criterion to disable it. In this study, we disable ECC after ten million cycles.

We use CACTI [40] which is modified for more detail modeling of ECC technique to determine the latency and power of L2 cache with/without ECC protection. Using these latency and power values, we simulate the SPEC2K benchmarks in SS-SERA. We evaluate the access latency and power consumption of L2 cache for different schemes (i.e., traditional full-ECC, AVF-aware ECC, no ECC), and the results are shown in Fig. 15 and Fig. 16. We normalize the access latency and power consumption values to that of unprotected L2 cache.

We can see that for most of the SPEC2K programs, access latency and power consumption of full ECC protected L2 cache are reduced significantly using AVF-aware ECC technique. Besides, we can see that for gcc, mcf, art and

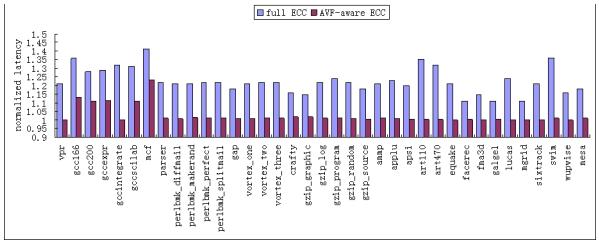


Fig. 15 Normalized access latency.

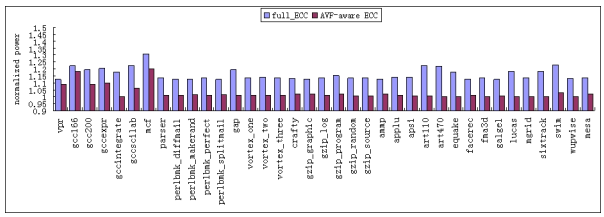


Fig. 16 Normalized power.

swim programs, full ECC protected L2 cache exhibit higher access latency and power consumption, this is because these four programs generate more L2 cache misses than other programs [41]. Using AVF-aware ECC technique, access latency and power consumption of full ECC protected L2 cache can also be reduced observably for these four benchmarks.

On average, for SPEC2K benchmarks, AVF-aware ECC technique reduces the access latency by 16.5% and the power consumption by 11.4% compared with traditional full ECC protection technique. We can infer that AVF-aware ECC technique is a good candidate for cost-effective cache protection.

6. Related Work

Many researches have employed analytical models for cache vulnerability computation. Asadi et al. [2] introduced the concept of critical time (CT) and critical word (CW) for accurate estimation of cache reliability and presented a soft error modeling method that captured different types of soft errors. They used the same estimating method to measure vulnerability of different cache memories [3], [42]. Zhang [4] analyzed the different access patterns of cache lines to find the susceptible time of blocks. Then they used the information of blocks' susceptible time to compute cache vulnerability factor. Wang et al. [43] and Yan and Zhang [44] employed the similar analysis method of Zhang [4] to measure cache vulnerability to soft errors. Mukherjee et al. [9] introduced the concept of Architectural Vulnerability Factor (AVF) for the measurement of soft error rate. And Biswas et al. [33] proposed lifetime analysis based method to compute AVF of cache structures.

Furthermore, studies have shown significant AVF variations of several microarchitectures across different execution phases of applications [25], [26]. Fu et al. [25] investigated the fuzzy correlations between AVFs and several spe-

cial performance metrics. Walcott et al. [26] extended their idea, and explored multivariate statistical relationships between AVF and a wide variety of performance metrics. Then they created a linear model of multiple variables to predict runtime AVF properly. Duan et al. [27] further proposed a versatile AVF predictor which was calibrated for different workloads, execution phases and processor configuration, and developed a fast AVF estimation to identify the intervals with high AVF for several structures (such as issue queue).

7. Conclusion

In this paper, using our improved cache AVF estimation framework, we perform a comprehensive study of dynamic characteristics and predictability of L2 cache AVF. The study results suggest that it is inadequate to use simple predictive models based on several performance metrics for accurate AVF prediction. Driven by the results, we propose to use BART method for accurate L2 cache AVF prediction. Experimental results show that compared with other predictive methods (i.e., linear regression and BRT), BART achieves higher R^2 than other two methods across different train/test splits and different model sizes, thus demonstrating the robustness and accuracy of BART. Then, some simple selecting rules on several key performance metrics are extracted by bump hunting technique, thus further reducing the complexity of L2 cache AVF predictor. We take an AVF-aware ECC technique as an example to illustrate the usage of the predictor and to demonstrate the prospect of AVF-aware soft error protection schemes. Experimental results show that AVF-aware ECC technique could provide performance and energy gains without sacrifice much reliability. We can infer that AVF-aware soft error protection techniques provide a delay and energy efficient fault tolerant solution for cache designs.

With dramatic scaling in feature size of VLSI technology, performance, power consumption and reliability are all becoming very important criteria for microprocessor designs. Processor designers are increasingly required to more carefully trade off performance, power and reliability. With the ability to predict cache AVF fast and accurately at runtime, comes the opportunity of AVF-aware fault tolerant technique which provides a cost-efficient soft error protection for caches.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant No.60970036 and No.60873016, National High Technology Development 863 Program of China under Grant No.2009AA01Z102 and No.2009AA01Z124.

References

[1] S.Z. Shazli, M.A. Aziz, M.B. Tahoori, and D.R. Kaeli, "A field analysis of system-level effects of soft errors occurring in microprocessors used in information system," IEEE International Test Conference, pp.1–10, Oct. 2008.

[2] H. Asadi, V. Sridharan, M.B. Tahoori, and D. Kaeli, "Balancing performance and reliability in the memory hierarch," Proc. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp.269–279, March 2005.

[3] V. Sridharan, H. Asadi, M.B. Tahoori, and D. Kaeli, "Reducing data cache susceptibility to soft errors," IEEE Trans. Dependable and Secure Computing, vol.3, no.4, pp.353–364, Oct.-Dec. 2006.

[4] W. Zhang, "Computing cache vulnerability to transient errors and its implication," Proc. 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Oct. 2005.

[5] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Siavasubramaniam, "ICR: In-cache replication for enhancing data cache reliability," Proc. International Conference on Dependable Systems and Networks (DSN), pp.291–300, June 2003.

[6] S. Kim and A. Somani, "Area efficient architectures for information integrity checking in cache memories," Proc. International Symposium on Computer Architecture (ISCA), pp.246–256, May 1999.

[7] H.T. Nguyen and Y. Yagil, "A systematic approach to SER estimation and solutions," Proc. International Reliability Physical Symposium, pp.60–70, Texas, 2003.

[8] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," IEEE Trans. Device and Materials Reliability, vol.5, no.3, pp.305–316, Sept. 2005.

[9] S.S. Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," Proc. 36th annual IEEE/ACM International Symposium on Microarchitecture, pp.29–40, IEEE Computer Society, Washington, DC, USA, 2003.

[10] S. Hareland, J. Maiz, M. Alavi, K. Mistry, S. Walstra, and C. Dai, "Impact of CMOS scaling and SOI on soft error rates of logic processes," Symposium on VLSI Technology, Digest of Technical Papers, pp.73–74, June 2001.

[11] T. Karnik, B. Bloechel, K. Soumyanath, V. De, and S. Borkar, "Scaling trends of cosmic rays induced soft errors in static latches beyond $0.18\,\mu$," Symposium on VLSI Circuits, Digest of Technical Papers, pp.61–62, June 2001.

[12] K. Reick, P.N. Sanda, S. Swaney, J.W. Kellington, M. Mack, M. Floyd, and D. Henderson, "Fault-tolerant design of the IBM power6 microprocessor," IEEE Micro, vol.28, no.2, pp.30–38, March 2008.

[13] AMD Athlon(TM) 64 Processor, http://www.amd.com

[14] Intel Pentium 4 Processor Technical Documentation, 2004. http://www.intel.com/design/pentium4/documentation.htm

[15] S. Rusu, H. Muljono, and B. Cherkauer, "Itanium 2 processor 6M: Higher frequency and larger L3 cache," IEEE Micro, vol.24, no.2, pp.10–18, March 2004.

[16] Sun Microsystems Inc., OpenSPARC T2 system-On-Chip (SOC) microarchitecture specification, May 2008.

[17] H. Ando, K. Seki, S. Sakashita, M. Aihara, R. Kan, K. Imada, M. Itoh, M. Nagai, Y. Tosaka, K. Takahisa, and K. Hatanaka, "Accelerated testing of a 90 nm SPARC64V microprocessor for neutron SER," Proc. IEEE Workshop on Silicon Errors in Logic-System Effects (SELSE), April 2007.

[18] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way multithreaded SPARC processor," IEEE Micro, vol.25, no.2, pp.21–29, March 2005.

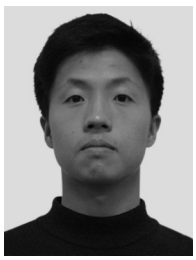
[19] M.A. Bajura, Y. Boulghassoul, R. Naseer, S. DasGupta, A.F. Witulski, J. Sondeen, S.D. Stansberry, J. Draper, L.W. Massengill, and J.N. Damoulakis, "Models and algorithmic limits for an ECC-based approach to harden sub-100-nm SRAM," IEEE Trans. Nucl. Sci., vol.54, no.4, pp.935–945, Aug. 2007.

[20] K.C. Mohr and L.T. Clark, "Delay and area efficient first-level cache soft error detection and correction," Proc. International Conference on Computer Design, pp.88–92, San Jose, CA, Oct. 2006.

[21] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," Proc. 40th

Annual IEEE/ACM International Symposium on Microarchitecture, pp.197–209, Dec. 2007.
[22] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. Irwini, "Soft error and energy consumption interactions: A data cache perspective," Proc. International Symposium on Low Power Electronics and Design (ISLPED), Aug. 2004.
[23] D. Yoon and M. Erez, "Memory mapped ECC: Low-cost error protection for last level caches," Proc. 36th International Symposium on Computer Architecture (ISCA), pp.116–127, June 2009.
[24] D.H. Yoon and M. Erez, "Virtualized and flexible ECC for main memory," Proc. 15th Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp.397–408, March 2010.
[25] X. Fu, J. Poe, T. Li, and J. Fortes, "Characterizing microarchitecture soft error vulnerability phase behavior," Proc. International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp.147–155, Sept. 2006.
[26] K.R. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic prediction of architectural vulnerability from microarchitectural state," Proc. International Symposium on Computer Architecture (ISCA), pp.516–527, June 2007.
[27] L. Duan, B. Li, and L. Peng, "Versatile prediction and fast estimation of architectural vulnerability factor from processor performance metrics," IEEE 15th International Symposium on High Performance Computer Architecture (HPCA), pp.129–140, Feb. 2009.
[28] N. Soundararajan, A. Parashar, and A. Sivasubramaniam, "Mechanisms for bounding vulnerabilities of processor structures," Proc. International Symposium on Computer Architecture (ISCA), pp.506–515, June 2007.
[29] H.A. Chipman, E.I. George, R.E. McCulloch, Bayesian Ensemble Learning. In Neural Information Processing Systems, 19, Scholkopf B, Platt J, Hoffman T, editors, Cambridge, MA: MIT Press; 2007.
[30] J. Friedman and N. Fisher, "Bump hunting in high-dimensional data," Statistics and Computing, vol.9, pp.123–143, 1999.
[31] D. Burger and T. Austin, The SimpleScalar Toolset, Version 3.0. http://www.simplescalar.com
[32] X. Fu, T. Li, and J. Fortes, "Sim-SODA: A framework for microarchitecture reliability analysis," Proc. Workshop on Modeling, Benchmarking and Simulation (Held in conjunction with International Symposium on Computer Architecture), June 2006.
[33] A. Biswas, R. Cheveresan, J. Emer, S.S. Mukherjee, P.B. Racunas, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," Proc. International Symposium on Computer Architecture (ISCA), pp.532–543, June 2005.
[34] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," Proc. 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp.45–57, Oct. 2002.
[35] E. Duesterwald, C. Cascaval, and S. Dwarkadas, "Characterizing and predicting program behavior and its variability," Proc. 12th International Conference on Parallel Architectures and Compilation Techniques (PACT), p.220, IEEE Computer Society, Washington, DC, USA, Sept.-Oct. 2003.
[36] J.H. Friedman, "Multivariate adaptive regression splines," Annals of Statistics, vol.19, no.1, pp.1–67, 1991.
[37] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag New York, New York, NY, 1995.
[38] L. Breiman, "Random forests," Mach. Learn., vol.45, no.1, pp.5–32, Oct. 2001.
[39] Friedman J. Greedy Function Approximation: A Gradient Boosting Machine. Annuals of Statistics, vol.29, pp.1189–1232, 2001.
[40] CACTI 6.0. http://www.cs.utah.edu/~rajeev/cacti6/
[41] S. Sair and M. Charney, Memory behavior of the SPEC2000 benchmark suite, Technical report, IBM, 2000.
[42] H. Asadi, V. Sridharan, M.B. Tahoori, and D. Kaeli, "Vulnerability analysis of L2 cache elements to single event upsets," Proc. Conference on Design, Automation, and Test in Europe, March 2006.
[43] S. Wang, J. Hu, and S.G. Ziavras, "On the characterization of data cache vulnerability in high-performance embedded microprocessors," Proc. 6th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (IC-SAMOS), pp.14–20, July 2006.
[44] J. Yan and W. Zhang, "Evaluating instruction cache vulnerability to transient errors," ACM SIGARCH Computer Architecture News, vol.35, no.4, pp.21–28, Sept. 2007.

Yu Cheng born in 1984. Ph.D. candidate at School of Computer, National University of Defense Technology, China. She is currently working towards her Ph.D. on reliability, ultra low power and high performance computing.



Anguo Ma born in 1983. Ph.D. candidate at School of Computer, National University of Defense Technology, China. He is currently working towards his Ph.D. on reliability, low power design, GPGPU and high performance computing.

Minxuan Zhang born in 1954. Professor at School of Computer, National University of Defense Technology, China. His research interests include computer architectures, low power, reliability and VLSI design.