Ma AG, Cheng Y, Xing ZC. Accurate and simplified prediction of AVF for delay and energy efficient cache design. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 26(3): 504–519 May 2011. DOI 10.1007/s11390-011-1150-7

Accurate and Simplified Prediction of AVF for Delay and Energy Efficient Cache Design

An-Guo Ma (马安国), Yu Cheng (成　玉), and Zuo-Cheng Xing (邢座程), *Senior Member*, *CCF*

School of Computer Science, National University of Defense Technology, Changsha 410073, China

E-mail: {anguo.ma, y.cheng, zc.xing}@nudt.edu.cn

Received January 5, 2011; revised April 2, 2011.

Abstract With continuous technology scaling, on-chip structures are becoming more and more susceptible to soft errors. Architectural vulnerability factor (AVF) has been introduced to quantify the architectural vulnerability of on-chip structures to soft errors. 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). Cache is one of the most susceptible components to soft errors and is commonly protected with error correcting codes (ECC). However, protecting caches closer to the processor (i.e., L1 data cache (L1D)) using ECC could result in high overhead. Protecting caches without accurate knowledge of the vulnerability characteristics may lead to over-protection. Therefore, designing AVF-aware ECC is attractive for designers to balance among performance, power and reliability for cache, especially at early design stage. In this paper, we improve the methodology of cache AVF computation and develop a new AVF estimation framework, soft error reliability analysis based on SimpleScalar. Then we characterize dynamic vulnerability behavior of L1D and detect the correlations between L1D AVF and various performance metrics. We propose to employ Bayesian additive regression trees to accurately model the variation of L1D AVF and to quantitatively explain the important effects of several key performance metrics on L1D AVF. Then, we employ bump hunting technique to reduce the complexity of L1D AVF prediction and extract some simple selecting rules based on several key performance metrics, thus enabling a simplified and fast estimation of L1D AVF. Based on the simplified and fast estimation of L1D AVF, intervals of high L1D AVF can be identified online, enabling us to develop the AVF-aware ECC technique to reduce the overhead of ECC. Experimental results show that compared with traditional ECC technique which provides complete ECC protection throughout the entire lifetime of a program, AVF-aware ECC technique reduces the L1D access latency by 35% and saves power consumption by 14% for SPEC2K benchmarks averagely.

Keywords AVF (architectural vulnerability factor) prediction, BART (Bayesian additive regression), AVF-aware ECC (error correction codes)

1 Introduction

Soft errors are becoming an important concern for microprocessor designs in ultra-deep submicron technology. As the largest structures in a processor, memory elements are most susceptible to soft errors[1-2]. It is reported that 50% soft errors happen in memory elements[3]. Caches, typically implemented as static random access memory (SRAM) arrays, face more serious challenges from soft errors in sub-100 nm technologies[4-5]. The per-bit soft error rate (SER) in modern SRAM is reported to be higher than 0.0001 FIT/bit[6-7]. One failure in time (FIT) is defined as 1 failure in 10^9 hours of device operation.

Recently, researches have found that not all raw soft errors affect the final program output, many raw errors can be masked at the architecture level[8-9], potentially motivating high level tradeoffs among reliability, performance, power consumption, area and other metrics. Mukherjee *et al.*[8] introduced the concept of 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. Many researches have characterized the time varying AVF characteristics of several important structures[10-11]. Furthermore, AVF prediction/estimation mechanisms have been introduced and incorporated into the dynamic fault tolerant system design to make tradeoffs between reliability and performance[11-13]. To our knowledge, they focused on a few given micro-architecture structures, e.g., issue queue (IQ), reorder buffer (ROB) and register file, but

Regular Paper

Supported by the National Natural Science Foundation of China under Grant Nos. 60970036 and 60873016, the National High Technology Development 863 Program of China under Grant Nos. 2009AA01Z102 and 2009AA01Z124.

not including caches.

Parity and error correction codes (ECC) based protection schemes have been widely adopted in modern microprocessors and systems[14-21]. Although ECC protection techniques enable the hardware to maintain a high level of reliability, they bring about significant area, performance and power costs. Moreover, protecting higher levels of cache (e.g., L1 data cache (L1D)) using ECC would induce higher overheads. Previous studies showed that single-error correction and double-error detection (SEC-DED) in L1 cache would increase its access latency by up to 95%, power consumption by up to 22%, and area cost by up to 18%[22-24]. A variety of techniques have been proposed to reduce the area, performance and power costs of ECC[25-30].

However, all these techniques assumed that the probability a soft error would result in an erroneous program outcome (i.e., AVF) is 100%, and therefore provided ECC protection throughout the entire execution lifetime of programs. Considering that protecting cache without accurate knowledge of the vulnerability characteristics may lead to over-protection, AVF-aware ECC techniques which apply ECC protection only to the execution points of high vulnerability would reduce the power consumption and memory access latency brought by ECC to some extent. Since L1D is on the critical path for data accesses, it is important to provide soft error protection for L1D with minimum impacts on performance and power consumption.

Following the idea, we characterize the dynamic L1D AVF behavior and investigate the correlation between the L1D AVF and several key performance metrics. Then by using Bayesian additive regression trees (BART)[31], we accurately predict L1D AVF across different execution phases of SPEC2K benchmarks. Furthermore, to facilitate the use of L1D AVF prediction, we employ bump hunting technique[32] to reduce the complexity of L1D AVF prediction. Therefore, runtime L1D vulnerability can be predicted based on several key performance metrics and execution points of high L1D AVF can be identified. This facilitates the implementation of the AVF-aware dynamic ECC technique based on simplified L1D AVF prediction.

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

1) We improve the methodology of cache AVF computation and develop a new AVF estimation framework, SS-SERA (Soft Error Reliability Analysis based on SimpleScalar). To evaluate our improved method, we estimate AVF of write-back L1D and compare against Sim-SODA which is a publicly available AVF calculation framework. Experimental results show that our method generates more accurate L1D AVF results. Based on the simulation results, we further characterize the dynamic vulnerability of L1D across different execution phases of SPEC2K benchmarks, and detect the correlations between L1D AVF and several key performance metrics.

2) We propose to use BART for accurate L1D AVF prediction across different execution phases of SEPC2K programs with different train/test splits. Then, a comprehensive performance comparison between BART and other competitive methods (i.e., the linear regression model and boosted regression trees (BRT)) is conducted, revealing the superiority and robustness of BART for L1D AVF prediction across different test/train splits and different model sizes.

3) We further employ bump hunting technique to achieve a simplified and fast estimation of L1D AVF, enabling an online L1D vulnerability estimation to identify the execution intervals of high L1D AVF. Then, an AVF-aware ECC technique which enables ECC protection only for the intervals of high L1D AVF is implemented and evaluated, to motivate new studies in AVF-aware design. Experimental results show that compared with traditional ECC technique which provides complete ECC protection throughout the entire lifetime of a program, the dynamic AVF-aware ECC technique reduces the L1D access latency by 35% and saves power consumption by 14% for SPEC2K benchmarks averagely.

The remainder of this paper is organized as follows. Section 2 discusses the related work. Section 3 describes the SS-SERA AVF estimation framework. Section 4 introduces the BART method and its basic features. Section 5 describes experimental setup and experimental evaluation results. Section 6 proposes the simplified and fast L1D AVF estimation. Finally, we conclude in Section 7.

2 Related Work

2.1 Estimation of AVF

There have been several representative architecture level tools developed for early quantitative evaluation of soft errors. Mukherjee *et al.*[8] introduced the concept of Architectural Vulnerability Factor (AVF) for the measurement of soft error rate, and employed Architecturally Correct Execution (ACE) analysis on the high level performance model to provide an early estimation of AVF. Biswas *et al.*[33] further computed AVFs of address-based structures on the same performance model. Fu *et al.*[34] implemented ACE analysis method on a cycle-accurate performance simulator and developed Sim-SODA, which is a unified simulation

framework used to calculate AVFs of various structures. Li *et al.*[35] developed SoftArch which is a probabilistic model of error generation and propagation process, and integrated the model into a trace-driven performance simulator to determine the soft error rates of some hardware structures.

Furthermore, studies have shown significant AVF variation of several microarchitectures across different execution phases of applications[10-11]. Fu *et al.*[10] investigated the fuzzy correlations between AVFs and several special performance metrics. Walcott *et al.*[11] 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.*[12] 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 (i.e., issue queue, reorder buffer).

2.2 Low-Cost ECC

ECC-based techniques are commonly employed in modern microprocessors to improve the reliability of cache, but incurring severe performance and power costs. Many studies have focused on low-cost ECC schemes[25-30].

Mohr and Clark[25] employed two-dimensional parity schemes to detect and correct single bit errors within a word. They applied product codes to memory arrays and used horizontal byte-parity codes to enable low-latency error detection with a minimal increase in area. Kim[26] proposed two-dimensional error coding which separated the high overhead multi-bit error correction from low overhead error detection, thus correcting multi-bit errors in cache with significant small latency. Li *et al.*[27] employed decoupled ECC which protected clean cache blocks with parity and dirty blocks with ECC, and reduced the power consumption of on-chip ECC by power-gating the SEC-DED ECC portion of clean cache lines, with only parity active. They proposed an early-write-back scheme to enhance the ability of using a less powerful error protection scheme for a longer time without sacrificing much reliability. Sadler *et al.*[28] proposed a new Punctured ECC Recovery Cache (PERC) scheme, where cache reads only reduced error detection to save energy while writes updated both detection and correction information.

Very recently, the above decoupled ECC was generalized for Memory-Mapped ECC (MME)[29]. More expensive error correction bits were stored within the memory hierarchy as data and only detection codes were maintained on chip for caches. Furthermore, Yoon and Erez[30] extended their work on MME to the virtualized ECC-based memory error tolerance mechanism, which mapped redundant information needed to correct errors into the memory namespace itself. They used this mechanism to develop two-tiered error protection techniques that separated error detection from the rare error correction, thus saving energy.

3 SS-SERA AVF Estimation Framework

3.1 AVF for Different Types of Soft Error

There are two types of soft errors: silent data corruption (SDC) and detected unrecoverable errors (DUE). SDC only happens in structures with no soft error protection, and induces the system to generate an erroneous outcome. Once a structure is protected by error detection schemes, SDC turns into DUE. DUE is further subdivided into false DUE and true DUE according to whether the detected error would indeed affect the final output of a program. False DUE represents benign detected errors which would not affect the correctness of outcome, and the remaining DUE which would induce the system to generate incorrect outputs is true DUE.

Architectural Vulnerability Factor (AVF)[8] is used to measure the soft error rate of on-chip structures, including caches[33]. A structure's soft error rate is the product of its raw error rate and its AVF. According to different types of soft errors, AVF is also subdivided into SDC AVF and DUE AVF. Soft error rate of a structure with no error protection is computed by (1). For a structure only protected with an error detection mechanism (without error correction), soft error rate of the structure is computed by (2). Soft error detection technique (such as parity) adds extra check bits in cache, thus potentially increasing true DUE. The true DUE AVF of a structure using soft error detection is no less than SDC AVF of the unprotected one.

$$soft\_error\_rate = raw\_error\_rate \times AVF_{SDC} \quad (1)$$

$$\begin{aligned} soft\_error\_rate &= raw\_error\_rate \times AVF_{DUE} \\ &= raw\_error\_rate \times AVF_{true\_DUE} \\ &\geqslant raw\_error\_rate \times AVF_{SDC}. \end{aligned} \quad (2)$$

Note that DUE AVFs of write-through and write-back data caches are different. For write-through data cache only protected by error detection techniques, its DUE AVF is zero, because data can be re-fetched from lower-level cache when detecting an error. For write-back data cache only protected by error detection techniques, DUE would induce the system to generate erroneous outputs, its DUE AVF could not be zero.

Protecting cache with error recovery schemes could possibly reduce SDC and DUE to zero.

3.2 Improved Methodology of Computing Cache AVF

There have been several representative architecture-level tools developed for SDC AVF estimation[33-35]. To our knowledge, Sim-SODA is the only publicly available architecture level simulator for AVF calculation, whereas it has its own limitations. Firstly, it is not proper to employ Sim-SODA to estimate cache AVF for floating point workloads, because the baseline simulator (i.e., sim-Alpha[36]) does not model floating point pipeline accurately. Secondly, it lacks accuracy for cache AVF calculation based on Sim-SODA.

To address the issues, we integrate an improved cache AVF computing model in a more popular simulator (i.e., SimpleScalar[37]), and developed an improved AVF estimation framework, named SS-SERA.

3.2.1 AVF Computing Equations in SS-SERA

Activities occurring during the lifetime of a bit in cache include "idle", "fill", "read", "write", and "evict". Based on the lifetime analysis technique, a bit's lifetime is divided into ACE, un-ACE and unknown components according to the activities during the periods. Fig.1 shows an example of lifetime classification of the data array of a write-back cache.

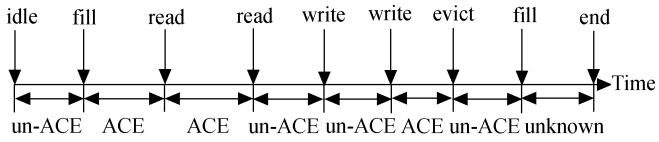


Fig.1. Example of lifetime classification of a write-back data cache. "End" represents the end of benchmark simulation.

The granularity of the lifetime analysis has a big impact on L1D data array's AVF. Empirically, we maintain the lifetime information at byte granularity. Therefore, AVF of L1D data array (*DA_AVF*) is the fraction of all bytes' lifetime during which the bytes are in the ACE state, computed by (3).

$$DA\_AVF = \frac{\sum_{i=1}^{B} ACE_i}{B \times total\_exec\_cycles} \tag{3}$$

B is the total number of bytes of data array. ACE_i represents the total residency cycles of byte i in ACE state.

The lifetime analysis for tag array is performed at bit granularity. Therefore, AVF of L1D tag array (*TA_AVF*) is the fraction of all bits' lifetime during which the bits are in the ACE state, computed by (4).

$$TA\_AVF = \frac{\sum_{i=1}^{M} \sum_{j=1}^{N} ACE_{ij}}{M \times N \times total\_exec\_cycles} \tag{4}$$

M is the total number of entries of tag array, and N is the bit length of each entry. ACE_{ij} represents the total residency cycles of bit j of entry i in ACE state.

3.2.2 Improved Lifetime Classification in SS-SERA

Based on traditional lifetime classification, ACE fraction of a byte/bit lifetime can be computed. For write-back data array, some un-ACE periods can conditionally be ACE, such as fill-to-evict and read-to-evict. Fig.2(a) shows the situation when fill-to-evict is ACE. Considering two consecutive 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 potentially ACE due to the inevitable written back of the entire cache line to the next cache level.

We eliminate the potential ACE situations by adding "dirty" bits associated with different portions of a cache line and adding an extra activity called "dirty_evict". We can identify the modified bytes according to their dirty bits values. When a byte is modified, its "dirty"

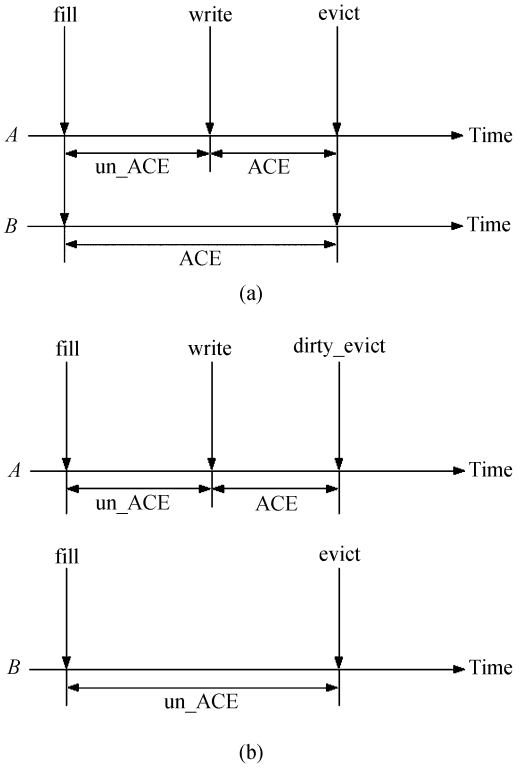


Fig.2. (a) Potential ACE condition. (b) Eliminating potential ACE.

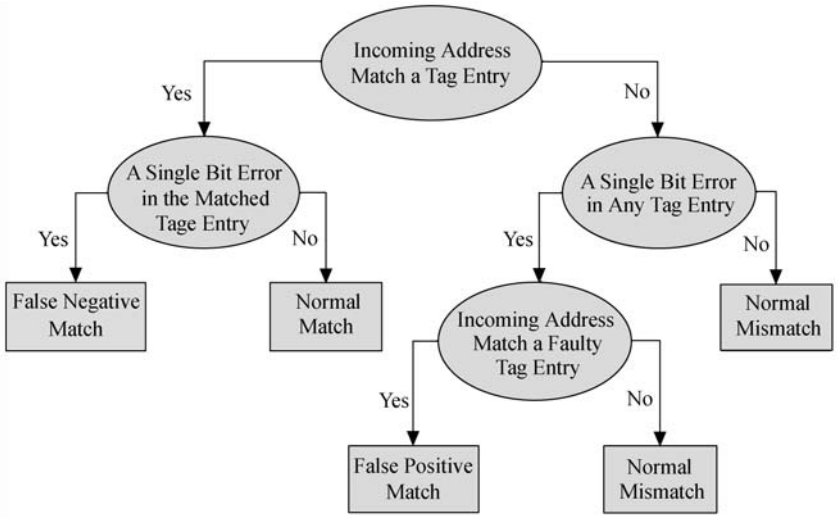


Fig.3. False negative and false positive situations.

bit is set to 1. We distinguish "dirty_evict" from "evict". "dirty_evict" represents the eviction of a modified byte, and "evict" denotes the eviction of an unmodified byte. If the "dirty" bit of a byte is set to 0, eviction of the byte would not result in a written back event. Then above fill-to-evict period of B is no longer ACE, as shown in Fig.2(b).

Although adding each byte of a cache line with a "dirty" bit may induce additional area overhead which is similar to that of providing a parity bit for each byte, hardware overhead of this byte-level dirty bit scheme is acceptable with relaxed die area target. If the die area is relatively constrained, we could add "dirty" bits at a larger granularity (e.g., a dirty bit per word). Compared with byte-level dirty bit scheme, the area overhead of the word-level dirty bit scheme could be reduced by 75%.

For tag array, lifetime analysis has been extended to identify false negative and false positive cases. Fig.3 illustrates the situation of false negative and false positive.

False negative match happens when an error on the primary matched tag bits, thus inducing the primary match to a mismatch. False negative match does harm to modified data entries of a write-back cache, because the modified data would be written back to incorrect location of the next cache level due to the faulty tag. We also employ the extra activity "dirty_evict" to tackle false negative match. For a tag entry, write-to-dirty_evict period is always identified as ACE, and there is no write-to-evict period. Note that write corresponds to the first write to any byte in the data array.

False positive match represents the match that should have mismatched, causing incorrect data transfer from wrong data entry. False positives only arise on the tag entries which are one bit different from the incoming tag bits. We employ Hamming-Distance-One (HDO) analysis technique[33] to identify the entry's particular bit whose fault will introduce the false positive match. If the bit exists, the bit is viewed as ACE, and the remaining bits in the same tag entry are un-ACE.

Accordingly, we list our improved classification of lifetime in Table 1. Note that since all tag bits associated with a modified data entry are ACE from the time that any byte in the entry was first modified until its eviction, there is no write that happened before each period in Table 1. Experimental results in Subsection 5.2 shows that SDC AVF of write-back L1D is reduced by 26.58% based on our improved classification of lifetime.

Table 1. Improved Lifetime Classification for Write-Back L1D

	ACE	un-ACE	unknown
Data Array (per-byte granularity)	fill-to-read, read-to-read, write-to-read, write-to-end, read-to-dirty_evict, write-to-dirty_evict	idle, fill-to-write, fill-to-evict, read-to-evict, read-to-write, write-to-write, evict-to-fill, evict-to-end, dirty_evict-to-end	fill-to-end, read-to-end
Tag Array (per-bit granularity)	fill-to-HDO, read-to-HDO, write-to-HDO, write-to-read, write-to-write, write-to-dirty_evict, write-to-end,	idle, fill-to-read, fill-to-write, read-to-write, fill-to-evict, read-to-evict, HDO-to-read, HDO-to-write, HDO-to-evict, HDO-to-end, evict-to-end, dirty_evict-to-end	fill-to-end, read-to-end

4 Prediction Methodology

Several predictive modeling approaches have been developed in the past few years, including linear regression, neural networks, multivariate adaptive regression splines (MARS)[38], support vector machines (SVM)[39], random forests[40], boosting regression trees (BRT)[41], and Bayesian additive regression trees (BART)[31], each of which uses different techniques to fit the predictive model.

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 tree model, the sum-of-trees model is more flexible and adaptive. Random forest, boosting, and BART are typical sum-of-trees-based methods, which use different patterns of tree models fitting.

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 modeling 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. BART uses Bayesian backfitting MCMC algorithm to iteratively sample from the posterior distribution, in order to achieve a convergent fitting quickly. The induced samples can provide a variety of inferential quantities of interest, such as partial dependence functions and variable selection.

4.1 Fitting of BART Model

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

$$Y = f(\boldsymbol{X}) + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2). \tag{5}$$

For BART, $f(\boldsymbol{X})$ is approximated by a summation of regression trees, computed by (6):

$$f(\boldsymbol{X}) = \beta_0 + \sum_{j=1}^{m} g_j(\boldsymbol{X}) \tag{6}$$

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 μ_b $(b = 1, \ldots, B)$. Each input variable $\boldsymbol{X} = (x_1, \ldots, x_p)$ 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 (6) can be presented as $g_j(\boldsymbol{X}; T_j, M_j)$, and BART predictive model can be transformed into (7):

$$\begin{aligned} &Y = \beta_0 + \sum_{j=1}^{m} g_j(\boldsymbol{X}; T_j, M_j) + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2), \\ &M_j = \{\mu_{j1}, \mu_{j2}, \ldots, \mu_{jB}\}. \end{aligned} \tag{7}$$

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 this paper, 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 $\boldsymbol{X}'(x_1, \ldots, x_p)$, BART predicts the response value Y' by an average of draws of all sampled trees.

4.2 Partial Dependence Function

Partial dependence function[41] reveals the marginal effect of a subset of variables on the response. Divide the p-dimensional variables $\boldsymbol{X}(x_1, \ldots, x_p)$ into two parts: $\boldsymbol{x}_s$ (variables of interest) and $\boldsymbol{x}_c$ (complement of $\boldsymbol{x}_s$), the partial dependence function is defined as:

$$f(\boldsymbol{x}_s) = \frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{x}_s, x_{ic}) \tag{8}$$

x_{ic} is the i-th observation value of $\boldsymbol{x}_c$, 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 $f(\boldsymbol{X})$. For each draw, $f(\boldsymbol{x}_s)$ in (8) is computed using (9):

$$f_k^*(\boldsymbol{x}_s) = \frac{1}{n}\sum_{i=1}^{n} f_k^*(\boldsymbol{x}_s, x_{ic}), \quad k \in \{1, \ldots, K\}. \tag{9}$$

Then, the average of $f_1^*(\boldsymbol{x}_s), \ldots, f_K^*(\boldsymbol{x}_s)$ yields an estimate of $f(\boldsymbol{x}_s)$:

$$f(\boldsymbol{x}_s) = \frac{1}{K}\sum_{k=1}^{K} f_k^*(\boldsymbol{x}_s). \tag{10}$$

4.3 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 $\boldsymbol{X}$, 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 (11):

$$v_i = \frac{1}{K}\sum_{k=1}^{K} z_{ik}^*. \tag{11}$$

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[31].

In summary, BART model consists of an ensemble of (hundreds to thousands of) regression trees, and the hardware implementation of BART-based predictive model is highly complex. In order to interpret the BART-based predictive model better, partial function and variable selection are employed in our experiment for better understanding the model.

5 Experimental Evaluation

5.1 Experimental Setup

All of the experiments are conducted with SS-SERA. SPEC2K INT and FP benchmarks compiled for the Alpha ISA are evaluated, and each of them is run for all of the 100-Million Instruction SimPoints[42]. Each SimPoint is partitioned into 25 intervals of 4 million instructions. The baseline configuration of SS-SERA is listed in Table 2. To simulate a SimPoint, we should fast forward $Num_{forward}$ instructions, calculated using (12):

Table 2. Baseline Configuration

Parameter	Value
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, 32 B line-size
L1 I-cache	64 KB, 4-way, 32 B line-size
Unified L2 cache	512 KB, 4-way, 64 B 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

$$Num_{forward} = (SN_{SimPoint} - 1) \times Size_{SimPoint} \tag{12}$$

$SN_{SimPoint}$ 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 SimPoint1942, simulation of the SimPoint starts just after fast forwarding 1941×100 M instructions from the start of program.

Because computing L1D AVF values of small intervals is quite involved, we have proposed to employ checkpoint mechanism and cooldown technique in SS-SERA to guarantee the correctness of AVF computation and to improve the accuracy of AVF estimation respectively. We have reduced the unknown components of L1D AVF computation to be less than 2% using appropriate cooldown period sizes. Besides, for SPEC2K benchmarks, the unknown components are shown to be mostly un-ACE, with little increase for SDC AVF[33]. We believe L1D AVF computation with unknown component less than 2% is accurate enough to represent the vulnerability of L1D. The detail of our schemes for estimating L1D AVF of small intervals is discussed in another paper.

We employ BART to predict cache AVF accurately across different execution phases of programs. Furthermore, the basic features of BART, including model-free variable selection and estimation of partial dependence functions, are used to achieve a better interpretation and visualization of the BART method. Meanwhile, we also conduct a comparison between BART and other competing methods which have been employed for AVF prediction (i.e., linear regression method and BRT).

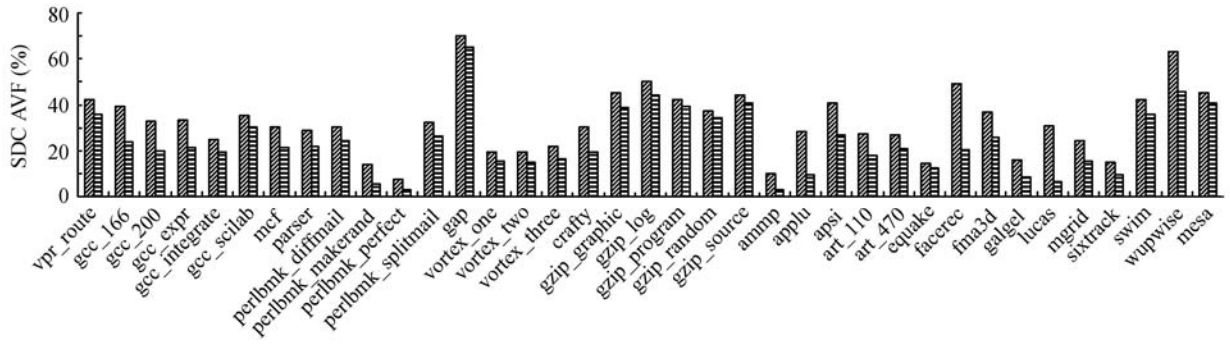


Fig.4. SDC AVF comparison of write-back L1D data array.

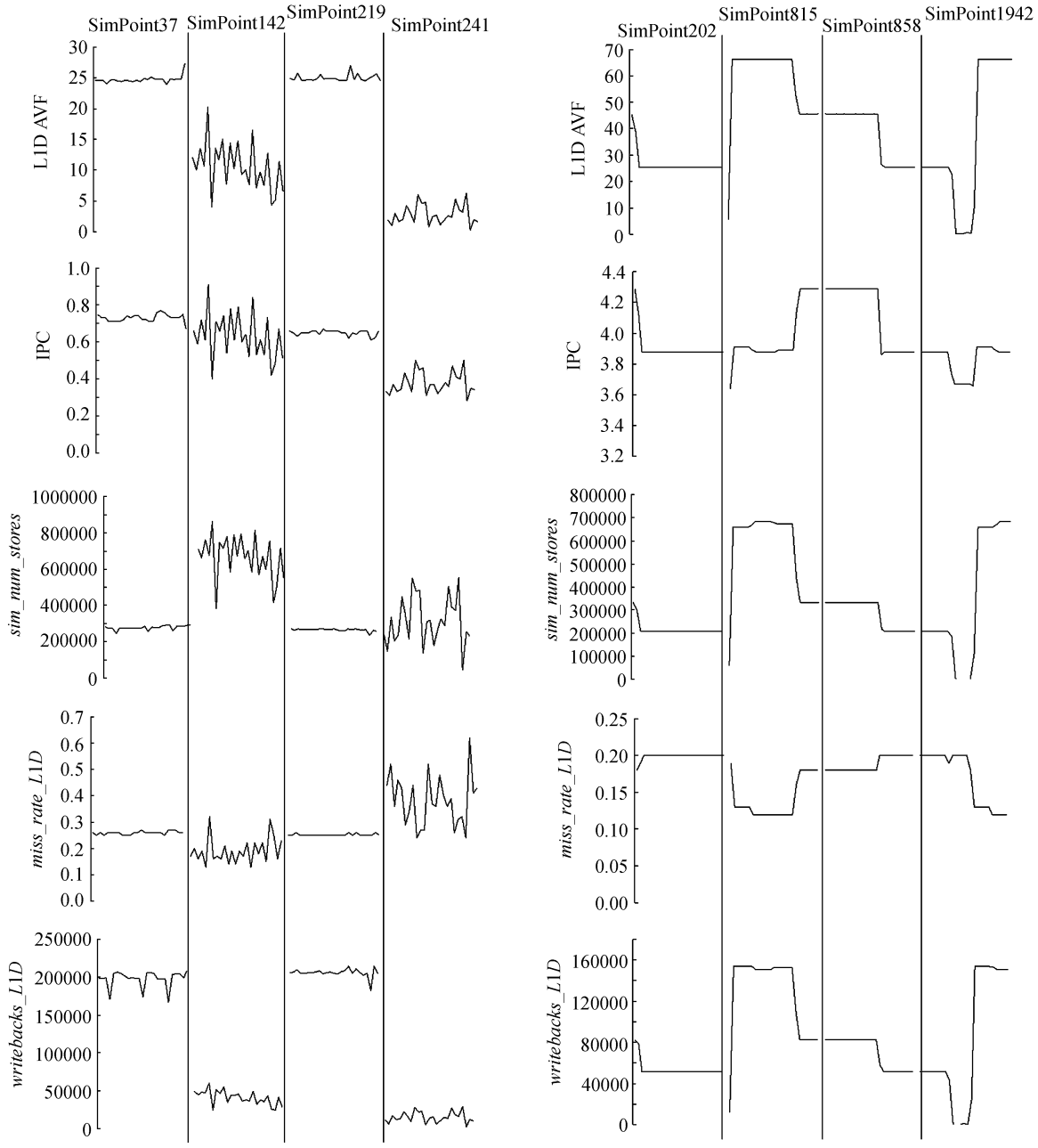


Fig.5. Profile of time-varying L1D AVF and several performance metrics.

5.2 Validating the Improved Methodology of Computing L1D AVF

As described in Section 3, we implement the improved cache AVF computing model in SS-SERA, to eliminate the potential ACE situations. In order to validate the improved methodology of computing cache AVF, we compare the L1D AVF results of SS-SERA with that of Sim-SODA. Fig.4 shows the comparison results.

Form left to right, the two bars associated with each benchmark represent SDC AVF of write-back cache computed using Sim-SODA and SS-SERA respectively. On average, using our improved lifetime classification, SDC AVF of write-back L1D is reduced by 26.58%.

5.3 Characterizing Time-Varying Behavior of L1D

Fig.5 shows the profile of runtime L1D AVF and several key performance metrics over several SimPoints of two benchmarks (i.e., mcf and swim). Other SimPoints and benchmarks exhibit similar time-varying behavior and their results are omitted for brevity. Each 100 M-sized SimPoints contains 25 points of plots which represent the L1D AVF and performance information of 4 M-sized intervals.

5.3.1 L1D AVF Variation

Experimental results show that L1D AVF exhibits different varying characteristics across different SimPoints and different programs. Take mcf for example, L1D AVF of SimPoint142 varies in a larger range (4.01%∼20.33%) than SimPoint37 (23.91%∼27.45%). Variation of L1D AVF is large for SimPoint1942 of swim (0.2%∼66.45%), but is much small for SimPoint219 of mcf (24.54%∼27%).

Table 3 lists the variance of L1D AVF for SEPC2K programs. Variance of L1D 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, as (13) shows. Lower WCoV indicates that intervals of a program exhibit more similar AVF behavior.

$$WCoV_{program\text{“}x\text{”}} = \sum_{s_i \in program\text{“}x\text{”}} CoV_{s_i} \times weight_{s_i},$$

$$CoV_{s_i} = sd_{s_i}/mean_{s_i} \tag{13}$$

s_i represents one of the representative SimPoints of program "x". sd_{s_i} is the standard deviation of L1D AVF for s_i, and $mean_{s_i}$ is the average L1D AVF of s_i.

Table 3. Variance of L1D AVF

Benchmarks	WCoV	Benchmarks	WCoV
gzip_graphic	0.08	vortex_one	0.17
gzip_log	0.10	vortex_two	0.22
gzip_program	0.14	vortex_three	0.12
gzip_random	0.11	crafty	0.06
gzip_source	0.14	ammp	0.08
vpr_route	0.14	applu	0.80
gcc_166	0.10	apsi	0.85
gcc_200	0.15	art_110	0.09
gcc_expr	0.17	art_470	0.20
gcc_integrate	0.26	equake	0.56
gcc_scilab	0.30	facerec	0.29
mcf	0.45	fma3d	0.70
parser	0.39	galgel	0.27
perlbmk_diffmail	0.09	lucas	0.01
perlbmk_makerand	0.22	mgrid	0.13
perlbmk_perfect	0.02	sixtrack	0.10
perlbmk_splitmail	0.37	swim	0.32
gap	0.21	wupwise	0.60
		mesa	0.11

5.3.2 Correlations Between L1D AVF and Performance Metrics

Although L1D AVF exhibits significant variation during program execution, there still exist fuzzy correlations between cache AVF and several performance metrics.

As the most frequently used metric for time-varying performance and vulnerability predictions[10-11,43], *IPC* exhibits a fuzzy relationship with L1D AVF. We can see that residence time of the ACE bytes and total execution cycles of a program in (3) are both affected by *IPC*, indicating that *IPC* definitely has significant influence on L1D AVF. Whether *IPC* shows positive or negative effects on L1D AVF, it further depends on the memory access characteristics of programs.

As shown in Fig.5, *IPC* has either positive or negative effects on L1D AVF for different SimPoints. The curves of *IPC* bear a strong resemblance to the profile of L1D AVF on SimPoint142/241 of mcf, showing the positive effects of *IPC* on L1D AVF. However, *IPC* curves change in an opposite direction to the L1D AVF profile on SimPoint37/219 of mcf, indicating the negative effects of IPC on L1D AVF.

The profiles of *sim_num_stores* and *writebacks_L1D* accord with the profile of L1D AVF for swim, but disagree with the profile of L1D AVF for mcf. The curves of *miss_rate_L1D* bear a weak similarity to the profile of L1D AVF of mcf and swim.

In summary, it is inadequate to use only several

special performance metrics to track L1D AVF, thus motivating us to characterize L1D AVF with a larger space of performance metrics.

5.4 Robust and Accurate Prediction of L1D AVF

The trends existing in Fig.5 suggest that it is inadequate to use only several special performance metrics to track L1D AVF. Simple predictive models, such as the simple linear regression model and multivariate regression model, are not powerful enough for accurate L1D AVF prediction for SPEC2K benchmarks. Hence, we employ BART which provides substantially higher predictive performance than previous models to model L1D AVF across different SimPoints and different programs.

Our dataset contains the performance metrics and L1D AVF for 7200 intervals of 288 SimPoints of SPEC2K benchmarks. Firstly, we create 18 independent train/test splits by randomly selecting 5/6 of the dataset as a train set and the remaining 1/6 as a test set. Thus, the train set contains the data of 6000 intervals, and the test set contains the data of 1200 intervals. We train the BART model on the train set, then apply the model to predict L1D AVF for the test set and compute the predictive *RMSE* of the 18 train/test splits respectively. *RMSE* is computed as (14). We consider relative *RMSE* (*RRMSE*), which is *RMSE* divided by the minimum *RMSE*, to facilitate the comparisons among different train/test splits.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2} \tag{14}$$

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.

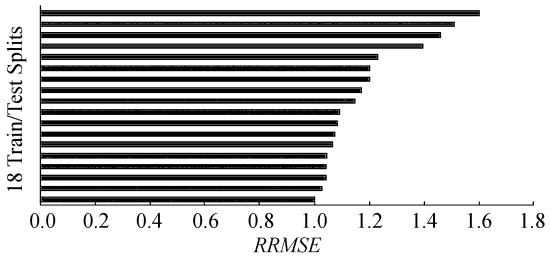


Fig.6. *RRMSE* values of the 18 train/test splits.

Fig.6 shows that *RRMSE* ranges in (1, 1.6), indicating that all the *RMSE* values are close to each other, further revealing the robustness of BART with different train/test splits.

We choose the train/test split of the minimum RMSE for the following illustrations of features of BART. Partial dependence function (explained in Subsection 4.2) 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.7(a), and their importance to representing the

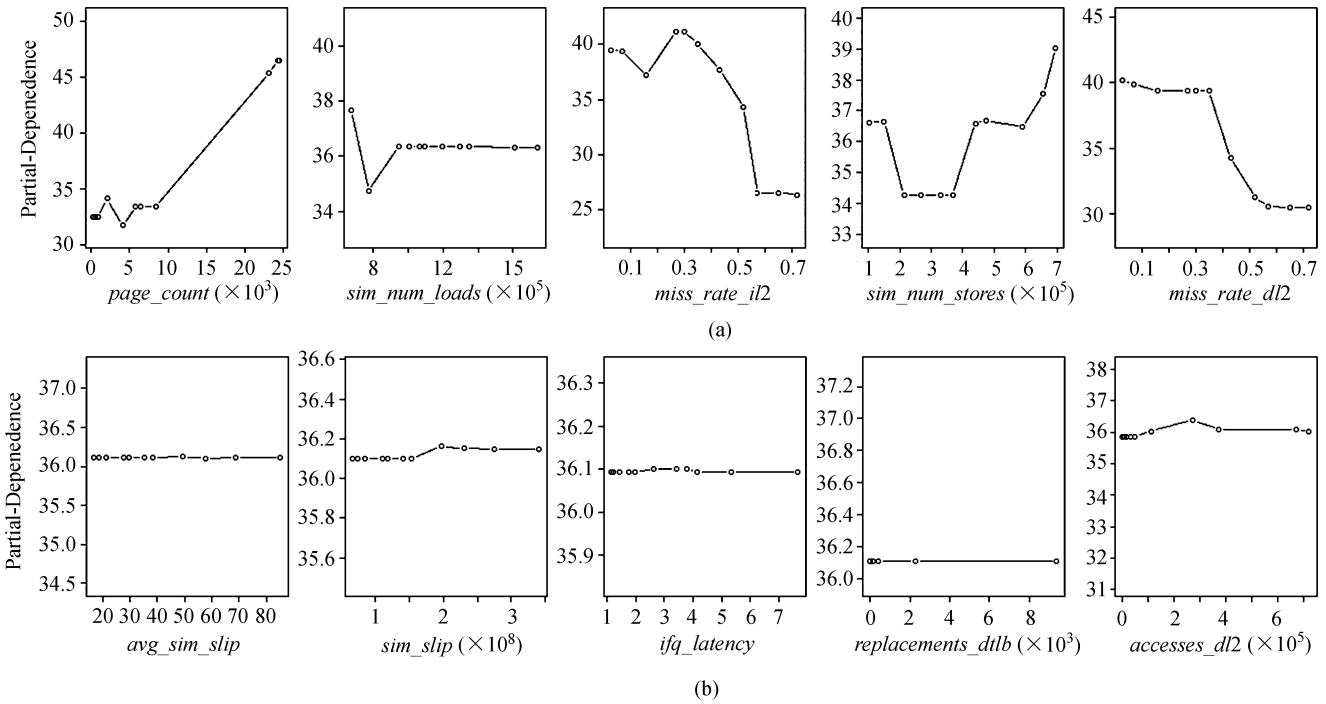


Fig.7. (a) Partial dependence of five influential variables. (b) Partial dependence of five insignificant variables.

variation of L1D AVF is testified by the nonzero marginal effects of them. By comparison, Fig.7(b) shows the insignificant effects of other five variables on L1D AVF, 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 Subsection 4.3). Fig.8 shows the values (computed by (11)) for all the predictor variables. We can identify 10 variables which show relatively strong influences on L1D AVF.

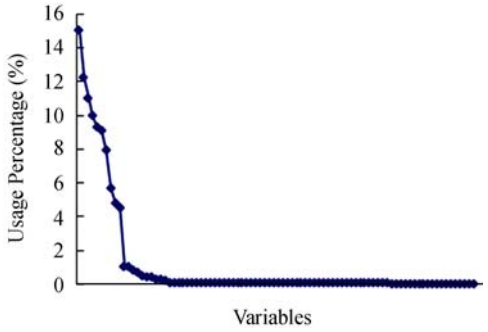


Fig.8. Usage percentage of variables.

Although BART works well for L1D AVF prediction, variables used in BART fitting are overmuch. For purpose of facilitating the use of L1D AVF predictor, the predictor must be of 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. Fig.9 shows the refitted BART inferences.

Fig.9 shows the sequence of σ draws over the iterations. The draws of σ nicely wander around the value $\sigma = 2$, implying that BART model is fitted well. Fig.9 plots posterior mean estimate $\hat{f}(x)$ against the true response value y. Vertical lines indicate the 90% posterior intervals for the response value. We can see that most $\hat{f}(x)$ values correlate well with the true response values, and the intervals tend to cover the true response values.

In order to visualize the results of prediction better, we compare the measured and predicted L1D AVF profiles on the test set, shown in Fig.10. We can see that BART predictive method faithfully detects the time-varying behavior of measured L1D AVF, and predicts L1D AVF with high fidelity.

5.5 Comparison of L1D 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[11-12].

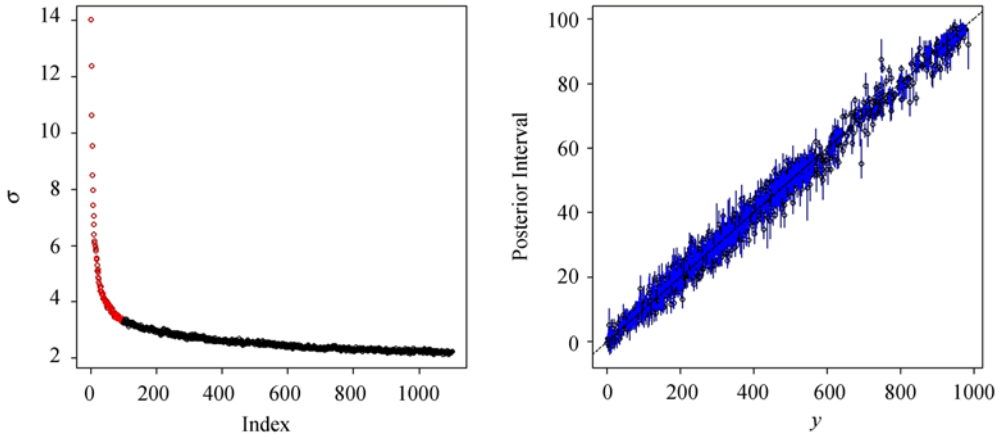


Fig.9. Refitted BART inferences.

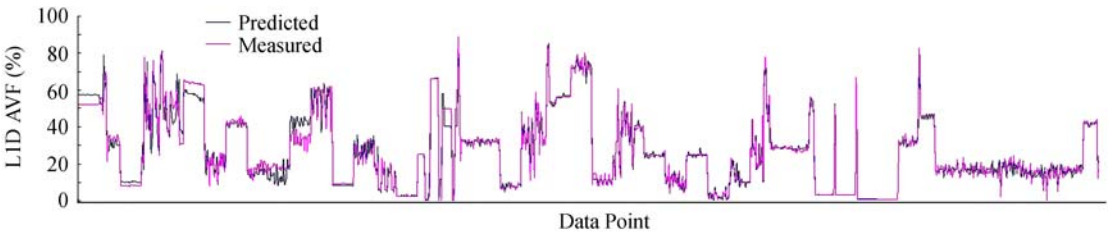


Fig.10. Measured and predicted L1D AVF profiles on the test set.

Firstly, we compare the three predictive models across different test/train splits.

Using the same 18 test/train split as used in Subsection 5.4, we compute the mean absolute error (MAE) of the three different predictive methods on both train set and test set, shown in Fig.11. For each of the three predictive models, MAEs on train set are close to each other across different train/test splits, yet MAEs on test set differentiate among different splits. In the mass, MAEs of BART are less than that of linear model and BRT on both train set and test set across different splits.

To further confirm the superiority of BART across different train/test splits, we use Multiple R-Squared (denoted as R^2) which is the square of correlation coefficient to explain the predictive results. R^2 varies between 0 and 1, computed by (15). Larger R^2 represents a better predictive fit, indicating that the corresponding performance metric correlates well with L1D AVF.

$$R^2 = \frac{Total\_Sum\_of\_Squares - Residual\_Sum\_of\_Squares}{Total\_Sum\_of\_Squares}$$

$$Total\_Sum\_of\_Squares = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

$$Residual\_Sum\_of\_Squares = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad (15)$$

y_i is the response value (i.e., L1D AVF) of each interval, $\bar{y}$ is the average L1D AVF of all the intervals in a program, and $\hat{y}$ is the predicted L1D AVF of each interval.

Fig.12 shows R^2 of different models on train set and test set. Each 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 model, BRT and BART. On average, R^2 of the simple linear model, BRT and BART on the train set is 40%, 80%, and 99%, and R^2 of the three models on the test set is 31%, 65%, and 80% respectively. BART achieves higher R^2 than the simple linear and BRT models on both train set and test set.

Secondly, using the best train/test split in Subsection 5.4, we compare these three predictive models over different model sizes, shown in Fig.13. R^2 of three models on the train 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 the train set gradually achieves a stable value. R^2 of three models on the test set varies irregularly when the model size is less than 8. We can see that R^2 of three models on the test set arrives at a stable value with model size equal to or more than 8. Compared with the simple linear and BRT models, BART obtains the best R^2 on train set and test set.

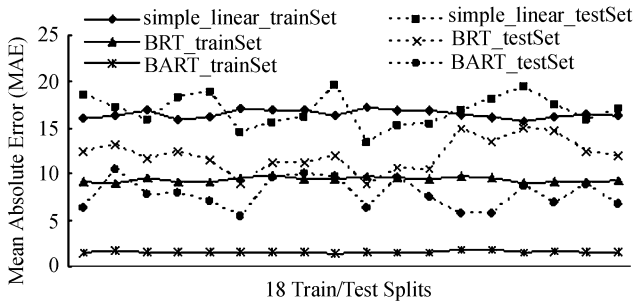


Fig.11. Mean absolute error of different models on 18 train/test splits.

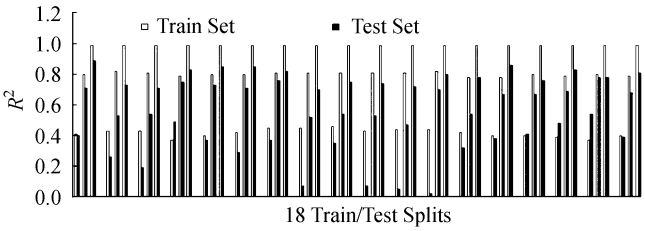


Fig.12. R^2 of different models on 18 train/test splits.

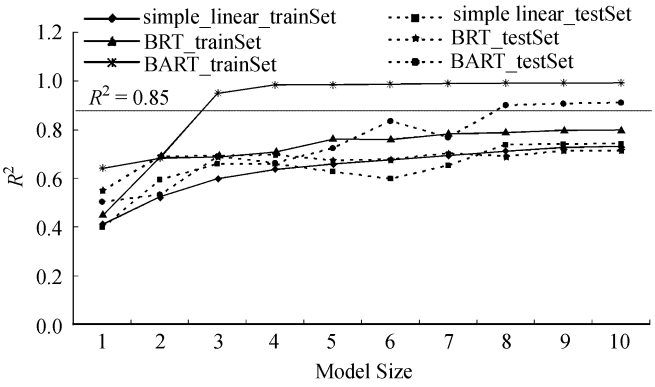


Fig.13. R^2 of different models over different model sizes.

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

6 Simplified and Fast Estimation of L1D AVF

6.1 Simplified and Fast L1D AVF Predictor

In order to integrate the AVF prediction into AVF-aware dynamic fault tolerant management schemes, we consider reducing the complexity of L1D AVF prediction to get a simplified and fast AVF predictor. BART method is proved to perform well for L1D

AVF prediction, but the dimensionality of the predictive function is high, even the refitted BART function which relies on only 10 variables is not suitable for dynamic prediction. So we further employ bump hunting technique[32] to obtain the simplified L1D 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 it has been successfully used for fast estimation of AVF of IQ and ROB[12]. In our study, we aim to find the top 10% intervals of high L1D AVF. We apply bump hunting technique to our dataset and extract 5 simple selecting rules on several key performance metrics, to identify intervals of high L1D AVF.

Fig.14 shows the simple selecting rules extracted by bump hunting technique. The rules can be simply implemented using several NAND and INV gates in hardware. Besides, since the dependent performance metrics of the rules can be easily monitored during runtime, simplified and fast L1D AVF prediction is available using the selecting rules in Fig.14.

if ($sim\_num\_loads > 777284.2$
& & $sim\_num\_stores > 380936.7$
& & $IPC > 1.88$
& & $ruu\_occupancy > 35.367$
& & $bpred\_add\_rate > 0.86$)
{
L1D is regarded to exhibit high architectural vulnerability across this interval.
}

Note: *sim_num_loads* and *sim_num_store* are total numbers of loads and stores committed respectively. *IPC* is instructions per cycle. *ruu_occupancy* is the average RUU occupancy. *bpred_addr_rate* is the branch address-prediction rate.

Fig.14. Simple selecting rules extracted by bump hunting technique.

We employ the selecting rules on part of test set, and the result is illustrated in Fig.15. Generally, we can identify intervals of high L1D AVF correctly, but there are also some intervals of low L1D AVF mistaken as intervals of high L1D AVF. Totally, 2.7% intervals of test set are falsely identified. The number of falsely identified intervals is considerably small, so we believe that the above simple selecting rules work effectively for fast L1D AVF estimation.

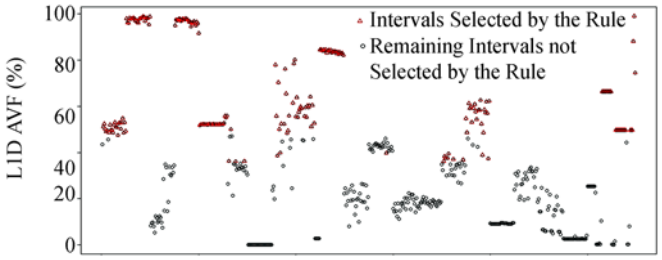


Fig.15. Fast estimation of L1D AVF on the test set.

6.2 AVF-Aware ECC Technique

ECC-based protection techniques have been widely employed in the modern microprocessor design to improve the reliability of cache, and SEC-DED is a representative of them. Previous studies[22-24] have shown that implementing SEC-DED can increase L1 cache access latency by up to 95%, and power consumption by up to 22%.

Provided that L1D ECC checker, consisting of encoder and decoder, is on the critical data path, we implement AVF-aware ECC technique in SS-SERA to reduce the potential over-protection by enabling ECC protection only for the execution points of high vulnerability.

In our experiment, cache is initially simulated with ECC (i.e., SEC-DED) disabled. To identify the top 10% intervals of high L1D AVF, the cache AVF threshold is set as 47. Every two million cycles, we get the instantaneous L1D AVF value using our simplified AVF predictor. If L1D AVF is above the threshold, ECC

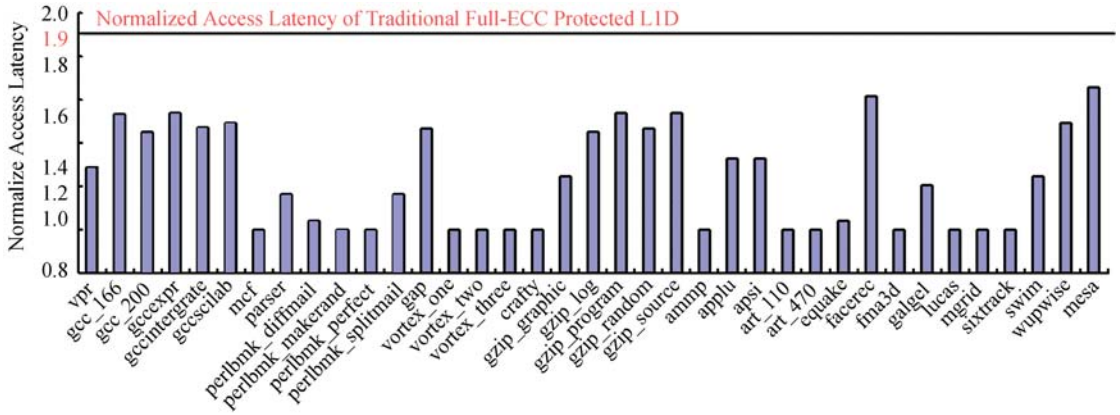


Fig.16. Normalized access latency.

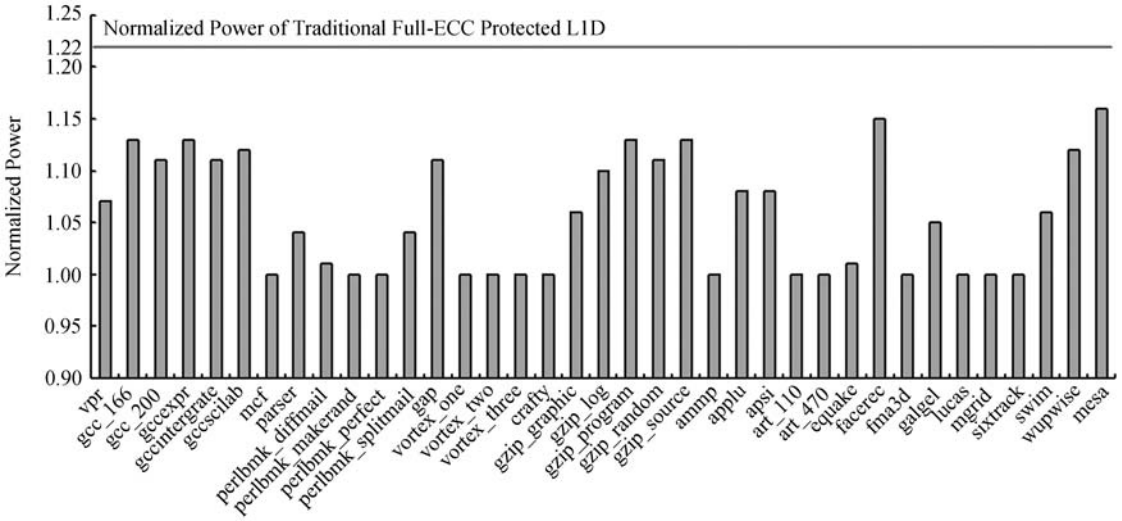


Fig.17. Normalized power.

protection is enabled for the following simulation intervals. Once ECC protection is enabled, we need a criterion to disable it. Hence, we disable ECC after ten million cycles in our experiment.

We use CACTI 6.0[44] which is modified for more detail modeling of ECC technique to determine the latency and power of L1D with/without ECC protection. Using these latency and power values, we simulate the SPEC2K benchmarks in SS-SERA again. We evaluate the access latency and power consumption of L1D for different schemes (i.e., traditional full-ECC, AVF-aware ECC, without ECC), as shown in Fig.16 and Fig.17. We normalize the access latency and power consumption values to that of unprotected L1D.

The experimental results show that most programs benefit from AVF-aware ECC technique. On average, for SPEC2K benchmarks, AVF-aware ECC technique reduces the access latency by 35% and the power consumption by 14% compared with traditional full ECC protection technique.

7 Conclusion

In this paper, we propose an improved methodology of computing cache AVF. Based on the more accurate estimation of L1D AVF, we characterize the time-varying L1D vulnerability behavior and detect the correlations between L1D AVF and several key performance metrics. We propose to employ BART to model the variation of L1D AVF, and the experimental results show that BART works well for accurate L1D AVF prediction. Then, in order to incorporate the AVF predictor into runtime fault tolerant management schemes, we employ bump hunting technique to reduce the complexity of L1D AVF prediction. Some simple selecting rules on several key performance metrics are extracted by bump hunting technique, thus enabling a simplified and fast estimation of L1D AVF. Based on the simplified estimation of L1D AVF, intervals of high L1D AVF can be identified online, thus motivating the development of AVF-aware ECC technique. We demonstrate that AVF-aware ECC technique could provide performance and energy gains without sacrifice much reliability.

With dramatic scaling in feature size of VLSI technology, processor designers are required to more carefully trade off performance, power and reliability. With the ability to predict AVF fast and accurately at runtime, comes the opportunity of AVF-aware fault tolerant technique. Our proposed AVF-aware ECC technique suggests computer architect a unique way of cache design which achieves good tradeoffs among reliability, performance and power, and it has significant advantages especially for applications which are tolerant to soft errors but more severe to performance/power.

Acknowledgements We thank Yu-Xing Tang for his help and suggestions in writing this paper. We thank the anonymous reviewers for their helpful comments and suggestions.

References

[1] Cai Y, Schmitz M T, Ejlali A, AlHashimi B M, Reddy S M. Cache size selection for performance, energy and reliability of time-constrained systems. In *Proc. the 2006 Asia and South Pacific Design Automation Conference* (*ASP-DAC*), Yokohama, Japan, Jan. 24-27, 2006, pp.923-928.

[2] Gaisler J. Evaluation of a 32-bit microprocessor with built-in concurrent error-detection. In *Proc. the 27th International Symposium on Fault-Tolerant Computing* (*FTCS 1997*), Seattle, USA, June 25-27, 1997, p.42.

[3] Mitra S, Seifert N, Zhang M, Shi Q, Kim K. Robust system

design with built-in soft-error resilience. *IEEE Computer*, February, 2005, 38(1): 43-52.

[4] Baumann R C. The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction. In *Proc. International Electron Devices Meeting* (*IEDM 2002*), San Jose, USA, Feb. 26-Mar. 1, 2002, pp.329-332.

[5] Agarwal A, Paul B C, Mukhopadhyay S, Roy K. Process variation in embedded memories: Failure analysis and variation aware architecture. *IEEE Journal of Solid-State Circuits*, September, 2005, 40(9): 1804-1814.

[6] Lambert D, Baggio J, Ferlet C V, Flament O, Saigne F, Sagnes B, Buard N, Carriere T. Neutron-induced SEU in bulk SRAMs in terrestrial environment: Simulations and experiments. *IEEE Trans. Nuc. Sci.*, 2004, 51(6): 3435-3441.

[7] Granlund T, Granbom B, Olsson N. Soft error rate increase for new generations of SRAMs. *IEEE Trans. Nuc. Sci.*, 2003, 50(6): 2065-2068.

[8] Mukherjee S S, Weaver C, Emer J, Reinhardt S, Austin T. A systematic methodology to compute the Architectural Vulnerability Factors for a high-performance microprocessor. In *Proc. the International Symposium on Microarchitecture* (*MICRO*), San Diego, USA, Dec. 3-5, 2003, pp.29-40.

[9] Wang N J, Quek J, Rafacz T M, Patel S J. Characterizing the effects of transient faults on a high-performance processor pipeline. In *Proc. the International Conference on Dependable Systems and Networks* (*DSN*), Florence, Italy, Jun. 28-Jul. 1, 2004, pp.61-70.

[10] Fu X, Poe J, Li T, Fortes J. Characterizing microarchitecture soft error vulnerability phase behavior. In *Proc. the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (*MASCOTS*), Monterey, USA, Sept. 11-14, 2006, pp.147-155.

[11] Walcott K R, Humphreys G, Gurumurthi S. Dynamic prediction of architectural vulnerability from microarchitectural state. In *Proc. the International Symposium on Computer Architecture* (*ISCA*), San Diego, USA, Jun. 9-13, 2007, pp.516-527.

[12] Duan L, Li B, Peng L. Versatile prediction and fast estimation of Architectural Vulnerability Factor from processor performance metrics. In *Proc. the 15th IEEE International Symposium on High Performance Computer Architecture* (*HPCA*), Raleigh, USA, Feb. 14-18, 2009, pp.129-140.

[13] Soundararajan N, Parashar A, Sivasubramaniam A. Mechanisms for bounding vulnerabilities of processor structures In *Proc. the International Symposium on Computer Architecture* (*ISCA*), San Diego, USA, Jun. 9-13, 2007, pp.506-515.

[14] Alpha 21264 Microprocessor Hardware Reference Manual. Digital Equipment Corporation, July 1999.

[15] Reick K, Sanda P N, Swaney S, Kellington J W, Mack M, Floyd M, Henderson D. Fault-tolerant design of the IBM Power6 microprocessor. *IEEE Micro*, March, 2008, 28(2): 30-38.

[16] AMD Athlon(TM) 64 Processor. http://www.amd.com, 2008.

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

[18] Rusu S, Muljono H, Cherkauer B. Itanium 2 processor 6 M: Higher frequency and larger L3 cache. *IEEE Micro*, March, 2004, 24(2): 10-18.

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

[20] Liu C, Gu Y, Sun L, Yan B, Wang D. R-ADMAD: High reliability provision for large-scale de-duplication archival storage systems. In *Proc. the 23rd International Conference on Supercomputing*, Yorktown Heights, USA, June 8-12, 2009, pp.370-379.

[21] Gao X, Chen Y J, Wang H D, Tang D, Hu W W. System architecture of Godson-3 multi-core processors. *Journal of Computer Science and Technology*, 2010, 25(2): 181-191.

[22] Shrivastava A, Lee J, Jeyapaul R. Cache vulnerability equations for protecting data in embedded processor caches from soft errors. In *Proc. the ACM SIGPLAN/SIGBED 2010 Conference on Languages, Compilers, and Tools for Embedded Systems* (*LCTES*), Stockholm, Sweden, Apr. 13-15, 2010, pp.143-152.

[23] Li J F, Huang Y J. An error detection and correction scheme for RAMs with partial-write function. In *Proc. the 2005 IEEE International Workshop on Memory Technology, Design, and Testing* (*MTDT*), Taipei, China, Aug. 3-5, 2005, pp.115-120.

[24] Phelan R. Addressing soft errors in ARM core-based designs. Technical Report, ARM, 2003.

[25] Mohr K C, Clark L T. Delay and area efficient first-level cache soft error detection and correction. In *Proc. International Conference on Computer Design*, San Jose, USA, Oct. 1-4, 2006, pp.88-92.

[26] Kim J, Hardavellas N, Mai K, Falsafi B, Hoe J. Multi-bit error tolerant caches using two-dimensional error coding. In *Proc. the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, Chicago, USA, Dec. 1-5, 2007, pp.197-209.

[27] Li L, Degalahal V, Vijaykrishnan N, Kandemir M, Irwini M. Soft error and energy consumption interactions: A data cache perspective. In *Proc. International Symposium on Low Power Electronics and Design* (*ISLPED*), Newport Beach, USA, Aug. 9-11, 2004, pp.132-137.

[28] Sadler N N, Sorin D J. Choosing an error protection scheme for a microprocessor's L1 data cache. In *Proc. International Conference on Computer Design* (*ICCD*), San Jose, USA, Oct. 1-4, 2006, pp.499-505.

[29] Yoon D, Erez M. Memory mapped ECC: Low-cost error protection for last level caches. In *Proc. the 36th International Symposium Computer Architecture* (*ISCA*), Austin, USA, Jun. 20-24, 2009, pp.116-127.

[30] Yoon D, Erez M. Virtualized and flexible ECC for main memory. In *Proc. the 15th Architectural Support for Programming Languages and Operating Systems* (*ASPLOS*), Pittsburgh, USA, Mar. 13-17, 2010, pp.397-408.

[31] Chipman H A, George E I, McCulloch R E. Bayesian ensemble learning. Neural Information Processing Systems, 19, Scholkopf B, Platt J, Hoffman T (eds.), Cambridge: MIT Press, MA, 2007.

[32] Friedman J, Fisher N. Bump hunting in high-dimensional data. *Statistics and Computing*, 1999, 9(2): 123-143.

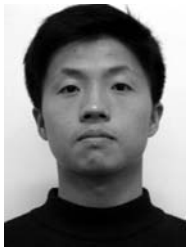
[33] Biswas A, Cheveresan R, Emer J, Mukherjee S S, Racunas P B, Rangan R. Computing architectural vulnerability factors for address-based structures. In *Proc. the International Symposium on Computer Architecture* (*ISCA*), Madison, USA, Jun. 4-8, 2005, pp.532-543.

[34] Fu X, Li T, Fortes J. Sim-SODA: A framework for microarchitecture reliability analysis. In *Proc. the Workshop on Modeling, Benchmarking and Simulation* (*Held in conjunction with International Symposium on Computer Architecture*), June, 2006.

[35] Li X, Adve S V, Bose P, Rivers A. Soft-Arch: An architecture-level tool for modeling and analyzing soft errors. In *Proc. the International Conference on Dependable Systems and Networks* (*DSN*), Yokohama, Japan, Jun. 28-Jul. 1, 2005, pp.496-505.

[36] SimAlpha Homepage. http://www.arch.cs.titech.ac.jp/∼kise/SimAlpha/index.htm, 2003.

[37] Burger D, Austin T. The SimpleScalar Toolset, Version 3.0. http://www.simplescalar.com, 2001.
[38] Friedman J H. Multivariate adaptive regression splines. *Annals of Statistics*, 1991, 19(1): 1-67.
[39] Vapnik V N. The Nature of Statistical Learning Theory. New York: Springer-Verlag New York, Inc., NY, 1995.
[40] Breiman L. Random forests. *Machine Learning*, October 2001, 45(1): 5-32.
[41] Friedman J. Greedy function approximation: A gradient boosting machine. *Annuals of Statistics*, 2001, 29(5): 1189-1232.
[42] Sherwood T, Perelman E, Hamerly G, Calder B. Automatically characterizing large scale program behavior. In *Proc. the 10th International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS*), San Jose, USA, Oct. 5-9, 2002, pp.45-57.
[43] Duesterwald E, Cascaval C, Dwarkadas S. Characterizing and predicting program behavior and its variability. In *Proc. the 12th International Conference on Parallel Architectures and Compilation Techniques*, New Orleans, USA, Sept. 27-Oct. 1, 2003, p.220.
[44] CACTI 6.0. http://www.cs.utah.edu/∼rajeev/cacti6/, 2009.



An-Guo Ma received his B.S. and M.S. degrees in computer science from National University of Defense Technology in 2003 and 2005 respectively. He is a Ph.D. candidate at School of Computer Science, National University of Defense Technology. His current research interests include reliability, low power design, GPGPU and high performance computing.

Yu Cheng received her B.S. and M.S. degrees in computer science and electronic science from National University of Defense Technology in 2005 and 2007 respectively. She is a Ph.D. candidate at School of Computer Science, National University of Defense Technology. Her current research interests include reliability, ultra low power and high performance computing.

Zuo-Cheng Xing is a professor at School of Computer Science, National University of Defense Technology. His main research interests include power-efficient processor architecture design, low-power architecture, GPGPU, and reliability. He is a senior member of CCF.