Journal of Systems Architecture 55 (2009) 457–467

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Accurate and efficient processor performance prediction via regression tree based modeling

Bin Li [a], Lu Peng [b,*], Balachandran Ramadass [b]

[a] Department of Experimental Statistics, Louisiana State University, Baton Rouge, LA 70803, USA
[b] Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803, USA

ARTICLE INFO

Article history:
Received 6 February 2009
Received in revised form 11 August 2009
Accepted 16 September 2009
Available online 20 September 2009

Keywords:
Modeling techniques
Modeling of computer architecture
Measurement
Evaluation
Modeling
Simulation of multiple-processor systems

ABSTRACT

Computer architects usually evaluate new designs using cycle-accurate processor simulation. This approach provides a detailed insight into processor performance, power consumption and complexity. However, only configurations in a subspace can be simulated in practice due to long simulation time and limited resource, leading to suboptimal conclusions which might not be applied to a larger design space. In this paper, we propose a performance prediction approach which employs state-of-the-art techniques from experiment design, machine learning and data mining. According to our experiments on single and multi-core processors, our prediction model generates highly accurate estimations for unsampled points in the design space and show the robustness for the worst-case prediction. Moreover, the model provides quantitative interpretation tools that help investigators to efficiently tune design parameters and remove performance bottlenecks.

1. Introduction

Computer architects usually evaluate new designs using employing cycle-accurate processor simulators which provide a detailed insight into processor performance, power consumption and complexity. A huge design space is composed by the product of the choices of many microarchitectural design parameters such as processor frequency, issue width, cache size/latency, branch predictor settings, etc. To achieve an optimal processor design, a wide configuration spectrum of the design space has to be tested before making a final decision. However, only configurations in a subspace can be simulated in practice due to long simulation time and limited resources, leading to suboptimal conclusions which might not be applied to the whole design space. In addition, more parameters brought by chip-multiprocessors (CMPs) make this problem more urgent. In this paper, we propose to use a state-of-the-art tree-based predictive modeling method combined with advanced sampling techniques from statistics and machine learning to explore the microarchitectural design space and predict processor performance. This bridges the gap between simulation requirements and simulation time and resource costs.

The proposed method includes the following four components. (1) The maximin space-filling sampling method that selects initial design representatives from a large amount of design alternatives. (2) The state-of-the-art predictive modeling method – Multiple Additive Regression Trees (MART) [8] which builds an ensemble of trees with a highly prediction accuracy. (3) An active learning method which adaptively selects the most informative design points needed to improve the prediction accuracy sequentially. (4) Interpretation tools for MART-fitted models which are able to show the importance and partial dependence of design parameters and shed light on the underlying working mechanism. These provide computer architects a quantitative and efficient approach to optimize processor performance by adapting key design factors.

For each workload, 500 initial design points were sampled based on the maximin distance methods chooses a subset of design points (from the entire design space) in which the smallest pairwise distance is maximized (detailed in Section 2.1). Then another 500 points were sampled according to an adaptive sampling scheme (described in Section 2.3). We repeat the sampling process until 3000 design points were sampled for each benchmark. An independent test set which consists of another 5000 points is used to evaluate the prediction performance of fitted models. After sampling and testing, the interpretation is performed based on the fitted MART model with all the 3000 sampled points.

According to our experiments on 12 SPEC CPU2000 and four SPLASH2 benchmarks, by sampling 3000 points drawn from a microarchitecture design space with nearly 15 million configurations for each SPEC program and a CMP design space with nearly 9.7 million points for each SPLASH2 workload, we can summarize the following results:

* Corresponding author. Tel.: +1 225 578 5535; fax: +1 225 578 5200.
E-mail addresses: bli@lsu.edu (B. Li), lpeng@lsu.edu (L. Peng), bramad2@lsu.edu (B. Ramadass).

1383-7621/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2009.09.004

1. Performance prediction: application-specific models predict performance, based on 5000 independent sampled design points, with median percentage error ranges from 0.3% to 3.0% for our single-core processor performance prediction in a design space with about 15 million points and 0.5–1.4% for the CMP performance prediction in a design space with about 9.7 million points.
2. Worst-case performance: for the single-core processor performance prediction, the worst percentage errors are within 7% for 10 out of the 12 benchmarks; the largest worst-case percentage error is 22.5% for *art*. In the CMP performance study, the worst percentage errors are within 16.1% for all the four benchmarks.
3. Relative performance: compared to a classical linear regression model with a random sampling scheme, our method typically reduces 88% and 98% average percentage error for the single-core and CMP study, respectively. Our method also has 8% and 37% lesser average percentage error in the single-core and CMP study separately than Multiple Additive Regression Trees with a random sampling scheme.
4. Model interpretation: the model can be used to explain variable importance and partial dependence to each variable. For the two selected benchmarks, we find that "Width/ALU" (processor issue width/the number of ALU units) and "L2Size" (L2 cache size) are key factors to *bzip2* while "Load store queue (LSQ)" is important to *mcf*. Tuning these factors helps to improve processor performance to these programs. The partial dependence plots clearly illustrate processor design trends and bottlenecks for the processor.

The remainder of this paper is organized as follows. Section 2 introduces the MART-aided methodology. Section 3 describes the experimental setup, while experiment results are presented in Section 4. The model interpretation tools and model discussion are demonstrated in Section 5. Section 6 introduces the related works. Section 7 concludes this paper.

2. Methodology and background

We propose to use the nonparametric tree-based predictive modeling method combined with advanced sampling techniques from statistics and machine learning to explore the microarchitectural design space efficiently. The fitted model based on hundreds of regression trees can be summarized, interpreted and visualized similarly to conventional regression models.

Table 1a
Single core processor simulation parameters.

Parameters	Values
Fetch/issue/commit width	2/4/8 instructions
Branch predictor: bimod predictor or 2-level predictor <L1>/<L2>/<history size>	Bimod predictor – 4096 entries or 8192 entries 2-level predictor: 1/4096/10, 1/8192/10, 2/4096/10, 2/8192/10, 4/4096/10, 4/8192/10
Branch target buffer <number of sets>/<assoc>	1024/4, 2048/2, 1024/8, 2048/4
Integer/floating ALUs	1/1, 2/1-related to issue width 2 2/1, 2/2-related to issue width 4 2/2, 4/4-related to issue width 8
Register update units	64/128/256 entries
Load store queue	16/32/64 entries
L1 inst. cache size	8 K, 16 K, 32 K, 64 KB
L1 data cache size	8 K, 16 K, 32 K, 64 KB
L2 unified cache size	256 K, 512 K, 1024 K, 2048 KB
Memory latency	100, 140, 180, 220, 260 cycles
L1 I/D and L2 block size	32B, 64B, 128B
L1 I/D cache associativity	1, 2, 4
L2 cache associativity	4, 8, 16

Table 1b
CMP simulation parameters.[a]

Parameters	Values
Core configuration	In order, out of order
Issue width	1, 2, 4
Number of cores	1, 2, 4, 8
Off chip bandwidth	4, 8, 16, 24, 32 bytes/cycles
Memory latency (cycles)	200, 250, 300, 350, 400, 450, 500
L2 cache size	1, 2, 4, 8 MB
L2 cache block size	32, 64, 128 B
L2 cache associativity	1, 2, 4, 8, 16 way
L2 cache hit latency	7, 9, 11, 13 cycles
L2 replacement policy	LRU, random
L1 I/D cache size	32 KB, 64 KB
L1 I/D cache block size	32, 64 B
L1 I/D cache associativity	1, 2, 4
Branch predictor	Hybrid, 2-level

[a] L2 cache is shared and L1 I/D cache are private to each core.

2.1. Maximin distance design

In experiment design, the distance-based space-filling sampling methods are popular, especially, when we believe that interesting features of the true model are just as likely to be in one part of the experimental region as another. Among them, the maximin distance design is commonly used. The maximin distance criterion chooses a subset of design points (from the entire design space) in which the smallest pairwise distance is maximized.

In our study, some architectural design parameters are nominal with no intrinsic ordering structure and the others are discrete with a small number of values (see Table 1). Hence, we define the following distance measure used in our study. Let wt_j be the weight for the jth design parameter. The distance between design points x_1 and x_2 are defined as

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sum_{j=1}^{p} [wt_j \times I(x_{1j} \neq x_{2j})], \tag{2.1}$$

where $wt_j = \log_2(\text{number of levels for } j\text{th design parameter})$ and $I(A)$ is an indicator function, equal to one when A holds, otherwise zero. Note that the weight for each design parameter is equal to its information entropy with uniform probability for each of its possible values.

2.2. Regression model with MART

MART is one of several techniques that aim to improve the performance of a single model by fitting many models and combining them for prediction. MART consists of two parts: classification and regression trees (CART [1]) and boosting technique.

CART analysis consists of two basic steps. The first step consists of tree building, during which a tree is built using recursive binary splitting. The term "binary" implies that we first split the space into two regions, and model the response by a constant for each region. Then we choose the variable and split-point to achieve the best fit again on one or both of these regions. Thus, each node can be split into two child nodes, in which case the original node is called a parent node. The term "recursive" refers to the fact that the binary partitioning process can be applied over and over again until some stopping criterion is reached. Each resulting node is assigned a value, which is based on the distribution of the observations in the training set that fall in that node. The second step consists of tree "pruning", which results in the creation of a sequence of simpler trees, through the cutting off the weakest links.

Tree-based methods are popular because they represent information in a way that is intuitive and easy to visualize, and have several other advantageous properties. First, the tree is inherently nonparametric and can handle mixed-type of input variables naturally, i.e. no assumptions are made regarding the underlying distribution of the values for the input variables, e.g. numerical data that are highly skewed or multi-modal, as well as categorical predictors with either ordinal or non-ordinal structure. This eliminates researchers' time which would otherwise be spent in determining whether variables are normally distributed, and making transformation if they are not. Second, the tree is adept at capturing non-additive behavior, i.e. complex interactions among predictors are routinely and automatically handled with relatively few inputs required from the analyst. This is in marked contrast to some other multivariate nonlinear modeling methods, in which extensive input from the analyst, analysis of interim results, and subsequent modification of the method are required. Third, the tree is insensitive to outliers, and unaffected by monotone transformations and differing scales of measurement among inputs. Despite these benefits, tree is not usually as accurate as its competitors, and small changes in training data can result in very different series of splits [8].

Boosting is a general method for improving model accuracy of any given learning algorithm, based on the idea that it is easier to find an average many rough rules of thumb than to find a single highly accurate prediction rule [20]. The idea of boosting has its roots in PAC (Probably Approximately Correct) learning [23]. Kearns and Valiant [12] proved the fact that "weak" learners, each performing only slightly better than a random decision, can be combined to form a "powerful" learner. In boosting, models (e.g. trees) are fitted iteratively to the training data, using appropriate methods to gradually increase emphasis on observations modeled poorly by the collection of trees. Boosting algorithms vary in how they quantify lack of fit and select settings for the next iteration.

In MART, boosting is a form of "functional gradient descent". MART approximates the underlying function $f(x)$ by an additive expansion

$$\hat{f}(x) = \sum_{m=1}^{M} \beta_m b(x;\gamma_m), \tag{2.2}$$

where the expansion coefficients $\{\beta_m\}_1^M$ and the tree parameters $\{\gamma_m\}_1^M$ are jointly fitted into the training data. The model is fitted in a forward "stagewise" (not "stepwise") fashion, meaning that existing trees are left unchanged as the model is enlarged. In MART, one starts with an initial guess $\hat{f}_0(\mathbf{x})$ and then for each iteration m = 1, ..., M:

(1) Compute the negative gradient $\{g_i\}_{i=1}^n$ as the working response

$$g_i = \frac{\partial L(y_i, \mathbf{x_i})}{\partial f(\mathbf{x_i})}|_{f(\mathbf{x_i})=\hat{f}(\mathbf{x_i})}, \tag{2.3}$$

where $L(y,\hat{f})$ is some loss function we expect to minimize. In this study, loss function L is the squared error loss function and g is partial derivative of L with respect to $f(x)$.

(2) Fit a regression tree $b(\mathbf{x};\gamma)$ and a gradient descent step size β which minimizes

$$\sum_{i=1}^{n} L(y_i, \hat{f}_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i;\gamma)). \tag{2.4}$$

Note that the gradient descent step size β can be viewed as the weight for the regression tree that minimizes the loss function.

(3) Update the estimate of f(x) as

$$\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \beta b(\mathbf{x};\gamma). \tag{2.5}$$

Namely, the updated estimator is the summation of the previous estimator and the weighted newly-fitted regression tree.

From a user's perspective, MART, as applied in this paper, has the following features. First, the model fitting process is stochastic – i.e. it includes a random or probabilistic component. Particularly, within each iteration, the regression tree is fitted based on a bootstrap sample of the training set. The observations selected to fit the regression tree are called in-bag samples, while the observations not selected are called out-of-bag samples. The stochasticity (from randomly select bootstrap samples within each iteration) improves predictive performance, reducing the variance of the final model, by only using a random subset of data to fit each new tree [7]. This means that, unless a random seed is set initially, final models will subtly differ each time they are run. Second, the gradient descent step determines the contribution of each tree to the growing model. Studies in Ref. [6] show that using small values of gradient descent step size always lead to better prediction performance. Hence, we fix β at 0.01 in this study. Third, the tree complexity controls whether interactions are fitted: a tree with depth of one ("stump" with only two terminal nodes) fits an additive model without including any interaction. In this study, we set the maximum depth for trees at three, which fits a model with up to three-way interactions. Finally, the choice of M, i.e., when to stop the boosting algorithm, is based on monitoring the estimation performance on out-of-bag samples, which are the set of observations not selected (from the training data) to fit trees within each iteration.

In this paper, MART is run by using the **gbm**, an **R** implementation of MART package produced by Greg Ridgeway [17]. For details of MART, we refer the readers to Ref. [8]. For details of applying MART in gbm, we refer the readers to its online manual at http://cran.r-project.org/web/packages/gbm/gbm.pdf.

2.3. Adaptive sampling

Adaptive sampling, also known as active learning in machine learning literature, involves sequential sampling schemes that use information learned from previous observations to guide the sampling process. Several empirical and theoretical studies have shown that adaptively selecting samples in order to learn a target function can outperform conventional sampling schemes. For example, Freund et al. [5], Saar-Tsechansky and Provost [18] and Sung and Niyogi [22] all employed this method.

In this paper, we select the 500 subsequent samples in the following four steps:

(1) Apply MART on the sampled points 20 times with different random seeds.
(2) For each of the MART-fitted model, predict the unsampled points in the design space.
(3) Sort these points (in a decreasing order) according to the coefficient of variance (CoV, the ratio of standard deviation to mean) for the model prediction.
(4) Set an initial distance threshold θ. First, include the least confident point (one with the largest CoV value). Then, include the next least confident points if its distance to the first point is above θ. After that, we include the least confident points if its distances to the first two points are above θ. Repeat the procedure. If all 500 points have been selected, increase the value of distance threshold θ slightly and restart the selection procedure until we reach the maximum threshold that allows us to select all 500 points.

Consider an easy unsampled point (in terms of its predictability). The MART models (with different random seeds) should have very similar predicted value on this point. However, for a difficult point (with high uncertainty in prediction), then different models will provide very different prediction values. Hence, to improve the model by sampling more points, we need to select the most uncertain ones, which can benefit the model greater than the easy ones.

The intuition of selecting the subsequent samples as above is based on the bias-variance decomposition. Note that the decomposition is originally proposed for the squared loss, but can be generalized to other losses such as zero-one losses for classification [3]. In practice, since the bias is unknown before measuring, we can only measure the variance of predictions. However, there are two concerns. First, we often care more about the relative scale than the absolute loss in practice. This makes us to sort the design points based on their CoV rather than the variances. Second, if we select the points strictly according to their CoV, the selected ones are often clustered. In other words, they tend to be close to each other in the design space. In order to achieve the global accuracy, we try to select sampling points as separated as possible, although the ones with large CoV should have higher preference to be selected.

2.4. Stopping criterion

The procedure is stopped based on either the time/cost constraint or the performance measure. The former depends on investigators' time and cost budget, while for the latter, we can monitor the procedure based on a cross-validation measure or prediction performance on an independent test set. Namely, if the prediction accuracy is above a pre-specified level and/or the improvement of prediction accuracy is below a pre-specified level, the procedure will be stopped. Since we consider the stopping issue is more on the user-end, in this study, we fix the total number of points we need to sample throughout the paper.

2.5. Interpreting the model

Although it producing a model with hundreds to thousands of trees, MART does not have to be treated like a black box. MART model can be summarized, interpreted and visualized similar to conventional regression models. This involves identifying those parameters that are most influential in contributing to its variation and visualizing the nature of dependence of the fitted model on these important parameters.

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

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

$$F_s(X_s) = E_{\mathbf{x}_{\setminus \mathbf{s}}}[f(x)], \tag{2.6}$$

where $E_{\mathbf{x}_{\setminus \mathbf{s}}}[.]$ means the expectation over the joint distribution of all the input variables with index not in **S**. In practice, partial dependence can be estimated from the training data by $\hat{F}_\mathbf{s}(\mathbf{x}_\mathbf{s}) = (1/n)\sum_{i=1}^{n} \hat{f}(\mathbf{x}_\mathbf{s}, \mathbf{x}_{i\setminus \mathbf{s}})$, where $\{\mathbf{x}_{i\setminus \mathbf{s}}\}_1^n$ are the data values of $\mathbf{x}_{\setminus \mathbf{s}}$.

3. Experimental setup

We have two parts of experiments: single core processor simulation and CMP simulation. For single core simulation, we modified the sim-outorder, the out-of-order pipelined simulator in SimpleScalar [2], to be an eight-stage Alpha-21264 like pipeline. Only 12 (eight integer and four floating point) CPU and memory intensive programs from SPEC2000 were selected. They are *art, bzip2, crafty, equake, fma3d, gcc, mcf, parser, swim, twolf, vortex* and *vpr*. We skipped a number of instructions for each SPEC program based on a previous work [19]. Then we collected the number of execution cycles for the next 100 million instructions. Since we use generic simulators and benchmarks, we believe that the validated model can be applied to other simulators and workloads. Table 1a lists 13 groups of design parameter choices for an out of order (OoO) executed pipelined processor. The cross-product of these choices is about 15 million design points per benchmark, representing an intractably large space to fully simulate.

For CMP simulation, we employed SESC [21] which has detailed out-of-order processor simulation for each core and memory subsystem. We downloaded pre-compiled binaries of four SPLASH2 applications from Ref. [21]. They are *barnes*, *fmm*, *radiosity* and *raytrace*. During execution of these programs, the number of slave threads was set the same as the number of cores. We collected execution cycles for 100 million instructions. For results of multiple cores, we took the average number of execution cycles of all cores as CMP's execution cycles. Table 1b listed 14 groups of design parameter choices for a CMP. For each program, the design space size is about 9.7 millions.

Fig. 1 shows a scheme plot of our sampling, modeling and interpretation procedure. For each workload, 500 initial design points were sampled based on the maximin distance criterion described in Section 2.1. Then another 500 points were sampled according to the adaptive sampling scheme described in Section 2.3. We repeated the sampling process until 3000 design points were sampled for each benchmark. Notice that for the 3000 points, we only explored approximately 0.02% of the total 15 million points in the design space for the single-core study and about 0.03% of the 9.7 million points. An independent test set consisting of another 5000 points was used to evaluate the prediction performance of fitted models. The interpretation was based on the fitted MART model with all the 3000 sampled points.

4. Prediction performance evaluation

Table 2 shows the average and maximum percentage error (PE)

$$\text{PE} = |(\text{Predicted} - \text{Actual})/\text{Actual}\ | \times 100\% \tag{4.1}$$

of our method on an independent test set with 5000 design points. We see that the average PE decreases as the training set sizes increase (via adaptive sampling). In the single-core study, by sampling only 0.02% of sample points from the design space, the mean average PE's are within 1% for 9 out of 12 benchmarks. For *art*, which achieves least accuracy, the mean average PE is 4.18%. Similarly, in the CMP study, by sampling only 0.03% of sample points from the design space, the mean average PE's are all within 2% for the four multithreaded benchmarks.

Lee and Brooks [14] proposed a linear regression modeling method for performance prediction. In their paper, they used the similar design space as ours. Their mean percentage error was 4.9%. İpek et al. [9] proposed a neural network based active learning method to explore the design space. Although their design spaces are quite different from ours, they reported 4–5% error on average on the CMP study based on training the model on 1.03% sample drawn from the design space (about 2500 sampled design

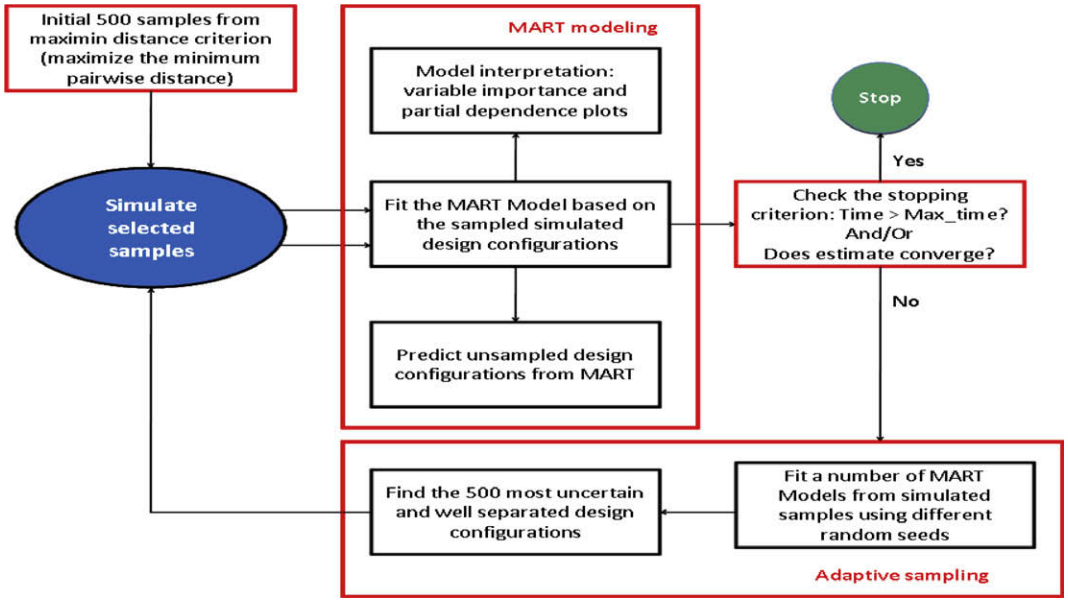

Fig. 1. Overview of the proposed MART model.

Table 2
Summary of relative prediction performance with specified error and sample size.

Benchmark	0.0067% Sample (1000 pts.)		0.013% Sample (2000 pts.)		0.02% Sample (3000 pts.)	
	Mean PE	Max PE	Mean PE	Max PE	Mean PE	Max PE
Single-core study						
art	6.30	42.79	4.63	24.95	4.18	22.56
bzip2	0.73	4.50	0.46	3.16	0.41	3.33
crafty	1.62	13.11	1.02	7.17	0.87	5.53
equake	2.65	18.69	2.26	15.77	2.13	15.04
fma3d	0.91	5.43	0.70	3.36	0.62	2.96
gcc	0.74	4.04	0.49	3.02	0.43	2.26
mcf	0.67	4.99	0.50	4.22	0.46	4.24
parser	0.83	4.90	0.52	3.65	0.42	2.30
swim	1.44	9.59	0.90	5.94	0.66	4.63
twolf	1.83	10.36	1.38	7.53	1.23	6.31
vortex	1.36	13.07	0.93	7.11	0.80	6.88
vpr	0.98	6.93	0.62	4.53	0.53	4.32
	0.01% Sample (1000 pts.)		0.02% Sample (2000 pts.)		0.03% Sample (3000 pts.)	
CMP study						
barnes	2.53	17.26	2.05	13.43	1.90	13.91
fmm	2.17	15.71	1.72	11.78	1.45	10.15
raytrace	1.12	21.90	0.83	17.36	0.75	16.02
radiosity	1.02	9.08	0.79	7.12	0.74	6.4

points). Hence, we conclude that our method has a highly compatible mean PE compared to those two methods.

One advantage for the tree-based methods is that they are highly stable and robust to the outliers (or extreme values). A common way to evaluate the robustness of a method is to check the worst-case performance. Table 2 also shows the maximum percentage errors among the 5000 testing points. By exploring 0.02% of the design space in the single-core study, the maximum PE is less than 10% in 10 out of 12 benchmarks. Even in the least accurate benchmark, *art*, the maximum PE is 22.55%. In the CMP study, the maximum PE is within 20% for the four benchmarks. As comparison, the maximum PE, in Lee and Brooks [14], is 20.298%. In the CMP study from İpek et al. [9], their maximum PE ranges from about 34% to 53%. Therefore, our method has a comparable maximum PE to Lee and Brooks' work and a highly compatible worst-case performance to İpek et al.'s approach. This shows strong robustness of our models.

Fig. 2 shows the average PE's (black line) with different number of sampled points in the single-core study. The *x*-axis represents the portion of full parameter space simulated to form the training sets, and the *y*-axis stands for the average percentage error rate across the independent test set. The dashed lines represent the mean plus/minus one standard deviation of PE's. We see that the average percentage error rate decreases monotonically as the training set size increases (via adaptive sampling).

Fig. 3 depicts the empirical Cumulative Distribution Function (CDF) plots of percentage errors (on test set) with about 0.02% sampled points in the single-core study. The *x*-axis shows the PE, and the *y*-axis shows the percentage of data points that achieve error less than each *x* value. We see that for 9 out of 12 benchmarks, more than 90% of the points are predicted with less than 2% error. The three with largest percentage errors are *art*, *equake* and *twolf*. For these three workloads, more than 90% of the points are predicted with less than 10%, 5% and 3% error,

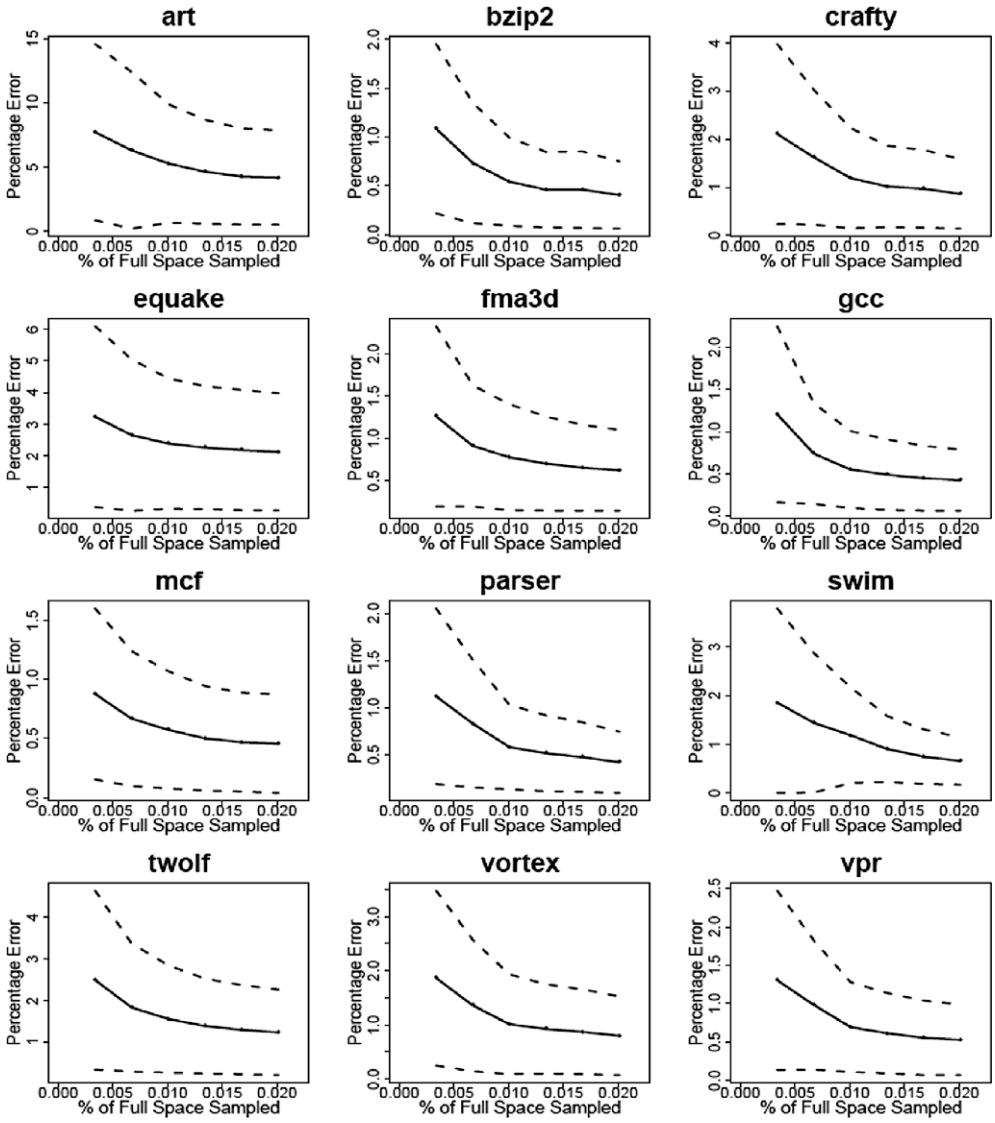

Fig. 2. Prediction accuracy of the models on the design space (adaptive sampling in single-core study).

respectively. Figs. 4 and 5 illustrate the average PE's (with standard deviations) across different number of sampled points and CDF plots in the CMP study. Note that for all the four benchmarks, 90% of the points are predicted with less than 4.2% error.

The reasons that our method achieves high prediction accuracy lie in twofolds: (1) the superior prediction performance inherited from MART; (2) and the proposed adaptive sampling scheme. To illustrate the robustness of our model, we compare the predictive ability of our method with another two regression models, differing in specification and data used to perform the fit. The two models are listed as follows.

(1) $R + M$: Fitting MART on the randomly selected sample points.
(2) $R + L$: Fitting traditional linear regression model with all possible two-way interactions on randomly selected sample points. This resembles the linear regression approach proposed in Ref. [14], which represents one of the best recent predictive models for processor performance.

Table 3 lists the relative percentage errors in $R + L$ and $R + M$ against our proposed method. The numbers in Table 3 are the ratios of the average and maximum PE (based on the test set) from the specified model and the proposed method. Note that in Table 3, the larger the numbers, the more improvements from using our proposed method. If the ratio is equal to one, it means that the average PEs are the same. From Table 3, we have the following two conclusions:

(1) The ratios for $R + L$ are larger than those for $R + M$ in both studies, which means using MART substantially improves the prediction performance under the random sampling scheme. This is due to the fact that MART is adept at capturing nonlinear and non-additive behavior, e.g. nonlinear dependence and interactions among independent variables are routinely and automatically handled.
(2) The ratios of relative prediction performance for $R + M$ are greater than 1 in most of cases of the single-core study and all the cases in the CMP study. The benefit of using adaptive sampling scheme is obvious, especially in the CMP study. For example, with 0.03% of the sampled points, using the proposed adaptive sampling scheme reduces more than 37% of the average PE for all of the four benchmarks and more than 95% worst-case PE for three out of the four bench-

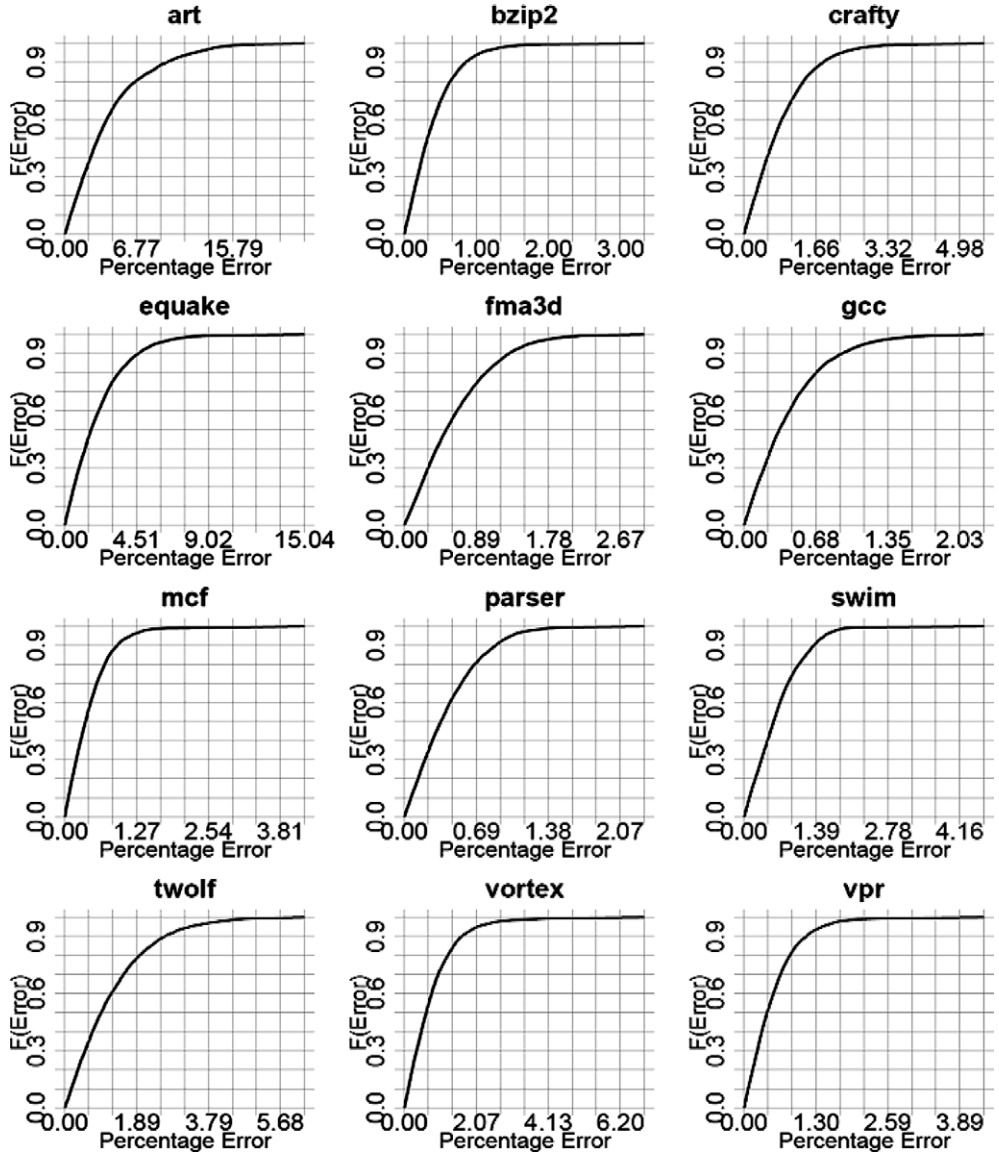

Fig. 3. Empirical CDF of prediction errors (single-core study).

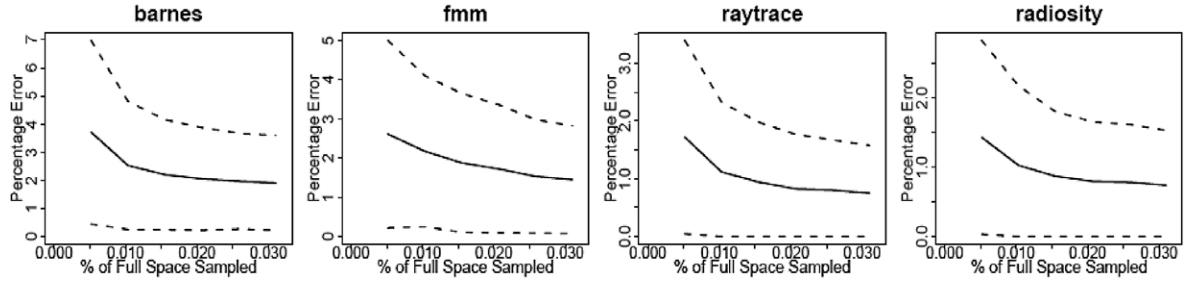

Fig. 4. Prediction accuracy of the models on the design space (adaptive sampling in the CMP study).

marks. Note that 37% comes from $(1.37-1)*100\%$ in the 'Mean PE' column for $R+M$, while 95% comes from $(1.95-1)*100\%$ in the 'Max PE' column for $R+M$. Multi-threaded programs running on CMPs generates high variability which can be easier caught by our adaptive sampling method compared to random sampling approach. The comparisons illustrate the robustness of our proposed method.

5. Model discussion, interpretation and visualization

5.1. Model discussion

5.1.1. Statistical justification of test set size

In this study, we evaluated the prediction performance based on 5000 independently and randomly selected test samples. Compared with the overall design space, it is still a small set. A natural

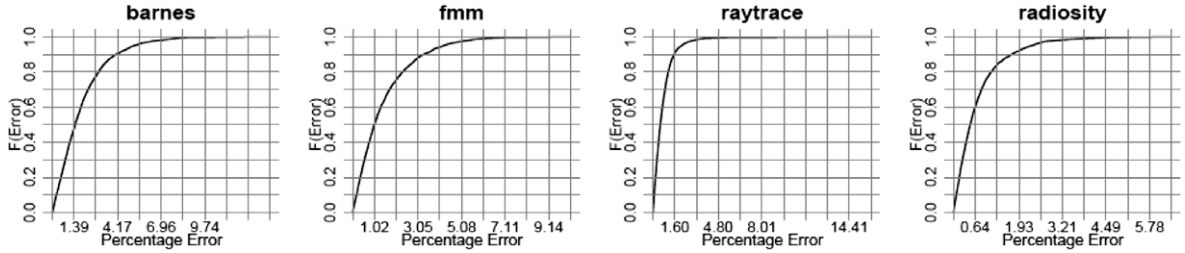

Fig. 5. Empirical CDF of prediction errors for the CMP performance study.

Table 3
Summary of relative predictive accuracy (against $A + M$) with specified sample size.

Benchmark	0.013% Sample (2000 pts.)				0.02% Sample (3000 pts.)			
	$R + L$		$R + M$		$R + L$		$R + M$	
	Mean PE	Max PE	Mean PE	Max PE	Mean PE	Max PE	Mean PE	Max PE
Single-core study								
art	4.05	3.76	1.12	1.09	4.44	3.89	1.00	1.04
bzip2	7.40	4.13	1.21	1.31	8.27	4.02	1.11	1.01
crafty	7.62	6.02	1.11	1.41	8.64	7.41	1.10	1.50
equake	2.74	2.50	1.04	1.00	2.81	2.54	1.04	0.94
fma3d	3.25	3.79	1.07	1.22	3.56	4.32	1.04	1.14
gcc	9.63	7.25	1.07	1.52	10.89	9.47	1.10	1.59
mcf	10.90	4.99	1.06	1.15	11.84	4.58	1.10	1.05
parser	7.83	4.06	1.06	0.90	9.45	6.33	1.08	1.19
swim	14.76	8.99	1.06	0.87	20.11	10.67	1.04	0.78
twolf	3.55	4.26	1.05	1.20	3.90	4.93	1.04	1.44
vortex	3.87	2.87	1.11	1.67	4.38	2.77	1.11	1.40
vpr	13.25	6.89	1.14	1.03	15.26	7.45	1.18	1.02
	0.02% Sample (2000 pts.)				0.03% Sample (3000 pts.)			
CMP study								
barnes	25.68	15.20	1.38	1.89	27.55	12.67	1.37	1.95
fmm	48.50	27.79	1.66	2.19	57.86	31.26	1.83	2.24
raytrace	65.58	10.18	1.64	1.15	72.29	10.12	1.69	1.09
radiosity	65.57	24.39	1.42	1.98	70.36	23.62	1.50	2.01

question is: whether the test sample size is large enough to have an accurate evaluation of the prediction performance. Hence, we applied bootstrapping technique to justify our choice of test sample size. Bootstrapping developed by Efron [4] is a general tool of estimating statistical properties of an estimator (e.g. the 95% confidence interval for the median and mean percentage errors). By using bootstrapping, we can estimate the confidence interval for the mean and median PE based on 5000 test samples. Note that the confidence interval based on bootstrapping does not assume any distribution assumption on the population (i.e. normal distribution assumption on PE's). However, bootstrapping procedure needs resampling the samples (in this case, they are the 5000 test samples) with replacement a large number of times (say 1000 times). We can check whether the test sample size is large enough based on the width of the bootstrap interval. Namely, if the width of the bootstrap interval (BI) is small, it indicates the test sample size is large enough and the mean and median PE based on this test samples has small sampling variation. Table 4 shows the 95% bootstrap interval based on 1000 bootstrapped samples for the mean and median PE in three benchmarks, which have the highest mean PE in single-core and CMP studies. We see that both upper and lower confidence limits for the 95% bootstrapped intervals are very close to their corresponding point estimates. For example, the mean PE for *art* is 4.18 which is very close to its upper and lower limits for the 95% bootstrap interval on mean PE. This indicates that the test sample size is large enough to have an accurate estimate of prediction performance in the study.

Table 4
Summary of bootstrap interval for mean and median PE.

Benchmark	Mean PE	95% BI on mean PE	Median PE	95% BI on median PE
art	4.18	(4.05,4.31)	3.01	(2.90, 3.11)
equake	2.13	(2.08, 2.19)	1.62	(1.57, 1.67)
barnes	1.90	(1.85, 1.94)	1.47	(1.42, 1.51)

5.1.2. Sensitivity study of sampling set size

Like other regression methods, MART typically predict better when trained on more data. On the other hand, data collection in architecture design space exploration is expensive, and a tradeoff exists between number of simulations and model accuracy. As we mentioned in Section 2.4, determination of the training sample size is an end-user issue. Namely, the stopping criterion is based on either the investigator's time and cost budget or convergence of prediction performance. Fig. 6 shows two typical curves of the percent of improvement in the single-core and CMP studies, which is defined as the proportion of improvement for each additional batch over the total improvement (in terms of the mean PE in the test set) in six batches. For example, suppose the mean PE based on the first 500 points (first batch) is 0.11. The mean PE based on the first 1000 points (two batches) is 0.06. The mean PE based on the total 3000 points (six batches) is 0.01. Then the total improvement is $0.11 - 0.01 = 0.1$. The improvement based on the second batch is $0.06 - 0.01 = 0.05$. Hence, the proportion of improvement for the second batch is then (0.05/

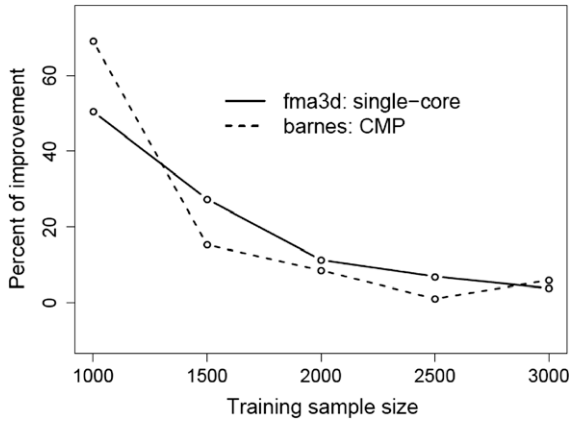

Fig. 6. Sensitivity of training set size.

0.1) × 100% = 50%. Based on Fig. 6, we see that the percent of improvement after the fourth batch (2000 design points) is relatively small comparing to the first three batches in both single-core and CMP studies. Hence, the results suggest the reasonable number of training samples to be 2000 for a large simulation study.

5.2. Variable importance

The proposed method can easily be used to analyze the importance of individual design parameters. Note the explanation of the variable importance measure is presented in Section 2.5. This illustrates an inside view and provide computer architects with efficient directions of improving processor performance. We select two workloads *bzip2* and *mcf* in the single-core study as an example. Fig. 7 shows the relative variable influence, scaled to have a sum added to 100. For *bzip2* shown in the left figure, the most important variables are "Width/ALU" and "L2Size" while "LSQ" is the most important factors for *mcf*. From these figures, we can see that CPU intensive programs such as *bzip2* are very sensitive to the instruction issue width and integer/floating point units as well as L2 cache size. Increasing these parameters provides an efficient way to improve processor performance. *Mcf* presents another sensitivity preference on LSQ size. This is reasonable because *mcf* has a considerable percentage of L2 cache misses due to its intensive pointer chasing. These outstanding load instructions tend to exhaust LSQ entries. The right part of Fig. 7 indicates that tuning LSQ entries will obtain greater performance benefits than other design parameters. Similarly, the variable importance method can also apply to multi-core processors. We found that the number of cores is the most important parameters with an importance factor over 90 in average. This indicates that thread-level parallelism is the key factor for CMPs performance.

5.3. Partial dependence plot

Another advantage of the proposed model is that visualization and interpretation of fitted functions in a MART model can be achieved through partial dependence plots even though functions fitted by the MART models can be highly variable in shape and are frequently non-linear. The partial dependence plots can provide computer architects with visible interactions between different design parameters and performance trends and bottlenecks. From the plots, they can select configurations with optimized performance given a cost budget. As an example, we illustrate the two-dimensional partial dependence plot of the execution cycle of *mcf* to the two most important variables in Fig. 8. From this figure, we can see that a processor with a large LSQ size and cache block size tends to have high performance (the white region marked by "H"). On the other hand, the bottom left region marked by "L" indicates low performance configurations with a large execution cycle suffering from the small sizes of the LSQ and cache blocks. Moreover, we can see the tradeoffs between the design alternatives from this figure. For example, to reach a performance design goal which demands about 6.5e + 08 for the execution cycle of *mcf*, one can design a processor with the following alternatives for the LSQ and the cache block size: (1) the LSQ size equals to 26 and the cache block size is larger than 88 (part A of the line marked with "6.5e + 08"); (2) the LSQ size ranges from 26 to 46 and the cache block size equals to 88 (part B of the line); (3) the LSQ size equals to 46 but the cache block size ranges from 32 to 88 (part C of the line). With the help of this tool, computer architects can make judicious choices with other constraints from power, cost and complexity.

6. Related work

Architectural design space exploration has recently emerged to be an interesting problem in this community as increasingly large number parameters brought by advanced circuit integration technology. İpek et al. [9] predicted performance of memory subsystems, processors and CMPs via artificial neural networks (ANNs). They combine neural network and active learning method to efficiently explore large design space and predict unsampled points in the design space. Lee et al. [14] proposed regression models for performance and power prediction. They considered prediction based on both linear and nonlinear regression models as well as model inference such as significance testing for each design parameter and assessing goodness of fit. They also used regression models in Pareto frontier analysis, pipeline depth analysis and multiprocessor heterogeneity study [15]. Joseph et al. [10] developed linear regression models that characterized the interactions between processor performance and microarchitectural parame-

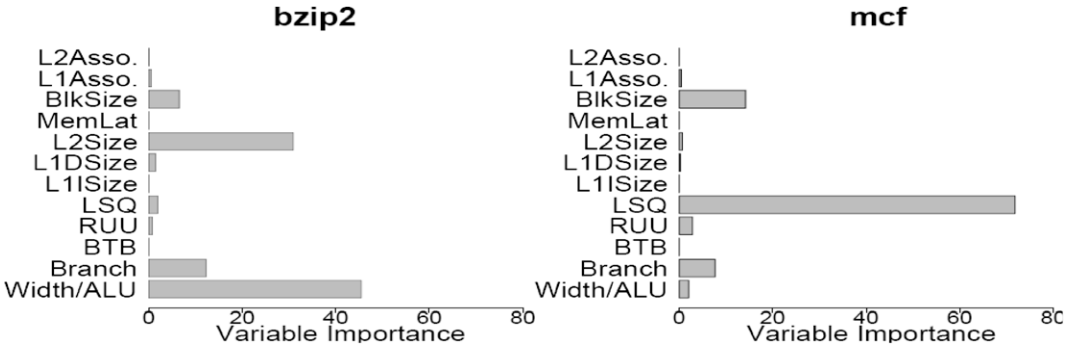

Fig. 7. Variable Importance of *bzip2* and *mcf*.

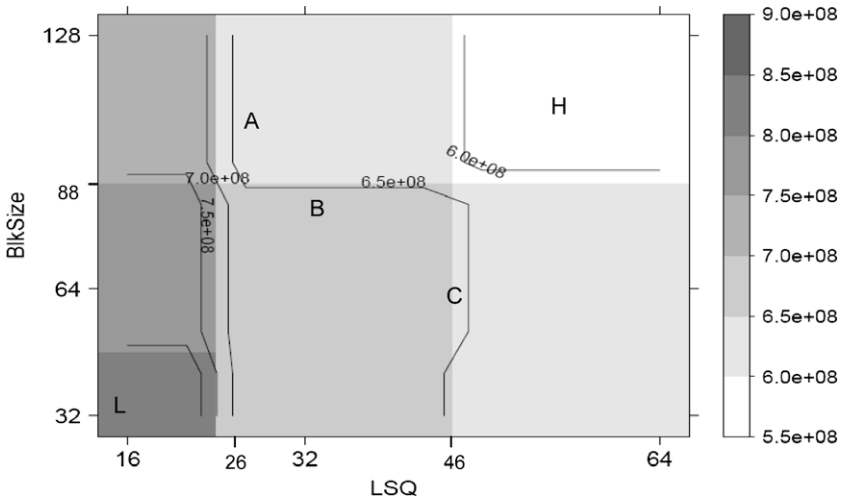

Fig. 8. The two-dimensional partial dependence plot of the execution cycle of *mcf* to the most important variables: "LSQ size" and "cache block size".

ters. They built the models by using Akaike's Information Criteria (AIC) directed iterative processes by which significance of the parameters to the CPI performance were ordered. They further proposed a non-linear regression model based on Radial Basis Function (RBF) networks for a superscalar processor performance prediction [11]. The RBF centers are chosen from the centers of the hyper-rectangles fitted from the regression trees. The working procedure requires multiple stages including building regression trees and RBF networks fitting, while MART fits the regression trees iteratively. Ould-Ahmed-Vall et al. [16] used a model tree to analyze performance of a subset of SPEC2006 running on an Intel Core 2 Duo processor.

Compared with the classical linear regression model, ANNs and RBF-networksour proposed method has the following features: (1) our proposed method is particularly well suited for the discrete (either ordinal or nominal variables) design space parameter; (2) MART achieves extremely accurate prediction which is supported by both lots of empirical evidence and theoretical proofs; (3) our method is highly robust to the tuning parameter values (need minimal knowledge to tune the model); (4) it also comes with model interpretation tools such as the measure of variable importance and the partial dependence plot, which provide computer architects a quantitative view for design alternatives and may shed light on the underlying mechanism. This paper extends the original conference version which appeared in Ref. [13] by more results in Sections 3–5.

7. Conclusion

In this paper, we propose a MART model which exploits microarchitectural design space and predicts performance of a single-core processor and a CMP. This model samples up to 0.02% of the full design space for a single-core processor with about 15 million points but achieves a very high accuracy. The median percentage error rate, based on an independent 5000 test points, ranges from 0.32% to 3.12% in 12 SPEC CPU2000 benchmarks. For a CMP design space with about 9.7 million points, the median percentage error is limited to a range from 0.50% to 1.89%. These results show that our model has highly compatible prediction performance to recently proposed regression and neural network models. The comparison of worst-case prediction also shows that our model has stronger robustness than both the linear regression ($R + L$) and the MART with random sampling ($R + M$) approaches. In addition, our model reflects performance trends and bottlenecks by showing the importance and partial dependence of processor design parameters.

References

[1] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Wadsworth, 1984.
[2] D. Burger, T. Austin, The Simple Scalar Tool Set, Version 2.0, Technical Report #1342, CS Department, University of Wisconsin–Madison, June 1997.
[3] P. Domingos, A unified bias-variance decomposition and its applications, in: Proceedings of the Seventeenth International Conference on Machine Learning, 2000, pp. 231–238.
[4] B. Efron, The Jackknife, the Bootstrap, and Other Resampling Plans, Society for Industrial and Applied Mathematics, Philadelphia, Penn., 1982.
[5] Y. Freund, H.S. Seung, E. Shamir, N. Tishby, Information, prediction, and query by committee, in: Proceedings of the Advances in Neural Information Processing Systems, 1993, pp. 483–490.
[6] J. Friedman, Greedy function approximation: a gradient boosting machine, The Annals of Statistics 29 (2001) 1189–1232.
[7] J. Friedman, Stochastic gradient boosting, Computational Statistics and Data Analysis 38 (2002) 367–378.
[8] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer, NY, 2001.
[9] E. İpek, S.A. McKee, B.R. Supinski, M. Schulz, R. Caruana, Efficiently exploring architectural design spaces via predictive modeling, in: Proceedings of the Twelfth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII), San Jose, CA, October 2006.
[10] P. Joseph, K. Vaswani, M. Thazhuthaveetil, Use of linear regression models for processor performance analysis, in: Proceedings of the Twelfth IEEE Symposium on High Performance Computer Architecture (HPCA-12), February 2006, pp. 99–108.
[11] P. Joseph, K. Vaswani, M. Thazhuthaveetil, A predictive performance model for superscalar processors, in: Proceedings of the International Symposium on Microarchitecture, December 2006.
[12] M. Kearns, L.G. Valiant, Cryptographic limitations on learning Boolean formulae and finite automata, Journal of the Association for Computing Machinery 41 (1994) 67–95.
[13] B. Li, L. Peng, B. Ramadass, Efficient MART-aided modeling for microarchitecture design space exploration and performance prediction, in: Proceedings of the 2008 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Annapolis, MD, June 2008.
[14] B. Lee, D. Brooks, Accurate and efficient regression modeling for microarchitectural performance and power prediction, in: Twelfth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII), San Jose, CA, October 2006.
[15] B. Lee, D. Brooks, Illustrative design space studies with microarchitectural regression models, in: Proceedings of the Thirteenth International Symposium on High Performance Computer Architecture (HPCA-13), February 2007.
[16] E. Ould-Ahmed-Vall, J. Woodlee, C. Yount, K.A. Doshi, S. Abraham, Using model trees for computer architecture performance analysis of software applications, in: Proceedings of the 2007 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), April 2007.
[17] G. Ridgeway, Generalized Boosted Models: A Guide to the gbm Package, <http://cran.r-project.org/web/packages/gbm/vignettes/gbm.pdf>, 2007.
[18] M. Saar-Tsechansky, F. Provost, Active learning for class probability estimation and ranking, in: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, August 2001, pp. 911–920.

[19] S. Sair, M. Charney, Memory Behavior of the SPEC2000 Benchmark Suite, Technical Report, IBM Corp., October 2000.
[20] R. Schapire, The boosting approach to machine learning – an overview, in: D.D. Denison, M.H. Hansen, C. Holmes, B. Mallick, B. Yu (Eds.), MSRI Workshop on Nonlinear Estimation and Classification, Springer, NY, 2002.
[21] SESC, <http://sesc.sourceforge.net/>.
[22] K. Sung, P. Niyogi, Active learning for function approximation, in: Proceedings of the Advances in Neural Information Processing Systems, vol. 7, 1995, pp. 593–600.
[23] L.G. Valiant, A theory of the learnable, Communications of the ACM 27 (1984) 1134–1142.

Bin Li received his Bachelor degree in Biophysics from Fudan University, China. He obtained his Master degree in Biometrics (08/2002) and Ph.D. degree in Statistics (08/2006) from The Ohio State University. He joined the Experimental Statistics department at Louisiana State University as an Assistant Professor in August, 2006. His research interests include statistical learning and data mining, statistical modeling on massive and complex data, and Bayesian statistics. He received the Ransom Marian Whitney Research Award in 2006 and a Student Paper Competition Award from ASA on Bayesian Statistical Science in 2005. He is a member of the Institute of Mathematical Statistics (IMS) and American Statistical Association (ASA).

Lu Peng received his Bachelor and Master degrees in Computer Science and Engineering from Shanghai Jiaotong University, China. He obtained his Ph.D. degree in Computer Engineering from the University of Florida in Gainesville in April 2005. He joined the Electrical and Computer Engineering department at Louisiana State University as an Assistant Professor in August, 2005. His research focus on memory hierarchy system, reliability, power efficiency and other issues in CPU design. He also has interests in Network Processor. He received an ORAU Ralph E. Powe Junior Faculty Enhancement Awards in 2007 and a Best Paper Award from IEEE International Conference on Computer Design in 2001. Dr. Peng is a member of the ACM and the IEEE Computer Society.

Balachandran Ramadass received his Bachelor's degree in electronics and communication engineering from Pondicherry University, India and his Master of Science degree in electrical and computer engineering in Louisiana State University. His research focuses are CMP architecture, on-chip power optimization, interconnection network, heterogeneous CMP architecture.