Efficient Design Space Exploration via Statistical Sampling and AdaBoost Learning

Dandan Li[*], Shuzhen Yao[*], Yu-Hang Liu[†], Senzhang Wang[*] and Xian-He Sun[†]
[*] School of Computer Science and Engineering, Beihang University, Beijing,China
[†] Department of Computer Science, Illinois Institute of Technology, Chicago, USA
lidandan04@163.com, {szyao, szwang}@buaa.edu.cn, {yliu242, sun}@iit.edu

ABSTRACT

Design space exploration (DSE) has become a notoriously difficult problem due to the exponentially increasing size of design space of microprocessors and time-consuming simulations. To address this issue, machine learning techniques have been widely employed to build predictive models. However, most previous approaches randomly sample the training set leading to considerable simulation cost and low prediction accuracy. In this paper, we propose an efficient and precise DSE methodology by combining statistical sampling and Adaboost learning technique. The proposed method includes three phases. (1) Firstly, orthogonal design based feature selection is employed to prune design space. (2) Sencondly, an orthogonal array based training data sampling method is introduced to select the representative configurations for simulation. (3) Finally, a new active learning approach ActBoost is proposed to build predictive model. Evaluations demonstrate that the proposed framework is more efficient and precise than state-of-art DSE techniques.

1. INTRODUCTION

In order to meet the increasing performance requirements at the cost of given budget, modern microprocessors have been adopting more and more complex architectures. When designing a new microprocessor, one major challenge is to find the promising architectural configurations in design space to satisfy different performance, power, temperature, and cost constraints. Generally, computer architects use cycle-accurate simulators to run benchmarks to evaluate different design architecture parameter combinations. The software simulator is often several orders of magnitude slower than the execution on real hardware. Therefore, full-search exploration of a wide range of architecture parameters is infeasible. In the past decade, in order to address the architectural design space exploration (DSE), previous works mainly focus on two directions. One direction is to investigate fast simulation techniques, e.g., Simpoint [15] and SMARTS [16]. However, the reduced time based on speeding up simulation is quite limited, since the design space is still extremely huge. The other direction is to use machine learning techniques to build regression models with a set of simulations to predict the unknown architectural responses, which can usually achieve desirable results.

However, as the simulation is time-consuming and the design space is extremely large, it is essential for DSE to sample a small training set for predictive model to learn the relationships between parameters and responses accurately. Most traditional approaches randomly sample a training data set from the entire design space to build regression models [7, 8, 9, 10, 11]. The problem of such method is that the sampled training data cannot fully represent the distribution of the entire sample space. Although Ïpek et al. [1] and Guo et al. [3,4] separately proposed using active learning and semi-supervised learning to improve the accuracy of predictive model, their initial training set are randomly sampled and accounted for a large proportion of the entire training data (e.g., random samples account for 75% of entire training set in [3]). Since the above approaches ignored the training set sampling strategy, they suffered from either higher simulation costs or lower prediction accuracy.

The focus of this paper is on designing an efficient sampling approach to sample more representative and informative training set. At the meantime, an effective and robust predictive model is also necessary to improve the prediction accuracy with low simulation costs. Therefore, a novel DSE framework is proposed in this paper. The proposed framework consists of three phases: design space pruning, orthogonal array based training data sampling and active learning-based AdaBoost.

1) In the first phase, we propose to apply a statistical method orthogonal design to efficiently determine the importance of each parameter. The orthogonality of orthogonal design and corresponding analysis of variance (ANOVA) enable us to calculate the contribution of each structural parameter. Although [2] proposed to utilize Plackett and Burman (PB) design to identify the key parameters, PB design is an experiment design with two levels. Orthogonal design is an experiment design of multiple parameters with multiple levels.

2) We next propose to apply orthogonal array (OA) to sample configurations as the initial training data for simulation. Compared with random sampling, the orthogonality of OA guarantees that the selected samples can better represent the distribution of the design space.

3) Finally, with the sampled initial training data, a novel Active Learning-based AdaBoost model (ActBoost) is proposed to improve the accuracy of the predictive model. Disagreement-based active learning is proposed to selectively sample informative samples for reducing simulation cost.

To summarize, this paper makes the following contributions:

- To our knowledge, this is the first work that employs orthogonal design to prune the design space and sample training set. For design space pruning, the rationale is that the architectural parameters with little effect on responses can be explored in a coarse-grained manner.
- A new learning-based method ActBoost is proposed to improve the prediction accuracy with a small training set.
- Experimental results demonstrate that the proposed framework is efficient and precise. Compared with random sampling, OA sampling could achieve more accurate prediction. Compared with three state-of-art DSE techniques, ActBoost

DAC '16, June 05-09, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4236-0/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2897937.2898012

significantly outperforms them on prediction accuracy while fixing the sample size. Besides, ActBoost could achieve comparable prediction accuracy with fewer samples, thus significantly reduce the simulation time.

The remainder of this paper is organized as follows. The next section presents the details of our approach. Section 3 introduces the experimental setup. Section 4 shows the evaluation results. We review related work in Section 5 and conclude this paper in Section 6.

2. ORTHOGONAL DESIGN AND ACTBOOST FOR DSE

In this section we introduce how to combine statistical sampling strategy and machine learning techniques to improve the accuracy of design space model with lower simulation cost. Figure 1 presents the framework of the proposed three-phase DSE, and the steps highlighted with deep color are our contributions. In the design space pruning phase, OA is applied to sample configurations for simulation and calculate the contribution of each parameter on performance. The parameters with little contribution are pruned to narrow the design space. In the initial training data sampling phase, OA sampling is also applied to select configurations in the reduced design space. The simulation results are used as the initial training set for the predictive model. In the final phase, we train ActBoost model to predict the performance of configurations not simulated.

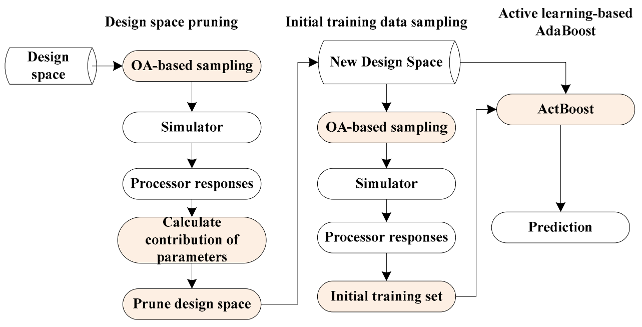

Figure 1. Three-phase DSE framework

To elaborate our proposal, we next give a brief introduction on orthogonal design.

2.1 Orthogonal Design

Although orthogonal design is widely used in statistical domain, this paper makes the first attempt to apply it for architectural design space exploration. Instead of testing all possible parameter combinations like the factorial design, orthogonal design conducts a minimum amount of representative experiments according to OA to investigate which parameters most significantly affect responses [12]. As the selected representative experiment number is usually very small, time and resources cost can be significantly reduced. OA follows the balance principle with the following two properties: (1) each level (value) of each parameter occurs the same number of times in each column, and (2) each possible level combination of any two given parameters occurs the same times in the array. Thus, OA has the characteristics of orthogonality and representativeness. The OA can be derived from deterministic algorithms [12] or looked up online. The arrays are selected by the number of parameters and the number of levels (values). For example, a design space of the target architecture considering 5 parameters, one with 4 levels and four with 2 levels, would require 64 $(4^1 \times 2^4)$ different simulations to test the effect of all parameters. However, for OA only 8 simulations are sufficient to calculate the effect of each parameter and it can be denoted by $L_8(4^1 \times 2^4)$, as shown in Table 1. Each row in this array represents a combination of levels, that is, a simulation. For example, the last row stands for a simulation in which parameter A is at level 4, parameter B at 2, parameter C and D at 1, and parameter E at 2.

Generally, we can easily find the exact appropriate orthogonal array online. When the corresponding orthogonal array with the same number of parameters and values in design space cannot be obtained online exactly, we make some accommodation and the highest value will be taken or the difference will be split. For example, the levels of 5 parameters are 3, 2, 2, 2, 2, but we could still select the OA $L_8(4^1 \times 2^4)$ and the appropriate strategy is to fill in the two entries for A=4 with 1, 2 or 3 randomly.

Table 1. Orthogonal array: $L_8(4^1 \times 2^4)$

No.	A	B	C	D	E
1	1	1	1	1	1
2	1	2	2	2	2
3	2	1	1	2	2
4	2	2	2	1	1
5	3	1	2	1	2
6	3	2	1	2	1
7	**4**	1	2	2	1
8	**4**	2	1	1	2

Analysis of variance (ANOVA) on the collected data from the orthogonal design of experiments can be used to calculate the effect of each parameter on performance characteristic. Thus we can further determine the levels and range of parameters, including the minimum, maximum values of parameters and the number of values. If the contribution of a parameter is large, its values can be further split for testing more values. Otherwise, we shrink the range of parameters and test fewer values.

2.2 Design Space Pruning

The intuition of this phase is that the configuration parameters with little effect on microprocessor performance could be fixed to a constant value or their number of values can be reduced. Specifically, it includes four steps as follows.

1) We first select the parameters and their value ranges according to the most current superscalar implementations.

2) Then we construct OA to sample design configurations, and perform corresponding simulations.

3) The contribution percentage of each parameter is calculated by statistical ANOVA technique that allows one to separate the effect of a single parameter on the response. The contribution percentage of parameter X can be calculated by $\rho_X = SS_X / SS_T$, where SS_X is sum-of-square of parameter X, and SS_T is the total sum-of-square of all the parameters. The contribution percentage gives an indication of the relative importance of each parameter.

4) The parameters are ranked according to their contributions. The parameters with small contribution can be fixed to a constant value or their levels can be reduced, thus the design space is pruned. For instance, suppose there are two parameters A and B with contribution percentage 20% and 2% respectively, and they both have 5 possible values. As B has a much lower impact than A on the performance, we could reduce the number of values of B.

2.3 Initial Training Data Sampling

In general, regression model can achieve promising generalization performance when the training and testing data follow the same distribution and sufficient training data are available. Because of the representative property of orthogonal array, the dataset sampled via orthogonal array can evenly cover the entire design space. In addition, different from PB design with only two values for each parameter, OA permits the parameters to have arbitrary values. Thus, we propose OA-based sampling method (OA sampling for short) to determine initial training set. Cycle-accurate simulations on these architectural configurations selected via OA sampling are conducted. The simulations and corresponding configurations constitute the initial training dataset for the predictive model.

Besides orthogonal design, two other popular designs of experiment methods are factorial design and random design. Factorial design gets increasingly complex with an increase in the number of parameters, which is obviously infeasible to sample all the configurations in the design space. Compared with random sampling which may lead to uncertainty, OA sampling could uniformly sample the representative architectural configurations that equally distributes in the entire design space. Therefore, the initial training dataset for predictive model via OA sampling can better capture the relationships between parameters and responses than random sampling.

2.4 ActBoost: Active Learning-based AdaBoost

Our goal is to train a more accurate predictive model with lower simulation cost. To improve the robustness and accuracy of the model, we employ an AdaBoost algorithm, AdaBoost.RT [14] with artificial neural network (ANN) as the weak learner to build the predictive model. Although the initial training dataset based on OA sampling is representative, usually it is too small to train an accurate predictive model. Thus the initial training set has to be enlarged to improve prediction, and the configurations included in training set (for simulation) should be informative for the generalization ability of predictive model. Thus, ActBoost is proposed, which trains two AdaBoost.RT models with active learning sampling strategy and output the averaged results of the two models as the final prediction.

AdaBoost.RT: It iteratively trains several base predictors $h_1, \cdots, h_T$, and provides a linear combination of these base predictors as the final predicting model H. Let $L = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2) \cdots (\mathbf{x_m}, y_m)\}$ be the training set, where $\mathbf{x}_i$ is the *i*-th design configuration and y_i is corresponding processor response. In the beginning, the base predictor is supplied with training set of *m* sample with uniform distribution $D_1(i) = 1/m$. The performance of the base predictor is evaluated by computing the prediction error. The error rate of the ANN predictor ε_t is computed using a pre-set threshold ϕ, which is used to demarcate prediction error as correct or incorrect. If the absolute relative error ($ARE_t(i)$) for any particular configuration is greater than ϕ, the predicted response for this configuration is considered to be incorrect, otherwise it is correct. As shown in Equation 1, the numbers of incorrect predictions are counted to calculate ε_t :

$$\varepsilon_t = \sum_{i: ARE_t(i) > \phi} D_t(i) \quad , \quad ARE_t(i) = \left| \frac{h_t(\mathbf{x_i}) - y_i}{y_i} \right| \qquad (1)$$

We set $\beta_t = {\varepsilon_t}^n$, where n is power coefficient (e.g. linear, square or cubic). The weight updating parameter (D_t for the *t*-th iteration) is computed to give larger weight to the examples whose error rate is large. As shown in Equation 2, at the end of the *i*-th iteration, AdaBoost.RT updates the distribution $D_{t+1}(i)$ with respect to each sample:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & if\ ARE_t(i) \le \phi \\ 1 & otherwise \end{cases} \qquad (2)$$

where Z_t is a normalization factor. After T iterations, the final regression model H shown in Equation 3 is ready to assign any input configuration $\mathbf{x}$ a response value,

$$H(\mathbf{x}) = \sum_{t=1}^{T} \left\{ \left(\log \frac{1}{\beta_t} \right) h_t(\mathbf{x}) \Big/ \sum_t \left(\log \frac{1}{\beta_t} \right) \right\} \qquad (3)$$

Algorithm 1: Active learning-based AdaBoost for DSE

```
Input: $L$ : set of labeled samples, $U$ : set of unlabeled samples
Output: Prediction of performance on configuration $\mathbf{x}$ not simulated
Create a pool $P$ by moving $p$ unlabeled samples from $U$ to $P$ randomly
  Use $L$ to train Adaboost.RT $H_1$ with $M_1$ hidden neurons
  Use $L$ to train Adaboost.RT $H_2$ with $M_2$ hidden neurons
  Run for $K$ iterations:
    1. Both $H_1$ and $H_2$ predict the $P$
       Obtain prediction results $h_j(\mathbf{x}_i)$ for each unlabeled sample $\mathbf{x}_i \in P$
    2. //Calculate $c.v$ of each unlabeled sample in $P$ and sort them
       $c.v(\mathbf{x}_i) = \frac{\sigma_i}{\mu_i}$ , $\sigma_i = \sqrt{\frac{1}{2T}\sum_{j=1}^{2T}(h_j(\mathbf{x}_i) - \mu_i)^2}$ , $\mu_i = \frac{1}{2T}\sum_{j=1}^{2T} h_j(\mathbf{x}_i)$
    3.Choose $N$ samples $\mathbf{x}_1^{'}, \mathbf{x}_2^{'}, \cdots, \mathbf{x}_N^{'}$ randomly from top $W$ ambiguous
      ones
    4. Simulate the $N$ samples to obtain corresponding responses as
       $y_1, y_2, \cdots, y_N$
    5. //Move the newly labeled samples from $P$ to L
       $P \leftarrow P - \{(\mathbf{x}_1^{'}, y_1), (\mathbf{x}_2^{'}, y_2), \cdots, (\mathbf{x}_N^{'}, y_N)\}$
       $L \leftarrow L \cup \{(\mathbf{x}_1^{'}, y_1), (\mathbf{x}_2^{'}, y_2), \cdots, (\mathbf{x}_N^{'}, y_N)\}$
    6. Rebuild $H_1$ , $H_2$ by new set $L$
    7. Replenish the $P$ by choosing $N$ examples from $U$ at random
  Output: $H^*(\mathbf{x}) \leftarrow \frac{1}{2}(H_1(\mathbf{x}) + H_2(\mathbf{x}))$
```

ActBoost: This algorithm aims to provide promising generalization performance by incrementally enlarging the labeled data set and boosting the learners. At each iteration of the training process, each learner estimates the responses of a pool of unlabeled samples. Algorithm 1 presents the pseudo-code of ActBoost algorithm. At the beginning, two diverse AdaBoost.RT models (say, H_1 and H_2) with distinct inductive biases are initialized by the initial training set. The ANNs in the two models have the same number of layers but different number of hidden neurons. Let $h_1, h_2, \cdots, h_T$ be the ANNs of H_1 and $h_{T+1}, h_{T+2}, \cdots, h_{2T}$ be the ANNs of H_2 . We employ active learning based on disagreement to sample unlabeled samples. Since the predictions of two AdaBoost.RTs' ANNs are different on test set, high disagreement indicates that including the points in the training set can lower model variance as well as prediction error. The algorithm randomly selects several samples on which the multiple learners have the largest disagreement. The disagreement of each sample can be measured by calculating the coefficient of variation ($c.v$) of the predictions of all the base ANN learners. The $c.v$ value of the *i*-th unlabeled configuration $\mathbf{x}_i$ is calculated by Equation 4,

$$c.v(\mathbf{x}_i) = \frac{\sigma_i}{\mu_i} \ , \ \sigma_i = \sqrt{\frac{1}{2T}\sum_{j=1}^{2T}(h_j(\mathbf{x}_i) - \mu_i)^2} \ , \ \mu_i = \frac{1}{2T}\sum_{j=1}^{2T} h_j(\mathbf{x}_i) \qquad (4)$$

where μ_i is the mean of predictions, σ_i is standard deviation, $2T$ is the number of all ANNs of two AdaBoost.RT models, and h_j is the j-th ANN. The unlabeled configurations are sorted according to $c.v$. In order to roughly keep the balance of data distribution, the selected configurations are randomly chosen from a small group of the most ambiguous points instead of being chosen directly from the most ambiguous ones. Then, the selected configurations are simulated and moved from the unlabeled set to the labeled set. After using the latest labeled set to rebuild two AdaBoost model, the pool P should be replenished with N unlabeled samples for next iteration. Finally, we average the predictions on each rebuilt AdaBoost models as the final prediction.

As the sampling approach depends on the disagreement between two AdaBoost models, the individual model helps each other in the learning process. And each selected sample will significantly boost the learners of the two models.

3. EXPERIMENTAL SETUP

3.1 Simulator and Benchmarks

Using the state-of-art cycle-accurate simulator GEM5 [13], we conduct our simulations on 12 representative benchmarks from the SPEC CPU2006 suite. Aided by SimPoint [15], 100 million simulated instructions for each benchmark are simulated to collect statistics. IPC (Instruction-per-Cycle) is used as the performance metric for processor response.

3.2 The Explored Design Space

At the beginning, we choose 15 parameters with two levels, the lowest and highest values. It only needs 16 simulations according to OA sampling ($L_{16}(2^{15})$). The contribution of each parameter on performance is different on different programs. We sample 16 configurations for each benchmark for simulation. Then we calculate the contribution percentage of each parameter on IPC via ANOVA and sort the parameters according to the values of contribution percentage. Due to the page limit, only five benchmarks are shown in Table 2, the sum ranks of each parameter are calculated based on all the 12 benchmarks.

Table 2. The ranking of parameters based on their contribution percentage (b1: gromacs b2: games b3:gcc b4:bwaves b5:cactusADM)

Parameter	b1	b2	b3	b4	b5	…	Sum
ROB Size	1	1	8	2	3	…	27
Load/ Store Queue Size	3	2	5	4	2	…	49
Issue /Fetch/ Commit Width	2	9	2	6	8	…	53
Register File	4	7	6	3	5	…	62
Instruction Queue Size	5	4	1	1	6	…	65
L1 Icache	7	6	9	7	7	…	71
L2 Ucache	6	5	10	5	4	…	73
L1 DCache	8	3	4	10	9	…	81
L1 D MSHR	9	8	7	11	1	…	82
ALUs	10	10	13	9	11	…	114
FPUs	11	11	14	8	10	…	129
Cacheline size	12	13	3	13	12	…	145
L1 Dcache hit latency	14	14	11	14	13	…	161
L2 hit latency	13	12	15	15	14	…	163
BTB Entries	15	15	12	12	15	…	165

Table 2 illustrates that *Cacheline size*, *L1* and *L2 hit latency* and *BTB Entries* have less impact on IPC, therefore, they can be set constant values and the design space is pruned to 11 parameters. The number of levels of each parameter is determined by its ranking. The parameters sorted in front can be set more levels. As shown in Table 3, *ROB size*, *Load/Store Queue Size*, *Register File* and *Instruction Queue Size* are set 8 levels, while *Issue /Fetch/Commit Width*, *Cache Size* and other parameters are set fewer levels. However, the design space still contains more than 8.3 million design configurations.

As architecture with ROB size less than instruction queue size is impossible to achieve optimal performance, we reduce the design space by 1.3 million options. Then from the remained 7 (8.3-1.3) million options, the initial training set are determined by OA sampling. The corresponding OA is $L_{64}(8^4 \times 4^4 \times 2^3)$ including 4 parameters with 8 levels, 4 parameters with 4 levels and 3 parameters with 2 levels. Thus, 64 configurations are cycle-accurate simulated as the initial training set of our proposal. We also sample 1000 design configurations uniformly at random (UAR) as presented by [7] as the test set in all following experiments, unless otherwise specified.

Table 3. Microprocessor design space

Variable Parameters	Values
ROB Size	64-176 , 16+
Load/ Store Queue Size	16-128, 16+
Issue /Fetch/ Commit Width	2, 4, 6, 8
Register File	80-136, 8+
Instruction Queue Size	16-128, 16+
L1 Icache	16, 32, 64, 128kB
L2 Ucache	512kB-4MB, 2*
L1 DCache	16, 32, 64, 128kB
L1 Dcache MSHR	2, 4
ALUs	2, 4
FPUs	2, 4
Total	**8,388, 608 Combinations**

According to the proposed three-phase DSE, after design space pruning, we demonstrate the effectiveness of OA sampling in initial sampling phase and ActBoost in active learning phase. For OA sampling, we compare the prediction accuracy of two predictive models based on OA sampling with random sampling respectively. Finally we compare ActBoost against three state-of-the-art predictive models [1, 4]. In the paper, we employ the Mean Squared Error (MSE) to evaluate the prediction performance.

4. EVALUATION

4.1 Performance Evaluation of OA Sampling

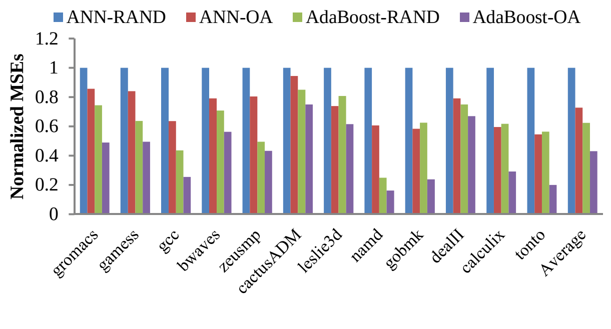

Figure 2. MSEs of AdaBoost and ANN models with random sampling vs. OA sampling (normalized to the MSE of ANN model with random sampling)

We first use OA to sample initial training set, and their simulations are used as the training set to train a relative weak predictive model. In order to avoid fortuity of random sampling, we conduct 10 times of randomly sampling 64 configurations to build 10 weak predictive models, and calculate the mean MSE of the 10 predictive models.

We carry out experiments to evaluate AdaBoost.RT and ANN with two sampling methods separately: random-sampling and OA sampling. As shown in Figure 2, for both methods OA sampling significantly outperforms random sampling on all the 12 benchmarks with the same training set size. On average, compared with random sampling the initial training data, OA sampling reduces prediction error by 32% and 27% of

AdaBoost.RT and ANN, respectively. Therefore, OA sampling is very efficient for sampling the initial training data. In addition, we can also see that AdaBoost.RT outperforms ANN. Compared with ANN, on average, AdaBoost.RT reduces the MSE by 38% and 41% with random sampling and OA sampling, respectively.

In addition, OA could be easily constructed for design space which spans from 1 to 50 parameters with various levels. Therefore, OA sampling is still effective and scalable for the diversified micro architecture DSE in reality.

4.2 Performance Evaluation of ActBoost

In our experiments, the initial training set of ActBoost consists of 64 sampled design configurations. The number of computation iterations K is set to 50, the number of samples to be labeled N is set to 4, and the number of candidate samples W is set to 16. It implies that 200 extra unlabeled design configurations will be labeled by cycle-accurate processor simulations, and the pool size holding evaluated unlabeled samples is set to 100. To obtain two diverse AdaBoost.RT models, hidden neurons of ANN in two AdaBoost.RT models are set to 8-unit and 6-unit with two layers respectively. The rest settings of ANN are the same as that in [1]. Besides, we randomly choose 3000 configurations as unlabeled set. After 50 active learning iterations, the training data are enlarged to 264 samples.

To study the effectiveness of ActBoost, we compare the prediction accuracy of our approach with three state-of-the-art predictive models: random-sampling-based ANN DSE approach (ANN-RS), intelligent-sampling-based ANN DSE approach (ANN-IS) [1], and co-training regression tree with active learning (COAL) approach [4]. In addition, to evaluate the efficiency of the proposed active learning approach, we also conduct experiments of random-sampling-based AdaBoost.RT model. ANN-IS shares the same initial training set with COAL that consists of 64 configurations sampled by random, and both AdaBoost.RT and ActBoost share the OA-sampling-based initial training set.

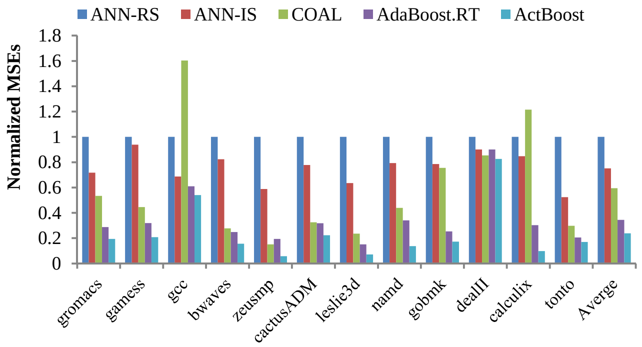

Figure 3. Normalized MSEs of different methods (normalized to the MSE of ANN-RS)

Figure 3 shows the normalized MSEs of different approaches. ActBoost outperforms other approaches over all 12 benchmarks. In average, ActBoost reduces the MSEs by 76% of ANN-RS. Most notably, ActBoost reduces the MSEs by 94% of ANN-RS on the benchmark *zeusmp*. Even on the benchmark with the least MSE reduction, that is, *gcc*, the MSE reduction can still achieve 50%. Comparing with ANN-IS, ActBoost reduces the MSE by 68% on average. Compared with COAL, ActBoost reduces the MSE by 60% on average. Therefore, ActBoost is more practical than state-of-the-art DSE techniques due to its higher accuracy. Moreover, compared to random-sample-based AdaBoost.RT model, ActBoost reduces the MSE by 31% on average. Thus we can conclude that the selective sampling via active learning based on two Boost models in ActBoost can significantly improve the prediction accuracy.

4.3 Sensitivity to Training Sample Size

The training sample size is fixed in the aforementioned experiments. In this subsection, we further study the prediction accuracy and training time of different methods with different training sample size. We increase training samples from 24 to 264 for fairness. The initial 24 samples of ActBoost are randomly selected from 64 OA sampled configurations. ANN-IS still shares the same initial training set with COAL that sampled by random.

On all the benchmarks, the experimental results reveal similar performance for different methods. Due to the page limitation, we take benchmark *leslie3d* as an illustrative example.

4.3.1 Prediction accuracy

Figure 4 represents the accuracy comparison of our method and other three methods with the increase of training sample size on the benchmark *leslie3d*. We can see that ActBoost requires fewer training samples than three other methods to achieve comparable prediction accuracy. When increasing the training samples number to 84, the prediction accuracy of COAL is close to ActBoost, but after that its MSE decreases with a relative slow speed. On average, ANN-RS requires about 40% more simulated configurations to achieve comparable prediction accuracy with ActBoost. In summary, ActBoost can achieve desirable prediction performance with smaller training sample size, thus the simulation and training time can be significantly reduced.

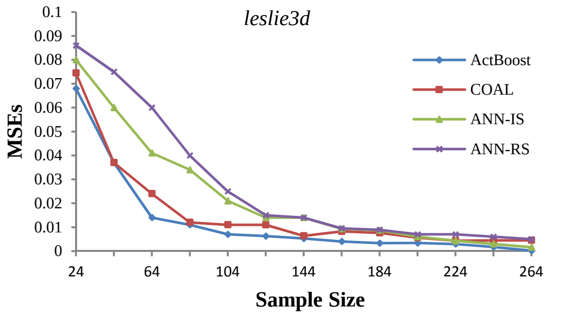

Figure 4. MSEs *vs.* training sample size for different methods

4.3.2 Training Time Cost

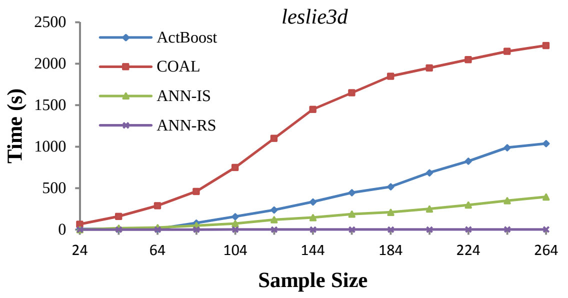

Figure 5. Training time vs. training sample size for different methods

We also study the training time costs with different sizes of training samples. Figure 5 illustrates that the training time almost linearly increases with respect to sample size for all the models. One can see that, the training time of ActBoost is only about 17 minutes (1038 seconds) for 264 samples, while COAL requires 37 minutes (2220 seconds). ANN-RS is the fastest which just requires less than 3 seconds. Although ANN-IS is faster than ActBoost, the more 11 minutes training time is worthy for high prediction accuracy and saving large amount of simulation time.

4.4 Optimal Configuration

Generally speaking, the actual optimal configuration cannot be obtained unless simulating the entire design space, which is infeasible. A compromise is using ActBoost to explore test dataset space as a case study to evaluate the effectiveness of ActBoost. We predict configurations of test set and find the optimal configuration, and then compare with the actual best simulation configuration of entire test set. Table 4 shows the optimal simulation configuration, the optimal predicted configurations and prediction error of ActBoost and other three state-of-art approaches for benchmark *leslie3d*. One can see that the optimal predicted configuration of ActBoost is the same with the optimal simulation configuration, and the prediction error is very low, only 0.5%. While the optimal predicted configurations of other approaches are different from the optimal simulation configuration. Although the actual IPC of the optimal predicted configuration of COAL is just slightly smaller than that of the real optimal simulation configuration (1.92 vs. 1.93), its prediction error is higher than ActBoost. The actual IPCs of optimal predicted configurations of ANN-IS and ANN-RS are much smaller, and prediction error is much higher.

Table 4. The optimal simulation configuration and predicted optimal configurations of different methods

Parameters	**Simulation**	**ActBoost**	**COAL**	**ANN-IS**	**ANN-RS**
ROB size	112	112	96	128	176
LSQ	64	64	96	112	128
Width	8	8	8	8	8
Register File	104	104	80	136	136
IQ	64	64	80	32	128
L1 Icache	128	128	128	32	128
L2 Ucache	512	512	4096	512	512
L1 DCache	128	128	32	16	128
MSHR	6	6	6	6	6
ALUs	4	4	4	2	2
FPUs	4	4	2	4	2
Actual	1.93	1.93	1.92	1.91	1.89
Predicted	---	1.92	1.8	2.04	2.16
Error	---	**0.5%**	6.3%	6.7%	14.3%

5. RELATED WORK

A number of analytic and statistical machine learning models based methodology have been applied to architectural DSE. Yi et al. [2] applied PB design to identify the key parameters and classify benchmarks for improving the simulation methodology. İpek et al. [1] proposed predictive model based on ANNs for DSE. Lee et al. [7] proposed spline-based regression model and composable performance regression model. Joshep et al. [5, 6] proposed to build linear regression models and Radial Basis Function (RBF) nonlinear regression model to predict the microprocessor performance. Khan et al. [9] and Dubach et al. [10] independently proposed to utilize prior knowledge to build cross-program ANN-based regression model to predict processor response of new applications across the entire architectural configurations. Guo et al. [3, 4] combined semi-supervised learning with active learning to build a predictive regression model. In another paper, Guo et al. [11] proposed a framework which built several distinct base regression models and then construct a metamodel to output the final prediction results.

However, since the above approaches ignored sampling strategy, they still suffer high simulation costs or low prediction accuracy. The key differences in our approach are twofold. Firstly, we employ rigorous statistic method to prune the design space and sample the initial training set. Secondly, we propose a new model ActBoost. These techniques help us achieve high prediction accuracy at low simulation cost.

6. CONCLUSIONS

This paper makes the first attempt to combine statistical method and machine learning technique for chip design space exploration. We proposed to employ statistical method to prune the design space and determine the training set. We also presented ActBoost which utilized two AdaBoost.RT models with active learning to build the predictive model. The efficient sampling strategy and accurate predictive model forms a concatenation optimization to achieve desirable efficiency and accuracy. Experimental results show that the proposed approach is efficient and precise, compared with state-of-the-art DSE approaches. Our approach thus enables efficient yet accurate design space exploration for modern processors.

7. ACKNOWLEDGMENTS

This work is supported in part by the Aeronautical Science Foundation of China under grant 2013ZC51023, the National Science Foundation under grant CNS-1162540 and grant CCF-0937877. The authors acknowledge financial support from China Scholarship Council.

8. REFERENCES

[1] E. İpek, S. McKee, B. de Supinski, M. Schulz, and R. Caruana, “Efficiently exploring architectural design spaces via predictive modeling,” in ASPLOS-12, 2006.

[2] J. Yi, D. J. Lilja, and D. M. Hawkins, “A statistically rigorous approach for improving simulation methodology,” in HPCA-9, 2003.

[3] Q. Guo, T. Chen, Y. Chen, etc., “Effective and efficient microprocessor design space exploration using unlabeled design configurations,” in IJCAI ’11, 2011.

[4] T. Chen, Y. Chen, Q. Guo, etc., “Effective and efficient microprocessor design space exploration using unlabeled design configurations,” ACM Transactions on Intelligent Systems Technology, vol. 5, no. 1, pp. 992–999, 2011.

[5] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, “Construction and use of linear regression models for processor performance analysis,” in HPCA-12, 2006.

[6] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, “A predictive performance model for superscalar processors,” in MICRO-39, 2006.

[7] B. C. Lee and D. Brooks, “Accurate and efficient regression modeling for microarchitectural performance and power prediction,” in ASPLOS-12, 2006.

[8] B. C. Lee, J. Collins, H. Wang, and D. Brooks, “CPR: Composable performance regression for scalable multiprocessor models,” in MICRO-41, 2008.

[9] S. Khan, P. Xekalakis, J. Cavazos, and M. Cintra, “Using predictive modeling for cross-program design space exploration in multicore systems,” PACT-16, 2007.

[10] C. Dubach, T. M. Jones, and M. F. P. O’Boyle, “Microarchitectural design space exploration using an architecture-centric approach,” MICRO-40, 2007.

[11] Q. Guo, T. Chen, Z. H. Zhou, etc., “Robust design space modeling,” ACM Transactions on Design Automation of Electronic Systems, vol. 20, no. 2, pp. 1–22, 2015.

[12] K.T. Fang, Y. Wang, Number-theoretic methods in statistics, Chapman and Hall, New York, 1994.

[13] N. Binkert et al., “The gem5 simulator.” ACM SIGARCH Computer Architecture News, vol. 39, no. 2, pp. 1–7, 2011.

[14] D. L. Shrestha and D. P. Solomatine, “Experiments with AdaBoost.RT, an improved boosting scheme for regression.” Neural Computation, vol. 18, no. 7, pp. 1678–1710, 2006.

[15] G. Hamerly, E. Perelman, and B. Calder, “How to use simpoint to pick simulation points,” ACM SIGMETRICS Performance Evaluation Review, vol. 31, no. 4, pp. 25-30, 2004.

[16] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, “SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling,” in ISCA-30, 2003.