Efficient Design Space Exploration by Knowledge Transfer

Dandan Li[1], Senzhang Wang[2,3], Shuzhen Yao[1], Yu-Hang Liu[4], Yuanqi Cheng[1] and Xian-He Sun[5]
[1] School of Computer Science and Engineering, Beihang University, Beijing, China
[2] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
[3] Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, China
[4] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[5] Department of Computer Science, Illinois Institute of Technology, Chicago, USA
{dandanli, szyao, chengyuanqi}@buaa.edu.cn, szwang@nuaa.edu.cn, liuyuhang@ict.ac.cn, sun@iit.edu

ABSTRACT

Due to the exponentially increasing size of design space of microprocessors and time-consuming simulations, predictive models have been widely employed in design space exploration (DSE). Traditional approaches mostly build a program-specific predictor that needs a large number of program-specific samples. Thus considerable simulation cost is required for each program. In this paper, we study the novel problem of transferring knowledge from the labeled samples of previous programs to help predict the responses of the new target program whose labeled samples are very sparse. Inspired by the recent advances of transfer learning, we propose a transfer learning based DSE framework TrDSE to build a more efficient and effective predictive model for the target program with only a few simulations by borrowing knowledge from previous programs. Specifically, TrDSE includes two phases: 1) clustering the programs based on the proposed orthogonal array sampling and the distribution related features, and 2) with the guidance of clustering results, predicting the responses of configurations in design space of the target program by a transfer learning based regression algorithm. We evaluate the proposed TrDSE on the benchmarks of SPEC CPU 2006 suite. The results demonstrate that the proposed framework is more efficient and effective than state-of-art DSE techniques.

Categories and Subject Descriptors

B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; I.2.6 [**Artificial Intelligence**]: Learning.

Keywords

Design space exploration; Processor design; Knowledge transfer

1. INTRODUCTION

Ever-increasing advances of integrated circuit and parallelism techniques make the size of processor design space growing exponentially. Finding the promising architectural configurations in design space to meet the performance, power and cost requirements is called design space exploration (DSE). Cycle-accurate simulators are generally used to evaluate different processor design parameter combinations. However, the software simulator is time-consuming and several orders of magnitude slower than the execution on real hardware. Therefore, it is infeasible to completely evaluate the large design space. In order to reduce the simulation time and cost of DSE, previous works mainly focus on two directions. One direction is to investigate fast simulation techniques, e.g., Simpoint [18] and SMARTS [19]. Although the time of per simulation is reduced, the size of design space is still extremely huge. The other direction is to use machine learning techniques to build predictive models with a set of simulation results of some configurations to predict the unknown responses of new architectural configurations. The combination of fast simulation techniques and machine learning techniques could usually achieve desirable results.

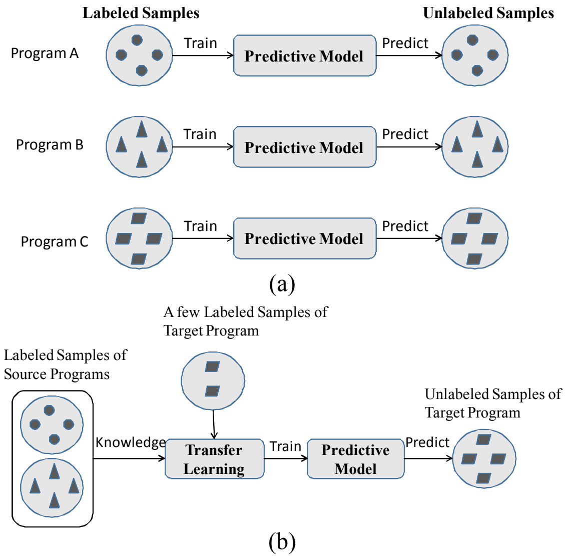


Figure 1. Illustration of DSE with (a) traditional machine learning method and (b) transfer learning method

However, most machine learning based methods aim to build a program-specific machine learning model like regression model for each program [1,3,4,6,7,8,13,14,15]. For instance, İpek et al. [1] proposed to employ artificial neural network (ANN) to build predictive model for architectural DSE. Guo et al. [3,4] proposed to employ semi-supervised learning regression approach and active learning approach with M5P (model tree) as the base predictor to improve the accuracy of prediction. The issue is that when a new program is considered, a large number of new simulations on the program are needed to obtain the corresponding responses as the training data. These data are then used to build a program-specific predictive model to predict the responses of those configurations not simulated. Figure 1(a) shows this traditional learning process of previous works. The simulation results of one program are independent of those of other programs. Given a program, only the labeled samples of this program are used as the training data, but the labeled samples of other programs are ignored. The labeled samples refer to the

CODES/ISSS '16, October 01-07, 2016, Pittsburgh, PA, USA
© 2016 ACM. ISBN 978-1-4503-4483-8/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2968456.2968457

architectural configurations with the corresponding simulated responses, while the unlabeled samples refer to those that are not simulated and thus the corresponding responses are unknown. As the simulations are usually extremely time-consuming, there is a large overhead of obtaining the labeled samples for each program.

To address this problem, recently some works proposed to build cross-program predictors [9,10,11,12]. Khan et al. [9] proposed a reaction-based cross-program predictive model by employing ANN as the base predictor. Dubach et al. [10, 11] proposed a linear regression model as the meta model above some program-specific models built via ANNs. Guo et al. [12] applied M5P as predictor and incorporated the inherent program characteristics as a part of the training data to build the predictive model. Although the experimental results showed the superior performance of the above methods compared to traditional program-specific methods, their performances are still not promising due to the following limitations.

- The off-line training cost is very high. That is, they require a large number of training data of each previous program (e.g. Khan et al. [9], Guo et al. [12] and Dubach et al. [10, 11] require 1000, 500 and 512 training instances of each prior program, respectively). In addition, a lot of previous programs (e.g. [9] requires 17 programs, [12] requires 24 programs and [10,11] requires 5 programs) are needed. Although the number of simulations on the new program is reduced, the off-line training cost is very high.
- The assumptions in [9] and [10,11] are that similar programs should present similar responses on the same configurations. That is a too coarse-grained knowledge transfer. Once two programs are dissimilar, the effectiveness of their models is undesirable or even worse than the methods without knowledge transfer.
- Previous works are not able to transfer knowledge from the sample granularity. That is they cannot select the helpful samples and filter the harmful ones from the previous programs for the new program. Even for two dissimilar programs, it's possible that some samples in one program could be potentially helpful to predict the simulation responses of the other program.

To address above mentioned problems, for the first time we study how to select useful samples and transfer the knowledge from one or several programs to help us build a better predictive model for the target program whose labeled samples are very sparse. Transfer learning has achieved significant success in many machine learning tasks including classification, regression, and clustering to address the problem of lacking enough training data in the target domain [22-26]. Figure 1(b) gives an illustration of transfer learning, which is different with traditional machine learning. It aims to transfer the knowledge from some source domains to a target domain when the latter has very few high-quality training data. Following the terms in transfer learning, we call the new program to be predicted as target domain program and the prior programs as the source domain programs. At the meantime, we study how to select source domain programs that are similar to the target domain program in order to improve the effectiveness of knowledge transfer. The intuition is that similar programs may contain more samples that present similar configurations and corresponding responses. Motivated by this idea, we propose a two-phase DSE framework TrDSE that includes the program clustering phase and the knowledge transfer phase. In the clustering phase, we firstly use orthogonal array sampling to sample a very small number of configurations for each benchmark for simulation, and then we propose to extract five distribution related features based on the simulation responses for each benchmark. With the extracted features, the hierarchical clustering algorithm is applied to cluster the benchmarks. In the knowledge transfer phase, based on the clustering results, a transfer learning based regression algorithm is employed to transfer knowledge from source domain programs to the similar target domain program. The program clustering phase provides us with guidance for the transfer learning among programs. The goal of TrDSE is to improve the prediction accuracy on the target program with a very few simulations by transferring the helpful labeled training samples of source domain programs.

To summarize, this paper makes the following contributions:

(1) A new program clustering algorithm is proposed based on orthogonal array sampling. We extract five data distribution related features for program clustering. The insight is that the programs with similar distributions of response could share more common knowledge.

(2) To our knowledge, this is the first work that employs transfer learning algorithm in DSE to transfer helpful samples from source domain programs to help predict the responses of the configurations in the target program, thereby significantly reducing the simulation cost.

(3) Experimental results demonstrate the effectiveness, efficiency as well as robustness of TrDSE. Different from the previous cross-program predicting models [9,10,11,12], TrDSE does not firmly require a relatively large number of training data of the source domain programs, thus is more efficient and robust. Compared with two state-of-art DSE techniques, TrDSE significantly reduces the number of simulations and improve the prediction accuracy.

The rest of the paper is organized as follows. Section 2 details 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. PROGRAM CLUSTERING AIDED TRANSFER LEARNING FOR DSE

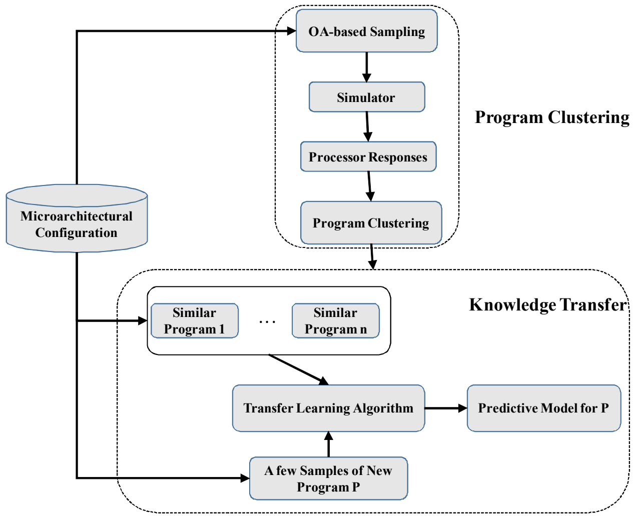


Figure 2. Framework of TrDSE

In this section we introduce the novel two-phase framework TrDSE for efficient design space exploration. TrDSE effectively combines program clustering technique and transfer learning technique in order to reduce the simulation cost in DSE and at the mean time train a more accurate prediction model. Figure 2 presents the framework of TrDSE. In the program clustering phase, we firstly propose to use OA sampling strategy to sample 12 configurations for simulation for each program, and then from these processor response dataset, we extract the values of five distribution related features constituting the datasets for clustering. Based on the distribution related dataset of each program, a hierarchical clustering algorithm is applied to cluster all the programs. In the knowledge transfer phase, according to the program clustering results, we propose a transfer learning algorithm to transfer the knowledge of similar source domain programs to the target program for training a better prediction model. To elaborate our proposal, we next give a brief introduction on orthogonal design.

2.1 Orthogonal Design

The program clustering phase is the basis and guidance for the transfer learning phase. It does not necessarily provide very accurate results, i.e. a relative similarity among programs is enough for guiding transfer learning. Therefore, in order to reduce the simulation cost, one efficient sampling strategy for computing the similarity between two programs is critical.

An efficient design of experiment (DoE) method is essential to investigate the relationship between parameters and responses, and the sampling strategy is the key of DoE. The widely used DoE methods in statistical domain include full factorial, random, Latin hypercube sampling (LHS), Plackett-Burman design (PB design) and orthogonal design [16]. Here, we give a brief introduction and comparison on these techniques. 1) A full factorial experiment is an experiment which takes on all possible combinations of all values (levels) across all factors (parameters). But in DSE, it is obviously infeasible to sample all the configurations in the huge design space for simulation. 2) Random sampling strategy may lead to uncertainty and some redundant [20] or less informative design points are included, which results in high cost. 3) LHS is a statistical method for generating a sample of plausible collections of parameter values from a multidimensional distribution. In the context of statistical sampling, a square grid containing sample positions is a Latin square if (and only if) there is only one sample in each row and each column. However, the number of samples should be determined by designer subjectively. 4) The goal of PB design [2] is to find an experimental design where each combination of levels for any pair of factors appears the same number of times. But each parameter is only associated with two levels, it is not suitable for microarchitectural design space with multiple parameters and levels.

Instead of testing all possible parameter combinations like the factorial design, orthogonal design conducts a minimum amount of representative experiments according to orthogonal array (OA) to cover the entire design space and investigate how each parameter affects responses. As the selected representative experiment number is usually very small, time and resources cost can be significantly reduced. The OA can be derived from deterministic algorithms [16] or looked up online. The arrays are selected by the number of parameters and the number of levels. 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 relationship between parameters and responses. However, for OA only 8 simulations are needed, 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.

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. For example, level 1 and level 2 of parameter B in the second column of $L_8(4^1 \times 2^4)$ occur 4 times respectively. (2) Each possible level combination of any two given parameters occurs the same times in the array. For example, for the first two columns (i.e., parameter A and parameter B) of $L_8(4^1 \times 2^4)$, the possible level combinations (1,1), (1,2), (2,1), (2,2), (3,1), (3,2), (4,1) and (4,2) occur and they occur once.

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

According to the two balancing properties, it can be shown that OA has the characteristics of representativeness and orthogonality (i.e., two columns are statistically independent). The OA based sampling strategy (OA sampling for short) has three advantages comparing with the other four designs of experiments methods. Firstly, it can evenly cover the entire design space and the samples are representative for the distribution of design space [16]. Secondly, the orthogonal design is an experiment design of multiple parameters with multiple levels. Thirdly, OA sampling is more efficient and is able to catch the relationship between the parameters with the responses via the minimum samples. Thus, we choose the OA sampling strategy in orthogonal design as the sampling strategy to reduce the simulation cost.

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.

If the parameters are all with 2 levels, the corresponding OA generally is called 2-level OA. For example, 11 parameters with 2 levels, the corresponding 2-level OA is $L_{12}(2^{11})$, just needing 12 experiments. When the levels of parameters are different, the corresponding OA generally is called mixed-level OA.

2.2 Program Clustering

We take the program similarity as the metric to measure the transferability between source and target domains. The intuition is that similar programs should share more knowledge in general. TrDSE aims to transfer the knowledge from the source domain programs that are similar to the target domain program. Thus program clustering is considered to help us select source domain programs for the target domain program. It could provide guidance for source domain program selection and clustering results, i.e., selecting more similar source domain programs for the target domain program.

In DSE, if the distributions of the simulation responses of two programs are similar on the same configurations, many samples of one program could be helpful and used to build the predictive model of the other program. Thus we compute the similarities between programs based on the distribution of simulation responses of programs. Our method is significantly different from previous works on measuring program similarities [27, 28], where the used features of programs are architecture irrelevant (such as instruction mix or branch direction). The purpose of program clustering is to provide a coarse guidance for knowledge transfer in TrDSE. The program clustering phase mainly includes the following 3 steps.

Step1. We propose to apply 2-level OA sampling (only considering the highest and lowest values of each parameter) to sample configurations from the entire design space, and then simulate each program on these configurations. For 11 investigated parameters in our paper, 2-level OA only needs to sample 12 configurations ($L_{12}(2^{11})$).

Table 2. Five data distribution related features for clustering

No.	Distribution related Features
1	The maximum value
2	The minimum value
3	The difference between maximum value and minimum value
4	Average value
5	Standard variance

Step 2. Based on the simulation results of Step 1, we extract five features related to response distribution for charactering each program. In this paper, we only consider instruction per cycle (IPC) as the processor response or simulation results on the program. The five features are extracted based on the response values, and are shown in Table 2, i.e., the maximum response value, the minimum response value, the difference between maximum value and minimum value, the average value and the standard variance. The values of the five features are extracted from the 12 samples of each program and they constitute the dataset for clustering.

Step 3. The hierarchical clustering algorithm [21] is utilized to cluster the program. It is a method of cluster analysis which seeks to build a hierarchy of clusters and the results are usually presented in a dendrogram. Based on the response distribution related features extracted from the samples of all the investigated programs, Euclidean distance is used as the measure of distance between pairs and centroid linkage clustering is applied as the linkage criterion. The similarity of two programs is expressed as the distance of the respective five feature data of them.

The related works about program clustering based on simulation response are Yi et al. [2] and Dubach et al. [11]. Yi et al. [2] apply PB design to sample configurations for each program, and then compute the effect of each parameter on response and rank them. They take the rank as the clustering data of each program, and calculate the Euclidean distance of any two programs for clustering. The method is effective, but loses some information of the response values. For example, although the ranks of parameter significance are the same for the two programs, the responses of the two programs on the same configuration may be different significantly. In this case, the training data of one program can't help predict the responses of the other program. Dubach et al. [11] propose to measure the program similarity by directly using the results of running benchmarks on 3000 randomly selected configurations. Then Euclidean distance is also used as the measure of distance. The simulation cost is too high, and it does not mine the distribution information of the data.

2.3 Transfer Knowledge among Different Programs

Based on the program clustering, we consider employing the knowledge of source domain programs to help train the predictive model of the target domain program. The intuition is that part of the labeled data in the source domain can be reused in the target domain after re-weighting. We propose to apply an instance-transfer learning algorithm Two-stage TrAdaBoost.R2 [22] to select the helpful samples and remove the harmful samples of source domain programs. However, we need to address the following three problems to apply this method to our task. How to sample the labeled samples from the source domain programs? How to sample the labeled samples from the target domain program? How to transfer the knowledge from source domain programs to the target domain program? We will address the three problems in three steps respectively as follows.

Step 1. Based on program clustering results, we select one or several similar programs that are in the same cluster of the target program, and select some labeled samples from these programs. The samples can be sampled randomly. The sample size can be flexible, because the instance-transfer learning is fine-grained.

Step 2. As the useful samples of source domain programs are selected based on the very limited number of labeled samples in the target domain program, these sampled labeled samples of the target program should be representative and informative. We apply the mixed-level orthogonal array sampling strategy to sample configurations for simulation in the target domain program. For the design space in Section 3.2, the corresponding mixed-level OA is $L_{64}(8^4 \times 4^4 \times 2^3)$ which includes 64 samples. However, in reality, running 64 simulations for each program is still time consuming. If the resources are limited, e.g., only 10 or 20 simulations can be simulated, we propose to randomly sample 10 or 20 configurations from the 64 configurations. Although the 10 or 20 configurations cannot cover the entire design space, the distribution of the 10 or 20 configurations can still approximately represent the distribution of the entire design space.

Step 3. We select one or several programs that are similar to the target domain program as the source domain programs for knowledge transfer. We then leverage the labeled training data of the source domain programs and the target domain program to select the helpful samples. Finally, a predictive model is trained by the samples of the target program and the helpful samples transferred from the source domain programs.

We propose to employ the instance-transfer learning algorithm two-stage TrAdaBoost.R2 [22] to leverage the training samples of source domain programs to help us build the predictive model for the target domain program. Two-stage TrAdaBoost.R2 is a variant algorithm of TrAdaBoost [23] for regression. TrAdaBoost aims to address the classification problem in the inductive transfer learning setting. It assumes that due to the difference in distributions between the source and target domains, some of the source domain data may be useful in learning for the target domain but some of them may not and could even be harmful. TrAdaBoost attempts to iteratively re-weight the source domain data to reduce the effect of the "bad" source data while encouraging the "good" source data to contribute more for the target domain. To avoid the overfitting problem in TrAdaBoost, Two-Stage TrAdaBoost.R2 is proposed to adjust the weights of the samples in two stages.

Algorithm 1 presents the pseudo-code of the Two-stage TrAdaBoost.R2. It takes two training datasets as input. Let T_{target} (of size m) be the training set of the target domain program and T_{source} (of size n) be the training set of source domain programs. If there are more than one source domain program, we simply combine all source data sets into a larger data set T_{source} . As the two-stage TrAdaBoost.R2 handles the reweighting of each training instance separately, there should be no harm in mixing data in this fashion. Let T be the combination of T_{source} and T_{target} such that the first n instances in T are those from T_{source} . In the beginning, the initial weight of instances in T is set to a uniform weight $1/(n+m)$. Stage 2 (line 1) is embedded inside stage 1 (the remainder of for-loop). In stage 1, the learner (M5P model is selected as the learner in this paper) is supplied with T of $n+m$ samples with distribution w^t. The error for each sample of source domain can be used in reweighting them. The adjusted error of each instance of source domain programs is calculated by Equation 1,

$$e_i^t = |y_i - h_t(x_i)| / D_t, \quad D_t = \max_{j=1}^{n} |y_j - h_t(x_j)| \qquad (1)$$

where $h_t(x_i)$ is the M5P base learner, y_i is the IPC value of cycle-accurate simulation, D_t is the largest prediction error of a sample, and e_i^t is the relative prediction error of the *i*-th sample. Then as shown in Equation 2, only the weights of the samples of source domain programs are adjusted downwards gradually until reaching a certain point (determined through cross validation in stage 2).

$$w_i^{t+1} = \begin{cases} w_i^t \beta_t^{e_i^t} / Z_t, & 1 \le i \le n \\ w_i^t / Z_t, & n+1 \le i \le n+m \end{cases} \qquad (2)$$

where w_i^{t+1} is the weight of *i*-th sample in the *(t+1)*-th iteration, Z_t is a normalizing constant. Note that the weighting factor β_t is chosen to approximately satisfy the total weight for the target instances are $\frac{m}{(n+m)} + \frac{t}{S-1}(1 - \frac{m}{(n+m)})$ using a binary search. In this way, the total weight of the target instances increases uniformly from $m/(n+m)$ to 1 in S steps. For more details of this algorithm one can refer to [22].

Algorithm 1: Two-stage TrAdaBoost.R2 for DSE

```
Input:
$T_{source}$ : training dataset of the source domain program with $n$ samples;
$T_{target}$ : training dataset of the target domain program with $m$ samples;
$T$ : the combination of $T_{source}$ and $T_{target}$, the first $n$ samples are
    those from $T_{source}$
$N$ : the maximum number of boosting iterations
$F$ : the number of folds for cross validation
$w_i^t$ : the weight of the $i$-th samples in the $t$-th iteration, $1 \le i \le n+m$
Initialize: $w_i^1 = 1/(n+m)$
For $t = 1, \ldots, S$ :
  1. Call AdaBoost.R2' with $T$, distribution $w^t$, $N$, and Learner to obtain
     $\mathrm{model}_t$, where AdaBoost.R2' is identical to AdaBoost.R2 except that the
     weights of the first $n$ samples are never modified. Similarly, use F-fold
     cross validation to obtain an estimated $error_t$ of the error of $\mathrm{model}_t$.
  2. Call Learner with $T$ and distribution $w^t$, and get a
     hypothesis $h_t : X \to \mathbb{R}$.
  3. Calculate the adjusted error $e_i^t$ for each instances:
       let $D_t = \max_{j=1}^{n} |y_j - h_t(x_j)|$
       then $e_i^t = |y_i - h_t(x_i)| / D_t$
  4. Update the weight vector:
       $w_i^{t+1} = \begin{cases} w_i^t \beta_t^{e_i^t} / Z_t, & 1 \le i \le n \\ w_i^t / Z_t, & n+1 \le i \le n+m \end{cases}$
     where $Z_t$ is a normalizing constant, and $\beta_t$ is chosen such that the resulting
     weight of the target (final $m$) samples is $\frac{m}{(n+m)} + \frac{t}{S-1}(1 - \frac{m}{(n+m)})$.
Output $\mathrm{model}_t$ where $t = \mathrm{argmin}_i error_i$
```

In stage 2, AdaBoost.R2 is applied as the regression model. AdaBoost.R2' is identical with AdaBoot.R2 except for keeping the source domain instances weight unchanged. The weights of source domain samples are adjusted downwards gradually in S steps, and the AdaBoost.R2' with the lowest cross-validation error are selected as the predictive model for DSE.

3. EXPERIMENT SETUP

3.1 Simulator and Benchmarks

Using the state-of-art cycle-accurate simulator GEM5 [17], we conduct our simulations on 27 benchmarks from the SPEC CPU 2006 suite. Aided by SimPoint [18], 100 million instructions of 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

In this paper, we investigate 11 important parameters for processor performance. The design space is shown in Table 3. We fix CPU clock as 2GHz and branch predictor as tournament. As architectures with ROB size less than the instruction queue size is impossible to achieve optimal performance, we can directly remove such architectures and reduce the design space from 8.3 million to 7 (8.3-1.3) million configurations. For the design space, the corresponding mixed-level 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. Two-level OA just needs 12 samples. According to the simulation results of the 12 samples of each program, the five features described in Section 2.2 are extracted for program clustering.

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**

Thus, 12 configurations are cycle-accurate simulated as the dataset for program clustering. We use uniform random sampling to pick 3000 architectural configurations and simulate them for each benchmark. Unless otherwise stated, all our predictors are validated using these 3000 samples, i.e., they are used as the test dataset in all following experiments.

3.3 Evaluation Methodology

In order to evaluate the performance of the predictive model, we use the relative mean absolute error (RMAE) defined as $rmae = \left| \frac{predicted\ value - real\ value}{real\ value} \right| * 100\%$. This metric explicitly demonstrates the predictive error of a regression model. A lower RMAE means a better predictive model.

The Pearson correlation coefficient is also used as a metric for better evaluating the fitness of learned regression model. The Pearson correlation coefficient between the predicted value and the real value is defined as $corr = \mathrm{cov}(pred, real) / \sigma_{pred} \cdot \sigma_{real}$.

$\mathrm{cov}(pred, real)$ is the covariance of the predicted value and the real value. σ_{pred} and σ_{real} represent their standard deviations. The Pearson correlation coefficient ranges from -1 to 1, where 1 means the predicted value perfectly models the shape of the real value.

3.4 Competitive Methods

Some input parameters should be set first in two-stage TrAdaBoost.R2. In our experiment, the number of cross-validation folds F is set to 10 and the number of iterations S is set to 200. The M5P algorithm [29] is chosen as the base learner in AdaBoost.R2'. The maximum number of boosting iterations N of AdaBoost.R2' is set at 20.

We compare TrDSE with the following baselines.

- A state-of-art program-specific predictive model based on artificial neural networks proposed by Ïpek et al. [1] (PSM for short). Following the setting in [1], the ANN adopts one 16-unit hidden layer, a learning rate of 0.001.
- We also compare TrSDE with an empirical architecture-centric approach based on linear model proposed by Dubach et al. [10,11] (EAC for short). EAC predicts the new program using knowledge of previous programs. As the most relevant work to TrDSE, we choose it as a baseline.

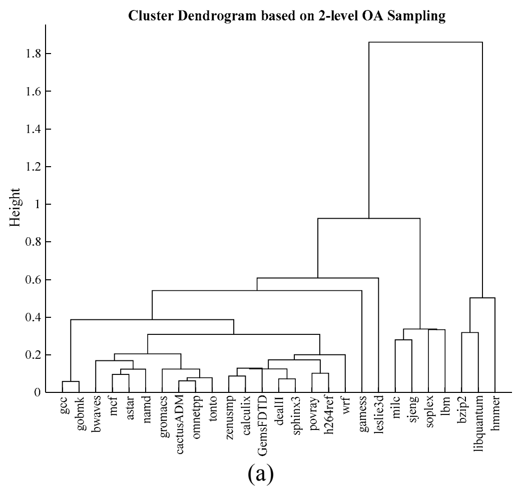
Another cross-program predicting related work, Khan et al. [9] proposed a cross-program reaction-based predictive model. Their experiments are preceded using 17 previous benchmarks with 1000 training samples per benchmarks to predict a new program. Although they proposed that only 8 further simulations for a new program are needed, the off-line training time cost is very high. Therefore, we do not compare our method with it due to its large number of previous benchmarks and training samples.

4. EVALUATION

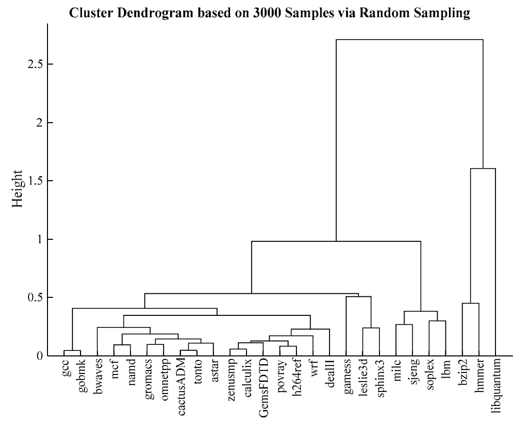
4.1 Evaluation of Program Clustering

The proposed program clustering approach utilizes the extracted distribution related feature data based on 2-level OA sampling. For the 11 parameters in the design space, the corresponding 2-level OA is $L_{12}(2^{11})$ requiring 12 simulations of each program. In order to evaluate the effectiveness of 2-level OA sampling, we compare the program clustering results based on 3000 randomly selected samples.

Figure 3 shows the dendrogram results by applying a hierarchical clustering method on the two different datasets of SPEC CPU 2006 benchmarks. The horizontal lines join two branches together and the height on the y-axis gives the average distance between them. The higher separation between branches, the less similar the programs in each branch are from each other. For example, the benchmarks *libquantum, bzip2*, *hmmer* are far away from other benchmarks, i.e., they are significantly different with other programs. We can see that the dendrograms of the two clustering results are almost the same, despite there are some differences, e.g., the nearest benchmark of *mcf* is *astar* in (a), while the nearest benchmark of *mcf* is *namd* in (b). The small differences between the two results can be ignored because the three benchmarks are all similar with each other. Here, we only consider the relative similarities between benchmarks. As shown in Table 4, the clustering results on the two program datasets are the same when we cluster the 27 benchmarks to 3 clusters. Thus one can conclude that the proposed 2-level OA sampling can effectively select the most representative samples, and can achieve the comparable clustering results with only a very small number of simulations for each program.



(a)



(b)

Figure 3. Hierarchical clustering results on SPEC CPU 2006 benchmarks with different sampling methods: (a) 2-level OA sampling (12 samples) (b) randomly sampling 3000 samples

Table 4. The program clustering result

No	Benchmarks	Number
1	*gcc, gamess, zeusmp, leslie3d, povray, calculix, h264ref, wrf, bwaves, mcf, cacyusADM, namd, gobmk, dealII, GemsFDTD, tonto, omnetpp, astar, sphinx3, gromacs*	20
2	*milc, sjeng, soplex, lbm*	4
3	*bzip2, hmmer, libquantum*	3

4.2 Evaluation of Knowledge Transfer

When transferring knowledge from source domain programs to the target domain program, the following three variables may significantly affect the prediction performance: the number of source domain programs, the available training data size of source domain programs, and the training data size of the target program.

In order to evaluate the effectiveness of TrDSE, we conduct experiments to test the performance with different settings of the three variables separately.

4.2.1 Knowledge Transfer from Multi Source Domain Programs

For TrDSE, the source domain programs that are most similar to the target domain program are selected for knowledge transfer based on the clustering result. For example, if we select 3 source domain programs for the target domain program *bzip2*, the programs *hmmer*, *libquantum* and *lbm* (or, *soplex*) are selected since they are the most similar 3 programs to *bzip2*.

In this section, we evaluate the performance of TrDSE by varying the training set size of the target program and the source domain program number. The training set size of the target program, i.e., the simulation number of the target program, is set to 10, 20, 30 and 40, respectively. The number of source domain programs varies from 1 to 5. The training set size of each source domain program is fixed at 50, i.e., the available training dataset of 5 source programs totally includes 250 samples. Thus for each program we need to conduct 20 experiments with each one associated with a variable setting.

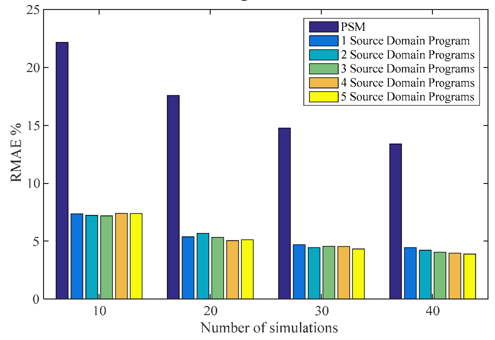
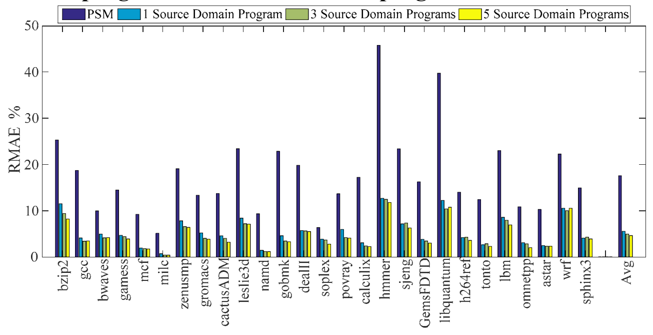
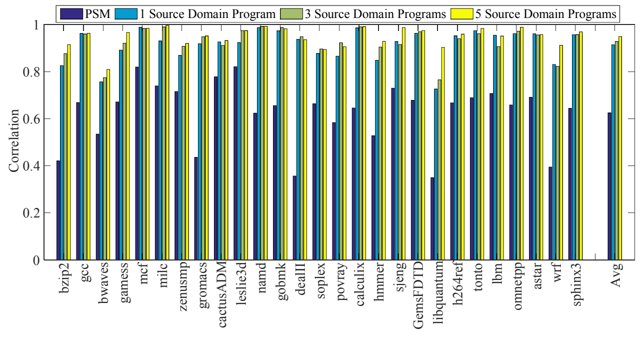


Figure 4. Average RMAE of TrDSE and PSM of 27 benchmarks with various training set sizes of the target program and source domain program numbers



(a) RMAE



(b) Pearson correlation coefficient

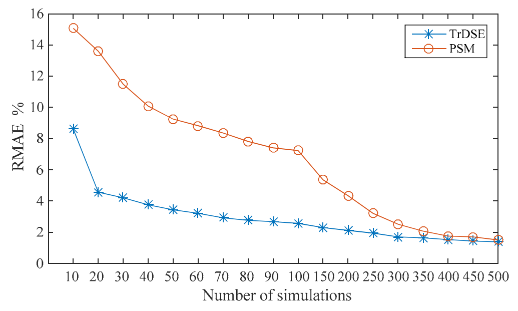
Figure 5. RMAE and Pearson correlation coefficient of PSM and TrDSE with 1, 3, 5 source domain programs and 20 training samples of the target program.

Figure 4 shows the average RMAE of TrDSE of all the benchmarks. PSM is the baseline approach. As can be seen, on average the RMAE of TrDSE with 10 training samples of the target program and one source domain program is only 7.36%, while the number for PSM is 22.7%, which shows a very significant improvement. The prediction error of TrDSE reduced to 5.38% with only 20 training samples of target domain program. One can conclude that the significant improvement of TrDSE benefits from transferring knowledge from similar source domain programs. One can also see that for TrDSE, more source domain programs could transfer more knowledge for the target program, but due to the limit training samples of the target program, the common knowledge that can be transferred is limit. That is why the decreasing trend of the prediction error becomes slow when the source domain program number is larger than 3. Figure 4 also shows that, when the training set size of the target program increases to 20, the prediction error is greatly reduced compared to the case with only 10 available training samples, and the prediction error is reduced to about 5.3% on average. When the training sample size of the target program keeps rising, the reduction of the prediction error becomes less significant.

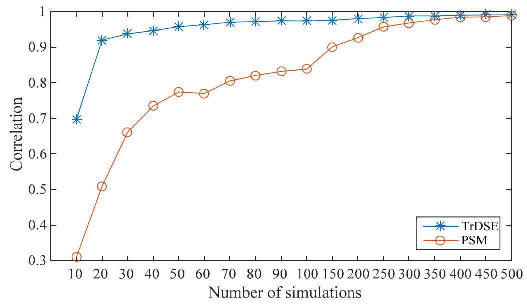
Due to page limitation, we can't show all the RMAE and Pearson correlation coefficients of all the variable combinations (20 experiments for each program) on all the benchmarks. As a representative case, we show the performance of TrDSE with source domain program numbers 1, 3, 5 respectively by fixing the training set size of target program at 20 in Figures 5. The performance improvement of TrDSE is particularly significant on the benchmarks that are hard to predict. For example, the prediction error of PSM on *hmmer* benchmark is 34.33% with 20 training samples, while TrDSE reduces the error to 12.69% with one source domain program. Comparing with the Pearson correlation coefficient 0.62 achieved by PSM, TrDSE dramatically improves it to 0.91 with one source domain program on average. For the benchmark *dealII*, the improvement is particularly significant, from 0.36 to 0.94.

One can see that 3 source domain programs is good option for knowledge transfer as the performance improvement in such a case is most significant. For many benchmarks, when the number of source domain programs is larger than 3, the prediction accuracy increases slowly. The reason could be that some dissimilar programs are included, which makes the knowledge transfer more difficult. In addition, the available knowledge from source domain programs is limit due to the very sparse training data of the target program.

4.2.2 Varying the Simulation Size of Target Program



(a) RMAE



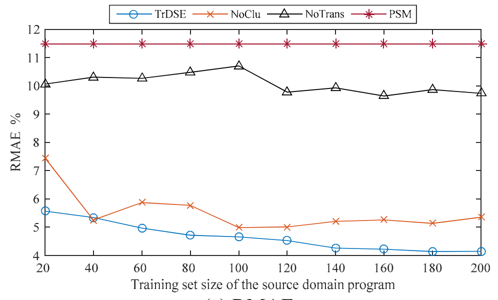
(b) Pearson correlation coefficient

Figure 6. RMAE and Pearson correlation coefficient of PSM and TrDSE by varying the simulation number of the target program

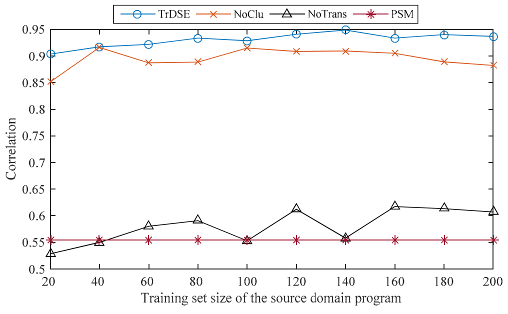
To evaluate the effectiveness of TrDSE with various simulation numbers (training dataset size) of the target program, we compare TrDSE with PSM by increasing the number of simulations from 10 to 500. We aim to compare the performance of TrDSE and PSM given the equal number of simulations from the target program. For TrDSE, we fix the number of source domain programs at 3, and use 50 training samples per source domain program. Figure 6 shows the average prediction error and Pearson correlation coefficient over all the programs of the two methods. One can see that TrDSE reduces the prediction error from 15% to 8.6% with only 10 simulations and improves the Pearson correlation coefficient from 0.3 to 0.7. When the number of simulations increases to 20, the prediction error of TrDSE drops to 4.5%, and the Pearson correlation coefficient increases to 0.92. With the increase of simulation number, the prediction performance of the two approaches both improves. By transferring knowledge from source domain programs, only 30 simulations of the target program are needed for TrDSE to achieve a 0.93 of Pearson correlation coefficient, while PSM needs 200 simulations to get the similar result.

4.2.3 *Robustness Analysis*

In this section we study the robustness of TrDSE by transferring knowledge among programs that are not very similar. We set the source domain program number to 1, and training set size of target program to 20. We investigate the prediction accuracy of TrDSE by varying the training set size of source domain program.



(a) RMAE



(b) Pearson correlation coefficient

Figure 7. RMAE and Pearson correlation coefficient of TrDSE, NoClu, NoTrans and PSM varying training set size of the source domain program

In addition, we also conduct experiments with the following 3 approaches as comparison. (1) NoClu: We randomly select a source domain program without considering its similarity to the target domain program, and then apply the two-stage transfer learning techniques to build a predictive model for the target program. (2) NoTrans: We randomly select the same amount of samples with TrDSE and NoClu from the most similar source domain program. The samples are then directly used for the target domain program to build a predictive model. (3) The third comparison method is PSM.

Figure 7 shows the average prediction error and Pearson correlation coefficient of all the 27 benchmarks of the four approaches. As the training data size of target domain program is fixed, the prediction error of PSM is constant and thus the curve in the figure is a line. One can see that, TrDSE is most correlated with the actual data. When the training set size of source domain programs increases to 140, the prediction error decreases to 4.2%, and Pearson correlation coefficient increases to 0.95.

One can see the performance of NoTrans is better than PSM. That demonstrates that the effectiveness of program clustering and there are some samples could be transferred and applied between the similar programs. The reason that NoTrans is not competitive with NoClu is that despite the two programs are similar, their response distributions are not exactly the same. That is there are a lot of helpful samples for the target domain program, but meanwhile some harmful ones also exist. The helpful samples could improve the prediction accuracy of NoTrans, while the harmful ones could lower the accuracy. Thus directly using the samples of source program may not be always helpful to the target program. The transfer learning algorithm of NoClu could filter the harmful samples for the predictive model of target program. The program clustering can guide the selection of source domain programs for transfer learning, which helps to sample the data from the source program that has the similar training data distribution with the target program.

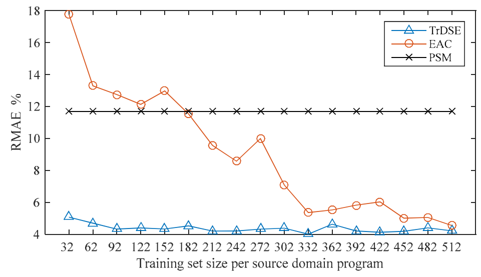
One can also see the prediction accuracy of NoClu is a little bit worse than TrDSE, but still much better than program-specific model PSM. This verifies the effectiveness of our method in transferring useful samples form the source domain program to the target domain program. Note that the prediction performance of NoClu is close to TrDSE. The reason is that the result given in Figure 7 is the average prediction error on the 27 benchmarks, and many of the benchmarks in SPEC CPU 2006 are similar. However, when NoClu selects the most dissimilar benchmark *libquantum* for *gcc*, the prediction error is 8.8% with 200 training samples of *libquantum*, while the error drops to 2.9% when TrDSE selects the very similar benchmark *gobmk*. The prediction error of PSM is 15%. In this case, NoClu is much worse than TrDSE but still outperforms PSM. It demonstrates the robustness of transfer learning and the necessity of program clustering.

4.2.4 *Comparison with Cross-Program Model*

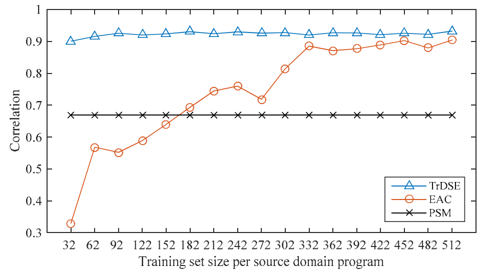
To further study the effectiveness and efficiency of TrDSE, we compare it with the state-of-art cross-program predicting model EAC proposed by Dubach el at. [10,11]. EAC uses prior knowledge from 5 benchmarks to help predict the new program. Notably, EAC requires 512 training samples for each of the 5 benchmarks. On this basis, 32 further simulations for the new program are also needed for building the predictive model. One critical drawback of EAC is that the cost of off-line training is too high, requiring 512 training data for each of 5 benchmarks.

For fairness, we also select 5 source domain programs for knowledge transfer, 32 training samples of the new target program for prediction. When varying the training set size per source domain program, we compare the performance of TrDSE with EAC. Note that the training set size of the 5 benchmarks is always the same and increases from 32 to 512 with an increase step of 30.

Figure 8 shows the average prediction accuracy over all the 27 programs of the three DSE approaches. As can be seen, EAC is remarkably inferior to TrDSE when the training set size of source programs is relatively small. It is even worse than PSM when the training set size of source domain programs is less than 182. Even with only 32 samples per source domain program, the prediction error of TrDSE is 5.1%, and the Pearson correlation coefficient is 0.9, a very promising result. When the training set size is larger than 182 per source program, EAC becomes better than PSM. When the training set size per source program increases to 512, EAC can also achieve promising performance with 4.5% prediction error and 0.9 Pearson correlation coefficient. To achieve the similar result, only 92 samples per source domain program are enough for TrDSE (prediction error of 4.3% and Pearson correlation coefficient of 0.92).



(a) RMAE



(b) Pearson correlation coefficient

Figure 8. Comparison of three methods

Table 5. The simulation cost and training cost of different methods

Approach	The number of simulation for each program		Training time	Total number of simulation
	Source program	Target program		
EAC	512	32	$\approx 46(s)$	$512\times 5+32\times 22=3264$
TrDSE	92	32	$\approx 21(s)$	$92\times 5+32\times 22=1164$
PSM	0	200	$\approx 17(s)$	$200\times 27=5400$

With a similar prediction performance (about 4.5%), the simulation cost and training cost of TrDSE, EAC and PSM are shown in Table 5. In general, a simulation for 100 million instructions requires an execution time of more than 10 minutes, which may be different depending on the program. For the sake of simplicity, we take the number of simulations as the metric to measure the simulation cost. According to Section 4.2.2, PSM needs about 200 training samples for each program to achieve a comparable prediction accuracy. For the 27 benchmarks used in our experiments, the 5 source programs are considered as the previous programs and the other 22 programs are the target programs. Thus the total number of simulations for TrDSE and EAC includes the number of simulations on both source programs and target programs. One can also see that, the training time of the three methods are all less than 1 minute, which could be ignored in DSE compared with the time consuming simulations. TrDSE is much more efficient than PSM and EAC to achieve a similar prediction performance, as TrDSE only needs less than one fifth simulations of PSM and one third simulations of EAC. Therefore, TrDSE could dramatically reduce the simulation time in DSE. This is mainly because TrDSE is a fine-grained knowledge transfer method that focuses on selecting the most useful training samples from the source domain programs, while EAC is a coarse-grained knowledge transfer method that aims to transfer the entire source domain programs.

4.2.5 Optimal Configuration

In general, the actual optimal configuration cannot be obtained unless simulating the entire design space, which is infeasible. A compromise is using TrDSE to explore test dataset space as a case study to evaluate its effectiveness. We predict the configurations of test set and sort them with the predicted IPC value, and the one with the largest IPC is considered as the optimal configuration predicted by the model. We next compare it with the actual best simulation configuration of the entire test set. Here, we fix the training set size of the target program as 50 and the number of source programs as 3. Since the prediction accuracy of EAC is lower than PSM when the training set size of source domain program is small, we only compare the performance of TrDSE with PSM in this section. Table 6 shows the optimal simulation configuration, and the optimal predicted configurations and prediction error of TrDSE and PSM for benchmark *libquantum* and *wrf*.

One can see that although the optimal predicted configuration of TrDSE is not totally the same with the optimal simulation configuration, most of the parameters are the same. And they do not yield significant performance differences. It is notable that on the benchmark *wrf*, the optimal predicted performance and the optimal simulation performance are very close, and meanwhile its prediction error is low (1.6%). Compared with PSM, TrDSE is more effective.

Table 6. The optimal simulation configuration and the predicted optimal configurations of different methods

Parameter	***libquantum***			***wrf***		
	Simulation	TrDSE	PSM	Simulation	TrDSE	PSM
ROB size	144	144	112	144	144	112
Width	8	6	8	8	8	8
L1 DCache	64kB	16kB	128kB	128kB	128kB	16kB
L1 Icache	16 kB	32kB	64 kB	128kB	128kB	128kB
L2 Ucache	4MB	2MB	4MB	1MB	4MB	512kB
MSHR	4	4	4	4	4	4
Register File	104	104	128	88	136	80
IQ	112	64	80	112	32	112
LSQ	96	112	64	112	96	128
ALUs	4	4	4	4	4	2
FPUs	4	4	2	4	4	4
Actual	3.8961	3.896	3.43	1.425	1.4246	1.35
Predicted	---	3.70	3.99	---	1.448	1.49
Error	---	5%	16.4%	---	1.6%	10.3%

5. RELATED WORK

1) Program-specific predictor for DSE. İpek et al. [1] proposed to employ ANNs to predict the performance of architectural configurations. Lee et al. [7] derived spline-based regression models for architectural performance and power prediction. Later Lee et al. [8] proposed a composable performance regression model for multiprocessor. Joshep et al. [5, 6] proposed to build linear regression models and a Radial Basis Function (RBF) nonlinear regression model to predict the microprocessor performance. Guo et al. [3, 4] combined semi-supervised learning with active learning to build a predictive regression model. Guo et al. [15] also proposed a framework which built several distinct base regression models and then construct a metamodel to output the final prediction results. Palermo et al. [13] proposed to combine the design of experiments and response surface modeling techniques for managing system-level constraints and identifying the Pareto front. Later Xydis, Palermo et al. [14] proposed spectral-aware pareto iterative refinement for supervised high-level synthesis. **2) Cross-program predictor for DSE.** Program specific models cannot transfer useful knowledge from previous programs to a new program. Thus, some cross-program predictive models are proposed [9,10,11,12].

Transfer Learning (TL). TL aims to transfer the knowledge from some previous tasks to a target task when the latter has fewer high-quality training data. Generally, it can be classified into three different settings: inductive transfer learning, transductive transfer learning, and unsupervised transfer learning [25]. Dai et al. [23] proposed a boosting algorithm TrAdaBoost, which is an extension of the AdaBoost algorithm, to address the inductive transfer learning problems. TrAdaBoost assumes that the source and target domain data use exactly the same set of features and labels, but the distribution of the data in the two domains are different. Pardoe and Stone [22] presented a two-stage algorithm TrAdaBoost.R2 to extend the TrAdaBoost algorithm for regression problems. The idea is to apply the techniques which have been proposed for modifying AdaBoost for regression [24] on TrAdaBoost. Furthermore, to avoid the overfitting problem in TrAdaBoost, [22] proposed to adjust the weights of the data in two stages.

6. CONCLUSION

This paper made the first attempt to combine program clustering and transfer learning techniques for more efficient chip design space exploration. We proposed a novel program clustering approach which extracted five distribution related features based on orthogonal array sampling, and then applied hierarchical clustering algorithm to cluster the programs. This approach could efficiently cluster programs with only 12 simulations for each program. Then we proposed a program clustering aided DSE approach via transfer learning techniques, which could transfer knowledge from other previous programs to the new target program. Empirical evaluation demonstrates TrDSE could significantly reduce the number of simulations in DSE and has strong robustness, and thus it is more efficient and effective comparing with other state-of-art approaches.

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. Shuzhen Yao is the corresponding author.

8. REFERENCES

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

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

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

[4] T. Chen, Y. Chen, Q. Guo, et al., “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’ 2006.

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

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

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

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

[11] C. Dubach, T. M. Jones, and M. F. P. O’Boyle, “An empirical architecture-centric approach to microarchitectural design space exploration,” IEEE Transactions on Computers, vol. 60, no. 10, pp. 1445-1458, 2011.

[12] Q. Guo, T. Chen, Y. Chen, et al., “Microarchitectural design space exploration made fast,” Microprocessors and Microsystems, vol. 37, no. 1, pp. 41-51, 2013.

[13] G. Palermo, C. Silvano, and V. Zaccaria, “ReSPIR: a response surface-based pareto iterative refinement for application-specific design space exploration,” TCAD, vol. 28, no.12, pp. 1816-1829, 2009.

[14] S. Xydis, G. Palermo, et al. “SPIRIT: Spectral-Aware pareto iterative refinement optimization for supervised high-level synthesis,” TCAD, vol. 34, no. 1, pp. 155-159, 2015.

[15] Q. Guo, T. Chen, Z. H. Zhou, et al., “Robust design space modeling,” TODAES, vol. 20, no. 2, pp. 1–22, 2015.

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

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

[18] 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.

[19] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, “SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling,” in ISCA’ 2003.

[20] S. Wang, Z. Li, and X. Zhang, “Bootstrap sampling based data cleaning and maximum entropy SVMs for large datasets,” in ICTAI’ 2012.

[21] S. C. Johnson, "Hierarchical Clustering Schemes" Psychometrika, vol. 32, no. 3, pp. 241-254, 1967.

[22] D. Pardoe, and P. Stone, “Boosting for regression transfer,” in ICML’ 2010.

[23] W. Dai, Q. Yang, G. Xue, and Y. Yu, “Boosting for Transfer Learning,” in ICML’ 2007.

[24] H. Drucker, “Improving regressors using boosting techniques,” in ICML’ 1997.

[25] S. J. Pan, and Q. Yang, “A survey on Transfer Learning,” TKDE, vol. 22, no. 10, pp. 1345-1359, 2010.

[26] S. Wang, H. Zhang, J. Zhang, et al., “Inferring diffusion networks with sparse cascades by structure transfer,” in DASFAA’ 2015.

[27] A. Phansalkar, A. Joshi, L. Eeckhout, and L.K. John, “Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites,” in ISPASS’ 2005.

[28] A. Phansalkar, A. Joshi, and L.K. John, “Subsetting the SPEC CPU2006 Benchmark Suite,” ACM SIGARCH Computer Architecture News, vol. 35, no. 1, pp. 69-76, 2007.

[29] Y. Wang and I.H. Witten, “Induction of model trees for predicting continuous classes,” in ECML’ 1997.