Cross-program Design Space Exploration by Ensemble Transfer Learning

Dandan Li[1], Shuzhen Yao[1], Senzhang Wang[2,3], Ying Wang[4]
[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] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{dandanli, szyao}@buaa.edu.cn, szwang@nuaa.edu.cn, wangying2009@ict.ac.cn

Abstract—**Due to the increasing complexity of the processor architecture and the time-consuming software simulation, efficient design space exploration (DSE) has become a critical challenge in processor design. To address this challenge, recently machine learning techniques have been widely explored for predicting the performance of various configurations through conducting only a small number of simulations as the training samples. However, most existing methods randomly select some samples for simulation from the entire configuration space as training samples to build program-specific predictors. When a new program is considered, a large number of new program-specific simulations are needed for building a new predictor. Thus considerable simulation cost is required for each program. In this paper, we propose an efficient cross-program DSE framework TrEE by combining a flexible statistical sampling strategy and ensemble transfer learning technique. Specifically, TrEE includes the following two phases which also form our major contributions: 1) proposing an orthogonal array based foldover design for flexibly sampling the representative configurations for simulation, and 2) proposing an ensemble transfer learning algorithm that can effectively transfer knowledge among different types of programs for improving the prediction performance for the new program. We evaluate the proposed TrEE on the benchmarks of SPEC CPU 2006 suite. The results demonstrate that TrEE is much more efficient and robust than state-of-art DSE techniques.**

I. INTRODUCTION

With the ever-increasing complexity of microprocessor architecture, the design space grows exponentially. The situation is deteriorated by the fact that the software simulation technology is extremely time-consuming, which makes the design space exploration (DSE) become one major challenge in microprocessors design. To address this issue, machine learning techniques have been widely employed to build predictive models with a small set of simulation results of some configurations to predict the unknown responses of new architectural configurations [1-8]. However, these methods mostly aim to build a program-specific prediction model like regression model for each program. For instance, İpek et al. [1] and Guo et al. [2,3] proposed to use artificial neural network (ANN) and model tree (M5P) respectively to build a predictive model for architectural DSE. However, when a new program is considered, a large number of new simulations on the program are needed to provide corresponding labeled samples for training a new program-specific predictor. 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, some recent works proposed to build cross-program predictors [9-13]. 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 simple linear regression model as the meta model above some program-specific predictors. Guo et al. [12] proposed to incorporate the inherent program characteristics as a part of the training data to build the predictive model. Li et al. [13] proposed to transfer knowledge among programs by directly selecting some training samples of previous programs to enrich the training set of the new program to construct a more accurate prediction model. Although it shows that these approaches are more efficient than traditional program-specific methods, their performances are still less than satisfactory due to the following limitations.

(1) Although these approaches can significantly reduce the number of simulations on the new program, their 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 previous program, respectively), and a lot of previous programs (17, 24 and 5 previous programs are needed respectively in [9], [12] and [10, 11]).

(2) The cross-program models are effective on a set of programs that are similar to each other, but may not be that effective when applied on dissimilar ones. Previous studies show that the prediction performance of cross-program predictors largely depends on the similarity between the previous programs and the new program. The performance may even become worse when directly transferring knowledge between two dissimilar programs [10,11,13].

(3) Without considering the sampling strategy, most of previous works use the random sampling method [9-12] to sample the configurations of the previous programs and the new programs for simulation. As random sampling does not consider the distributions of the samples in the configuration space, it is not capable to uniformly sample the design points. It may lead to select some redundant or less informative design points, which results in a higher simulation cost. A more effective sampling method is necessary and essential to select more representative training samples for building a more accurate prediction model.

To address the above issues, this paper proposes a novel cross-program DSE framework TrEE (ensemble transfer learning for DSE). TrEE includes two phases: the training data sampling phase and ensemble transfer learning phase. For the training data sampling strategy, orthogonal array (OA) sampling strategy proposed by Li et al. [8] is proven to be effective in improving the accuracy of the prediction model compared with the random sampling. However, due to the inherent characteristics of OA sampling, the sample size in the OA is usually fixed and very small, and thus the number of configurations for simulation cannot be determined flexibly. In the sampling phase, we improve OA sampling by using foldover design technology, and thereby OA can be expanded to flexibly sample a much larger number of training data without losing the properties of orthogonality and representativeness. In the ensemble transfer learning phase, following the terms used in machine learning [14], we call the previous programs with large training set as the source domain programs, while call the new programs for simulation and prediction with very sparse training samples as the target domain programs. A new ensemble transfer learning algorithm is proposed to transfer the knowledge from source domain programs to the target program, aiming to build an effective and efficient cross-program predictive model for DSE based on a small set of simulations of the target program.

To summarize, this paper makes the following primary contributions:

(1) Since the sample size of OA sampling is small and insufficient for constructing accurate predictors, an OA based foldover sampling strategy is proposed to expand the OA and make it capable to flexibly sample as many representative and evenly distributed configurations for simulation as required.

(2) A new ensemble transfer learning algorithm is proposed to effectively transfer the knowledge from the source domain programs to the target program, thereby significantly reducing the simulation cost. Different from other previous cross-program DSE methods, TrEE can even transfer knowledge between two dissimilar programs and achieve much better performance.

(3) Experimental results demonstrate the efficiency and effectiveness of TrEE. Compared with random sampling, our OA based foldover sampling can better select the representative samples. Compared with two state-of-art cross-program DSE techniques, TrEE significantly reduces the number of simulations and improves the prediction accuracy.

The remainder of this paper is organized as follows. The next section presents the details of our approach. Section III introduces the experimental setup. Section IV shows the evaluation results. We review related work in Section V and conclude this paper in Section VI.

II. FOLDOVER DESIGN AND ENSEMBLE TRANSFER FOR DSE

In this section we introduce the novel framework TrEE for effective and efficient DSE. TrEE effectively combines OA based foldover sampling and ensemble transfer learning to reduce the simulation cost in DSE, and the entire framework is shown in Fig. 1. An efficient sampling strategy is essential to select representative training data for constructing an accurate predictive model. The training sets for the source domain programs and the target program are both sampled in the sampling phase. With the sampled datasets, the transfer learning model is then used to predict the responses of those not simulated configurations on the target program.

Specifically, in the sampling phase, we propose a new design of experiment method: OA based foldover design sampling (foldover sampling for short), as shown in the upper part of Fig. 1. The OA based foldover design expands orthogonal design by transforming the levels of each parameter in OA. Due to its symmetrical structure and statistical properties, the table created by foldover still have the properties of orthogonality and representativeness. Thus, it could flexibly sample any number of representative design points that are evenly distributed throughout the design space. In the transfer learning phase as shown in the middle part of Fig. 1, we firstly construct some source domain program predictors based on the training sets of the source domain programs. Then based on the sparse training set of the target domain program, we construct a weak learner (we call it as base learner) for the target program. Finally, based on both the predictors of source domain programs and the base learner of the target program, a nonlinear meta model is constructed for the target program as shown in the bottom of the figure. That is, through assembling the source domain program predictors and the base learner of the target program, the meta learner achieves the knowledge transfer from the source domain programs to the target program, thereby training a better predictive model for the target program with very few simulations. Next, we will introduce the two major phases of our model in detail in the following sections.

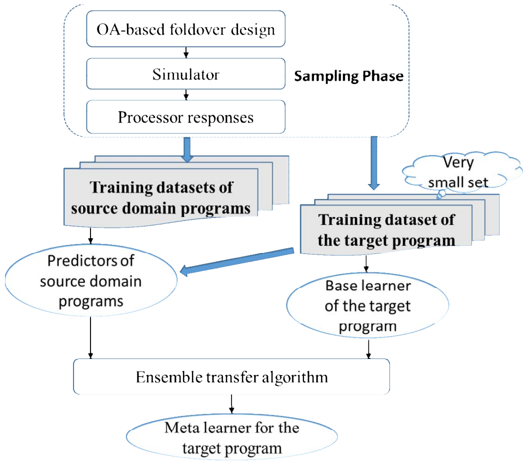

OA-based foldover design
Simulator
Sampling Phase
Processor responses
Very small set
Training datasets of source domain programs
Training dataset of the target program
Predictors of source domain programs
Base learner of the target program
Ensemble transfer algorithm
Meta learner for the target program

Fig. 1. Framework of TrEE

A. OA based Foldover Design

An efficient design of experiments (DoE) is critical to investigate the relationship between the parameters and the responses, and the sampling strategy is the key of DoE. A good sampling strategy should be able to make the design points evenly distributed throughout the design space. In other words, the design points should be representative. Li et al. [8] proposed to use orthogonal array (OA) in orthogonal design to sample the initial training data set, which achieved better results than random sampling. Each row of parameter combination in OA represents a configuration design needed for simulation. However, a major limitation of OA is that the sample set size is fixed and usually very small, which is insufficient to help train a very precise predictive model. Due to such a limitation of OA, the processor architects cannot flexibly select any number of configurations for simulation. To address this issue, the paper proposes a new foldover design method to improve the flexibility of traditional OA sampling.

To elaborate the foldover design, we firstly give a brief introduction on orthogonal design. Orthogonal design conducts a minimum amount of representative experiments according to OA to cover the entire design space and investigate how each parameter affects responses of the experiments. The OA can be derived from deterministic algorithms [15] 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 relationship between parameters and responses. However, for OA only 8 simulations are needed, and it can be denoted by $L(8;4,2,2,2,2)$ or $L_8(4^1 \times 2^4)$, as shown in Table I(a). 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 3, parameter B at 1, parameter C and D both at 0, and parameter E at 1. OA follows the balance principle with the following two properties. 1) Each level of each parameter occurs the same number of times in each column. For example, level 0 and level 1 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 (0,0), (0,1), (1,0), (1,1), (2,0), (2,1), (3,0) and (3,1) occur and they occur once. The advantages of OA based sampling are that it has the properties of representation and orthogonality.

Many parameters are needed to consider in microprocessor DSE. For example, there are 11 parameters in the design space shown in Table II, and 4 parameters with 8 levels, 4 parameters with 4 levels, 3 parameters with 2 levels. The corresponding OA is $L_{64}(8^4 \times 4^4 \times 2^3)$, i.e. 64 representative design points. However, 64 design points as the training set for the huge design space are insufficient for constructing an accurate predictive model. Thus, in order to flexibly select an arbitrary number of representative and informative samples in the design space, the idea is that we expand the size of OA and keep the balance principle of the array at the same time. To this aim, we propose to use foldover design to improve the OA sampling.

Foldover is a classic technique of DoE in statistics. It reverses or transforms the values (levels) assigned to one or more factors in the initial design (OA here) for running additional experiments, providing more data on potential combinations of factors [16]. If all the levels of factors in the initial OA design are transformed, it is called a full foldover design. Due to the good geometric symmetry structure and statistical property, foldover design has been widely used in practice [16]. For example, Georgiou et al. [17] proposed a foldover OA method for computer experiments.

Motivated by the above methods, we consider to combine the uniformity of OA and the advantages of foldover design to construct more representative design configurations for simulation. Thus, we propose to use full foldover design to fold the OA in order to expand the sample size.

Let $d \in L(n;q_1,\ldots q_j,\ldots q_s)$ be an orthogonal array, n is the number of configurations, s is the number of parameters, $0,1,\ldots(q_j-1)$ is the levels of the *i*-th column (parameter). The design matrix d is $X=(\mathbf{x}^1,\ldots\mathbf{x}^j,\ldots\mathbf{x}^s)$, $\mathbf{x}^j$ is the *j*-th column of X, $j=1,\ldots s$. Suppose $\Gamma=\{\gamma=(\gamma_1,\ldots,\gamma_s)|\gamma_1=0,1,\ldots(q_1-1),\ \cdots,$ $\gamma_s=0,1,\ldots(q_s-1)\}$ is the foldover plan space, and then we define $\forall\gamma=(\gamma_1,\ldots,\gamma_s)\in\Gamma$ as a possible foldover plan for the orthogonal array d. And the *j*-th parameter column of d is mapped to $\mathbf{x}^j(\gamma_j)=(\mathbf{x}^j\oplus\gamma_j)=(X_{1j}+\gamma_j,\ldots,X_{nj}+\gamma_j)'(\bmod\ q_j)$. That is, the levels of the *j*-th parameters in OA firstly add a number γ_j and then take a modular operation on q_j. For initial design d, different new matrix are given by different foldover plans γ.

We take the OA shown in Table I(a) as an example to elaborate the OA based foldover design. The OA can be expressed as $L(8;4,2,2,2,2)$, here $n=8$, $s=5$, $q_1=4$, and $q_2,q_3,q_4,q_5=2$. We set the foldover plan $\gamma=(3,1,1,1,0)$, and then the OA is mapped to Table I(b). For example, the level values of the parameter A first add 3, and then modulo 4. As a result, the first column of Table I(a) $(0,0,1,1,2,2,3,3)^T$ are transformed to first column of Table I(b) $(3,3,0,0,1,1,2,2)^T$, which forms a new parameter column.

The combinational table (jointed table) of the initial OA and the new arrays transformed by foldover design is taken as the sampling array. Processor architects can sample the configurations from the combinational table for simulation. According to the particular requirements of training set size, processor architects could determine how many rounds of foldover plans are needed. Due to the geometric symmetry of foldover design, the new array created by folding OA is also an OA. Thus, the design points in the combinational table are still evenly distributed in the design space, maintaining the properties of orthogonality and representativeness. Therefore, the OA based foldover design not only keeps the advantages of OA sampling, but also can flexibly expand the sampling set size, thereby more effectively captures the relationship between parameters and the experiment responses.

TABLE I. ORTHOGONAL ARRAY AND FOLDOVER DESIGN

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

No.	A	B	C	D	E
1	**0**	0	0	0	0
2	**0**	1	1	1	1
3	**1**	0	0	1	1
4	**1**	1	1	0	0
5	**2**	0	1	0	1
6	**2**	1	0	1	0
7	**3**	0	1	1	0
8	**3**	1	0	0	1

(b) Foldover design

No.	A	B	C	D	E
1	**3**	1	1	1	0
2	**3**	0	0	0	1
3	**0**	1	1	0	1
4	**0**	0	0	1	0
5	**1**	1	0	1	1
6	**1**	0	1	0	0
7	**2**	1	0	0	0
8	**2**	0	1	1	1

B. Ensemble Transfer Learning for DSE

In this section, we introduce how to explore ensemble transfer learning among programs for DSE. Previous cross-program DSE methods that make use of transfer learning techniques can be fit into model-driven transfer learning which makes use of models trained on the source domain programs, and sample-driven transfer learning which uses the source domain training samples directly to enrich the target domain data. Sample-driven transfer works well when the source domain and target domain programs are similar. However, in real scenarios the programs can vary significantly, making sample-driven transfer methods infeasible to use. Model-driven transfer learning algorithm mainly explore the relationship between models rather than samples, and thus is much more robust without requiring two programs for knowledge transfer very similar.

Pardoe and Stone proposed a model driven transfer learning algorithm: Transfer Stacking [14], in which a meta-level model combines multiple low-level models including a single base learner trained on the target domain data and multiple predictors previously trained on source data. It performs linear regression to find a linear combination of the low-level models that best fit the target domain data. Inspired by Transfer Stacking, we propose a new ensemble transfer learning algorithm for cross-program DSE, which improves the transfer stacking algorithm by incorporating nonlinear meta-level learner and feature augmentation technique.

The main idea of our model TrEE is that, through considering the prediction results of source program predictors on the samples of the target program as important references and features for constructing the final meta model. The final meta model can capture the nonlinear relationship among programs. As illustrated in Fig. 2, similar to a typical machine learning task, the ensemble transfer learning model in TrEE contains two phases: training phase and predicting phase.

In the training phase, the training sets $T_1, T_2, \ldots, T_S$ of the source domain programs are trained for constructing the predictors of source domain programs $h_1, h_2, \ldots, h_S$ respectively. Meanwhile, a base weak learner h' is constructed based on the training set T_{target} of the target program. Here, we call the source program predictors and the base learner of the target program as the low-level learners for brevity. We take the prediction results of low-level learners as the features for the meta-level model. Here, different with Transfer Stacking, the configurations are also considered as the feature of the meta learner. The idea is that there are some common features that are useful to predict responses for all the programs, and the learner can find such features and effectively transfer knowledge among different programs through reusing such features across all the programs. This is also called feature augmentation [18].

Note that the base learner and the meta-level learner are both constructed based on the training set of the target program. The difference is that base learner only utilizes configurations as features, while the instance features of the meta learner contain not only the configurations, but also the responses of both source domain program predictors and base learner.

At the predicting phase, we use the meta learner to predict the performance responses of the new configurations that has not been simulated for the target program. Specifically, the new configurations, the prediction results of the source domain program predictors and the base learner are considered as the feature inputs for the meta-learner. Then the meta learner gives a response prediction based on these features.

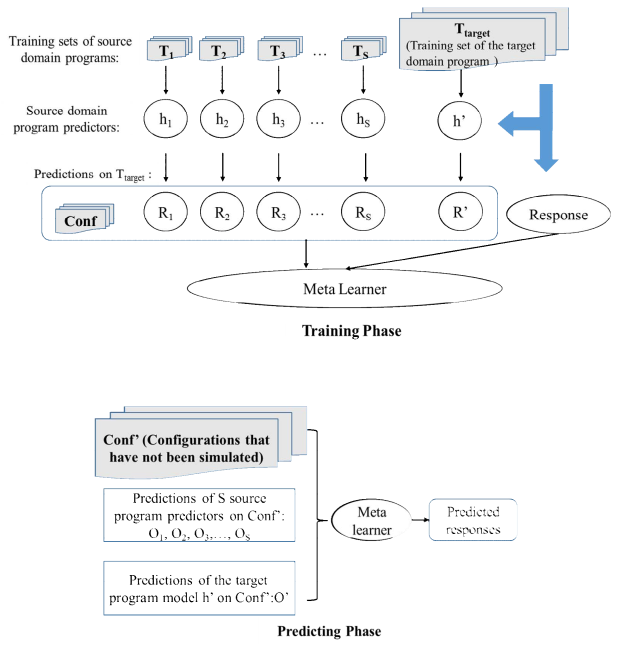

Training sets of source domain programs:
T1
T2
T3
…
TS
Ttarget (Training set of the target domain program)
Source domain program predictors:
h1
h2
h3
…
hS
h'
Predictions on Ttarget :
Conf
R1
R2
R3
…
RS
R'
Response
Meta Learner
Training Phase
Conf' (Configurations that have not been simulated)
Predictions of S source program predictors on Conf': O1, O2, O3,…, OS
Predictions of the target program model h' on Conf':O'
Meta learner
Predicted responses
Predicting Phase

Fig. 2. The framework of the proposed TrEE model

Algorithm 1 presents the pseudo-code of the TrEE model. In the beginning, based on the training sets $T_1, T_2, \ldots, T_S$ of the source domain programs, the predictors are constructed as $h_t = \mathcal{L}(T_t)$. For the low-level models $\mathcal{L}$, AdaBoost is used to boost ANN to build the source program predictors and the base learner due to its promising prediction performance in DSE [8]. The base learner of the target program h' is also constructed using AdaBoost.

In order to avoid overfitting, F fold cross validation is performed in step 3 for the base learner so that the output for each sample in T_{target} is obtained when it is out-of-sample, i.e., the output $R_i^{'}$ equals to the output of the base learner for the fold where sample $\mathbf{x}_i$ is in the validation set. In step 4 and step 5, the meta-level model h_{meta} is constructed based on the dataset generated by catenating the configuration and the prediction results of the low-level models. Finally SVM is employed as the meta-learner algorithm. Based on the meta-level training samples, we perform SVM to find a nonlinear combination of the source program predictors and the base learner that fits the data of the target program.

TrEE does not require a high similarity between programs, and could transfer the knowledge of the source domain programs via exploring the model-level nonlinear relationship based on the diverse performance responses on the same training samples. For parameter setting in the algorithm, we set the number of boosting iterations of AdaBoost as 10, and the cross validation as 10 folds.

Algorithm 1: Ensemble transfer learning model for DSE

```
Input: $T_{\text{target}}=\{(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),\ldots,(\mathbf{x}_m,y_m)\}$ : training set of the target program with $m$ samples; $T_1,T_2,\ldots,T_S$ : training datasets of the $S$ source domain programs; $\mathcal{L}$ : low-level learners algorithm (AdaBoost); $F$ : the number of folds for cross validation; $h_t$ : source domain program model; $h'$ : target domain program model; $M$ : meta-model algorithm SVM; $h_{meta}$ : meta-model
1.//Training $S$ source domain program predictors $h_t$ via the algorithm $\mathcal{L}$ on the data sets $T_t$
    for $t=1,2,\cdots,S$ do
        $h_t=\mathcal{L}(T_t)$
    end for
2.//Call the algorithm $\mathcal{L}$ with the full training set $T_{\text{target}}$ and get the base learner of the target program $h'$
    $h'=\mathcal{L}(T_{\text{target}})$
3. Perform $F$ fold cross validation on $T_{\text{target}}$ using algorithm $\mathcal{L}$ . For $1\le i\le m$ , let $R_i^{'}$ equal the output of the learned model for the fold where sample $\mathbf{x}_i$ is in the validation set.
4.//Generate the training set for the meta-model
    $D'=\phi$
    for each $\mathbf{x}_i\in T_{\text{target}}$ do
      for $t=1,2,\cdots,S$ do
          $R_{i,t}=h_t(\mathbf{x}_i)$ //the output of the source domain program predictors
      end for
      $D'=D'\cup\left((\mathbf{x}_i,R_{i,1},R_{i,2},\cdots R_{i,S},R_i^{'}),y_i\right)$
    end for
5.//Training the meta-model $h_{meta}$ via the algorithm $M$ on the data set $D'$
    $h_{meta}=M(D')$
Output $H(\mathbf{x})=h_{meta}\left(\mathbf{x},h_1(\mathbf{x}),h_2(\mathbf{x}),\cdots h_S(\mathbf{x}),h'(\mathbf{x})\right)$
```

III. Experiment Setup

A. Simulator and Benchmarks

Using the state-of-art cycle-accurate simulator GEM5 [19], we conduct our simulations on 27 benchmarks from the SPEC CPU 2006 suite. Aided by SimPoint [20], 100 million instructions of each benchmark are simulated to collect statistics. IPC (Instruction-per-Cycle) is used as the performance metric for processor response.

B. The Explored Design Space

In this paper, we investigate 11 important parameters for processor performance. That is, we choose to vary the 11 parameters, while other parameters for describing the architectural components are fixed. For example, we fix CPU clock as 2GHz, branch predictor as tournament, L1 Dcache associativity as 2-way, L1 Icache associativity as 2-way and L2 Cache associativity as 8-way, etc. The parameters and the corresponding design space is shown in Table II. For the design space, 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. We fold the OA over with 7 different foldover plans, i.e., 7 rounds of foldover and, then join them together forming a sampling table with 512 ($= 64\times 7+64$) architectural design configurations. We also randomly select 2000 configurations in the design space as the dataset for evaluation. We simulate them for each benchmark. From the training dataset with 512 samples created by foldover design, we select the specified number of training data for the source programs and the target program in TrEE in the following experiments. If the required training set size is smaller than 64, we directly randomly sample it from the OA dataset. Otherwise we select it from one or several foldover arrays according to the different dataset sizes.

TABLE II. 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**

C. Evaluation Methodology

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

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 matches the shape of the real value.

D. Competitive Methods

We compare TrEE with the following baselines.

(1) PSM is a state-of-art program-specific predictive model based on artificial neural networks [1]. Following the setting in [1], the ANN adopts one 16-unit hidden layer, a learning rate of 0.001.

(2) EAC is an empirical architecture-centric approach based on a simple linear model combining several individual models of previous programs [10,11].

(3) TrDSE is a program cluster aided transfer learning for DSE approach [13]. TrDSE firstly clusters the programs and then using transfer learning algorithm to transfer knowledge from similar programs for the new program.

IV. Evaluation

A. Evaluation of Sampling Strategy

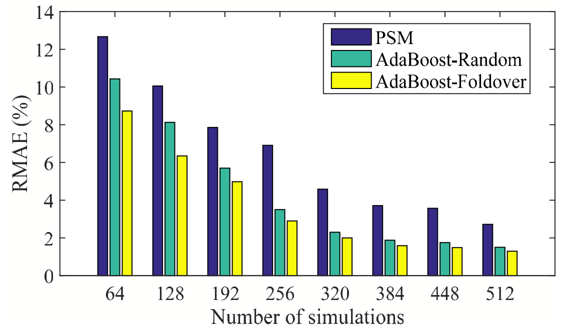


Fig. 3. Average RMAE of PSM and AdaBoost with random sampling vs. foldover sampling with various numbers of simulations

To evaluate the effectiveness of the proposed foldover sampling strategy, we compare the accuracies of PSM (program-specific predictive model based ANN with random-sampling), AdaBoost with random sampling and foldover sampling respectively when varying the number of simulations. We conduct 5 round experiments with the two methods with random sampling, and then calculate their average prediction errors. Fig. 3 shows the results on all the 27 benchmarks. One can see that for the AdaBoost method, foldover sampling outperforms random sampling significantly when the simulation size is less than 256. Foldover sampling reduces prediction error from 10.4% (random sampling) to 8.7% when the simulation size is 64, and from 8.1% (random sampling) to 6.3% when the simulation size is 128. The prediction error of foldover sampling with 192 simulations (4.9%) is close to that of random sampling (4.6%) with 320 samples, which means that compared to random sampling, foldover sampling reduces nearly 40% simulations while achieves similar prediction performance. When the simulation size becomes large enough like 320, the prediction performance of foldover sampling even outperforms random sampling. In addition, one also can see that AdaBoost significantly outperform PSM. In order to achieve a prediction error of 3.7%, AdaBoost with foldover sampling needs less than 256 simulations, while PSM needs 384 simulations. That is, the foldover sampling based AdaBoost reduces 33% simulations of PSM. This experiment shows that foldover sampling is much more effective to select representative samples than random sampling. It also shows AdaBoost is a more accurate prediction model than PSM. Thus in the following experiments we use foldover sampling to select samples, and use AdaBoost as the prediction model.

B. Evaluation of Ensemble Transfer Learning

The goal of TrEE is to reduce both the off-line training cost (i.e., the training samples of the source domain programs) and the simulation size of the target program; meanwhile construct a predictive model with precise prediction. The off-line training cost depends on the following two factors: the number of source domain programs and the available training set size of each source domain program. In order to evaluate the effectiveness of TrEE, we conduct experiments to test the performance with different settings of the three factors separately. Unless otherwise stated, all the training sets in TrEE are sampled based on the foldover sampling and the source domain programs are selected randomly.

1) Varying the Simulation Size of Target Program

To evaluate the effectiveness of TrEE with various number of simulations (training dataset size) of the target program, we compare the RMAE of TrEE and PSM by increasing the number of simulations from 10 to 500. In this experiment, we randomly select 3 programs and 128 training samples for each program as the source domain programs for each target program.

Fig. 4 shows the average RMAE over all the programs of the two methods. With the increase of simulation number, the prediction performance of the two approaches both improves. One can see that TrEE reduces the prediction error from 21.7% of PSM to 10.6% with only 10 simulations. When the number of simulations increases to 20, RMAE of TrEE drops to 8.1%, while the number achieved by PSM is 14.5%, which shows a significant improvement. By transferring knowledge from source domain programs, only 40 simulations of the target program are needed for TrEE to achieve a prediction error of 5.5%, while PSM needs 300 simulations to achieve a similar performance with the prediction error of 5.96%.

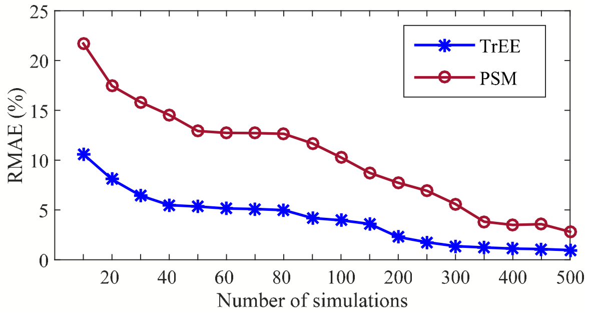
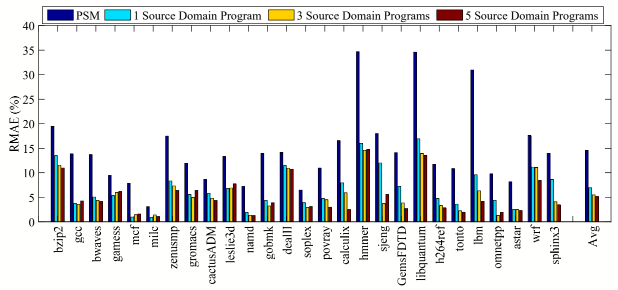


Fig. 4. Average RMAE of PSM and TrEE by varying the simulation number of the target domain program

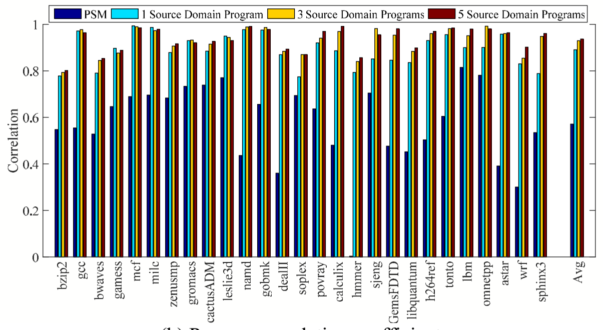
2) Varying the Number of Source Domain Programs

In this section, we evaluate the performance of TrEE by varying the number of the source domain program. The training set size of the target program, i.e., the simulation number of the target program, is set to 40. The training set size of each source domain program is also fixed at 128. Fig. 5 shows the RMAE and Pearson correlation coefficient of TrEE with the number of source domain programs as 1, 3 and 5 respectively. From Fig. 5(a) one can see that TrEE significantly outperforms PSM, and it reduces the prediction error 14.5% of PSM to 6.9% on average when using a source domain program. For some benchmarks that are hard to predict as they are dissimilar to other programs such as *libquantum* and *lbm,* the improvement of TrEE is more significant, which implies TrEE can also effectively transfer knowledge among dissimilar programs. In Fig. 5(b), comparing with the Pearson correlation coefficient 0.57 achieved by PSM, TrEE dramatically improves it to 0.93 with three source domain programs on average. For benchmark *gcc*, the improvement is even more significant, from 0.55 by PSM to 0.97 by TrEE with only one source domain program.

One can see that with the increase of source domain programs number, the prediction accuracy of the ensemble transfer learning model improves. When using 3 source domain programs, the average prediction error of TrEE is reduced to 5.5%, comparing with 6.9% when only one source domain program is used. When using 5 source domain program, the average RMAE is further reduced to 5.17%.



(a) RMAE



(b) Pearson correlation coefficient

Fig. 5. RMAE and Pearson correlation coefficient of PSM and TrEE with 1,3,5 source domain programs and 40 training samples of the target program

3) Robustness Comparison

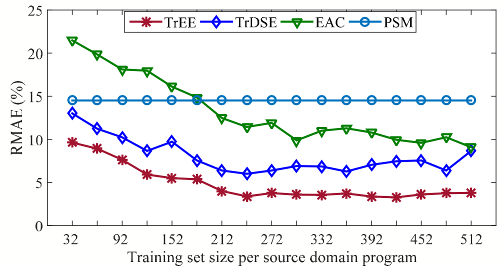


Fig. 6. Robustness comparison of the four methods

In this section we study the robustness of TrEE by transferring knowledge among programs that are not very similar. We compare it with two state-of-art cross-program predicting model EAC proposed by Dubach el at. [10, 11] and TrDSE proposed by Li el at. [13]. PSM is also taken as the baseline.

To be fair, the three cross-program approaches all use the same number of training samples. Here, the number of source domain programs is set as 3, and the training data size of the target program is set as 32. To compare the off line training cost, we compare the prediction accuracy of different methods with different sizes of training set for each source domain program. We increase the training set size of the 3 source domain programs from 32 to 512 with an increase step of 30. Without considering the similarity between programs, we just randomly select three benchmarks as the source programs, *bzip2, leslie3d* and *hmmer*. Fig. 6 shows the average prediction accuracy over all the other 24 programs of the four approaches. As the training data size of the target program is fixed as 32, the prediction error of PSM is constant and thus the curve in the figure is a line.

From Fig. 6 one can see that TrEE consistently outperforms the other methods. And EAC is remarkably inferior to TrEE. It is even worse than PSM when the training samples per source domain program is less than 182. When the training set size is larger than 182 per source program, EAC just starts to become better than PSM. With only 32 samples per source domain program, TrEE reduces the prediction error from 14.5% of PSM to 9.6%. When the training set size per source domain program increases to 212, the prediction error dramatically reduces to 4%, a very promising result. While the prediction error of TrDSE and EAC are reduced to 6.4% and 12.5% respectively. When the training set size per source domain program increases larger than 300, the prediction accuracy of TrEE grows slowly. That is because based on the limit training set of the target program, less more available knowledge of the source domain programs could be transferred for the target program.

Compared with EAC, TrEE shows higher robustness. The RMAE curve of TrEE is less volatile and consistently lower than that of EAC, showing a more powerful knowledge transfer ability and better robustness. This is mainly because TrEE is a model driven transfer learning approach that can capture the nonlinear relationship between the programs. While EAC is a linear model, it is difficult for EAC to fit programs for various filed applications. TrDSE is an instance driven transfer learning approach whose performance is largely affected by the instance similarity of the source domain program to the target domain. The critical drawback of TrDSE and EAC is that they overwhelmingly depend on the similarity among programs.

TABLE III. **SIMULATION AND MODEL TRAINING COST OF TREE AND PSM**

Simulation and model training cost	**PSM**	**TrEE**
The simulation size of the source domain program	0	212
The simulation size of the target domain program	350	32
Average training time per program model	≈ 17 seconds	≈ 43.7 seconds
Total number of simulations	$350 \times 27 = 9450$	$212 \times 3 + 32 \times 24 = 1404$
Average DSE time for each program	≈ 58.3 hours	≈ 8.7 hours

Here we also give an example to show the efficiency of the proposed TrEE. According to the experiment in Section IV.B.(1), PSM needs about 350 training samples to achieve the similar prediction error of 4% to TrEE with 212 simulations of the source domain program and 32 simulations of the target domain program. We compare the simulation cost and model training cost of the two methods in such an experiment setting as shown in Table III. For the 27 benchmarks tested in our experiments, 3 benchmarks are considered as the source domain programs and the other 24 benchmarks are the target programs. Thus the total number of simulations for TrEE includes the number of simulations on both source domain programs and the target programs. Thus, TrEE needs 1404 simulations for DSE on the 27 benchmarks, while PSM need 9450 simulations. TrEE only needs less than one-sixth simulations of PSM. In general, a simulation for 100 million instructions requires an execution time of about 10 minutes on average. The predictive model training time of the two methods are no more than one minute. TrEE needs about 234 hours in total for DSE on all the 27 benchmarks, and about 8.7 hours for each program on average. While PSM needs about 1575 hours in total for DSE and 58.3 hours for each program on average.

4) Better Configuration

In general, the actual optimal configuration cannot be obtained unless simulating the entire design space consisting of 8 million design configurations, which is infeasible. A compromise is made by comparing the promising configuration deduced by TrEE with the configurations deduced by the other three approaches. To be specific, we randomly sample 3000 configurations from the entire design space, and use these approaches to predict the performance responses on these configurations. Then we simulate the promising configurations found by these approaches and compare the corresponding processor performance responses (IPC) directly. We take benchmark *games* as an illustrative example. Table IV shows the promising design configurations found by the four methods, the predicted IPC and the actual simulation response. One can see that, the configuration found by TrEE is the best comparing with those of the other three methods, and the predicted response of TrEE is very close to the actual performance of this configuration, with only 0.6% prediction error. The actual performance of the configuration found by TrDSE is better than that of EAC and PSM, and the prediction error is also low. Although the configuration found by EAC is better than that of PSM, its prediction error is higher. Therefore, TrEE is more effective for exploring the better configuration than the other methods.

TABLE IV. **THE PROMISING DESIGN CONFIGURATIONS FOUND BY THE DIFFERENT METHODS (*GAMESS*)**

Parameters	**TrEE**	**TrDSE**	**EAC**	**PSM**
ROB size	176	144	112	96
Width	8	8	8	8
L1 DCache	128kB	32kB	64kB	32kB
L1 Icache	64kB	32kB	32kB	128kB
L2 Ucache	512kB	1MB	2MB	1MB
MSHR	4	4	4	4
Register File	128	120	128	136
IQ	32	80	64	32
LSQ	96	96	96	128
ALUs	4	4	4	4
FPUs	2	2	4	2
Actual	1.69	1.66	1.64	1.63
Predicted	1.68	1.68	1.89	1.87
Error	**0.6%**	1.2%	15.2%	14.7%

V. RELATED WORK

Recently, a number of statistical and machine learning techniques have been applied in architectural design. To efficiently explore processor design space, İpek et al. [1] proposed to construct regression model based on ANNs to predict the performance of architectural configurations. Joshep et al. [4, 5] introduced to build linear regression models for modeling the relationship between the architectural parameters and the performance. To accurately predict the performance of microprocessors, Joshep et al. [5] later used Radial Basis Function (RBF) networks to construct a nonlinear model. Lee et al. [6, 7] introduced spline-based regression model and composable performance regression model to predict IPC and power consumption for multiprocessor design. To further reduce the simulation cost, Guo et al. [2, 3] employed semi-supervised learning and active learning to build a predictive regression model. Later, Guo et al. [21] proposed a framework which built a metamodel above several distinct regression models to construct a robust predictive model. To improve the prediction model with less simulations, Li et al. [8] proposed to apply orthogonal array sampling technique to sample the training dataset and then construct an AdaBoost based active learning model. In addition, some studies also tried to use analytic models to cope with DSE. Ying et al. [22] uses a greedy policy to perform on-line design space search for reconfigurable machine learning processors. Palermo et al. [23] proposed to use Response Surface Modeling (RSM) techniques to refine the design space exploration. Xydis and Palermo et al. [24] proposed a spectral-aware pareto iterative refinement for supervised high-level synthesis.

However, all of the above works studied constructing the program-specific models, which cannot transfer knowledge from previous programs to a new program. To address this issue, some previous works investigated how to construct cross-program DSE model. Khan et al. [9] and Dubach et al. [10, 11] both proposed simple cross-program meta models above previously trained predictors. To build a general predictive model for all programs, Guo et al. [12] took the inherent program characteristics as a part of input of the model. Li et al. [13] introduced a program cluster aided transfer learning framework for transferring similar training samples from the similar previous programs to the new program. Although these cross-program DSE methods reduced some simulation cost for the new program compare with program-specific models, they may not effective among dissimilar programs. Another shortcoming is that most of them ignored the sampling strategy for predictive model.

VI. Conclusion

This paper proposed an efficient and robust DSE framework TrEE which combines foldover based sampling and ensemble transfer learning for cross-program DSE. The proposed foldover sampling expands classical OA sampling technique, and breaks the limit of the small and fixed sample number of OA sampling. On this basis, we proposed an new ensemble transfer learning model TrEE, which could transfer knowledge from other previous programs to the new program. One critical advantage of TrEE is its robustness since it performs well in knowledge transfer for DSE even if source domain programs are dissimilar to the target program. Empirical evaluation demonstrates TrEE outperforms state-of-art approaches in efficiency, effectiveness, and robustness in DSE.

Acknowledgment

This work is supported in part by the NSFC under grants 61602237 and 61504153, and the Director's Project Fund of Key Laboratory of Trustworthy Distributed Computing and Service (BUPT), Ministry of Education (Grant No. 2017KF03).

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

[3] 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.

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

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

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

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

[8] D. Li, S. Yao, Y. Liu, et al., “Efficient Design Space Exploration via Statistical Sampling and AdaBoost Learning,” in DAC’ 2016.

[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] D. Li, S. W, S. Yao, et al., “Efficient Design Space Exploration by Knowledge Transfer,” in CODES+ISSS’ 2016

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

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

[16] K.T. Fang, D. K. Lin and H. Qin, “A note on optimal foldover design,” Statistics & Probability Letters, vol. 62, no. 3, pp. 245-250, 2003.

[17] S.D. Georgiou, “Orthogonal designs for computer experiments,” Journal of Statistical Planning and Inference, vol. 141, no.4, pp. 1519-1525, 2011.

[18] H. Daumé, III and D. Marcu, “Domain adaptation for statistical classifiers,” Journal of Artificial Intelligence Research,” vol. 26, pp. 101-126, 2006.

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

[20] 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.

[21] Q. Guo, T. Chen, Z. H. Zhou, O. Temam, L. Li, D. Qian, and Y. Chen, “Robust design space modeling,” ACM Transactions on Design Automation of Electronic Systems, vol. 20, no. 2, pp. 1–22, 2015.

[22] Y. Wang, H. Li, and X. Li, "Real-Time Meets Approximate Computing: An Elastic CNN Inference Accelerator with Adaptive Trade-off between QoS and QoR," In Proc. of DAC, 2017.

[23] G. Palermo, C. Silvano, and V. Zaccaria, “ReSPIR: a response surface-based pareto iterative refinement for application-specific design space exploration,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 28, no.12, pp. 1816-1829, 2009.

[24] S. Xydis, G. Palermo, V. Zaccaria, and C. Silvano, “SPIRIT: Spectral-Aware pareto iterative refinement optimization for supervised high-level synthesis,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 34, no. 1, pp. 155-159, 2015.