Effective and Efficient Microprocessor Design Space Exploration Using Unlabeled Design Configurations*

Qi Guo[1,3,4], Tianshi Chen[1,3], Yunji Chen[1,3], Zhi-Hua Zhou[2], Weiwu Hu[1,3], and Zhiwei Xu[1]

[1] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[2] National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China
[3] Loongson Technologies Corporation Limited, Beijing 100190, China
[4] Graduate University of Chinese Academy of Sciences, Beijing 100149, China
{guoqi, chentianshi, cyj, hww, zxu}@ict.ac.cn, zhouzh@lamda.nju.edu.cn

Abstract

During the design of a microprocessor, Design Space Exploration (DSE) is a critical step which determines the appropriate design configuration of the microprocessor. In the computer architecture community, supervised learning techniques have been applied to DSE to build models for predicting the qualities of design configurations. For supervised learning, however, considerable simulation costs are required for attaining the labeled design configurations. Given limited resources, it is difficult to achieve high accuracy. In this paper, inspired by recent advances in semi-supervised learning, we propose the COMT approach which can exploit unlabeled design configurations to improve the models. In addition to an improved predictive accuracy, COMT is able to guide the design of microprocessors, owing to the use of comprehensible model trees. Empirical study demonstrates that COMT significantly outperforms state-of-the-art DSE technique through reducing mean squared error by 30% to 84%, and thus, promising architectures can be attained more efficiently.

1 Introduction

When designing a microprocessor, the first and probably the most important step is to decide appropriate design configurations satisfying different performance/power/temperature/reliability constraints, which is called as *Design Space Exploration* (DSE). It has become a great challenge to computer architects, since the size of design space grows exponentially with the number of interactive design parameters (e.g., cache size, queue size, issue width, etc.), and the simulation required by evaluating the quality of each configuration of design parameters is quite time-consuming. Moreover, the difficulty of DSE task is further exacerbated by the increasing amount and complexity of computer workloads with significantly different characteristics.

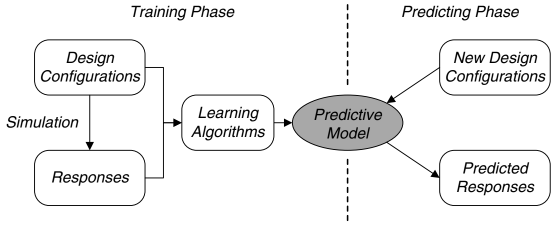

Figure 1: A framework of predictive modeling for DSE.

Traditionally, architects employed large-scale cycle-accurate architectural simulation on representative benchmarks to explore the design space. However, time-consuming simulation makes it intractable to explore the entire design space. For instance, during the design of Godson-3, which is a 16-core chip-multiprocessor (CMP) with a reconfigurable architecture [Hu *et al.*, 2009], it takes several weeks to simulate only one design configuration on SPEC CPU2000 benchmark suite. To reduce the simulation costs, several fast simulation approaches were proposed to reduce the number of simulated instructions with respect to each design configuration [Hamerly *et al.*, 2006; Genbrugge and Eeckhout, 2009; Joshi *et al.*, 2006]. However, the number of design configurations to simulate is still quite large. To reduce the number of simulated configurations and thus reduce the overall simulation costs, predictive modeling was proposed [Joseph *et al.*, 2006; Ïpek *et al.*, 2006; Lee *et al.*, 2008]. As illustrated in Figure 1, a predictive modeling approach contains two phases, i.e., training phase and predicting phase. In the training phase, some design configurations are simulated. Along with the corresponding responses (e.g., performance or energy response) obtained by simulations, these labeled design configurations are utilized to train a predictive model that characterizes the relationship between the design parameters and processor responses. In the predicting phase, such a predictive model is employed to predict the responses of *new* design configurations that are not involved in the training set. Since simulations are only required in the training phase, predictive modeling is relatively efficient in comparison with traditional approaches. However, considerable simulation costs are required to attain the labeled design configu-

*This work was partially supported by the CAS Frontier Research Project, the JiangsuSF (BK2008018), the NSFC (61073097, 61021062, 61003064, 60921002, 60736012), the 973 Program (2010CB327903, 2011CB302502, 2011CB302803) and the National S&T Major Project of China (2009ZX01028-002-003, 2009ZX01029-001-003, 2010ZX01036-001-002).

rations for supervised learning, which encumbers the models from achieving high prediction accuracies given limited computational resources and stringent design-to-market pressure. In fact, the design configurations that have not been simulated may also be effective in enhancing the prediction accuracy of a predictive model, which are completely overlooked by previous investigations on DSE. Furthermore, previous predictive modeling approaches often employ difficult-to-interpret learning models such as ANNs (Artificial Neural Networks), and the trained models are black-boxes that could not offer insights about how different design parameters affect the performance or energy of microprocessors.

To circumvent these deficiencies of previous techniques, in this paper, we propose the COMT (Co-Training Model Tree) approach for the challenging DSE problem. The key intuition is that similar architectural configurations would behave similarly, and thus, some unlabeled design configurations can be used to enhance the prediction accuracy. COMT works in co-training style [Blum and Mitchell, 1998], a representative of the disagreement-based semi-supervised learning paradigm [Zhou and Li, 2010], where two models label unlabeled data for each other. To enable the learned model to be comprehensible, COMT employs model trees [Quinlan, 1992] that are trained by M5P [Wang and Witten, 1997], an optimized algorithm for constructing model trees. COMT initializes two models using the labeled training set, and each model is then refined by using instances labeled by the other model. Here, a key issue is how to select appropriate unlabeled instances to label. For this, COMT considers the neighboring properties of training examples to estimate the labeling confidence of unlabeled instances. Experiments show that, given the same simulation budget to attain the labels of training design configurations, COMT can reduce by 30-84% mean squared error of the state-of-the-art DSE technique.

The rest of the paper proceeds as follows. Section 2 introduces some related work. Section 3 presents our COMT approach. Section 4 reports on empirical results. Section 5 concludes this paper.

2 Related Work

Design Space Exploration. Many investigations reduce the simulation costs for DSE by analyzing program characteristics. Hamerly *et al.* [2006] simulated only some representative program phases rather than the whole program. From the perspective of program, Joshi *et al.* [2006] found a reduced representative subset of programs based on inherent microarchitecture-independent characteristics by cluster analysis. Moreover, statistical simulation was employed to construct a synthesized shorter program to emulate the execution characteristics of the original program [Genbrugge and Eeckhout, 2009]. Unlike the above approaches, predictive modeling techniques reduce simulated design configurations by learning the relationship between design parameters and processor responses. Following the supervised learning framework, the above task was accomplished by linear regression model [Joseph *et al.*, 2006] or ANNs [İpek *et al.*, 2006], where ANNs are most widely used. However, since the usefulness of unlabeled design configurations is completely ignored, the above approaches suffer from either high simulation costs (for achieving high accuracies) or low prediction accuracy (given limited computational resources).

Semi-Supervised Learning. Semi-Supervised Learning (SSL) is a mainstream methodology for exploiting unlabeled data to improve the prediction accuracy. Generally, SSL can be classified into four categories [Zhou and Li, 2010], that is, generative methods [Fujino *et al.*, 2005], S3VMs (Semi-Supervised Support Vector Machines) [Xu and Schuurmans, 2005], graph-based methods [Zhu *et al.*, 2003], and disagreement-based methods [Blum and Mitchell, 1998; Zhou and Li, 2010]. Generative methods conduct maximum likelihood estimation to determine the parameters of models, where the labels of unlabeled data are treated as missing values. S3VMs usually utilize unlabeled data to adjust the decision boundary built from labeled examples. In graph-based methods, the SSL problem can be addressed by propagating the label information in a graph constructed from labeled and unlabeled data where each node corresponds to one instance. The key of disagreement-based methods is to generate multiple learners, let them collaborate to exploit unlabeled data, and maintain a disagreement among the base learners. The line of research started by Blum and Mitchell [1998]'s seminal work on *co-training*. Zhou and Li [2005] proposed a Semi-Supervised Regression (SSR) approach, COREG, which employs two kNN regressors to conduct the data labeling and the predictive confidence estimation. COREG utilizes kNN as the base regressor since it is easy to update and smoothly consistent with the manifold assumption of SSL. In COREG, the most confidently labeled example is determined as the one which makes the regressor most *consistent* with labeled data.

Comprehensibility. In DSE practice, learned predictive models with comprehensible results are very helpful for architects and systems designers. However, most strong learning systems, such as SVMs (Support Vector Machines), ANNs and ensembles, are complicated black-box models, and it is a great challenge to improve the comprehensibility. Currently, decision trees are regarded as one of the most comprehensible learning models, since the prediction process of decision trees is explicit [Zhou and Jiang, 2004]. Therefore, model tree, a variant of decision tree that is applicable to regression problems, is employed in COMT for attaining interpretable models.

3 Our Proposal

3.1 COMT

Let $L = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{|L|}, y_{|L|})\}$ be the labeled example set, where $\mathbf{x}_i$ is the i-th design configuration with d interested design parameters, and y_i is corresponding processor response, e.g., performance metric like IPC (Instruction-per-Cycle). Let U be the unlabeled data set, i.e., the set of design configurations without simulation.

The most critical issue in SSR is how to estimate the labeling confidence such that the most confident unlabeled instance can be selected to label. Following Zhou and Li [2005]'s approach, we consider that the error of the regressor on the labeled data set should decrease the most if the most confident unlabeled instance is labeled. Formally, for each unlabeled instance $\mathbf{x}_u$, the quality of $\mathbf{x}_u$ can be measured using a criterion as shown in Equation 1:

$$\Delta_u = \frac{1}{|L|} \sum\nolimits_{\mathbf{x}_i \in L} \Big((y_i - h(\mathbf{x}_i))^2 - \big(y_i - h'(\mathbf{x}_i)\big)^2 \Big), \qquad (1)$$

where h is the original regressor, and h' is the regressor refined with example $\mathbf{x}_u$ and its label. In other words, the instance with the best *mean squared error* (MSE) reduction on the labeled set will be selected to label. Such a criterion is adopted in COMT.

Another challenge during the design of COMT is that, repeatedly measuring the MSE of model tree on the entire labeled data set in each iteration is time-consuming. To address this problem, COMT utilizes the local information of constructed model tree to improve the update efficiency. Specifically, given an unlabeled instance which would be labeled by the original model tree, it should fit into a linear model that lies in a leaf of the tree. Thus, we approximately measure the labeling confidence of each unlabeled instance by computing only the MSE of its *siblings* locating in the same leaf of the tree, where siblings refer to those labeled examples. In this case, the Promising Labeled Example (PLE) of an unlabeled data set, denoted by $\widetilde{\mathbf{x}}$, is determined by maximizing the local error reduction (Δ_u) defined in Equation 2:

$$\Delta_{\mathbf{x}_u} = \sum\nolimits_{\mathbf{x}_i \in \Omega_u} \Big((y_i - m(\mathbf{x}_i))^2 - \big(y_i - m'(\mathbf{x}_i)\big)^2 \Big), \qquad (2)$$

where Ω_u is sibling set of $\mathbf{x}_u$ in the original tree, m is the original regressor, and m' is the regressor refined by $(\mathbf{x}_u, y_u)$, $y_u = m(\mathbf{x}_u)$.

Algorithm 1 presents the pseudo-code of COMT algorithm. As the first step, M5P algorithm is used to construct two diverse model trees (say, m_1 and m_2) via employing distinct parameters M_1 and M_2 respectively, where M_1 and M_2 determine the minimal number of examples in each leaf of m_1 and m_2, respectively [Wang and Witten, 1997]. Let L_1 and L_2 be the current labeled data sets of m_1 and m_2 respectively, and L_1 and L_2 are initialized by the same labeled data set L. In each iteration, COMT uses the PLE determined by m_1 to augment the labeled set L_2, and vice versa. After using the latest labeled sets to update two model trees, we should replenish the pool U' with unlabeled examples to size p for next iteration. Finally, we average the prediction on each updated model tree as the final prediction.

Notice that, in contrast to the original co-training [Blum and Mitchell, 1998], COMT does not work with two views. Similar to COREG [Zhou and Li, 2005] and other single-view disagreement-based SSL approaches, the validity of COMT can be theoretically justified by the large difference between the two base learners [Wang and Zhou, 2007].

Algorithm 1: Pseudo-code of COMT Algorithm

```
Data: $L$ : Set of labeled examples.
      $U$ : Pool of unlabeled instances.
      $p$ : Size of the pool with unlabeled instances.
      $t$ : Number of learning iterations.
      $M_1, M_2$ : Minimal number of examples in the leaf of
      model trees.
begin
    $L_1 \leftarrow L$; $L_2 \leftarrow L$;
    Create a pool $U'$ with $p$ unlabeled examples;
    Train model tree $m_1$ with labeled set $L_1$ by parameter $M_1$;
    Train model tree $m_2$ with labeled set $L_2$ by parameter $M_2$;
    for $l \leftarrow 1$ to $t$ do
        for $j \in \{1, 2\}$ do
            for each $\mathbf{x}_u \in U'$ do
                $\Omega_u \leftarrow Sibling(m_j, \mathbf{x}_u)$;
                $y_u \leftarrow m_j(\mathbf{x}_u)$;
                Obtain new model $m'_j$ by adding $(\mathbf{x}_u, y_u)$;
                $\Delta_{\mathbf{x}_u} \leftarrow \sum_{\mathbf{x}_i \in \Omega_u} ((y_i - m_j(\mathbf{x}_i))^2 - (y_i - m'_j(\mathbf{x}_i))^2)$;
            end
            if there exists a $\Delta_{\mathbf{x}_u} > 0$ then
                $\widetilde{\mathbf{x}}_j \leftarrow \arg\max_{\mathbf{x}_u \in U'} \Delta_{\mathbf{x}_u}$; $\widetilde{y}_j \leftarrow m_j(\widetilde{\mathbf{x}}_j)$;
                $\pi_j \leftarrow \{(\widetilde{\mathbf{x}}_j, \widetilde{y}_j)\}$; $U' \leftarrow U' - \{\widetilde{\mathbf{x}}_j\}$;
            end
            else
                $\pi_j \leftarrow \phi$;
            end
        end
        if $\pi_1$ is $\phi$ AND $\pi_2$ is $\phi$ then
            exit;
        end
        $L_1 \leftarrow L_1 \cup \pi_2$; $L_2 \leftarrow L_2 \cup \pi_1$;
        Update $m_1$, $m_2$ by new set $L_1$, $L_2$, respectively;
        Replenish pool $U'$ to size $p$ with randomly selected
        candidate unlabeled examples from $U$;
    end
    Output $m^*(\mathbf{x}) \leftarrow \frac{1}{2}(m_1(\mathbf{x}) + m_2(\mathbf{x}))$;
end
```

3.2 Comprehensible Models for DSE

In contrast to uninterpretable models such as ANNs, comprehensible predictive models obtained from COMT are very helpful for establishing the relationship between design parameters and processor responses. For example, according to the weights of different design parameters in the linear models associated with the leaves of a model tree, we can determine the key architectural parameters that significantly influence the processor responses. In addition to offering explicit information about design parameters, such comprehensible models can further assist computer architects to deduce promising architecture under some given design specifications.

In industry, the typical design specification could be "*given a certain power dissipation (e.g., 120 Watt) constraint, we should maximize the performance of a processor*". Without a comprehensible model which characterizes the detailed relationship between design parameters and processor responses, one may have to estimate the (predicted) responses (e.g., performance) of all design configurations in a brute-force way to search for the most promising one. Unlike conventional approaches that employ uninterpretable models such as ANNs, COMT can offer linear models in the leaves of its model trees. These linear models can be tackled with Linear Programming (LP) techniques, which enables rapid deductions of promising architectures.

To be specific, by executing the entire co-training process twice, we construct two predictive models for performance and power, respectively. After that, we obtain two models consisting of several linear models with decision rules. For each *performance-power linear model pair*, denoted by $(I(LM_i), W(LM_j))$, we obtain a LP problem as,

$$\text{Maximize} \qquad I(LM_i),$$
$$s.t. \begin{cases} W(LM_j) \leq 120 \\ \text{contraints extracted from decision rules} \\ \text{default constraints on design parameters.} \end{cases}$$

In this LP, there are two categories of constraints on design parameters. The first category of constraints are extracted from the decision rules associated with the linear models $I(LM_i)$ and $W(LM_j)$ of the model trees. The other category includes default constraints on design parameters, which confine the ranges of parameters. After solving all such LPs obtained from performance-power linear model pairs, we effectively and efficiently gain the promising architecture. Apparently, there are mn LPs for architects to solve, where m and n are the numbers of linear models on the performance and power tree, respectively. Since the number of variables in each LP is only the number of interested design parameters (usually tens of parameters), the worst-case computation time for solving this problem can be neglected compared with tremendous simulation costs.

4 Empirical Study

4.1 Benchmarks for Processor Evaluation

The most common way to evaluate the performance of a processor during DSE is to measure the execution time of many popular benchmarks on cycle-accurate simulators and here we employ renowned Simplescalar [Austin *et al.*, 2002] as the prototype simulator of a modern superscalar microprocessor. Regarding benchmarks, the most successful standardized benchmark suite covering various application fields has been created by SPEC (Standard Performance Evaluation Corporation) since the late 1980s [Hennessy and Patterson, 2003]. Since SPEC benchmarks are real programs with slightly modification for portability, they are widely utilized by architects, researchers, and computer vendors for performance evaluation. For example, vendors of desktop computers and servers periodically submit the performance results measured by SPEC benchmarks to *www.spec.org* to provide fair comparisons with other machine products.

SPEC CPU2000 is the fourth generation benchmark suite of SPEC series, which consists of a set of 11 integer benchmarks (CINT2000) and 14 floating-point benchmarks (CFP2000). SPEC CPU2000 aims at providing fair evaluations of desktop *general-purpose* processors. To validate the effectiveness of COMT on different programs, we consider **12** representative programs with distinct behaviors from SPEC CPU2000 as *applu*, *art*, *bzip2*, *crafty*, *equake*, *galgel*, *gcc*, *lucas*, *mcf*, *swim*, *twolf*, and *vpr*. In our experiments, we randomly generate 400 design configurations without assuming the superiority of any specific one. During such a sampling process, every configuration violating any given constraint on design parameters will simply be discarded and replaced by a new one. Simulating the 400 configurations for all benchmarks consumes about 6800 machine hours on a cluster with 32 AMD Operton processors. Among the 400 simulated configurations, 300 design configurations are considered as the training data and the rest 100 configurations are adopted as the test data. Table 1 presents the performance statistics of the 300 training examples over the above benchmarks, where IPC (Instruction-per-Cycle) is utilized as the performance metric. From the statistics we observe that the variances of IPC (e.g., *equake*) are large on several benchmarks, which may make the performance prediction hard.

Table 1: Statistical features of the responses (IPC) of simulated configurations over different benchmarks.

Benchmarks	Avg.	Variance	Benchmarks	Avg.	Variance
applu	1.93	0.61	*gcc*	1.48	0.35
art	1.27	0.56	*lucas*	1.85	0.79
bzip2	2.13	0.75	*mcf*	1.54	0.31
crafty	1.52	0.42	*swim*	1.73	0.39
equake	2.19	0.76	*twolf*	2.21	0.70
galgel	1.67	0.39	*vpr*	2.2	0.72

4.2 Performance Evaluation of COMT

Table 2 shows the microprocessor design space that covers 10 important design parameters and contains more than 70M design configurations. To demonstrate the effectiveness of COMT, we compare the prediction accuracy of our approach with state-of-the-art predictive models introduced by İpek *et al.* [2006], where supervised ANNs are utilized to construct predictive models. Following the same setting utilized by İpek *et al.* [2006], the ANN adopts one 16-unit hidden layer, a learning rate of 0.001, and a momentum value of 0.5. For the proposed COMT, we set the number of computation iterations, i.e., t in Algorithm 1, to 100, and set the pool size holding evaluated unlabeled examples in each iteration, i.e., p in Algorithm 1, also to 100. To obtain two diverse model trees, we set M_1 and M_2 to 4 and 10, respectively. Besides, we randomly generate 5000 design configurations as the unlabeled set for COMT.

In Figure 2, the detailed comparisons of prediction results on test data are presented, where İpek *et al.* [2006]'s ANN-based model, the supervised M5P model, and COMT are involved, all use the labeled training set stated in the last section. We can clearly see that COMT significantly outperforms other approaches over all 12 benchmarks. Most notably, on the benchmark *galgel*, COMT reduces MSE by 84% of the ANN-based model. On the easy-to-predict benchmark *mcf*, COMT can reduce MSE by 68% of the ANN-based model. Even on the benchmark with the least MSE reduction (*crafty*), COMT still reduces MSE by 30% of the ANN-based model. Hence, we can conclude that COMT is much more practical than state-of-the-art predictive modeling technique because of its high accuracy.

On the other hand, from the comparisons between COMT and the supervised M5P model presented in Figure 2, COMT reduces MSE by 12% (*gcc*) to 65% (*bzip2*) of the supervised M5P model. The above fact demonstrates that semi-supervised learning is effective for enhancing the prediction accuracy of predictive modeling for DSE. In addition to measuring the performance of COMT with one training/testing

Table 2: Investigated microprocessor design space.

Abbr.	Parameter	Value
WIDTH	Fetch/Issue/Commit Width	2,4,6,8
FUNIT	FPALU/FPMULT Units	2,4,6,8
IUNIT	IALU/IMULT Units	2,4,6,8
L1IC	L1-ICache	8,16,32,64,128,256KB
L1DC	L1-DCache	8,16,32,64,128,256KB
L2UC	L2-UCache	256,512,1024,2048,4096KB
ROB	ROB size	16-256 with a step of 16
LSQ	LSQ size	8-128 with a step of 8
GSHARE	GShare size	1,2,4,8,16,32K
BTB	BTB size	512,1024,2048,4096
Total	**10 parameters**	**70,778,880 Options**

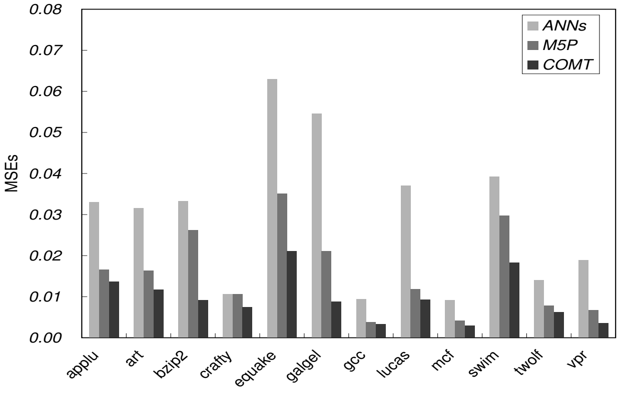

Figure 2: MSEs of state-of-the-art DSE approaches (ANN-based approach), the supervised M5P model and COMT over different benchmarks.

split, for each benchmark we also perform 5-fold cross validation on all 400 simulated design configurations. Figure 3 compares the MSEs along with the corresponding standard deviations of COMT and previous approaches. Taking the benchmark *galgel* as an example again, with 5-fold cross validation, the average MSEs of ANNs, the supervised M5P and COMT are 3.64E-2, 2.04E-2 and 9.34E-3, respectively, and the standard deviations of ANNs, the supervisde M5P and COMT are 7.79E-3, 2.59E-3 and 1.21E-3, respectively. The program with less MSE reduction is *crafty*, where the average MSEs of ANNs, the supervised M5P and COMT are 1.23E-2, 9.79E-3 and 8.73E-3 respectively, and COMT still exhibits more stable prediction accuracy, as evidenced by the smaller standard deviation compared with ANNs and the supervised M5P. In summary, the cross validation results further confirm that COMT can significantly outperform state-of-the-art DSE approach.

In addition, the impact of the number of unlabeled examples considered in each iteration, i.e., the pool size p in Algorithm 1, on the prediction accuracy is also studied. Table 3 presents the prediction results of COMT with respect to different pool sizes. Generally speaking, on the benchmarks, the best prediction accuracy is often achieved when a larger pool size is utilized. A potential explanation to this observation is that more unlabeled examples may, in general, offer more opportunities for COMT to exploit promising unlabeled examples that are effective for enhancing prediction accuracy.

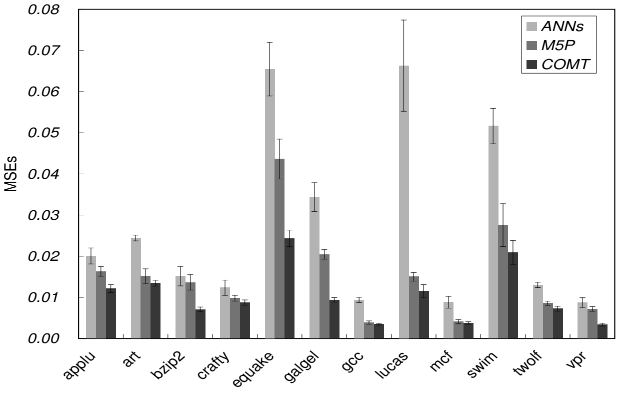

Figure 3: Mean MSEs and corresponding standard deviations of the ANN-based approach, the supervised M5P model and COMT in 5-fold cross validations.

Table 3: MSEs of COMT under different pool sizes p.

	Pool Size				
Benchmarks	**20**	**40**	**60**	**80**	**100**
applu	1.34E-2	1.52E-2	1.21E-2	1.29E-2	**1.02E-2**
art	1.29E-2	1.56E-2	1.45E-2	**1.17E-2**	1.38E-2
bzip2	2.67E-2	1.17E-2	1.37E-2	**1.07E-2**	1.16E-2
crafty	9.02E-3	9.20E-3	9.05E-3	8.92E-3	**8.58E-3**
equake	2.38E-2	1.88E-2	1.81E-2	**1.81E-2**	1.85E-2
galgel	1.51E-2	1.16E-2	1.04E-2	1.06E-2	**8.39E-3**
gcc	4.25E-3	3.42E-3	**2.80E-3**	3.15E-3	2.99E-3
lucas	1.30E-2	1.17E-2	1.07E-2	**1.04E-2**	1.09E-2
mcf	3.65E-3	4.01E-3	3.52E-3	3.61E-3	**3.47E-3**
swim	2.31E-2	2.34E-2	1.88E-2	1.59E-2	**1.28E-2**
twolf	9.80E-3	9.36E-3	8.48E-3	8.33E-3	**6.76E-3**
vpr	3.58E-3	3.52E-3	3.56E-3	**3.19E-3**	3.28E-3

4.3 Sensitivity to Training Iteration t

In this subsection, we further study the impact of the number of iteration, i.e., t in Algorithm 1, on the prediction accuracy of COMT. The experiment conducted here follows the same parameter setting (except t) as the previous experiments.

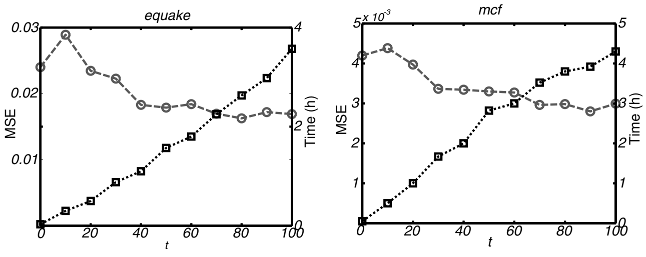

Figure 4: Prediction accuracies (MSE) and computational costs (Time) of COMT with respect to different numbers of training iterations (t).

Figure 4 offers an illustrative example about the relationship between the MSE of COMT and t, where two benchmarks, *equake* and *mcf*, are considered. In general, the MSE (red curves in Figure 4) roughly decreases as the number of iterations increases, especially when t is greater than 10. However, when the number of iterations has become large enough, the MSE may decrease with a relatively slower speed. An explanation to this observation is that those promising unlabeled examples which are valid for enhancing prediction accuracy have eventually become exhausted. In this case, the newly selected confident unlabeled examples, which are actually redundant in the context of the latest labeled data set, contribute little to the prediction accuracy of COMT. In the meantime, the training costs of COMT grow almost linearly with respect to t (black curves in Figure 4). Hence, when the computational resources are limited, it would be beneficial if t is decided by trading off the gained prediction accuracy and training costs, which can be achieved by setting more sophisticated stopping criterion for COMT. The related investigation will be conducted in our future work.

4.4 Deployment of Comprehensible Models

COMT can provide comprehensible models for efficient DSE. Ideally, to validate the effectiveness of our comprehensible models, we have to simulate the entire design space consisting of 70M design configurations, then compare the configuration deduced by COMT and the actual optimal configuration. However, this is infeasible due to intractably large simulation costs. A compromise is made by comparing the promising configuration deduced by COMT with the configuration searched by the state-of-the-art DSE approach (ANN-based modeling). To be specific, we simulate the two architectural configurations on an illustrative benchmark, and compare the corresponding processor responses directly.

```
              Performance Tree                |              Power Tree
WIDTH <= 3                                    |   WIDTH <= 3              LM1
|    LSQ <= 24                                |   WIDTH > 3
|    |    L2UC <= 393216                      |   |    ROB <= 96          LM2
|    |    |    LSQ <= 12     LM1              |   |    ROB > 96           LM3
......                                        |   ......
                                              |   W(LM1): 0.3824WIDTH + 0.5968FUNIT
I(LM1): 0.0131WIDTH + 0.0033IUNIT             |           + 0.1974IUNIT + 0.0001L1IC
          + 0.0043LSQ + 0.9028                |           + 0.0001L1DC + 0.0417ROB
                                              |       + 0.0285LSQ + 0.0001BTB + 8.7457
```

Figure 5: Sections of performance and power model trees.

As an example, we employ the benchmark *mcf* in the illustrative experiment. Suppose the power dissipation cannot exceed 50 watt according to microprocessor design specification. Due to the page limit, we only present sections of the built performance and power model tree in Figure 5. These trees imply several LPs, one of which is for the linear model pair $(I(LM_1), W(LM_1))$:

$$\max(0.0131\,WIDTH + 0.0033IUNIT + 0.0043LSQ + 0.9028)$$

$$s.t.\left\{\begin{array}{l} 0.3842\,WIDTH + 0.5968FUNIT + 0.1974IUNIT \\ 0.0001L1IC + 0.0001L1DC + 0.0417ROB \\ +0.0285LSQ + 0.0001BTB + 8.7457 \le 50 \\ 2 <= WIDTH <= \mathbf{3}, 2 <= FUNIT <= 8 \\ 2 <= IUNIT <= 8, 8K <= L1IC <= 256K \\ 8K <= L1DC <= 256K, 256K <= L2UC <= \mathbf{384}K \\ 16 <= ROB <= 256, 8 <= LSQ <= \mathbf{12} \\ 1K <= GSHARE <= 32K, 512 <= BTB <= 4096. \end{array}\right.$$

It is notable that constraints on $WIDTH <= 3$, $L2UC <= 384K$, and $LSQ <= 12$ are extracted from the decision rules while others are determined by the ranges of design parameters. By solving all the LPs transformed from performance-power linear model pairs, we attain the promising configuration as shown in Table 4, where the promising configuration found by its opponent (state-of-the-art DSE approach using ANNs) is also presented. Table 5 further presents the corresponding performance and power responses of these two configurations (namely, "ANN-conf" and "COMT-conf"). Although the actual performance (obtained by cycle-accurate simulations) of ANN-conf is slightly better than that of COMT-conf, the power dissipation of ANN-conf exceeds the power budget ($54.47 > 50$). Hence, COMT finds the promising configuration complies with the design specification, while its opponent fails. In practice, COMT has already been utilized for the Godson series [Hu *et al.*, 2009], in the early design phase of its next-generation processor cores.

Table 4: Comparison of promising design configurations attained by the ANN-based approach and COMT.

Parameter	ANN-conf	COMT-conf
WIDTH	8	8
FUNIT	2	2
IUNIT	8	8
L1IC	16KB	8KB
L1DC	32KB	128KB
L2UC	4096KB	4096KB
ROB	128	64
LSQ	128	32
GSHARE	1KB	1KB
BTB	2048	512

Table 5: Comparison of performance and power responses with respect to ANN-conf and COMT-conf.

Configuration	Responses	Predicted	Actual	Error
ANN-conf	Performance (IPC)	1.98	2.11	6.16%
	Power (Watt)	49.72	*54.47*	8.72%
COMT-conf	Performance (IPC)	2.11	2.03	**3.94%**
	Power (Watt)	48.24	49.13	**1.81%**

5 Conclusions and Future Work

In contrast to traditional brute-force DSE approaches, in this paper, we propose the COMT approach. By exploiting unlabeled design configurations, COMT significantly improves prediction accuracies and reduces excessive simulation costs. Moreover, due to the comprehensibility of model trees employed by COMT, computer architects can now analytically deduce the promising microprocessor architecture without exhaustive comparisons among a large number of design configurations.

For the computer architecture community, the COMT approach enables efficient and comprehensible deduction of the optimal architecture for every given benchmark program, which greatly benefits the DSE for pre-silicon microprocessor design. Furthermore, the COMT approach is especially promising for post-silicon microprocessor reconfiguration. COMT and its future versions can help finding the optimal architecture suitable for a given program or even the given program features, and thus guiding the architecture reconfiguration to adapt to various programs. This task was rarely studied by the architecture community, but would be a crucial step towards the development of an *elastic processor* (by which we call a processor whose architecture parameters can be dynamically reconfigured to suit different programs) and a *computer tribe* (by which we call a series of downward compatible elastic processors). Unlike the Field Programmable Gate Array (FPGA) whose reconfiguration may have to modify millions of controlling parameters, an elastic processor only employs a moderate number of reconfigurable parameters (e.g., 20), which avoids the problem of dimension explosion when building performance/power predictive models (via machine learning techniques) for guiding architecture reconfiguration.

The development of efficient elastic processors relies heavily on the advance of machine learning techniques, as evidenced by the impact of semi-supervised learning on DSE. To build an elastic processor, there are still open problems for machine learning and computer architecture researchers to work together. For example:

- *Program Feature Selection*: How to select adequate program features such that programs sharing similar features have similar responses on the same computer architecture?
- *Program Feature-driven DSE*: How to take program features into account in DSE?
- *Program Tribing*: How to cluster programs such that programs in the same cluster share the same "promising" architecture configurations?

Acknowledgments: We want to thank the reviewers for helpful comments and suggestions.

References

[Austin *et al.*, 2002] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.

[Blum and Mitchell, 1998] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th Annual Conference on Learning Theory (COLT)*, pages 92–100, 1998.

[Fujino *et al.*, 2005] A. Fujino, N. Ueda, and K. Saito. A hybrid generative/discriminative approach to semi-supervised classifier design. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, pages 764–769, 2005.

[Genbrugge and Eeckhout, 2009] D. Genbrugge and L. Eeckhout. Chip multiprocessor design space exploration through statistical simulation. *IEEE Trans. Computers*, 58(12):1668–1681, 2009.

[Hamerly *et al.*, 2006] G. Hamerly, E. Perelman, J. Lau, B. Calder, and T. Sherwood. Using machine learning to guide architecture simulation. *Journal of Machine Learning Research*, 7:343–378, 2006.

[Hennessy and Patterson, 2003] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Francisco, CA, 3rd edition, 2003.

[Hu *et al.*, 2009] W. Hu, J. Wang, X. Gao, Y. Chen, Q. Liu, and G. Li. Godson-3: A scalable multicore RISC processor with X86 emulation. *IEEE Micro*, 29(2):17–29, 2009.

[İpek *et al.*, 2006] E. İpek, S. A. McKee, R. Caruana, B. R. Supinski, and M. Schulz. Efficiently exploring architectural design spaces via predictive modeling. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 195–206, 2006.

[Joseph *et al.*, 2006] P.J. Joseph, V. Kapil, and M.J. Thazhuthaveetil. Construction and use of linear regression models for processor performance analysis. In *Proceedings of the 12th International Symposium on High Performance Computer Architecture (HPCA)*, pages 99–108, 2006.

[Joshi *et al.*, 2006] A. Joshi, A. Phansalkar, L. Eeckhout, and L. K. John. Measuring benchmark similarity using inherent program characteristics. *IEEE Trans. Computers*, 55(6):769–782, 2006.

[Lee *et al.*, 2008] B. C. Lee, J. Collins, H. Wang, and D. Brooks. Cpr: Composable performance regression for scalable multiprocessor models. In *Proceedings of the 41st International Symposium on Microarchitecture (MICRO)*, pages 270–281, 2008.

[Quinlan, 1992] J. R. Quinlan. Learning with continuous classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence (AJCAI)*, pages 343–348, 1992.

[Wang and Witten, 1997] Y. Wang and I.H. Witten. Induction of model trees for predicting continuous classes. In *Proceedings of the 9th European Conference on Machine Learning (ECML)*, pages 128–137, 1997.

[Wang and Zhou, 2007] W. Wang and Z.-H. Zhou. Analyzing co-training style algorithms. In *Proceedings of the 18th European Conference on Machine Learning (ECML)*, pages 454–465, 2007.

[Xu and Schuurmans, 2005] L. Xu and D. Schuurmans. Unsupervised and semi-supervised multi-class support vector machines. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, pages 904–910, 2005.

[Zhou and Jiang, 2004] Z.-H. Zhou and Y. Jiang. NeC4.5: Neural ensemble based C4.5. *IEEE Trans. Knowledge and Data Engineering*, 16(6):770–773, 2004.

[Zhou and Li, 2005] Z.-H. Zhou and M. Li. Semi-supervised regression with co-training. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 908–913, 2005.

[Zhou and Li, 2010] Z.-H. Zhou and M. Li. Semi-supervised learning by disagreement. *Knowledge and Information Systems*, 24(3):415–439, 2010.

[Zhu *et al.*, 2003] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of 16th International Conference on Machine Learning (ICML)*, pages 912–919, 2003.