An Active Learning Method for Empirical Modeling in Performance Tuning

Jiepeng Zhang
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China
hitzjp@mail.ustc.edu.cn

Jingwei Sun
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China
sunjw@mail.ustc.edu.cn

Wenju Zhou
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China
zhouwenj@mail.ustc.edu.cn

Guangzhong Sun
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China
gzsun@ustc.edu.cn

***Abstract*—Tuning performance of scientific applications is a challenging problem since performance can be a complicated nonlinear function with respect to application parameters. Empirical performance modeling is a useful approach to approximate the function and enable efficient heuristic methods to find sub-optimal parameter configurations. However, empirical performance modeling requires a large number of samples from the parameter space, which is resource and time-consuming. To address this issue, existing work based on active learning techniques proposed PBU Sampling method considering performance before uncertainty, which iteratively performs performance biased sampling to model the high-performance subspace instead of the entire space before evaluating the most uncertain samples to reduce redundancy. Compared with uniformly random sampling, this approach can reduce the number of samples, but it still involves redundant sampling that potentially can be improved.**

We propose a novel active learning based method to exploit the information of evaluated samples and explore possible high-performance parameter configurations. Specifically, we adopt a Performance Weighted Uncertainty (PWU) sampling strategy to identify the configurations with either high performance or high uncertainty and determine which ones are selected for evaluation. To evaluate the effectiveness of our proposed method, we construct random forest to predict the execution time of kernels from SPAPT suite and two typical scientific parallel applications *kripke*, *hypre*. Experimental results show that compared with existing methods, our proposed method can reduce the cost of modeling by up to 21x and 3x on average meanwhile hold the same prediction accuracy.

***Index Terms*—Performance Modeling, Active Learning, Sampling Strategy, Machine Learning**

I. Introduction

High-Performance Computing (HPC) ecosystems are comprised of increasingly complex hardware, software stacks, applications, etc. Each component of them can provide many configurable parameters that have profound impact on performance. As existing researches [1] illustrate, performance of a HPC application could be increased by 10x even 1000x with carefully selected parameters. Therefore, tuning configurable parameters is an essential problem to maximize the performance of HPC applications.

Tuning is a challenging problem since the relation between performance and configurable parameters is a complicated function in many HPC applications. Domain experts can use mathematical formulas to present the function, by getting insights into an application and manually constructing analytical performance models [2]–[4]. But this type of method is labor-intensive and not portable for different applications.

Without requiring domain knowledge, the relation between performance and configurable parameters can be regarded as a black-box function. Given a configuration of parameter, we can execute applications and measure their performance, though the impact of this configuration is not comprehensible. The tuning process can be automatically conducted by evaluating possible configurations and searching the one with best performance (e.g. shortest execution time). However, since the parameter space, namely all possible configurations, grows exponentially with respect to the number of parameters, it is infeasible to find the exactly optimal configuration via traversing the space.

Empirical performance modeling (EPM) is a practical approach to enable efficient heuristic auto-tuning that finds sub-optimal parameters with reasonable cost [5]–[7]. EPM adopts machine learning techniques to build a surrogate model as an approximation of the black-box performance function. After randomly sampling and learning from a portion of the parameter space, the surrogate model can predict the performance of any given configuration instead of measuring its actual execution. The key factor of successful auto-tuning is building an accurate surrogate model to precisely identify high-performance configurations and prune the exploration on poor-performance configurations. To train such an accurate model, a large number of performance samples is required. Since every individual execution and performance measurement of a HPC application may expend minutes to hours, the whole training

1530-2075/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPS47924.2020.00034

process on thousands of samples can be very expensive.

To reduce the expensive cost of so many evaluations in EPM, existing works [7]–[9] utilized active learning techniques to build surrogate models with fewer samples by reducing data redundancy. Active learning is a machine learning technique to interactively query the annotator for labels. Instead of indiscriminately random sampling, active learning iteratively identifies and evaluates the most informative configurations to achieve the same even better accuracy with fewer samples. In addition to reducing data redundancy with active learning, Balaprakash et al. [8] considered that in performance modeling for auto-tuning, more emphasis should be placed on high-performance configurations rather than the poor-performance ones. Specifically, they performed Performance Biased Uncertainty Sampling (PBUS) in active learning to model the high-performance subspace rather than the entire parameter space. However, due to the limitations of this biased sampling, sometimes the redundancy problem can become more severe.

To resolve this problem, we propose a novel active learning based method to make a better exploration-exploitation balance. Specifically, our method iterates between constructing random forest and using it to determine which configurations to evaluate. For the latter determination step, we design a Performance Weighted Uncertainty (PWU) strategy to identify the configurations with either high performance or high uncertainty. PWU can quantify the contribution that each configuration can make to the modeling, based on a metric that combines performance and uncertainty in a novel way. Owing to this novel quantification method, PWU can make a better balance between the exploration of possible high-performance configurations and the exploitation of evaluated samples, so less data redundancy is achieved.

In this work, we mainly make these contributions:

- We propose an active learning method for empirical modeling in performance tuning to build random forest with higher accuracy of high-performance configurations and further reduce data redundancy. Owing to the random forest, our method enables efficient modeling of the vast parameter space with mixed numerical/categorical features and is robust to data with outliers and noise.
- We design a Performance Weighted Uncertainty sampling strategy to identify the configurations with either high performance or high uncertainty for evaluation by quantifying the contribution that each configuration can make to the model. Compared with existing works, the redundancy problem is alleviated when adopting our PWU strategy.
- Experimental results show that compared with existing work, our proposed active learning method with PWU strategy can reduce the cost of modeling by up to 21x and 3x on average meanwhile achieve the same accuracy. Besides, with the same time cost, our method can achieve higher prediction accuracy.

The remaining sections of this paper is structured as follows. Section II describes the framework and detailed processes of our proposed active learning method. Section III introduces the experimental setup. Section IV presents the results and analysis of our experiments. Section V lists the related work and section VI concludes the paper.

II. Methodology

Traditional empirical modeling methods perform random uniform sampling in the parameter space, evaluate their performance and build models using these evaluated samples. To reduce the cost of empirical modeling, we propose a PWU sampling strategy in active learning to model the high-performance subspace and further reduce data redundancy among them. An overview of our proposed active learning method is illustrated in Fig.1.

Generally, performance can be execution time, throughput rate, system utilization, etc. In this paper, we measure the execution time of programs as its performance and the wall time is chosen. The shorter the execution time, the higher the performance, and vice versa.

To illustrate our active learning method, first we introduce the active learning technique and the surrogate model we choose in this work and then explain our novel sampling strategy in detail.

A. Active Learning

Active learning [10] is a machine learning technique to interactively query the annotator for labels. In statistics literature it is also called optimal experimental design [11]. The key idea behind active learning is that a machine learning algorithm can perform better with less training if it is allowed to choose the data from which it learns. A typical active learning process is to iteratively select the most informative samples from an unlabeled data pool. Through choosing data purposefully instead of choosing blindly, active learning can reduce the information redundancy of data, so less training data can offer equivalent even higher accuracy compared with random sampling.

Specifically, in empirical performance modeling, a parameter space is usually too large to enumerate, therefore a data pool containing sufficient parameter configurations can be sampled randomly from this parameter space and this data pool can be used to represent the entire parameter space. Its effectiveness has been proven in several works [5], [7]. Details of active learning are listed in algorithm 1.

Algorithm 1 includes two phases: cold-start phase (Line 1-4) and iteration phase (Line 5-9). In the beginning, because the training set is empty and the model is not initialized, the cold-start phase is required to sample a few configurations from data pool to initialize the model. Then the following iteration phase begins. First, a previously designed sampling strategy utilizes the trained model to determine which configurations from the pool to be selected. Programs with these configurations as inputs are executed to measure their performance. Finally, the algorithm appends new evaluated samples to the training set and updates the model using the new training set. This process continues until it reaches the maximum number of

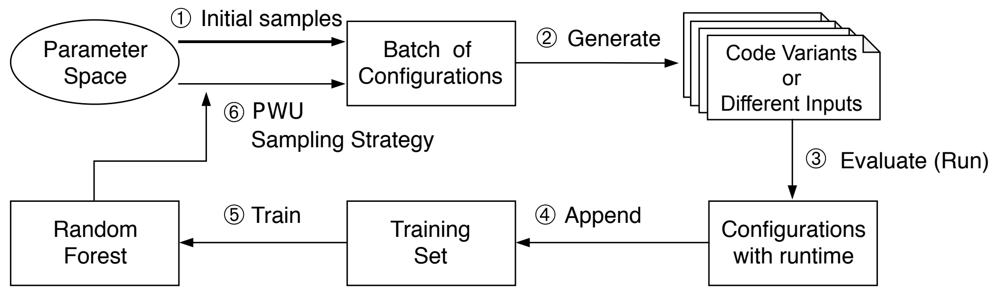


Fig. 1: Overview of our active learning approach to build a performance model. After an initial small set of configurations is sampled randomly (①), a loop (②-⑥) can start. These sampled configurations are used to generate different code variants when tuning compilation parameters or programs with different inputs when tuning parameters of inputs (②). Then these code variants or programs are executed to measure their execution time (③). When new configurations with execution time are appended to training set (④), random forest can be trained from scratch or updated partially (⑤). Next our proposed UPS sampling strategy utilizes the trained random forest to identify the most profitable configurations for next iteration (⑥). This loop continues until the size of training set reaches a given maximum value.

Algorithm 1: Active Learning Algorithm

```
Data: unlabeled data pool $\mathcal{X}_{pool}$, size of initial training set $n_{init}$, batch size $n_{batch}$, maximum size of training set $n_{max}$,
      learning algorithm RandomForest, SamplingStrategy
Result: training set $< \mathcal{X}_{train}, \mathcal{Y}_{train} >$, model $\mathcal{M}$
1 $\mathcal{X}_{train} \longleftarrow$ Sample $n_{init}$ configurations from $\mathcal{X}_{pool}$ randomly
2 $\mathcal{Y}_{train} \longleftarrow$ Evaluate($\mathcal{X}_{train}$) // Evaluate the sampled configurations
3 $\mathcal{M} \longleftarrow$ RandomForest($\mathcal{X}_{train}$, $\mathcal{Y}_{train}$) // Construct a random forest from scratch using
     training set
4 $\mathcal{X}_{pool} \longleftarrow \mathcal{X}_{pool} \setminus \mathcal{X}_{train}$
5 while size of $\mathcal{X}_{train}$ does not reach $n_{max}$ do
6     $\mathcal{X}_{batch} \longleftarrow$ SamplingStrategy($\mathcal{M}$, $\mathcal{X}_{pool}$, $n_{batch}$) // The sampling strategy utilizes tthe
          fitted random forest to select a batch of configurations
7     $\mathcal{Y}_{batch} \longleftarrow$ Evaluate($\mathcal{X}_{batch}$) // Evaluate the selected configurations
8     $\mathcal{X}_{train} \longleftarrow \mathcal{X}_{train} \bigcup \mathcal{X}_{batch}; \mathcal{Y}_{train} \longleftarrow \mathcal{Y}_{train} \bigcup \mathcal{Y}_{batch}; \mathcal{X}_{pool} \longleftarrow \mathcal{X}_{pool} \setminus \mathcal{X}_{batch}$ // Append new labeled
          samples to training set
9     $\mathcal{M} \longleftarrow$ RandomForest($\mathcal{X}_{train}$, $\mathcal{Y}_{train}$) // Construct a random forest from scratch or update
          it partially using the new training set
  end
```

evaluations. Through iteratively selecting the configurations with the most contribution to modeling, active learning can achieve the same even higher prediction accuracy, compared with random uniform sampling.

B. Surrogate Model

In active learning, besides outputting the prediction of a sample, the model must also be able to give the uncertainty of this prediction. A common choice of model is Gaussian Process or its variants. A Gaussian Process learning method (GP) [12] fits the training data with a Gaussian process and gives a prediction with a confidence interval. It usually works well for numerical features but not categorical features and fits only noise-free or Gaussian noise observations. To overcome these weaknesses, we adopt random forest to construct our active learning model.

Random forest [13] is an ensemble learning method for classification and regression tasks. It constructs a collection of decision trees with the training set and when predicting, the most popular voted class (for classification) or mean prediction (for regression) of all trees is outputted. A decision tree is a tree-like learning model that breaks down a complex decision-making process into a tree-like decision relationship. It constructs a tree recursively that at each node a split feature with the highest gain of information is selected, until some conditions are satisfied.

To eliminate the tendency to overfit of decision trees, random forest introduces two kinds of randomness into the construction of trees. One is bootstrap aggregating or bagging. For each tree, a subset is sampled with replacement from the training set and then the tree is fitted to this subset with another randomness, random subspace method. When applying the random subspace method, a tree does not utilize all of the features but a random subset of the features. This randomness makes a good balance between the strength of the individual tree and the correlation of different trees. A range of benefits are provided as follow:

1) Theoretical proof against overfitting
2) Robustness to outliers and noise
3) Effectiveness on categorical features

Moreover, random forest have another important feature that can be utilized in active learning. When applying to a regression task using random forest, the variance of predictions of different trees in forest can be calculated as the uncertainty of its prediction [14]. An accurate measure of uncertainty is the precondition of different sampling strategies in active learning. Benefiting from the reasonable balance between the strength and the correlation, the variance of different trees in random forest won't be biased too much by some incapable trees and can be an accurate representative of the uncertainty of prediction.

C. Sampling Strategy

The sampling strategy is critical in active learning. An effective sampling strategy can identify a batch of samples that contribute most to the optimization target. However, due to the large number of combinations of batch, designing an effective sampling strategy is challenging. Practically, we can apply a greedy solution to this problem. A common method is to sort the samples by scores, which represent how much contribution a sample can make. Once the samples are sorted by their scores, we can simply choose a batch of samples which have the highest scores. As a result, compared with random sampling, less training samples with reduced redundancy can achieve comparative accuracy.

The key component of the above strategy is the scoring method, namely how to quantify the contribution that a configuration can make to the modeling. Generally, the above uncertainty scoring method is effective to reduce the redundancy. But in this work, we aim to not only reduce the data redundancy but also achieve high accuracy of high-performance samples regardless of the accuracy of poor-performance samples. We distinguish the high-performance samples from the poor-performance ones by their performance rankings [1]. The top α percentage of samples in the performance rankings are considered as high-performance and the remaining ones are not. α lies between 0 and 1 and is usually a small value, such as 0.01, 0.05, 0.10 in this paper. For example, $\alpha = 0.01$ implies that only the top 1% percentage of samples are high-performance.

Suppose we have built a random forest with b individual trees to predict the execution times of n configurations in the data pool, for a specific configuration $x_i (i = 1, 2, ...n)$, we calculate its prediction mean μ_i and uncertainty σ_i using the method proposed in [14]. For these n configurations, the prediction mean and its uncertainty can be represented by two vectors: $\boldsymbol{\mu} = [\mu_1, \mu_2, ..., \mu_n], \boldsymbol{\sigma} = [\sigma_1, \sigma_2, ..., \sigma_n]$. To calculate the scores of these configurations, we treat Performance as the Weight of Uncertainty (**PWU**):

$$\boldsymbol{s} = \frac{\boldsymbol{\sigma}}{\boldsymbol{\mu}^{(1-\alpha)}} \tag{1}$$

where all operations adopt an entry-wise way and α represents the proportion of high-performance configurations. The choice of α will be discussed in Section III-C.

To make it clear, the PWU scoring method in Equation 1 is based on an intuition that the configurations with high performance and high uncertainty should be preferred and both factors should be combined together instead of considering one factor before another like PBUS method did in [8], of which the limitation is demonstrated in Section IV-C. When adopting the PWU scoring method, the samples with high performance, namely short execution time, are going to contribute more weight. For example, given two samples with the same uncertainty, the one with higher performance is preferred. Thus more high-performance samples with high uncertainty are selected.

Further, when α approaches 1, all configurations are considered as high-performance, namely modeling the whole parameter space. In this case, $\boldsymbol{s}$ will be reduced to $\boldsymbol{\sigma}$ and it'll select the most uncertain samples, which is the most common practice in active learning. When α is close to 0, $\boldsymbol{s}$ becomes $\frac{\boldsymbol{\sigma}}{\boldsymbol{\mu}}$, which is known as coefficient of variation (CV) in statistics. The coefficient of variation represents the ratio of the standard deviation to the mean, and it is a useful statistic for investors in finance to evaluate risk-return trade-off, corresponding to the uncertainty-performance trade-off in this paper.

III. Experimental Setup

In order to evaluate the effectiveness of our active learning method, we conduct a series of experiments to construct models for 12 kernels and two typical parallel applications on two different platforms. Details are listed below.

A. Benchmarks

Kernels

SPAPT [15] is a set of extensible and portable search problem in automatic performance tuning. Since the transformation and compilation of some kernels are very time consuming, only 12 of 18 kernels are chosen for modeling and they are still very representative. Every search problem contains a serial implementation of a computation kernel, the size of this problem, and some configurable compilation parameters such as loop unrolling, cache tiling, register tiling, etc. Among these problems, the number of compilation parameters ranges from 8 to 38 and the size of search space ranges from 10^{10} to 10^{30}.

For example, ADI is a stencil code kernel for matrix subtraction, multiplication and division. Its main computation code is listed in Listing 1 and the input size is controlled by variable N. The types and possible values of compilation parameters are listed in Table I. These compilation parameters control the transformation of code through a DSL language [16].

Parallel Applications

Two parallel applications are chosen: *kripke*, *hypre*.

kripke [17] is a mini-app developed at LLNL, designed to be a proxy for a fully functional discrete-ordinates transport code. It has some parameters that affect only its performance except its output and correctness. These parameters, which are listed in Table II, are treated as the inputs of our performance model.

hypre is a software library for the solution of large sparse linear systems on massively parallel computers. *new_ij* [18] is a test driver for it and in this work, we use *new_ij* to solve a 27pt 3D laplacian problem and we mark this problem as *hypre*. Table III shows the performance parameters of *hypre*. Besides the number of process (#process), the other parameters are all categorical.

B. Platform and Data Collection

The kernels and applications are executed on two different platforms A and B respectively. Details of node configuration are shown in Table IV.

For kernels, they are small serial programs, compiled with GCC 7.3 and executed on CentOS 7. The execution time of these kernels is usually less than one second and their performance tends to be affected by the system noise. To reduce the effect of system noises, two approaches are adopted. First, when running programs, only necessary services and processes are reserved. Second, the program of one configuration is executed 35 times [8] for average performance.

For applications, they are compiled with GCC 7.3 and executed on CentOS 7 with openmpi 3.1. The same as the kernels, to reduce the effect of the unstable network, every parameter configuration is evaluated several times for average performance.

Listing 1: Main computation code of ADI kernel

```
for (i1=0; i1<=N-1; i1++)
for (i2=1; i2<=N-1; i2++){
X[i1][i2] = X[i1][i2] - X[i1][i2-1] *
    A[i1][i2] / B[i1][i2-1];
B[i1][i2] = B[i1][i2] - A[i1][i2] *
    A[i1][i2] / B[i1][i2-1];
}
```

TABLE I: Compilation parameters of ADI kernel

Type	Number	Values
tile	8	1, 16, 32, 64, 128, 256, 512
unrolljam	4	1, 2, 3, ..., 31
regtile	4	1, 8, 32
scalarreplace	2	True, False
vector	2	True, False

C. Evaluation

Baseline

TABLE II: Parameters of *kripke*

Name	Values
layout	DGZ, DZG, GDZ, GZD, ZDG, ZGD
gset	1, 2, 4, 8, 16, 32, 64, 128
dset	8, 16, 32
pmethod	sweep, bj
#process	1, 2, 4, 8, 16, 32, 64, 128

TABLE III: Parameters of *hypre*

Name	Values
solver	0-15, 18, 20, 43-45, 50-51, 60-61
coarsening	pmis, hmis
smtype	0-8
#process	8, 16, 32, 64, 128, 256, 512

TABLE IV: Node configuration of two platforms

Specification	Platform A	Platform B
CPU type	E5-2680 v3	E5-2680 v4
CPU frequency	2.5GHz	2.4GHz
#core	24	28
memory	64GB	128GB
network	-	100Gbps OPA

First, we refined the random sampling method. Different from the traditional random uniform sampling, it is to sample randomly from the top $p\%$ configurations in predicted performance rankings, denoted by **BRS** (Biased Random Sampling).

To pay more efforts to the high-performance configurations, existing work [8] adopted a performance biased sampling before exploring the uncertain space in active learning (Performance Biased Uncertainty Sampling, **PBUS**). In addition to the case that both performance and uncertainty are taken into account, we also evaluated the sampling methods that consider either performance or uncertainty, denoted by **BestPerf** and **MaxU** respectively.

Evaluation Metrics

Since we aim to build a model with higher accuracy for high-performance configurations, simply calculating the prediction error of all configurations is not suitable in this work. Therefore we calculate the prediction error of the top $100\alpha\%$ samples in performance rankings [1], [8] as one of our evaluation metrics. For the measurement of prediction error, we choose the Root Mean-Square Error (**RMSE**) [8], [9].

For a test set that contains n random samples and is sorted by performance from high to poor, its $RMSE$ is calculated as follows:

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}, m = \lfloor n \times \alpha \rfloor \tag{2}$$

where y_i, $\hat{y}_i$ are the observation and the prediction of the i-th sample respectively. α represents the acceptable proportion in performance rankings and is usually set to a small value. For example, $\alpha = 0.05$ means that the top 5% configurations in performance rankings are considered as high-performance and the others are poor-performance.

In addition to the evaluation of prediction error level, the cost to achieve a good level must also be measured. Here we calculate the cumulative time of labeling the samples, namely executing the programs with corresponding configurations, as the cost. Formally, the Cumulative time Cost (**CC**) is calculated as follows:

$$CC = \sum_{i=1}^{n} y_i \tag{3}$$

where n is the size of training data and y_i represents the execution time of i-th sample.

In our experiments, a test set is sampled randomly from the parameter space and the label (execution time) of every configuration is measured in advance. In algorithm 1, at the end of each iteration, the built random forest will be evaluated using the above metrics.

D. Algorithm Parameters

10,000 configurations are sampled randomly and uniformly from the parameter space as a surrogate of the whole space, and later experiments have shown its sufficiency. For per experiment, we randomly split them into 7000 samples as data pool in Algorithm 1 and 3000 samples as test set for evaluating the model.

A large initial size of training set is contrary to our goal of reducing the modeling cost so it is set to a small value, 10. The batch size n_{batch} is set to 1, which means one sample will be selected and evaluated at each iteration. The maximum evaluation number of samples n_{max} is set to 500 because the model begins to converge when collecting about 500 samples.

The choice of α in Section II-C depends on the practical modeling demands. A large value implies that more configurations are considered as high-performance and a small value means more demanding requirement of performance. Like Jayaraman et al. did in [1], we set α with 0.05 and evaluate two corresponding metrics ($RMSE$ and CC) after every sampling. Moreover, we also set α with different values (0.01, 0.10) to evaluate the effectiveness of our method at different modeling demands.

IV. Experimental Results

To compare our proposed active learning method with existing methods, we conduct the active learning process for 12 kernels from SPAPT and two scientific parallel applications via different methods.

To reduce the impact of randomness from data preprocessing and model training in random forest, for each kernel or scientific application, totally 10 random experiments of active learning process in algorithm 1, are conducted. In each experiment, the built model will be evaluated at each iteration and the results are averaged over the 10 experiments.

A. Overall Analysis

First, we set α with 0.01, implying that only the top 1% samples in performance rankings are considered high-performance and acceptable. Fig.2 and Fig.4 (a) represent the RMSE varying with the number of labeled samples via different sampling methods for 12 kernels and two parallel applications. For all but one program our proposed PWU sampling method first reaches a low error level and maintains an obvious advantage over the other baselines almost all the time.

In terms of modeling cost, Fig.3 and Fig.4 (b) represent the CC to label a specific number of samples via different sampling methods. Although the BestPerf and BRS(Biased Random Sampling) consume the least time cost, the rewards (RMSE) are not ideal because the data redundancy is not taken into account. Compared with PBUS, our PWU method not only consumes less cost but also achieves lower error level for all kernels. For two parallel benchmarks, PWU costs much more than baselines, but in Fig.5 which plots the RMSE varying with the time cost, our method still outperforms the others or at least achieves comparable ability. Specifically, Fig.7 lists the ratio of time cost to reach a low error level with sampling via PBUS and PWU. Compared with PBUS, our proposed method PWU is able to reduce the cost of modeling by up to 21x and 3x on average.

B. Discussion on α

α represents the proportion of high-performance configurations across the whole parameter space and changes with different demands. To test the robustness of our method, apart from 0.01, we set α with more values: 0.05, 0.10. Taking the kernel atax for illustration, Fig.6 shows the comparison of PBUS and PWU at different values of α. Apparently, our proposed method achieves similar results and performs best with different α, which implies that our design is effective and robust.

C. Case Study

Kernel: *atax*

To analyze in depth the difference of our PWU sampling and PBUS method, we drew the selected samples in a two-dimensional coordinate system according to its predicted performance and uncertainty in Fig.9. It it obvious that PBUS puts too much weight into the low uncertainty area and our proposed PWU sampling method can identify the samples with more uncertainty, namely less redundancy, and make a better balance between exploration and exploitation.

Performance Tuning

To illustrate the ability of our built surrogate model, we treat the prediction of surrogate model as the true observation to avoid repeated executions of target program. That is to say, the surrogate model enables negligible cost of thousands of annotations. We performed model based tuning processes with two kinds of annotators. One is true annotator, namely the ground truth, which executes program to measure its

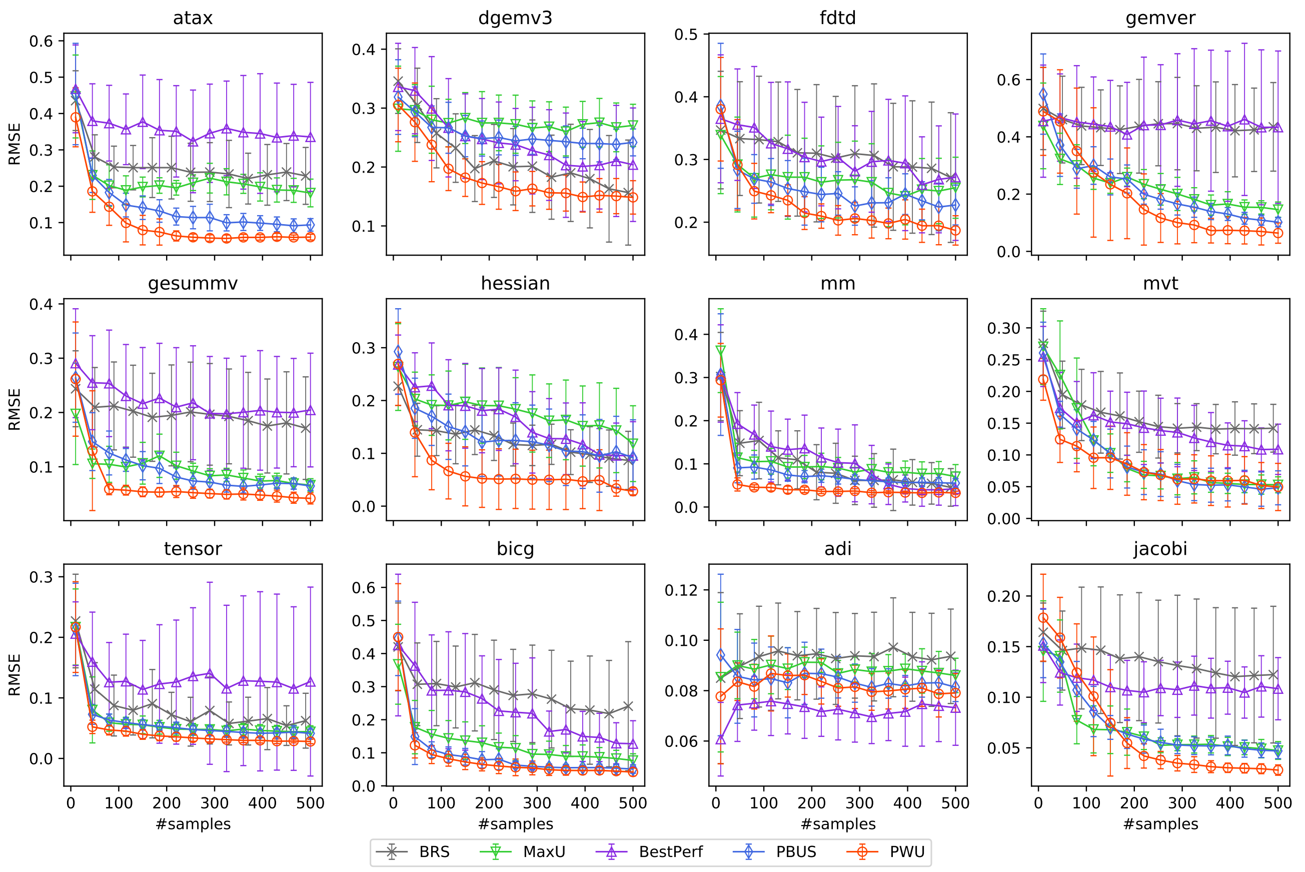


Fig. 2: RMSE varying with the number of samples in training set via different sampling methods for 12 kernels.

performance as label and the other is surrogate model, which treats the prediction of surrogate model as the observation. At each iteration, we selected the best prediction performance from the above test set. Fig.8 shows the tuning results of kernel `atax` via these two methods and it implies that in terms of tuning, the quality of the surrogate model is comparative to, even better than, the ground truth.

V. Related Work

Analytical Modeling

With the help of domain experts, researchers can analyze the code of applications manually and deduce a mathematical expression model based on computation models such as LogP. Kerbyson et al. presented an analytical model to reveal bottlenecks and possible effective parameter tuning of a multidimensional hydrodynamics code SAGE [19]. Barker et al. developed a performance model of a large-scale hydrodynamics code Krak by separating inter-process communication from computation and modeling them individually [4]. However, these analytical modeling methods require expert knowledge and are not portable for different applications.

Empirical Modeling

Existing researches show that learning an accurate model from empirical configuration-performance data is practical and effective. Ipek et al. designed neural networks to predict execution time of SMG application and achieved low prediction error even across a large multidimensional parameter space [20]. In addition to the program-inputs features, Huang et al. extracted runtime features from the instrumented programs to build LASSO models with interpretability [21]. Hunter et al. proposed some new features for NP-hard problems including SAT, TSP, MIP and compared the effectiveness of 11 learning algorithms [14]. Compared with these works that focus on extracting informative features and designing effective learning algorithms like Random Forest, in this paper we aim to reduce modeling cost via efficient sequential sampling.

Sequential Modeling

Instead of random uniform sampling in the parameter space in conventional empirical modeling, sequential modeling utilizes existing data to infer and select samples that contribute most to modeling goals like low prediction error. Hutter et al. sequentially built random forest and calculated the EI to select the most promising parameter configuration [22]. Balaprakash et al. designed an active learning method to select a batch of samples in each iteration for building accurate models with

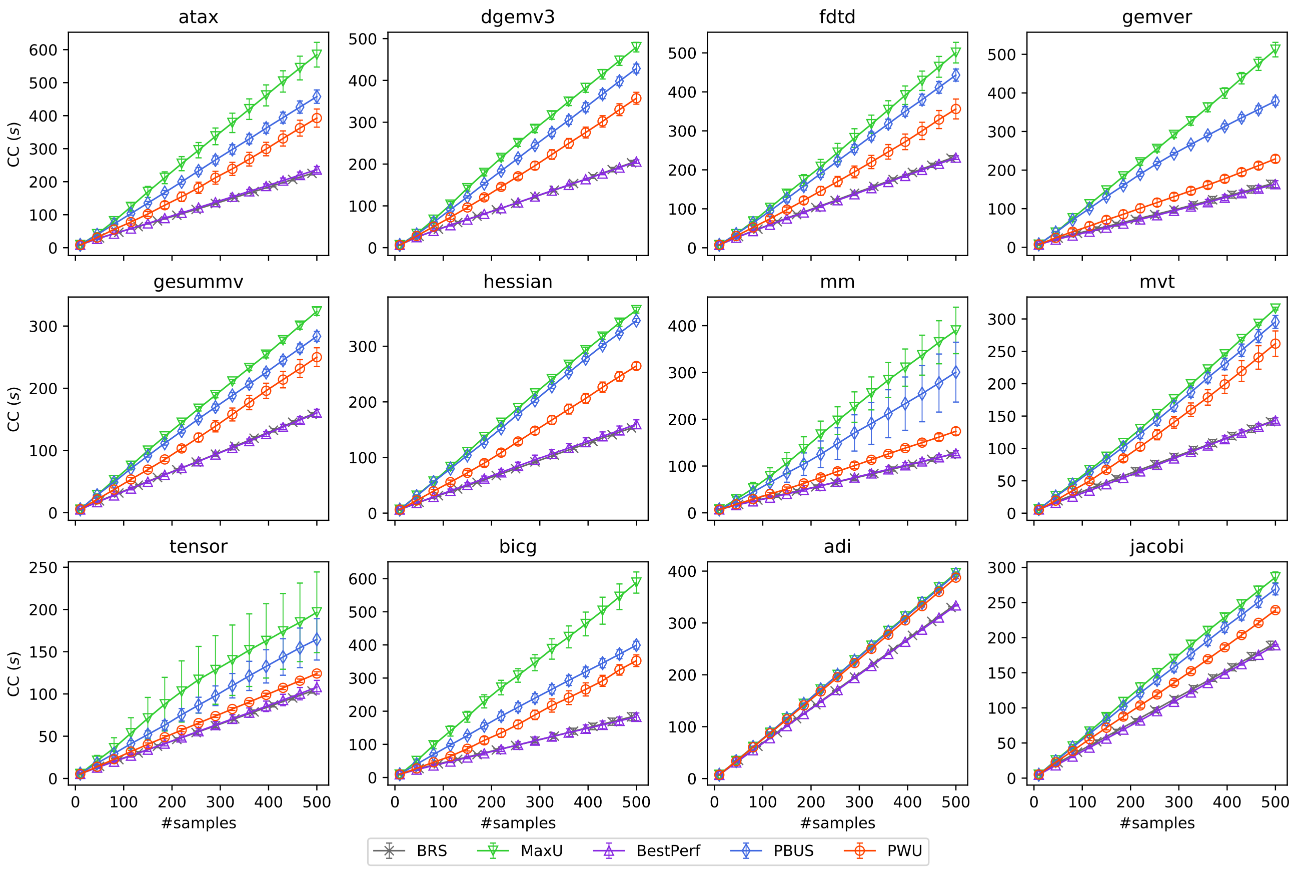


Fig. 3: CC varying with the number of samples in training set via different sampling methods for 12 kernels.

less cost [8]. Based on active learning, Nelson et al. built random forest to assist searching the parameter space of transforming efficient tensor contractions for GPUs [7] and Ogilvie et al. constructed dynamic trees to predict the performance of configurable compilation parameters [9]. Instead of modeling the entire parameter space, Balaprakash et al. [8] also designed an active learning based PBUS sampling strategy to model the high-performance subspace for performance tuning. In this paper, we have shown its limitations and advanced it by designing a more efficient sampling strategy.

VI. Conclusion

In this work, we propose an effective active learning method to iteratively build accurate performance models for searching high-performance configurations in the parameter space. At each iteration, it constructs a random forest and utilizes the forest model to identify a batch of configurations based on a sampling strategy for next evaluation. For the latter step, we design an Performance Weighted Uncertainty (PWU) sampling strategy to identify the configurations with high performance or high uncertainty. To evaluate the effectiveness of our method, we have constructed random forests via different methods to predict the performance of 12 kernels and two parallel applications. Experimental results show that our active learning method with PWU sampling strategy can avoid the limitation of existing methods and performs best. Compared with the best of existing method PBUS, our proposed method PWU is able to reduce the cost of modeling by up to 21x and 3x on average, meanwhile achieve the same even better accuracy.

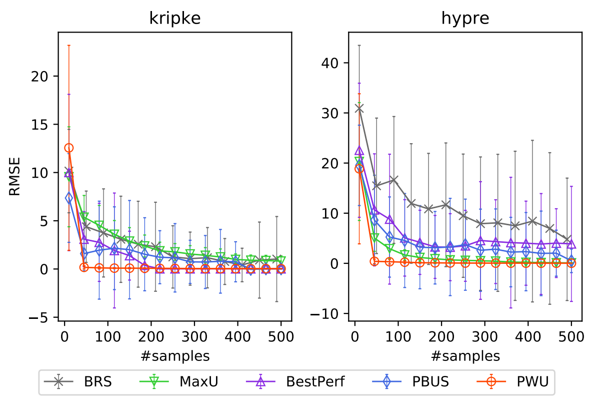
In future works, we can investigate the relation of different kernels and the portability of performance models to avoid building models from scratch when encountering new kernels or platforms.

ACKNOWLEDGMENT

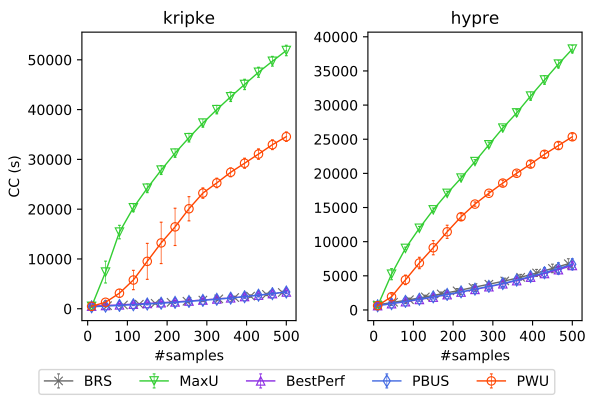
This work is supported by Youth Innovation Promotion Association of CAS and National Natural Science Foundation of China (No.61772485 and No.61432016). The evaluations of kernels and applications in this paper have been done on the supercomputing system in the Supercomputing Center of University of Science and Technology of China. Guangzhong Sun (gzsun@ustc.edu.cn) is the corresponding author of this paper.

References

[1] J. J. Thiagarajan, N. Jain, R. Anirudh, A. Gimenez, R. Sridhar, A. Marathe, T. Wang, M. Emani, A. Bhatele, and T. Gamblin, "Bootstrapping parameter space exploration for fast tuning," in *Proceedings*



(a) RMSE



(b) CC

Fig. 4: RMSE and CC varying with the number of samples in training set for two applications: *kripke*, *hypre*

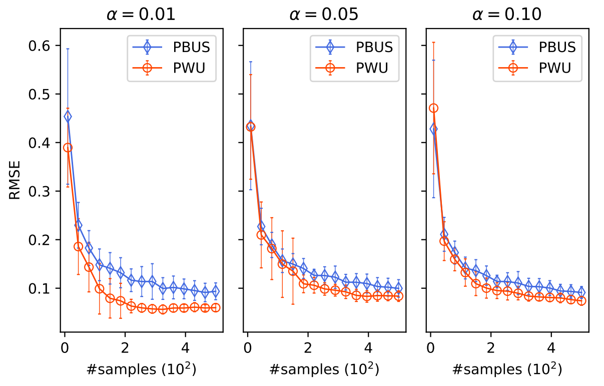


Fig. 6: RMSE varying with #samples at different α values

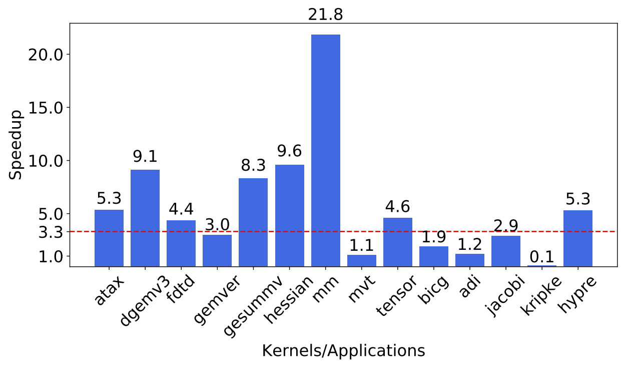


Fig. 7: The speedup of the cumulative time cost achieved by our proposed PWU method, compared with PBUS

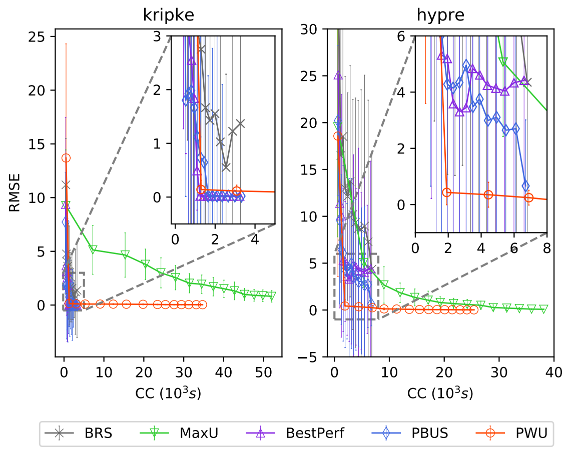


Fig. 5: RMSE varying with the cumulative time cost

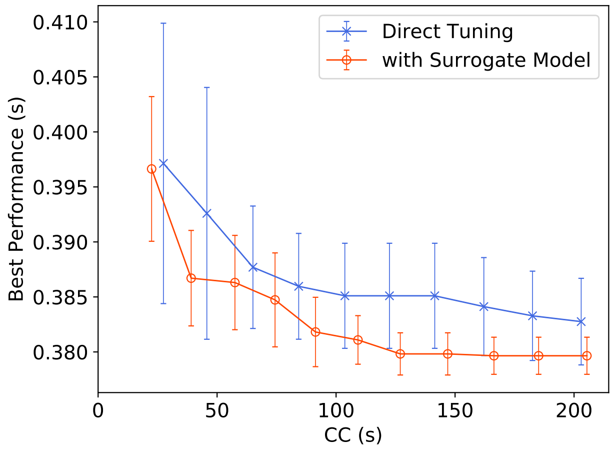


Fig. 8: Comparison of direct tuning and tuning with surrogate model

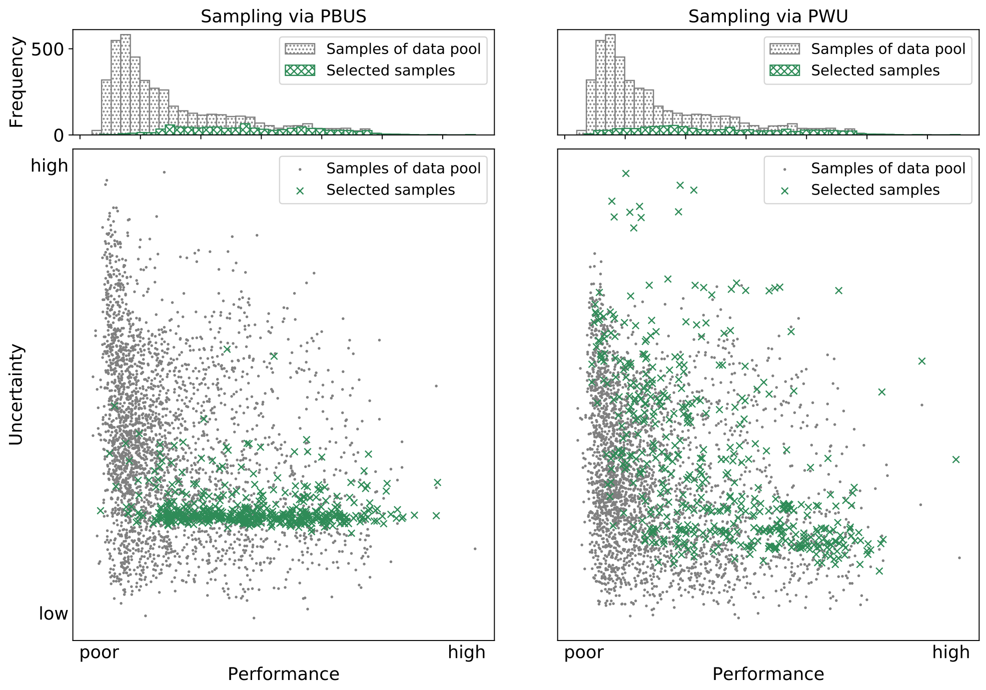


Fig. 9: The distribution of selected samples via PBUS and PWU for kernel ATAX. Taking the left graph for an example, a sample is drawn in the two-dimensional coordinate system according to its predicted performance and uncertainty. The grey points marked by "·" represent the distribution of samples in the data pool, and the green points marked by "×" represent the distribution of selected samples from the pool.

of the 2018 International Conference on Supercomputing, pp. 385–395, ACM, 2018.

[2] M. J. Clement and M. J. Quinn, "Analytical performance prediction on multicomputers," in *Supercomputing'93: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, pp. 886–894, IEEE, 1993.

[3] D. Sundaram-Stukel and M. K. Vernon, "Predictive analysis of a wavefront application using loggp," *ACM SIGPLAN Notices*, vol. 34, no. 8, pp. 141–150, 1999.

[4] K. J. Barker, S. Pakin, and D. J. Kerbyson, "A performance model of the krak hydrodynamics application," in *2006 International Conference on Parallel Processing (ICPP'06)*, pp. 245–254, IEEE, 2006.

[5] P. Balaprakash, S. M. Wild, and P. D. Hovland, "Can search algorithms save large-scale automatic performance tuning?," *Procedia Computer Science*, vol. 4, pp. 2136–2145, 2011.

[6] J. Bergstra, N. Pinto, and D. Cox, "Machine learning for predictive auto-tuning with boosted regression trees," in *2012 Innovative Parallel Computing (InPar)*, pp. 1–9, IEEE, 2012.

[7] T. Nelson, A. Rivera, P. Balaprakash, M. Hall, P. D. Hovland, E. Jessup, and B. Norris, "Generating efficient tensor contractions for gpus," in *2015 44th International Conference on Parallel Processing*, pp. 969–978, IEEE, 2015.

[8] P. Balaprakash, R. B. Gramacy, and S. M. Wild, "Active-learning-based surrogate models for empirical performance tuning," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 1–8, IEEE, 2013.

[9] W. F. Ogilvie, P. Petoumenos, Z. Wang, and H. Leather, "Minimizing the cost of iterative compilation with active learning," in *Proceedings of the 2017 International Symposium on Code Generation and Optimization*, pp. 245–256, IEEE Press, 2017.

[10] B. Settles, "Active learning literature survey," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2009.

[11] F. Pukelsheim, *Optimal design of experiments*. SIAM, 2006.

[12] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*, pp. 63–71, Springer, 2003.

[13] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[14] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artificial Intelligence*, vol. 206, pp. 79–111, 2014.

[15] P. Balaprakash, S. M. Wild, and B. Norris, "Spapt: Search problems in automatic performance tuning," *Procedia Computer Science*, vol. 9, pp. 1959–1968, 2012.

[16] B. Norris, A. Hartono, and W. Gropp, "Annotations for productivity and performance portability," *Petascale Computing: Algorithms and Applications. Computational Science. Chapman & Hall/CRC Press, Taylor and Francis Group*, pp. 443–462, 2007.

[17] A. J. Kunen, T. S. Bailey, and P. N. Brown, "Kripke-a massively parallel transport mini-app," tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2015.

[18] R. D. Falgout, J. E. Jones, and U. M. Yang, "The design and implementation of hypre, a library of parallel high performance preconditioners," in *Numerical solution of partial differential equations on parallel computers*, pp. 267–294, Springer, 2006.

[19] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *SC'01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, pp. 39–39, IEEE, 2001.

[20] E. Ipek, B. R. De Supinski, M. Schulz, and S. A. McKee, "An approach to performance prediction for parallel applications," in *European Conference on Parallel Processing*, pp. 196–205, Springer, 2005.

[21] L. Huang, J. Jia, B. Yu, B.-G. Chun, P. Maniatis, and M. Naik, "Predicting execution time of computer programs using sparse polynomial regression," in *Advances in neural information processing systems*, pp. 883–891, 2010.

[22] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*, pp. 507–523, Springer, 2011.