An Extensible Thread Throttling Method for Multiple OpenMP Parallel Programs

XIAOXUAN LUO, South China University of Technology, Guangzhou, China
WEIWEI LIN, South China University of Technology, Guangzhou, China and Pengcheng Laboratory, Shenzhen, China
JIACHUN LI, South China University of Technology, Guangzhou, China
FAN CHEN, South China University of Technology, Guangzhou, China
HAOCHENG ZHONG, South China University of Technology, Guangzhou, China
KEQIN LI, Department of Computer Science, State University of New York, New Paltz, United States

OpenMP is one of the most popular parallel frameworks in the HPC area. Many researchers have proposed OpenMP thread throttling techniques for searching the optimal configuration of parallelism to improve computational efficiency. However, existing research mainly focuses on the optimal solution and ignores the average performance of the program during the search process. In addition, there are various types of workloads in HPC production environments. The OpenMP configuration needs to be adjusted according to the real-time running status of programs. Otherwise, it may lead to a deviation of the actual improvement in the real-time environment from the theory. In this article, we propose an OpenMP thread throttling method. The method uses the search results of historical workloads to train the performance vertex prediction model, quickly identifies the approximate range of the optimal number of threads for unknown workloads, and searches in a small range with a neighborhood-sampling-based bidirectional hill-climbing search algorithm. The method improves real-time optimization efficiency in HPC systems with multiple unknown loads. Through experiments, we demonstrate the advantages of our method compared to a variety of commonly used thread throttling methods. With minor differences in the optimal solutions, the average performance and convergence speed of our method during the search can be improved by up to 10.6% and 22.7% compared to the best method.

CCS Concepts: • **Computer systems organization** → **Real-time system architecture**; • **Theory of computation** → **Models of learning**; • **Software and its engineering** → *System administration;*

Additional Key Words and Phrases: Real-time system management, performance optimization, OpenMP, resource allocation, knowledge-sharing modeling

This work is supported by Shandong Provincial Natural Science Foundation Project (ZR2024LZH012), Guangxi Key Research and Development Project(2024AB02018), Guangdong Provincial Natural Science Foundation Project (2025A1515010113), and the Major Key Project of PCL, China under Grant PCL2025AS11.
Authors' Contact Information: Xiaoxuan Luo, South China University of Technology, Guangzhou, Guangdong, China; e-mail: 202210188550@mail.scut.edu.cn; Weiwei Lin (corresponding author), South China University of Technology, Guangzhou, Guangdong, China and Pengcheng Laboratory, Shenzhen, China; e-mail: linww@scut.edu.cn; Jiachun Li, South China University of Technology, Guangzhou, Guangdong, China; e-mail: jclee@scut.edu.cn; Fan Chen, South China University of Technology, Guangzhou, Guangdong, China; e-mail: 787078310@qq.com; Haocheng Zhong, South China University of Technology, Guangzhou, Guangdong, China; e-mail: cshczhong@mail.scut.edu.cn; Keqin Li, Department of Computer Science, State University of New York, New Paltz, New York, United States; e-mail: lik@newpaltz.edu.

ACM 1539-9087/2026/09-ART77
https://doi.org/10.1145/3769679

ACM Reference Format:
Xiaoxuan Luo, Weiwei Lin, Jiachun Li, Fan Chen, Haocheng Zhong, and Keqin Li. 2026. An Extensible Thread Throttling Method for Multiple OpenMP Parallel Programs. *ACM Trans. Embedd. Comput. Syst.* 25, 5, Article 77 (September 2026), 22 pages. https://doi.org/10.1145/3769679

1 Introduction

Modern HPC systems execute jobs using **thread-level parallelism** (**TLP**) to support large-scale computational tasks. OpenMP is a commonly used parallel framework in the HPC environment, which accelerates programs by splitting the parallelizable regions into multiple threads. Resource allocation is a major performance influencing factor for OpenMP programs. Since the performance of OpenMP programs will not grow linearly with the number of threads, increased overhead, such as data synchronization and shared memory access after reaching the parallel bottleneck can lead to degradation in the performance of OpenMP programs [19]. Considering that it is difficult for HPC users to accurately evaluate the optimal resources required for a job, thread allocation optimization for OpenMP programs has become an important way to improve system efficiency [26].

Thread throttling (or concurrency throttling) is a popular method for determining an OpenMP program's optimal resources. The primary optimization principle of thread throttling is searching the performance of feasible solutions in the parameter space to obtain a global (or nearly) optimal configuration of the program threads. Existing methods perform well in optimal solution searching. However, most existing algorithms require a large amount of sampling for a particular program to identify the changing patterns of performance. Any program changes will result in the optimization process needing to be rebuilt. Such changes include variations in the type of application, different application configurations for the same application (e.g., number of particles, size of the simulation space) and input data. All these changes affect the program behavior and thus the optimization of the optimal solution [35]. Therefore, real-time optimization capabilities are critical to address the various OpenMP programs in HPC production environments. In addition, existing methods only focus on the final optimization results without considering the program performance during the optimization process, which is not reasonable in practice. If some OpenMP programs run only a few repetitions, existing methods can not obtain enough sampling points to evaluate the optimal number of threads, resulting in a severe degradation of program performance. In this article, we refer to the above optimization problem as extensibility problem. An optimization algorithm with good extensibility is able to quickly converge to the global optimal solution in a wide variety of workloads, while maintaining a high average optimization effect during the search process.

In order to solve the extensibility problems, we propose a novel OpenMP thread throttling method. The optimization objective of our method is to improve the optimization efficiency and the program performance during the optimization process. We analyze the optimization results for known workloads with the knowledge migration theory as a premise, extracting cross-workload information to extend the optimization for unknown workloads. Specifically, the main contributions are:

- We propose an OpenMP performance characterization method combining Gaussian **radial basis function** (**RBF**) kernel and thread scalability evaluation algorithm, which can identify program performance bottlenecks by feature interactions and evaluate program performance trends without hardware resource constraints.
- We propose an OpenMP performance vertex prediction model. The model is trained using optimization results from multiple historical workloads to predict the approximate range of the optimal thread for an unknown workload, thus achieving multi-workload expanding for OpenMP performance optimization.

— We propose an OpenMP thread throttling algorithm that accelerates the search convergence with the OpenMP performance vertex prediction model and determines the optimal solution by a neighborhood-sampling-based bidirectional hill-climbing method. The algorithm can obtain a better solution in the preliminary sampling phase and improve the program performance during the optimization process.

The rest of this article is organized as follows: Section 2 summarizes related work. Section 3 presents the methodology of our method. Section 4 presents the detailed implementation. Section 5 presents experimental results. Section 6 summarizes this article.

2 Related Work

Thread throttling is divided into dynamic and static categories. Dynamic thread throttling determines optimal number of threads for each OpenMP parallel region, while static thread throttling uses only one specific thread throughout the program. The allocated resources in dynamic thread throttling will change frequently, leading to increased overhead in data migration and cache invalidation. It is difficult to achieve the theoretically optimal optimization effect in practical usage [30, 31]. On the other hand, the extra resources released by dynamic thread throttling can hardly be utilized to accelerate other OpenMP programs. This is due to the need to prevent the number of allocated threads from exceeding the number of cores. Therefore, static thread throttling is more robust and valuable for HPC production environments. However, dynamic thread throttling can also be modified to fit static scenarios, so this section does not strictly distinguish between dynamic and static algorithms.

In order to accommodate the differences caused by input set, processor architecture, and parallel regions, Lorenzon et al. [19] proposed a user-transparent OpenMP optimization technique called Aurora. The method sequentially evaluates scalability when the threads grow multiplicatively and exponentially, thus determining the optimal solution search range. Then, Aurora searches the selected space using hill-climbing to obtain the optimal solution with high speed and low overhead. Similarly, marques et al. [21] proposed TBFT optimize the number of threads in OpenMP programs using exponential search with an improved hill-climbing method based on binary search. Schwarzrock et al. [29] proposed Hoder, which uses the Fibonacci search strategy to obtain the optimal number of program threads. In node-level systems, using odd threads will lead to load imbalance, affecting program performance. Therefore, the authors exclude the odd number of threads from the search space to further speed up the optimal solution acquisition. In order to reduce the performance loss due to frequent adjustment of threads, Schwarzrock et al. [32] also proposed an optimization method based on the NUMA architecture for thread throttling results. The method takes the original thread optimization results as a basis and corrects them using the **Weighted Moving Average** (**WMA**) data smoothing technique to minimize the thread variations.

The above methods require a large number of samplings to obtain the optimal thread configuration, leading to high training costs. To reduce the optimization cost, scholars model factors such as scalability, performance, and power consumption to implement thread throttling algorithms. Bai et al. [4] analyzed the performance influencing factors of multicore programs and proposed a BPI model that considers the metrics of **memory-level parallelism** (**MLP**), **instruction-level parallelism** (**ILP**), and TLP . The authors classified parallel programs based on BPI features and built scalability prediction models for each type of program. With the help of scalability metrics, the authors designed a thread throttling method for determining the optimal number of threads. Coutinho et al. [7] used machine learning algorithms to solve the problem of high sampling overhead. The authors use multiple linear regression to establish the correlation between parameters and the energy efficiency of programs. It can use a small number of sample points to determine program energy efficiency values for various parameter configurations, thus guiding optimizing

decisions. The method can achieve an average prediction accuracy of 96% using only 1% of the sample space, and the search results are close to the optimal solution in most cases.

In addition, some studies combined extra factors with the number of threads to design optimization strategies, including processor aging [23], system temperature [22], and placement policies [20]. They are not presented in detail due to the deviation from our optimization scenarios.

3 Methodology

3.1 Basic Idea

The optimal thread of an OpenMP program is not a random result that cannot be evaluated, but rather an comprehensive result of multiple factors such as hardware architecture and program characteristics. Theoretically, it is feasible to evaluate the approximate performance of different thread configurations of an OpenMP program by profiling specific hardware and software [6, 34]. The main reason why existing methods are unable to perform multi-workload expanding is that, they cannot determine the relationship between the threads and the optimization objective in advance, thus requiring a large number of samples. In addition, the ultimate goal of building the performance function is to determine the optimal solution, and the rest of the sampling points are of little value.

Based on the above considerations, we utilize the knowledge migration theory to change the way of acquiring optimization information about unknown workloads from the sampling process to the knowledge extraction process [14]. Specifically, we do not directly establish the correspondence between the performance of an OpenMP program and the number of threads. Instead, the optimal solution is evaluated based on the system profiling data, avoiding wasting sampling points on low-value data fitting. System profiling data (e.g., hardware utilization and performance monitoring unit events) are important metrics for evaluating the running states of OpenMP programs. Many studies have proved that system profiling data can provide a reliable basis for performance analysis and modeling [1, 8, 17].

With system profiling data, the differences in optimal solutions due to multiple workloads can be transformed into differences in optimal solutions presented by multiple system states. Thus, the historical workload optimization information, which is challenging to utilize by traditional methods, can be obtained by knowledge extraction. Such information can be used to rapidly identify the approximate range of optimal solutions for unknown workloads. Figure 1 shows a schematic comparison of our method with the traditional method.

In terms of the optimization objective, **Energy Delay Product** (**EDP**) has gradually replaced performance as the main optimization objective in recent years. However, the optimal EDP solution of a program is not completely affected by the system efficiency but also by application characteristics such as job size, which makes it difficult to accurately predict using only system profiling data. The EDP formula is shown in Equation (1). Regularly, energy can be expressed as the product of power consumption and program time. In this case, the EDP grows quadratically with job runtime but linearly with average system power consumption. This causes a greater focus on system power consumption under small-scale programs and job performance under large-scale programs, resulting in deviation in the system state represented by the EDP optimum between workloads. Considering that EDP trades off both system overhead and program performance, a high thread number setting (lower EDP due to high performance) may have similar EDP values to a low thread number setting (lower EDP due to low overhead). This is not conducive to ensuring a unique mapping from the system profiling data to the optimal number of threads, making it difficult to correlate the pattern of change in the system profiling data with the prediction objectives.

$$EDP = Energy \cdot Time = Power \cdot Time^2. \tag{1}$$

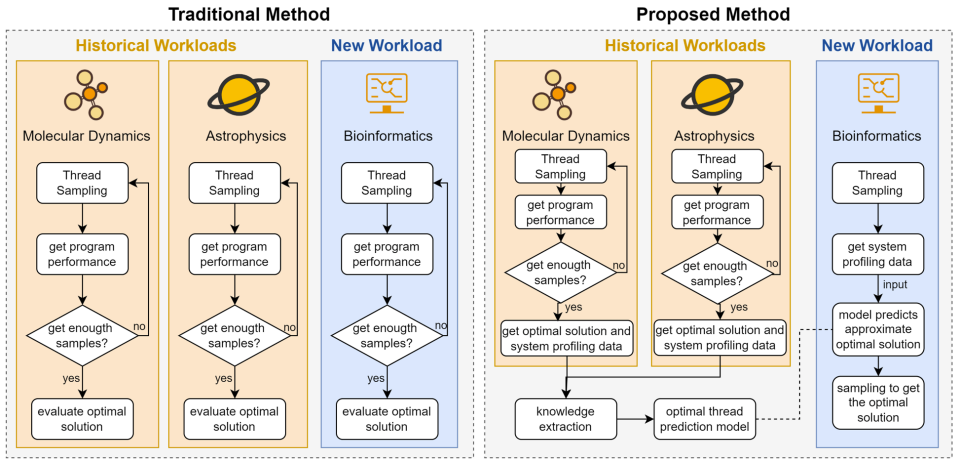

Fig. 1. Schematic diagram of the principle of multi-workload optimization for the traditional method and the proposed method.

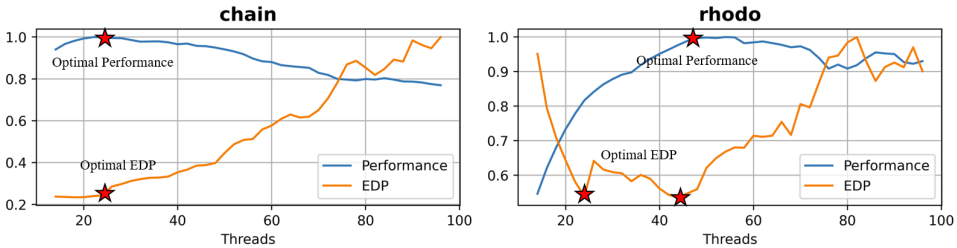

Fig. 2. Performance curves vs. EDP curves for chain and rhodo workloads. The horizontal coordinate is the number of threads used by the program. The vertical coordinate is the normalized performance and EDP.

In contrast, the performance vertex is a relatively stable value, that is, reached when the system parallel overhead exceeds the parallel gain. The system profiling data is sufficient to characterize the parallel efficiency without introducing more information characterizing the job properties, thus reducing the complexity of performance vertex prediction. As shown in Figure 2, the optimal performance solution corresponds to the same number of threads as the EDP optimal solution for the chain workload, and the difference between the two is insignificant under such workloads. However, for the rhodo workload, in addition to the EDP optimal solution corresponding to the performance optimal solution, another EDP optimal solution emerges due to the lower power consumption. This affects the unique mapping of the system profiling data to the optimal number of threads and reduces the accuracy of the optimal solution prediction.

3.2 Problem Statement

OpenMP thread throttling aims to optimize the objective function $L(t)$ by adjusting the number of threads t used by the program. In the research scenario of this article, the objective function is the runtime function T of the program. We try to obtain the performance optimal solution t^* for each OpenMP program with low cost.

$$t^* = \arg\min_t L(t) = \arg\min_t T(t). \tag{2}$$

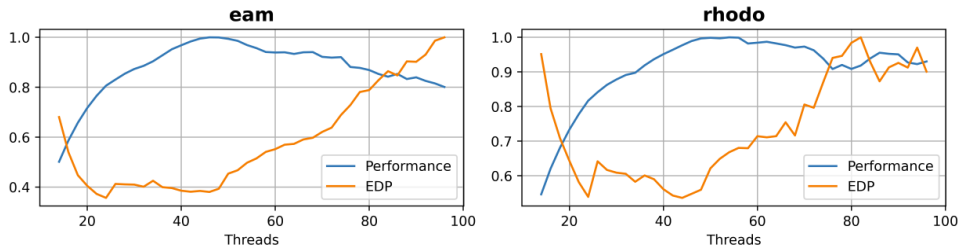

Fig. 3. Performance curves vs. EDP curves for eam and rhodo workloads. The horizontal coordinate is the number of threads used by the program. The vertical coordinate is the normalized performance and EDP.

Existing methods fall into two main classes of solution processes. The first one is the elimination-solving method, represented by the hill-climbing method. This method evaluates the program's performance at specific threads and eliminates some low-performance solutions to update the candidate solution set S. It will finally converge to the optimal solution t^* by an iteration loop.

$$S_{k+1} = \left\{ t_{new} | t_{new} \in S_k \; and \; \frac{\sum_{t \in S_{k+1}} L(t)}{|S_{k+1}|} < \frac{\sum_{t \in S_k} L(t)}{|S_k|} \right\}. \tag{3}$$
$$stop\, if |S_{k+1} = 1|, \; t^* = t_{k+1}$$

The second type is the model-based solving method. This type of method uses the performance result of n feasible solutions as the training data D of the agent model $f(t;\theta)$. The agent model can evaluate the trend between threads t and the optimization objective y, thus obtaining the optimal solution t^*.

$$D = \{(t_i), y_i | i = 1, 2, \ldots, n\}, \tag{4a}$$

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} [y_i - f(t_i;\theta)]^2, \tag{4b}$$

$$t^* = \arg\min_{t} f(t;\hat{\theta}). \tag{4c}$$

The above methods require a large amount of sampling in the process of obtaining the optimal solution, which may not be satisfied in the production environment. Moreover, the performance of the OpenMP program during the sampling process should also be used as an essential index to evaluate the merits of the optimization algorithm. They do not consider the performance of the OpenMP program during the sampling process as a metric to evaluate the optimization effect. We convert the evaluation function of the optimization algorithms from considering only the optimal solution (Equation (5a)) to a comprehensive consideration of the average performance during the entire optimization process (Equation (5b)). The number of samples n is identical for all comparison algorithms, and its value should be large enough to ensure that most of the optimization algorithms can search for the optimal solution. The new evaluation function considers both the optimization efficiency and the optimal solution effect. It can comprehensively evaluate and compare different optimization algorithms.

$$E_{old} = L(t^*) = T(t^*), \tag{5a}$$

$$E_{new} = \sum_{i=1}^{n} L(t_i) = \sum_{i=1}^{n} T(t_i), \tag{5b}$$
$$where\, t^* \in S = \{t_i | i = 1, 2, \ldots, n\}.$$

Existing methods perform extensive sampling to identify the performance distribution of an OpenMP program. Considering that only the optimal solution is ultimately valuable, we transform the OpenMP thread throttling problem from a single-workload independent regression problem to a multi-workload joint prediction problem, thereby significantly reducing the sampling overhead. Specifically, multiple OpenMP thread throttling optimization tasks have some similarities. Figure 3 illustrates two workloads with similar performance and EDP trends. The system profiling dataset M can characterize workloads' operating behavior. Meanwhile, the operating behavior will determine the efficiency of the system hardware, which will affect the specific value of the optimal solution. Therefore, we train the optimal solution prediction model P by generalizing the knowledge from the composition dataset D, which includes the system profiling data and the optimal solution of n workloads. For an unknown workload, we initially perform a few samples to obtain the runtime system profiling dataset M_{new}. Then, the local optimal solution $\hat{t}^*_{new}$ is determined with the optimal solution prediction model. Eventually, the global optimal solution t^*_{new} is converged by some neighborhood sampling.

$$D = \{(M_i, y_i)|i = 1, 2, \dots, n\}, \tag{6a}$$

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} [y_i - \hat{P}(M_i; \theta)]^2, \tag{6b}$$

$$\hat{t}^*_{new} = P(M_{new}; \hat{\theta}). \tag{6c}$$

Based on the above prediction method, we are not only able to significantly reduce the number of samples, but also improve the extensibility of the optimization algorithm. Existing thread throttling methods are only effective for specific programs. They need to re-sample and perform feasible solution reduction or construct agent models for new unknown programs. In contrast, our method has the natural advantage of extensibility and can accelerate the optimization of unknown programs by leveraging the optimal solution information of historical programs.

3.3 Feasibility and Challenge

In order to assess the feasibility and challenges of the proposed method, this section conducts preliminary experiments and analyses using various benchmarks. A detailed description of the experimental environment and workload will be introduced in Section 5.

For the feasibility verification of optimal solution prediction modeling, two workloads are randomly selected as test workloads for each of the three applications NPB, LAMMPS, and PENNANT. The system profiling data and number of threads corresponding to the highest performance for the remaining workloads (10 in total) comprise the training dataset. The Random Forest Regression algorithm is used to construct performance vertex prediction model. On the one hand, the type of system profiling features that characterize the differences in optimal solutions for multiple workloads are difficult to identify manually. It requires feature selection with the help of algorithms such as Random Forest Regression. On the other hand, a relatively simple model prevents the model's own design from affecting the feasibility assessment of performance vertex prediction modeling, while reducing the probability of model overfitting.

Figure 4 summarizes the sampling optimal number of threads (true value), the average of the predicting optimal number of threads, and the MAE for the six test workloads. Regarding MAE, most of the test workloads have large prediction errors. This phenomenon is reasonable due to the relative simplicity of the model. Regarding the relationship between the number of optimal solutions for different workloads, the model can identify differences in the distribution of optimal solutions to some extent by system profiling data. It tends to output larger values for relatively

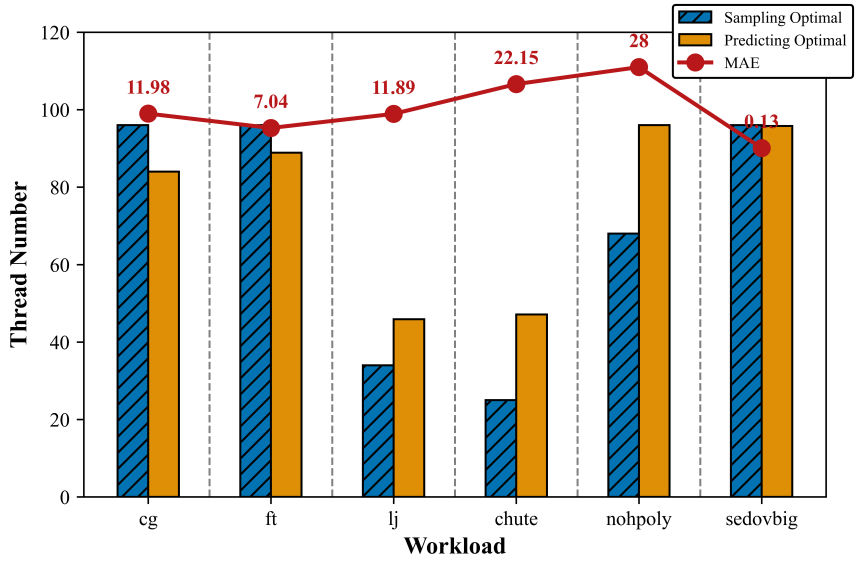

Fig. 4. Test results for constructing a vertex prediction model for OpenMP program performance using random forest regression.

large-scale workloads such as cg, ft, and sedovbig, while also recognizing small workloads such as lj and chute. However, the model performs poorly in distinguishing subtle differences (lj, chute) and some workloads (nohpoly).

Possible reasons for the poor performance of the above model include:

— The number of the optimal threads is heavily influenced by performance bottlenecks. Many OpenMP programs contain multiple bottlenecks interacting with each other. This makes models that consider each feature in isolation less effective and require consideration of the joint effect of multiple bottlenecks [25].
— Due to hardware limitations, many programs achieve maximum performance when running with all cores. The optimal solutions are similar does not mean these workloads have similar operating behavior. Take Figure 5 as an example, the performance growth rate of cg is higher than ft. Cg should have a numerically larger optimal solution when computational resources are adequate. Therefore, the impact of hardware resource constraints needs to be taken into account when establishing the mapping of system profiling data to the optimal number of threads.
— Because of noise and fluctuations, the system profiling data cannot be completely accurate, and there is no guarantee that the performance vertex prediction model will maintain a high accuracy. Therefore, sampling in the neighborhood of the predicted optimal solution is needed to determine the true optimal solution.

In summary, although simply using random forest regression to model the performance vertex of OpenMP programs does not work well, it can indirectly demonstrate the feasibility of using system profiling data to evaluate the differentiating features of programs. We should consider challenges such as correlation between features, OpenMP program scalability assessment, and local optimization of suboptimal solutions.

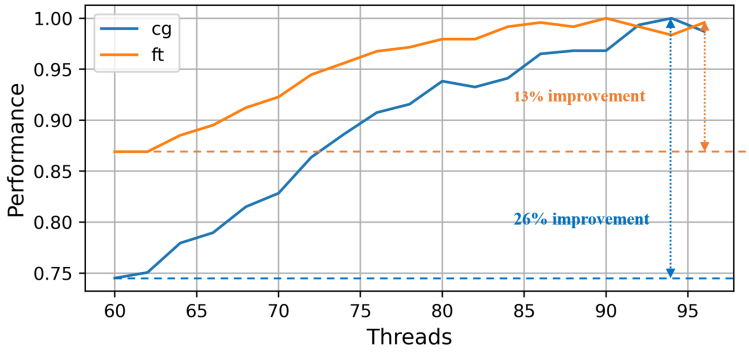

Fig. 5. Normalized performance variation curves for cg vs. ft. Both workloads have similar performance optimal solutions, but their performance growth rates differ greatly.

4 Method Design

4.1 Characterization of Performance Trends

The optimal performance solution for OpenMP programs results from multiple bottleneck interactions. To jointly identify performance bottlenecks exhibited by multiple systems profiling data, we use the Gaussian RBF kernel to map the original feature space to a high-dimensional feature space. The RBF kernel can accurately evaluate nonlinear relationships between features and efficiently handle the complexity of high-dimensional feature spaces [3]. In our scenario, the joint action paradigm for system profiling data is unclear, and the bottleneck features have local similarities, which is more suitable for the RBF kernel. The composition of the high-dimensional dataset D_h is shown in Equation (7a). The kernel matrix K is obtained by applying the RBF kernel transform to the system profiling dataset M for n workloads. The composition of the elements in the kernel matrix is shown in Equation (7c), where k is the RBF kernel, m_i represents the ith feature in the system profiling dataset M, and σ is the kernel bandwidth parameter.

$$D_h = \{(RBF(M^i), y^i) | i = 1, 2, \ldots, n\}, \tag{7a}$$

$$RBF(M^i) = K^i, \tag{7b}$$

$$K_{ij} = k(m_i, m_j) = \exp\left(-\frac{\|m_i - m_j\|^2}{2\sigma^2}\right). \tag{7c}$$

For the program scalability problem, i.e., the differential behavior cannot be evaluated between multiple OpenMP programs whose optimal number of threads takes a value close to the total number of cores. We evaluate the optimal solutions under non-hardware constraints by scaling the optimal number of threads according to the difference in performance decay rates. The pseudo-code is shown in Algorithm 1. We compute the acceleration ratio on the normalized performance at a granularity of 0.1 and evaluate the overall acceleration ratio decay rate by the difference between neighboring acceleration ratios. Eventually, the combination of acceleration ratio values for performance from 0.9 to 1.0 is used to evaluate the number of threads when the program reaches peak performance (the first time the acceleration ratio is negative). Different hardware and software environments may affect the selection of specific scaling granularity. The optimal granularity setting can be selected by testing the impact of multiple granularities such as 0.1, 0.15, 0.2, and so on. on the accuracy of the performance vertex prediction model.

ALGORITHM 1: Thread Scalability Evaluation Algorithm.

Input: The sampling performance dataset P of a OpenMP program whose optimal number of threads closes to the number of cores.
Output: The scaled optimal number of threads t_s without hardware resources limitation.
for performance value p in [0, 1, 0.1] **do**
Get the first number of threads t_p whose performance exceed p.
Add t to the set S that used to calculate the acceleration ratio.
end for
for all index i in S **do**
$acc = 1/(S[i+1] - S[i])$
Add acc to the acceleration ratio set A.
end for
Calculate the average loss l of acceleration ratio by difference the neighboring elements in A.
Get the last two threads $t_{0.9}$, $t_{1.0}$ from S and the last acceleration ratio $acc_{0.9-1.0}$ from A
The additional threads $t_{add} = (acc_{0.9-1.0} - l)/l * (t_{1.0} - t_{0.9})$
The scaled threads $t_s = t_{1.0} + t_{add}$

After using the above algorithm, the optimal number of threads for cg and ft in Figure 5 are scaled to 121 and 96, respectively. It prevents programs with different running behaviors from generating the same optimal number of threads due to hardware resource constraints. The subsequent evaluation section will compare the accuracy of models with or without scaling to the optimal number of threads using scalability evaluation to demonstrate the effectiveness of the above approach.

4.2 Performance Vertex Modeling

While the RBF kernel is effective in obtaining the joint interaction relationship between features, it also leads to an explosive growth of the feature space. Most system profiling features are non-significant and do not accurately reflect the differences in performance bottlenecks across OpenMP programs. Moreover, different programs may have distinct performance influences, making it difficult to use a few specific metrics to construct a model applicable to multiple workloads [5, 9]. To reduce the overhead of model training, feature dimensionality reduction is required before using the RBF kernel to ensure the validity of the information in the high-dimensional feature space.

Unfortunately, it is difficult to apply common reduction techniques in our scenario. Unsupervised dimensionality reduction algorithms such as PCA are mainly realized by eliminating redundant information, which can eliminate some of the low variance irrelevant features. However, the fact that most system profiling features change with program performance does not mean these features help identify the differences in multi-workload performance vertex. For example, the memory bottleneck of a program is not large, but the amount of memory accesses may change as performance changes. At this point, it is not reasonable to use memory accesses as a constraint on the optimal number of threads for the program [36]. Such patterns of change in data that are irregular or non-value to modeling are challenging to identify by existing unsupervised dimensionality reduction algorithms. For supervised feature selection algorithms [16, 27], it is difficult to construct a reasonable objective function to obtain a set of effective features in the low-dimensional space. As mentioned before, the interaction of some of the bottlenecks affects the value of the optimal number of threads, so some of the significant features do not correlate strongly with the prediction objective in the low-dimensional space.

To identify the interaction between features in high-dimensional space and the performance of OpenMP programs, we improve the SAEA-PRG feature screening algorithm. SAEA-PRG [18]

can effectively select feature features in high-dimensional space by using more efficient sampling algorithms with a more accurate agent model. On this basis, we introduce the kernel function to improve the algorithm's ability to analyze the joint influence of features. The proposed KSAEA-PRG (Kernel SAEA-PRG) algorithm retains the original algorithm in the data sampling and random grouping mechanism. In evaluating feasible solutions for each grouping, we use the RBF kernel to map the original feature space into a high-dimensional feature space, thus filtering out the interaction features that can characterize the values of OpenMP performance vertices. KSAEA-PRG is able to solve the problem of difficult identification of performance bottleneck interactions using the RBF kernel and prevent the problem of excessive complexity of feature selection in high-dimensional space with the efficient feasible solution evaluation mechanism of SAEA-PRG.

In terms of specific prediction algorithm selection, we prefer to use lightweight algorithms rather than complex algorithms such as deep neural networks. On the one hand, training complex models requires collecting large amounts of data, which is inconsistent with our goal of maximizing the average performance during optimization and inexpensively expanding to unknown workloads. On the other hand, the main advantage of complex models is the ability to reduce the cost of manual intervention and data processing through their powerful analytical capabilities. However, we have already performed high-dimensional data transformation, feature analysis, and hyperparameter optimization. The lightweight algorithms with high-quality datasets are not necessarily worse than the complex algorithms. Finally, considering that the actual applied environment of OpenMP thread throttling is a server using the CPU as the main computational resource. Even if the complex model can run properly, the optimization gain will be reduced due to the running overhead of the algorithm itself.

The OpenMP performance vertex prediction model can be applied to quickly evaluate the optimal number of threads based on the unknown workloads' operating behavior, improving the optimization algorithm's extensibility. Nevertheless, such extensibility does not mean the model is perfectly adapted to all unknown workloads. For unknown workloads whose operating behavior is dramatically different from the historical workloads, the filtered characteristics may not contain enough valid information to guide the vertex judgment of the unknown workload. Therefore, the workloads used to train the performance vertex prediction model should be typical, with characteristics covering most possible OpenMP applications. The study of OpenMP program typicality analysis is beyond the scope of this article. We assume that the experimental workloads are typical. In addition, it is not common for HPC users to submit jobs that are completely unknown. Most user jobs originate from applications deployed by system administrators. Reports published in real supercomputers [13, 33] show that most of the jobs originate from a few application types, and jobs from the same application have some similarity. Therefore, the algorithm proposed in this article is adapted to typical HPC optimization scenarios.

4.3 Local Search Strategy

In HPC production environments, instability factors such as workload fluctuations and monitoring noise are prevalent [24, 37]. The model cannot achieve utterly accurate performance vertex prediction. The performance vertices output by the model can only be used as a reference value. Searching within a specific range of the reference value is necessary to verify the actual performance vertex.

Theoretically, the reference value should be located near the global optimal solution. Optimization needs to be performed only in a small neighborhood of the reference value, thus skipping most unnecessary and low-performance sampling points to improve the average performance of unknown workload during the optimization. Traditional thread throttling methods search the entire thread space from a fixed starting position. It is a unidirectional, wide-range search scenario, so search

ALGORITHM 2: Neighborhood-Sampling-based Bidirectional Hill-Climbing Search Algorithm.

Input: The system profiling data D of an OpenMP program, the upper quartile of performance vertex prediction error e, the performance vertex prediction model M.
Output: The global optimal number of threads t^* of the OpenMP program.

```
Predict the reference optimal number of threads t̂* using D and M.
Determine the local search range R = [t̂* − e, t̂* + e]
low = t̂* − e
high = t̂* + e
mid = t̂*
while mid == low or mid == high do
    Sampling the performance P of low, high and mid (skip existing samples)
    if P_mid is maximum then
        low = mid − (mid − low)/3
        high = mid + (high − mid)/3
    else if P_low is maximum then
        low = low − (mid − low)
        high = mid
        mid = (low + high)/2
    else
        low = mid
        high = high + (high − mid)
        mid = (low + high)/2
    end if
end while
t* = mid
```

algorithms such as exponential search and Fibonacci search are used. In contrast, our scenario is a bidirectional small-range search. We implement a neighborhood-sampling-based bidirectional hill-climbing search algorithm to determine the global optimal number of threads. The algorithm combines the reference value output from the performance vertex prediction model, the model accuracy, and the improved binary search (Algorithm 2).

First, the algorithm uses the performance vertex prediction model to identify referenced optimal number of threads and uses the model's accuracy to determine the search range. An error between the predicted and actual values is calculated for each test workload during the model's training. The errors are sorted from low to high and the upper quartile error is selected as neighborhood size. This balances effectiveness and overhead by covering most programs' error range and preventing a few programs with poor predictions from leading to an oversized search space.

Subsequently, the algorithm performs the sampling with an improved binary search (Figure 6). Unlike ordinary binary search, we can not guarantee the optimal solution lies in the initial sampling interval. When the highest performance point lies in the interval boundary, the algorithm will extend the upper or lower bound of the search interval. This mechanism allows the search interval to be gradually adjusted to the vicinity of the actual optimal solution when the predicted vertices deviate significantly from the actual values, preventing poor solutions due to incorrect vertex predictions. When the midpoint has the highest performance, the sampling points of the next iteration do not directly use the median values of the left and right intervals. It adopts an asymmetric approach because the global optimal solution should be closer to the reference optimal solution. In summary, the algorithm will gradually narrow the search range and finally converge.

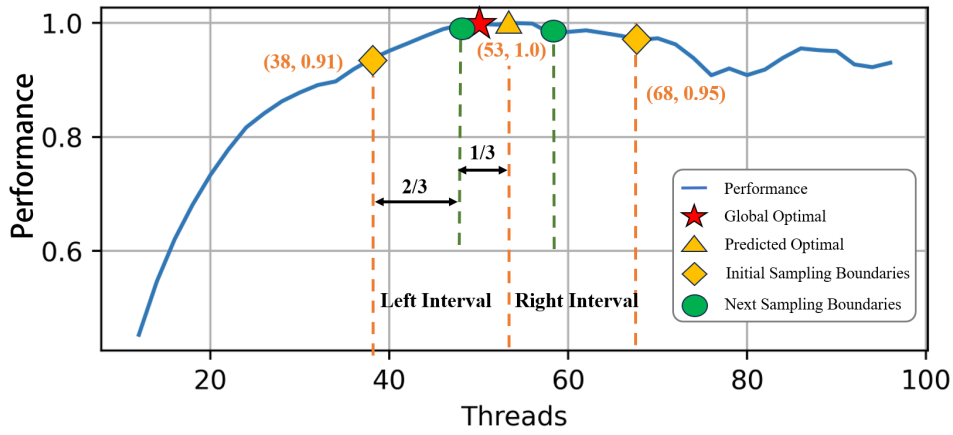

Fig. 6. Schematic diagram of the search process of neighborhood-sampling-based bidirectional hill-climbing search algorithm. It shows the sampling points of the first two searches.

The complexity of the proposed search algorithm is $O(log_3 2e)$, where e is the upper quartile of performance vertex prediction error. In the worst case, the algorithm requires $\lceil \frac{\hat{e}-e}{e} \rceil + \lceil 2 \log_3 2e \rceil + 1$ samples to converge to the optimal solution, where $\hat{e}$ is the error between the predicted and actual values of the performance vertex. When inaccurate model predictions result in an initial search range that does not cover the optimal solution, the algorithm needs to shift the search interval $\lceil \frac{\hat{e}-e}{e} \rceil$ times. Each shift is a distance e and requires an additional sampling. After determining the search interval (of length $2e$), at least $\log_3 2e$ iterations are needed to converge to the optimal solution. Each iteration requires at most two additional samples (when the maximum performance is P_{mid}). However, the initial interval evaluation needs to sampling three times and therefore needs to be plus one in the end.

5 Evaluation

Table 1 describes the hardware and software configurations in the experiments. To be more consistent with the HPC production environment, in addition to the commonly used NPB [10], we have included two actual HPC applications, PENNANT [11] and LAMMPS [15].

5.1 Data Set

We collected system profiling data and runtimes for all benchmarks running at different numbers of threads with a step of two. Each sample was repeated five times and averaged. Multiple tools were used to obtain the operational status of different server components, including the Linux /proc filesystem, perf, and Redfish. Specifically, the dataset used for the experiments contains the following system profiling features (218 types):

— **Server Status:** Server power, CPU power, memory power, cpu temperture, and so on.
— **CPU Utilization:** User, system, nice, idle, and iowait utilization.
— **Memory Usage:** Capacity of memory allocated to the cache, buffer, anonymous pages, and so on.
— **I/O Events:** Number of I/O requests, throughput, queuing delay, utilization, and so on.
— **PMU Events:** Cycles, instructions, IPC, cache hits, TLB hits, branch prediction, pipeline stalls, and so on.

Table 1. Experimental Environment

System Configuration	
CPU	Kunpeng920 7260
Memory	16*HMA84GR7JJR4N-WM (32 GB)
Disk	SAMSUNG MZ7LH480 (480 GB)
Kernel	Linux 4.18.0
OS	CentOS 7.7
Parallelism	OpenMP 4.5

Application Configuration		
Type	Version	Benchmarks
NPB	3.4.3	bt, cg, ep, ft, mg, and sp
PENNANT	0.9	leblancbig, nohpoly, nohsquare, sedovbig, and sedovflat
LAMMPS	2023-Aug-Stable2	chain, eam, lj, chute, and rhodo

— **Microarchitecture Bottlenecks:** Front-end bound, bad speculation, retiring, back-end bound, and so on.
— **NUMA Access:** Numa hits and misses, local memory allocation, foreign memory allocation, and so on.

5.2 Necessity Experiment

This section experimentally validates the necessity of our designs for improving the prediction accuracy of OpenMP performance vertices, including high-dimensional spatial transformations, scalability evaluation, and feature selection. The experiment compared the prediction accuracy of several modeling methods for unknown OpenMP program vertices:

— **None:** Train the performance vertex prediction model without any processing of the training data.
— **PCA:** Dimensionality reduction of training data using PCA.
— **Kernel PCA:** Dimensionality reduction of training data using Kernel PCA.
— **spFSR [38]:** Dimensionality reduction of training data using spFSR, a feature selection algorithm based on statistical analysis.
— **SAEA-PRG [18]:** Dimensionality reduction of training data using SAEA-PRG.
— **KSAEA-PRG:** Dimensionality reduction of training data using the KSAEA-PRG proposed in this article.
— **KSAEA-PRG+Scaling:** Adjusting prediction targets for training data Using the thread scalability evaluation algorithm Prior to KSAEA-PRG dimensionality reduction.

The prediction algorithm used in all tests was the Random Forest regression algorithm to remove the influence of the algorithm itself on accuracy. To evaluate the error of each modeling method, we randomly selected ten workloads as training workloads. Their profiling data with the performance optimal solution were used as training data to build the performance vertex prediction model. The remaining six workloads were used as test workloads, and their profiling data were input into the performance vertex prediction model. The model-predicted values were compared with the actual performance optimal solutions to calculate the MAE as the errors. The above process was repeated thirty times to cover most training and test workload combinations. Figure 7 shows the error box-plots with tables for each modeling method.

Mean Error	15.92	14.89	13.08	18.28	12.04	10.29	9.63
Error STD	8.86	7.69	5.23	8.57	10.25	4.51	5.35

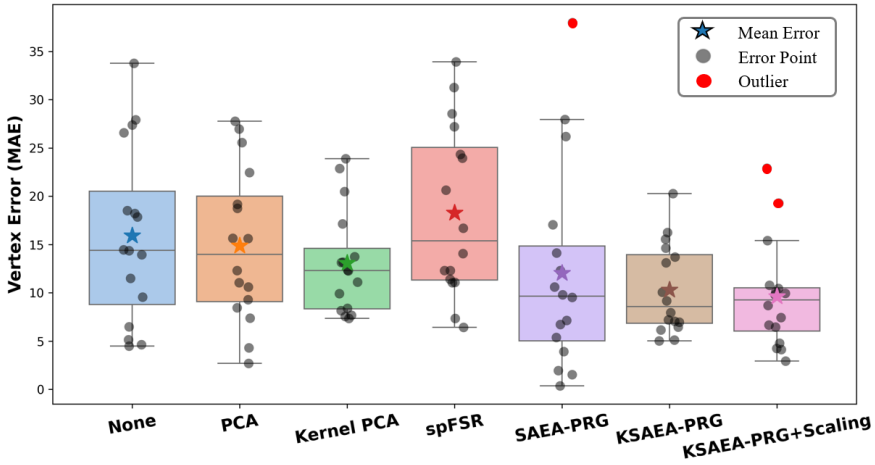

Fig. 7. Error distribution and summary of different modeling methods for predicting performance vertices.

In terms of the average prediction errors, the accuracy of the performance vertex prediction models is improved after performing feature selection in most cases, demonstrating the necessity of feature selection on system profiling data. spFSR achieves feature selection mainly through statistical analysis. It does not consider the effect of features on model accuracy and the interaction between features, so there is a negative gain. Similarly, PCA mainly reduces redundant features to achieve dimensionality reduction, which can decrease the interference of irrelevant features on the model to a certain extent. However, its accuracy enhancement effect is limited, only 6.5% compared to None. Even so, the accuracy can be further improved in Kernel PCA using kernel transformations, which increases the prediction accuracy by 12.2% compared to PCA, proving the effectiveness of the RBF kernel for identifying the interactions between features. Due to the excellent behavior in coping with high-dimensional feature spaces and filtering significant features most beneficial to the model, SAEA-PRG outperforms other methods even without using our designs. The model accuracy is dramatically improved after using the KSAEA-PRG proposed in this article. Combined with the thread scalability evaluation algorithm mentioned in Section 4, the model accuracy is further improved. Eventually, the average accuracy of the performance vertex prediction model built using our method is improved by 39.5% compared to the model built without any data processing, and there is still an improvement of 20% compared to the accuracy of the SAEA-PRG model. In the upper quartile, our model improves 47.1% and 28% compared to None and SAEA-PRG, respectively.

In terms of the error distribution across workloads, the performance vertex prediction model using the kernel space transformation has a minor variance. Both the upper and lower bounds of error are closer to the mean and median for both the Kernel PCA vs. PCA comparison and the KSAEA-PRG and SAEA-PRG comparison. This means the model is more stable and does not predict some workloads exceptionally well or some exceptionally poorly. One plausible explanation is that by establishing interactions between bottlenecks, the model recognizes the generic mapping of profiling data to performance vertices rather than just a training data fitting. In PCA and SAEA-PRG, while some workloads achieve very high prediction accuracy, there are also some workloads whose

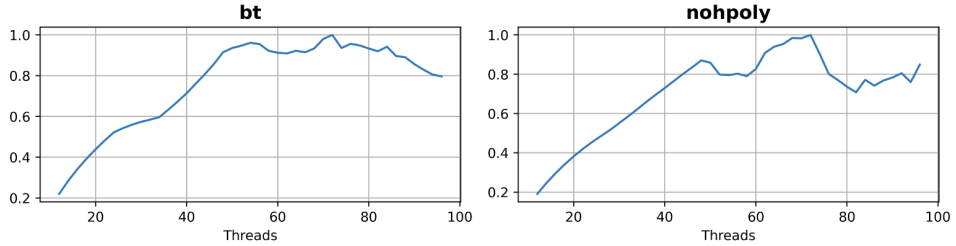

Fig. 8. Performance change curves of bt and nohpoly.

predicted values deviate significantly from the actual values. As mentioned earlier, some workloads have similarities in profiling data and performance trends (Figure 3), and a large amount of training data with similarity under the influence of certain random factors may overfit the model. Such overfitting models may not accurately predict the performance vertex of the test workloads.

In addition, there are some outlier workloads where the prediction error deviates significantly from the mean. The main reason is described in Section 4.2, i.e., the performance trends of some workloads may not be consistent with all the training workloads, resulting in the model not including the performance vertex prediction patterns for such unseen workloads. In the KSAEA-PRG+Scaling box-plot, the two outlier workloads are bt and nohpoly (Figure 8). They have multiple distinct performance vertices, whereas other workloads have only one performance vertex or are monotonically increasing (performance peaks at the maximum number of cores). The performance trends can reflect the resource usage characteristics of workloads to some extent and the mapping model of the system profiling data to the performance vertices. Since these two workloads have significantly different characteristics from the others. In most cases, the model cannot effectively utilize the training data to predict the performance vertex of these two outlier workloads.

5.3 EDP Vertex Prediction

Section 4.1 analyzes the differences between performance vertices and EDP vertices. Multiple EDP vertices may occur because the EDP is affected by both performance and power factors. This leads to the difficulty for the prediction model to recognize a uniform mapping pattern from the system state to the optimal number of threads. We conclude that performance vertex prediction is more reasonable compared to EDP vertex prediction.

This section uses a similar approach to EDP vertex modeling as in the previous section, and experimentally compares the difference in accuracy between EDP vertex prediction and performance vertex prediction. When the prediction target is transformed from the performance vertex to the EDP vertex, five benchmarks are changed, namely eam (45→24), rhodo (47→24), mg (55→42), leblancbig (96→51), and nohpoly (68→48). The prediction results are shown in Figure 9.

Different models in the EDP vertex prediction behave similarly to the performance vertex prediction. Compared to PCA and spFSR, SAEA-PRG has higher prediction accuracy due to its excellent feature selection capability. After introducing the kernel function, the degree of dispersion of data points in PCA and SAEA-PRG decreased significantly. Thread scalability evaluation can also improve the prediction accuracy. However, the EDP predictions perform much less well than the performance predictions in terms of overall prediction accuracy values. Each EDP model has an average 44.93% decrease compared to the performance model with the same method. The prediction effect of the proposed model also decreases by 48.18%. This means that a few unstable prediction targets have a pronounced negative impact on the entire model.

	None	PCA	Kernel PCA	spFSR	SAEA-PRG	KSAEA-PRG	KSAEA-PRG+Scaling
Mean Error	22.77	21.93	19.54	23.98	17.04	15.84	14.27
Error STD	11.26	13	10.45	10.89	14.1	8.28	7.86

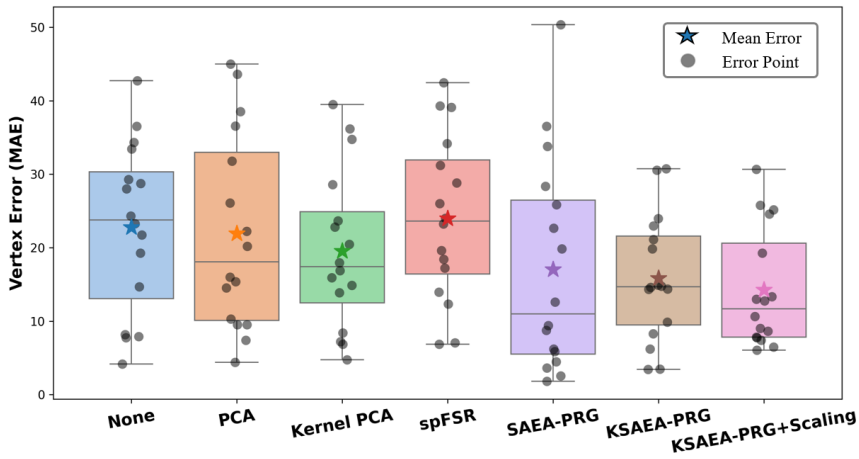

Fig. 9. Error distribution and summary of different modeling methods for predicting EDP vertices.

5.4 Comparison Experiment

This section will compare the proposed thread throttling algorithm with existing algorithms. The evaluation includes the performance of the optimal solution, the convergence speed, and the average performance during the entire optimization process. Comparison algorithms are:

- **Default:** OpenMP will run the program using all cores by default.
- **TBFT:** The thread throttling in TBFT [21].
- **Hoder:** The thread throttling in Hoder [29].
- **Bayesian:** Bayesian optimization is a popular parameter optimization algorithm based on the agent model. We implemented an optimization algorithm for OpenMP threads using the TPE sampler in Optuna [2].

Each algorithm performed thirty search iterations, which was enough for every algorithm to converge to the optimal solution. The server ran the workloads using the optimal number of threads after convergence. For the Default optimization, all thirty iterations were run using all cores. For Bayesian optimization, the algorithm was considered to converge after its optimal solution remained unchanged for five iterations. Every thirty rounds of iteration served as a complete test, and the new test restarted the search without using the historical sampling results. Since the Default, Hoder, and TBFT optimization processes were less affected by random factors, only three complete tests were performed for each workload, and the results were averaged. The sampling distribution strongly influenced the effectiveness of Bayesian optimization. Twenty complete tests per workload were performed, and the results were averaged. Our method was influenced by the performance vertex prediction model, and the final results were also averaged over twenty complete tests. For each test, ten workloads were randomly selected for modeling, and the effect of optimizing the remaining six workloads from scratch was tested.

The experimental results are shown in Table 2. The experiment reflects the search efficiency of the optimization algorithm through the average performance of workloads in the first ten iterations, the first twenty iterations and all iterations, the search effectiveness through the performance of the optimal solution, and the search speed through the number of convergence iterations.

Table 2. Optimization Result of Different Algorithms

Algorithm	Default	TBFT	Hoder	Bayesian	**Ours**
Performance (First 10)	0.7848	0.6141	0.7584	0.6693	**0.8676**
Performance (First 20)	0.7848	0.7619	0.8271	0.7582	**0.8945**
Performance (All)	0.7848	0.8135	0.8495	0.8077	**0.9035**
Optimal Performance	0.7848	0.9167	0.8943	0.9404	**0.9216**
Convergence Iteration	1	13.5	11.37	20.97	**8.79**

Bold text indicates the optimization effect of the algorithm proposed in this paper.

Hoder utilizes Fibonacci search to quickly skip poorly performing feasible solutions in the pre-search phase. It performs better in terms of search efficiency as well as convergence speed. However, even for the best-performing Hoder, the average performance for the first ten iterations is not as good as if run with all cores. Since the table is the average result of multiple realistic machine runs, even with the same thread configuration, some fluctuation still exists over multiple runs. When the optimal solution performance reaches about 0.95, we consider it has acquired the global optimal solution. From the results, TBFT and Hoder do not search for the global optimal solution in most tests. The main reason is the fluctuation problem mentioned above. Fluctuations may guide TBFT and Hoder to search in the wrong direction when the workload performance of the search boundaries is similar. Bayesian optimization evaluates trends in workload performance based on an agent model and may sample multiple times for the same thread configuration. Therefore, Bayesian optimization can escape from the interference of workload fluctuations and converge to (or nearly) the global optimal solution. However, its average workload performance is not significantly superior due to the slow convergence.

Our method has advantages in most aspects. Regarding the average performance during the search, Our method utilizes the predicted vertices to obtain a better feasible solution during the preliminary sampling phase. It improves the search efficiency by 10.6%, 8.1%, and 6.4% compared to the best comparison algorithm in the first ten iterations (Default), the first twenty iterations (Hoder), and all iterations (Hoder). Since our optimization goal is to maximize the search efficiency for unknown workloads, the improvement decreases as the number of sampling points increases. Our method is also affected by workload fluctuations; thus, the optimal solution performance is worse than Bayesian optimization's. However, most search boundaries are located near the optimal solution, and the fluctuations do not guide the algorithm in searching in a direction that deviates significantly from the optimal solution. Our method still has advantages compared to TBFT and Hoder. Regarding the convergence speed, our method shows a significant improvement compared to most compared algorithms. Compared to TBFT, Hoder, and Bayesiam, the improvements are 34.9%, 22.7%, and 53%, respectively.

Figure 10 shows the sampling process for a particular complete test of the five optimization algorithms, from which the features described above are also reflected. In this test, Hoder convergence values deviated significantly from the other algorithms. Bayesian optimization happened to sample the global optimal solution at the 16th iteration and converged after five iterations. The final converged values of TBFT and our algorithm are also close to the global optimal solution. Regarding search scope, TBFT and Bayesian optimization essentially searched the entire thread space. The Hoder searched in a relatively more minor scope but still across about fifty cores. Our algorithm spans under forty cores and narrows the search to about ten cores after the first five iterations.

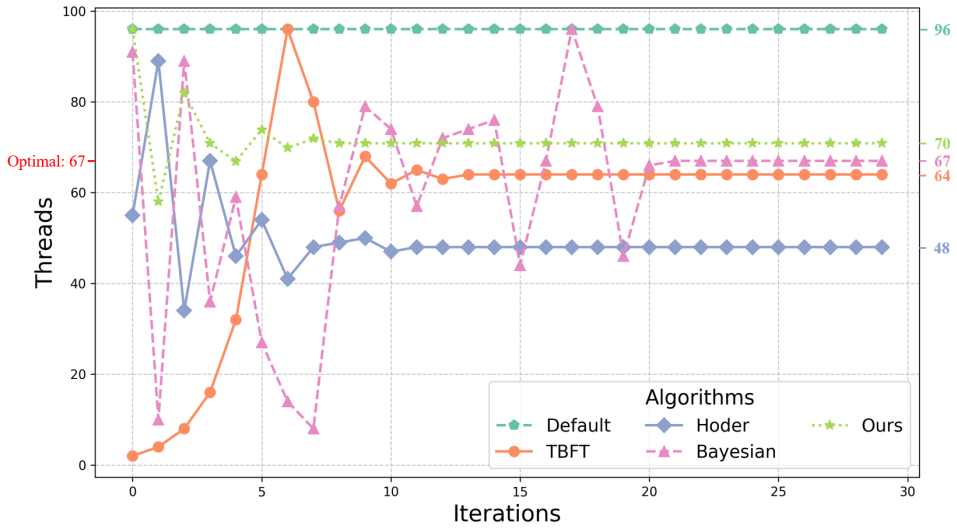

Fig. 10. Sampling distribution of one complete optimization of different optimization algorithms.

5.5 Discussion

In this section, we further analyze the experimental results and our method. According to the experiment, we can verify the effectiveness of some method designs.

Verification 1. Existing optimization frameworks treat the search processes for different workloads as completely independent. However, we believe that the search processes of multiple workloads can be guided by each other, and the extensibility of optimization algorithms can be significantly improved by extracting their knowledge. In this article, we demonstrate the feasibility of the new optimization framework in the OpenMP thread throttling scenario, i.e., using system profiling data to predict the approximate number of threads for different OpenMP workloads to reach maximum performance.

Verification 2. Due to the complex processes involved in instruction computation, inter-process communication, data synchronization, and so on., some server components may not be able to reach their peak physical performance. In addition to the values of individual system profiling feature, the interactions between different features can more accurately assess the joint efficiency of server components when running an OpenMP program.

Verification 3. The EDP behavior of an OpenMP program is influenced by multiple factors, including the program itself and the power consumption of the server. This leads to a more complex analysis and less accurate modeling of EDP compared to performance.

Verification 4. The similarity of multiple OpenMP programs affects the results of characterization and optimization. A single model may not be able to cope with all types of workload in a complex environment. In addition, a uniform mapping pattern of input data to predicted targets may also not exist between different types of workloads. Workload classification and multi-model collaboration in large-scale HPC clusters are necessary.

We also have observed some problems that may be encountered with thread throttling optimization in production environments.

Observation 1. When evaluating the effect of an optimization algorithm, it is not reasonable to consider only the performance of the optimal solution. If the convergence speed of the optimization algorithm is slow, its overall optimization effect may not be as good as the default optimization algorithm, even if the converged solution is well.

Observation 2. Due to workload fluctuations, optimization algorithms may not always search for the globally optimal solution when applied in practice. It needs to be combined with other measures to adapt to the instability of the production environment (e.g., multiple sampling confirmations in Bayesian optimization).

Observation 3. For unknown workloads where the mapping of system profiling data to performance vertices is significantly different from historical workloads, our method can not effectively determine the approximate range of the optimal solution. This affects the optimization efficiency and the average performance of the program during the optimization process. However, in most cases the globally (or nearly) optimal solution can be obtained eventually.

The most significant limitations of this method are introducing additional model training overhead. However, this limitation can be resolved with appropriate modifications. If explicit model training is unacceptable, our method could be modified to a form similar to Bayesian agent model building, where the internal model is continuously updated during the optimization process. As stated in Section 4.3, we include fault tolerance mechanisms for inaccurate predictions, which have little effect on the optimal solution, except for the additional sampling overhead. The advantages of our method can be exploited once the model has matured. Another possible solution is to increase the range of the initial search interval in Algorithm 2 when the model is immature, thus mitigating the additional search overhead associated with inaccurate predictions. The above modifications do not affect the optimization principle of this algorithm. In addition, since the resource allocation for jobs occurs in the scheduling phase, the optimization algorithms are executed before the job is run, so the additional overhead incurred does not affect the job performance. Modern monitoring systems have minimized their operating overheads in order to support 24/7 data collection. Many studies have shown that current monitoring system overheads are small enough to have no significant performance impact on programs [12, 28].

6 Conclusion

Distinguishing from existing studies, we consider not only the performance of the optimal solution but also the efficiency of searching the unknown workload, which is a common problem in practice. To realize a extensible thread throttling technique, we innovatively propose the OpenMP performance vertex prediction model to determine the approximate range of optimal solutions for unknown workloads using historical search results. In the small search space, it uses a neighborhood-sampling-based bidirectional hill-climbing search algorithm, which eventually converges to the globally optimal solution. This article discusses the theoretical feasibility as well as the challenges of the above optimization method. Several designs, such as feature selection, kernel space transformation, and thread scalability evaluation, are used to improve thread throttling. The necessity of algorithm designs for improving optimization efficiency for unknown workloads is finally demonstrated through experiments.

Considering the growing importance of green computing and heterogeneous computing, in the future we will continue to improve this method and extend the applicable scenarios. This includes using hardware-level evaluations to determine the energy efficiency characteristics of servers and improve EDP optimization effectiveness, as well as designing multi-server scalable optimization techniques in conjunction with differentiated analysis of different hardware variations.

References

[1] Ilya Afanasyev and Dmitry Lichmanov. 2021. Evaluating the performance of Kunpeng 920 processors on modern HPC applications. In *Proceedings of the International Conference on Parallel Computing Technologies*. Springer, 301–321.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2623–2631.

[3] Mahdi A Almahdawi and Omar De La C Cabrera. 2024. Learning with multiple kernels. *IEEE Access* 12 (2024), 56973–56980.

[4] Xiuxiu Bai, Endong Wang, Xiaoshe Dong, and Xingjun Zhang. 2015. A scalability prediction approach for multi-threaded applications on manycore processors. *The Journal of Supercomputing* 71, 11 (2015), 4072–4094.

[5] Xiaolin Chang, Ruofan Xia, Jogesh K Muppala, Kishor S Trivedi, and Jiqiang Liu. 2016. Effective modeling approach for IaaS data center performance analysis under heterogeneous workload. *IEEE Transactions on Cloud Computing* 6, 4 (2016), 991–1003.

[6] Florina M Ciorba, Christian Iwainsky, and Patrick Buder. 2018. OpenMP loop scheduling revisited: Making a case for more schedules. In *Proceedings of the Evolving OpenMP for Evolving Architectures: 14th International Workshop on OpenMP, IWOMP 2018, Barcelona, Spain, September 26–28, 2018, Proceedings 14*. Springer, 21–36.

[7] AM Coutinho Demetrios, Daniele De Sensi, Arthur Francisco Lorenzon, Kyriakos Georgiou, Jose Nunez-Yanez, Kerstin Eder, and Samuel Xavier-de Souza. 2020. Performance and energy trade-offs for parallel applications on heterogeneous multi-processing systems. *Energies* 13, 9 (2020), 2409.

[8] Tamara Dancheva, Unai Alonso, and Michael Barton. 2024. Cloud benchmarking and performance analysis of an HPC application in Amazon EC2. *Cluster Computing* 27, 2 (2024), 2273–2290.

[9] Jered Dominguez-Trujillo, Keira Haskins, Soheila Jafari Khouzani, Christopher Leap, Sahba Tashakkori, Quincy Wofford, Trilce Estrada, Patrick G Bridges, and Patrick M Widener. 2020. Lightweight measurement and analysis of HPC performance variability. In *Proceedings of the 2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 50–60.

[10] Jill Dunbar. 2024. NAS Parallel Benchmarks. Retrieved August 16, 2024 from https://www.nas.nasa.gov/software/npb.html.

[11] Charles R. Ferenbaugh. 2016. The PENNANT Mini-App. Retrieved August 16, 2024 from https://github.com/lanl/PENNANT.

[12] Sascha Hunold, Jordy I Ajanohoun, Ioannis Vardas, and Jesper Larsson Träff. 2022. An overhead analysis of mpi profiling and tracing tools. In *Proceedings of the 2nd Workshop on Performance EngineeRing, Modelling, Analysis, and VisualizatiOn Strategy*. 5–13.

[13] Matthew D Jones, Joseph P White, Martins Innus, Robert L DeLeon, Nikolay Simakov, Jeffrey T Palmer, Steven M Gallo, Thomas R Furlani, Michael Showerman, Robert Brunner, Andry Kot, Gregory Bauer, Brett Bode, Jeremy Enos, and William Kramer. 2017. Workload analysis of blue waters. arXiv:1703.00924. Retrieved from https://arxiv.org/abs/1703.00924 (2017).

[14] Rajat Kumar, Amit Mankodi, Amit Bhatt, Bhaskar Chaudhury, and Aditya Amrutiya. 2020. Cross-platform performance prediction with transfer learning using machine learning. In *Proceedings of the 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 1–7.

[15] Sandia National Laboratories. 2024. LAMMPS Molecular Dynamics Simulator. Retrieved August 16, 2024 from https://www.lammps.org/

[16] Min Li, Huan Ma, Siyu Lv, Lei Wang, and Shaobo Deng. 2024. Enhanced NSGA-II-based feature selection method for high-dimensional classification. *Information Sciences* 663 (2024), 120269.

[17] Davide Li Calsi and Vittorio Zaccaria. 2023. Interruptible remote attestation of low-end IoT microcontrollers via performance counters. *ACM Transactions on Embedded Computing Systems* 22, 5 (2023), 1–19.

[18] Shulei Liu, Handing Wang, Wei Peng, and Wen Yao. 2022. A surrogate-assisted evolutionary feature selection algorithm with parallel random grouping for high-dimensional classification. *IEEE Transactions on Evolutionary Computation* 26, 5 (2022), 1087–1101.

[19] Arthur Francisco Lorenzon, Charles Cardoso De Oliveira, Jeckson Dellagostin Souza, and Antonio Carlos Schneider Beck. 2018. Aurora: Seamless optimization of openmp applications. *IEEE Transactions on Parallel and Distributed Systems* 30, 5 (2018), 1007–1021.

[20] Guangqiang Luan, Pu Pang, Quan Chen, Shuai Xue, Zhuo Song, and Minyi Guo. 2022. Online thread auto-tuning for performance improvement and resource saving. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 3746–3759.

[21] Sandro Matheus VN Marques, Matheus S Serpa, Antoni N Muñoz, Fábio D Rossi, Marcelo C Luizelli, Philippe OA Navaux, Antonio Carlos S Beck, and Arthur F Lorenzon. 2022. Optimizing the EDP of OpenMP applications via concurrency throttling and frequency boosting. *Journal of Systems Architecture* 123 (2022), 102379.

[22] Sandro Matheus Vila Nova Marques, Fábio Diniz Rossi, Marcelo Caggiani Luizelli, Antonio Carlos Schneider Beck, and Arthur Francisco Lorenzon. 2023. Seamless thermal optimization of parallel workloads. *IEEE Design & Test* 40, 5 (2023), 34–41.

[23] Thiarles S Medeiros, Luan Pereira, Fábio D Rossi, Marcelo C Luizelli, Antonio Carlos S Beck, and Arthur F Lorenzon. 2021. Mitigating the processor aging through dynamic concurrency throttling. *Journal of Parallel and Distributed Computing* 156 (2021), 86–100.

[24] Martin Molan, Andrea Borghesi, Daniele Cesarini, Luca Benini, and Andrea Bartolini. 2023. RUAD: Unsupervised anomaly detection in HPC systems. *Future Generation Computer Systems* 141 (2023), 542–554.

[25] Ananya Muddukrishna, Peter A Jonsson, Artur Podobas, and Mats Brorsson. 2016. Grain graphs: OpenMP performance analysis made easy. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 1–13.

[26] DA Nikitenko, PA Shvets, and VV Voevodin. 2020. Why do users need to take care of their HPC applications efficiency? *Lobachevskii Journal of Mathematics* 41, 8 (2020), 1521–1532.

[27] Litao Qu, Weibin He, Jianfei Li, Hua Zhang, Cheng Yang, and Bo Xie. 2023. Explicit and size-adaptive PSO-based feature selection for classification. *Swarm and Evolutionary Computation* 77 (2023), 101249.

[28] Thomas Roehl, Jan Treibig, Georg Hager, and Gerhard Wellein. 2014. Overhead analysis of performance counter measurements. In *Proceedings of the 2014 43rd International Conference on Parallel Processing Workshops*. IEEE, 176–185.

[29] Janaina Schwarzrock, Charles C de Oliveira, Marcus Ritt, Arthur F Lorenzon, and Antonio Carlos S Beck. 2020. A runtime and non-intrusive approach to optimize edp by tuning threads and cpu frequency for openmp applications. *IEEE Transactions on Parallel and Distributed Systems* 32, 7 (2020), 1713–1724.

[30] Janaina Schwarzrock, Michael Guilherme Jordan, Guilherme Korol, Charles C de Oliveira, Arthur F Lorenzon, and Antonio Carlos S Beck. 2019. On the influence of data migration in dynamic thread management of parallel applications. In *Proceedings of the 2019 IX Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE, 1–8.

[31] Janaina Schwarzrock, Michael Guilherme Jordan, Guilherme Korol, Charles C de Oliveira, Arthur F Lorenzon, Mateus Beck Rutzig, and Antonio Carlos S. Beck. 2021. Dynamic concurrency throttling on numa systems and data migration impacts. *Design Automation for Embedded Systems* 25, 2 (2021), 135–160.

[32] Janaina Schwarzrock, Hiago Mayk G De A Rocha, Arthur F Lorenzon, and Antonio Carlos S Beck. 2022. Smoothing on dynamic concurrency throttling. In *Proceedings of the 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 962–971.

[33] Pavel Shvets, Vadim Voevodin, and Dmitry Nikitenko. 2020. Approach to workload analysis of large HPC centers. In *Proceedings of the International Conference on Parallel Computational Technologies*. Springer, 16–30.

[34] Fadi N Sibai. 2014. Performance modeling and analysis of parallel gaussian elimination on multi-core computers. *Journal of King Saud University-Computer and Information Sciences* 26, 1 (2014), 41–54.

[35] Melissa C Smith and Gregory D Peterson. 2012. Optimization of shared high-performance reconfigurable computing resources. *ACM Transactions on Embedded Computing Systems (TECS)* 11, 2 (2012), 1–22.

[36] Shaojie Tan, Qingcai Jiang, Zhenwei Cao, Xiaoyu Hao, Junshi Chen, and Hong An. 2024. Uncovering the performance bottleneck of modern HPC processor with static code analyzer: A case study on kunpeng 920. *CCF Transactions on High Performance Computing* 6, 3 (2024), 343–364.

[37] Laurens Versluis, Mehmet Cetin, Caspar Greeven, Kristian Laursen, Damian Podareanu, Valeriu Codreanu, Alexandru Uta, and Alexandru Iosup. 2021. A holistic analysis of datacenter operations: Resource usage, energy, and workload characterization–extended technical report. arXiv:2107.11832. Retrieved from https://arxiv.org/abs/2107.11832 (2021).

[38] Guo Feng Anders Yeo, Irene Hudson, David Akman, and Jeffrey Chan. 2023. Sequential feature selection and instance selection using SpFSR and SpFixedIS. In *Proceedings of the 2023 6th International Conference on Artificial Intelligence and Big Data (ICAIBD)*. 843–848.

Received 13 December 2024; revised 2 July 2025; accepted 21 September 2025