Heliyon 9 (2023) e16777

Contents lists available at ScienceDirect

Heliyon

journal homepage: www.cell.com/heliyon

Adaptive variable sampling model for performance analysis in high cache-performance computing environments

Mincheol Shin [a], Mucheol Kim [a,*], Geunchul Park [b], Ajith Abraham [c]

[a] Department of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea
[b] Division of Supercomputing, Korea Institute of Science and Technology Information, Daejeon, Republic of Korea
[c] Faculty of Computing and Data Science, FLAME University, Lavale, Pune, Maharashtra, India

ARTICLE INFO

Keywords:
High performance computing
Data science
Decision support
Performance prediction
Machine learning

ABSTRACT

High-performance computing provides computing power for a variety of scientific disciplines, supporting advancements by offering insights beyond metacognition. Maximizing computing performance without wasting resources is a major research issue. Predicting the performance of a computer's next state is effective for scheduling. However, hardware performance monitors representing the computer's state require high expert knowledge, and there is no standardized model. In this paper, we propose an adaptive variable sampling model for performance analysis in high-performance computing environments. Our method automatically classifies the optimal variables from numerous variables related to performance prediction and predicts performance using the sampled variables. The optimal variables for performance analysis do not require expert knowledge during the sampling process. We conducted experiments in various architectures and applications to validate this method. This model performed at least 24.25% and up to 58.75% faster without any loss in accuracy.

1. Introduction

Due to population growth and infrastructure development, the requirements and complexity of smart cities are increasing, requiring the ability to solve various city problems (population, economy, health, energy, climate, etc.) [1–3]. High-performance computing (HPC) is an essential technology in solving not only real-world problems such as logistics and energy, but also science and engineering problems of hyper-cognition in the realization of smart cities. With the increase in the amount of data produced in modern cities, applications in various areas such as population, environment, and climate that are difficult to derive with existing computing capabilities are being improved and expanded [4,5].

Meanwhile, to realize the exascale Computing era, the development of energy efficiency as well as the performance improvement of sustainable HPC must be prioritized. Energy consumed by HPC systems accounted for 1.5% of total electrical energy usage in 2010, and since then, it has continued to increase as the demand for computing power grows [6]. In addition, HPC energy efficiency improvement (i.e. energy saving policy, scheduling) is required to achieve the 20 MW target, which is an exascale performance condition, but it is not consistent with the rate of increase in demand for large scale computing [7].

In order to effectively use computer resources, it is important to predict their state. In the absence of a standard model for hardware performance monitors (HPM) in the analysis of HPC systems, identifying and selecting the optimal variables for analysis proves to be a

* Corresponding author.
E-mail address: mucheol.kim@gmail.com (M. Kim).

https://doi.org/10.1016/j.heliyon.2023.e16777
Received 14 April 2023; Received in revised form 24 May 2023; Accepted 26 May 2023
Available online 5 June 2023

considerable challenge [41–44]. The study by Roundtree et al. [8] emphasizes the importance of CPU frequency and memory access for performance prediction in HPC environments. Even though performance can be improved through various indicators (CPU load, memory usage, I/O activity, network bandwidth) [46], an approach that considers HPM at the operating system (OS) level is not likely to reflect the characteristics of the architecture (Zhang et al. [9]). To configure HPM in a highly interactive and complex HPC environment, it is necessary to use hardware events that require a high level of computer science knowledge; correlation analysis between performance indicators through data mining is also required. In spite of the difficulty of analyzing many variables, the improvement in performance and prediction from using various HPM will provide new insights [45].

In this paper, we propose an adaptive variable sampling model for performance analysis in high-performance computing environments. Our method automatically classifies numerous variables involved in HPC performance prediction and effectively predicts performance using these variables. The optimal variables for performance analysis are determined without requiring expert knowledge during the sampling process. The predictive model is trained using an efficient support vector machine (SVM) for high-dimensional data. As minimizing the overhead is crucial when making decisions in HPC based on prediction results, we have chosen SVM as our prediction model. Experiments are conducted using eight benchmarks specified from NAS Parallel Benchmarks (NPB) to validate our approach. To evaluate the proposed model, we predict the behavior of untrained applications and investigate whether our approach can effectively predict performance for new tasks [10]. The contributions of the proposed approach can be summarized as follows.

- Our sampling strategy identifies the optimal variables required for performance prediction without the need for HPC domain knowledge, depending on the set parameter values.
- The proposed model enables efficient performance prediction in various application and architecture environments with optimal variables for performance prediction.

The paper is organized as follows. Section 2 describes related work. In Section 3, we propose an adaptive performance prediction model for effective resource management. Section 4 presents an evaluation of our proposed method using several applications and benchmarks. Section 5 summarizes the contributions.

2. Related work

In HPC, system profiling data provides insights into performing in a variety of task environments, such as computing optimization, performance prediction, computing design, and failure prediction. Furthermore, there are diverse monitoring and performance measurement tools available for aggregating system profiling data [11–13].

Many researchers have suggested system performance profiling-based approaches for system optimization [14–19,41–43]. Due to the development of the architecture and the complexity of the composition, as shown in (Fig. 1), the variable representing the state of the resource in the HPC environment has been increasing continuously. However, Modern processors typically provide 4–8 hardware counters to measure hundreds of events (i.e. cache and translation lookaside buffer misses) [20]. Existing techniques for measuring events are inadequate for data profiling due to the small number of counters. To improve accuracy and to solve omission problems that occur in the multiplexer [20,21], In Ref. [20], the authors have presented a method that replaces outliers after multiplexer sampling, fills in the missing values, improves the quality of data, and designates the interaction and ranking of events. In Ref. [21], the authors have presented NeoMPX for Performance Application Programming Interface using a divided curved-area method and curved-area

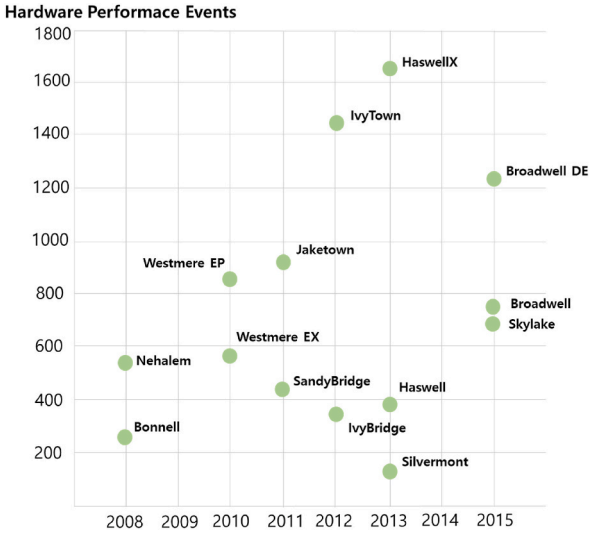

Fig. 1. Number of events according to architecture development.

method to improve the accuracy of the multiplexer. In Ref. [22], the authors have presented a set of profiling tools designed to integrate with a field programmable gate array accelerated simulation platform. However, it was too slow to observe end-to-end system-level level operation in a multi-cluster environment and is at an inadequate level of abstraction to diagnose performance.

In addition, there have been many studies on the characterization of applications in HPC. Many researchers have conducted work to improve job scheduling, resource management, and performance optimization based on application program characterization [23,24]. In Ref. [25], the authors have presented a method of classifying the characteristics of application programs considering EM(expectation maximization) clustering-based resource interference. In Ref. [24], the authors have presented application characterization through micro-architecture metrics and OS-level metrics. They have done a top-down analysis method to classify bottlenecks through a drill-down structured in a hierarchical manner. In Ref. [25], the authors have presented an ASTPI(Average Stall Time Per Instruction) based workload classifier to distinguish between CPU-intensive and memory-intensive workloads.

[41] proposed an analysis of machine learning (ML) algorithms to collect knowledge about the performance of these applications through hardware events and derived performance metrics [42]. raised the issue of it usually taking a long time to profile and train the model, especially for the cost-expensive applications. Consequently, they proposed APMT, an offline performance modeling tool with hardware counter-supported profiling, to overcome the drawbacks of manual work (analytical model) and unguaranteed model accuracy (empirical model) [43]. presented significant performance overhead or performance deviation problems that occur during the collection process using multiplexing methods and proposed a systematic performance diagnosis method focused on building accurate and interpretable performance models using performance counters.

3. An adaptive variable sampling model for performance analysis in high-performance computing environments

In this paper, we propose an adaptive variable sampling model for performance analysis in high-performance computing environments (Fig. 2). The proposed model proceeds in two stages. First, our sampling strategy can solve the multiplexing-based collection problem by using fewer variables without domain knowledge. Second, the training set generated using the profiling data predicts the future state of the computer using an SVM-based performance prediction model.

3.1. Data collection and data-driven feature sampling strategy

In previous performance prediction approaches, variable selections were performed through domain knowledge. Domain-based prediction research has difficulty reflecting the HPC environment, GPUs, computational accelerators, and so on. There are three difficulties in predicting performance through performance monitoring events (PMU). First, the Perf tool provides little guidance on how to interpret the results [27]. Because modern processors generate hundreds to thousands of events representing the current state of a computer [20,27], it is difficult to select the type and amount of data necessary to grasp the characteristics of applications [28]. Therefore, selecting variables and interpreting results require considerable domain knowledge and an understanding of the architecture. Second, it is known that collecting a large number of variables using multiplexers results in the loss of collected variables [20, 29,30]. Using the limited number of built-in counters in computers to gather multiple variables increases the uncertainty of the collected variables. Moreover, in the prediction model, there are complexity issues [31] and the curse of dimensionality problems [32–34] during the computational process. Third, collecting a lot of events comes with energy consumption. The change in power consumption as the number of instructions increases shows that event collection is influencing energy efficiency (Table 1). Therefore, extraction of effective essential performance indicators can reduce HPC power consumption.

The proposed method collects the state information of various HPC applications through NPB. We select NPB benchmark classes as D class for performance analysis [10]. Because the proposed model collects data every second, it cannot collect enough data for a problem size smaller than class D. Data collected using Perf [13] is expressed as a time series (Equation (1)).

$$Data = \begin{Bmatrix} T_1\{E_1, E_2, E_3, \cdots E_n\} \\ T_2\{E_1, E_2, E_3, \cdots E_n\} \\ \vdots \\ T_n\{E_1, E_2, E_3, \cdots E_n\} \end{Bmatrix} \tag{1}$$

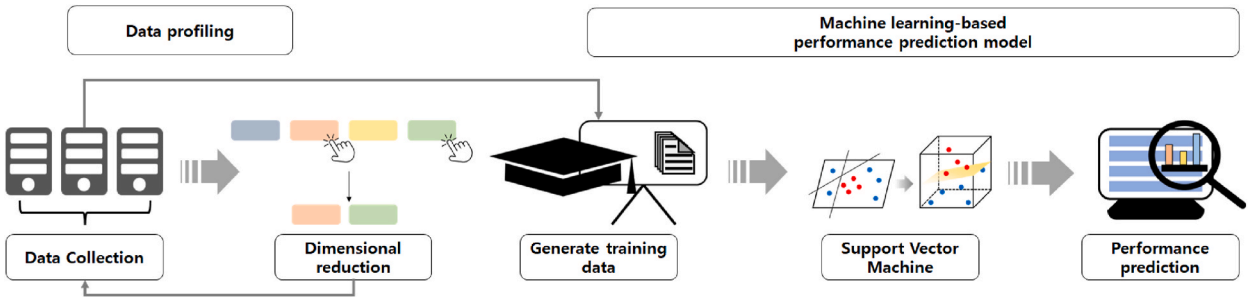

Fig. 2. Adaptive variable sampling model for performance analysis in high-performance computing environments process.

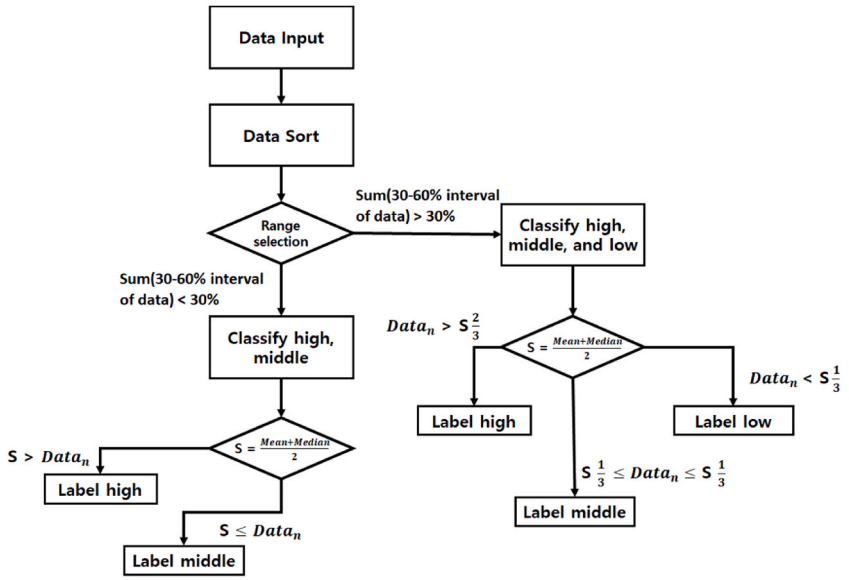

Fig. 3. Labeling flow chart.

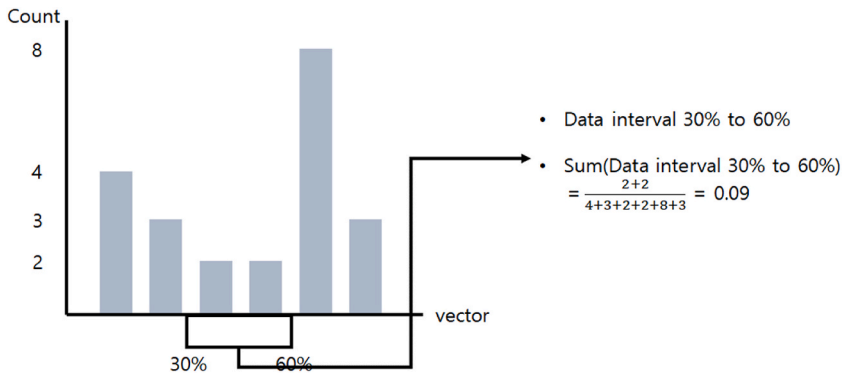

Fig. 4. Data interval example.

Table 1
Power value according to the number of instructions.

Command	Command count	Power(W)	Command	Command count	Power(W)
NULL	0	33.2	skb:*	3	34.27
H/w + S/w + PMU event	40	116.42	workqueue:*	4	34.22
block:*	19	44.63	task:*	2	32.38
net:*	9	38.4	timer:*	13	39.78
sock:*	2	32.27	syscalls:*	592	422.09
writeback:*	25	49.42	sched:*	23	51.74
kmem:*	12	40.57	signal:*	2	33.8
vmscan:*	15	42.05			

Table 2
Event items collected by Perf.

Event	branch-load-misses, branch-misses, bus-cycles, cache-misses, cache-references, CPU-clock, CPU-cycles, dTLB (data Translation Lookaside Buffer)-load-misses, iTLB (instruction Translation Lookaside Buffer)-loads, L1-icache-load-misses, msr/aperf/,msr/mperf/, page-faultstask-clock

T represents a certain collection interval, and E_n refers to the n^{th} hardware/software event of Perf. Perf collects events consisting of H/W (hard-ware event), S/W (soft-ware event), and Trace Point. Because Perf provides different items for each architecture, the events for each architecture are classified and collected (Table 2).

The proposed feature selection method finds the most characteristic variable in the entire dataset by the principal component (PC) matrix and the cumulative sum of variances.

The variable with the important feature represents the largest data variance among the data (Equation (2)). The elements are defined as follows.

- $Cov(X,Y)$: It represents $Var(X)$, the distribution of data X at $Cov(X,X)$, whereas $Cov(X, Y)$ refers to the relevance between element X and Y.
- X_i, Y_i: ith element of X and Y
- $\overline{X}$, $\overline{Y}$: Means of element X and Y
- n: n raw data

$$Cov(X,Y) = \frac{\sum_{i}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{n-1} \tag{2}$$

The purpose of PCA is to find and project an axis that maximizes the variance in the original data (Table 3). When PCA uses the Lagrange multiplier method to project data X onto an arbitrary axis P, which is a unit vector V, the variance is as follows (Equation (3)).

- C: Eigenvector
- λ: Eigenvalue of C variance when projected by eigenvector

$$L(X_{V\rightarrow}, \lambda) = X_{V\rightarrow}{}^{T} C X_{V\rightarrow} - \lambda\left(X_{V\rightarrow}{}^{T} X_{V\rightarrow} - 1\right) \tag{3}$$

C is calculated through the partial differentiation of $X_{V\rightarrow}$, using the Lagrange function L. (Equation (4)):

$$C = X_{V\rightarrow} \lambda X_{V\rightarrow}{}^{T} \tag{4}$$

The PC, the column vector of the eigenvector, maximizes the variance in the process of projecting the eigenvector. To determine the dimensions, the variance cumulative sum ratio of each PC is calculated (line 3, equation (5)).

- CP_n: Cumulative sum of n-dimensional variance
- $M_{n,i}$: Eigenvector of i^{th} N-dimensional data

Table 3
Dimensionality reduction algorithm.

Algorithm 1. Dimensionality Reduction (D, V, R)

Input: $D \leftarrow$ Collected raw data list
$V \leftarrow$ Cumulative sum of variance
$R \leftarrow$ Ratio of eigenvector selection
Output:$F \leftarrow$ Feature selection list
/*
M : Principal component matrix
M_n: N-dimensional principal component
$M_{(n,i)}$: Eigenvector of ith N-dimensional data
CP_n: Cumulative sum of n-dimensional variance
A : Selected dimension
variables_matrix_list : A list of the variable matrix selected through the PCA
Result_L: key variable list
*/
1. $D \leftarrow Normalization(D)$
2. $(M, CP) \leftarrow PCA(D)$
3. $A \leftarrow CP_n > V$
for n in range(A)
for i in range $(\text{len}(M_n))$
4. $variables\_matrix\_list \leftarrow R * list.Max(|M_n|) < |M_{(n,i)}| < list.Max(|M_n|)$
for n in range (len (*variables_matrix_list*).columns)
For i in range (len (*variables_matrix_list*)
5. if $|Pearson\ Correlation\ Coefficient(L_{(n,1)}, L_{(n,i+1)})| < 0.7$
$Result\_L.append(L_{(i)})$
6. else
pass
7. return *Result_L*

- V: Cumulative sum of variance
- n: N-dimension
- A: Variable saved after deciding the dimension of the PC

$$\sum_{1}^{n} CP_n \begin{cases} A = i \quad if \quad CP_n > V \\ Pass \quad otherwise \end{cases} \tag{5}$$

To extract the representative variables in the PC dimensions (line 4), it is necessary to select the highest eigenvector (Equation (6)) and $M_{(n,i)}$ within the range through repetition up to the selected A-dimensions, save and return it in L (Equation (7), line 5). The elements are defined as follows:

- P: Eigenvector of ith N-dimensional data in the PC matrix M
- R: Ratio of eigenvector selection
- L: A list of the variables selected through the PCA
- *List M*: PC matrix M in which the n-value remains unchanged when the equation is repeated from 1 to i

$$\sum_{1}^{i} List\ M_{(n,i)} \begin{cases} Max = |P| \quad if \quad |P| > Max \\ Pass\ otherwise \end{cases} \tag{6}$$

$$\sum_{1}^{i} List\ M_{(n,i)} \begin{cases} L \leftarrow P \quad if \quad R * Max < M_{(n,i)} < Max \\ Pass\ otherwise \end{cases} \tag{7}$$

The key variable has the greatest influence on each eigenvector (line 4). If there is a correlation between the key variables, the complexity is improved during the performance prediction process, but the accuracy is not greatly affected. Therefore, Pearson's correlation coefficiency [35] is used to remove variables having high correlation. The first value of each L should be included in *Result_L* (line5). To remove the highly correlated variables, we calculate the Pearson correlation coefficient between the key variables and remove those whose value is 0.7 or greater (line 5, 6). Finally, the derived variables of *Result_L* are applied to the model (line 7).

This strategy is a method for sampling columns necessary for prediction, so it does not create new variables compared to feature selection. This allows for analyzing the relationships, associations, and important factors beyond the cognitive scope among the variables during the prediction process.

3.2. Machine learning-based performance prediction model

The variety of applications running on HPC systems is highly diverse. Therefore, to achieve effective scheduling, it is necessary to perform effective predictions even for new forms of data that have not been previously learned. Another challenge can arise when using predictive models that require more resources to achieve high performance predictive accuracy. Therefore, we predict the next state of the computer system using an SVM model that solves the hyperplane problem that maximizes the margin in the classification of performance metrics.

3.2.1. Generate training data based on data profiling

SVM, a supervised learning model, performs training based on the labels in the data. However, separate labeling work is required for the data analysis because attribute information for each application task does not exist. Depending on the HPC environment, the bandwidth and resource values of the idle state and the maximum load state are different. Using the absolute values of resources for labeling is not appropriate since it varies depending on the computer environment (architecture, node connection). We generate data labels for each resource through their usage and calculate high and low usage situations relative to the specific computer environment for labeling (Table 4). shows a method for creating data labels by comparing the overall usage of resources (CPU, Memory).

3.2.2. SVM-based performance prediction model

Kernel SVM makes it possible to classify hyperplanes that cannot be linearly classified by increasing the dimensions of the data. Due to a soft margin from some noise, generalization is guaranteed for both noisy and untrained data. "Kernel trick" during the process of the SVM approach is a mathematical technique that can achieve the same result as adding multiple polynomial features, even without

Table 4
Workload calculation algorithm.

Algorithm 2 Workload calculation algorithm
1. Perform normalization on data D.
2. Create and initialize resource variables for each attribute.
3. Add the normalized value corresponding to the resource (CPU, Memory) of each attribute.
4. End the algorithm and return the variable L_D.

(Fig. 3) shows the labeling flow chart. We sort the profiling data processed in Algorithm 1 and present the data as a frequency distribution. The data interval that defines the range from the minimum to the maximum determine the classification model in the histogram data (Fig. 4).

calculating the actual value of the feature function that maps the data to a high-dimensional space. We define the hyperplane in equation (8).

$$f(x) = \Phi(x)^T w + b \tag{8}$$

$\Phi(x)$ is a basis function that maps the p-dimensional x data to a higher m-dimension. The optimal hyperplane is calculated using the Wolfe duality theorem (Equation (9)) [36].

$$Max\ L_D = \sum_{i=1}^{N} a_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j y_i y_j \Phi(x_i)' \Phi(x_j)$$

$$subject\ to \sum_{i=1}^{N} a_i y_i = 0 \tag{9}$$

$$0 \le a \le C, i = 1 \cdots N$$

The inner product of the quadratic polynomial x from Equation (9) can be expressed as follows (Equations 10 and 11).

$$\Phi(x) = \Phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1 x_2 \\ x_2^2 \end{pmatrix} \tag{10}$$

$$\Phi(a)^T \Phi(b) = \begin{pmatrix} a_1^2 \\ \sqrt{2}a_1 a_2 \\ a_2^2 \end{pmatrix}^T \bullet \begin{pmatrix} b_1^2 \\ \sqrt{2}b_1 b_2 \\ b_2^2 \end{pmatrix} = a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2 = (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \bullet \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 \tag{11}$$

The two-dimensional inner product of a and b is calculated in Equation (12).

$$a^T \bullet b = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \bullet \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 \tag{12}$$

We can confirm that the inner product calculation using the kernel trick and the general inner product calculation value are the same (Equations 11 and 12). Well-known cases that satisfy the kernel trick are the Gaussian Radial Basis Function (RBF) (Equation (13)), r-order polynomial kernel (Equation (14)), and sigmoid kernel (Equation (15)) [37–40]. We have shown the best performance using the RBF kernel.

$$K(a,b) = exp\left(-\gamma ||a-b||^2\right) \tag{13}$$

$$K(a,b) = \left(a^T b + 1\right)^r \tag{14}$$

Table 5
Specifications for each node.

Node		Information	
Master Node		OS	CentOS Linux release June 7, 1810 (Core)
		CPU	Intel® Xeon® CPU E5-2620 0 @ 2.00 GHz
		Core	24 total, 6 cores for each CPU, 4 CPUs
		Memory	31 GiB
Slave Node	KNL [a]Node	OS	CentOS Linux release June 7, 1810 (Core)
		CPU	Intel® Xeon Phi™ CPU 7290 @ 1.50 GHz
		Core	288 total, 72 cores for each CPU, 4 CPUs
		Memory	188 GiB
	SLX[b] Node	OS	CentOS Linux release March 7, 1611 (Core)
		CPU	Intel® Xeon® Gold 6152 CPU @ 2.10 GHz
		Core	88 total, 22 cores for each CPU, 4 CPUs
		Memory	188 GiB
	EPYC[c] Node	OS	CentOS Linux release August 7, 2003 (Core)
		CPU	AMD EPYC 7451 24-Core Processor
		Core	Total 96, 24 Cores for each CPU, 4 CPUs
		Memory	251 GiB

[a] Knights Landing.
[b] Skylake
[c] AMD EPYC.

$$K(a,b) = tanh\left(ka^T b - \delta\right)^r \tag{15}$$

4. Experimental environment and results

Our paper evaluates the accuracy of prediction by performing the proposed approach on various HPC architectures and benchmark applications.

4.1. Experimental setup

We construct the experimental environment with consisting of one master node and two slave nodes for each architecture. The specifications for each node and the software version used in the OS environment for each model implementation are as (Table 5) and (Table 6). Cross-validation is a method that divides the data into equally K fold data and the model is trained using a differently partitioned datasets (*E(i)*) in each task (Fig. 5) [37]. To evaluate the accuracy of the proposed approach, we divide the training and test data for eight NPB applications (BT, CG, EP, FT, IS, MG, LU, SP) (See in Table 7).

4.2. Experimental results

The data profiling process serves two purposes. The first objective is to avoid the accuracy and resource efficiency issues caused by multiplexers. Second, we find the optimal variable for prediction in the variable list without a high-level understanding of HPC.

To test whether the data profiling process is effective in predicting the next task state, we predicted the accuracy by comparing the original data with 30, 50, 70, 80, where the ratio of eigenvector selection is sharply distinct (Figs. 6 and 7). We conduct the experiment except for the case where the ratio of eigenvector selection value is 90, because data reduction hardly occurred. The experimental results display improved accuracy or similar results in most conditions. We carry out the event list for our experiments to intel architecture. Therefore, our experimental data learned with Intel-based architecture shows relatively low accuracy in AMD's EPYC architecture because performance variables between architectures are different.

Different architectures have different performance variables. We carry out the event list for our experiments to intel architecture which most of our experimental environments are based on. Therefore, experimental data trained with Intel-based architectures show relatively low accuracy on AMD's EPYC architecture.

Experimental results show a valid range of accuracy even when making predictions with data reduced by 15%–70%. It provides insight that a lot of data causes a dimensional curse, or that there are key variables in predicting the next state. In addition, it shows that the proposed method effectively predicts the resource utilization status only with profiling data composed of a small amount of performance indicators.

Our proposed method reduces the execution time by 24–58% (See Tables 8–11). This demonstrates that our model is a more efficient method than using all variables. The fact that it performs quickly without sacrificing accuracy suggests that it has little impact on the operating state of the HPC and presents potential for practical application. It shows that our model can make predictions faster and more accurately than using all the data. In most cases, it was confirmed that the ratio of eigenvector selection was performed faster when the selection was small, and it can be seen that the performance prediction time decreases linearly according to the number of variables.

Our experiments perform multiple applications on different architectures to test whether our model exhibits generalizable performance. We handle ratio of eigenvector selection parameters to generate dimension-reduced data to predict performance, without having to understand each hardware event. The experimental results show a 24–59% improvement in execution speed compared without loss of accuracy. Our approach represents consistent accuracy regardless of the various applications (eight NPB states), architectures (KNL, SLX, EPYC), and node environments (single, multi). Therefore, when applied to the scheduling model, there is little loss of accuracy, but it shows the possibility of significant time-saving effects.

5. Conclusions

Due to the outstanding performance of HPC, the demand to solve various real-world problems such as logistics and energy continues to increase. Analyzing the state of a computer is critical to implementing an effective HPC operating method, however it is a difficult problem because it requires very high domain knowledge. In this paper, we proposed an adaptive variable sampling model for performance analysis in high-performance computing environments. The proposed model can obtain key variables required for performance analysis through sampling strategies without domain knowledge such as frequency control, power capping, perf. In addition, it is unnecessary to continuously collect all variables in the process of exploring variables used for performance prediction.

We predict performance on minimal resources using a kernel svm model. By utilizing the proposed model to perform performance predictions, we achieved performance predictions at least 24.25% and up to 58.75% faster than methods using all data, without any loss in performance. This result demonstrates that our method appropriately samples the necessary variables for performance prediction and reduces the model's complexity. Experimental results show that the proposed model generates performance predictions about 24%–58% faster without performance loss compared to the case of using all performance indicators. It results demonstrates that our model not only effectively samples the variables, but also is a lightweight performance prediction model.

In a future work, we should expand the scope of variable exploration in the performance prediction process and provide practical

Table 6
Software version.

Software		Version
Perf	KNL	Perf version 3.10.0–1062.18.1.el7.x86_64.debug
	SLX	Perf version 3.10.0–1062.12.1.el7.x86_64.debug
	EPYC	Perf version 3.10.0–1127.8.2.el7.x86_64.debug

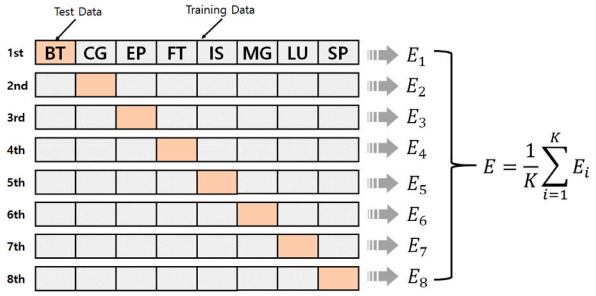

Fig. 5. Model evaluation method (cross validation).

Table 7
Benchmark specifications[a] [10].

Kernels	IS	Integer sort, random memory access
	EP	Embarrassingly parallel
	CG	Conjugate gradient, irregular memory access and communication
	MG	Multi-grid on a sequence of meshes, long- and short-distance communication, memory intensive
	FT	Discrete 3D fast Fourier transform, all-to-all communication
Pseudo Applications	BT	Block tri-diagonal solver
	SP	Scalar penta-diagonal solver
	LU	Lower-upper Gauss-Seidel solver

[a] https://www.nas.nasa.gov/software/npb.html.

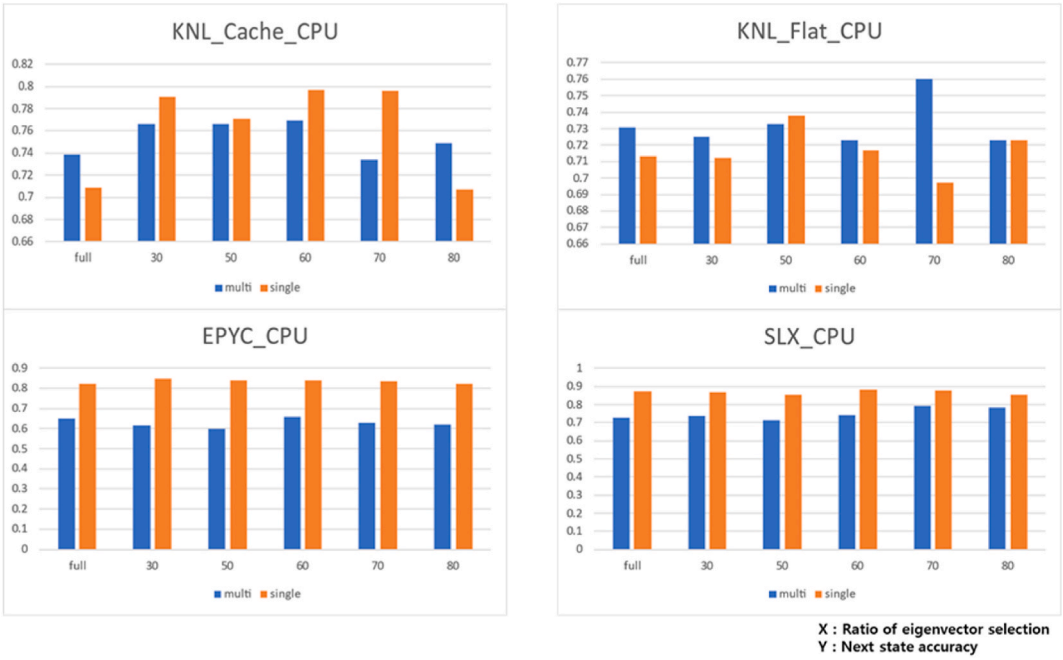

Fig. 6. Next state accuracy prediction (CPU).

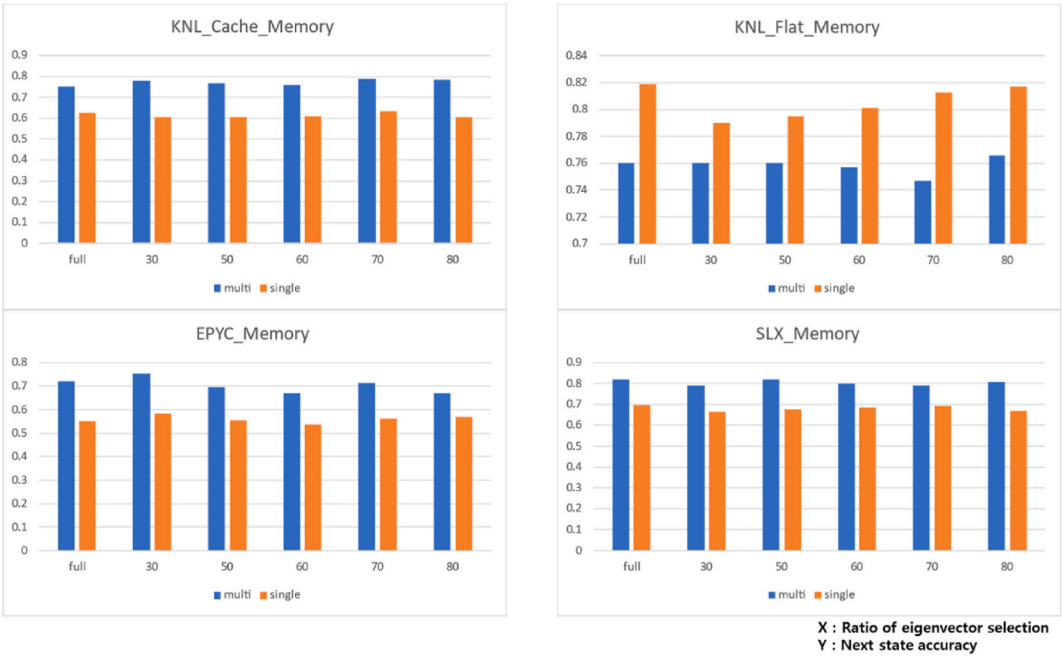

Fig. 7. Next state accuracy prediction (Memory).

Table 8
Ratio of reduction in execution time in KNL_Cache.

Architecture		KNL_Cache				
Ratio of eigenvector selection		30	50	60	70	80
CPU	Multi	36%	33%	35%	29%	24%
	Single	45%	40%	38%	32%	23%
Memory	Multi	44%	37%	42%	33%	26%
	single	37%	39%	31%	31%	24%
Average		40.50%	37.25%	36.50%	31.25%	24.25%

Table 9
Ratio of reduction in execution time in KNL_Flat.

Architecture		KNL_Flat				
Ratio of eigenvector selection		30	50	60	70	80
CPU	Multi	41%	37%	35%	37%	26%
	Single	31%	42%	32%	40%	21%
Memory	Multi	44%	38%	37%	34%	31%
	Single	40%	41%	34%	34%	29%
Average		39.00%	39.50%	34.50%	36.25%	26.75%

Table 10
Ratio of reduction in execution time in EPYC.

Architecture		EPYC				
Ratio of eigenvector selection		30	50	60	70	80
CPU	Multi	39%	33%	34%	34%	24%
	Single	93%	92%	33%	31%	26%
Memory	Multi	37%	36%	32%	29%	26%
	Single	65%	74%	32%	28%	24%
Average		58.50%	58.75%	32.75%	30.50%	25.00%

Table 11
Ratio of reduction in execution time in SLX.

Architecture		SLX				
Ratio of eigenvector selection		30	50	60	70	80
CPU	Multi	40%	34%	33%	33%	26%
	Single	40%	40%	33%	28%	24%
Memory	Multi	42%	36%	37%	31%	24%
	Single	37%	40%	34%	32%	23%
Average		39.75%	37.50%	34.25%	31.00%	24.25%

insights in combination with HPC power consumption.

Declaration of competing interest

The authors declare that they have no financial or personal interests that could potentially influence or bias the research reported in this paper.

Acknowledgments

This research was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1A2C109550811) and part by Korea Institute for Advancement of Technology (KIAT) grant funded by the Korea Government (MOTIE) (P0012724, The Competency Development Program for Industry Specialist).

References

[1] S.E. Bibri, A foundational framework for smart sustainable city development: theoretical, disciplinary, and discursive dimensions and their synergies, Sustain. Cities Soc. 38 (2018) 758–794, https://doi.org/10.1016/j.scs.2017.12.032.
[2] P. Duan, M. Askari, K. Hemat, Z.M. Ali, Optimal operation and simultaneous analysis of the electric transport systems and distributed energy resources in the smart city, Sustain. Cities Soc. 75 (2021), https://doi.org/10.1016/j.scs.2021.103306.
[3] J. Colding, M. Colding, S. Barthel, The smart city model: a new panacea for urban sustainability or unmanageable complexity? Environ. Plan. B Urban Anal. City Sci. 47 (1) (2020) 179–187, https://doi.org/10.1177/2399808318763164.
[4] C. Silvano, G. Agosta, A. Bartolini, A.R. Beccari, L. Benini, J. Bispo, R. Cmar, J.M.P. Cardoso, C. Cavazzoni, J. Martinovič, G. Palermo, M. Palkovič, P. Pinto, E. Rohou, N. Sanna, K. Slaninová, Autotuning and Adaptivity Approach for Energy Efficient Exascale HPC Systems: the ANTAREX Approach. 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016, 708–713.
[5] M.M. Rathore, A. Paul, W.H. Hong, H.C. Seo, I. Awan, S. Saeed, Exploiting IoT and big data analytics: defining Smart Digital City using real-time urban data, Sustain. Cities Soc. 40 (2018) 600–610, https://doi.org/10.1016/j.scs.2017.12.022.
[6] P. Arabas, E. Niewiadomska-Szynkiewicz, Energy-efficient workload allocation in distributed HPC system, in: International Conference on High Performance Computing & Simulation (HPCS), 2019, pp. 747–753, https://doi.org/10.1109/HPCS48598.2019.9188240.
[7] C. Silvano, G. Agosta, A. Bartolini, A.R. Beccari, L. Benini, J. Bispo, R. Cmar, J.M.P. Cardoso, C. Cavazzoni, J. Martinovič, G. Palermo, M. Palkovič, P. Pinto, E. Rohou, N. Sanna, K. Slaninová, Autotuning and Adaptivity Approach for Energy Efficient Exascale HPC Systems: the ANTAREX Approach. 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016, 708–713.
[8] B. Rountree, D.K. Lowenthal, M. Schulz, B.R. de Supinski, Practical performance prediction under dynamic Voltage frequency scaling, in: International Green Computing Conference and Workshops, IGCC 2011, 2011, https://doi.org/10.1109/IGCC.2011.6008553.
[9] J. Zhang, R.J. Figueiredo, Application classification through monitoring and learning of resource consumption patterns, in: 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, IEEE Computer Society, 2006, https://doi.org/10.1109/IPDPS.2006.1639378.
[10] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, M. Yarrow, The NAS Parallel Benchmarks 2.0, 1995.
[11] S. Mark, Collectl, http://collectl.sourceforge.net/. Accessed on 2018-10-31.
[12] D. Wieers, Dstat, http://dag.wiee.rs/home-made/dstat/dstat.1.html. Accessed on 2009-11-25.
[13] Multiple Authors, perf: Linux profiling with performance counters. https://perf.wiki.kernel.org/index.php/Main_Page. Accessed on 2018-05-18.
[14] S. Wang, B. Luo, W. Shi, D. Tiwari, Application configuration selection for energy-efficient execution on multicore systems, J. Parallel Distr. Comput. 87 (2016) 43–54, https://doi.org/10.1016/j.jpdc.2015.09.003.
[15] A. Shahid, M. Fahad, R.R. Manumachu, A. Lastovetsky, Improving the accuracy of energy predictive models for multicore CPUs by combining utilization and performance events model variables, J. Parallel Distr. Comput. 151 (2021) 38–51, https://doi.org/10.1016/j.jpdc.2021.01.007.
[16] F. Liang, C. Feng, X. Lu, Z. Xu, Performance characterization of hadoop and data MPI based on amdahl's second law, in: Proceedings - 9th IEEE International Conference on Networking, Architecture, and Storage, NAS 2014, Institute of Electrical and Electronics Engineers Inc., 2014, pp. 207–215, https://doi.org/10.1109/NAS.2014.39.
[17] S. Eyerman, L. Eeckhout, T. Karkhanis, J.E. Smith, A performance counter architecture for computing accurate CPI components, ACM SIGPLAN Not. 41 (2006) 175–184, https://doi.org/10.1145/1168918.1168880.
[18] M. Jarus, A. Oleksiak, T. Piontek, J. Weglarz, Runtime power usage estimation of HPC servers for various classes of real-life applications, Future Generat. Comput. Syst. 36 (2014) 299–310, https://doi.org/10.1016/j.future.2013.07.012.
[19] M. Witkowski, A. Oleksiak, T. Piontek, J. Weoglarz, Practical power consumption estimation for real life HPC applications, Future Generat. Comput. Syst. 29 (2013) 208–217, https://doi.org/10.1016/j.future.2012.06.003.
[20] Y. Lv, B. Sun, Q. Luo, J. Wang, Z. Yu, X. Qian, Counterminer: mining big performance data from hardware counters, in: Proceedings of the Annual International Symposium on Microarchitecture, MICRO, IEEE Computer Society, 2018, pp. 613–626, https://doi.org/10.1109/MICRO.2018.00056.
[21] Y.C. Wang, J. Wang, J.K. Chen, S.C. Zuo, X.M. Su, J. Lin, NeoMPX: characterizing and improving estimation of multiplexing hardware counters for PAPI, in: Proceedings - IEEE International Conference on Cluster Computing, ICCC, Institute of Electrical and Electronics Engineers Inc., 2020, pp. 47–56, https://doi.org/10.1109/CLUSTER49012.2020.00015.
[22] S. Karandikar, A. Ou, A. Amid, H. Mao, R. Katz, B. Nikolic, K. Asanovic, FirePerf: FPGA-accelerated full-system hardware/software performance profiling and co-design, in: International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS, Association for Computing Machinery, 2020, pp. 715–731, https://doi.org/10.1145/3373376.3378455.
[23] J. Choi, G. Park, D. Nam, Interference-aware co-scheduling method based on classification of application characteristics from hardware performance counter using data mining, Cluster Comput. 23 (2020) 57–69, https://doi.org/10.1007/s10586-019-02949-7.

[24] A. Yasin, A Top-Down method for performance analysis and counters architecture, in: ISPASS 2014 - IEEE International Symposium on Performance Analysis of Systems and Software, IEEE Computer Society, 2014, pp. 35–44, https://doi.org/10.1109/ISPASS.2014.6844459.
[25] M. Stillwell, F. Vivien, H. Casanova, Dynamic fractional resource scheduling for HPC workloads, in: Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, IPDPS, 2010, https://doi.org/10.1109/IPDPS.2010.5470356.
[27] M. Burtscher, B.-D. Kim, J. Diamond, J. Mccalpin, L. Koesterke, J. Browne, PerfExpert, An Easy-To-Use Performance Diagnosis Tool for HPC Applications, 2010.
[28] J. Zhang, R.J. Figueiredo, Application classification through monitoring and learning of resource consumption patterns, in: 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, IEEE Computer Society, 2006, https://doi.org/10.1109/IPDPS.2006.1639378.
[29] M. Dimakopoulou, S. Eranian, N. Koziris, N. Bambos, Reliable and efficient performance monitoring in linux, in: International Conference for High Performance Computing, Networking, Storage and Analysis, SC, IEEE Computer Society, 2016, pp. 396–408, https://doi.org/10.1109/SC.2016.33.
[30] R. v Lim, D. Carrillo-Cisneros, W. Alkowaileet, I. Scherson, Computationally efficient multiplexing of events on hardware counters, in: Linux Symposium, Citeseer, 2014, pp. 101–110.
[31] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, R. Hundt, Google-Wide Profiling: a continuous profiling infrastructure for data centers, IEEE Micro 30 (2010) 65–78, https://doi.org/10.1109/MM.2010.68.
[32] A. Hinneburg, D.A. Keim, Optimal Grid-Clustering : towards Breaking the Curse of Dimensionality in High-Dimensional Clustering, 1999. http://www.ub.uni-konstanz.de/kops/volltexte/2008/7041/.
[33] D.L. Donoho, High-dimensional data analysis: the curses and blessings of dimensionality, AMS Math Challenges Lecture 1 (2000) 32.
[34] O.O. Aremu, D. Hyland-Wood, P.R. McAree, A machine learning approach to circumventing the curse of dimensionality in discontinuous time series machine data, Reliab. Eng. Syst. Saf. (2020) 195, https://doi.org/10.1016/j.ress.2019.106706.
[35] H. Zhou, Z. Deng, Y. Xia, M. Fu, A new sampling method in particle filter based on Pearson correlation coefficient, Neurocomputing 216 (2016) 208–215, https://doi.org/10.1016/j.neucom.2016.07.036.
[36] P. Wolfe, A duality theorem for non-linear programming, Q. Appl. Math. 19 (1961) 239–244.
[37] J.P. Vert, K. Tsuda, B. Scholkopf, A primer on kernel methods, Kernel methods in computational biology 47 (2004) 35–70.
[38] Y. Goldberg, M. Elhadad, splitSVM: fast, space-efficient, non-heuristic, polynomial kernel computation for NLP applications, in: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers, 2008, pp. 237–240.
[39] H.-T. Lin, C.-J. Lin, A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods, n.d, https://www.researchgate.net/publication/2478380.
[40] S. An, W. Liu, S. Venkatesh, Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression, Pattern Recogn. 40 (2007) 2154–2162.
[41] M. Ferro, V.P. Kloh, M. Gritz, V. de Sá, B. Schulze, Predicting runtime in HPC environments for an efficient use of computational resources, in: Anais do XXII Simpósio em Sistemas Computacionais de Alto Desempenho, SBC, 2021, October, pp. 72–83.
[42] N. Ding, V.W. Lee, W. Xue, W. Zheng, APMT: an automatic hardware counter-based performance modeling tool for HPC applications, CCF Transac. High Perform. Comput. 2 (2020) 135–148.
[43] N. Ding, S. Xu, Z. Song, B. Zhang, J. Li, Z. Zheng, Using hardware counter-based performance model to diagnose scaling issues of HPC applications, Neural Comput. Appl. 31 (2019) 1563–1575.
[44] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, F. Monrose, SoK: the challenges, pitfalls, and perils of using hardware performance counters for security, in: 2019 IEEE Symposium on Security and Privacy (SP), IEEE, 2019, May, pp. 20–38.
[45] L.L. Woo, Hardware performance counters (HPCs) for anomaly detection, Hardware Supply Chain Security: Threat Modelling, Emerging Attacks and Countermeasures (2021) 147–165.
[46] C. Woralert, J. Bruska, C. Liu, L. Yan, High frequency performance monitoring via architectural event measurement, in: 2020 IEEE International Symposium on Workload Characterization (IISWC), IEEE, 2020, October, pp. 114–122.