Journal of Network Intelligence
Taiwan Ubiquitous Information
 ISSN 2414-8105
Volume **10**, Number **4**, November **2025**

Modeling and Application of Performance Optimization for Parallel Algorithms

Jun-Feng Wang, Gang-Yi Ding, Yu-Gang Li

School of Computer Science
Beijing Institute of Technology
Beijing 100081, P. R. China
bitwind@qq.com, 694458752@qq.com, 703034970@qq.com

Fu-Quan Zhang*

Fujian Provincial Key Laboratory of Information Processing and Intelligent Control
Minjiang University
Fuzhou 350117, P. R. China
8528750@qq.com

*Corresponding author: Fu-Quan Zhang
Received July 5, 2023, revised August 12, 2024, accepted January 15, 2025.

Abstract. *In many large-scale scientific and engineering research fields, simulation computing has played a significant role in its development. This type of computing has the characteristics of high complexity and large computational load, and traditional serial algorithms can no longer meet their computing needs. Parallel processing has become a key technology for solving problems. How to give full play to the performance of high-performance computing systems and design parallel algorithms with optimal performance has become a hot issue in current research. This paper proposes a parallel algorithm performance optimization analysis model to guide the design and optimization of parallel algorithms. This model quantitatively analyzes the design optimization of parallel algorithms from two aspects: parallel systems and parallel algorithms. It provides expressions for the acceleration, efficiency, cost, and other indicators of the algorithm, and points out performance bottlenecks in the algorithm. This provides theoretical guidance for maximizing the performance of parallel systems and optimizing parallel algorithms. Based on this model, this article conducts research and analysis on the parallel optimization design of large matrix multiplication algorithms, and conducts experimental verification, greatly improving the parallel efficiency of matrix multiplication.*
Keywords: computational intensive applications; performance optimization analysis models; large matrix multiplication

1. **Introduction.** In the process of scientific development, theory, calculation, and experiment are the three major means of scientific research [1], playing a significant role in its development process. In practical research, due to the lack of suitable theoretical models, inability to achieve experimental conditions, or excessively expensive experimental costs, simulation calculations have become the only or main means to solve such problems [2]. This type of computing has the characteristics of high complexity and large computational load, and traditional serial algorithms can no longer meet its computing needs. Parallel processing has become a key technology to solve its computing problems.

At the same time, the architecture of high-performance computer systems continues to develop, the performance continues to improve, and the hardware scale continues to

expand. The architecture has developed from a single CPU to a CPU+GPU heterogeneous parallel system. The computing power has increased to tens of billions of times, and the integration degree has reached millions of computing cores, providing sufficient computing power for parallel computing. Researchers are actively investing in the study of parallel algorithm design, and various parallel computing models are constantly being proposed [3]. However, there is a lack of a parallel model that considers both machine performance and guides algorithm parallel optimization in current research.

To address this issue, this paper proposes a performance optimization model for parallel algorithms. This model quantitatively analyzes the design optimization of parallel algorithms from two aspects: parallel systems and parallel algorithms. In terms of parallel systems, it gives a quantitative analysis from three aspects: parallelism, storage performance, and the relationship between parallelism and storage, in order to maximize the computing power and storage performance of parallel systems. In terms of parallel algorithm, it provides a method from the decomposition, dependency and execution of parallel algorithms, and gives the feasibility analysis of the performance of parallel algorithms in terms of time complexity and scalability.

The model provides the expressions of acceleration, efficiency, overhead and other indicators of the algorithm, and points out the performance bottlenecks in the algorithm, which provides methods and theoretical support for the performance optimization analysis of parallel algorithms. Finally, based on this model, this paper selects a typical algebraic application: large matrix multiplication, and conducts parallel optimization design of parallel algorithms from the aspects of decomposition, dependency, execution and memory, which greatly improves the parallel efficiency of large matrix multiplication.

The organization of this paper is as follows. The current development status of parallel computing performance modeling research is introduced in Section 2. In Section 3, a performance optimization analysis model for parallel algorithms is presented. Section 4 introduces the research on model-based parallel optimization of large matrix multiplication. Section 5 provides a summary and analysis.

2. **Related Works.**

The performance optimization of parallel algorithms is a hot topic in current research, and performance models are mainly divided into three types: analytical performance models, empirical performance models, and hybrid performance models.

2.1. **Analytical Performance Model.**

The analytical performance model is based on a detailed analysis of parallel algorithms and selects feature parameters to evaluate their performance.

Samuel *et al.* [4] proposed the Roofline Model in 2009, which further improves application performance by analyzing the upper performance limit. Sabela *et al.* [5] used the performance analysis tools STREAM and BenchIT to establish a performance analysis model for Intel Xeon Phi KNL [6, 7] multi-core processors. Based on the model, the communication path between computing cores was optimized, and the optimal number of threads for application execution was further provided. Bauer *et al.* [8] provided a performance analysis model of *su3_rmd* in Quantum Chromodynamics (QCD). Hoisie *et al.* [9] proposed a performance model for the Sweep3D tool by analyzing factors such as the logical relationship of computing tasks, parallel execution order, and data partitioning method. Mathis *et al.* [10] presented a performance model for a particle transport program. Kyle *et al.* [11] used the ASPEN model tool to establish a performance model for programs under analysis.

The analytical performance model is convenient for establishing a mathematical framework for performance analysis of application programs, and performance optimization

methods can be derived through calculation. However, this model has high requirements for users, and the modeling process is relatively complex.

2.2. **Empirical Performance Model.** The empirical performance model is a black-box analysis model that does not focus on the internal principles of the application, but establishes a model for performance analysis based on statistical analysis of its execution behavior.

Bradley *et al.* [12, 13] established a performance model for the NPB tool based on regression statistical analysis. Bhattacharyya *et al.* [14] constructed performance models for NPB and HPCCG tools based on compression analysis, assembly, and compilation techniques. Lee *et al.* [15] developed performance models for MultiGrid computing and High Performance Linpack using polynomial regression and artificial intelligence algorithms. Calotoiu *et al.* [16] established models for Sweep3D, MILC, and HOMME, and implemented predictive analysis of performance bottlenecks. Prasanna [17] and Valerie [18] developed multi-input performance models based on machine learning algorithms. Marahe [19] analyzed prediction accuracy across different artificial intelligence model frameworks.

The modeling cost of empirical performance models is relatively low and easy to automate. However, as it relies on statistical analysis of large amounts of experimental data, its prediction accuracy is easily affected by data quality. Meanwhile, feature selection for empirical models is challenging: (1) too few parameters increase prediction error; (2) too many parameters make model construction computationally expensive.

2.3. **Hybrid Performance Model.** A hybrid performance model combines multiple modeling approaches. It compensates for the shortcomings of analytical models when facing complex problems and for the empirical models' lack of theoretical interpretability.

Martin *et al.* [20] proposed the Perf Expert model to solve the problem of system serial performance analysis. Alexeev *et al.* [21] proposed a hybrid model for FMO applications. Kim *et al.* [22] built a hybrid model for CCSM applications, which analyzes internal coupling relationships and predicts execution time during the coupling process.

The hybrid performance model combines the strengths of multiple models but lacks universality and is often tailored for specific applications. Moreover, such models are difficult to reuse, which limits their scope of application.

3. **A Performance Optimization Analysis Model for Parallel Algorithms.** In this section, we propose a performance optimization analysis model for parallel algorithms, which includes both parallel system analysis and parallel algorithm analysis.

3.1. **Parallel System Analysis.** Parallel systems are quantified from three dimensions: system parallelism, the relationship between parallelism levels and storage, and storage access performance. Parallel functions H_P, storage function H_M, relation function H_B, and storage performance analysis parameters are defined.

The parallel function is defined as $H_P = \{p_0, p_1, \ldots, p_{hp}\}$, where hp indicates the number of super steps in the system (the number of parallel layers), and the i-th layer contains p_i nodes ($1 \leq i \leq hp$).

The relation function is defined as $H_B = \{hl_1, hl_2, \ldots, hl_{hp}\}$, where hl_i ($1 \leq i \leq hp$) provides communication methods on the i-th layer. When hl_i is a positive integer k, it indicates communication through the k-th level storage; when hl_i is k^+, it indicates message-passing communication.

The storage function is $H_M = \{U_i, l_i, B_i, L_i\}, \quad 1 \leq i \leq h_m + h_c$, where U_i is the size of MU_i, l_i is the reference block length of MU_i, B_i is the access bandwidth, and L_i is the

access latency. Typically, $l_{i-1} < l_i, \quad B_{i-1} > B_i, \quad L_{i-1} < L_i$. Functions $f_i(l_i, B_i, L_i)$ provide performance formulas for hierarchical memories MU_i $(1 \le i \le h_m)$, and sequentially for communication functions when $h_m \le i \le h_m + h_c$.

Storage Performance Analysis Parameters:

- Spatial complexity: M_0, the storage space required for the problem to be solved.
- Storage access complexity: M_1, the actual storage access volume when implemented on a specific system.
- Air-to-storage ratio: $\beta = M_0/M_1$.
- Air ratio: $r_1 = W_1/M_0$, the number of calculations per unit storage space, where W_1 is the total computation count.
- Visiting ratio: $r_2 = W_1/M_1$, the number of computations supported per unit of actual storage access.
- Buffer ratio: $r = C_c/M_0$, where C_c is the cache capacity of the system.
- Compute–access time ratio: $\beta_1 = W_1/(M_1 \times t_1)$, representing the number of computations supported by memory accesses completed per unit time, where t_1 is the average access time.

3.2. **Parallel Algorithm Analysis.** The research of parallel Analysis of algorithms starts from the problem decomposition. The decomposed problem is regarded as the basic element of the model, and the dependency relationship between them is defined according to the actual situation of the problem. The decomposition, dependency, execution and memory access of the problem are given in the form of a matrix, which describes the essential characteristics of its corresponding algorithm. The model makes feasibility analysis on the performance of parallel algorithms in terms of time complexity and scalability, and gives the expressions of acceleration, efficiency, overhead and other indicators of the algorithm, which can be used to analyze the performance of complex algorithms in solving practical problems, and point out the performance bottleneck in the algorithm, providing methods and theoretical support for the performance optimization analysis of parallel algorithms.

Definition 1 (Calculation Problem) A calculation problem β_{N_r} can be expressed as a mathematical problem with a relationship between input and output functions:

$$\beta_{N_r} : In(\beta_{N_r}) \to Out(\beta_{N_r}) \tag{1}$$

Where N_r represents the size of the input data, and $r \in \mathbb{N}$. Calculation problems β_{N_r} can be defined as the following triplets:

$$\beta_{N_r} = (N_r, In(\beta_{N_r}), Out(\beta_{N_r})) \tag{2}$$

The key to parallel algorithm design is problem decomposition, where some or all of the decomposed problems can be executed in parallel, and there is usually a certain dependency relationship between these problems. We represent the dependency relationships between problems by defining dependency groups and dependency matrices.

Definition 2 (Dependency Group) Define relationship group (ε, π), and define π_ε as the strict internal dependency relationship of ε. If any part A in ε depends on part B in ε, it is denoted as $A\pi_\varepsilon B$, and recorded as $A \leftarrow B$. If there is no dependency relationship between A and B, it is recorded as $A \not\leftarrow B$. The relationship group (ε, π) with strict internal dependency relationship π_ε is called a dependency group, denoted as $(\varepsilon, \pi, \pi_\varepsilon)$.

Definition 3 (Dependency Matrix) Given the dependency group $(\varepsilon, \pi, \pi_\varepsilon)$, matrix F, with a scale of $r_D \times c_D$. Element of matrix $d_{i,j} \in (\varepsilon, \pi)$, $\forall i \in [0, r_D - 1]$, $\forall s, j \in [0, c_D - 1]$, $d_{i,j} \not\leftrightarrow d_{i,s}$, $\forall i \in [1, r_D - 1]$, $\forall j \in [0, c_D - 1]$, $\exists q \in [1, c_D - 1]$ $d_{i,j} \leftarrow d_{i-1,q}$, and other elements set to 0, matrix F is called a dependency matrix.

According to the definition of the dependency matrix, matrix F is unique. c_F represents concurrency of the dependency group $(\varepsilon, \pi, \pi_\varepsilon)$, r_F represents the dependency of ε. Concurrency measures the concurrency between subproblems in a dependency group, which depends on the number of columns in matrix F.

Definition 4 (Problem Decomposition) Given a problem β_{N_r}, a subset of any finite computational problem $\{\beta_{N_i}\}$, $i = 0, \ldots, k-1$, $\beta_{N_r} \leftarrow \beta_{N_i}$ is called a decomposition of problem β_{N_r} where $N_i < N_r$, $\sum_{i=0}^{k-1} N_i \geq N_r$. β_{N_i} is the subproblem of β_{N_r}. A decomposition of β_{N_r} can be expressed as:

$$D_k(\beta_{N_r}) := \{\beta_{N_0}, \ldots, \beta_{N_{k-1}}\} \tag{3}$$

The definition of problem decomposition is represented as follows:

$$D_k(\beta_{N_r}) = \left(\sum_{i=0}^{k-1} N_i,\ In(\beta_{N_i}),\ Out(\beta_{N_i}) \right) \tag{4}$$

Definition 5 (Decomposition Matrix) In Definition 3, the dependency matrix $D_k(\beta_{N_r})$ of the problem represents the dependency relationship between β_{N_r} subproblems, and in Definition 2, the dependency group $(D_k(\beta_{N_r}), g_{sol})$ is given, where g_{sol} is the arbitrary relationship between the two elements β_{N_i} and β_{N_j} in $D_k(\beta_{N_r})$, and has a strict partially ordered set relationship $\pi_{D_k(\beta_{N_r})}$. Construct a decomposition matrix F based on the dependency matrix $D_k(\beta_{N_r})$, denoted as $M_D(D_k(\beta_{N_r}))$, or abbreviated as M_{D_k}. For a given $D_k(\beta_{N_r})$, we use c_{D_k} to represent the number of columns in the matrix, which is also the unique concurrency of β_{N_r}, and use r_{D_k} to represent the number of rows in the matrix, which is also the unique dependency of β_{N_r}. Concurrency measures the concurrency between subproblems of β_{N_r}.

Definition 6 (Calculation Operation) Define M_p as a parallel computing environment with $P > 1$ computing nodes, and define the operator I^j as the corresponding relationship between R^s and R^t, where $s, t \in \mathbb{N}$ and are all positive integers. The set $\mathrm{Cop}_{M_p} := \{I^j\}, j \in [0, q-1], q \in \mathbb{N}$, where the operator I^j is not duplicated, represents the logical computing power of M_p.

Definition 7 (Problem Solvable) $\exists D_k(\beta_{N_r}) \in \mathbb{D}\beta_{N_r} : \forall \beta_{N_j} \in D_k(\beta_{N_r}), \exists I^j \in \mathrm{Cop}_{M_p} : I^j[\beta_{N_j}] = S(\beta_{N_j})$, which means that if there is any relationship $\theta : \beta_{N_j} \in D_k(\beta_{N_r}) \in \mathbb{D}\beta_{N_r} \rightarrow I^j \in \mathrm{Cop}_{M_p}$, and if the above conditions are met, we can say that problem β_{N_r} can be solved in the parallel computing environment M_p.

An algorithm is a combination of a series of operations to solve a problem. We define an algorithm as a set of operators to solve the problems β_{N_r}.

Definition 8 (Algorithm) For a given $D_k(\beta_{N_r})$, the algorithm for solving problem β_{N_r} is represented as:

$$A_{(D_k(\beta_{N_r}),M_p)} = \{I^{(i_0)}, I^{(i_1)}, \ldots, I^{(i_k)}\} \tag{5}$$

And that $I^{(i_k)} \circ I^{(i_{k-1})} \circ \cdots \circ I^{(i_0)}[\beta_{N_r}] = S(\beta_{N_r})$, thus there exists a correspondence between two mappings:

$$\gamma : \beta_{N_v} \in D_k(\beta_{N_r}) \in D\beta_{N_r} \leftrightarrow I^{(i_j)} \in A_{(D_k(\beta_{N_r}),M_p)} \tag{6}$$

where $j \in [0, \mathrm{card}(\mathrm{Cop}_{M_p}) - 1]$. Each ordered subset of $A_{(D_k(\beta_{N_r}),M_p)}$ is a subalgorithm of $A_{(D_k(\beta_{N_r}),M_p)}$. For the sake of symbol simplicity and without ambiguity, the algorithm is abbreviated as $A_{(k,P)}$.

Definition 9 (Granularity Set of Algorithm) For a given algorithm $A_{(k,P)}$, a subset $G(A_{(k,P)})$ composed of its different operators defines the granularity set of $A_{(k,P)}$. Two algorithms $A^i_{(k,P)} = \{I^{(i_0)}, I^{(i_1)}, \ldots, I^{(i_k)}\}$, $A^j_{(k,P)} = \{I^{(j_0)}, I^{(j_1)}, \ldots, I^{(j_k)}\}$, if $G(A^i_{(k,P)}) \equiv G(A^j_{(k,P)})$, then these two algorithms have the same granularity.

Definition 10 (Algorithm's Quotient Set AL/ϱ) A corresponding relationship has been defined as follows:

$$\varphi : A_{(k,P)} \in AL \to D_k(\beta_{N_r}) \in D\beta_{N_r} \tag{7}$$

To derive full mapping of ϱ, which is the equivalence relation of AL itself, such that:

$$\varrho(A_{(k,P)}) = \{\tilde{A}_{(k,P)} \in AL : \varphi(\tilde{A}_{(k,P)}) = \varphi(A_{(k,P)})\} \tag{8}$$

The set $\varrho(A_{(k,P)})$ is composed of a class of algorithms corresponding to the same decomposition $D_k(\beta_{N_r}) \in D\beta_{N_r}$ in the algorithm set AL. ϱ derives quotient set AL/ϱ, whose elements are disjoint and are finite subsets of algorithm set AL determined by equivalence relation ϱ, that is, they are equivalence classes under ϱ.

Definition 11 (Complexity) The cardinality of $A_{(k,P)}$, defined as $C(A_{(k,P)})$, is called the complexity of $A_{(k,P)}$.

$$C(A_{(k,P)}) := \mathrm{card}(A_{(k,P)}) = k \tag{9}$$

$C(A_{(k,P)}) = k$ is equivalent to the number of nonempty elements of M_{D_k}. According to Formula (6), the double mapping of γ can be obtained as follows:

$$\mathrm{card}(A_{(k,P)}) = \mathrm{card}(D_k(\beta_{N_r})) = k, \quad \forall A_{(k,P)} \in \varrho(A_{(k,P)}) \tag{10}$$

Each algorithm belonging to the same equivalence class has the same complexity. Therefore, each element $\varrho(A_{(k,P)})$ in the quotient set AL/ϱ is associated with an integer (complexity), and we can sort the equivalence classes in the quotient set, thus obtaining the algorithm to solve the minimum complexity of problem β_{N_r}.

Definition 12 (Execution Matrix) According to Definition 2, we introduce a dependency group $(P(A_{(k,P)}), *, \pi_{A_{(k,P)}})$, where $P(A_{(k,P)})$ is the set of all subalgorithm sets of algorithm $A_{(k,P)}$, and $\pi_{A_{(k,P)}}$ is a strict partially ordered set relationship between any two elements in $P(A_{(k,P)})$. We construct the matrix F of size $r_E \times c_E$ which is used as the dependency matrix. Next, we define the matrix as an execution matrix and use the symbol $M_E(A_{(k,P)}) = (e_{i,j})$ to represent it. If there is no ambiguity, it is abbreviated as $M_{E_{(k,p)}}$.

Because $\mathrm{card}(A_{(k,P)}) = \mathrm{card}(D_k(\beta_{N_r}))$, M_{D_k} and $M_{E_{(k,p)}}$ have the same number of nonempty elements k, as long as $P \geq 1$. If $c_E = P = c_{D_k}$, then there exists algorithm $A_{(k,P)}$, whose execution matrix $M_{E_{(k,p)}}$ and decomposition matrix M_{D_k} have exactly the same structure.

Definition 13 (Amplification Factor) If $A_{(k_i,P)}$ and $A_{(k_j,P)}$ have the same granularity set, then the ratio $Sc_{\mathrm{up}}(A_{(k_i,P)}, A_{(k_j,P)}) := \frac{k_i}{k_j}$ is the amplification ratio of $\varrho(A_{(k_i,P)})$ relative to $\varrho(A_{(k_j,P)})$. By Definition 11, we obtain:

$$Sc_{\mathrm{up}}(A_{(k_i,P)}, A_{(k_j,P)}) := \frac{C(A_{(k_i,P)})}{C(A_{(k_j,P)})} \tag{11}$$

Definition 14 (Row Execution Time) We define $T_r(A_{(k,P)})$ as the execution time of the r-th row of matrix $M_{E_{(k,p)}}$ as follows:

$$T_r(A_{(k,P)}) := \max_{j \in [0, c_E - 1]} t_{r_j} \tag{12}$$

Definition 15 (Execution Time)

$$T(A_{(k,P)}) := \sum_{r=0}^{r_E - 1} T_r(A_{(k,P)}) \tag{13}$$

$$\beta^{calc}_{M_{E_{(k,p)}}} := \sum_{r=0}^{r_E-1} \beta^{calc}_{(r,M_{E_{(k,p)}})}, \quad \beta^{calc}_{M_{E_{(k,p)}}} \geq r_E \tag{14}$$

$$T(A_{(k,P)}) = \beta^{calc}_{M_{E_{(k,p)}}} \cdot t_{calc} \tag{15}$$

$$\beta^{calc}_{sum,M_{E_{(k,p)}}} := \sum_{r=0}^{r_E-1} \sum_{r=0}^{c_E-1} \beta^{calc}_{(ij,M_{E_{(k,p)}})} \tag{16}$$

Definition 16 (Parallel Execution Time) $T_{\text{par}}(A_{(k,P)}) := \sum_{r=0}^{r_{\text{par}}-1} T_{i_r}(A_{(k,P)})$ defines the parallel execution time of $A_{(k,P)}$.

Definition 17 (Serial Execution Time) $T_{\text{seq}}(A_{(k,P)}) := \sum_{q=0}^{r_{\text{seq}}-1} T_{i_q}(A_{(k,P)})$ defines the serial execution time of $A_{(k,P)}$.

$$T(A_{(k,P)}) = T_{\text{par}}(A_{(k,P)}) + T_{\text{seq}}(A_{(k,P)}) \tag{17}$$

This indicates that by looking at matrix $M_{E_{(k,p)}}$, the model provides the execution time of the parallel and serial parts of algorithm $A_{(k,P)}$.

$$R^{calc}(A_{(k,P)}) := \frac{\beta^{calc}_{M_{E_{(k,p)}}}}{r_E} \tag{18}$$

R^{calc} is a parameter of algorithm $A_{(k,P)}$, depending on the subalgorithm with the highest computational complexity in $A_{(k,P)}$.

$$T(A_{(k,P)}) = R^{calc}(A_{(k,P)}) \cdot r_E \cdot t_{calc} = \sum_{r=0}^{r_E-1} \beta^{calc}_{(r,M_{E_{(k,p)}})} \cdot t_{calc} \tag{19}$$

If $P = 1$, due to $r_E = C(A_{(k,1)}) = k$, so $R^{calc}(A_{(k,1)}) := (\beta^{calc}_{all,M_{E_{(k,p)}}})/k$.

From Formula (17), it can be obtained that:

$$T(A_{(k,1)}) = k \cdot R^{calc}(A_{(k,1)}) \cdot t_{calc} \tag{20}$$

$$T(A_{(k,P)}) \geq r_D \cdot R^{calc}(A_{(k,P)}) \cdot t_{calc} \tag{21}$$

The minimum value is only obtained when $r_E = r_D$.

$$T(A_{(k,P)}) = (r_{\text{seq}} + r_{\text{par}}) \cdot R^{calc}(A_{(k,P)}) \cdot t_{calc} \tag{22}$$

Definition 18 (Acceleration Ratio AL/ϱ) For a given β_{N_r}, $A_{(k',1)} \in \varphi^{-1}(D_{k'}(\beta_{N_r}))$, two different decompositions $D_k(\beta_{N_r})$ and $D_{k'}(\beta_{N_r})$, where M_1 and M_P only differ in the number of processing units. If $G(A_{(k,P)}) = G(A_{(k',P)})$, then the acceleration of $A_{(k,P)}$ relative to $A_{(k',1)}$ is:

$$Sp(A_{(k,P)}, A_{(k',1)}) := Sc_{\text{up}}(A_{(k,P)}, A_{(k',1)}) \cdot \frac{T(A_{(k,1)})}{T(A_{(k,P)})} = \frac{k'}{k} \cdot \frac{\beta^{calc}_{sum,M_E(A_{(k,P)})}}{\beta^{calc}_{M_E(A_{(k,P)})}} \tag{23}$$

Definition 19 (Algorithm Cost) The cost definition of algorithm $A_{(k,P)}$ is as follows:

$$Q(A_{(k,P)}) = c_E \cdot r_E \cdot R^{calc}(A_{(k,P)}) \cdot t_{calc} \tag{24}$$

Definition 20 (Algorithm Overhead) The algorithm overhead is defined as follows:

$$Oh(A_{(k,P)}) := Q(A_{(k,P)}) - Q(A_{(k,1)}) = (c_E \cdot \beta^{calc}_{M_{E_{(k,P)}}} - \beta^{calc}_{sum,M_{E_{(k,1)}}}) \cdot t_{calc} \tag{25}$$

Definition 21 (Algorithm Efficiency) The algorithm efficiency is defined as follows:

$$Ef(A_{(k,P)}) := \frac{Sp(A_{(k,P)})}{P} = \frac{\beta^{calc}_{sum,M_{E_{(k,1)}}}}{N^E_P \cdot R^{calc}(A_{(k,P)})} \tag{26}$$

4. **Research on Model Based Parallel Optimization of Large Matrix Multiplication.** In algebra, matrix operation is the most basic data operation, which is a typical calculation-intensive computation, and has extensive applications in mathematical analysis, differential equations, computer science and other disciplines. This chapter focuses on the parallel optimization analysis of matrix block multiplication algorithms. We have selected two classic parallel algorithms for matrix block multiplication for comparative analysis.

Algorithm 1. $C = AB$ (where matrices A, B, and C are all $n \times n$ matrices), divides matrices A and C into n_p submatrices A_{nk} and C_{nk} of size $n_0 \times n$ (where n_p is the number of processes, $n_0 = n/n_p$, $0 \le k \le n_p - 1$). The $(k+1)$-th submatrix is stored on node k. Similarly, divide matrix B into n_p submatrices B_{nk} of $n \times n_0$, with the $(k+1)$-th submatrix stored on node k.

In the operation, each node k independently performs the corresponding local operation $C_{nk} += A_{nk}B_{nk}$, and then passes the submatrix B_{nk} to the adjacent node $\mathrm{mod}(k-1+n_p, n_p)$. This process is repeated n_p times, and in the last iteration only data calculation is performed without data transfer.

Algorithm 2. $C = AB$ (where matrices A, B, and C are all $n \times n$ matrices), divides matrices A, B, and C into n_p submatrices A_k, B_k, and C_k of size $n_1 \times n_1$ (assuming that the matrix can be evenly divided, $n_1 = n/\sqrt{n_p}$), and stores the $(k+1)$-th submatrix on node k (where $0 \le k \le n_p - 1$).

In the operation, each node k independently performs the local operation $C_{nk} += A_{nk}B_{nk}$, and then passes submatrix A_k to $\mathrm{mod}(k-1+\sqrt{n_p}, \sqrt{n_p})$, and submatrix B_{nk} to $\mathrm{mod}(k-\sqrt{n_p}+n_p, n_p)$. This process is repeated $\sqrt{n_p}$ times, and the last time only performs data calculation without data transfer.

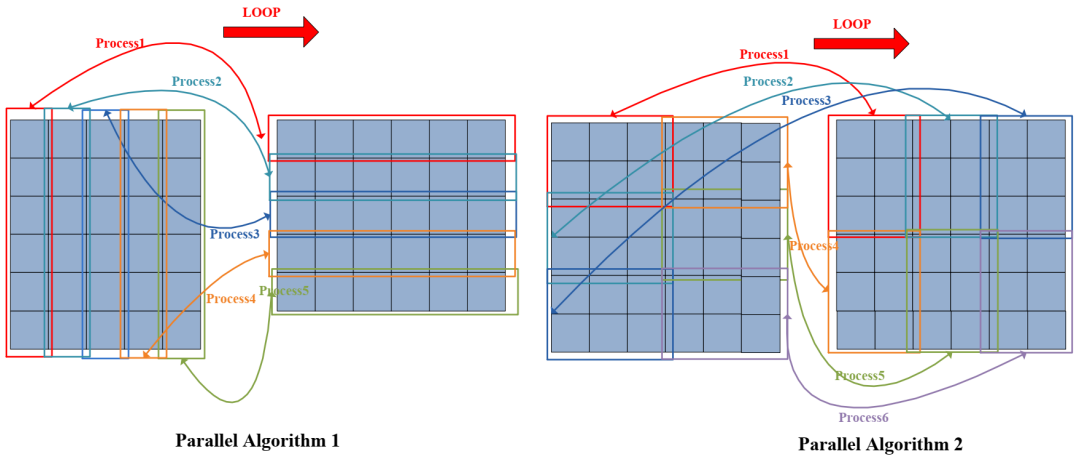

Figure 1. Parallel schematic diagram of Algorithm 1 and Algorithm 2.

For the two parallel algorithms mentioned above, we conduct a comprehensive analysis based on the performance analysis model from multiple aspects such as the degree of parallelism and communication between parallelisms, to determine the performance advantages and disadvantages of the two algorithms.

Let B_{nn} represent the computational problem of multiplying two matrices of size $n \times n$. Algorithm (1), $D^1(B_{nn}) \in DB_{nn}$ is a decomposition of problem B_{nn}, and its decomposition matrix is:

$$M_D(D^1(B_{nn})) = \begin{bmatrix} B^1_{(1n_0)} & B^1_{(2n_0)} & \cdots & B^1_{(n_pn_0)} \\ B^2_{(1n_0)} & B^2_{(2n_0)} & \cdots & B^2_{(n_pn_0)} \\ \vdots & \vdots & \ddots & \vdots \\ B^{(n_p)}_{(1n_0)} & B^{(n_p)}_{(2n_0)} & \cdots & B^{(n_p)}_{(n_pn_0)} \\ B^n_{nn} & \varnothing & \cdots & \varnothing \end{bmatrix}$$

The parallelism of Algorithm (1) is n_p. The dependency of Algorithm (1) is $r_{D^1} = n_p+1$. Let's assume problem B_{nn} is solvable on machine M_p ($p = n_p$). "XX" represents matrix multiplication, and "++" represents matrix addition.

For the given $D^1(B_{nn})$, $A_{(D^1(B_{nn}),M_{n_p})} = \{XX_1, XX_2, \ldots, XX_{n_p}, XX_{(n_p+1)}, \ldots, XX_{(n_0n_p)}, ++\}$ is the algorithm to solve problem B_{nn}. So the execution matrix of this algorithm is:

$$M_{E_{(n,n_p)}} = \begin{bmatrix} XX_1 & XX_2 & \cdots & XX_{n_p} \\ XX_{(n_p+1)} & XX_{(n_p+2)} & \cdots & XX_{(2n_p)} \\ \vdots & \vdots & \ddots & \vdots \\ XX_{((n_p-1)n_p+1)} & XX_{((n_p-1)n_p+2)} & \cdots & XX_{(n_pn_p)} \\ B^n_{nn} & \varnothing & \cdots & \varnothing \end{bmatrix}$$

Algorithm 2. $D^2(B_{nn}) \in DB_{nn}$ is a decomposition of Problem B_{nn}, and its decomposition matrix is:

$$M_D(D^2(B_{nn})) = \begin{bmatrix} B^1_{(1n_1)} & B^1_{(2n_1)} & \cdots & B^1_{(n_pn_1)} \\ B^2_{(1n_1)} & B^2_{(2n_1)} & \cdots & B^2_{(n_pn_1)} \\ \vdots & \vdots & \ddots & \vdots \\ B^{\sqrt{n_p}}_{(1n_1)} & B^{\sqrt{n_p}}_{(2n_1)} & \cdots & B^{\sqrt{n_p}}_{(n_pn_1)} \\ B^n_{nn} & \varnothing & \cdots & \varnothing \end{bmatrix} \tag{27}$$

The parallelism of Algorithm (2) is n_p. The dependency of Algorithm (2) is $r_{D^2} = \sqrt{n_p}+1$. Let's assume problem B_{nn} is solvable on machine M_p ($p = n_p$). "XX" represents matrix multiplication, and "++" represents matrix addition.

For a given $D^2(B_{nn})$, $A_{(D^2(B_{nn}),M_{n_p})} = \{XX_1, XX_2, \ldots, XX_{n_p}, XX_{(n_p+1)}, \ldots, XX_{(\sqrt{n_p}n_p)}, ++\}$ is the algorithm to solve problem B_{nn}. So the execution matrix of this algorithm is:

$$M_{E_{(n,n_p)}} = \begin{bmatrix} XX_1 & XX_2 & \cdots & XX_{n_p} \\ XX_{(n_p+1)} & XX_{(n_p+2)} & \cdots & XX_{(2n_p)} \\ \vdots & \vdots & \ddots & \vdots \\ XX_{((\sqrt{n_p}-1)n_p+1)} & XX_{((\sqrt{n_p}-1)n_p+2)} & \cdots & XX_{(\sqrt{n_p}n_p)} \\ B^n_{nn} & \varnothing & \cdots & \varnothing \end{bmatrix}$$

Below is a detailed analysis of the computational and communication costs of two matrices.

(1) The total computational complexity of the algorithms: Both algorithms are $2n^3$.

(2) Calculation complexity of each node in each iteration round:

Algorithm 1: $N_1 = 2 \times \frac{n}{n_p} \times \frac{n}{(nn_p)} \times n = \frac{2n^3}{n_p^2}$

Algorithm 2: $N_2 = \left(\frac{n}{\sqrt{n_p}}\right)^3 \times 2 = \frac{2n^3}{(n_p\sqrt{n_p})}$

(3) Traffic at each node in each iteration:

Algorithm 1: $NC_1 = \frac{n^2}{n_p}$

Algorithm 2: $NC_2 = \frac{2n^2}{n_p}$

(4) Total number of iterations of the algorithm:

Algorithm 1: $NT_1 = n_p$

Algorithm 2: $NT_2 = \sqrt{n_p}$

(5) Ratio of computation and communication volume per iteration:

Algorithm 1: $NCR_1 = \frac{NC_1}{N_1} = \frac{n_p}{2n}$

Algorithm 2: $NCR_2 = \frac{NC_2}{N_2} = \frac{\sqrt{n_p}}{n}$

Based on the above analysis, we have the following conclusions:

Algorithm 1: $sp(A^1_{(nn,n_p)}) = \frac{r_{E_{(n,1)}}}{r_{E_{(n,n_p)}}} = \frac{n_p^2+1}{n_p+1}$

Algorithm 1:

$$Q(A^1_{(nn,n_p)}) = c_{E_1} \times r_{E_1} \times N_1 = (n_p+1) \times n_p \times \left(\frac{2n^3}{n_p^2}\right) = \frac{2n^3(n_p+1)}{n_p}$$

$$Oh(A^1_{(nn,n_p)}) = Q(A^1_{(nn,n_p)}) - Q(A^1_{(nn,1)}) = (n_p-1) \times N_1 = (n_p-1) \times \left(\frac{2n^3}{n_p^2}\right)$$

$$Ef(A^1_{(nn,n_p)}) = \frac{r_{E(A_{(nn,1)})}}{r_{E(A_{(nn,n_p)})} \times c_{E(A_{(nn,n_p)})}} = \frac{n_p^2+1}{n_p^2+n_p}$$

Algorithm 2:

$$sp(A^2_{nn}) = \frac{r_{E_{(n,1)}}}{r_{E_{(n,n_p)}}} = \frac{\sqrt{n_p}\,n_p+1}{\sqrt{n_p}+1}$$

$$Q(A^2_{(nn,n_p)}) = c_{E_2} \times r_{E_2} \times N_2 = (\sqrt{n_p}+1) \times n_p \times \left(\frac{2n^3}{n_p\sqrt{n_p}}\right) = \frac{2n^3(\sqrt{n_p}+1)}{\sqrt{n_p}}$$

$$Oh(A^2_{(nn,n_p)}) = Q(A^2_{(nn,n_p)}) - Q(A^2_{(nn,1)}) = (n_p-1) \times N_2 = (n_p-1) \times \left(\frac{2n^3}{n_p\sqrt{n_p}}\right)$$

$$Ef(A^2_{(nn,n_p)}) = \frac{r_{E(A_{(nn,1)})}}{r_{E(A_{(nn,n_p)})} \times c_{E(A_{(nn,n_p)})}} = \frac{\sqrt{n_p}\,n_p+1}{\sqrt{n_p}\,n_p+n_p}$$

From the above analysis, we can conclude that Algorithm (1) is better than Algorithm (2) in terms of parallel acceleration and algorithmic efficiency, which can also be seen from the decomposition matrix and execution matrix of Algorithm (1) and Algorithm (2).

However, further incorporating the impact of algorithmic communication, the ratio of the total number of communications between the two algorithms to the total amount is: $\frac{\sqrt{n_p}-1}{n_p-1}$ and $2 \times \frac{\sqrt{n_p}-1}{n_p-1}$ From the above formula, it can be concluded that the communication volume of Algorithm (2) is smaller than Algorithm (1). The memory to space ratio of two algorithms: $r_1 = W_1/M_0 = O(n)$ and is constant. The memory access efficiency of both algorithms is relatively high.

The space to memory ratio of the two algorithms is: $\beta = \frac{2\sqrt{n_p-1}}{n_p-1}$ When the number of computing nodes is infinitely expanded, the value of the space-to-memory ratio tends to 0, so the memory access performance of Algorithm (2) is higher than that of Algorithm (1). Moreover, as the number of computing nodes continues to expand, this advantage becomes increasingly apparent.

The above theoretical calculations will be verified through experiments. The experimental results show that Algorithm (2) outperforms Algorithm (1) in performance, and the advantage is more pronounced when the number of computing nodes is 16 than when the number of computing nodes is 4. From the perspective of practical experiments, it has been verified that Algorithm (2) has higher memory access performance than Algorithm (1).

The experiment comparison diagram is shown in Figures 2 to 6. The experimental results are shown in Table 1.

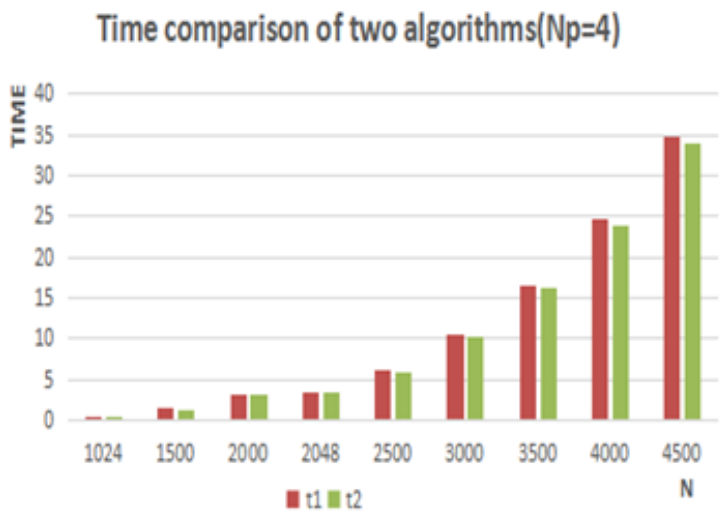

Figure 2. Time comparison of two algorithms ($N_p = 4$).

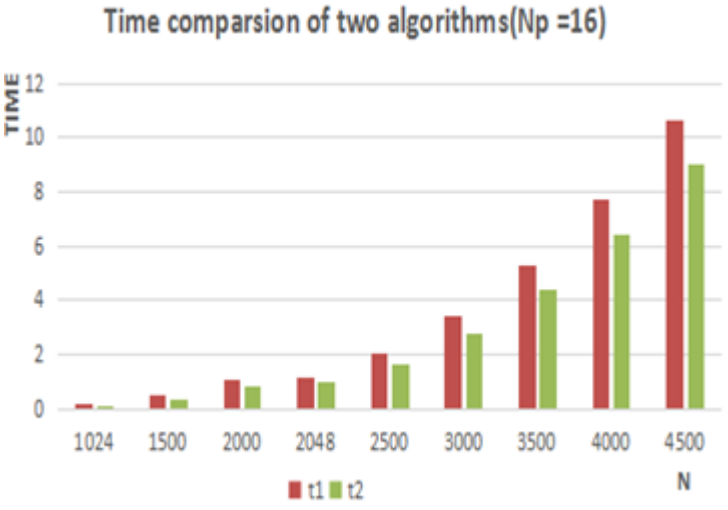

Figure 3. Time comparison of two algorithms ($N_p = 16$).

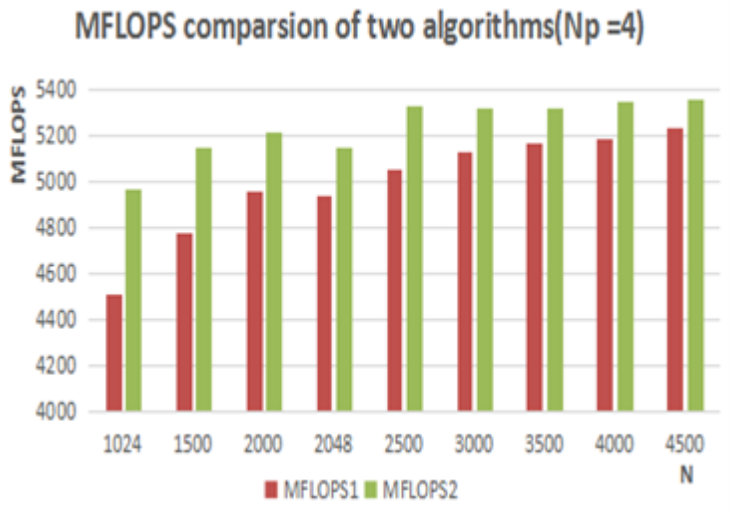

Figure 4. MFLOPS comparison of two algorithms ($N_p = 4$).

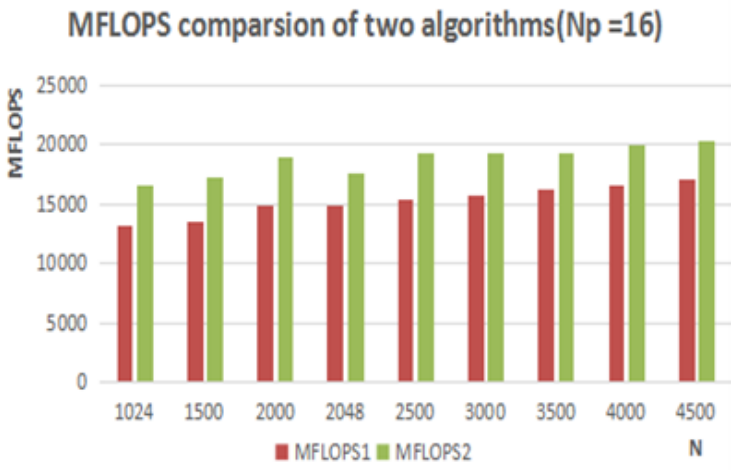

Figure 5. MFLOPS comparison of two algorithms ($N_p = 16$).

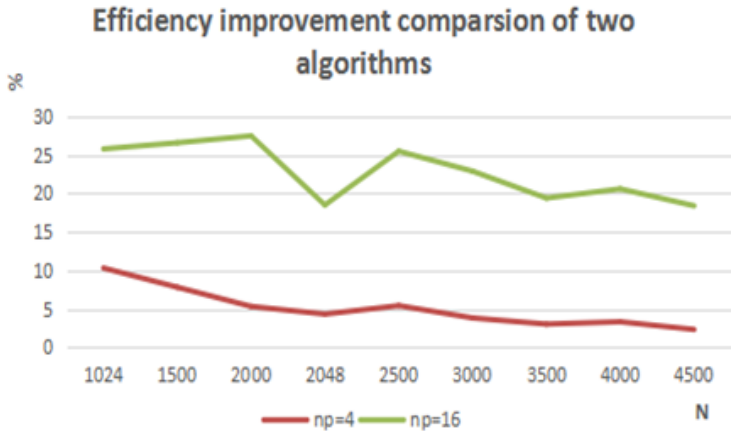

Figure 6. Efficiency improvement comparison of two algorithms.

Table 1. Performance Test Comparison of Two Algorithms.

N_p	N	T_1 (s)	**MFLOPS**$_1$	T_2 (s)	**MFLOPS**$_2$	**Efficiency improvement (%)**
4	1024	0.476	4507.87	0.432	4973.23	10.3
	1500	1.412	4779.00	1.310	5151.03	7.8
	2000	3.228	4955.93	3.065	5219.62	5.3
	2048	3.481	4935.50	3.337	5147.73	4.3
	2500	6.184	5052.96	5.866	5327.16	5.4
	3000	10.520	5128.65	10.143	5324.07	3.8
	3500	16.581	5171.54	16.105	5324.32	3.0
	4000	24.690	5184.24	23.908	5353.94	3.3
	4500	34.799	5237.26	34.028	5355.83	2.3
16	1024	0.164	13111.60	0.130	16491.33	25.8
	1500	0.498	13555.72	0.393	17158.46	26.6
	2000	1.088	14802.26	0.848	18871.37	27.5
	2048	1.158	14830.96	0.977	17580.89	18.5
	2500	2.035	15355.66	1.622	19271.64	25.5
	3000	3.447	15664.58	2.806	19274.87	22.9
	3500	5.301	16175.69	4.438	19320.21	19.4
	4000	7.710	16600.78	6.395	20015.73	20.6
	4500	10.671	17079.21	9.013	20221.43	18.4

5. **Conclusions.** This article proposes a performance optimization model for computationally intensive parallel algorithms based on the characteristics of computationally intensive applications, providing methods and theoretical support for the performance optimization analysis of such parallel algorithms. Finally, this paper selects the parallel algorithm of large matrix multiplication commonly used in mathematical analysis, differential equations, and computer science, conducts detailed optimization analysis and research based on the model, and performs extensive experimental verification, which greatly improves the parallel efficiency of matrix multiplication.

This model has important theoretical guidance for performance optimization analysis in parallel computing and has significant application value for improving the simulation speed and efficiency of computationally intensive applications in large and complex engineering.

REFERENCES

[1] L. Kang, R.-S. Chen, Y.-C. Chen, C.-C. Wang, X. Li, and T.-Y. Wu, "Using Cache Optimization Method to Reduce Network Traffic in Communication Systems Based on Cloud Computing," *IEEE Access*, vol. 7, pp. 124397–124409, 2019.

[2] T.-Y. Wu, A. Shao, and J.-S. Pan, "CTOA: Toward a Chaotic-Based Tumbleweed Optimization Algorithm," *Mathematics*, vol. 11, no. 10, p. 2339, 2023.

[3] T.-Y. Wu, H. Li, and S.-h. Chu, "CPPE: An Improved Phasmatodea Population Evolution Algorithm with Chaotic Maps," *Mathematics*, vol. 11, no. 9, p. 1977, 2023.

[4] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Floating-Point Program and Multicore Architectures," *Office of Scientific & Technical Information Technical Reports*, vol. 52, no. 4, pp. 65–76, 2009.

[5] D. Hackenberg, D. Molka, and W. E. Nagel, "Comparing Cache Architectures and Coherency Protocols on x86-64 Multicore SMP Systems," in *Proc. IEEE/ACM Int. Symp. on Microarchitecture*, pp. 413–422, 2010.

[6] A. Sodani, R. Gramunt, and J. Corbal, "Knights Landing: Second-Generation Intel Xeon Phi Product," *IEEE Micro*, pp. 34–46, 2016.

[7] S. Ramos and T. Hoefler, "Capability Models for Manycore Memory Systems: A Case Study with Xeon Phi KNL," in *Proc. IEEE Int. Parallel and Distributed Processing Symp.*, pp. 297–306, 2017.

[8] G. Bauer, S. Gottlieb, and T. Hoefler, "Performance Modeling and Comparative Analysis of the MILC Lattice QCD Application SU3_RMD," in *Proc. IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing*, pp. 652–659, 2012.

[9] A. Hoisie, O. Lubeck, and H. Wasserman, "Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures Using Multidimensional Wavefront Application," *Sage Publications, Inc.*, pp. 330–346, 2000.

[10] M. M. Mathis, D. J. Kerbyson, and A. Hoisie, "A Performance Model of Non-Deterministic Particle Transport on Large-Scale Systems," in *Proc. Int. Conf. on Computational Science*, pp. 905–915, 2003.

[11] K. L. Spafford and J. S. Vetter, "Aspen: A Domain Specific Language for Performance Modeling," in *Proc. Int. Conf. on High Performance Computing, Networking, Storage and Analysis*, IEEE Computer Society Press, p. 84, 2012.

[12] D. Bailey, "The NAS Parallel Benchmarks," in *Proc. Annu. Conf.*, pp. 158–165, 1991.

[13] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. D. Supinski, and M. Schulz, "A Regression-Based Approach to Scalability Prediction," in *Proc. Int. Conf. on Supercomputing*, pp. 368–377, 2008.

[14] A. Bhattacharyya and T. Hoefler, "Pemogen: Automatic Adaptive Performance Modeling During Program Runtime," in *Proc. 23rd Int. Conf. on Parallel Architectures and Compilation*, ACM, pp. 393–404, 2014.

[15] B. C. Lee, D. M. Brooks, B. R. D. Supinski, M. Schulz, K. Singh, and S. A. McKee, "Methods of Inference and Learning for Performance Modeling of Parallel Applications," in *Proc. ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP)*, San Jose, CA, USA, pp. 249–258, 2007.

[16] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes," in *Proc. High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2013.

[17] P. Balaprakash, A. Tiwari, S. M. Wild, L. Carrington, and P. D. Hovland, "Automomml: Automatic Multi-Objective Modeling with Machine Learning," in *Proc. Int. Conf. on High Performance Computing*, pp. 219–239, 2016.

[18] X. Wu, C. Lively, V. Taylor, H. C. Chang, L. Bo, K. Cameron, T. Dan, and S. Moore, "MuMI: Multiple Metrics Modeling Infrastructure," *Springer International Publishing*, pp. 53–65, 2014.

[19] A. Marathe, R. Anirudh, N. Jain, A. Bhatele, and T. Gamblin, "Performance Modeling Under Resource Constraints Using Deep Transfer Learning," in *Proc. ACM/IEEE Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, USA, 2017.

[20] M. Burtscher, B. D. Kim, J. Diamond, J. M. Calpin, and J. Browne, "PerfExpert: An Easy-to-Use Performance Diagnosis Tool for HPC Applications," in *Proc. High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, 2010.

[21] Y. Alexeev, A. Mahajan, S. Leyffer, G. Fletcher, and D. G. Fedorov, "Heuristic Static Load-Balancing Algorithm Applied to the Fragment Molecular Orbital Method," in *Proc. High Performance Computing, Networking, Storage and Analysis*, pp. 56–62, 2012.

[22] C. K. Luk, S. Hong, and H. Kim, "CCSM: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping," in *Proc. IEEE/ACM Int. Symp. on Microarchitecture (MICRO-42)*, pp. 45–55, 2009.