An Optimization Method for Application Parallel Parameters Integrated into Supercomputing Job Scheduling Systems

Authors: Zhang Wenshuai, Li Huimin, Li Jing, Pan Bicai, Zhang Wenshuai, Li Huimin, Li Jing

Date: 2024-09-20T00:00:00+00:00

Abstract

With the development of high-performance computing architectures, both software and hardware have adopted multi-level parallel structures. When computational tasks from different vertical levels and horizontal groupings are distributed across different processors on various nodes, numerous allocation and mapping possibilities arise. As a result, users are finding it increasingly difficult to determine optimal computational parallel parameters and hardware resource usage. We have investigated an optimization method that can assist users in automatically determining optimal application parallel parameters and hardware usage, thereby enabling efficient scaling to large-scale computing. Furthermore, we propose a scheme for deeply integrating this optimization method with job scheduling systems, which has achieved promising application results.

Full Text

A Parallel Parameter Optimization Method for Supercomputing Applications Integrated with Job Scheduling Systems

Wenshuai Zhang[1], **Huimin Li**[1], **Jing Li**[2], **Bicai Pan**[3] [1]Network Information Center, Supercomputing Center, University of Science and Technology of China, Hefei, Anhui 230026, China [2]School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230026, China [3]Department of Physics, University of Science and Technology of China, Hefei, Anhui 230026, China

Abstract: With the evolution of high-performance computing architectures, both software and hardware have adopted multi-layered parallel structures.

When computational tasks from different vertical layers and horizontal groupings are distributed across processors on different nodes, numerous allocation schemes become possible. These schemes are typically determined at runtime through multiple parallel parameters specified by users, significantly impacting computational efficiency. As computational scale and complexity increase, the configurable space for these parallel parameters expands dramatically, making it increasingly difficult for users to identify optimal values. Such runtime optimization problems are common in scientific computing applications, yet relevant research and solutions remain scarce. In this work, we use the VASP application as a case study to first analyze its multi-layer parallel structure and demonstrate substantial runtime speed variations caused by different parallel parameter configurations. We then propose a fully automated runtime optimization method based on a reduced parallel efficiency metric. This approach not only helps users quickly and easily determine optimal application parallel parameters but also identifies the most efficient computational resource usage, enabling applications to scale efficiently to large-scale parallel computing. Finally, we integrate this optimization method with a cluster job scheduling system and apply it to real VASP calculation jobs submitted by users. Statistical results show that our scheme significantly improves job execution speed and supercomputing resource utilization efficiency, demonstrating strong prospects for practical engineering applications.

Keywords: parallel computing; job scheduling; runtime optimization; supercomputing; VASP

Funding: Chinese Academy of Sciences Pilot Project Class A "Earth Big Data Science Engineering—Cloud Service Infrastructure Platform Operation and Security" (Grant No. XDA19020102). Related methods have been granted patent protection ("A Job Runtime Parameter Optimization Method for Supercomputing Cluster Scheduling," Patent No. ZL202111268933.5).

Authors: Wenshuai Zhang (male, born 1987, Ph.D., CCF member C7539M) focuses on supercomputing platform technologies, high-performance computing, and first-principles calculations; Huimin Li, Ph.D., Senior Engineer; Jing Li, Ph.D., Professor; Bicai Pan, Ph.D., Professor.

Driven by scientific computing demands, computing hardware has evolved from early homogeneous multi-core processors (SMP) to heterogeneous architectures with NUMA partitions, and from single nodes to large-scale clusters [1-3]. Consequently, computing systems have developed increasingly complex hierarchical structures. Correspondingly, many scientific computing programs have progressed from simple OpenMP parallelism to MPI parallelism and heterogeneous acceleration support [4-6]. Both software and hardware have become progressively more complex during their evolution and maturation [7-10].

When dealing with mature applications featuring diverse computational capabilities and algorithms, different computational instances exhibit enormous

variations in runtime characteristics [11-13], requiring algorithmic and runtime parameter adjustments to improve load balancing, optimize data placement, and reduce data movement and latency. Related research and implementations have focused on optimizing configuration parameters for critical computational functions [14,15]. For example, methods based on trial-run analysis or machine learning predictions using historical runtime data have been used to improve the speed of frequently repeated FFTW (Fastest Fourier Transform in the West) [16] function computations [17-18], yielding clear benefits for scientific computing applications in quantum chemistry and other domains that involve extensive Fourier transforms. Some studies have also optimized not only function-level runtime parameters but also system and application-level parameters [19-22]. For instance, based on a runtime system designed for exascale computing (OCR/OCR-Vx) [20,21], Dokulil and Benkner achieved over twofold performance improvements for subsequent similar iterative computations by performing a few iterations and performance analyses on Stencil2d and Seismic applications to optimize the mapping of application data blocks and computational tasks to NUMA nodes [22]. However, this runtime allocation optimization requires applications to be adapted and built on the OCR/OCR-Vx framework and does not address task distribution across multiple nodes.

When computational tasks from multiple vertical layers and horizontal groupings are distributed across different node clusters and processors, the parameter configuration space becomes larger and the runtime environment more complex and variable. Different task allocation approaches can lead to significantly different load balancing levels and parallel efficiency. The optimal runtime configuration depends on numerous factors including computational algorithms, data scale, memory and storage performance, number of computational cores, core performance, core topology, node topology, and communication bandwidth. Consequently, relying on experience or built-in code models to estimate optimal runtime parameters or task allocation schemes becomes increasingly infeasible. A typical challenge is that even if the problem of automatically optimizing parallel task allocation is solved within the program, the optimization of computational resource quantity configuration remains unresolved. This is because the amount of computational resources is generally provided by users or the scheduling system before computation begins and cannot be optimized or adjusted through internal program parameters. Therefore, applications typically allow users to manually adjust computational task partitioning methods and configuration parameters based on computational objectives and hardware resources. These parallel task allocation parameters are generally set in input files or runtime commands before program execution and are referred to as application parallel parameters in subsequent discussions. They affect only the grouping strategy of computational tasks without influencing the computational results.

This paper uses the VASP (Vienna Ab initio Simulation Package) [23] application as an example to explore how to automatically optimize application parallel parameters and computational core usage in supercomputing platforms, as

well as how to integrate runtime optimization schemes with scheduling systems. VASP is one of the most resource-intensive computational software packages globally, making its performance optimization significantly valuable for reducing global energy consumption, improving user computational efficiency, and accelerating scientific research progress. Moreover, VASP' s approximation models and parallel algorithms are widely present in other quantum chemistry software such as QUANTUM ESPRESSO [24]; thus, optimization methods for VASP can generally be applied to similar software.

Existing performance testing articles on VASP [25-28] primarily aim to analyze new hardware performance through a small number of test cases to guide hardware/software design and construction, while also helping users understand principles and experiences for improving runtime efficiency. However, users still need to manually test parallel parameters, and differences in user expertise continue to affect job runtime efficiency, or they must perform time-consuming exhaustive tests across the parameter space to achieve optimal performance. Additionally, previous work has predicted VASP job runtime based on application input parameters and historical runtime data [29], but the training data consisted of unoptimized runs provided by users and did not contain performance difference information for the same job under different runtime parameters, making it difficult to adapt to performance optimization needs.

Currently, research specifically targeting application parallel parameter tuning remains very limited. Wang Yurui et al. proposed a simple automatic optimization algorithm [30] that uses a fixed-variable method and binary search, selecting parallel efficiency based on a fixed number of computational cores as the optimization metric to sequentially optimize multiple parallel parameters. However, this approach still requires a large search space and numerous test runs.

This paper proposes a fully automated runtime optimization method based on a reduced parallel efficiency metric that not only helps users optimize runtime speed but also helps them determine the most efficient computational core usage. This reduced efficiency metric does not use a fixed number of computational cores as a baseline but is instead determined jointly by the ratio of core-hour consumption and speedup ratio from two test runs, balancing both computational speed and resource efficiency. Furthermore, the value space for each application parallel parameter is significantly reduced based on requirements for computational load balancing and local communication, decreasing the number of tests required. Additionally, since quantum calculations in VASP involve numerous repeated iterations, this work uses only a small number of iterative calculations for testing and analysis, further reducing optimization time. This optimization scheme can be conveniently integrated with job scheduling systems, utilizing idle node resources in cluster systems to pre-complete runtime optimization for jobs in the system, enabling jobs to run directly with optimized application parallel parameters after computational resources are allocated. This scheme can also be extended to other applications with multi-layer parallel computational

structures.

The remainder of this paper is organized as follows: Section 1 briefly describes VASP' s application parallel parameters and presents results from exhaustive parameter space testing. Section 2 describes the automated application parallel parameter optimization method. Section 3 presents and analyzes VASP application parallel parameter optimization results. Section 4 describes the integration of runtime optimization with the scheduling system and demonstrates its application effectiveness.

---

1 Application Parallel Parameters and Exhaustive Testing

This section introduces the common multi-layer parallel structure in supercomputing cluster applications, using VASP as an example, and briefly describes the parallel parameters that users can control and configure at runtime. We then present test results from exhaustive parameter space testing of VASP application cases to illustrate the challenges users face when configuring optimal parallel parameters for applications with multi-layer parallel structures.

1.1 VASP Application Parallel Structure

[Figure 1: see original paper] shows the basic pattern of task parallel partitioning in VASP runtime, illustrating the correspondence between tasks and CPU cores under three-level parallelism. Here, KPAR represents the number of simultaneous k-point tasks, NPAR represents the number of simultaneous band tasks, and NCORE represents the number of computational cores used for a single band calculation task. Therefore, the total number of computational cores is the product of these three parameters: KPAR $\times$ NPAR $\times$ NCORE. Since these three parameters determine both the basic parallel task partitioning scheme and the total computational resource requirements, they constitute the parallel parameters we aim to optimize. In some calculations, an additional layer of parallelism for multiple image instances is added above KPAR parallelism, with the parallel count denoted as IMAGES. There is almost no communication or dependency between multiple image instances, making this parallelism highly efficient and treatable as a constant during optimization. In this case, the total number of computational cores becomes KPAR $\times$ NPAR $\times$ NCORE $\times$ IMAGES.

1.2 Exhaustive Test Results

Before introducing the new automatic parallel parameter optimization method, we first briefly present empirical conclusions from our early testing. In document [31], we analyzed five different types of test cases across six hardware platforms. The results showed that some commonly used parameter settings, including recommendations from VASP official documentation, clearly deviate from optimal

values in certain computational scenarios. For example, the suggestion "NPAR = 4 ~ approx. SQRT(number of cores)" is clearly unsuitable for small systems with multiple k-points and large-scale single k-point cases. The recommendation "KPAR > Computer Nodes is not advised" is also inappropriate for small systems with multiple k-points. Similar phenomena have been observed by other VASP users, such as Geun Ho Gu from the University of Delaware [32]: "I have found that this instruction does not always hold up, and, really, this parameter is heavily dependent on the batch server / node configuration. So, it is wise to do your own test to optimize this parameter (and other parameters as well)."

We also discovered that with a fixed total number of computational cores, the optimal value space for NCORE is relatively smaller than that for NPAR, and the optimal configuration should have single-node core counts divisible by NCORE. Combined with Figure 1, this can be understood as ensuring that the smallest parallel computational resource group (NCORE CPU cores) completes assigned computational tasks within the same node, thereby significantly reducing inter-node communication. Tests showed that when computing on a 28-core node, changing NCORE from 8 to 7 reduced MPI communication time from 199 seconds to 131 seconds, substantially improving parallel computational efficiency. While these tests provided clear constraints on NCORE values and narrowed the optimal space, they did not yield a deterministic empirical model for application parallel parameter optimization.

The above empirical findings regarding optimal NCORE values suggest that for specific applications, identifying the smallest parallel task group and restricting its value space according to divisibility rules (node core counts) can substantially reduce the parameter space requiring optimization. Based on this, we conducted an exhaustive test within the reduced parameter space, with results shown in Figure 2 [Figure 2: see original paper].

The test case in Figure 2 involves a system composed of 59 atoms. Due to system symmetry, there are 8 reduced k-points in the Brillouin zone, representing a relatively small computational system. Even so, the exhaustive test produced substantial data; we present only representative results here. Based on the exhaustive test results, we plotted computational times for five parameter configurations or configuration optimizations.

The results show that under default configuration, computational time is longer than with our four selected parameter configuration schemes. The optimal process count for this system is below 100. When we optimized NCORE and KPAR separately based on general experience, runtime speed improved, but optimal computational time remained around 30-80 seconds, with parallel efficiency decreasing significantly beyond 600 processes. When using commonly configured empirical values, computational speed was relatively balanced but still far from optimal parameters. When seeking optimal values across the full space, we observed that for this small system, VASP could efficiently scale to 1,280 cores, with optimal computational time as low as 8 seconds—approximately 1,000× faster than using the same number of cores with default parallel parameter

configuration.

These exhaustive tests demonstrate that parallel scalability efficiency results depend significantly on whether runtime parameter optimization is performed simultaneously. In other words, when comparing parallel scalability efficiency of a specific case across two hardware platforms, each platform' s runtime parameters must first be optimized to their respective optimal values. It is important to emphasize that using exhaustive testing to find optimal parameters is cumbersome work that lacks practicality for complex systems and cannot be promoted for real-world scenarios.

---

2 Automated Runtime Optimization Method

Based on the above analysis, we propose a quantitative metric for searching optimal runtime parameters that can evaluate the relative merits of two different runtime configurations using different amounts of hardware resources. Using this metric, we design a scheme that balances optimization of computational efficiency and speed.

2.1 Reduced Parallel Efficiency Metric

This paper uses the following criteria to determine the relative merits of two parallel parameter configurations (denoted as Case1 and Case2):

1) When runtime costs are equal for two different job parameter configurations, the one with shorter computational time is optimal;
2) When runtime costs differ, judgment is based on the reduced parallel efficiency metric value Er.

Here, runtime cost refers to the fixed asset value of computational resources required, depreciation costs during operation, and other resource consumption costs such as electricity. In this paper, we assume these costs are linearly related to the number of computational cores and that different configurations do not use different hardware models, meaning depreciation and electricity costs remain relatively constant. Therefore, runtime cost is measured by the number of computational cores, also referred to as runtime resource count in subsequent discussions.

The reduced parallel efficiency Er is expressed as:

$$E_r = \frac{\text{SpeedUp}}{\text{Cost}} = \frac{T_1/T_2}{R_2/R_1} = \frac{R_1 \cdot T_1}{R_2 \cdot T_2}$$

where we assume the runtime resource count for Case1 is less than that for Case2; R_1 is the runtime resource count for Case1, R_2 is for Case2; T_1 is the job computational time for Case1, and T_2 is for Case2. The exponent n can generally be

set to any integer greater than 1. When n = 2, the reduced parallel efficiency represents the parallel efficiency corresponding to each doubling of computational resources, where "reduced" means it can equivalently compare parallel efficiency values at different hardware scaling factors (i.e., different R_2/R_1) on the same scaling basis.

In contrast, traditional parallel efficiency metrics Ep struggle to balance speed and resource utilization efficiency when comparing scenarios with different computational resource counts. For example, when comparing two parallel scaling instances: one using 4× CPU cores with 64% parallel efficiency versus another using 2× CPU cores with 64% parallel efficiency, both have the same parallel efficiency value but clearly different hardware resource utilization efficiencies. Therefore, this paper employs reduced parallel efficiency to enable fully automatic search and optimization across multi-dimensional parallel parameter spaces with unequal parallel scaling multiples. The Er formula includes both SpeedUp and Cost, indicating that it considers not only speed differences between configurations but also differences in resource cost usage, jointly determining the operational merits of different parameter configurations as a comprehensive evaluation metric.

2.2 Optimization Steps

This paper proposes the following steps to obtain optimal application parallel parameters, assuming the application has three parallel levels: P1, P2, and P3, with Proc_{max} being the maximum total cores available.

a) Run an extremely short trial calculation, stopping after collecting case characteristic data;
b) Based on trial results, determine candidate value lists P1_{Candidates}, P2_{Candidates}, P3_{Candidates} for parameters P1, P2, P3. Following load balancing requirements, these can be selected as factor lists of the total computational tasks corresponding to each parallel parameter;
c) Based on case scale and limited experience, guess initial values P1_{ini}, P2_{ini}, P3_{ini} that require relatively few computational cores. Generally, initial core count can be set to single-CPU core count, with each parallel parameter using application-preset default values;
d) Starting from initial values, attempt parameter values by increasing each parameter according to its list. Using the aforementioned merit criteria, identify superior parallel parameters and select the attempt parameter with maximum Er as the current best parameter, which becomes the new starting point for subsequent walks. This process forms a Markov chain of attempt parameter points with reduced random probability, where each walk moves parameter points along the path with optimal parallel efficiency. To accelerate search, a high reduced efficiency threshold Er_{fine} can be set, where any attempt parameter exceeding this value is immediately accepted as the new optimal value;
e) Stop parameter optimization and parallel scaling tests when Er efficiency

metrics for all parameters no longer exceed the preset efficiency boundary Er_{min}, or when total core count exceeds the maximum.

The optimization steps can be represented by Algorithm 1. The process is equally applicable to jobs with only two parallel levels (P1, P2) or other numbers of levels. For single-level jobs, the optimization reduces to finding the maximum core count within the predetermined maximum that satisfies the Er_{min} efficiency metric.

For VASP applications, NCORE, NPAR, and KPAR correspond to P1, P2, and P3 in the algorithm. In step a), the output reduced k-point count is the total task count for parallel parameter KPAR, the output NBANDS count is the total task count for parallel parameter NPAR, and the single-node total core count can be used to determine the minimum parallel granularity NCORE parameter value list, which generally consists of factors of the former. When optimizing VASP jobs for Gamma-Only calculation types, the P3 parallel level does not exist, and the above optimization process applies only to the two-dimensional P1-P2 space, with P3 always set to 1.

The search process can be analogized to common steepest-descent search algorithms, where parameter points continuously walk and search along the optimal parallel scaling path, with the final stopping point estimated as the optimal application parallel parameter. Notably, this optimization process does not use a global target value (such as traditional parallel efficiency metrics) but instead uses the Er metric to find the optimal scaling path to achieve optimal runtime status. It is important to explain that simply using a specific parallel efficiency value as a unified optimization target would cause parallel parameters to trend toward smaller computational scales, preventing computations from reaching desired scales and speeds.

---

3 Parallel Parameter Optimization Results

This section demonstrates the optimization effects on parallel parameters and scalability for three test cases. All calculations below are based on VASP version 6.3.3, though other versions yield the same conclusions.

3.1 Case 1 Optimization Results

Table 1 shows the runtime parameter optimization results for the $Ni_{56}O_1H_2$ system. The first row represents user default settings, subsequent rows show the test path before optimization stops, and the final column "Loop+ Time" indicates the total time for multiple predetermined iterative calculations. NC is shorthand for NCORE, as in subsequent tables.

Table 1. Optimization results for $Ni_{56}O_1H_2$ calculation

Loop+	Processes	Parameter KPAR×NPAR×NC	Time (s)	Er (%)
1	80	1$×80×1||||2|160|2×80×10||||3|320|2×8×20||||4|640|4×8×10||||5|1280|4×16×10||||6|1		

This case matches the exhaustive test case from Section 1. Compared to the exhaustive testing, our optimization method uses only 7 tests, each requiring only 5 electronic iteration steps, to obtain optimal parallel parameters and scale computational cores to 1,280. Notably, when scaling from 640 to 1,280 cores, parallel efficiency remains as high as 94%, suggesting VASP's parallel scalability for this case exceeds 1,280 cores. Furthermore, this calculation case involves ionic iteration processes with total electronic iterations generally numbering in the thousands, making 35 electronic iterations a negligible fraction of the total computational task. The testing process does not increase total core-hour consumption—a cost-effective feature shared by subsequent cases.

3.2 Case 2 Optimization Results

Table 2 shows a transition state calculation case where the product of the three parallel parameters does not equal the total number of processes because it generates 5 images calculated simultaneously, creating a 5× difference between the first two values. The first row shows user default settings, with subsequent rows showing the test path before optimization stops.

Table 2. Optimization results for $Ni_{48}C_2O_2$ calculation

Loop+	Time (s)	Parameter KPAR×NPAR×NC	Processes	Er (%)
1		1$×4×10||||2||4×2×5||||3||4×2×10||||4||8×2×10||||5||4×4×10||||6||4×2×20||||7||8×2×2		

Parallel parameter optimization for this case reveals two characteristics: 1) In the final optimized result at 1,600 cores, NPAR and NCORE values are 4 and 10, respectively—identical to user settings at 200 cores. However, at 200 cores, the clearly superior parameters should be 2 and 5, demonstrating that optimal parallel parameters are significantly influenced by hardware resource usage and values optimal at one resource scale can deviate substantially from optimal performance at another scale. 2) Superlinear acceleration is commonly observed during optimization, such as the 143% value in the table, typically because the previous computational configuration being compared against had poor parallel efficiency.

3.3 Case 3 Optimization Results

Table 3 shows a calculation case with many atoms but few k-points (5 reduced k-points). The first row shows user default settings, with subsequent rows showing

the test path before optimization stops. Due to the small number of k-points, the KPAR parameter space is limited, requiring only 5 tests to obtain optimization results. This optimization yields remarkable results, providing nearly 100× acceleration over the user' s original calculation. Since this case also involves ionic iteration with longer total job times, this accelerated configuration offers excellent application benefits.

Table 3. Optimization results for $C_{100}I_{48}Cr_{16}$ calculation

Loop+	Processes	Parameter KPAR×NPAR×NC	Time (s)	Er (%)
1		1$×80×1||||2||1×16×10||||3||1×16×20||||4||5×16×20||||5||1×40×20||||6||1×16×40||||7		

3.4 Summary of Optimization Results

Using our proposed automated runtime optimization method, this section demonstrates parallel parameter optimization results for three typical cases. Multiple superlinear acceleration data points indicate that our method can effectively correct poorly performing parallel parameter configurations. Unlike reference [30], which uses a fixed core count as a baseline for calculating parallel efficiency, our reduced parallel efficiency metric enables comparability between test cases under dynamic baselines, allowing continuous evaluation and balancing of computational efficiency and performance without fixing core counts or other parallel parameters during each new test. Consequently, for the first case, only 7 tests were needed to achieve optimization effects comparable to those obtained through dozens of tests using the local space exhaustive method in the previous section.

Furthermore, the latter two cases demonstrate the method' s good case-agnostic adaptability, finding parallel parameter configurations that balance efficiency and performance without requiring testing and fitting for different case types. This significantly simplifies application difficulty and reduces overfitting risks associated with fitting models, ensuring more stable expected performance. In the next section, we validate the method' s stability through testing on a large number of user jobs.

---

4 Job Scheduling System Integration Method and Application Effects

This section proposes a workflow for integrating the aforementioned parallel parameter optimization method into the job scheduling system, as shown in Figure 3 [Figure 3: see original paper]. Users simply submit jobs and wait for optimization results.

[Figure 3: see original paper] shows the integration flow of the job scheduling system with parallel parameter optimization. In practice, the test jobs used for optimization can be assigned a "preemptible" attribute and run immediately on currently idle cluster nodes. Since optimization test jobs typically have brief runtimes, they have a high probability of completing execution before potential preemption. Therefore, such jobs can operate without affecting other user jobs while reducing cluster node idle time and increasing cluster utilization efficiency.

After users accept successful optimization results, they can choose to automatically adopt the optimized parallel parameters or be notified for manual configuration. Particularly, by increasing the scalability test size limit for the optimization process, users can receive more recommendations for large-scale computing settings. Automatically updating job scales can dynamically adjust small user jobs into large-scale computational jobs, fully realizing the original intent of large-scale supercomputing cluster construction and enabling more high-performance large-scale simulations.

The workflow shown in Figure 3 has been implemented in the supercomputing cluster at the University of Science and Technology of China, using VASP version 5.4.4, cluster nodes with E5-2680 v3 or v4 CPUs, and inter-node interconnects using FDR InfiniBand or 100 Gbps OPA networks. Limited by cluster and actual job scales, this test did not perform large-scale scalability optimization, only allowing optimization tests within 1.5× the user' s original computational resource request. The resulting computational acceleration effects and average core-hour consumption are shown in Figure 4 [Figure 4: see original paper], where computational jobs are normal continuously submitted user jobs grouped in sets of 100. The results show that the scheduling system' s automatic parameter optimization plugin can accelerate VASP jobs by approximately 1.6–3.2× on average while reducing average core-hour consumption by 15–35%. Further analysis indicates that if optimization targets are limited to VASP jobs using multi-k-point calculations and the Blocked-Davidson algorithm (accounting for about half of all jobs), this runtime optimization method achieves significantly better acceleration effects, with average speedup reaching 2.33×.

[Figure 4: see original paper] Results of fully automated parameter optimization

Further testing remains for future research. We hope to eventually enable users to forego queue selection and resource request amounts entirely, allowing the job scheduling system to automatically configure optimal resource requests. This approach could be better implemented when web portals become the mainstream usage mode for supercomputing. Particularly, because the reduced parallel efficiency metric allows users to control the balance between computational speed and resource utilization efficiency by adjusting the metric' s threshold boundary. Setting a higher acceptable boundary for Er provides a "priority on efficiency, secondary on speed" option; setting a lower boundary provides a "priority on speed, secondary on efficiency" option; and setting an intermediate boundary achieves a default balanced option.

5 Conclusion

First, this paper demonstrates that in multi-layered hardware/software computing scenarios, computational speed and parallel scalability depend significantly on the optimization level of application parallel parameters. Particularly, due to hardware diversity and variations in computational data scale and algorithms, performance in multi-dimensional parallel parameter spaces is complex and variable, making it difficult for users and developers to determine optimal points based on experience. To help applications simply and quickly achieve optimal runtime speed and scalability, fully automated parallel parameter space search and performance optimization has become a necessary research problem.

We then propose a runtime optimization method based on a reduced parallel efficiency metric that addresses the difficulty of evaluating parameter points in unequally-spaced discrete multi-dimensional parallel parameter spaces while simultaneously optimizing for both computational speed and hardware utilization efficiency. This method can significantly improve speed for most VASP jobs using equivalent computational resources and can enhance computational scale and ultimate speed while ensuring hardware utilization efficiency.

Finally, we design an integration scheme between this runtime optimization method and job scheduling systems, demonstrating its practicality and optimization effects through deployment in a production system. The proposed optimization method can help numerous users conveniently optimize their job scales, increase the proportion of large-scale computations, accelerate research progress without increasing core-hour costs, and help supercomputing clusters more easily support large-scale simulations rather than being fragmented into numerous small-scale computing devices, thereby better aligning with the value proposition of supercomputing clusters.

The method described herein applies not only to VASP but also to other computational programs with multi-layer parallel structures, communication and memory access constraints, and complex variability, such as Quantum ESPRESSO. Due to workload and space limitations, we have not conducted large-scale testing and summarization. Additionally, based on this automated runtime optimization method, we can easily obtain large quantities of high-quality runtime data, laying a foundation for subsequent machine learning-based runtime optimization methods.

We anticipate that this method will not only support computing providers in delivering faster and more efficient runtime environments but also enable the creation of user-friendly cloud computing interfaces. Such interfaces would allow users to compute at any balance point between "computational speed priority" and "resource utilization efficiency priority" without needing to set parallel parameters or computational resource counts, ultimately enabling computational researchers to focus efficiently on computational problems themselves.

References

[1] MARTORELL X, AYGUADÉ E, NAVARRO N, et al. Thread fork/join techniques for multi-level parallelism exploitation in NUMA multiprocessors[C]//Proceedings of the 13th international conference on Supercomputing. 1999: 294-301. DOI: 10.1145/305138.305206

[2] LU K, WANG Y, GUO Y, et al. MT-3000: a heterogeneous multi-zone processor for HPC[J]. CCF Transactions on High Performance Computing, 2022, 4(2): 150-164. DOI: 10.1007/s42514-022-00095-y

[3] DONGARRA J, STERLING T, SIMON H, et al. High-performance computing: clusters, constellations, MPPs, and future directions[J]. Computing in Science & Engineering, 2005, 7(2): 51-59. DOI: 10.1109/MCSE.2005.34

[4] DIAZ J, MUNOZ-CARO C, NINO A. A survey of parallel programming models and tools in the multi and many-core era[J]. IEEE Transactions on parallel and distributed systems, 2012, 23(8): 1369-1386. DOI: 10.1109/TPDS.2011.308

[5] HACKENBERG D, JUCKELAND G, BRUNST H. Performance analysis of multi-level parallelism: inter-node, intra-node and hardware accelerators[J]. Concurrency and Computation: Practice and Experience, 2012, 24(1): 62-72. DOI: 10.1002/cpe.1725

[6] TU B, ZOU M, ZHAN J, et al. Research on hierarchical parallel computing models for multi-core processor clusters[J]. Chinese Journal of Computers, 2008, 31(11): 1948-1955. DOI:10.3321/j.issn:0254-4164.2008.11.009.

[7] ZHANG Y, YUAN L, CHEN Y, et al. Research on multi-level discontinuous nonlinear scalability phenomena in high-performance computing[J]. Chinese Journal of Computers, 2020, 43(6): 973-989. DOI: 10.11897/SP.J.1016.2020.00973

[8] ZANG D, CAO Z, SUN N. Development of high-performance computing[J]. Science & Technology Review, 2016, 34(14):22-28. DOI:10.3981/j.issn.1000-7857.2016.14.002

[9] JIN Z, LU Z, LI H, et al. The origin of high-performance computing: current status and development considerations for scientific computing applications[J]. Bulletin of Chinese Academy of Sciences, 2019, 34(6): 625-639. DOI:10.16418/j.issn.1000-3045.2019.06.004.

[10] WANG T. High-performance computing in quantum chemistry[J]. Journal of East China Normal University (Natural Science), 2018, (4): 109-119. DOI: 10.3969/j.issn.1000-5641.2018.04.011

[11] GARCIA-GASULLA M, BANCHELLI F, PEIRO K, et al. A generic performance analysis technique applied to different CFD methods for HPC[J]. International Journal of Computational Fluid Dynamics, 2020, 34(7-8): 508-528.

[12] GAVIN J P. Optimisation of VASP Density Functional Theory (DFT)/Hartree-Fock (HF) hybrid functional code using mixed-mode parallelism[EB/OL]. (2012-10-31)[2023-11-20]. http://www.hector.ac.uk/cse/distributedcse/reports/vasp02/vasp02.

[13] RICHARD Catlow, FRS, SCOTT Woodley, NORA De Leeuw, ANDREW Turner. Optimising the performance of the VASP 5.2.2 HECToR[EB/OL]. (2010-11-9) [2023-11-20].http://www.hector.ac.uk/cse/distributedcse/reports/vasp01/vasp01_{collectives}.pdf

[14] RASCH A, SCHULZE R, STEUWER M, et al. Efficient auto-tuning of parallel programs with interdependent tuning parameters via auto-tuning framework (ATF)[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2021, 18(1): 1-26. DOI: 10.1145/3427093

[15] MENON H, BHATELE A, GAMBLIN T. Auto-tuning parameter choices in hpc applications using bayesian optimization[C]//2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2020: 831-840.

[16] FRIGO M, JOHNSON S G. Fftw: An adaptive software architecture for the fft[C]//Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP' 98 (Cat. No. 98CH36181): volume 3. IEEE, 1998: 1381-1384.

[17] GU L, LI X. Dft performance prediction in fftw[C]//Languages and Compilers for Parallel Computing: 22nd International Workshop, LCPC 2009, Newark, DE, USA, October 8-10, 2009, Revised Selected Papers 22. Springer, 2010: 140-156.

[18] GUARRASI M, ERBACCI G, EMERSON A. Auto tuning of the fftw library for massively parallel supercomputers[J/OL]. PRACE: Partnership Advanced Computing Europe, 2014, 1: 1-12.

[19] BECKINGSALE D, PEARCE O, LAGUNA I, et al. Apollo: Reusable models for fast, dynamic tuning of input-dependent code[C]//2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2017: 307-316.

[20] MATTSON T G, CLEDAT R, CAVÉ V, et al. The Open Community Runtime: A runtime system for extreme scale computing[C]//2016 IEEE High Performance Extreme Computing Conference (HPEC). IEEE, 2016: 1-7.

[21] DOKULIL J, SANDRIESER M, AND BENKNER S. OCR-Vx-an alternative implementation of the Open Community Runtime[C]//International Workshop on Runtime Systems for Extreme Scale Programming Models and Architectures in conjunction with SC15, Austin, Texas, USA, 2015.

[22] DOKULIL J, BENKNER S. Automatic placement of tasks to NUMA nodes in iterative applications[C]//2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). IEEE, 2020: 192-195.

[23] KRESSE G, FURTHMÜLLER J. Efficiency of ab initio total energy calculations for metals and semiconductors using a plane wave basis set[J]. Computational materials science, 1996, 6(1): 15-50.

[24] Giannozzi P, Baroni S, Bonini N, et al. QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials[J]. Journal of physics: Condensed matter, 2009, 21(39): 395502.

[25] STEGAILOV V, SMIRNOV G, VECHER V. Vasp hits the memory wall: Processors efficiency comparison[J]. Concurrency and Computation: Practice and Experience, 2019, 31(19): 5136.

[26] STEGAILOV V, VECHER V. Efficiency analysis of intel and amd x86_{64} architectures for ab initio calculations: a case study of vasp[C]//Supercomputing: Third Russian Supercomputing Days, RuSCDays 2017, Moscow, Russia, September 25–26, 2017, Revised Selected Papers 3. Springer, 2017: 430-441.

[27] ZHAO Z, MARSMAN M, WENDE F, et al. Performance of hybrid MPI/OpenMP VASP on Cray XC40 based on Intel Knights landing many integrated core architecture[J]. CUG Proceedings, 2017.

[28] MCINTOSH-SMITH S, PRICE J, DEAKIN T, et al. A performance analysis of the first generation of HPC-optimized Arm processors[J]. Concurrency and Computation: Practice and Experience, 2019, 31(16): e5110. https://doi.org/10.1002/cpe.5110

[29] WU G B, SHEN Y, ZHANG W S, et al. Runtime prediction of jobs for backfilling optimization[J]. Journal of Chinese Computer Systems, 2019, 40(1): 6-12.

[30] WANG Y, DU Z, JIANG J, et al. Modeling the Parallel Efficiency of Density Functional Theory Based Jobs on Sunway TaihuLight[C]//2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC). IEEE, 2019: 199-204. DOI: 10.1109/CSE/EUC.2019.00046

[31] ZHANG W S. Previous Benchmark for VASP[EB/OL]. [2023-06-30]. https://scc.ustc.edu.cn/_{upload}/article/files/cd/05/bfefd0ec4fc08d2843fded77bca5/3118fa63-8f2d-4fd5-8582-161dfe1342e8.pdf.

[32] GEUN Ho Gu. It is wise to do your own test to optimize this parameter[EB/OL]. [2023-06-30]. https://www.researchgate.net/post/NCORE-and-NPAR-in-VASP.

Note: Figure translations are in progress. See original paper for figures.