Bulletin of Natural Sciences Research
Vol. 16, No. 1, 2026.

DOI: https://doi.org/10.5937/bnsr16-64069
Original Scientific Paper

PERFORMANCE, ACCURACY, AND PREDICTIVE CHUNK OPTIMIZATION OF PARALLEL SIMPSON AND TRAPEZOIDAL INTEGRATION METHODS

SANJA ĐUROVIĆ[1*], MARKO SMILIĆ[1], ČASLAV STEFANOVIĆ[1], STEFAN PANIĆ[1], MILAN SAVIĆ[1]

[1]Faculty of Sciences and Mathematics, University of Priština in Kosovska Mitrovica, Kosovska Mitrovica, Serbia

ABSTRACT

Parallel numerical integration is a fundamental component of many scientific and engineering applications, yet its performance depends strongly on workload partitioning strategies. In shared-memory environments, the choice of chunk size directly affects load balancing, scheduling overhead, and cache utilization, particularly when adaptive numerical methods are employed. This paper presents an analysis of the performance, accuracy, and scalability of four numerical integration techniques: Simpson's rule, Adaptive Simpson's rule, the Trapezoidal rule, and the Adaptive Trapezoidal rule, implemented in a multi-threaded execution model. Execution-time distributions are analyzed across a wide range of chunk sizes and thread counts, revealing substantial performance variability and method-dependent optimal chunk configurations. In addition to performance evaluation, numerical accuracy is assessed to highlight trade-offs between computational efficiency and error reduction. Based on empirical observations, optimal chunk ranges are identified and used to construct a predictive model that estimates near-optimal chunk configurations as a function of the number of threads. The proposed approach enables efficient parameter selection without exhaustive tuning and demonstrates the feasibility of predictive chunk optimization for parallel numerical integration.

Keywords: Parallel numerical integration, Simpson's rule, Trapezoidal rule, Adaptive quadrature, Chunk optimization.

INTRODUCTION

Numerical integration plays a critical role in scientific computing, simulation, optimization, and data-driven modeling. Many real-world problems require repeated evaluation of definite integrals, often under strict performance constraints (Lloyd et al., 2020; Shahvandi, 2021). With the widespread availability of multi-core processors, parallelization has become a natural approach to accelerate numerical integration algorithms. However, achieving efficient parallel performance is not trivial and depends on several implementation-specific factors beyond the mathematical formulation of the integration method (Quintero-Monsebaiz et al., 2021; Tonarelli et al., 2025).

One of the most influential factors in shared-memory parallelism is workload partitioning. In loop-based parallel numerical integration, the integration interval is typically divided into subintervals, with each thread assigned a subset (Chapman et al., 2007).

The granularity of this division, commonly expressed as chunk size, determines how much work each thread receives at a time. An inappropriate chunk size can lead to poor load balancing, excessive scheduling overhead, cache inefficiencies, and ultimately degraded performance. This effect is even more pronounced when adaptive integration methods are used, as they introduce irregular, data-dependent workloads (Estébanez et al., 2014).

Despite extensive research on numerical integration and parallel computing, many studies focus primarily on algorithmic accuracy or theoretical complexity, treating chunk size as a secondary or fixed parameter. As Estébanez et al. (2014) and Laberge et al. (2019) note, suboptimal chunk configurations can significantly degrade performance, yet systematic exploration of chunk granularity is rare in numerical integration studies. Smilić et al. (2025) analyzed the performance of sequential and parallel implementations of Simpson's rule, focusing on execution time, speedup, and efficiency, while holding scheduling parameters fixed throughout the experiments. In practice, default scheduling parameters are often used, even though they may be far from optimal for a given method, problem size, or hardware configuration. Abdul Khalib et al. (2013) investigated the impact of Open Multi-Processing (OpenMP) scheduling clauses and chunk sizes on speedup in multicore environments, showing that performance can vary significantly with these parameters and that default configurations are commonly assumed. Silva et al. (2025) demonstrated that auto-tuning the chunk size in OpenMP dynamic scheduling yields significantly better performance than default configurations across different problem sizes and hardware platforms, while Korndörfer et al. (2022) showed that automated selection of scheduling

*Corresponding author: sanja.djurovic@pr.ac.rs

algorithms and chunk parameters consistently outperforms default OpenMP settings. Furthermore, adaptive methods such as Adaptive Simpson's and Adaptive Trapezoidal rules, described by Burden and Faires (2016) and Davis and Rabinowitz (2007), exhibit non-uniform execution patterns that make performance tuning particularly challenging.

This work addresses these challenges by systematically analyzing the performance of four widely used numerical integration methods across varying chunk sizes and thread counts. In addition to execution time, numerical accuracy is evaluated to provide a balanced view of performance–accuracy trade-offs. Building on empirical observations, we propose a predictive approach to chunk selection that estimates optimal chunk ranges based solely on the number of threads. This enables automated, portable performance optimization without exhaustive benchmarking.

The main contribution of this paper is a systematic investigation of chunk-size sensitivity in parallel numerical integration across adaptive and non-adaptive methods in a shared-memory environment. Unlike many previous studies that treat scheduling parameters as fixed or rely on exhaustive empirical tuning, this work introduces the concept of chunk order as a robust abstraction that reduces performance variability and enables stable cross-configuration comparison.

The remainder of the paper is organized as follows: Section 2 reviews numerical methods; Section 3 presents the proposed optimization and prediction approach; Section 4 discusses the experimental results; and Section 5 concludes the paper and outlines directions for future research.

NUMERICAL METHODS

This section briefly describes the numerical integration methods considered in this study. The focus is on their computational characteristics and suitability for parallel execution rather than on detailed mathematical derivations.

Simpson's Rule

Simpson's rule is a classical numerical integration technique that approximates the integrand using second-order polynomials over pairs of subintervals. The integration interval is divided into an even number of equally spaced subintervals, and the integral is computed as a weighted sum of function evaluations at the endpoints and midpoints (Smilić et al., 2025).

From a parallelization perspective, Simpson's rule is well-suited to data-parallel execution because of its uniform structure and predictable workload. Each subinterval pair can be processed independently, making static or dynamic scheduling straightforward (Smilić et al., 2025). However, optimal performance still depends on selecting an appropriate chunk size that balances overhead and concurrency.

Adaptive Simpson's Rule

Adaptive Simpson's rule improves on the classical method by recursively subdividing the integration interval based on local error estimates. Regions with higher curvature in the integrand receive finer subdivision, while smoother areas are evaluated with fewer subintervals (Burden & Faires, 2016).

Although this adaptivity significantly improves numerical accuracy for complex integrands, it creates irregular workloads that complicate parallel execution. Subintervals may require vastly different amounts of computation, leading to load imbalance if scheduling is not careful. As a result, chunk size selection plays a critical role in mitigating performance degradation caused by dynamic task generation (Dinan et al., 2007).

Trapezoidal Rule

The Trapezoidal rule approximates the integrand using linear interpolation on each subinterval. It is one of the simplest numerical integration methods and requires minimal computational overhead (Burden & Faires, 2016).

Its simplicity makes it highly scalable in parallel environments, as each subinterval can be processed independently with minimal synchronization. However, the Trapezoidal rule generally offers lower accuracy than Simpson-based methods, particularly for functions with high curvature (Burden & Faires, 2016). As with Simpson's rule, the chunk size affects performance by influencing scheduling efficiency and memory access patterns.

Adaptive Trapezoidal Rule

The Adaptive Trapezoidal rule extends the basic Trapezoidal method by recursively refining subintervals where the estimated error exceeds a predefined threshold. This approach achieves higher accuracy while maintaining a relatively simple computational structure (Liu et al., 2017).

Compared with the Adaptive Simpson's rule, the Adaptive Trapezoidal method typically has lower per-task computational cost but still exhibits workload irregularity (Davis & Rabinowitz, 2007). Consequently, its performance is sensitive to chunk size and scheduling strategy, making it a suitable candidate for detailed performance analysis (Estébanez et al., 2014).

OPTIMIZATION AND PREDICTION

Adaptive Chunking Based on Function Oscillation

In parallel numerical integration, uniform workload partitioning can lead to suboptimal performance when the integrand varies non-uniformly across the integration domain. Regions with high curvature or rapid oscillations typically require more computational effort, particularly for adaptive

integration methods. To address this issue, an adaptive chunking strategy adjusts the granularity of the workload based on local function behavior.

Let $f(x)$ denote the integrand over the interval $[a,b]$. The interval is initially divided into N coarse subintervals of equal length. Local variation estimates based on differences in function values are commonly used in adaptive numerical integration to identify regions of increased variability and potential oscillatory behavior (Davis & Rabinowitz, 2007).

For each subinterval $[x_i, x_{i+1}]$, a local oscillation metric is defined as follows:

$$\Delta_i = \left| f(x_{i+1}) - f(x_i) \right|, \quad x_i = a + i\frac{b-a}{N}. \tag{1}$$

Based on this estimate, an adaptive chunk weight is assigned as follows:

$$c_i = 1 + \alpha \cdot \Delta_i. \tag{2}$$

where α is a scaling factor that controls sensitivity to local oscillations. This formulation provides finer granularity for regions with higher oscillatory behavior while maintaining low runtime overhead (Plaškota, 2015).

Chunk Size Exploration and Measurement Protocol

Chunk sizes are examined as an exponential function of the number of threads (Barker et al., 2012). For a given thread count T, the total number of chunks C is defined as:

$$C \in \{T, 2T, 4T, \ldots, M \cdot T\}. \tag{3}$$

where T is the number of threads and M is a predefined upper multiplier. This logarithmic exploration enables efficient coverage of coarse and fine-grained scheduling regimes (da Silva et al., 2025).

For each configuration (T,C), execution time is measured over K trials and averaged as:

$$\bar{t}(T,C) = \frac{1}{K}\sum_{k=1}^{K} t_k. \tag{4}$$

where t_k denotes the execution time of the k-th trial (Bielecki & Śmiałek, 2023).

Identification of Optimal Chunk Order

Instead of selecting a single optimal chunk size, chunk configurations are grouped by their order of magnitude. The chunk order o is defined as:

$$o = \lfloor \log_{10}(C) \rfloor. \tag{5}$$

For each thread count T, the chunk order that minimizes average execution time is considered optimal, consistent with defining the best chunk size as the one that maximizes measured performance (Laberge et al., 2019). This abstraction reduces sensitivity to performance noise and highlights stable efficiency regions.

Predictive Chunk Optimization Model

Based on empirical mappings between thread count and optimal chunk order, a predictive model is developed. Let the dataset be defined as:

$$D = \{(T_i, o_i) \mid i = 1, \ldots, n\} \tag{6}$$

where T_i is the number of threads and o_i is the corresponding optimal chunk order.

For a new thread count T, prediction is performed using a k-nearest neighbors (k-NN) approach (Bishop, 2006). The distance metric is defined as:

$$d(T, T_i) = |T - T_i| \tag{7}$$

The predicted chunk order is obtained via majority voting:

$$\hat{o}(T) = \arg\max_{o} \sum_{(T_i, o_i) \in S_k(T)} I(o_i = o). \tag{8}$$

where $S_k(T)$ denotes the set of k nearest neighbors and $I(\cdot)$ is the indicator function (Hastie et al., 2009).

Monotonicity Constraint

To ensure physically meaningful predictions, a monotonicity constraint is applied:

$$\hat{o}(T) \geq \max_{T_i < T} o_i. \tag{9}$$

This constraint enforces non-decreasing chunk-order predictions as thread counts increase, reflecting the intuition that higher parallelism benefits from equal or coarser workload granularity (Tefera et al., 2024).

Model Validation

The predictive model is validated using Leave-One-out Cross-Validation (LOOCV). For each data point (T_j, o_j), a prediction is made using all remaining samples. Prediction accuracy is computed as:

$$Accuaracy = \frac{1}{n}\sum_{j=1}^{n} I\left(\hat{o}(T_j) = o_j\right) \tag{10}$$

where n is the number of evaluated thread configurations (Hastie et al., 2009).

RESULTS AND DISCUSSION

This section presents and discusses experimental results for all four integration methods. The analysis focuses on

execution-time distributions, scalability with respect to thread count, numerical accuracy, and the effectiveness of the predictive chunk-optimization model. Performance variability across chunk sizes is examined in detail, highlighting each method's sensitivity to workload partitioning. Trade-offs between execution time and numerical error are also discussed to provide practical guidance for method selection. Finally, the predicted chunk ranges are compared with empirically observed optimal configurations to validate the proposed predictive approach. The experiments were conducted on the hardware platform summarized in Table 1.

The evaluation uses an oscillatory Fresnel-type integrand of the form $F(x) = \frac{2}{\sqrt{2\pi}} \int_0^{10} \cos x^2 dx$, whose oscillation frequency increases with x. This provides a representative workload for comparing adaptive and non-adaptive integration methods.

Figure 1 shows the variability in execution time across different chunk sizes with a fixed number of threads. For the Simpson and Trapezoidal methods, the performance distribution remains narrow, indicating stable execution times and limited sensitivity to workload partitioning, owing to their uniform computational structure. In contrast, Adaptive Simpson and Adaptive Trapezoidal show significantly higher performance variability, especially as the number of threads increases. This behavior stems from irregular, data-dependent workloads, where suboptimal chunk sizes can cause load imbalance and longer execution times.

Table 1. Processor specifications.

Model processor	Frequency	Turbo Clock	Cores	Threads
AMD Ryzen 7 5700U	1.8GHZ	4.3GHz	8	16

Although overall execution time decreases with higher parallelism, sensitivity to chunk size becomes more pronounced, underscoring the importance of appropriate workload granularity for adaptive integration methods. This observation motivates a more detailed investigation of chunk-size effects and justifies the focus on chunk-order-based analysis in the remainder of this section.

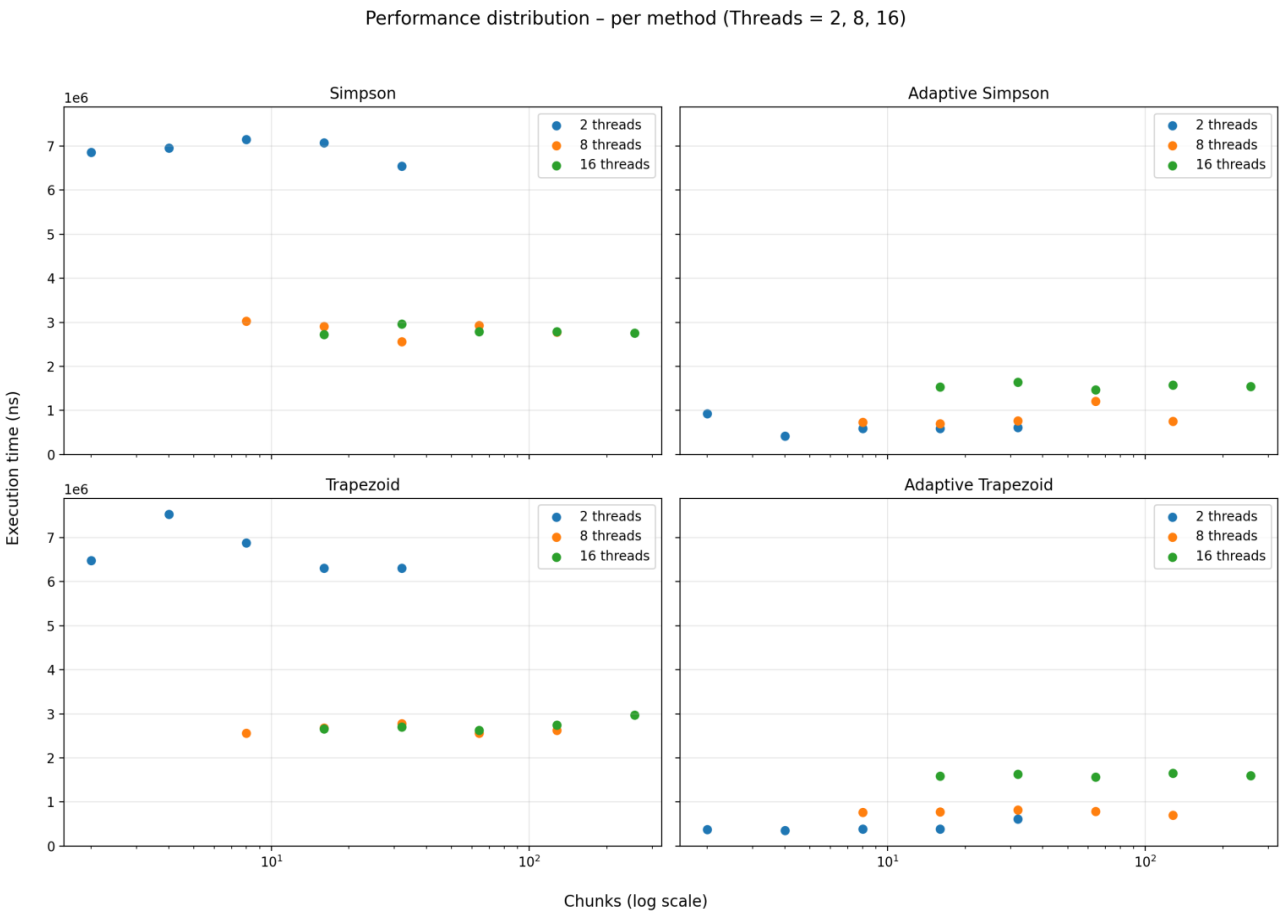

Figure 1. Performance distribution – per method (Threads = 2, 8, 16).

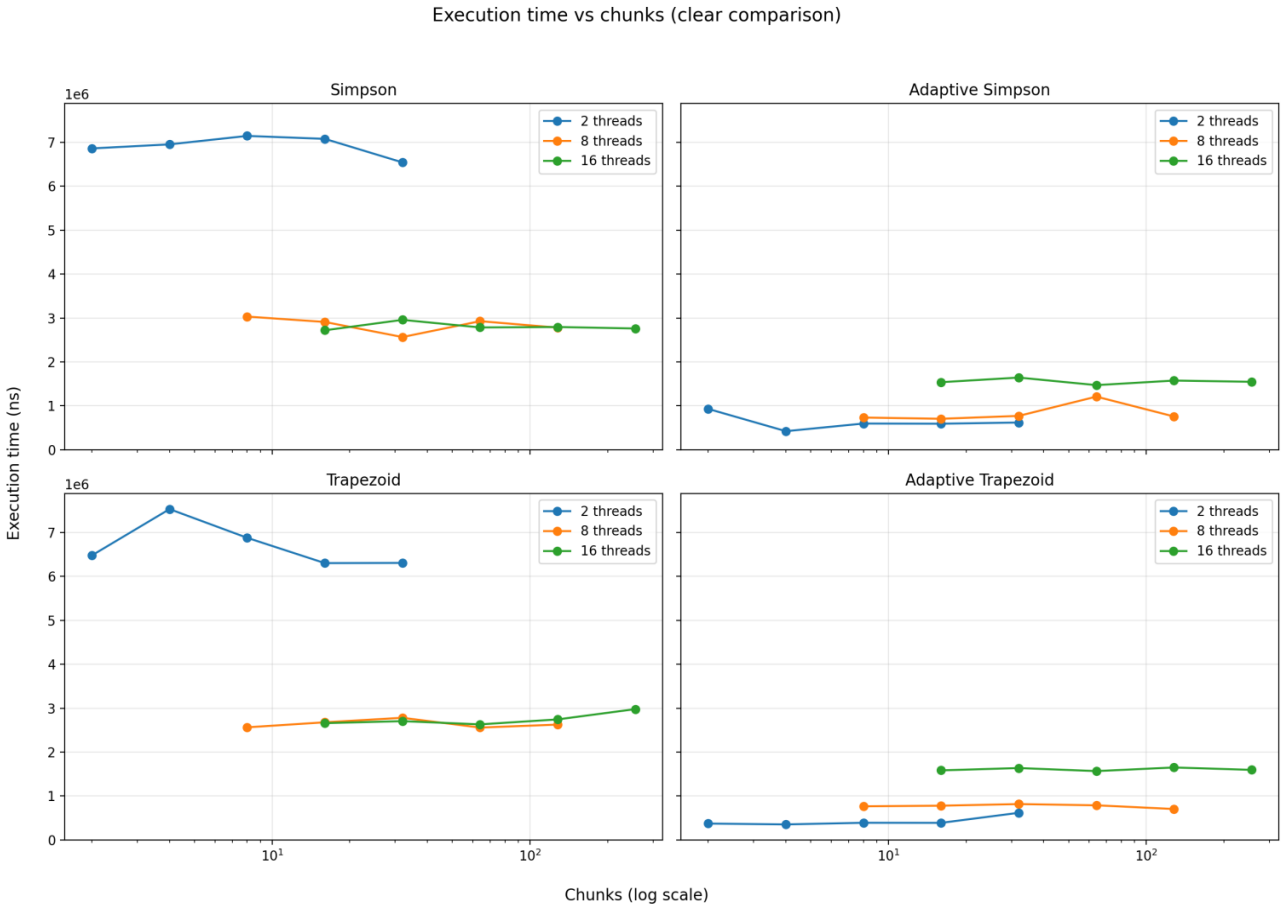

Figure 2. Execution time vs chunks.

Figure 2 shows the relationship between execution time and chunk size across all four integration methods at different thread counts. For non-adaptive methods, execution time varies smoothly with chunk size, with performance minima typically occurring at intermediate chunk sizes. Very small chunks introduce noticeable scheduling overhead, while excessively large chunks reduce parallel efficiency due to limited load balancing. In contrast, adaptive methods show more pronounced, irregular sensitivity to chunk size.

Execution time varies moderately across the explored chunk-size range, particularly at higher thread counts. This behavior reflects the non-uniform, data-dependent workload inherent in adaptive integration, where inappropriate chunk granularity can amplify load imbalance and idle time among threads. Notably, for adaptive methods, the location of execution-time minima shifts substantially with increasing thread count, whereas for non-adaptive methods it remains comparatively stable.

Overall, the results show that the optimal chunk size is both method- and concurrency-dependent, and that adaptive methods require substantially more careful tuning to reach near-optimal performance. Given the pronounced variability and noise sensitivity in execution-time measurements across individual chunk sizes, selecting a single optimal chunk size is unreliable. To address this, the following analysis focuses on the optimal chunk order, enabling a more robust comparison of performance trends as a function of the number of threads.

Figure 3 shows how the optimal chunk order evolves with increasing thread count across all considered integration methods. Across methods, the optimal chunk order generally increases with parallelism, though the trend is not strictly monotonic. Non-adaptive methods show relatively moderate fluctuations, whereas adaptive methods show larger variations and higher optimal orders, reflecting the need for coarser workload granularity to mitigate load imbalance at higher thread counts. Adaptive methods consistently exhibit higher optimal chunk orders than non-adaptive ones, reflecting the need for coarser-grained scheduling to amortize load imbalance and task management overhead.

Figure 4 shows the range of chunk sizes that yield the fastest 10% of configurations across different thread counts and methods. As the number of threads increases, this range expands across all methods. This indicates that near-optimal performance can be achieved over a relatively wide range of chunk sizes, particularly for non-adaptive methods.

Adaptive methods show greater variation in the width of the TOP-10% range than non-adaptive methods. These results indicate that the set of chunk sizes associated with similar performance levels depends on both the integration method and the degree of parallelism.

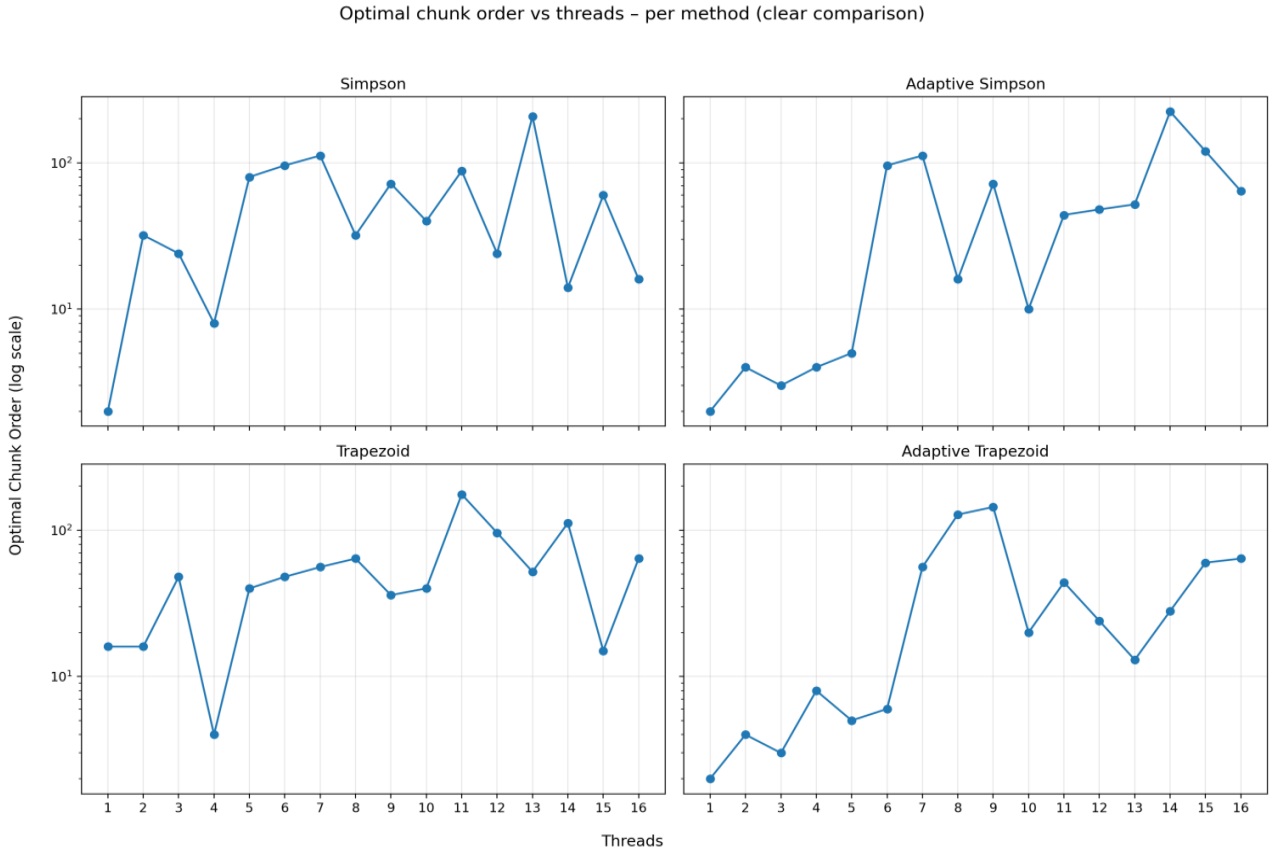

Figure 3. Optimal chunk order vs threads.

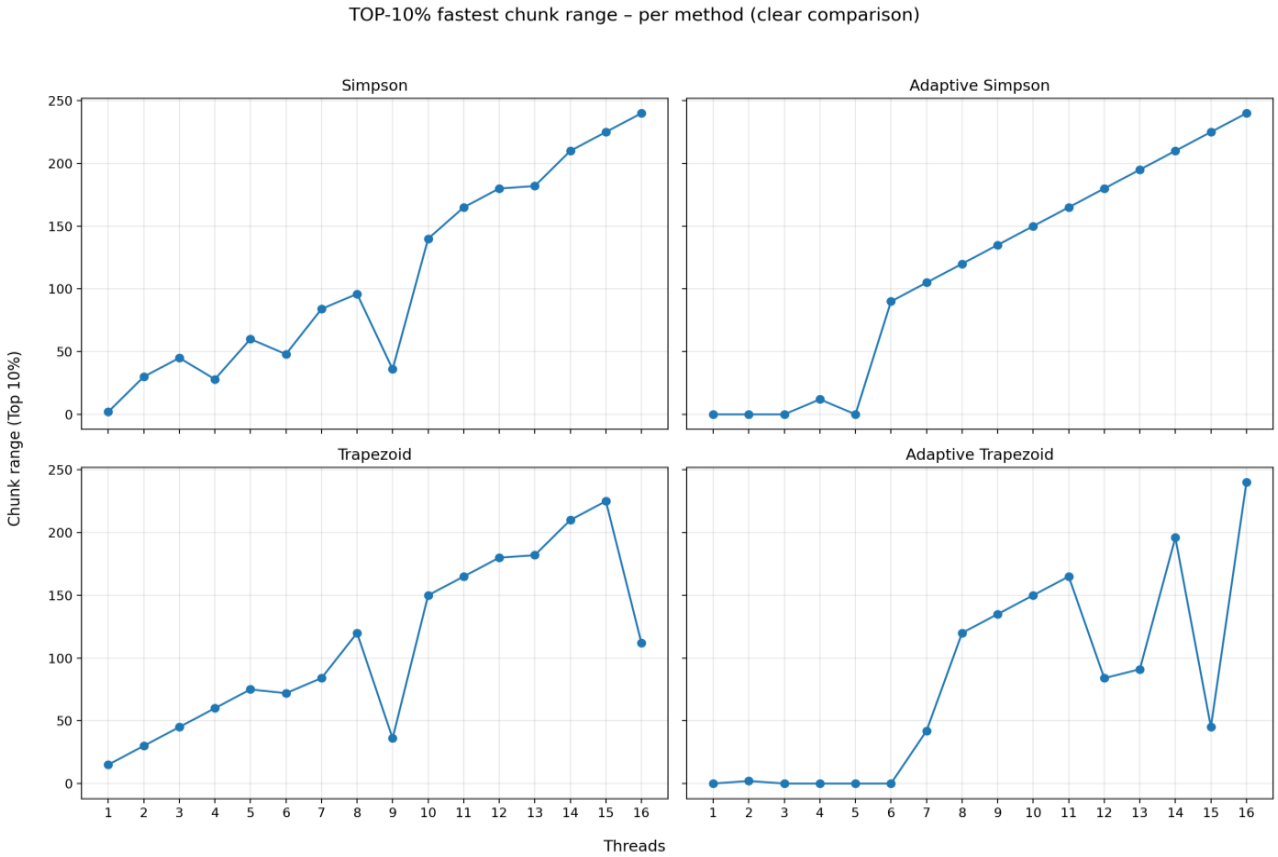

Figure 4. Top 10% fastest chunk range vs threads.

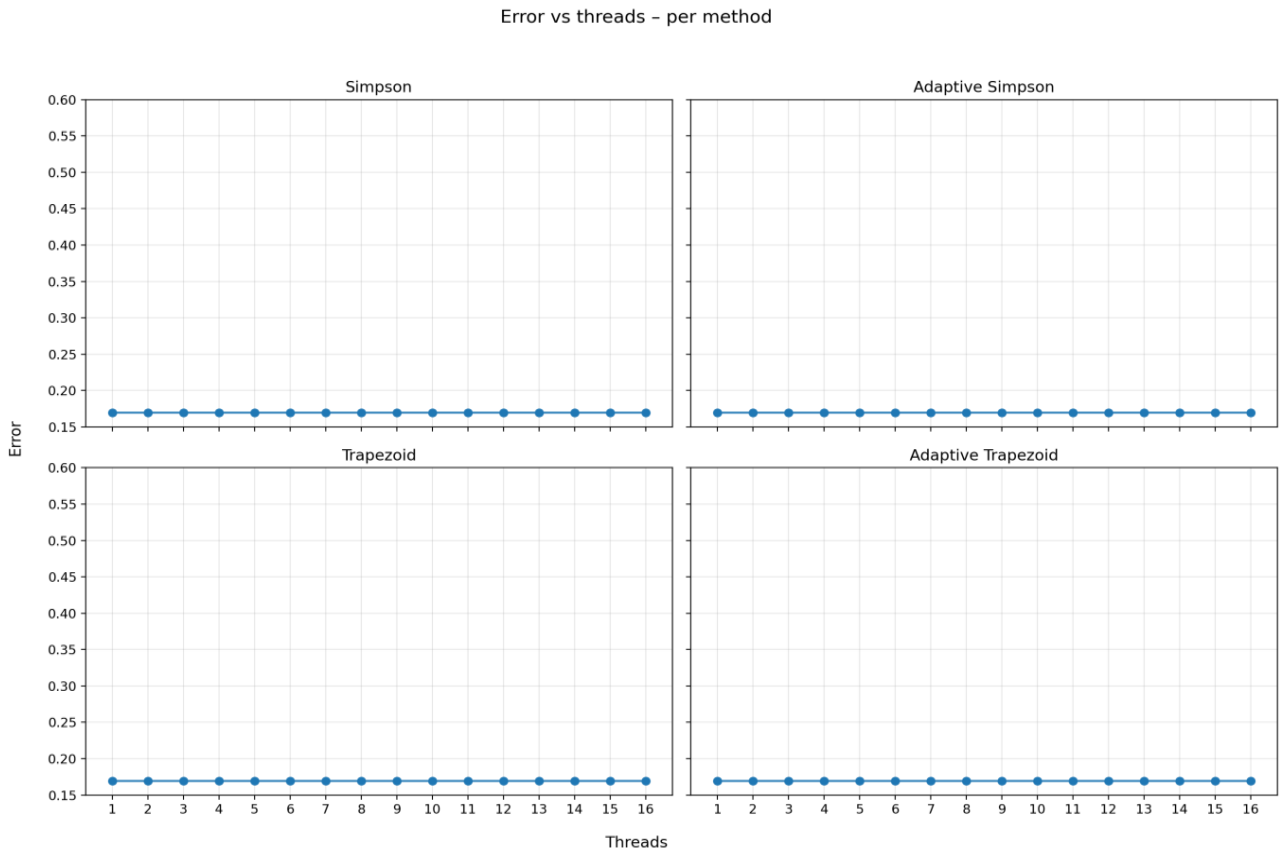

Figure 5. Error vs threads.

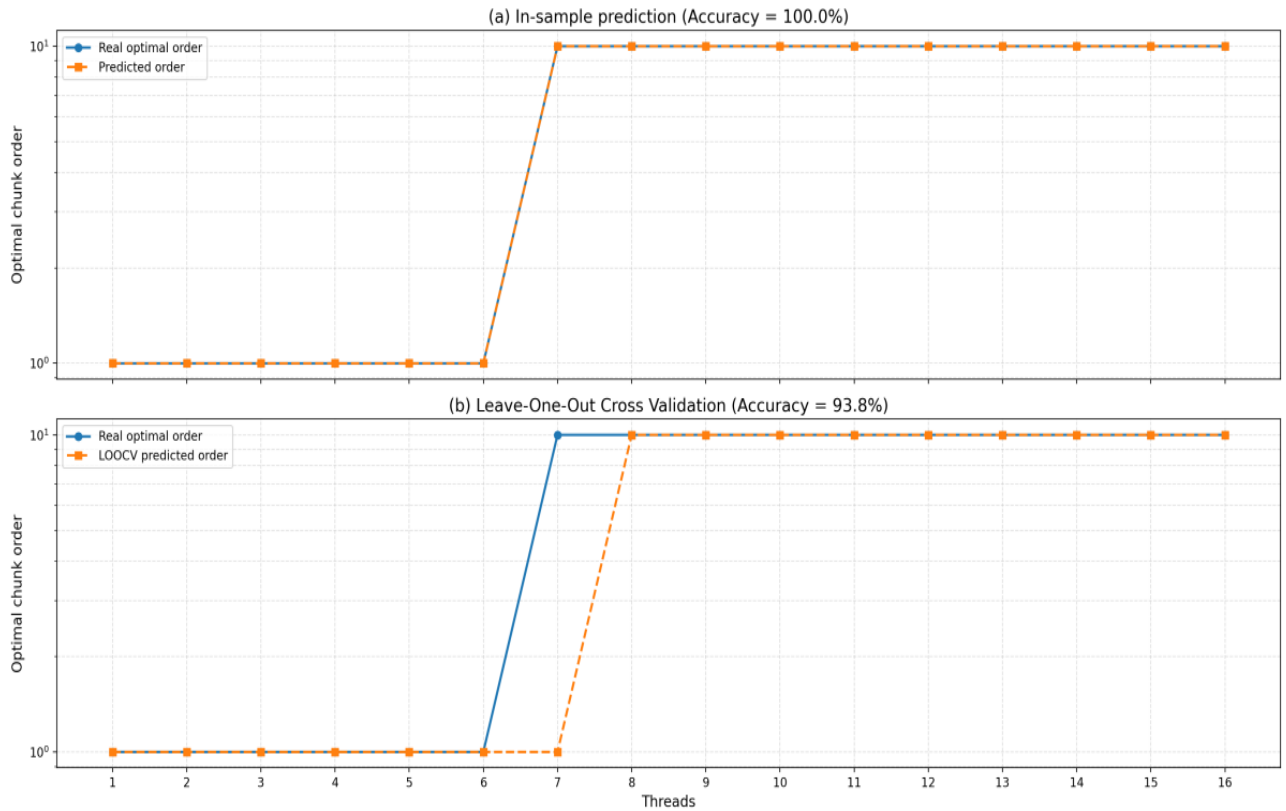

Figure 6. Optimal order prediction (in-sample and LOOCV).

Figure 5 shows the numerical error as a function of the number of threads for all four integration methods. Across the entire range of thread counts, the error remains constant for each method, indicating no dependence on the degree of parallelism. This behavior is consistent across both adaptive and non-adaptive methods, with identical error values across all thread configurations. The results confirm that varying the number of threads does not affect numerical accuracy for the tested implementations.

Figure 6 shows the performance of the proposed predictive model for estimating the optimal chunk order as a function of the number of threads. In the in-sample evaluation, the predicted chunk order exactly matches the empirically observed optimal order across all thread counts, yielding 100% accuracy.

To assess the model's robustness and generalization, LOOCV is used. Under this scheme, the model achieves 93.8% accuracy, with only a single mismatch near the transition between chunk orders. Even in this case, the predicted value differs by only one order of magnitude, which has a negligible impact on execution time given the previously observed TOP-10% performance ranges. From a practical standpoint, this level of accuracy is sufficient to eliminate the need for exhaustive tuning while preserving near-optimal performance. These results demonstrate that the proposed model reliably captures the relationship between thread count and optimal chunk granularity. By predicting the chunk order rather than an exact chunk size, the model provides a stable and practical mechanism for performance-oriented parameter selection without exhaustive empirical tuning.

Overall, the results indicate that chunk-size sensitivity depends strongly on the integration method, with adaptive methods requiring significantly more careful scheduling. By focusing on chunk order rather than exact chunk values, the proposed predictive model can identify and effectively exploit stable performance trends.

LIMITATIONS AND POSSIBLE EXTENSIONS

Although the results demonstrate that the proposed chunk-order-based approach provides stable and accurate guidance for chunk selection, several limitations should be acknowledged. First, the experimental evaluation was conducted on a shared-memory multicore platform, and the observed performance trends may differ on heterogeneous or distributed-memory systems. Second, the study focuses on one-dimensional numerical integration problems and a specific set of test functions; more complex integrands or higher-dimensional problems may exhibit different workload characteristics, particularly for adaptive methods. Future work will therefore investigate the applicability of the proposed approach to distributed-memory systems, explore runtime integration within auto-tuning frameworks, and extend the evaluation to a broader class of numerical workloads.

CONCLUSION

This paper presents a comprehensive study of performance, accuracy, and chunk-size optimization for parallel numerical integration using Simpson and Trapezoidal methods, including their adaptive variants. The results show that chunk size is a critical parameter that significantly affects execution time and scalability, often more than the choice of numerical method. Adaptive methods offer improved accuracy but introduce irregular workloads that require careful scheduling to achieve good performance.

By identifying optimal chunk ranges and modeling their relationship to thread count, this work presents a practical predictive approach to chunk selection. The proposed method reduces the need for manual tuning and enables efficient parallel execution across varying levels of concurrency. These findings underscore the importance of performance-aware parameter selection in parallel numerical algorithms and provide a foundation for further research on automated optimization strategies in scientific computing.

From a practical perspective, the proposed chunk-order-based predictive model can support runtime or auto-tuning systems by providing efficient scheduling guidance without exhaustive parameter search. In practical runtime environments, the model can be invoked at application startup or during dynamic scheduling to estimate a near-optimal chunk order based solely on the number of threads. Future work will investigate its applicability to distributed-memory systems and extend the evaluation to a broader class of numerical workloads.

ACKNOWLEDGMENTS

The authors gratefully acknowledge support from the Serbian Ministry of Science, Technological Development, and Innovation (Contract No. 451-03-34/2026-03).

REFERENCES

Abdul Khalib, N., Kamaruddin, S. S., Ibrahim, Z. & Mohamed, Z. 2013. Performance evaluation of OpenMP scheduling policies and chunk sizes, International Journal of Computer Science and Network Security, 13 (2), pp. 1-8.

Barker, K., Chard, K., Foster, I., Katz, D. S., Wilde, M. & Ripeanu, M. 2012. Scientific software design for emerging platforms, IEEE Computer, 45 (11), pp. 47-53.

Bielecki, J. & Śmiałek, M. 2023. Estimation of execution time for computing tasks, Cluster Computing, 26, pp. 3943-3956. https://doi.org/10.1007/s10586-022-03774-1

Bishop, C. M. 2006. Pattern Recognition and Machine Learning, Springer.

Burden, R. L. & Faires, J. D. 2016. Numerical Analysis, 10th edn., Cengage Learning.

Chapman, B., Jost, G., & van der Pas, R. 2007. Using OpenMP: Portable Shared Memory Parallel Programming, MIT Press.

da Silva, F. H. S., Calheiros, R. N. & Schnorr, L. M. 2025. Auto-tuning OpenMP dynamic scheduling for improved performance across problem sizes and architectures, Computers & Operations Research, 167, 106041. https://doi.org/10.1016/j.cageo.2025.105932

Davis, P. J. & Rabinowitz, P. 2007. Methods of Numerical Integration, 2nd edn., Dover Publications.

Dinan, J., Olivier, S., Sabin, G., Prins, J., Sadayappan, P. & Tseng, C.-W. 2007. Dynamic load balancing of unbalanced computations using message passing, Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1-8. https://doi.org/10.1109/IPDPS.2007.370581

Estébanez, J., Llanos, D., Orden, D. & Palop, B. 2014. On the choice of the best chunk size for the speculative execution of loops, Journal of Supercomputing, 69 (1), pp. 475-497.

Hastie, T., Tibshirani, R. & Friedman, J. 2009. The Elements of Statistical Learning, 2nd edn., Springer. https://doi.org/10.1007/978-0-387-84858-7

Korndörfer, J. H., Müller, M., Eleliemy, A. & Ciorba, F. M. 2022. Automated scheduling algorithm selection and chunk parameter calculation in OpenMP, IEEE Transactions on Parallel and Distributed Systems, 33 (11), pp. 2802-2815. https://doi.org/10.1109/TPDS.2022.3189270

Laberge, G., Shirzad, S., Diehl, P., Kaiser, H., Prudhomme, S. & Lemoine, A. S. 2019. Scheduling optimization of parallel linear algebra algorithms using supervised learning, arXiv preprint arXiv:1909.03947.

Liu, D., Wu, J. & Zhang, X. 2017. The adaptive composite trapezoidal rule for Hadamard finite-part integrals on an interval, Journal of Computational and Applied Mathematics, 325, pp. 165-174. https://doi.org/10.1016/j.cam.2017.04.041

Lloyd, S., Irani, R. & Ahmadi, M. 2020. Using neural networks for fast numerical integration and optimization, IEEE Access, 8, pp. 84519-84531. https://doi.org/10.1109/ACCESS.2020.2991966

Plaškota, L. 2015. Automatic integration using asymptotically optimal adaptive Simpson quadrature, Numerische Mathematik, 131, pp. 173-198. https://doi.org/10.1007/s00211-014-0684-3

Quintero-Monsebaiz, R., Meneses-Viveros, A., Carranza, F., Cortés, C. G., González-Zamudio, A. & Vela, A. 2021. Multidimensional adaptive and deterministic integration in CUDA and OpenMP, Journal of Supercomputing, 77, pp. 12075-12097. https://doi.org/10.1007/s11227-021-03752-1

Shahvandi, M. K. 2021. Applications of numerical integration in geodesy and geophysics, Acta Geophysica, 69, pp. 1-17. https://doi.org/10.1007/s11600-020-00525-x

Smilić, M., Milić, D., Stefanović, Č., Đurović, S., Đošić, D. & Matejić, M. 2025. Analysis of sequential and parallel execution of Simpson's rule for numerical integration in Java, Proceedings of the 60th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST), Ohrid, North Macedonia, pp. 1-4. https://doi.org/10.1109/ICEST66328.2025.11098324

Tefera, A. G., Tarekegn, G. & Ayalew, B. 2024. Constraint-Guided Learning of Data-Driven Health Indicator Models, arXiv preprint arXiv:2407.13853.

Tonarelli, M., Riva, S., Benedusi, P., Ferrandi, F. & Krause, R. 2025. Adaptive multidimensional quadrature on multi-GPU systems, arXiv preprint arXiv:2511.01573.