Fast-Tracking Scalability Analysis: The PaScal Paramount Approach

Reilta Christine Dantas Maia[1], **Samuel Xavier-de-Souza**[2]

[1]Programa de Pós-Graduação em Engenharia Elétrica e de Computação
Universidade Federal do Rio Grande do Norte (UFRN) – Natal, RN – Brazil

[2]Departamento de Engenharia de Computação e Automação
Universidade Federal do Rio Grande do Norte (UFRN) – Natal, RN – Brazil

reilta.maia.701@ufrn.edu.br, samuel@dca.ufrn.br

Abstract. *Scalability analysis of High-Performance Computing applications is essential, but its process is time-consuming and costly. This work introduces PaScal Paramount, a new feature of the PaScal Analyzer tool that accelerates this analysis by estimating the total performance from the execution of a small fraction of the parallel code. Experiments demonstrated that the approach reduces analysis time by up to 89% while maintaining low estimation error and generating performance patterns faithful to the full execution. The feature offers an adjustable trade-off between speed and accuracy, proving to be an effective solution for gaining rapid insights into the behavior of applications, thereby guiding their optimization process.*

1. Introduction

Scalability is a metric that refers to the ability of an application or algorithm to efficiently leverage an increasing number of processors/resources as they become available [Kumar and Gupta 1994]. With advancements in fields such as artificial intelligence, scientific simulations, and big data, the demand for applications requiring high processing power has significantly increased [Chen et al. 2021]. In this context, scalability analysis becomes fundamental when choosing the best architecture, predicting system behavior under different loads, identifying performance bottlenecks, and guiding both the development of new algorithms and the evolution of hardware [Ramachandran et al. 1994]. The advancement of supercomputers, which have already surpassed the petaflops mark, reinforces the need to understand and model how applications perform in different configurations and architectures, ensuring that the available resources are fully utilized [Cameron et al. 2005].

Despite its importance, conducting scalability analysis presents significant challenges. The complex interaction between application characteristics and system architecture makes it difficult to accurately predict performance at a large scale. Traditional metrics, such as speedup, indicate trends but do not explain the origin of bottlenecks. To gain a deeper understanding, it is necessary to quantify the overheads that degrade scalability, such as workload imbalance and communication latencies—a task that often requires running the application in various configurations, varying both the input and the computational resources.

To address this gap and simplify the process of scalability analysis, the Parallel Scalability Suite (PaScal Suite) was developed. The framework includes two tools, the PaScal Analyzer [da Silva et al. 2022] and the PaScal Viewer [da Silva et al. 2019]: one for collecting execution data from different configurations and another for visualizing and interpreting this data. This approach aims to offer a more accessible and organized means for researchers and developers to evaluate the behavior of their parallel applications.

Although the PaScal Analyzer facilitates scalability analysis, this process can still be quite time-consuming, demanding significant time and resources. In this work, we propose the PaScal Paramount feature, which uses a technique called Paramount Iteration [Tavares et al. 2019]. This approach aims to reduce the time and resources required to obtain a scalability analysis through partial executions, making the data collection process less costly while delivering accurate estimates.

The results obtained from the implementation of PaScal Paramount demonstrate its effectiveness and potential. The new approach allowed for a significant reduction in analysis time and good accuracy in execution/performance estimates. The application of PaScal Paramount in High-Performance Computing (HPC) scenarios has the potential to aid in analyses by accelerating data collection and overcoming prohibitive time-consuming scalability analysis.

The remainder of this paper is organized as follows: first, the PaScal Analyzer and its workflow are presented in Section 2. Section 3 details the PaScal Paramount workflow, explaining how it integrates with and modifies the existing workflow. In Section 4, the results obtained with the implementation of PaScal Paramount are discussed. In Section 5, related works are presented, and finally, Section 6 summarizes the contributions and points to directions for future work.

2. PaScal Analyzer Workflow

The PaScal Analyzer is a tool focused on performance scalability analysis for parallel programs written in C/C++ for shared-memory systems, with support for POSIX Threads and OpenMP. Its workflow is designed to be minimally intrusive while offering flexibility and precision in collecting execution data [da Silva et al. 2022].

The main innovation of the PaScal Analyzer lies in its minimally intrusive approach, which results in an overhead of less than 1% on the application's performance. Instead of collecting an excessive amount of data, the tool focuses on essential measurements for scalability analysis, such as execution time and energy consumption. This strategy makes it lighter and easier to use.

The Analyzer's workflow (Figure 1) begins with the execution of the target application, using user-defined parameters. The analysis itself starts with the instrumentation of the application's source code. This instrumentation can be done in two ways: manually, where the user inserts specific routine calls to delimit the code regions to be monitored; or automatically, where the tool is responsible for identifying and encapsulating parallel blocks. These approaches rely on the use of a wrapper library to intercept calls to the underlying parallel APIs (such as OpenMP and POSIX Threads), allowing for the collection of execution time data with less than 1% overhead.

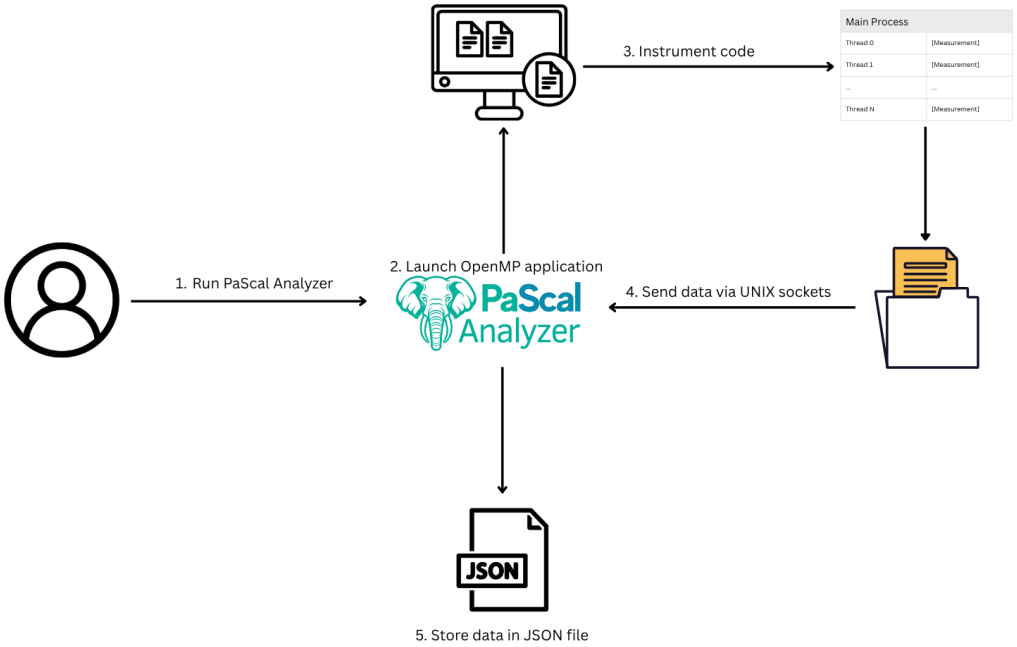


Figure 1. PaScal Analyzer Workflow.

Source: Adapted from [Gomes 2025].

During the application's execution, measurements from all threads are transmitted via `UNIX` sockets to an active instance of the `Analyzer`. It then unifies the collected data, organizes the measurements into a standardized structure, and generates an output file in JSON format containing information such as the total execution time and the time per parallel region. This file can be visualized with the Viewer, the graphical tool that allows for an intuitive interpretation of the data and the identification of scalability bottlenecks in specific code regions.

3. PaScal Analyzer with Paramount Iteration

The optimization and performance evaluation of HPC applications often demand full executions, which can be time-consuming and incur high costs, for instance, in cloud environments. To mitigate this problem, [Tavares et al. 2019] proposed the Paramount Iteration technique.

3.1. Paramount Iteration

A technique aimed at optimizing the performance analysis of HPC applications. Its premise is to perform a partial execution of the application, analyzing the behavior of the initial loops to infer the performance and scalability of the program as a whole. The technique is based on the observation that most parallel codes are iterative and tend to behave predictably after an initial warm-up period [Yang et al. 2005]. By measuring the execution time of a portion of the main loop's iterations, it is possible to efficiently estimate the performance, scalability, and cost of the application in different configurations, without the need for full executions [Tavares et al. 2019].

However, it is important to note that the effectiveness of `Paramount Iteration` depends on the nature of the application. It is best suited for codes with

deterministic loops that maintain predictable behavior throughout the execution. Applications whose loops are non-deterministic, dependent on random variables, or that significantly change their behavior over time may yield unsatisfactory estimation results [Tavares et al. 2019].

3.2. PaScal Paramount

To leverage the benefits of the `Paramount Iteration` technique, a new feature called `PaScal Paramount` was implemented in the `PaScal Analyzer`. The goal is to allow the user to obtain scalability analysis data in an accelerated manner, avoiding the long wait for full executions, especially for complex applications.

The feature is triggered through a call to the `pascal_paramount(id,paramount,total_it)` routine. This function is used to manually instrument the source code in the user's region of interest. The new routine leverages the existing time measurement routines in the `Analyzer`, `pascal_start(id)` and `pascal_stop(id)`, which were slightly modified to collect and store the time of each iteration per thread.

The operational flow of `PaScal Paramount` can be seen in the flowchart in Figure 2:

1. **Application Start:** The application begins to execute, and the initial time is recorded.
2. **Instrumentation:** In the region of interest, the call to the `pascal_paramount(id, paramount, total_it)` function initiates the partial analysis routine. The user specifies the region `id`, the number of iterations to be analyzed (`paramount`), and the total number of loop iterations (`total_it`).
3. **Partial Measurement:** The tool checks if the number of executed iterations is less than the defined `paramount` value. If so, measurement is performed for each iteration of the loop. Start and end times are collected per thread, and this data is stored for the subsequent estimation. Otherwise, the tool enters estimation mode.
4. **Estimation and Finalization:** When the number of iterations reaches the `paramount` value, the application stops the measurement. Based on the collected times, the `PaScal Analyzer` estimates the time for the remaining iterations. Initially, this estimation was done using the average of the times, but it was adjusted to use the median to mitigate the impact of cache misses and other initial warm-up effects. The application is then prematurely terminated.

The implementation of `PaScal Paramount` uses `C++` atomic variables to ensure data consistency and avoid race conditions, a critical concern when multiple threads concurrently access and manipulate shared counters. This lock-free approach is more efficient than traditional locking mechanisms, such as mutexes, as it avoids synchronization overhead. These atomic operations are crucial for safely managing the count of executed iterations and for coordinating the application's premature termination, ensuring it only exits after all threads have completed their measurement and estimation phases. The final result, which includes both the partial measurements and the estimate, is consolidated into a JSON output file, providing a complete scalability analysis in a fraction of the total execution time.

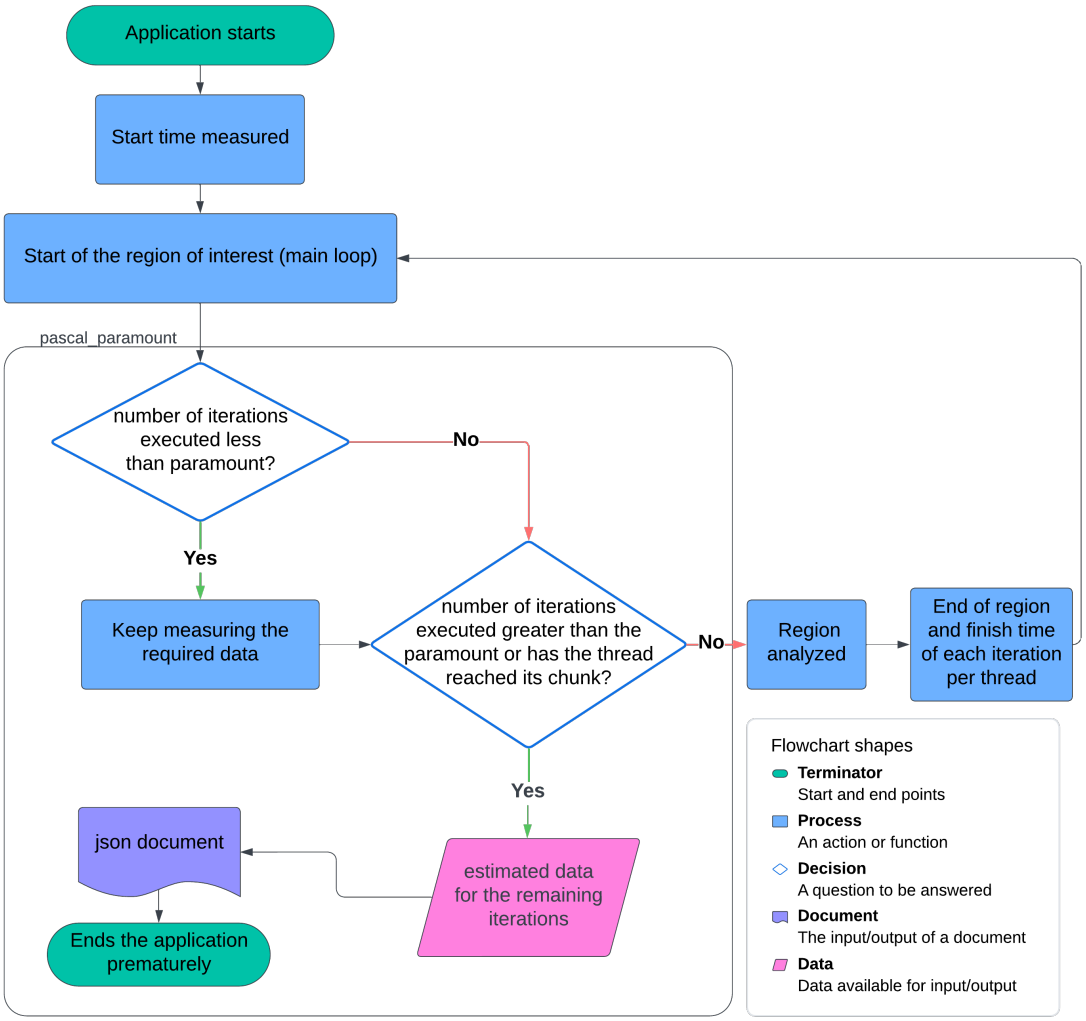


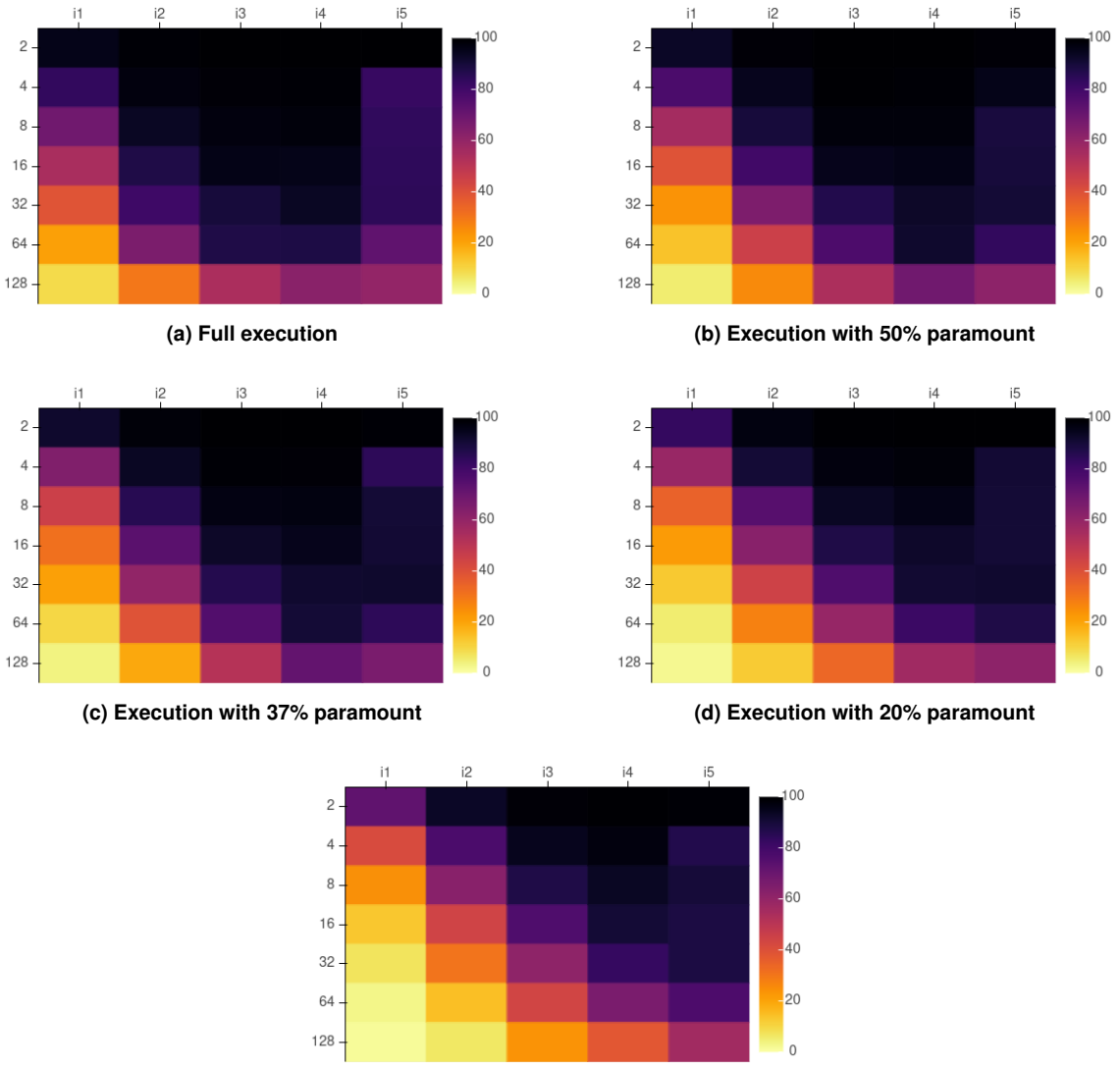
Figure 2. PaScal Paramount Workflow

4. Experiments and Results

In this section, we present the results of the tests conducted to validate the implementation of the approach presented in Subsection 3.2. The experiments were performed on a computational node in the `amd-512` partition of the supercomputer at the Federal University of Rio Grande do Norte. This node is equipped with 2 `AMD EPYC 7713 2.0GHz CPUs`, featuring a total of 128 processing cores and 512 GB of RAM.

For the experiments, we used a matrix multiplication algorithm. As inputs, we considered matrix orders of 256, 512, 1024, 2048, and 4096, as well as `paramount` rates of 10%, 20%, 37%, and 50% of the iterations.

In the execution with the `PaScal Analyzer`, we defined combinations of 128, 64, 32, 16, 8, 4, 2, and 1 processing cores with the aforementioned inputs. For better visualization of the graphs, we chose not to perform tests with all variations of the available core counts. The `Analyzer` was configured to repeat each combination of core count and input (problem size) 10 times. In the main loop, where the parallelism occurs, we used the `pascal_start()` and `pascal_stop()` routines to collect data from the re-



(a) Full execution

(b) Execution with 50% paramount

(c) Execution with 37% paramount

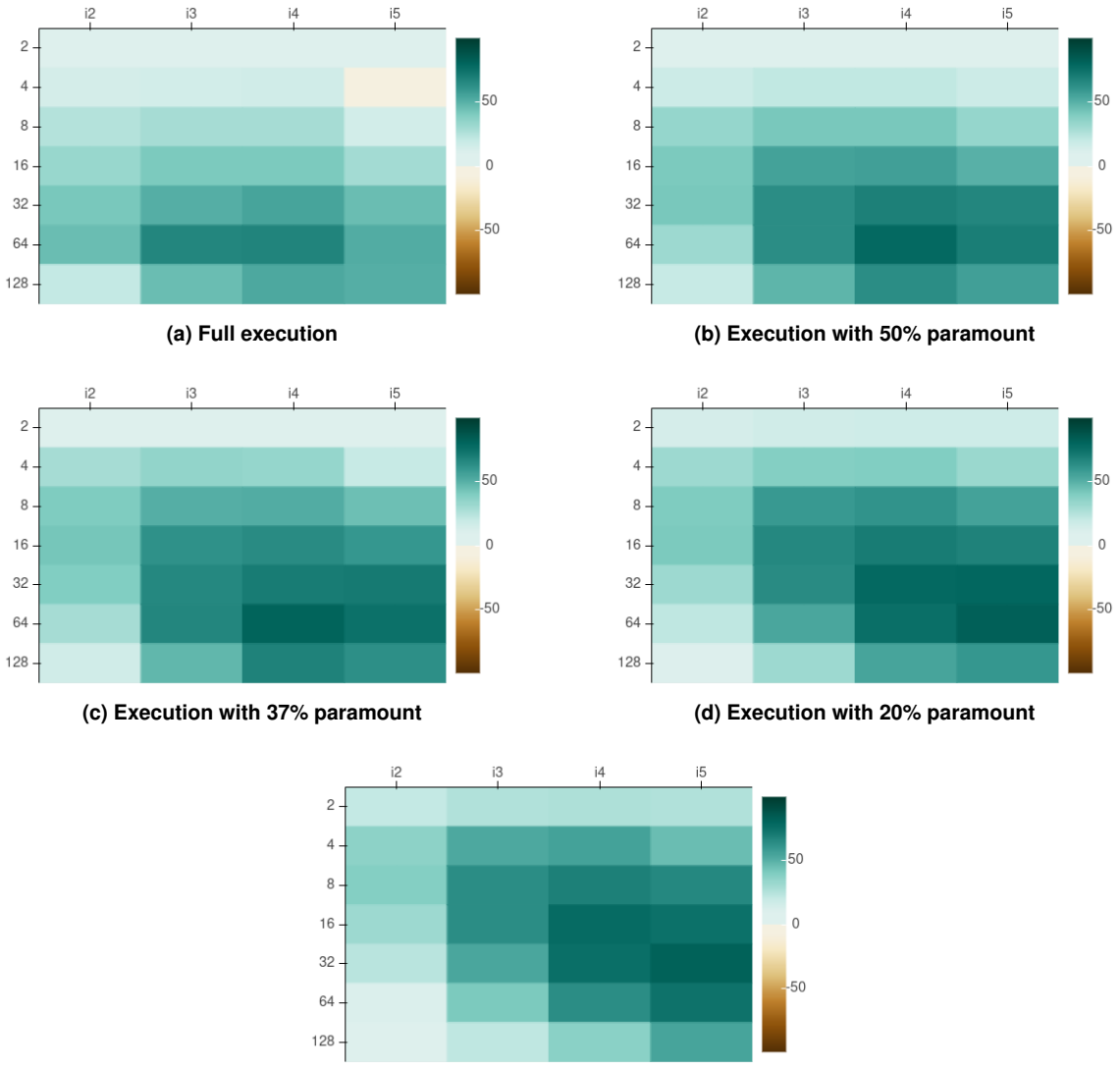
(d) Execution with 20% paramount

(e) Execution with 10% paramount

Figure 3. Efficiency diagram of the matrix multiplication algorithm. The number of cores varies on the vertical axis. The problem size varies on the horizontal axis.

gion during the full execution of the algorithm. To test `pascal_paramount()`, it was used in place of `pascal_start()`, where it only fully executes the defined portion of iterations and estimates the remainder.

With the help of the `PaScal Viewer`, diagrams were generated to visualize the collected data. Figure 3 displays the efficiency of the matrix multiplication algorithm. In the full execution (Figure 3a), a characteristic performance pattern is observed: efficiency is high for larger problems (i3 to i5) with few cores but degrades sharply when many cores (128) are applied to small problems (i1, i2), highlighting the impact of communication overhead. When comparing with the partial executions, it is noticeable that the results estimated by `PaScal Paramount` reproduce this behavior with high fidelity. In particular, the executions with `paramount` rates of 20% (Figure 3d), 37% (Figure 3c), and 50% (Figure 3b) generate diagrams that are very similar to the full execution, to



(a) Full execution

(b) Execution with 50% paramount

(c) Execution with 37% paramount

(d) Execution with 20% paramount

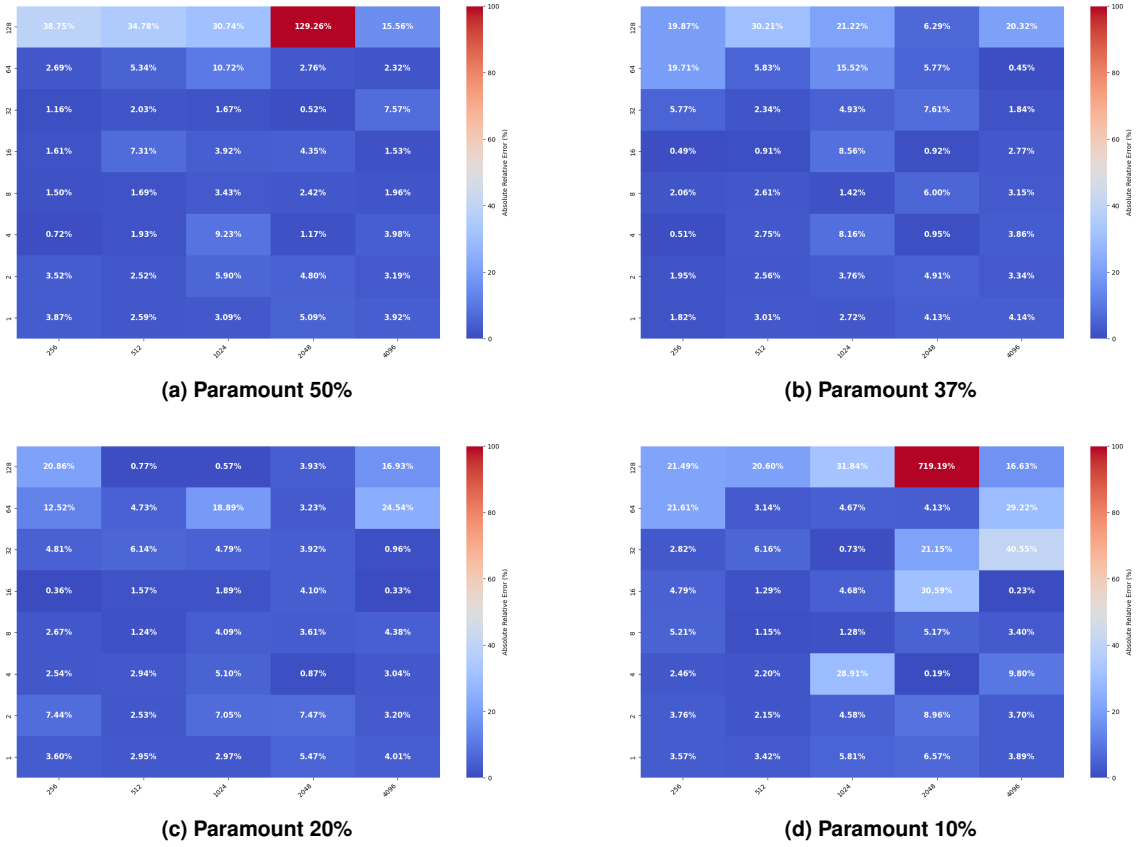
(e) Execution with 10% paramount

Figure 4. Scalability diagram of the matrix multiplication algorithm. The number of cores varies on the vertical axis. The problem size varies on the horizontal axis.

the point that a user could analyze the application based on these executions, validating the estimate's accuracy in capturing performance nuances.

Figure 4, which illustrates the application's scalability, reinforces this conclusion. Again, the patterns observed in the partial executions (Figures 4b-4e) are consistent with that of the full execution (Figure 4a), demonstrating that the method is equally effective for predicting how the application's performance behaves with the addition of more computational resources. This confirms that the approach is capable of capturing the application's scalability trend, allowing the developer to identify bottlenecks and optimal performance points without needing to execute the code in its entirety.

Figure 5 presents heatmaps of the absolute relative error, comparing the total execution time estimated by `PaScal Paramount` with the time from the full execution. The objective of this analysis is to quantify the precision of the estimation approach. In



(a) Paramount 50% **(b) Paramount 37%**

(c) Paramount 20% **(d) Paramount 10%**

Figure 5. Heatmaps of the absolute relative error of the executions for each input with pascal_paramount compared to the full executions. The number of cores varies on the vertical axis. The inputs vary on the horizontal axis.

general, a predominance of dark blue is observed in the graphs, which indicates a very low relative error in most of the tested scenarios. This validates the premise that the estimation based on the median of the initial iterations is, in most cases, quite accurate. However, a detailed analysis reveals a clear trade-off between the fraction of executed iterations and the robustness of the estimate.

The 10% `paramount` rate configuration (Figure 5d) is a good example of this. Although many of its estimates are reasonable, it shows an anomalous error of 719.19% for the configuration with 128 cores and the fourth input. This single extreme point raises the mean error of this configuration to 25.81%, masking the fact that the typical error is considerably lower (8.03%). This demonstrates that, while being the fastest approach, it carries a significant risk of producing inaccurate estimates in computationally sensitive scenarios, possibly due to system warm-up phases that do not reflect the application's stable behavior. A similar, though less extreme, phenomenon occurs in the 50% `paramount` configuration (Figure 5a), which, despite being mostly accurate, also registers an outlier of 129.26%.

In contrast, the 37% `paramount` (Figure 5b) and 20% `paramount` (Figure 5c) configurations demonstrate remarkable stability. Their mean errors were 5.44% and 4.78%, respectively, and, more importantly, they do not exhibit the extreme outliers seen in the other configurations. This suggests that the fraction of execution is already suffi-

(a) Paramount 50% **(b) Paramount 37%**

(c) Paramount 20% **(d) Paramount 10%**

Figure 6. Heatmaps of the time-saving percentages of the executions for each input with pascal_paramount compared to the full executions. The number of cores varies on the vertical axis. The paramount rate varies on the horizontal axis.

cient to capture the representative behavior of the application, thus ensuring greater reliability and predictability in the results.

In summary, Figure 5 validates the overall accuracy of `PaScal Paramount` but also serves as a guide for the user. The 10% rate is viable for very fast exploratory analyses, where the general trend is more important than numerical precision and the risk of a point error is acceptable. For most use cases that require high confidence in the data, the 20% and 37% rates prove to be the most robust choices, offering an effective balance between a drastic reduction in analysis time and high, consistent accuracy in the estimates.

Figure 6, in turn, quantifies the main benefit of `PaScal Paramount`: the time savings in performing the scalability analysis. The results demonstrate the percentage of time saved by the partial executions compared to the full execution, validating the main proposition of the approach.

The analysis shows that the smaller the fraction of executed iterations (`paramount` rate), the greater the time savings. The savings are substantial in all scenarios, starting with an average of 45-58% for the 50% `paramount` rate (Figure 6a) and

scaling progressively. With the 37% rate (Figure 6b), the savings already reach the 60-75% range. The benefit becomes even more expressive in the 20% (Figure 6c) and 10% (Figure 6d) configurations, where the time savings consolidate between 75-85% and can even exceed 90%, respectively.

However, the choice of the ideal `paramount` rate involves a cost-benefit analysis, weighing this time gain against numerical accuracy (Figure 5) and the visual fidelity of the performance patterns (Figures 3 and 4). The 10% rate, for example, is ideal for a quick exploratory analysis. It offers the maximum acceleration, with time savings approaching 90%, allowing for agile validation of code changes. However, as seen in Figure 5d, this speed comes with a higher risk of numerical imprecision, including the possibility of extreme outliers.

On the other hand, where maximum confidence and visual fidelity are crucial, the 37% and 50% rates represent the most robust choice. As demonstrated in Figures 3 and 4, these are the rates that produce the performance diagrams most faithful to the full execution. The user still benefits from expressive time savings (in the 40-75% range), with the added security of a more numerically stable and visually representative estimate.

Nevertheless, the joint analysis of the results points to the 20% rate as the optimal balance point for most use cases for the analyzed application. This configuration combines massive time savings, close to 80%, approaching the agility of the 10% rate, while maintaining excellent visual fidelity and, crucially, the numerical stability of the higher rates. In the tests performed, the 20% rate showed the lowest mean error and an absence of outliers, making it the ideal rate for the analysis of the tested application.

Thus, the results not only validate the acceleration provided by `PaScal Paramount` but also demonstrate its flexibility, allowing the user to adjust the tool according to their specific needs, whether it be maximum speed for exploration or maximum fidelity in the analysis of results.

5. Related Work

Performance prediction and analysis of parallel applications are classic challenges in high-performance computing. Traditional approaches often fall into two extremes. On one side, there are abstract analytical models, which are fast but fail to capture complex system details; on the other, there are detailed simulation techniques, which are accurate but extremely costly in time and resources, as pointed out by [Adve and Sakellariou 2000]. The search for methods that combine accuracy and low cost is, therefore, an active field of research.

A fundamental premise that has enabled significant advancements in this area is the observation that most scientific applications are iterative in nature. [Yang et al. 2005] demonstrated that, after a brief initialization period, the performance behavior of a parallel program tends to stabilize. Based on this, they proposed that short partial executions could be used to predict the performance of the full execution with high accuracy, eliminating the need for complex program modeling and drastically reducing the cost of analysis.

Following this principle, several approaches have emerged. [Zhang et al. 2017], for example, presented a compiler-based technique (`LLVM`) to statically analyze iteration frequency and combine it with instruction measurements to predict performance on

large-scale systems from a single-node execution. More directly and in line with our work, [Tavares et al. 2019] formalized the partial execution technique under the name `Paramount Iteration`. They successfully applied it to predict the performance of cloud applications, demonstrating its utility for estimating costs and choosing hardware configurations without the need for full and expensive executions.

The present work fits into this context, utilizing the foundations established by [Yang et al. 2005] and the `Paramount Iteration` technique from [Tavares et al. 2019]. Our contribution, however, is distinguished by integrating this technique directly into an existing and validated scalability analysis suite, the `PaScal Suite`. The foundation for our implementation is the `PaScal Analyzer` [da Silva et al. 2022], a tool designed to collect performance data with minimal intrusion on the target code. While previous works focused on predicting a final performance value, our goal with `PaScal Paramount` is to accelerate the scalability analysis workflow. That is, instead of just estimating the total time, our tool quickly generates the same scalability and efficiency diagrams as a full execution, bridging the gap between numerical prediction and practical, visual performance analysis.

6. Conclusion

Scalability analysis is a pillar in the development of HPC applications, yet its cost in time and computational resources represents a significant obstacle for researchers and developers. This work addressed this challenge with the introduction of `PaScal Paramount`, a new feature integrated into the `PaScal Analyzer` tool. By implementing the `Paramount Iteration` technique, the tool drastically accelerates the collection of performance data by estimating the application's full behavior from the execution of a small fraction of its iterations.

The experimental results demonstrated the effectiveness and robustness of the approach. `PaScal Paramount` achieved a reduction in analysis time of up to 89% for larger-scale problems, an expressive gain that makes experimentation more agile. More importantly, this acceleration was achieved while maintaining a generally low estimation error and, crucially, preserving the fidelity of the visual patterns of efficiency and scalability, especially with `paramount` rates of 37% and 50%. The analysis revealed a clear and useful trade-off, allowing the user to opt for maximum speed for exploratory analyses (lower rates) or for maximum fidelity to the application's pattern (higher rates).

The main contribution of this research is, therefore, a flexible solution that makes scalability analysis more practical and accessible. By providing quick and reliable insights into the code's behavior, `PaScal Paramount` empowers users to understand and guide the optimization process of their applications more efficiently, reducing the cycle between identifying a bottleneck and its solution.

For future work, we plan to develop a more advanced estimation model to better handle unexpected errors. Furthermore, researching ways to automate the choice of the ideal paramount rate, based on an initial analysis of the application, is a promising direction for further enhancing this tool.

7. Acknowledgments

This work was supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and the Programa de Mestrado e Doutorado Acadêmico para Inovação (MAI/DAI). We also acknowledge the computing support from the Núcleo de Processamento de Alto Desempenho (NPAD/UFRN).

References

Adve, V. and Sakellariou, R. (2000). Application representations for multiparadigm performance modeling of large-scale parallel scientific codes. *The International Journal of High Performance Computing Applications*, 14(4).

Cameron, K. W., Ge, R., and Feng, X. (2005). High-performance, power-aware distributed computing for scientific applications. *Computer*, 38(11):40–47.

Chen, J., Becker, B. A., Ouyang, Y., and Shen, L. (2021). What Influences Students' Understanding of Scalability Issues in Parallel Computing. *The Journal of Computational Science Education*, 12(2):58–65.

da Silva, A. B., Cunha, D. A., Silva, V. R., de A. Furtunato, A. F., and Xavier-de Souza, S. (2019). PaScal Viewer: A Tool for the Visualization of Parallel Scalability Trends. In *International Workshop on Extreme-Scale Programming Tools*, pages 250–264. Springer.

da Silva, V., da Silva, A., Valderrama, C., Manneback, P., and Xavier-de Souza, S. (2022). A Minimally Intrusive Approach for Automatic Assessment of Parallel Performance Scalability of Shared-Memory HPC Applications. *Electronics*, 11:689.

Gomes, J. F. P. (2025). Extending the PaScal Analyzer for MPI Scalability Analysis: Design, Implementation, and Validation. B.s. thesis, Universidade Federal do Rio Grande do Norte.

Kumar, V. P. and Gupta, A. (1994). Analyzing scalability of parallel algorithms and architectures. *Journal of parallel and distributed computing*, 22(3):379–391.

Ramachandran, U., Venkateswaran, H., Sivasubramaniam, A., and Singla, A. (1994). Issues in understanding the scalability of parallel systems. In *Proceedings of the First International Workshop on Parallel Processing, Bangalore, India*, pages 399–404.

Tavares, W., Reis, L., Brunetta, J., and Borin, E. (2019). Aplicação da técnica Paramount Iteration nas aplicações BLAST e DNN-ROM na nuvem computacional. In *Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 228–239. SBC.

Yang, L. T., Ma, X., and Mueller, F. (2005). Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution. IEEE.

Zhang, M., Hao, M., and Snir, M. (2017). Predicting HPC parallel program performance based on LLVM compiler. *Cluster Computing*, 20(2):1233–1244.