An Approach Based on Partial Executions for Evaluating the Scalability of Parallel Applications

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

[1] Departamento de Engenharia de Computação e Automação
Universidade Federal do Rio Grande do Norte (UFRN)
59.078-970 – Natal – RN – Brazil

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

Abstract. *Scalability analysis is crucial for developers to gauge how their applications perform under varying resource and workload conditions. However, analyzing complex applications can be time-consuming. This study proposes a partial execution approach in the Parallel Scalability Analyzer tool, leveraging the Paramount Iteration technique. This approach holds significant potential for reducing scalability analysis time.*

1. Introduction

The use of parallelization techniques and the increase in computational resources are alternatives for solving increasingly complex problems. However, according to [Nguyen et al. 2016], optimizing the performance of parallel applications is becoming increasingly challenging, given that they need to scale as computational resources increases. In this context, scalability analysis is a great ally for developers and researchers to help identify application bottlenecks. However, these analyses require execution times that can be long, depending on the complexity of the application. In this context, this study proposes the implementation of a partial execution approach with the Paramount Iteration technique in the scalability analysis tool, Parallel Scalability Analyzer [da Silva et al. 2022], to make the analysis process more agile. This technique estimates the total execution time by focusing on a specific percentage of iterations, eliminating the need for complete execution.

2. Proposed Approach

The Paramount Iteration technique was proposed by [Tavares et al. 2019] as an alternative to help choose cloud computing configurations, since it is necessary to carry out tests to evaluate the performance and scalability of applications in various configurations. The proposed technique eliminates the need to run applications in their entirety. Just a few portions of the applications would be enough to estimate their behavior. It was based on the premise that the applications have a repetitive nature, since most parallel codes are iterative and behave predictably after a minimal initial period [Yang et al. 2005].

The approach proposed in this work was designed with the objective of reducing the execution time for obtaining results and analyses of applications, considering that execution can be extremely time-consuming, especially for applications in high-performance computing environments. To achieve this, a manual instrumentation routine, `pascal_paramount`, was developed, which allows the user to delimit a region of interest in the source code.

The user specifies three key parameters for `pascal_paramount`: the region identifier (`id`), the number of iterations to be effectively executed and measured (`paramount`), and the total number of iterations for the region (`total_it`). To ensure data integrity and avoid synchronization overhead in multithreaded environments, atomic variables are employed for controlling iterations and handling critical data.

After executing the paramount iterations, the time for the remaining iterations is estimated based on the median of the measured times – a choice made to mitigate the impact of initial overheads (e.g., cache misses). Once the estimation is completed for all threads, the application is terminated prematurely, and the total estimated time is calculated, significantly optimizing the analysis process.

Validation tests were conducted on the compute nodes of the High-Performance Computing Center (NPAD/UFRN) using two applications: a matrix-based code and a ray tracing code. In the matrix tests, execution time estimates using `pascal_paramount()` showed relative errors often below 1% and achieved time reductions of over 80% with smaller paramount values (e.g., 10%).

For the ray tracing application, which features more irregular iteration behavior, estimation errors were higher—often exceeding 10%—with lower gains in execution time. These results demonstrate that the approach is particularly effective for applications with consistent per-iteration workloads.

For future work, the intention is to develop a more advanced model to identify and handle estimates that behave as outliers. An estimate is considered an outlier when its measured iterations diverge significantly from the application's average behavior, a common issue in codes with high workload variability. The proposed model will discard these discrepant estimates and replace them with more representative measurements. Furthermore, we aim to create an automated routine to define the ideal Paramount Iteration rate based on tests from a few representative resource configurations and workloads. This approach will eliminate user intervention in selecting the rate, using the best-found rate to estimate remaining configurations and thus obtain scalability results more quickly.

References

[da Silva et al. 2022] 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.

[Nguyen et al. 2016] Nguyen, H. T., Wei, L., Bhatele, A., Gamblin, T., Boehme, D., Schulz, M., Ma, K.-L., and Bremer, P.-T. (2016). VIPACT: A visualization interface for analyzing calling context trees. In *Proceedings of the Workshop on Visual Performance Analysis*, pages 1–8. IEEE Press.

[Tavares et al. 2019] 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 et al. 2005] Yang, L. T., Ma, X., and Mueller, F. (2005). Cross-platform performance prediction of parallel applications using partial execution. IEEE.