Performance Evaluation of N-Body Simulations on AWS with StarPU, OpenMP and MPI Runtime Systems

Nicolas Vanz[1], **Vanderlei Munhoz**[1,2], **Márcio Castro**[1]

[1] Federal University of Santa Catarina – Florianópolis – Brazil

[2]University of Bordeaux – Talence – France

`nicolas.vaz@ufsc.br, vanderlei.munhoz-pereira-filho@inria.fr`
`marcio.castro@ufsc.br`

Abstract. *Cloud Computing provides a cost-effective option for High Performance Computing (HPC) workloads, but brings new challenges. This study evaluates StarPU, a task-based runtime for heterogeneous architectures, in cloud environments by running an N-Body simulation under different cluster configurations and comparing it with traditional HPC runtime systems (OpenMP and MPI). Results show that StarPU excels in single-node setups, especially with GPU acceleration, while scalability varies, struggling with CPU-intensive workloads but performing well in hybrid and GPU-only scenarios. These insights highlight the need for careful infrastructure selection and architectural strategies to ensure good performance in cloud-based HPC.*

1. Introduction

Cloud Computing offers notable advantages for computational workloads, including on-demand access to advanced resources, scalable provisioning, and a cost-efficient pay-per-use model. However, extending these benefits to High Performance Computing (HPC) introduces several challenges, especially in addressing the stringent demands of scientific applications [Netto et al. 2018, Guidi et al. 2021, Dancheva et al. 2024].

To address these challenges, the HPC landscape has increasingly turned to heterogeneous computing architectures, integrating hardware accelerators like Graphic Processing Units (GPUs) to enhance parallel processing capabilities. GPUs have become central to the architecture of leading supercomputing systems, delivering significant performance improvements for compute-intensive, parallelizable tasks [Zwart et al. 2007, Nylons et al. 2007, Wei et al. 2022]. Complementing this hardware evolution, task-based parallelism has emerged as an effective programming paradigm. By decomposing applications into smaller, independent tasks defined by explicit data dependencies, task-based frameworks enable dynamic scheduling, allowing workloads to be efficiently distributed across heterogeneous resources based on current system conditions [Augonnet et al. 2011, Bueno et al. 2012, Hoque et al. 2017]. This adaptive approach contrasts with traditional static allocation methods [Graham et al. 2006, Chandra 2001], improving load balancing and reducing idle time in heterogeneous HPC environments [Pereira 2023].

In this study, we explore the practical viability of deploying a task-based HPC application on a general-purpose public cloud platform. Focusing on a representative N-Body simulation, we compare two execution models on Amazon Web Services (AWS): a dynamic task-scheduling approach using the StarPU [Augonnet et al. 2011] framework to manage heterogeneous resources, and a conventional HPC setup combining Open Multi-Processing (OpenMP) [Chandra 2001] and Message Passing Interface (MPI) [Graham et al. 2006] for parallelization. To facilitate the deployment and management of these configurations, we leverage the HPC@Cloud Toolkit [Munhoz and Castro 2024], which streamlines infrastructure handling and accelerates testing in the cloud.

In summary, this work contributes to the field through the following advancements: (i) a comprehensive analysis of the advantages and challenges of leveraging heterogeneous computing in cloud HPC environments, along with strategies to effectively harness performance from hardware heterogeneity; (ii) a comparative evaluation of modern HPC frameworks, such as StarPU, highlighting their benefits and limitations in relation to traditional approaches that combine OpenMP and MPI.

The remainder of this paper is organized as follows. Section 2 introduces the fundamental concepts of cloud HPC and the StarPU runtime system. Section 3 describes the N-Body problem and its implementations using StarPU and OpenMP/MPI. Section 4 details the experimental setup, while Section 5 presents and analyzes the results. Section 6 reviews related work. Finally, Section 7 summarizes the main findings and concludes the paper.

2. Background

This section provides the background to contextualize our work at the intersection of HPC and cloud environments. We begin by discussing the fundamental characteristics of traditional HPC systems and cloud computing infrastructures (Section 2.1). Next, we introduce StarPU, a runtime system designed to simplify and optimize task scheduling across heterogeneous hardware platforms (Section 2.2). Finally, we present the HPC@Cloud Toolkit, a lightweight, cloud-agnostic framework used in this study to automate the deployment and management of HPC clusters in public cloud environments (Section 2.3).

2.1. High Performance Computing and Cloud Computing

HPC refers to the use of powerful computational systems designed to solve complex scientific and engineering problems that require extensive processing power, high-speed interconnects, and large-scale data handling capabilities. Traditional HPC infrastructures are typically composed of tightly-coupled clusters or supercomputers, where nodes are interconnected through high-bandwidth, low-latency networks, such as InfiniBand. These environments are optimized to deliver peak performance by providing fine-grained control over resource allocation, process placement, and data locality. HPC applications often rely on parallel programming paradigms like MPI and OpenMP to exploit the parallelism available in these architectures, enabling efficient execution of simulations, numerical methods, and large-scale data analyses.

In contrast, Cloud Computing has emerged as a flexible, on-demand resource provisioning model, offering scalability, elasticity, and cost-efficiency through a pay-per-use pricing model. Data centers are designed to serve a broad spectrum of workloads, prioritizing ease of deployment, service availability, and operational scalability for enterprise applications, web services, and data-driven workflows [Liberman García 2015]. The abstraction layers provided by virtualization and containerization technologies are central to Cloud Computing, enabling resource sharing among multiple tenants and simplifying infrastructure management.

Despite its numerous advantages, the convergence of HPC workloads with cloud environments presents significant challenges. Scientific applications often demand low-latency communication, high-throughput data exchanges, and deterministic execution, which are difficult to achieve within virtualized, multi-tenant cloud architectures. Large-scale simulations typically involve datasets that exceed the memory capacity of individual nodes, requiring distributed data partitioning and frequent synchronization across nodes. However, in cloud settings, the inherent variability in network performance, resource contention, and limited control over hardware affinity exacerbate communication overheads and introduce unpredictable performance fluctuations. These limitations pose critical barriers to efficiently executing HPC workloads in public cloud infrastructures, especially when compared to the tightly-coupled nature of traditional HPC clusters.

To address these challenges, the HPC community has increasingly embraced heterogeneous computing architectures as a strategy to enhance computational throughput and overall efficiency. The adoption of specialized hardware accelerators, most notably GPUs, has revolutionized performance capabilities in supercomputing systems [Zwart et al. 2007, Nylons et al. 2007, Wei et al. 2022]. GPUs, with their massively parallel processing cores, are particularly well-suited for workloads involving dense numerical computations, matrix operations, and data-parallel tasks. Heterogeneous computing holds particular significance in cloud environments, where the availability of specific hardware resources is inherently uncertain, both within a single provider and across different providers, especially when portability to alternative cloud platforms is a consideration. However, efficiently harnessing the full potential of heterogeneous resources requires sophisticated programming models and runtime systems capable of dynamically managing resource allocation, workload distribution, and data movement, further complicating the convergence of HPC workloads with cloud-native environments.

2.2. StarPU

StarPU [Augonnet et al. 2011] is a runtime system designed to simplify the development and execution of task-based parallel applications on heterogeneous architectures. It abstracts the complexities of hardware heterogeneity by (i) allowing developers to provide multiple implementations of the same task, each optimized for a different type of processing unit, such as a CPU or a GPU; and (ii) enabling programmers to express applications as task graphs, where tasks are defined with explicit data dependencies. This flexibility allows developers to describe alternative execution strategies for tasks depending on the available hardware resources. StarPU then dynamically schedules these tasks across CPUs and GPUs, optimizing resource utilization while minimizing idle times and unnecessary data movements.

One of StarPU's key strengths lies in its ability to transparently manage the offloading of computational kernels, synchronization and data coherency across heterogeneous systems. This is particularly valuable when dealing with diverse hardware resources, as statically mapping and synchronizing tasks in such environments is not only complex but often non-portable. StarPU's dynamic scheduling capabilities allow applications to fully exploit the specific strengths of Central Processing Units (CPUs) and GPUs in a portable and scalable manner.

StarPU provides several schedulers. In this work, we utilize the Deque Model Data-Aware (DMDA) scheduler [Augonnet et al. 2011], which bases its scheduling decisions on estimated task execution periods and data transfer expenses. It maintains performance models for each kernel on various processing units, refining execution time predictions as tasks finish. By accounting for both computation duration and data movement costs, DMDA minimizes total execution time, offering significant advantages in public cloud settings, where data transfer can be a major limitation.

2.3. HPC@Cloud

The HPC@Cloud Toolkit [Munhoz and Castro 2024] is an open-source framework that automates the deployment and management of HPC clusters in cloud environments[1]. It simplifies resource provisioning and cluster configuration, allowing users to rapidly set up customized HPC infrastructures for various workloads.

Unlike AWS ParallelCluster, for example, which targets production-grade deployments with deep AWS integration, HPC@Cloud offers a lightweight, cloud-agnostic approach focused on ease of use and fast prototyping. It toolkit utilizes user definitions in the form of HashiCorp Configuration Language (HCL) files, used by Terraform, an industry-standard Infrastructure as Code (IaC) tool widely adopted in enterprise environments. The toolkit manages the entire lifecycle of cloud resources by implementing the generated Terraform plans to provision resources at the start of an experiment, while also ensuring comprehensive decommissioning after completion, thereby avoiding orphaned resources and unnecessary costs. The toolkit was utilized in this study to provision the experimental cloud environment, avoiding dependency on vendor-specific HPC solutions.

3. N-Body Simulations

The N-Body Problem is a fundamental topic in computational physics and astrophysics, as it models the dynamics of particles interacting under mutual forces, typically governed by Newtonian gravity or electrostatic interactions [Heggie 2005]. The core complexity of this problem lies in the fact that every particle exerts a force on, and is simultaneously influenced by, every other particle in the system. Consequently, the computational workload scales quadratically with the number of bodies involved for direct calculation approaches. This inherent computational demand turns the N-Body Problem a good candidate for evaluating the performance of HPC architectures and parallel computing frameworks.

Traditionally, solutions for the N-Body Problem involve either direct methods, which calculate all pairwise interactions, or approximate methods like the Barnes-Hut algorithm [Barnes and Hut 1986] and the Fast Multipole Method (FMM) [Wang 2021], which lower computational demands by grouping distant particles and estimating their effects. These techniques often employ parallelization with MPI or hybrid MPI and OpenMP strategies to distribute computations over multiple nodes. For simplicity, this work explores a direct method, which grows quadratically to the number of simulated bodies.

The implemented solution for the N-Body problem employs a direct computation technique, methodically determining the gravitational forces exerted on each particle by assessing all pairwise interactions present in the system. Although the direct method produces highly accurate results, our main objective is to utilize this algorithmically intensive workload to stress-test and evaluate different Cloud Computing setups and tools, rather than capitalizing on its precision advantages for scientific exploration.

[1]HPC@Cloud Toolkit's repository: `https://github.com/lapesd/hpcac-toolkit`

In the algorithm, we apply the Equation 1 [Nylons et al. 2007, Zwart et al. 2007] to determine the force F between two particles i and j during a simulation time-step. This equation simplifies Newton's law of universal gravitation by postulating that the bodies have identical masses.

$$\vec{F}_{ij} = \frac{\vec{r}_j - \vec{r}_i}{\left(|\vec{r}_j - \vec{r}_i|^2 + \epsilon^2\right)^{3/2}} \tag{1}$$

Thus, the overall force acting on a particle i is determined by summing the forces applied by all remaining particles as stated in Equation 2:

$$\vec{F}_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} \vec{F}_{ij} \tag{2}$$

We thus derive the acceleration for each particle, which correlates with the total force. Subsequently, we update the velocity (Equation 3) and position (Equation 4).

$$\vec{v}_i(t + \Delta t) = \vec{v}_i(t) + \vec{a}_i(t) \cdot \Delta t \tag{3}$$

$$\vec{r}_i(t + \Delta t) = \vec{r}_i(t) + \vec{v}_i(t) \cdot \Delta t \tag{4}$$

From a computational perspective, our entire system was modeled using two data arrays: one for position vectors and the other for velocity vectors. We defined two tasks, designed to: (i) compute the total force on each particle; and (ii) update their positions. Figure 1 illustrates the task execution flow of the N-Body simulation. At each simulation step, a set of tasks is launched to compute the inter-particle forces in two phases. During the first phase, the positions array is accessed in read-only mode (read dependency), while the velocities array is updated by the tasks (write dependency). To enable parallel execution, the velocities array is partitioned into segments and distributed across the available compute nodes, whereas the positions array is replicated to provide the necessary input for all force calculations. This data segmentation and distribution are managed either implicitly by StarPU or explicitly via manual partitioning in the MPI+OpenMP implementation. Following the force computation, a new set of tasks is generated to perform position integration. In this second phase, however, the positions array is updated by the tasks (write dependency), while the velocities array is accessed in read-only mode (read dependency). As before, the arrays are partitioned and assigned to parallel tasks. Once all tasks have completed, data synchronization is performed to ensure consistency across nodes, thereby preparing the system state for the next simulation step.

3.1. StarPU Implementation

In the StarPU implementation, tasks are executed across a pool of heterogeneous workers, comprising both CPUs and GPUs. Since each task's input data dependencies are determined by the specific array indices it accesses or modifies, the data arrays are asynchronously partitioned and assigned to the corresponding tasks by StarPU at runtime considering current data distribution and resource usage. We define the partitioning strategy (number and size of partitions), while StarPU dynamically determines when to partition

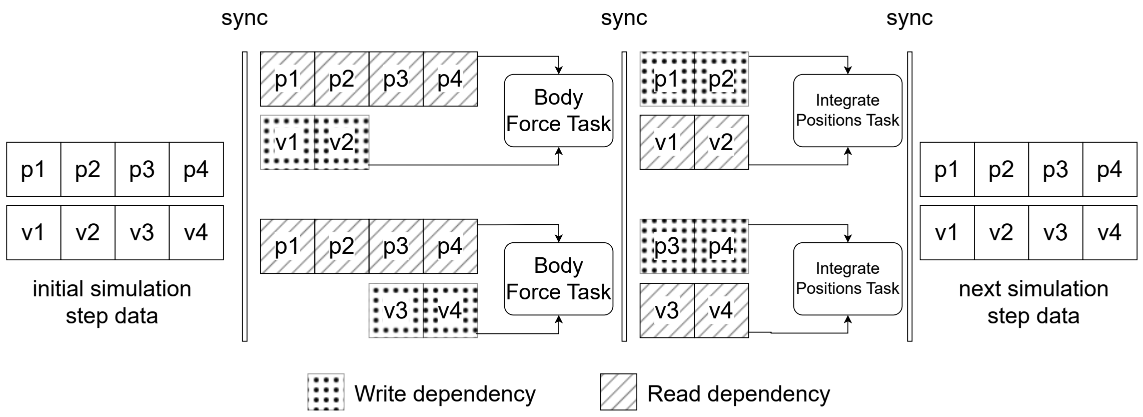

Figure 1. Illustration of data dependencies and task management of the implemented N-Body solvers.

and transfer data based on task scheduling. Task distribution across nodes is performed statically, with tasks evenly divided among the nodes. Within each node, scheduling decisions are managed by the DMDA scheduler. The StarPU task distribution approach adheres to its core principle of providing the same task behavior on different processing units: the total number of tasks always matches the number of available workers in the cluster (CPUs + GPUs), in contrast with the MPI+OpenMP approach. Additionally, one CPU core per node is reserved for scheduling operations.

Figure 1 illustrates how positions (p_i) and velocities (v_i) are handled throughout each simulation phase. The data arrays are dynamically segmented and reassembled to align with the specific requirements of the current tasks. For example, computing the total force on each body requires access to the entire position array, whereas updating particle positions only necessitates a localized subset of the data. StarPU implicitly manages data dependencies by analyzing the task graph scheduling, orchestrating data transfers and maintaining coherence across processing units.

Parallelism in this implementation is achieved by decomposing every simulation step into numerous fine-grained tasks, which can be scheduled for execution on any available processing unit. StarPU dynamically orchestrates task scheduling and resource allocation at runtime, leveraging performance models to decide which processing unit, CPU or GPU, should execute each task based on current workload conditions and resource availability. This adaptive scheduling approach enables effective load balancing and resource utilization across heterogeneous cloud infrastructures.

3.2. OpenMP+MPI Implementation

In the implementation that combines OpenMP with MPI, parallelism is achieved through a static data decomposition approach. At each simulation time step, the arrays are statically partitioned, distributed to the respective nodes, and later gathered to synchronize results. This segmentation process is repeated in every iteration step, imposing explicit data management and task distribution responsibilities on the programmer. Unlike the StarPU implementation, where task distribution is dynamic and guided by runtime scheduling policies, the MPI+OpenMP approach enforces a fixed distribution strategy: at each time step, large computational tasks are spawned and assigned to each node. Within a node, these tasks are executed using a fork-join model with multiple CPU threads or offloaded to GPUs through OpenMP target offloading directives.

In the multi-node implementation, the number of tasks depends on the target processing unit. GPU workloads are handled by spawning one task per available GPU, while CPU workloads are done by spawning one task per physical core in the cluster. Parallelism in this model is thus realized by generating coarse-grained tasks per node, which are further processed locally. On the CPU side, tasks are subdivided into smaller chunks for execution by OpenMP threads, while for GPU execution, the entire workload segment is offloaded to the accelerator using the `target` directive.

4. Experimental Setup

We designed an experimental setup leveraging the HPC@Cloud Toolkit for automated provisioning and configuration. The experimental infrastructure was structured into two configurations: (i) single node, where each experiment was executed on a standalone instance; and (ii) homogeneous multi-node clusters, comprising 2, 4, and 8 nodes, with all nodes utilizing identical instance types. This experimental design facilitated an evaluation of performance across different hardware configurations, enabling the assessment of both raw computational efficiency and scalability as the number of nodes increased.

All experiments were conducted using the `g6.16xlarge` instance type, featuring 64 vCPUS (32 physical cores), a NVIDIA L4 GPU with 24 GiB of memory, and 256 GiB of RAM. To ensure consistency and reduce variability across runs, simultaneous multithreading (e.g., Hyper-Threading) was disabled on all nodes. The software stack employed in the experiments included StarPU 1.4.7, OpenMP 5.0, and OpenMPI 5.0.7, with all system images based on the official release of Amazon Linux 2023.6.20250218. For the OpenMP implementation, the number of threads was restricted to match the number of physical CPU cores, ensuring a fair comparison with the StarPU configuration, which is designed to schedule tasks exclusively on physical cores.

5. Results and Discussion

We begin our analysis by evaluating performance in three distinct single-node scenarios: CPU-only execution, GPU-only execution, and hybrid execution leveraging both the CPU and GPU. These experiments establish a baseline for assessing computational efficiency on a single instance. We then extend the study to multi-node (cluster) configurations to investigate the scalability of the implementation and evaluate the efficiency of horizontal scaling across distributed cloud resources.

We evaluated execution times for two input sizes: a "small" input consisting of 2^{19} bodies and a "large" input with 2^{20} bodies. Each experiment was executed ten times, and the reported results represent the average execution time. Performance was compared between the StarPU-based implementation and an alternative approach that combines OpenMP and MPI.

5.1. Single-node Results

Figure 2 presents the execution times (in seconds) of the N-Body simulation on a single node, comparing the performance of CPU-only, GPU-only, and hybrid CPU-GPU configurations. Across all configurations and input sizes, the StarPU-based implementation consistently outperformed the OpenMP-based approach. Notably, configurations leveraging GPU acceleration with StarPU achieved substantial performance improvements. The experiments revealed that using the GPU for force computation led to significant

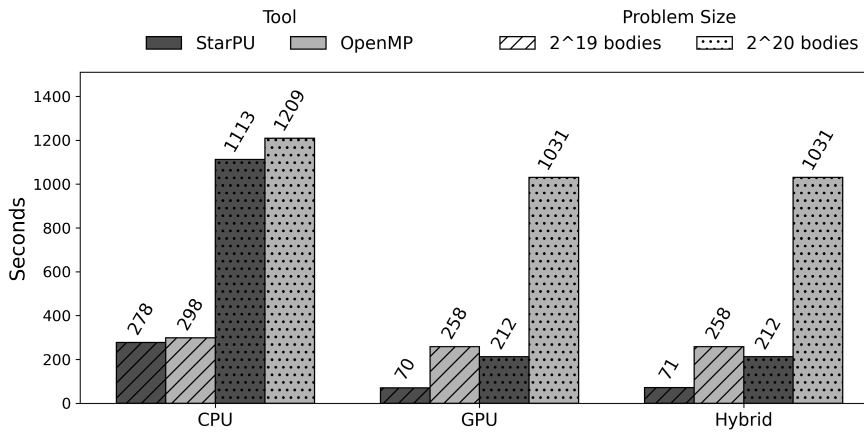

Figure 2. Execution time (in seconds) on a single node.

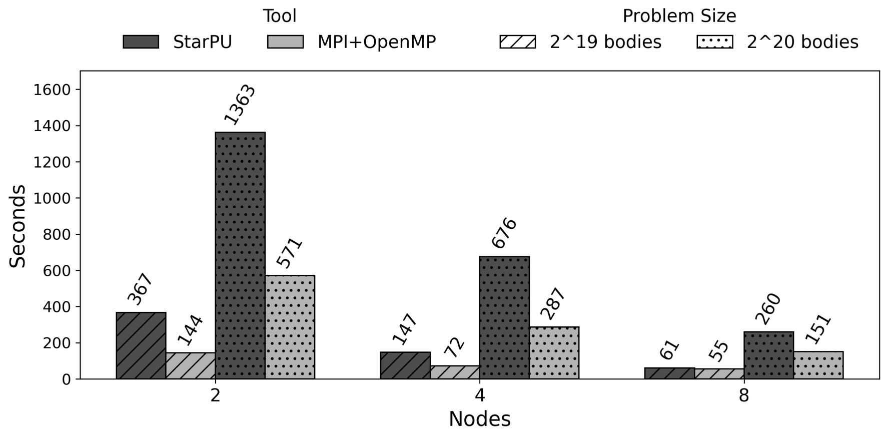

Figure 3. Execution times (in seconds) on clusters with only CPU tasks.

speedups, reducing execution time by approximately 80% to 85% compared to a 32-core CPU-only configuration. This highlights the effectiveness of GPU acceleration for the computationally intensive phases of the simulation.

In the hybrid execution strategy, workloads were divided between CPU and GPU resources. In the StarPU implementation, this distribution was handled dynamically at runtime, based on StarPU's internal scheduling and performance models. In contrast, we employed the OpenMP's `target` directive to offload the most computationally demanding tasks (force calculations) to the GPU, while delegating lighter tasks to the CPU. Interestingly, the hybrid configuration did not yield significant performance gains in either implementation. This result suggests that the overhead associated with executing fine-grained tasks on the GPU (such as data transfer and synchronization costs) offset any potential speedup. Consequently, the overall performance of the hybrid configuration remained comparable to that of the GPU-only setup, indicating that the granularity of the lighter tasks was insufficient to benefit from GPU offloading.

5.2. Multi-node Results

Figure 3 presents the execution times for the CPU-only configuration in a multi-node environment. In this scenario, the MPI+OpenMP implementation demonstrated excellent

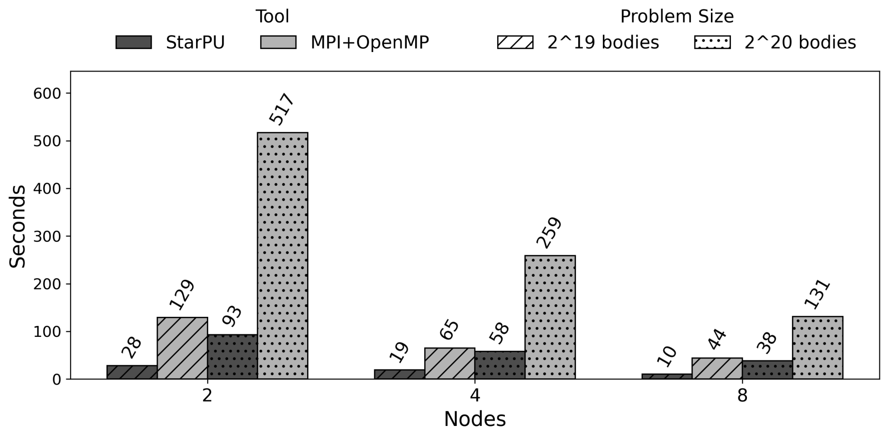

Figure 4. Execution times (in seconds) on clusters with only GPU tasks.

scalability. Conversely, StarPU encountered difficulties in managing the large number of CPU tasks spread across multiple nodes, leading to diminished scalability and increased execution times. Notably, the two-node StarPU configuration exhibited higher execution times than the single-node setup, suggesting that the overhead associated with task distribution and inter-node communication outweighed the benefits of parallelism at this scale.

Figure 4 illustrates the execution times in a multi-node configuration with GPU-only execution. The results highlight that StarPU manages GPU tasks significantly more efficiently than the MPI+OpenMP implementation for the given setup, achieving lower execution times and demonstrating excellent scalability as the number of nodes increases. Notably, the CUDA-based implementation further reduces execution time in the StarPU configuration, leveraging fine-grained kernel optimizations and efficient data management, an advantage not observed in the MPI+OpenMP approach, where the overhead of managing offload directives and data transfers limits performance gains.

Figure 5 presents the execution times in a multi-node configuration with hybrid CPU+GPU execution. The results demonstrate that StarPU was able to reduce in 2% to 10% the execution time by balancing workloads between CPUs and GPUs, dynamically exploiting available resources when possible. In contrast, the MPI+OpenMP implementation showed 1% to 5% higher execution times. This outcome highlights StarPU's ability to determine when and how to utilize CPU resources to minimize execution time when data movement and synchronization become more expensive, such as in multi-node setups, while the rigid task distribution strategy of MPI+OpenMP limits its efficiency in these environments.

6. Related Work

Nylons et al. proposed a CUDA-based implementation of the All-Pairs N-Body algorithm, tackling the inherent O(N^2) computational complexity through a tiling strategy that promotes data locality and minimizes memory bandwidth constraints [Nylons et al. 2007]. Furthermore, they employed loop unrolling and fine-grained control over thread synchronization to further enhance throughput, achieving significant performance gains. Building upon some of the algorithmic techniques, our work extends these concepts to heterogeneous computing environments by employing a task-based dynamic scheduling model that integrates both CPU and GPU resources.

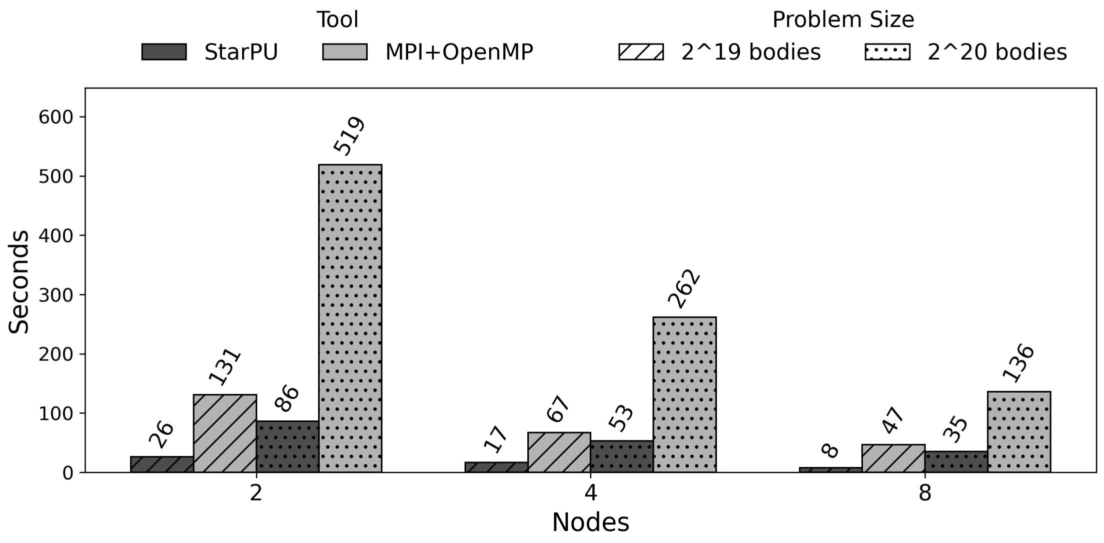

Figure 5. Execution times (in seconds) on clusters with CPUs+GPUs.

Teylo et al. investigated the use of ephemeral public cloud infrastructures to execute bag-of-tasks applications, offering valuable insights into leveraging cost-effective and flexible resources for HPC workloads [Teylo et al. 2023]. Their work primarily addresses embarrassingly parallel applications, which inherently align with the cloud's elastic resource provisioning and exhibit minimal inter-task communication overheads. In a complementary direction, our study also examines an embarrassingly parallel problem, the N-Body simulation, but in a distinct computational context. Specifically, we focus on on-demand cloud instances and employ StarPU, a runtime system that orchestrates computations using a task graph-based model. This approach introduces dynamic scheduling mechanisms to manage task execution across CPU and GPU resources.

Zhuang et al. demonstrated that large-scale Earth science simulations on public cloud platforms can achieve performance and cost metrics comparable to traditional supercomputing environments [Zhuang et al. 2020]. By optimizing MPI configurations and using reproducible workflows with AWS ParallelCluster and Spack, they showcased efficient scaling of the GEOS-Chem model. Their work addresses prior concerns about cloud inefficiency in HPC, focusing on specialized, high-performance configurations.

This research deviates from prior studies by comparing explicitly the dynamic scheduling offered by StarPU to traditional HPC tools in on-demand cloud instances, while avoiding the use of HPC marketed solutions. By isolating the runtime and scheduling behavior of StarPU under these conditions, the research provides a novel perspective on its adaptability and efficiency in on-demand cloud environments, thus contributing to the understanding of task-based parallelism in cloud-based HPC contexts.

7. Conclusion

This work evaluated the effectiveness of task-based dynamic scheduling using StarPU for executing N-Body simulations on AWS, and compared its performance to a traditional MPI+OpenMP implementation. The results showed that StarPU consistently outperformed the MPI+OpenMP approach on a single node, especially when GPU acceleration was utilized for force computations. However, in hybrid CPU+GPU configurations, neither approach yielded additional performance benefits. The overhead associated with offloading fine-grained tasks, such as particle position updates, to the GPU offset any potential gains, resulting in performance comparable to GPU-only execution.

In multi-node CPU-only executions, MPI+OpenMP exhibited excellent scalability, while StarPU faced challenges managing a large volume of distributed CPU tasks, leading to performance degradation. Conversely, in GPU-only and hybrid multi-node scenarios, StarPU outperformed MPI+OpenMP. These results underscore the importance of dynamic scheduling frameworks for hybrid cloud HPC workloads.

Overall, this study demonstrates that task-based parallelism plays a crucial role in effectively utilizing the heterogeneous resources available in public cloud environments. However, it is important to recognize a key trade-off associated with StarPU: while performance model calibration is essential for optimizing task scheduling and improving application efficiency, the calibration process itself incurs computational overhead and additional costs. A practical approach would be to conduct calibration during development phase, store the resulting performance models, and update them only when substantial changes occur in workload characteristics or the underlying cloud infrastructure.

As future work, we plan to expand our performance evaluation to include alternative solutions to the N-Body problem, such as the Barnes-Hut algorithm [Barnes and Hut 1986] and FMM [Wang 2021], which introduce hierarchical approximations to reduce computational complexity and offer new opportunities for parallelization.

Acknowledgements

This work was partially funded by *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq), *Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina* (FAPESC) through the call N^{o} 18/2024 and Amazon Web Services (AWS) through the CNPq/AWS call N^{o} 64/2022 (Cloud Credits for Research).

References

Augonnet, C., Thibault, S., Namyst, R., and Wacrenier, P.-A. (2011). StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23(2):187–198.

Barnes, J. and Hut, P. (1986). A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324(6096):446–449.

Bueno, J., Planas, J., Duran, A., Martorell, X., Ayguadé, E., Badia, R. M., and Labarta, J. (2012). Productive programming of gpu clusters with ompss. In *Parallel and Distributed Processing Symposium (IPDPS)*.

Chandra, R. (2001). *Parallel programming in OpenMP*. Morgan kaufmann.

Dancheva, T., Alonso, U., and Barton, M. (2024). Cloud benchmarking and performance analysis of an HPC application in Amazon EC2. *Cluster Computing*, 27(2):2273–2290.

Graham, R. L., Woodall, T. S., and Squyres, J. M. (2006). Open mpi: A flexible high performance mpi. In *Parallel Processing and Applied Mathematics: 6th International Conference, PPAM 2005, Poznań, Poland, September 11-14, 2005, Revised Selected Papers 6*, pages 228–239. Springer.

Guidi, G., Ellis, M., Buluç, A., Yelick, K., and Culler, D. (2021). 10 years later: Cloud computing is closing the performance gap. In *Companion of the ACM/SPEC International Conference on Performance Engineering*, ICPE '21, page 41–48, New York, NY, USA. Association for Computing Machinery.

Heggie, D. (2005). The classical gravitational n-body problem. *Encyclopedia of Mathematical Physics*.

Hoque, R., Hérault, T., Bosilca, G., and Dongarra, J. (2017). Dynamic task discovery in PaRSEC: a data-flow task-based runtime. In *ScalA '17: The 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*.

Liberman García, A. (2015). *The evolution of the Cloud: the work, progress and outlook of cloud infrastructure*. PhD thesis, Massachusetts Institute of Technology.

Munhoz, V. and Castro, M. (2024). Enabling the execution of hpc applications on public clouds with hpc@cloud toolkit. *Concurrency and Computation: Practice and Experience*, 36(8):e7976.

Netto, M. A. S., Calheiros, R. N., Rodrigues, E. R., Cunha, R. L. F., and Buyya, R. (2018). HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges. *ACM Comput. Surv.*, 51(1).

Nylons, L., Harris, M., and Prins, J. (2007). Fast n-body simulation with cuda. *GPU gems*, 3:62–66.

Pereira, R. (2023). *Efficient Use of Task-based Parallelism in HPC Parallel Applications*. Theses, Ecole normale supérieure de lyon - ENS LYON.

Teylo, L., Arantes, L., Sens, P., and Drummond, L. M. d. A. (2023). Scheduling Bag-of-Tasks in Clouds using Spot and Burstable Virtual Machines. *IEEE Transactions on Cloud Computing*, 11(1):984–996.

Wang, Q. (2021). A hybrid fast multipole method for cosmological n-body simulations. *Research in Astronomy and Astrophysics*, 21(1):003.

Wei, J., Chen, M., Wang, L., Ren, P., Lei, Y., Qu, Y., Jiang, Q., Dong, X., Wu, W., Wang, Q., Zhang, K., and Zhang, X. (2022). Status, challenges and trends of data-intensive supercomputing. *CCF Transactions on High Performance Computing*, 4(2):211–230.

Zhuang, J., Jacob, D. J., Lin, H., Lundgren, E. W., Yantosca, R. M., Gaya, J. F., Sulprizio, M. P., and Eastham, S. D. (2020). Enabling high-performance cloud computing for earth science modeling on over a thousand cores: Application to the geoschem atmospheric chemistry model. *Journal of Advances in Modeling Earth Systems*, 12(5):e2020MS002064. e2020MS002064 2020MS002064.

Zwart, S. F. P., Belleman, R. G., and Geldof, P. M. (2007). High-performance direct gravitational n-body simulations on graphics processing units. *New Astronomy*, 12(8):641–650.