Hybrid MPI + OpenMP Parallelization of the N-Body Problem: Design, Implementation, and Performance Analysis

Caio Reis[1]

[1]Department of Computer Science, Federal University of São João del-Rei, Brazil

December 17, 2025

Abstract

This work presents the design, implementation, and performance evaluation of a hybrid parallel solution for the classical N-Body problem, developed in C++ using the Message Passing Interface (MPI) and OpenMP. The N-Body problem models the gravitational interaction among N particles and is characterized by an $O(N^2)$ computational complexity, making it a representative benchmark for high-performance computing. A sequential baseline is first implemented to validate correctness and identify computational bottlenecks. The parallel solution adopts a hybrid programming model in which MPI is used to distribute the particle set across processes, while OpenMP accelerates force computation loops within each process. Communication is handled through an all-to-all data exchange using `MPI_Allgatherv`, allowing each process to access a consistent global snapshot of particle positions at every simulation step. Performance is evaluated through strong scaling experiments, comparing pure MPI, pure OpenMP, and hybrid configurations under fixed total core counts. The results demonstrate that MPI-based parallelism provides near-linear speedup for moderate process counts, while hybrid configurations offer competitive performance by reducing communication overhead at higher core counts. The study highlights key trade-offs between distributed and shared-memory parallelism and reinforces the effectiveness of hybrid approaches for computationally intensive scientific simulations. **Palavras-chave:** Parallel Computing; Hybrid Programming;

MPI; OpenMP; N-Body Simulation; Performance Analysis.

1 Introduction

The N-Body problem is a fundamental computational challenge in scientific computing, arising in domains such as astrophysics, molecular dynamics, and plasma physics. It consists of computing the forces exerted among N particles and updating their positions over time according to physical laws. The straightforward formulation requires evaluating all pairwise interactions, leading to an $O(N^2)$ computational complexity per simulation step. As the number of particles increases, the problem quickly becomes infeasible for sequential execution.

Parallel computing offers a natural approach to addressing this limitation. By distributing the workload across multiple processing elements, it is possible to significantly

reduce execution time and enable simulations with larger problem sizes. However, efficient parallelization of the N-Body problem presents challenges related to data distribution, communication overhead, and load balancing, particularly in distributed-memory environments.

This work explores a hybrid parallel solution that combines two widely adopted parallel programming paradigms: MPI for distributed-memory parallelism and OpenMP for shared-memory parallelism. The hybrid approach aims to exploit both inter-node and intra-node parallelism, allowing the simulation to scale across multiple processes while efficiently utilizing multi-core architectures.

The objectives of this project are threefold: (i) to design a modular and maintainable hybrid parallel architecture for the N-Body problem, (ii) to implement and validate the correctness of the solution through comparison with a sequential baseline, and (iii) to evaluate the performance of different parallel configurations through systematic benchmarking and scalability analysis.

2 Methodology

The simulation models the motion of particles in a two-dimensional space under mutual gravitational attraction. At each timestep, the force acting on a particle is computed by summing the contributions from all other particles according to Newton's law of universal gravitation. Particle velocities and positions are then updated using the symplectic Euler integration method, which offers improved numerical stability over the standard explicit Euler scheme.

A sequential version of the algorithm is first implemented to serve as a correctness reference and performance baseline. Profiling of the sequential execution confirms that the force computation dominates runtime, accounting for the vast majority of execution time.

For the parallel solution, a hybrid programming model is adopted. The particle set is decomposed into contiguous blocks and distributed among MPI processes using a one-dimensional block decomposition strategy. Each process is responsible for updating a subset of particles, ensuring exclusive write access and avoiding race conditions.

At the beginning of each timestep, all processes exchange their local particle data using `MPI_Allgatherv`, creating a global snapshot of particle positions. This snapshot is used as read-only input during force computation. Within each process, the force calculation loop over local particles is parallelized using OpenMP with static scheduling, as each iteration performs an equivalent amount of work.

Performance evaluation is conducted through strong scaling experiments, where the problem size is held constant while varying the number of MPI processes and OpenMP threads. Execution time is measured using high-resolution timers, excluding input and output operations to focus exclusively on computation and communication costs.

3 Development

The implementation is written in C++17 and organized into modular components to separate concerns and facilitate maintainability. Core data structures represent particles using aligned memory layouts to improve cache efficiency. A custom MPI datatype is defined to allow efficient transmission of particle structures while respecting alignment and padding constraints.

The parallel execution flow consists of three main phases within each timestep: communication, computation, and integration. During the communication phase, MPI collectives are used to synchronize particle positions across processes. The computation phase performs the $O(N^2)$ force calculations, and the integration phase updates particle velocities and positions.

OpenMP directives are applied exclusively to rank-local loops, ensuring that MPI communication is not interleaved with multithreaded regions. This separation simplifies reasoning about correctness and avoids undefined behavior. Static scheduling is employed due to the uniform workload associated with each particle interaction.

Automation scripts are developed to execute standardized benchmark configurations, including sequential, pure MPI, pure OpenMP, and hybrid setups. These scripts generate timing data that is later analyzed using a dedicated post-processing workflow.

4 Results

Performance results are obtained for a fixed problem size of $N = 5000$ particles and a fixed number of simulation steps. Figure 1 shows the strong scaling behavior of the pure MPI implementation, illustrating near-linear speedup as the number of MPI processes increases up to ten ranks.

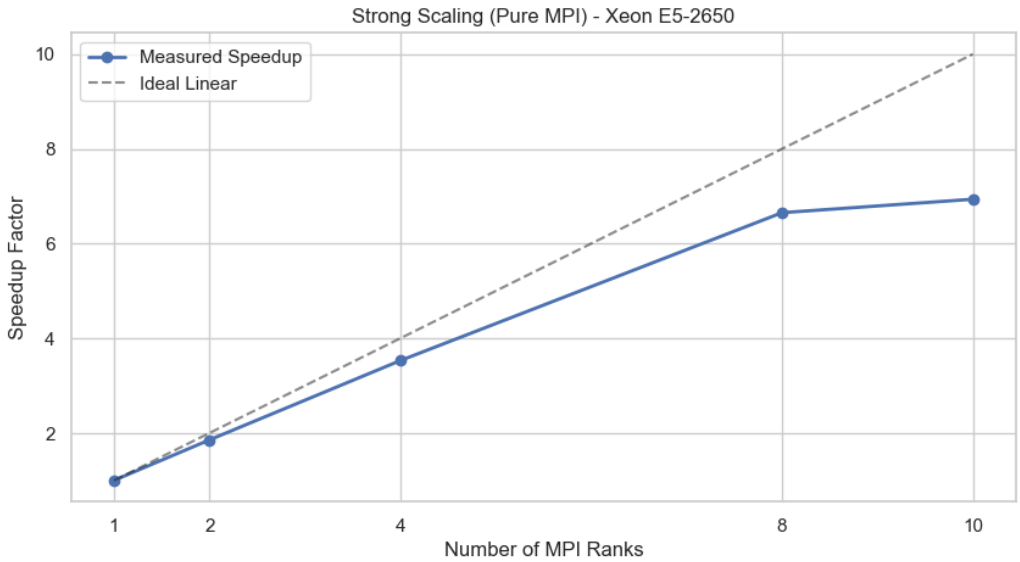

Figure 1: Speedup as a function of the number of MPI processes for the pure MPI configuration.

Figure 2 presents the corresponding parallel efficiency, which gradually decreases as communication overhead becomes more significant at higher process counts.

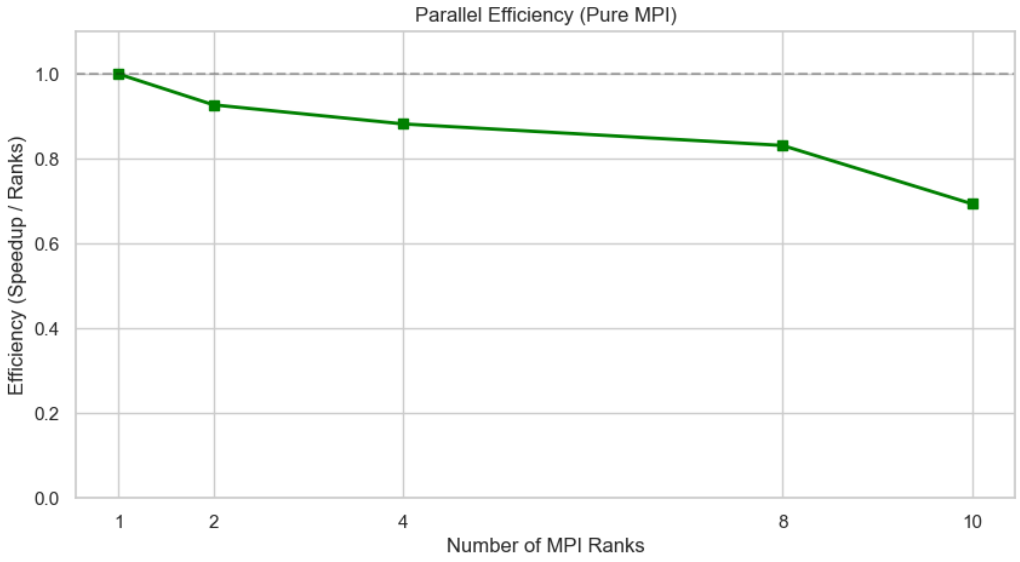

Figure 2: Parallel efficiency of the pure MPI implementation.

Hybrid configurations are evaluated by comparing execution times under fixed total core counts. Figure 3 compares pure MPI, pure OpenMP, and hybrid configurations using ten total cores, while Figure 4 presents the same comparison for four total cores.

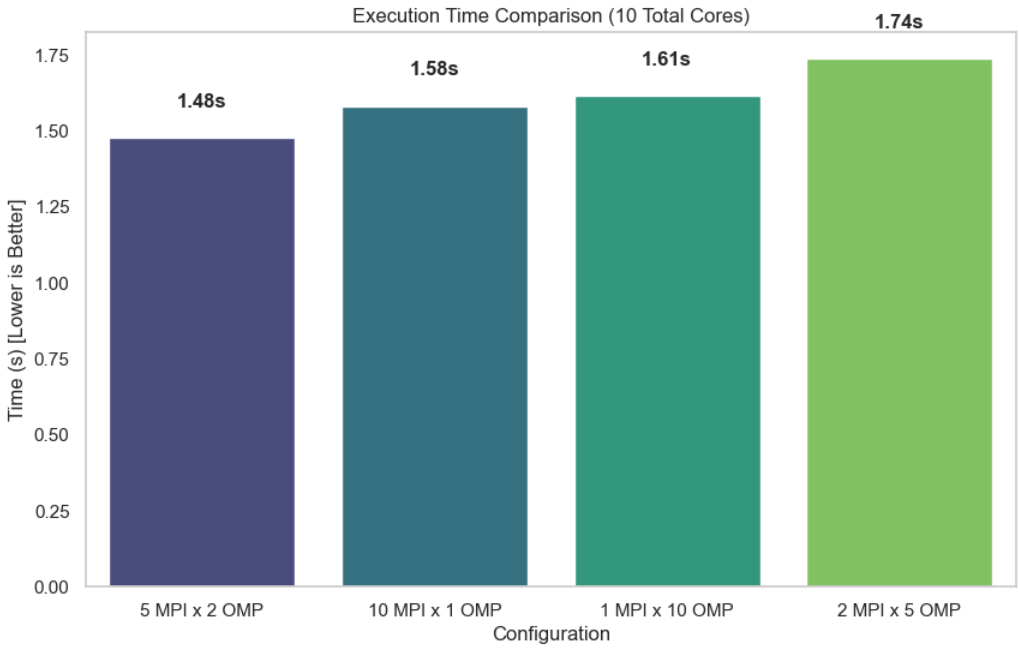

Figure 3: Execution time comparison for different configurations using ten total cores.

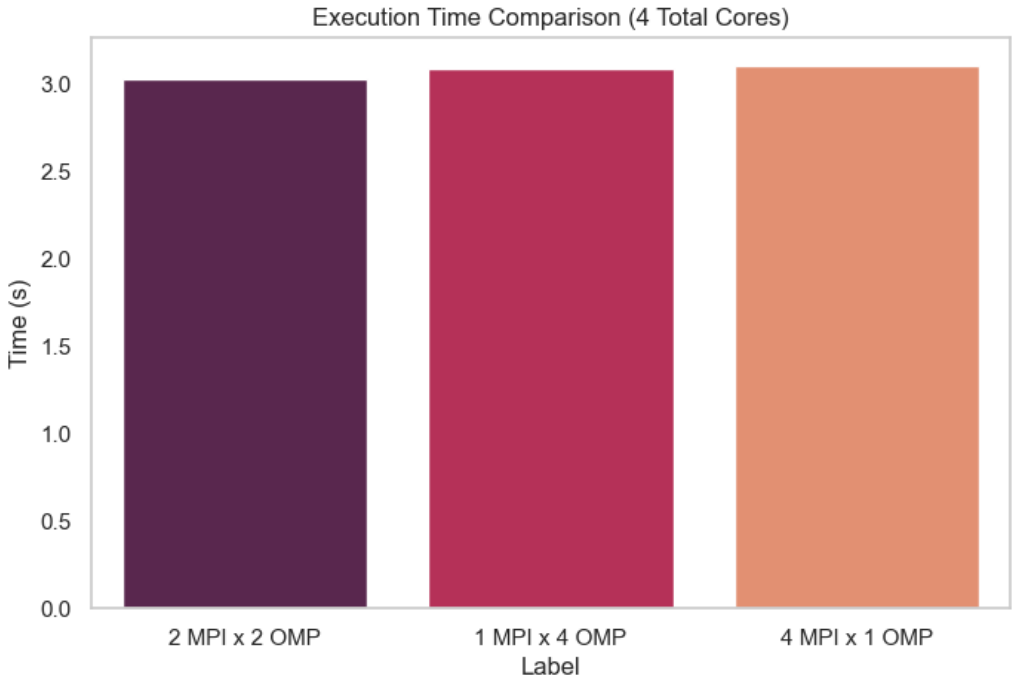

Figure 4: Execution time comparison for different configurations using four total cores.

The numerical results show that pure MPI achieves the highest performance for the tested configurations, while hybrid approaches offer competitive performance by reducing the number of MPI processes involved in collective communication. Pure OpenMP configurations perform adequately at low core counts but exhibit limited scalability due to memory bandwidth constraints.

Overall, the results confirm that hybrid MPI + OpenMP parallelization provides a flexible and effective strategy for the N-Body problem, balancing communication and computation costs on modern multi-core systems.