Hybrid Parallel Computing for the Mandelbrot Set: MPI and OpenMP Integration

By
Muhammad Mubashar Shahzad
[SM3600012]

Report submitted in partial fulfillment of
the requirements for the project of

High Performance Computing

DEPARTMENT OF MATHEMATICS AND GEOSCIENCES

UNIVERSITY OF TRIESTE

2024

TABLE OF CONTENTS

ABSTRACT

This project implements a hybrid MPI and OpenMP approach to compute and visualize the Mandelbrot set on the ORFEO high-performance computing cluster. By distributing tasks across multiple processors with MPI and using multi-threading within each processor through OpenMP, the study evaluates both strong and weak scaling. Results show that while OpenMP strong scaling plateaus due to parallel overhead, OpenMP weak scaling increases execution time proportionally with some inefficiencies. MPI strong scaling remains stable up to a point but faces communication overhead issues beyond a certain number of tasks, and MPI weak scaling reveals a non-linear increase in execution time, indicating decreasing parallel efficiency. These findings offer insights into optimizing hybrid parallel implementations for computationally intensive tasks.

Chapter 1

INTRODUCTION

The Mandelbrot set is a well-known fractal that serves as a benchmark for testing parallel computing strategies. Defined by the iteration of the function

$$f_c(z) = z^2 + c,$$

where both z and c are complex numbers, the set includes points in the complex plane for which the magnitude of z remains bounded over successive iterations. This computational problem is highly suitable for parallel processing, as each point can be computed independently.

This project employs a hybrid approach using Message Passing Interface (MPI) and Open Multi-Processing (OpenMP) to efficiently compute and visualize the Mandelbrot set. By utilizing MPI for task distribution across multiple processors and OpenMP for threading within each processor, we aim to optimize the computation and reduce execution time. Our performance evaluation includes both strong and weak scaling experiments conducted on the ORFEO high-performance computing cluster.

Strong scaling evaluates the performance as the number of processors increases with a fixed problem size, while weak scaling assesses the performance when both the number of processors and the problem size increase proportionally. These experiments provide insights into the efficiency and scalability of the hybrid implementation, helping to optimize the computation of the Mandelbrot set.

Mandelbrot Set Visualization

To visualize the Mandelbrot set, each point in the complex plane is mapped to a pixel in a 2D grid. Points that belong to the Mandelbrot set are colored black, while other points are shaded according to the number of iterations required to determine that they do not belong to the set.

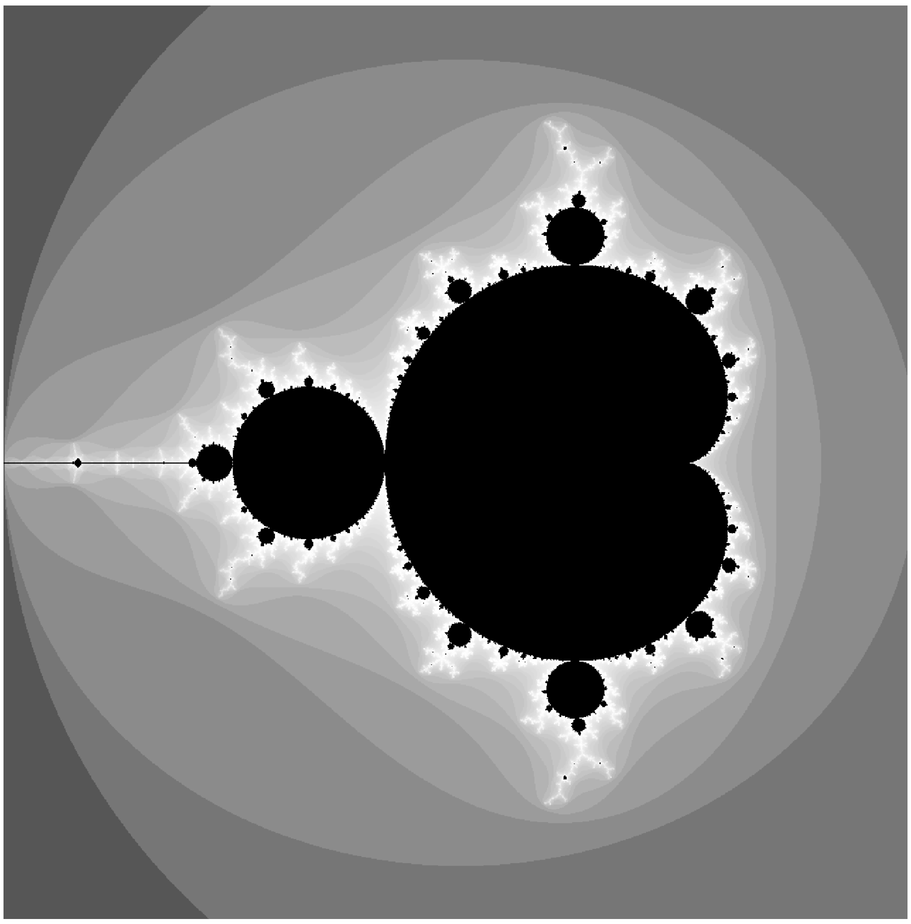

Figure 1-1: A visualization of the Mandelbrot set, displaying the intricate fractal pattern characterized by self-similarity and complex boundaries. The black regions represent points that remain bounded under iteration, while the varying shades of gray illustrate the rate at which points escape to infinity.

Chapter 2

METHODOLOGY

2.1 Computational Architecture

The project utilizes the ORFEO cluster, particularly the THIN partition, which consists of nodes equipped with Intel Xeon Gold CPUs, each with 12 cores. The nodes are interconnected via a high-speed network, allowing efficient data transfer. This setup supports both MPI for distributed memory parallelism and OpenMP for shared memory parallelism, providing a robust environment for hybrid parallel computing.

2.2 Mandelbrot Set Implementation

The Mandelbrot set is computed and visualized using a hybrid MPI and OpenMP approach. Each point in the complex plane is computed independently, making it suitable for parallel processing. The implementation involves:

1. **MPI Parallelization**: Distributing the workload across multiple processes, each handling a section of the complex plane.
2. **OpenMP Parallelization**: Further dividing the workload within each process among multiple threads.

The core components include:

- **mandelbrot function**: Determines if a point belongs to the Mandelbrot set by iterating the function $f_c(z) = z^2 + c$.
- **write_pgm_image function**: Generates a PGM image from the computed matrix representing the Mandelbrot set.
- **Main function**: Coordinates the computation by parsing command-line arguments, initializing MPI, distributing tasks, and gathering results.

2.3 Experiment Plan

The experiments are designed to evaluate both strong and weak scaling:

1. **Strong Scaling**: Keeping the problem size constant while varying the number of processing units to measure how the solution time decreases.
2. **Weak Scaling**: Increasing both the problem size and the number of processing units proportionally to measure how the solution time changes.

Scripts used:

- **strong_scaling_openmp.sh**: Evaluates performance by increasing the number of OpenMP threads with a single MPI task.
- **strong_scaling_mpi.sh**: Assesses performance by increasing the number of MPI tasks with one OpenMP thread per task.
- **weak_scaling_openmp.sh**: Examines performance by scaling the problem size with the number of OpenMP threads.
- **weak_scaling_mpi.sh**: Evaluates performance by scaling the problem size with the number of MPI tasks.

Chapter 3

RESULTS AND DISCUSSION

3.1 MPI Strong Scaling

The strong scaling plot for MPI (Figure 3-1) shows a non-monotonic relationship between execution time and the number of MPI tasks. Initially, the execution time decreases as the number of tasks increases, indicating effective load distribution and parallel efficiency. However, after a certain point, the execution time increases sharply, suggesting communication overhead and synchronization issues start to dominate, reducing the efficiency gains. At the highest number of tasks, the execution time decreases again, likely due to an optimized use of available resources or a reduction in task contention. These fluctuations highlight the need for careful balancing between computational load and communication overhead in MPI applications.

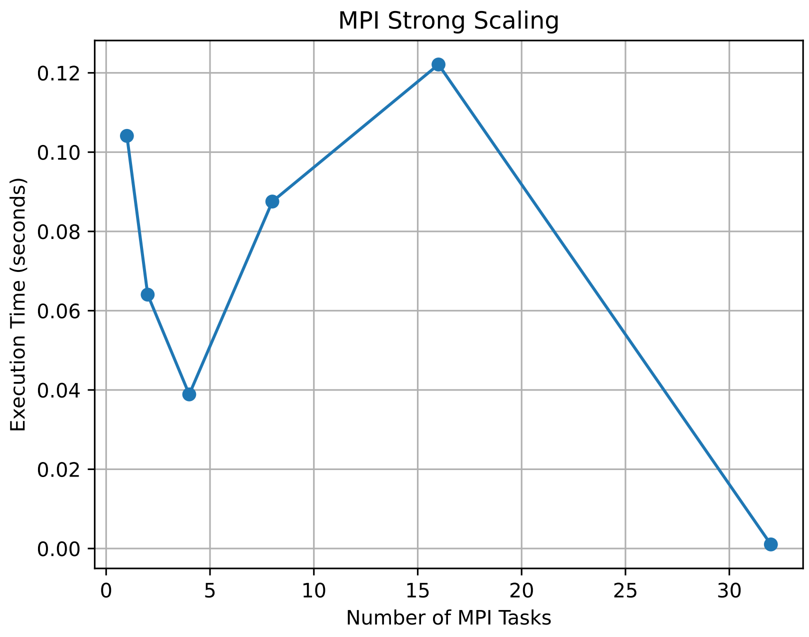

Figure 3-1: The graph illustrates the execution time versus the number of MPI tasks. The execution time initially decreases, then exhibits an increase before decreasing again, indicating varying efficiency levels with different task counts.

3.2 MPI Weak Scaling

The weak scaling results for MPI (Figure 3-2) reveal a steady increase in execution time as the number of MPI tasks and the problem size both grow. This linear increase suggests that the MPI implementation scales proportionally with the problem size, but the increase in execution time indicates that communication overhead and synchronization costs rise with the number of tasks. The non-linear segments in the graph further emphasize inefficiencies that might be due to uneven load distribution or increased latency in communication as more nodes are involved. This trend underscores the challenges in maintaining parallel efficiency when scaling up both tasks and problem size in MPI.

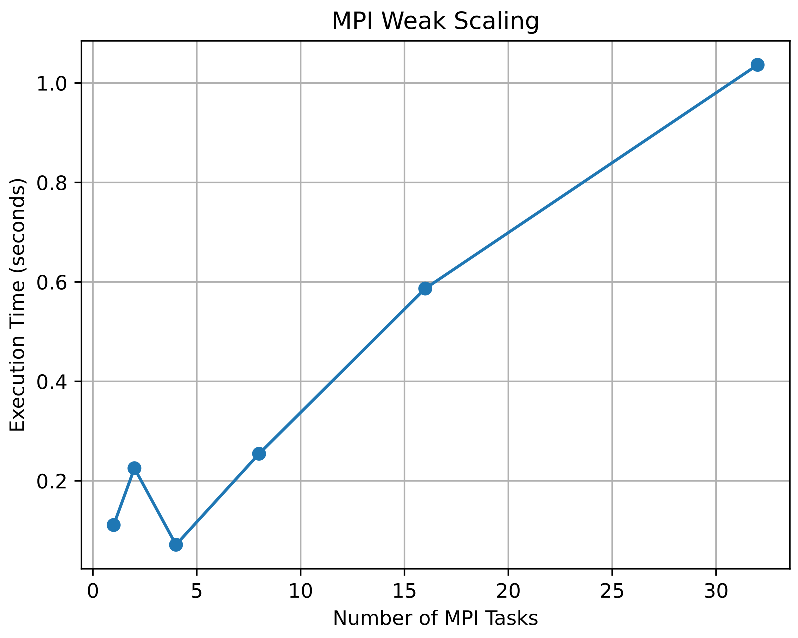

Figure 3-2: The plot shows execution time against the number of MPI tasks with a proportionally increased problem size. The execution time rises steadily, reflecting the increasing computational demand with additional tasks.

3.3 OpenMP Strong Scaling

For OpenMP strong scaling (Figure 3-3), the execution time significantly drops as the number of threads increases, showing the benefits of parallel computation. The execution time plateaus after reaching a certain number of threads, which indicates that additional threads beyond this point do not contribute to further performance gains. This plateau is likely due to factors such as memory bandwidth limitations or increased overhead from managing a larger number of threads. The initial rapid decrease in execution time suggests that OpenMP effectively utilizes the multi-core architecture up to an optimal thread count.

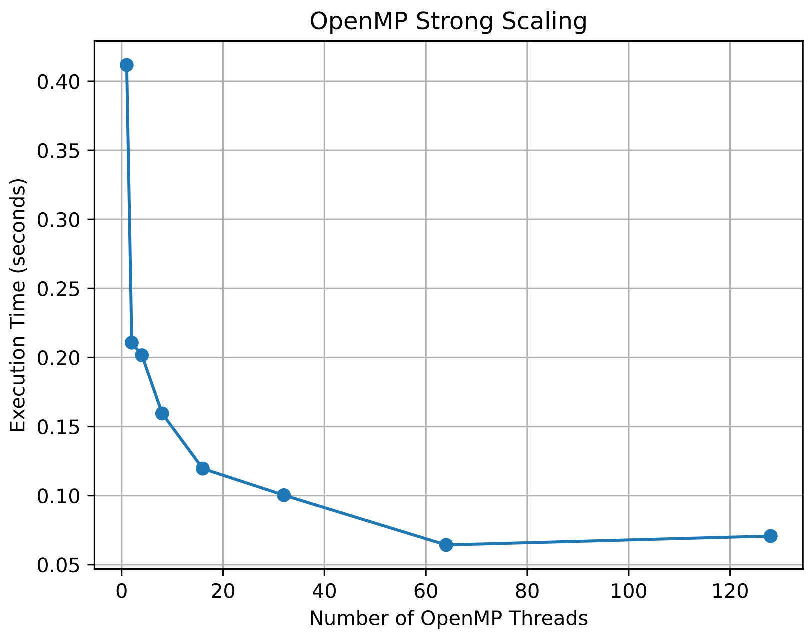

Figure 3-3: The graph depicts execution time versus the number of OpenMP threads. There is a significant decrease in execution time as the number of threads increases, stabilizing after a certain point, which indicates efficient parallelization up to a threshold.

3.4 OpenMP Weak Scaling

The OpenMP weak scaling plot (Figure 3-4) demonstrates a linear increase in execution time as the number of threads and the problem size both increase. This behavior indicates that while the computation scales with the number of threads, the execution time rises proportionally due to the increased workload. The linearity of the increase implies that OpenMP maintains a consistent level of parallel efficiency across different thread counts, but it also highlights that the overall computational demand grows with the problem size, thus leading to longer execution times.

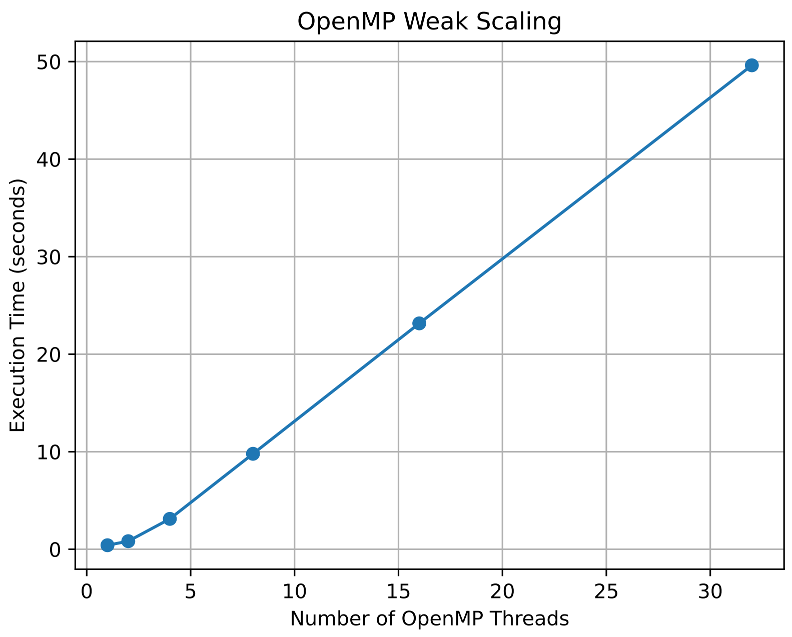

Figure 3-4: The chart represents execution time against the number of OpenMP threads with a proportionally increased problem size. The linear increase in execution time demonstrates the computational demand scaling with thread count.

3.5 Conclusion

The analysis of MPI and OpenMP scaling experiments reveals critical insights into the performance characteristics of hybrid parallel computing. MPI strong scaling shows varying efficiency with different task counts, emphasizing the importance of balancing load distribution and communication overhead. MPI weak scaling indicates that communication costs become significant as both tasks and problem size increase. OpenMP strong scaling demonstrates effective use of multi-core processors up to an optimal thread count, while OpenMP weak scaling shows consistent parallel efficiency but increased computational demand with larger problem sizes. These findings highlight the complexities and trade-offs in optimizing parallel computations for high-performance computing environments. Future work should focus on refining load balancing strategies and minimizing communication overhead to enhance scalability and performance further.