High-Performance Computing – Exercise 2c

Hybrid Parallel Mandelbrot Set Computation: MPI + OpenMP Scaling Analysis

Tanja Derin [SM3800013]
Data Science and Artificial Intelligence [346SM]
Supervised professor: Luca Tornatore
July 2025

Abstract

The Mandelbrot set serves as a classical benchmark for evaluating parallel computing techniques due to the parallel structure of its computation. This project presents a hybrid MPI + OpenMP implementation designed to generate high-resolution images of the Mandelbrot set. The computation is distributed across MPI processes, with each process utilizing OpenMP threads to compute pixel values independently. Strong and weak scaling tests were performed on the EPYC nodes of the Orfeo cluster to assess performance. The results highlight effective scalability, with OpenMP providing efficient intranode parallelism and MPI ensuring distributed load balancing across nodes. The hybrid approach demonstrates both computational efficiency and adaptability for high-performance environments.

1 Introduction

The Mandelbrot set is a fractal structure defined over the complex plane, characterized by the iterative function $f_c(z) = z^2 + c$, where z and c are complex numbers. A point c is considered to belong to the Mandelbrot set if the magnitude of the sequence generated from $z_0 = 0$ remains bounded after a finite number of iterations. In practical applications, a cutoff threshold $I_{\max}$ is introduced to determine membership, alongside a condition that terminates iteration when $|z_n| > 2$.

The problem of computing the Mandelbrot set is embarrassingly parallel, as the value associated with each pixel in the image can be computed independently. This characteristic makes it a suitable candidate for evaluating parallel programming paradigms, particularly hybrid models combining message passing and shared memory approaches.

This project implements a hybrid MPI + OpenMP solution to compute high-resolution images of the Mandelbrot set. The implementation follows a static domain decomposition strategy under the SPMD (Single Program Multiple Data) model. Within each process, OpenMP threads parallelize the pixel computations across available CPU cores. The resulting matrix of pixel intensities is saved as a PGM file, which can be visualized to verify correctness.

The objective is to demonstrate the effectiveness of hybrid parallelism in solving computationally intensive tasks, and to evaluate the scalability of the proposed approach through strong and weak scaling tests conducted on a multi-node high-performance computing cluster.

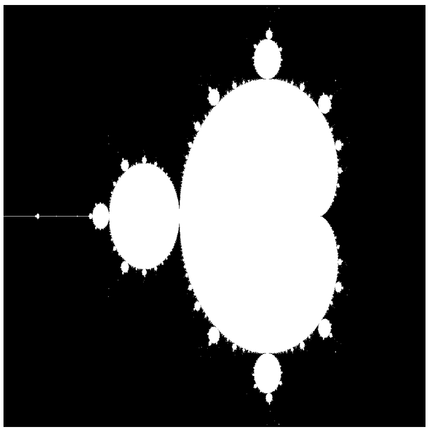


Figure 1: Mandelbrot set output

2 Hybrid Parallel Implementation

The implementation adopts a hybrid parallel programming model using `MPI` (Message Passing Interface) for inter-process communication and `OpenMP` for intra-process parallelism. This design aims to leverage both distributed-memory and shared-memory parallelism to maximize performance on multi-node clusters.

2.1 MPI Decomposition

The 2D domain representing the Mandelbrot image is decomposed row-wise across MPI processes. Each process computes a contiguous block of rows. When the total row count is not divisible evenly, the remainder is distributed to the first few processes to ensure balanced workload:

```
int rows_per_process = ny / size;
int remainder = ny % size;
int start_row = rank * rows_per_process + (rank < remainder ? rank : remainder)
    ;
int local_rows = rows_per_process + (rank < remainder ? 1 : 0);
```

Listing 1: Row-wise MPI decomposition with remainder handling

2.2 Parallel File Output with MPI-IO

Each process writes its block directly to a shared output file using `MPI_File_write_at`. The file header is written only by rank 0, followed by a barrier to synchronize all processes:

```
if (rank == 0) {
    header_size = snprintf(header, sizeof(header), "P5\n%d %d\n%d\n", xsize,
        ysize, maxval);
    MPI_File_write(file, header, header_size, MPI_CHAR, &status);
}
MPI_Bcast(&header_size, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);

MPI_Offset offset = header_size + (MPI_Offset)(rank * local_rows * xsize);
MPI_File_write_at(file, offset, image, local_rows * xsize, MPI_UNSIGNED_CHAR, &
    status);
```

Listing 2: MPI parallel file write with offset after header

2.3 OpenMP Thread-Level Parallelism

Within each MPI process, the Mandelbrot pixel values are computed using OpenMP. The outer loop over rows is parallelized, and dynamic scheduling is applied to improve load balance across threads:

```
#pragma omp parallel for schedule(dynamic)
for (int j = 0; j < local_rows; j++) {
    for (int i = 0; i < nx; i++) {
        double cx = xL + i * dx;
        double cy = yL + (start_row + j) * dy;
        int iter = compute_mandelbrot(cx, cy, Imax);
        local_image[j * nx + i] = (unsigned char)(iter == Imax ? 255 : 0);}}
```

Listing 3: Pixel-wise Mandelbrot computation with OpenMP

2.4 Timing and Synchronization

Compute and I/O phases are timed using `MPI_Wtime()` and synchronized with `MPI_Barrier()` to ensure consistency:

```
MPI_Barrier(MPI_COMM_WORLD);
double start_compute = MPI_Wtime();
// Computation region
double end_compute = MPI_Wtime();
```

Listing 4: Timing critical regions with MPI

2.5 SPMD Execution and Scalability

The implementation adheres to the Single Program Multiple Data (SPMD) model, where all MPI processes execute the same program code but operate on different subsets of the data. In this case, each process is assigned a unique portion of the image rows to compute based on its rank and the total number of processes. While the root process (`rank 0`) handles additional tasks such as writing the image header and aggregating timing results, the computational logic remains uniform across all ranks.

The hybrid parallel approach leverages both inter-node and intra-node parallelism. Inter-node scaling is achieved through MPI, which partitions the image domain across nodes, while intra-node parallelism is provided by OpenMP, which distributes pixel computations among threads within each process. This design enables flexible scalability: the number of MPI ranks and OpenMP threads can be independently adjusted to test different hardware configurations.

3 Performance Measurement and Testing Setup

All performance tests were carried out on the EPYC partition of the ORFEO HPC cluster. This partition consists of 8 compute nodes, each equipped with two AMD EPYC 7H12 CPUs, providing a total of 128 cores and 512 GiB of DDR4 memory per node. The nodes are interconnected via a 100 Gb/s network, and each CPU exposes multiple NUMA domains to the operating system.

The hybrid MPI + OpenMP implementation was tested in both strong and weak scaling scenarios. OpenMP runs were executed within a single node using up to 128 threads, while MPI runs scaled across multiple nodes with up to 256 processes. Each job was submitted through the Slurm workload manager with exclusive resource allocation to ensure consistent performance.

Performance metrics such as compute time, I/O time, and total execution time were measured using `MPI_Wtime()` and aggregated using `MPI_Reduce` to capture the longest timings across all processes. These values were stored in CSV format for further analysis and visualization.

To interpret the scalability trends observed during strong and weak scaling, two classical performance models were used: Amdahl's Law and Gustafson's Law. Amdahl's Law evaluates the theoretical limit of speedup in strong scaling scenarios under the assumption of a fixed problem size and a constant serial fraction f. The speedup $S(p)$ with p processors is given by:

$$S(p) = \frac{1}{f + \frac{1-f}{p}},$$

where f denotes the fraction of the workload that is strictly serial.

Conversely, Gustafson's Law models weak scaling performance under increasing workload, offering a more realistic measure of parallel efficiency when the problem size scales with the number of processors. The corresponding speedup expression is:

$$S(p) = p - f \cdot (p - 1),$$

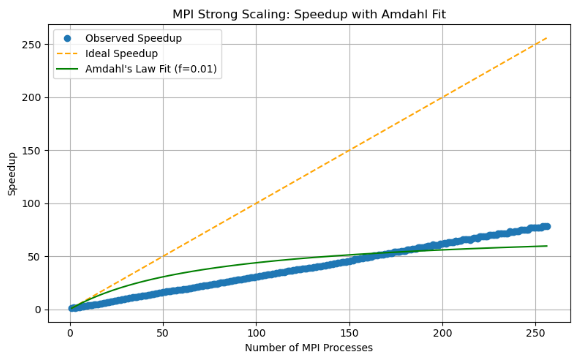
where f again represents the non-parallelizable portion of the computation.

Both models were fitted to the experimental data and are discussed in detail in the following results sections.

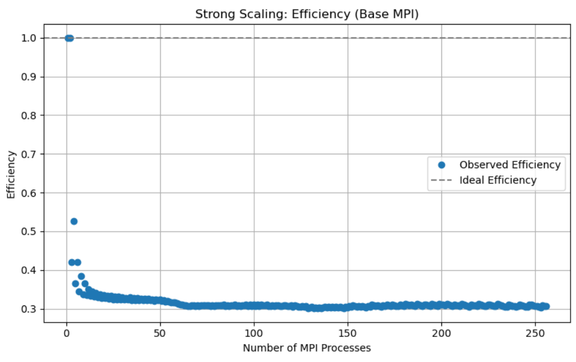
4 Strong Scaling Results

This section presents the strong scaling performance of the hybrid Mandelbrot implementation using both MPI and OpenMP. Experiments were executed on the `EPYC` partition of the ORFEO cluster. MPI tests were run across two nodes using up to 256 processes, while OpenMP experiments were confined to a single node with up to 128 threads.

4.1 MPI Strong Scaling



(a) Speedup with Amdahl's Law Fit



(b) Strong Scaling Efficiency

Figure 2: MPI Strong Scaling Results

The MPI speedup curve in Figure 2a exhibits a sublinear trend, increasingly deviating from the ideal line as the number of processes grows. The fitted Amdahl curve indicates a serial fraction of $f = 0.01$, which, despite being small, imposes a significant theoretical limit on scalability at higher process counts.

Figure 2b shows that efficiency declines steadily with the number of MPI processes, from near-perfect values at small scales to approximately 30% at 256 processes. This degradation is attributed to communication overhead, synchronization delays, and possible workload imbalance across distributed ranks.

4.2 OpenMP Strong Scaling

The OpenMP implementation demonstrates nearly ideal speedup across all tested thread counts (Figure 3). The fitted Amdahl's Law curve yields a serial fraction close to zero, suggesting that the workload is highly parallelizable within a single node and that the implementation scales efficiently in a shared memory environment.

As shown in Figure 4, efficiency remains consistently above 97%, with only minor drops at higher thread counts. These fluctuations are likely due to NUMA effects, thread contention, or operating system-level scheduling noise.

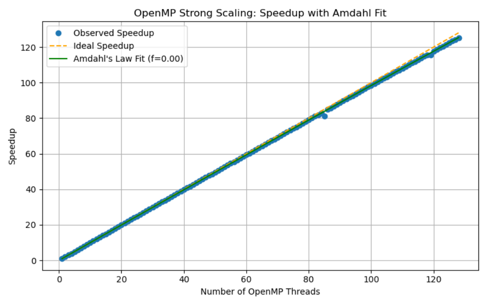


Figure 3: OpenMP Strong Scaling: Speedup with Amdahl's Law Fit

The strong scaling results highlight the contrasting scalability of MPI and OpenMP. While MPI encounters limitations due to internode communication overhead, OpenMP achieves near ideal speedup within a single node. These trends underscore the strengths of the hybrid model in combining inter- and intra- node parallelism. The Amdahl's Law fits quantify the serial bottlenecks and help visualize the theoretical speedup limits of each approach.

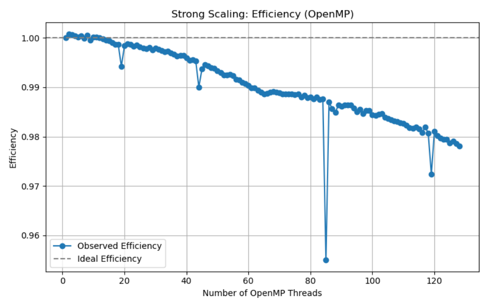


Figure 4: OpenMP Strong Scaling: Efficiency

5 Weak Scaling Results

Weak scaling tests were also conducted for both
MPI and OpenMP implementations on the `EPYC` partition of the ORFEO cluster. The workload was scaled proportionally with the number of processing units, keeping the computational load per core or thread constant. This setup allowed for evaluating scalability under increasing problem sizes.

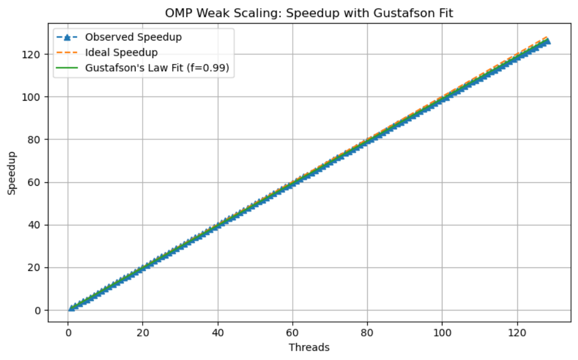
For each configuration with p MPI processes or OpenMP threads, the image resolution was set to $\sqrt{p \cdot C} \times \sqrt{p \cdot C}$, with $C = 10^6$ representing the number of pixels per core or thread. This ensures that each processing unit consistently handles one million pixels, regardless of scale.

For example:

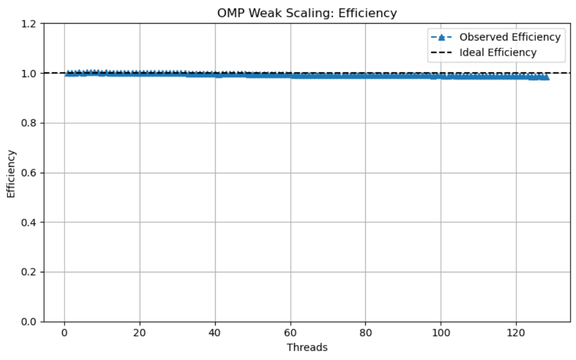
- 1 process/thread computes a 1000×1000 image,
- 4 processes/threads compute a 2000×2000 image,
- 256 processes/threads compute a 16000×16000 image.

All runs were performed with a maximum iteration count $I_{\max} = 255$, and strong binding and placement were used to ensure consistent performance.

5.1 OpenMP Weak Scaling



(a) Speedup with Gustafson's Law Fit ($f = 0.99$)



(b) Weak Scaling Efficiency

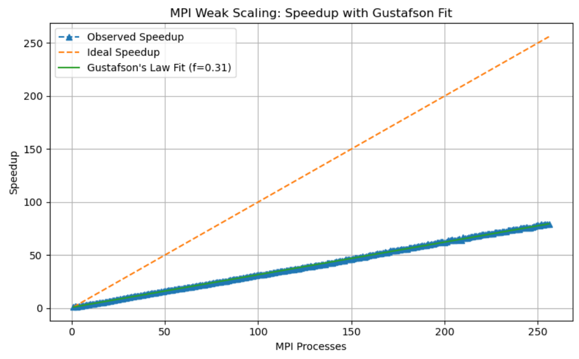
Figure 5: OpenMP Weak Scaling Results

The OpenMP implementation exhibits near ideal weak scaling. The observed speedup aligns closely with the ideal trend and the Gustafson's Law fit, which yields a parallel fraction of $f = 0.99$.

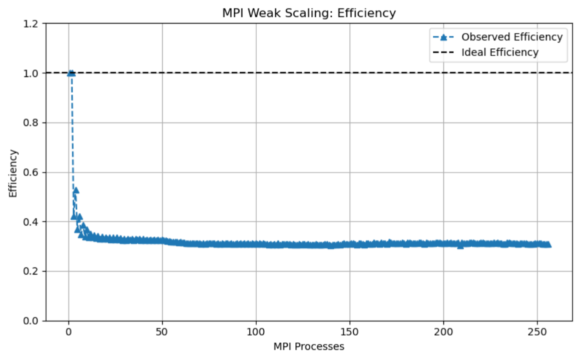
Efficiency remains consistently high across the tested range, confirming minimal parallel overhead and excellent resource utilization.

5.2 MPI Weak Scaling

In contrast, the MPI implementation shows more modest scalability. The fitted parallel fraction of $f = 0.31$ indicates a significant impact from communication overhead and potential load imbalance. Efficiency drops steadily with increasing process count, falling below 35% at 256 processes.



(a) Speedup with Gustafson's Law Fit ($f = 0.31$)



(b) Efficiency

Figure 6: MPI Weak Scaling Results

The results confirm that the OpenMP implementation scales more efficiently under weak scaling conditions. Its high parallel fraction and near-constant efficiency reflect low synchronization costs and effective shared-memory parallelism. The MPI version, while still scalable, suffers from increased communication latency and task imbalance as more nodes and processes are introduced. These observations are well captured by the Gustafson's Law fits, offering quantitative insights into the scalability limits of each approach.

6 Potential Improvements: Dynamic Load Balancing

Although the presented implementation achieves good parallel performance, particularly with OpenMP, further improvements to the MPI component could enhance scalability and efficiency. A good approach is the adoption of a dynamic master–worker strategy, where the master process handles scheduling by dynamically assigning row blocks to worker processes in response to their availability. Each worker sends a work request when idle, computes a block of rows using OpenMP, and overlaps communication with computation via non-blocking messaging. This adaptive task distribution mitigates load imbalance, avoids idle time, and reduces communication bottlenecks—factors that can significantly degrade MPI efficiency at scale.

In addition, prefetching work requests during computation and using fine-grained scheduling within OpenMP further contribute to improved utilization. These enhancements, if integrated into the current implementation, could help overcome the scalability limitations observed in the strong and weak scaling tests and reduce the impact of MPI overhead across multiple nodes.

7 Conclusion

The hybrid MPI + OpenMP implementation leveraged distributed and shared memory parallelism, achieving good performance in both strong and weak scaling scenarios. OpenMP exhibited near-ideal scaling behavior within a single node, with speedup trends closely matching the theoretical maximum and efficiency consistently above 97%. MPI, on the other hand, showed more limited scalability at higher process counts, primarily due to increasing communication overhead and synchronization delays.

The observed scaling trends were effectively captured using classical models:

- Amdahl's Law highlighted the performance ceiling in strong scaling, revealing how even a small serial fraction can constrain speedup;
- Gustafson's Law provided a more realistic assessment in weak scaling, emphasizing how parallelism can remain effective as workload scales with resources.

Overall, the models offered valuable interpretability for identifying bottlenecks and understanding the limits of parallel efficiency.