103084 - High Performance Computing
Hybrid OMP-MPI Mandelbrot Fractal

Jordi Rafael Lazo Florensa

18 June 2023

Master's degree in Computer Engineering

Universitat de Lleida
Escola Politècnica Superior

Contents

List of Figures

List of Tables

1 Introduction

The file named *mandelbrot-seq.c* is a C program that generates an image of the Mandelbrot set. The Mandelbrot set is a complex mathematical set that exhibits a repeating pattern of fractals at different scales.

The program uses nested for-loops to iterate over each pixel in the image. For each pixel, it computes a corresponding complex number by mapping the pixel's coordinates to the complex plane. It then performs a series of iterations using the Mandelbrot formula to determine whether the complex number is part of the Mandelbrot set or not. If the number is part of the set, it is colored black, and if it is not part of the set, it is colored with varying shades of blue based on how many iterations were required to determine that it was not part of the set.

So in this assignment a parallel version will be implemented using MPI (Message Passing Interface) and OpenMP to distribute the work across multiple processes and threads.

2 Scalability of the Program

The scalability of a parallel program refers to its ability to efficiently utilize additional computational resources as the problem size or number of processing units increases. In this implementation, the scalability can be analyzed based on the usage of MPI and OpenMP.

1. MPI Scalability:
    - The code distributes the workload across multiple processors using MPI. Each processor calculates a portion of the fractal image independently.
    - As the number of processors (size) increases, the workload is divided into smaller portions, and each processor computes a smaller portion of the image.
    - In theory, the MPI approach can scale well with the increasing number of processors, as each processor operates independently without requiring much communication with other processors.
    - However, the scalability may be limited by the communication overhead incurred when sending and receiving data between processes.
    - It's important to note that the scalability of the MPI implementation can be affected by factors such as the size of the problem and the communication pattern.
2. OpenMP Scalability:
    - Within each MPI process, the code utilizes OpenMP to parallelize the computation loop, distributing the work among multiple threads.
    - The *#pragma omp parallel for* directive parallelizes the inner loop of the x-coordinate calculations, allowing multiple threads to perform computations concurrently.
    - The parallel region created by OpenMP allows for efficient utilization of multiple cores or processors within each MPI process.

- The scalability of the OpenMP implementation depends on factors such as the number of available cores, the workload distribution, and the synchronization overhead among threads.
- If the available cores can handle the increased number of threads efficiently, the OpenMP implementation can contribute to improved scalability within each MPI process.

3 Obtained Results

3.1 Execution time

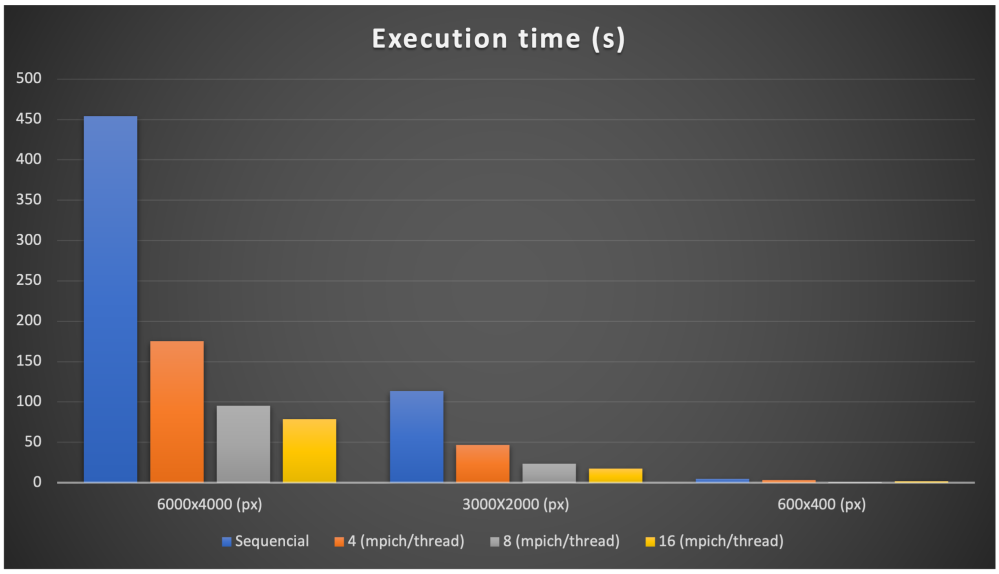

Figure 1: Execution time of all mandelbrot hybrid versions

Execution time (s)				
	Sequencial	4(mpich/threads)	8(mpich/threads)	16(mpich/threads)
6000x4000 (px)	453,98	175,12	95,40	78,89
3000X2000 (px)	113,52	46,84	23,66	17,70
600x400 (px)	4,66	3,48	0,92	1,61

Table 1: Execution time of Mandelbrot version using MPI and OMP

The Table 1 and Figure 1 demonstrates the impact of utilizing MPI and OpenMP in parallelizing the computation and the corresponding reduction in execution times for different problem sizes. The program shows good scalability for the larger problem sizes, but the scalability might be affected by overhead or contention issues for the smallest problem size with 16 processors.

3.2 Speed up

Speedup is calculated as the ratio of the sequential execution time to the parallel execution time. A speedup greater than 1 means that the parallel version is faster than the sequential version and as can be seen in Figure 2, this is true for the results obtained using 4 and 8 processes.

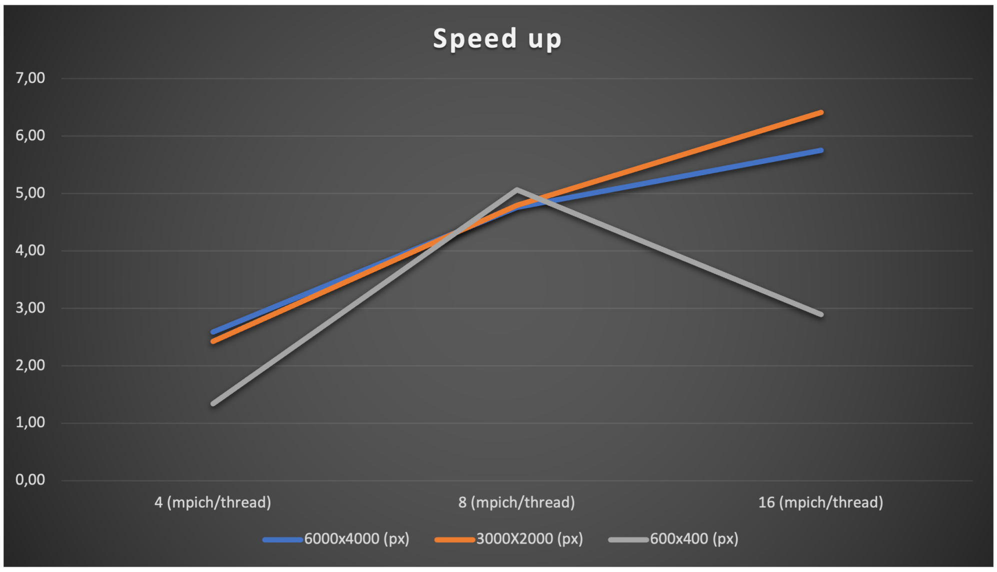

Figure 2: Speed up of the different executions

SpeedUp			
	4 (mpich/threads)	8 (mpich/threads)	16 (mpich/threads)
6000x4000 (px)	2,59	4,76	5,75
3000X2000 (px)	2,42	4,80	6,41
600x400 (px)	1,34	5,07	2,89

Table 2: Speed up of Mandelbrot version using MPI and OMP

Looking at the Table 2 and Figure 2 show the speedup values for different problem sizes and numbers of processors using MPI and OpenMP. The speedup values indicate the improvement in performance achieved by utilizing parallel processing. The program demonstrates good speedup for the larger problem sizes but shows some limitations in achieving optimal speedup for the smallest problem size with 16 processors.

3.3 Efficiency

The efficiency values indicate the relative performance achieved per processor. Higher efficiency values indicate better utilization of resources and more effective parallelization.

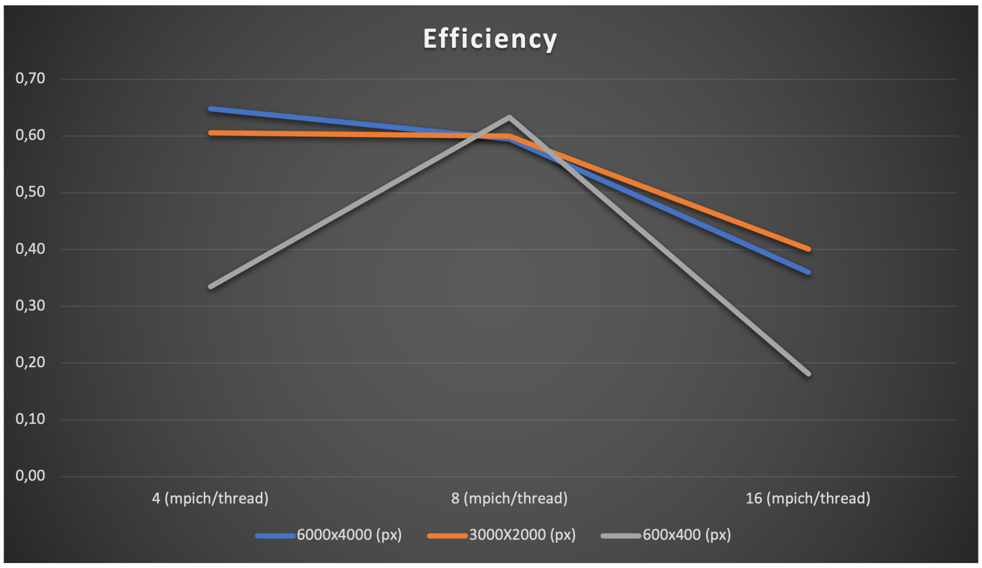

Figure 3: Efficiency of the different executions

Efficiency			
	4 (mpich/threads)	8 (mpich/threads)	16 (mpich/threads)
6000x4000 (px)	0,65	0,59	0,36
3000X2000 (px)	0,61	0,60	0,40
600x400 (px)	0,33	0,63	0,18

Table 3: Efficiency of Mandelbrot version using MPI and OMP

The Table 3 and Figure 3 values indicate the relative performance achieved per processor. The program demonstrates diminishing returns in terms of efficiency for the largest problem size, reasonable efficiency for the moderate problem size, and scalability limitations with reduced efficiency for the smallest problem size.

4 Comparison of the three versions

Execution time (s) (OMP)			
	2 (threads)	4 (threads)	8 (threads)
6000x4000 (px)	335	300	172
3000X2000 (px)	83	75	44
600x400 (px)	3	3	2

Table 4: Execution time of the parallel version of Mandelbrot fractal using OMP

The Table 4 we can observe:

- Similar to the hybrid and MPI approaches, there is a reduction in execution time when moving from sequential execution to the OpenMP approach with 2 processors.

- As the number of processors increases from 2 to 4 and 8, there is further reduction in execution time for all problem sizes. This demonstrates the scalability of the OpenMP approach.
- However, compared to the hybrid and MPI approaches, the reduction in execution time is generally smaller, indicating that the combination of MPI and OpenMP provides better performance for the given problem sizes.

Execution time (s) (MPI)			
	2 (mpich)	4 (mpich)	8 (mpich)
6000x4000 (px)	466,94	270,91	294,74
3000X2000 (px)	117,08	59,47	63,83
600x400 (px)	4,68	2	1,59

Table 5: Execution time of the parallel version of Mandelbrot fractal using MPI

In the Table 5 we can observe:

- Similar to the hybrid approach, there is a significant reduction in execution time when moving from sequential execution to the MPI approach with 2 processors.
- As the number of processors increases from 2 to 4 and 8, there is further reduction in execution time for all problem sizes. This demonstrates the scalability of the MPI approach.
- However, the reduction in execution time is not as substantial compared to the hybrid approach, indicating that the combination of MPI and OpenMP provides better performance for the given problem sizes.

Execution time (s) (Hybrid)			
	4 (mpich/threads)	8 (mpich/threads)	16 (mpich/threads)
6000x4000 (px)	175,12	95,40	78,89
3000X2000 (px)	46,84	23,66	17,70
600x400 (px)	3,48	0,92	1,61

Table 6: Execution time of the parallel version of Mandelbrot fractal using MPI and OMP

In the Table 6 we can observe:

- The execution time decreases significantly when moving from the sequential execution to the hybrid approach with 4 processors. This indicates the benefits of parallelization in reducing the overall execution time.
- As the number of processors increases from 4 to 8 and 16, there is further reduction in the execution time for all problem sizes. This demonstrates the scalability of the hybrid approach.
- The largest problem size of 6000x4000 pixels shows the most substantial reduction in execution time, indicating the effectiveness of parallelization in handling computationally intensive tasks.

5 Conclusions

The hybrid (MPI + OpenMP) approach stands out as the most effective solution, offering superior performance and scalability compared to the MPI and OpenMP approaches individually. By combining both MPI and OpenMP parallelization techniques, the hybrid approach demonstrates its ability to significantly reduce the execution time of the program, surpassing the performance of the sequential version.

The hybrid approach excels in scalability as well, as evidenced by the continuous reductions in execution time observed for all problem sizes as the number of processors increases. This scalability can be attributed to the balanced workload distribution among the processors, allowing for efficient parallel execution and effective handling of computationally intensive tasks.

The inter-node parallelism offered by MPI enables the distribution of the workload across multiple nodes, facilitating communication and coordination among different processes. This ensures efficient utilization of resources in a distributed computing environment. On the other hand, the intra-node parallelism provided by OpenMP allows for parallel execution within each node, effectively exploiting the available multi-core or multi-threaded architectures.

By combining both MPI and OpenMP, the hybrid approach achieves a synergistic effect, harnessing the advantages of both techniques. The inter-node parallelism helps in distributing the workload among different nodes, while the intra-node parallelism ensures efficient utilization of resources within each node. This combination optimizes the overall performance and scalability of the program.

In conclusion, the hybrid (MPI + OpenMP) approach proves to be the most optimal choice, offering superior performance and scalability for the program. By leveraging the benefits of both MPI and OpenMP parallelization techniques, it maximizes the utilization of available resources, efficiently handles computationally intensive tasks, and achieves significant reductions in execution time.