Assignment Report #4
Zooming In on the Mandelbrot Set

Student Name: Dias Suleimenov (ID: 202158836)
Course: PHYS 421 Parallel Computing

Submitted: October 21, 2025

Software packages used:

GCC/13.3.0	Main compiler
OpenMPI/5.0.5	OpenMPI library
Python 3.12.3 + Matplotlib 3.10.7	Visualization

AI tools used:

Gemini 2.5 Pro	Code generation, script writing
Copilot	Code generation, script writing

Abstract

The report investigates the parallelization of the Mandelbrot set generation using a master-worker algorithm implemented in C++ with OpenMPI. The performance is evaluated by measuring execution times across varying numbers of processes. The results demonstrate significant speedup and high efficiency, with a peak efficiency of 96.2% at 16 processes and a maximum speedup of 56.7x at 64 processes. The findings highlight the effectiveness of the parallelization approach while also noting the overhead associated with lower process counts.

1 Introduction

The Mandelbrot set is one of the simplest and best-known examples of a fractal structure. It is defined as the set of complex numbers $\{c\}$ for which the sequence

$$z_{n+1} = z_n^2 + c,$$

does not diverge to infinity when iterated starting from $z_0 = c$. That is, the sequence of complex numbers $z_0, z_1, z_2, \ldots, z_n$ remains bounded in magnitude as $n \to \infty$.

In this report, we explore the efficiency of proposed parallelized algorithm of Mandelbrot escape-time algorithm.

2 Methodology

The generation of Mandelbrot set is done used escape-time algorithm, for each pixel (representing some complex number) it is measured how much iterations it took before exceeding $|z| > 2$. Then the resulted set of escape-times used for coloring the mandelbrot set.

In order to implement parallelized generator of Mandelbrot set, the library OpenMPI and C++ language was used. Algorithm works in following way: the master-worker slices grid of pixels to blocks and sends it to slave-workers. Where slave-workers compute their assigned blocks and send the results back to the master process, when all blocks for frames are collected the master colors the image and saves it.

The scalability of the algorithm is analyzed by measuring time (using bash command `time`) required for processing with given ammount of cores. The serial code is used as baseline for comparison.

3 Results

The videos of the generated Mandelbrot set with zooming can be found at repository.

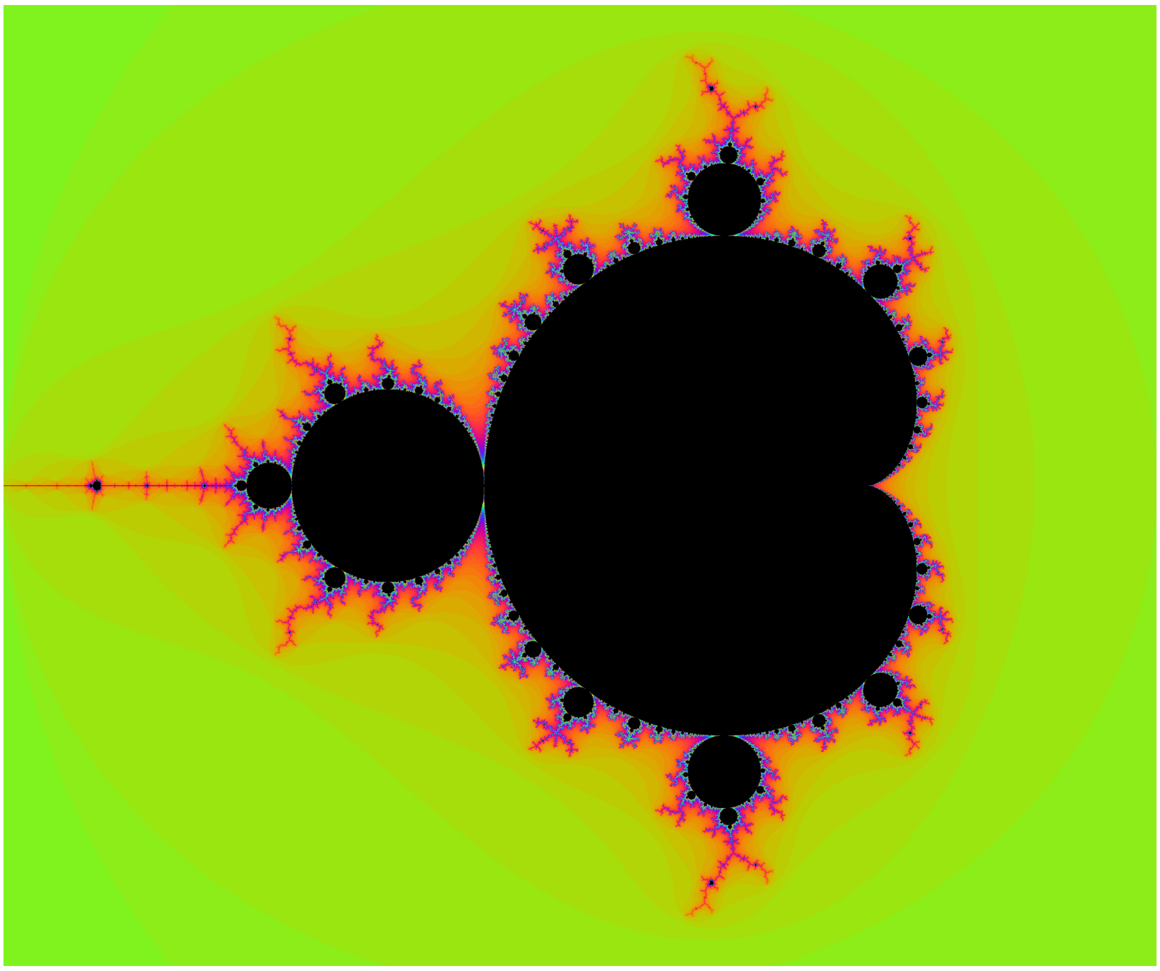

Figure 1: Image of Mandelbrot set covering the area $-2 < x < 1$, $-1.25 < y < 1.25$ and the depth of iteration $n_{\max} = 1000$.

Table 1: Parallel performance for following parameters: $n_{frames} = 20$, $N_x = 3840$, $N_y = 2160$, $n_{max} = 5000$.

# of Processes (P)	Avg. Time (s)	Speedup	Efficiency (%)
1	1858.00	1.00	100.0
2	1797.56	1.03	51.7
4	609.25	3.05	76.2
8	263.32	7.06	88.2
16	120.68	15.40	96.2
32	61.14	30.39	95.0
64	32.78	56.67	88.6

4 Discussion

As can be see from Figure 2 and Table 1, the growth of efficiency is not linear and it is not unexpected. Because factually parallelized version uses $N - 1$ slave workers for computation, and one master worker for saving data. What is more interesting that efficiency really high with increase of processes, especially at 16-32 processes. This may be caused due to improved cache hit rate as the chunks of arrays to process is decreased. However after that, it decreasing, which may suggest on the increased overhead of communication.

5 Conclusion

In this report a Mandelbrot set generator was successfully parallelized using a C++/MPI master-worker algorithm, and its scalability was evaluated. The implementation demonstrated excellent performance, achieving a peak efficiency of 96.2% with 16 processes and a maximum speedup of 56.7x on 64 processes. The downside of this approach was also noted on low ammount of processes, namely using $N - 1$ workers for computation.

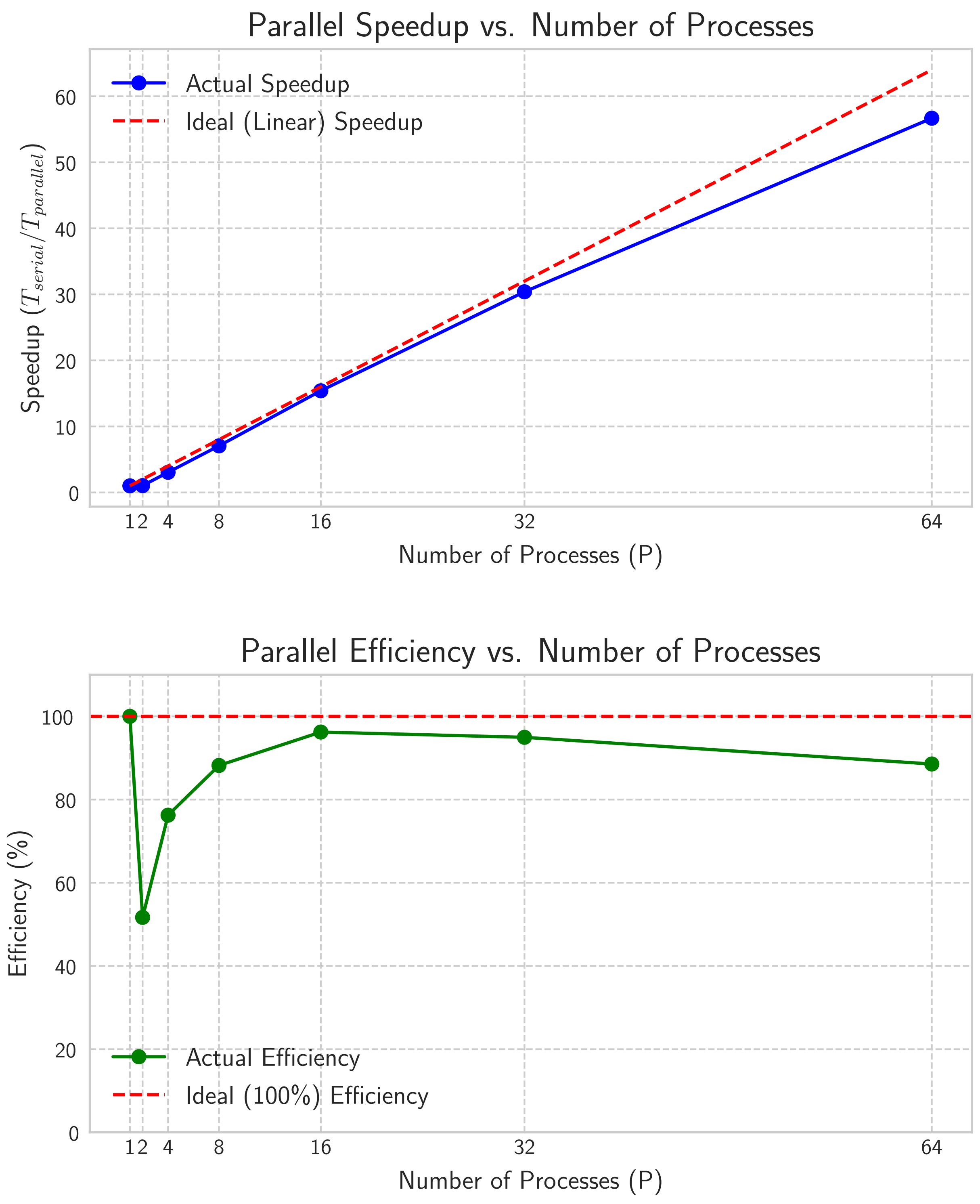

Figure 2: The speedup and efficiency of parallelized algorithm relative to number of processes.