Fundaments of HPC
Third Assignment
Exercise 1: The Mandelbrot Set

Nicola DOMENIS

January 1, 2020

1 Introduction

We present the second assignment in the course of FHPC. We will discuss about:

Main Exercise
Creation of an openmp code capable of rendering images of the Mandelbrot set;

Secondary Exercise
Testing the strong and weak scalability of the program;

2 Main exercise: visualizing Mandelbrot sets

We started by implementing the serial program as "serial_mandelbrot.c". We followed the instructions on the assignment. We use the input data to create a matrix that will contain the values that count the steps of the series to reach non-convergence, given by the condition $|z| > 2$. There are three nested for cycles. Two for cycles are used to scan the matrix and one verifies if the point to which the cells refers to belongs or not to the mandelbrot set. Then we recognized that to parallelize this problem it was necessary to act on the main for cycle. We added an OMP for command to deal with the parallelization. Then we tested the program with various parameters. Notably we risk of exhausting the memory of the computer if we make a too big matrix. Having a large number of iterations is also a source of computational effort from the machine. Finally, the two points of input might be problematic to explore the Mandelbrot set, since it is easy to miscalculate the position and end up into a zone that fully belongs to the Mandelbrot set, thus being uninteresting. Interesting results came up by substituting the series formula $z_{n+1} = z_n^2 + c$ with other formulas. We report some interesting images:

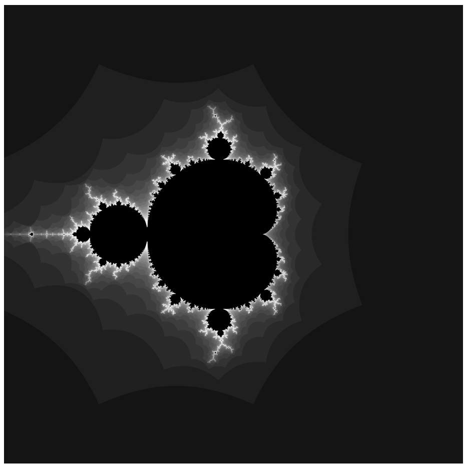

Figure 1: Representation of the Mandelbrot set

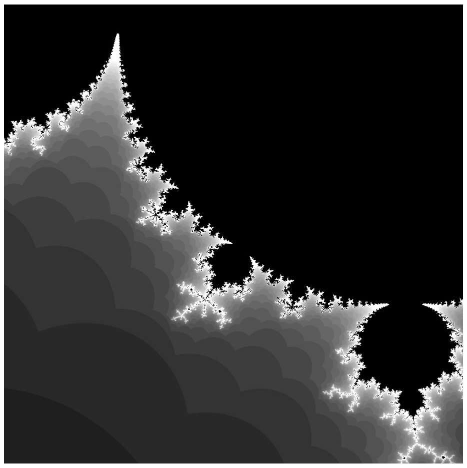

Figure 2: Particular of the Mandelbrot set

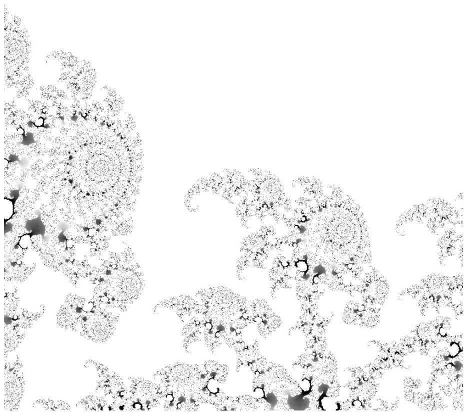
Figure 3: Deep particular of the Mandelbrot set

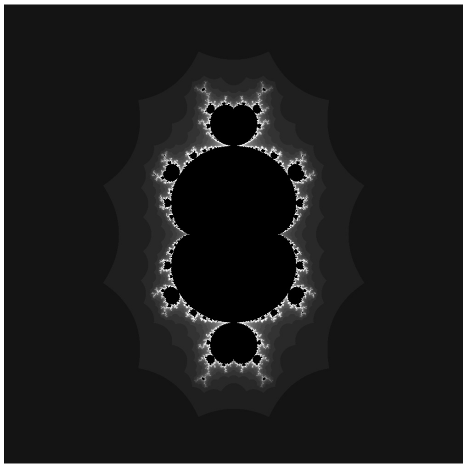
Figure 4: Mandelbrot set using $z_{n+1} = z_n^3 + c$

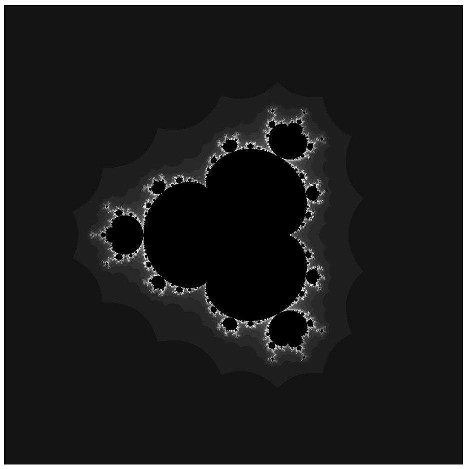

Figure 5: Mandelbrot set using $z_{n+1} = z_n^4 + c$

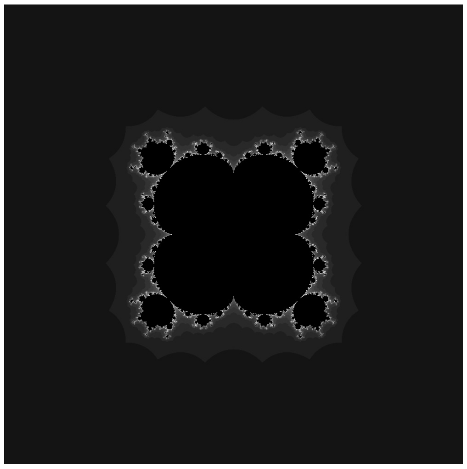

Figure 6: Mandelbrot set using $z_{n+1} = z_n^5 + c$

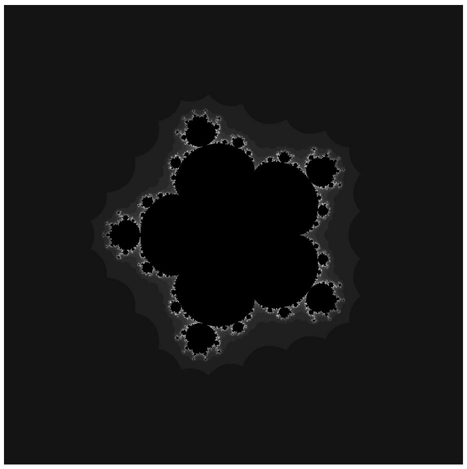

Figure 7: Mandelbrot set using $z_{n+1} = z_n^6 + c$

3 Strong and Weak Scalability Test

We report the graphs of the elapsed time during the strong and weak scalability tests:

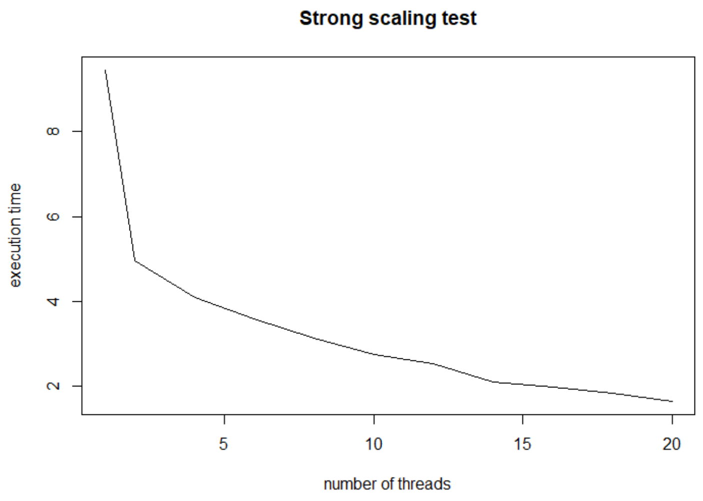

Figure 8: Strong scaling test for $N = 10^4$

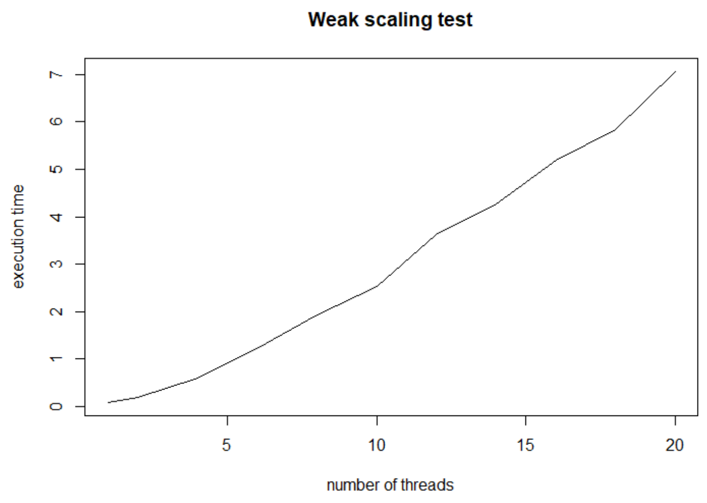

Figure 9: Weak scaling test for $N = 10^3$multiplied by the number of threads

We see that the code scales well in the strong scaling case, while it doesn't have a constant execution time while we execute the weak scaling test. The reason for this might be that the workscale isn't distributed very well among the threads. By increasing N we are just packing more points that do converge in the execution task of some threads that have to calculate all the iterations before stating that the points don't diverge. The workload unbalance can be mitigated by the increase of threads in the strong scaling case, but it is not sufficient if the size of the problem grows, as in the weak scaling test.