The Mandelbrot set
Hybrid MPI/OpenMP implementation

Alessandro Della Siega

May 2024

1 Introduction

The goal of this assignment is to compute and visulize the Mandelbrot set. The Mandelbrot set properties make this problem embarrassingly parallel, and it is a perfect candidate for a hybrid MPI/OpenMP implementation.

The analysis of the computation is done in the context of a hybrid MPI/OpenMP implementation. We will evaluate the strong scaling and the weak scaling as follows:

- MPI scaling: we will consider a single OpenMP thread per MPI process and we will increase the number of MPI processes;
- OpenMP scaling: we will consider a single MPI process and we will increase the number of OMP threads.

1.1 Computational architecture

The computation will be performed on the ORFEO cluster. In particular, we will take advantage of the nodes of the EPYC partition which consists of 8 nodes equipped with two AMD EPYC 7H12 CPUs. Each node is equipped with:

- 128 cores;
- 256 threads;
- 512 GB of RAM.

2 Implementation

The encoding of the problem is straightforward. An image of the Mandelbrot set is a two-dimensional array of pixels. Each pixel corresponds to a point in the complex plane. However, the image is stored as a one-dimensional array of pixels. Each pixel is stored as a `unsigned char`, its maximum value can be 255.

Let us present a brief summary of C implementation:

- `main.c`, it contains the main function in which MPI is initialized, the amount of work is distributed among the MPI processes, and the computation is performed. In the end, the computation is gathered and the image is saved. The main function also takes care of the timing of the computation;

- `mandelbrot.c`, in this file we define two functions. The first is the iterative quadratic map that characterizes the points of the Mandelbrot set. The latter is the function whose task is to evaluate, within an OpenMP parallel region, the Mandelbrot set on a given array of pixels;

- `image_utils.c`, in this file we define the function that saves the image in the desired PGM format;

- `timing_utils.c`, here the function that saves the timing results on a `.csv` file.

2.1 MPI parallelization

In order to distribute the work among the MPI processes we decided to adopt a sequential fashion.

Suppose to have N processes and $M = n_x \times n_y$ pixels to compute. The root process divides the one-dimensional array in N blocks of approximately M/N pixels each. Obviously, the last block may contain less pixels if M is not divisible by N since the remainder is distributed starting from the first process, with respect to the rank. The first process is assigned the first block of pixels, the second process the second block, and so on. The last process is assigned the last block of pixels. The gathering of the results is done by the root process, which calls `MPI_Gatherv` to collect the results from all the other processes and saves the image.

2.2 OpenMP parallelization

The OpenMP parallelization is performed in `compute_mandelbrot_set` in the file `mandelbrot.c`. The parallel region is defined at the beginning of the function and the parallel for directive is used to parallelize the loop that iterates over the pixels of the image, i.e. the elements of the one-dimensional array. The scheduling policy is set to `dynamic`: OpenMP assigns one iteration to each thread. When the thread finishes, it will be assigned the next iteration that has not been executed yet.

2.3 Experimental setup

For both the MPI and OpenMP scaling, we will consider the strong scaling and the weak scaling. The strong scaling consists in fixing the dimension n_x, n_y of the Mandelbrot image and increasing the number P/T of processes/threads. While, the weak scaling consists in fixing the amount of work per process/thread and increasing the number of processes/threads. Let us define the following setup:

- MPI strong scaling: we fix $T = 1$,

  $$n_x = 10000$$

  $$n_y = 10000$$

  and $P = 1, 2, 4, 8, 16, 32, 64, 96, 128$.

- MPI weak scaling: we fix $T = 1$,

  $$n_x = 2000 \times \texttt{round}\{\sqrt{P}\}$$

  $$n_y = 2000 \times \texttt{round}\{\sqrt{P}\}$$

  for $P = 1, 2, 4, 8, 16, 32, 64, 96, 128$;

- OMP strong scaling: we fix $P = 1$,

  $$n_x = 10000$$

  $$n_y = 10000$$

  for $T = 1, 2, 4, 8, 16, 32, 64, 96, 128, 160, 192, 224, 256$;

- OpenMP weak scaling: we fix $P = 1$

  $$n_x = 2000 \times \texttt{round}\{\sqrt{T}\}$$

  $$n_y = 2000 \times \texttt{round}\{\sqrt{T}\}$$

  for $T = 1, 2, 4, 8, 16, 32, 64, 96, 128, 160, 192, 224, 256$.