TD/TP 2: Mandelbrot set

September 25, 2023

1 Introduction

The Mandelbrot set consists of all points c of the complex plane C for which the following iterative scheme does not diverge:

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases} . \tag{1}$$

Writing $z = x + iy$ and $c = a + ib$, the equations (1) can be rewritten as

$$\begin{cases} x_{n+1} = x_n^2 - y_n^2 + a \\ y_{n+1} = 2x_n y_n + b \end{cases}$$

with initial conditions $x_0 = y_0 = 0$.

One can show that, if there exists an integer n such that $|z_n| > 2$ (i.e. $|z_n|^2 = (x_n^2 + y_n^2) > 4$), then the iterative scheme (1) diverges. To learn more about this subject, one can refer to [1, 2].

2 Storage of images in memory

An image is a two-dimensional array. Each element of this array is called a *pixel*, which is the abbreviation of *picture element*. The value of this point can be, depending on the type of image, a value of grayscale intensity, a color, or a radiance value. In memory, this array is organised row-wise: we first find the first row, then the second, etc. In particular, we will manipulate images encoded on a single byte (for one pixel), and the value of each pixel (therefore an integer between 0 and 255) represents the index in a table of colors.

In C, this can be translated as:

```c
int w, h;         // width and height of the image

unsigned char *image = malloc(w * h);

for (int i = 0; i < h; i++)
  for (int j = 0; j < w; j++)
    image[i * w + j] = 0;   // accessing pixel i,j
```

By convention, the pixel with coordinates $(0, 0)$ is the top-left corner of an image printed on the screen; it is therefore also the first element in the array `Image` in memory.

3 Image storage format

There is a huge variety of formats for storing images in a file. We will use the Sun Rasterfile format, which has the advantage of being very simple to manipulate and which can be visualized through most image displaying software[1].

[1] For example `display` from the software bundle `ImageMagick` on Linux

A rasterfile file consists of a header which described the main characteristics of the image (size, encoding, ...), followed by a list of 1-byte words which give the table of colors (if necessary). Then, finally, comes the image itself, stored as an array of raw data.

4 Sequential algorithm

Synopsis of the algorithm:

1. For the center of each pixel of the image:
   (a) Compute the number of iterations before which the iterative scheme diverges (more precisely, before which we reach some $|z_n| > 2$). The maximal number of iterations will be limited by some user-defined depth.
   (b) Update the value of the corresponding pixel:
   If the depth is reached, pixel_color $\leftarrow$ 255
   Else, pixel_color $\leftarrow$ NumberIterations % 255
2. Save the image.

5 Questions

1. Discuss how the algorithm presented in Section 4 may be parallelized.
2. Propose a simple parallel version of this algorithm. Use an MPI routine for collective communication. Run it on at least 100 "processors" (=CPU cores).
3. Compare the running times for each processor. Do this with various numbers of processors and record the resulting parallelization efficiency. In particular, you can focus on the case of 8 processors with the default values.
4. Propose another load-balancing strategy to improve performance.
5. Implement (in C) the algorithm which corresponds to this load-balancing strategy.

References

[1] The Fractal Geometry of the Mandelbrot Set, *Robert L. Devaney* `http://math.bu.edu/DYSYS/FRACGEOM/`

[2] The Spanky Fractal Database, *Noël Giffin*, `http://spanky.triumf.ca/www/welcome1.html`

[3] Mandelbrot hard zoom into spirals and galaxies, `https://www.youtube.com/watch?v=jm_Q1FO9bP4`