Assignment 2
Mandelbrot and Julia Sets

CS/SE 4F03 – 2014

January 31, 2014

Due date: February 25, 2014 in class

General remarks

- You are allowed to work in groups of at most four.
  - Each student in a group receives the same grade.
  - Only one group member should submit to `svn`.
- If you are a graduate student, you must not work in a group.
- As we progress with this assignment, new information may be added to this description, and the provided source code may be updated.

1 Introduction

1.1 Mandelbrot set

Let z_0 be a complex number. The Mandelbrot set is the set of values z_0 in the complex plane for which the iteration

$$z_{k+1} = z_k^2 + z_0, \quad k = 0, 1, \ldots, \tag{1}$$

is bounded as $k \to \infty$. It can be shown that this sequence is unbounded, if for some k, $|z_k| > 2$.

1.2 Julia sets

Given a complex constant c, for each point z_0 in the complex plane, generate

$$z_{k+1} = z_k^2 + c, \quad k = 0, 1, \ldots \tag{2}$$

If this sequence is bounded, then z_0 is in the Julia set corresponding to c.

1.3 Goal

The goal of this assignment is to produce beautiful, high-resolution images of Julia and Mandelbrot sets using distributed parallel computing.

Serial program. You are given a serial program that generates a bmp image. You can download it from `http://www.cas.mcmaster.ca/~nedialk/COURSES/4f03/code/julia.zip`. Study the makefile and the C files, and in particular `julia.c`. We use a simple coloring scheme, but you can replace it with your own in the file `savebmp.c`. You are free to modify the coloring scheme.

2 Visualization

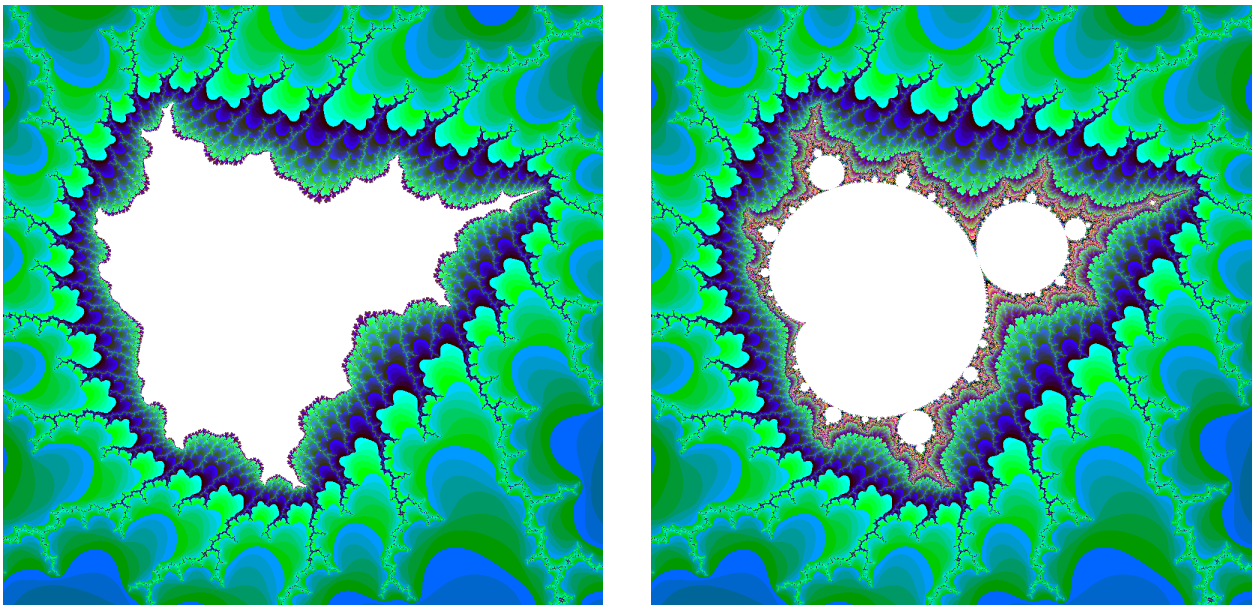
2.1 Mapping pixels

Consider an image of $m \times n$ pixels. We map each pixel to a complex number. This number is our z_0. We iterate using formula (1) or (2) until

- $|z_k| > 2$ or
- $k = M$, where M is the maximum number of iterations allowed.

If $|z_k| \leq 2$, then $k = M$, and it is likely that z_0 is in the set. We can color z_0 white (or black). Otherwise, $z_k > 2$ at iteration $k \leq M$, and the sequence becomes unbounded. We record the time when z_k "escapes" using a color corresponding to the iteration number k; that is, we color z_0 with a color depending on this k.

In Figure 1, we show such a visualization.



(a) $M = 100$ (b) $M = 2000$

Figure 1: Image computed with (1), where the initial points $z_0 = x + yi$ are from the region $x \in [-0.181862, -0.181772]$, $y \in [1.0190851.019175]$. The image is with width and height of 1,200 pixels. The white region are points for which we conclude they are from the Mandelbrot set. If we decrease M the white region expands and vice versa.

2.2 Converting pixel coordinates to complex numbers

Assume we are given a pixel with coordinates (x, y). Let the complex region be given as $x_1 < x_2$ and $y_1 < y_2$. Let w be the width of the image and let h be its height. If we denote $z_0 = a_0 + b_0 i$, we compute

$$a_0 = x_1 + \frac{x}{w}(x_2 - x_1)$$
$$b_0 = y_1 + \frac{y}{h}(y_2 - y_1).$$

See `julia.c`.

2.3 Evaluating $z^2 + c$

Let $z = a + bi$ and $c = c_r + c_i i$. Then

$$z^2 + c = a^2 - b^2 + c_r + (2ab + c_i)i.$$

That is, we can compute on each iteration the real part $a^2 + b^2 + c_r$ and the imaginary part $2ab + c_i$.

We need to check when $|z| = \sqrt{a^2 + b^2} > 2$. That is, we can check when

$$a^2 + b^2 > 4.$$

2.4 Coloring

There are various ways to color the pixels. We generate colors and a bmp file in `savebmp.c`.

3 Why parallel computing?

By depth of the image we mean the maximum number of iterations M. The size of the image and the depth determine the level of detail and the computation time. M could be several hundred or in the thousands.

Interesting points are the ones for which we reach M and $|z_M| \leq 4$: we do not know for sure if they are in the set. By increasing M, some of such points may escape.

4 Tasks

The goal is to produce a parallel MPI implementation. A subtle issue is how to distribute the work equally on processors.

4.1 Investigation

Read on the internet about the theory of these sets. Good sources are e.g.

- `http://www.mathworks.com/moler/exm/chapters/mandelbrot.pdf`
- `http://classes.yale.edu/julias/Mandelset/welcome.html`
- `http://en.wikipedia.org/wiki/Julia_set`
- `http://en.wikipedia.org/wiki/Mandelbrot_set`

You can explore such sets at e.g. `http://aleph0.clarku.edu/~djoyce/julia/explorer.html`.

4.2 Parallel algorithm

We will be discussing in class and during tutorials the issues involved in producing a good parallel implementation. For this assignment, you are required to describe in words and using pseudo code how you would compute these sets in parallel and produce a parallel MPI implementation.

4.3 Implementation

You should produce a program that takes as input the following list of parameters

```
flag cr ci xmim xmas ymin ymax height width maxit image stats
```

where

parameter	description
`flag`	if 0 iterate with (1); otherwise iterate with (2). If 0, the next two number do not matter
`cr ci`	the real and imaginary part of the constant c in (2)
`xmim xmax`	specify the interval $[x_{\min}, x_{\max}]$ for the real part of z_0
`ymim ymax`	specify the interval $[y_{\min}, y_{\max}]$ for the imaginary part of z_0
`height`	height of the image in pixels
`width`	width of the image in pixels
`maxit`	maximum number of iterations allowed
`image`	name of a file into which the resulting image is stored
`stats`	name of a file that stores data about an execution

- In `savebmp.c`, we store the image into a file with extension `bmp`, so your file name should be `image.bmp`.
- Each line of the `stats` file is

  `process time iterations`

  - `process` is the process number
  - `time` is the CPU time taken by this process, and
  - `iterations` is the maximum number of iterations reached by `process`

  The lines should be sorted in increasing order by process number. In the provided serial implementation, such a file is not used.

Name your executable `julia`. The image in Figure 1(b) is computed by

```
./julia 0 -0.4 0.6 -0.181862 -0.181772 1.019085 1.019175 1200 1200 2000 image.bmp
```

Using MPI, you should run e.g.

```
mpirun -np 8 ./julia 0 -0.4 0.6 -0.181862 -0.181772 1.019085 1.019175 1200 1200 2000 image.bmp stats
```

5 What to submit

1. Description of your parallel algorithm. Discuss how well it distributes the work.
2. Hard copy of your source code
3. Try to discover interesting images. Submit up to 5 images which you find interesting. Convert them to PDF format, and submit the PDF files to `https://websvn.mcmaster.ca/se4f03/macid/A2/images`. Include the image number in the name of the file, e.g. `image-1.pdf`.

   In a text file in this directory, name it `params`, list the parameters you have used to generate the bmp files. You can simply have each line in this file of the form

   `mpirun -np 8 ./julia 0 -0.4 0.6 -0.181862 -0.181772 1.019085 1.019175 1200 1200 2000 image.bmp stats`
4. Your stats files under `https://websvn.mcmaster.ca/se4f03/macid/A2/stats`. The file corresponding to image number i should have name `stats-i`.
5. You source code under `https://websvn.mcmaster.ca/se4f03/macid/A2/`.

   In this directory, there must be a makefile such that when `make` is typed, the executable `julia` is created.

6 Evaluation

The evaluation of your work will include assessing your algorithm, implementation, and quality of images.