Parallelized Mandelbrot Set Member Calculation

Program One
CS 3331 Fall 2023
Due: Sep. 18, Monday at 11:00pm

Motivation

The Mandelbrot set is a set of complex numbers. When a certain recurrence relation is applied to the numbers outside the set, the values quickly approach infinity. The values for numbers inside the set remain finite. Near the edge of the set, the speed at which the terms approach infinity varies wildly.

The recurrence relation is

$$z_{n+1} = {z_n}^2 + C$$

where $z_0 = C$ and C is a complex number. As it turns out, if the size[1] of z_i exceeds 2 for some value of i, then it can be shown that the terms will eventually become infinite. The value of i when z_i exceeds 2 varies wildly for values near the set members.

Plots of the Mandelbrot set can produce beautiful images. For this project, you will take an existing program that plots the Mandelbrot set and parallelize it.

Existing Code

The program you are given is invoked as:

mandelPlot `tlr tli real-side-length imag-side-length max-iterations \`
`horizontal-pixels vertical-pixels image-file`

It identifies whether the points within a rectangle of the complex plane are in the mandelbrot set by applying the recurrence relation above to each point. The coordinate of the top left point in the rectangle is: `tlr` + `tli`I. The length of each side in the real and complex plane is given by `real-side-length` and `imag-side-length` respectively. There is a one-to-one correspondance between the values tested for inclusion in the Mandelbrot set and pixels in the graph that is created by the program. Hence the number of points explored is `horizontal-pixels*vertical-pixels` and the distance between two points along any axis is `side-length`/(`pixels`-1).

The recurrence relation given earlier is applied to each point in turn. At most `max-iterations` terms are calculated. If after this number of terms the value is still below two, the point is taken to be in the set.

The program also plots the points through creation of a PPM file named `image-file`. The number of iterations performed at a particular point determines its color in the final plot. The PPM file format specifies the number of pixels in each row and column as part of a header written to the file. Following the header are three bytes for each pixel that represent the red, green, and blue color values for the point. The value of each component must be 255 or lower. The pixel positions are implicit; the color values are listed in row major order. The first color value corresponds to the upper left corner of the image. The remaining colors are applied row wise until all the points have been specified.

You can use the command

```
gimp filename
```

[1]The size of z_i is the length of the hypotenuse of a triangle where the sides are the complex and imaginary components of z_i

to view the PPM files you create on a departmental Linux system.

Several color schemes are available. The color scheme is chosen through a macro which should **only be defined at the point of compilation**, e.g. `gcc -DCOLORSCHEME=1 -omandelPlot mandelPlot.cc`. Following are example color schemes, invocations, and the images they produce.

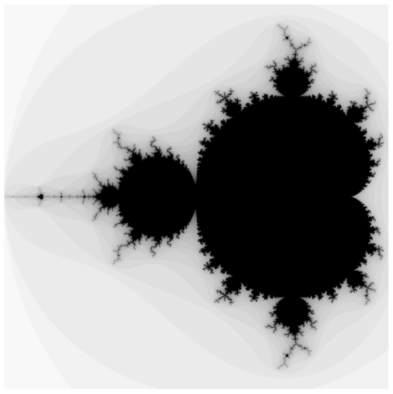

Figure 1: COLORSCHEME=1, parameters: -2+1.25I, 2.5 side length, 40 iterations max, 1500 pixels

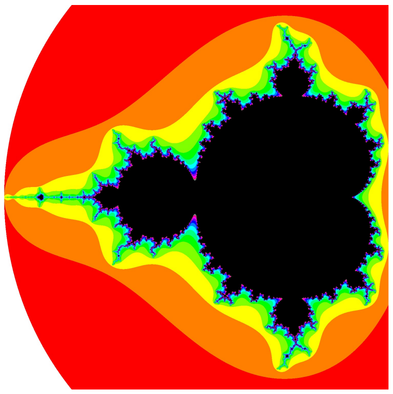

Figure 2: COLORSCHEME=3, parameters: 0.39+0.39I, 0.08 side length, 50 iterations max, 1500 pixels

Requirements

You will parallelize the calculation of points in the Mandelbrot set. Your application is invoked as: `pmandel tlr tli real-side-length imag-side-length max-iterations horizontal-pixels vertical-pixels image-file nprocs`. The first eight arguments have the same meaning as above. `nprocs` specifies a number of processes that should be created to perform the calculation. The main program spawns `nprocs` children. Each child calculates an equal sized horizontal strip of the rectangle and writes it to a file named `image-fileX`, where *X* is an integer identifier for the process performing the calculation. This is depicted in Figures 3 and 4.

For simplicity, you may assume that the number of processes will divide evenly into the length of the vertical side and the number of pixels. The image file created by each child should be viewable (have its own header and color values).

After spawning the children, the main process waits for each child to complete. It then concatenates the image files into a single PPM file that plots the entire area. This complete image file is named `image-file`. Note that

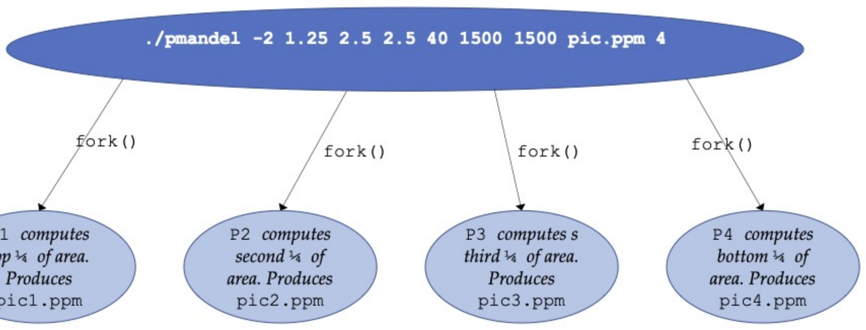
./pmandel -2 1.25 2.5 2.5 40 1500 1500 pic.ppm 4
fork()
fork()
fork()
fork()
P1 computes top ¼ of area. Produces pic1.ppm
P2 computes second ¼ of area. Produces pic2.ppm
P3 computes s third ¼ of area. Produces pic3.ppm
P4 computes bottom ¼ of area. Produces pic4.ppm

Figure 3: Process tree

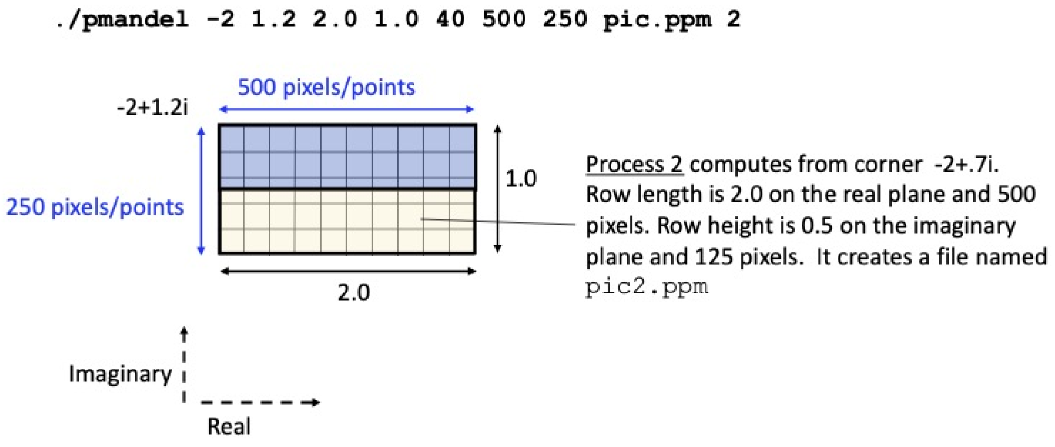
./pmandel -2 1.2 2.0 1.0 40 500 250 pic.ppm 2
-2+1.2i
500 pixels/points
250 pixels/points
1.0
2.0
Process 2 computes from corner -2+.7i. Row length is 2.0 on the real plane and 500 pixels. Row height is 0.5 on the imaginary plane and 125 pixels. It creates a file named pic2.ppm
Imaginary
Real

Figure 4: Distribution of Points among Child Processes

there can be only one header in the consolidated file and it must contain the appropriate header and row/value pairs.

Notes

- Do not modify the PPM file header format (the pixel values will change of course).
- Do not delete the child process image files. These will be used for grading.
- The parent and all the child processes **must** run concurrently.
- Before beginning its calculations, each of the child processes must print their calculation parameters using the same `snprintf` and `write` statements given in the distributed code (with appropriate parameters of course). Each child process must also print its PID immediately prior to exiting using the same `snprintf` and `write` statements given in the distributed code. This is required for grading.
- The `mandelPlot` code is in: `/home/campus13/jmayo/public/cs3331/projects/project1/`. A makefile in that directory provides an overview of the code structure.

- I suggest you spend some time learning the code. Once you understand how it operates, the parallelization will be much simpler.

Collaboration

No collaboration is allowed for this project.

Submission

Keep all your files in a single directory to facilitate grading. Submit your files through Canvas. Your submission will contain `pmandel.c`, a `makefile`, and any additional source code or header files. Typing `make` from a directory with your submitted files should create an executables named `pmandel`. This binary will be used to grade your program. Your makefile should compile `pmandel` with COLORSCHEME defined as 1. Typing `make clean` should remove all object files, binaries and image files. **Be sure to check your submission after it is uploaded.** Remember that submission of the wrong files will not be considered in the grading. Also be sure to run your code on `guardian.it.mtu.edu` or a CS lab machine. The fact that your code ran on your own machine will not be considered in the grading.