Parallelized Mandelbrot Set Member Calculation

Program One
CS 3331 Spring 2023
Due: Jan. 27, Friday 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:

`mandel tlr tli side-length max-iterations pixels image-file`

It identifies whether the points in a square 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 square is: `tlr` + `tliI`. The length of a side of the square in the complex plane is given by `side-length`. There is a one-to-one correspodance between the values tested for inclusion in the Mandelbrot set and pixels. Hence the number of points explored is `pixels*pixels` and the distance between two points along any axis is `side-length/pixels`.

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 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 depicted.

You can use the command

`gimp filename`

to view the PPM files you create.

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 -omandel mandel.c`. Following are example color schemes, invocations, and the images they produce.

[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

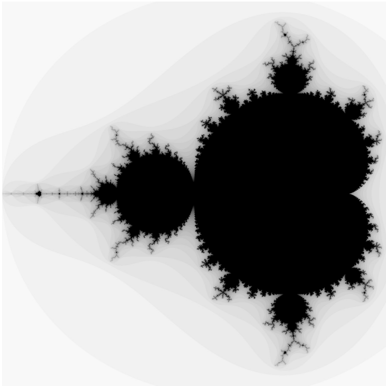

Figure 1: COLORSCHEME=1, parameters: -2+1.25I, 2.5 side length, 40 iterations max, 1500 pixels

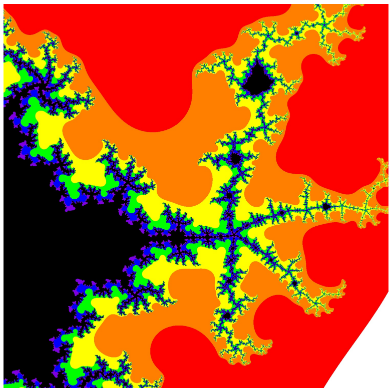

Figure 2: COLORSCHEME=3, parameters: 0.39+0.39I, 0.08 side length, 50 iterations max, 1500 pixels

Requirements

You will parallelize the calculation of points in the Mandelbrot set. Your application is invoked as: `pmandel tlr tli side-length max-iterations pixels image-file nprocs`. The first six 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 square 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 a side and the number of pixels. The image file created by each child should be viewable (have its own header and color values). Notice that you will have to modify the program given to you so that it can operate on a rectangle instead of a square.

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

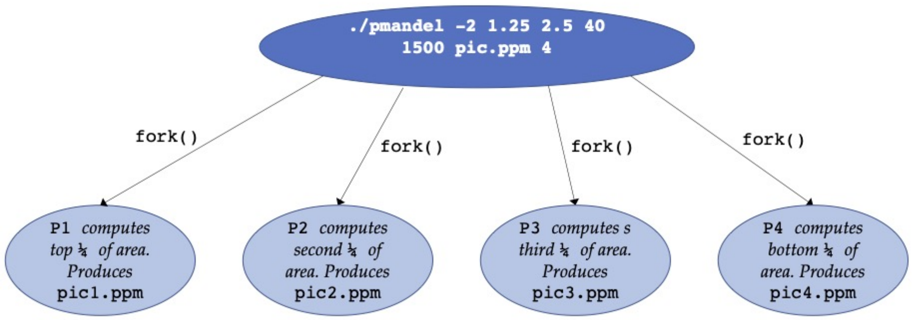

Figure 3: Process tree

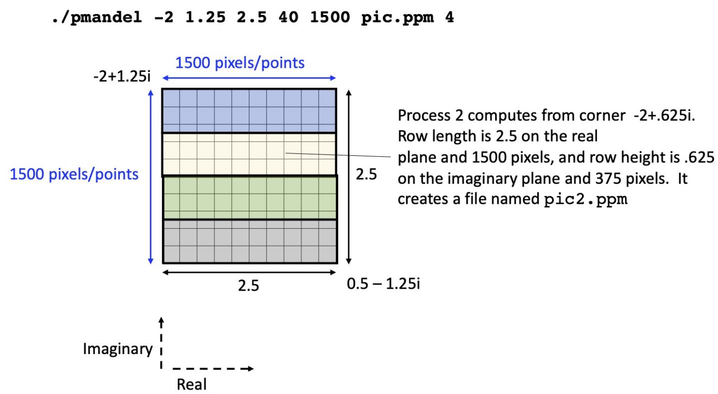

Figure 4: Distribution of Points among Child Processes

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 `mandel` code is in: `/home/campus13/jmayo/public/cs3331/projects/project1/mandel.c`
- I suggest you spend some time learning the distributed 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. The project is due on January 27, Friday, at 11pm. 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.