Homework 2: Mandelbrot Set

Deadline: July 18, 2019, 8:00

Contents

Goal

This assignment helps you get familiar with:

1. Pthread
2. Hybrid parallelism (MPI + OpenMP)
3. Load balancing techinques

In this assignment, you are asked to parallelize the sequential Mandelbrot Set program using:

1. (hw2a) – pthread / std::thread
2. (hw2b) – MPI + OpenMP

Problem Description

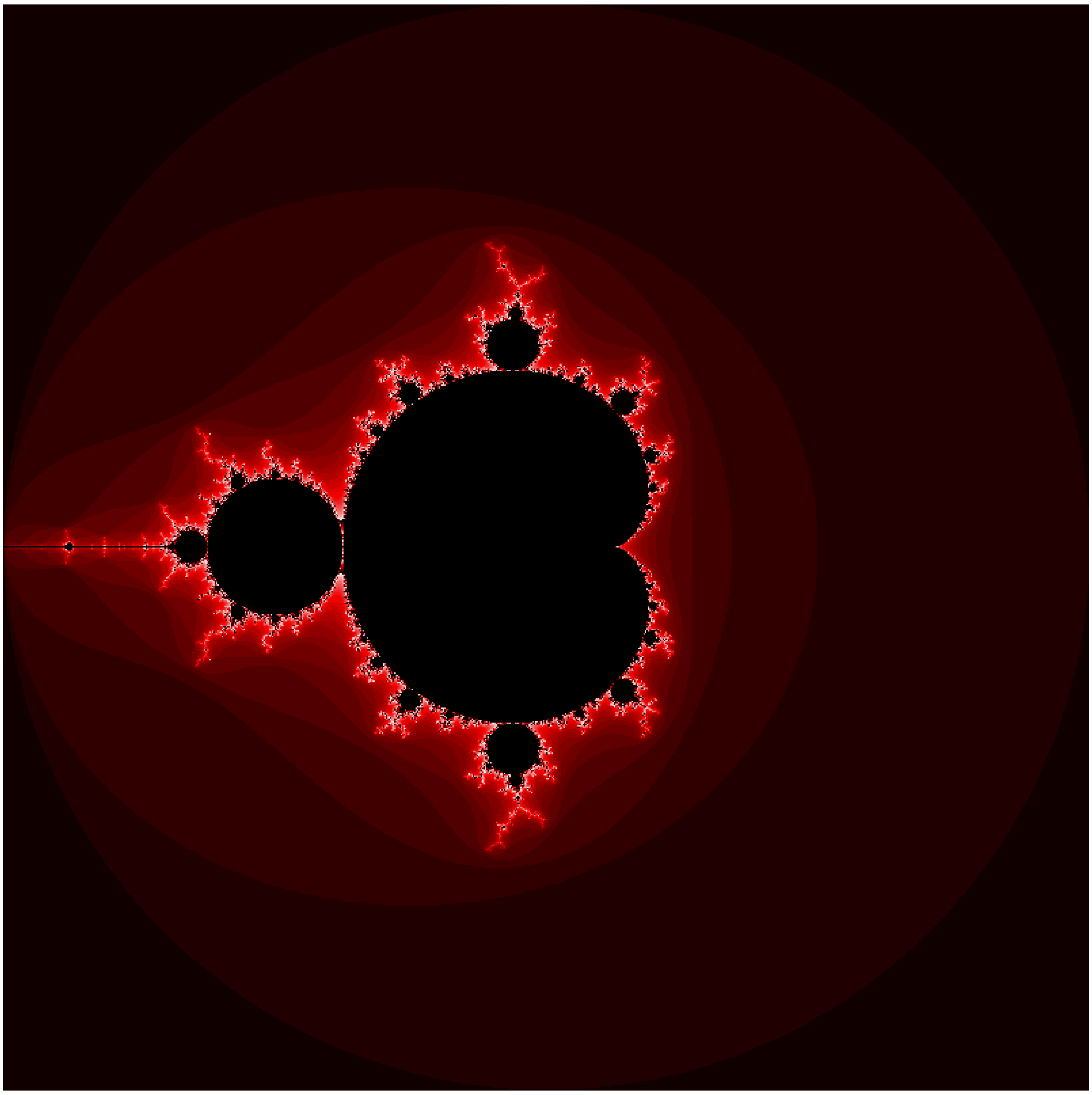
Mandelbrot Set is a set of complex numbers that are quasi-stable when computed by iterating the function:

$$\begin{cases} Z_0 & = C \\ Z_k & = Z_{k-1}^2 + C \end{cases}$$

- C is some complex number: $C = a + bi$
- Z_{k+1} is the $(k+1)_{\text{th}}$ iteration of the complex number
- if $|Z_k| \leq 2$ for any k, C belongs to Mandelbrot Set

What exactly is Mandelbrot Set?

- It is fractal: An object that display self-similarity at various scale; Magnifying a fractal reveals small-scale details similar to the larger-scale characteristics
- After plotting the Mandelbrot Set determined by thousands of iterations:



You can get a felling about Mandelbrot Set at http://tilde.club/~david/m/.

Requirements

In this homework, you are asked to implement 2 versions:

1. `hw2a` – thread version, using Pthread or std::thread.
2. `hw2b` – hybrid version, using OpenMP **and** MPI.

Both versions follow the same input & output format.

Input specification

The input is specified from the command line. Ther are no input files.
Your program should accept the following `srun` command:

```
srun -n$procs -c$threads ./executable $out $iters $x0 $x1 $y0 $y1 $w
$h
```

For example, the image in Problem Description is created by:

```
srun -n1 -c1 ./hw2seq out.png 10000 -2 2 -2 2 800 800
```

The meaning of the arguments are:

- `$procs` – int; [1, 48]; number of processes. Always 1 for the thread version (`hw2a`).
- `$threads` – int; [1, 12]; number of threads per process.
- `$out` `(argv[1])` – string; the path to the output file.
- `$iters` `(argv[2])` – int; [1, 2×10^8]; number of iterations. (the largest *int* is around 2.1×10^9)
- `$x0` `(argv[3])` – double; [-10, 10]; inclusive lower bound of the real axis.
- `$x1` `(argv[4])` – double; [-10, 10]; non-inclusive upper bound of the real axis.
- `$y0` `(argv[5])` – double; [-10, 10]; inclusive lower bound of the imag axis.
- `$y1` `(argv[6])` – double; [-10, 10]; non-inclusive upper bound of the imag axis.
- `$w` `(argv[7])` – int; [1, 16000]; number of points in the x-axis for output.
- `$h` `(argv[8])` – int; [1, 16000]; number of points in the y-axis for output.

Output specification

Your programs should produce a PNG image at `$out`, visualizing the Mandelbrot Set in the given range.
We provide a sequential version to show how the pixels are rendered. See Resources.

Grading

1. Correctness (60%)
   Weights:

   a. thread version (30%)
   b. hybrid version (30%)

   Several test cases will be used to test your implementations. You get 30 points for an implementation if you passed all the test cases, (30×0.7^k) points if there are k failed test cases.
   For each test case, you pass it if:

   - Your implementation produced a valid PNG image.
   - At least 99.6% of the pixels in your output are identical to the corresponding pixel produced by the sequential version. You are advised not to use this tolerance for optimizations.
   - The execution time of your implementation is *shorter* than **the execution time of the sequential version + 30 seconds**.

2. Performance (30%)
   Weights:

   a. thread version (10%)
   b. hybrid version (20%)

   You need to pass all the correctness test cases in the corresponding version in order to get performance points.
   Points are given according to the relative performance of your program among all the students.
3. Demo (10%)

   - Each student is given 5 minutes to explain the implementation followed by some questions from TA.
   - Points are given according to your understanding and explanation of your code, and your answers of the TA questions.

Submission

Put your source code under at `~/homework/hw2` in `apollo31`:

- Your Makefile – `~/homework/hw2/Makefile`
- Source code for thread version – `~/homework/hw2/hw2a.cc` or `~/homework/hw2/hw2a.c`
- Source code for hybrid version – `~/homework/hw2/hw2b.cc` or `~/homework/hw2/hw2b.c`

`make hw2a` and `make hw2b` will be used to compile your code.
Use `hw2a-judge` and `hw2b-judge` to check and submit your code **before the deadline**. You can submit **as many times as you want**.

Resources

Resources are given under `/home/ipl19/x/hw2/`:

- Sequential version – `hw2seq.c`
- Example Makefile – `Makefile`
- (TODO) Testcases `cases/`

Submissions can be viewed at http://ipl.cs.nthu.edu.tw/s/hw2a and http://ipl.cs.nthu.edu.tw/s/hw2b.
In order to compare your image with the the answer, use:

```
hw2-diff answer.png yourimg.png
```

Here are a few programs that be used to get the output PNG to your computer for display:

- `rsync`
- `scp`
- `sftp`
- MobaXterm
- FileZilla

Final Notes

Contact TA via afg984@gmail.com or iLMS immediately if you find any problems with the homework specification, judge scripts, example source code or the test cases.
You are allowed to discuss and exchange ideas with others, but you are required to write the code on your own. You’ll got 0 points if we found you cheating.