Last Update: 2023.10.17

CS542200 Parallel Programming
Homework 2: Mandelbrot Set

Due: Sun Nov 12, 2023 23:59

1 Goal

In this assignment, you are asked to parallelize the sequential *Mandelbrot Set* program, and learn the following skills:

- Get familiar with thread programming using Pthread and OpenMP.
- Combine process and thread to implement a hybrid parallelism solution .
- Understand the importance of load balance.

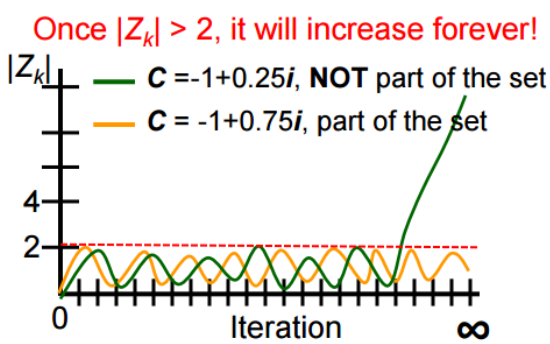
2 Problem Description

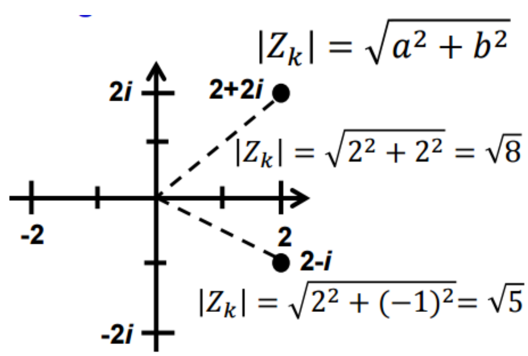
The *Mandelbrot Set* is a set of complex numbers that are quasi-stable when computed by iterating the function:

$$Z_k = C, \qquad k = 0$$

$$Z_k = Z_{k-1}^{\ 2} + C, \quad k \geq 1$$

- C is some complex number: $C = a + bi$
- Z_k is the k^{th} iteration of the complex number
- if $\left|Z_k\right| \leq 2$ for any k, C belongs to the *Mandelbrot Set*

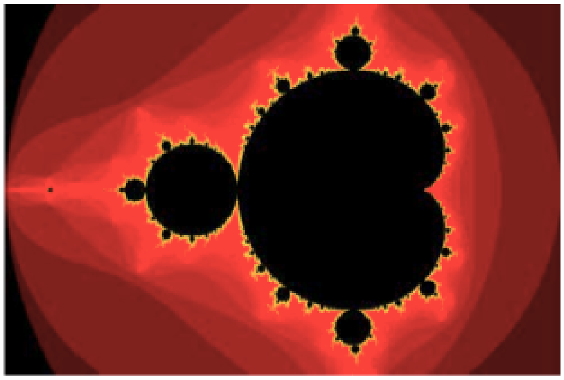




What exact is the *Mandelbrot Set*?

- 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:



For more information, please refer to lecture notes.

3 Input / Output Format

3.1 Input specification

The input parameters are specified from the command line. There are no input files.

Your programs should accept the following `srun` command:

```
srun -n $procs -c $t ./exe $out $iter $x0 $x1 $y0 $y1 $w $h
```

For example, the image in Sec.2 is created by:

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

The meanings of the arguments are:

- `$procs` – int; [1, 48]; number of processes. Always 1 for the Pthread version.
- `$t` – int; [1, 12]; number of threads per process. (technically, this is the number of CPUs you can use per process; you are allowed to use more or fewer threads)
- `$out` – string; the path to the output file.
- `$iter` – int; [1, 2×10^8]; number of iterations. (the largest int is around 2.1×10^9)
- `$x0` – double; [-10, 10]; inclusive bound of the real axis.
- `$x1` – double; [-10, 10]; non-inclusive bound of the real axis.
- `$y0` – double; [-10, 10]; inclusive bound of the imaginary axis.
- `$y1` – double; [-10, 10]; non-inclusive bound of the imaginary axis.
- `$w` – int; [1, 16000]; number of points in the x-axis for output.
- `$h` – int; [1, 16000]; number of points in the y-axis for output.

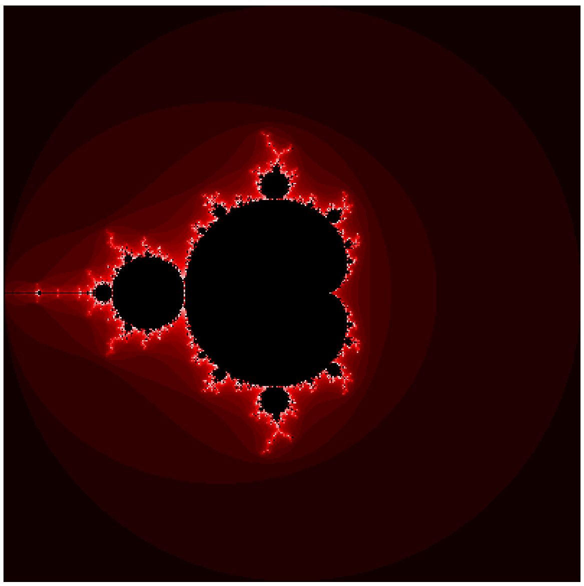
3.2 Output specification

Your programs should produce a PNG image at `$out`, visualizing the *Mandelbrot Set* in the given range.

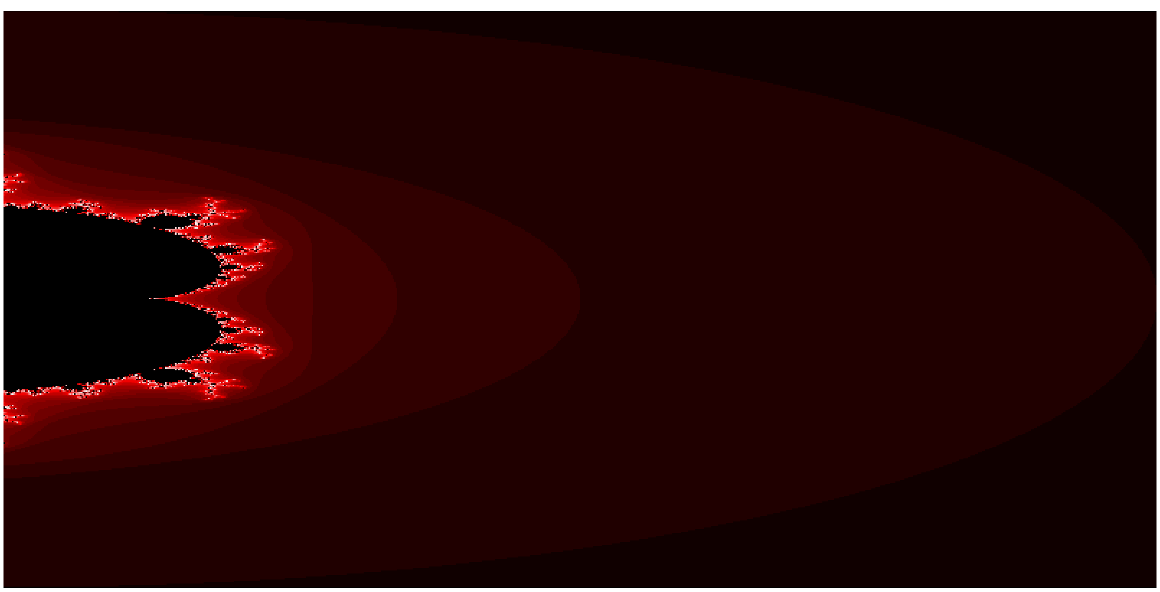
We provide a sequential version to show how the pixels are rendered.

Example 1: `srun ./exe $out $iter -2 2 -2 2 400 400`

x axis:[-2 2), 400 pints = {-2, -1.99, -1.98, … 1.98, 1.99}



Example 2: `srun ./exe $out $iter 0 2 -2 2 800 400`



4 Working Items

In this assignment, you are asked to parallelize the sequential *Mandelbrot Set* program by implementing the following two versions:

1. `pthread`: Single node shared memory programming using Pthread
    - This program only needs to be run on a single node.
2. `hybrid`: Multi-node hybrid parallelism programming using MPI + OpenMP
    - This program must be run across multiple nodes.
    - MPI processes are used to balance tasks among nodes, and OpenMP threads are used to perform computations.
    - Pthread library could also be used to create additional threads for communications.

Requirements:

- **Must follow the input/output file format and execution command line arguments specifications described in Section 3.**
- **No mathematical optimization is permitted.** That means the computations must be performed on each and every pixel.

5 Report

Your report must contain the following contents, and you can add more as you like.

1. Title, name, student ID

2. Implementation

Explain your implementations, especially in the following aspects:

- ✔ How you implement each of requested versions, especially for the hybrid parallelism.
- ✔ How do you partition the task?
- ✔ What technique do you use to reduce execution time and increase scalability?
- ✔ Other efforts you made in your program

3. Experiment & Analysis

Explain how and why you do these experiments? Explain how you collect those measurements? Show the result of your experiments in plots, and explain your observations.

i、 Methodology

(a). **System Spec** (If you run your experiments on your own machine)

Please specify the system spec by providing the CPU, RAM, storage and network (Ethernet / InfiniBand) information of the system.

(b). **Performance Metrics**

How do you measure computing time of your programs? How do you compute the values in the plots?

ii、 Plots: Scalability & Load Balancing & Profile

- Experimental Method:
  - Test Case Description: Explain the test data and its sizes you've chosen.
  - Parallel Configurations: Describe the number of processes and threads used, and how nodes and cores are distributed.
- Performance Measurement:
  - Use a profiler (like IPM) for performance analysis.
  - Provide basic metrics like execution time.
- Analysis of Results:
  - Display results generated by the profiler; this could be in the form of tables, charts, or other visualization tools.
  - Conduct strong scalability experiments, and plot the speedup and time profile results.
  - Show how balanced it is in each of your experiments by plots..
- Optimization Strategies:
  - Based on the analysis results, propose potential optimization strategies.
  - If optimizations have been implemented, provide a comparison of performance before and after the enhancements.
- The plot must contain at least 4 different scales (number of processes, threads) for both single node and multi-node environments.
- **Make sure your plots are properly labeled and formatted.**
- You are recommended to choose a proper problem size to ensure the experiment results are accurate and meaningful.

iii、 Discussion (must base on the results in the plots)

(a). Compare and discuss the **scalability** of your implementations.

(b). Compare and discuss the **load balance** of your implementations.

iv、 Others

- You are strongly encouraged to conduct more experiments and analysis of your implementations.

4. Experience & Conclusion

- ✔ Your conclusion of this assignment.
- ✔ What have you learned from this assignment?
- ✔ What difficulties did you encounter in this assignment?
- ✔ If you have any feedback, please write it here. Such as comments for improving the spec of this assignment, etc.

6 GRADING

1. **[45%] Correctness** (`pthread` version: 15%, hybrid version: 30%)
   - We will use several test cases to test your implementations. You will get full points for an implementation if you pass all the test cases, no points for it otherwise.
   - For each test case, your implementation will be considered correct if:
     - ❖ Your implementation produced a valid PNG image.
     - ❖ At least **99.8%** of the pixels in your output image are identical to the corresponding pixel produced by the sequential version.
     - ❖ You can use **`hw2-diff`** to check the difference between two png images. e.g. **`hw2-diff slow01.png myout.png`**
     - ❖ The **execution time** of your implementation is **shorter** than:
       - ➢ <u>the execution time of the sequential version</u> + <u>30 seconds</u>
2. **[15%] Performance** (`pthread` version: 5%, hybrid version: 10%)
   - We will use several different test cases (denoted C) to run your code.
   - Your performance score will be given by:

$$\sum S(C) \times \frac{T_{best}(C)}{T_{yours}(C)}$$

     - ❖ C is a test case: the set of input parameters excluding parallelism settings. e.g. `$x0`, `$x1`, `$y0`, `$y1`, `$w`, `$h`.
     - ❖ $S(C)$ is the score allocated to the test case.

- ❖ $T_{best}(C)$ is the shortest execution time of all students for the test case, excluding incorrect implementations.
- ❖ $T_{yours}(C)$ is your shortest execution time of {`pthread`, `hybrid`} for the test case, excluding incorrect implementations.
- ❖ $\sum S(C) = 15$

3. **[30%] Report**
   - Grading is based on your evaluation, discussion and writing. If you want to get more points, design or conduct more experiments to analyze your implementation.

4. **[10%] Demo**
   - i、 Explain your implementations.
   - ii、 Explain the key results and findings from your report.
   - iii、 (Optional) Your extra efforts. (Why do you deserve a higher score?)

5. **Policy**
   - i、 **0 points will be given to cheater** (even copying code from the Internet).
   - ii、 **No late submissions after the deadline will be accepted.**

7 Submission & Reminder

Upload these files to `eeclass` with **NO compression** before the deadline

- `hw2a.cc` (pthread version)
- `hw2b.cc` (hybrid version)
- `Makefile` (optional)
- `hw2_{student_ID}.pdf`

Note:

1. Deadline: **November 12, 2023 23:59**.
2. Refer to /home/pp23/share/hw2 on apollo for the sequential version of *Mandelbrot Set*, Makefile and test cases.

3. Your Makefile must be able to build the corresponding targets of the implementations: `pthread`, `hybrid`. If you're unsure how to write a Makefile, you can use the provided example Makefile as-is.

4. **Self-checking scripts** are provided to verify the correctness of your code. Type the following commands under your source code directory for testing: **`hw2a-judge`** for pthread version, **`hw2b-judge`** for hybrid version.

5. **A scoreboard system** will be available for you to checkout the current ranking of your implementation.
   https://apollo.cs.nthu.edu.tw/pp23/scoreboard/hw2a
   https://apollo.cs.nthu.edu.tw/pp23/scoreboard/hw2b

6. Resources are limited, start your work ASAP. Do not leave it until the last day!