CS542200 Parallel Programming
Homework 2: Mandelbrot Set
Revision 5
Due: Wed Nov 7 23:59, 2018

1 GOAL

This assignment helps you to:

- Get familiar with OpenMP.
- Know the differences between different memory architectures we've learned so far.
- Understand the importance of load balance.

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

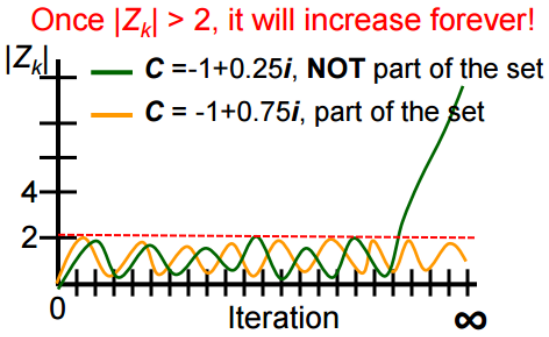
1. MPI with static scheduling
2. MPI with dynamic scheduling
3. OpenMP (scheduling of your choice)
4. Hybrid – MPI + OpenMP (scheduling of your choice)

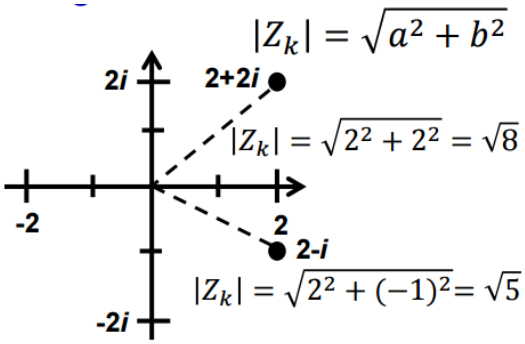
2 PROBLEM DESCRIPTION

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

$$Z_k = \begin{cases} \mathrm{C}, & \mathrm{k} = 0 \\ Z_{k-1}^2 + C, & \mathrm{k} \geq 1 \end{cases}$$

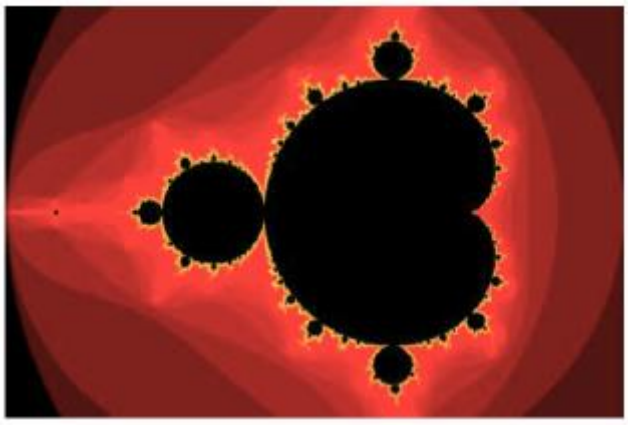
- C is some complex number: $\mathrm{C} = \mathrm{a} + \mathrm{b}i$
- Z_k is the k_{th} iteration of the complex number
- if $|Z_k| \leq 2$ for any k, C belongs to the *Mandelbrot Set*





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 $t $left $right $lower $upper $w $h $out
```

The meanings of the arguments are:

- `$procs` – int; [1, 48]; number of processes. Always 1 for the OpenMP version.
- `$t` – int; [1, 12]; number of threads per process. Always 1 for the MPI versions. (technically, this is the number of CPUs you can use per process; you are allowed to use more or fewer threads)
- `$left` – double; [-10, 10]; inclusive lower bound of the real axis.
- `$right` – double; [-10, 10]; non-inclusive upper bound of the real axis.
- `$lower` – double; [-10, 10]; inclusive lower bound of the imaginary axis.
- `$upper` – double; [-10, 10]; non-inclusive upper 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.
- `$out` – string; the path to the output file.

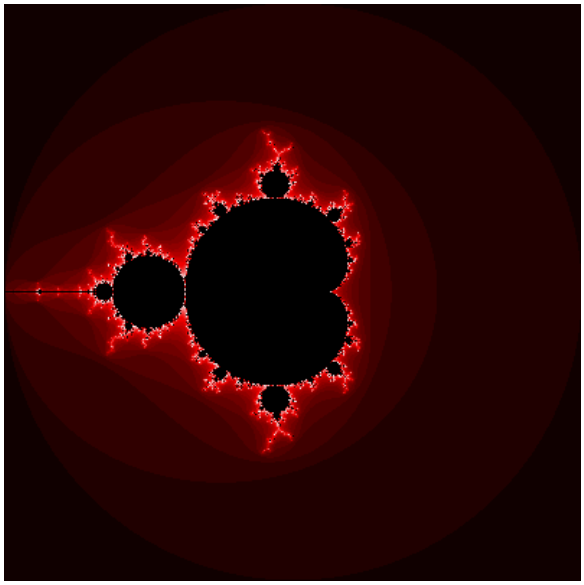
3.2 Output specification

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

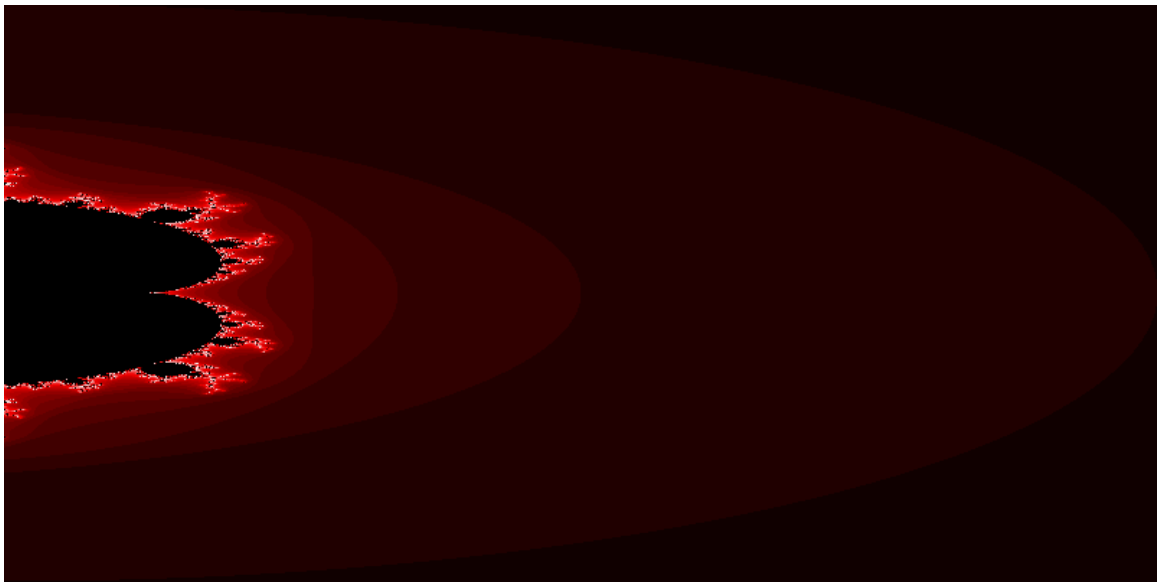
We provide a sequential version to show how the pixels are rendered.

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

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



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



4 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 and dynamic scheduling algorithm.
   - ✓ 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

   (a). Conduct **strong scalability** experiments, and plot the speedup. The plot must contain at least 4 different scales (number of processes, threads) for both single node and multi-node environments.

   (b). Design your own experiments to show how **balance** it is in each of your experiments by plots.

   (c). Make sure your plots are properly labeled and formatted.

   (d). 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

Conduct more interesting experiments to analyze performance. For example:

- In the hybrid version, hat is the best distribution between MPI tasks & threads, why?
- What's the best distribution of cores between nodes, why?

4. Experience & Conclusion

It could include these following aspects:

- ✓ Your conclusion of this assignment.
- ✓ What have you learned from this assignment?
- ✓ What difficulty 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.

5 GRADING

1. **Correctness** (50%)

You are required to implement four different versions of *Mandelbrot Set*:

- `mpi_static` – MPI with static scheduling (10%)
- `mpi_dynamic` – MPI with dynamic scheduling (15%)
- `openmp` – OpenMP with scheduling of your choice (10%)
- `hybrid` – MPI + OpenMP with scheduling of your choice (15%)

We will use several test cases to test your implementations. You get full points for an implementation if you passed 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 **99.6%** of the pixels in your output image are the identical to the corresponding pixel produced by the sequential version.
- The **execution time** of your implementation is **shorter** than:
  - the execution time of the sequential version + 30 seconds

2. **Performance** (15%)

   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. `$left`, `$right`, `$lower`, `$upper`, `$w`, `$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 {`mpi_static`, `mpi_dynamic`, `openmp`, `hybrid`} for the test case, excluding incorrect implementations.
   - $\sum S(C) = 15$

3. **Report** (25%)

   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. **Demo** (10%)

   i、 Explain your implementations.

   ii、 Explain the key results and findings from your report.

   iii、 (Optional) Your extra efforts. (Why do you deserve more bonus points?)

6 SUBMISSION

Upload these files to `apollo31`, and place them under `~/homework/hw2/`:

- `mpi_static.c`
- `mpi_dynamic.c`
- `openmp.c`
- `hybrid.c`
- `Makefile`

Upload this file to iLMS:

- `report.pdf`

Note:

1. You can also write your code in C++ by submitting `*.cc` files.
2. Your Makefile must be able to build the corresponding targets of the implementations: `mpi_static`, `mpi_dynamic`, `openmp`, `hybrid`. If you're unsure how to write a Makefile, you can use the provided example Makefile as-is.
3. Your submission time for your source code will be based on the time on `apollo31` and your submission time for your report will be based on iLMS.
4. Late submission policy:

`score *= 1`		Before deadline
`score *= 0.95`	After deadline	Before deadline+3days
`score *= 0.9`	After deadline+3days	Before deadline+7days
`score *= 0`	After deadline+7days	

7 REMINDER

1. Refer to iLMS for the location of the sequential version of *Mandelbrot Set*, Makefile and test cases.
2. Since we have limited resources, please start your work ASAP. Do not leave it until the last day!
3. 0 will be given to cheaters (even copying code from the Internet), but discussion on code is encouraged.
4. Asking questions through iLMS is welcomed!