The Chinese University of Hong Kong, Shenzhen

CSC4005

Parallell Programming

CSC4005 Project2
Mandelbrot Set Computation

Author:
Zijuan Lin

Student Number:
121090327

October 31, 2022

Contents

1 How to run the code

The submitted files include the pthread version and MPI version of Mandelbrot Set Computation.

To run the MPI version, one should first compile the **mpi.cpp** with
mpic++ mpi.cpp -o mpi -std=c++11
command to get text results, or compile with
mpic++ mpi.cpp -o mpig -I/usr/include -L/usr/local/lib -L/usr/lib -lglut -lGLU -lGL -lm -DGUI -std=c++11
to link GUI libraries in order to get an image as outcome as well.

After compilation, user can run the program by using command **mpirun -np 4 ./mpig 800 800 100** to get a result with text and image, or use the command **mpirun -np 4 ./mpi 800 800 100** to get plain result if the first way of compilation is done.

To run the pthread version, one should compile the **pthread.cpp** with
g++ pthread.cpp -o pthread -lpthread -O2 -std=c++11
to get the text results, or compile with
g++ pthread.cpp -o pthreadg -I/usr/include -L/usr/local/lib -L/usr/lib -lglut -lGLU -lGL -lm -lpthread -DGUI -O2 -std=c++11
to get an image as outcome as well.

After compilation, user can run the program by using command **./pthreadg 800 800 100** to get a result with text and image, or use the command **./pthread 800 800 100** to get plain result if the first way of compilation is done.

Results:

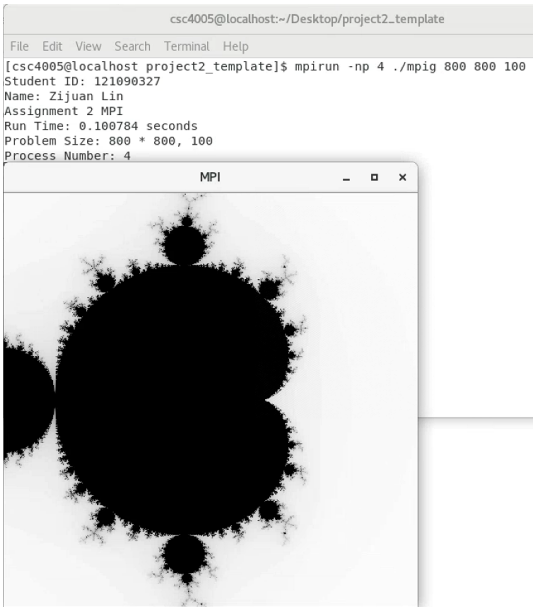

Figure 1: The text an image result of MPI version

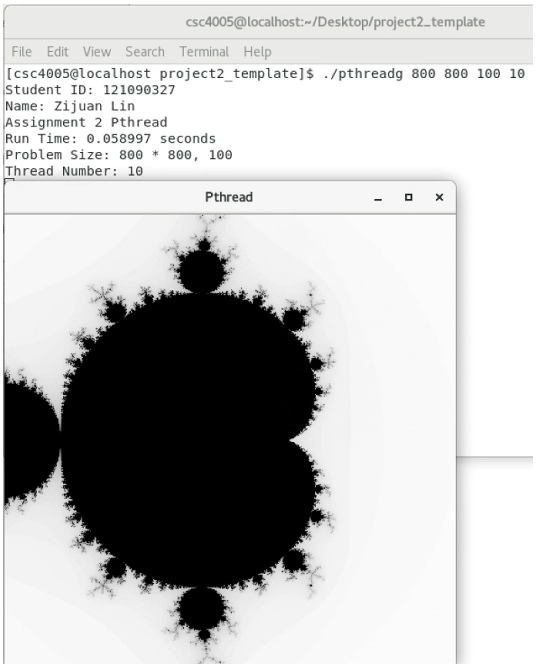

Figure 2: The text an image result of pthread version

2 Problem Statement

Mathematical basics In the project, we are required to calculate the Mandelbrot Set parallelly. The user will provide the width(**Y_RESN**) and height(**Y_RESN**) of the resulting graph as well as the max iteration value**max_iteration**. The iteration of each point z in complex plane is

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

Here c is provided by This iteration will end when $|z| \geq 2$ or $n + 1 = max\_iteration$, and the "color" of point $(z.real, z.imag)$ in the resulting graph is $n/max\_iteration$.

3 Program Design

MPI Version In the MPI version, we construct a struct as follows.

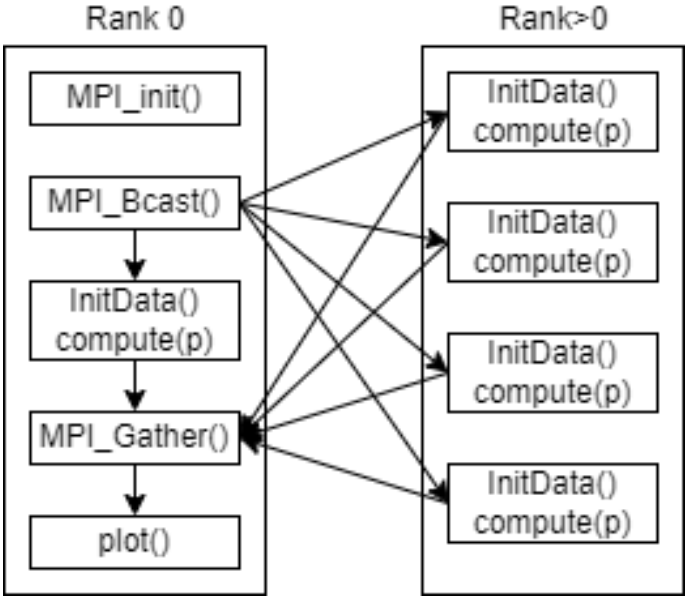

Figure 3: The structure of MPI version program

After **MPI_init()** and conduct point initialization in each rank, rank 0 will broadcast the number of points(which is **slice** in the code below) need to be calculated by each rank. Then each rank will start to calculate the "color" of each point and store the values in an array. After the calculation, the arrays will be collected by **MPI_Gather()** and be assigned to the points in rank 0 and plot the resulting image. The core part of calculation conduct in each rank is as follows:

```
float *res=new float[slice];
Point *p=data;
p+=rank*slice;

for (int i=0;i<slice;i++){
    compute(p);
    res[i]=p->color;
    p++;
}
```

By adding **rank*slice** to **p** we can let each rank calculate the correct fraction of points.

Pthread Version In the pthread version, we use a struct **Args** to assign the tasks need to be executed by each thread. Thread can share the same memory space, so we don't need to collect the value and merge them. The structure of pthread version is as follows:

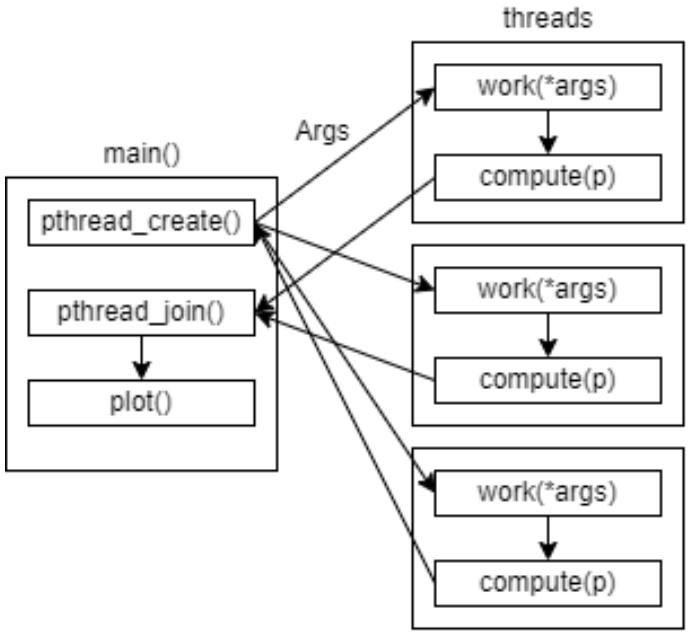

Figure 4: The structure of pthread version program

The division of points to be calculated by each thread is calculated by the below fraction of code.

```
Args* my_arg = (Args*) args;
```

```
int thd = my_arg->a;
int total = my_arg->b;
int slice;
if (total_size%total==0)
    slice=total_size/total;
else
    slice=(total_size/total)+1;
p+=thd*slice;
```

4 Experimental Results

To test the performance of programs as well as the advantage over sequential version, we conduct several experiments.

Baseline The baseline of the experiment is the running time of sequential program.

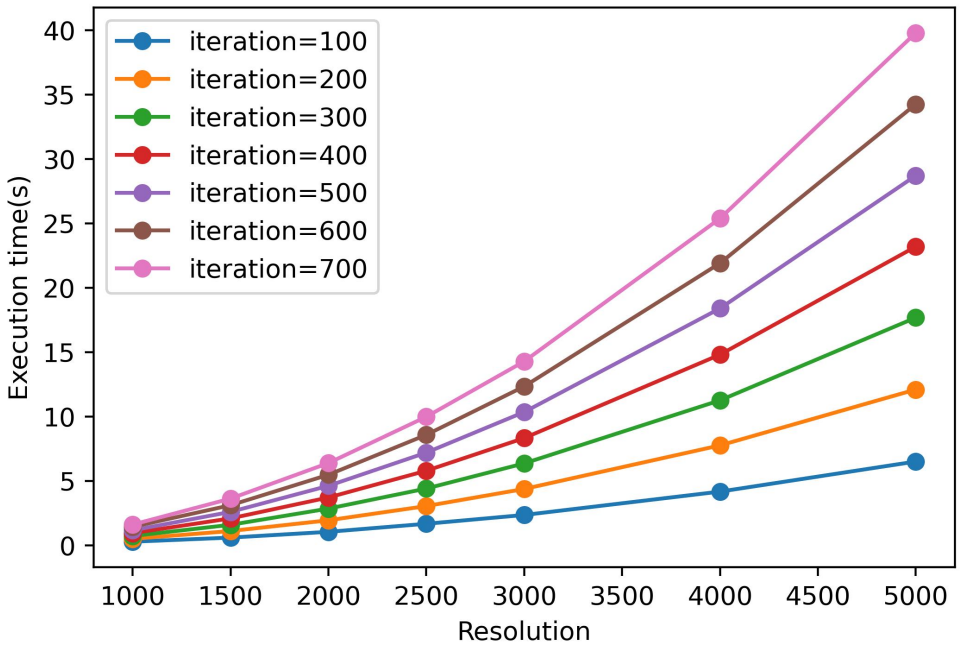

Figure 5: The running time of sequential version

MPI version To test the MPI version, we've run the program using [4,6,8,10,12,14,16,18,20] processes in HPC. The results are as follows.

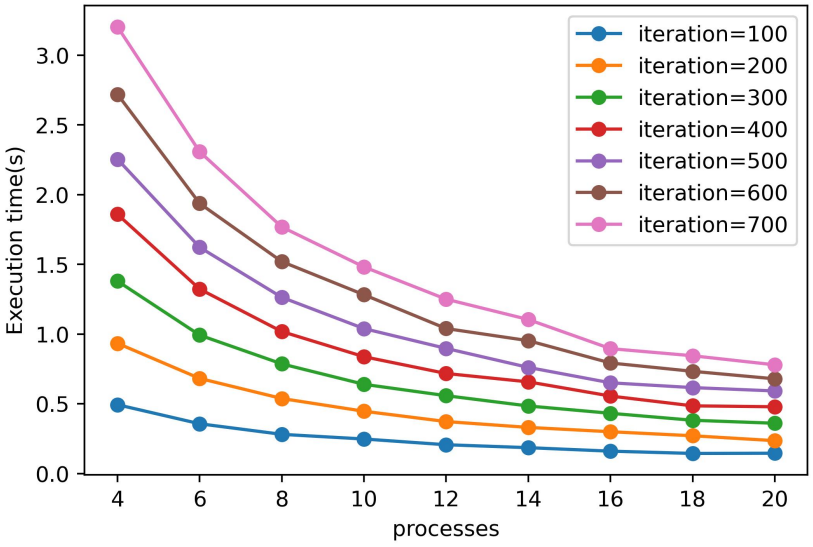

Figure 6: The running time of MPI version when **X_RESN=Y_RESN=2000**

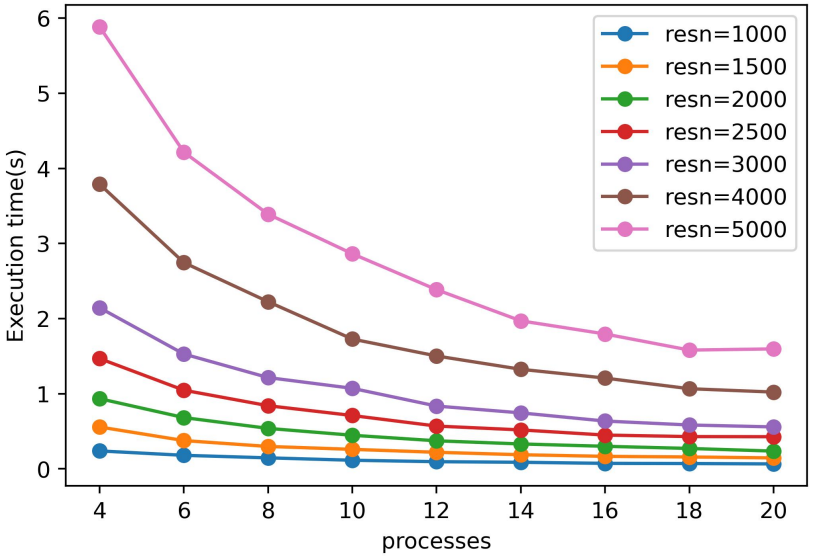

Figure 7: The running time of MPI version when **max_iteration=200**

Pthread version To test the performance of pthread implementation, we've tested the cases of thread=[1,2,4,6,8,10,12,14,16,18,20,40,60,80,100].

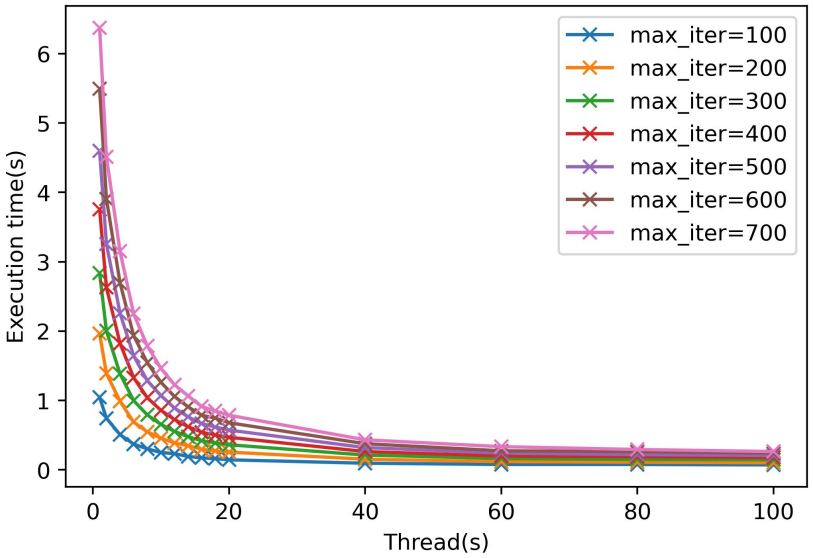

Figure 8: The running time of pthread version when **X_RESN=Y_RESN=2000**

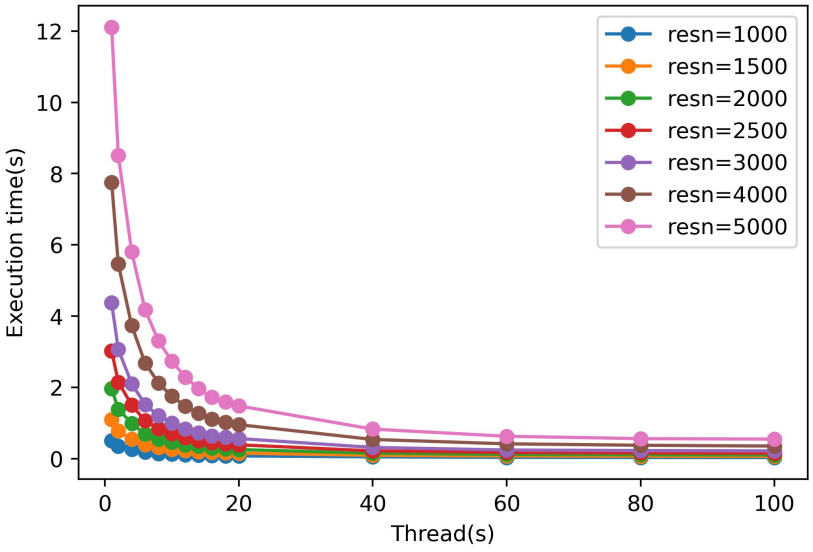

Figure 9: The running time of pthread version when **max_iteration=200**

5 Result Analysis

MPI version From the above figures, we can see that the parallel strategy has accelerated the computation. However, from the following figure we can see that the acceleration effect is not so significant, though the effect is linear. This results from the unbalanced workload. The operation of assigning the calculated colors into points in rank 0 can be one of the main issue.

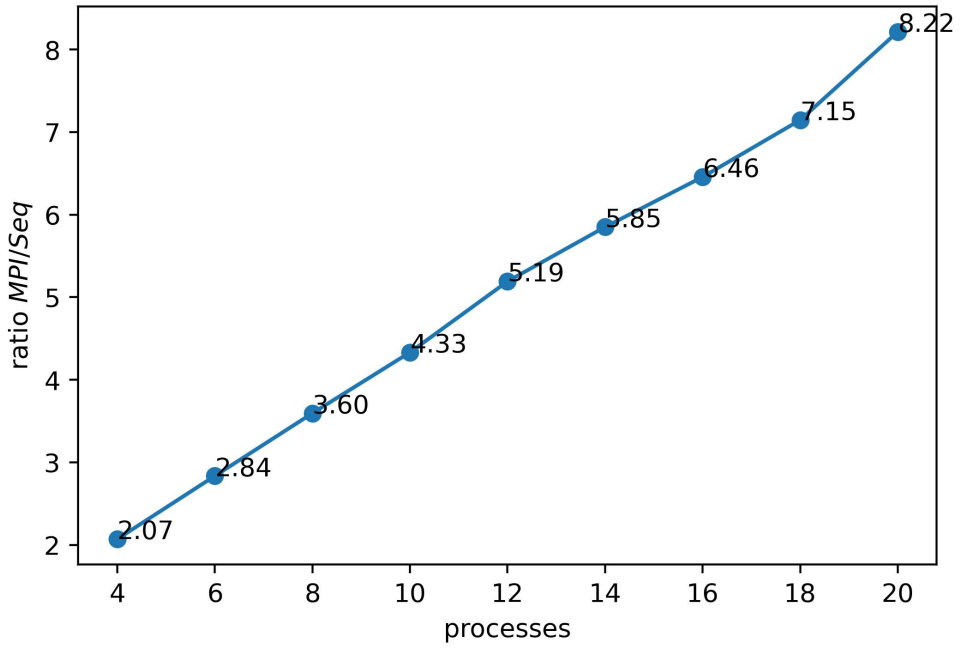

Figure 10: The time ratio of MPI implementation and sequential implementation when X_RESN=Y_RESN=2000, max_iteration=200

Pthread version From the above figures we can observe a steeper curve in the Execution time-Thread graph than MPI implementation, which shows a significant improvement in computing speed when the number of involving threads is small, as well as a much stronger marginal effect as the number of thread increases. This is caused by the delay in creating and joining threads has becoming a bigger fraction in total time cost as number of threads involving in the computation increases.

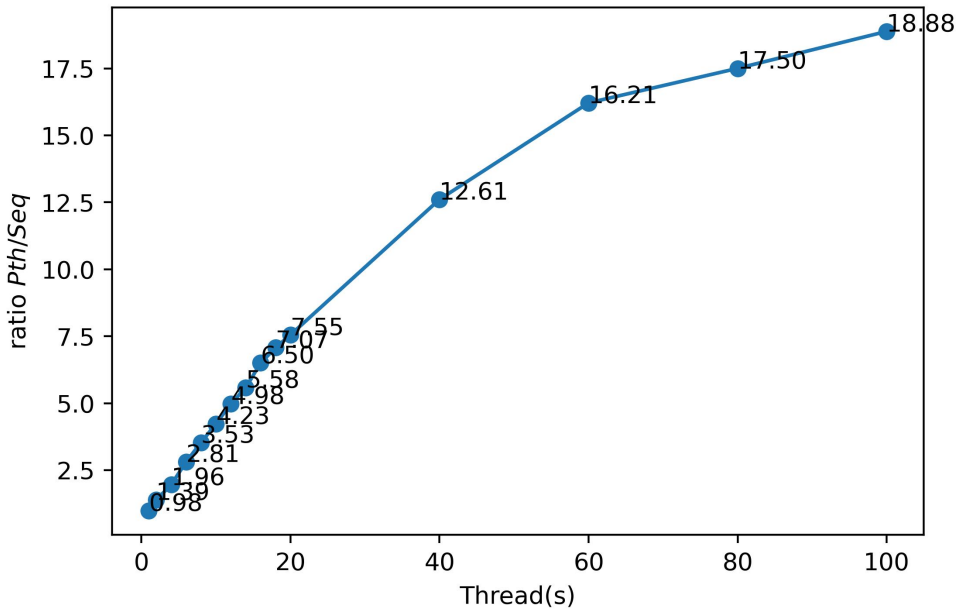

Figure 11: The execution time ratio of pthread implementation and sequential implementation when X_RESN=Y_RESN=2000, max_iteration=200

Pthread VS MPI We've plot a figure to show the acceleration effect of MPI implementation and Pthread implementation. Compared with MPI, we can see that pthread implementation can achieve similar result as using MPI. However, since threads share the same memory space, the program will not need to transfer a big array of result as MPI implementation does. Thus as pthread implementation will have a more significant time reduction as the number of thread increases before the marginal effect becomes too significant.

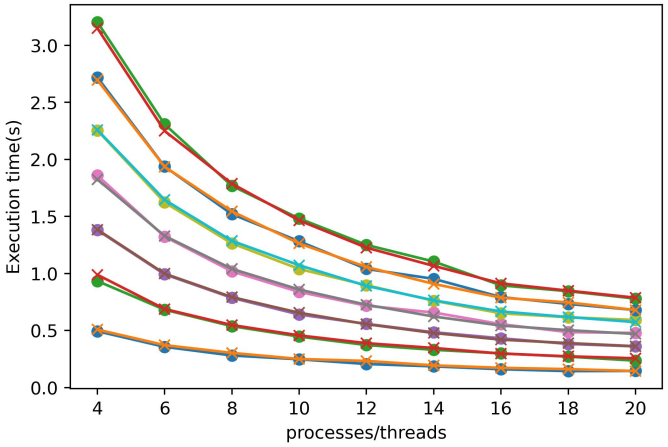

Figure 12: The time ratio of MPI implementation and Pthread implementation when X_RESN=Y_RESN=2000.
From top to bottom the MAX˙ITERATION is [700,600,500,400,300,200,100]

6 Conclusion

From the experiment we can see the implementation of multi-thread and multi-process strategies can accelerate a complex of computation, which suggests the great potential in image processing. However, we can see that both of the implementation cannot reach the goal of $t_{para} \cdot process = t_{seq}$, which means they are not so effective and still have potential to improve the performance. The key is to reduce the sequential tasks conducted in a single rank(or thread) and distribute them to others. For example, we can try to draw the figure seperately by each rank(or thread) and merge them together in one window.

7 Suffix

Raw data of experiments are provided in .xlsx files.

The **MPI_res.xlsx** provides the data of experimenting MPI implementation.
The **pth_res.xlsx** provides the data of experimenting pthread implementation.