CSC4005: Distributed and Parallel Computing

Lab 2: Mandelbrot Set Computation

Name: 陈雅茜

Student ID: 117010032

Date: Nov 3rd, 2020

香港中文大學（深圳）

THE CHINESE UNIVERSITY OF HONG KONG, SHENZHEN

Content

1. Introduction

Message Passing Interface (MPI) is a standardized and portable message-passing standard designed by a group of researchers from academia and industry to function on a wide variety of parallel computing architectures. POSIX Threads, usually referred to as pthreads, is an execution model that exists independently from a language, as well as a parallel execution model. It allows a program to control multiple different flows of work that overlap in time.

The main differences between MPI and Pthread is that, Pthread is based on the share memory while MPI is for distributed-memory communication.

In this project, in order to master the multiprogramming skill, we demonstrate the Mandelbrot Set in sequential structure, parallel structure through MPI, and parallel structure through Pthread architecture. The Mandelbrot set is the set of values of c in the complex plane for which the orbit of the critical point z = 0 under iteration of the quadratic map $z_{k+1} = z_k^2 + c$ remains bounded.Thus, a complex number c is a member of the Mandelbrot set if, when starting with $z_0 = 0$ and applying the iteration repeatedly, the absolute value of z_n remains bounded for all k > 0.

In this report, the context totally has eight different sections, design, test, compile, result, performance analysis, conclusion, experience and bonus. In the design part, the author detailly introduces the implementation of the project test part provides the designing detail for testing algorithm; compile part gives the instruction to run our project; result part shows the project output and corresponding running time for both sequential sorting and parallel sorting algorithm; performance analysis shows the possible reasons for the result, bonus part includes all the detail for implementing the dynamic version of project, and in the appendix, all the codes are enclosed. The project is implemented on C++ language and is tested on Linux OS, ubuntu 18.04.

2. Design

In this section, the report gives the detailed design for Mandelbrot Set demonstration under sequential, MPI, and Pthread architecture.

2.1 Sequential Architecture

The overall flow for Mandelbrot Set demonstration under sequential architecture mainly has three sections: window initialization, point calculation and point draw.

2.1.1 Window Initialization

In the window initialization section, we set the parameters for window, background, and foreground. The code for this section is shown in below.

The process is divided into eight main sections: set the window variables, set the Mandlebrot variables, connect to Xserver, get screen size, set window size, set window position, create opaque window, and create graphics context. You have to operate them sequentially.

2.1.2 Point Calculation

In this section, we verify that if the given pixel belongs to Mandelbrot Set or not one by one by applying the iteration repeatedly, to see if the absolute value of z_n remains bounded for all n > 0. Since it is not pratical to check if the value is bounded or not using infinite iterations, in this project, we assume that the value is bounded if its magnitude is always smaller than the given threashold in the first m iterations, where m is the given number of maximum iteration by the end-user.

According to the Mandelbrot Set definition, in each iteration, the z_n value will be updated based on the following formula:

$$z_{k+1} = z_k^2 + c$$

since the definition of z_k is $z_k = a + bi$, we can further simplify the above formula into

$$z_{k+1} = (a + bi)^2 + c$$

$$z_{k+1} = a^2 + 2abi + b^2 i^2 + c$$

$$z_{k+1} = a^2 + 2abi - b^2 + c$$

therefore, the real part for z_{k+1} value should be $z_{k+1,real} = z_{k,real}^2 - z_{k,imag}^2 + c_{real}$
the imaginary part for z_{k+1} value should be $z_{k+1,imag} = 2z_{k,real}z_{k,imag} + c_{imag}$
The magnitude of z_k is defined as

$$z_k = \sqrt{a^2 + b^2}$$

As defined the initial value for c is

$$c = \frac{x - height/2}{height/4} + \frac{y - width/2}{width/4} \times i$$

The pseudocode for iteration calculation is shown in below

```
for every_pixel in the window:
    initialize result z_0 to be 0;
    initialize c;
    initialize count number k to be 0;
    do  {                                              /* iterate for pixel color */
    calculate z_k+1 according to c and z_k;
    calculate magnitude of z;
    count++;
    } while (lenth < threshold && count < MAX_iteration);
```

2.1.3 Point Draw

In this section, the corresponding pixel whose complex value is in Mandelbrot Set will be turned to black on the window with the API XDrawPoint. Since, in each iteration for each pixel, the value k will be added by 1, and the iteration terminates when the complex maginitude gets beyond the threshold or the maximum number of iterations have been achieved, if the pixel value k equals to the maximum number of iterations, the point satisfies the requirement to be in Mandelbrot Set.

The pseudocode in this section is shown in below

```
if (count number == MAX_iteration) XDrawPoint (pixel);
```

2.1.4 Overall Flow

The flow is shown in below

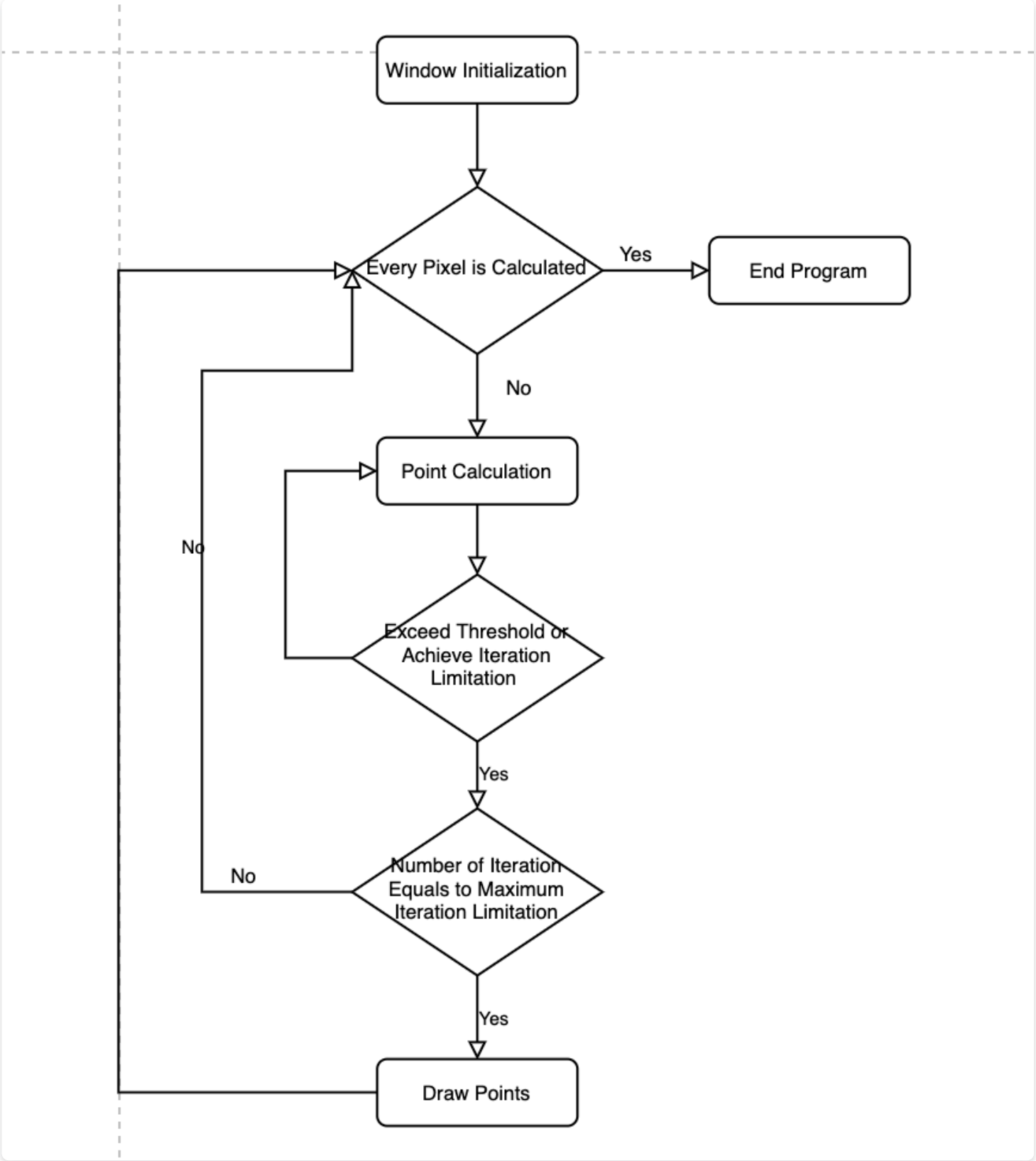

Fig.1 Flow for Sequential Approach

2.2 MPI Architecture

The design for Mandelbrot Set demonstration under MPI architecture is almost the same with that of sequential version, the only difference is that under the MPI architecture, in the point calculation we divide the task into several chunks and assign each chunk to different processors. After calculating the value k for each pixel, we gather the value from different processors and use the master processor to draw the figure.

2.2.1 Point Calculation

This section shows the point calculation using MPI architecture with static pixel division. Assume that the number of processors used in this project is N and the number of pixels is $W \times H$. In our design, the first N-1 processors are assigned with $\lfloor W \times H \div N \rfloor$ pixels, the last processor is assigned with $W \times H - \lfloor W \times H \div N \rfloor \times (N-1)$ pixels. For examples, if $W = H = 20$ and the number of processors equals to 3, the first 2 processors have to calculate 133 pixels while the last one have to calculate 134 pixels.

The pseudocode for this section is shown in below

```
    initialize the parameters;
        for (every pixel in the chunk){
        initialize result z_0 to be 0;
        initialize c;
        initialize count number k to be 0;
        do  {
        calculate z_k+1 according to c and z_k;
        calculate magnitude of z;
        count++;
        } while (lenth < threshold && count < MAX_iteration);
        put into the local array;
    }
}
    gather array from slave into master;
```

2.2.2 Overall Flow

The overall flow for MPI approach is shown below

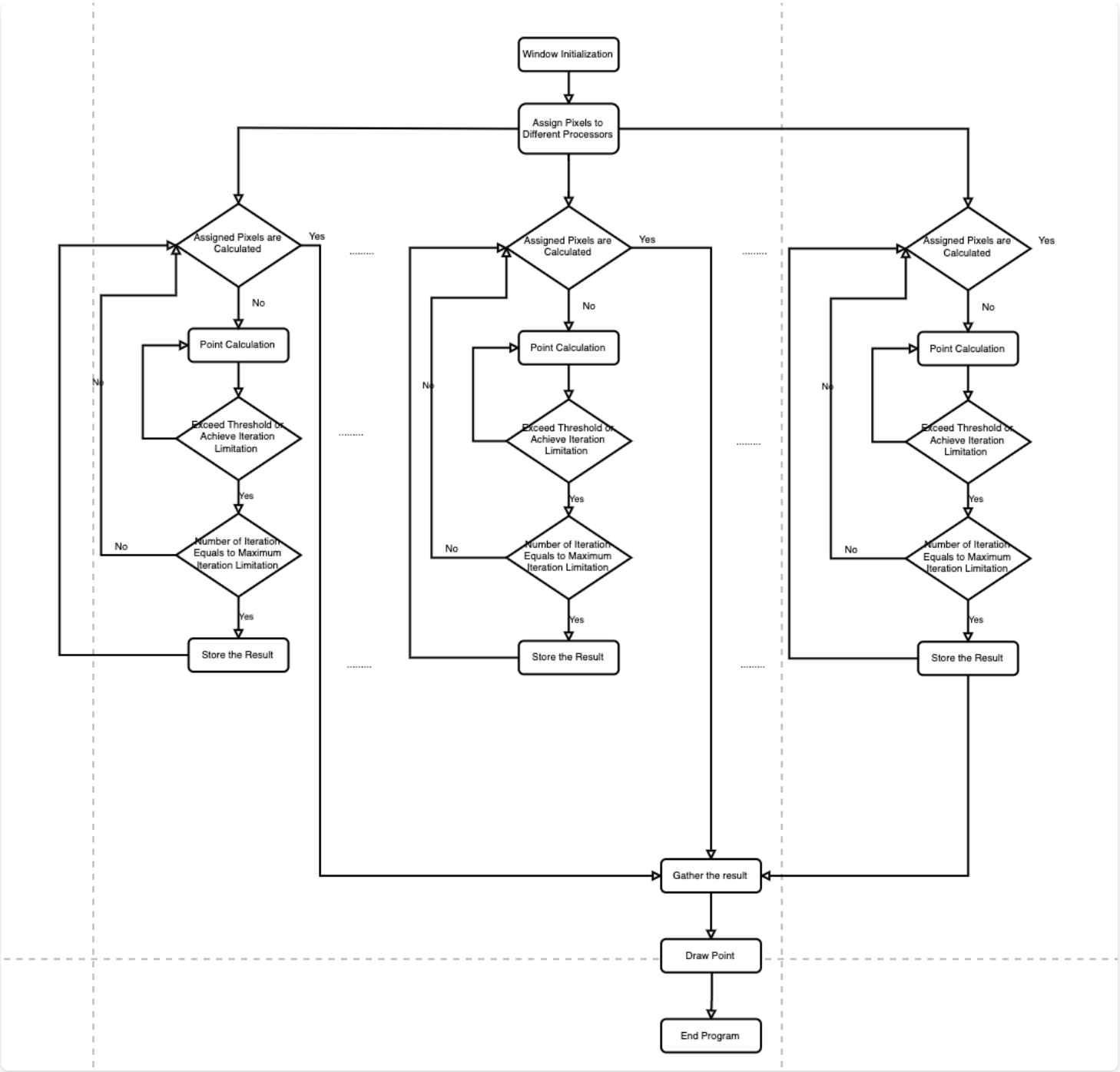

Fig.2 Flow for MPI Approach

2.3 Pthread Architecture

The idea for Mandelbrot Set demonstration under Pthread architecture is almost the same with that of MPI version, except that instead of assigning pixels to different processors, we assign pixels to different threads using pthread_create() API. Another difference between MPI version and Pthread version is that, instead of calculating the result separately and gather the result to the master processor to draw the figure, in the Pthread version, we let each thread to draw the figure on their

own. However, it will cause some problems if more than one thread is drawing the figure. In this case, we use the lock and unlock to ensure that only one thread is drawing the figure at a time.

The pseudocode for this section is shown below

```
    for (every thread){
        creat input data;
        creat thread and put the data into the thread;
        do the Mandelbrot_calc;
        if (rc){
            check thread creation;
            return EXIT_FAILURE;
        }
    }
    for (every thread){
        join the thead;
    }
}

void* Mandelbrot_calc(void* arg){
    initialize the parameters;
    for (every pixel in the chunk){
        initialize result z_0 to be 0;
        initialize c;
        initialize count number k to be 0;
        do  {
        calculate z_k+1 according to c and z_k;
        calculate magnitude of z;
        count++;
        } while (lenth < threshold && count < MAX_iteration);
        put into the local array;
    }
        pthread_mutex_lock(&mutex_draw);
        draw pixel if count == MAX_Iteration;
        pthread_mutex_unlock(&mutex_draw);
        }
    end the thread;
}
```

2.3.2 Overall Flow

The overall flow for Pthread approach is shown below

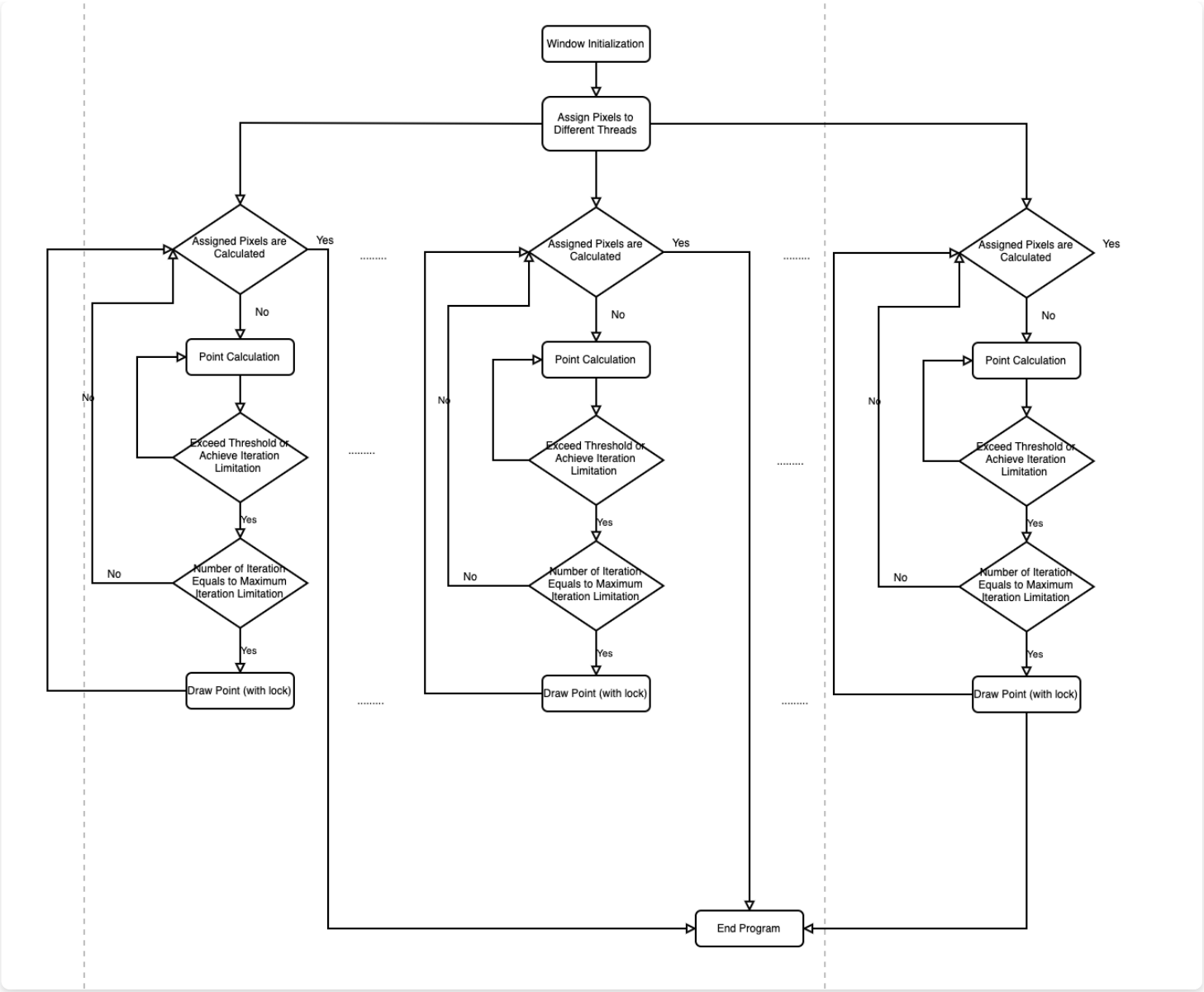

Fig.3 Flow for Pthread Approach

3. Test

In this section we briefly introduce the methods for testing. There are roughly three different aspects for testing, first is test for different number of threads and processor, the second is test for different window size, the third is test for different maximum iteration.

3.1 Test for Different Number of Threads and Processors

In order to compare the performance under sequential, MPI, and Pthread architecture, and observe the performance trendency with varying number of threads/processors. In the first type of test, we test the project with fixed window size, maximum iteration, and threshold with varying number of threads. The number of threads and processors from 1 to 20.

3.2 Test for Different Window Size

To see the influence of window size, in this section we test the project with fixed number of threads/processors (set to be 4) and fixed maximum limitation (set to be 100) with varying window size under Pthread, MPI, and sequential architure. We test the window size from 100×100 to 800×800.

3.3 Test for Different Maximum Iteration

To see the influence of maximum iteration, in this section we test the project with fixed number of threads/processors (set to be 4) and fixed window size (800×800) with maximum limitation under Pthread, MPI, and sequential architure. We test the maximum limitation from 50 to 800.

4. Compile

In this section, we introduce the detailed instruction to compile the project.

4.1 Compiling

The compiling command is taken under the direction \home\stu_id. For mac user, it is required to first install XQuartz and do the following command in XQuartz terminal.

```
g++ Mandelbrot_Set_Sequential.cpp -lX11 -o /code/stu_id/Mandelbrot_Set_Sequential
mpic++ Mandelbrot_Set_MPI.cpp -lX11 -o /code/stu_id/Mandelbrot_Set_MPI
g++ Mandelbrot_Set_Pthread.cpp -lX11 -o /code/stu_id/Mandelbrot_Set_Pthread -lpthread
```

Then the executable file transportation_sort and transportation_sort_seq are generated under /code/stu_id direction.

4.2 Running

- Under /code/stu_id direction, user should type "touch script.pbs" to create a script file.

```
touch script.pbs
```

- Then type the command "vim script.pbs" to modify the script.pbs file.
- Type the command which is similar to the below line to execute the file.

```
#!/bin/bash
#PBS -l nodes=1:ppn=5,mem=1g,walltime=00:05:00
#PBS -q batch
#PBS -m abe
#PBS -V
timeout 60 mpiexec -n $j -f /home/mpi_config /code/117010032/Mandelbrot_Set_MPI $W $H $M
```

In which, $j stands for the number of processors, $W stands for the window width,$H stands for the window height, $M stands for the number of maximum iteration, and $T stands for threshold. With the same idea the .pbs file to execute the Pthread version is shown in below.

```
#!/bin/bash
#PBS -l nodes=1:ppn=5,mem=1g,walltime=00:05:00
#PBS -q batch
#PBS -m abe
#PBS -V
timeout 60 /code/117010032/Mandelbrot_Set_Pthread $j $W $H $M $T
```

$j stands for the number of threads. The .pbs file to execute the Pthread version is shown in below.

```
#!/bin/bash
#PBS -l nodes=1:ppn=5,mem=1g,walltime=00:05:00
#PBS -q batch
#PBS -m abe
#PBS -V
timeout 60 /code/117010032/Mandelbrot_Set_Sequential $W $H $M $T
```

Then the end user is able to execute the .pbs file with the command

```
qsub script.pbs
```

4.3 Checking the State

To check the state of the project, the command shown in below will be used

```
qstat
```

Than the system will automatically return the state of all the tasks run on this server. The possible return is shown in the figure10. There are totally three different types of states, C, R and Q, in which C represents for completion, R stands for running and Q stands for waiting in the queue.

```
117010032@master2:/code/117010032$ qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2969.master2              m_10000.pbs      117010031       00:09:02 C batch
2970.master2              m_10000.pbs      117010031       00:00:00 R batch
```

Fig.4 Result for State Checking

If you don't need the task anymore, the command to kill the task is shown in below

```
qdel < queue id >
```

4.4 Checking the Output

With the execution command, the system will generate two files under the foler of execution files. script.pbs.o < queue id > is used to store the output information. Due to the security concern, in order to open the file, you should first change the limits power. Chmod 777 allows everyone to read, delete and modify the file.

```
chmod 777 script.pbs.o < queue id >
```

Then you are able to open and modify the file with the command

```
vim script.pbs.o < queue id >
```

With the same idea, users are able to open the script.pbs.e < queue id > file which contains the error information using the below command

```
chmod 777 script.pbs.e < queue id >
vim script.pbs.e < queue id >
```

5. Result

In this section, we have a subsection for output snapshot and three different subsections which are correspond to the three subsections under the test part.

5.1 General Result for Project

In this subsection, it gives the general project output. Figure11 gives the output with widow size 800×800 under sequential structure. Figure12 and Figure13 gives the output with widow size

800×800 and 4 processors under MPI and Pthread structure separately.

```
root@ecs-kc1-large-2-linux-20200730225035:~/cyq# ./Mandelbrot_Set_Sequential 800
 800 100 4
Name: Yaqian Chen
Student ID: 117010032
Assignment 2, Mandelbrot Set, Sequential Implementation
Execution Time is: 0.339168 seconds.
```

Fig.5 Result for General Output under Sequential Structure

```
root@ecs-kc1-large-2-linux-20200730225035:~/cyq# mpiexec -n 3 ./Mandelbrot_Set_M
PI 800 800 100 4
Name: Yaqian Chen
Student ID: 117010032
Assignment 2, Mandelbrot Set, MPI Implementation
Execution Time is: 0.401491 seconds.
```

Fig.6 Result for General Output under MPI Structure

```
root@ecs-kc1-large-2-linux-20200730225035:~/cyq# ./Mandelbrot_Set_Pthread 3  800
 800 100 4
Name: Yaqian Chen
Student ID: 117010032
Assignment 2, Mandelbrot Set, Pthread Implementation
Execution Time is: 0.129945 seconds.
```

Fig.7 Result for General Output under Pthread Structure

The output figure with window size of 800×800, maximum number of iteration set to be 100 and threshold set to be 4 is shown in below.

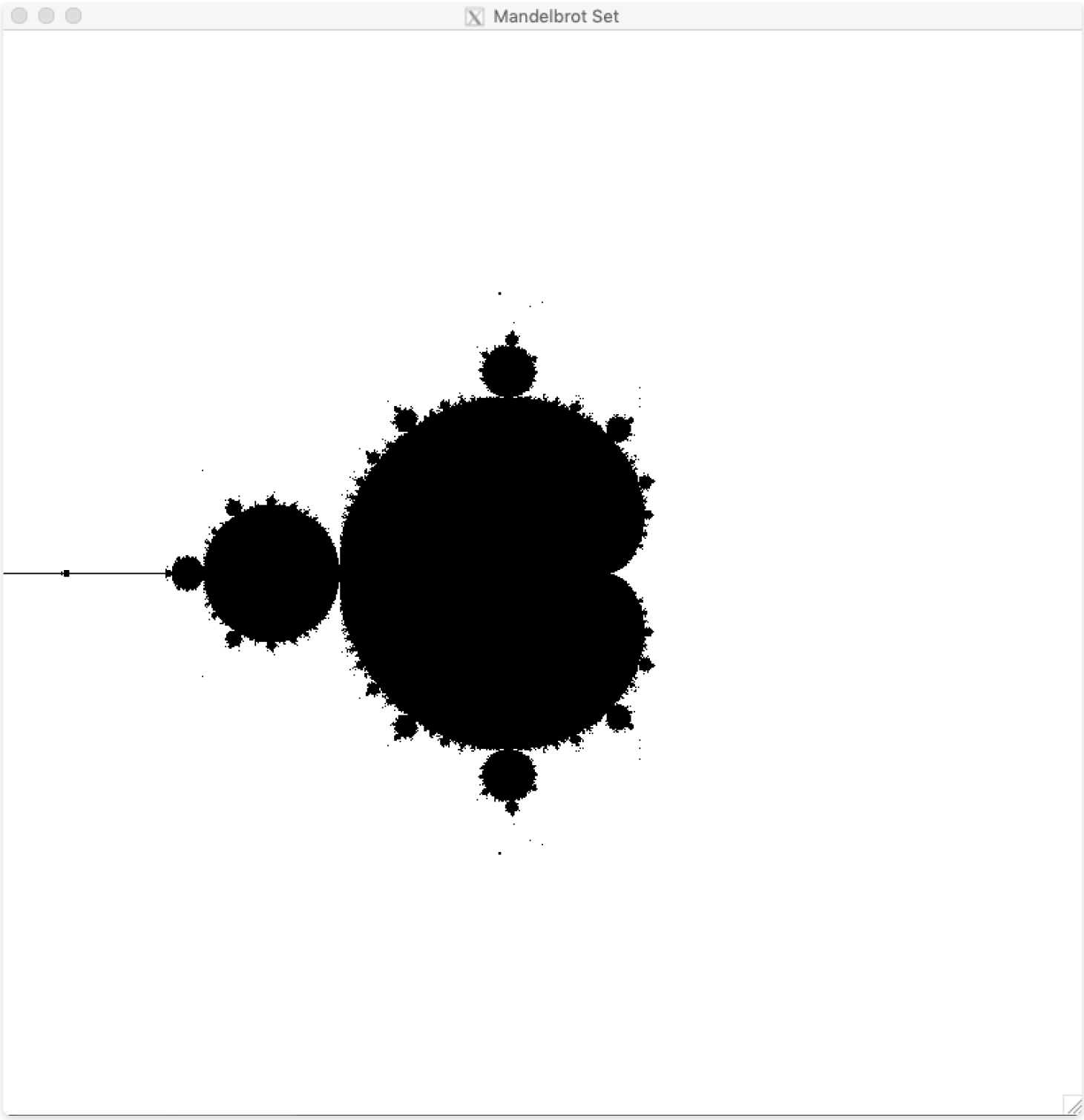

Fig.8 Output Figure for Mandelbrot set

5.2 Result for Different Number of Threads and Processors

This section displays the performance result for fixed window size, maximum iteration, and threshold with varying number of threads under Pthread architure and varying number of processors under MPI architure. The performance with the same set of sequential version is also included to give

the comparison. We test the number of threads and processors from 1 to 20. The result is shown in below.

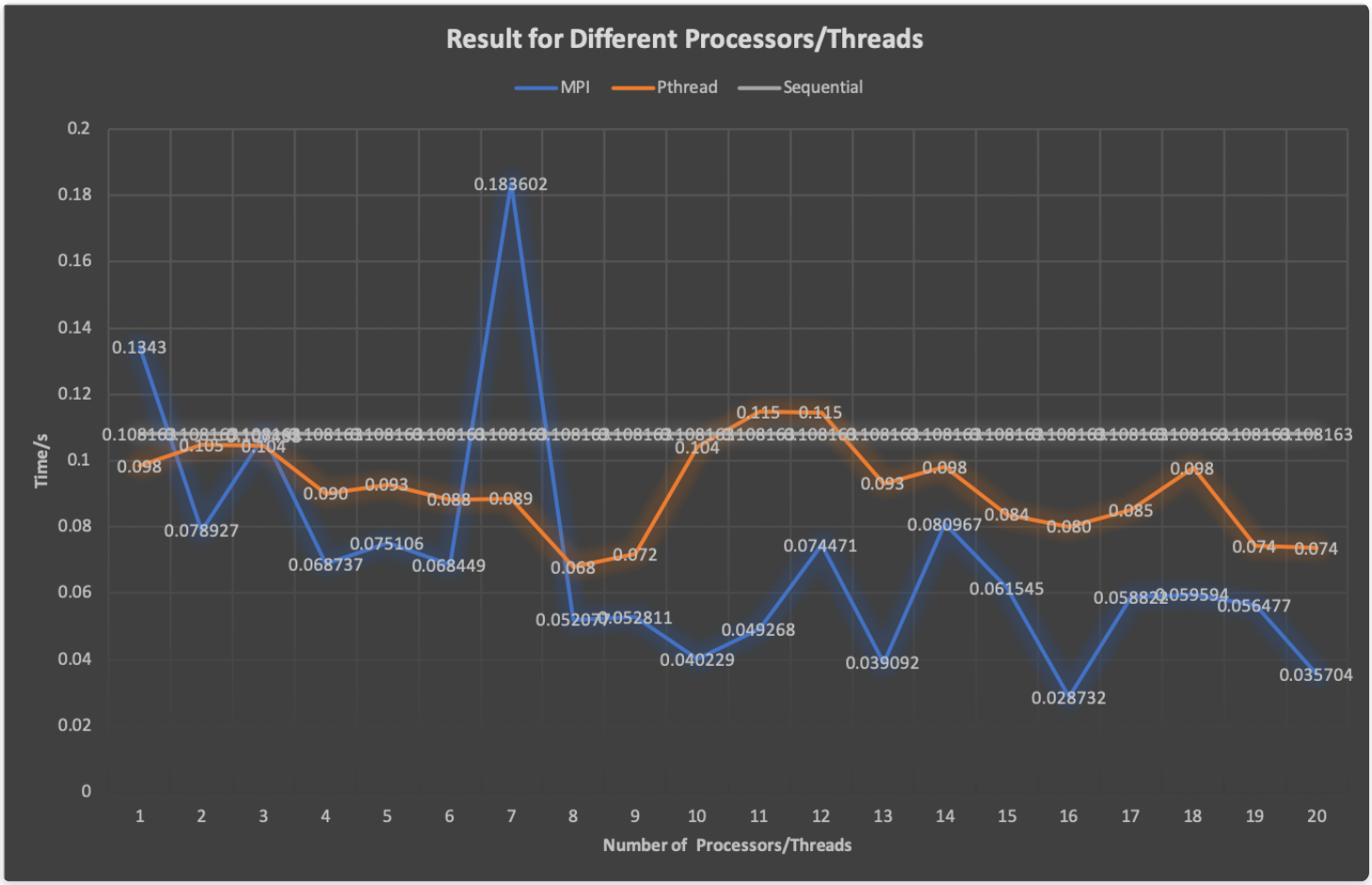

Fig.9 Result for Different Number of Processors/Threads

In figure 9, x axis represents the number of processors/threads, the y axis represents the running time. The blue line represents the performance under the MPI architecture, Orange line represents that under Pthread architecture while the grey line represents the Sequential one.

5.3 Result for Different Window Size

This section displays the performance result for fixed number of threads/processors (set to be 4) and fixed maximum limitation (set to be 100) with varying window size under Pthread, MPI, and sequential architure. We test the window size from 100×100 to 800×800. The result is shown in below.

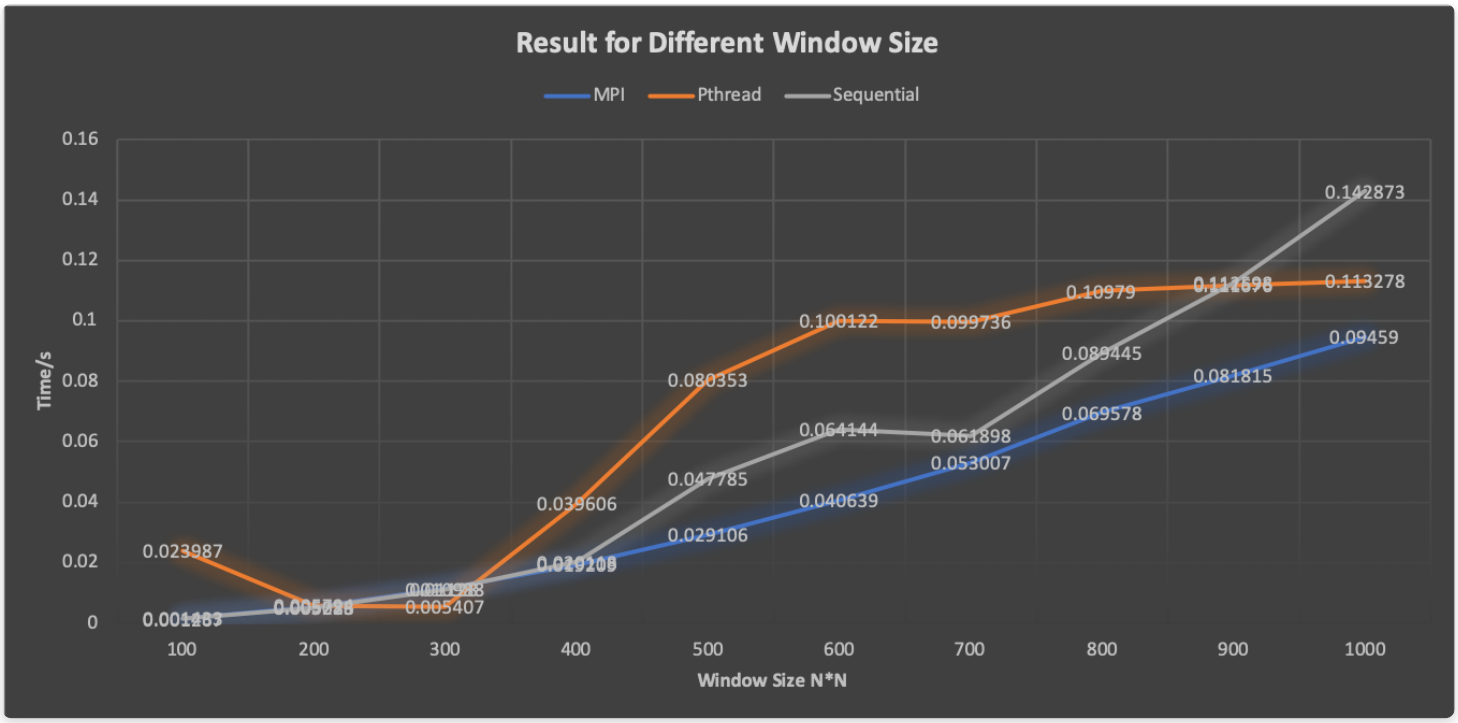

Fig.10 Result for Different Window Size

In figure 10, x axis represents the height/width, with x value N, the window size should be $N \times N$, the y axis represents the running time. The blue line represents the performance under the MPI architecture, Orange line represents that under Pthread architecture while the grey line represents the Sequential one.

5.4 Result for Different Maximum Iteration

This section displays the performance result for fixed number of threads/processors (set to be 4) and fixed window size (800×800) with maximum limitation under Pthread, MPI, and sequential architure. We test the maximum limitation from 50 to 800. The result is shown in below.

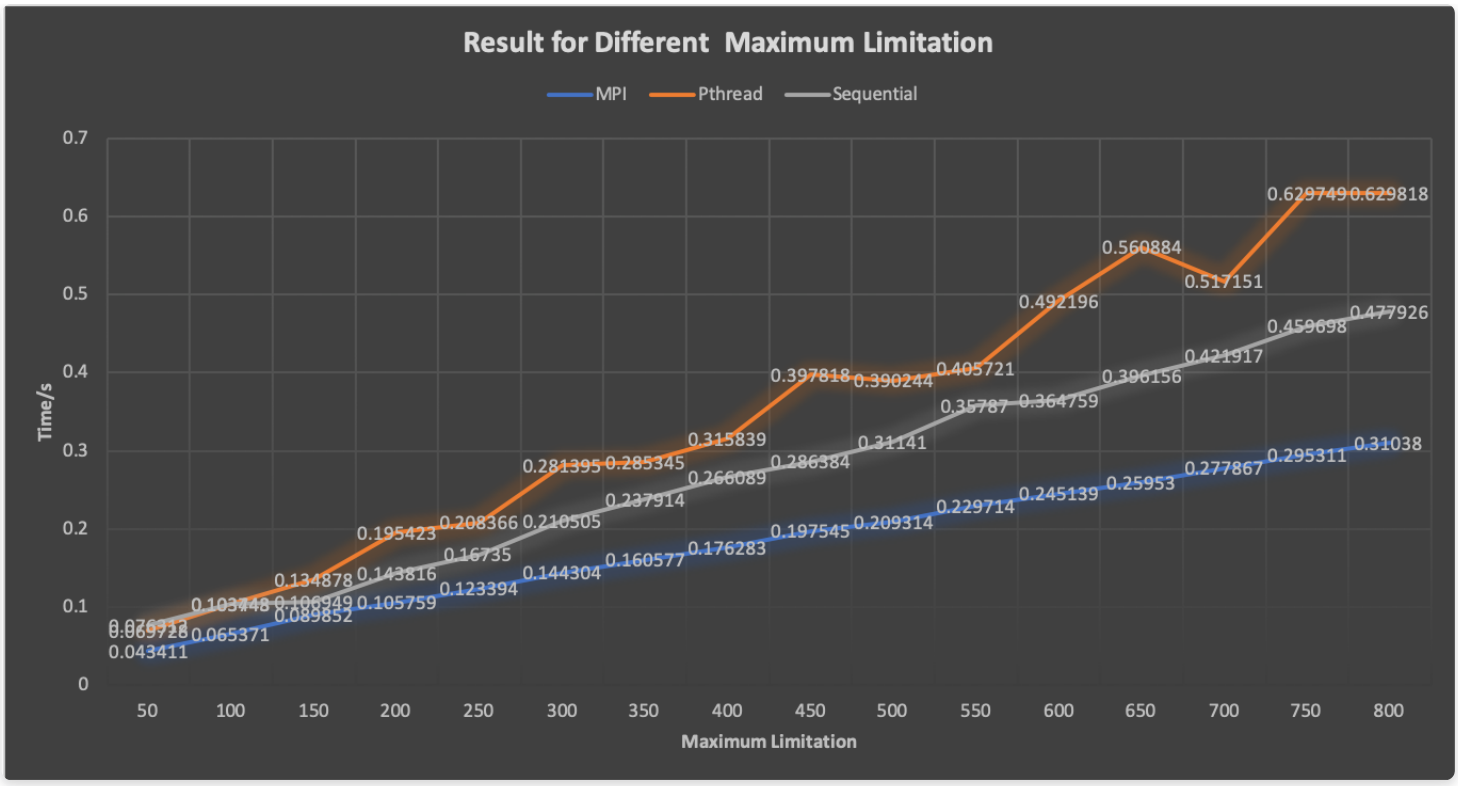

Fig.11 Result for Different Maximum Limitation

In figure 11, x axis represents the maximum limitation, the y axis represents the running time. The blue line represents the performance under the MPI architecture, Orange line represents that under Pthread architecture while the grey line represents the Sequential one.

6. Performance Analysis

By looking at the result given in the above section, we can get several simple observations like general trend, fluctuation, line intersection, outlier, and the sudden growth. In the following section, we give the observation and in the subsection of the observation we also the possible reasons behind. By detailed analyzing those phenomena, we are able to better understand the multiprocessing and the strategies on designing better preformed program.

6.1 Analysis from Result for Different Number of Threads and Processors

6.1.1 General Trend

As we can see from figure 9 given in section 5.2, the general trend for both MPI and Pthread algorithm is that with the increasing processors or the number of threads, the execution time will decrease. Although, there is fluctuation in the figure, like the sudden peak when number of processors increases to six, however, according to the discussion with other classmates, this is not

the general case.

However, we can still observe that the general fluctuation in MPI is greater than that in Pthread.

One possible explanation for the fluctuation in the figure is that, the tasks are unbalanced divided into different threads and processors.

6.1.2 Comparison between MPI and Sequential

With smaller number of processors, there is no big difference between sequential and MPI. However, with larger number of processors, the runtime for MPI is much lesser than the runtime of sequential which is reasonable since MPI enables the processes to be executed simultaneously.

6.1.3 Comparison between Pthread and Sequential

The speed up for pthread algorithm is not that obvious, however, in general the runtime under pthread method is smaller than the runtime of sequential.

6.1.4 Comparison between MPI and Pthread

When the number of threads/procesors is small, the runtime for pthread is smaller than the runtime of MPI. However, when the number of threads/procesors increases, the runtime for pthread is larger than the runtime of MPI.

One possible explanation is that, since Pthread architecture is based on the shared memory architecture while MPI based on distributed memory. With larger number of threads, the probability that one thread has to wait for another thread to extract the data from the memory also increases which is costly.

However, when the number of thread/processor is small, since the overhead for pthread to get data from shared memory is shorter than for processors to pass the message, the runtime for Pthread is smaller than the runtime for MPI.

6.2 Analysis from Result for Different Window Size

When the window size is small, the runtime for all three methods are almost the same, However, with increased window size, sequential architecture performance is worse than MPI and Pthread.

One possible explanation is that, Pthread and MPI both have overhead time like data transition and preparation which will not change with the window size. When the window size is relatively small, the computation time is relatively small. Although, MPI and Pthread are able to save some computation time by allowing them to execute simultaneously, the time saved is limited and is relatively smaller or almost the same amount with the overhead.

6.3 Analysis from Result for Different Maximum Iteration

The general trend for increasing maximum limitation is that the runtime for all three algorithms will increase. The MPI method has the lowest increase rate and the Pthread has the largest.

7. Conclusion

In this project, we demonstrate the Mandelbrot Set with sequential, MPI and Pthread approaches. We launched three different tests to see the program profermance: fixed limitation and window size with varying number of threads and processors; fixed number of threads and processors, with varying window size; and varying maximum iteration limitation.

- With imbalanced task assignment, the fluatuation is severe.
- With smaller number of processors, there is no big difference between sequential, Pthread and MPI due to the overhead. However, with larger number of processors/threads, the runtime for MPI and Pthread is much lesser than the runtime of sequential.
- When the number of threads/procesors is small, the runtime for pthread is smaller than the runtime of MPI. However, when the number of threads/procesors increases, the runtime for pthread is larger than the runtime of MPI.

8. Experience

Some other findings observed by comparing my project result with classmates' project are shown in this section.

- There are two different functions provided by MPI library which are MPI_scatter() and MPI_scatterv(). MPI_scatter() has better preformance compared to MPI_scatterv(), however, MPI_scatterv() is more flexible and is easy to implement.
- There are two ways for project termination. The first one is checking every time for each iteration or do the comparison for N-1 times where N is the array length. Do the comparison for N-1 times actually saves time since it does not require checking.
- There are two times to get the current time to calculate the execution time. One is using MPI_time and another one is using clock() function. Clock() function is faster compared to MPI_time().
- clock() function is inaccurate when timing the Pthread method, in this situation, we use timeval struct instead.
- There are multiple ways to implement the dynamic process to execute the MPI and Pthread architecture. One way besides mine approach is that we use the queue structure to assign the task to whichever is free. This method has generallly better performance than mine.

It is possible to execute the file in two ways. It is possible to directly run the file by

```
#!/bin/bash
#PBS -l nodes=1:ppn=5,mem=1g,walltime=00:05:00
#PBS -q batch
#PBS -m abe
#PBS -V
timeout 60 mpiexec -n $j /code/117010032/Mandelbrot_Set_MPI $W $H $M
```

In which, $jstandsforthenumberofprocessors$,W stands for the window width, $Hstandsforthewindowheight$,M stands for the number of maximum iteration, and $T stands for threshold. In this way, the execution time is longer compared to the running way with config file.

```
#!/bin/bash
#PBS -l nodes=1:ppn=5,mem=1g,walltime=00:05:00
#PBS -q batch
#PBS -m abe
#PBS -V
timeout 60 mpiexec -n $j -f /home/mpi_config /code/117010032/Mandelbrot_Set_MPI $W $H $M
```

- It is also possible to speed up the execution by optimizing the compile structure. With -O1, -O2 and -O3, the execution time is around 1/4 of the original one (different processor number and array size will get different result).

9. Bonus

9.1 Design for Dynamic MPI

For dynamic MPI approach, the design is very much the same with that of static MPI approach. The only difference is that, we divide the window pixels into small chunks. In each task, the processor need to finish one chunk. Whenever, the slave processor is idle, it will send its rank to the master and master will assign another task to it.

The general flow for dynamic MPI should be

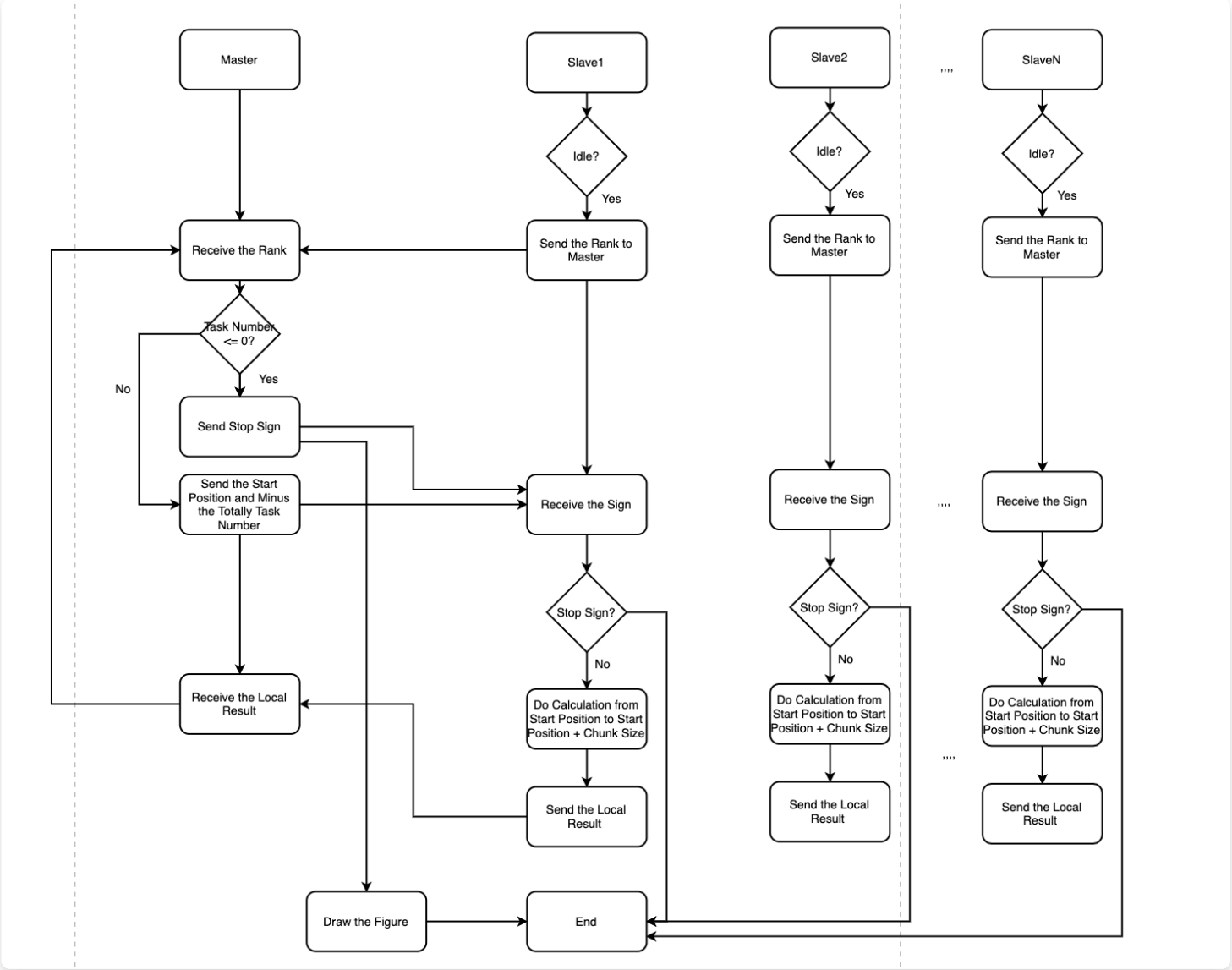

Fig.12 Flow for Dynamic MPI Approach

9.2 Design for Dynamic Pthread

For dynamic Pthread approach, the design is very much the same with that of static Pthread approach. The only difference is that, we divide the window pixels into small chunks. In each task, the thread need to finish one chunk. Whenever, the slave thread is idle, it will by itself fetch another new task.

The general flow for dynamic Pthread should be

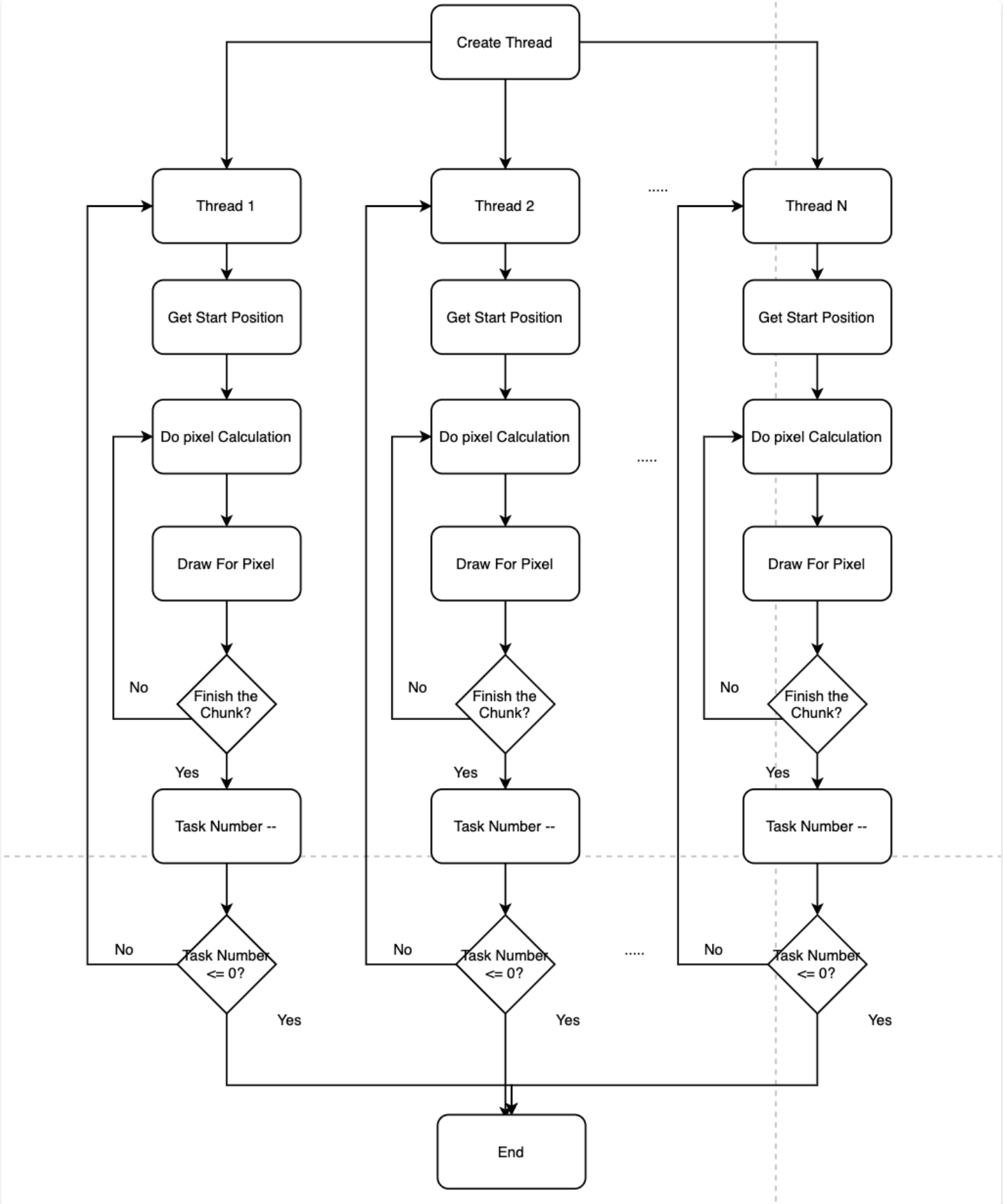

Fig.13 Flow for Dynamic Pthread Approach

9.3 Test for Dynamic MPI

Here we only analysis the dynamic MPI approach to demonstrate the performance of dynamic structure and the difference between the dynamic approach and the static ones.

There are mainly three different subsections for the test process.

9.3.1 Test for Different Chunk Size

First, we test the process with fixed window size to be 800×800, fixed number of processors to be 4 and we change the chunk size from 20 to 400.

9.3.2 Test for Different Number of Processors

Second, we test the process with fixed window size to be 800×800, fixed chunk size to be 10 and we change the number of processors used in the program from 2 to 20.

9.3.3 Test for Different Window Size

Third, we test the process with fixed number of processors to be 4, fixed chunk size to be 10 and we change the window size from 10×10 to 200×200.

9.4 Compile

In this section, we introduce the detailed instruction to compile the dynamic project.

9.4.1 Compiling

The compiling command is taken under the direction \home\stu_id. For mac user, it is required to first install XQuartz and do the following command in XQuartz terminal.

```
mpic++ Mandelbrot_Set_MPI_dy.cpp -lX11 -o /code/stu_id/Mandelbrot_Set_MPI_dy
g++ Mandelbrot_Set_Pthread_dy.cpp -lX11 -o /code/stu_id/Mandelbrot_Set_Pthread_dy -lpth
```

Then the executable file transportation_sort and transportation_sort_seq are generated under /code/stu_id direction.

9.4.2 Running

- Under /code/stu_id direction, user should type "touch script.pbs" to create a script file.

```
touch script.pbs
```

- Then type the command "vim script.pbs" to modify the script.pbs file.
- Type the command which is similar to the below line to execute the file.

```
#!/bin/bash
#PBS -l nodes=1:ppn=5,mem=1g,walltime=00:05:00
#PBS -q batch
#PBS -m abe
#PBS -V
timeout 60 mpiexec -n $j -f /home/mpi_config /code/117010032/Mandelbrot_Set_MPI_dy $W $H
```

In which, $j stands for the number of processors, $W stands for the window width,$H stands for the window height, $M stands for the number of maximum iteration, $T stands for threshold, and $C means the chunk size. With the same idea the .pbs file to execute the Pthread version is shown in below.

```
#!/bin/bash
#PBS -l nodes=1:ppn=5,mem=1g,walltime=00:05:00
#PBS -q batch
#PBS -m abe
#PBS -V
timeout 60 /code/117010032/Mandelbrot_Set_Pthread $j $W $H $M $T $C
```

9.5 Result

9.5.1 Result for Different Chunk Size

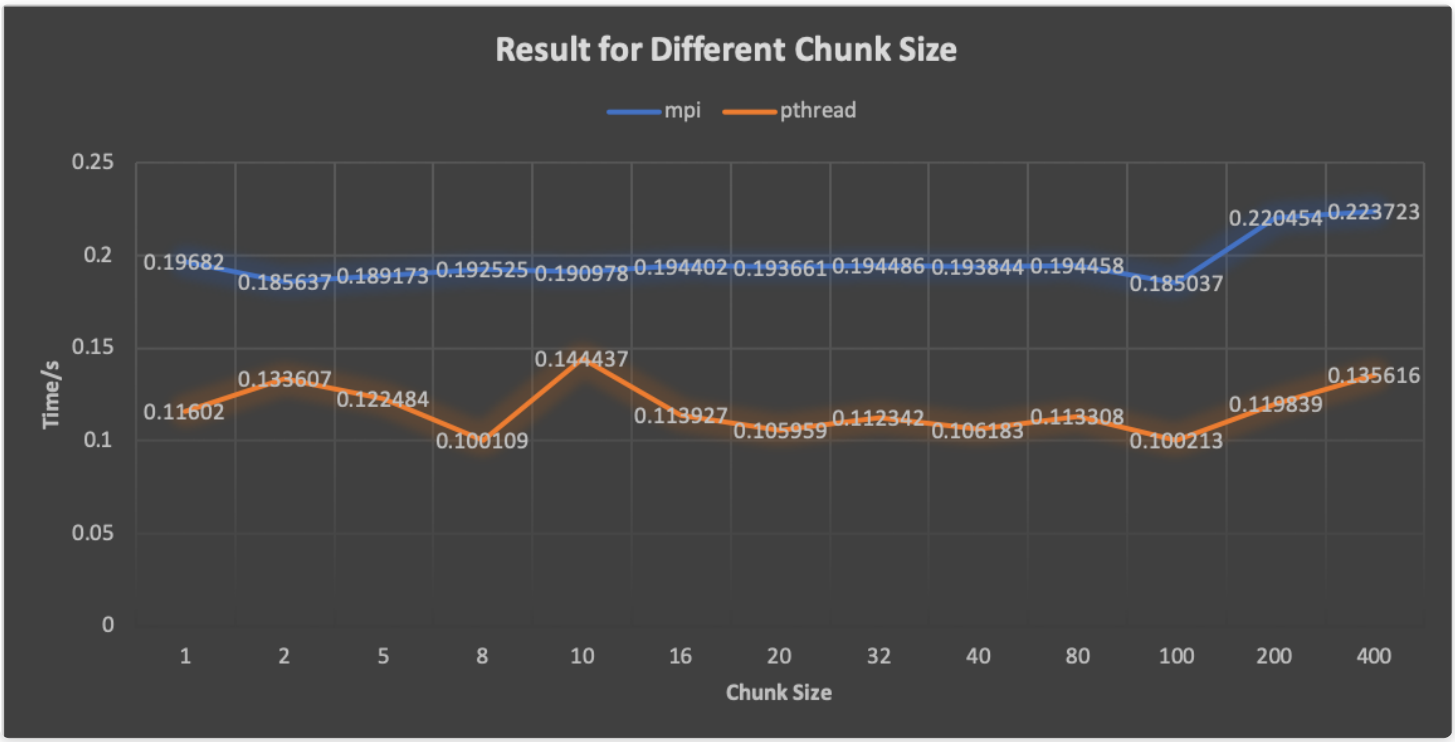

Fig.12 Result for Different Window Size

In figure 12, x axis represents the chunk size, the y axis represents the running time. The blue line represents the performance under the MPI architecture, Orange line represents that under Pthread architecture.

9.5.2 Result for Different Number of Processors

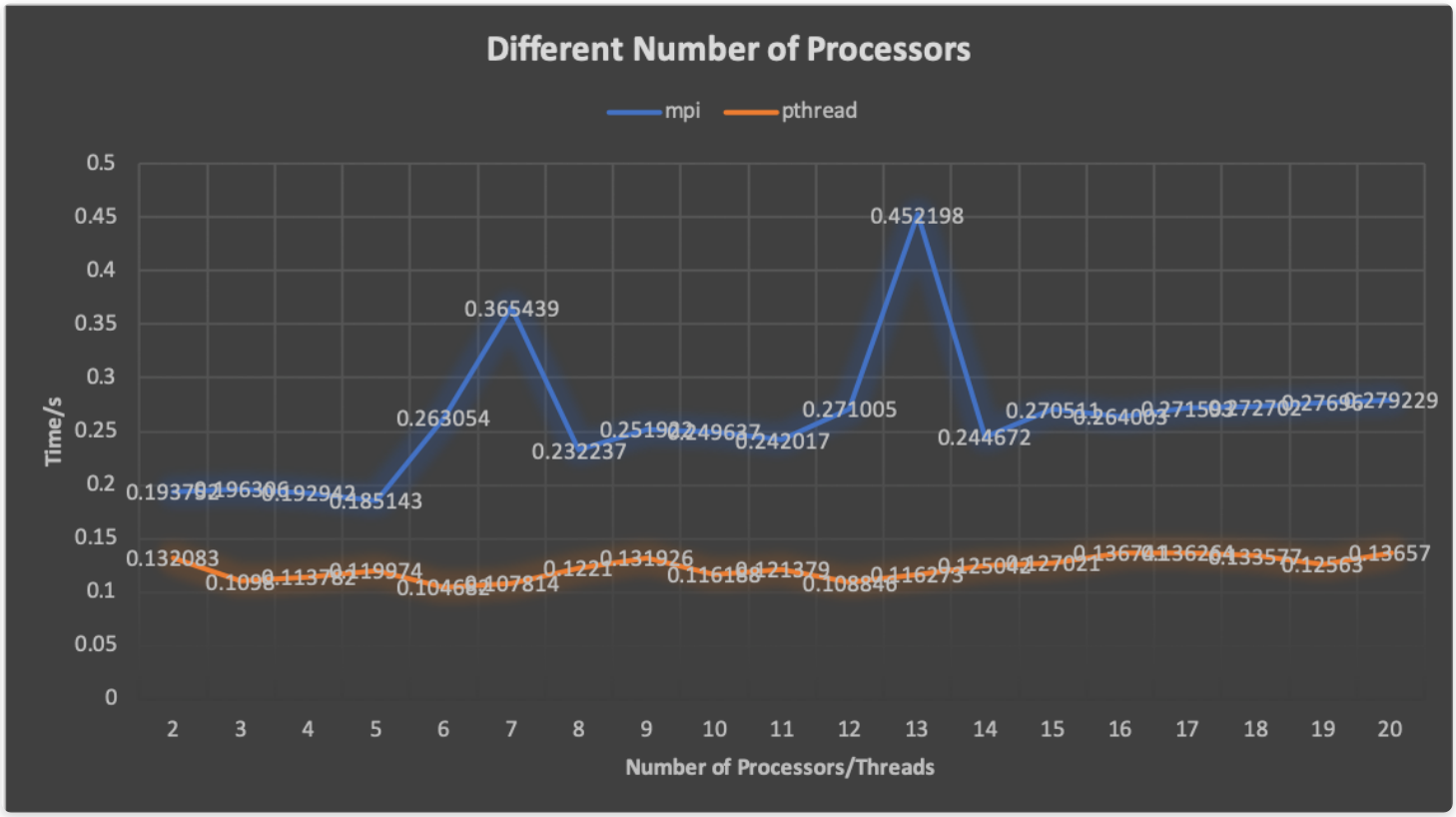

Fig.13 Result for Different Number of Processors/Threads

In figure 13, x axis represents the number of processors/threads, the y axis represents the running time. The blue line represents the performance under the MPI architecture, Orange line represents that under Pthread architecture.

9.5.3 Resutl for Different Window Size

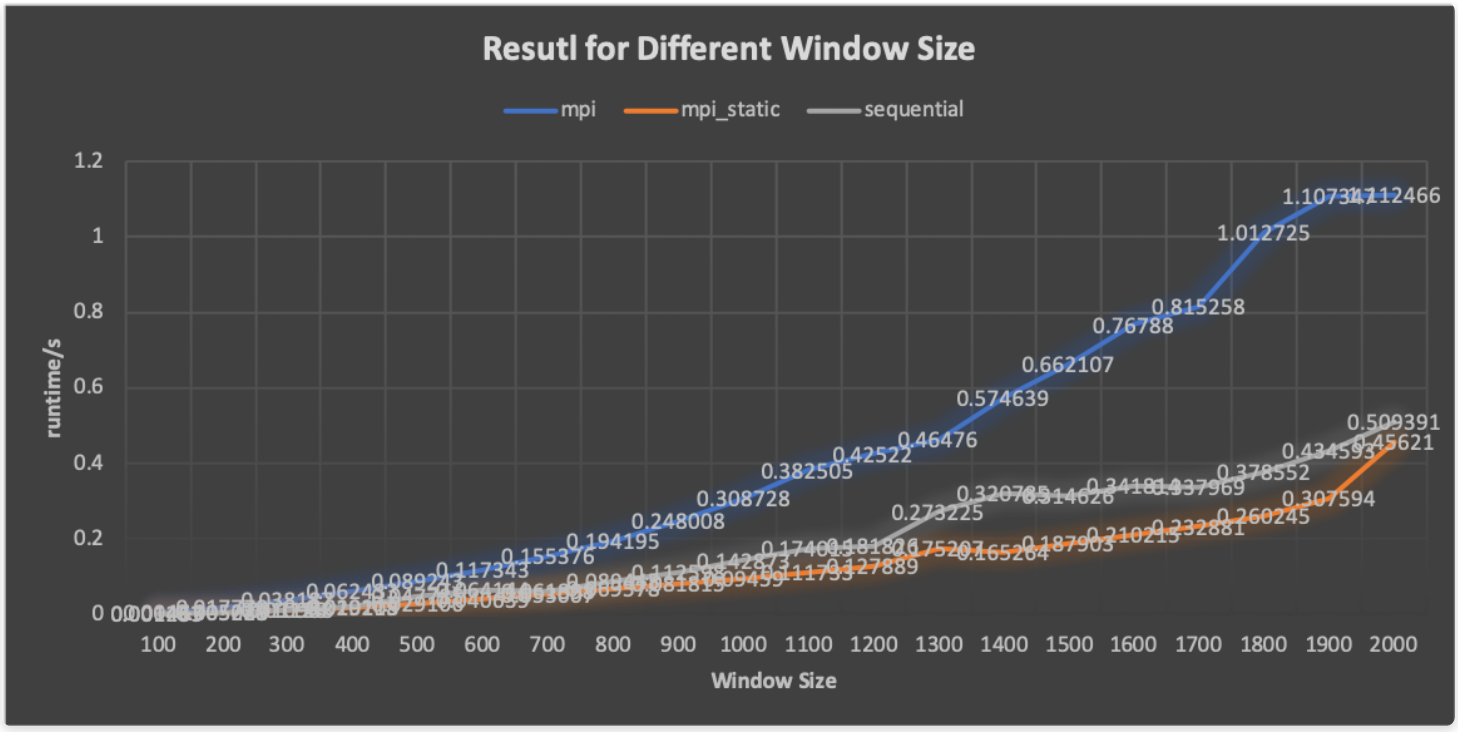

Fig.14 Result for Different Window Size

$\qquad$ In figure 14, x axis represents the number of processors/threads, the y axis represents the running time. The blue line represents the performance under the dynamic MPI architecture, Orange line represents that under static MPI architecture, and the grey line represents the sequencial architecture.

9.6 Performance Analysis

9.6.1 Analysis from Result for Different Chunk Size

As we can see from the figure 12, there is no big different for the performance between different chunk size, while generally speaking, with larger chunk size the running time will be greater.

The possible explanation for this phenomenon is that, when the chunk size is small, the overhead time is relatively large, since master process has to assign more tasks to different processors on average. When the chunk size get larger, the size in each task is larger and the task is less balanced than before, therefore, also requries more time to complete.

Generally speaking, the overhead time taken when the chunk size is small is on average shorter than the time waste caused by unbalance task assignment.

9.6.2 Analysis from Result for Different Number of Threads and Processors

As we can observe from figure 13, Pthread generally have better performance than MPI approach. Generally speaking, with larger number of processors/threads, the runtime increases. Also, we can compare the result in figure 13 with the result given by figure 9, that process with dynamic

approach generally shows less fluatuation than that of static approach.

The possible reasons for the above three phenomena are that:

First, since in my approach, in each iteration, the master and slave processor have to do at least three communications (send the rank to master, send the startposition to slave, and send the local array to master), which is extremely time consuming. Also, the test window size is 800×800, which is not large enough and therefore, the computation time is relatively shorter than the communication time. Therefore, Pthread generally have better performance than MPI approach.

Second, greater number of processors/threads means greater overhead time to assign the task to each thread and processor. Since the test window size is 800×800, which is not large enough and therefore, the computation time is relatively shorter than the overhead, therefore, with larger number of processors/threads, the runtime increases.

Third, since the approach is dynamic, each processor/thread should take generally the same time to finish their task. Since the task assignment is more balanced than the static version, the fluctuation is less severe than the static version.

9.6.3 Analysis from Result for Different Window Size

The result shown in figure 14 is not as we expected. As we can observe, that under the dynamic MPI architecture, the process requries the greatest amount of runtime, while in static architecture requires the least.

However, according to the discussion with classmates, if we implement the dynamic architecture in another way, the runtime for dynamic MPI will be much lesser than the static ones (use the queue structure to assign the task to whichever is free. This method has generallly better performance than mine.s).

The reason why my performance is not as good as the architecture using queue structure is that, in my approach, in each iteration, the master and the slave process need to communicate at least for three times. The overhead is too large that the time saved by dynamic assignment cannot cover the time waste in communication. However, as we can also observe from the figure 14, when the window size is large enough, the increase speed for dynamic approach is slower than the static approach. Therefore, it is reasonable for us to expect that when the window size is large enough, finally the time saved in opitmizing task assignment will outstrip the overhead.

10. Appendix

10.1 Code for Static Sequential

```
/* Sequential Mandelbrot program */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <stdio.h>
#include <string.h>
#include <sys/time.h>
#include <math.h>

typedef struct complextype
        {
        float real, imag;
        } Compl;


int main (int argc, char* argv[])
{
        Window          win;                            /* initialization for a window */
        unsigned
        int             width, height,                  /* window size */
                        x, y,                           /* window position */
                        border_width,                   /*border width in pixels */
                        display_width, display_height,  /* size of screen */
                        screen;                         /* which screen */

        char            *window_name = "Mandelbrot Set", *display_name = NULL;
        GC              gc;
        unsigned
        long            valuemask = 0;
        XGCValues       values;
        Display         *display;
        XSizeHints      size_hints;
        Pixmap          bitmap;
        XPoint          points[800];
        FILE            *fp, *fopen ();
        char            str[100];

        XSetWindowAttributes attr[1];

       /* Mandlebrot variables */
        int i, j, k;
        Compl   z, c;
        float   lengthsq, temp;
        float threshold;
        int MAX_Iteration;

        /* connect to Xserver */

        if (  (display = XOpenDisplay (display_name)) == NULL ) {
           fprintf (stderr, "drawon: cannot connect to X server %s\n",
                                XDisplayName (display_name) );
```

```
  exit (-1);
  }

  /* get screen size */

  screen = DefaultScreen (display);
  display_width = DisplayWidth (display, screen);
  display_height = DisplayHeight (display, screen);

  /* set window size */

width = atoi(argv[1]);
height = atoi(argv[2]);
threshold = atoi(argv[4]);
MAX_Iteration = atoi(argv[3]);


  /* set window position */

  x = 0;
  y = 0;

  /* create opaque window */

  border_width = 4;
  win = XCreateSimpleWindow (display, RootWindow (display, screen),
                         x, y, width, height, border_width,
                         BlackPixel (display, screen), WhitePixel (display, scree

  size_hints.flags = USPosition|USSize;
  size_hints.x = x;
  size_hints.y = y;
  size_hints.width = width;
  size_hints.height = height;
  size_hints.min_width = 300;
  size_hints.min_height = 300;

  XSetNormalHints (display, win, &size_hints);
  XStoreName(display, win, window_name);

  /* create graphics context */

  gc = XCreateGC (display, win, valuemask, &values);

  XSetBackground (display, gc, WhitePixel (display, screen));
  XSetForeground (display, gc, BlackPixel (display, screen));
  XSetLineAttributes (display, gc, 1, LineSolid, CapRound, JoinRound);

  attr[0].backing_store = Always;
  attr[0].backing_planes = 1;
  attr[0].backing_pixel = BlackPixel(display, screen);
```

```
        XChangeWindowAttributes(display, win, CWBackingStore | CWBackingPlanes | CWBack

        XMapWindow (display, win);
        XSync(display, 0);

        struct timeval startTime;
        struct timeval endTime;

        /* Calculate and draw points */
        printf("Name: Yaqian Chen\nStudent ID: 117010032\nAssignment 2, Mandelbrot Set,
        gettimeofday(&startTime, NULL);
        for(i=0; i < width; i++)
        for(j=0; j < height; j++) {

          z.real = z.imag = 0.0;
          c.real = ((float) j - height/2)/(height/4);               /* scale factors fo
          c.imag = ((float) i - width/2)/(width/4);
          k = 0;

          do  {                                             /* iterate for pixel color

            temp = z.real*z.real - z.imag*z.imag + c.real;
            z.imag = 2.0*z.real*z.imag + c.imag;
            z.real = temp;
            lengthsq = z.real*z.real+z.imag*z.imag;
            k++;
          } while (lengthsq < threshold && k < MAX_Iteration);

        if (k == MAX_Iteration) XDrawPoint (display, win, gc, j, i);

        }
    gettimeofday(&endTime, NULL);
    XFlush (display);
    printf("Execution Time is: %f seconds.\n", (double)((endTime.tv_sec - startTime.tv_s
    sleep (30);
    return 0;
        /* Program Finished */

}
```

10.2 Code for Static Pthread

```
#include <pthread.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/time.h>
#include <time.h>
#include <iostream>

using namespace std;
Window          win;                            /* initialization for a window */
unsigned
int             width, height,                  /* window size */
                x, y,                           /* window position */
                border_width,                   /*border width in pixels */
                display_width, display_height,  /* size of screen */
                screen,                         /* which screen */
                NUM_thread,
                MAX_ITERATION;
float           lengthsq;
float           threshold;
char            *window_name = "Mandelbrot Set", *display_name = NULL;
GC              gc;
unsigned
long            valuemask = 0;
XGCValues       values;
Display         *display;
XSizeHints      size_hints;
Pixmap          bitmap;
XPoint          points[800];
FILE            *fp, *fopen ();
char            str[100];
pthread_mutex_t mutex_draw;
struct timeval startTime;
struct timeval endTime;

void* Mandelbrot_calc(void *t);

typedef struct complextype
        {
        float real, imag;
        } Compl;

typedef struct _thread_data{
    int thread_id;
    int *data;
}thread_data;
```

```
int main(int argc, char* argv[]){


    XSetWindowAttributes attr[1];

    /* connect to Xserver */

    if (  (display = XOpenDisplay (display_name)) == NULL ) {
        fprintf (stderr, "drawon: cannot connect to X server %s\n",
                            XDisplayName (display_name) );
    exit (-1);
    }

    /* get screen size */

    screen = DefaultScreen (display);
    display_width = DisplayWidth (display, screen);
    display_height = DisplayHeight (display, screen);

    /* set window position */

    x = 0;
    y = 0;

    /* create opaque window */
    NUM_thread = atoi(argv[1]);
        width = atoi(argv[2]);
        height = atoi(argv[3]);
    MAX_ITERATION = atoi(argv[4]);
    threshold = atoi(argv[5]);

    border_width = 4;
    win = XCreateSimpleWindow (display, RootWindow (display, screen),
                            x, y, width, height, border_width,
                            BlackPixel (display, screen), WhitePixel (display, screen));

    size_hints.flags = USPosition|USSize;
    size_hints.x = x;
    size_hints.y = y;
    size_hints.width = width;
    size_hints.height = height;
    size_hints.min_width = 300;
    size_hints.min_height = 300;

    XSetNormalHints (display, win, &size_hints);
    XStoreName(display, win, window_name);

    /* create graphics context */

    gc = XCreateGC (display, win, valuemask, &values);
```

```
    XSetBackground (display, gc, WhitePixel (display, screen));
    XSetForeground (display, gc, BlackPixel (display, screen));
    XSetLineAttributes (display, gc, 1, LineSolid, CapRound, JoinRound);

    attr[0].backing_store = Always;
    attr[0].backing_planes = 1;
    attr[0].backing_pixel = BlackPixel(display, screen);

    XChangeWindowAttributes(display, win, CWBackingStore | CWBackingPlanes | CWBackingPi

    XMapWindow (display, win);
    XSync(display, 0);

    /* Calculate and draw points */

    int* wholeArray = new int[height * width];

    printf("Name: Yaqian Chen\nStudent ID: 117010032\nAssignment 2, Mandelbrot Set, Pthr
    gettimeofday(&startTime, NULL);
        pthread_mutex_init(&mutex_draw, NULL);
    pthread_t thread[NUM_thread];
    thread_data input_data[NUM_thread];
    for (int i=0;i<NUM_thread;i++){
        input_data[i].data = wholeArray;
        input_data[i].thread_id = i;
        int rc = pthread_create(&thread[i],NULL,Mandelbrot_calc,&input_data[i]);
        if (rc){
            fprintf(stderr,"error:pthread_create,rc: %d\n",rc);
            return EXIT_FAILURE;
        }
    }

    for (int i=0;i<NUM_thread;++i){
        pthread_join(thread[i],NULL);
    }
    gettimeofday(&endTime, NULL);
    XFlush (display);
    printf("Execution Time is: %f seconds.\n", (double)((endTime.tv_sec - startTime.tv_s
    sleep(30);
}

void* Mandelbrot_calc(void* arg){
    int i, j, k;
    Compl   z, c;
    float   temp;
    thread_data* input_data = (thread_data *)arg;
    int thread_id = input_data->thread_id;
    int* wholeArray = input_data->data;
    int sepNum = width/NUM_thread * height;
    int recvCount = (thread_id == NUM_thread - 1) ? width * height - sepNum * (NUM_threa
    for(int i= thread_id * sepNum/height; i < thread_id * sepNum/height + recvCount/heig
```

```
        for(j=0; j < height; j++) {

        z.real = z.imag = 0.0;
        c.real = ((float) j - (height/2))/(height/4);               /* scale factors fo
        c.imag = ((float) i - (width/2))/(width/4);
        k = 0;

        do  {                                             /* iterate for pixel color */

            temp = z.real*z.real - z.imag*z.imag + c.real;
            z.imag = 2.0*z.real*z.imag + c.imag;
            z.real = temp;
            lengthsq = z.real*z.real+z.imag*z.imag;
            k++;

        } while (lengthsq < threshold && k < MAX_ITERATION);
        pthread_mutex_lock(&mutex_draw);
        if (k == 100) XDrawPoint (display, win, gc, j, i);
        pthread_mutex_unlock(&mutex_draw);
        }
    pthread_exit(NULL);
}
```

10.3 Code for Static MPI

```
#include "mpi.h"
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
#include <iostream>

using namespace std;

typedef struct complextype
        {
        float real, imag;
        } Compl;

int main(int argc, char* argv[]){
    int rank, size;
    int *reps, *result;
    Window win;
    GC gc;
    Display *display;
    /* Mandlebrot variables */
    int i, j, k;
    Compl   z, c;
    float   lengthsq, temp;
    float   threshold;
    int MAX_Iteration;

    unsigned int    width, height,                  /* window size */
                    x, y,                           /* window position */
                    border_width,                   /*border width in pixels */
                    display_width, display_height,  /* size of screen */
                    screen;                         /* which screen */

    char            *window_name = "Mandelbrot Set", *display_name = NULL;
    unsigned
    long            valuemask = 0;
    XGCValues       values;
    XSizeHints      size_hints;
    Pixmap          bitmap;
    XPoint          points[800];
    FILE            *fp, *fopen ();
    char            str[100];
    struct timeval startTime;
    struct timeval endTime;
    MPI_Init(&argc, &argv);
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
MPI_Comm_size(MPI_COMM_WORLD, &size);
XSetWindowAttributes attr[1];
/* connect to Xserver */

if (  (display = XOpenDisplay (display_name)) == NULL ) {
    fprintf (stderr, "drawon: cannot connect to X server %s\n",
                        XDisplayName (display_name) );
exit (-1);
}
/* get screen size */

screen = DefaultScreen (display);
display_width = DisplayWidth (display, screen);
display_height = DisplayHeight (display, screen);

/* set window size */
width = atoi(argv[1]);
height = atoi(argv[2]);
threshold = atoi(argv[4]);
MAX_Iteration = atoi(argv[3]);

/* set window position */

x = 0;
y = 0;

/* create opaque window */

border_width = 4;
win = XCreateSimpleWindow (display, RootWindow (display, screen),
                        x, y, width, height, border_width,
                        BlackPixel (display, screen), WhitePixel (display, screen));

size_hints.flags = USPosition|USSize;
size_hints.x = x;
size_hints.y = y;
size_hints.width = width;
size_hints.height = height;
size_hints.min_width = 300;
size_hints.min_height = 300;

XSetNormalHints (display, win, &size_hints);
XStoreName(display, win, window_name);

/* create graphics context */

gc = XCreateGC (display, win, valuemask, &values);

if(rank==0){
    XSetBackground (display, gc, WhitePixel (display, screen));
    XSetForeground (display, gc, BlackPixel (display, screen));
```

```
        XSetLineAttributes (display, gc, 1, LineSolid, CapRound, JoinRound);

        attr[0].backing_store = Always;
        attr[0].backing_planes = 1;
        attr[0].backing_pixel = BlackPixel(display, screen);

        XChangeWindowAttributes(display, win, CWBackingStore | CWBackingPlanes | CWBack

        XMapWindow (display, win);
        XSync(display, 0);
        printf("Name: Yaqian Chen\nStudent ID: 117010032\nAssignment 2, Mandelbrot Set,
        gettimeofday(&startTime, NULL);
    }
    int count = 0;
    int sepNum = width/size * height;
    int* wholeNumList = new int[size];
    int* disNumList = new int[size];
    for (int i = 0; i < size; i++){
        wholeNumList[i] = sepNum;
        if (i == size - 1){
            wholeNumList[i] = width * height - sepNum * (size - 1);
        }
        disNumList[i] = i * sepNum;
    }
    int recvCount = (rank == size - 1) ? width * height - sepNum * (size- 1) : sepNum;
    int* wholeArray = new int[height * width];
    int* sepArray = new int[height * recvCount];
    for(int i= rank * sepNum/height; i < rank * sepNum/height + recvCount/height ; i++)
    for(int j=0; j < height; j++) {
        z.real = z.imag = 0.0;
        c.real = ((float) j - height/2)/(height/4);               /* scale factors for
        c.imag = ((float) i - width/2)/(width/4);
        k = 0;
        do  {         /* iterate for pixel color */
        temp = z.real*z.real - z.imag*z.imag + c.real;
        z.imag = 2.0*z.real*z.imag + c.imag;
        z.real = temp;
        lengthsq = z.real*z.real+z.imag*z.imag;
        k++;
        } while (lengthsq < threshold && k < MAX_Iteration);
        sepArray[count++] = k;
    }
}
    MPI_Gatherv(sepArray, recvCount, MPI_INT, wholeArray, wholeNumList, disNumList, MPI_
    if (rank==0){
        gettimeofday(&endTime, NULL);
        int count = 0;
        for (int i = 0; i < width; i++){
            for (int j = 0; j < height; j++){
                    if (wholeArray[count++] == MAX_Iteration) XDrawPoint (display, win,
            }
```

```
        }
        XFlush (display);
        printf("Execution Time is: %f seconds.\n", (double)((endTime.tv_sec - startTime.
        sleep (30);
    }
    MPI_Finalize();
    return 0;
}
```

10.4 Code for Dynamic Pthread

```
#include <pthread.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/time.h>
#include <time.h>
#include <iostream>

using namespace std;
Window          win;                            /* initialization for a window */
unsigned
int             width, height,                  /* window size */
                x, y,                           /* window position */
                border_width,                   /*border width in pixels */
                display_width, display_height,  /* size of screen */
                screen,                         /* which screen */
                NUM_thread,
                MAX_ITERATION;
float           lengthsq;
float           threshold;
char            *window_name = "Mandelbrot Set", *display_name = NULL;
GC              gc;
unsigned
long            valuemask = 0;
XGCValues       values;
Display         *display;
XSizeHints      size_hints;
Pixmap          bitmap;
XPoint          points[800];
FILE            *fp, *fopen ();
char            str[100];
pthread_mutex_t mutex_draw, mutex_task;
struct timeval startTime;
struct timeval endTime;
int chunk;
int taskNum;

void* Mandelbrot_calc(void *t);

typedef struct complextype
        {
        float real, imag;
        } Compl;

typedef struct _thread_data{
    int thread_id;
    int *data;
```

```
}thread_data;

int main(int argc, char* argv[]){


    XSetWindowAttributes attr[1];

    /* connect to Xserver */

    if (  (display = XOpenDisplay (display_name)) == NULL ) {
        fprintf (stderr, "drawon: cannot connect to X server %s\n",
                            XDisplayName (display_name) );
    exit (-1);
    }

    /* get screen size */

    screen = DefaultScreen (display);
    display_width = DisplayWidth (display, screen);
    display_height = DisplayHeight (display, screen);

    /* set window position */

    x = 0;
    y = 0;

    /* create opaque window */
    NUM_thread = atoi(argv[1]);
        width = atoi(argv[2]);
        height = atoi(argv[3]);
    MAX_ITERATION = atoi(argv[4]);
    threshold = atoi(argv[5]);
    chunk = atoi(argv[6]);
    taskNum = width/chunk;

    border_width = 4;
    win = XCreateSimpleWindow (display, RootWindow (display, screen),
                            x, y, width, height, border_width,
                            BlackPixel (display, screen), WhitePixel (display, screen));

    size_hints.flags = USPosition|USSize;
    size_hints.x = x;
    size_hints.y = y;
    size_hints.width = width;
    size_hints.height = height;
    size_hints.min_width = 300;
    size_hints.min_height = 300;

    XSetNormalHints (display, win, &size_hints);
    XStoreName(display, win, window_name);
```

```
    /* create graphics context */

    gc = XCreateGC (display, win, valuemask, &values);

    XSetBackground (display, gc, WhitePixel (display, screen));
    XSetForeground (display, gc, BlackPixel (display, screen));
    XSetLineAttributes (display, gc, 1, LineSolid, CapRound, JoinRound);

    attr[0].backing_store = Always;
    attr[0].backing_planes = 1;
    attr[0].backing_pixel = BlackPixel(display, screen);

    XChangeWindowAttributes(display, win, CWBackingStore | CWBackingPlanes | CWBackingP

    XMapWindow (display, win);
    XSync(display, 0);

    /* Calculate and draw points */

    int* wholeArray = new int[height * width];

    printf("Name: Yaqian Chen\nStudent ID: 117010032\nAssignment 2, Mandelbrot Set, Pth
    gettimeofday(&startTime, NULL);
        pthread_mutex_init(&mutex_draw, NULL);
    pthread_mutex_init(&mutex_task, NULL);
    pthread_t thread[NUM_thread];
    thread_data input_data[NUM_thread];
    for (int i=0;i<NUM_thread;i++){
        input_data[i].data = wholeArray;
        input_data[i].thread_id = i;
        taskNum -= 1;
        int rc = pthread_create(&thread[i],NULL,Mandelbrot_calc,&input_data[i]);
        if (rc){
            fprintf(stderr,"error:pthread_create,rc: %d\n",rc);
            return EXIT_FAILURE;
        }
    }

    for (int i=0;i<NUM_thread;++i){
        pthread_join(thread[i],NULL);
    }
    gettimeofday(&endTime, NULL);
    XFlush (display);
    printf("Execution Time is: %f seconds.\n", (double)((endTime.tv_sec - startTime.tv_s
    sleep(30);
}

void* Mandelbrot_calc(void* arg){
    int i, j, k;
    Compl   z, c;
    float   temp;
```

```
    thread_data* input_data = (thread_data *)arg;
    int thread_id = input_data->thread_id;
    int* wholeArray = input_data->data;
    int startPosition = thread_id * chunk;
    while(1){
        for(int i= startPosition; i < startPosition + chunk ; i++){
                for(j=0; j < height; j++) {
                z.real = z.imag = 0.0;
                c.real = ((float) j - (height/2))/(height/4);               /* scale fa
                c.imag = ((float) i - (width/2))/(width/4);
                k = 0;
                do  {                                                  /* iterate for pixel

                    temp = z.real*z.real - z.imag*z.imag + c.real;
                    z.imag = 2.0*z.real*z.imag + c.imag;
                    z.real = temp;
                    lengthsq = z.real*z.real+z.imag*z.imag;
                    k++;
                } while (lengthsq < threshold && k < MAX_ITERATION);
                pthread_mutex_lock(&mutex_draw);
                if (k == MAX_ITERATION) {
                    XDrawPoint (display, win, gc, j, i);
                    }
                pthread_mutex_unlock(&mutex_draw);
                }
        }
    if (taskNum > 0){
        pthread_mutex_lock(&mutex_task);
        startPosition = (width/chunk - taskNum) * chunk;
        taskNum -= 1;
        pthread_mutex_unlock(&mutex_task);
    }
    else pthread_exit(NULL);
    }
}
```

10.5 Code for Dynamic MPI

```
#include "mpi.h"
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
#include <iostream>
#include <queue>

using namespace std;

typedef struct complextype
        {
        float real, imag;
        } Compl;

int main(int argc, char* argv[]){
    int rank, size;
    int *reps, *result;
    Window win;
    GC gc;
    Display *display;
    /* Mandlebrot variables */
    int i, j, k;
    Compl   z, c;
    float   lengthsq, temp;
    float   threshold;
    int MAX_Iteration;

    unsigned int    width, height,                  /* window size */
                    x, y,                           /* window position */
                    border_width,                   /*border width in pixels */
                    display_width, display_height,  /* size of screen */
                    screen;                         /* which screen */

    char            *window_name = "Mandelbrot Set", *display_name = NULL;
    unsigned
    long            valuemask = 0;
    XGCValues       values;
    XSizeHints      size_hints;
    Pixmap          bitmap;
    XPoint          points[800];
    FILE            *fp, *fopen ();
    char            str[100];
    struct timeval startTime;
    struct timeval endTime;
    MPI_Init(&argc, &argv);
    MPI_Comm comm = MPI_COMM_WORLD;
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
XSetWindowAttributes attr[1];
/* connect to Xserver */

if (  (display = XOpenDisplay (display_name)) == NULL ) {
    fprintf (stderr, "drawon: cannot connect to X server %s\n",
                        XDisplayName (display_name) );
exit (-1);
}
/* get screen size */

screen = DefaultScreen (display);
display_width = DisplayWidth (display, screen);
display_height = DisplayHeight (display, screen);

/* set window size */
width = atoi(argv[1]);
height = atoi(argv[2]);
threshold = atoi(argv[4]);
MAX_Iteration = atoi(argv[3]);

int chunk  = atoi(argv[5]);
int taskNum = width/chunk;
int startPosition = 0;
/* set window position */

x = 0;
y = 0;

/* create opaque window */

border_width = 4;
win = XCreateSimpleWindow (display, RootWindow (display, screen),
                        x, y, width, height, border_width,
                        BlackPixel (display, screen), WhitePixel (display, screen));

size_hints.flags = USPosition|USSize;
size_hints.x = x;
size_hints.y = y;
size_hints.width = width;
size_hints.height = height;
size_hints.min_width = 300;
size_hints.min_height = 300;

XSetNormalHints (display, win, &size_hints);
XStoreName(display, win, window_name);

/* create graphics context */

gc = XCreateGC (display, win, valuemask, &values);
```

```
if(rank==0){
    XSetBackground (display, gc, WhitePixel (display, screen));
    XSetForeground (display, gc, BlackPixel (display, screen));
    XSetLineAttributes (display, gc, 1, LineSolid, CapRound, JoinRound);

    attr[0].backing_store = Always;
    attr[0].backing_planes = 1;
    attr[0].backing_pixel = BlackPixel(display, screen);

    XChangeWindowAttributes(display, win, CWBackingStore | CWBackingPlanes | CWBack

    XMapWindow (display, win);
    XSync(display, 0);
    printf("Name: Yaqian Chen\nStudent ID: 117010032\nAssignment 2, Mandelbrot Set,
    int startPosition = 0;
    gettimeofday(&startTime, NULL);
}
int* wholeArray = new int[height * width];
while (1){
    if (rank != 0){
        int count = 1;
        int sepNum = chunk * height + 1;
        int* sepArray = new int[sepNum];
        MPI_Send(&rank, 1, MPI_INT, 0, 0, comm);
        MPI_Recv(&startPosition, 1, MPI_INT, 0, 1, comm, MPI_STATUS_IGNORE);
        if (startPosition < 0) break;
        sepArray[0] = startPosition;
        for(int i= startPosition; i < startPosition + chunk; i++){
            for(int j=0; j < height; j++) {
                z.real = z.imag = 0.0;
                c.real = ((float) j - height/2)/(height/4);               /* scale
                c.imag = ((float) i - width/2)/(width/4);
                k = 0;
                do  {         /* iterate for pixel color */
                temp = z.real*z.real - z.imag*z.imag + c.real;
                z.imag = 2.0*z.real*z.imag + c.imag;
                z.real = temp;
                lengthsq = z.real*z.real+z.imag*z.imag;
                k++;
                } while (lengthsq < threshold && k < MAX_Iteration);
                sepArray[count++] = k;
            }
        }
        MPI_Send(sepArray, sepNum, MPI_INT, 0, 2, comm);
    }
    if (rank == 0){
        int rank_recv;
        int sepNum = chunk * height + 1;
        int stopSign = -1;
        int* sepArray = new int[sepNum];
```

```
            MPI_Recv(&rank_recv, 1, MPI_INT, MPI_ANY_SOURCE, 0, comm, MPI_STATUS_IGNORE
            if (startPosition == -1){
                for (int i = 1; i < size; i++){
                    MPI_Send(&startPosition, 1, MPI_INT, i, 1, comm);
                }
                break;
            }
            MPI_Send(&startPosition, 1, MPI_INT, rank_recv, 1, comm);
            MPI_Recv(sepArray, sepNum, MPI_INT, rank_recv, 2, comm, MPI_STATUS_IGNORE);
            int start = sepArray[0]*height;
            for (int i = 0; i < sepNum - 1; i++){
                wholeArray[start + i] = sepArray[1 + i];
            }
            startPosition += chunk;
            taskNum -= 1;
            if (taskNum < 0) startPosition = -1;
        }
    }
    if (rank==0){
        gettimeofday(&endTime, NULL);
        int count = 0;
        for (int i = 0; i < width; i++){
            for (int j = 0; j < height; j++){
                    if (wholeArray[count++] == MAX_Iteration) XDrawPoint (display, win,
            }
        }
        XFlush (display);
        printf("Execution Time is: %f seconds.\n", (double)((endTime.tv_sec - startTime
        sleep (30);
    }
    MPI_Finalize();
    return 0;
}
```