

CSC4005 FA22 HW02

Haoran Sun (haoransun@link.cuhk.edu.cn)

1 Introduction

The Mandelbrot set is the set of complex numbers $c \in \mathbb{C}$ that does not diverge under the following iteration, start from $z_0 = c$.

$$z_{n+1}^2 = z^2 + c \quad (1)$$

The set can be visualized through numerical computation. By setting a rectangular region (should be aligned to the real and imaginary axis) in the complex plane, one can divide it into a mesh of points. Setting a maximum iteration number m_{\max} and then perform iteration on each point using the equation 1 until it diverges. Let $m(c)$ be the number of iteration steps that the point diverges, then $m(c)/m_{\max}$ could be an estimator of the density of Mandelbrot set around this point. Then, the density of Mandelbrot set could be visualized according to $m(c)$ and m_{\max} .

However, the computation is intensive. Suppose we are going to obtain an image with resolution $m \times n$, the computational complexity would be $O(nm)$. In order to accelerate the computation, in this assignment, two parallel scheme are implemented: MPI and Pthread. Programs are tested under different resolutions and numbers of CPU cores. Speed-up factor and CPU efficiency are also analyzed.

2 Method

2.1 Program design and implementation

All programs are implemented using C++ programming language. MPICH and Pthreads are used for parallelism. For visualization, OpenGL is used to implement a graphical render while STB (single-file public domain libraries) is used to plot png images.

Please refer to Figure A.1 for the MPI program flowchart and Figure A.2 for the Pthreads program flowchart. The sequential version were written in `src/main.seq.cpp`, MPI version `src/main mpi .cpp`, Pthreads `src/main.pthread.cpp`.

2.2 Usage

For convenience, one can directly execute `demo.seq.sh`, `demo.mpi.sh`, and `demo.pthread.sh` under `hw02/` to have a first glimpse of the program.

```
cd hw02
./scripts/demo.seq.sh
./scripts/demo.mpi.sh
./scripts/demo.pthread.sh
./scripts/demo.pthread_ds.sh # ds means dynamic scheduling
```

The program is compiled using CMake build system. One can have a look at `CMakeLists.txt` and `src/CMakeLists.txt` to check compilation requirements. If one wants to build the program with

the GUI feature, he can run the following commands to configure and start compilation under hw02 directory. To disable the GUI feature, one can set `-DGUI=OFF` in the configure process. The compiled programs would be placed in build/bin directory.

```
cmake -B build -DCMAKE_BUILD_TYPE=Release -DGUI=ON
cmake --build build
```

One can run the program using the following commands, where `xmax`, `xmin`, `ymax`, `ymin` set the range of the rectangular region in the complex plane; `ndim` sets the resolution (partition of the mesh) on x -direction (real line); `record` determines whether the runtime data would be saved; `save` controls if the image would be saved. After executing the program, a GUI window should be prompted to display the density and an image `mandelbrot_$jobtype.png` similar to Figure 1 would be saved.

```
paras="--ndim 2000 --xmin -0.125 --xmax 0.125 --ymin 0.6 --ymax 0.7"
record="--record 0"
save="--save 1"
./build/bin/main.seq $paras $save $record           # sequential program
mpirun -np 4 ./build/bin/main.mpi $paras $save $record # mpi program
./build/bin/main.pthread -nt 4 $paras $save $record   # pthreads program
```



Figure 1: Sample output image

2.3 Performance evaluation

In order to evaluate the parallel code, the program was executed under different configurations. With 20 different CPU core numbers (from 1 to 20 with increment 1, $p = 1, 2, \dots, 20$) and 20 different x -resolutions (from 500 to 10000 with increment 500, $n = 500, 1000, \dots, 10000$), 400 cases in total were sampled. Recorded runtime and CPU time were analyzed through the Numpy package in Python. Figures were plotted through the Matplotlib and the Seaborn packages in Python. Analysis code were written in `analysis/main.ipynb`.

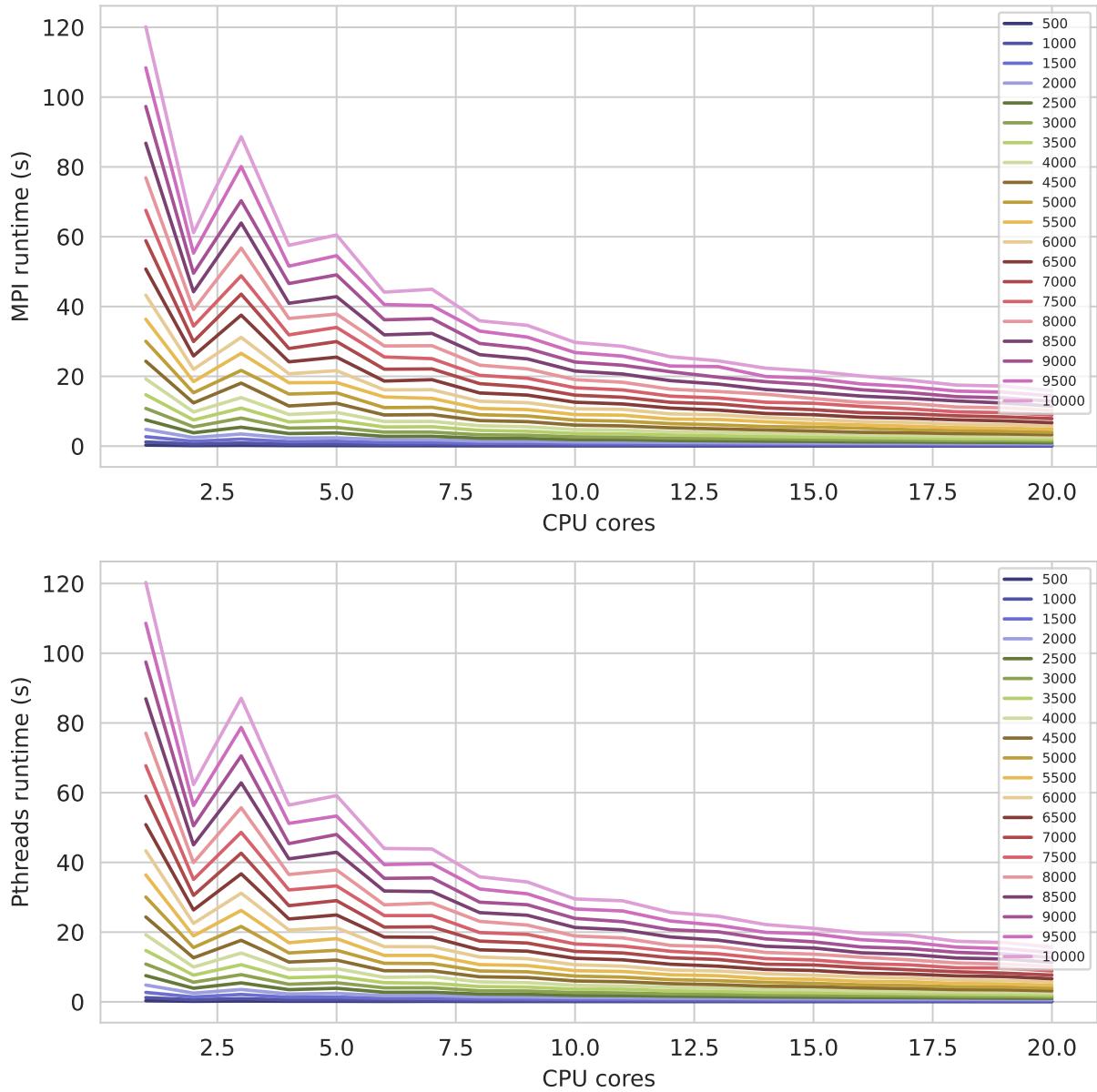


Figure 2: Runtime vs CPU cores plot.

3 Result and Discussion

3.1 Runtime

The graph of running time versus CPU cores and versus array size were plotted in Figure 2 and 3, respectively. From Figure 3, the plot clearly shows a perfect $O(n^2)$ the complexity of the algorithm, which is consistent with the theoretical analysis. From figure 2, however, for a fixed array size, the runtime does not monotonically decrease with the increase of CPU cores. It shows a zig-zag pattern from CPU cores ranging from 1 to 7. The reason behind this is the static scheduling

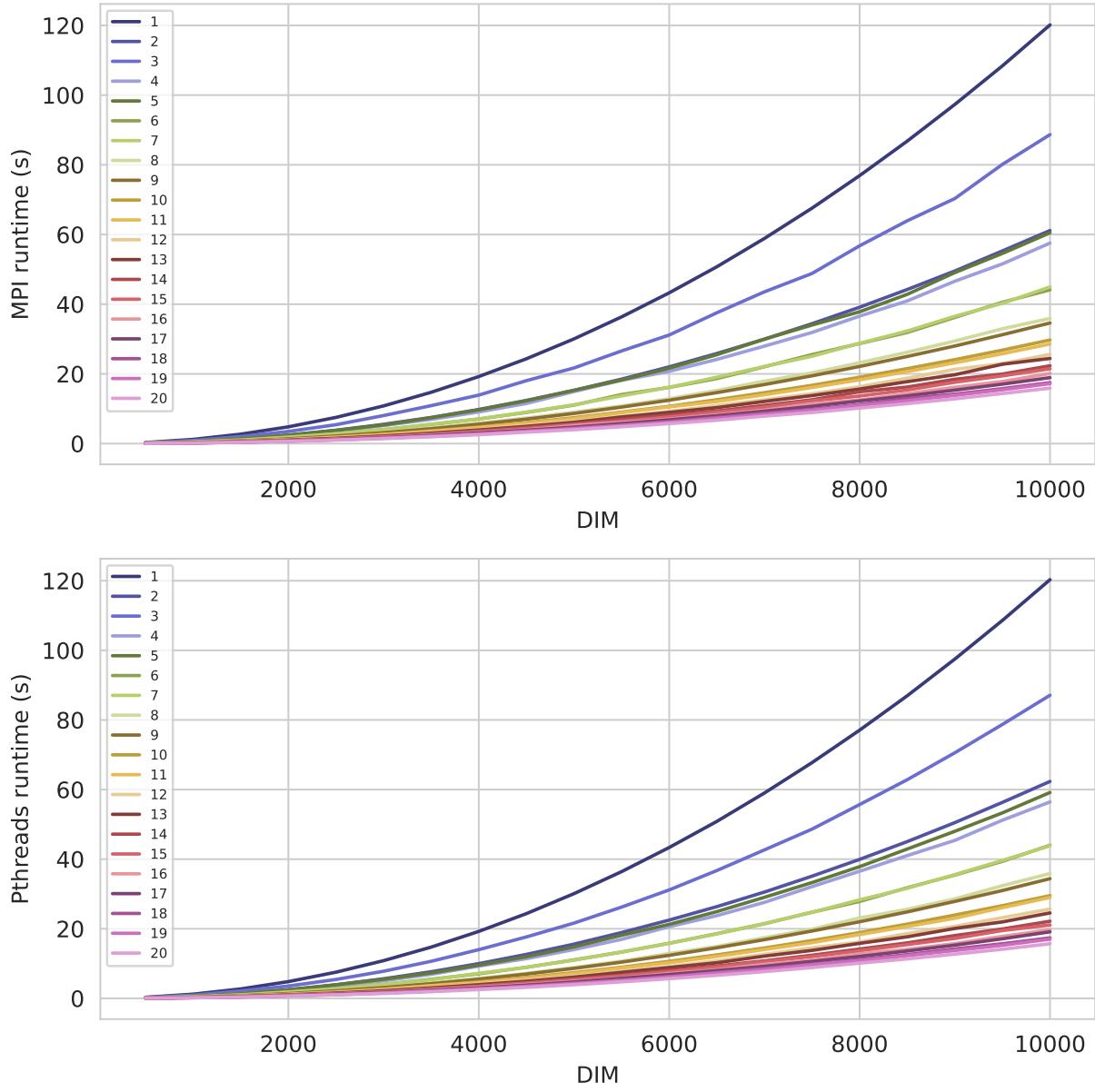


Figure 3: Runtime vs x -resolution plot.

algorithm used in the parallel program. According to the flowchart in Figure A.1 and A.2, the mesh is averagely distributed to each process/thread. Nevertheless, the computation time of each thread may not be the same, which means some thread may finish their job very fast and keep waiting other threads finish their job. This issue would be further discussed in the following sections.

Due to the static scheduling algorithm, the performance of MPI and Pthreads are approximately the same, as Figure A.1 shown.

3.2 Performance analysis

The heatmap of acceleration is plotted in Figure 4. It should be noticed that when the array size is large ($n \geq 1000$), the speed-up ratio does not change a lot, which means the time spent on communication (MPI) and shared-memory operations (Pthreads) are comparatively a small part of the overall execution time. Moreover, from the first row of the heatmaps, the acceleration rate of Pthread is relatively lower than MPI's speed-up ratio, which indicates the creation and joining of threads could be time-consuming.

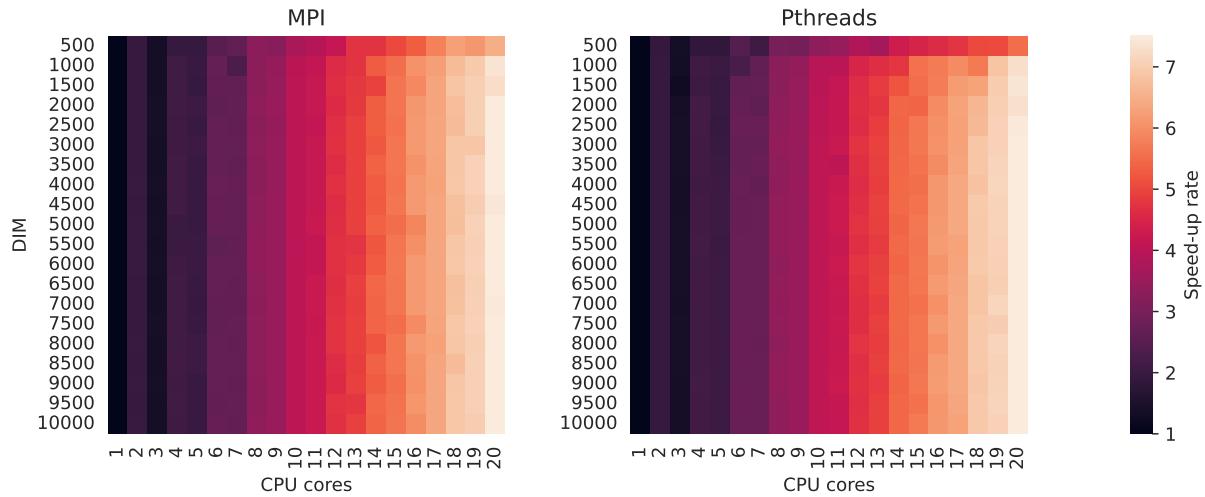


Figure 4: Speed-up ratio of two parallel programs.

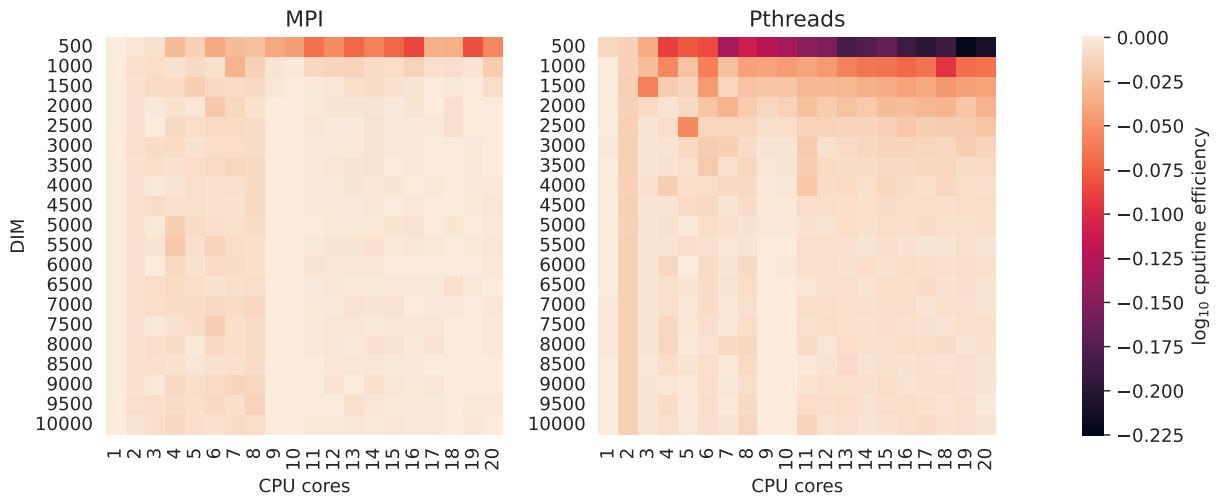


Figure 5: \log_{10} CPU time efficiency.

The heatmap of CPU time spent on computation is plotted in Figure 5. It should be clear that when the array size is large, the computation efficiency of the parallel program is the same as the computational efficiency of the sequential program. Then, we can again confirm that the relatively-low efficiency of the parallel program is caused by the scheduling algorithm which performs partition.

To solve this issue, one may apply dynamic scheduling. The dynamic scheduling would constantly check the status of each process/thread and utilize spare computational resources dynamically and efficiently. For example, the dynamic scheduling scheme in this homework is completed by set several variables: `max_idx` is the number of total jobs, `curr_idx` records how many jobs have been finished, `jobsizes` is the data points that each process will process in one iteration. In each iteration, the thread will first check if there are unfinished jobs. If there is, a computing job with a size equal to `jobsizes` will be assigned to this thread. If not, the program will terminate the calculation. In this case, `jobsizes` is set to be 500×500 . Note that Pthreads mutex lock is useful to avoid racing problems. One can refer to `src/main.pthread_ds.cpp` and `mandelbrot_loop_pt_ds` in `src/utils.h` for detailed implementation. Dynamic scheduling is useful when the problem size is large. As Figure 6 shown, in the Pthreads implementation of dynamic scheduling, it can further speed up the naive Pthreads calculation up to more than 2.5 times.

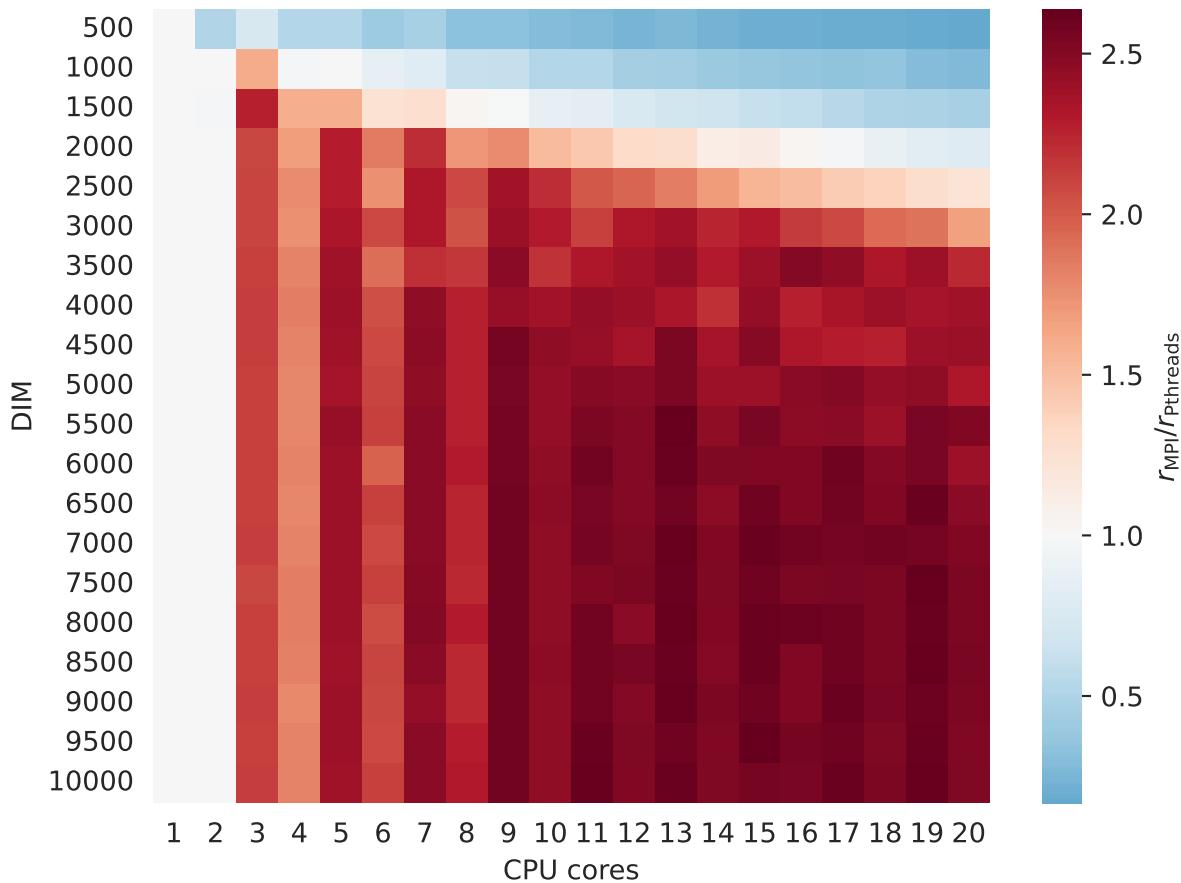


Figure 6: Dynamic scheduling speed up vs naive parallel speed-up

4 Conclusion

In conclusion, two parallel computing schemes for Mandelbrot set were implemented and their performances were compared and evaluated. One should use dynamic scheduling when dealing with large-scale computations to fully utilize computational resources.

A Supplementary figures

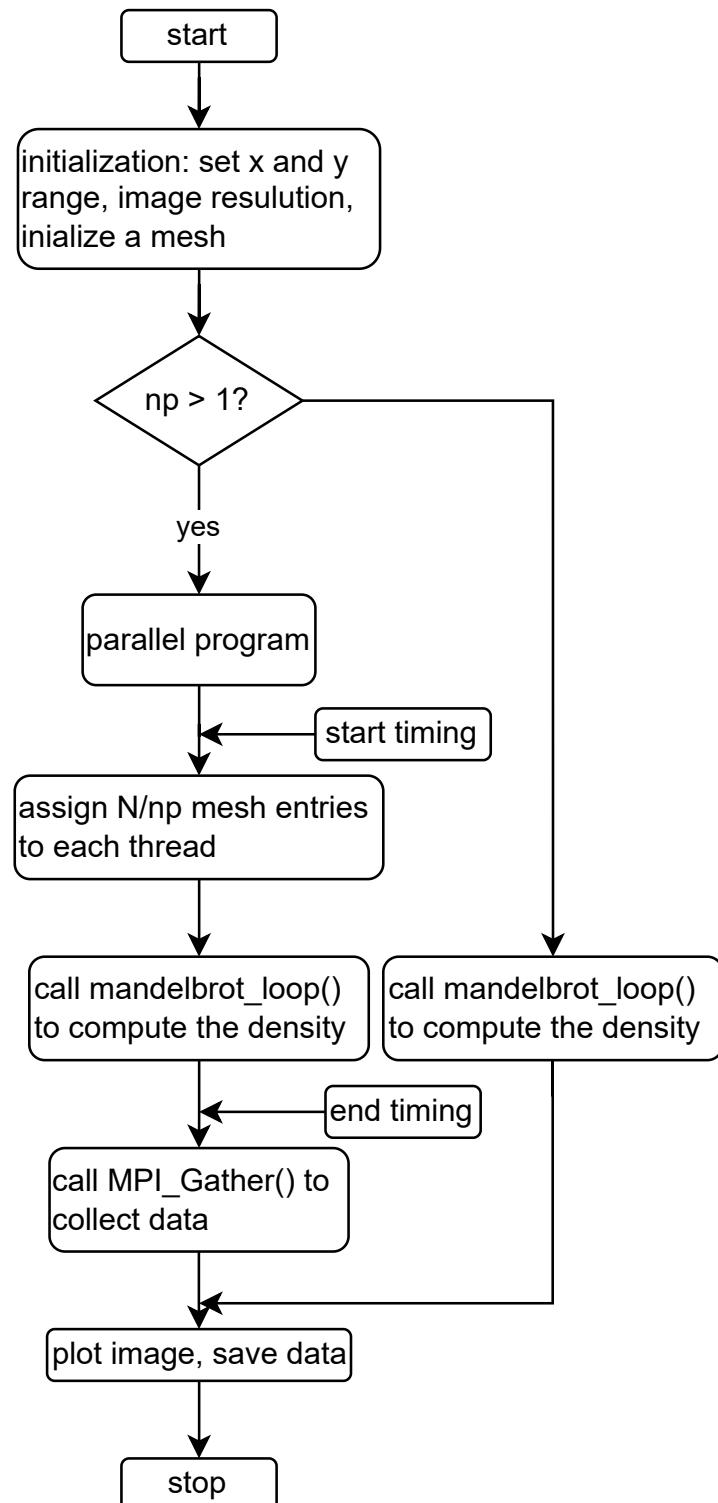


Figure A.1: MPI program flowchart

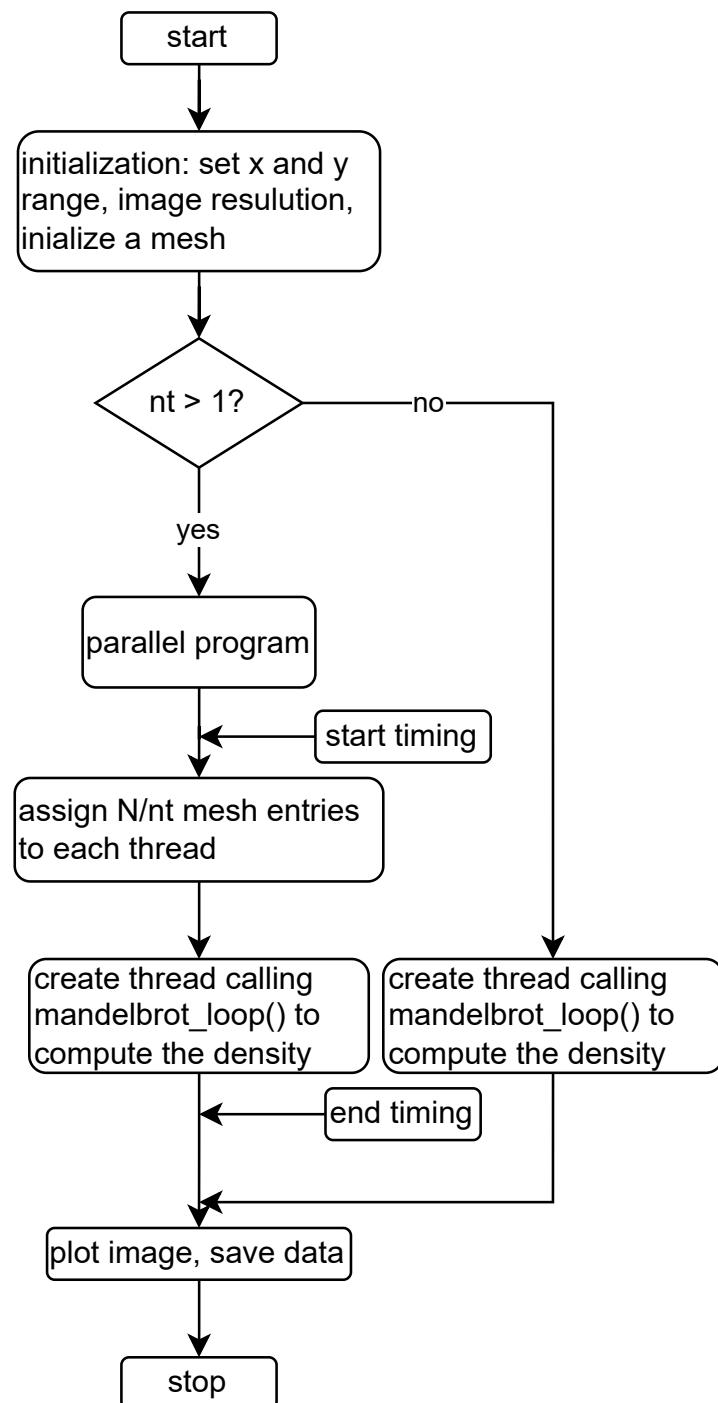


Figure A.2: Pthreads program flowchart

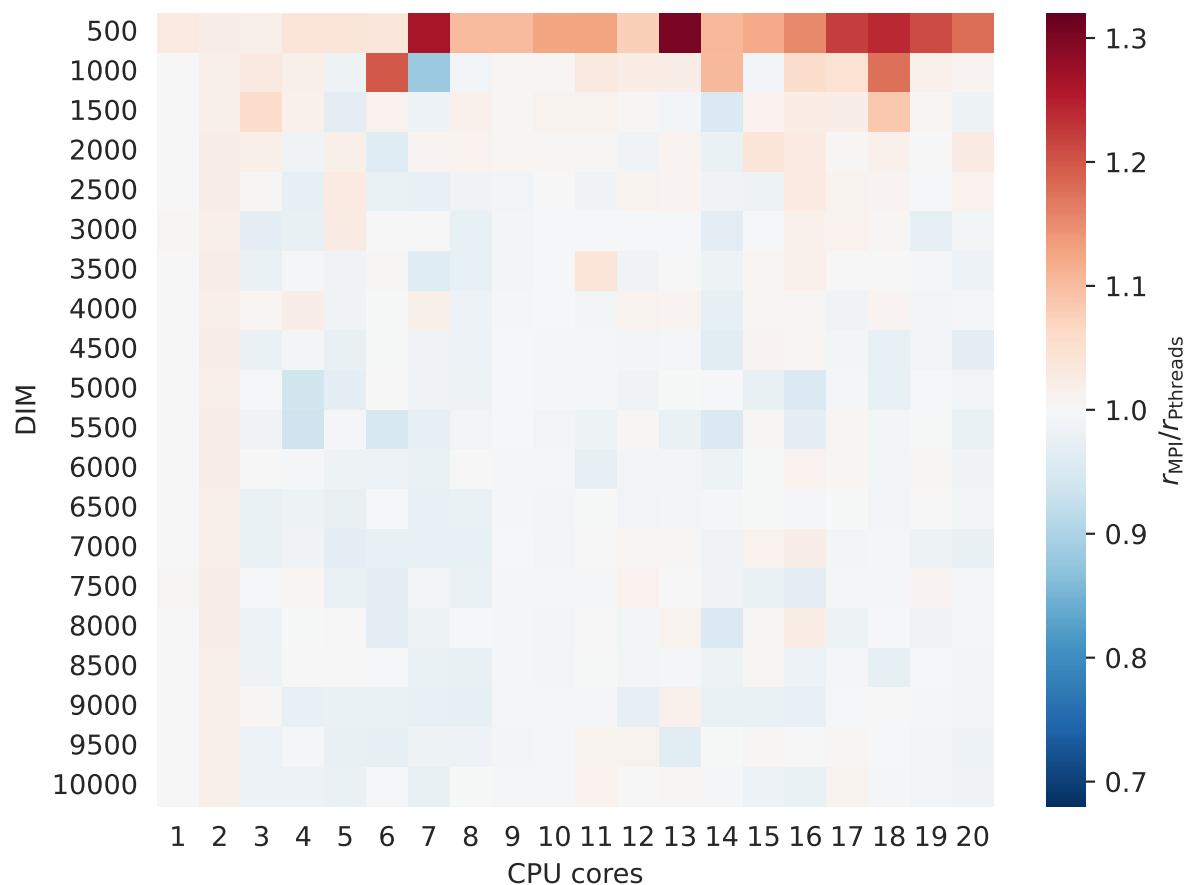


Figure A.3: Ratio of MPI and Pthreads speed-up rate.

B Source code

CMakeLists.txt

```

1 cmake_minimum_required(VERSION 3.20)
2 project(hw01)
3
4 # set output path
5 set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/lib)
6 set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/lib)
7 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)
8
9 # set include libraires
10 include_directories(src)
11
12 set(CMAKE_CXX_STANDARD 11)
13
14 # add src folder
15 add_subdirectory(src)

```

src/CMakeLists.txt

```

1 find_package(MPI REQUIRED)
2
3 # options
4 # gui option
5 option(GUI "OPENGL Rendering" OFF)
6
7 # include & requirements
8 # pthread
9 set(THREADS_PREFER_PTHREAD_FLAG ON)
10 find_package(Threads REQUIRED)
11 # mpi
12 find_package(MPI REQUIRED)
13 include_directories(${MPI_INCLUDE_PATH})
14 # opengl & glut
15 if(GUI)
16     find_package(OpenGL REQUIRED)
17     find_package(GLUT REQUIRED)
18     include_directories(${OPENGL_INCLUDE_DIRS} ${GLUT_INCLUDE_DIRS})
19     add_definitions(-DGUI)
20 endif()
21
22 # executable
23 add_executable(main.seq main.seq.cpp)
24 add_executable(main.pthread main.pthread.cpp)
25 add_executable(main.pthread_ds main.pthread_ds.cpp)
26 add_executable(main.mpi main.mpi.cpp)
27 target_link_libraries(main.pthread Threads::Threads)
28 target_link_libraries(main.pthread_ds Threads::Threads)
29 target_link_libraries(main.mpi ${MPI_LIBRARIES})
30 if(GUI)
31     target_link_libraries(main.seq ${OPENGL_LIBRARIES} ${GLUT_LIBRARIES})
32     target_link_libraries(main.mpi ${OPENGL_LIBRARIES} ${GLUT_LIBRARIES})
33     target_link_libraries(main.pthread ${OPENGL_LIBRARIES} ${GLUT_LIBRARIES})
34     target_link_libraries(main.pthread_ds ${OPENGL_LIBRARIES} ${GLUT_LIBRARIES})
35 endif()

```

src/main.seq.cpp

```

1 #include <stdio.h>
2 #include <iostream>
3 #include <fstream>
4 #include <cstdlib>
5 #include <string.h>

```



```

6 #include <chrono>
7 #include <thread>
8 #include "utils.h"
9
10 int main(int argc, char* argv[]) {
11     // initialization
12     float xmin = -2.0e-0;
13     float xmax = 0.6e-0;
14     float ymin = -1.3e-0;
15     float ymax = 1.3e-0;
16     int DIM = 500;
17     int save = 1;
18     int iter = 200;
19     int record = 0;
20
21     // parse argument
22     char buff[200];
23     for (int i = 0; i < argc; i++){
24         strcpy(buff, argv[i]);
25         if (strcmp(buff, "-n") == 0 || strcmp(buff, "--ndim") == 0){
26             std::string num(argv[i+1]);
27             DIM = std::stoi(num);
28         }
29         if (strcmp(buff, "--xmin") == 0){
30             std::string num(argv[i+1]);
31             xmin = std::stof(num);
32         }
33         if (strcmp(buff, "--xmax") == 0){
34             std::string num(argv[i+1]);
35             xmax = std::stof(num);
36         }
37         if (strcmp(buff, "--ymin") == 0){
38             std::string num(argv[i+1]);
39             ymin = std::stof(num);
40         }
41         if (strcmp(buff, "--ymax") == 0){
42             std::string num(argv[i+1]);
43             ymax = std::stof(num);
44         }
45         if (strcmp(buff, "--iter") == 0){
46             std::string num(argv[i+1]);
47             iter = std::stof(num);
48         }
49         if (strcmp(buff, "--save") == 0){
50             std::string num(argv[i+1]);
51             save = std::stoi(num);
52         }
53         if (strcmp(buff, "--record") == 0){
54             std::string num(argv[i+1]);
55             record = std::stoi(num);
56         }
57     }
58     // postprocessing
59     int xDIM = DIM;
60     int yDIM = int(DIM*(ymax-ymin)/(xmax-xmin));
61
62     // print info
63     print_info(xDIM, yDIM);
64
65     // allocation and initialization
66     std::complex<float> *Z = (std::complex<float> *)malloc(sizeof(std::complex<float>)*
67                                         yDIM*xDIM);
68     char *map = (char *)malloc(sizeof(char) * xDIM * yDIM);
69     mandelbrot_init(Z, xDIM, yDIM, xmin, xmax, ymin, ymax);
70
71     // start time

```



```

72     auto t1 = std::chrono::system_clock::now();
73
74     // MAIN program
75     mandelbrot_loop(Z, map, 0, xDIM*yDIM, iter);
76
77     // end time
78     auto t2 = std::chrono::system_clock::now();
79     auto dur = t2 - t1;
80     auto dur_ = std::chrono::duration_cast<std::chrono::duration<double>>(dur);
81     double t = dur_.count();
82
83     // record data
84     if (record==1) runtime_record("seq", DIM, 1, t, t);
85
86     // save png
87     if (save==1) mandelbrot_save("seq", map, xDIM, yDIM);
88
89     // end time
90     runtime_print(DIM, 1, t, t);
91
92     // rendering
93     #ifdef GUI
94         // copy memory
95         map_glut = (char *)malloc(sizeof(char)*xDIM*yDIM);
96         memcpy(map_glut, map, sizeof(char)*xDIM*yDIM);
97         // plot
98         xDIM_glut = xDIM;
99         yDIM_glut = yDIM;
100        render("seq");
101        free(map_glut);
102    #endif
103
104    // free arrays
105    free(Z);
106    free(map);
107
108    return 0;
109 }
```

src/main.mpi.cpp

```

1 #include <stdio.h>
2 #include <iostream>
3 #include <fstream>
4 #include <cstdlib>
5 #include <string.h>
6 #include <chrono>
7 #include <thread>
8 #include <mpi.h>
9 #include "utils.h"
10
11
12 int main(int argc, char* argv[]) {
13     // mpi initialization
14     MPI_Init(NULL, NULL);
15     // fetch size and rank
16     int size, rank;
17     MPI_Comm_size(MPI_COMM_WORLD, &size);
18     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
19
20     // initialization
21     float xmin = -2.0e-0;
22     float xmax = 0.6e-0;
23     float ymin = -1.3e-0;
24     float ymax = 1.3e-0;
25     int DIM = 500;
```



```

27   int save = 1;
28   int iter = 200;
29   int record = 0;
30
31   // parse argument
32   char buff[200];
33   for (int i = 0; i < argc; i++){
34     strcpy(buff, argv[i]);
35     if (strcmp(buff, "-n") == 0 || strcmp(buff, "--ndim") == 0){
36       std::string num(argv[i+1]);
37       DIM = std::stoi(num);
38     }
39     if (strcmp(buff, "--xmin") == 0){
40       std::string num(argv[i+1]);
41       xmin = std::stof(num);
42     }
43     if (strcmp(buff, "--xmax") == 0){
44       std::string num(argv[i+1]);
45       xmax = std::stof(num);
46     }
47     if (strcmp(buff, "--ymin") == 0){
48       std::string num(argv[i+1]);
49       ymin = std::stof(num);
50     }
51     if (strcmp(buff, "--ymax") == 0){
52       std::string num(argv[i+1]);
53       ymax = std::stof(num);
54     }
55     if (strcmp(buff, "--iter") == 0){
56       std::string num(argv[i+1]);
57       iter = std::stof(num);
58     }
59     if (strcmp(buff, "--save") == 0){
60       std::string num(argv[i+1]);
61       save = std::stoi(num);
62     }
63     if (strcmp(buff, "--record") == 0){
64       std::string num(argv[i+1]);
65       record = std::stoi(num);
66     }
67   }
68   // postprocessing
69   int xDIM = DIM;
70   int yDIM = int(DIM*(ymax-ymin)/(xmax-xmin));
71
72   // pre-defined variables
73   std::complex<float> *Z;
74   std::complex<float> *Z_;
75   char *map;
76   char *map_;
77   int start_idx = xDIM*yDIM/size * rank;
78   int end_idx = xDIM*yDIM/size * (rank+1);
79   if (rank==size-1) end_idx = xDIM*yDIM;
80   if (rank==0){
81     // print info
82     print_info(xDIM, yDIM);
83     // allocation and initialization
84     Z = (std::complex<float> *)malloc(sizeof(std::complex<float>)*yDIM*xDIM);
85     map = (char *)malloc(sizeof(char) * xDIM * yDIM);
86     mandelbrot_init(Z, xDIM, yDIM, xmin, xmax, ymin, ymax);
87   }
88   // allocate local variables for each process
89   Z_ = (std::complex<float> *)malloc(sizeof(std::complex<float>) * (end_idx-start_idx)
90                                         );
91   map_ = (char *)malloc(sizeof(char) * (end_idx-start_idx));
92
93   // timing

```



```

93     double t1, t2, t1_, t2_;
94
95     // MAIN program
96     // start timing
97     t1 = MPI_Wtime();
98     // CASE 1: sequential
99     if (size==1){
100         t1_ = MPI_Wtime();
101         mandelbrot_loop(Z, map, 0, xDIM*yDIM, iter);
102         t2_ = MPI_Wtime();
103     }
104     // CASE 2: parallel
105     else {
106         // distribute the data
107         int scale = sizeof(std::complex<float>) / sizeof(int);
108         if (rank==0) {
109             for (int i = 1; i < size; i++){
110                 int start = xDIM*yDIM/size * i;
111                 int end   = xDIM*yDIM/size * (i+1); if (i==size-1) end = xDIM*yDIM;
112                 MPI_Send((int *) (Z+start), (end-start)*scale, MPI_INT, i, 0,
113                           MPI_COMM_WORLD);
114             }
115             for (int i = 0; i < xDIM*yDIM/size; i++) Z_[i] = Z[i];
116         }
117         else {
118             MPI_Recv((int *) Z_, (end_idx-start_idx)*scale, MPI_INT, 0, 0,
119                       MPI_COMM_WORLD, MPI_STATUS_IGNORE);
120         }
121         // // print check
122         // printf("rank %d start_idx %d end_idx %d print %f + %fi\n",
123         //        rank, start_idx, end_idx, std::real(Z_[0]), std::imag(Z_[0]));
124
125         // start timing
126         t1_ = MPI_Wtime();
127
128         // execution
129         mandelbrot_loop(Z_, map_, 0, end_idx-start_idx, iter);
130
131         // end timing
132         t2_ = MPI_Wtime();
133
134         // gather data
135         MPI_Gather(map_, xDIM*yDIM/size, MPI_CHAR, map, xDIM*yDIM/size, MPI_CHAR, 0,
136                     MPI_COMM_WORLD);
137         MPI_Barrier(MPI_COMM_WORLD);
138         // tail case
139         if (xDIM*yDIM%size != 0){
140             if (rank == size-1) MPI_Send(map_+xDIM*yDIM/size, xDIM*yDIM%size, MPI_CHAR,
141                                         0, 2, MPI_COMM_WORLD);
142             if (rank == 0) MPI_Recv(map+xDIM*yDIM/size*size, xDIM*yDIM%size, MPI_CHAR,
143                                     size-1, 2, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
144         }
145         MPI_Barrier(MPI_COMM_WORLD);
146     }
147     // end timing
148     t2 = MPI_Wtime();
149
150     // end time
151     double t = t2 - t1;    // overall execution time
152     double t_ = t2_ - t1_; // cpu time on calculation
153     double *time_arr = (double *)malloc(sizeof(double) * size);
154     double t_sum = 0;
155     MPI_Gather(&t_, 1, MPI_DOUBLE, time_arr, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
156     MPI_Barrier(MPI_COMM_WORLD);
157     for (int i = 0; i < size; i++){
158         t_sum += time_arr[i];
159     }

```



```

155     MPI_Barrier(MPI_COMM_WORLD);
156
157     // record data
158     if (rank==0 && record==1){
159         runtime_record("mpi", DIM, size, t, t_sum);
160         runtime_record_detail("mpi", DIM, size, t, time_arr);
161     }
162
163     // save png
164     if (rank==0 && save==1) mandelbrot_save("mpi", map, xDIM, yDIM);
165
166     // sync
167     MPI_Barrier(MPI_COMM_WORLD);
168
169     // end time
170     if (rank==0) runtime_print(DIM, size, t, t_sum);
171
172     // rendering
173     #ifdef GUI
174     if (rank==0){
175         // copy memory
176         map_glut = (char *)malloc(sizeof(char)*xDIM*yDIM);
177         memcpy(map_glut, map, sizeof(char)*xDIM*yDIM);
178         // plot
179         xDIM_glut = xDIM;
180         yDIM_glut = yDIM;
181         render("mpi");
182         free(map_glut);
183     }
184     #endif
185
186     // free arrays
187     if (rank==0){
188         free(Z);
189         free(map);
190     }
191     free(Z_);
192     free(map_);
193     MPI_Barrier(MPI_COMM_WORLD);
194
195     // mpi finalization
196     MPI_Finalize();
197
198     return 0;
199 }
```

src/main(pthread).cpp

```

1 #include <stdio.h>
2 #include <iostream>
3 #include <fstream>
4 #include <cstdlib>
5 #include <string.h>
6 #include <chrono>
7 #include <thread>
8 #include <pthread.h>
9 #include "utils.h"
10
11
12 int main(int argc, char* argv[]) {
13     // initialization
14     float xmin = -2.0e-0;
15     float xmax = 0.6e-0;
16     float ymin = -1.3e-0;
17     float ymax = 1.3e-0;
18     int DIM = 500;
19     int save = 1;
```



```

20     int iter = 200;
21     int record = 0;
22
23 // pthread specific args
24     int nt = 1;
25
26 // parse argument
27     char buff[200];
28     for (int i = 0; i < argc; i++){
29         strcpy(buff, argv[i]);
30         if (strcmp(buff, "-n") == 0 || strcmp(buff, "--ndim") == 0){
31             std::string num(argv[i+1]);
32             DIM = std::stoi(num);
33         }
34         if (strcmp(buff, "-nt") == 0 || strcmp(buff, "--nthread") == 0){
35             std::string num(argv[i+1]);
36             nt = std::stoi(num);
37         }
38         if (strcmp(buff, "--xmin") == 0){
39             std::string num(argv[i+1]);
40             xmin = std::stof(num);
41         }
42         if (strcmp(buff, "--xmax") == 0){
43             std::string num(argv[i+1]);
44             xmax = std::stof(num);
45         }
46         if (strcmp(buff, "--ymin") == 0){
47             std::string num(argv[i+1]);
48             ymin = std::stof(num);
49         }
50         if (strcmp(buff, "--ymax") == 0){
51             std::string num(argv[i+1]);
52             ymax = std::stof(num);
53         }
54         if (strcmp(buff, "--iter") == 0){
55             std::string num(argv[i+1]);
56             iter = std::stof(num);
57         }
58         if (strcmp(buff, "--save") == 0){
59             std::string num(argv[i+1]);
60             save = std::stoi(num);
61         }
62         if (strcmp(buff, "--record") == 0){
63             std::string num(argv[i+1]);
64             record = std::stoi(num);
65         }
66     }
67 // postprocessing
68     int xDIM = DIM;
69     int yDIM = int(DIM*(ymax-ymin)/(xmax-xmin));
70
71 // print info
72     print_info(xDIM, yDIM);
73
74 // allocation and initialization
75     std::complex<float> *Z = (std::complex<float> *)malloc(sizeof(std::complex<float>)*
76                                     yDIM*xDIM);
77     char *map = (char *)malloc(sizeof(char) * xDIM * yDIM);
78     mandelbrot_init(Z, xDIM, yDIM, xmin, xmax, ymin, ymax);
79     Ptargs *args = (Ptargs *)malloc(sizeof(Ptargs) * nt);
80     pthread_t *threads = (pthread_t *)malloc(sizeof(pthread_t) * nt);
81
82 // start time
83     auto t1 = std::chrono::system_clock::now();
84     double *time_arr = (double *)malloc(sizeof(double)*nt);
85
// MAIN program

```



```

86     // create threads
87     for (int i = 0; i < nt; i++){
88         // calculate start and end index
89         int start_idx = xDIM*yDIM/nt * i;
90         int end_idx = xDIM*yDIM/nt * (i+1);
91         args[i] = {Ptargs}{.Z=Z, .map=map, .start_idx=start_idx, .end_idx=end_idx, .iter
92             =iter, .id=i, .time_arr=time_arr};
93         if (i==nt-1) args[i].end_idx = xDIM*yDIM;
94
95         // create independent threads
96         pthread_create(&threads[i], NULL, mandelbrot_loop_pt, (void *)(&args[i]));
97     }
98     // join threads
99     for (int i = 0; i < nt; i++){
100        pthread_join(threads[i], NULL);
101    }
102
103    // end time
104    auto t2 = std::chrono::system_clock::now();
105    auto dur = t2 - t1;
106    auto dur_ = std::chrono::duration_cast<std::chrono::duration<double>>(dur);
107    double t = dur_.count();
108    double t_sum = 0;
109    for (int i = 0; i < nt; i++) t_sum += time_arr[i];
110
111    // record data
112    if (record==1){
113        runtime_record("pth", DIM, nt, t, t_sum);
114        runtime_record_detail("pth", DIM, nt, t, time_arr);
115    }
116
117    // save png
118    if (save==1) mandelbrot_save("pth", map, xDIM, yDIM);
119
120    // end time
121    runtime_print(DIM, nt, t, t_sum);
122
123    // rendering
124    #ifdef GUI
125        // copy memory
126        map_glut = (char *)malloc(sizeof(char)*xDIM*yDIM);
127        memcpy(map_glut, map, sizeof(char)*xDIM*yDIM);
128        // plot
129        xDIM_glut = xDIM;
130        yDIM_glut = yDIM;
131        render("pthread");
132        free(map_glut);
133    #endif
134
135    // free arrays
136    free(Z);
137    free(map);
138
139    return 0;
140 }
```

src/utils.h

```

1 #define STB_IMAGE_WRITE_IMPLEMENTATION
2 #include <stdio.h>
3 #include <iostream>
4 #include <complex>
5 #include "stb_image_write.h"
6 #include <sys/stat.h>
7 #include <sys/types.h>
8
```



```

9 #ifdef GUI
10 #include <GL/glut.h>
11 #include <GL/glu.h>
12 #include <GL/gl.h>
13
14 char *map_glut;
15 int xDIM_glut, yDIM_glut;
16 int width = 1000;
17 int xwidth, ywidth;
18 #endif
19
20 typedef struct dsargs{
21     int jobszie;
22     int curr_idx;
23     int max_idx;
24     pthread_mutex_t *mutex_ptr;
25 } Dsargs;
26
27 typedef struct ptargs{
28     std::complex<float> *Z;
29     char *map;
30     int start_idx;
31     int end_idx;
32     int iter;
33     int id;
34     double *time_arr;
35     Dsargs *dsptr;
36 } Ptargs;
37
38 void print_info(int xDIM, int yDIM){
39     printf("Name: Haoran Sun\n");
40     printf("ID: 119010271\n");
41     printf("HW: Mandelbrot Set Computation\n");
42     printf("Set xDIM to %d, yDIM to %d\n", xDIM, yDIM);
43 }
44
45 void mandelbrot_init(std::complex<float> *Z, int xDIM, int yDIM, float xmin, float xmax,
46     float ymin, float ymax){
47     for (int i = 0; i < yDIM; i++){
48         for (int j = 0; j < xDIM; j++){
49             float x = (xmax-xmin)/xDIM*j + xmin;
50             float y = (ymin-ymax)/yDIM*i + ymax;
51             // printf("%f %f\n", x, y);
52             Z[i*xDIM+j] = std::complex<float>(x, y);
53         }
54     }
55 }
56
57 char mandelbrot_iter(std::complex<float> z, std::complex<float> z0, int iter){
58     std::complex<float> p = z;
59     for (int i = 0; i < iter; i++){
60         z = z * z + z0;
61         if (std::real(z * std::conj(z)) > 4) return 255 - 255 * i/iter;
62     }
63     return 0;
64 }
65
66 void mandelbrot_loop(std::complex<float> *Z, char *map, int start_idx, int end_idx, int
67     iter){
68     for (int i = start_idx; i < end_idx; i++){
69         map[i] = mandelbrot_iter(Z[i], Z[i], iter);
70     }
71 }
72
73 void *mandelbrot_loop_pt(void *vargs){
74     // transfer args
75     Ptargs args = *(Ptargs *)vargs;

```



```

74     double *time_arr = args.time_arr;
75     int id = args.id;
76     // start time
77     auto t1 = std::chrono::system_clock::now();
78
79     // main loop
80     mandelbrot_loop(args.Z, args.map, args.start_idx, args.end_idx, args.iter);
81
82     // end time
83     auto t2 = std::chrono::system_clock::now();
84     auto dur = t2 - t1;
85     auto dur_ = std::chrono::duration_cast<std::chrono::duration<double>>(dur);
86     double t = dur_.count();
87     time_arr[id] = t;
88
89     return NULL;
90 }
91
92
93 void *mandelbrot_loop_pt_ds(void *vargs){
94     // transfer args
95     Ptargs args = *(Ptargs *)vargs;
96     double *time_arr = args.time_arr;
97     int id = args.id;
98     int max_idx = args.dsprt->max_idx;
99     // start time
100    auto t1 = std::chrono::system_clock::now();
101
102    // get mutex
103    pthread_mutex_t *mutex_ptr = args.dsprt->mutex_ptr;
104    while (true){
105        // read parameters from global scheduling parameters
106        pthread_mutex_lock(mutex_ptr);
107        int start_idx = args.dsprt->curr_idx;
108        int end_idx = start_idx + args.dsprt->jobszie;
109        if (end_idx > max_idx) end_idx = max_idx;
110        args.dsprt->curr_idx = end_idx;
111        pthread_mutex_unlock(mutex_ptr);
112        if (start_idx>=max_idx) break;
113        // main loop
114        // printf("id = %d, s = %d, e = %d, m = %d\n", id, start_idx, end_idx, max_idx);
115        mandelbrot_loop(args.Z, args.map, start_idx, end_idx, args.iter);
116    }
117
118    // end time
119    auto t2 = std::chrono::system_clock::now();
120    auto dur = t2 - t1;
121    auto dur_ = std::chrono::duration_cast<std::chrono::duration<double>>(dur);
122    double t = dur_.count();
123    time_arr[id] = t;
124
125    return NULL;
126 }
127
128 void mandelbrot_save(const char *jobtype, char *map,
129     int xDIM, int yDIM){
130     char filebuff[200];
131     snprintf(filebuff, sizeof(filebuff), "mandelbrot_%s.png", jobtype);
132     stbi_write_png(filebuff, xDIM, yDIM, 1, map, 0);
133     printf("Image saved as %s.\n", filebuff);
134 }
135
136 void runtime_record(const char *jobtype, int N, int nt, double t, double t_sum){
137     const char *folder = "data";
138     mkdir(folder, 0777);
139     FILE* outfile;
140     char filebuff[200];

```



```

141     snprintf(filebuff, sizeof(filebuff), "./%s/runtime_%s.txt", folder, jobtype);
142     outfile = fopen(filebuff, "a");
143     fprintf(outfile, "%10d %5d %10.4f %10.4f\n", N, nt, t, t_sum);
144     fclose(outfile);
145     printf("Runtime added in %.txt", filebuff);
146 }
147
148 void runtime_record_detail(const char *jobtype, int N, int nt, double t, double *
149   time_arr){
150   const char *folder = "data";
151   mkdir(folder, 0777);
152   FILE* outfile;
153   char filebuff[200];
154   snprintf(filebuff, sizeof(filebuff), "./%s/runtime_detailed_%s_%d.txt", folder,
155   jobtype, nt);
156   outfile = fopen(filebuff, "a");
157   fprintf(outfile, "%10d %5d %10.4f ", N, nt, t);
158   for (int i = 0; i < nt; i++){
159     fprintf(outfile, "%10.4f ", time_arr[i]);
160   }
161   fprintf(outfile, "\n");
162   fclose(outfile);
163   printf("Detailed runtime added in %.txt", filebuff);
164 }
165
166 void runtime_print(int N, int nt, double t, double t_sum){
167   printf("Execution time: %.2fs, cpu time: %.2fs, #cpu %2d\n", t, t_sum, nt);
168 }
169
170 #ifdef GUI
171
172 void display_test(){
173   glClear(GL_COLOR_BUFFER_BIT);
174
175   glBegin(GL_POLYGON);
176   glVertex2f(0, 0);
177   glVertex2f(1, 0);
178   glVertex2f(1, 1);
179   glVertex2f(0, 1);
180   glEnd();
181
182   glFlush();
183 }
184
185 void plot(){
186   // display test
187   // initialization
188   glClear(GL_COLOR_BUFFER_BIT);
189   glColor3f(0.0f, 0.0f, 0.0f);
190
191   // draw points
192   GLfloat pointSize = 1.0f;
193   glPointSize(pointSize);
194   glBegin(GL_POINTS);
195   glClear(GL_COLOR_BUFFER_BIT);
196   for (int i = 0; i < yDIM_glut; i++){
197     for (int j = 0; j < xDIM_glut; j++){
198       int c0 = (unsigned char) map_glut[i*xDIM_glut+j];
199       float c = c0 / 255.0;
200       glColor3f(c, c, c);
201       glVertex2f(j, yDIM_glut-i);
202     }
203   }
204   glEnd();
205
206   // flush

```



```
206     glFlush();
207 }
208
209 void resize(int x, int y){
210     glutReshapeWindow(xwidth, ywidth);
211 }
212
213
214 void render(const char *jobtype){
215     // glu init
216     int glufoo = 1;
217     char q[] = " ";
218     char *glubar[1];
219     glubar[0] = q;
220     glutInit(&glufoo, glubar);
221     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
222
223     // set x and y width
224     xwidth = width;
225     ywidth = yDIM_glut*width/xDIM	glut;
226     glutInitWindowSize(xwidth, ywidth);
227     glutCreateWindow(jobtype);
228     glMatrixMode(GL_PROJECTION);
229     gluOrtho2D(0, xDIM_glut, 0, yDIM_glut);
230
231     // display func
232     glutDisplayFunc(plot);
233     // glutDisplayFunc(display_test);
234     glutReshapeFunc(resize);
235
236     glutMainLoop();
237 }
238
239 #endif
```

