CSC4005

Distributed and Parallel Computing

Assignment #2

Hajun Lee

117010437

Introduction

In assignment2, we should implement 'Mandelbrot set computation' by using MPI and Pthread.

It designed for communication between local threads and multiprocessors. Quasi-stable is that the numbers of the requirement of Mandelbrot set won't exceed threshold even though it have repeated several times of computation.

For MPI version, it should defined tasks for calculate X_RESN * Y_RESN / N points and gather them together after computation quasi-stable in master program.

For Pthread version, it should defined number of threads for calculate X_RESN * Y_RESN / N points and then save to quasi-stable point.

Method

Set of points in a complex plane that are quasi-stable (will increase and decrease, but not exceed some limit) when computed by iterating the function is below.

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

z_{k+1} : (k+1) iteration of the complex number of z.

c : complex number of the point in the complex plan.

For c, we can calculate with height and width

C = (x - height/2) / (height / 4) + (y – width / 2) / (width / 4) * i

First, start with initial value 0, iteration will stop when z_k is hight then threshold or the maximum number of iterations.

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

In the graph, each pixel should compute z_k and draw pixel on the graph when z_k is Quasi-stable

Design

1. Pthread.cpp

Before main function:

1. Include Header files and defined namespace.
2. Modified constant variables.

Main function:

1. Define structure about complex type and Thread_data, and initialize global variables.
2. Initialize the number of threads, X value of the graph and Y value of the graph which are width and height from the input value
3. Configuration of Xlib drawing
    A. Initialize related variables
    B. Get screen size and error condition
    C. Set window size
    D. Set window position
    E. Create opaque window
    F. Create graphics context
4. Timer start
5. For master, create threads and calculate left area and join threads.
6. For slave, execute cal_func() to initialize local variables, apply and calculate the result and save the result.
7. Timer stop and print personal information.
8. Draw and show the graph on the screen.

2. MPI.cpp

Before main function:

1. Include Header files and defined namespace.
2. Modified constant variables.

Main function:

1. Define structure about complex type and initialize global variables.
2. Initialize X value of the graph and Y value of the graph which are width and height from the input value.
3. Configuration of Xlib drawing
    A. Initialize related variables
    B. Get screen size and error condition
    C. Set window size
    D. Set window position
    E. Create opaque window
    F. Create graphics context
4. Timer start
5. For master, create threads and calculate left area and join threads.
6. For slave, execute cal_func() to initialize local variables, apply and calculate the result and save the result.
7. Timer stop and print personal information.
8. Draw and show the graph on the screen.
9. MPI finish

Performance Analysis:

Pthread:

For pthread, basically when the size bigger, execution time is also will be bigger and when number of threads increase, execution time decreases.

MPI:

For MPI, basically when the size bigger, execution time is also will be bigger and when number of threads increase, execution time decreases.

Compare between MPI and pthread:

To compare between MPI and pthread, when the size is small, MPI has better performance than pthread but when size is big enough, pthread has better performance than MPI

pthread.cpp Code:

C: > Users > Andy > Desktop > pthread.cpp > ...

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

using namespace std;


#define         X_RESN  800       /* x resolution */
#define         Y_RESN  800       /* y resolution */
#define         NUM_THREADS 4

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

typedef struct _thread_data {
    int thread_id;
}thread_data;


int *result;
```

```
72  int main ()
73  {
74      Window          win;                            /* initialization for a window */
75      unsigned
76      int             width, height,                  /* window size */
77                      x, y,                           /* window position */
78                      border_width,                   /*border width in pixels */
79                      display_width, display_height,  /* size of screen */
80                      screen;                         /* which screen */
81
82      char            *window_name = "Mandelbrot Set", *display_name = NULL;
83      GC              gc;
84      unsigned
85      long            valuemask = 0;
86      XGCValues       values;
87      Display         *display;
88      XSizeHints      size_hints;
89      Pixmap          bitmap;
90      XPoint          points[800];
91      FILE            *fp, *fopen ();
92      char            str[100];
93
94      XSetWindowAttributes attr[1];
95
96      /* Mandlebrot variables */
97      int i, j, k;
98      Compl   z, c;
99      float   lengthsq, temp;
00
01      /* connect to Xserver */
02
03      if (  (display = XOpenDisplay (display_name)) == NULL ) {
04          fprintf (stderr, "drawon: cannot connect to X server %s\n",
05                      XDisplayName (display_name) );
06      exit (-1);
07      }
08
09      /* get screen size */
10
11      screen = DefaultScreen (display);
12      display_width = DisplayWidth (display, screen);
13      display_height = DisplayHeight (display, screen);
14
15      /* set window size */
16
17      width = X_RESN;
18      height = Y_RESN;
19
```

```
120     /* set window position */
121
122     x = 0;
123     y = 0;
124
125     /* create opaque window */
126
127     border_width = 4;
128     win = XCreateSimpleWindow (display, RootWindow (display, screen),
129                                x, y, width, height, border_width,
130                                BlackPixel (display, screen), WhitePixel (display, screen));
131
132     size_hints.flags = USPosition|USSize;
133     size_hints.x = x;
134     size_hints.y = y;
135     size_hints.width = width;
136     size_hints.height = height;
137     size_hints.min_width = 300;
138     size_hints.min_height = 300;
139
140     XSetNormalHints (display, win, &size_hints);
141     XStoreName(display, win, window_name);
142
143     /* create graphics context */
144
145     gc = XCreateGC (display, win, valuemask, &values);
146
147     XSetBackground (display, gc, WhitePixel (display, screen));
148     XSetForeground (display, gc, BlackPixel (display, screen));
149     XSetLineAttributes (display, gc, 1, LineSolid, CapRound, JoinRound);
150
151     attr[0].backing_store = Always;
152     attr[0].backing_planes = 1;
153     attr[0].backing_pixel = BlackPixel(display, screen);
154
155     XChangeWindowAttributes(display, win, CWBackingStore | CWBackingPlanes | CWBackingPixel, attr);
156
157     XMapWindow (display, win);
158     XSync(display, 0);
159
160
161     struct timeval timeStart, timeEnd, timeSystemStart;
162     double totalTime = 0, systemRunTime;
163     gettimeofday(&timeStart, NULL);
164
```

```
165
166     //INIT THREAD
167
168     result = (int *)malloc(sizeof(int) * (X_RESN * Y_RESN));
169
170     pthread_t thread[NUM_THREADS];
171     thread_data input_data[NUM_THREADS];
172
173     int tot_part = X_RESN / NUM_THREADS;
174
175     Compl   z, c;
176     int i, j, k;
177     double  lengthsq, temp;
178     int tempA;
179
180     for (tempA = 0; tempA < NUM_THREADS; tempA++) {
181         input_data[tempA].thread_id = tempA;
182
183         int rc = pthread_create(&thread[tempA], NULL, cal_func, &input_data[tempA]);
184         if (rc) {
185             fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
186             return EXIT_FAILURE;
187         }
188     }
189
190     if (X_RESN % NUM_THREADS != 0){
191         int tot_left = X_RESN % NUM_THREADS;
192         int tot_start = NUM_THREADS * tot_part;
193         int tot_part = tot_left;
```

```
190     if (X_RESN % NUM_THREADS != 0){
191         int tot_left = X_RESN % NUM_THREADS;
192         int tot_start = NUM_THREADS * tot_part;
193         int tot_part = tot_left;
194
195         for (i = tot_start; i < tot_start + tot_part; i++) {
196             for (j = 0; j < Y_RESN; j++) {
197                 z.real = z.imag = 0.0;
198                 c.real = ((float)j - Y_RESN / 2) / (Y_RESN / 4);  //scale factors for 800 x 80
199                 c.imag = ((float)i - X_RESN / 2) / (X_RESN / 4);
200                 k = 0;
201
202                 do {
203
204                     temp = z.real*z.real - z.imag*z.imag + c.real;
205                     z.imag = 2.0*z.real*z.imag + c.imag;
206                     z.real = temp;
207                     lengthsq = z.real*z.real + z.imag*z.imag;
208                     k++;
209
210                 } while (lengthsq < 12 && k < 100); //lengthsq and k are the threshold
211
212                 if (k >= 100) {
213                     result[i * Y_RESN + j] = 1;
214                 }
215             }
216         }
217     }
```

```
33
34  void *cal_func(void *arg) {
35      thread_data *input_data = (thread_data *)arg;
36      int thread_id = input_data -> thread_id;
37      int tot_part = X_RESN / NUM_THREADS;
38      int tot_start = thread_id * tot_part;
39
40      /* Calculate points */
41      int i, j, k;
42      Compl   z, c;
43      float   lengthsq, temp;
44
45      for (i = tot_start; i < tot_start + tot_part; i++){
46          for(j=0; j < Y_RESN; j++) {
47              z.real = z.imag = 0.0;
48              c.real = ((float) j - 400.0)/200.0;               /* scale factors for 800 x 800 win
49              c.imag = ((float) i - 400.0)/200.0;
50              k = 0;
51
52              do  {                                             /* iterate for pixel color */
53
54              temp = z.real*z.real - z.imag*z.imag + c.real;
55              z.imag = 2.0*z.real*z.imag + c.imag;
56              z.real = temp;
57              lengthsq = z.real*z.real+z.imag*z.imag;
58              k++;
59              } while (lengthsq < 12.0 && k < 100);
60              if (k >= 100) {
61                  result[i*Y_RESN + j] = 1;
62              }
63          }
64
65      }
66
67      pthread_exit(NULL);
68      //exit
69  }
70
```

```
220
221     for (tempA = 0; tempA < NUM_THREADS; tempA++){
222         pthread_join(thread[tempA], NULL);
223
224     }
225
226     gettimeofday(&timeEnd, NULL);
227     totalTime = (timeEnd.tv_sec - timeStart.tv_sec) + (double)(timeEnd.tv_usec - timeStart.tv_u
228
229
230     printf("NAME: Hajun Lee\n");
231     printf("STUDENT ID: 117010437\n");
232     printf("ASSIGNMENT 2, PTHREAD VERSION\n");
233     printf("TOTAL TIME IS %lf\n", totalTime);
234
235
236     for (i = 0; i < X_RESN; i++) {
237         for (int j = 0; j < Y_RESN; j++) {
238             if (result[i * Y_RESN + j] == 1) {
239                 XDrawPoint(display, win, gc, j, i);
240                 usleep(1);
241             }
242         }
243     }
244
245     usleep(200000);
246     XFlush (display);
247     sleep (30);
248     /* Program Finished */
249
250     return 0;
251
252 }
```

MPI.cpp Code:

C: > Users > Andy > Desktop > MPI.cpp > ...

```
1
2   /* Sequential Mandelbrot program */
3
4   #include <X11/Xlib.h>
5   #include <X11/Xutil.h>
6   #include <X11/Xos.h>
7   #include <stdio.h>
8   #include <string.h>
9   #include <math.h>
10  #include <mpi.h>
11  #include <time.h>
12  #include <stdlib.h>
13  #include <iostream>
14
15  using namespace std;
16
17
18  #define         X_RESN  800       /* x resolution */
19  #define         Y_RESN  800       /* y resolution */
20  #define         MASTER    0
21
22  typedef struct complextype
23          {
24          float real, imag;
25          } Compl;
26
```

```
30  int main(int argc, char *argv[])
31  {
32      Window          win;                        /* initialization for a window */
33      unsigned
34      int             width, height,              /* window size */
35                      x, y,                       /* window position */
36                      border_width,               /*border width in pixels */
37                      display_width, display_height,  /* size of screen */
38                      screen;                     /* which screen */
39
40      char            *window_name = "Mandelbrot Set", *display_name = NULL;
41      GC              gc;
42      unsigned
43      long            valuemask = 0;
44      XGCValues       values;
45      Display         *display;
46      XSizeHints      size_hints;
47      Pixmap          bitmap;
48      XPoint          points[800];
49      FILE            *fp, *fopen ();
50      char            str[100];
51
52      XSetWindowAttributes attr[1];
53
54     /* Mandlebrot variables */
55      int i, j, k;
56      int a;
57      Compl   z, c;
58      float   lengthsq, temp;
59
60      /* connect to Xserver */
61
62      if (  (display = XOpenDisplay (display_name)) == NULL ) {
63        fprintf (stderr, "drawon: cannot connect to X server %s\n",
64                      XDisplayName (display_name) );
65      exit (-1);
66      }
67
68      /* get screen size */
69
70      screen = DefaultScreen (display);
71      display_width = DisplayWidth (display, screen);
72      display_height = DisplayHeight (display, screen);
73
74      /* set window size */
75
76      width = X_RESN;
77      height = Y_RESN;
78
```

C: > Users > Andy > Desktop > MPI.cpp > ...

```
77          height = Y_RESN;
78
79          /* set window position */
80
81          x = 0;
82          y = 0;
83
84          //init MPI
85          int NumofTask;
86          int task;
87
88          //start time
89          struct timeval timeStart, timeEnd, timeSystemStart;
90          double totalTime = 0, systemRunTime;
91
92          MPI_Init(&argc, &argv);
93          MPI_Comm_size(MPI_COMM_WORLD, &NumofTask);
94          MPI_Comm_rank(MPI_COMM_WORLD, &task);
95
96          int part_width = X_RESN / NumofTask;
97          unsigned long tot_part[part_width * Y_RESN]= {0};
98          unsigned long total[NumofTask * part_width * Y_RESN] = {0};
99
100         if(task == MASTER){
101
102         /* create opaque window */
103
104             border_width = 4;
105             win = XCreateSimpleWindow (display, RootWindow (display, screen),
106                                        x, y, width, height, border_width,
107                                        BlackPixel (display, screen), WhitePixel (display, screen));
108
109             size_hints.flags = USPosition|USSize;
110             size_hints.x = x;
111             size_hints.y = y;
112             size_hints.width = width;
113             size_hints.height = height;
114             size_hints.min_width = 300;
115             size_hints.min_height = 300;
116
117             XSetNormalHints (display, win, &size_hints);
118             XStoreName(display, win, window_name);
119
120             /* create graphics context */
121
122             gc = XCreateGC (display, win, valuemask, &values);
123
```

```
121
122             gc = XCreateGC (display, win, valuemask, &values);
123
124             XSetBackground (display, gc, WhitePixel (display, screen));
125             XSetForeground (display, gc, BlackPixel (display, screen));
126             XSetLineAttributes (display, gc, 1, LineSolid, CapRound, JoinRound);
127
128             attr[0].backing_store = Always;
129             attr[0].backing_planes = 1;
130             attr[0].backing_pixel = BlackPixel(display, screen);
131
132             XChangeWindowAttributes(display, win, CWBackingStore | CWBackingPlanes | CWBackingPixel,
133
134             XMapWindow (display, win);
135             XSync(display, 0);
136
137             gettimeofday(&timeStart, NULL);
138         }
```

```
142
143         /* Calculate and draw points */
144         for(a = 0; a < NumofTask; a++){
145             if(task == a){
146
147                 for(i= part_width * a; i < part_width * (a + 1); i++)
148                 for(j=0; j < Y_RESN; j++) {
149
150                 z.real = z.imag = 0.0;
151                 c.real = ((float) j - 400.0)/200.0;               /* scale factors for 800 x 800 winc
152                 c.imag = ((float) i - 400.0)/200.0;
153                 k = 0;
154
155                 do  {                                             /* iterate for pixel color */
156
157                     temp = z.real*z.real - z.imag*z.imag + c.real;
158                     z.imag = 2.0*z.real*z.imag + c.imag;
159                     z.real = temp;
160                     lengthsq = z.real*z.real+z.imag*z.imag;
161                     k++;
162
163                 } while (lengthsq < 4.0 && k < 100);
164
165                 if (k == 100){
166                     tot_part[i - (part_width * a) + j * part_width] = 1;
167                 }
168
169             }
170
171             MPI_Gather(&tot_part, part_width * Y_RESN, MPI_INT, total, part_width * Y_RESN, MPI_INT,
172
173         }
174         }
175
```

```
175
176     if (task == MASTER){
177         gettimeofday(&timeEnd, NULL);
178         totalTime = (timeEnd.tv_sec - timeStart.tv_sec) + (double)(timeEnd.tv_usec - timeStart.tv_use
179
180         printf("NAME: Hajun Lee\n");
181         printf("STUDENT ID: 117010437\n");
182         printf("ASSIGNMENT 2, MPI VERSION\n");
183         printf("TOTAL TIME IS %lf\n", totalTime);
184
185         int l;
186
187         for(l = 0; l < X_RESN * Y_RESN; l++){
188             if(total[l] == 1){
189                 XDrawPoint (display, win, gc, (l % (part_width * Y_RESN)) / part_width, (l % (part_wi
190                 usleep(1);
191             }
192         }
193
194
195
196
197         XFlush (display);
198         sleep (30);
199         }
200         /* Program Finished */
201
202
203 }
204
```

How to run a code (pthread.cpp):

To run the pthread version,

1. Compile it with below.
    - g++ pthread.cpp -o ./output.out -lX11 -lpthread
2. Run it with below.
    - ./output 4 400 400

How to run a code (MPI.cpp):

To run the MPI version,

1. Compile it with below.
    - mpic++ MPI.cpp -o ./output.out -lX11
2. Run it with below.
    - Mpiexec -n 4 ./output.out 400 400