Assignmen3 Report

CSC4005 **Yu Chen**
Prof. Yeh-Ching Chung 115010124
TA: Hongliang Zhu & Peipei Zhu 11 November 2018

Instruction

This assignment implemented three version of N-body simulation program with graphical output. One is sequential program. The other two are parallel program using multiprocess by MPI library and multithread by pthread library. The complied executable file is in the cluster at direction assignment3 under my user root, named as `Nbodyseq`, `Nbodypth`, and `Nbodympi`. For MPI program, use command `mpiexec -np 4 ./Nbodympi` to run the program with 4 process. The number of process can be changed by the parameter `-n`. The Then re-build by `mpicc`. The threads number used in `Nbodypth` can be changed by change the parameter `nthread`.

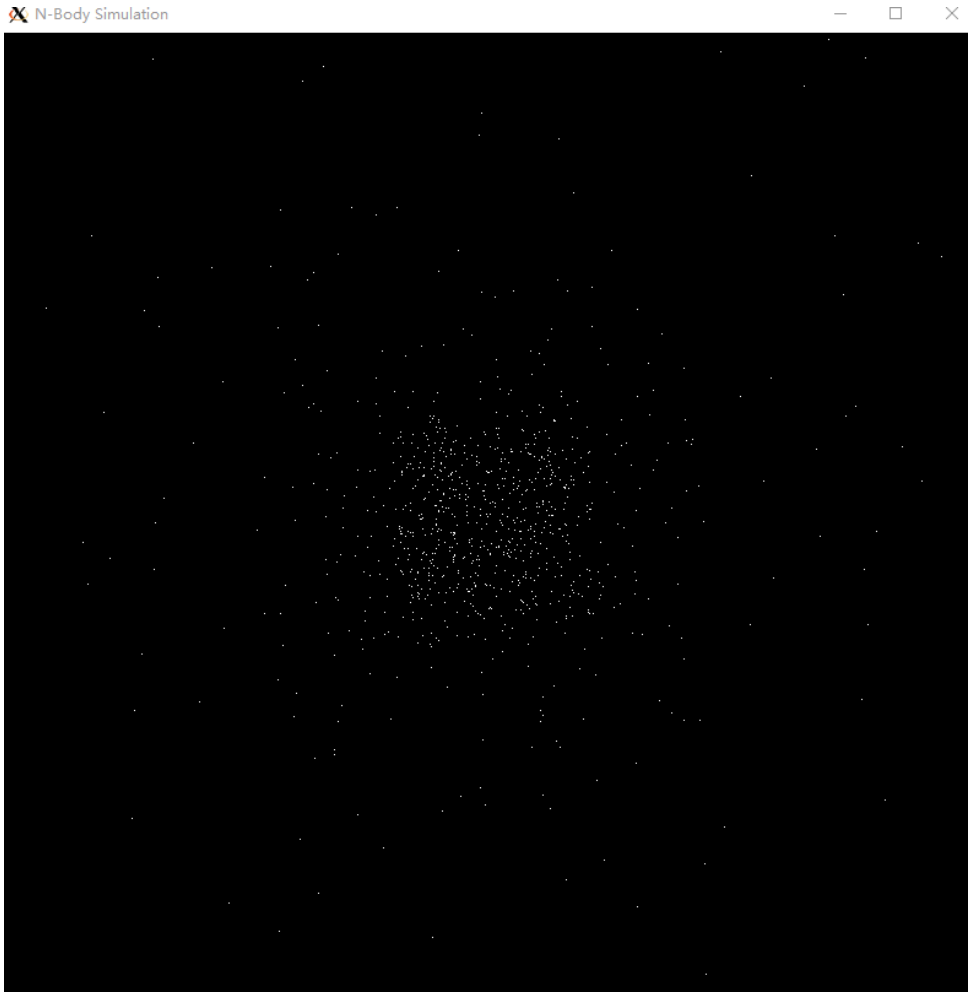

Figure 1: A sample simulation screen shot.

Fig. 1 is a sample output with 1000 bodies simulated with a time step $0.1s/frame$, the program runs 1000 iterations. The random generated bodies has a position constrain in range of x[300,500] and y[300,500]. The mass of bodies ranges in [50, 100], which has a 10^16 scale.

Here because the gravity constant is set as 6.6 which leaves out the 10^-16. The gravity force between each body is the same for these programming tricky. These parameters can be changed by constant value defined at beginning of each source code c file. The running time of the program will be shown in terminal when the calculation of the picture finished.

Design

The structure of sequential N-body simulation program mainly consist by Xlib part and point calculation part. The Xlib part is the same in all three version. It generated a $800x800$ graphical window to show the result image. It sometimes refresh slower than calculation. The code of Xlib refers to the instruction note provided in assignment2.

The force calculation and position update part is a particle-particle algorithm, the most straight forward one.The method is to:

- Accumulate forces by finding the force F(i,j) of particle j on particle i,
- Integrate the equations of motion, and
- Update time counter

The force between each body is $F = Gm_a m_b/r^2$, for calculation in a pixel graph, the force is decomposition into two orthogonal vectors, says Fx and Fy.

MPI design

The calculation job for each body is almost the same. So, the master assigned the job equally to each slaves. The job contains the body structure. The slaves processes update position of bodies in the job in each iteration. The number of body contains in the job is decided by:

$$Job = N_{body}/N_p \tag{1}$$

After all the slaves finished its job, they send back their job in a new body array structure $local\_nBody$ back to master by MPI_Gather. The master process draw the frames on screen frame by frame.

Pthread design

First, initialize the pthread. In this task `pthread_mutex` is used to insure data correctness. (Actually, if no `pthread_mutex` there is segmentation fault.) A variable `startx` is used to save the row number of the next task.

Previous paragraph describe the main stream codes design. The mainly running codes is in the function invoked in each thread to calculation the n^2 forces. The function `forcecal` is the threads function. It firstly check whether all the bodies are finished. If not, get the next body's number. The update of the `startx` is at the condition of all other threads is locked so that the body would not be misused by two different threads. Each thread calculate the force then the master thread update positions of all the bodies after thread join. The the simulation of that frame draw on the figure.

Experiment & Analysis

In this assignment, I haven't test for multiprocess for MPI program because some problem occurred when use processes more than 4. The test uses 4 process or threads, which is representative number for multitasks. Experiment compares three methods of implementation.

	200	400	600	800	1000	2000
sequential	1.44	4.55	9.86	17.26	27.29	104.25
MPI	1.50	2.42	4.01	6.02	8.96	30.58
pthread	3.84	7.61	/	11.87	17.66	62.12

Table 1: 4process/threads, 1000 steps for different number of body.

The table. 1 is the performance test for three versions program. The process/thread used in MPI and pthread is 4. From table we can see that the MPI takes great advantage when body number reaching 2k. The speed up factor is increasing with iteration number, which is 3.41 for 2k iterations. the 600 bodies of pthread lost. The pthread program takes advantage when body number reaches 800. Its speed up factor is 1.68 when body number reaches 2k. The performance should further increase for a lager number.

Experience

Algorithm Improvement

This assignment only implementede the most simplest algorithm, the particle particle model. It has a time complexity of $O(n^2)$, which cannot afford a large number simulation. There are many other trade off solution like tree method. It uses Barnes-Hut algorithm. It improves the performance to a time complexity to $O(NlogN)$.

Display Improvement

All the three version of the program displays the result on a 800x800 windows. All the bodies are considered as particles. There is no color to reflect the weight. Particles only take up one pixel on screen, no radius to reflect the volumn of bodies on 2D screens. I didn't set the boundary of these body.

It is always that the figure printing time is much longer than running time in pthread program of this assignment. I cannot find out the reason for this. The time stamp of pthread is right. Compare with the running time of sequential and MPI program, the running time for pthread is also reasonable. Maybe there is some problem when Xlib using multi-threads or I miss some sets in Xlib.

Boundary

I did not set the boundary reflect at edges. Not because I cannot add two if codes to change the x, y direction. But I think it is a astronomy simulation and the university would not have boundary. Just a small joke.

Track

In the coding processing, I forgot to clean the foreground so that every bodies left its track on screen. Something wrong but interesting. A figure is shown in Fig. 2

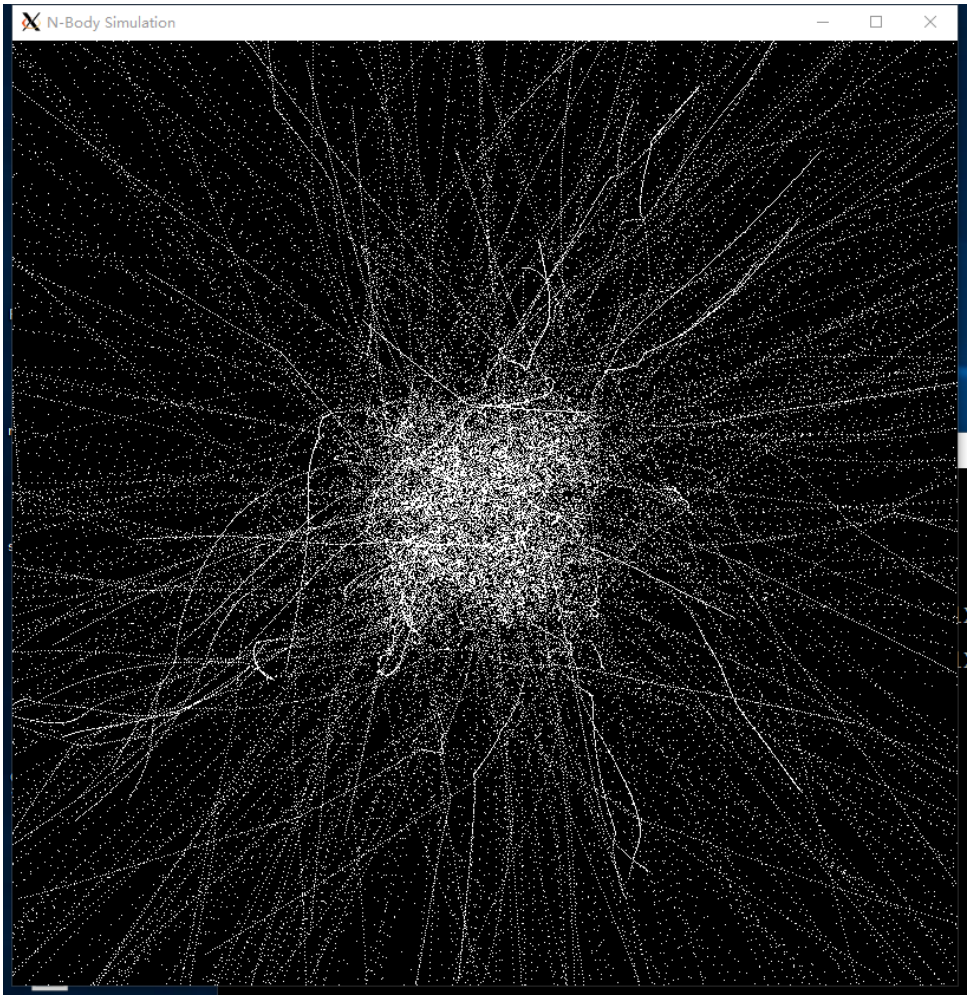

Figure 2: A "Big Bang".

Source Code

The assignment is demo in a server. The source code for the submission time is compressed into a zip file and submitted to the Blackboard. A plaintext version is in below.

The sequential version.

```
#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 <time.h>


const int X_RESN = 800;
const int Y_RESN = 800;

const double G = 6.6;
```

```
15 const int NUMB = 1000;
16 const int MAXX = 500;
17 const int MINX  = 300;
18 const int MAXY = 500;
19 const int MINY  = 300;
20 const int MAXW = 100;
21 const int MINW  = 50;
22 const int ITERATION = 1000;
23 const double T = 0.1;
24
25 struct Body{
26 /* position */
27 double x,y;
28 /* velocity */
29 double vx,vy;
30 /* weight */
31 double w;
32 };
33
34
35 int main () {
36 Window          win;        /* initialization for a window */
37 unsigned
38 int             width, height,              /* window size */
39 x, y,                   /* window position */
40 border_width,    /* border width in pixels */
41 display_width,
42 display_height,          /* size of screen */
43 screen;                   /* which screen */
44
45 char            *window_name = "N-Body Simulation",
46 *display_name = NULL;
47 GC              gc;
48 unsigned
49 long            valuemask = 0;
50 XGCValues       values;
51 Display         *display;
52 XSizeHints      size_hints;
53 Pixmap          bitmap;
54 XPoint          points[800];
55
56 XInitThreads();
57
58 XSetWindowAttributes attr[1];
59
60 if ((display = XOpenDisplay(display_name)) == NULL) {
61 fprintf(stderr, "drawon: cannot connect to X server %s\n",
62 XDisplayName(display_name));
63 //exit(-1);
64 }
65
66 /* get screen size */
67 screen = DefaultScreen(display);
68 display_width = DisplayWidth(display, screen);
69 display_height = DisplayHeight(display, screen);
```

```
70
71  /* set window size */
72  width = X_RESN;
73  height = Y_RESN;
74
75  /* set window position */
76  x = 0;
77  y = 0;
78
79  /* create opaque window */
80  border_width = 4;
81  win = XCreateSimpleWindow (display,
82  RootWindow(display, screen),
83  x, y, width, height, border_width,
84  BlackPixel(display, screen),
85  WhitePixel(display, screen));
86
87  size_hints.flags = USPosition|USSize;
88  size_hints.x = x;
89  size_hints.y = y;
90  size_hints.width = width;
91  size_hints.height = height;
92  size_hints.min_width = 300;
93  size_hints.min_height = 300;
94
95  XSetNormalHints(display, win, &size_hints);
96  XStoreName(display, win, window_name);
97
98  /* create graphics context */
99  gc = XCreateGC(display, win, valuemask, &values);
100
101 XSetBackground(display, gc, WhitePixel(display, screen));
102 XSetForeground(display, gc, BlackPixel(display, screen));
103 XSetLineAttributes(display, gc, 1, LineSolid, CapRound,
104 JoinRound);
105
106 attr[0].backing_store = Always;
107 attr[0].backing_planes = 1;
108 attr[0].backing_pixel = BlackPixel(display, screen);
109
110 XChangeWindowAttributes(display, win,
111 CWBackingStore | CWBackingPlanes | CWBackingPixel, attr);
112
113 XMapWindow(display, win);
114 XSync(display, 0);
115
116 int i,j,k;
117 struct Body Nbody[NUMB];
118 double vx[NUMB];
119 double vy[NUMB];
120 double deltaX, deltaY;
121 double distance;
122 double F;
123 int scr = DefaultScreen(display);
124 int pm = XCreatePixmap(display,win,X_RESN,Y_RESN,DefaultDepth(display,scr));
```

```
125
126 /*Initialization with random weight and position with 0 inital speed*/
127 srand(time(NULL));
128 for(i=0;i<NUMB;i++)
129 {
130 Nbody[i].x = rand() % (MAXX-MINX)+MINX;
131 Nbody[i].y = rand() % (MAXY-MINY)+MINY;
132 Nbody[i].vx = 0;
133 Nbody[i].vy = 0;
134 Nbody[i].w = rand() % (MAXW-MINW)+MINW;
135 }
136
137 /*Iterations with NUMB square force calculation*/
138 for(k=0;k<ITERATION;k++)
139 {
140 XSetForeground(display,gc,0);
141 XFillRectangle(display,pm,gc,0,0,X_RESN,Y_RESN);
142
143 /* Update speed for each point.*/
144 for(i=0;i<NUMB;i++)
145 {
146 for(j=0;j<NUMB;j++)
147 {
148 if (j==i) continue;
149 deltaX = Nbody[j].x - Nbody[i].x;
150 deltaY = Nbody[j].y - Nbody[i].y;
151 distance = sqrt((deltaX * deltaX) + (deltaY * deltaY));
152 if(distance == 0) continue;
153 F = G * Nbody[j].w / (distance*distance);
154 vx[i] = vx[i] + T * F * deltaX/distance;
155 vy[i] = vy[i] + T * F * deltaY/distance;
156 }
157 }
158 /* update position for each point.*/
159 for(i=0;i<NUMB;i++)
160 {
161 Nbody[i].x = Nbody[i].x + vx[i] * T;
162 Nbody[i].y = Nbody[i].y + vy[i] * T;
163 Nbody[i].vx = vx[i];
164 Nbody[i].vy = vy[i];
165 }
166 /*Draw the points. */
167
168 XSetForeground(display, gc, WhitePixel(display,scr));
169 for(i=0;i<NUMB;i++)
170 {
171 XDrawPoint(display, pm, gc, Nbody[i].y, Nbody[i].x);
172 }
173 //XClearArea(display, win, 0, 0, X_RESN, Y_RESN, 0);
174 XCopyArea(display,pm,win,gc,0,0,X_RESN,Y_RESN,0,0);
175 //XFlush(display);
176 //sleep(0.1);
177 }
178 XFreePixmap(display,pm);
179 XCloseDisplay(display);
```

```
return 0;
}
```

The pthread version.

```
#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 <pthread.h>
#include <time.h>


const int X_RESN = 800;
const int Y_RESN = 800;

const double G = 6.6;
const int NUMB = 500;
const int MAXX = 500;
const int MINX  = 300;
const int MAXY = 500;
const int MINY  = 300;
const int MAXW = 100;
const int MINW  = 50;
const int ITERATION = 100;
const double T = 0.1;

struct Body{
/* position */
double x,y;
/* velocity */
double vx,vy;
/* weight */
double w;
}Nbody[505];


int i,j,k;
double vx[505];
double vy[505];


Window win;
GC gc;
Display *display;

pthread_mutex_t mutex;
int startx;

void forcecal(void* para)
{
startx = (int)para;

```

```
52 double deltaX, deltaY;
53 double distance;
54 double F;
55 // I don't know why but here should minors
56 //some value to avoid segmentation fault.
57 while(startx < NUMB - 8){
58 /* Update speed for each point.*/
59 for(j=0;j<NUMB;j++)
60 {
61 if (j==startx) continue;
62 deltaX = Nbody[j].x - Nbody[startx].x;
63 deltaY = Nbody[j].y - Nbody[startx].y;
64 distance = sqrt((deltaX * deltaX) + (deltaY * deltaY));
65 //printf("%d\n",Nbody[startx].x);
66 if(distance == 0) continue;
67 F = G * Nbody[j].w / (distance*distance);
68 if(distance > 5)
69 {
70 vx[startx] = vx[startx] + T * F * deltaX/distance;
71 vy[startx] = vy[startx] + T * F * deltaY/distance;
72 }
73 }
74 /* retrieve and update the next job */
75 pthread_mutex_lock(&mutex);
76 startx++;
77 pthread_mutex_unlock(&mutex);
78 }
79 }
80
81 int main (void)
82 {
83 XInitThreads();
84
85 Window          win;                /* initialization for a window */
86 unsigned
87 int             width, height,                  /* window size */
88 x, y,                           /* window position */
89 border_width,                   /*border width in pixels */
90 display_width, display_height,  /* size of screen */
91 screen;                         /* which screen */
92
93 char            *window_name = "Mandelbrot Set", *display_name = NULL;
94 //GC              gc;
95 unsigned
96 long            valuemask = 0;
97 XGCValues       values;
98 //Display         *display;
99 XSizeHints      size_hints;
100
101 XSetWindowAttributes attr[1];
102
103 /* connect to Xserver */
104
105 if (  (display = XOpenDisplay (display_name)) == NULL ) {
106 fprintf (stderr, "drawon: cannot connect to X server %s\n",
```

```
XDisplayName (display_name) );
}

/* get screen size */

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

/* set window size */

width = X_RESN;
height = Y_RESN;

/* 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);

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 | CWBackingPixel, attr);

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

```

```
int scr = DefaultScreen(display);
int pm = XCreatePixmap(display,win,X_RESN,Y_RESN,DefaultDepth(display,scr));

/*Initialization with random weight and position with 0 inital speed*/
srand(time(NULL));
for(i=0;i<NUMB;i++)
{
Nbody[i].x = rand() % (MAXX-MINX)+MINX;
Nbody[i].y = rand() % (MAXY-MINY)+MINY;
Nbody[i].vx = 0;
Nbody[i].vy = 0;
Nbody[i].w = rand() % (MAXW-MINW)+MINW;
//printf("%f\n", Nbody[i].w);
}
printf("The pthread begins\n");

int nthread, n;
nthread = 4;
n = 0;

pthread_t tid[nthread];
pthread_mutex_init(&mutex, NULL);

for(k=0;k<ITERATION;k++)
{
printf("The iter %d begins\n", k);
XSetForeground(display,gc,0);
XFillRectangle(display,pm,gc,0,0,X_RESN,Y_RESN);

for (i = 0; i < nthread; i++)
{
int startx = i;
pthread_create(&tid[i], NULL, &forcecal, (void*)(startx));
}

for (i = 0; i < nthread; i++)
{
pthread_join(tid[i], NULL);
}
/* update position for each point.*/

for(i=0;i<NUMB;i++)
{
//printf("%f\n", vx[i]);
Nbody[i].x = Nbody[i].x + vx[i] * T;
Nbody[i].y = Nbody[i].y + vy[i] * T;
Nbody[i].vx = vx[i];
Nbody[i].vy = vy[i];
}
/*Draw the points. */

XSetForeground(display, gc, WhitePixel(display,scr));

for(i=0;i<NUMB - 8;i++)
{
```

```
217 if(Nbody[i].y <= 800 && Nbody[i].x <= 800)
218 XDrawPoint(display, pm, gc, (int)Nbody[i].y, (int)Nbody[i].x);
219 }
220 XCopyArea(display,pm,win,gc,0,0,X_RESN,Y_RESN,0,0);
221 XFlush(display);
222 //sleep(0.1);
223 }
224 XFreePixmap(display,pm);
225 XCloseDisplay(display);
226 //sleep(10);
227 return 0;
228 }
```

The mpi version.

```
1  #include <X11/Xlib.h>
2  #include <X11/Xutil.h>
3  #include <X11/Xos.h>
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <string.h>
7  #include <math.h>
8  #include <time.h>
9  #include "mpi.h"
10
11
12 const int X_RESN = 800;
13 const int Y_RESN = 800;
14
15 const double G = 6.6;
16 const int NUMB = 100;
17 const int MAXX = 500;
18 const int MINX  = 300;
19 const int MAXY = 500;
20 const int MINY  = 300;
21 const int MAXW = 100;
22 const int MINW  = 50;
23 const int ITERATION = 1000;
24 const double T = 0.1;
25
26 struct Body{
27 /* position */
28 double x,y;
29 /* velocity */
30 double vx,vy;
31 /* weight */
32 double w;
33 };
34
35
36
37 int main (int argc, char *argv[])
38 {
39 XInitThreads();
40
41 Window win;
```

```
42 GC gc;
43 Display *display;
44
45 int i, j, k;
46 int size, rank;
47
48 double startTime, endTime;
49 startTime = MPI_Wtime();
50
51 MPI_Init(&argc, &argv);
52 MPI_Comm_size(MPI_COMM_WORLD, &size);
53 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
54
55 MPI_Datatype MPIBody;
56 MPI_Type_contiguous(5, MPI_DOUBLE, &MPIBody);
57 MPI_Type_commit(&MPIBody);
58
59 MPI_Status status;
60 int job = NUMB /size;
61
62 double vx[NUMB];
63 double vy[NUMB];
64 double deltaX, deltaY;
65 double distance;
66 double F;
67
68 struct Body *local_nBody;
69 local_nBody = (struct Body*)malloc(NUMB * sizeof(struct Body));
70 struct Body* Nbody = (struct Body*)malloc(NUMB * sizeof(struct Body));
71
72 if (rank == 0)
73 {
74
75 Window          win;              /* initialization for a window */
76 unsigned
77 int             width, height,                  /* window size */
78 x, y,                            /* window position */
79 border_width,                    /*border width in pixels */
80 display_width, display_height,   /* size of screen */
81 screen;                          /* which screen */
82
83 char            *window_name = "N-body Simulation", *display_name = NULL;
84 GC              gc;
85 unsigned
86 long            valuemask = 0;
87 XGCValues       values;
88 Display         *display;
89 XSizeHints      size_hints;
90 Pixmap          bitmap;
91 XPoint          points[800];
92 FILE            *fp, *fopen ();
93 char            str[100];
94
95 XSetWindowAttributes attr[1];
96
```

```
/* Mandlebrot variables */


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

/* get screen size */

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

/* set window size */

width = X_RESN;
height = Y_RESN;

/* 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);

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);
```

```
152
153 XChangeWindowAttributes(display, win,
154 CWBackingStore | CWBackingPlanes | CWBackingPixel, attr);
155
156 XMapWindow (display, win);
157 XSync(display, 0);
158
159 int scr = DefaultScreen(display);
160 int pm = XCreatePixmap(display,win,X_RESN,Y_RESN,DefaultDepth(display,scr));
161
162 /*Initialization with random weight and position with 0 inital speed*/
163 srand(time(NULL));
164 for(i=0;i<NUMB;i++)
165 {
166 Nbody[i].x = rand() % (MAXX-MINX) + MINX;
167 Nbody[i].y = rand() % (MAXY-MINY) + MINY;
168 Nbody[i].vx = 0;
169 Nbody[i].vy = 0;
170 Nbody[i].w = rand()% (MAXW-MINW) + MINW;
171 }
172
173 for (i = 1; i < size; i++)
174 MPI_Send(Nbody, NUMB, MPIBody, i, i, MPI_COMM_WORLD);
175 }
176 else
177 {
178 MPI_Recv(Nbody, NUMB, MPIBody, 0, rank, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
179 }
180
181 for(k=0;k<ITERATION;k++)
182 {
183 int startPoint = job * rank;
184
185 /* Update speed for each point in job.*/
186 for(i=startPoint;i<job + startPoint;i++)
187 {
188 for(j=0;j<NUMB;j++)
189 {
190 if (j==i) continue;
191 deltaX = Nbody[j].x - Nbody[i].x;
192 deltaY = Nbody[j].y - Nbody[i].y;
193 distance = sqrt((deltaX * deltaX) + (deltaY * deltaY));
194 if(distance == 0) continue;
195 F = G * Nbody[j].w / (distance*distance);
196 if(distance > 5)
197 {
198 vx[i] = vx[i] + T * F * deltaX/distance;
199 vy[i] = vy[i] + T * F * deltaY/distance;
200 }
201 }
202 }
203 MPI_Barrier(MPI_COMM_WORLD);
204
205 /* update position for each point in job.*/
206 for(i=startPoint;i<job + startPoint;i++)
```

```
{
Nbody[i].x = Nbody[i].x + vx[i] * T;
Nbody[i].y = Nbody[i].y + vy[i] * T;
Nbody[i].vx = vx[i];
Nbody[i].vy = vy[i];
}

for(i=0;i<job;i++)
{
local_nBody[i].x = Nbody[startPoint+i].x;
local_nBody[i].y = Nbody[startPoint+i].y;
local_nBody[i].vy = Nbody[startPoint+i].vy;
local_nBody[i].vx = Nbody[startPoint+i].vx;
}

MPI_Gather(local_nBody, job, MPIBody, Nbody, job, MPIBody, 0, MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
/*Draw the points. */
if(rank == 0)
{
//              int scr = DefaultScreen(display);
//              int pm = XCreatePixmap(display,win,X_RESN,Y_RESN
//                                  ,DefaultDepth(display,scr));
//              XClearWindow(display, win);
//              XSetForeground(display, gc, WhitePixel(display,scr));
//              for(i=0;i<NUMB;i++)
//              {
//      XDrawPoint(display, pm, gc, Nbody[i].y, Nbody[i].x);
//              }
// XCopyArea(display,pm,win,gc,0,0,X_RESN,Y_RESN,0,0);

for (j = 1; j < size; j++)
MPI_Send(Nbody, NUMB, MPIBody, j, j, MPI_COMM_WORLD);
}
else
{
MPI_Recv(Nbody, NUMB, MPIBody, 0, rank, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
}

if(rank == 0)
{
endTime = MPI_Wtime();
double totaltime = endTime - startTime;
printf("Run time is: %fs\n", totaltime);
}

MPI_Finalize();

return 0;
}
```

References

[1] The Latex Template used for assignment is cite from overleaf. *https* : *//www.overleaf.com/latex/templates/ece* − 100 − *template/pjrrfybfggqt*

[2] The source code for sequential program is spread by the instructor from the site. *http* : *//www.cs.nthu.edu.tw/ ychung/homework/para*$_p$*rogramming/mandelbrot.htm*