实验三

姓名：吴燕晶
学号：SA17011125

（一）实验题目：利用 MPI 解决 N 体问题

① 实验描述：N 体问题是指找出已知初始位置、速度和质量的多个物体在经典力学情况下的后续运动。在本次实验中，你需要模拟 N 个物体在二维空间中的运动情况。通过计算每两个物体之间的相互作用力，可以确定下一个时间周期内的物体位置。在本次实验中，N 个小球在均匀分布在一个正方形的二维空间中，小球在运动时没有范围限制。每个小球间会且只会受到其他小球的引力作用。为了方便起见，在计算作用力时，两个小球间的距离不会低于其半径之和，在其他的地方小球位置的移动不会受到其他小球的影响（即不会发生碰撞，挡住等情况）。你需要计算模拟一定时间后小球的分布情况，并通过 MPI 并行化计算过程。

② 实验要求：

1. 有关参数要求如下：
   引力常数数值取 6.67*10^-11
   小球重量都为 10000kg
   小球半径都为 1cm
   小球间的初始间隔为 1cm，例：N=36 时，则初始的正方形区域为 5cm*5cm
   小球初速为 0.
2. 对于时间间隔，公式如下
   delta_t=1/timestep
   其中，timestep 表示在 1s 内程序迭代的次数，小球每隔 delta_t 时间更新作用力，速度，位置信息。结果中程序总的迭代次数=timestep*模拟过程经历的时间，你可以根据你的硬件环境自己设置这些数值，理论上来说，时间间隔越小，模拟的真实度越高。
3. 你的程序中，应当实现下面三个函数
   compute_force()：计算每个小球受到的作用力
   compute_velocities(): 计算每个小球的速度
   compute_positions(): 计算每个小球的位置
   典型的程序中，这三个函数应该是依次调用的关系。

   如果你的方法中不实现这三个函数，应当在报告中明确说明，并解释你的方法为什么不需要上述函数的实现。
4. 报告中需要有 N=64 和 N=256 的情况下通过调整并行度计算的程序执行时间和加速比。

（二）实验环境

① 虚拟机：VMware
② 操作系统：Ubuntu16.04
③ 内存：4G

④ 处理器：4

（三）算法设计与分析

首先我们需要定义小球的数据结构。一个小球它含有六个特征：x 方向的速度，y 方向的速度，x 方向的位置，y 方向的位置，x 方向的加速度，y 方向的加速度。算法的可以分成五个函数分别实现。

1．Compute_force

该函数用来计算两个小球之间的引力作用的大小，并且将力的大小转换成加速度的大小。

对于两个小球 Sq 和 Sk，要计算这两个小球之间的引力：

F = G*Mq*Mk/(Sq-Sk)^2

根据 F=ma，可以计算加速度。因为小球的加速度是由方向的，所以我们可以根据下面的式子计算加速度，对于小球 Sq 有：

a = G*Mk*(Sq-Sk)/|Sq- Sk|^3

计算出了小球 Sq 的加速度，对于小球 Sk 只需要取反就可以。

将得到的加速度乘以 delta_t 就是小球在一段时间内的加速度。

该函数会返回两个小球 x，y 方向的加速度。

2．Compute_acce

该函数将会根据传入的小球 x 方向的加速度和 y 方向的加速度，对小球自身 x，y 方向的加速度进行设置。

3．Compute_velocities

该函数用来计算小球当前的速度，小球当前的速度等于小球原来的速度加上当前的加速度。

4．Compute_positions

该函数用来计算小球当前的位置，小球当前的位置等于等于小球原来的位置加上当前的速度乘以时间间隔

5．Compute_allforce

计算处于同一个 block 内的小球所受到的其他小球的引力的大小，进一步得到一个 block 内的小球的加速度的大小。这个函数主要利用了一个两重循环的遍历算法，并利用 compute_force 函数计算两个小球的加速度改变量。

6．Balls

（1）按照进程数的大小，将所有的小球划分成大小相等的块

（2）定义 force 变量，用来记录当前想要传递给下一个块的内容。force 记录的是经历过的块内的小球对于目标块内的小球的影响。它的数据结构和 ball 的数据结构相同。

（3）初始化小球的加速度，速度和位置；同时初始化 force 变量。

（4）定义一个 MPI 数据结构 mpiball，该数据结构是用来定义 force 变量的 MPI 数据结构，而 force 变量的数据结构和 ball 的数据结构相同。

（5）一个循环，表示进行了多少次迭代，这里设置的是 10000。在每一个循环内：

①计算当前块内所有小球之间的相互作用力对各自加速度的影响，并将其保存到 force 变量中

②将 force 变量发送给下一个进程，所谓的下一个进程指的是进程号为(myrank+1)%size 的进程，设置 tag 标签为 myrank+i*size

③当前进程接受前一个进程发送来的 tag 标签不等于 myrank+i*size 的消息，其会将内容保存到 force 变量中，并且调用 compute_allforce 改变当前的 force 的内容，然后将现在的 force

传递给下一个进程。这里的前一个进程指的是进程号为(myrank-1+size)%size 的进程。

④当前进程接收来自于前一个进程发送来的 tag 标签为 myrank+i*size 的消息。通过以上过程我们可以知道此时的 force 变量记录了当前进程内的小球受到其他所有小球的影响。

⑤更新当前块内所有小球的加速度，速度和位置

⑥更新 force 变量的值

（四）核心代码

1．初始化小球的质量，G 值， delta_t

```
//define the weight of each ball
#define m 10000

//define the value of G
#define g 6.67*pow(10, -11)

//defime the delta time
#define delta_t 0.00001
```

2．定义小球的数据结构

```
//define the structure of ball
typedef struct myball{
  double vx; // the speed of x
  double vy; // the speed of y
  double px; // the position of x
  double py; // the position of y
  double ax; // the accelation of x
  double ay; // the accelation of y
}ball;
```

3．Compute_force 函数

```
// compute the force
// return a array which the length is 4 and it record the acc between to point
void compute_force(ball *b1, ball *b2, double acc[]){
  double diff_x, diff_y;
  diff_x = b1->px - b2->px;
  diff_y = b1->py - b2->py;
  double acc_x, acc_y;
  double distance;
  distance = sqrt(diff_x*diff_x + diff_y*diff_y);
  double temp_force = g*m/pow(distance, 3);
  acc_x = diff_x * temp_force;
  acc_y = diff_y * temp_force;
  acc[0] = acc_x * delta_t;
  acc[1] = acc_y * delta_t;
  acc[2] = (-acc_x) * delta_t;
  acc[3] = (-acc_y) * delta_t;
}
```

4．Compute_acce 函数

```
// compute the accelerate
void compute_acce(ball *b1, double acc_x, double acc_y){
  b1->ax = acc_x;
  b1->ay = acc_y;
}
```

5. Compute_velocities

```
// compute the speed
void compute_velocities(ball *b1){
  b1->vx += b1->ax;
  b1->vy += b1->ay;
}
```

6. Compute_positions

```
// compute the position
void compute_positions(ball *b1){
  double vx = b1->vx;
  double vy = b1->vy;
  b1->px += vx*delta_t;
  b1->py += vy*delta_t;
}
```

7. Compute_allforce

```
void compute_allforce(ball balls[], ball force[], int n){
  double acc[4];
  for(int i=0; i<n; i++)
    for(int j=i+1; j<n; j++){
      compute_force(&balls[i], &balls[j], acc);
      force[i].ax += acc[0];
      force[i].ay += acc[1];
      force[j].ax += acc[2];
      force[j].ay += acc[3];
    }

}
```

8. Balls

（1）初始化

```
  double start, end, diff;
  int size, myrank;
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
 // printf("myrank is %d\n", myrank);
 // printf("the size is %d\n", size);
  int count;
  count = n/size;
  if(myrank == 0){
    start = MPI_Wtime();
  }
```

（2）定义 force 变量

```
ball balls[count+1];
ball force[count+1];
```

（3）初始化小球的速度，加速度和位置，同时初始化 force 变量

```
for(int i=0; i<row; i++)
  for(int j = 0; j<col; j++){
    balls[i*col+j].vx = 0;
    balls[i*col+j].vy = 0;
    balls[i*col+j].px = i;
    balls[i*col+j].py = j+myrank*count;
    balls[i*col+j].ax = 0;
    balls[i*col+j].ay = 0;
  }

for(int i=0; i<count; i++){
  force[i].vx = 0;
  force[i].vy = 0;
  force[i].px = balls[i].px;
  force[i].py = balls[i].py;
  force[i].ax = 0;
  force[i].ay = 0;
}
```

（4）定义 mpiball 结构

```
  // define the send data type
  MPI_Datatype mpiball;
  MPI_Datatype oldtype[6] = {MPI_DOUBLE, MPI_DOUBLE, MPI_DOUBLE, MPI_DOUBLE, MPI
_DOUBLE, MPI_DOUBLE};
  int blockcount[6]= {1,1,1,1,1,1};
  MPI_Aint offset[5];
  MPI_Status stat;
  //initialize the address
  MPI_Address(&(force[0].vx), &offset[0]);
  MPI_Address(&(force[0].vy), &offset[1]);
  MPI_Address(&(force[0].px), &offset[2]);
  MPI_Address(&(force[0].py), &offset[3]);
  MPI_Address(&(force[0].ax), &offset[4]);
  MPI_Address(&(force[0].ay), &offset[5]);
  for(int i=5; i>=0; i--){
    offset[i] -= offset[0];
  }
  MPI_Type_struct(6, blockcount, offset, oldtype, &mpiball);
  MPI_Type_commit(&mpiball);
```

（5）主循环

```
  int k = 1;
  while(k<10000){

    // calculate current block all force and send the force to next
    compute_allforce(balls, force, count);
   // printf("%d finish calculate the force\n", myrank);
    MPI_Send(force, count, mpiball, (myrank+1)%size, myrank+k*size, MPI_COMM_WOR
LD);
   // printf("%d finish the first send\n", myrank);
    for(int i=0; i<size; i++){
      if(i!=myrank){
        MPI_Recv(force, count, mpiball, (myrank-1+size)%size, i+k*size, MPI_COMM
_WORLD, &stat);
     //   printf("%d finish the second recv\n", myrank);
        compute_allforce(balls, force, count);
      //  printf("%d finish calculate the force 2\n", myrank);
        MPI_Send(force, count, mpiball, (myrank+1)%size, i+k*size, MPI_COMM_WORL
D);
       // printf("%d finish the second send\n", myrank);
      }
    }
   // printf("finish the circling");
    MPI_Recv(force, count, mpiball, (myrank-1+size)%size, myrank+k*size, MPI_COM
M_WORLD, &stat);
   // printf("finish the first receive");
    for(int i=0; i<count; i++){
      compute_acce(&balls[i], force[i].ax, force[i].ay);
      compute_velocities(&balls[i]);
      compute_positions(&balls[i]);
    }
 //   printf("%d finish the %d time", myrank, k);
    for(int i=0; i<count; i++){
                                                          163,1         68%
```

```
  // printf("finish the first receive");
    for(int i=0; i<count; i++){
      compute_acce(&balls[i], force[i].ax, force[i].ay);
      compute_velocities(&balls[i]);
      compute_positions(&balls[i]);
    }

    for(int i=0; i<count; i++){
      force[i].vx = 0;
      force[i].vy = 0;
      force[i].px = balls[i].px;
      force[i].py = balls[i].py;
      force[i].ax = 0;
      force[i].ay = 0;
    }
    k++;
  }
```

（6）串行程序

```
void serial_time(int n){
  double start, end, diff;
  start = MPI_Wtime();
  ball balls[n];
  int row = (int)sqrt(n);
  for(int i=0; i<row; i++){
    for(int j =0; j<row; j++){
      balls[i+j*row].vx = 0;
      balls[i+j*row].vy = 0;
      balls[i+j*row].px = i;
      balls[i+j*row].py = j;
      balls[i+j*row].ax = 0;
      balls[i+j*row].ay = 0;
    }
  }
  printf("finish the initial");
  int k = 1;
  while(k<10000){
    compute_allforce(balls, balls, n);
    for(int i=0; i<n; i++){
      compute_velocities(&balls[i]);
      compute_positions(&balls[i]);
    }
    k++;
  }
  end = MPI_Wtime();
  diff = end - start;
  printf("the serial time is %f\n", diff);
}
```

（五）实验结果

用 MPI 实现

运行时间

规模\进程数	1	2	3	4
N=64 Te=10000 Delta_t=0.00001	2.075067	1.165778	0.591496	1.660255
N=256 Te=10000 Delta_t=0.00001	34.314328	19.225123	9.486434	29.493004

加速比

规模\进程数	1	2	3	4
N=64 Te=10000	1	1.779985	3.508169	1.249849

Delta_t=0.00001				
N=256 Te=10000 Delta_t=0.00001	**1**	**1.784869**	**3.617200**	**1.163473**

（六）分析与总结

1. 从运行时间上可以明显的看出，N 值增大，运行时间增加。从进程 1 到 4 来看，并行化程序会使运行时间减少。进程数为 1 到 3，可以明显的看出，运行时间随着进程数的增加而减少。而当进程数增加到 4 的时候可以看出运行时间相比于 3 个进程又增加了。运行时间在进程数为 3 的时候达到最小值。

2. 从加速比上可以明显的看出，N 值增大，运行时间增加。从进程 1 到进程 4 来看，并行化程序会使程序加速。进程数为 1 到 3，可以明显的看出，加速比逐渐的升高。但是当进程数达到 4 的时候，加速比下降了，加速的效果不明显。加速比在进程数为 3 的时候达到最大值。

3. 运行时间在进程数为 3 的时候达到最小，加速比也在进程数为 3 的时候达到最大，这说明并行程序并不是进程数越多越好。在一定范围内增加进程数会使运行时间增加，但如果进程数过多的话，通信过于频繁，此时的通信开销过大，并行化计算所减少的时间不足以抵消通信开销，甚至会使计算时间比串行的还慢。总而言之，我们在并行化程序的时候要选择合适的进程数，这样才能够让并行化的效果达到最佳。