实验三

实验题目

利用 MPI 解决 N 体问题

实验环境

操作系统：windows10 1803 pro IDE：vs2017
编译配置：debug x64
硬件配置：2.2GHZ Intel i5 CPU + 16GB RAM

实验内容

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。

实验要求

1. 有关参数要求如下：
   a) 引力常数数值取 6.67*10^-11
   b) 小球重量都为 10000kg
   c) 小球半径都为 1cm
   d) 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
   e) 小球初速为 0.
   f) 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。

2. 你的程序中，应当实现下面三个函数
   a) compute_force()：计算每个小球受到的作用力
   b) compute_velocities(): 计算每个小球的速度
   c) 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。

3. 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。

算法设计与分析

1. 初始化质量 mass，间隔 gap，初始速度，位置等常量并广播

```
void Gen_init_cond(double masses[], vect_t pos[], vect_t loc_vel[], int n, int loc_n)// 
{
    const double mass = 10000, gap = 0.01, speed = 0;
    if (my_rank == 0)
    {
        int ny = ceil(sqrt(n));
        double y = 0.0;
        for (int i = 0; i < n; i++)
        {
            masses[i] = mass;
            pos[i][X] = i * gap;
            if ((i + 1) % ny == 0)
                ++y;
            pos[i][Y] = y;
            vel[i][X] = 0.0;
            vel[i][Y] = 0.0;
        }
    }
    // 同步质量，位置信息，分发速度信息
    MPI_Bcast(masses, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(pos, n, vect_mpi_t, 0, MPI_COMM_WORLD);
    MPI_Scatter(vel, loc_n, vect_mpi_t, loc_vel, loc_n, vect_mpi_t, 0, MPI_COMM_WORLD);
}
```

2. 计算每个球受到的引力

```
void Compute_force(int loc_part, double masses[], vect_t loc_forces[], vect_t pos[], int n, int loc_n)
{
    const int part = my_rank * loc_n + loc_part;
    int k;
    vect_t f_part_k;
    double len, fact;
    for (loc_forces[loc_part][X] = loc_forces[loc_part][Y] = 0.0, k = 0; k < n; k++)
    {
        if (k != part)
        {
            f_part_k[X] = pos[part][X] - pos[k][X];
            f_part_k[Y] = pos[part][Y] - pos[k][Y];
            len = sqrt(f_part_k[X] * f_part_k[X] + f_part_k[Y] * f_part_k[Y]);
            fact = -G * masses[part] * masses[k] / (len * len * len);
            f_part_k[X] *= fact;
            f_part_k[Y] *= fact;
            loc_forces[loc_part][X] += f_part_k[X];
            loc_forces[loc_part][Y] += f_part_k[Y];
        }
    }
}
```

3 更新每个球的速度与位置

```
void Update_part(int loc_part, double masses[], vect_t loc_forces[], vect_t loc_pos[],
    vect_t loc_vel[], int n, int loc_n, double delta_t)// 更新位置
{
    const int part = my_rank * loc_n + loc_part;
    const double fact = delta_t / masses[part];
    loc_pos[loc_part][X] += delta_t * loc_vel[loc_part][X];
    loc_pos[loc_part][Y] += delta_t * loc_vel[loc_part][Y];
    loc_vel[loc_part][X] += fact * loc_forces[loc_part][X];
    loc_vel[loc_part][Y] += fact * loc_forces[loc_part][Y];
}
```

4. 核心循环

```
177     // 开始计算并计时
178     if (my_rank == 0)
179         start = MPI_Wtime();
180     for (step = 1; step <= n_steps; step++)
181     {
182         // 计算每小球受力，更新小球状态，然后同步小球位置
183         for (loc_part = 0; loc_part < loc_n; Compute_force(loc_part++, masses, loc_forces, pos, n, loc_n));
184         for (loc_part = 0; loc_part < loc_n; Update_part(loc_part++, masses, loc_forces, loc_pos, loc_vel, n, loc_n, delta_t));
185         MPI_Allgather(MPI_IN_PLACE, loc_n, vect_mpi_t, pos, loc_n, vect_mpi_t, MPI_COMM_WORLD);
186         if (step % output_freq == 0)
187             Output_state(step * delta_t, masses, pos, loc_vel, n, loc_n);
188     }
189     // 打印计时
190     if (my_rank == 0)
191     {
192         finish = MPI_Wtime();
193         printf("Elapsed time = %f ms\n", (finish - start) * 1000);
194         free(vel);
195     }
```

实验结果

N = 64，时间间隔 1ms，迭代 5000 次后的结果截图：

```
 0 X:  13.83150 Y:  0.00129
 1 X: -13.76900 Y:  0.30590
 2 X:  4.78127 Y:  0.55458
 3 X: -4.69854 Y: -0.55607
 4 X:  0.01382 Y:  0.00188
 5 X:  13.84859 Y:  0.00094
 6 X: -13.79760 Y: -0.30804
 7 X:  2.53043 Y:  1.23645
 8 X: -2.26922 Y:  1.01244
 9 X:  0.60310 Y:  0.99081
10 X:  0.07393 Y:  0.94081
11 X:  0.19742 Y:  1.07862
12 X:  0.17182 Y:  0.96174
13 X:  2.47905 Y:  0.76333
14 X: -2.94652 Y:  1.01618
15 X:  2.76779 Y:  1.53133
16 X: -2.20009 Y:  2.00033
17 X:  0.01944 Y:  1.99587
18 X:  0.71108 Y:  2.11582
19 X: -0.30405 Y:  1.88060
20 X:  0.35131 Y:  2.00470
21 X:  2.57014 Y:  1.99973
22 X: -2.43561 Y:  2.47177
23 X:  2.73368 Y:  3.03794
24 X: -2.14745 Y:  3.05629
25 X:  0.03827 Y:  2.99154
26 X:  0.70785 Y:  3.13758
27 X: -0.17480 Y:  2.86255
28 X:  0.50420 Y:  3.00979
```

```
29 X:  2.66861 Y:  2.95717
30 X: -2.21038 Y:  2.94718
31 X:  2.90045 Y:  4.12355
32 X: -2.06344 Y:  4.13344
33 X:  0.16346 Y:  4.00131
34 X:  1.25235 Y:  3.79701
35 X: -0.56401 Y:  4.20293
36 X:  0.52274 Y:  3.99887
37 X:  2.75616 Y:  3.87493
38 X: -2.20778 Y:  3.86791
39 X:  1.88047 Y:  5.63212
40 X: -1.96141 Y:  4.99979
41 X:  0.24541 Y:  5.00092
42 X:  1.77587 Y:  11.69527
43 X:  1.47288 Y:  4.36385
44 X:  0.60338 Y:  4.99928
45 X:  2.81119 Y:  5.00019
46 X: -3.42782 Y: -1.69156
47 X:  20.07531 Y:  8.18286
48 X: -1.87831 Y:  5.65582
49 X:  1.16517 Y:  6.45862
50 X: -0.45737 Y:  5.53775
51 X: -16.10468 Y:  3.82161
52 X:  0.63093 Y:  6.00024
53 X:  2.88613 Y:  6.00033
54 X: -2.27711 Y:  6.34259
55 X:  3.09550 Y:  6.97320
56 X: -1.64113 Y:  7.12421
57 X:  0.54606 Y:  6.99930
58 X:  1.20676 Y:  8.30262
59 X: -0.35548 Y:  5.69945
60 X:  0.79130 Y:  6.98942
61 X:  2.92836 Y:  7.03548
62 X: -1.89138 Y:  6.87573
63 X:  0.63001 Y:  7.99991
```

运行时间及加速比：

时间间隔1ms				
迭代5000步				
规模/线程数	1	2	4	8
64	563.623 ms	382.367 ms	306.252 ms	19963.959 ms
256	8651.594 ms	5867.792 ms	4793.257 ms	28812.963 ms
加速比				
规模/线程数	1	2	4	8
64	0.993	1.474	1.840	0.029
256	0.972	1.474	1.805	0.301

分析总结

与前两次实验一样，较大规模数据使用 MPI 能够显著降低程序的运行时间，但使用线程数太多导致的通信成本也可能会显著增加并行程序的运行时间。