并行计算实验 3 实验报告

实验题目：使用 MPI 接口编程模拟二维情况下的 n 体运动

实验环境: 实验所使用电脑操作系统为 windows10 操作系统，IDE 为 Visual Studio2017，OpenMP 实验环境由 VS2017 提供，MPI 使用 MS-MPI, 代码编写和测试均由 VS2017 实现。

算法设计与分析:

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，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。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）。

根据实验指导的提示，首先我构造了代表球体的结构体，结构体记录了球体的横纵向加速度，横纵向速度，横纵向位置，结构体数组代替所有的球体。随后我构建了三个函数，分别用来计算球体的位置，速度，加速度。

需要注意的是，上述三个函数中，只有计算加速度的函数在计算时需要其他球体的信息，计算速度和位置的函数都是根据本身的信息对相应属性进行更新。

其中，计算加速度时，函数遍历所有的球体信息，计算所有其他球体对自身的横纵向加速度。计算速度时，只要根据已经更新了的自身的加速度乘上时间片得到过去的时间里速度的改变。计算位置时，只要根据已经更新了的速度信息相应地计算在过去的一个时间片内位置的改变即可。

由于本次模拟需要并行化实现，为了节约通信成本，我们把所有的球体按照处理器的数量分块，每个处理器负责对应的一些球体的信息计算。在每次模拟循环开始后，所有的处理器向其他所有处理器发送自己所负责的球体的信息。这样一来所有的处理器都可以接收到所有球体的更新信息，随后处理器顺次计算负责的那些球体的加速度，速度，位置。

在规定的周期模拟结束后，约定一个根处理器，所有其他处理器把模拟数据发送给根处理器，由根处理器统一输出到结果。

核心代码:

```
void compute_force(int index)
{
    ball_list[index].ax = 0;
    ball_list[index].ay = 0;
    for (int i = 0; i<N; i++)
    {
        if (i != index)
        {
            double dx = ball_list[i].px - ball_list[index].px;
            double dy = ball_list[i].py - ball_list[index].py;
            double d = (dx*dx + dy * dy);
            if (d<r*r)d = r * r;
            d *= sqrt(d);

            ball_list[index].ax += GM * (dx) / d;
            ball_list[index].ay += GM * (dy) / d;

        }

    }

}
```

上图为计算加速度的函数，该函数遍历所有球体信息，计算相互作用力并分解向量到横纵方向，便于后续计算和处理信息。

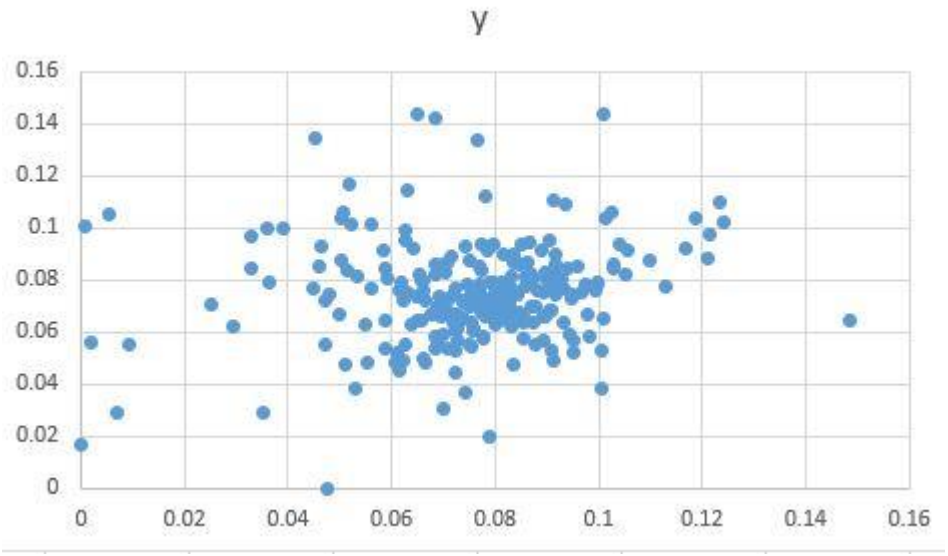
```
void compute_velocities(int index)
{

    ball_list[index].vx += ball_list[index].ax*delta_t;
    ball_list[index].vy += ball_list[index].ay*delta_t;

}
void compute_positions(int index)
{

    ball_list[index].px += ball_list[index].vx*delta_t;
    if (ball_list[index].px>((size - 1) / 100.0))ball_list[index].px = (size - 1) / 100.0;
    if (ball_list[index].px<0)ball_list[index].px = 0;
    ball_list[index].py += ball_list[index].vy*delta_t;
    if (ball_list[index].py>((size - 1) / 100.0))ball_list[index].py = (size - 1) / 100.0;
    if (ball_list[index].py<0)ball_list[index].py = 0;

}
```

上图为计算速度和位置的函数，首先速度由初速度加上时间片乘上加速度得到。
位置由初位置加上速度乘上时间片得到。

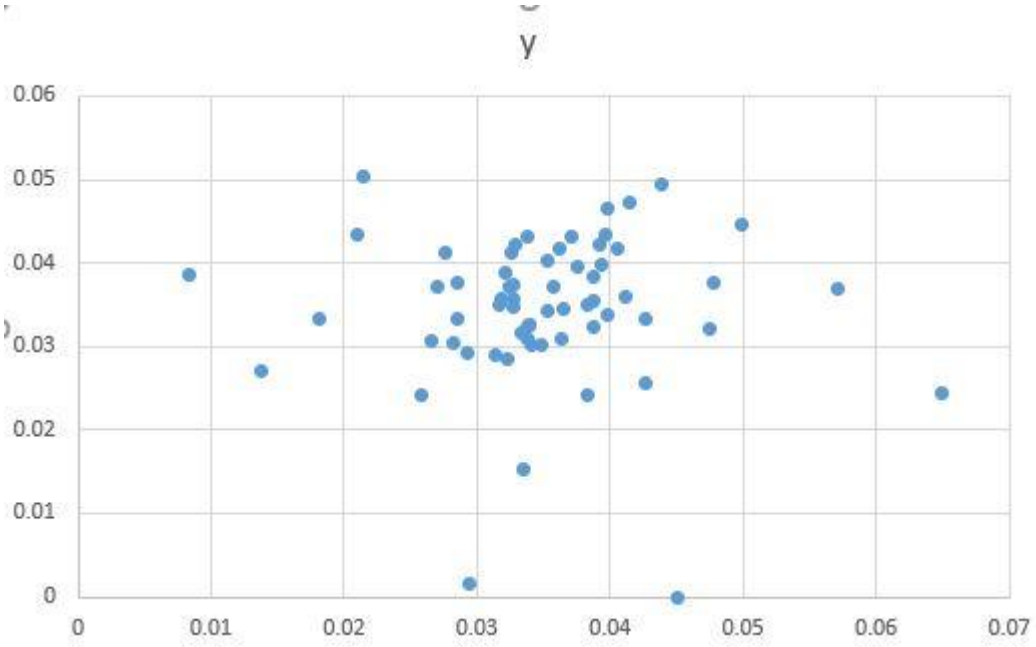
```
starttime = clock();
for (int i = 0; i<cycle_times; i++)
{
    for (int j = 0; j<numprocs; j++)
    {
        if (j != myid)
            MPI_Bsend((ball_list + (N / numprocs)*myid), sizeof(ball)*N / numprocs, MPI_BYTE, j, i * 10 + myid, MPI_COMM_WORLD);
    }
    for (int j = 0; j<numprocs; j++)
    {
        if (j != myid)
        {
            MPI_Status status;
            MPI_Recv((ball_list + (N / numprocs)*j), sizeof(ball)*N / numprocs, MPI_BYTE, j, i * 10 + j, MPI_COMM_WORLD, &status);
        }
    }
    for (int j = (N / numprocs)*myid; j<(N / numprocs)*(myid + 1); j++)
    {
        compute_force(j);
    }

    MPI_Barrier(MPI_COMM_WORLD);
    for (int j = (N / numprocs)*myid; j<(N / numprocs)*(myid + 1); j++)
    {
        compute_velocities(j);
        compute_positions(j);
    }

    MPI_Barrier(MPI_COMM_WORLD);
}
endtime = clock();
```

上图是模拟过程的主函数，可以看到和算法思路相同，使用 MPI 接口函数实现。

实验结果：



球体个数：256　　处理器：4　　周期：20000



球体数：64　处理器：4　周期：20000

可以看到，由于只考虑相互吸引力作用，不考虑碰撞时的排斥力，在模拟足够长的周期后，球体总是趋向于聚集。

运行时间：

规模/进程数	1	2	3	4
64	1.948s	1.638s	1.711s	1.944s
256	23.431s	14.835s	12.914s	14.071s

加速比：

规模/进程数	1	2	3	4
64	1	1.189	1.138	1.002
256	1	1.579	1.814	1.665

实验分析与总结：MPI 是基于进程通信的并行计算，所以进程间不会共享数据，要使用其他进程的数据的时候需要进行通信。在进行进程间通信时，我们要注意根据不同的需求选择不同的信息函数，如 MPI_Bsend,MPI_Sen，MPI_recv 等。