实验三

姓名：王嵘晟
学号： PB17111614

利用 MPI 解决 N 体问题

实验环境

操作系统： Windows 10
IDE： Visual Studio 2019 X64 Debug 模式 MPI环境： MS-MPI V10
硬件配置： Intel CORE i7 6550U

算法设计与分析

对于复杂的 N 体问题，做了以下简化：

1. 将每个小球看作有质量的质点
2. 为了方便计算受力情况，当小球间距小于 MIN_DIS = 0.0001 m 时，在计算受力情况时将间距当作 MIN_DIS 来计算
3. 计算速度和位置时同理，取了一个对于时间的划分 DELTA_T = 0.0001 s，认为小球在这个时间内速度变化为匀变速，加速度恒定。由于时间间隔较小，所以认为小球移动为匀速运动
4. 将小球的受力情况与速度分为 x 分量与 y 分量，方便计算。

对于小球的受力情况，根据万有引力公式：
$$F=\frac{Gm_{1}m_{2}}{r^2}$$
应用到这个问题后可以整理得到：
$$F_x=\frac{GMMd_x}{(d_x^2+d_y^2)^\frac{3}{2}}$$

$$F_y=\frac{GMMd_y}{(d_x^2+d_y^2)^\frac{3}{2}}$$

由此可以得到速度与位移的计算公式:
$$v_x += \frac{F_x}{M}\times \Delta t$$

$$v_y += \frac{F_y}{M}\times \Delta t$$

$$x += v_x\times \Delta t$$

$$y += v_y\times \Delta t$$
经过 iternum 次迭代后得到最终结果。
并行部分：对于小球这6个相关属性变量做通信，根据线程数划分。

核心代码

```
/// <summary>
/// 全部使用国际单位制，小球初始间距0.01。
/// 由于小球不存在碰撞但可能会有质心间距极小情况，所以在此设定一个最小间距为0.0001即初始
```

```
间距的1/100
/// </summary>
typedef struct ball {
    double x;          // 横坐标
    double y;          // 纵坐标
    double f_x;        // x轴受力
    double f_y;        // y轴受力
    double v_x;        // x轴速度
    double v_y;        // y轴速度
}Ball;

/// <summary>
/// F=GMM/r^2, Fx= x /sqrt(x^2+y^2) * F
/// </summary>
/// <param name="balls"></param>
/// <param name="num"></param>
void ComputeForce(Ball* balls, int num)            // 给 balls[num] 计算受力
{
    int i;
    double d_x, d_y;          // 水平 竖直距离
    double d_x_ab, d_y_ab;
    double f_x, f_y;
    f_x = 0.0;
    f_y = 0.0;
    for (i = 0; i < N; i++)
    {
        if (i == num)
            continue;
        d_x = balls[i].x - balls[num].x;
        d_y = balls[i].y - balls[num].y;
        d_x_ab = d_x >= 0 ? d_x : (-1 * d_x);
        d_y_ab = d_y >= 0 ? d_y : (-1 * d_y);
        if (d_x_ab < MIN_DIS)
        {
            if (d_x >= 0)
            {
                d_x = MIN_DIS;
            }
            else
            {
                d_x = -MIN_DIS;
            }
        }
        if (d_y_ab < MIN_DIS)
        {
            if (d_y >= 0)
            {
                d_y = MIN_DIS;
            }
            else
            {
                d_y = -MIN_DIS;
            }
        }
```

```
            f_x += G * M * M * d_x / pow(pow(d_x, 2) + pow(d_y, 2), 1.5);
            f_y += G * M * M * d_y / pow(pow(d_x, 2) + pow(d_y, 2), 1.5);
        }
        balls[num].f_x = f_x;
        balls[num].f_y = f_y;
    }
    /// <summary>
    /// v = a*t = F * t / m
    /// 设定一个最小时间 Δt , v = Σa*t
    /// </summary>
    /// <param name="balls"></param>
    /// <param name="num"></param>
    void ComputeVelocities(Ball* balls, int num)
    {
        balls[num].v_x += balls[num].f_x / M * DELTA_T;
        balls[num].v_y += balls[num].f_y / M * DELTA_T;
    }

    /// <summary>
    /// 在 Δt 时间内看作匀速运动
    /// </summary>
    /// <param name="balls"></param>
    /// <param name="num"></param>
    void ComputePositions(Ball* balls, int num)
    {
        balls[num].x += balls[num].v_x * DELTA_T;
        balls[num].y += balls[num].v_y * DELTA_T;
    }
```

主函数并行部分：

```
    // MPI 初始化
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &mypid);
    MPI_Comm_size(MPI_COMM_WORLD, &mpi_threads_num);

    startTime = MPI_Wtime();
    startPos = N * mypid / mpi_threads_num;
    endPos = N * (mypid + 1) / mpi_threads_num;

    for (k = 0; k < iternum; k++)
    {
        for (i = startPos; i < endPos; i++)
        {
            ComputeForce(balls, i);
        }
        for (i = startPos; i < endPos; i++)
        {
            ComputeVelocities(balls, i);
        }
        for (i = startPos; i < endPos; i++)
        {
```

```
            ComputePositions(balls, i);
        }
        MPI_Barrier(MPI_COMM_WORLD);

        if (mypid != 0)
        {
            // 更新的位置信息发送给 thread 0
            MPI_Send(&(balls[startPos]), 6 * (endPos - startPos + 1), MPI_DOUBLE,
0, iternum * 10 + mypid, MPI_COMM_WORLD);
        }
        else
        {
            // thread 0 接受其他 thread 发送来的信息
            for (i = 1; i < mpi_threads_num; i++)
            {
                MPI_Recv(&(balls[N * i / mpi_threads_num]), 6 * N, MPI_DOUBLE, i,
iternum * 10 + i, MPI_COMM_WORLD, &status);
            }
            // 接受全部信息后广播，更新其他 thread 的位置信息
            MPI_Bcast(balls, 6 * N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        }
        MPI_Barrier(MPI_COMM_WORLD);
    }
    endTime = MPI_Wtime();
```

实验结果

时间

规模\进程数	1	2	4	8
N=25 iternum=500	0.118872s	0.075708s	0.046791s	6.904926s
N=64 iternum=500	0.812122s	0.423867s	0.284208s	7.162476s
N=256 iternum=500	12.855535s	6.740138s	4.006196s	13.849497s

加速比(与串行相比)

规模\进程数	1	2	4	8
N=25 iternum=500	1	1.570138	2.540489	0.017216
N=64 iternum=500	1	1.915983	2.857491	0.113386
N=256 iternum=500	1	1.90731	3.208913	0.928231

实验结论

1. N体问题整体来看数据量不算太大，迭代次数使用500次计算量中等。对于加速效果，依然可以体现出随着计算规模的增大加速效果越来越好
2. 由于物理内核限制，4线程时加速效果最好

3. 使用 MPI 对于解决物理中的实际复杂问题有一定的帮助