实验三

PB17081504 廖洲洲

实验题目

利用 MPI 解决N体问题

实验环境

操作系统：Win 10

IDE：Microsoft Visual Studio 2015

编译器：cl.exe

硬件配置：CPU：Intel Core i7-8550U；CPU核心数:4；内存:8G

算法设计与分析

- 有关参数

```
#define N 64                    //小球个数
#define G 6.67e-11              //万有引力常数
#define M 10000                 //小球质量
#define Cycle 50000             //总的周期数，每周期时间=TotalTime/Cycle
#define CycleTime 0.01          //每周期的时间
```

- 定义小球结构体

```
struct ball {
    double ax, ay;              //加速度
    double vx, vy;              //速度
    double px, py;              //位置
};
```

计算加速度需要求得小球所受的合外力，因此需要知道所有小球的位置。故需要在每个进程的每个周期循环后向其他进程发送其负责的小球相关数据。由于每个进程都需要别的进程的数据，同时也需要给别的进程发送数据，故选择MPI中的广播功能作为进程通信方式。

计算速度和位置只需要自己的加速度即可。

- 主要算法思想

> 将程序分给n个进程，每个进程平均负责计算一组小球的位置。程序外循环为周期循环，每个周期计算当前周期小球的位置，由于小球被分给了不同的进程计算，而计算加速度需要所有小球的位置，故在每个周期结束后，使用广播向所有进程广播小球位置，实现进程同步。计算小球位置时，首先根据小球位置计算出各个小球的加速度，然后再计算出小球新的速度，再计算出小球新的位置。为了避免两个小球过近出现力无穷大的情况，当两个小球的距离小于0.001m时使用0.001m计算。

由于计算每个小球的加速度需要球每个小球与其他所有小球的力，故对于一个线程的时间复杂度为O($Cycle * N^2$),空间复杂度为O(N)

核心代码

```
    //main函数核心循环
    startTime = MPI_Wtime();
    //每个周期计算小球的位置，周期结束时向其他进程广播本进程负责的小球位置
    partition = N / size;//负责的小球个数
    low = rank*partition;//负责的小球的最小编号
    high = (rank + 1)*partition - 1;//负责的小球的最大编号
    for (i = 0; i < Cycle; i++) {
        for (j = low; j <= high; j++) {
            //首先更新各个小球在当前位置受到的力，计算出加速度
            compute_force(j, ballList);
        }
        for (j = low; j <= high; j++) {
            //计算出各个小球新的位置
            compute_positions(j, ballList);
        }
        for (j = 0; j < size; j++) {
            //每个进程广播自己负责的小球信息，实验了数据同步
            MPI_Bcast(&ballList[j*partition], sizeof(ball)*partition, MPI_BYTE,
j, MPI_COMM_WORLD);
        }
        MPI_Barrier(MPI_COMM_WORLD);
    }
    endTime = MPI_Wtime();


    //计算加速度、速度、位置的相关函数
void compute_force(int index, ball *ballList) {
    //计算编号为index的小球所受到的加速度
    //初始化加速度为0，再遍历其它小球，计算新的加速度
    double px, py;
    double dx, dy,d;
    px = ballList[index].px;
    py = ballList[index].py;
    ballList[index].ax = 0;
    ballList[index].ay = 0;
    for (int i = 0; i < N; i++) {
        if (i != index) {
            dx = ballList[i].px - px;
            dy = ballList[i].py - py;
            d = sqrt(dx*dx + dy*dy);
            if (d < 0.001) {
                //如果两球距离过小，使用1cm作为距离计算
                d = 0.001;
                dx = 0.001*dx / d;
                dy = 0.001*dy / d;
            }
            ballList[index].ax += G*M*dx / (d*d*d);
            ballList[index].ay += G*M*dy / (d*d*d);
        }
    }
}

void compute_velocities(int index, ball * ballList) {
    //计算每个小球的速度
```

```
    ballList[index].vx += ballList[index].ax*CycleTime;
    ballList[index].vy += ballList[index].ay*CycleTime;
}

void compute_positions(int index, ball * ballList) {
    //计算每个小球的位置
    double oldVx, oldVy;
    //保存旧的速度值
    oldVx = ballList[index].vx;
    oldVy = ballList[index].vy;
    compute_velocities(index, ballList);
    ballList[index].px += (oldVx + ballList[index].vx)*CycleTime / 2.0;
    ballList[index].py += (oldVy + ballList[index].vy)*CycleTime / 2.0;
}
```

实验结果

N=64 CycleTime（每周期时间）=0.01s

运行时间（单位s）

规模(周期数)进程数	1	2	4	8
5000	0.296494	0.154361	0.116779	0.144644
10000	0.706362	0.303846	0.267726	0.337912
50000	5.304150	3.018286	1.242270	1.734022
100000	11.059648	5.423501	4.486377	6.432215

加速比

规模\进程数	1	2	4	8
10000	1	1.920783	2.538933	2.049819
50000	1	2.324737	2.638377	2.090373
100000	1	1.757338	4.269724	3.058871
500000	1	2.039208	2.465162	1.719415

N=256 CycleTime（每周期时间）=0.01s

运行时间

规模\进程数	1	2	4	8
10000	6.049876	3.197611	1.659807	1.249792
50000	71.222821	45.394439	32.140079	41.055763
100000	170.219044	97.742511	62.819452	64.389268
500000	809.904365	430.126721	326.015264	344.361256

加速比

规模\进程数	1	2	4	8
10000	1	1.891999	3.644927	4.840706
50000	1	1.568977	2.216013	1.734783
100000	1	1.741505	2.709655	2.643593
500000	1	1.882944	2.484253	2.351903

分析与总结

- 遇到使用8个进程程序运行时间特别长的情况，需要近10分钟才能出结果。后来发现是因为后台程序过多，CPU资源不足，进程间资源调度可能会花费了较多时间。关闭后台运用后，程序运行时间大大缩短，比较符合预期。
- 使用广播通信方式时，要注意<Address,Count,Datatype>对root进程来说，其既定义了发送缓冲也定义了接收缓冲，其他进程只定义了接收缓冲。也就是，当其他进程执行广播语句，发现Root不是自己是会进行接收操作，若是自己则会进行发送操作。这种发送利于整个数组的同步。
- 程序运行规模较大，故并行效果明显，随着进程数的增多，运行时间减少，加速比也随着增大。但同样的，进程数越多，加速比的增长也更加缓慢
- 进程数增大为8时，除了N=256，cycle=1000的情况下，其他程序运行时间较进程数为4时都增大了，加速比减小了。这可能是因为8个进程同时处理大规模数据时，计算机资源不足，计算机对资源的调配与缓存等花费了较多时间。因此，对于并行度的设置，不能一味的求大，而是要找到一个最佳的设置，使其能够合理地利用计算机资源。

实验结果截图

N=64,周期数100000，每周期时间0.01s时不同并行度结果截图（N=256小球过多，略）

```
E:\Visual Studio Projects\MPI_NBody\x64\Debug>mpiexec -n 1 MPI_NBody.exe
运行时间:11.059648
球最终位置
        (27.31,27.31)   (20.75,5.57)    (31.94,-13.20)  (4.01,-15.93)   (-5.58,-14.35)  (-31.06,-13.88) (-20.74,5.59)   (-28.04,28.11)
        (5.60,20.76)    (42.94,42.94)   (54.17,26.94)   (8.03,3.30)     (-7.93,3.33)    (-54.09,26.94)  (-42.43,42.51)  (-5.52,20.81)
        (-13.22,31.93)  (26.94,54.17)   (16.77,16.77)   (20.01,37.23)   (-21.98,36.53)  (-16.71,16.75)  (-26.87,54.16)  (12.59,32.83)
        (-13.03,7.35)   (3.29,8.01)     (37.07,19.84)   (26.39,26.40)   (-24.63,24.73)  (-36.26,21.81)  (-3.26,8.00)    (14.11,6.21)
        (-14.34,-5.60)  (3.16,-7.94)    (36.36,-22.14)  (24.97,-24.88)  (-21.88,-21.88) (-34.50,-26.88) (-3.18,-7.96)   (16.08,-3.99)
        (-13.88,-31.06) (26.94,-54.10)  (17.52,-17.48)  (22.02,-36.14)  (-26.84,-34.47) (-17.48,-17.49) (-26.88,-54.10) (13.27,-31.87)
        (5.56,-20.52)   (43.37,-43.29)  (54.17,-26.87)  (8.01,-3.09)    (-7.94,-3.17)   (-54.10,-26.88) (-44.22,-44.22) (-5.51,-20.67)
        (26.96,-26.89)  (20.58,-5.49)   (32.83,12.59)   (6.23,14.09)    (-7.31,13.15)   (-31.85,13.29)  (-20.67,-5.54)  (-25.75,-25.75)

E:\Visual Studio Projects\MPI_NBody\x64\Debug>mpiexec -n 2 MPI_NBody.exe
运行时间:5.423501
球最终位置
        (27.31,27.31)   (20.75,5.57)    (31.94,-13.20)  (4.01,-15.93)   (-5.58,-14.35)  (-31.06,-13.88) (-20.74,5.59)   (-28.04,28.11)
        (5.60,20.76)    (42.94,42.94)   (54.17,26.94)   (8.03,3.30)     (-7.93,3.33)    (-54.09,26.94)  (-42.43,42.51)  (-5.52,20.81)
        (-13.22,31.93)  (26.94,54.17)   (16.77,16.77)   (20.01,37.23)   (-21.98,36.53)  (-16.71,16.75)  (-26.87,54.16)  (12.59,32.83)
        (-13.03,7.35)   (3.29,8.01)     (37.07,19.84)   (26.39,26.40)   (-24.63,24.73)  (-36.26,21.81)  (-3.26,8.00)    (14.11,6.21)
        (-14.34,-5.60)  (3.16,-7.94)    (36.36,-22.14)  (24.97,-24.88)  (-21.88,-21.88) (-34.50,-26.88) (-3.18,-7.96)   (16.08,-3.99)
        (-13.88,-31.06) (26.94,-54.10)  (17.52,-17.48)  (22.02,-36.14)  (-26.84,-34.47) (-17.48,-17.49) (-26.88,-54.10) (13.27,-31.87)
        (5.56,-20.52)   (43.37,-43.29)  (54.17,-26.87)  (8.01,-3.09)    (-7.94,-3.17)   (-54.10,-26.88) (-44.22,-44.22) (-5.51,-20.67)
        (26.96,-26.89)  (20.58,-5.49)   (32.83,12.59)   (6.23,14.09)    (-7.31,13.15)   (-31.85,13.29)  (-20.67,-5.54)  (-25.75,-25.75)
```

```
E:\Visual Studio Projects\MPI_NBody\x64\Debug>mpiexec -n 4 MPI_NBody.exe
运行时间:4.486377
球最终位置
        (27.31,27.31)   (20.75,5.57)    (31.94,-13.20)  (4.01,-15.93)   (-5.58,-14.35)  (-31.06,-13.88) (-20.74,5.59)   (-28.04,28.11)
        (5.60,20.76)    (42.94,42.94)   (54.17,26.94)   (8.03,3.30)     (-7.93,3.33)    (-54.09,26.94)  (-42.43,42.51)  (-5.52,20.81)
        (-13.22,31.93)  (26.94,54.17)   (16.77,16.77)   (20.01,37.23)   (-21.98,36.53)  (-16.71,16.75)  (-26.87,54.16)  (12.59,32.83)
        (-13.03,7.35)   (3.29,8.01)     (37.07,19.84)   (26.39,26.40)   (-24.63,24.73)  (-36.26,21.81)  (-3.26,8.00)    (14.11,6.21)
        (-14.34,-5.60)  (3.16,-7.94)    (36.36,-22.14)  (24.97,-24.88)  (-21.88,-21.88) (-34.50,-26.88) (-3.18,-7.96)   (16.08,-3.99)
        (-13.88,-31.06) (26.94,-54.10)  (17.52,-17.48)  (22.02,-36.14)  (-26.84,-34.47) (-17.48,-17.49) (-26.88,-54.10) (13.27,-31.87)
        (5.56,-20.52)   (43.37,-43.29)  (54.17,-26.87)  (8.01,-3.09)    (-7.94,-3.17)   (-54.10,-26.88) (-44.22,-44.22) (-5.51,-20.67)
        (26.96,-26.89)  (20.58,-5.49)   (32.83,12.59)   (6.23,14.09)    (-7.31,13.15)   (-31.85,13.29)  (-20.67,-5.54)  (-25.75,-25.75)

E:\Visual Studio Projects\MPI_NBody\x64\Debug>mpiexec -n 8 MPI_NBody.exe
运行时间:6.432215
球最终位置
        (27.31,27.31)   (20.75,5.57)    (31.94,-13.20)  (4.01,-15.93)   (-5.58,-14.35)  (-31.06,-13.88) (-20.74,5.59)   (-28.04,28.11)
        (5.60,20.76)    (42.94,42.94)   (54.17,26.94)   (8.03,3.30)     (-7.93,3.33)    (-54.09,26.94)  (-42.43,42.51)  (-5.52,20.81)
        (-13.22,31.93)  (26.94,54.17)   (16.77,16.77)   (20.01,37.23)   (-21.98,36.53)  (-16.71,16.75)  (-26.87,54.16)  (12.59,32.83)
        (-13.03,7.35)   (3.29,8.01)     (37.07,19.84)   (26.39,26.40)   (-24.63,24.73)  (-36.26,21.81)  (-3.26,8.00)    (14.11,6.21)
        (-14.34,-5.60)  (3.16,-7.94)    (36.36,-22.14)  (24.97,-24.88)  (-21.88,-21.88) (-34.50,-26.88) (-3.18,-7.96)   (16.08,-3.99)
        (-13.88,-31.06) (26.94,-54.10)  (17.52,-17.48)  (22.02,-36.14)  (-26.84,-34.47) (-17.48,-17.49) (-26.88,-54.10) (13.27,-31.87)
        (5.56,-20.52)   (43.37,-43.29)  (54.17,-26.87)  (8.01,-3.09)    (-7.94,-3.17)   (-54.10,-26.88) (-44.22,-44.22) (-5.51,-20.67)
        (26.96,-26.89)  (20.58,-5.49)   (32.83,12.59)   (6.23,14.09)    (-7.31,13.15)   (-31.85,13.29)  (-20.67,-5.54)  (-25.75,-25.75)
```