Lab3-report

- 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。

  实验中参数设定如下：

  - 引力常数数值取$\mathbf{6.67 * 10^{-11}}$
  - 小球重量都为 $\mathbf{10000}kg$
  - 小球半径都为$\mathbf{1}cm$
  - 小球间的初始间隔为$\mathbf{1}cm$，例：$N = \mathbf{36}$时，则初始的正方形区域为$\mathbf{5}cm * \mathbf{5}cm$
  - 小球初速为$\mathbf{0}$
  - 对于时间间隔，公式为：$delta\_t = \frac{1}{timestep}$

    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。

    在实验中我取的$timestep = 1000$，模拟时间$T = 10s$

- 实验环境

  - 操作系统：Ubuntu 18.04 lts 64bits
  - 编译器
    - mpi 3.2.1 (for mpi)
  - CPU: Inte i5-8250U CPU & 3.40Ghz * 8
  - Memory： 8GB

- 算法设计与分析

  在本次实验中，要求模拟N体运动，并且为了简化模型条件，忽略小球的客观物理实体情况（不会考虑小球的碰撞等问题），在算法设计和编码实现时，整个实验的算法流程可以分为单体的受力、速度和位置更新算法，MPI并行化实现两个部分。

  - 单体的受力、速度和位置更新算法

在实验中，N体运动的模型进行了部分简化，其中可以忽略小球在运动过程中的形体上的冲突问题，当两个小球的距离小于半径之和（实验中取的为0.02m），则取小球间的距离为0.02m。初始条件下进行位置的初始化，其分布是小球在二维正方形空间的均匀分布。在初始化之后，按照按照给定的模拟时间和1s内的迭代次数进行循环模拟。对于每个小球的单次迭代过程，其模拟顺序为受力分析、速度更新和位置更新。由于小球的运动是在二维空间上的，所以需要进行力的分解。两个球体间的引力计算公式表示为：$F = \frac{GMm}{R^2}$，在分析每个小球的受力时，依次遍历其他小球的位置，然后两个小球间的引力进行x轴和y轴方向上的分解，并且要累加所有其他小球对当前小球的两个方向上引力和，这样得到两个方向上的加速度分别为x方向加速度$\sum_i a_x = \frac{GM}{R_i^2} \cdot \frac{\Delta x_i}{R_i}$，y方向加速度$\sum_i a_y = \frac{GM}{R_i^2} \cdot \frac{\Delta y_i}{R_i}$，其中距离表示为$R_i = \sqrt{(\Delta x_i)^2 + (\Delta y_i)^2}$，然后根据方向加速度计算$x$方向上的速度值$v_x = v_x + a_x \cdot \Delta t$，$y$方向的速度值为$v_y = v_y + a_y \cdot \Delta t$。更新速度之后，即可得到该$\Delta t$时间内小球从位置$(x, y)$移动到位置$(x + v_x \cdot \Delta t, y + v_y \cdot \Delta t)$。这样就实现了单次迭代过程中每个小球从引力分析开始到更新位置坐标的过程，如果是T个周期进行模拟，每一次模拟过程相同。

具体代码实现时，以上更新维护过程分别三个函数中依次实现，引力分析并且得到分方向的加速度值在函数`compute_force(int k)`中实现，速度的更新在函数`compute_velocities(int k)`，位置坐标的更新在函数`compute_positions(int k)`中实现。

- **MPI并行实现**

在用MPI实现并行时，首先要调用MPI函数`MPI_Init(..)`进入MPI环境并完成所有的初始化工作，然后，调用`MPI_Comm_rank`函数获得当前进程在指定通信域中的编号，将自身与其他程序区分。然后要获取指定通信域的进程数，调用`MPI_Comm_size`函数获取指定通信域的进程个数。如果返回的rank为0的话说明是主线程。在并行程序中需要指定模拟的时间周期数T和小球的数目N。

各个线程接收到参数之后，根据线程数目$size$进行任务的分配，根据效率最优的原理每一个线程承担的任务量相同均为$n = \frac{N}{size}$，即每一个线程执行的模拟小球数量都为n。首先根据线程号$rank$进行初始位置的初始化，我在实验中的任务划分原则为线程号$rank = i$的线程负责从$\frac{N}{size} \cdot i$到$\frac{N}{size} \cdot (i+1)$编号的小球的模拟，在初始化时根据小球编号在二维正方形空间中进行间隔为1cm的位置设定即可。

由于在模拟过程中每个小球的受力跟其他小球的位置相关，所以在此过程中要对线程进行数据的同步，也就是在每一个周期模拟之前需要把线程的小球位置情况广播到其他线程，我所采用的是多对多通信方式，调用的MPI接口函数为`MPI_Allgather`，即设定一个全局接收缓冲区，多对多通信接收到的结果保存在这个buff中，然后在本个周期内模拟时就得到了全部的N个小球的位置信息。在线程通信时并没有必要把全部的信息进行广播收集通信，因为对于其他线程而言使用到的数据仅为小球的位置，所以速度值和加速度值没必要进行线程同步，仅仅同步位置这一数据即可，这样就降低了通信代价。

所有线程同步的模拟完成后rank为0的线程进行汇总，得到程序的运行时间和N体问题的最终模拟结果。最后通过`MPI_Finalize`函数从MPI环境中退出。

- **算法时间复杂度及优化分析**

在模拟时，按照以上分析的算法流程，每个时间周期内，每个小球需要遍历所有数目N个小球的位置，一共在T的时间周期数内模拟N个小球的运动，所以其时间复杂度为$O(N^2 T \frac{1}{size})$。MPI并行算法理论上的时间效率与线程数目成正比关系。

- 核心代码
  - 单体的受力、速度和位置更新算法

```
// 每个线程中第k个小球的受引力计算，进而得到方向上的分解加速度
void compute_force(int k)
{
    float dis_x, dis_y, a;
    k = N / size * rank + k;
    a_x = 0.0;
    a_y = 0.0;

    for(int i = 0; i < N; i++)
    {
        if(i == k)
            continue;
        dis_x = fabs(all_balls_x[i] - all_balls_x[k]);
        dis_y = fabs(all_balls_y[i] - all_balls_y[k]);
        dis_x = dis_x > 0.02 ? dis_x : 0.02;
        dis_y = dis_y > 0.02 ? dis_y : 0.02;
        a = G * M / pow(dis_x * dis_x + dis_y * dis_y, 1.5);
        a_x += a * (all_balls_x[i] - all_balls_x[k]);
        a_y += a * (all_balls_y[i] - all_balls_y[k]);
    }
}
// 每个线程中第k个小球的速度更新
void compute_velocities(int k)
{
    curr_balls_vx[k] += a_x / TIME_STEP;
    curr_balls_vy[k] += a_y / TIME_STEP;
}
// 每个线程中第k个小球的位置坐标更新
void compute_positions(int k)
{
    curr_balls_x[k] += curr_balls_vx[k] / TIME_STEP;
    curr_balls_y[k] += curr_balls_vy[k] / TIME_STEP;
}
```

  - **MPI**执行部分

```
    // 根据线程编号，初始化小球位置
    int k = 0;
    for(int i = N / size * rank; i < N / size * (rank + 1); i++)
    {
        curr_balls_x[k] = i % (N / size);
        curr_balls_y[k++] = i / (N / size);
    }
    ....
    for(int i = 0; i < TIME * TIME_STEP; i++)
    {
        // 线程多对多通信进行同步
        MPI_Allgather(curr_balls_x, N / size, MPI_FLOAT, all_balls_x, N / size,
MPI_FLOAT, MPI_COMM_WORLD);
        MPI_Allgather(curr_balls_y, N / size, MPI_FLOAT, all_balls_y, N / size,
MPI_FLOAT, MPI_COMM_WORLD);
        // 模拟过程
        for(int j = 0; j < N / size; j++)
        {
            compute_force(j);
            compute_velocities(j);
            compute_positions(j);
        }
    }
```

- 实验结果

  针对实验要求中的两个参数N测试时间结果如下：（实验中设定的timestep=1000，时间T=10s）

  - 运行时间

规模\线程数	1	2	4	8
N=64	3.036419s	2.072088s	1.413075s	1.065423s
N=256	37.130213s	21.492535s	16.426047s	12.707623s

  - 加速比

规模\线程数	1	2	4	8
N=64	1	1.47	2.15	2.85
N=256	1	1.73	2.31	3.05

- 结果分析

  从实验结果来看，$N = 256$规模下的的运行时间较长，并且每一个规模下多线程的并行化都起到了明显的加速效果，且线程数越多加速比越大，但是并不是与线程数成线性关系，这是因为随着线程数的增加，通信开销也会变大。

- 总结和收获

此次实验，通过MPI模拟了N体运动的模型，并且在线程通信时使用了多对多的通信方式，通过实验可以看到并行化之后可以提高程序效率。