实验 3

PB17111626 秦亚璇

1 实验题目:

利用 MPI 解决 N 体问题

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。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。初始情况下，N 个小球等间隔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。小球可以看成质点。小球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。

2 实验环境(操作系统,编译器,硬件配置等):

操作系统 Ubuntu 16.04,编译器 g++，硬件配置 ThinkPad X1 Carbon(16G)

3 算法设计与分析(写出解题思路和实现步骤)

N 体问题实质上就是每次给一个很小的间隔进行加速度，速度，位置的更新的问题，具体的更细微的要求我们稍后再做考虑。出于并行的考虑，每次更新的时候我们都希望按照之前实验的思路将 N 个小球（注意他们是质点，且假定可以相互穿透）的位置计算分散到 m 个进程上进行计算，这里最致命的问题是，每个进程都只更新了自己的部分，别的部分依然陈旧，照此进行计算，会造成越来越大的偏差位移，为解决这个问题，我们每次计算结束之后都需要让所有的进程之间相互通信达到信息的统一。此外就是在上一个实验中遭遇到的 point，由于存在时间周期模拟，我们要极力避免出现信息存在时间差，所有一定要在必要的部分加入语句强行对其进程，一定程度的时间损失换来绝对的正确性。最后就是时间间隔的选取，这个地方按照基础物理知识不难理解，越小的时间间隔，模拟越精确，在计算机计算能力可满足的情况下，我选择了 0.0001 作为时间间隔。此外就是计算技巧（该部分有参考），我之前一直在思索，这样连方向都难以确定的力学问题要以什么样的方式来表示，角度和大小足够复杂，牵涉了各种各样的数学问题，最终在参考了网上已有的代码之后，选择了 x，y 轴的分解，所有问题迎刃而解。最后是很细微的点，设计好平面的大小之后，要进行边界条件的判断，我假定边界是墙，走到边界，也只能在边界的部分停下来，这里还有多谢球的假设是质点，处理极其简单。

4 核心代码(写出算法实现的关键部分,如核心的循环等)

```
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 == 0)
                continue;
            ball_list[index].ax += 6.67e-3 * dx / sqrt(d) / d;
            ball_list[index].ay += 6.67e-3 * dy / sqrt(d) / d;
        }
    }
}
void compute_velocities(int index)
{
    ball_list[index].vx += ball_list[index].ax * 0.0001;
    ball_list[index].vy += ball_list[index].ay * 0.0001;
}
void compute_positions(int index)
{
    compute_velocities(index);
    ball_list[index].px += ball_list[index].vx * 0.0001;
    if(ball_list[index].px < 0 )
        ball_list[index].px = 0;
    if(ball_list[index].px > bian)
        ball_list[index].px = bian;
    ball_list[index].py += ball_list[index].vy * 0.0001;
    if(ball_list[index].py < 0)
        ball_list[index].py = 0;
    if(ball_list[index].py > bian)
        ball_list[index].py = bian;
}
```

```
begin = MPI_Wtime();
int mm = N / numofprocess;
for (t = 0; t < 100; t++)
{
    for (j = 0; j < numofprocess; j++)
        if (j != rankid)
            MPI_Bsend((ball_list + mm * rankid), sizeof(ball) * mm, MPI_BYTE, j,0, MPI_COMM_WORLD);
    for (j = 0; j < numofprocess; j++)
        if (j != rankid)
        {
            MPI_Status status;
            MPI_Recv((ball_list + mm * j), sizeof(ball) * mm, MPI_BYTE, j,0, MPI_COMM_WORLD, &status);
        }
    for (j = mm * rankid; j < mm * (rankid + 1); j++)
        compute_force(j);
    MPI_Barrier(MPI_COMM_WORLD);
    for (j = mm * rankid; j < mm * (rankid + 1); j++)
        compute_positions(j);
    MPI_Barrier(MPI_COMM_WORLD);
}
end = MPI_Wtime();
```

5 实验结果

运行时间（100 周期）

规模\进程数	1	2	4	8
64	0.0062s	0.0033s	0.0043s	6.0s
256	0.095s	0.050s	0.031s	6.2s

加速比（100 周期）

规模\进程数	1	2	4	8
64	1	1.88	1.44	0.001
256	1	1.9	3.06	0.015

6 分析与总结

并行计算本身有明显的加速效果，从上面的数据也不难看出，两倍左右是很正常的加速比，甚至会达到 3-5 倍，但是要妥善衡量规模和进程之间此消彼长的关系，这是计算开销和

通信开销之间的 tradeoff，诚然我也想过也见过太多的进程加速效果不会很好，但是这次实验给了我更加直观的感受，不正确的运用并行本身是一种灾难，减速效果会大大出乎意料。另外就是并行计算本身有很大的先决局限性，所有程序都是建立在大家数据统一的基础上的，所以彼此等待，相互沟通是一个必不可少的条件，必须严格执行。