实验二

PB17081504 廖洲洲

实验题目

利用 MPI 进行蒙特卡洛模拟

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

1.设置v_max=15,p=0.4，其中CarNum是模拟的车辆总数，cycle是模拟周期数

2.声明一个车辆结构体

```
struct car
{
    int v;//当前速度
    int x;//当前车头位置
};
```

车辆间距可由该辆车的前一辆车车头位置-该辆车的车头位置-1(车身长度)可得。

3.主要算法思想

程序包含size个进程，各个进程维护一个数组元素为car的carList数组，该数组大小为CarNum/size，数组中每个元素代表道路上一辆车。对于编号为rank的进程，其数组中元素代表的车辆编号为rank*CarNum/size~(rank+1)*CarNum/size-1。即相当于将道路上的车辆连续划分为size份，第i份车辆由编号为i的进程维护。

在每个周期，更新各辆车的位置和速度

对每个进程维护的车辆，如果它不是第一辆车，那么它能从数组的前一位置得到它前面一辆车的位置，从而可以按要求模拟车辆的运动。

对于每个进程k(k>0，第一个进程的第一辆车的d=无穷大)维护的车辆，如果它是第一辆车，那么它的前一辆车是进程k-1维护的最后一辆车。因此进程k在除第0周期外每个周期需要先接收进程k-1发来的其上一辆车的位置。同时，在每个周期结束时，各个进程m(m<size-1，最后一个进程不用向后发数据)要将其最后一辆车的位置发送给进程m+1。

由于每个进程只维护其所负责的那一段车辆大小的数组，故空间复杂度O(CarNum)，程序运行时间会随着进程数的增多而减少，单个进程时间复杂度为O(cycle*CarNum)。

核心代码

```
    //main函数主循环
    for (i = 0; i < partition; i++) {
        carList[i].v = 0;
        carList[i].x = CarNum - (rank*partition + i);
    }
    //初始时第一辆车的前车位置
    preCarX = CarNum - (rank*partition - 1);
    for (j = 0; j < Cycle; j++) {
        if (j > 0&&rank>=1) {
            //接收第一辆车的上一辆车的位置
            MPI_Recv(&preCarX, 1, MPI_INT, rank - 1, (j-1) * 10 + rank-1,
MPI_COMM_WORLD, &status);
        }
        for (i = partition-1; i >= 0; i--) {
            //车运动
            if (i == 0) {
                carMove(&carList[i], preCarX, rank*partition + i);
            }
            else {
                carMove(&carList[i], carList[i - 1].x,rank*partition + i);
            }
        }
        if (rank < size - 1) {
            //向下一个进程发送最后一辆车的位置
            MPI_Send(&(carList[partition - 1].x), 1, MPI_INT, rank + 1, j * 10 +
rank, MPI_COMM_WORLD);
        }
    }


    //carMove函数
int carMove(car * thisCar, int preCarX,int carNum) {
    if (carNum == 0 && thisCar->v<v_max) {
        //第一辆车d=无穷大，速度加一
        thisCar->v += 1;
    }
    else if (carNum>0) {
        int d = preCarX - thisCar->x - 1;
        if (d > thisCar->v&&thisCar->v<v_max) {
            //d>v，速度加1
            thisCar->v += 1;
        }
        else {
            //d<=v，速度降到d
            thisCar->v = d;
        }
    }
    double newp = rand() / (RAND_MAX + 1.0);
    if (newp <= p&&thisCar->v >= 1) {
        thisCar->v = thisCar->v - 1;
    }
    thisCar->x += thisCar->v;
    return 1;
}
```

实验结果

运行时间(单位s)

规模\进程数	1	2	4	8
车辆：100000，周期：2000	28.926785	16.917243	8.863483	6.150736
车辆：500000，周期：500	36.933861	19.660663	11.062099	7.168110
车辆：1000000，周期：300	44.598975	23.135263	13.835642	7.792565

加速比

规模\进程数	1	2	4	8
车辆：100000，周期：2000	1	1.709899	3.263591	4.702979
车辆：500000，周期：500	1	1.878566	3.338775	5.152524
车辆：1000000，周期：300	1	1.927749	3.223484	5.723273

分析与总结

- 每个进程维护的车辆数组大小为数组大小为CarNum/size，而不是CarNum，这样更能节省空间
- 使用MPI_Send和MPI_Recv要注意MPI_Recv的参数tag要和MPI_Send的参数tag相同，同时也要注意消息的发送和接收源的正确性。如果发送和接受的tag不一致，将会导致发送接收错误或者某进程一直接收不到消息，则该进程会一直阻塞在接收消息的位置。
- 本实验数据规模较大，因此使用MPI对程序运行时间大大减少。
- 随着进程数的增加，加速比也相应的增加。但是加速比的增加会随着进程数的增加而放缓。当进程数为2时，加速比比较接近2。当进程数为8时，加速比平均为5.2左右。可以看出加速比/进程数在随着进程数的增大而减小。