# 实验二

姓名：王嵘晟
学号： PB17111614

# 利用 MPI 进行蒙特卡洛模拟

## 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

设定最高车速70，减速概率0.5。由于堵车情况一开始所有车辆都是首尾相接的，只有最前面的车移动了后面的车才会有移动的可能。所以并行部分给每个线程分配等量的车数，并对车的位置和速度做进程间通信。判断车速和移动的依据为速度小于前车距离则速度++，直到车速等于前车距离，而完成加速后用一个随机数来决定是否减速。随后移动距离等于单位时间内的车速。由于最后一个线程控制道路最前面一部分的车辆，所以此时线程通信除了最后一个线程外需要接收后一个线程的位置信息。完成这一切后将速度信息和位置信息发送给thread 0，随后将结果输出。时间复杂度为O(car_num × iternum / mpi_threads_num)

## 核心代码

```
int Move(int v, int d)          // 速度变化函数
{
    if (v < d)
    {
        if (v < V_MAX)
            v++;
    }
    else
    {
        v = d;
    }
    double r = (rand() % 100) / 100.0;
    if (r < P && v > 0)
    {
        v --;
    }
    return v;
}

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

```
// 信息传递
start = MPI_Wtime();
startpos = mypid * car_num / mpi_threads_num;                  // 0到(线程数-1)/线程
数 * car_num
endpos = (int)((mypid + 1) * car_num / mpi_threads_num - 1);
for (i = 0; i < iternum; i++)
{
    // 划分
    if (mypid != 0)
    {
        MPI_Send(&(car_pos[startpos]), 1, MPI_INT, mypid - 1, i * 10 + mypid,
MPI_COMM_WORLD);
    }

    for (j = startpos; j <= endpos; j++)
    {
        car_speed[j] = move(car_speed[j], car_pos[j + 1] - car_pos[j] - 1);
        car_pos[j] += car_speed[j];
    }
    if (mypid != mpi_threads_num - 1)
    {
        MPI_Recv(&(car_pos[endpos + 1]), 1, MPI_INT, mypid + 1, i * 10 + mypid +
1, MPI_COMM_WORLD, &status);
    }
    MPI_Barrier(MPI_COMM_WORLD);
}

//  通过Send 将各个thread中数据集中到thread 0中
if (mypid != 0)
{
    MPI_Send(&(car_pos[startpos]), endpos - startpos + 1, MPI_INT, 0, 10 * iternum
+ mypid, MPI_COMM_WORLD);
    MPI_Send(&(car_speed[startpos]), endpos - startpos + 1, MPI_INT, 0, 10 *
(iternum + 1) + mypid, MPI_COMM_WORLD);
}
MPI_Barrier(MPI_COMM_WORLD);

if (mypid == 0)
{
    for (i = 0; i < mpi_threads_num; i++)
    {
        if (i == 0)
            continue;
        MPI_Recv(&(car_pos[i * (endpos - startpos + 1)]), endpos - startpos + 1,
MPI_INT, i, 10 * iternum + i, MPI_COMM_WORLD, &status);
        MPI_Recv(&(car_speed[i * (endpos - startpos + 1)]), endpos - startpos + 1,
MPI_INT, i, 10 * (iternum + 1) + i, MPI_COMM_WORLD, &status);
    }
    for (i = 0; i < car_num; i++)
    {
        cout << "car:" << i << "  pos  " << car_pos[i] << "  speed  " <<
car_speed[i] << endl;
    }
```

```
}

finish = MPI_Wtime();
t = finish - start;
if (mypid == 0)
{
    printf("Time: %lfs", t);
}
MPI_Finalize();
```

# 实验结果

时间

| 规模\进程数 | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| 100000×2000 | 26.004335s | 17.298415s | 10.488736s | 29.770967s |
| 500000×500 | 34.561371ss | 26.378425s | 14.685196s | 36.741238s |
| 1000000×300 | 36.059020s | 22.607459s | 14.988512s | 28.752256s |

加速比(与串行相比)

| 规模\进程数 | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| 100000×2000 | 1 | 1.503759 | 2.148850 | 0.873334 |
| 500000×500 | 1 | 1.310214 | 2.353484 | 0.940670 |
| 1000000×300 | 1 | 1.595005 | 2.405777 | 1.254608 |

# 实验结论

1. 对于大量数据的蒙特卡洛模拟，使用 MPI 做并行处理的加速比随着数据规模的增大而增大。
2. 由于物理内核限制，4线程时加速效果最好
3. 整体来看加速效果一般，根据模拟结果分析原因应该是迭代次数过少导致模拟结果受车辆的数变化n对模拟结果影响较小。