实验二

学号：SA17011125
姓名：吴燕晶

（一）实验题目：利用 MPI 进行蒙特卡洛模拟

① 实验描述：利用蒙特卡洛算法解决以下问题，并用 MPI 实现并行化

在道路交通规划上，需要对单条道路的拥堵情况进行估计。根据 Nagel-Schreckenberg 模型，车辆的运动满足以下规则：

1. 假设当前速度是 v 。
2. 如果前面没车，它在下一秒的速度会提高到 v + 1 ，直到达到规定的最高限速v_{max}。
3. 如果前面有车，距离为 d，且 d <= v，那么它在下一秒的速度会降低到 d - 1 。
4. 前三条完成后，司机还会以概率 p 随机减速 1 个单位，速度不会为负值。
5. 基于以上几点，车辆向前移动 v（这里的 v 已经被更新）个单位

② 实验要求：

1. v_{max}，p 的值请自行选取，要求v_{max}不低于 10，p 不为 0 即可
2. 实验规模：
   a) 车辆数量为 100 000，模拟 2000 个周期后的道路情况。
   b) 车辆数量为 500 000 模拟 500 个周期后的道路情况。
   c) 车辆数量为 1 000 000 ，模拟 300 个周期后的道路情况。
3. 有兴趣的同学可以在车辆数量很低（如 100）的情况分析道路上的车辆分布情况
4. 实验报告的提交方式与上次相同

（二）实验环境

① 虚拟机：VMware
② 操作系统：Ubuntu16.04
③ 内存：4G
④ 处理器：4

（三）算法设计与分析

1．将 N 辆车根据进程数均匀划分。每个进程控制 N/进程数（以下称之为 len）辆车的位置。

2．针对每个进程初始化每一辆车的位置，且每一辆车的位置是不能重复的。

3．除了 0 号进程以外，其他进程将其控制的第一个车辆的位置告知前一个进程并且 tag=0

4．每一个进程执行 Te 次循环，当前的循环为 k：

对于 len 辆车：

① 将当前车的速度加 1

② 如果当前的车辆是第 len 辆车，并且当前的进程不是最后一个的话，那么当前的进程需要接收来自于后一个进程 tag=k-1 的数据 d。然后计算当前车距离前一辆车的距离。

③ 否则的话直接根据相应的公式计算当前车距离前一辆车的距离

④ 判断当前车距离前一辆车的距离是不是比当前的速度要大，是的话更新当前的速度为车距

⑤ 以 p 的概率将当前的速度减 1

⑥ 更新当前的位置

⑦ 如果当前的进程不是第一个进程，并且计算的是第 1 车的速度和位置的话，就把当前的车的位置发送给前一个进程， tag= k

（四） 核心代码

1．将 N 辆车根据线程数均匀划分。

```
// Allocation and Initial
// initialize the speed of verticals and positions
int *tempv;
int *temppos;

MPI_Status status;
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
MPI_Comm_size(MPI_COMM_WORLD, &myprocs);

int len = N/myprocs;

tempv = (int*)malloc(len*sizeof(int));
temppos = (int*)malloc(len*sizeof(int));
if(temppos!=NULL&&tempv!=NULL){
  memset(tempv, 0, sizeof(int)*len);
  memset(temppos, 0, sizeof(int)*len);
}else{
 printf("malloc failure");
 return;
}
```

2．初始化车的位置，除了 0 号进程以外，其他进程将其控制的第一个车辆的位置告知前一个进程并且 tag=0

```
if(myid==0){
  start = MPI_Wtime();
  //initial position
  for(int i= 0; i<len; i++){
    temppos[i] = i + myid*len;
  }
}else{
  for(int i=0; i<len; i++){
    temppos[i] = i+myid*len;
  }
  MPI_Send(&temppos[0], 1, MPI_INT, myid-1, 0, MPI_COMM_WORLD);
}
```

3．更新每一个周期车辆的位置和速度

```
int k = 1;
// dynamic change the speed and the position
while(k<=Te){

  // change the speed and position of all verticals
  for(int i = 0; i<len; i++){

    // suppose v = v+1
    if(tempv[i]<vmax){
      tempv[i]++;
    }

    // define v = d - 1
    int d;
    if(i==(len-1)&&myid!=(myprocs-1)){
      MPI_Recv(&d, 1, MPI_INT, myid+1, k-1, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
;
      d = d - temp[i];
   }else if(i==(len-1)&&myid==(myprocs-1)){
      d = L - temppos[i];
    }else{
      d = temppos[i+1] - temppos[i];
    }
    d--;
    if(tempv[i]>d){
      tempv[i] = d;
    }
    // random sudden deceleration
    int t = rand()%100;
    if(t<30){
                                                    78,13         57%
    int t = rand()%100;
    if(t<30){
      if((tempv[i]-1)>=0){
        tempv[i]--;
      }
    }

    // new postition
    temppos[i]+=tempv[i];

    // send new position of temppos[0]
    if(i==0&&myid!=0){
      MPI_Send(&temppos[0], 1, MPI_INT, myid-1, k, MPI_COMM_WORLD);
    }
  }
  k++;
}
```

（五）实验结果

用 MPI 实现

运行时间

规模\进程数	1	2	4	8
N=100000 Te=2000	4.509401	2.389431	1.448312	2.094634
N=500000 Te=500	5.679087	2.910477	1.871359	2.294702
N=1000000 Te=300	6.769731	3.581015	2.530697	2.266495

加速比

规模\进程数	1	2	4	8
N=100000 Te=2000	1	1.887228	3.113557	2.152835
N=500000 Te=500	1	1.951257	3.034739	2.474869
N=1000000 Te=300	1	1.890450	2.675046	2.986872

（六）分析与总结

1．从运行时间来看，当进程数为 1 到进程数为 4，随着进程数的增加，运行时间都是不断的减少的。

2．从运行时间来看，当进程数为 8 时，当 N=100000,Te=2000 或者 N=500000,Te=500 时，运行时间却比进程数为 4 的时候，运行时间增加了。而 N=1000000, Te=300 时，运行时间比进程数为 4 的时候减少。这是由于随着进程数的增加进程之间的通讯量也增加了，当规模比较小的时候，进程之间的通信占据了大量的时间，而当规模比较大的时候，进程之间的通信时间相对占比比较少，这时候才体现出了并行的优点。

3．从加速比上来看，其规模和运行时间一样。当进程数为 1 到 4 时，其加速比是不断增加的。但是当进程数到达 8 的时候，前两个规模虽然和串行相比，虽然运行速度还是提高的，但是加速比相对于进程数为 4 的时候，却没有提高。而规模比较大的时候，加速比还是会继续提高。

4．综上所述，在一定的范围内下提高进程数，会使得运行时间减少，提高加速比。但是进程数过多的时候，由于进程之间通行的影响却会使得运行时间增加，加速比减低。