实验 2

实验题目：利用 MPI 进行蒙特卡洛模拟.

实验环境：实验所使用电脑操作系统为 windows10 操作系统，IDE 为 Visual Studio2017，OpenMP 实验环境由 VS2017 提供，MPI 使用 MS-MPI，代码编写和测试均由 VS2017 实现。

算法设计与分析：
首先所有车辆规定好初速度，最大速度，车与车之间的间距，随机降低速度的概率 p.
车辆的行驶和所处情形使用一个结构体来表示，poisition 记录车辆所在位置，distance 记录该车辆与前一辆车的距离，speed 记录该车辆当前的速度。
首先应该明确在串行执行模拟时的情况，车辆在程序中作为结构体数组保存，每一个周期对数组中所有数据进行逐个更新，首先更新距离，随后根据距离来判断速度的可能改变，然后在车辆没有达到最低速度时令其的速度按照概率 p 随机衰减，完成速度数据的更新，最后按照速度的更新值更新该车辆所处位置，每个周期对结构体数组中所有数据进行逐个更新，然后循环执行数个周期，完成模拟。

随后考虑并行化的情况，由于每辆车辆都需要前一辆车辆的位置信息来更新距离，所以按照步长把数据划分给处理器需要耗费很大的通信资源，时间消耗大，转而考虑把数据分块，每个处理器处理某一整块数据，这样一来，每个处理器只需要把所分配块的第一个车辆数据传给负责前一块数据的处理器。在车辆数据大的情况下，极大减小了通信时间消耗。

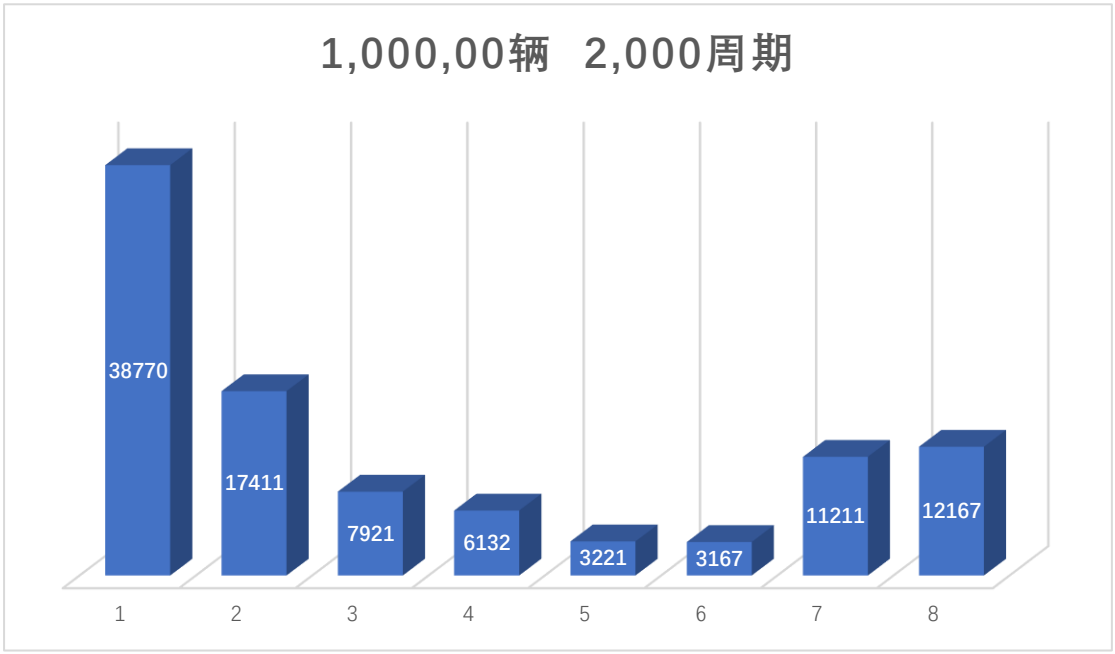
此外在模拟结束后，为了统计相关信息，设计了一个程序块，其中除了第 n-1 号处理器外每个处理器都把自己模拟完成后的数据发送给第 n-1 号处理器，第 n-1 号处理器在收集数据后进行数据汇总和分析后写入到外部文件作为模拟结果的记录。

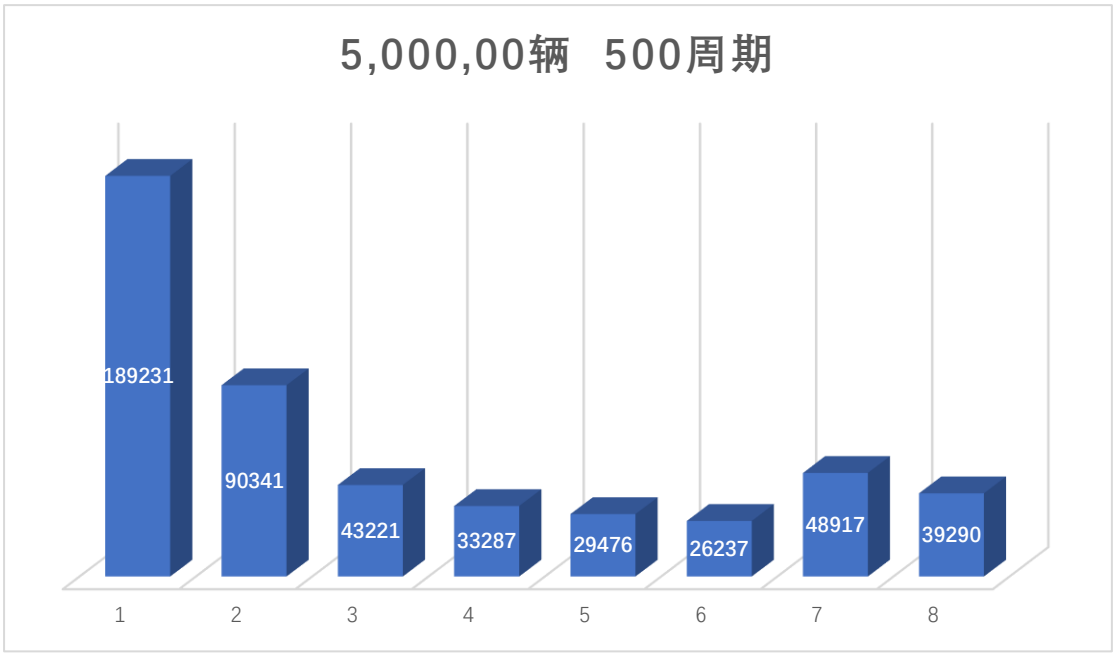
核心代码：

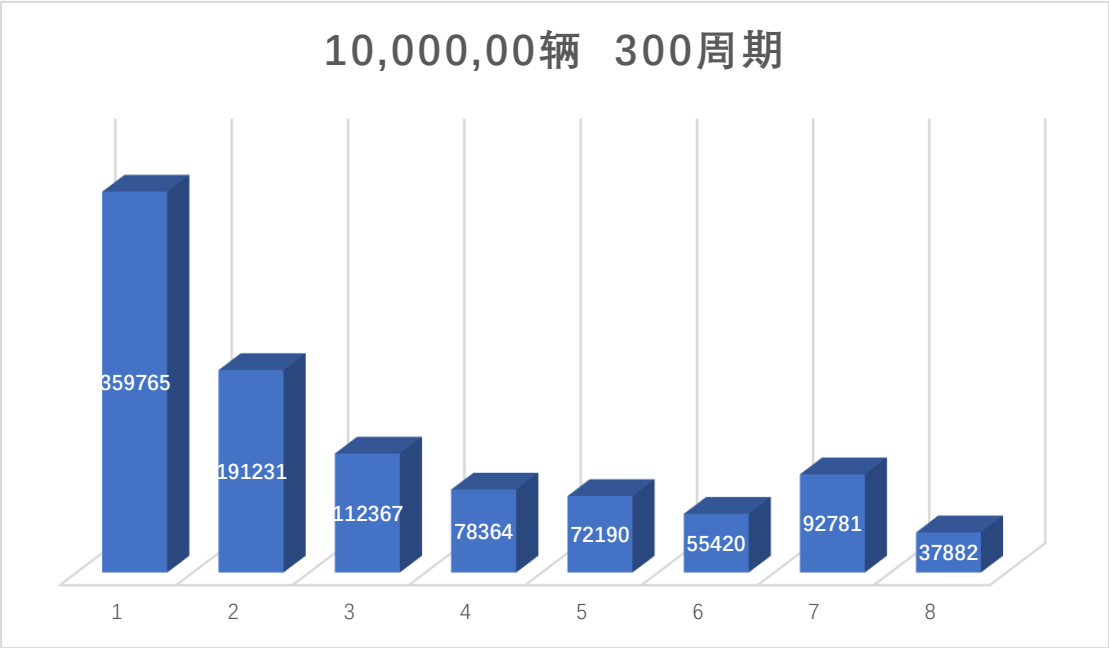
```
for (j = 0; j<2000; j++){
    i = (num_car / numprocs * my_rank);
    if (my_rank != 0)MPI_Bsend(&(cars[i].poisition), 1, MPI_INT, my_rank - 1, j * 10 + my_rank, MPI_COMM_WORLD);

    for (; i<num_car / numprocs * (my_rank + 1) - 1; i++){
        cars[i].distance = cars[i + 1].poisition - cars[i].poisition;
        if (cars[i].distance>cars[i].speed && cars[i].speed<vmax)cars[i].speed++;
        if (cars[i].distance <= cars[i].speed)cars[i].speed = cars[i].distance - 1;
        srand(i*num_car + j);
        if (cars[i].speed>1){
            int r = rand() % 10;
            if (r<p)cars[i].speed--;
        }
        //更新位置
        cars[i].poisition += cars[i].speed;

    }
    if (my_rank != numprocs - 1){
        int temp;
        MPI_Status status;
        MPI_Recv(&(temp), 1, MPI_INT, my_rank + 1, j * 10 + my_rank + 1, MPI_COMM_WORLD, &status);
        //更新距离
        cars[i].distance = temp - cars[i].poisition;
    }
    //更新速度
    if (cars[i].speed<vmax)cars[i].speed++;
    if (cars[i].distance <= cars[i].speed)cars[i].speed = cars[i].distance - 1;
    srand((unsigned)time(NULL));
    if (cars[i].speed>1 && rand() % 10< p)cars[i].speed--;

    //更新位置
    cars[i].poisition += cars[i].speed;
}
```

```
MPI_Barrier(MPI_COMM_WORLD);

if (my_rank != numprocs-1)MPI_Send((cars + my_rank * num_car / numprocs), sizeof(car)*num_car / numprocs, MPI_BYTE, numprocs-1, my_rank, MPI_COMM_WORLD);
else {
    MPI_Status status;
    for(int j=0;j<numprocs-1;j++)MPI_Recv((cars + my_rank * num_car / numprocs), sizeof(car)*num_car / numprocs, MPI_BYTE, j, j, MPI_COMM_WORLD, &status);

    int a;
    for (a = 0; a<num_car; a++){
        fprintf(fp, "%d %d:%d %d %d\n", my_rank, a, cars[a].speed, cars[a].poisition, cars[a].distance);
    }

    for (i = 0; i<num_car; i++){
        count[cars[i].speed]++;
        pos_count[cars[i].poisition / 1000]++;
    }

    int k;
    for (k = 0; k<10; k++){
        fprintf(fp2, "%d\t:%d\n", k, count[k]);
    }
    for (k = 0; k<2 * 8 + num_car * 8 / 1000; k++){
        fprintf(fp2, "%d\t%d\n", k, pos_count[k]);
    }
}


MPI_Barrier(MPI_COMM_WORLD);
```

实验结果：并行度为 4 时测量结果：







运行时间:

规模/进程数	1	2	3	4
100000，2000	37.658	21.109	15.545	13.581
500000，500	42.153	24.103	17.634	14.931
1000000，300	55.639	31.746	22.817	19.440

加速比:

规模/进程数	1	2	3	4
100000，2000	1	1.784	2.423	2.897
500000，500	1	1.749	2.480	2.823
1000000，300	1	1.753	2.439	2.928

实验分析与总结:

由上述模拟可以知道，三种规模的模拟中，最后有大量的车速度为最低速度 1，在尝试在源码中消除随机减速这一对速度的影响后，情况有所好转。可见，随机减速最终会导致比较严重的堵车情况。

在实验中遇到的一个容易犯错误的点是在输出结果之前，应该使用 MPI_Barrier(MPI_COMM_WORLD);语句控制所有进程完成数据处理后再进行数据的发送和输出。