**实验2**

学号 姓名

**实验题目:**

**利用MPI进行蒙特卡洛模拟**

**实验环境(操作系统,编译器,硬件配置等):**

**本次实验运行在NVIDIA DGX（本次实验只使用了CPU：Intel Xeon E5-2698 v4 @ 2.2GHz, 20 cores/40 threads）硬件平台的ubuntu 16.04 RTS系统上，采用了支持C++11标准的gcc 7.3编译器来进行编译工作和cmake 3.9来辅助编译（只是为了程序写起来比较简单）。**

**算法设计与分析(写出解题思路和实现步骤)**

**按照题目要求，在第n时刻中通过第n - 1时刻的结果求出v与d，并将这些结果继续用于下一次计算直到程序运行结束。**

**为了减少通信量以及无谓的内存拷贝，同时尽可能加大并行度，对这个问题进行分析时，可以发现如下特征：第n时刻时第i辆车的速度（以及n步时的路程）仅仅取决于第n - 1时刻时该车的速度与位置，以及这一时刻时第i + 1辆车的位置。因此可以采用如下的编程模型：**

**1、在每一个线程中保存两份本地的v与d数组，用它们交替作为n - 1时刻与n时刻的v与**

**d的数据，这样就确保了线程计算过程中不会出现任何形式的数据拷贝**

**2、每个线程最靠后的车在时刻n的速度依赖于n - 1时刻的下一个线程的最靠前的车的速度，因此这些数据将作为两个时刻之间进程之间互相通信的数据，以确保程序的正常执行**

**核心代码(写出算法实现的关键部分,如核心的循环等)**

**具体的代码都可以在https://github.com/thoh-testarossa/pc-exp上找到**

**Mont Carlo Part：**

**void** montCarlo\_oneStep\_v(**int** pid, **int** numproc, **int** \*d\_sor, **int** upper\_d, **int** \*v\_sor, **int** \*v\_des, **int** vmax, **int** scale, **double** p)  
{  
 **double** p\_generated;  
  
 std::random\_device r;  
 std::uniform\_real\_distribution<**double**> uniform\_dist(0.0, 1.0);  
 std::default\_random\_engine e1(r());  
  
 **for**(**int** i = 0; i < scale; i++)  
 {  
 //Step 1: To determine a new speed value  
 **if**(i != scale - 1)  
 {  
 **if**(v\_sor[i] >= d\_sor[i + 1] - d\_sor[i]) v\_des[i] = d\_sor[i + 1] - d\_sor[i] - 1;  
 **else** v\_des[i] = v\_sor[i] + 1 <= vmax ? v\_sor[i] + 1 : vmax;  
 }  
 **else** {  
 **if**((pid < numproc - 1) && (v\_sor[i] >= upper\_d - d\_sor[i])) v\_des[i] = upper\_d - d\_sor[i] - 1;  
 **else** v\_des[i] = v\_sor[i] + 1 <= vmax ? v\_sor[i] + 1 : vmax;  
 }  
  
 //Step 2: Random slow down  
 p\_generated = uniform\_dist(e1);  
 **if**(p\_generated <= p && v\_des[i] > 0) v\_des[i]--;  
 }  
  
}  
  
**void** montCarlo\_oneStep\_d(**int** pid, **int** \*d\_sor, **int** \*d\_des, **int** \*v\_des, **int** scale)  
{  
 **for**(**int** i = 0; i < scale; i++) d\_des[i] = d\_sor[i] + v\_des[i];  
}

**Process part：**

**int** \*d\_sor = my\_d\_1, \*d\_des = my\_d\_2, \*v\_sor = my\_v\_1, \*v\_des = my\_v\_2;  
  
**for**(**int** time = 0; time < TIMELIMIT; time++)  
{  
 **if** (myid > 0)  
 //Prepare the upper d which will be sent to previous process  
 upper\_d\_to\_previous\_process = time % 2 == 0 ? my\_d\_2[0] : my\_d\_1[0];  
 **if** (myid > 0)  
 //MPI\_SEND the distance of the min\_id obj to the "previous" process  
 MPI\_Send((**void** \*)&upper\_d\_to\_previous\_process, 1, MPI\_INT, myid - 1, time \* numprocs + myid, MPI\_COMM\_WORLD);  
 **if** (myid < numprocs - 1)  
 //MPI\_RECEIVE the distance of the min\_id obj from the "next" process  
 MPI\_Recv((**void** \*)&upper\_d\_from\_next\_process, 1, MPI\_INT, myid + 1, time \* numprocs + myid + 1, MPI\_COMM\_WORLD, &s);  
  
 **if**(time % 2 == 0)  
 d\_sor = my\_d\_2, d\_des = my\_d\_1, v\_sor = my\_v\_2, v\_des = my\_v\_1;  
 **else** d\_sor = my\_d\_1, d\_des = my\_d\_2, v\_sor = my\_v\_1, v\_des = my\_v\_2;  
  
 montCarlo\_oneStep\_v(myid, numprocs, d\_sor, upper\_d\_from\_next\_process, v\_sor, v\_des, VMAX, my\_scale, P);  
 montCarlo\_oneStep\_d(myid, d\_sor, d\_des, v\_des, my\_scale);  
}

**实验结果:**

**实验运行时采用的参数为：vmax = 20，p = 0.514，另外每部车之间在初始会有一个[0, 50]之间的距离分布以及[0, 20]之间的速度分布**

**运行时间**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 规模, 步数\线程 | 1 | 2 | 4 | 8 | 16 | 32 |
| 100000, 2000 | **41.806** | **20.984** | **10.629** | **5.703** | **2.947** | **2.545** |
| 500000, 500 | **52.254** | **26.208** | **13.439** | **6.723** | **3.890** | **2.700** |
| 1000000, 300 | **62.680** | **31.481** | **15.886** | **8.092** | **4.750** | **3.139** |
| 200000, 100000 | **4178.046** | **2093.302** | **1044.811** | **523.649** | **276.516** | **181.035** |

**运行时间**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 规模, 步数\线程 | 1 | 2 | 4 | 8 | 16 | 32 |
| 100000, 2000 | **1** | **1.992** | **3.933** | **7.331** | **14.186** | **16.427** |
| 500000, 500 | **1** | **1.994** | **3.888** | **7.772** | **13.433** | **19.353** |
| 1000000, 300 | **1** | **1.991** | **3.946** | **7.746** | **13.196** | **19.968** |
| 200000, 100000 | **1** | **1.996** | **3.999** | **7.979** | **15.11** | **23.079** |

**分析与总结**

**从上面的实验可以看出，程序的并行性能被发挥得很好，基本都能以理论最大加速比来运行。**

**另外也可以看出来，即使在最下面的那一组实验，步数比其他三组多得多（也就是通信量多得多）的情况下，程序的加速比仍然稳定在一个非常高的水平，不如说还由于数据规模的显著增加而提升了，也可以看出MPI进程间通信的开销其实是很小的。**

**备注(可选)**

**车辆很少时，比如64辆车，在对比试验中可以发现，其实有没有不允许撞车的条件看起来结果都差不多。当然这也可能是因为初始车距比较大的缘故（均值25 > vmax）**