作业报告

虽然对这个任务具体干什么直到做完还是不清楚。但是通过对单线程代码分析后发现对我们所求的Alphas、Betas、Gammas矩阵来说，其中每一个元素都需要遍历矩阵U\_t中的第i行和第j行的所有列，将对应的元素平方或交叉项相加。那么我们可以将U\_t按列划分，在每个进程中对矩阵的某一块计算对应的Alphas、Betas、Gammas矩阵，完成后传回0号进程，将所有的Alphas、Betas、Gammas矩阵相加即得到最终的结果。

因此，思路就很明了了。首先0号进程读入矩阵，划分后传给其他线程。每个线程单独计算一些列对应的三个矩阵。最后将结果传回0号进程，求和。其中一定要注意防止内存泄漏，要在合适的位置写delete [] var;语句。

分别测试了M=N=10, 100, 1000, 2000, 4000的参数设置，均通过了Validation\_mpi.cpp代码的检测。测试平台为Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz,8核心16线程，Ubuntu 18.04，并行运算使用8个进程即将所有核心都占满。检测过程中我发现数据量小的时候运行时间容易产生偏高的情况，看起来像是因为CPU调度而等了一会，大约若干毫秒左右。随着矩阵规模增大，串行和并行用时的比值先上升后下降。规模小时，多核运算带来的提升太小，线程间通信成本相对较。规模中等时，最并行计算的优势，即通信成本没有很大。数据量大是，进程间通信成为了限制条件，多进程速度逐渐下降。我猜这里用openmp会好一点。

另外，我发现OpenMPI在进程间传递数据的速度很慢,即算完之后很久不能完成数据传输。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 计算用时（除去异常值5次平均） | | |  |
| 矩阵规模  （M=N） | 串行/ms  （reduction） | 并行/ms  （reduction-multi） | 串行/并行比值 | |
| 10 | <0.001 | 0.135 | 0.007 | |
| 20 | 0.006 | 0.246 | 0.024 | |
| 40 | 0.048 | 0.145 | 0.331 | |
| 80 | 0.399 | 0.373 | 1.079 | |
| 100 | 0.826 | 0.625 | 1.321 | |
| 200 | 6.718 | 2.535 | 2.650 | |
| 400 | 53.174 | 16.405 | 3.241 | |
| 800 | 423.861 | 94.374 | 4.491 | |
| 1000 | 868.088 | 234.968 | 3.694 | |
| 2000 | 7779.216 | 3037.640 | 2.561 | |
| 4000 | 62313.020 | 23756.320 | 2.623 | |