高斯消元法的Openmp实现

计算机科学与技术
1511186
梁宸

February 25, 2020

1 简介

本次编程作业使用C++语言使用OpenMP实现的多线程版本的高斯消元法程序, 实验处理器环境为intel i7, 操作系统环境为Linux Mint 18.3 64bit.

2 算法设计

本次程序与上次pthread实现的算法设计基本思路相似, 考虑数据依赖性, 算法最外层枚举行的循环不能并行化, 依然要按次序串行执行; 而对于外层循环的每行, 需要做两件事, 一是将这一行所有的元素除以当前行首的元素; 二是将余下所有行按每行行首元素的比例消除. 而其中步骤一一定要先于步骤二完成. 因此这两步操作可以分别并行化.

2.1 任务划分

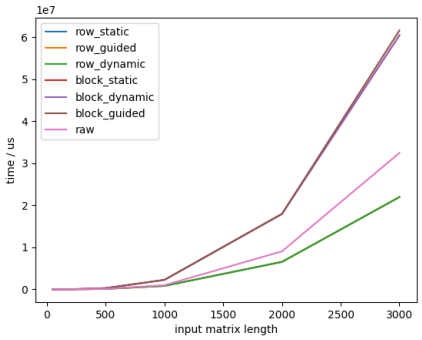
与上次报告中的讨论相似, 考虑分割的任务大小. 若粒度很小, 比如每个线程只对一个YMM寄存器大小的块进行操作, 然后调度, 这样虽然各个线程的负载更均衡, 理论上算法可伸展性更好, 但会有更多的同步通信开销, 并且使得内存访问更加随机不可预测, 影响缓存命中; 当划分任务粒度很大时, 这里比如每个线程一次完成连续的一行, 会更利于缓存命中, 也能减少通信开销, 并且在本程序的情况下, 当处理的矩阵大小相同时, 每个线程执行的指令耗时应该是十分接近的, 因此预测这样按行划分会更符合实际环境.

2.2 线程开销优化

事实上较新版本的OpenMP的实现采用了线程池等模式, 而不是在每次调用并行执行的时候临时创建线程, 所以似乎不用特地去优化线程的创建销毁等额外开销. 经过实践测试, 手动优化线程创建开销, 与编译器自动实现的版本比, 性能没什么显著区别.

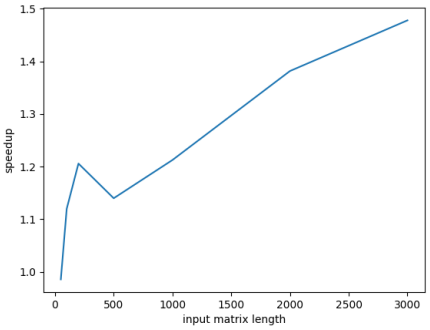
3 结果与分析

首先针对上述两种任务划分方式, 以及OpenMP提供的循环调度的几种方式, 进行测试, 结果如下:



可以看到按行划分的性能远高于按块执行, 这与之前的分析预期一致. 而其中按块划分时, 采用静态的循环调度性能最好, 这是因此静态的调度时内存访问更加连续, 更有利于缓存命中.

采用按行划分, 动态调度, 与单线程的avx版本的加速比如下:



可以看到在输入规模很小时, 因额外的线程创建, 同步等开销, 加速比较低. 为单峰即使在最大测试规模下, 加速比仍有上升的趋势, 但限于本机的计算性能, 难以继续扩大输入规模.

4 问题与改进思路

在当前的测试结果下, 本次OpenMP实现性能要差于上次PThread实现. 原因我认为一是OpenMP不易像PThread一样实现细致的线程调度, 从而更好地针对具

体情况优化, 也难以具体地操控线程的行为. 但无疑与Pthread相比, 编程更为简单.