**CS205 C/ C++ Program Design**

**Assignment 3**

**Name**: 巫晓, **SID**: 11912803

## Part 1. Life is short, show me the code.

为了方便多个版本的迭代，我将我的代码放在了GitHub上

<https://github.com/XiaoLing12138/cppwork/tree/master/P1/CODE>

## Part 2. Result & Verification

在期中项目中，我们需要实现对两个矩阵的乘法。

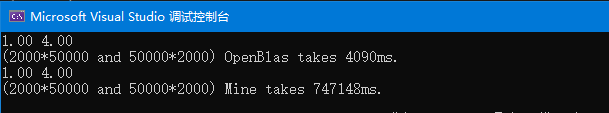
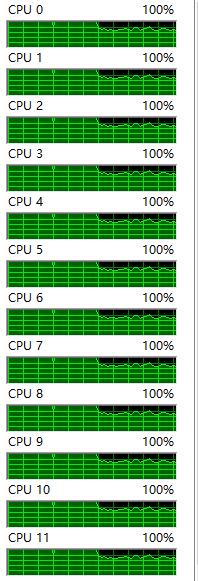
这个功能的原理非常简单，可以类比与上一次作业中的复合点积。但是这次的数据大小依旧十分恐怖，一个矩阵有100M个浮点数。

首先我参考了openCV的矩阵储存方法，使用一个struct来作为矩阵。

在第一个版本中，我实现了最简单的运算，用三层循环嵌套计算。当然，有相应的语句判断矩阵是否合法。显而易见的，这个方法的耗时非常久，100M的矩阵相乘估计得跑到地老天荒。

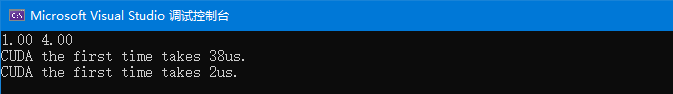
在第二个版本中，我使用了多线程的方法加速计算，并将其与openBlas自带的矩阵乘法函数比较。其结果是非常令人震撼的，同为2000\*50000与50000\*2000的矩阵相乘，我的多线程算法需要747s左右，而openBlas的cblas\_sgemm却仅需要3.9s左右，比我的快将近186倍。不得不感叹，自己与高手的距离还是太远了。

附上CPU占用图和耗时图：



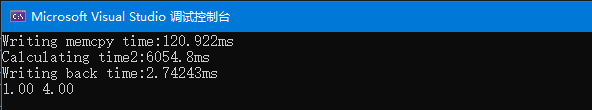
接下来需要考虑如何计算加速。

有了上一次作业的经历，这次我直接使用了cuda加速，用GPU计算而不是CPU，在version3-4中，我发现了一个非常重要的错误，即kernel函数会立即返回每次调用的thread，而如果chrono计时在kernel函数后面，它会立即停止计时，而此时kernel的多线程计算并没有完全跑完，导致得出的耗时（假）非常小，大概40us左右。而二次调用kernel就可以初见端倪，第二次调用只要2us，这是非常诡异的。

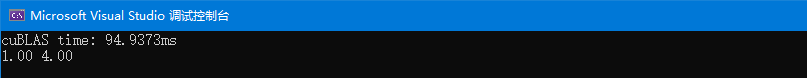


在version5中，我改为使cudaEvent进行计时，它会等kernel结果进行同步之后再停止计时，此时的耗时是正确的。此时，大概耗时6s，与openBlas相比还是慢了一些，说明openBlas做的优化非常足，仅仅使用cuda强大的算力已经不能与其相提并论了。不过相较于我原来使用的方法，已经提升了125倍左右。

此外，我在kernel中，使用了分块运算（block与thread）和共享内存（\_\_shared\_\_），减少了内存的占用，也相应的加快了访问数据的速度。



在version6中，我想到了cuda也有自己自带的blas库，叫做cuBlas。于是我尝试使用cuBlas进行计算，其结果也是非常惊人的，计算2000\*50000与50000\*2000的矩阵乘法仅仅需要95ms左右，比我多线程版的快878倍。这个结果应该是算法优化+GPU强大算力的结果。



如果在继续优化的话，方向应当为使用cuda+算法优化，就像cuBlas一样。

不过不得不说，openBlas能在CPU上跑出这么高的效率，作者实在是吾辈楷模！

## Part 3. Difficulties & Solutions, or others

1. 难点：矩阵的验证

解决：判断一下即可

1. 难点：计算过于费时

解决：使用多线程进行计算

1. 难点：CPU算力有限

解决：使用cuda将矩阵放在GPU上进行计算

1. 难点：float可能存在精度缺失

解决：可以考虑使用double

1. 难点：运算能不能再快一些

解决：使用cuBlas，算法+算力的结合！