**姓名：钱泽凯**

**学号：1190202011**

**作业有两问：**

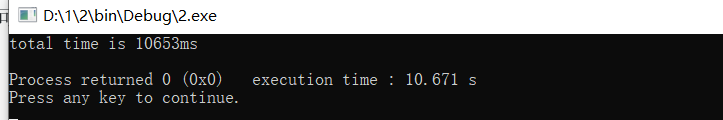
1. **请编写程序，实现两个矩阵相乘**

先编写CPU版程序，然后给出GPU代码每个矩阵用线性数组表示考虑多个block考虑矩阵尺寸不是block尺寸的整倍数

CPU版本：**（具体代码内容见附件中matrix\_mul\_cpu.cpp）**

根据以上矩阵乘法的描述，便可以很快地实现矩阵乘法，三层循环，最内层循环做向量的乘加，最外的两层则做输出矩阵的元素遍历。

设置数组大小为1024\*1024，类型为浮点型，同时加入时间函数，进行计时，结果如图所示：

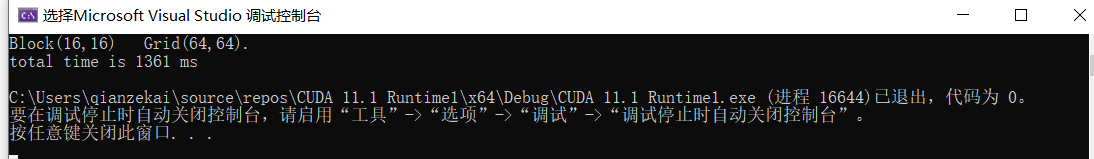


用时10653ms，可见用时还是比较多的。

GPU版本：**（具体代码内容见附件中matrix\_mul\_gpu.cu）**

1. 若考虑多个block考虑矩阵尺寸是block尺寸的整倍数：

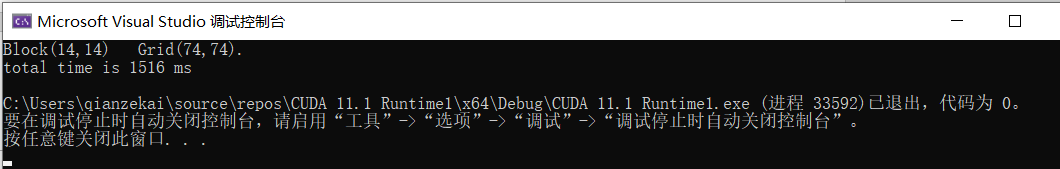
此时在CUDA版本中，我们使用了1024个线程，每个线程执行一行的向量乘加，且每块中含有16\*16个线程，其他地方和CPU版本基本类似。运行结果如图所示：



速度远高于CPU版本，用时1361ms。

1. 若考虑多个block考虑矩阵尺寸不是block尺寸的整倍数：

此时在CUDA版本中，我们使用了1024个线程，每个线程执行一行的向量乘加，且每块中含有14\*14个线程，这样矩阵的尺寸就不是block的整数倍了。其他地方和CPU版本基本类似。运行结果如图所示：



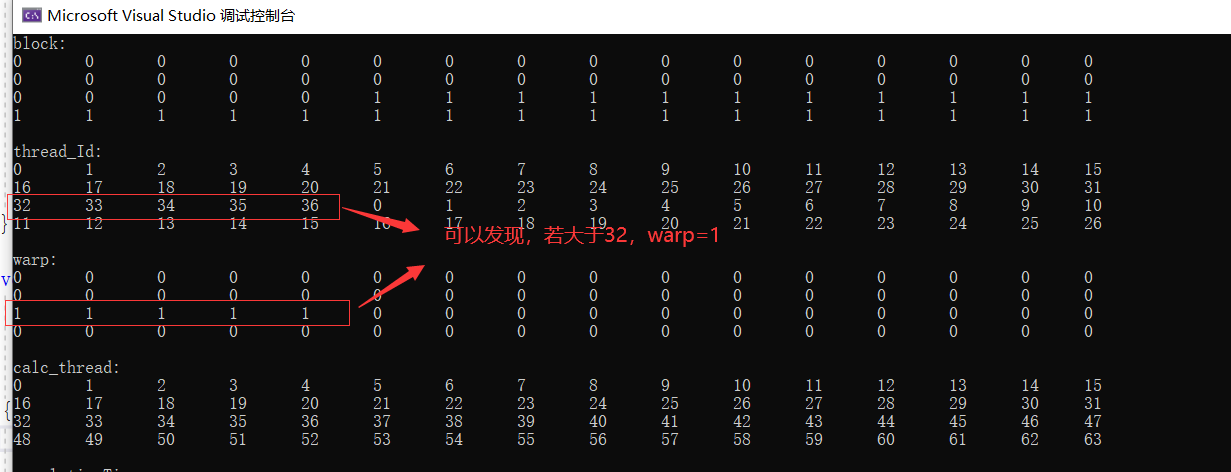
速度稍慢，1516ms，但是同样远快于cpu版本。

1. **理解线程束的调度机制**

**具体代码内容见附件中WarpScheduling.cu。**

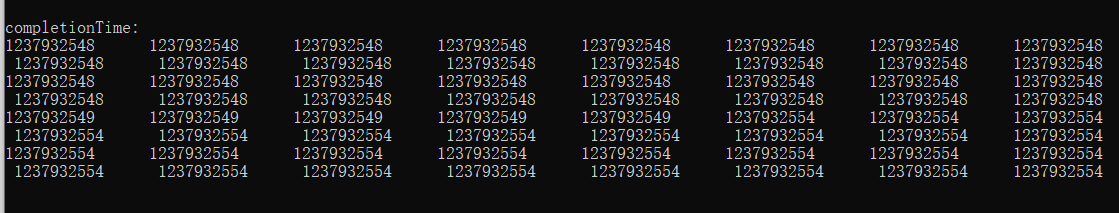
1. 验证warp的线程数量：

当一共64个数据，block=2，每块的thread为37时，程序运行结果如下图所示，可以看到，当threadId大于32时，warp才能等于1，则说明一个warp有32个线程。

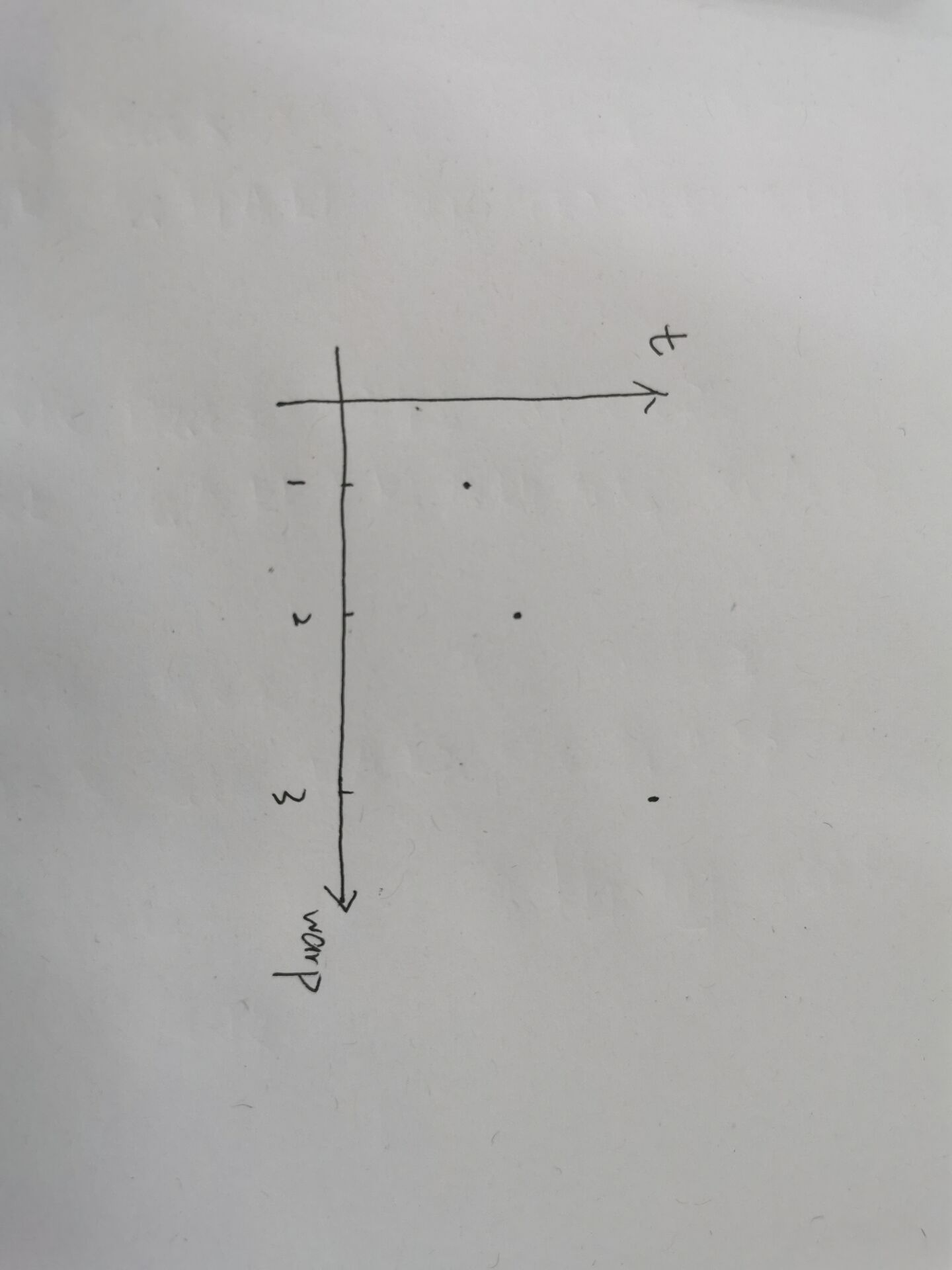


1. 加入计时功能，对warp的调度时间进行输出，并绘出散点图进行分析

当一共64个数据，block=2，每块的thread为37时，对warp的调度时间进行输出如下图所示，



绘制的散点图如下：



通过直观的观察发现，warp是CUDA 中每一个SM 执行的最小单位，函数的执行是以线程束(warps)作为单位，一个warp里的所有线程几乎是同时执行的。每个warp中的线程同时结束 。

1. 变大block和grid的大小会如何？给出对线程束调度机制的理解

一个kernel函数会有一个grid，grid底下又有数个block，每个block是一个thread群组。在同一个block中thread可以通过共享内存（shared memory）来通信，同步。而不同block之间的thread是无法通信的。

CUDA的设备在实际执行过程中，会以block为单位。把一个个block分配给SM进行运算；而block中的thread又会以warp（线程束）为单位，对thread进行分组计算。目前CUDA的warp大小都是32，也就是说32个thread会被组成一个warp来一起执行。同一个warp中的thread执行的指令是相同的，只是处理的数据不同。

基本上warp 分组的动作是由SM 自动进行的，会以连续的方式来做分组。一个SM 一次只会执行一个block 里的一个warp，但是SM 不见得会一次就把这个warp 的所有指令都执行完；当遇到正在执行的warp 需要等待的时候（例如存取global memory 就会要等好一段时间），就切换到别的warp来继续做运算，借此避免为了等待而浪费时间。

实际上，warp 也是CUDA 中，每一个SM 执行的最小单位；如果GPU 有16 组SM 的话，也就代表他真正在执行的thread 数目会是32\*16 个。

举例：

比如说如果有一个block 里有128 个thread 的话，就会被分成四组warp，第0-31 个thread 会是warp 1、32-63 是warp 2、64-95是warp 3、96-127 是warp 4。而如果block 里面的thread 数量不是32 的倍数，那他会把剩下的thread独立成一个warp；比如说thread 数目是66 的话，就会有三个warp：0-31、32-63、64-65 。由于最后一个warp 里只剩下两个thread，所以其实在计算时，就相当于浪费了30 个thread 的计算能力；