# 异构计算试验报告

---实验1：基于CUDA的GPU计算PI值

第一部分：实验环境

OS：Windows 10

CPU：intel(R) Core(TM) i7-10510U CPU@ 1.80Ghz

GPU: NVIDIA GeForce MX250

编译器： cl :v19.29.30133

nvcc: Cuda compilationn tools, release 11.4, V11.4.120

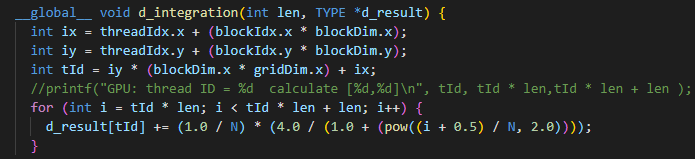
第二部分：实验内容

使用CUDA编程模型并行计算PI值，研究GPU与CPU效率的比较，分析不同GPU线程分块对性能的影响

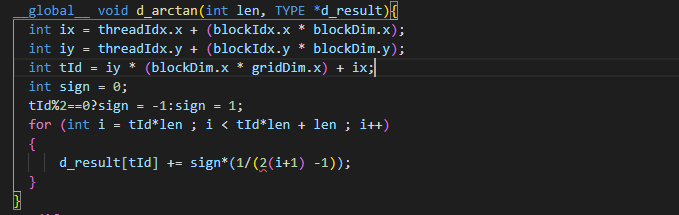
第三部分：实验原理

数学原理为

1. 积分求解



1. 幂级数求解



使用CUDA编程模型，程序执行过程为

CUDAMALLOC -> CUDAMEMCPY(HTOD) ->KERNEL->CUDAMEMCPY(DTOH)

在执行和函数时指定<<<grid，block>>>来控制线程的数量和结构

第四部分：性能分析

#define Ｎ　迭代次数

1. CPU VS GPU

1.1单核比较

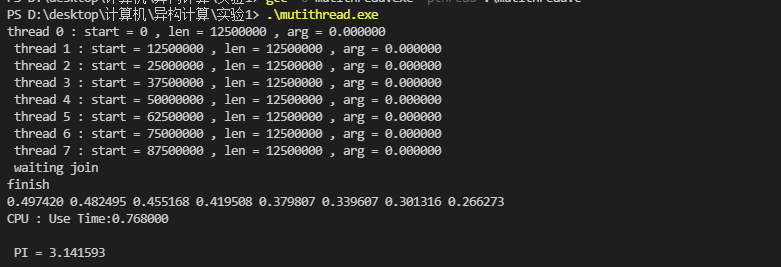
N = (1000\*1000\*100)=100M

CPU（thread = 1）：4.887（s）

GPU(grid = 1,block=1)：用时过长..无法测量

1.2多核比较

首先要发挥出最大的CPU性能，由于机器的CPU是逻辑8核的，所以理论上开启8个线程可以发挥最大性能。所以我写了一个8线程并行的程序来计算，同样循环N=（1000\*1000\*100）次



可以看出新的时间为0.768s快了不少

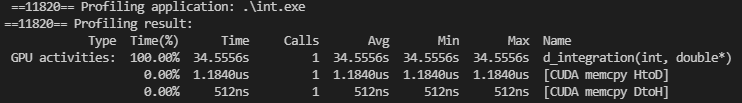
（PS：这里的程序是使用GCC编译的，而CUDA程序NVCC调用的是MSCV的编译器，Windows平台上NVCC不支持GCC，但是结果应该是差不多的）

1. GPU性能分析

接下来就是找出GPU的最高性能了。

首先在grid = 1下探究block里的线程数对性能的影响。这里可以直接用nvprof分析，非常方便。

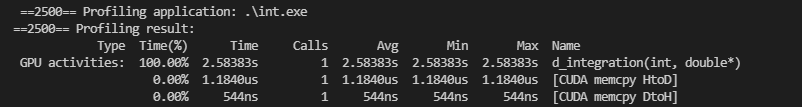
2.1 grid = (1,1,1) block = (10,1,1)



一共用时：34.5556s 线程数：10

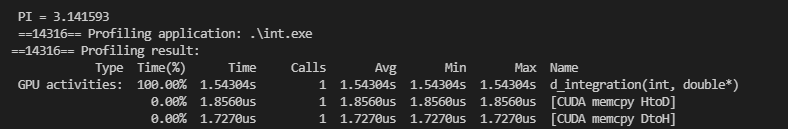
可以注意到，在本程序中，主要的时间是消耗在计算上的，由于没有什么要拷贝的内存，所以瓶颈只会出现在计算上

2.2 grid=(1,1,1) block=(100,1,1)



一共用时：2.58383s 线程数：100

2.3grid=(1,1,1) block=(1000,1,1)



一共用时：1.543s 线程数：1000

每个block的最大线程数限制是1024，这里就不能再增加block了

2.4 ThreadIdx，BlockIdx，BlockDim和GridDim

再每个线程中都可以使用这四个变量，经过实验可以发现

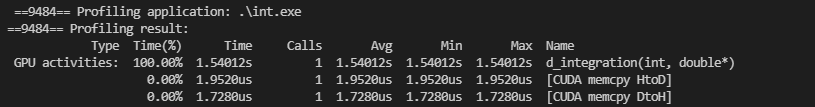
1. 每个变量都有x，y，z三个维度
2. BlockDim和GridDim是所有线程统一的，指示线程的组织结构
3. 线程间只有ThreadIdx和BlockIdx这两个变量不一样，且通过这两个变量可以算出唯一的线程号
4. 对于一个典型的二维Block和Grid的唯一索引计算方式为

     int ix = threadIdx.x + (blockIdx.x \* blockDim.x);

     int iy = threadIdx.y + (blockIdx.y \* blockDim.y);

    int id = iy \* (blockDim.x \* gridDim.x) + ix;

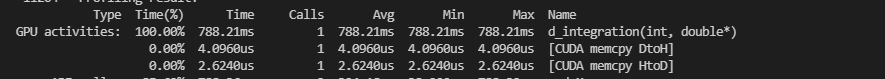
2.5 grid = (1,1,1) block =(500,2,1)



一共用时：1.54s 线程数：1000

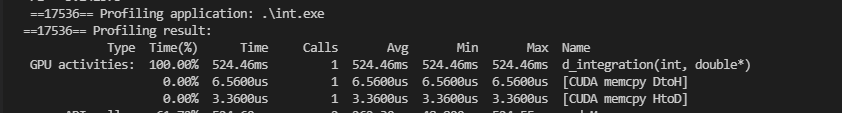
和前面block（1000，1，1）接近

2.6 grid = (2,1,1) block=(500,2,1)

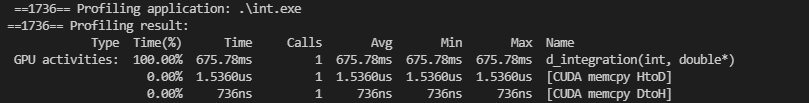


一共用时：0.788s 线程数：2000

2.7grid = (1,3,1) block=(500,2,1)

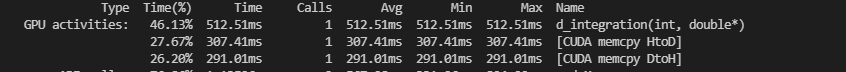


一共用时：0.524s 线程数：3000

2.8grid=(1,3,1)block=(100,2,1)  


一共用时：0.675s 线程数：600

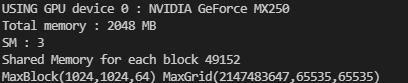
2.9grid=(1000,100,1) block=(500,2,1)



一共用时：1.4s 线程数：100000000

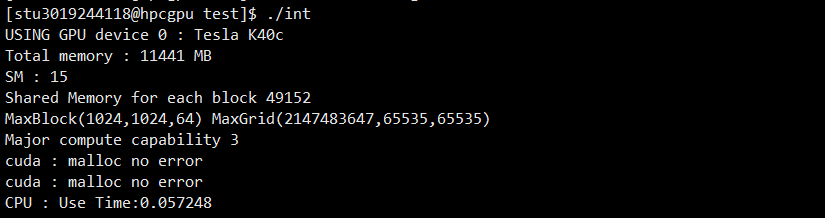
这里已经为每一次循环都分配了一个线程，可以看到内存拷贝的时间都增加了很多，但是执行的时间还是没有怎么变化，那么最快的执行时间大概就是500ms左右了

加速表格(时间ms)



由于GPU的SM一共只有三个，所以预期Grid = 3的时候速度会最快，但实际上超过3之后速度基本也保持不变

放到服务器上试试



服务器有15个SM，果然执行的速度要快了很多，只需要57ms

第五部分：总结

总的来说，GPU的线程数和CPU的线程数有某种程度的相似，在线程比较敖少的时候，增加线程可以看到明显的性能提升，随着线程超过某一个阈值，性能保持稳定。GPU稍微与CPU不同的一点在于，GPU创建的线程好像是没有开销的（就算创建了非常多线程，比如十万个，性能好像也没有影响）对比之下，CPU就不同了，大量的线程会导致操作系统的崩溃，并且性能也会下降很多。

还有一个很神奇的一点，Windows10自带的任务管理器里GPU的利用率在某些情况下是非常不准的，准确的利用率还是需要通过nvidia-smi 命令查看

源码地址

https://github.com/zzzzzec/Heterogeneous-Computing/tree/master/%E5%AE%9E%E9%AA%8C1