基于CUDA的并行矩阵乘法算法 (Tiled Matrix Multiplication)

1 算法说明

CUDA 并行计算框架广泛应用于神经网络、计算机视觉等领域，因为这些领域存在着大量的浮点数矩阵计算，其算法往往可并行性极强，运算量大，非常适合使用 GPU 计算。因此我们的研究也从矩阵乘法展开。

在本次实验中，我们测试了 CUDA C Programming Guide 中给出的矩阵乘积的例子。我们先使用 CPU 实现了形如 $C = A \cdot B$ （测试矩阵的大小为 2048×2048）的矩阵乘法，随后将 CPU 代码移植到 CUDA，将 CPU 上的值传入 GPU，使用 GPU 进行并行计算，并将结果同 CPU 结果对比。

GPU 计算算法相较于 CPU 运算其优化的思路如下：

1. 使用了块并行的思想，将要计算的矩阵 A、B 分解成块大小为 32×32 的子矩阵 (Submatrix)，每一个线程块 (Block) 结构运算一个子矩阵乘法，从而提高乘法运算的并行性；
2. 为了引入共享内存的概念降低 GPU 带宽使用，从而在一个线程块中所有的线程都是共享参数的，从而无需每次计算都从全局内存中获取数据。

1.1 程序输入

本程序无用户输入部分，在 `matrix2.cu` 中的 `main()` 函数中定义了矩阵的大小，在代码的头部全局定义了计算中的块大小。计算矩阵 A、B 中的参数为程序随机生成的单精度浮点数方阵，为了确保计算数据不溢出，我们约定这些参数的取值范围为 $(-1, 1)$。

1.2 程序输出

程序输出包含了 CPU 计算结果和 GPU 计算结果两部分。

1. CPU 和 GPU 分别执行矩阵乘法运算所需时间；
2. CPU 和 GPU 分别执行矩阵乘法运算产生的数值结果上的误差。

1.3 算法过程

1.3.1 初始化

1. 初始化运算矩阵 A、B 和 结果矩阵 C，使用 Cuda 内存管理接口为其在 CPU 内存中分配内存，并为运算矩阵赋浮点数随机数初值；
2. 初始化线程块及对应线程；
3. 初始化计时程序。

1.3.2 GPU 矩阵乘法运算

1. 将数据从 CPU 传输到 GPU 的设备内存上；

2. 将常规 CPU 算法中的两层 for 循环串行运算改为通过 CUDA 中的线程编号，即 `theradIdx.x` 和 `threadIdx.y` 进行并行运算。
3. 计算完成后，需调用 `CudaFree()` 函数来释放内存。

1.3.3 共享内存优化

在以上操作的基础上，我们利用了 CUDA 的共享内存对程序进行优化。

可将 CUDA C 关键词 `__shared__` 添加到变量声明中，这将使这个变量驻留在共享内存中，从而线程块中的每一个线程都共享这块内存，使得一个线程块中的多个线程能够在计算上进行通信和协作。而且，共享内存缓冲区驻留在物理 GPU 上，而不是驻留在 GPU 之外的系统内存中。因此，在访问共享内存时的延迟要远远低于访问普通缓冲区的延迟，提高了效率。

矩阵乘法的并行运算，每次计算矩阵的一块数据。利用共享内存的共享功能，每次将一块数据保存到共享内存中，使得一个线程块中的所有线程同时调用数据计算当前块对应的矩阵乘法结果值。

在进行乘法计算前和乘法计算后都需调用 `__syncthreads()` 函数对线程块中的线程进行同步。

1.3.4 CPU 矩阵乘法算法

使用两层循环计算矩阵中的每一个值。

1.3.5 结果汇总

1. 分别获取两种算法计算出结果的耗时，并将耗时输出；
2. 分别计算两种算法计算出结果的误差，并将误差输出。

2 机器配置

2.1 硬件环境

- GPU: NVIDIA GeForce GTX 1050 (Pascal Architecture)
- CPU: Intel i7-7700HQ

2.2 软件环境

- Microsoft Visual Studio Community 2017 15.9.10
- Nvidia Cuda 10.1.120

2.3 运行环境

- Visual C++ 2017

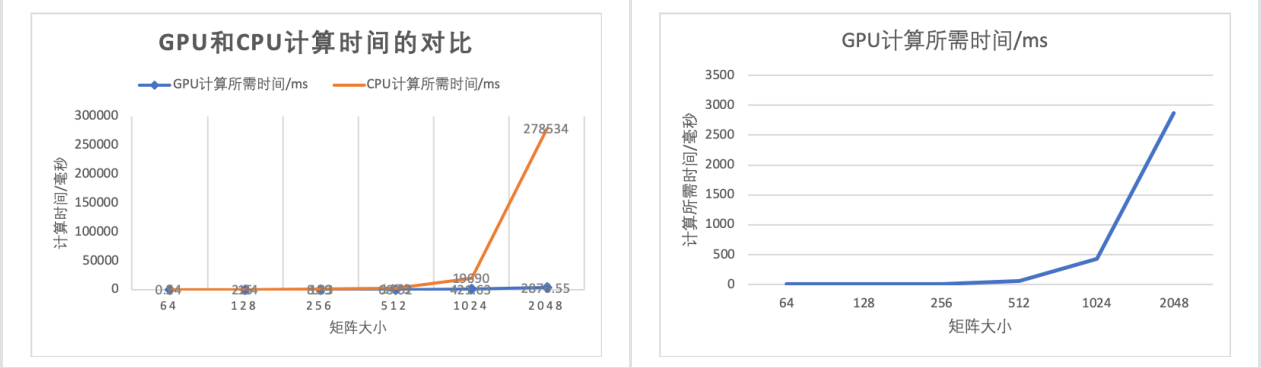
3 结果分析

3.1 CUDA 程序运行结果

本部分将展示使用上述程序在 CPU 和 GPU 环境中分别执行不同大小的矩阵乘法算法所需时间，由于每个矩阵乘法所得结果所有值计算误差绝对值之和均小于 10^{-10}，故我们认为误差可以忽略不计。

矩阵大小	GPU 计算所需时间/ms	CPU 计算所需时间/ms
64×64	0.54	2.0
128×128	2.34	15.0
256×256	8.33	129.0
512×512	60.02	1473.0
1024×1024	421.63	19690.0
2048×2048	2876.55	278534.0

（注：本表统计的是三次运行结果的平均值；GPU 计算时间包含了数据在 CPU 和 GPU 上传递的开销）



左图展示的是 GPU 和 CPU 执行时间的对比图，右图是左图中 GPU 计算时间折线部分的放大。可以从图中得出结论，当矩阵的秩增大时，计算时间也随之增大，这个过程不是线性的，且 GPU 所消耗的时间的递增速率也远远低于 CPU。由此可见，当矩阵的大小越大、可并行的计算量越大，使用 GPU 的算法的优势就越大，加速比也越高。

3.2 不同大小的线程块对运算速度的影响

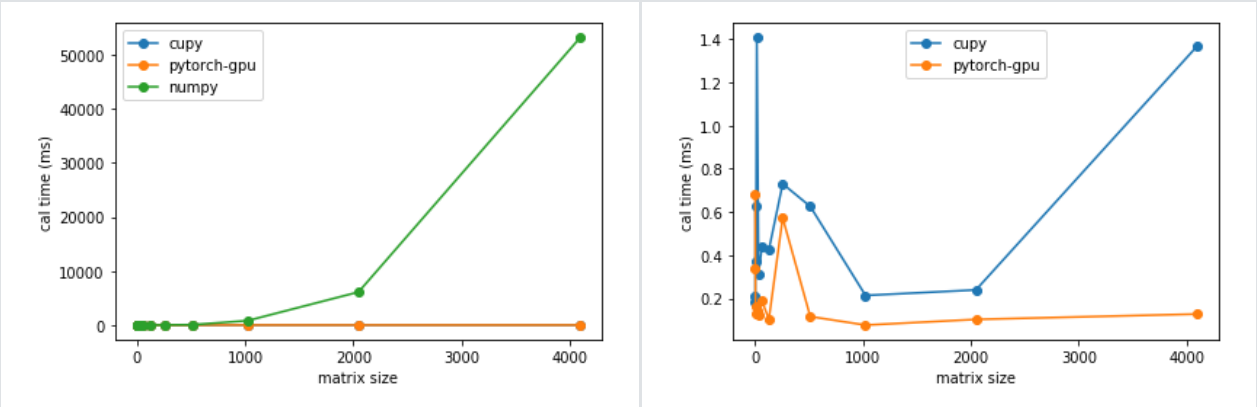
本部分研究了改变线程块 (Block) 的大小对 Cuda 程序计算速度的影响。由于在本程序中，矩阵的大小必须是线程块大小的整数倍，因此，我们测试了线程块为 $4 \times 4, 8 \times 8, 16 \times 16$ 和 32×32 时计算 1024×1024 大小的矩阵乘法的情况，并汇总如下：

线程块的大小	程序运算时间/ms	误差
4×4	2395.0	0.000000000
8×8	685.3	0.0
16×16	505.1	0.0
32×32	425.1	0.0

由于以上数据浮动较大，我们进行了多次测试并取平均值。以上误差均为 0，因为其中的误差小于 10^{-10}，完全可以忽略不计。由以上数据可知，线程块划分得越大，计算的并行性就越高，其运行的速度就越快。而线程块的大小也有一定的限制，在测试环境中，当线程块的大小达到 32×32 之后，其大小便无法再增大，因为超出了显卡的支持范围。

3.3 与 Cupy, PyTorch-gpu, Numpy 等计算框架比较

我们了解到 python 中有使用 Cuda 接口改写 `Numpy` 库的 `Cuda` 框架；同时，几个主流的深度学习框架如 `tensorflow` 和 `PyTorch` 等均含有 gpu 版本，其底层也涉及了 Cuda 的支持，因此我们将它们同 `Numpy` 在矩阵乘法方面的计算性能进行了比较，比较的结果如下：



在上面的图片中，左图是 Cupy, PyTorch-gpu 同 Numpy 执行相同的矩阵运算的执行时间之间的比较，右图则是左图中 Cupy 和 PyTorch-gpu 之间的比较曲线。由此可见，Cupy 和 PyTorch-gpu 这些经过封装和优化的基于 gpu 并行计算的程序接口对运算的提升非常显著。

并且，`Cupy` 提供了简单易用的程序接口，而不需像 CUDA 编程那样了解多线程和内存管理机制，编写复杂的内核函数和底层操作等。相比之下，`Cupy` 上手和入门难度低，学习曲线平缓，并且效果通常优于自己编写的 CUDA 程序。