南開大學
Nankai University

计算机学院

并行程序设计期末报告

CUDA 学习总结

姓名： 高祎珂

学号：2011743

专业：计算机科学与技术

2022 年 6 月 15 日

目录

1 概述

我的 CUDA 学习是在英伟达官方网站上进行的，完成了里面的 CUDA 的学习，庚哥学习过程跟着 CUDA 课程安排走，先是了解了如何进行简单 CUDA 程序的编写，接着有了解了 GPU 与 CPU 之间的内存交换问题，最后通过可视化方法对 GPU 加速有了更清晰的感受，对于课堂的讲述的 GPU 的知识有了更深刻的认识，对于 GPU 与 CPU 之间的函数联系，内存联系也更加清楚了。

2 编写基本的 CUDA 加速程序

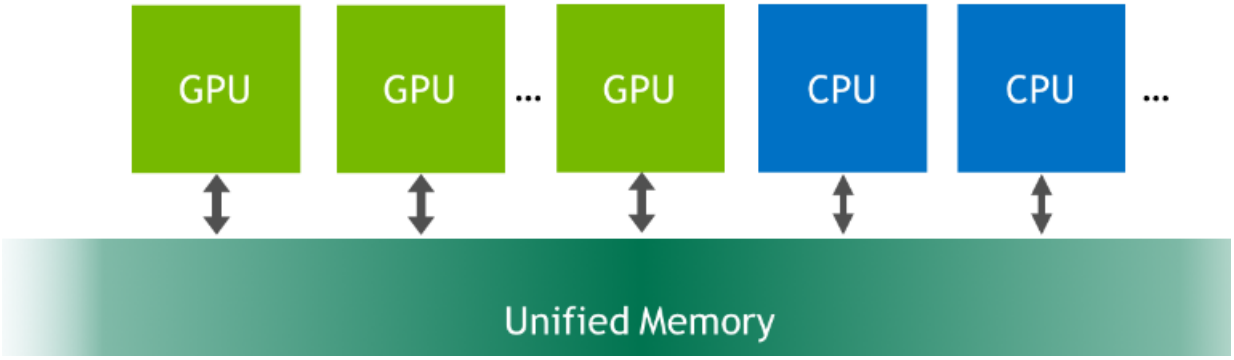
通过这部分的学习了解了 CUDA 的编程实现，CUDA 程序有其自己的命令来识别是 GPU 上运行还是 CPU 运行，即“__global__”，且核函数的声明也有其自己的固定的格式，即“GPUFunction<<<num_of_blocks,threads_of_per_block>>>()”；因为 GPU 是核函数启动方式为异步，还需要加入 cudaDeviceSynchronize()，使得能完成 GPU 函数，如果不加，因为 GPU 上运算量可能回答，最终导致函数还没运行完但是主机函数已经结束了导致程序异常结束，此函数也可以实现同步。

接下来学习了 CUDA 提供的索引变量，有了索引变量函数的编写其实和多线程函数的编写就很相似了。比较重要的变量分别是正在执行核函数的线程（位于线程块内）索引：threadIdx.x，线程所在的线程块（位于网格内）索引：blockIdx.x，块中线程数的特殊变量：blockDim.x。而这里为什么又“.x”呢？是因为这是一维的，CUDA 默认为.x，其实根据数据的特点，我们还可以编写二维甚至三维的网格和块。有了这几个变量我们就可以通过表达式：threadIdx.x + blockIdx.x × blockDim.x 来得到线程索引，这样就和我们编写多线程函数很相似了，编写循环函数过程判断条件的处理也和之前的相似，只不过这里线程总数为 gridDim.x * blockDim.x，利用上网格数据。

后续还学习了 CUDA 的错误处理，根据这个我们可以找到错误所在范围，CUDA 有自己提供的错误处理，我们还可以自己编写错误处理函数进行判断。

3 CUDA 内存管理技术

这部分内容主要是充分利用 nsys 性能分析器进行优化，这部分对于内存的讲解最开始没怎么读懂，主要是对页错误的产生不怎么理解这个概念，后来通过查阅资料对这个知识点理解了，这个部分是对上一部分的深入，主要是讲述了在计算内部 GPU 和 CPU 之间是如何进行的。在此部分介绍了流多处理器（SM），这是 GPU 的处理单元，在核函数执行期间，将线程块提供给 SM 以供其执行。为支持 GPU 执行尽可能多的并行操作，通常应该选择线程块数量数倍于指定 GPU 上 SM 数量的网格大小来提升性能。



接下来是对于统一内存以及页错误的学习。统一内存（UM）是可从系统中的任何处理器访问的单个内存地址空间（参见上图）。这种硬件/软件技术允许应用程序分配可以从 CPU 或 GPU 上运行的代码读取或写入的数据。根据上一部分分配内存调用 cudaMallocManaged() ，替换对 malloc() 或 new 的调用，这是一个分配函数，它返回一个可从任何处理器访问的指针。分配 UM 时，内存尚未驻留在主机或设备上。主机或设备尝试访问内存时会发生页错误，此时主机或设备会批量迁移所需的数据。同理，当 CPU 或加速系统中的任何 GPU 尝试访问尚未驻留在其上的内存时，会发生页错误并触发迁移。这样会使 DtoH 和 HtoD 操作占据了程序运行的绝大部分，大大影响程序进程，通过查阅资料找到了一些解决方法：

1. 如果我们将初始化从 CPU 转移到 GPU，则添加内核不会出现页面错误。
2. 多次运行内核并查看分析器中的平均时间.
3. 使用统一内存预取在初始化数据后将数据移动到 GPU。

在本课程中，他提供了异步内存预取方法，也就是使用统一内存预取在初始化数据后将数据移动到 GPU，CUDA 提供了函数 cudaMemPrefetchAsync() 可以实现我们的目的，通过此方法，进行 nsys profile 分析我们可以看到，明显减少了数据迁移数量，运行时间也得到了优化。通过这个部分的学习，我也认识到在 GPU 编程中对于内存的处理也是比较重要的部分。

4 异步流及可视化

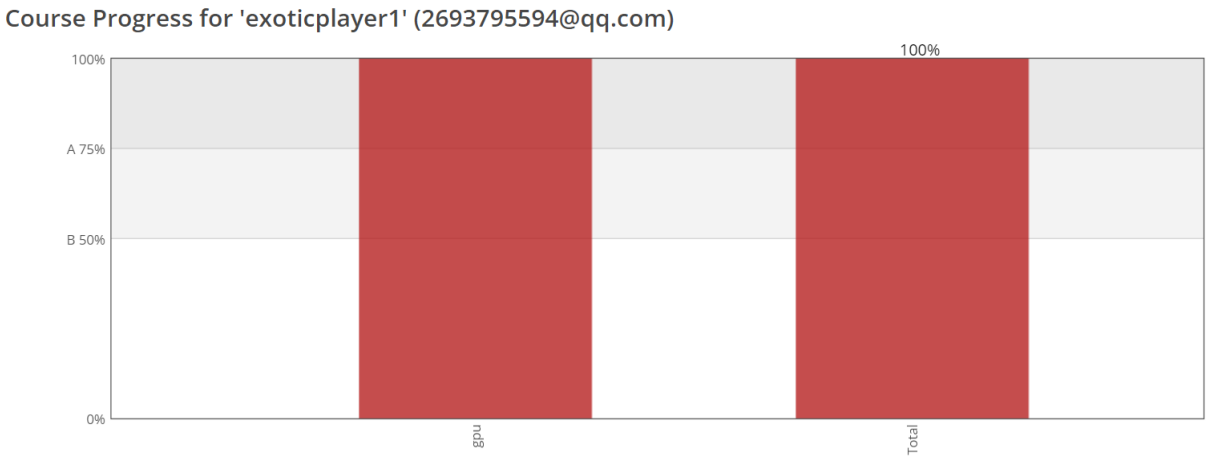
之前的学习在主函数里基本是串行的进行核函数,但是通过本节的学习,我又了解到了并发 CUDA 流知识，我们可以声明非默认 CUDA 流，通过非默认 CUDA 流，可以实现不同核函数的并行，这是新的可以提升性能的方法。通过本实验还进行了程序分析的可视化，这更直观的看到之前的内存预取的作用。

通过可视化，我更加的清楚了数据预取对数据迁移的影响，数据预取大幅度的减少了数据迁移次数，使用核函数进行向量初始化也会在一定程度上减少数据迁移次数，提高性能，可视化操作类似于之前的 VTune 性能分析，这样可以更直接的观察到程序性能。

在 CUDA 编程中，流是由按顺序执行的一系列命令构成。在 CUDA 应用程序中，核函数的执行以及一些内存传输均在 CUDA 流中进行。除默认流以外，我们还可创建并使用非默认 CUDA 流，此举可支持执行多个操作，例如在不同的流中并发执行多个核函数。多流的使用可以为加速应用程序带来另外一个层次的并行，并能提供更多应用程序的优化机会。并且了解了创建非默认 CUDA 流的方法。进行了函数编写，生成了可执行程序，查看了该程序达到的效果，可以看到程序性能分析中，实现了核函数的并行。

5 学习记录


NVIDIA
DEEP LEARNING INSTITUTE
CERTIFICATE OF COMPETENCY
This NVIDIA DLI Certificate has been awarded to
Yike Gao
for the successful completion of
加速计算基础 —— CUDA C/C++
Will Ramey
Senior Director, Developer Programs
June 15, 2022


Course Progress for 'exoticplayer1' (2693795594@qq.com)
100%
100%
A 75%
B 50%
0%
gpu
Total