图形处理器中通用计算流多处理器的设计与实现

王锡淮 16337236　徐立 16337263

2018 年 1 月 4 日

摘要

GPU 的发展速度已经超过了摩尔定律的预期，而且采用的是有别于应用在 CPU 的 Single Instruction Multiple Data（SIMD）的 Single Instruction Multiple Threads（SIMT）模型，SIMT 较 SIMD 模型而言更加灵活，所以 GPU 其并行计算的强大又灵活能力能够有力地加速深度学习的训练过程，GPU 大吞吐量和低平均延时的特性也使得 GPU 能处理海量的数据。所以了解 GPU 的内部架构和具体实现能够帮助掌握 GPU 并行编程的原理和避免在编程时犯下由于对 GPU 结构的不了解而导致的错误。但是目前各大公司对于自身设计的 GPU 的具体实现方法和具体到细节的结构都是保密的，所以我们根据找到的一些概念和从 GPU 编程（Compute Unified Device Architecture，CUDA）中了解到的 GPU 结构，结合自己的理解完成了一个具有通用计算能力的流多处理器的设计与基本实现，这个流多处理器包括几部分：指令缓存区、多线程化的指令单元、数个流处理器、共享存储器和组播网络。同时我们设计了一个简单的指令集来对这个 GPU 加以验证。该流多处理器的基本功能都被实现出来，比如并行的指令传输和并行的存储器访问。这个设计的意义其一在于能够帮助理解 GPU 的结构，其二在于能够为完整地设计 GPU 提供参考价值和意义。但是本实验没有完成专用功能部件，设计的指令也有欠缺，同时没有考虑多个流多处理器之间的交互，这是进一步研究应该完成的。

关键词： 图形处理器，流多处理器，通用计算，单指令多线程

Abstract

The speed of development of the GPU has exceeded the expectation of Moore's Law and uses a Single Instruction Multiple Threads (SIMT) framework that is different from Single Instruction Multiple Data (SIMD) used in the CPU. SIMT is more flexible than the SIMD framework, so GPU's powerful parallel computing power and flexibility to speed up the process of deep learning training, GPU throughput and low average latency feature also enables the GPU to handle massive amounts of data. The GPU's large throughput and low average latency also enable the GPU to process huge amounts of data. So understanding the GPU's internal architecture and implementation can help master the principles of GPU parallel programming and avoid making mistakes in programming due to poor understanding of the GPU's architecture. However, most of the current companies are confidential about how to design their own GPU and the details of the structure. So according to the concepts we found and the GPU structure we learned from Compute Unified Device Architecture (CUDA), combined with our own understanding of the completion of a general-purpose computing stream multi-processor design and the basic realization of the stream multi-processor includes several parts: instruction cache, multi-threaded instruction unit, a number of stream processors, shared memory and multicast network. At the same time we designed a simple instruction set to verify this GPU. The basic functions of the streaming multiprocessor are implemented, such as parallel instruction transfer and parallel memory access. One of the significance of this design is to help understand the structure of the GPU, and the other is to provide a reference value and meaning for the complete design of the GPU. However, this experiment did not complete the special function unit, the design of the instruction is also lacking, and we do not consider the interaction between multiple stream multi-processors, which further researches should complete.

Keyword: Graphics Processing Units, Multiprocessor, General-purpose computing，Single Instruction Multiple Threads

1 背景

近几年来深度学习的迅猛发展是众所周知的，其能够迅猛发展的原因，除了数据量的大量增加，和深度学习算法的发展之外，还有就是硬件的发展，尤其是硬件计算能力的大幅提高，让许多在过去不可能实现的任务得以实现。图形处理器（GPU）的发展便在其中起到了无可替代的作用，其发展速度甚至已经超过了摩尔定律，GPU 并行计算的能力，非常适合于神经网络的训练过程，这是因为一方面 GPU 有大量的核心，即流处理器（Stream Processor），这些流处理器能够并行计算，能够同时进行在神经网络训练过程中多个节点的运算；另一方面是 GPU 相对于中央处理器（CPU）大吞吐量和低平摊延时的结构也能满足训练过程中海量数据进出的要求；还有就是 GPU 是基于 Single Instruction Multiple Threads（SIMT）模型，相对于 CPU 的 Single Instruction Multiple Data（SIMD）模型更加灵活，比如 GPU 支持各个线程跳转到不同的分支 [1]。GPU 在深度学习领域中所用到的功能主要是其通用计算方面的能力，所以我们选择了 GPU 中与其通用计算能力相关的部分进行研究学习，具体就是一个流多处理器（Stream Multiprocessor）的设计与实现。

既然 GPU 在深度学习领域中能发挥这样重要的作用，而深度学习又是当前技术浪潮中的浪巅，我们有足够的好奇，对于 GPU 内部架构与设计的好奇，了解 GPU 的结构对于我们今后学习深度学习以及并行计算打下基础，完成 GPU 中一个流多处理器的设计与实现也能成为我们进一步学习完整 GPU 的具体实现的基石。而对于这个题目本身，目前各大公司对于自己设定的 GPU 的具体实现都是保密的，我们找不到一个完整的实现方法，能找到的只有一些概念和在对 Compute Unified Device Architecture（CUDA）编程的学习中获得的关于 GPU 内部结构的描述。所以在设计我们自己的流多处理器时，每一个部件的实现以及部件与部件之间的关系都要由自己设计决定。因此我们的实现是一个流处理器的完整实现，能提供一种流多处理器的实现思路，具有参考价值与意义。本篇论文要设计并实现一个流多处理器，并设计一个指令集以验证。

2 技术路线

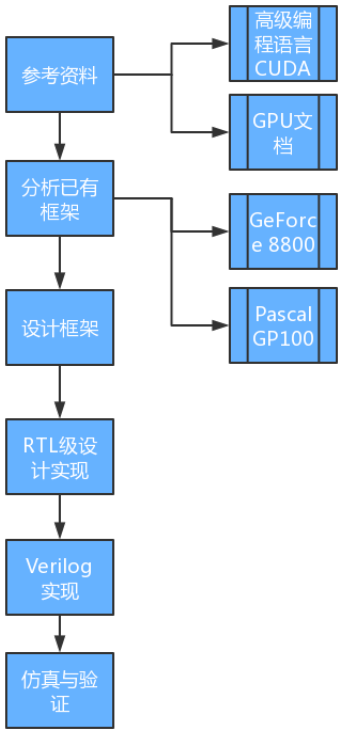

图 1: 技术流程图

技术路线图如图1所示。我们为了解 GPU 整体的模型，我们参考了 Nivida 公司有关 GPU 的文献 [3]，同时也通过了解 CUDA 编程来了解 GPU 的结构 [2]。我们也参考了两个 Tesla 系列 GPU 的流多处理器结构，如图2和图3所示。

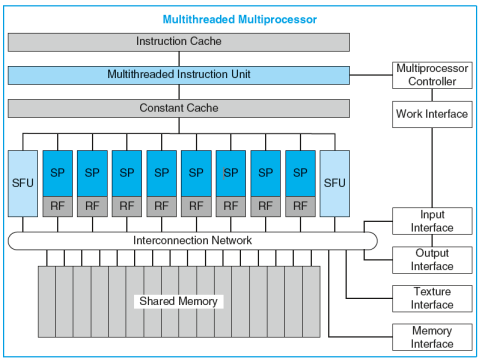

图 2: GeForce8800 的流多处理器结构 [1]

图 3: gp100 的流多处理器结构 [3]

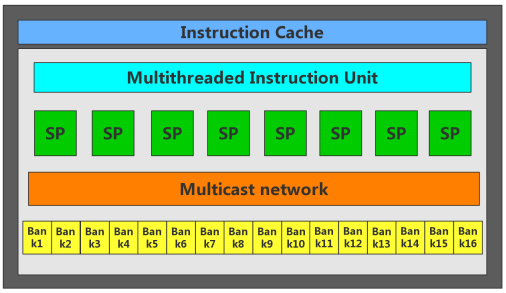

图 4: 本实验的流多处理器结构

然后基于这些，我们设计了自己的流多处理器，如图4所示。之后我们设计了 RTL 级的实现方案，并用 Verilog 语言实现了我们的设计方案，最后仿真与验证了我们设计的指令集。

3 实现过程

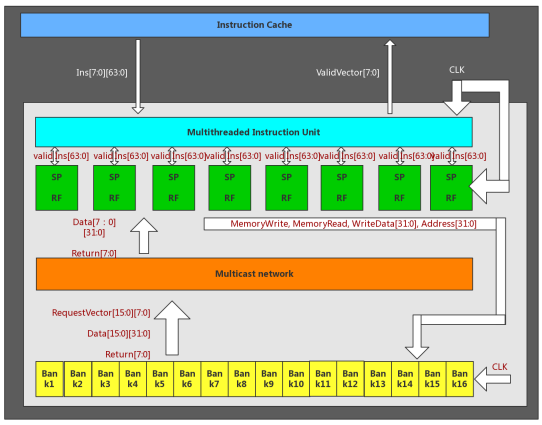

图 5: 本实验的数据通路图

首先从总体流程上讲，如数据通路图图5所示，流多处理器从指令缓存区读取指令，然后经由多线程化的指令单元预处理过后的指令被并行传入 8 个流处理器中，其中在这些指令里，流入当前被占用的流处理器的指令是空指令，随后 8 个流处理器同时处理指令，实现指令的并行处理。流处理器处理某些指令，比如 ld.shared 和 st.shared 时，会进行内存的读写操作，在内存的读取操作时需要通过组播网络来解决一部分的数据冲突。还有其它类型的冲突需要后文讲到的另外的冲突解决方案。

3.1 指令缓存区

本实验的指令集设计为 64 位指令集，具有三种指令形式，如下表1所示。

指令集的内容包括了基本的整数和浮点算术运算、取最大值和最小值、内存的读和写、设置同步屏障等。

指令缓存区（Instruction Cache）的功能是存储指令，并随时准备将指令传给多线程化的指令单元。为了做到只向当前仍然空闲的流处理器发送正常的指令，还需要多线程化的指令单元向指令缓存区发送各个流处理器的占用情况，对于被占用的流处理器，发送的将是空指令。

操作码（10 位）	d	a	b	c

操作码（10 位）	d	a	立即数（32 位）

操作码（10 位）	d（32 位）	保留（22 位）

表 1: 3 种指令格式，其中的 a，b，c，d 等未说明的长度都是 11 位

3.2 多线程化的指令单元

多线程化的指令单元（Multithreaded Instruction Unit）的作用是收集各个流处理器的占用情况，传送到指令缓存区；还有将从指令缓存区得到的指令在时钟边沿到来时分配给对应的流处理器。

3.3 流处理器

流处理器，其结构如图6所示，是处理指令的中心，在没有内存访问冲突的情况下，一般由 4-5 个周期完成一条指令。其中流处理器的组成主要包括一个控制单元，一个数据寄存器，一个算术逻辑单元。这里的算术逻辑单元并不是一般的算术逻辑单元，而是集合了浮点数运算和一些特殊功能的运算单元，因为该流处理器适用于通用计算，浮点运算是必须的，特殊功能可以加速某些常用的特殊运算，符合加速大概率事件的原则。具体上，在第一个周期中处理器接收指令；在第二个周期中长度为 10 位的操作码被传入控制单元中确定控制信号的生成；算术逻辑单元在第三周期进行对应的运算；如果是一般的计算指令，则结果在第四个周期被写回寄存器；而如果是从内存中读取数据的指令，则存储器在第四周期被访问；若存储器成功传回数据，则在第五个周期才进行写回操作，否则流处理器会一直等待存储器的写回数据传入，这里涉及到了后文详细叙述的在内存访问冲突时将并行的访问变成串行访问的方法。

3.4 共享存储器

共享存储（Shared Memory），是在一个线程块（Threads block）或者说写作线程阵列（Cooperative thread block）中的各个流处理器能共享的数据，其生命周期为当前程序运行时间。一个拥有 32 线程数的流多处理器的共享存储块通常被分为 16 个或者 32 个存储体（bank），对应着不同地址的数据的内容，一个简单的共享存储样例的大致结构如图8所示。

存储器访问 流处理器访问共享存储有多种不同的情况需要不同的处理，以下一一列举。

从存储器读取数据 分为三种情况，像图7展示的那样。

- 各线程读取不相同的存储体，这种情况如图7最左图所示，不发生冲突。
- 多个线程读取相同存储体中的同一个数据存储地址。这种情况下，通过了组播网络进行组播，将数据存储地址中的数据复制多份，传输给产生了读取请求的流处理器。
- 读取相同 bank 中的不同内存地址。这种情况没有特别有效的解决方法，只能将并行访问改为串行访问，效率会大大降低。所以在编程时应注意尽量避免这种情况的出现。将并行访问变为串行访问的方法是利用时钟以及向每个流处理器发送是否完成存储器访问的信号，将产生了访问请求的流处理器

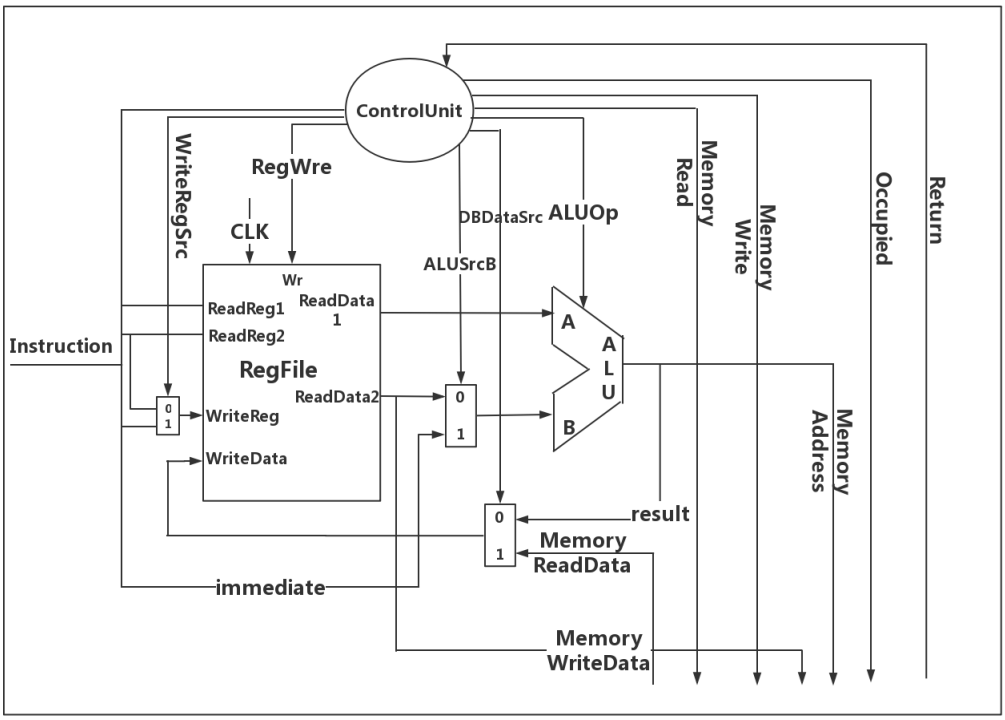

图 6: 流处理器结构

编号放进队列中，每一周期完成一个访问请求。

向存储器写入数据 多个线程同时向同一个存储体写入会发生冲突，这种情况是未定义行为，在硬件上也没有特别有效的解决方法。只能在编程的时候尽量避免，防止这种的情况的出现。

3.5 组播网络

组播网络（Multicast Network），可以数据复制多份传输给产生了读取请求的流处理器。主要用来避免上文中提到的多个线程访问同一个存储体中同一个数据存储地址发生冲突。

3.6 专用功能部件

一个完整流多处理器中还应包含双精度浮点数计算单元和专用功能部件（SFU），由多线程化的指令单元预处理后传输。双精度浮点数的运算可以仿照单精度浮点数的运算。而特殊函数，比如三角函数、指数函数、对数函数、开方等一般不能直接求出精确数值的运算，可以用 Coordinate Rotation Digital Computer（CORDIC）算法做近似计算。这个算法使得各种函数的计算只涉及移位和加法，易于在硬件上实现 [5]。实现这样的 SFU 能够加速大概率事件。

4 效果评价

我们了解了 GPU 的一个流多处理器的结构，以及使用 CUDA 调用 GPU 并行计算的一些思想与方法，同时设计并实现了一个 GPU 中组成一个流多处理器需要的基本模块，比如流处理器，共享存储器，指令寄存器，并行指令单元等，也实现了浮点运算器，包括浮点加法和浮点乘法器。这样也就实现了指令的并行传输，指令的并行处理，组播网络，数据的并行传输等功能，

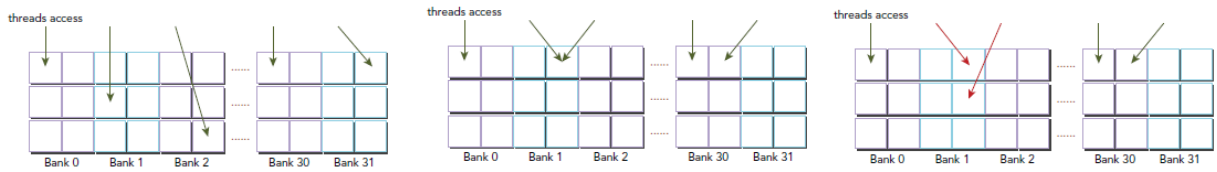

图 7: 存储器访问情况 [2]

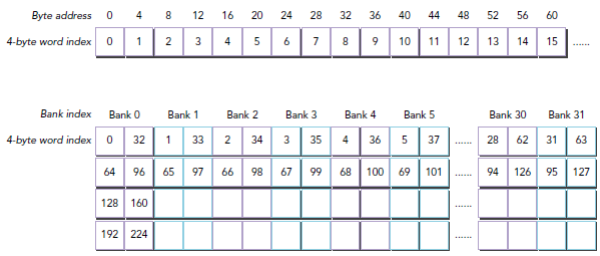

图 8: 共享存储样例 [2]

基本可以用来完成在图形处理器上的通用计算功能。尤其值得一提的是如何处理在访问内存共享单元时遇到的冲突，包括在硬件设计时作为硬件设计者如何将并行的命令转换为串行执行，以及作为编程人员在使用硬件时如何避免访问同一个 bank 的不同内存单元的方法；还有就是如何将指令传输到空闲的流处理器，这里体现了许多并行的思想。总的来讲，我们完成了预期目标。

但是我们的不足在于实现的指令集并不完善，同时没有实现 SFU 的功能，这是进一步的设计可以考虑的。还有本设计并没有考虑多个流多处理器之间的交互，这也是在 GPU 的设计中要重点考虑的。

参考文献

[1] Nickoll,J.，D.Kirk. *Graphics and Computing GPUs[M].*

[2] Cheng,J.,M.Grossman,T.McKercher. *Professional CUDA C Programming* [*M*]. Indianapolis: John Wiley & Sons, 2014：204-212.

[3] *GP100 Pascal Whitepaper[DB/OL].*2016. `https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf`

[4] *IEEE 754-2008，IEEE 浮点数算术标准 [S]*

[5] Jack E. Volder *The Birth of CORDIC.* Journal of VLSI Signal Processing 25: Kluwer Academic Publishers, Netherlands, 2000: 101–105.