CUDA微架构与指令集（1）-微架构概述

https://zhuanlan.zhihu.com/p/158917926

None Mon May, 24 03:19

今天开一个新的系列，主要聊CUDA微架构和指令集，包括一些特定指令的使用场景和性能状态，也有一些warp调度和指令执行逻辑上的性能影响等等，概括说来就是一些指令级别的优化思路和方法。不过我没在NVIDIA工作过，也没做过硬件设计，所以这些琐碎的知识点大多来自我多年做算法程序实现和性能优化上的经验，也包括对CUDA官方文档的理解，当然也有很多细节来自NVIDIA的论坛和stackoverflow上的一些讨论。因为看不到设计文档，NV也不可能公开这些信息，所以很多问题并没有确切的信息，以个人理解为主，仅供参考。

首先，要明确什么是CUDA的微架构和指令集。CUDA的微架构（Micro-architecture）一般指的是Streaming Multiprocessor（简称**SM**）的架构，并不包括外围的一些辅助功能，比如memory控制单元，copy engine之类。指令集（Instruction Set Architecture，简称ISA）就不用解释了，就是GPU执行的机器码的形式。CUDA的机器码也被称为SASS，有人说是Streaming ASSembly，存疑。CUDA还有另外一个构建在SASS以上的虚拟中间代码指令集叫PTX。由于GPU的指令都是从SM发出的，所以微架构与指令集之间有很强的关联。微架构的设计就决定了指令集所支持的指令形式，以及大部分的指令throughput和latency。但也有一些例外，比如仿存指令虽然是SM发出，但却需要外部单元的配合，因此仿存指令的带宽和延迟其实并不都是微架构决定的。

这里还要说明一下为什么要研究CUDA微架构和指令集。每个程序都有两个重要的方面，功能和性能。功能就是能接受什么输入，进行什么处理，然后完成什么输出。而性能，就是让程序完成设计功能所需要的成本。通常说的性能多数是指时间成本，所花时间越短则性能越好。更广义的性能也可以包括空间成本（如内存占用），运行成本（比如能耗），升级和维护成本，甚至可移植性好不好等等。对指令集的理解和运用，直接决定了程序的功能和性能最终能够达到的状态。程序写的好不好，有的是与架构无关的算法或流程设计问题，也有的是与架构相关的优化或适配问题。而这里的架构相关，则有相当一部分会体现在微架构和指令集上（当然，driver等也会有体现）。注意这里没有提到programming model和compiler的问题，也没有提到driver更上层的API，因为这些是用户运用指令集的手段，不是最终形式。当然，好的programming model和toolchain可以让人更高效的完成高质量的程序编写，也是大多数人运用这些功能的方式。但对微架构和指令集的准确理解和运用，是合理有效的使用这些工具的重要技术支撑。因此，要想对CUDA程序进行更深入细致的优化，了解指令集和微架构是CUDA编程进阶的必要课程。

其次，微架构和指令集是有演进的。它涉及很多问题，设计需求，市场需求，工艺状态，甚至软件生态的支持等等，所以架构总是在不断变化的。CUDA用Compute Capability（有时候简称**CC**）来区分各个微架构，比如5.0，6.1，7.5等等，通常用`SM_50`，`SM_61`，`SM_75`（有时也不用下划线）来表示。CC有大版本小版本（major和minor），大版本一般有代号，比如3.x称为

Kepler，包括3.0，3.5，3.7等，5.x称为**Maxwell**，包括5.0，5.2，5.3等，6.x是**Pascal**，包括6.0，6.1，6.2等。当然也有例外，Kepler以前的不太熟，比较近的7.x并没有一个统一的代号，7.0是**Volta**，7.5是**Turing**。最新的**Ampere**新架构卡A100是8.0，但是“x.0”是近两代Tesla计算卡的惯常位置，将来的GeForce消费卡可能会是8.1，8.2，或者8.5之类。有些CC只出现在Tesla或Quadro系列中，比如6.0就好像只有P100，7.0只有V100。有些CC只在文档里看到过，比如7.2，也不知道是怎么回事，也许是太监了……（补注：今天偶然看到有款jetson的产品Jetson AGX Xavier是7.2，果然是我out了……）

用户可以在NV官网的CUDA GPUs列表里查看各个卡对应的SM版本。这个列表并不全，比如常见的笔记本显卡MX150、MX250之类就不在列表里，其他如GTX1660之类好像也没有。NV好像也没有提供特定的工具来查看当前设备的CC，至少我试过nvidia-smi, nv控制面板中的系统信息，设备管理器等，都只能查到显卡型号，没法查Compute Capability，也是奇怪的很。当然，要真想查也是可以的。一是用CUDA的API函数cudaGetDeviceProperties，二是跑一遍CUDA sample里的 `deviceQuery` ，这两种方法都要先装CUDA SDK。不知道copy一个编译好的deviceQuery能不能运行，存疑，也许要用那个driver api的版本 `deviceQuerydrv` 。所以如果你没能在官方列表找到你的卡，又没有装CUDA SDK，估计就只能上网搜或者问啦！不过按现在的形势，市面上能买到的NV的独立显卡，几乎可以肯定是支持CUDA的。

Compute Capability决定了这个设备支持一些什么样的编程功能（参考Features and Technical Specifications），也包括每个SM内一些资源的数目等等。CC和SASS指令集之间有比较直接的对应关系，大版本之间的差异往往还是比较大的，同一个大版本内的小版本的指令集几乎是不变的（早期的可能比较乱，近一些的Volta和Turing算是例外），具体可以参考CUDA Binary Utilities中提供的Instruction Set Reference。当然，这个东西也就勉强看看，靠这点信息是不可能学会的，这也是为什么要开这个系列的原因。另外，小版本之间可能会有一些功能配比上的微调，主要是一些特定功能单元（F16、F64或TensorCore之类）相关。所以很多指令的throughput和latency都与CC有关，这也是一些微架构相关的细致优化比较敏感的部分。注意这里说的指令集是最终的机器码SASS指令集，不是中间语言PTX。

这里需要强调的是，**Compute Capability只与设备的SM架构版本有关，与设备绝对算力的强弱无关**。因为CC主要指的是**微架构**，多数是SM内的功能。而一款芯片除了SM内部设计以外，还有SM的个数，还有很多其他与SM配套的功能单元，特别是memory控制方面的单元，这些东西对性能的绝对值影响往往更大。比如一款常见的笔记本显卡MX150的CC是6.1，Pascal架构，只有2G显存，3个SM（384个CUDA Core），64bit的显存位宽，也就比亮机卡好那么一丢丢（好像也没有这么弱……比多数核芯显卡还是强的）。而著名的GTX 690战术核显卡才是3.0，土豪专享的TITAN Black和双芯TITAN Z也不过是3.5，真正能爆的核弹原型GTX 590甚至还是2.0。一般说来，同一系（**系**一般指的是显卡型号中的代数，比如750，780是7系，980是9系之类）的多数卡CC是相近甚至相同的。因为一代SM架构设计完成后，可以通过控制SM的数目、显存容量、显存带宽等来组合成高中低端不同层次的产品，这样虽然SM的架构是相同的，芯片的性能

却有高低之分。令大家羡慕不已的老黄刀法就体现在这些地方。当然，有时候为了清库存或者与对手竞争，也会临时从上一代甚至上上代架构里拉出一个异类，比如7系多数是Kepler架构，但730是Fermi架构，据说是630的马甲卡。也有一些时候也许是新架构成熟度不够，先放在前一代的中低端卡来试试水，也是有可能的事情。比如14年发布的750/750Ti，还有很多8系的笔记本显卡都是Maxwell架构，但Maxwell的性能旗舰GTX 980Ti/TitanX到2015年才发布。所以显卡的"系"和CC之间也不完全是对应的。有兴趣的朋友可以去wiki上搜list of NVIDIA Graphics Processing Units，有详细的架构配置和发布时间信息，还是能品出很多奇妙的味道来~ 另外呢，芯片的性能也不是一个定值，散热设计等也会影响芯片的运行频率，这也是不同厂商的显卡差异所在，即使它们用的是同样的芯片。

从设计上来讲，芯片的各个功能单元的配比调整还是比较容易的，但SM的改动则相对更复杂和困难，工作量更大，周期也更长，设计上一般都是尽量公用的。当然显卡的SM与CPU的core类似，也是可以屏蔽掉一部分。这样也许可以在完整版的基础上，根据坏掉的SM的个数来挑出不同级别的芯片来。很多人都知道，同一系的i3、i5和i7有可能是从同一个wafer上筛选出来的，GPU芯片应该也有这种操作，但一般可能会分得更细。比如Turing系列的芯片很多，高端的2080Ti用的TU102，2080用的TU104，2070的TU106，还有更低端的1660用的TU116之类。我猜测，这里的不同的代号应该是从不同wafer上下来的，die的size都不一样，应该没法靠刀法来阉割。而TU116又有细分的TU116-200，TU116-300，TU116-400等等，这个也许是用刀法能搞定的。这里所有TU开头的芯片的CC都是7.5，但是从整个芯片的性能到SM内的一些具体功能单元数目（最典型的是TensorCore，RayTracing单元，还有F64单元等），其实都是可能有差异的。所以CC本质上就只是一个设计上粗略的区分，既不决定所有功能，更不决定绝对性能。

微架构的概述就先说这么多，其实说这么多也就说了版本的问题，大部分具体的功能还是会体现在指令集上。所以下次再大致概述一下指令集，然后就可以开始具体找一些实例来分享了~