湖南大學

HUNAN UNIVERSITY

计算机设计

学生姓名	周珍冉
学生学号	201708010610
专业班级	智能 1702
指导老师	吴强
完成日期	2019.12.9

Tesla GPU 架构分析

Fermi、Kepler、Maxwell、Pascal、Volta、Turing

一、 GPU 介绍

1. 起源

GPU 缩写为 Graphics Processing Unit，一般称为视觉处理单元。GPU 被广泛用于嵌入式系统、移动电话、个人电脑、工作站和电子游戏解决方案当中。现代的 GPU 对图像和图形处理是十分高效率的，这是因为 GPU 被设计为很高的并行架构这样使得比通用处理器 CPU 在大的数据块并行处理算法上更具有优势。

NVIDIA 公司在 1999 年发布 GeForce 256 图形处理芯片时首先提出 GPU 的概念。从此 NVIDIA 显卡的芯片就用这个新名字 GPU 来称呼。GPU 使显卡削减了对 CPU 的依赖，并执行部分原本 CPU 的工作，尤其是在 3D 图形处理时。GPU 所采用的核心技术有钢体 T&L、立方环境材质贴图与顶点混合、纹理压缩及凹凸映射贴图、双重纹理四像素 256 位渲染引擎等，而硬体 T&L 技术能够说是 GPU 的标志。

2. 工作流程

- ✧ **顶点处理**：这阶段 GPU 读取描述 3D 图形外观的顶点数据并根据顶点数据确定 3D 图形的形状及位置关系，建立起 3D 图形的骨架。在支持 DX8 和 DX9 规格的 GPU 中，这些工作由硬件实现的 VertexShader（定点着色器）完成。
- ✧ **光栅化计算**：显示器实际显示的图像是由像素组成的，我们需要将上面生成的图形上的点和线通过一定的算法转换到相应的像素点。把一个矢

量图形转换为一系列像素点的过程就称为光栅化。例如，一条数学表示的斜线段，最终被转化成阶梯状的连续像素点。

- ✧ **纹理帖图**：顶点单元生成的多边形只构成了 3D 物体的轮廓，而纹理映射（texturemapping）工作完成对多变形表面的帖图，通俗的说，就是将多边形的表面贴上相应的图片，从而生成“真实”的图形。TMU（Texturemapping unit）即是用来完成此项工作。
- ✧ **像素处理**：这阶段（在对每个像素进行光栅化处理期间）GPU 完成对像素的计算和处理，从而确定每个像素的最终属性。在支持 DX8 和 DX9 规格的 GPU 中，这些工作由硬件实现的 Pixel Shader（像素着色器）完成最终输出，由 ROP（光栅化引擎）最终完成像素的输出，1 帧渲染完毕后，被送到显存帧缓冲区。

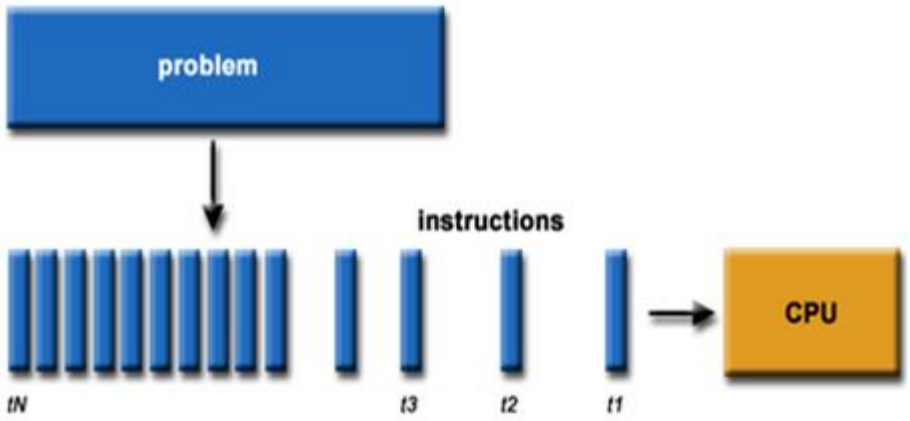
3. GPU 与 CPU

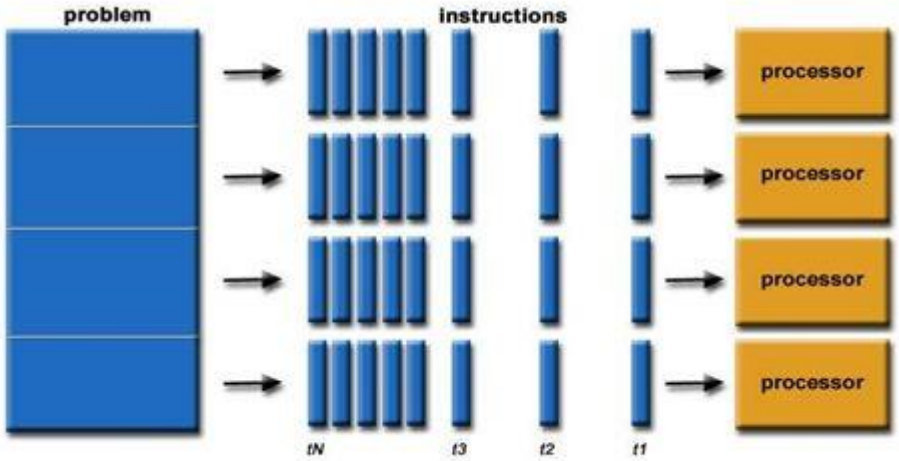
对于 GPU 来说，它的任务是在屏幕上合成显示数百万个像素的图像，也就是同时拥有几百万个任务需要并行处理，因此 GPU 被设计成可并行处理很多任务，而不是像 CPU 那样完成单任务。

因此 CPU 和 GPU 架构差异很大，CPU 功能模块很多，能适应复杂运算环境；GPU 构成则相对简单，目前流处理器和显存控制器占据了绝大部分晶体管。

CPU 中大部分晶体管主要用于构建控制电路（比如分支预测等）和 Cache，只有少部分的晶体管来完成实际的运算工作。而 GPU 的控制相对简单，且对 Cache 的需求小，所以大部分晶体管可以组成各类专用电路、多条流水线，使得 GPU 的计算速度有了突破性的飞跃，拥有了更强大的处理浮点运算的能力。

从硬件设计上来讲，CPU 由专为顺序串行处理而优化的几个核心组成。另一方面，GPU 则由数以千计的更小、更高效的核心组成，这些核心专为同时处理多任务而设计。





4. GPU 加速技术

A． CUDA

为充分利用 GPU 的计算能力，NVIDIA 在 2006 年推出了 CUDA（ComputeUnifiedDevice Architecture，统一计算设备架构）这一编程模型。CUDA 是一种由 NVIDIA 推出的通用并行计算架构，该架构使 GPU 能够解决复杂的计算问题。它包含了 CUDA 指令集架构（ISA）以及 GPU 内部的并行计算引擎。开发人员现在可以使用 C 语言来为 CUDA 架构编写程序。

B． OpenCL

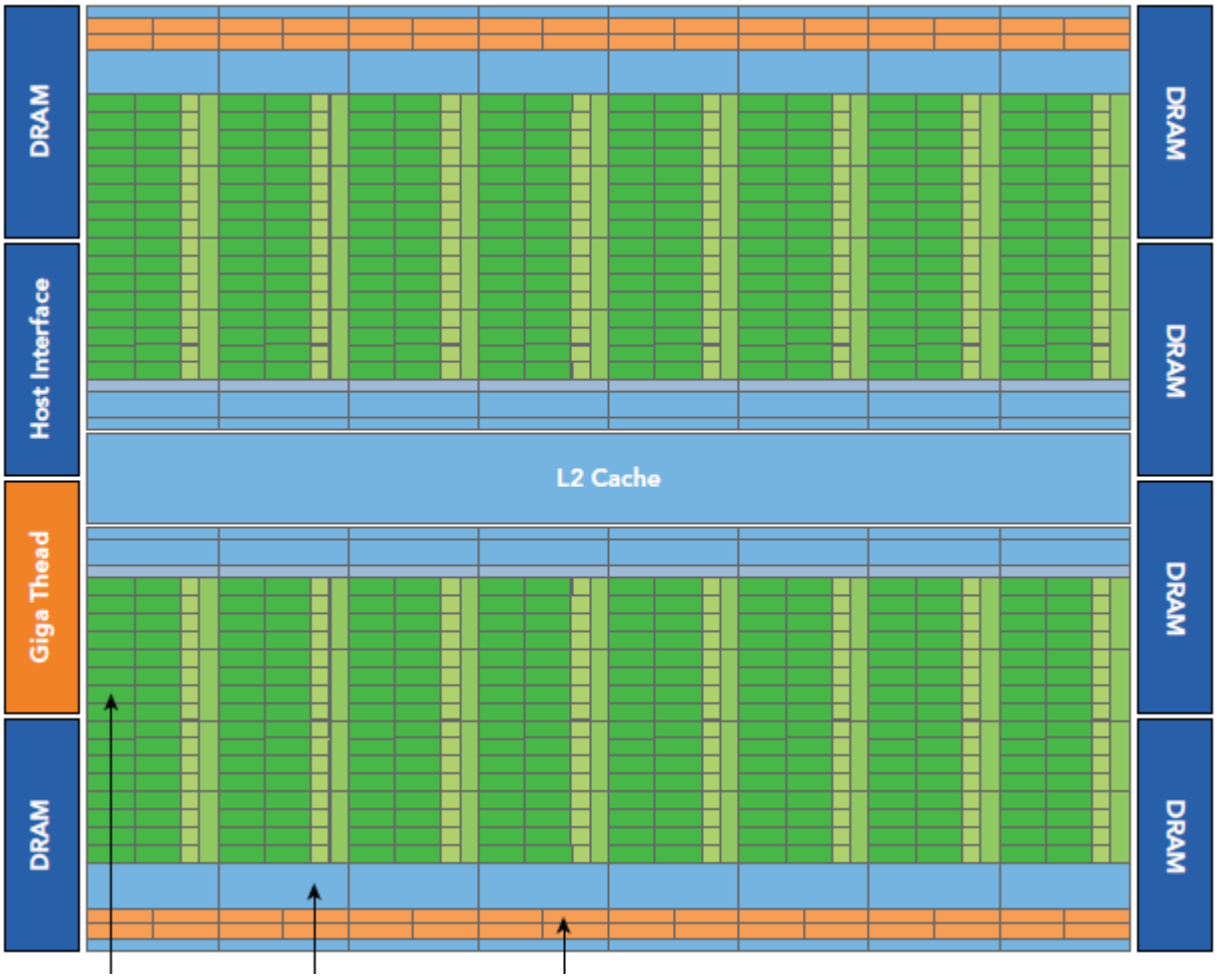
OpenCL 全称 Open Computing Language 即开放计算语言。OpenCL 为异构平台提供了一个编写程序，尤其是并行程序的开放的框架标准。OpenCL 所支持的异构平台可由多核 CPU、GPU 或其他类型的处理器组成。OpenCL 的目标是面向任何一种并行处理器，OpenCL 是第一种真正的开放自由版权编程标准，适用于异构系统上的通用计算。而异构平台可由 CPU、GPU、DSP、FPGA 或其他类型的处理器搭建。

二、 Fermi 架构

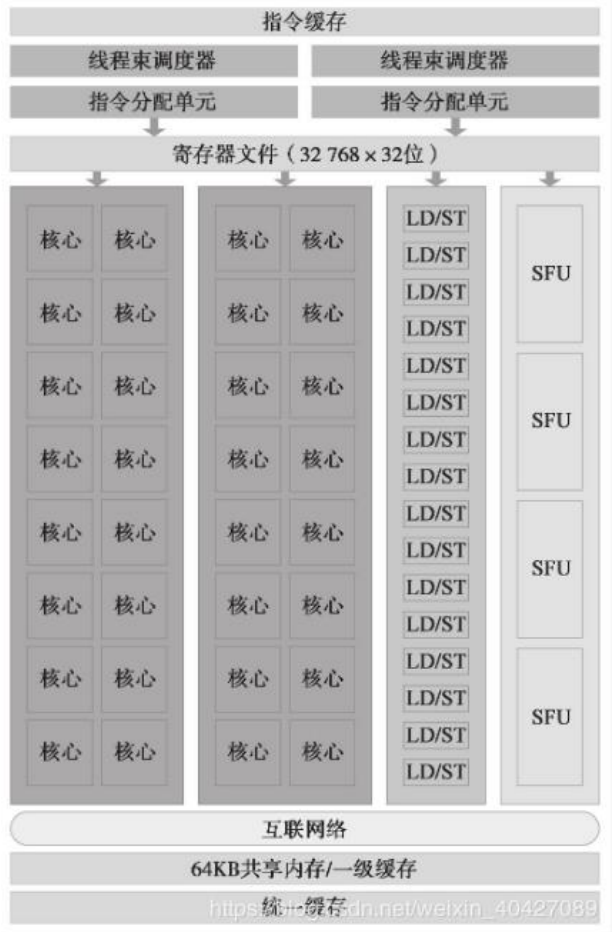
Fermi 是第一个完整的 GPU 计算架构。

- ✧ 512 个 accelerator cores 即所谓 CUDA cores（包含 ALU 和 FPU）
- ✧ 16 个 SM，每个 SM 包含 32 个 CUDA core
- ✧ 六个 384 位 GDDR5 DRAM，支持 6GB global on-board memory
- ✧ GigaThread engine（图左侧）将 thread blocks 分配给 SM 调度
- ✧ 768KB L2 cache
- ✧ 每个 SM 有 16 个 load/store 单元，允许每个 clock cycle 为 16 个 thread（即所谓 half-warp，不过现在不提这个东西了）计算源地址和目的地址

- ✧ Special function units（SFU）用来执行 sin cosine 等
- ✧ 每个 SM 两个 warp scheduler 两个 instruction dispatch unit，当一个 block 被分配到一个 SM 中后，所有该 block 中的 thread 会被分到不同的 warp 中。
- ✧ Fermi（compute capability 2.x）每个 SM 同时可处理 48 个 warp 共计 1536 个 thread。



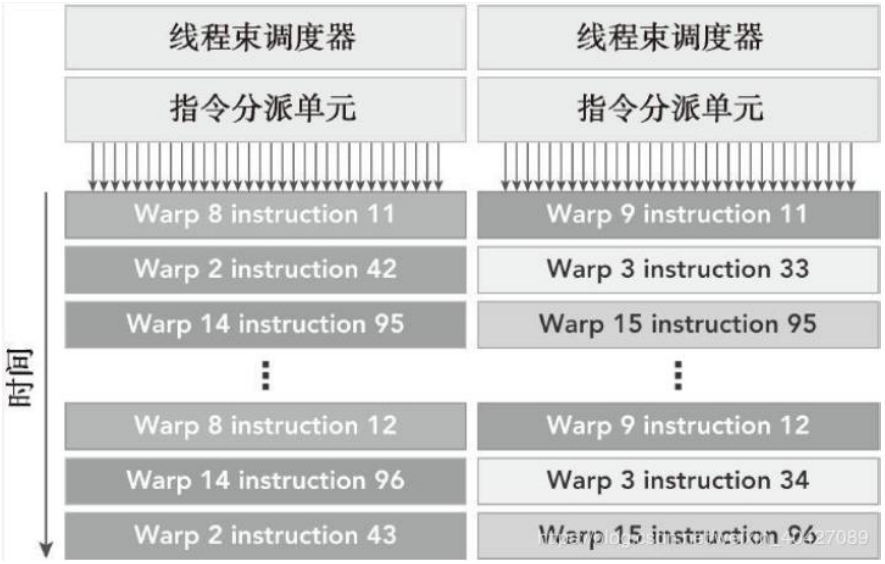
Fermi 架构的 GPU 有一个被 16 个 SM 共享的 L2 缓存，大小为 768K。每个 SM 由执行单元（CUDA cores）、调度分配 warp 的单元、shared memory，register file，L1 cache 组成，结构图如下：



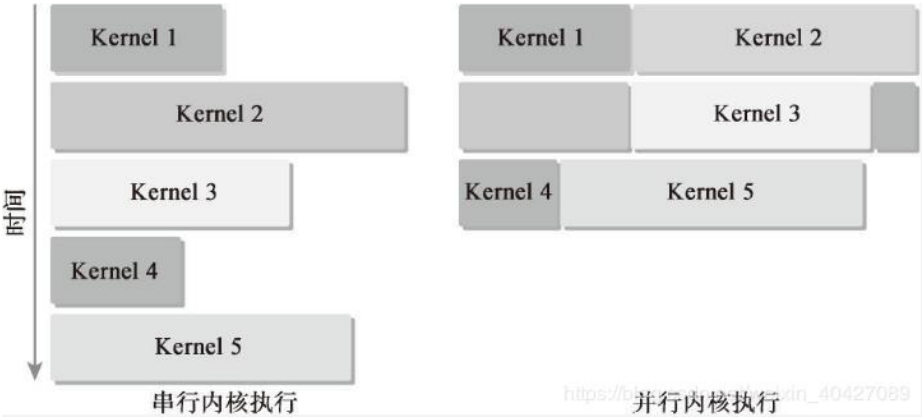
LD/ST 单元（加载/储存单元）：每一个 SM 有 16 个加载/储存单元，允许每个 cycle 有 16 个线程（线程束的一半）计算源地址和目的地址。

SFU（特殊功能单元）：执行固有指令，如正弦、余弦、平方根和插值，速度为：1 指令/cycle

线程束调度器和指令调度单元：当一个线程块被指定给一个 SM 时，所有的线程会被分成线程束。一个 SM 有两个线程束调度器，它们会分别选择两个线程束，再把指令从线程束发送到一个组上。组是一个概念，指的是拥有 16 个 CUDA 核心，16 个 LD/ST 单元和 4 个 SFU 单元的一个集合。因为我们知道，在硬件底层来说，核函数都是汇编语言代码，都是由一句句的指令顺序组成的。因此每一个 CUDA 核心都可以被分配到某一个线程的某一条指令并进行运算，如下图所示：



Fermi 架构还支持并发内核执行，如下图所示：



使用 Fermi 架构的 Tesla GPU：未找到资料（但由于其是第一代架构，是其余 GPU 架构的基础，所以重点分析了一下）。

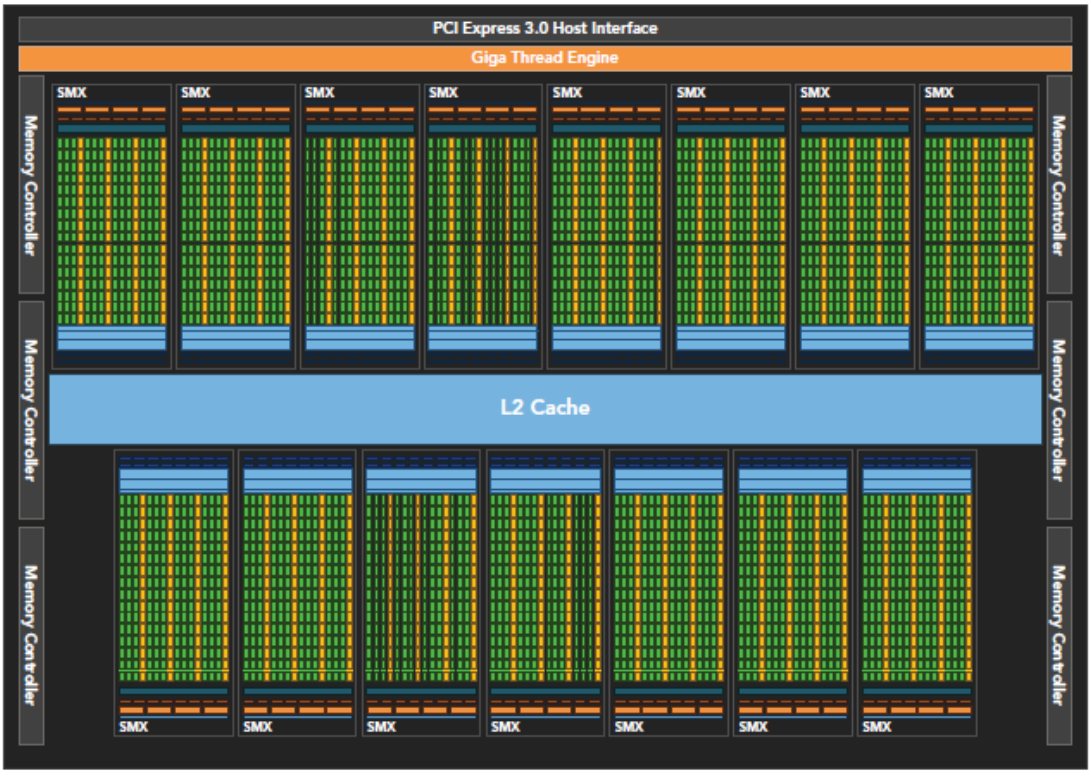
三、 Kepler 架构

Kepler 架构是英伟达于 2012 年推出的新的 GPU 架构，其与 Fermi 架构的最大区别是：SM 单元的结构有所变化。Kepler 相较于 Fermi 更快，效率更高，性能更好。

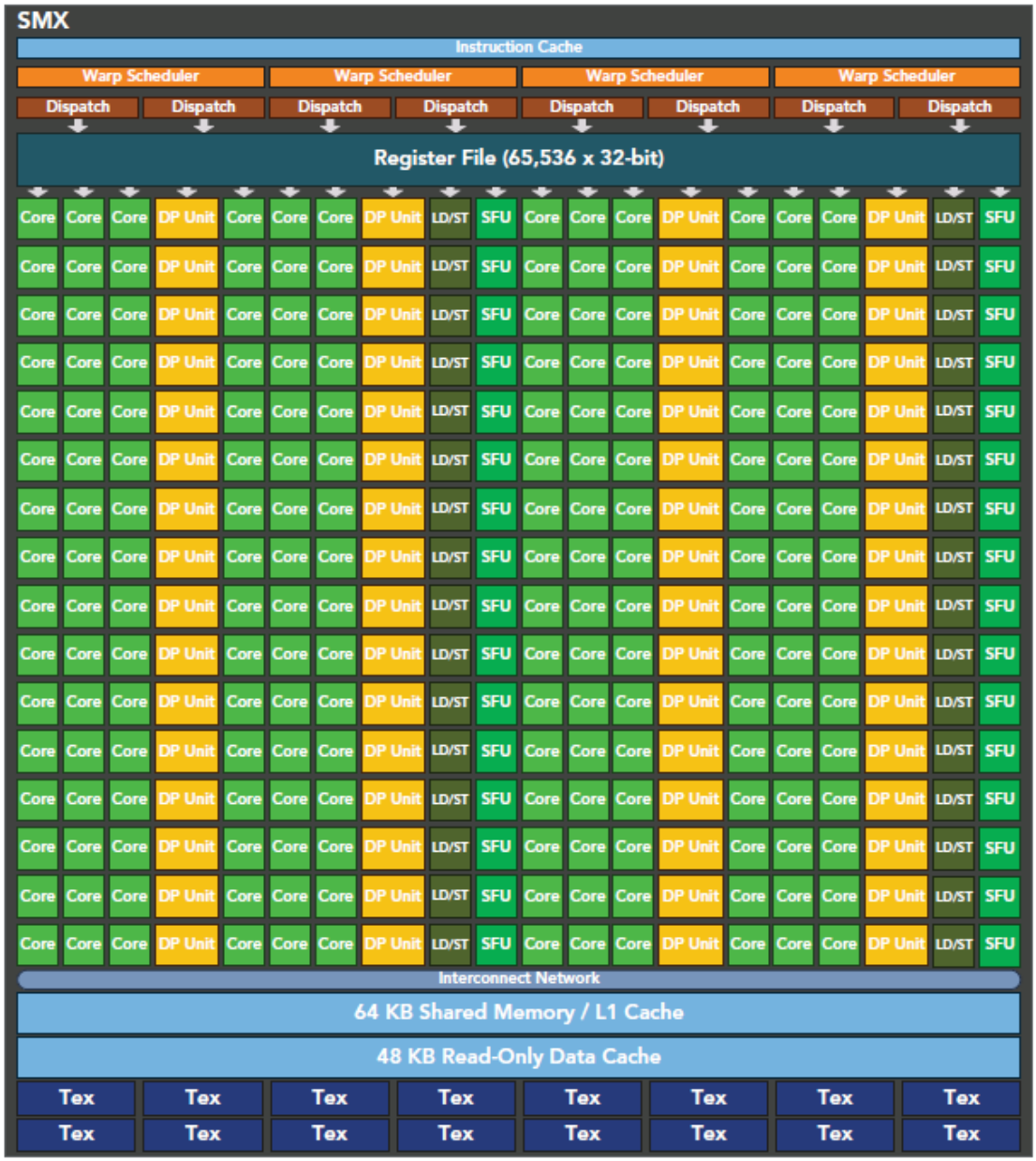
Kepler 组成部分：

✧ 15 个 SM

- ✧ 6 个 64 位 memory controller
- ✧ 192 个单精度 CUDA cores, 64 个双精度单元, 32 个 SFU, 32 个 load/store 单元（LD/ST）
- ✧ 增加 register file 到 64K
- ✧ 每个 Kepler 的 SM 包含四个 warp scheduler、八个 instruction dispatchers, 使得每个 SM 可以同时 issue 和执行四个 warp。
- ✧ Kepler K20X（compute capability 3.5）每个 SM 可以同时调度 64 个 warp 共计 2048 个 thread。



Kepler 架构下的 SM 单元:



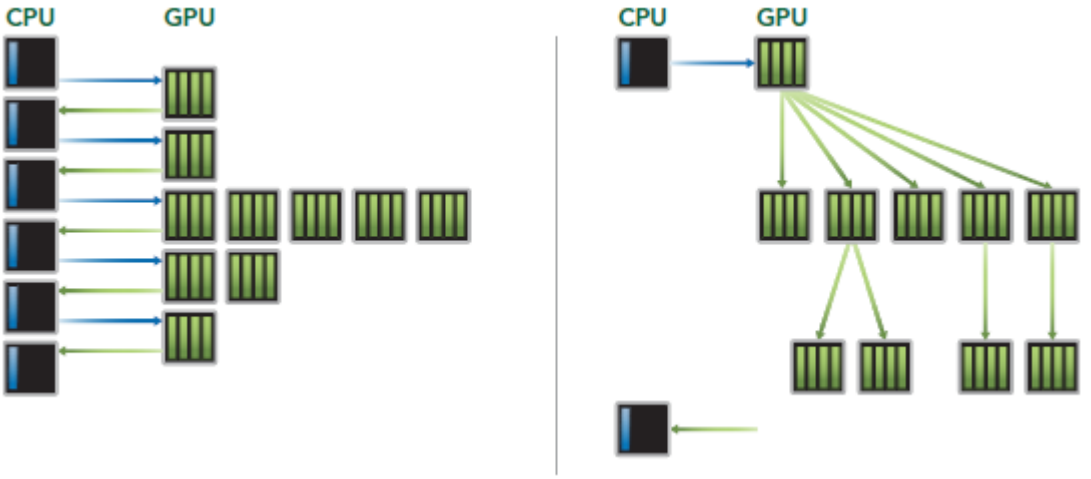
一个 Kepler 架构下的 SM 单元有 4 个线程束调度器，8 个调度器。192 个单精度 CUDA 核心，64 个 DP Unit（双精度单元），32 个特殊功能单元以及 32 个 LD/ST 单元。

按照分析 Fermi 架构的 SM 单元的方法，根据这些数量来分析 Kepler 架构的一些性能。一个 SM 单元有 4 个线程束调度器，也就是说在一个 SM 上可以同时发送和执行 4 个线程束。而一个组有 32 个 SFU，32 个 LD/ST 单元，48 个 CUDA 核心和 16 个 DP Unit。

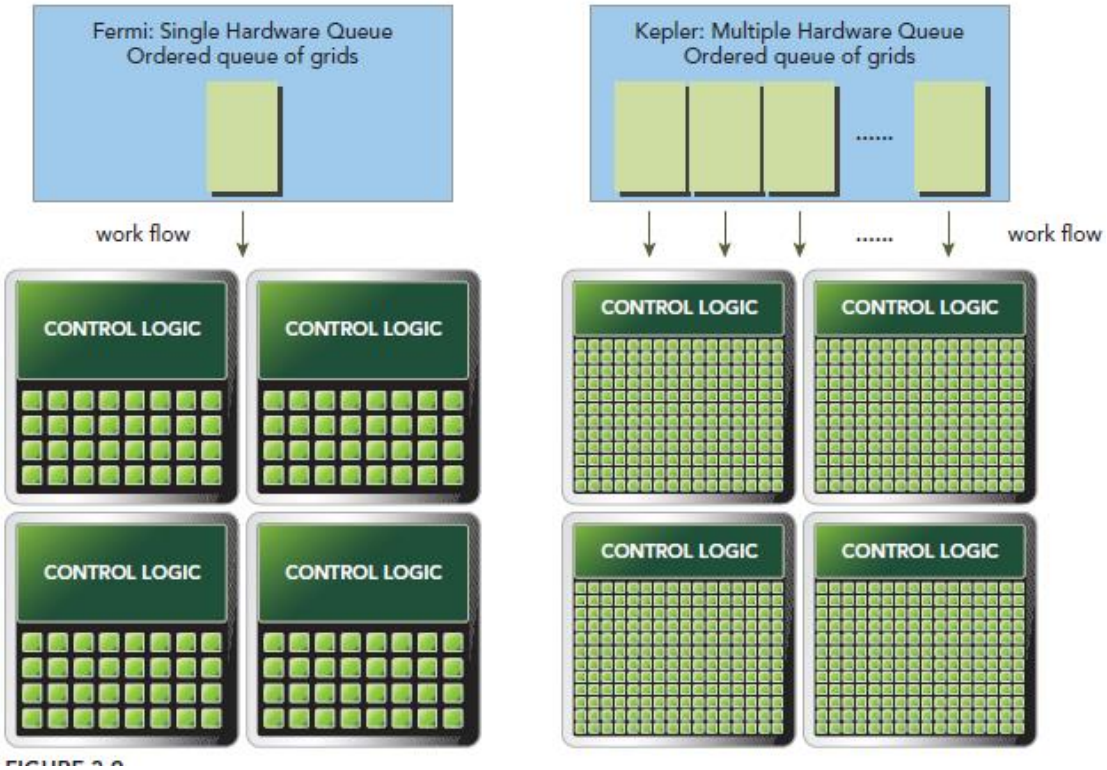
一点很大的突破——从 Kepler 架构开始，由于引入了 DP Unit，在 CUDA 上使用 double 型变量成为了可能，这使得计算可以得到的精度更高。

Kepler 架构的两个新特性：

- **动态并行**：Kepler 架构允许 GPU 动态并行，可以在一个内核中启动一个新的内核，也就是嵌套内核。有了这个特性，任何 kernel 内都可以启动其它的 kernel 了。这样直接实现了 kernel 的递归以及解决了 kernel 之间数据的依赖问题。此举可以减少 GPU 与 CPU 的通信，减小工作负载。



- **Hyper-Q 技术**：Hyper-Q 技术是 Kepler 架构开始拥有的新技术，增加了 CPU 和 GPU 之间硬件上的联系，使 CPU 可以在 GPU 上同时运行更多的任务。这样就可以增加 GPU 的利用率减少 CPU 的闲置时间。Fermi 依赖一个单独的硬件上的工作队列来从 CPU 传递任务给 GPU，这样在某个任务阻塞时，会导致之后的任务无法得到处理，Hyper-Q 解决了这个问题。它可以在主机与 GPU 之间提供 32 个硬件工作队列，以减少任务在队列中阻塞的可能性。这种技术可以保证在 GPU 上可以有更多的并发执行，更大程度上提升 GPU 的整体性能。

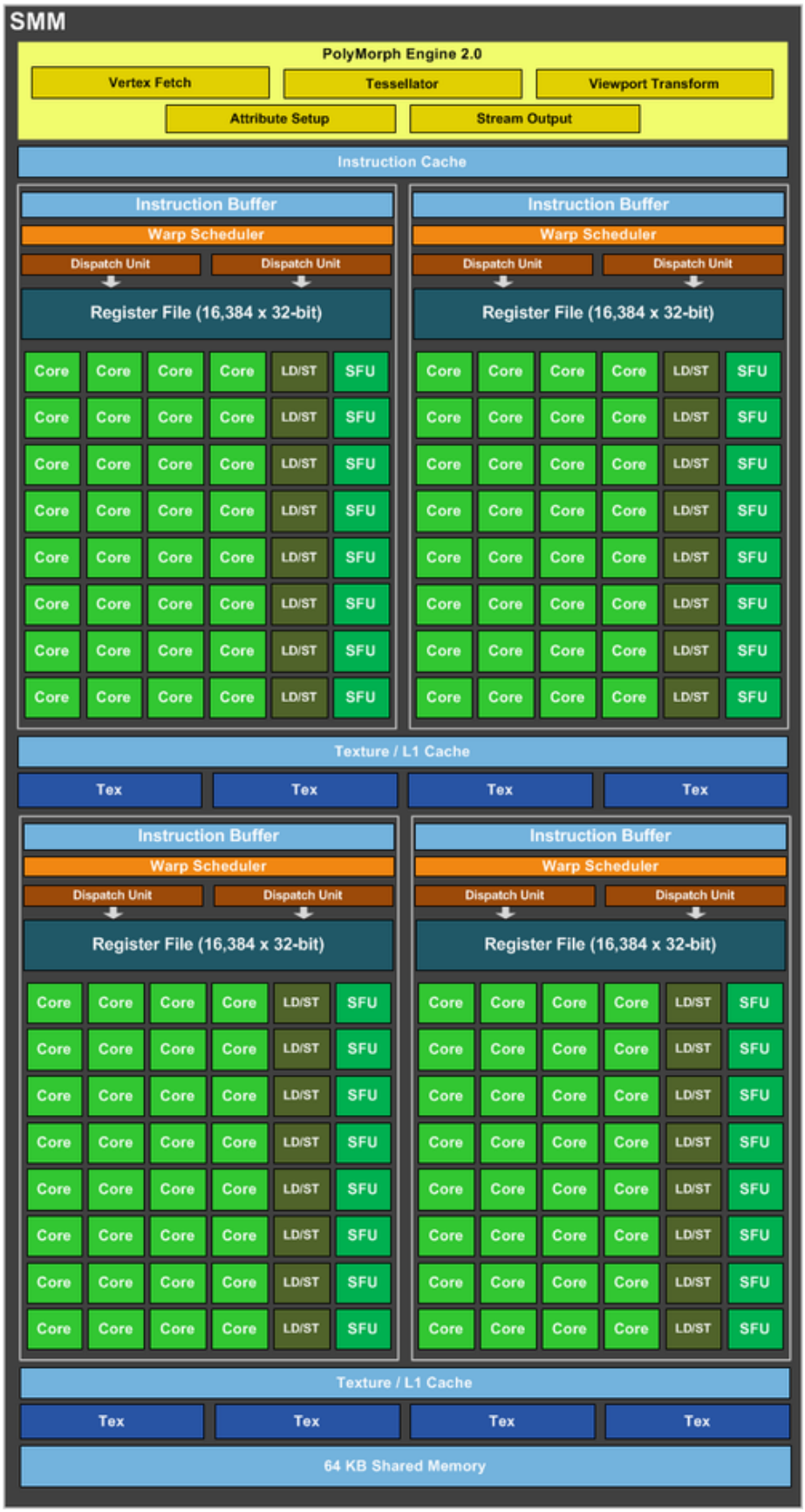


使用 Kepler 架构的 Tesla GPU：K80、K40、K20、K10

四、 Maxwell 架构

Maxwell 并没有全面革新的技术改进，而是在之前 Fermi 和 Kepler 的基础上做了很多的改进而得来的。它在流式多处理器方面采用了一种全新设计，可大幅提高每瓦特性能和每单位面积的性能。

控制逻辑分区、负载平衡、时钟门控粒度、调度、每时钟周期发出指令条数等方面的改进以及其它诸多增强之处让 Maxwell SM 能够在效率上远超 Kepler SMX。全新的 Maxwell SM 架构能够在 GM107 中将 SM 的数量增加至五个(相比之下 GK107 中只有两个)，而芯片面积却仅增加 25%。



Maxwell 效率上的提升主要归功于全新的 Maxwell SM 架构，即 SMM 。

这种全新的 SM 架构可大幅提升节能性，而且在着色器有限的工作场合中可令每个 CUDA 核心的性能提升 35%。实现这些进步需要对架构进行大量重大更改。NVIDIA 重新编写了 SM 调度器架构和算法，使其更加智能，避免了不必要的停顿，同时进一步降低了调度每条指令所需的能耗。

NVIDIA 在 Maxwell 更改了 SM 的组织方式。每个 SM 分为四个独立的处理块，每个处理块具备自己的指令缓冲区、调度器以及 32 个 CUDA 核心。Kepler 的方法是划分为非 2 幂 (non-power-of-two) 数量的 CUDA 核心，其中一些是共享核心，这种方法现已弃用。新的划分方法简化了设计与调度逻辑、节省了面积与功耗、降低了计算延迟。

成对的处理块共享四个纹理过滤单元和一个纹理高速缓存。计算一级高速缓存的功能也与纹理高速缓存相结合，而共享显存是一个独立的单元(类似首款 CUDA GPU—— G80 中所使用的方法)，被全部四个块共享。

总体而言，在这一全新设计上，每个 SMM 的尺寸得到大幅缩减，而性能却能够达到一个 Kepler SMX 的 90%。更小的面积让 NVIDIA 能够在每颗 GPU 中实现更多数量的 SMM 。通过对比 GK107 和 GM107 SM 总数的相关指标可发现，GM107 有五个 SM，而前者只有两个。GM107 的峰值纹理性能比前者高 25%，CUDA 核心数量多 1.7 倍，着色器性能大约高 2.3 倍。

使用 Maxwell 架构的 Tesla GPU：M60、M40

五、 Pascal 架构

Pascal 是 Maxwell 的接替者，增强了异步计算功能实现硬件层了对 DirectX API 的更高版本（DirectX 12 Feature Level 12_1）的支持。除了架构上的改进，

还使用了更好的 16nm FF+工艺 （对比 Maxwell 所使用的 28nm），晶体管密度和性能大幅度提升，功耗发热进一步降低 ， 高端产品还配备带宽更高的 HBM2 显存，性能和能耗比都有了很大提升。

Pascal 的 GPC 有 6 个 SM，每个 SM 只含有 64 个 CUDA Core，虽然 GP100 SM 只有 Maxwell SM 中 CUDA 核心数的一半，但总的 SM 数目增加了，每个 SM 保持与上一代相同的寄存器组，则总的寄存器数目增加了。这意味着 GP100 上的线程可以使用更多寄存器，也意味着 GP100 相比旧的架构支持更多线程、warp 和线程块数目。与此同时，GP100 总共享内存量也随 SM 数目增加而增加了，带宽显著提升不至两倍。拥有 64 个 FP32 单元 32 个 FP64 单元，FP64 与 FP32 比例达到了 1：2，双精度性能大幅度提高，而 Pascal 的 FP32 单元可以同时执行 2 个 FP16 半精度运算，因此 FP16 浮点性能也同样获得极大提升。和以前架构相同，支持 IEEE 754-2008 标准，支持 FMA 运算，支持异常值处理。

Pascal SM 架构图中可以看到，一个 GP100 SM 分成两个处理块，每块有【 32768 个 32 位寄存器 + 32 个单精度 CUDA 核心 + 16 个双精度 CUDA 核心 + 8 个特殊功能单元（SFU） + 8 个存取单元 + 一个指令缓冲区 + 一个 warp 调度器 + 两个分发单元】。

Pascal SM 架构相比 Kepler 架构简化了数据通路，占用面积更小，功耗更低。提供更高级的调度和重叠载入/存储指令来提高浮点利用率。

GP100 中新的 SM 调度器架构相比 Maxwell 更智能，具备高性能、低功耗特性。

Pascal 的关键技术：TSMC 16nm FF+工艺 、NVIDIA® NVLink™ 高速互连技术、HBM2 第二代 3D 堆栈式高带宽内存、依靠 Async shaders 从硬件层面完整实现 AsyncCompute 和支持 DirectX 12Feature Level 12_1。

使用 Pascal 架构的 Tesla GPU：P4、P40、P100

六、 Volta 架构

基于台积电专门为 NVIDIA 设计的最新 12nm FFN 高精度制程封装技术，GV100 在 815 平方毫米的芯片尺寸中，内部集成了高达 211 亿个晶体管结构。相较于 Pascal 系列 GPU，GV100 不但在计算性能上有了长足的进步，同时还增加了许多令人眼前一亮的新特性。包括进一步精简的 GPU 编程和应用部署流程，以及针对 GPU 资源利用情况的深度优化。其结果是，GV100 在提供强大计算性能的同时还非常省电。

Tesla V100 拥有业界领先的浮点和整型运算性能，峰值运算性能如下（基于 GPU Boost 时钟频率）：

双精度浮点（FP64）运算性能：7.5 TFLOP/s;

单精度（FP32）运算性能：15 TFLOP/s;

混合精度矩阵乘法和累加：120 Tensor TFLOP/s。

GV100 由许多图形处理集群（Graphics Processing Cluster，GPC）、纹理处理集群（Texture Processing Cluster，TPC）、流式多处理器（Streaming Multiprocessor，SM）以及内存控制器组成。一个完整的 GV100 GPU 由 6 个 GPC、84 个 Volta SM、42 个 TPC（每个 TPC 包含 2 个 SM）和 8 个 512 位的内存控制器（共 4096 位）。其中，每个 SM 有 64 个 FP32 核、64 个 INT32 核、32 个 FP64 核与 8 个全新的 Tensor Core。同时，每个 SM 也包含了 4 个纹理处理单元（texture units）。

更具体地说，一个完整版 Volta GV100 中总共包含了 5376 个 FP32 核、5376 个 INT32 核、2688 个 FP64 核、672 个 Tensor Core 以及 336 个纹理单元。每个内存控制器都链接一个 768 KB 的 2 级缓存，每个 HBM2 DRAM 堆栈都由一对内存控制器控制。整体上，GV100 总共包含 6144KB 的二级缓存。下图展示了带有 84 个 SM 单元的完整版 Volta GV100，需要注意的是，不同的产品可能具有不同的配置，比如 Tesla V100 就只有 80 个 SM。

Volta SM 主要特性：

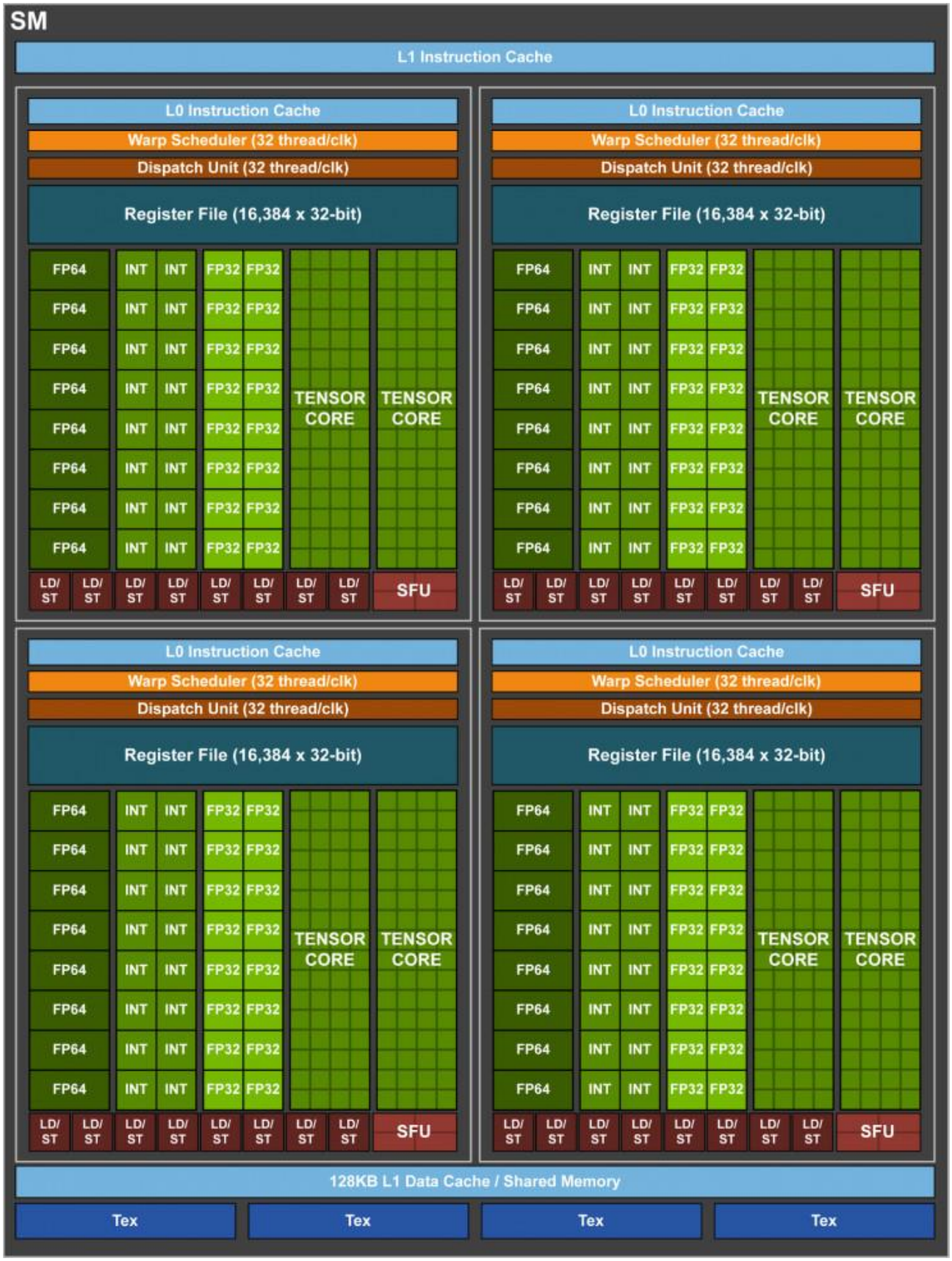
✧ 为深度学习矩阵计算建立的新型混合精度 FP16/FP32 Tensor Core;

✧ 为更高性能、更低延迟而强化的 L1 高速数据缓存;

✧ 为简化解码和缩短指令延迟而改进的指令集;

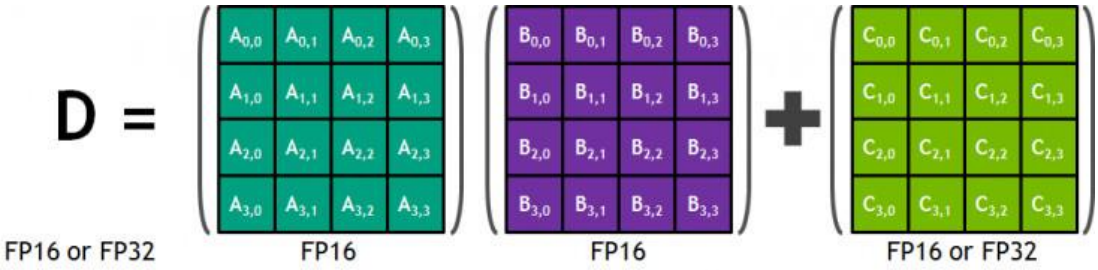
✧ 更高的时钟频率和能效。

下图显示了 Volta GV100 SM 单元的基本结构:

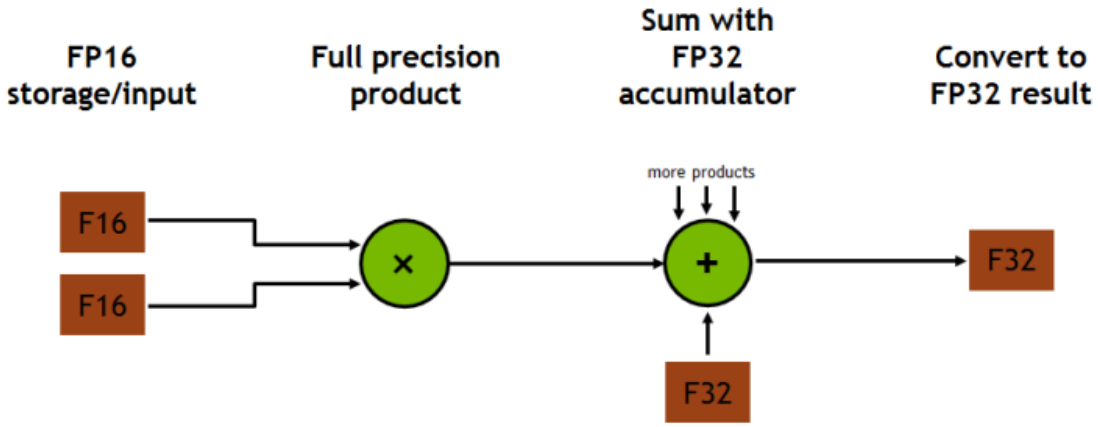

SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

Tensor Core：

全新的 Tensor Core 是 Volta GV100 架构中最重要的一项新特性，在训练超大型神经网络模型时，它可以为系统提供强劲的运算性能。Tesla V100 的 Tensor Core 可以为深度学习相关的模型训练和推断应用提供高达 120 TFLOPS 的浮点张量计算。在 Volta 架构中，每个 Tensor Core 都包含一个 4x4x4 的矩阵处理队列，来完成神经网络结构中最常见的 D=AxB+C 运算。其中 A、B、C、D 是 4 个 4×4 的矩阵，因此被称为 4x4x4。如下图所示，输入 A、B 是指 FP16 的矩阵，而矩阵 C 和 D 可以是 FP16，也可以是 FP32。

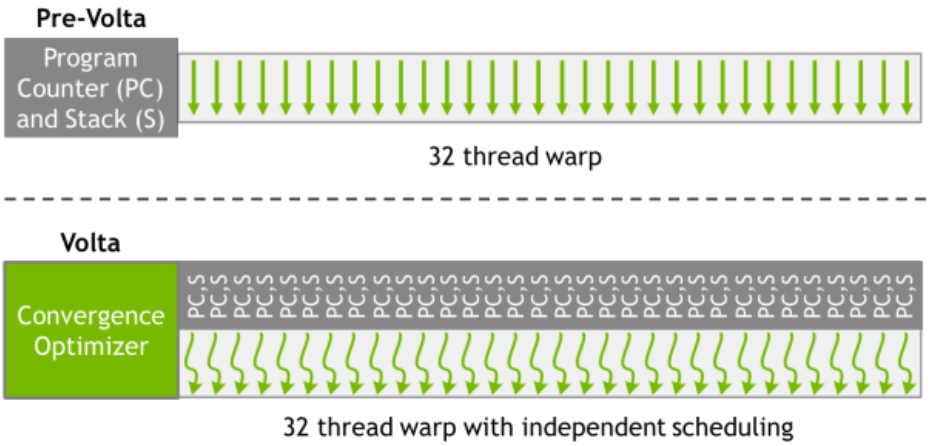


具体到硬件架构：



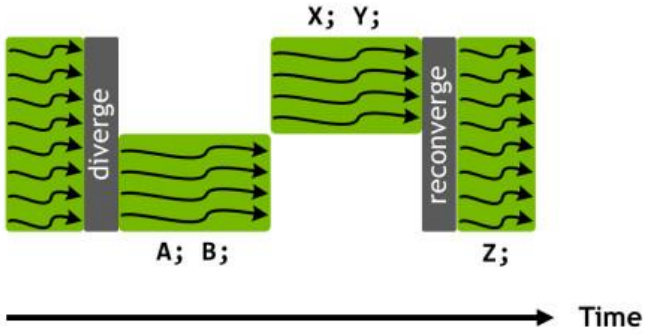
Independent Thread Scheduling：

Volta 的另一个重要架构改动是所谓独立的线程调度。在 Volta 之前，Nvidia 一直使用的 SIMT（Single Instruction, Multiple Threads）架构最重要的特征就是一个“warp”中的 32 个线程是共享一个 PC（Program Counter）和栈（Stack）的。而在 Volta 中，则每个线程有了自己的 PC 和 Stack。如下图所示。



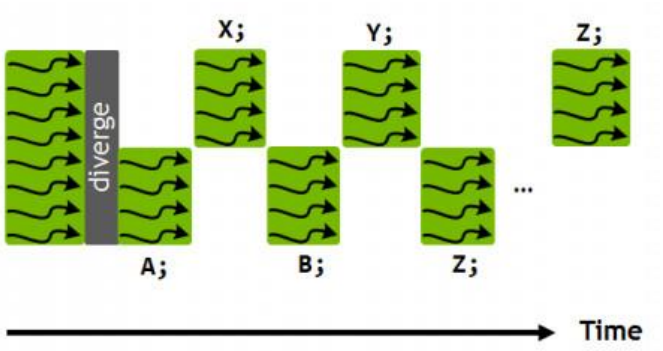
由于 Volta 之前的架构中多个线程共享 PC 和 Stack，在线程调度的时候粒度是比较粗的。比如下面这种情况。当出现分支的适合，对于 threadIdx < 4 的线程就要执行 A，B；否则是 X，Y。Pascal 的一个“warp”的 32 个线程共享一个 PC，并结合一个“活动掩码”（active mask），指定任何给定时间哪个线程是活动的。 这意味着不同的程序分支使某些线程无效。“warp”的不同部分顺序执行，直到再次收敛，此时掩模被恢复，所有线程再次一起运行。

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```



在 volta 中，情况则发生了变化，如下图所示，可以实现不同的调度方式，程序中 if 和 else 分支的语句现在可以及时交错。当然，这里程序的执行还是 SIMT 方式，即在任何给定的时钟周期，warp 中的所有活动线程执行相同的指令，从而保留以前架构的执行效率。

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```



从这里可以看出，独立的线程调度的重点是实现了更细致的调度粒度，当然代价是增加了很多硬件开销（每个 thread 都要有自己的 PC 和 Stack）。

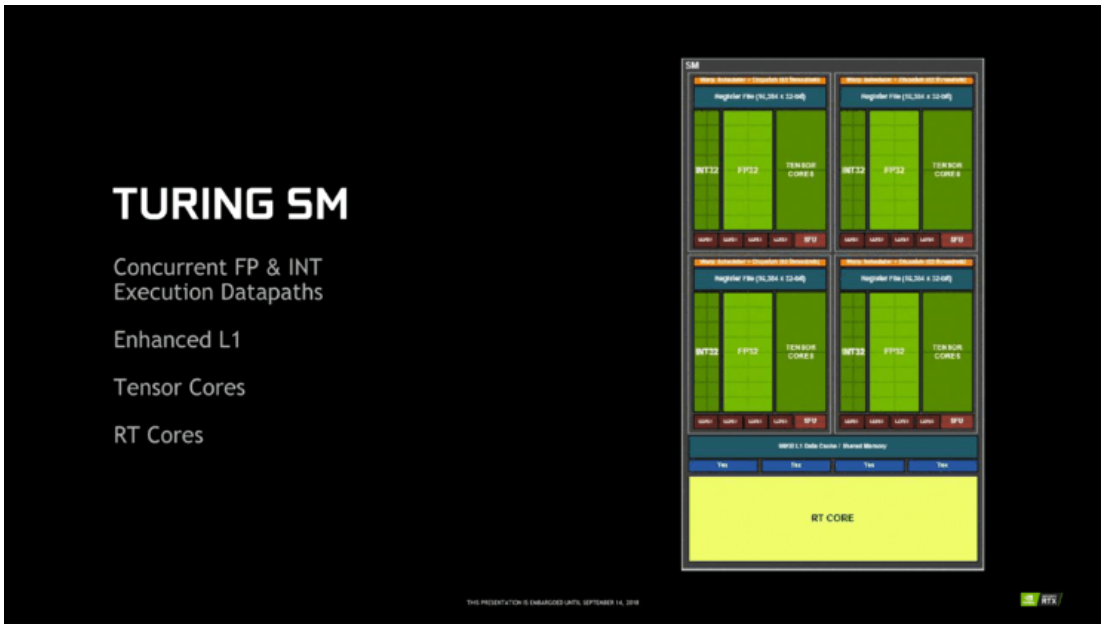
其他改进：

- ✧ 能够同时执行 FP32 和 INT32 指令
- ✧ 增强 L1 Data Cache 和 Shared Memory，可以支持更灵活的 cache 和 share memory 的使用，可以充分利用 shared memory 在性能上的优势
- ✧ STARVATION-FREE ALGORITHMS

使用 volta 架构的 Tesla GPU：V100

七、 Turing 架构

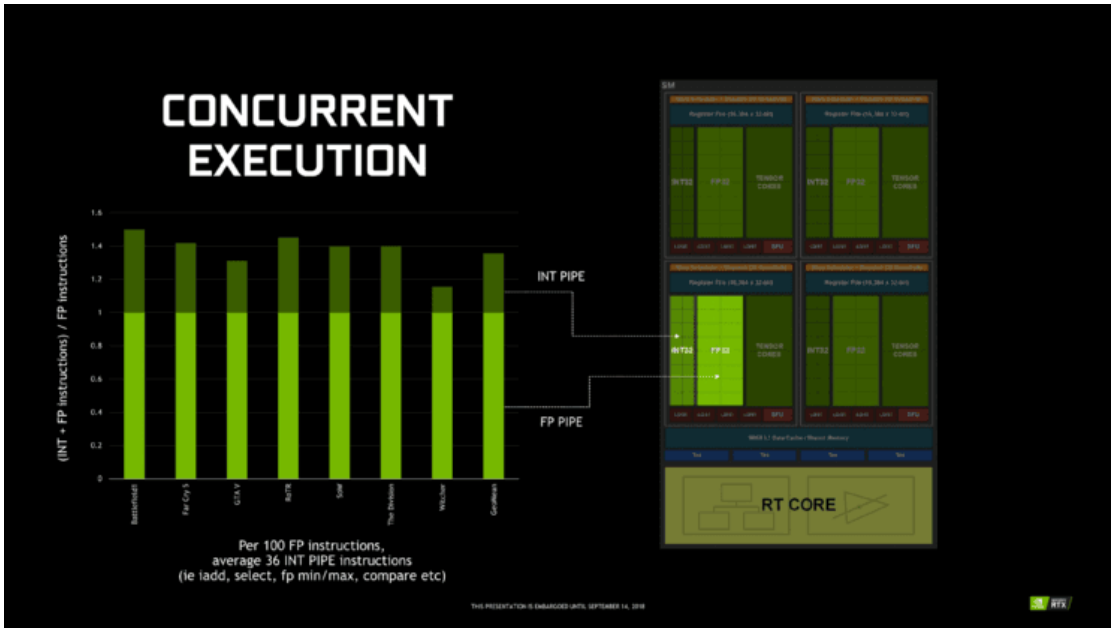
Turing 架构的发布，标志着计算机图形学在消费级市场上开始从虚假的视觉欺骗向着真实的追光逐影发展。Turing 架构增设了专用的光线追踪单元 RT Core，并辅以 Tensor Core 来进行 AI 降噪。



与 Volta 一样，Turing SM 被划分为 4 个子核（或处理块），每个子核具有单个 warp 调度器和调度单元，而 Pascal 的 2 个分区设置是每个子核的 warp 调度器具有两个相对的调度端口。

从广义上讲，这样的变化意味着 Turing 失去了在一个时钟周期内从线程发出第二条非依赖指令的能力。Turing 可能与 Volta 在两个周期内执行指令相同，但调度程序可以在每个周期发出独立指令，因此 Turing 最终可以通过这种方式维护双向指令级并行（ILP），同时仍然具有两倍于 Pascal 的调度程序数量。

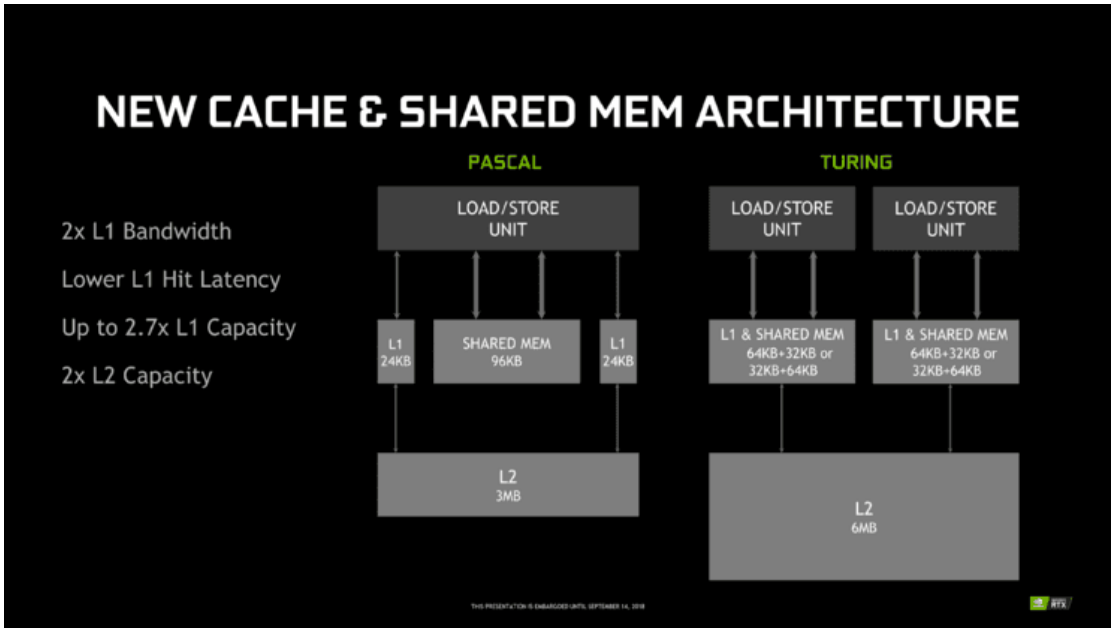
Turing 有独立的线程调度模型。Turing 有每个线程的调度资源，有一个程序计数器和每个线程的堆栈来跟踪线程的状态，以及一个收敛优化器来智能的将活动的同 warp 线程分组到 SIMT 单元中。



就 CUDA 和 ALU (算术逻辑单元) 而言, Turing 子核具有 16 个 INT32 单元, 16 个 FP32 单元和 2 个 Tensor 单元，与 Volta 子核的设置相同。使用像 Volta 这样的拆分 INT/FP 数据路径模型，Turing 还可以同时执行 FP 和 INT 指令，而这与 RT Core 密切相关。Turing 与 Volta 的不同之处在于 Turing 没有 FP64 单元，其 FP64 的吞吐量只有 FP32 的 1/32。

对于 Turing 的第二代 Tensor Core 和 RT Core 来说这种设计似乎是为了最大化 Tensor Core 的性能，而最大限度的减少了破坏性并行性或与其他计算工作负载的协调情况, 其中 4 个独立调度的子核和粒度线程处理对于在混合游戏导向工作负载下实现最高性能非常有用。

在内存方面，Turing 的每个子核都有一个类似 Volta 的 L0 指令缓存，具有相同大小的 64 KB 寄存器文件。在 Volta 中，这对于减少 Tensor Core 的延迟很重要，而在 Turing 中这可能同样有利于 RT Core。Turing SM 每个子核也有 4 个加载/存储单元，低于 Volta 中的 8 个，但仍然保持 4 个纹理单元。



新的 L1 数据高速缓存和共享内存（SMEM）进一步向上扩展，它已被改进并统一为单个可分区内存块，这是 Volta 的另一项创新。对于 Turing 来说，这看起来是一个组合的 96 KB L1/SMEM，传统图形工作负载分为 64KB 专用图形着色器 RAM 和 32 KB 纹理高速缓存和寄存器文件溢出区域。同时，计算工作负载可以将 L1/SMEM 划分最多 64 KB 作为 L1，其余 32 KB 作为 SMEM，反之亦然（Volta 的 SMEM 最高可配置为 96 KB）。

使用 Turning 架构的 Tesla GPU：T4。

八、 总结心得

本次报告在查阅资料时由于对显卡的不了解，一直找不到英伟达 Tesla GPU 的架构到底是哪个，费了好大劲才知道原来他的 GPU 架构是随着时间技术而不断更新的，每个类型以首字母显示。虽然架构在不断的发展进步，但是其基本结构组成没有太大的改变。