白皮书

# 英伟达™ (NVIDIA®) 下一代 CUDA™ 计算架构：

# Kepler™ GK110

*有史以来速度最快，效率最高的 **HPC** 架构*

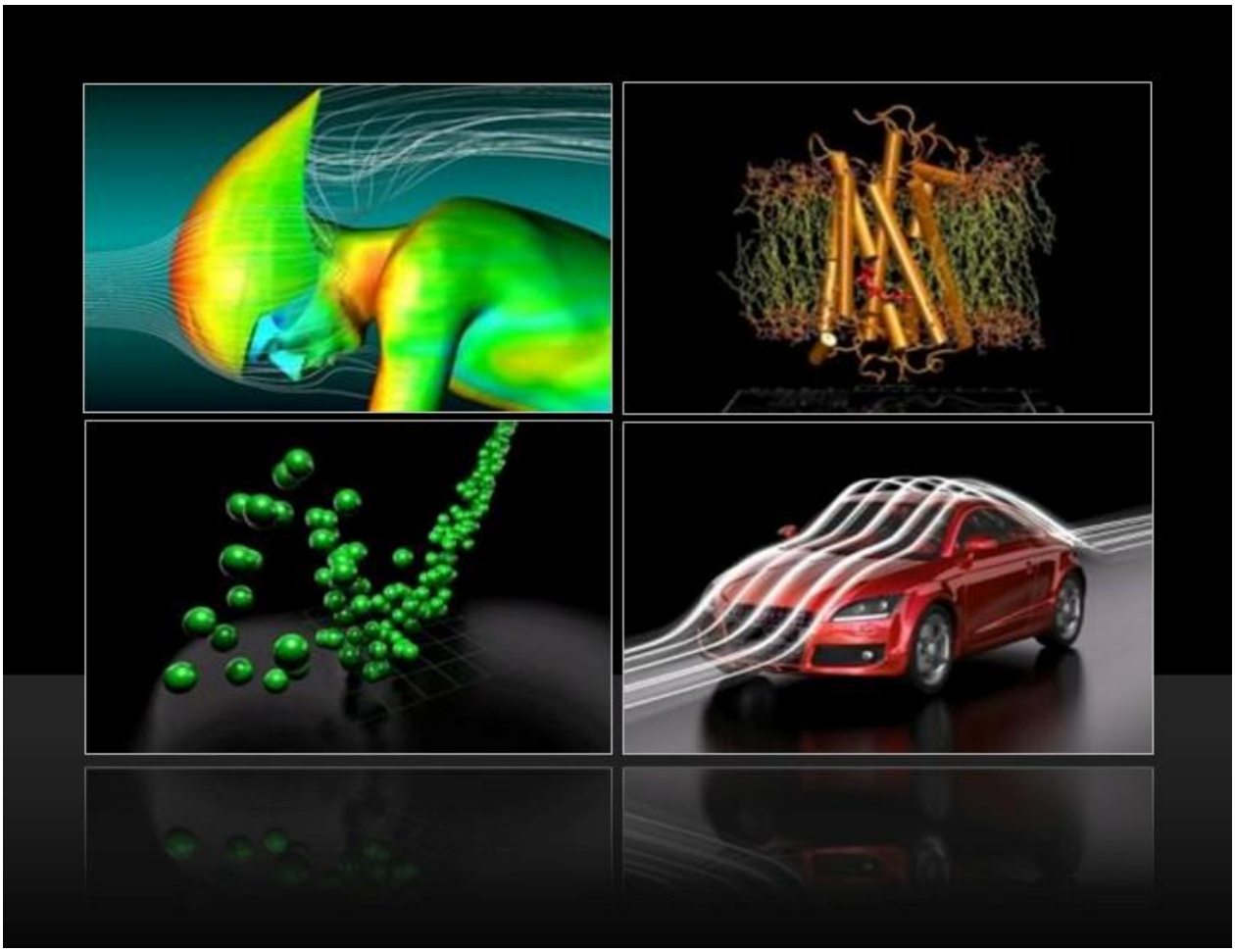
V1.0

# 目录

# 新一代 GPU 计算架构

随着科学、医学、工程和金融各领域对高性能并行计算需求的增加，英伟达™ (NVIDIA®) 以无比强大的 GPU 计算架构来不断创新和满足这种需求。英伟达现有的 Fermi GPU已经重新定义和加速了以下领域的高性能计算（HPC）的功能，如地震处理、生化模拟、天气和气候建模、信号处理、计算金融、计算机辅助工程、计算流体力学和数据分析。英伟达的新 Kepler GK110 GPU 大大提高了并行计算标准，并将会帮助解决世界上面临的最困难的计算问题。

通过提供比上一代 GPU 更强大的处理功能以及优化和提高 GPU 上并行执行工作负载的新方法，Kepler GK110 简化了并行程序的创建，将对会对高性能计算引起进一步改革。

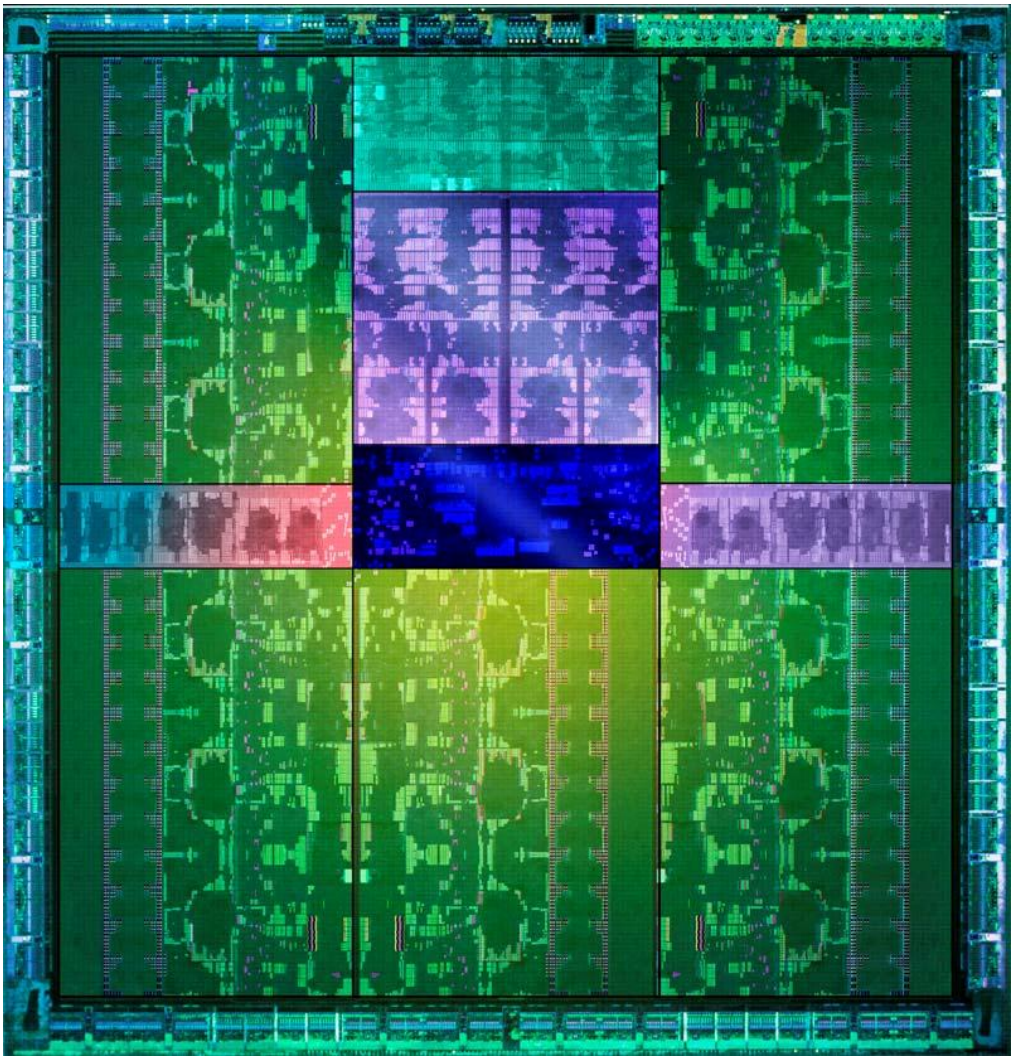


# Kepler GK110 – 性能卓越，效率极高

Kepler GK110 由 71 亿个晶体管组成，不仅速度最快，而且还是有史以来架构最复杂的微处理器。GK110 新加了许多注重计算性能创新功能，目的是要成为英伟达™ Tesla® 和 HPC 市场上的并行处理动力站。

Kepler GK110 会提供超过每秒 1 万亿次双精度浮点计算的吞吐量，DGEMM 效率大于 80%，而之前的 Fermi 架构的效率是 60 – 65%。

除了大大提高的性能之外，Kepler 架构在电源效率方面有 3 次巨大的飞跃，使 Fermi 的性能/功率比提高了 3 倍。



**Kepler GK110 模具照片**

Kepler GK110 的以下新功能提高 GPU 的利用率，简化了并行程序设计，并有助于 GPU 在各种计算环境中部署，无论是从个人工作站还是到超级计算机：

- **Dynamic Parallelism –** 能够让 GPU 在无需 CPU 介入的情况下，通过专用加速硬件路径为自己创造新的工作，对结果同步，并控制这项工作的调度。这种灵活性是为了适应程序执行过程中并行的数量和形式，编程人员可以处理更多的各种并行工作，更有效的将 GPU 用为计算用途。此功能允许结构较简单，一但较复杂的任务方便有效地运行，能使较大部分的应用程序在整个 GPU 上运行。此外，程序能够更容易的创建，CPU 能为其他任务释放。

- **Hyper-Q –** Hyper - Q 允许多个 CPU 核同时在单一 GPU 上启动工作，从而大大提高了 GPU 的利用率并削减了 CPU 空闲时间。Hyper - Q 增加了主机和 Kepler GK110 GPU 之间的连接总数（工作队列），允许 32 个并发、硬件管理的连接（与 Fermi 相比，Fermi 只允许单个连接）。Hyper - Q 是一种灵活的解决方案，允许来自多个 CUDA 流、多个消息传递接口（MPI）进程，甚至是进程内多个线程的单独连接。以前遇到跨任务虚假串行化的应用程序，限制了 GPU 的利用率，而现在无需改变任何现有代码性能就能大幅度提升。

- **Grid Management Unit –**使 Dynamic Parallelism 能够使用先进、灵活的 GRID 管理和调度控制系统。新 GK110 Grid Management Unit (GMU) 管理并按优先顺序在 GPU 上执行的 Grid。GMU 可以暂停新 GRID 和等待队列的调度，并能中止 GRID，直到其能够执行时为止，这为 Dynamic Parallelism 这样的强大运行提供了灵活性。GMU 确保 CPU - 和 GPU - 产生的工作负载得到妥善的管理和调度。

- **英伟达™ GPUDirect™–** 英伟达™ GPUDirect™ 能够使单个计算机内的 GPU 或位于网络内不同服务器内的 GPU 直接交换数据，无需进入 CPU 系统内存。GPUDirect 中的 RDMA 功能允许第三方设备，例如 SSD、NIC、和 IB 适配器，直接访问相同系统内多个 GPU 上的内存，大大降低 MPI 从 GPU 内存发送/接收信息的延迟。还降低了系统内存带宽的要求并释放其他 CUDA 任务使用的 GPU DMA 引擎。Kepler GK110 还支持其他的 GPUDirect 功能，包括 Peer - to - Peer 和 GPUDirect for Video。

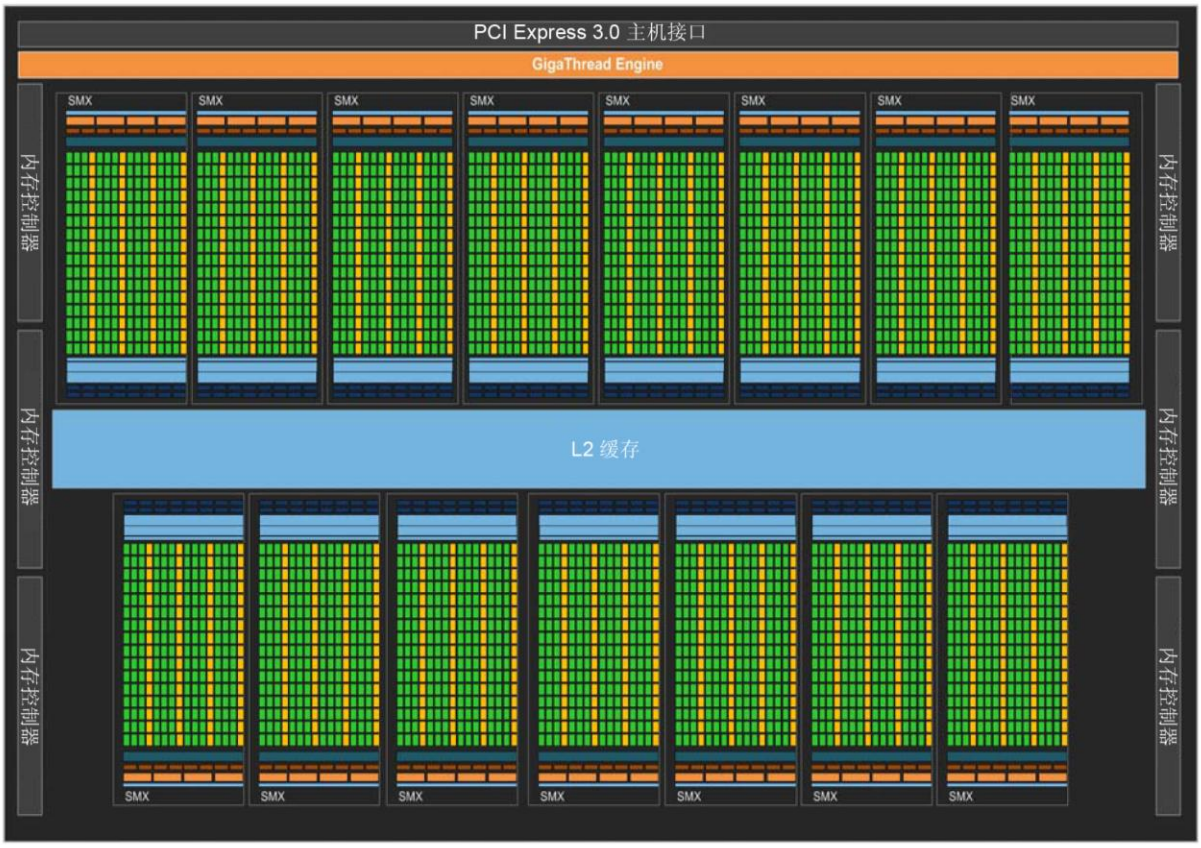
# GK110 Kepler 架构概述

Kepler GK110 专为英伟达™ Tesla® 打造，其目标是成为世界上并行计算性能最高的微处理器。GK110 不仅大大超过由 Fermi 提供的原始计算能力，而且非常节能，显著减少电力消耗，同时产生的热量更少。

完整 Kepler GK110 实施包括 15 SMX 单元和六个 64 位内存控制器。不同的产品将使用 GK110 不同的配置。例如，某些产品可能部署 13 或 14 个 SMX。

在下面进一步讨论的该架构的主要功能，包括：

- 新 SMX 处理器架构
- 增强的内存子系统，在每个层次提供额外的缓存能力，更多的带宽，且完全进行了重新设计，DRAM I/O 实施的速度大大加快。
- 贯穿整个设计的硬件支持使其具有新的编程模型功能




**Kepler GK110 完整芯片框图**

Kepler GK110 支持新 CUDA Compute Capability 3.5。（有关 CUDA 的简介请参考附录 A - CUDA 快速回顾）。下表对比了 Fermi 和 Kepler GPU 架构的不同计算能力的参数：

| | FERMI GF100 | FERMI GF104 | KEPLER GK104 | KEPLER GK110 |
|---|---|---|---|---|
| 计算能力 | 2.0 | 2.1 | 3.0 | 3.5 |
| 线程/Warp | 32 | 32 | 32 | 32 |
| 最大 Warp / 多处理器 | 48 | 48 | 64 | 64 |
| 最大线程 / 多处理器 | 1536 | 1536 | 2048 | 2048 |
| 最大线程块 / 多处理器 | 8 | 8 | 16 | 16 |
| 32 位寄存器 / 多处理器 | 32768 | 32768 | 65536 | 65536 |
| 最大寄存器 / 线程 | 63 | 63 | 63 | 255 |
| 最大线程 / 线程块 | 1024 | 1024 | 1024 | 1024 |
| 共享内存大小配置（字节） | 16K<br>48K | 16K<br>48K | 16K<br>32K<br>48K | 16K<br>32K<br>48K |
| 最大 X Grid Dimension | 2^16 - 1 | 2^16 - 1 | 2^32 - 1 | 2^32 - 1 |
| Hyper - Q | 不支持 | 不支持 | 不支持 | 支持 |
| Dynamic Parallelism | 不支持 | 不支持 | 不支持 | 支持 |

**Fermi 和 Kepler GPU 的计算能力**

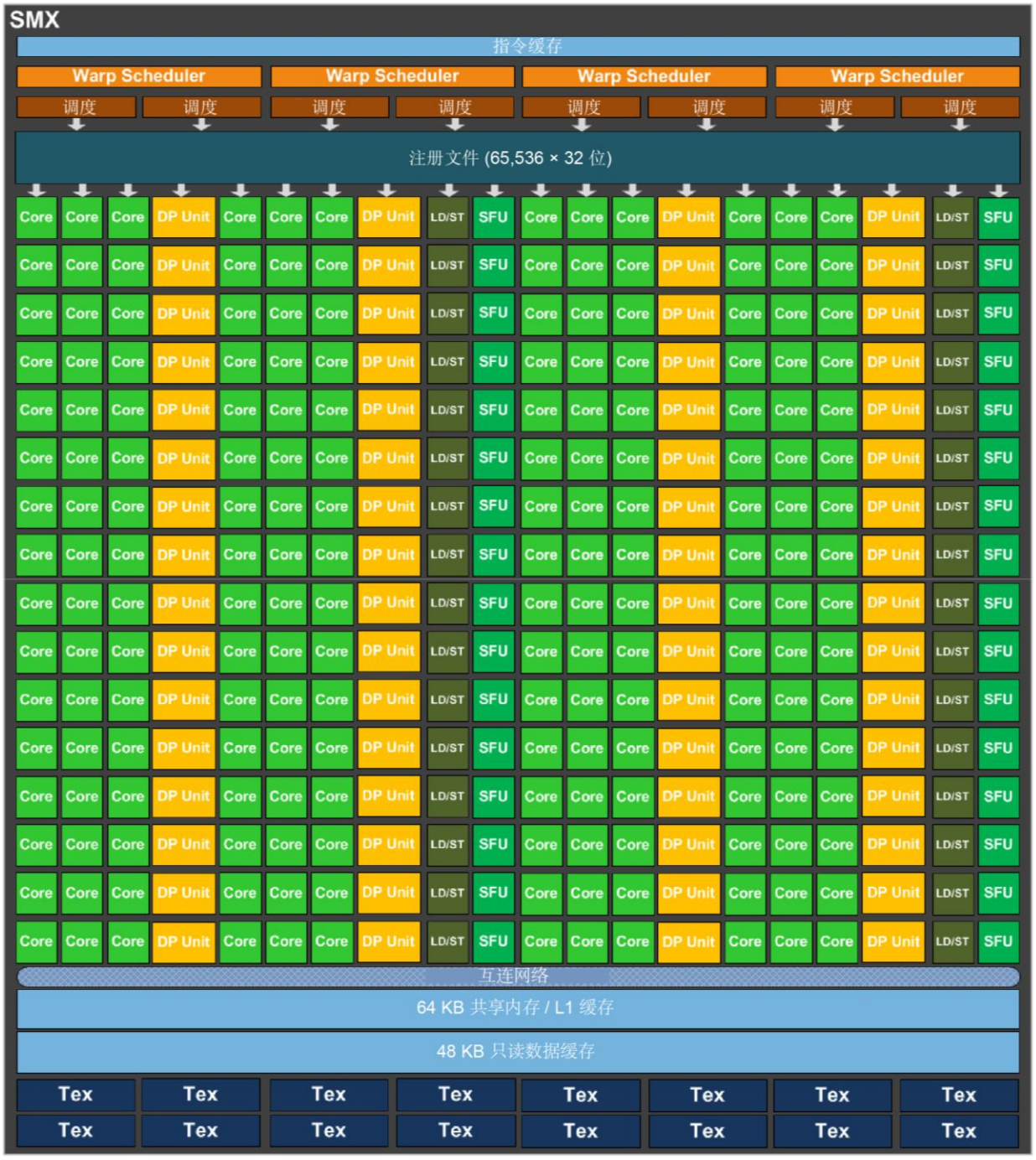
## 性能/功率比

Kepler 架构的一个主要设计目标是提高电源效率。设计 Kepler 时，英伟达工程师应用从 Fermi 中积累的经验，以更好地优化 Kepler 、实现高效运行。台积电的 28nm 制造工艺在降低功耗方面起着重要的作用，但许多 GPU 架构需要修改，以进一步降低功耗，同时保持出色的性能。

Kepler 每一个硬件设备都经过设计和擦洗，以提供卓越的性能/功率比。出色性能/功率比的最佳案例是 Kepler GK110 新流式多处理器 (SMX) 中的设计，与最近 Kepler GK104 引入的 SMX 单元的许多方面类似，但计算算法包括更多双精度单位。

## 流式多处理器 (SMX) 架构

Kepler GK110 的新 SMX 引入几个架构创新，使其不仅成为有史以来最强大的多处理器，而且更具编程性，更节能。




**SMX: 192 个单精度 CUDA 核、64 个双精度单元、32 个特殊功能单元 (SFU) 和 32 个加载/存储单元 (LD/ST)。**

**SMX 处理核架构**

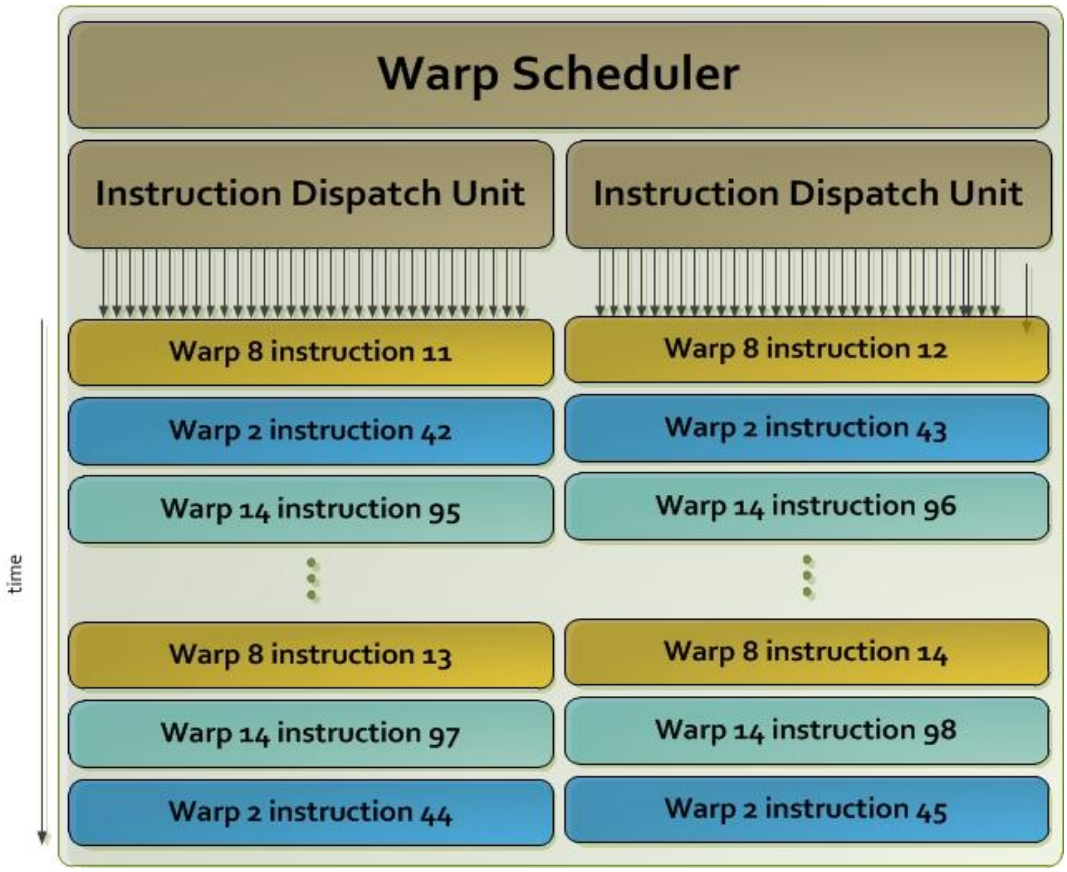
每个 Kepler GK110 SMX 单元具有 192 单精度CUDA 核，每个核完全由 - 浮点和整数算术逻辑单元组成。Kepler 完全保留 Fermi 引入的 IEEE 754-2008 标准的单精度和双精度算术，包括积和熔加 (FMA) 运算。

Kepler GK110 SMX 的设计目标之一是大大提高 GPU 的双精度性能，因为双精度算术是许多 HPC 应用的核心。Kepler GK110 的 SMX 还保留了特殊功能单元 (SFU) 以达到和上一代 GPU 类似的快速超越运算，所提供的 SFU 数量是 Fermi GF110 SM 的 8 倍。

与 GK104 SMX 单元类似，GK110 SMX 单元内的核使用主 GPU 频率而不是 2 倍的着色频率。2x 着色频率在 G80 Tesla 架构的 GPU 中引入，并用于之后所有的 Tesla 和 Fermi - 架构的 GPU。在更高时钟频率上运行执行单元使芯片使用较少量的执行单元达到特定目标的吞吐量，这实质上是一个面积优化，但速度更快的内核的时钟逻辑更耗电。对于 Kepler，我们的首要任务是的性能/功率比。虽然我们做了很多面积和功耗方面的优化，但是我们更倾向优化功耗，甚至以增加面积成本为代价使大量处理核在能耗少、低 GPU 频率情况下运行。

**Quad Warp Scheduler**

SMX 以 32 个并行线程为一组的形式调度进程，这 32 个并行线程叫做 Warp。而每个 SMX 中拥有四组 Warp Scheduler 和八组 Instruction Dispatch 单元，允许四个 Warp 同时发出执行。Kepler 的 Quad Warp Scheduler 选择四个 Warp，在每个循环中可以指派每 Warp 2 个独立的指令。与 Fermi 不同，Fermi 不允许双精度指令和部分其他指令配对，而 Kepler GK110 允许双精度指令和其他特定没有注册文件读取的指令配对 例如加载/存储指令、纹理指令以及一些整数型指令。




**每个 Kepler SMX 包含 4 组 Warp Scheduler，每组 Warp Scheduler 包含两组 Instruction Dispatch 单元。单个 Warp Scheduler 单元如上所示。**

我们努力优化 SMX Warp Scheduler 逻辑中的能源。例如，Kepler 和 Fermi Scheduler 包含类似的硬件单元来处理调度功能。其中包括：

a) 记录长延迟操作（纹理和加载）的寄存器
b) Warp 内调度决定（例如在合格的候选 Warp 中挑选出最佳 Warp 运行）
c) 线程块级调度（例如，GigaThread 引擎）

然而，Fermi 的 scheduler 还包含复杂的硬件以防止数据在其本身数学数据路径中的弊端。多端口寄存器记录板会纪录任何没有有效数据的寄存器，依赖检查块针对记录板分析多个完全解码的 Warp 指令中寄存器的使用情况过，确定哪个有资格发出。

对于 Kepler，我们认识到这一信息是确定性的（数学管道延迟是不变量），因此，编译器可以提前确定指令何时准备发出，并在指令中提供此信息。这样一来，我们就可以用硬件块替换几个复杂、耗电的块，其中硬件块提取出之前确定的延迟信息并将其用于在 Warp 间调度阶段屏蔽 Warp，使其失去资格。

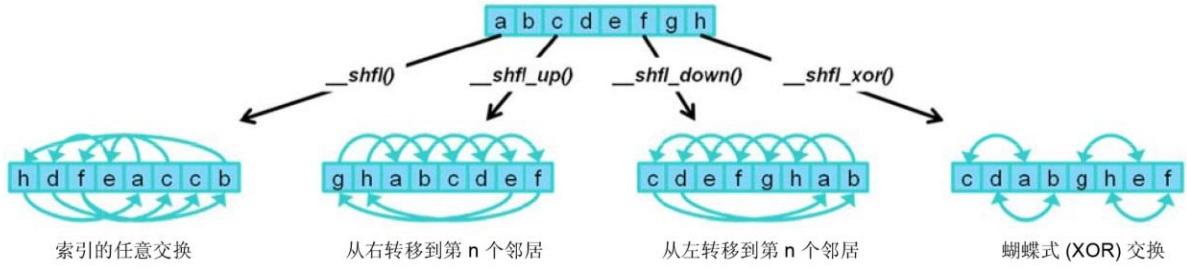
### 新 ISA 编码：每个线程 255 个寄存器

可由线程访问的寄存器的数量在 GK110 中已经翻了两番，允许线程最多访问 255 个寄存器。由于增加了每个线程可用的寄存器数量，Fermi 中承受很大寄存器压力或泄露行为的代码的速度能大大的提高。典型的例子是在 QUDA 库中使用 CUDA 执行格点 QCD（量子色动力学）计算。基于 QUDA fp64 的算法由于能够让每个线程使用更多寄存器并减少的本地内存泄漏，所以其性能提高了 5.3 倍。

### Shuffle 指令

为了进一步提高性能，Kepler 采用 Shuffle 指令，它允许线程在 Warp 中共享数据。此前，Warp 内线程之间的数据共享需要存储和加载操作以通过共享内存传递数据。使用 Shuffle 指令，Warp 可以读取来自Warp 内其他线程中任意排列的值。Shuffle 支持任意索引引用（即任何线程读取任何其他线程）。有用的 Shuffle 子集包括下一线程（由固定量弥补抵消）和 Warp 中线程间 XOR “蝴蝶”式排列，也称为 CUDA 性。

Shuffle 性能优于共享内存，因此存储和加载操作能够一步完成。Shuffle 也可以减少每个线程块所需共享内存的数量，因为数据在 Warp 级交换也不需要放置在共享内存中。在 FFT 的情况下，需要共享一个 Warp 内的数据，通过使用 Shuffle 获得 6%的性能增益。




**此示例表明某些变量可以在 Kepler 中使用 Shuffle 指令。**

### 原子运算

原子内存运算对并行编程十分重要，允许并发线程对共享数据结构执行正确的读 - 修改 - 写运算。原子运算如 add、min、max 和 compare，swap 在某种意义上也是也是原子运算，如果在没有其他线程干扰的情况下执行读、修改和写运算。原子内存运算被广泛用于并行排序、归约运算、建制数据结构而同时不需要锁定线程顺序执行。

Kepler GK110 全局内存原子运算的吞吐量较 Fermi 时代有大幅的提高。普通全局内存地址的原子运算吞吐量相对于每频率一个运算来说提高了 9 倍。独立的全局地址的原子运算的吞吐量也明显加快，而且处理地址冲突的逻辑已经变得更有效。原子运算通常可以按照类似全局负载运算的速度进行处理。此速度的提高使得原子运算足够快得在内核内部循环中使用，消除之前一些算法整合结果所需要的单独的归约传递。Kepler GK110 还扩展了对全局内存中 64 - 位原子运算的本机支持。除了 atomicAdd、atomicCAS 和 atomicExch（也受 Fermi 和 Kepler GK104 支持）之外，GK110 还支持以下功能：

- atomicMin
- atomicMax
- atomicAnd
- atomicOr
- atomicXor

其他不受本机支持的原子运算（例如 64 位浮点原子运算）可以使用 compare - and - swap (CAS) 指令模拟。

**纹理改进**

GPU 的专用硬件纹理单元对于需要取样或过滤图像数据的计算机程序来说是宝贵的资源。Kepler 中的纹理吞吐量与 Fermi 相比有明显提高，每个 SMX 单元包含 16 纹理过滤单元，对比 Fermi GF110 SM 提高了 4 倍。

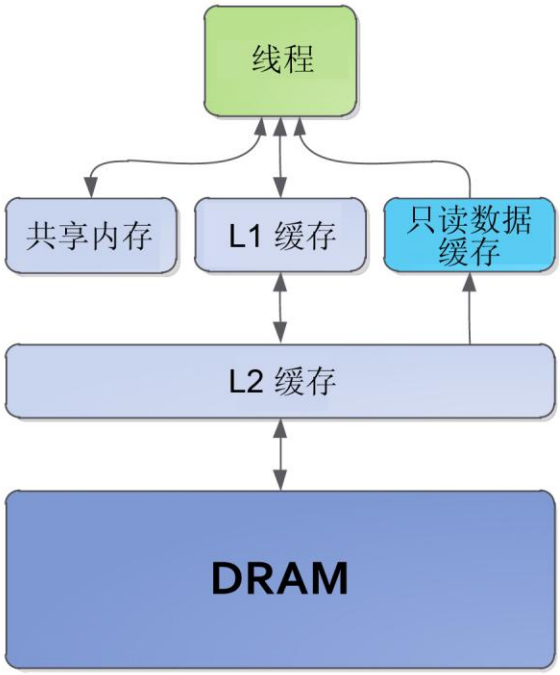
此外，Kepler 改变了管理纹理状态的方法。在 Fermi 时代，为让 GPU 引用纹理，必须在固定大小绑定表中分配“槽”才能启动 Grid。表中槽数量最终限制程序一次可以读取多少个独特的纹理。最终，在 Fermi 中限制程序仅可以同时访问 128 纹理。

Kepler 中有无绑定纹理，不需要额外步骤：纹理状态已保存为内存中的对象，硬件按需获取这些状态对象，绑定表过时。这有效地消除了计算程序引用独特纹理数量的任何限制。相反，程序可以在任何时间映射纹理和通纹理处理周围，因为他们将任何其他指针

# Kepler 内存子系统 – L1、L2、ECC

Kepler 的内存层次结构与 Fermi 类似。Kepler 架构支持统一内存加载和存储的请求路径，每个 SMX 多处理器有一个 L1 缓存。Kepler GK110 还使编译器指示为只读数据增设一个新的缓存，如下所述。

## Kepler 内存层次结构




### 64 KB 可配置共享内存和 L1 缓存

在 Kepler GK110 架构（如在上一代 Fermi 架构）中，每个 SMX 有 64 KB 的片上存储器，可配置为 48 KB 的 共享存储器和 16 KB 的 L1 缓存，或配置为 16 KB 的共享存储器和 48 KB 的 L1 缓存。Kepler 目前在配置共享存储器的分配和 L1 缓存方面的灵活性更大，允许共享存储器和 L1 缓存之间以 32KB/32KB 划分。为了支持 SMX 单元增加的吞吐量，用于 64 位或更大负载运算的共享存储器带宽相对 Fermi SM 也增加一倍，到每主频 256B。

### 48KB 只读 - 数据缓存

除 L1 缓存之外，Kepler 为只读数据引入 48 KB 缓存为了函数的持续时间。在 Fermi 时代，该缓存只能由纹理单元访问。专家程序员通常发现它的优势是通过将数据映射为纹理来加载数据，但这种方法有很多局限性。

在 Kepler 中，除了大大提高了该缓存的容量之外，还伴随着纹理功力的提高，我们决定让缓存为一般负载运算直接访问 SM。使用只读的路径好处极大，因为它使负载和工作组的影响远离共享/L1缓存路径。此外，其他情况下，只读数据缓存更高的标签带宽支持全速非对齐内存访问模式。

该路径的使用是由编译器自动管理（通过参数 C99 访问任何变量或称为常量的数据结构）。标准关键字 “const_restrict” 将被编译器标记以通过只读数据缓存加载。

### 改进的 L2 缓存

Kepler GK110 GPU 具有 1536KB 的专用 L2 缓存内存，是 Fermi 架构中 L2 的 2 倍。L2 缓存是 SMX 单元之间主要数据统一点，处理所有加载、存储和纹理请求并提供跨 GPU 之间有效、高速的数据共享。Kepler 上的 L2 缓存提供的每时钟带宽是 Fermi 中的 2 倍。之前不知道数据地址的算法，如物理求解器、光线追踪以及稀疏矩阵乘法，从高速缓存层次结构中获益匪浅。需要多个 SM 读取相同数据过滤和卷积内核也从中受益。

### 内存保护支持

与 Fermi 相同，Kepler的注册文件、共享内存、L1 缓存、L2 缓存和 DRAM 内存受单错纠正双错检测 (SECDED) ECC 代码保护。此外，只读的数据缓存 - 通过奇偶校验支持单错纠正，在奇偶校验错误的情况下，缓存单元自动使失效，迫使从 L2 读取正确的数据。

ECC 校验位从 DRAM 获取必定消耗一定量的带宽，这会导致启用 ECC和停用 ECC的运算之间的差异，尤其对于内存带宽敏感的应用程序。基于 Fermi 的经验，Kepler GK110 对 ECC 校验位获取处理进行了几项优化。结果，经内部的计算应用测试套件测量，开启和关闭 ECC 的性能三角洲已经平均降低 66%。

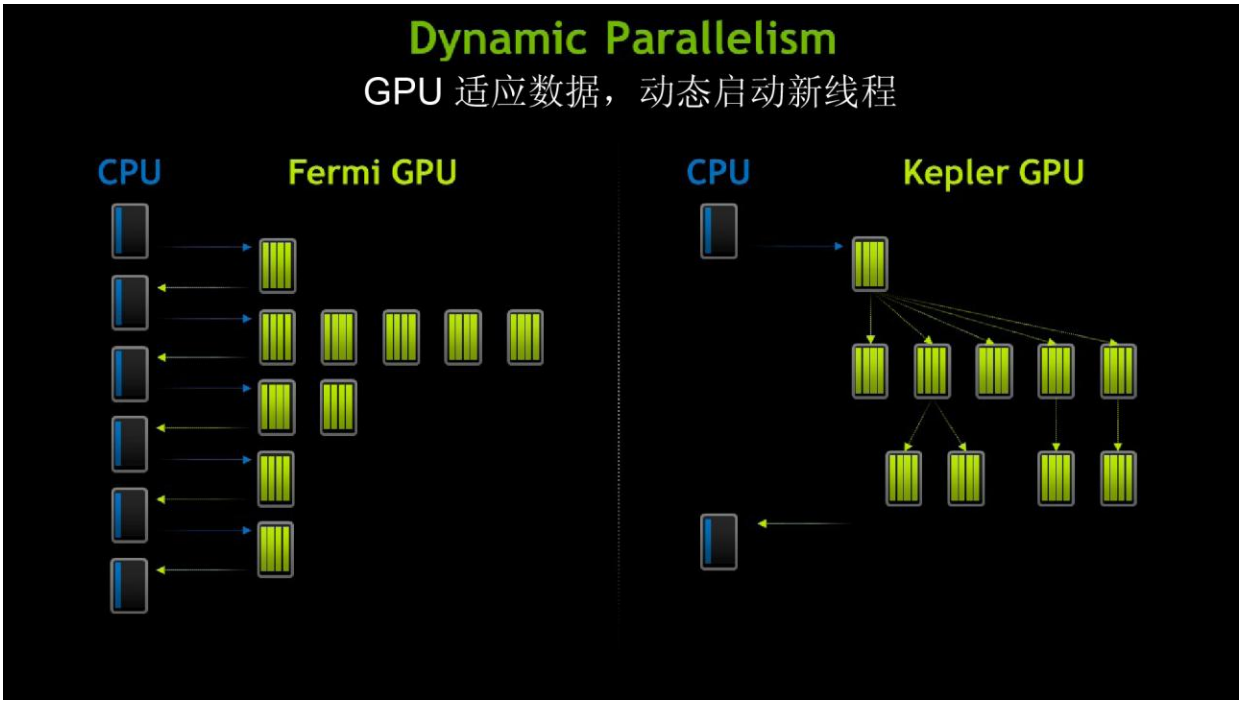
## Dynamic Parallelism

在混合 CPU - GPU 系统中，由于 GPU 的性能/功率比提高，使应用程序中大量并行代码完全在 GPU 高效运行，提高了可扩展性和性能。为了加快应用程序的额外并行部分的处理，GPU 必须支持更加多样化的并行工作负载类型。

Dynamic Parallelism 是 Kepler GK110 引入的新功能，能够让 GPU 在无需 CPU 介入的情况下，通过专用加速硬件路径为自己创造新的工作，对结果同步，并控制这项工作的调度。

在内核启动时，如果问题的规模和参数已知，那么 Fermi 在处理大型并行数据结构时效果非常好。所有的工作是从主机 CPU 启动，会运行到完成，并返回结果返回到 CPU。结果将被用来作为最终的解决方案的一部分，或通过 CPU 进行分析，然后向 GPU 发送额外的处理请求以进行额外处理。

在 Kepler GK110中，任何一个内核都可以启动另一个内核，并创建处理额外的工作所需的必要流程、事件以及管理依赖，而无需主机 CPU 的介入。T该架构能让开发人员更容易创建和优化递归和数据依赖的执行模式，并允许更多的程序直接运行在 GPU 上。可以为其他任务释放系统 CPU，或可以用功能少的 CPU 配置系统以运行相同的工作负载。



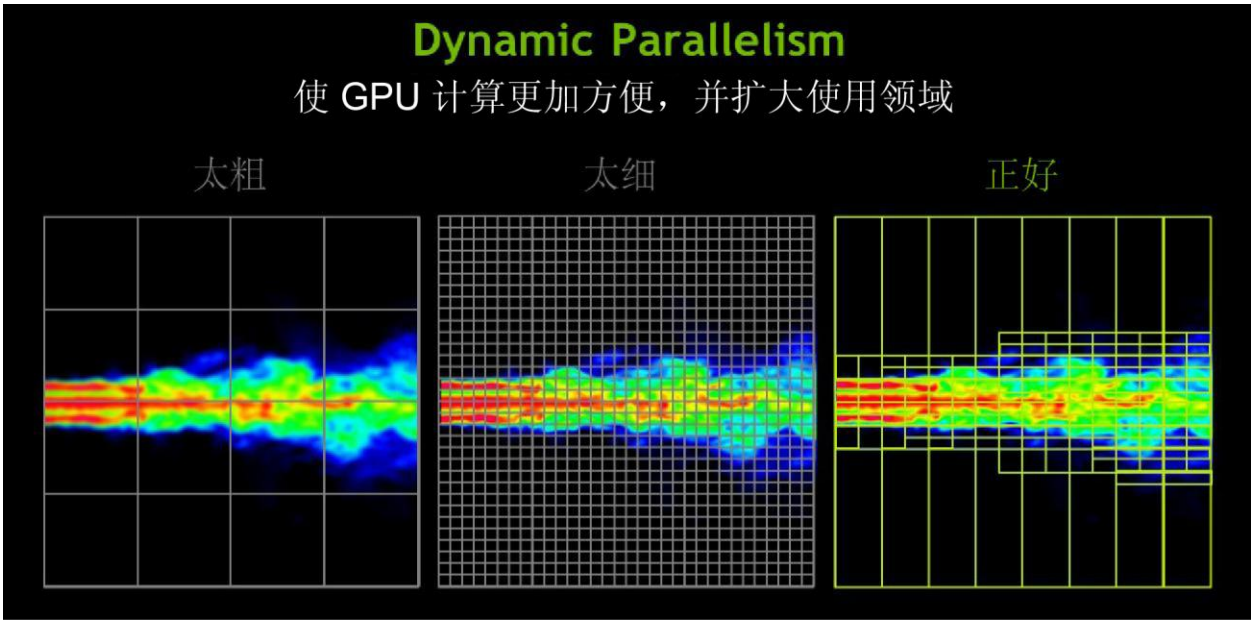

**Dynamic Parallelism 允许应用程序中更多的并行代码直接由 GPU 本身启动（右侧图像），而不需要 CPU 的干预（左侧图像）。**

Dynamic Parallelism 允许更多种并行算法在 GPU 上执行，包括不同数量的并行嵌套循环、串行控制任务线程的并行队或或卸载到 GPU 的简单的串行控制代码，以便促进应用程序的并行部分的数据局部化。

因为内核能够根据GPU 中间结果启动额外工作负载，程序员现在可以智能处理负载平衡的工作，以集中其大量资源在需要处理能力最大或与解决方案最有关的问题上。

一个例子是动态设置数值模拟的 Grid。 通常 Grid 主要集中在变化最大的地区，需要通过数据进行昂贵的前处理。另外，均匀粗 Grid 可以用来防止浪费的 GPU 资源，或均匀细 Grid 可以用来确保捕获所有功能，但这些选项的风险是在不太被注意的地区缺少模拟功能或“过度消费”的计算资源。

有了 Dynamic Parallelism，可以在运行时以数据依赖形式动态确定 - Grid 解决方案。以粗 Grid 开始，模拟“放大”注意的区域，同时避免在变化不大区域中不必要的计算。虽然这可以通过使用一系列的 CPU 启动的内核来完成，但是通过分析数据、作为单个模拟内核部分启动额外工作让 GPU 细化 Grid 本身要简单的多，消除了 CPU 的中断以及CPU和GPU之间的数据传输。



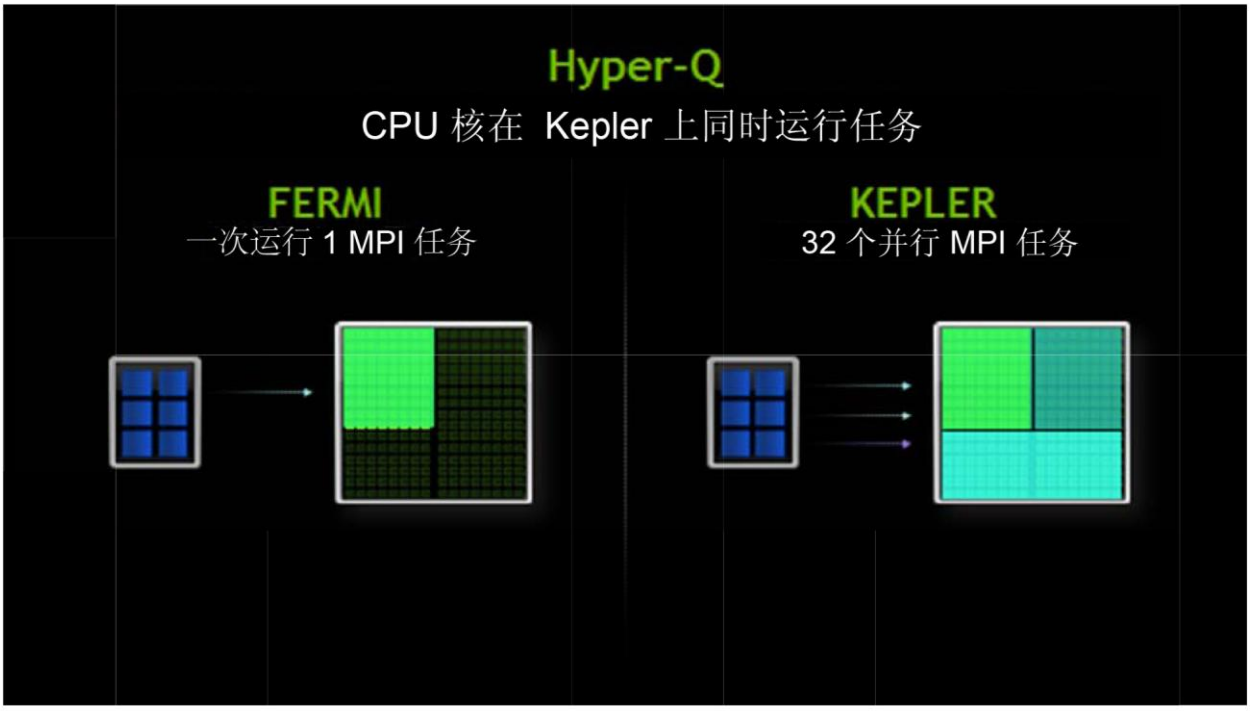

**–图片归属查尔斯·里德**

上面的例子说明了在数值模拟，采用动态调整 Grid 的好处。为了满足峰值的精度要求，固定的分辨率仿真必须运行在整个模拟域过于精细的分辨率上，而多分辨率 Grid 根据当地的变化为每个区域应用正确的模拟分辨率。

## Hyper - Q

原来的一个困难是， GPU 始终要优化调度来自多个数据流的工作负载。Fermi 结构支持从单独数据流的 16 路并发内核启动，但最终数据流都复用相同的硬件工作队列。这允许虚假的数据流内依赖，要求在单独数据流内的其他内核可以执行之前就完成一个数据流内依靠的内核。虽然在某种程度上这可以通过使用广度优先启动顺序缓解，但是随着程序的复杂性的增加，这可以成为越来越难以有效地管理。

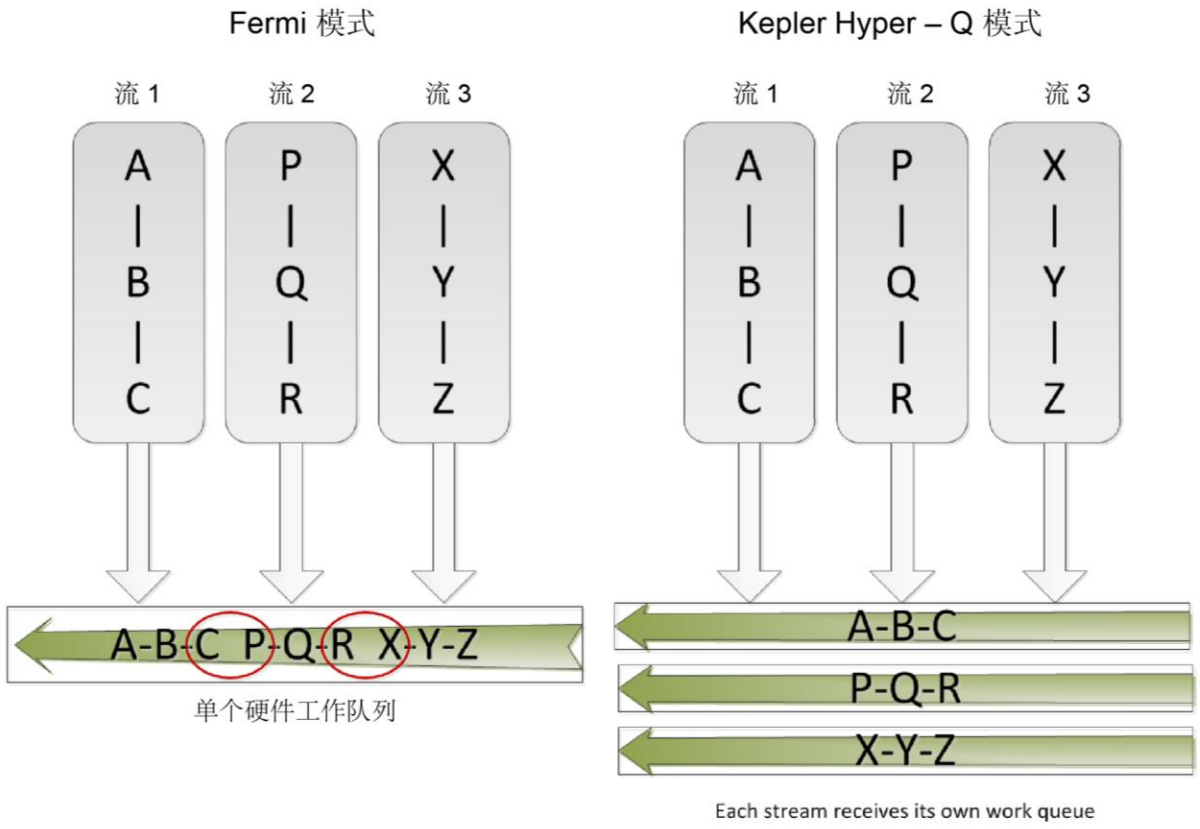
Kepler GK110 使用新 Hyper - Q 特征改进了这一功能。Hyper - Q 允许 32 个并发，硬件管理的连接（ 对比 Fermi 的单一连接），增加了主机和 GPU 中 CUDA Work Distributor （CWD）逻辑之间的连接总数（工作队列)。Hyper - Q 是一种灵活的解决方案，允许来自多个 CUDA 流、多个消息传递接口（MPI）进程，甚至是进程内多个线程的单独连接。以前遇到跨任务虚假串行化任务的应用程序，限制了 GPU 的利用率，而现在无需改变任何现有代码，性能就能得到 32 倍的大幅度提升。




**Hyper - Q 允许 CPU 和 GPU 之间更多的并发连接。**

每个 CUDA 流在其自己硬件工作队列管理，优化流间的依赖关系，一个流中的运算将不再阻止其他流，使得流能够同时执行，无需特别定制的启动顺序，消除了可能的虚假依赖。

Hyper‐Q 在基于 MPI 的并行计算机系统中使用会有明显的优势。通常在多核 CPU 系统上运行时创建传统基于 MPI‐的算法，分配给每个 MPI 进程的工作量会相应地调整。这可能会导致单个 MPI 进程没有足够的工作完全占据 GPU。虽然一直以来多个 MPI 进程都可以共享 GPU，但是这些进程可能会成为虚假依赖的瓶颈。Hyper‐Q 避免了这些虚假的依赖，大大提高了 MPI 进程间共享 GPU 的效率。



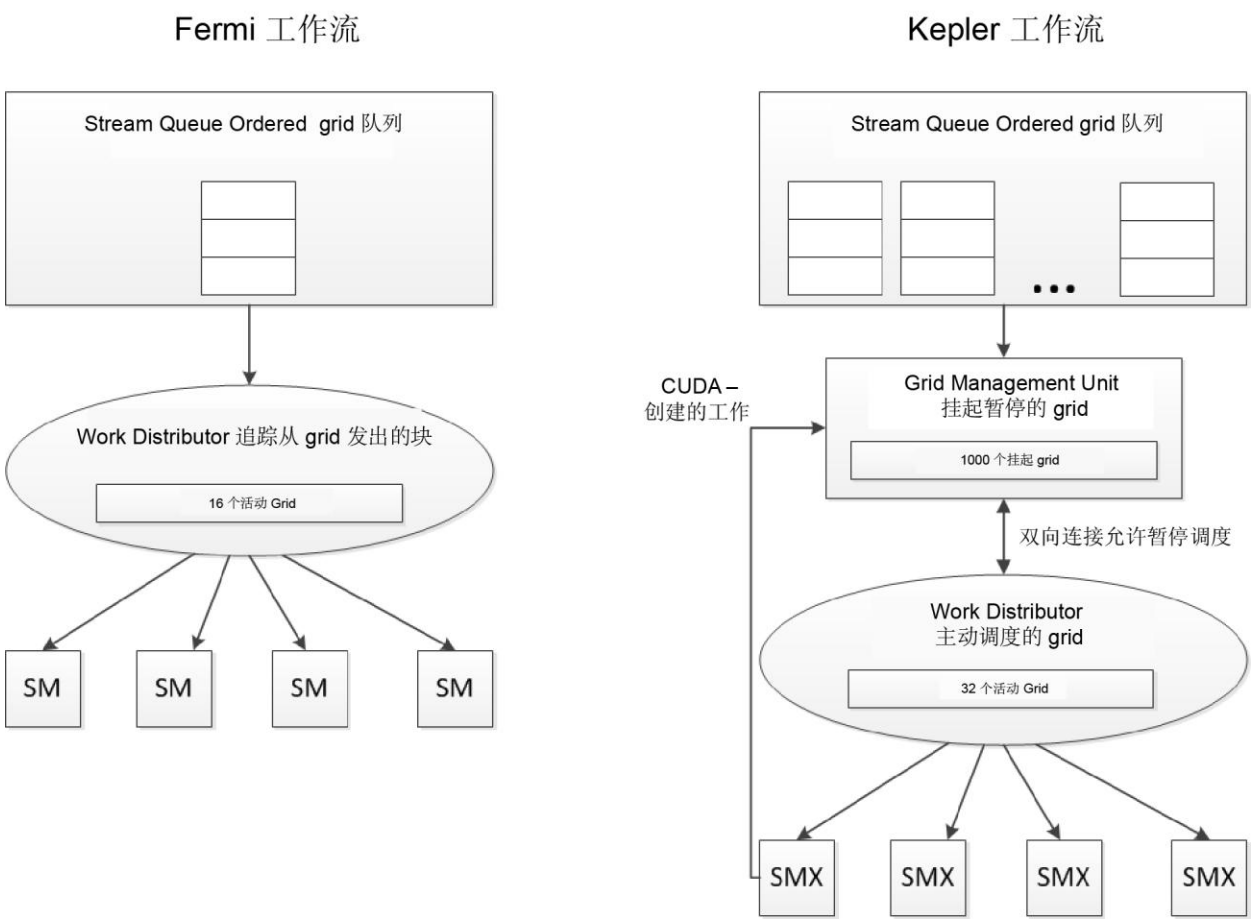

**Hyper‐Q 与 CUDA 流一起工作：左侧显示 Fermi 模式，仅 (C,P) 和 (R,X) 可以同时运行，因为单个硬件工作队列导致的流内依赖。Kepler Hyper‐Q 模式允许所有流使用单独的工作队列同时运行。**

## Grid Management Unit – 有效地保持 GPU 的利用率

Kepler GK110 中的新功能，如 CUDA 内核能够利用 Dynamic Parallelism 在 GPU 上直接启动工作，需要 Kepler 中 CPU - to - GPU 工作流提供比 Fermi 设计增强的功能。Fermi 中，线程块的 Grid 可由 CPU 启动，并将一直运行到完成，通过 CUDA Work Distributor (CWD) 单元创建从主机到 SM 的简单单向工作流。Kepler GK110 目的是通过 GPU 有效管理 CPU 和 CUDA 创建的工作负载来改进 CPU - 到 - GPU 的工作流。

我们讨论了 Kepler GK110 GPU 允许内核直接在 GPU 上启动工作的能力，重要的是要理解在 Kepler GK110 架构所做的变化，促成了这些新功能。Kepler 中，Grid 可从 CPU 启动，就和 Fermi 的情况一样，但是新 Grid 还可通过编程由 CUDA 在 Kepler SMX 单元中创建。要管理 CUDA 创建的 Grid 和主机生成的 Grid，在 Kepler GK110 中引入新 Grid Management Unit (GMU)。该控制单元管理并优先化传送到 CWD 要发送到 SMX 单元执行的 Grid。

Kepler 中的 CWD 保留准备好调度的 Grid，并能调度 32 个活动的 Grid，这是 Fermi CWD 容量的两倍。Kepler CWD 通过双向链接进行通信，允许 GMU 暂停新 Grid 的调度并保留挂起和暂停的 Grid，直到需要。GMU 也有到 Kepler SMX 单元的直接连接，允许 Grid 通过 Dynamic Parallelism 在 GPU 上启动其他工作，以将新工作传回到 GMU 进行优先化和调度。如果暂停调度的额外工作量的内核，GMU 将保持其为不活动，知道以来工作完成。




**重新设计的 Kepler HOST 到 GPU 的工作流显示新 Grid Management Unit，允许其管理主动调度的 Grid、暂停调度、保留挂起和暂停的 Grid。**

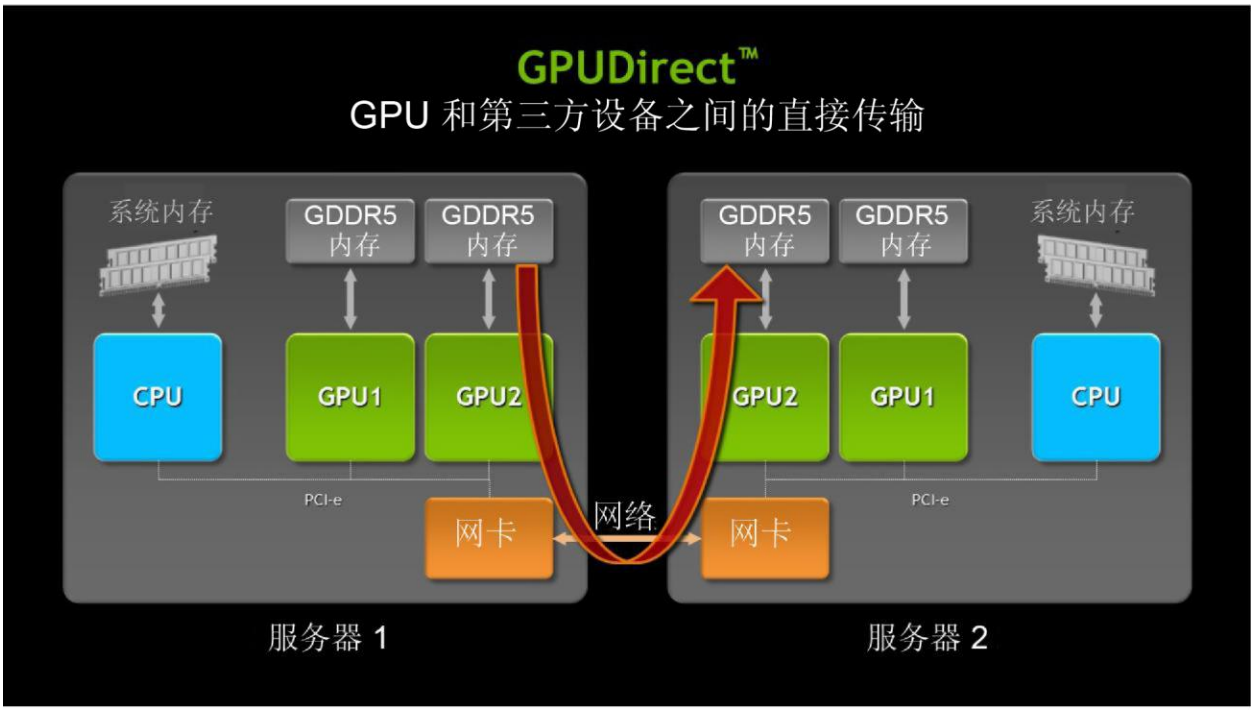
## 英伟达™（NVIDIA®）GPUDirect™

当处理大量的数据时，提高数据吞吐量并降低延迟，对于提高计算性能是至关重要的。Kepler GK110 支持英伟达™ GPUDirect 中的 RDMA，目的是通过允许第三方设备，如 IB 适配器、NIC 和 SSD，直接访问 GPU 内存 - 来提高性能。使用 CUDA 5.0 时，GPUDirect 提供以下重要功能：

- 无需 CPU方面的数据缓冲, NIC 和 GPU 之间的直接内存存取 (DMA)
- 显着改善 GPU和其他网络节点之间的 MPISend/ MPIRecv 效率。
- 消除了 CPU 带宽和延迟的瓶颈
- 与各种第三方网络、捕获和存储设备一起工作

如逆时偏移（用于石油和天然气勘探地震成像）这样的应用程序，将大量影像数据分布在多个 GPU。数以百计的 GPU 必须合作，以紧缩的数据，经常通信中间结果 GPUDirect 利用 P2P 和 RDMA 功能为服务器内或服务器之间“ GPU - 到 - GPU” 的通信的情况分配更高的总带宽。

Kepler GK110 还支持其他功能 GPUDirect，如 Peer - to - Peer 和 GPUDirect for Video。




**GPUDirect RDMA 允许网络适配器这样的第三方设备访问 GPU 内存， - 这还转换为跨节点 GPU 之间直接传输。**

# 结束语

随着 Fermi 在 2010 年的推出，英伟达引入高性能计算 (HPC) 行业的新时代，而这基于 CPU 和 GPU 协同工作解决计算密集型工作负载的混合计算模式。如今，有了 Kepler GK110 GPU，英伟达又一次提高了 HPC 行业的标准。

Kepler GK110 设计的初衷就是利用卓越的电源效率达到最大化计算性能和吞吐量。该架构有很多创新，如 SMX、Dynamic Parallelism 和 Hyper - Q 使混合计算大大简化和加快了编程，适用于更广泛的应用。Kepler GK110 GPU 将用于许多系统，从工作站到超级计算机，解决 HPC 中最严峻的挑战。

# 附录 A - CUDA 快速回顾

CUDA 组合的硬件/软件平台，让英伟达™ GPU 执行使用 C、C++、Fortran 和其他语言编写的程序。CUDA 程序调用称为内核的并行功能，跨许多线程执行。程序员或编译器将这些线程组织到线程块和线程块 Grid，如图 1 所示。线程块内的每个线程执行的内核的一个实例。每个线程在其线程块和 Grid、程序计数器、寄存器、逐线程专有存储器、输入和输出结果内都有线程 ID 和块 ID。

线程块是一组并发执行的线程，可以通过屏障同步和共享内存相互合作。线程块在其 Grid 中有块 ID。Grid 是一个线程块阵列，执行相同的内核，从全局内存读取输入，将结果写入全局内存，并同依赖内核调用。在 CUDA 的并行编程模式中，每个线程都有逐线程专有存储器空间，用于寄存器溢出，函数调用和 Cautomatic的数组变量。每个线程块都有逐线程专有存储器空间，用于用于线程间通信、数据共享、在并行算法中共享结果。线程块 Grid 在内核进行广泛的全局同步后在全局内存空间中共享结果。

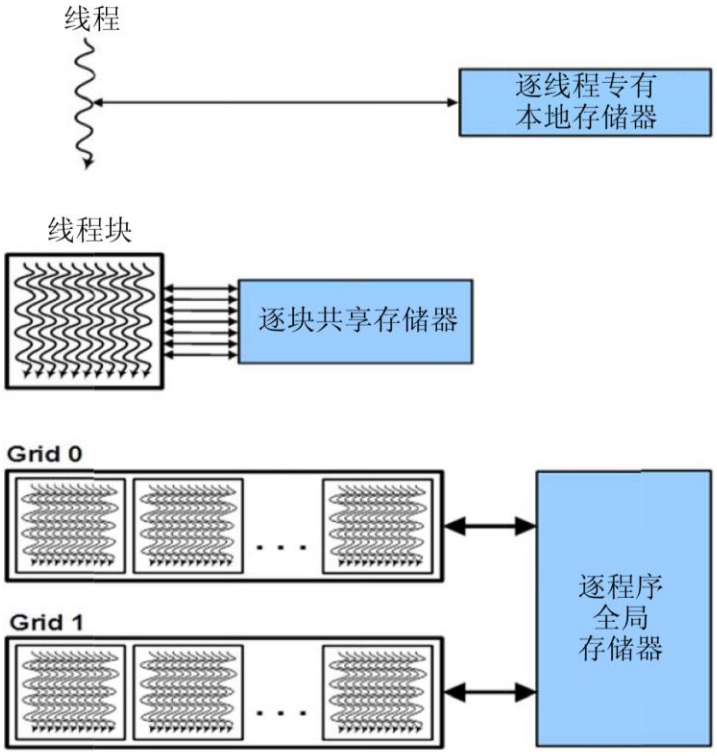


**图 1：线程、块和 Grid 的 CUDA 层次结构，以及对应的逐线程专有存储器空间、逐块共享存储器空间和逐应用程序全局存储器空间。**

## CUDA 硬件执行

线程的 CUDA 层次结构映射到 GPU 上处理器的层次结构；一个 GPU 执行一个或多个 Grid；流式多处理器（ Fermi 上的 SM / Kepler 上的 SMX）执行一个或多个线程块；SMX 中的 CUDA 核和其他执行单元执行线程指令。SMX 以 32 个线程为一组的形式执行，这 32 个线程叫做 Warp。虽然程序员一般可以忽略 Warp 执行功能的正确性，并专注于个别标线程编程，但他们可以通过让 Warp 中的线程执行相同的代码路径和存取附近的地址的存储器而大大提高性能。

## 注

本白皮书提供的所有信息，包括评论、意见、英伟达™ （NVIDIA®）设计规格、公版显卡、文件、图纸、诊断、列表和其它文件（无论统称还是单论都可称为“材料”）均“按本文编撰时的实际情况”表述。英伟达™（NVIDIA®）不对这些材料做出任何明确、暗示、法定或其它方式的担保，并明确拒绝承担任何暗示的不侵权、适销性和特定用途适用性担保责任。

我们认为，本文中所提供信息均准确可靠。然而，对于因使用此类信息导致的后果，或因使用信息导致侵犯专利权或任何第三方权利的情形，英伟达公司不承担任何责任。本文没有暗示或以任何其它形式提供英伟达公司专利或专利权的许可。本文提及的规格随时可能更改，恕不另行通知。本文将取代之前所提供的所有内容。未经英伟达公司明确书面批准，英伟达公司产品不得被用作救生设备或系统的关键组件。