CUDA编程模型综述

- 吕文龙
- 14210720082

摘要

介绍了通用GPU编程以及CUDA出现的背景, CUDA C语言语法以及CUDA在GPU上的执行模型. 介绍了最新版本CUDA的一些特性以及基于CUDA的并行计算扩展thrust库. 介绍了用于profile CUDA程序的工具. 并以一个蒙特卡洛算法为例, 展示了GPU编程的加速比.

背景介绍: GPU, GPGPU 与 CUDA

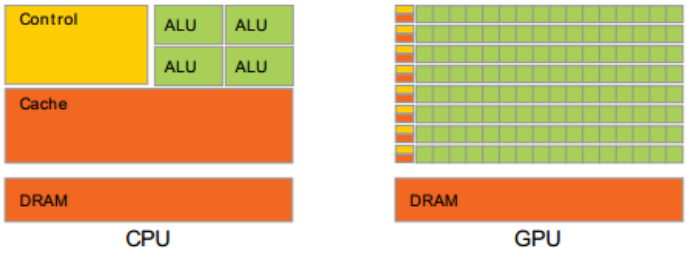
GPU 与 CPU 特性对比

GPU最初是作为CPU的协处理器, 为了加速图形计算而设计的. 图形计算要求对大量像素点的重复高密度计算, 此种需求造成了CPU与GPU在设计时侧重点的差异.

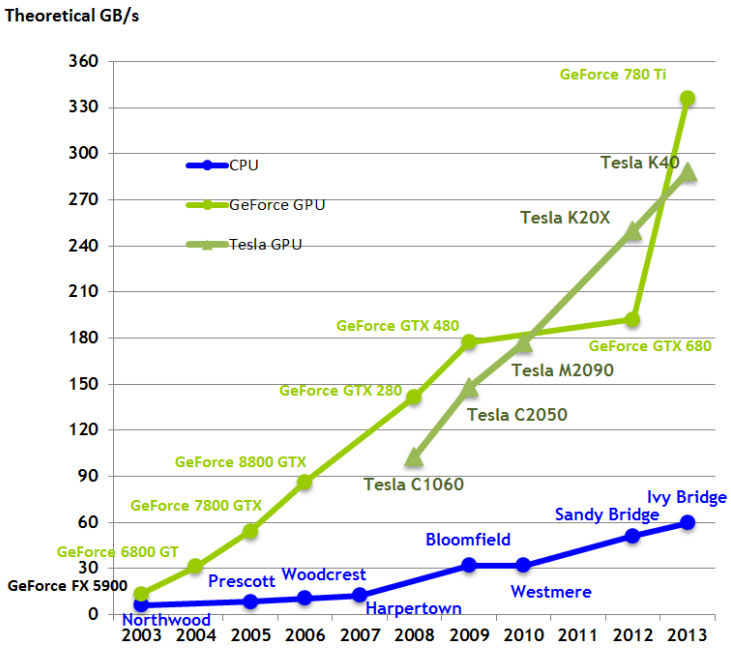
GPU相对CPU主要有以下三点优势:

- 更强的浮点数计算能力(与之对应的是更弱的逻辑控制功能)
- 更大的内存带宽
- 轻量级的线程切换

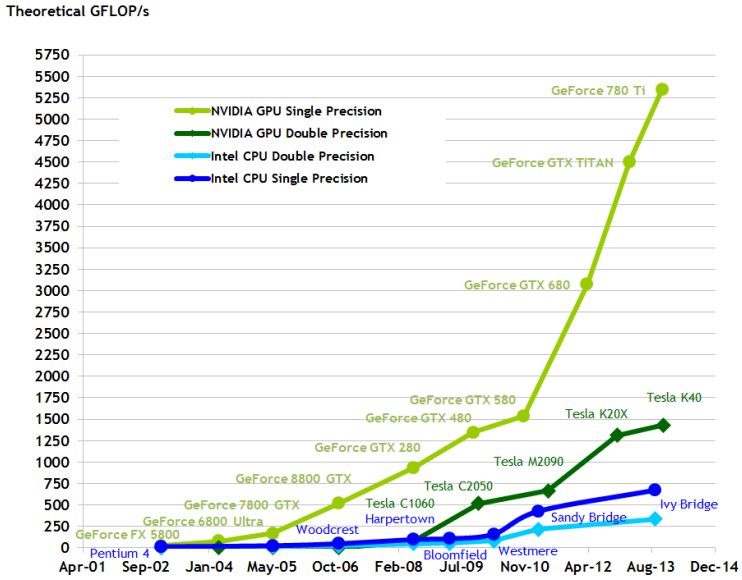
具体而言, 相比GPU, CPU需要处理更加通用的, 普适性的任务. 因而在设计时, CPU更侧重于逻辑操作, 会将大量的硬件用于控制逻辑以及缓存, 而GPU则更侧重高密度的浮点数运算. 因此在硬件设计上, 会更倾向于添加大量并行的运算单元,下图为CPU与GPU的硬件组成对比图:



设计上的差异, 使得在浮点数处理方面, 同时代的GPU往往远胜CPU, 下图为CPU-GPU的浮点数处理能力(GFLOPS)对比:



下图为不同时代CPU与GPU的内存带宽对比:



CPU的一个核心(或者通过超线程技术虚拟将一个核心虚拟成的多个核心). 同一时刻通常只能运行一个线程的指令, 当线程数超过核心数时, 多个核心共享计算资源, 而当进行线程切换时, 必须要花费大量的clock cycle来保存上下文.

而GPU上具有大量的核心, 并且能够通过硬件来管理线程, 可以实现零开销的线程切换. 当一个线程因为访存或者线程同步而等待时, 可以在下一个时钟周期立刻切换到准备计算的线程.

目前的GPU上没有很复杂的缓存机制, 也没有为单个计算单元设计复杂的流水线等指令级并行机制(至少目前为止这一点仍然是成立的, 不过随着GPU应用领域不再局限于图形处理方面而转向各种通用计算领域, GPU上应该也会出现强大的缓存与 Instruction level parallism 机制). 它主要通过零开销的线程切换来隐藏延迟. 正式由于这个特性, 在后面测试的例子中可以看到, 要使GPU达到加速比, GPU上同时运行的线程应该是数倍于GPU的计算单元(SP).