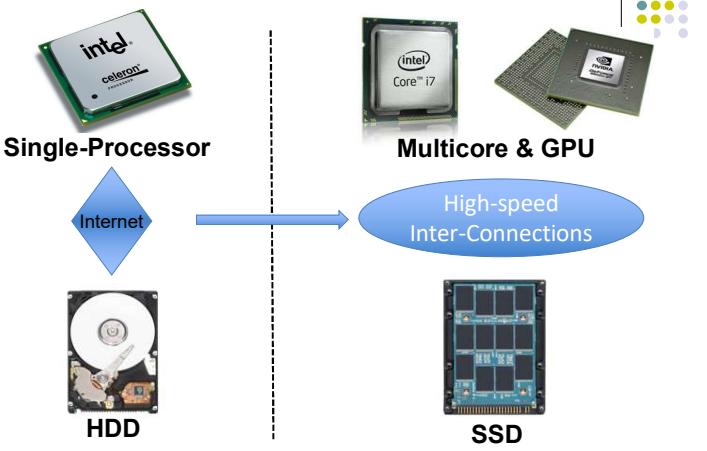
Big Data Processing with GPUs

毛 波
厦门大学 信息学院



异构计算

- 处理器芯片经历了从专用到通用，再从通用到专用的2次转变。其中，可存储指令的冯诺依曼体系和1971年X86 CPU的诞生是第一次转折的诱因；摩尔定律的减速和以GPU为代表的异构运算的崛起是第二次转折的诱因。
- 异构时代，芯片需集成多个模块来满足不同的需求。例如汽车芯片集成了GPU、CPU、NPU等至少10种处理单元。

课堂问题

- 你知道GPU市场占有率最高的公司是？

https://www.bilibili.com/s/video/BV1kb4y1U7Re

TOP 500 List (1-5)

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70,870.0	93,750.0	2,589

TOP 500 List (6-10)

6	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646
7	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
8	**JUWELS Booster Module** - Bull Sequana XH2000 , AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation ClusterSuite, Atos Forschungszentrum Juelich (FZJ) Germany	449,280	44,120.0	70,980.0	1,764
9	**HPC5** - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband, DELL EMC Eni S.p.A. Italy	669,760	35,450.0	51,720.8	2,252
10	**Voyager-EUS2** - ND96amsr_A100_v4, AMD EPYC 7V12 48C 2.45GHz, NVIDIA A100 80GB, Mellanox HDR Infiniband, Microsoft Azure Azure East US 2 United States	253,440	30,050.0	39,531.2	

主要内容

Ⅰ. Introduction to GPU

Ⅱ. GPU Architecture

Ⅲ. CUDA Programming

Part Ⅰ Introduction to GPU

1. GPU的发展

2. CPU和GPU比较

3. GPU的应用和资源

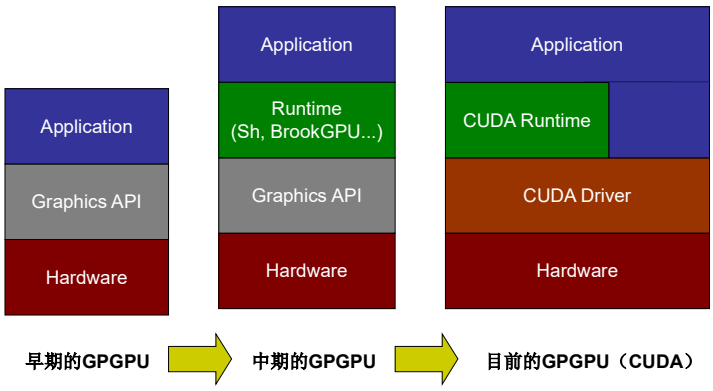
1.1 GPU与GPGPU

- 图形处理器(GPU, Graphics Process Unit)
  - 发展速度超过CPU
  - 今天的GPU不仅具备高质量和高性能图形处理能力，还可用于通用计算
- 用于通用计算的GPU(General-Purpose Computing on GPU，GPGPU)
  - 随着内部单元数量的快速增长及可编程性的持续改进，已经演化成为一个新型的并行计算平台
  - 一个必须引起重视的研究领域和技术

1.2 GPU的发展阶段

- 第一代GPU(～1999年)：部分功能从CPU分离实现硬件加速
  - GE(Geometry Engine)为代表，只能起到3D图像处理的加速作用，不具有软件编程特性
- 第二代GPU(1999年-2002年)：进一步硬件加速和有限编程性
  - 1999年NVIDIA GeForce 256将T&L(Transform and Lighting)等功能从CPU分离出来，实现了快速变换
  - 2001年NVIDIA和ATI分别推出的GeForce3和Radeon 8500，图形硬件的流水线被定义为流处理器，出现了顶点级可编程性，同时像素级也具有有限的编程性，但GPU的编程性比较有限
- 第三代GPU(2002年以后)：方便的编程环境(如CUDA)
  - 2002年ATI发布的Radeon 9700和2003年NVIDIA GeForce FX的推出
  - 2006年NVIDIA与ATI分别为推出了CUDA(Computer Unified Device Architecture，统一计算架构)编程环境和CTM(Close To the Metal)编程环境

1.2 GPGPU的发展阶段 Cont.



1.3 GPGPU的时代已到来

- 随着GPU可编程性不断增强，特别是CUDA等编程环境的出现，使GPU通用计算编程的复杂性大幅度降低。
- 由于可编程性、功能、性能不断提升和完善，GPU已演化为一个新型可编程高性能并行计算资源。
- 全面开启GPU面向通用计算的新时代已到来。

Part Ⅰ Introduction to GPU

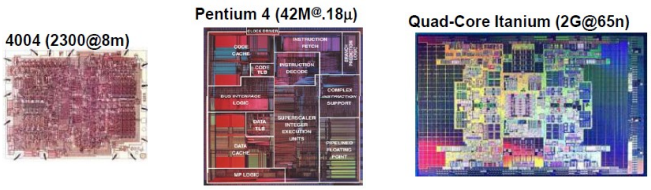
1. GPU的发展

2. CPU和GPU比较

3. GPU的应用和资源

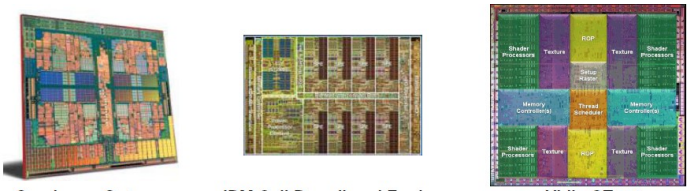
2.1 单核时代的摩尔定律

- CPU时钟频率每18个月翻一番
- CPU制造工艺逐渐接近物理极限
- 功耗和发热成为巨大的障碍



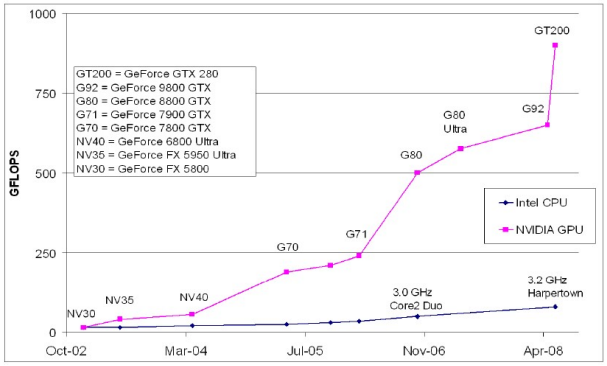
2.2 GPU是多核技术的代表之一

- 在一块芯片上集成多个较低功耗的核心
- 单个核心频率基本不变（一般在1-3GHz）
- 设计重心转向到多核的集成技术
- GPU是一种特殊的多核处理器

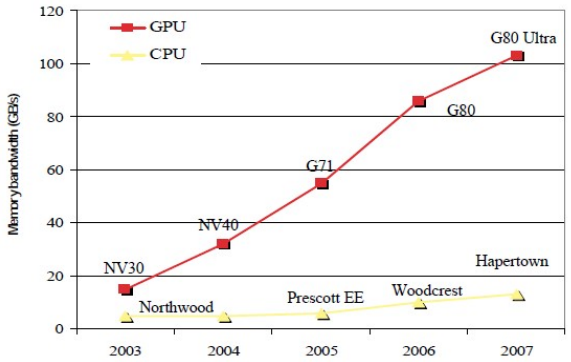


Quad-core Opteron　IBM Cell Broadband Engine　nVidia GT200

2.3 GPU和CPU浮点计算能力对比



2.4 GPU和CPU存储器带宽对比



2.5 GPGPU的优势

- CPU：更多资源用于缓存和逻辑控制
- GPU：更多资源用于计算，适用于高并行性、大规模数据密集型、可预测的计算模式。

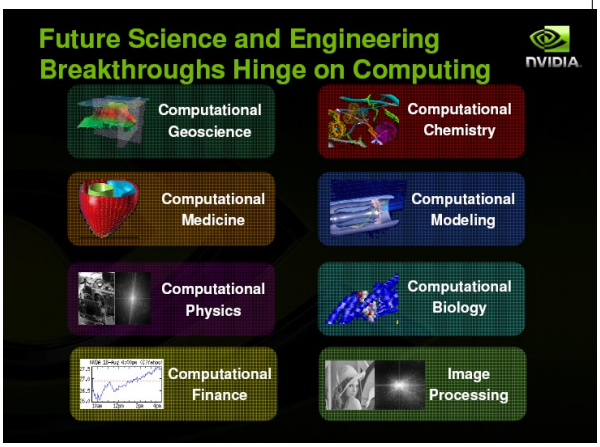
Part Ⅰ Introduction to GPU

1. GPU的发展

2. CPU和GPU比较

3. GPU的应用和资源

3.1 GPU的应用



3.2 GPU的资源

- NVIDIA CUDA Homepage
  - Contains downloads, documentation, examples and links
  - http://www.nvidia.com/cuda
- Programming Guide
- CUDA Forums
  - http://forums.nvidia.com
  - The CUDA designers actually read and respond on the forum
- Supercomputing 2007 CUDA Tutorials
  - http://www.gpgpu.org/sc2007/
- CUDA中文网站
  - http://www.cuda.net/zone_tech.html

主要内容

Ⅰ. Introduction to GPU

Ⅱ. GPU Architecture

Ⅲ. CUDA Programming

Part Ⅱ GPU Architecture

1. 已有的两类GPU结构

2. 存储器层次结构

3. 线程组织结构

1.1 支持通用计算的两类GPU结构

- 基于流处理器阵列的主流GPU结构
  - 以NVIDIA的GeForce8800GTX和ATI的HD 2900为代表
  - GeForce 8800GTX包含了128个流处理器，HD 2900包含了320个流处理器。这些流处理器可以支持浮点运算、分支处理、流水线、SIMD（Single Instruction Multiple Data，单指令流多数据流）等技术。
- 基于通用计算核心的GPU结构
  - Intel Larrabee核心是一组基于x86指令集的CPU核，CPU核拓展了x86指令集，并包含大量向量处理操作和若干专门的标量指令，同时还支持子例程以及缺页中断。
- 前者相对于后者具有更高的聚合计算性能，而后者则在可编程性上具有更大的优势。

1.2 基于流处理器阵列的GPU结构图

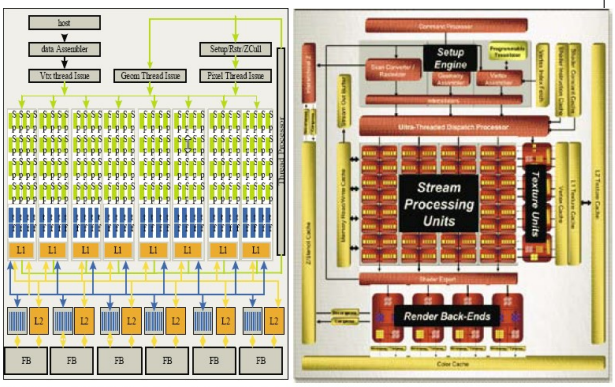


图1 GeForce 8800GTX（左）、HD 2900（右）

1.3 基于通用计算核心的GPU结构图

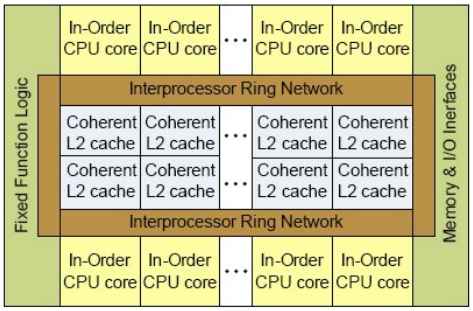
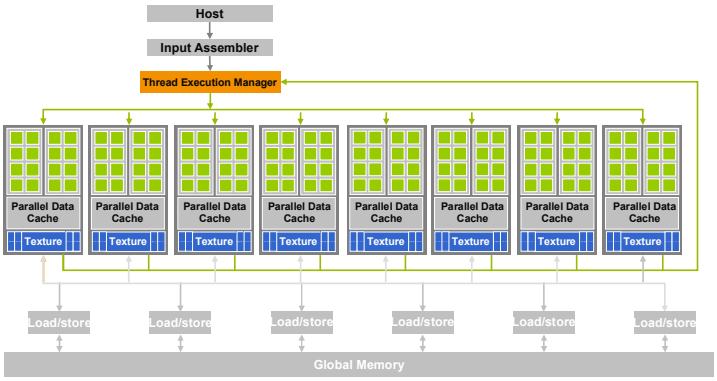


图2 Larrabee多核结构示意图

1.4 NVIDIA G80系列细解：Device Architecture



1.4 NVIDIA G80系列细解：SM(stream multiprocessor)

- GPU主要的组成单元，共16个SM
- 每个SM包含
  - 8个SP(scalar processor)，主频为1.35GHZ，所有SP受控同一个指令单元，同步执行
  - 两个SFU(special function unit)
  - 一个指令cache(I cache)
  - 一个常数cache(C cache) 8KB
  - 一个纹理cache(T cache) 6~8KB
  - 一个多线程发射单元(MT issue)
  - 一个16KB的shared memory，用于线程块内共享数据，访存速度很快
  - 8192个32位字大小的寄存器文件供共享
- 线程的创建、管理和执行由硬件调度，调度本身没有额外开销。

Part Ⅱ GPU Architecture

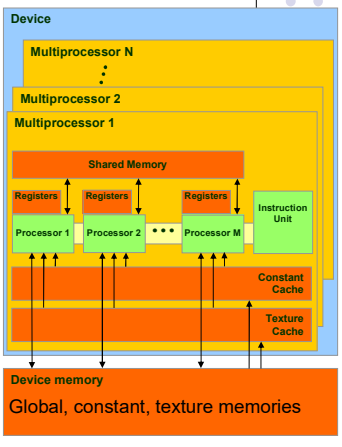
1. 已有的两类GPU结构

2. 存储器层次结构

3. 线程组织结构

2.1 存储器层次结构

- The local, global, constant, and texture spaces are regions of device memory
- Each multiprocessor has:
  - A set of 32-bit registers per processor
  - On-chip shared memory
    - Where the shared memory space resides
  - A read-only constant cache
    - To speed up access to the constant memory space
  - A read-only texture cache
    - To speed up access to the texture memory space

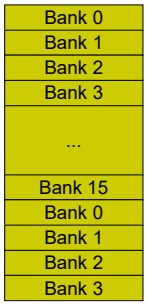


2.2 基本访存开销

存储器类型	位置	是否被缓存	访问速度
寄存器（Registers）	芯片上	不被缓存	几乎没有额外延迟
共享存储器（Share Memory）	芯片上	不被缓存	同寄存器
全局存储器（Device Memory）	设备上	不被缓存	400-600时钟周期
本地存储器（Local Memory）	设备上	不被缓存	400-600时钟周期
固定存储器（Constant Memory）	设备上	被缓存	被缓存时：同寄存器 未被缓存：400-600时钟周期
纹理存储器（Texture Memory）	设备上	被缓存	被缓存时：同寄存器 未被缓存：400-600时钟周期

2.3 共享存储器与存储体冲突

- 访问共享存储器速度很快，只要不存在存储体冲突（Bank Conflict），其速度与寄存器一样
- 共享存储器划分：分为16个存储体(bank)，每个bank按连续4byte循环分配的。不同bank的数据可以并发访问。
- 存储体冲突：同一个bank的访问请求被序列化，造成多倍的访问延迟。
  - 例外：对同一个内存地址的访问使用广播方式，不会造成额外延迟

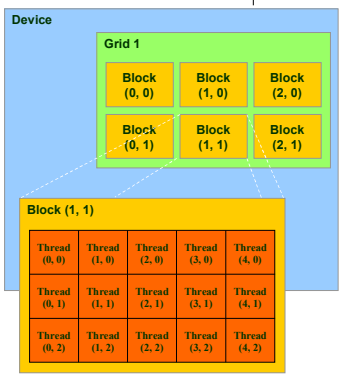


Part Ⅱ GPU Architecture

1. 已有的两类GPU结构
2. 存储器层次结构
3. 线程组织结构

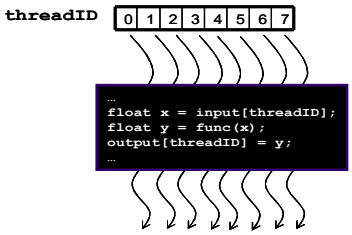
3.1 CUDA中的线程层次

- 线程：
  - CUDA中的基本执行单元；
  - 硬件支持，开销很小；
  - 所有线程执行相同的代码（STMD）
- 线程块：
  - 若干线程还可以组成块(Block，每个块至多512个线程)
  - 线程块可以呈一维、二维或者三维结构
  - 每个线程块分为若干个组(称为warp)，每个warp包含32个线程，物理上以SIMD方式并行
- 线程网格：
  - 若干个线程块可以组织成网格grid
  - Grid可以是一维或二维结构



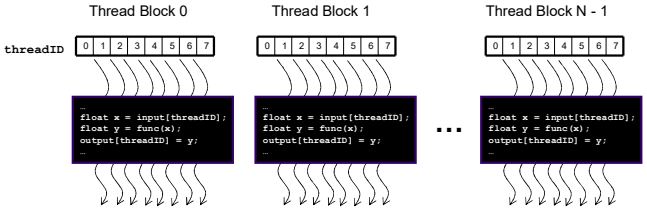
3.2 线程块ID和线程ID

- Thread id:
  a) local id: thread id in a block
  b) global id: thread id in a grid
- Compute thread global id : blockDim*blockId+threadId
- Each thread uses IDs to decide what data to work on



3.3 线程块中的线程合作

- Divide monolithic thread array into multiple blocks
  - Threads within a block cooperate via **shared memory, atomic operations** and **barrier synchronization**
  - Threads in different blocks cannot cooperate



主要内容

Ⅰ. Introduction to GPU

Ⅱ. GPU Architecture

Ⅲ. CUDA Programming

Part Ⅲ CUDA Programming

1. CUDA软件架构
2. CUDA编程语言
3. 内核函数
4. 运行时API

1 CUDA的软件架构

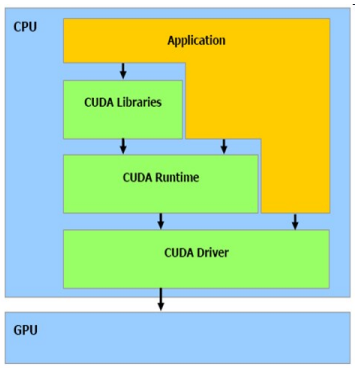


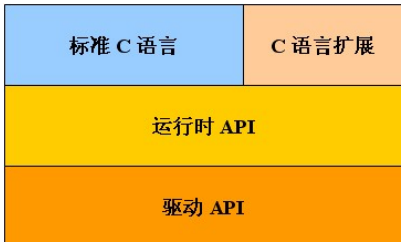
图 1-3. 统一计算设备架构软件堆栈

1 CUDA软件架构（续）

- 三个部分
  - 开发库（CUDA Library）， 目前包括两个标准的数学运算库CUFFT和CUBLAS
  - 运行时环境（CUDA Runtime），提供开发接口和运行时组件，包括基本数据类型的定义和各类计算、内存管理、设备访问和执行调度等函数
  - 驱动（CUDA Driver），提供了GPU的设备抽象级的访问接口，使得同一个CUDA应用可以正确的运行在所有支持CUDA的不同硬件上

2 CUDA编程语言

- CUDA编程语言主要以C语言为主，增加了若干定义和指令。



2 函数限定符

- 函数类型限定符需要指定函数的执行位置（主机或设备）和函数调用者（通过主机或通过设备）
- 在设备上执行的函数受到一些限制，如函数参数的数目固定，无法声明静态变量，不支持递归调用等等
- 用 _global_ 限定符定义的函数是从主机上调用设备函数的唯一方式，其调用是异步的，即立即返回

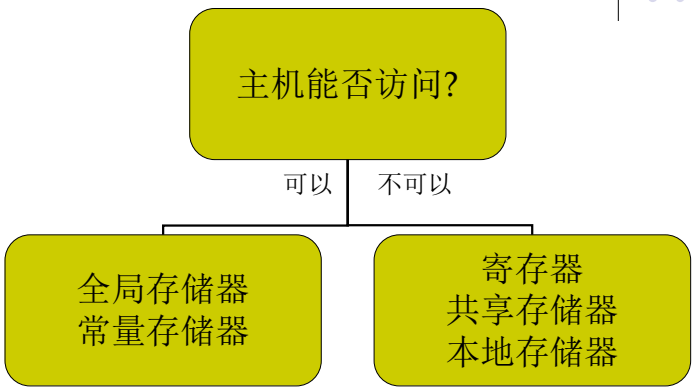
函数限定符	在何处执行	从何处调用	特性
device	设备	设备	函数的地址无法获取
global	设备	主机	返回类型必须为空
host	主机	主机	等同于不使用任何限定符

2 变量限定符

- _shared_ 限定符声明的变量只有在线程同步执行之后，才能保证共享变量对其他线程的正确性。
- 不带限定符的变量通常位于寄存器中。若寄存器不足，则置于本地存储器中

限定符	位于何处	可以访问的线程	主机访问
device	全局存储器	线程网格内的所有线程	通过运行时库访问
constant	固定存储器	线程网格内的所有线程	通过运行时库访问
shared	共享存储器	线程块内的所有线程	不可从主机访问

主机能访问哪里的变量?



44

3 内核函数 （Kernel）

- 内核函数是特殊的一种函数，是从主机调用设备代码唯一的接口，相当于显卡环境中的主函数
- 内核函数的参数被通过共享存储器传递，从而造成可用的共享存储器空间减少（一般减少100字节以内）
- 内核函数使用__global__函数限定符声明，返回值为空

```
__global__ void KernelDemo(float* a, float* b, float* c)
{
    int i = threadIdx.x;
    c[i] = a[i] + b[i];
}
```

4 运行时API

- 设备管理
  - cudaGetDeviceCount()： 获得可用GPU设备的数目
  - cudaGetDeviceProperties()： 得到相关的硬件属性
  - 使用cudaSetDevice()： 选择本次计算使用的设备
  - 默认使用第一个可用的GPU设备，即device 0
- 内存管理
  - cudaMalloc()： 分配线性存储空间
  - cudaFree()： 释放分配的空间
  - cudaMemcpy()：内存拷贝
  - cudaMallocPitch()：分配二维数组空间并自动对齐
  - cudaMemcpyToSymbol()：将主机上的一块数据复制到GPU上的固定存储器

4 内存拷贝cudaMemcpy()

- 由于主机内存和设备内存是完全不同的两个内存空间，因此必须严格指定数据所在的位置。
- 四种不同的传输方式
  - 主机到主机（HostToHost）
  - 主机到设备（HostToDevice）
  - 设备到主机（DeviceToHost）
  - 设备到设备（DeviceToDevice）
- 其中主机到设备和设备到主机的传输需要经过主板上的PCI-E总线接口，一般带宽在1～2GB/s左右。而设备到设备的带宽可达40GB/s以上

4 CUDA程序的生命周期

- CUDA程序的生命周期：
  1. 主机代码执行
  2. 传输数据到GPU
  3. GPU执行
  4. 传输数据回CPU
  5. 继续主机代码执行
  6. 结束
- 如果有多个内核函数，需要重复2～4步

CPU、GPU、DPU...

（from Nvidia@May. 2020）