AI 芯片 – GPU 详解

GPU基础概念

ZOMI

Talk Overview

1. AI 计算体系
   - 深度学习计算模式
   - 计算体系与矩阵运算
2. AI 芯片基础
   - 通用处理器 CPU
   - 从数据看 CPU 计算
   - 通用图形处理器 GPU
   - AI专用处理器 NPU/TPU
   - 计算体系架构的黄金10年

1. **硬件基础**
   - GPU 工作原理
   - GPU AI编程本质
2. **英伟达 GPU 架构**
   - 从 Fermi 到 Hopper 架构
   - Tensor Code 和 NVLink 详解
3. **GPU 图形处理流水线**
   - 图形流水线基础
   - GPU 逻辑模块划分
   - 图形处理算法到硬件

Talk Overview

1. **硬件基础**
   - GPU 工作原理
   - GPU AI编程本质
2. **英伟达 GPU 架构**
   - GPU基础概念
   - 从 Fermi 到 Volta 架构
   - Turing 到 Hopper 架构
   - Tensor Code 和 NVLink 详解
3. **GPU 图形处理**
   - GPU 逻辑模块划分
   - 算法到 GPU 硬件
   - GPU 的软件栈
   - 图形流水线基础
   - 流水线不可编译单元
   - 光线跟踪流水线

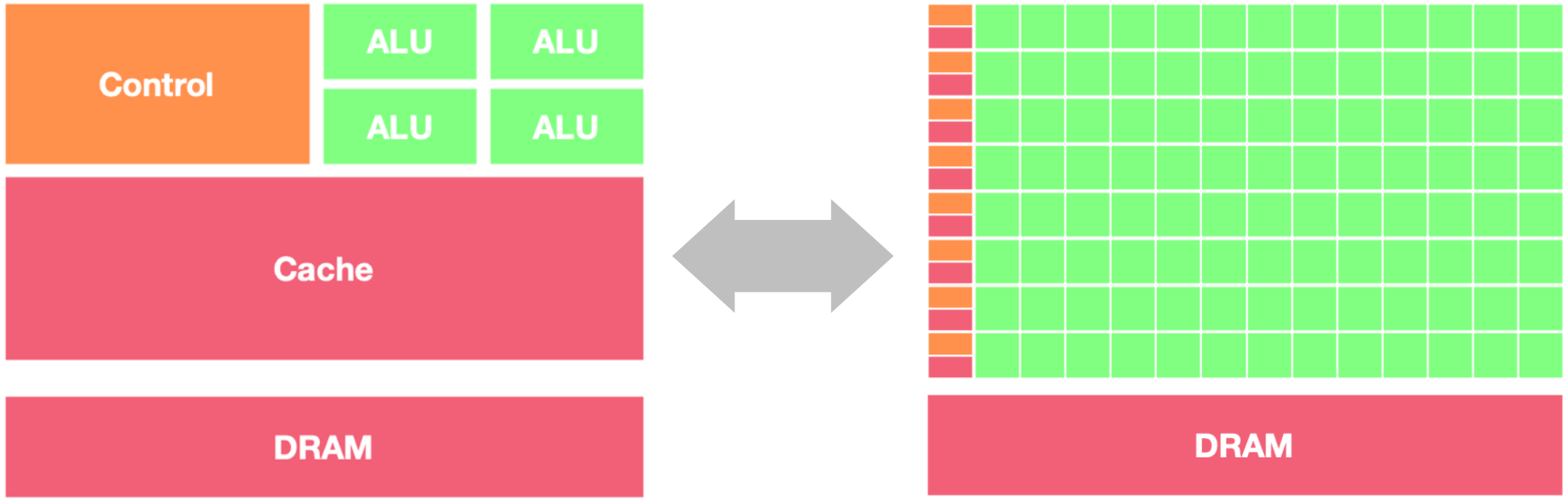
Talk Overview

I. GPU 基础概念

- 硬件基本概念（SM、SP、CUDA Core）
- CUDA 并行计算平台 + CUDA 线程层次结构
- 算力计算 NVIDIA Peak FLOPs

GPU vs CPU

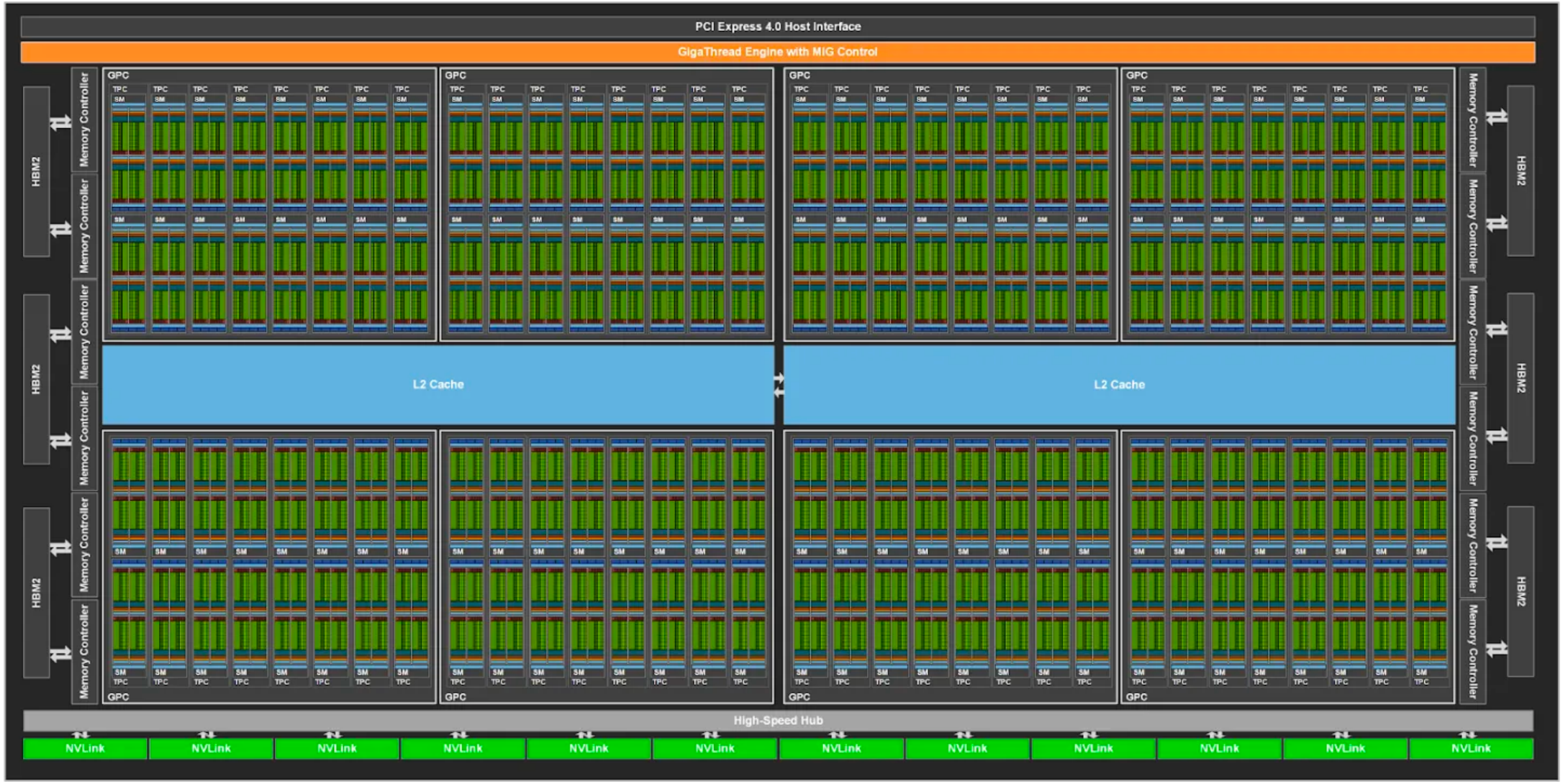
- GPU几乎主要由计算单元ALU组成，仅有少量的控制单元和存储单元。GPU采用了数量众多的计算单元和超长的流水线，但只有非常简单的控制逻辑并省去了Cache。
- CPU不仅被Cache占据了大量空间，而且还有有复杂的控制逻辑和诸多优化电路，相比之下计算能力只是CPU很小的一部。



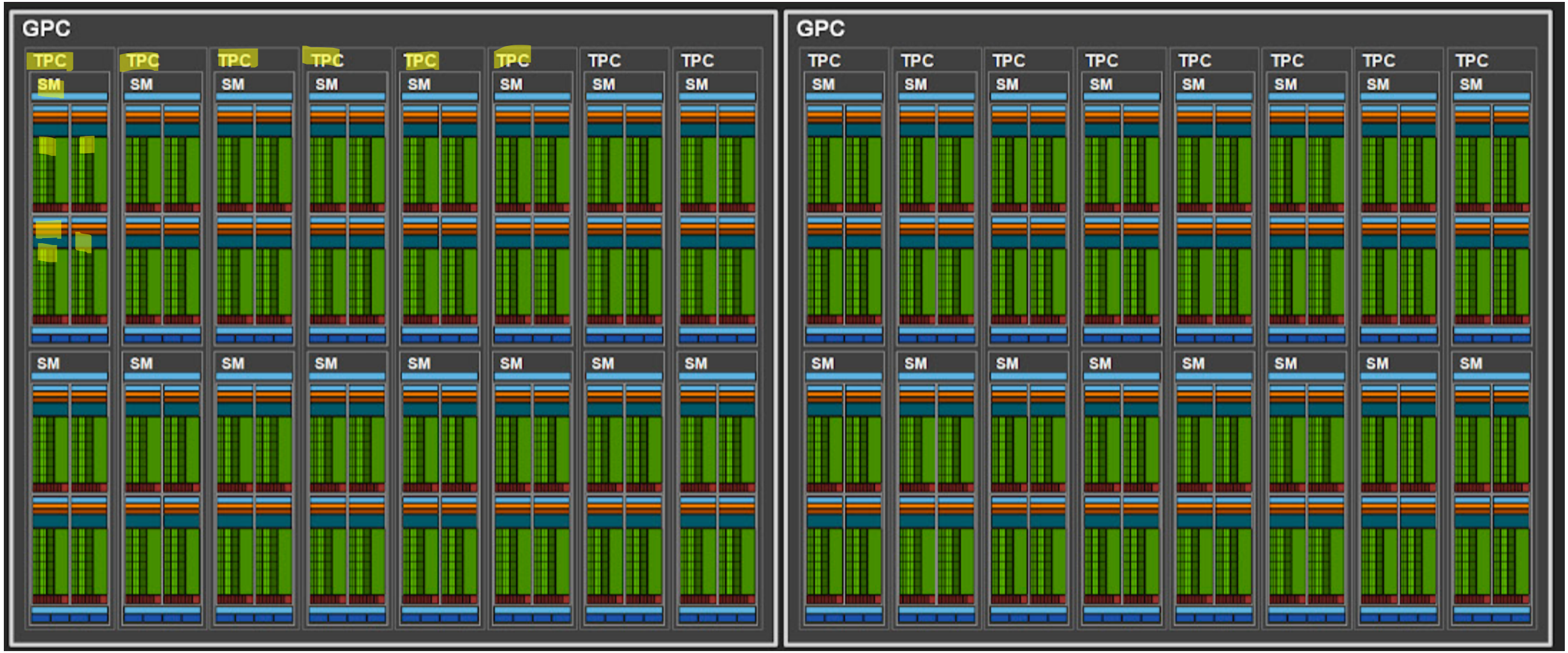
Architecture

硬件基本概念

A100 GPU架构图



A100 GPU架构图



GPU概念之间的关系

- GPC —— 图形处理簇，Graphics Processing Clusters
- TPC —— 纹理处理簇，Texture Processing Clusters
- SM —— 流多处理器，Stream Multiprocessors
- HBM —— 高带宽存储器，High Bandwidth Memory

- 包含关系为：GPC > TPC > SM > CORE

SM (Streaming Multiprocessor)

- **SM**：从 G80 提出的概念，中文称流式多处理器，核心组件包括CUDA核心、共享内存、寄存器等。SM包含许多为线程执行数学运算的 Core，是 NVIDA 的核心。

- **在CUDA中的作用**：可以并发地执行数百个线程。一个 block 上线程是放在同一个 SM，一个 SM 的有限 Cache 制约了每个 block 的线程数量。

SM (Streaming Multiprocessor)

	GP 100
CUDA Core	64
Register File	256KB
Shared Memory	64KB
Active Thread	2048
Active Block	32
Active Grid	8

SM（Streaming Multiprocessor）

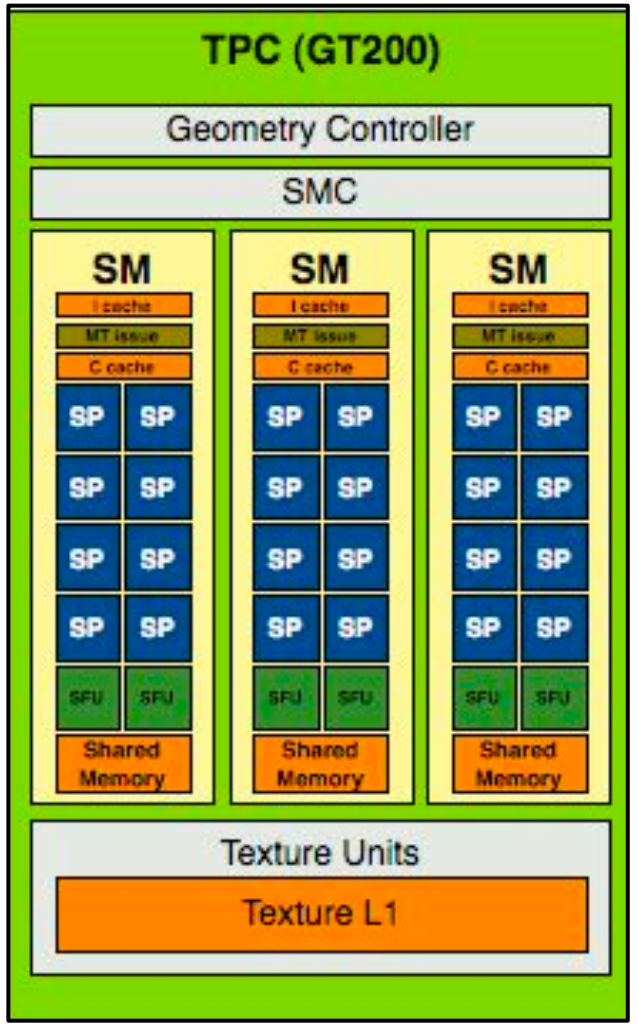
- **SM**：流式多处理器，核心组件包括CUDA核心、共享内存、寄存器等。SM包含许多为线程执行数学运算的Core，是 NVIDA 的核心。。主要包括：

1. CUDA Core：向量运行单元（FP32-FPU、FP64-DPU、INT32-ALU）；
2. Tensor Core：张量运算单元（FP16、BF16、INT8、INT4）；
3. Special Function Units：特殊函数单元 SFU（超越函数和数学函数，e.g. 反平方根、正余弦等）；
4. Warp Scheduler：线程束调度器（XX Thread / clock）；
5. Dispatch Unit：指令分发单元（XX Thread / clock）；
6. Multi level Cache：多级缓存（L0/L1 Instruction Cache、L1 Data Cache & Shared Memory）；
7. Register File：寄存器堆；
8. Load/Store：访问存储单元LD/ST（负责数据处理）；

SP，流处理器 (Stream Processor)

- **SP**：最基本的处理单元，Streaming processor，也称为CUDA core，最后线程具体的指令和任务都是在SP上处理的。GPU进行并行计算，也就是很多个SP同时做处理。

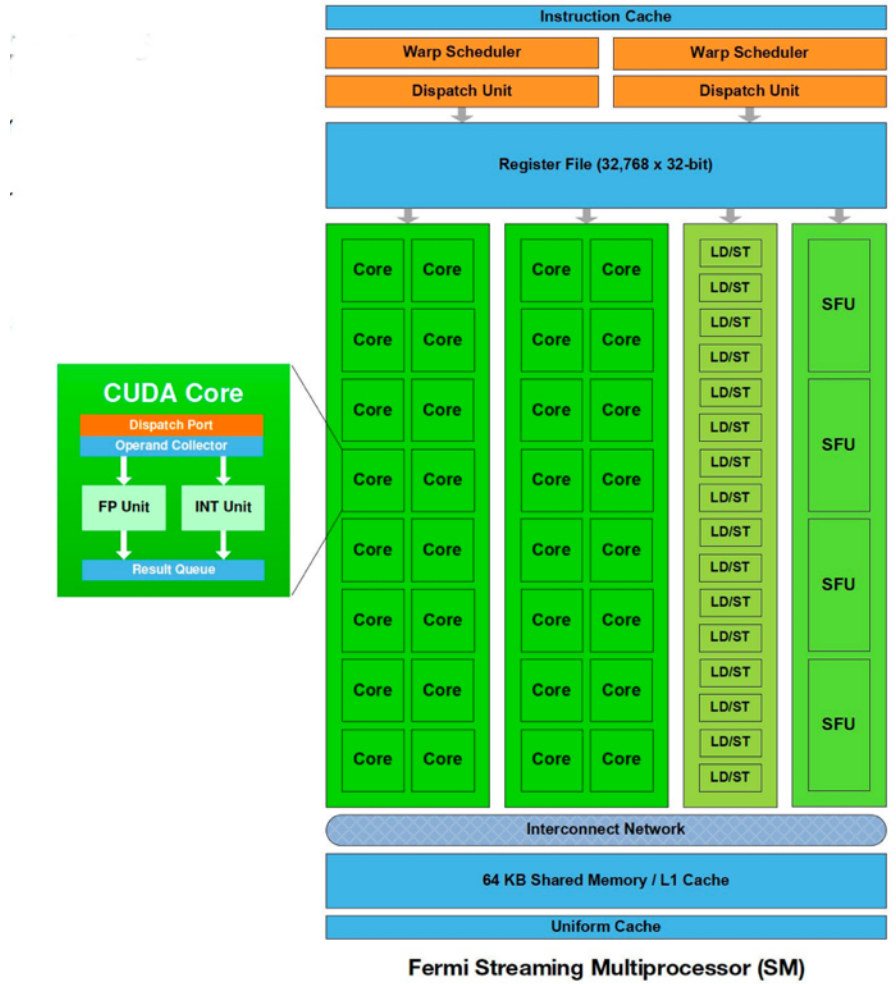
- **SP的消亡**：Fermi 架构后，SP被改称为CUDA Core，通过 CUDA 来控制具体的指令执行。所以对于现在的N卡架构来说，流处理器数量即CUDA Core数量。



CUDA Core，CUDA 核

- **CUDA Core的提出：**CUDA Core 在 Fermi 架构里提出，是最小的运算执行单元。

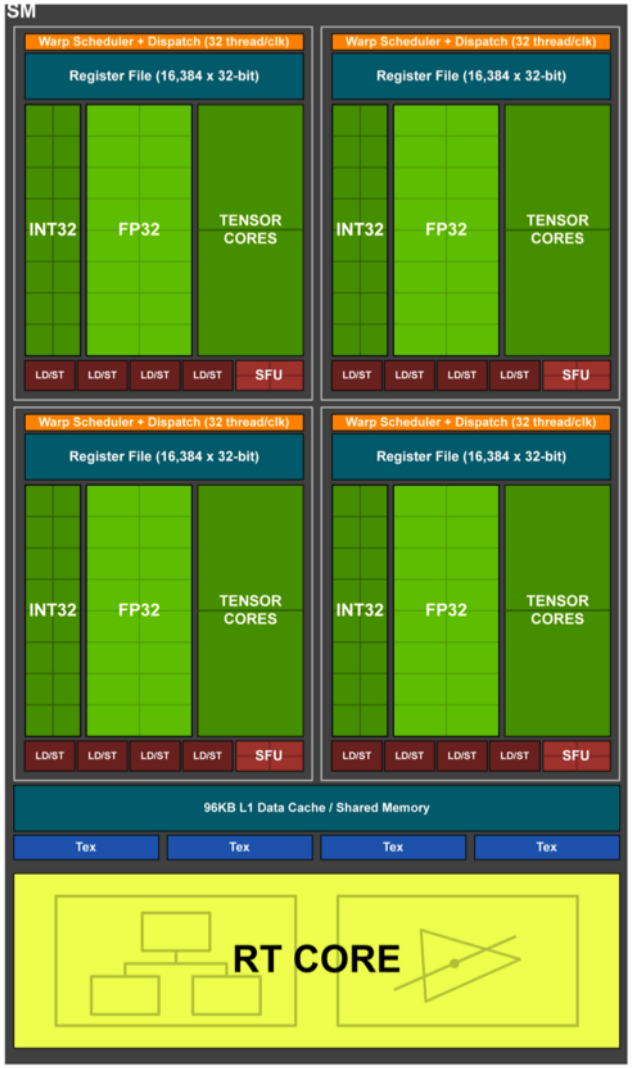
- 在 Fermi 架构中，一个 SM 中包含了有 2 组各 16 个 CUDA Core，每个 CUDA Core 包含了一个整数运算单元 ALU (Integer Arithmetic Logic Unit) 和一个浮点运算单元 FPU (Floating Point Unit) 。



CUDA Core，CUDA 核

- **取消 CUDA Core**：到了 Volta 架构，CUDA Core 又和 Fermi架构时期发生了变化。从这里开始就没有以前的 CUDA Core 了，而是变成了单独的 FP32 FPU 和 INT32 ALU。

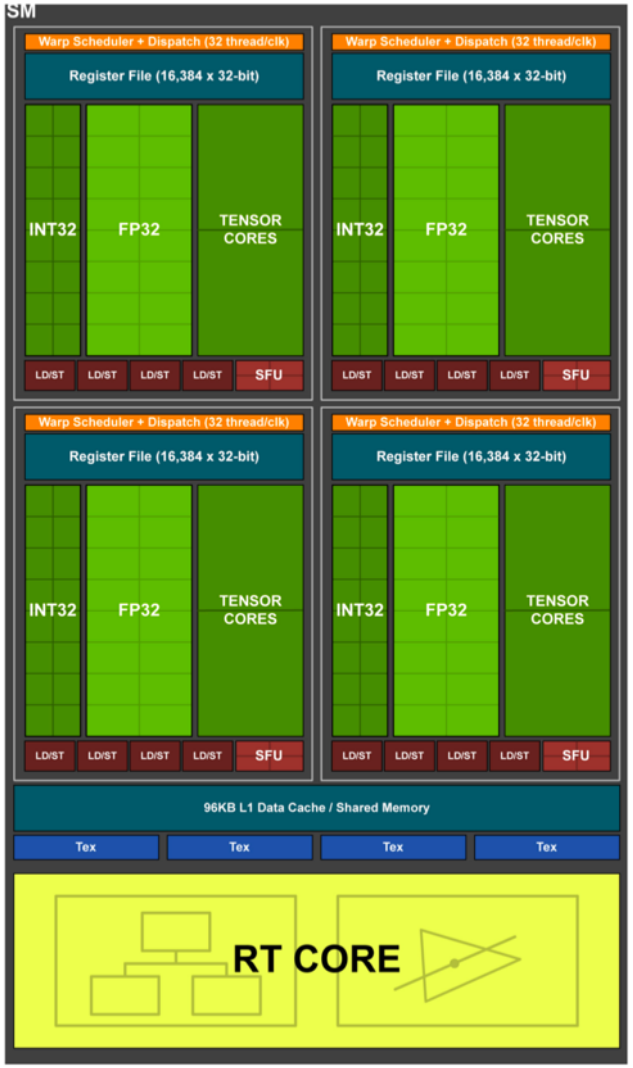
- **并行执行**：因为 FP32:INT32 是 1:1，所以还是很方便把它们合并成原来的 CUDA Core 去称呼。这样做的好处是每个 SM 现在支持 FP32 和 INT32 的并发执行。



Warp,线程束

- **Warp：** 线程束。逻辑上，所有Thread是并行；但是，从硬件的角度来说，并不是所有的 Thread能够在同一时刻执行，这里就需要Warp的引入。

- Warp 是 SM 基本执行单元，一个 Warp 包含32个并行 Thread，这32个 Thread 执行于 SIMT模式。也就是说所有 Thread 以锁步的方式执行同一条指令，但每个 Thread 会使用各自的 Data 执行指令分支。如果在 Warp 中没有32个 Thread 需要工作，那么 Warp 虽然还是作为一个整体运行，但这部分 Thread 是处于非激活状态的。

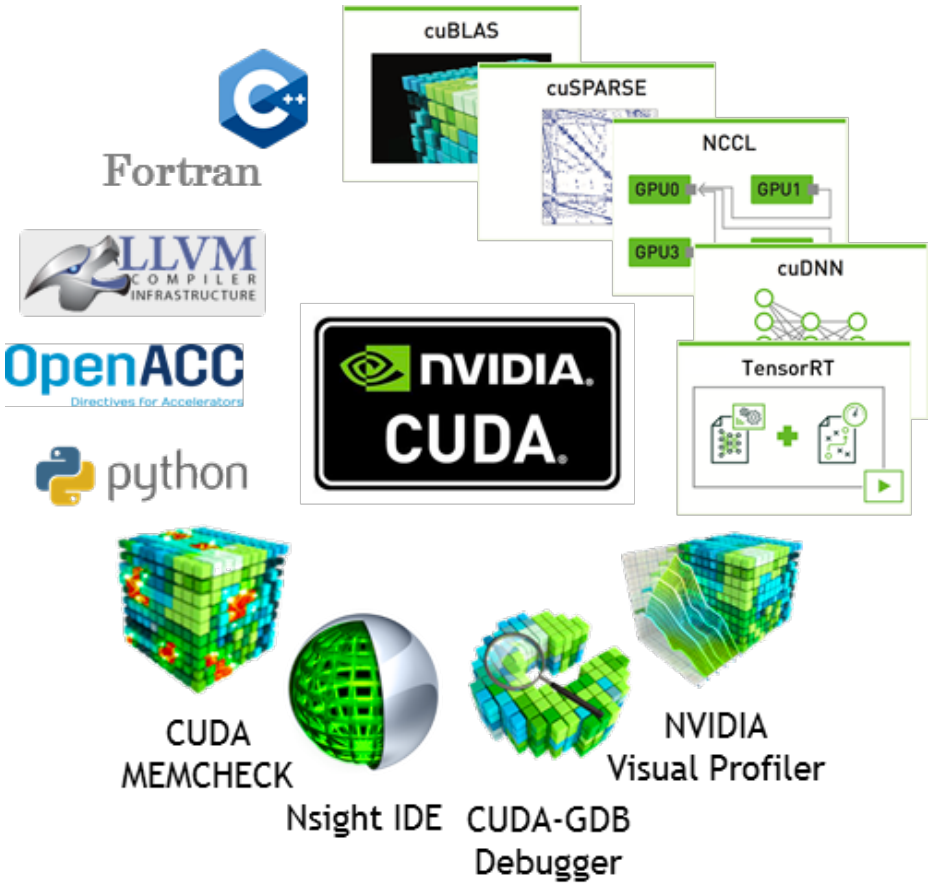


CUDA

基本概念

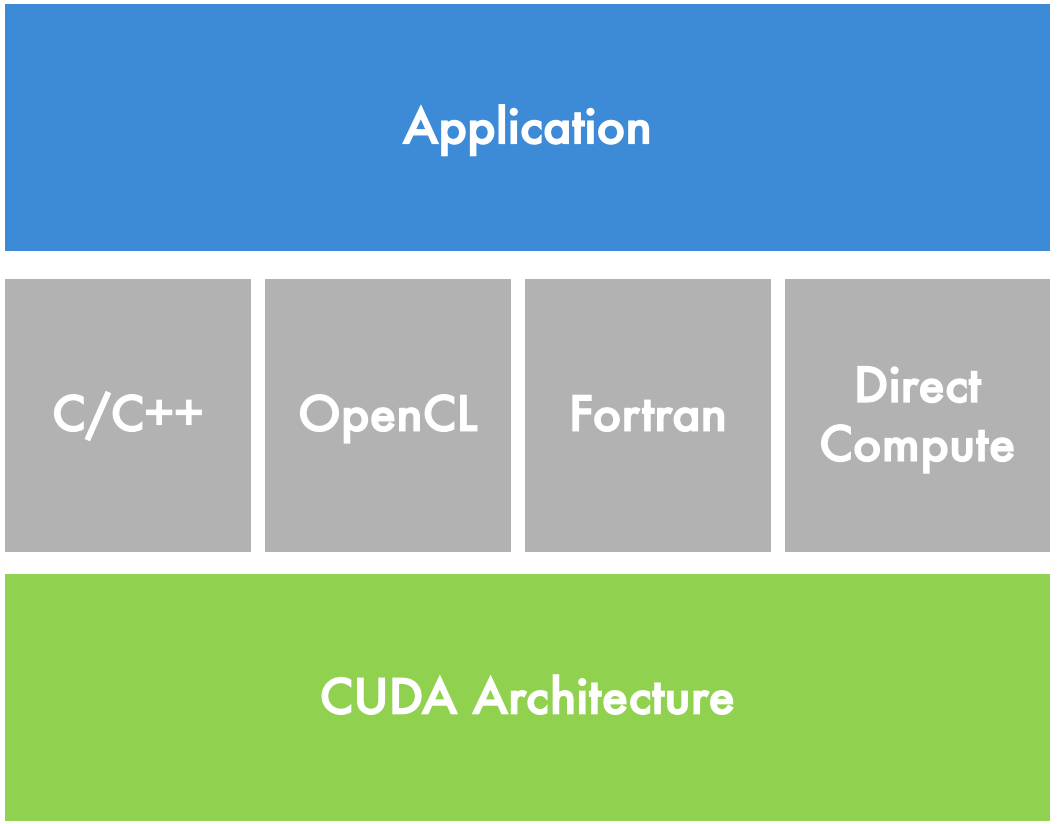
CUDA (Compute Unified Device Architecture)

- 2006 年 11 月， 推出了 CUDA，通用并行计算平台和编程模型，用于图形处理单元（GPU）上的通用计算。



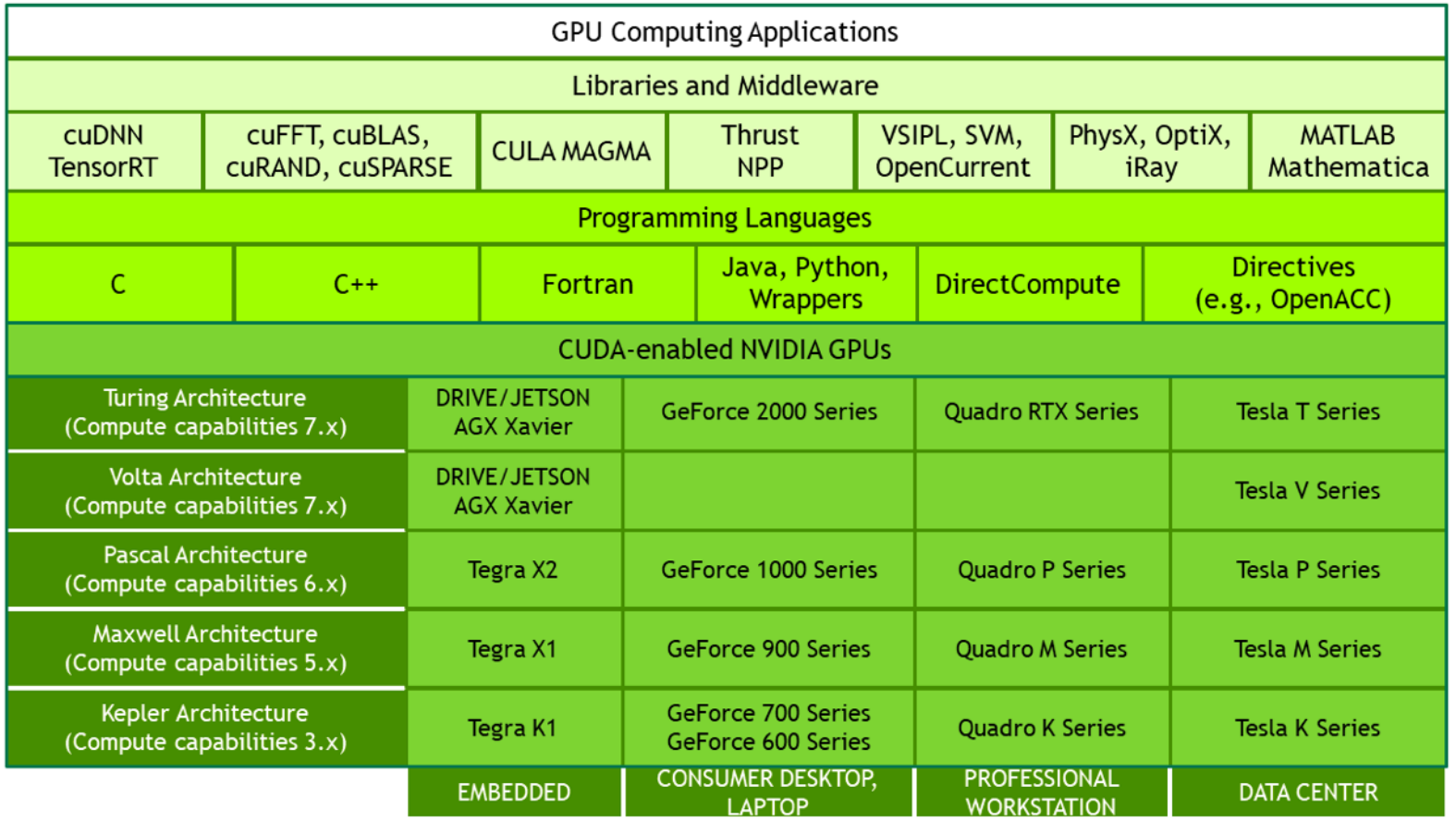
CUDA (Compute Unified Device Architecture)

- 并行计算架构（Parallel Computing Architecture）和编程模型（Programming Model），编程体系。
- 基于 LLVM 构建了 CUDA 编译器，方便开发者使用 C 进行开发。
- 提供了对其它编程语言的支持，如C/C++，Python，Fortran等语言。支持 OpenCL 和 DirectCompute 等应用程序接口。



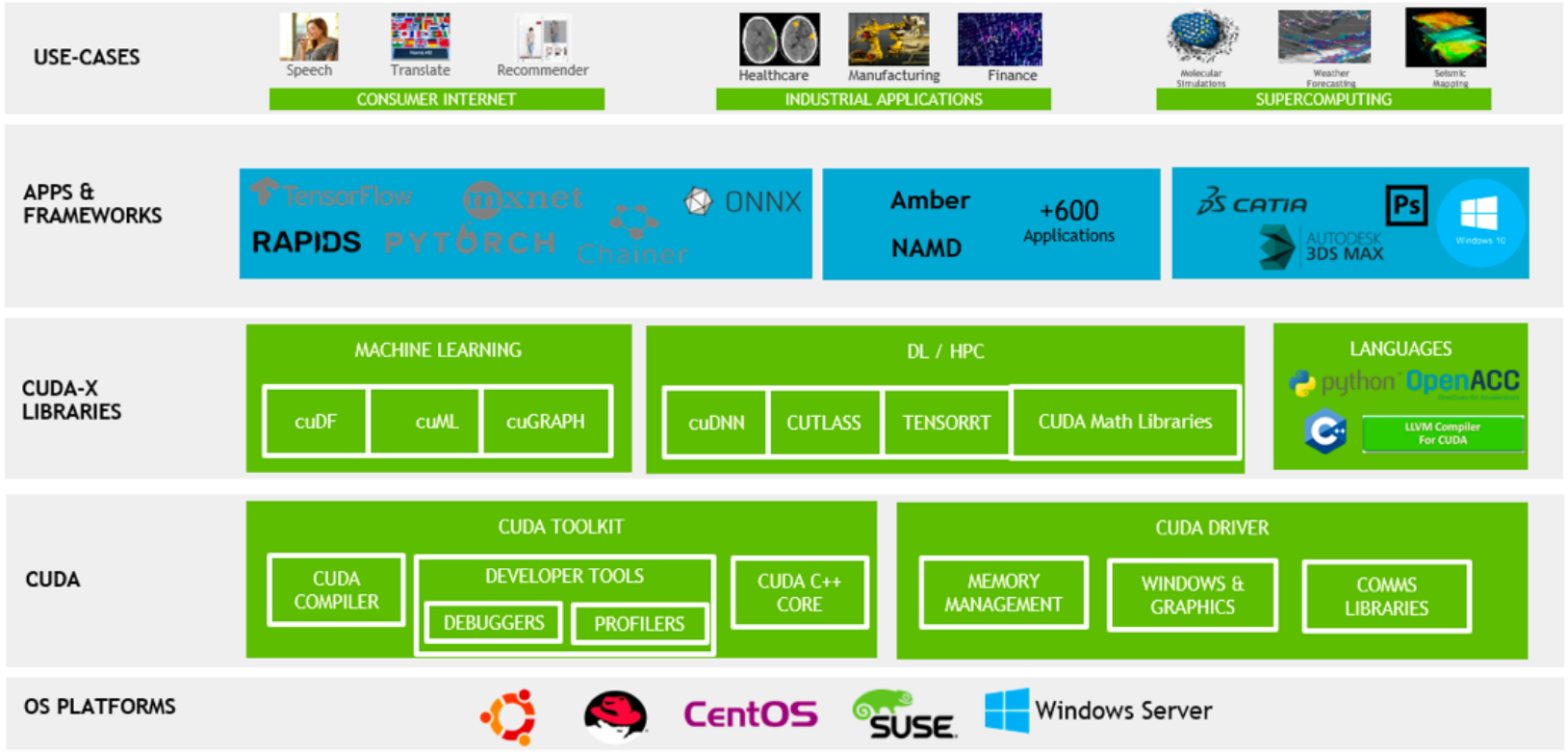
CUDA

- CUDA 并行编程模型作为一款通用接口，为熟悉 C 等标准编程语言的程序员保持较低的学习曲线。



CUDA (Compute Unified Device Architecture)

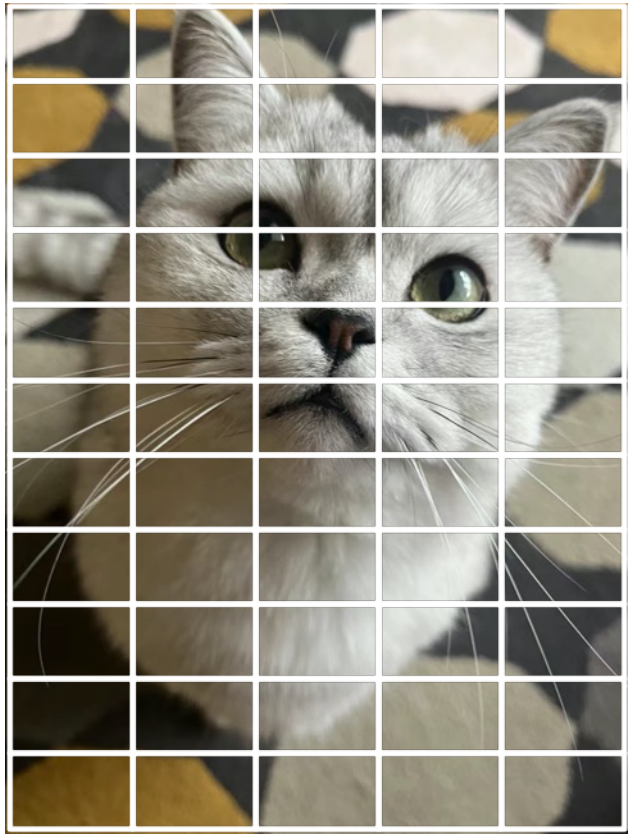
- CUDA在软件方面组成：一个CUDA库、一个应用程序编程接口（ API ）及其运行库(Runtime)、两个较高级别的通用数学库，即 CUFFT 和 CUBLAS。



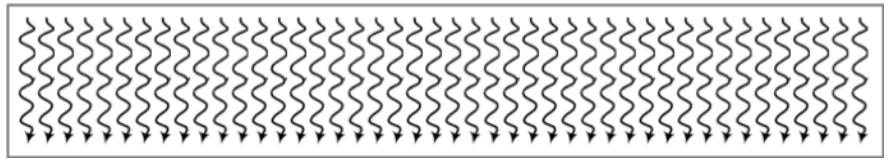
CUDA

线程层次结构

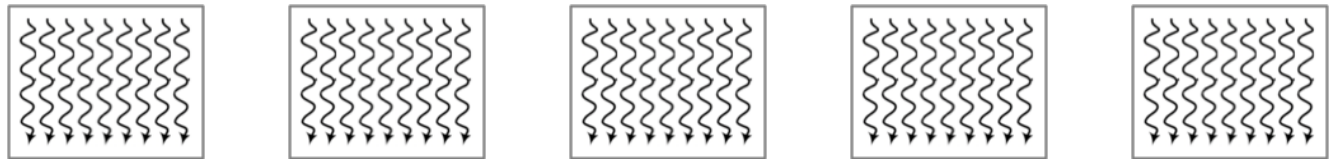
线程层次结构



网格 Grid 表示所有要执行的任务



网格 Grid 中包含了很多相同线程 Threads 数量的块 Blocks

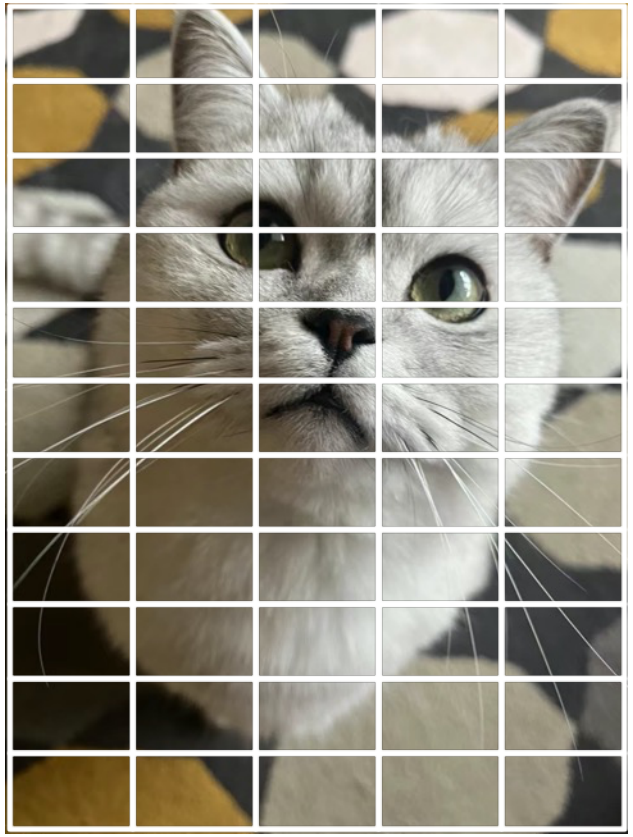


块 Block 中的线程数独立执行

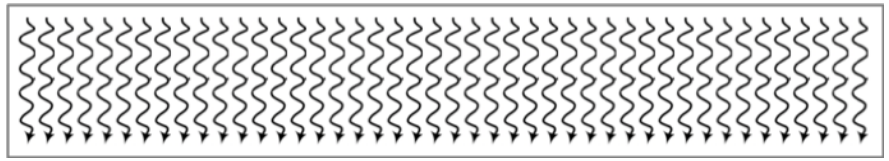
可以通过本地数据共享

同步交换数据

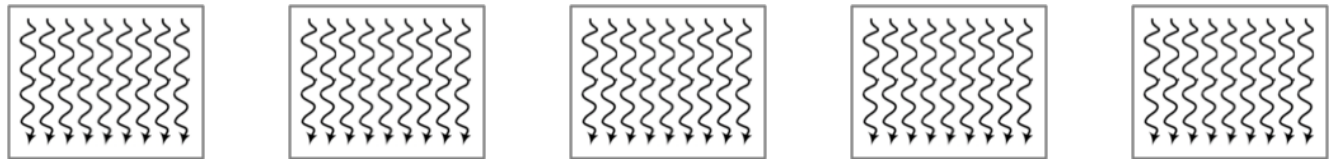
线程层次结构



网格 Grid 表示所有要执行的任务



网格 Grid 中包含了很多相同线程 Threads 数量的块 Blocks

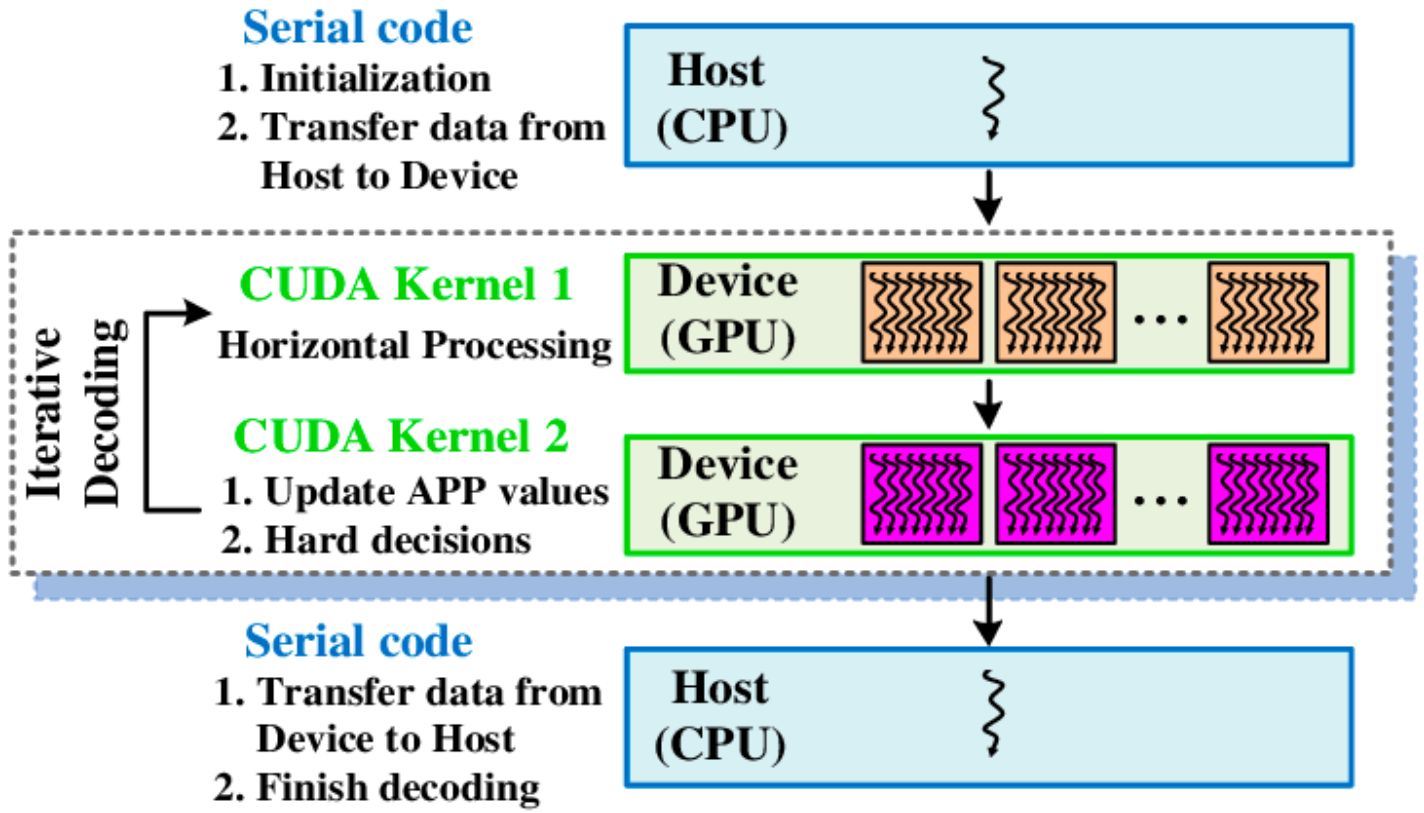


- 将问题划分为独立线程块，并行解决的子问题
- 子问题划分为可以由块内线程并行协作解决

线程层次结构：核 kernel

- **主设概念**：CUDA引入主机端（host）和设备（device）概念。CUDA 程序中既包含host程序，又包含device程序。

- **互相通信**：host与device之间可以进行通信，这样它们之间可以进行数据拷贝。



线程层次结构：核 kernel

- CUDA 执行流程中最重要的一个过程是调用CUDA的核函数来执行并行计算，kernel是CUDA中一个重要的概念。
- 在 CUDA 程序构架中，Host 代码部分在CPU上执行，是普通C代码；当遇到数据并行处理的部分，CUDA 就会将程序编译成GPU能执行的程序，并传送到GPU，这个程序在CUDA里称做核（ kernel ）。
- Device 代码部分在 GPU 上执行，此代码部分在 kernel 上编写（ .cu文件 ）。 kernel 用 __global__ 符号声明，在调用时需要用 <<<grid, block>>> 来指定kernel要执行及结构。

线程层次结构：核 kernel

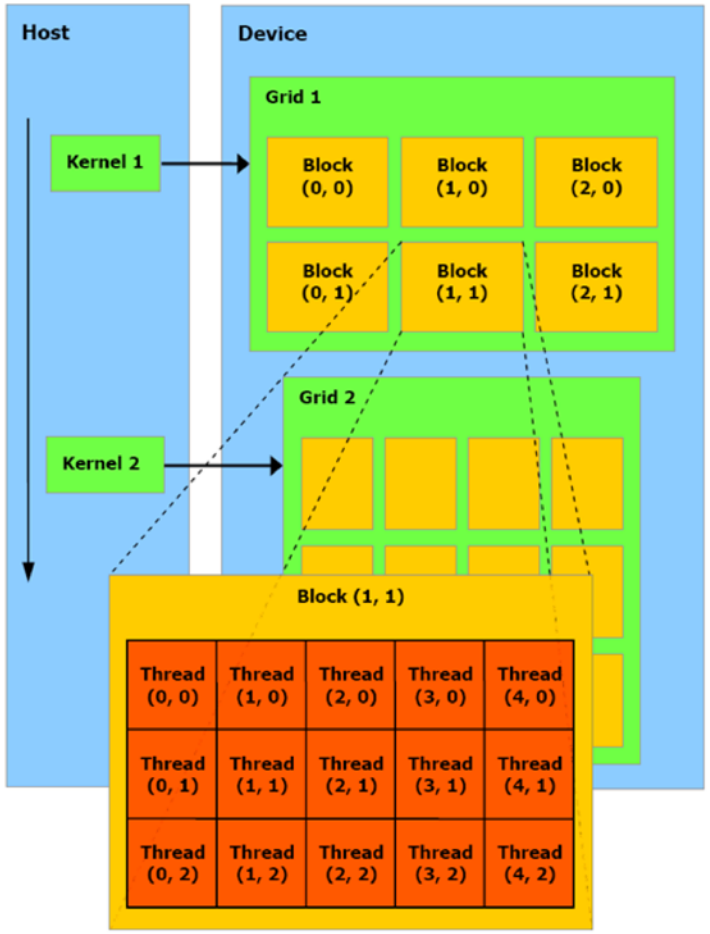
```cpp
#include <iostream>
#include <math.h>
#include <sys/time.h>

// function to add the elements of two arrays
void add(int n, float *x, float *y)
{
  for (int i = 0; i < n; i++)
      y[i] = x[i] + y[i];
}

int main(void)
{
  int N = 1<<25; // 30M elements

  float *x = new float[N];
  float *y = new float[N];

  // initialize x and y arrays on the host
  for (int i = 0; i < N; i++) {
    x[i] = 1.0f;
    y[i] = 2.0f;
  }

  struct timeval t1,t2;
  double timeuse;
  gettimeofday(&t1,NULL);

  // Run kernel on 30M elements on the CPU
  add(N, x, y);

  // Free memory
  delete [] x;
  delete [] y;

  return 0;
}
```

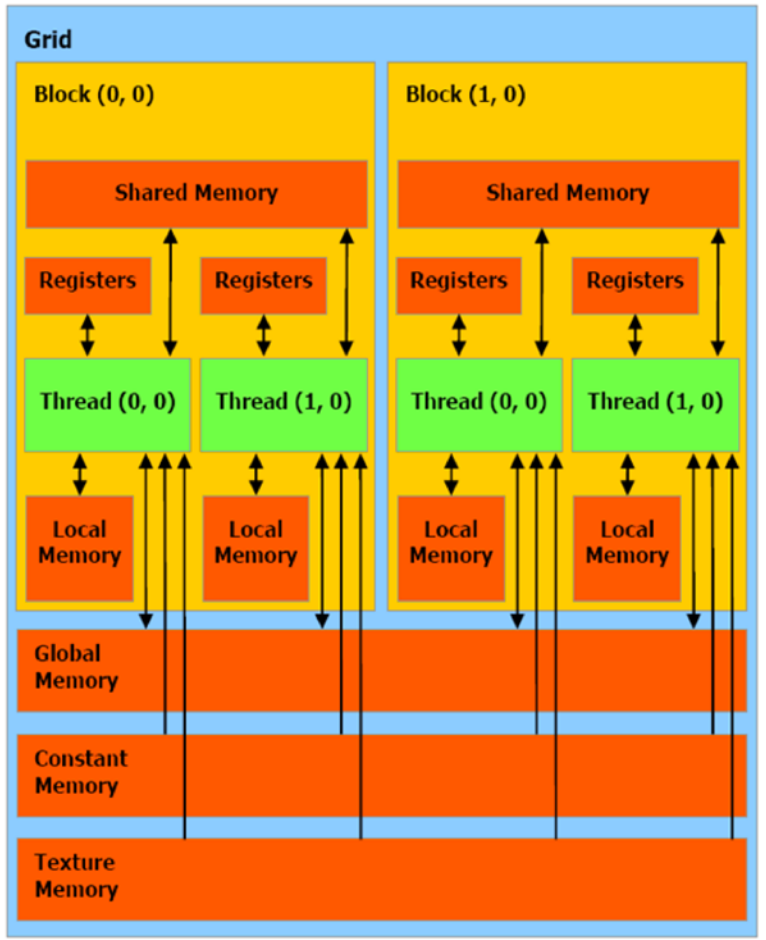
```cpp
#include <iostream>
#include <math.h>
// Kernel function to add the elements of two arrays
// __global__ 变量声明符，作用是将add函数变成可以在GPU上运行的函数
// __global__ 函数被称为kernel
__global__
void add(int n, float *x, float *y)
{
  for (int i = 0; i < n; i++)
    y[i] = x[i] + y[i];
}

int main(void)
{
  int N = 1<<25;
  float *x, *y;

  // Allocate Unified Memory – accessible from CPU or GPU
  // 内存分配，在GPU或者CPU上统一分配内存
  cudaMallocManaged(&x, N*sizeof(float));
  cudaMallocManaged(&y, N*sizeof(float));

  // initialize x and y arrays on the host
  for (int i = 0; i < N; i++) {
    x[i] = 1.0f;
    y[i] = 2.0f;
  }

  // Run kernel on 1M elements on the GPU
  // execution configuration，执行配置
  add<<<1, 1>>>(N, x, y);

  // Wait for GPU to finish before accessing on host
  // CPU需要等待cuda上的代码运行完毕，才能对数据进行读取
  cudaDeviceSynchronize();

  // Free memory
  cudaFree(x);
  cudaFree(y);

  return 0;
}
```

线程层次结构 I：网格 grid

- kernel 在 device 上执行时，实际上是启动很多线程，一个 kernel 所启动的所有线程称为一个网格（grid）

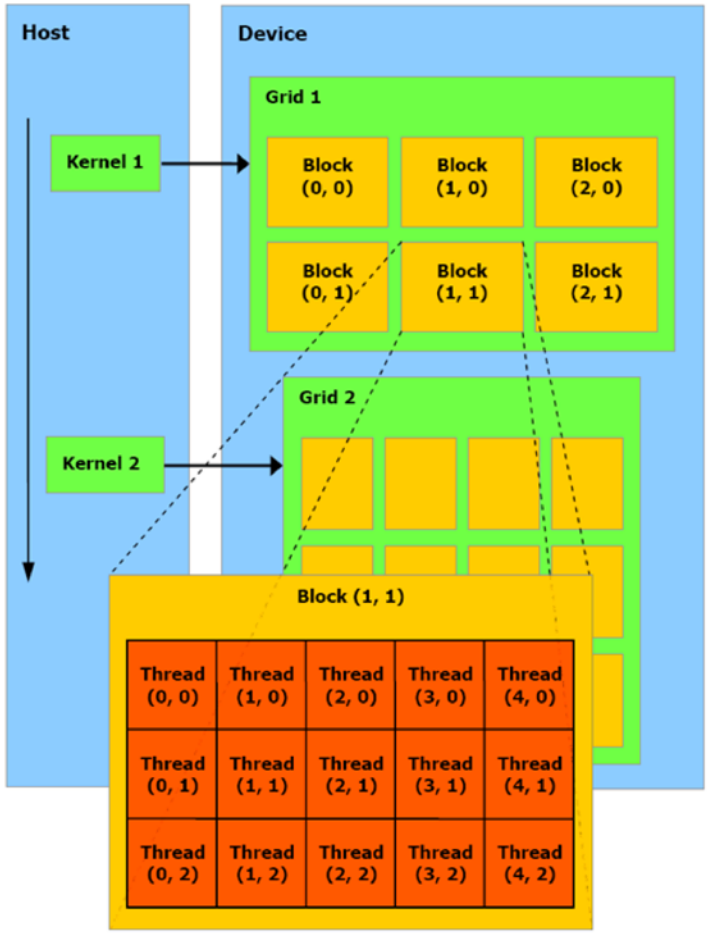
- 同一个网格上的线程共享相同的全局内存空间。grid是线程结构的第一层次。

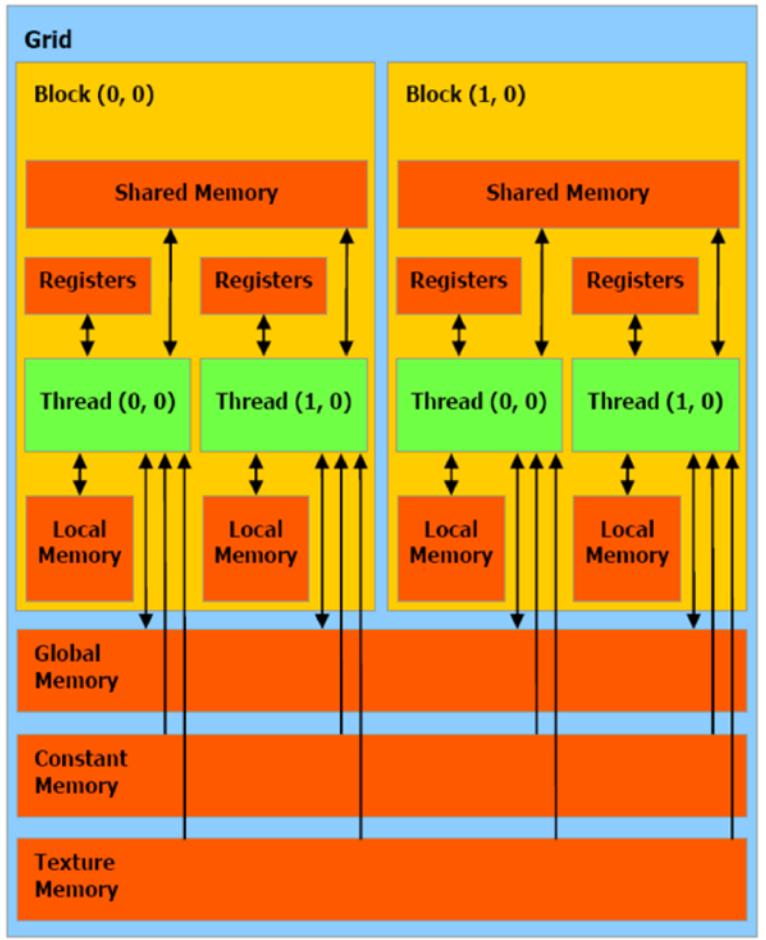




线程层次结构 II：线程块 Block

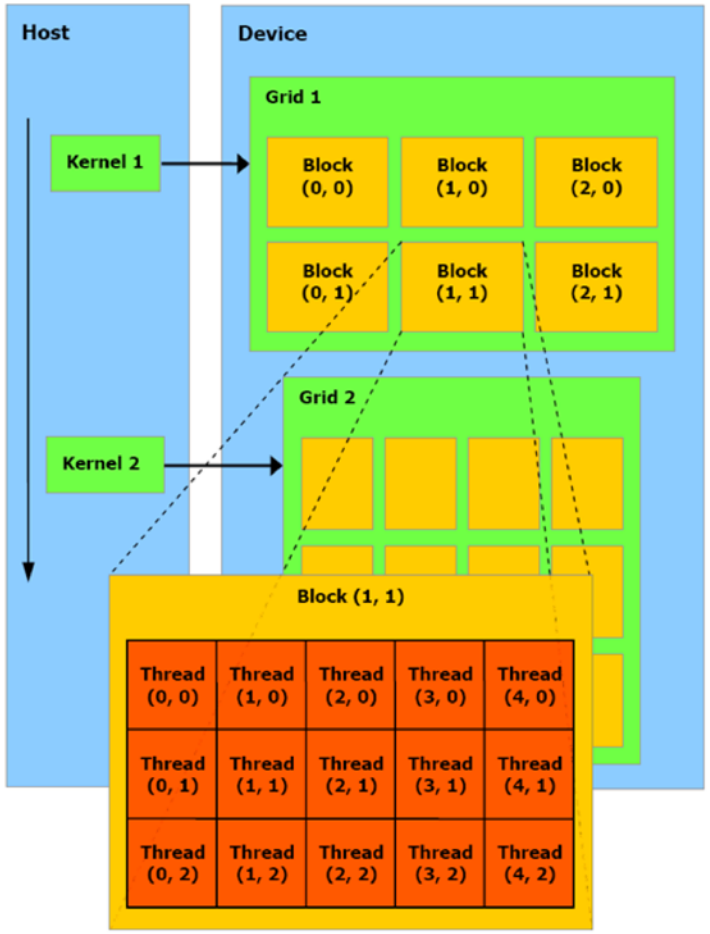
- Grid 分为多个线程块（block），一个 block 里面包含很多线程。
- Block 间并行执行，并且无法通信，也没有执行顺序。
- 每个 block 包含共享内存（Shared Memory），可以里面的 Thread 共享。

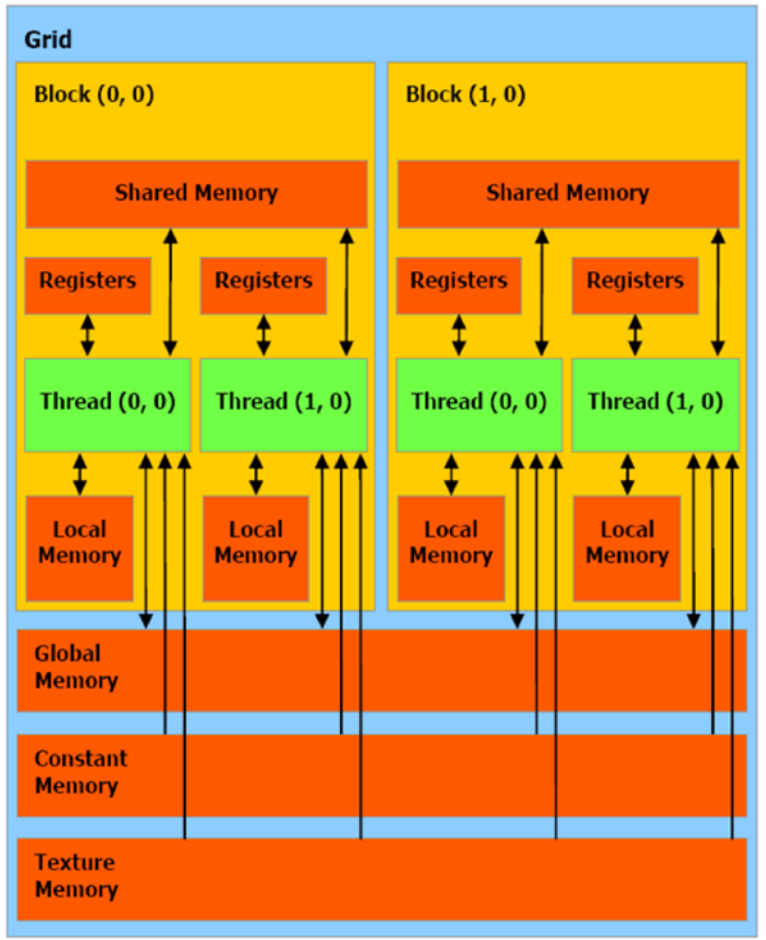




线程层次结构 III：线程 thread

- CUDA 并行程序，实际上会被多个 threads 来执行；
- 多个 threads 会被群组成一个线程 block；
- 同一个 block 中 threads 可以同步，也可以通过 shared memory 通信。



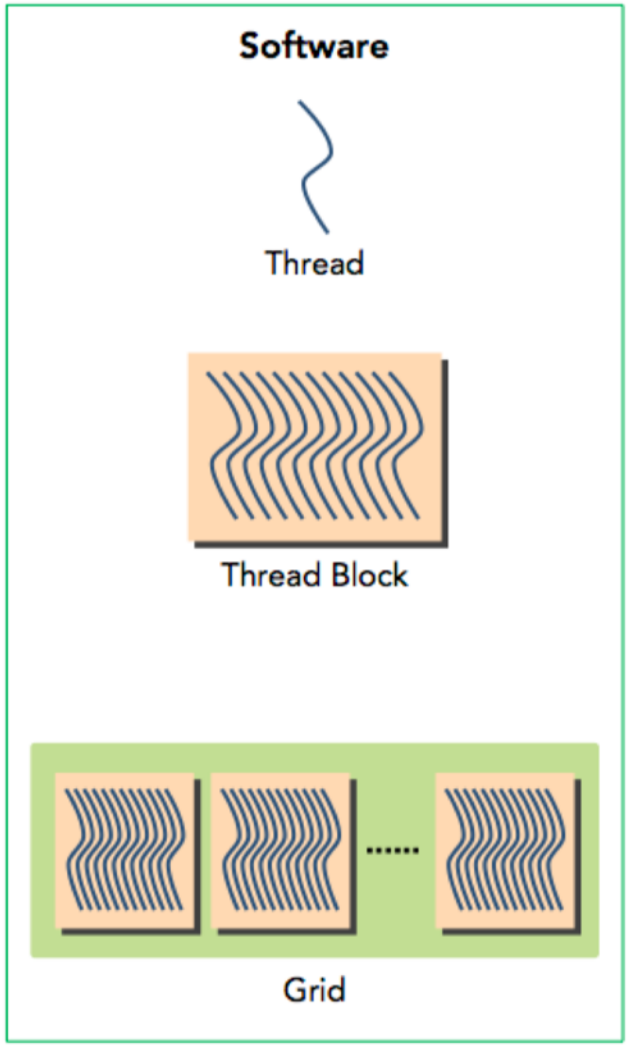


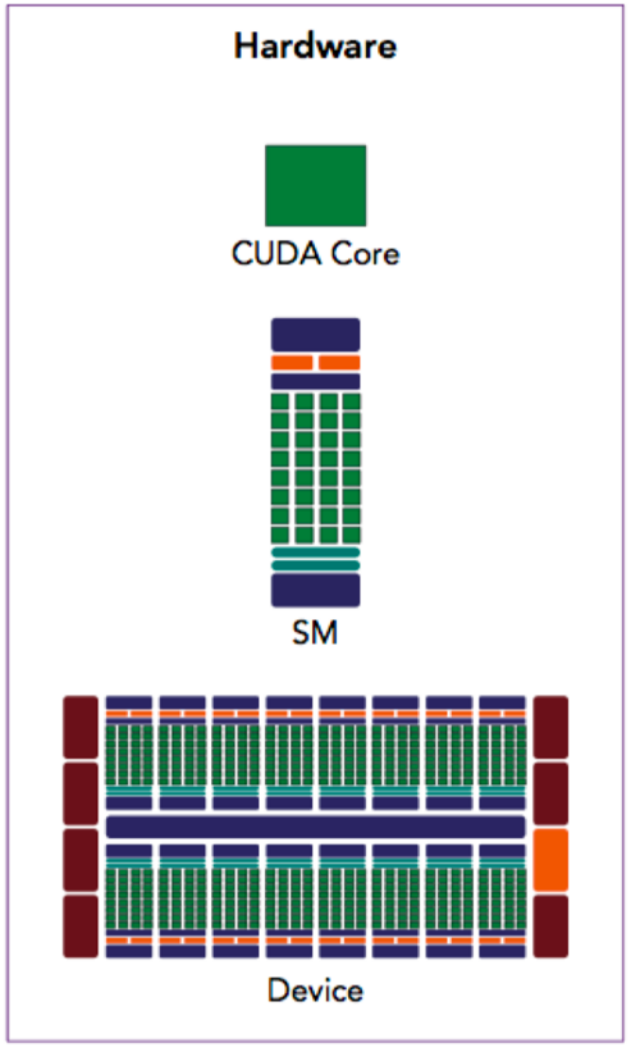
回到硬件：Wrap

- **Wrap**：GPU 执行程序时的调度单位，SM的基本执行单元。
- Warp 包含 32 个线程的集合，这个线程集合被“编织在一起”并且“步调一致”的形式执行。
- 同一个 Warp 中的每个线程都将以不同数据资源执行相同的指令，这就是所谓 SIMT 架构(Single-Instruction, Multiple-Thread，单指令多线程)。

CUDA 跟 NVIDIA 硬件架构的关系

- Block 线程块只在一个 SM 上通过 Wrap 进行调度。
- 一旦在 SM 上调起了 Block 线程块，就会一直保留到执行完 Kernel。
- SM 可以同时保存多个 Block 线程块，快间并行的执行。





NVIDIA

算力计算

峰值算力

- GPU 算力跟计算核心个数、核心频率、核心单时钟周期能力三个因素有关。GPU 峰值计算能力，公式如下：

$$Peak\ FLOPS = F_{clk} * N_{SM} * F_{req}$$

1. F_{clk} 为 GPU 时钟周期内指令执行数 (FLOPS/Cycle)
2. N_{SM} 为 GPU SM 数量 (Cores)
3. F_{req} 为运行频率 (GHz)

峰值算力

- A100 中 FP32 Tensor Core 指令吞吐 64 FLOPS/Cycle ，核心运行频率为 1.41GHz ，SM 数量为108 ，那么有：$Peak\ FLOPS = 1.41 * 108 * 64 * 2 = 19{,}491\ GFLOPS$ ~ 1.95 TFLOPS

Table 1. NVIDIA A100 Tensor Core GPU Performance Specs

Peak FP64[1]	9.7 TFLOPS
Peak FP64 Tensor Core[1]	19.5 TFLOPS
Peak FP32[1]	19.5 TFLOPS
Peak FP16[1]	78 TFLOPS
Peak BF16[1]	39 TFLOPS
Peak TF32 Tensor Core[1]	156 TFLOPS | 312 TFLOPS[2]
Peak FP16 Tensor Core[1]	312 TFLOPS | 624 TFLOPS[2]
Peak BF16 Tensor Core[1]	312 TFLOPS | 624 TFLOPS[2]
Peak INT8 Tensor Core[1]	624 TOPS | 1,248 TOPS[2]
Peak INT4 Tensor Core[1]	1,248 TOPS | 2,496 TOPS[2]

1 - Peak rates are based on GPU Boost Clock.
2 - Effective TFLOPS / TOPS using the new Sparsity feature

总结

GPU概念之间的关系

- GPC —— 图形处理簇，Graphics Processing Clusters
- TPC —— 纹理处理簇，Texture Processing Clusters
- SM —— 流多处理器，Stream Multiprocessors
- HBM —— 高带宽存储器，High Bandwidth Memory

- 包含关系为：GPC > TPC > SM > CORE

- SM 中包含 Poly Morph Engine、L1 Cache、Shared Memory、CUDA Core等
- CUDA Core 中包含 ALU、FPU、Execution Context、Thread Detach、Command Decode等

Reference 引用&参考

1. https://zhuanlan.zhihu.com/p/620257581 大佬们，A100显卡上的tensorcore有自己的私有寄存器
2. https://old.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.21-Monday-Pub/HC29.21.10-GPU-Gaming-Pub/HC29.21.132-Volta-Choquette-NVIDIA-Final3.pdf
3. https://ieeexplore.ieee.org/author/37086370271
4. https://www.computer.org/csdl/proceedings-article/hcs/2019/08875651/1ehCtCN4SUE
5. https://resources.nvidia.com/en-us-genomics-ep/ampere-architecture-white-paper
6. https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/
7. https://zhuanlan.zhihu.com/p/231302709
8. https://www.zhihu.com/zvideo/1367794004235542528
9. https://www.bilibili.com/read/cv15145865?from=search
10. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html?highlight=matrix%20multiply
11. https://nyu-cds.github.io/python-gpu/02-cuda/
12. https://zhuanlan.zhihu.com/p/258196004?utm_id=0
13. https://blog.csdn.net/qq_42059060/article/details/121274184