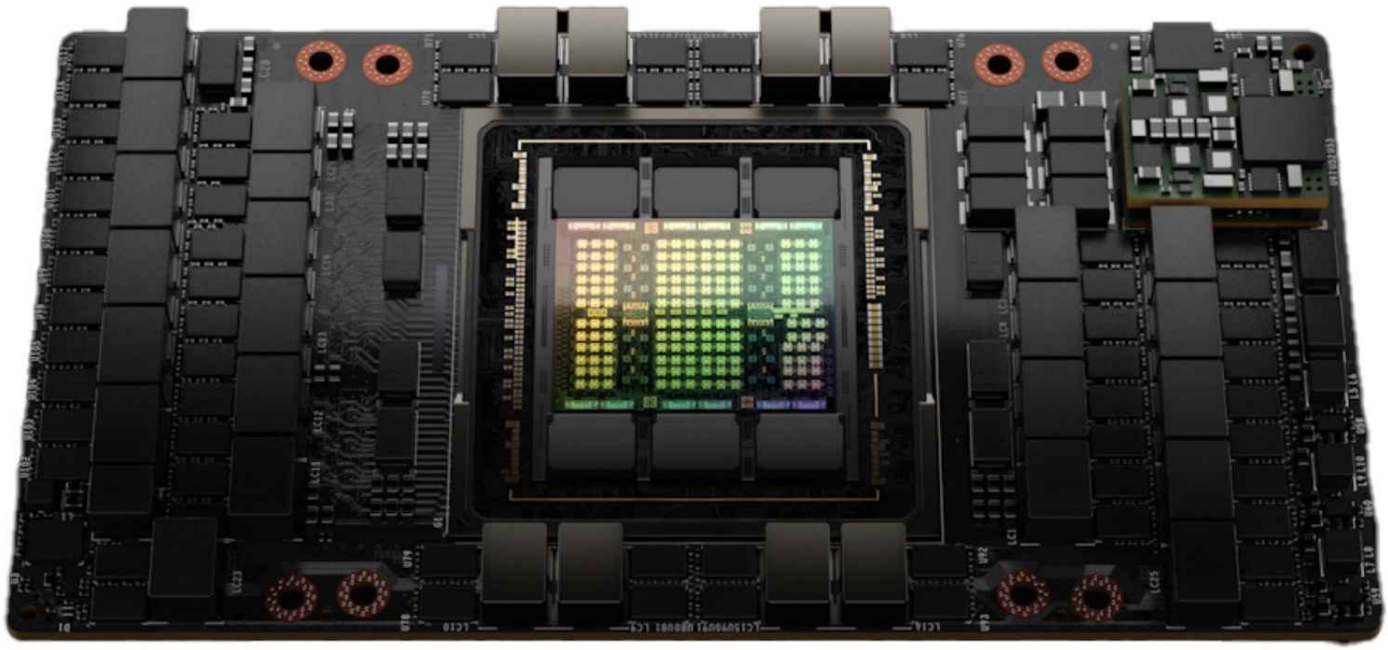
GPU Arch:自顶向下分析

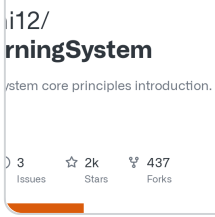


本文很多内容都取自ZOMI酱，大家对机器学习系统框架，GPU底层架构想要有一个更加深入的了解可以关注ZOMI。

https://space.bilibili.com/517221395

ZOMI酱的个人空间_哔哩哔哩_bilibili

ZOMI酱，AI系统/移动视觉/强化学习，给UP投币赚的全数捐给《中国儿童基金会》，，bilibili是国内知名的视频弹…



https://github.com/chenzomi12/De…

GitHub - chenzomi12/DeepLearningSystem: Deep Learning…

更多PEFT&MLSys相关精彩内容 Modest Understandings on LLM

https://www.bilibili.com/video/BV1Az4y1B7Da/

GPU Arch：自顶向下分析【浅谈底层·1】_哔哩哔哩_bilibili

更多信息:http://yxinyu.com/飞书文档:https://readpaper.feishu.cn/docx/UwT2dQsiko6u0RxoiXRcBtwfnAf, 视频播放…

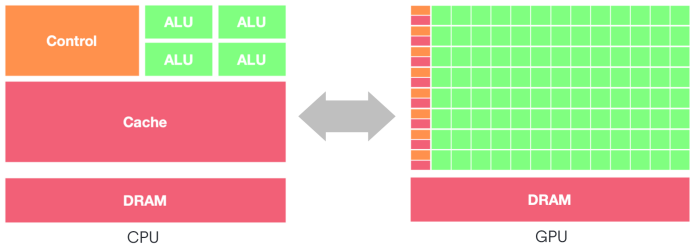
Intro

随着人工智能特别是以GPT为代表的生成式AI的迅猛发展，GPU已经成为了一种不可或缺的工具，甚至企业都以拥有多少高端GPU作为抓住风口能力的衡量标准。相比之下，CPU虽然在传统计算领域占据主导地位，但在处理AI任务时却不及GPU出色。本文将探讨为什么AI计算通常选择GPU而不是

CPU，并分析GPU在AI计算中的优势，同时，从底层原理探讨从Volta到最新的Hopper四代NVIDIA GPU架构的演进，展示其不断提升的性能和功能。

Why not CPU?

GPU主要由计算单元ALU组成。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很小的一部分。



通过视频，我们可以感受一下GPU的重要特性

Parallelism

https://www.youtube.com/watch?v=-P28LKWTzrI

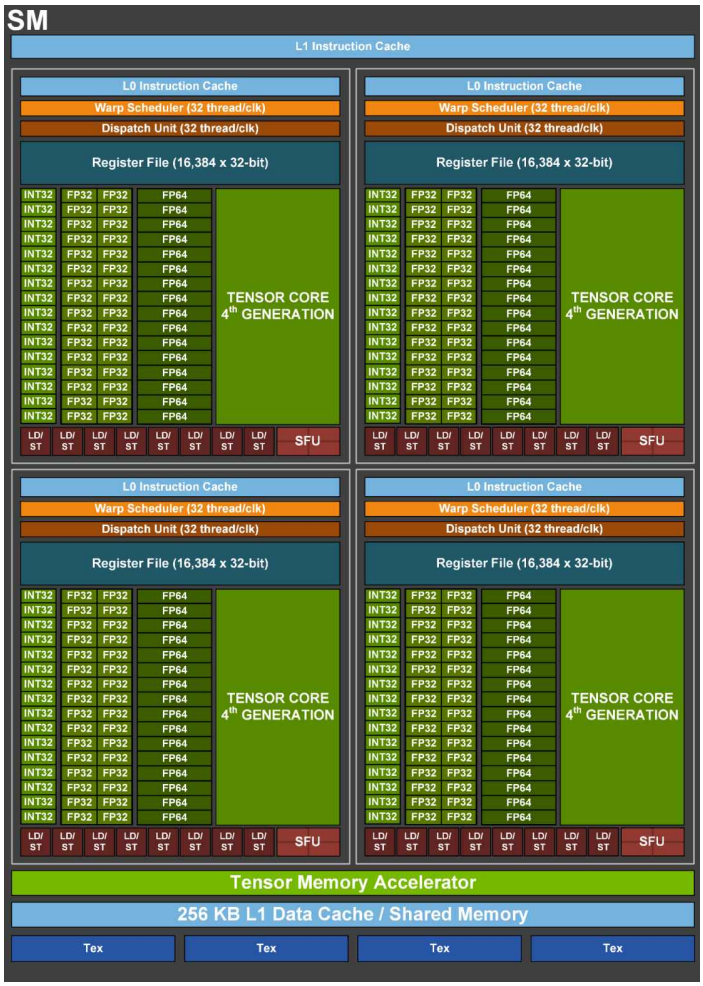
Insight into GPU

接下来，我们以Hopper架构为例自顶向下介绍一下GPU，让我们对GPU有一个更加清晰的认知。

Hopper-H100

1. **GPC(Graphics Processing Clusters)**：GPC负责处理图形渲染和计算任务。每个GPC包含多个TPC，以及与其相关的专用硬件单元和缓存。
2. **TPC(Texture Processing Clusters)**：TPC负责执行纹理采样和滤波操作，以从纹理数据中获取采样值，并应用于图形渲染中的相应像素。在CUDA计算中，每个TPC有两个SM处理计算任务。
3. **HBM（High-Bandwidth Memory）**：HBM是高带宽内存，也就是我们常说的显存。它通过将内存芯片直接堆叠在逻辑芯片上，提供了极高的带宽和更低的能耗，从而实现了高密度和高带宽的数据传输。
4. **L2 Cache**：L2 Cache是GPU中更大容量的高速缓存层，它位于多个流多处理器（SM）之间共享。L2 Cache还可以用于协调SM之间的数据共享和通信。

1. **SM(Streaming Multiprocessor)**：SM是是GPU的主要计算单元，负责执行并行计算任务。每个SM都包含多个流多处理器（CUDA核心），可以同时执行多个线程块中的指令。SM通过分配线程、调度指令和管理内存等操作，实现高效的并行计算。
2. **WARP(Wavefront Parallelism)**：WARP指的是一组同时执行的Thread。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。
3. **Dispatch Unit**：从指令队列中获取和解码指令，协调指令的执行和调度，将其分派给适当的执行单元，以实现高效的并行计算。
4. **L1 Cache/SMEM**：L1 Cache包含指令缓存(Instruction Cache)和数据缓存（Data Cache），在SM内部存储最常用的指令和数据，每个SM独享一个L1 Cache，提供低延迟和高带宽的访问。
5. **Register File**: 用于存储临时数据、计算中间结果和变量。离计算单元最近，访问速度非常快。
6. **SFU(Special Function Unit)**：SFU在GPU中用于加速特定类型的计算操作。如三角函数等。



Hopper

TensorCore，Tensor Memory Accelerator是主要用于加速AI计算负载中的矩阵乘法和累加(Matrix Multiply-Accumulate, MMA)，我们稍后对其进行更加具体的讲解。

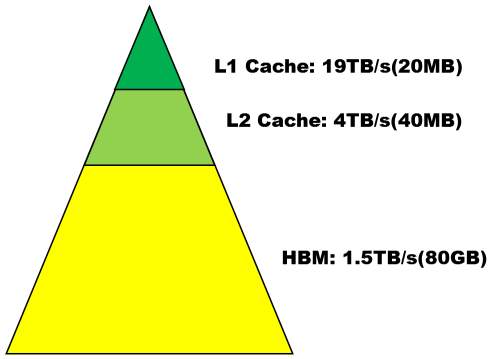
> 💡 **SIMD和SIMT有什么区别?**
>
> TL;DR: SIMT相对于SIMD在复杂计算的情况下，对开发人员和编译器的要求更低，更好优化。在数据的比特位小的时候，在SIMT中增加SIMD来增加GPU的计算能力，这就是为什么有些GPU的16位/8位计算的算力会是32位计算的多倍的情况。从现代GPU编程角度看SIMD与SIMT
>
> 为什么要有**Warp**?

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。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，便于编程优化，降低GPU计算资源浪费。

通过上面自顶向下的分析，我们知道，对于GPU中的存储部分访问速度由快到慢，计算部分从大到小排列为

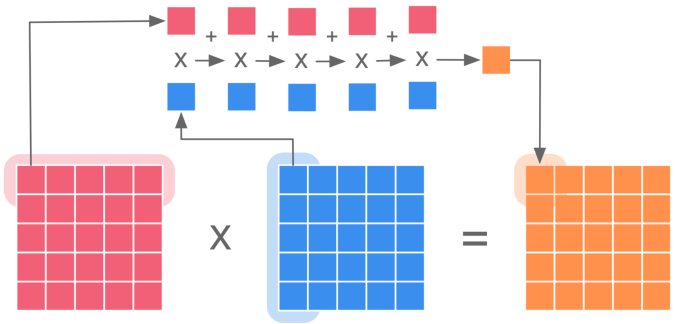
$$\text{Mem Speed:(L1 Cache/SMEM)>L2 Cache>HBM}$$
$$\text{Compute Unit:GPC>TPC>SM>(TensorCore, SFU, INT32, FP32..)}$$

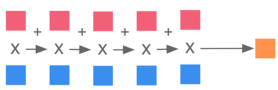
其中访存速度如下图所示(以A100为例)



LD/ST Mem Matters

接下来，我们引入一个计算强度(Compute Intensity)的概念来表示一段代码的计算时间复杂度与访存时间复杂度的比例。简单来讲，就是一个数据我访问一次需要做多少次运算。下面是一个计算强度的例子





For an NxN matrix:

- N row elements multiply with N column elements
- N additions create the final result
- This is done N^2 times, once for each result element

Arithmetic complexity is therefore: $N^2 \times (2N) = 2N^3 = O(N^3)$

Number of data loads: $O(N^2)$

Arithmetic intensity scales as: $N^3 / N^2 = O(N)$

对于N阶方阵的乘法操作，其计算时间复杂度为 $\mathrm{O}(N^3)$ ，访存时间复杂度为 $\mathrm{O}(N^2)$ ，其计算强度为 $\mathrm{O}(N)$.

根据计算强度的定义可知，对于计算能力固定的一个设备，如果想让计算单元达到更高的利用率，则对于越慢的内存，我们需要算法的计算强度就越大，即访问一次内存，多进行几次运算。

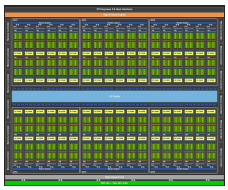
右图展示了当数据在GPU不同存储设备上时，计算单元达到最大利用率所需的计算强度。

我们可以通过并行提升整体计算强度以充分利用GPU能力。当我们的计算强度不够的时候就要想办法优化IO来降低端到端的延迟。

Data Location	Bandwidth (GB/sec)	Compute Intensity	Tensor Core
L1 / Shared	19,400	8	32
L2 Cache	4,000	39	156
HBM	1,555	100	401
NVLink	300	520	2,080
PCIe	25	6,240	24,960

From Vlota To Hopper

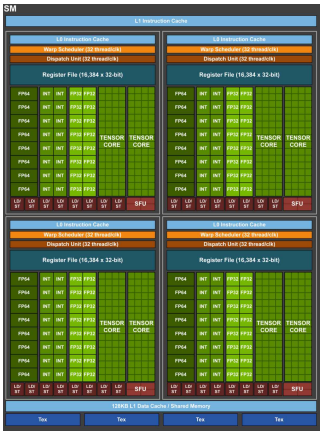
Vlota-V100



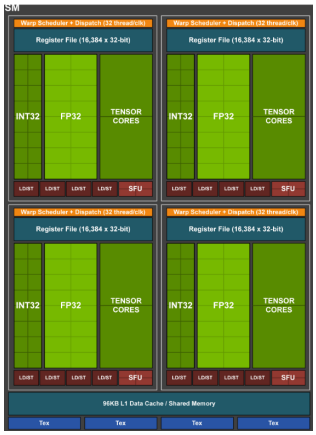
Turing-2080Ti

Ampere-A100

Hopper-H100



Vlota



Turing-(RT Core)

Ampere

Hopper

Arch	Volta	Turing	Ampere	Hopper
Year	2017	2018	2020	2022
Transistor	12nm/21.1B	12nm/18.6B	7nm/28.3B	4nm/80B
SM	80	92	108	132
L1 Cache	128KB	96KB	192KB	256KB
Per SM	32 FP64+64 Int32+64 FP32 +8 Tensor Core	64 Int32+64 FP32 +8 Tensor Cores	64 FP32+64 INT32+32 FP64 +4 Tensor Cores	128 FP32+64 INT32+64 FP64 +4 Tensor Cores
Superiority	NVLink2.0 Tensor Core 1.0	RT Core 1.0 Tensor Core 2.0	MIG 1.0 Tensor Core 3.0	MIG 2.0 Tensor Core 4.0
Product	V100	RTX2080Ti	A100	H100

Volta

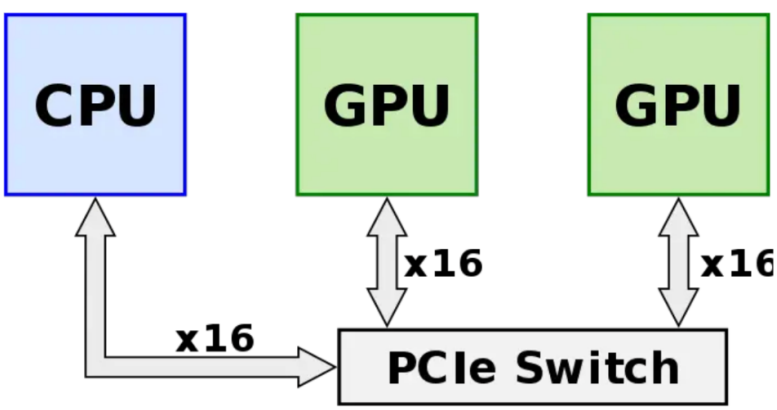
NVLink

❓ NVLink是什么？为什么需要他？

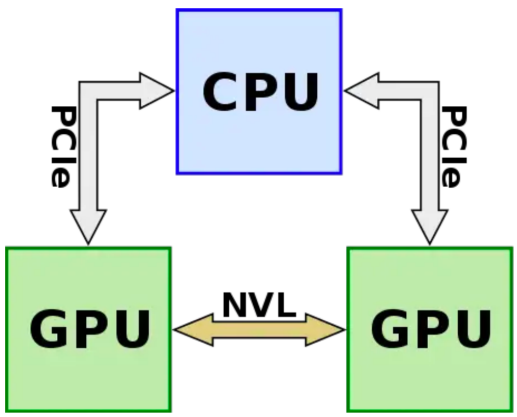
大模型通常具有巨大的参数数量和复杂的结构，需要处理大量的数据。分布式训练将这些大型模型分割成多个部分，由多个GPU或计算节点并行处理，每个部分处理自己的数据子集。然后通过全局通信，参数同步等方式进行梯度传播，此时GPU之间的通信带宽就变的越来越重要。

在NVLink出现之前，GPU与GPU之间的数据交互通过PCIe（Peripheral Component Interconnect Express）总线进行。但PCIe存在两个问题，一是PCIe总线的带宽相对有限，其中PCIe 4.0x16的最大带宽也就64GB/s，二是PCIe总线的延迟相对较高，在GPU之间传输数据时，每次数据传输都需要通过CPU和主机内存来完成。这种传输路径会导致额外的延迟，并降低数据传输的效率。然而，深度学习应用中需要更高的带宽和更低的延迟，PCIe显然是无法满足当下的神经网络训练需求。

NVLink利用高带宽、低延迟的通信通道，直接将多个GPU连接在一起，实现快速、高效的数据传输和共享。通过NVLink，GPU之间的数据交互可以直接在GPU之间进行，而无需通过CPU和主机内存。这种直接内存访问（DMA）的方式大大减少了数据传输的复制和延迟，提高了数据共享的效率。此外，NVLink还提供了一致的内存空间，使得多个GPU能够共享同一份内存，简化了程序设计和数据管理的复杂性。



Before NVLINK



AFTER NVLINK

主要思想就是GPU间采用NVLINK进行互联，实现GPC对卡间HBM的高速访问，和PCIe进行互补。

NVLINK1.0在Pascal架构中提出，从Volta开始成型，每条NVLINK提供50GB/s的通信带宽，在Volta架构每个GPU最多支持6条NVLINK，提供300GB/s的GPU间通信带宽。

💡 一定程度上可以说，GPU的计算速度决定了训练的速度，GPU间的通信带宽决定了在保证训练速度的情况下你可以训练多大的模型。

美国政府颁布的出口管制规定，出口至中国的芯片数据传输速率**不得超过每秒600GB**就是限制的这一部分，可以在很大程度上限制国内大模型的训练发展。所以Nvidia就通过降低NVLINK的速度，把A100砍成A800，最大互联带宽到400GB/s的方式来符合政府的出口禁令。

TensorCore

💡 **Tensor Core有什么用，为什么需要它?**

深度学习和机器学习中的许多任务都涉及大规模的矩阵运算，例如矩阵乘法(GEMM)和卷积(Conv)操作。这些运算通常需要大量的计算，并且在传统的GPU架构中，这些矩阵运算需要编码成 FMA(Fused-Multiply-Add)操作，再把结果从ALU到寄存器搬来搬去，每个浮点数计算都需要多个指令周期。这限制了GPU在处理这些任务时的效率，导致性能瓶颈。

Tensor Core通过引入特殊的硬件单元来解决这个问题。它每周期能执行 4x4x4 GEMM，即 64 个 FMA。即每个Tensor Core提供了相当于64个ALU，同时能耗上还有优势。

Tensor Core1.0虽然只支持 FP16 数据，但输出可以是 FP32，相当于 64 个 FP32 ALU 提供的算力。

Turing

图灵架构相比于上代Volta架构，仍采取NVLINK2.0，但条数增加到8条。

Tensor Core2.0相比于上代只支持FP16数据，新增INT8，INT4，INT精度模式，提供更快的运算速度，并且为DLSS技术赋能。

新增了RT Core，主要用于图形渲染中的光线追踪功能，我们这里不再展开。

Ampere

安培架构相比于上代，升级到NVLINK3.0，但条数增加到12条，提供600GB/s的GPU间互联带宽。

Tensor Core3.0相比于上代新增FP64、TF32、BF16精度模式。

新增稀疏矩阵加速运算技术，使标准操作的性能提高了一倍。

新增MIG(Multi-Instance GPU)技术，可以将单个 GPU 划分为多个GPU实例，每个实例的 SMs 在整个内存系统中都有独立的独立路径—片上交叉条端口、二级缓存库、内存控制器和 DRAM 地址总线都是唯一分配给单个实例的。利用这种能力，云服务提供商可以使用 MIG 来提高其 GPU 服务器的利用率，在不增加成本的情况下提供最多 7 倍的 GPU 实例。

Hopper

Hopper架构升级到NVLINK4.0，条数增加到18条，提供900GB/s的GPU间互联带宽。

	Second Generation	Third Generation	Fourth Generation
NVLink bandwidth per GPU	300GB/s	600GB/s	900GB/s
Maximum Number of Links per GPU	6	12	18
Supported NVIDIA Architectures	NVIDIA Volta™ architecture	NVIDIA Ampere Architecture	NVIDIA Hopper™ Architecture

Tensor Core在本代新增了FP8的运算能力，去除了IN4计算逻辑。

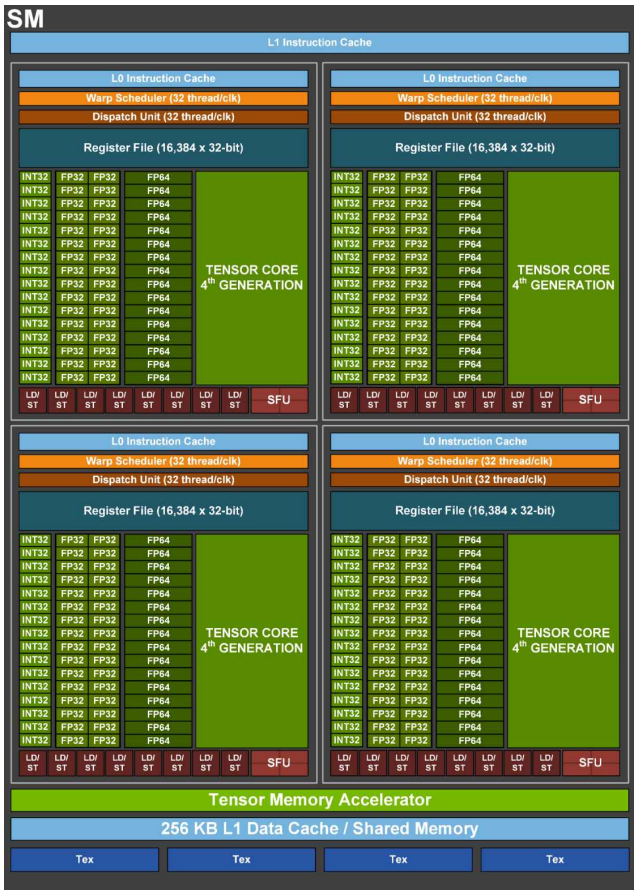
	Supported CUDA Core Precisions									Supported Tensor Core Precisions								
	FP8	FP16	FP32	FP64	INT1	INT4	INT8	TF32	BF16	FP8	FP16	FP32	FP64	INT1	INT4	INT8	TF32	BF16
NVIDIA Tesla P4	No	No	Yes	Yes	No	No	Yes	No	No	No	No	No	No	No	No	No	No	No
NVIDIA P100	No	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	No
NVIDIA Volta	No	Yes	Yes	Yes	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No	No
NVIDIA Turing	No	Yes	Yes	Yes	No	No	Yes	No	No	No	Yes	No	No	Yes	Yes	Yes	No	No
NVIDIA A100	No	Yes	Yes	Yes	No	No	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
NVIDIA H100	No	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	No	Yes	No	No	Yes	Yes	Yes

前三代 Tensor Core 基于WARP Level 进行编程:通过 SIMT 完成矩阵计算，将数据从全局内存加载到寄存器上，再通过 Warp Scheduler 调用Tensor Core完成矩阵乘法，最后将结果写出到寄存器。 但这样存在一些问题，Tensor Core 准备数据时，warp 内线程分别加载矩阵数据， 每一个线程都会获取独立矩阵块地址。多级缓存Cache的存储空间限制，单个 warp 的矩阵计算规格有上限。

为了解决这些问题，Hopper Tensor Core相比于Ampere有很大改进。新加入**TMA(Tensor Memory Accelerator)，**实现硬件异步数据加载，即全局内存中的数据可 以被异步地加载到共享内存。 采用单线程schedule模型，不需要所有线程都参与，同时使用 TMA 之前只需要一次性配置好首地址、偏移量等Tensor描述信息。

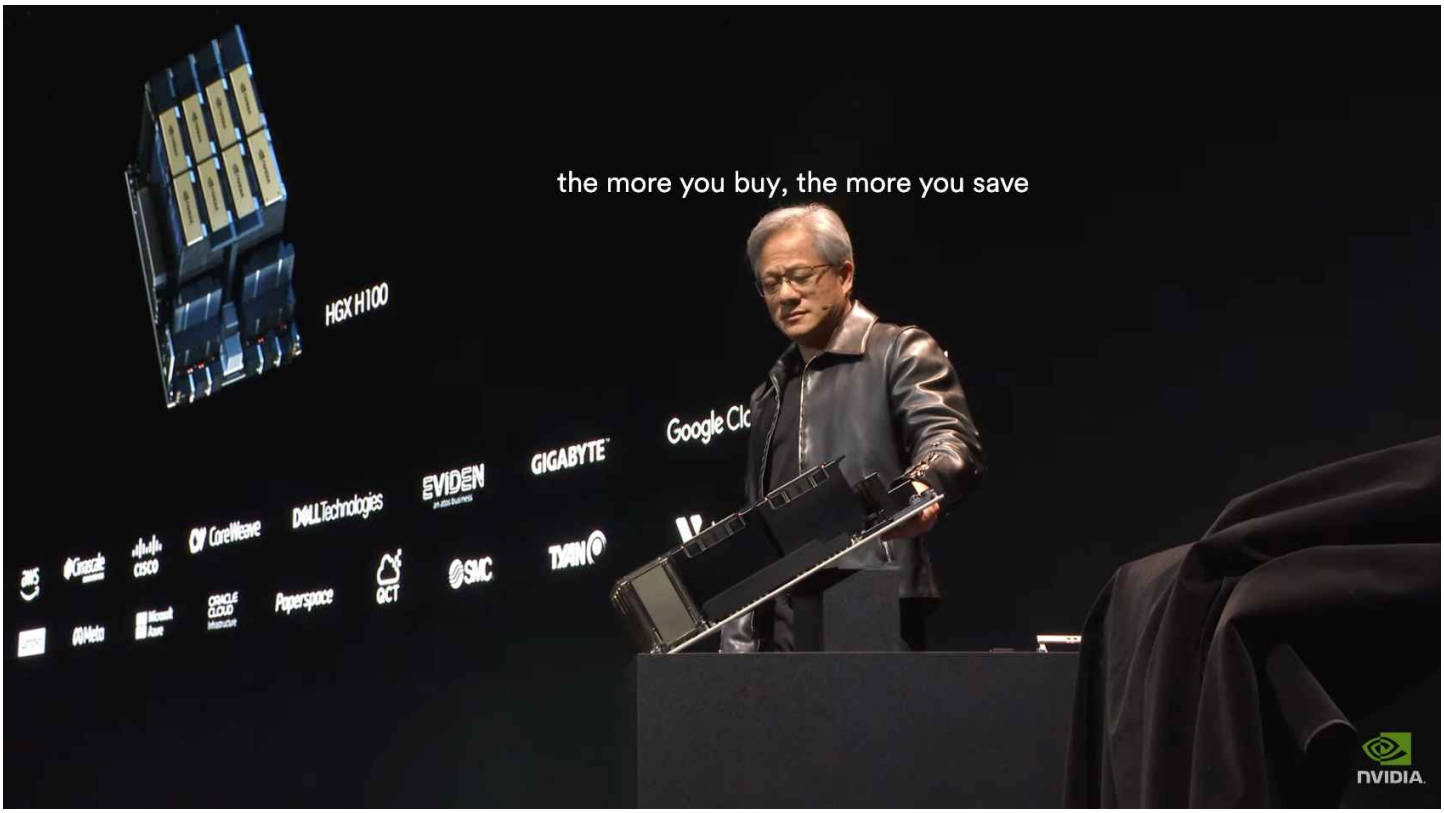
TMA 将 SM 组织成一个更大的计算和存储单元，完成数据从 global 到 shared memory 的异步加载，数据到寄存器的计算，最后通过硬件实现了矩阵乘法的流水线。

Ampere

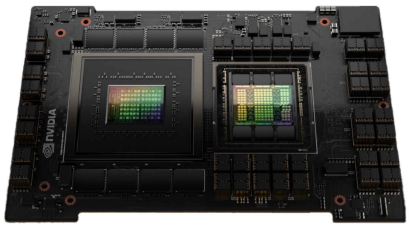


Hopper

至此，更快的运算速度，更大的通信带宽结合就完成了当下供不应求的最强加速卡H100。下面以Nvidia CEO黄仁勋在Computex 2023上的一句话作为结尾。The more you buy, the more you save.



对H100用到的其他技术(例如Transformer Engine，DPX Instructions)和一体式解决方案Grace Hopper Superchip感兴趣的同学可以点击预约下期。



信息收集