GENERAL-PURPOSE GPU · CORE TECHNOLOGIES

# GPGPU 核心技术介绍

从 SIMT 执行模型到层次化内存，深入理解 GPU 通用计算的核心技术原理。

COURSE MAP

# From *Memory* to Model.

先建立存算分离的架构原点，再看 GPU 如何演化、如何执行、如何优化。






00 架构原点：存算分离
冯·诺依曼架构解释 GPU 设计的长期问题。

01 历史脉络：图形 → 通用计算
从固定管线、统一着色器走到 CUDA 和 AI 计算。

02 核心原理：SIMT 执行模型
Thread、Warp、Block、Grid 构成 CUDA 心智模型。

03 性能优化：分支、访存、搬运
从单卡 kernel 到多 GPU 系统，看性能如何真正跑出来。

VON NEUMANN ARCHITECTURE · MEMORY WALL

# 冯·诺依曼架构：存算分离的起点

指令和数据放在存储器里，计算单元通过总线取数、执行、写回。这个清晰模型也带来了最核心的瓶颈：数据必须被搬到计算旁边才能被处理。


## 从存算分离到近存计算

在硬件层面不断拉近存储和计算的空间距离

**01 · 分离**

存储器负责保存，计算单元负责执行。结构清晰，但每次计算都依赖取数路径。

**02 · 瓶颈**

当 ALU 变快，带宽、延迟、能耗逐渐被数据搬运主导。

**03 · 回答**

GPU 用层次化内存、高带宽封装和近存数据复用来缓解这个结构性问题。

一条主线：计算越来越快，数据搬运越来越贵

SECTION 01 · HISTORY

# 从图形加速到通用计算

先看 GPU 如何从服务像素的固定管线，逐步演化成可编程的大规模并行处理器。


01 GPU 发展的三个时代

02 固定管线到统一着色器

03 CUDA 与 AI 计算生态

THREE ERAS · 1963 – 2026

# GPU 发展的三个时代

1963 – 2006

## 图形加速时代

从 Sketchpad 到
统一着色器架构

2006 – 2017

## CUDA 通用计算

GPU 从图形渲染器
变成并行计算器

2018 – 至今

## AI 工厂时代

超节点互联
万亿参数大模型

三个时代，一条主线：从"专用"到"通用"，从"单卡"到"集群"

1963 – 2006 · FROM SKETCHPAD TO G80

# 早期 GPU：从图形加速到可编程着色器

| | | |
|---|---|---|
| 1963 | Sketchpad | Ivan Sutherland · 第一个交互式图形程序 · 图灵奖 |
| 1982 | SGI | Jim Clark 创立 · Geometry Engine · OpenGL 标准化 |
| 1993 | NVIDIA | 黄仁勋等三人创立 · Denny's 餐厅 · 4 万美元起步 |
| 1999 | GeForce 256 | "世界上第一个 GPU" · 硬件 T&L · 固定功能管线巅峰 |
| 2006 | GeForce 8800 | G80 · 统一着色器架构 · 128 SP · CUDA 兼容 |

BEFORE / AFTER · 1996 – 2006

# 固定功能管线 → 统一着色器架构

BEFORE 固定管线

## 硬件写死

光照、纹理、几何变换全部"烧入硅片"。开发者无法自定义渲染效果。顶点和像素着色器物理分离，负载不均时空闲浪费。

- 3dfx Voodoo (1996)：纯 3D 加速附加卡
- GeForce 256 (1999)：硬件 T&L 固定管线
- DirectX 7.0：无着色器编程能力

AFTER 统一着色器

## 软件定义

同一组硬件单元动态分配执行顶点、像素、几何着色器任务。利用率大幅提升。为 CUDA 通用计算奠定硬件基础。

- Xenos (2005)：Xbox 360 · 首个统一架构
- GeForce 8800 (2006)：PC 首个统一架构
- DirectX 10 / SM 4.0：128 流处理器

2006.11.08 · NVIDIA GTC

# CUDA 诞生：GPU 通用计算的民主化时刻

Jensen Huang 在发布 GeForce 8800 GTX 的同一天揭开了 CUDA 的面纱。从此开发者可以用类 C 语言直接在 GPU 上编写并行程序——彻底告别了将计算"伪装"成图形操作的痛苦时代。

Ian Buck 从斯坦福的 BrookGPU 出发加入 NVIDIA，将学术探索推向产业化。

Thread → Warp → Block → Grid 

WHY GPU FOR DEEP LEARNING

# 深度学习为什么选择 GPU?

大规模并行

## 矩阵乘法天然适合 SIMT 并行

神经网络的核心运算是矩阵乘加，可分解为数千个独立线程并行执行，GPU 的 SIMT 模型天然匹配。

01

高带宽

## HBM 显存带宽远超 CPU 内存

H100 的 HBM3 提供 3 TB/s 带宽，是 DDR5 的数十倍。训练时海量梯度数据需要极致带宽。

02

生态护城河

## CUDA + cuDNN 软硬件协同

2014 年 cuDNN 发布后，所有主流框架（PyTorch、TensorFlow）底层都跑在 CUDA 上，生态不可替代。

03

SECTION 02 · EXECUTION MODEL

# GPGPU 核心原理

从使用范式、并行模型和线程层级入手，建立写 GPU 程序前必须具备的执行模型。

01 GPU 与 GPGPU 的使用范式

02 SIMD / SIMT / MIMD

03 Thread → Warp → Block → Grid

CONCEPT · GRAPHICS VS COMPUTE

# GPU → GPGPU
# 从图形处理器到通用计算引擎

GPU　图形处理器

## 为像素而生

设计初衷是加速 3D 图形渲染流水线：顶点变换 → 光栅化 → 像素着色 → 帧缓冲输出。固定功能硬件单元（TMU、ROP）主导。

- 专用硬件：ROP、TMU、固定管线
- 编程接口：OpenGL / DirectX / Vulkan
- 数据模型：三角形、纹理、像素

GPGPU　通用计算 GPU

## 为计算而生

将 GPU 视为大规模并行处理器，执行任意数据并行算法。关键突破：统一着色器 + 可编程性 + 共享内存 + 通用编程语言（CUDA）。

- 通用 ALU：SIMT 执行模型
- 编程接口：CUDA / OpenCL / HIP
- 数据模型：线程、Block、Grid
- 硬件取舍：Tensor Core / HBM / NVLink 优先

演进路径 ■ 固定管线 → 可编程着色器 → 统一着色器 → **GPGPU / CUDA / AI 加速**

SIMD → SIMT → MIMD

# 三种并行执行模型

核心问题

## 一条指令，如何操作多个数据？

从向量到线程再到多核，三种模型给出了不同粒度的答案。

### 01 SIMD

单指令多数据 · 向量架构。一条指令对多个数据元素锁步执行。向量宽度对编译器"可见"。适合规则的数据并行。

### 02 SIMT

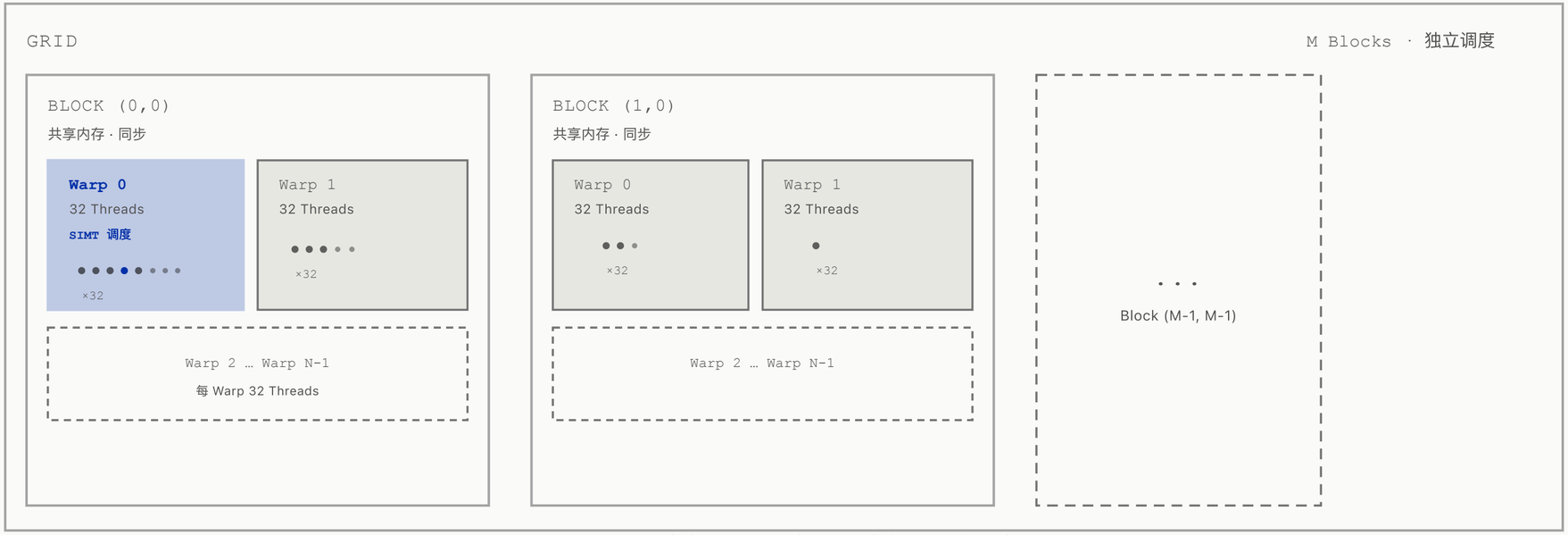
单指令多线程 · GPU 核心模型。编程视角是独立标量线程；传统硬件以 warp 共享 PC + active mask 发射指令，Volta 后线程执行状态更独立。

### 03 MIMD

多指令多数据 · 多核独立。不同核心执行不同指令流。Tenstorrent Wormhole 为代表，强调多核独立调度 + 高带宽 NoC 互联。

THREAD → WARP → BLOCK → GRID

# SIMT 线程组织：四层嵌套




**Grid = M × M Blocks · 每个 Block 含 N 个 Warp · 每个 Warp 含 32 个 Thread**

- Warp = 最小调度单元 (SIMT)
- Block = 共享内存 & 同步边界
- Grid = 一次 Kernel 启动的全部 Block

SECTION 03 · PERFORMANCE

# 性能优化与现代架构

有了执行模型之后，接下来换到性能视角：分支、访存、搬运和系统互联，决定 GPU 算力能不能真正跑出来。

01 分支发散与 warp 执行效率

02 层次化内存与显式数据搬运

03 单卡 kernel 到多 GPU 系统

WARP DIVERGENCE

# 分支分歧：SIMT 的阿喀琉斯之踵

IDEAL 无分歧

## 统一路径

Warp 内 32 个线程全部走同一分支。硬件满负荷执行，所有 ALU 都在做有用功。

- if (tid < 16) → 线程 0-15 走 A，16-31 走 B
- 理想情况：同一 warp 内线程走相同路径
- 优化策略：按 warp 边界设计条件分支

REALITY 分支发散

## 串行回放

部分线程走 if、部分走 else。硬件用 SIMT Stack 处理：先执行一条路径（另一路径线程被 masked off），再回放另一条。

- if (data[tid] > 0) → 数据依赖导致不规则分支
- 最坏情况：32 个线程走 32 条不同路径
- 硬件通过 IPDom 汇合点重新同步

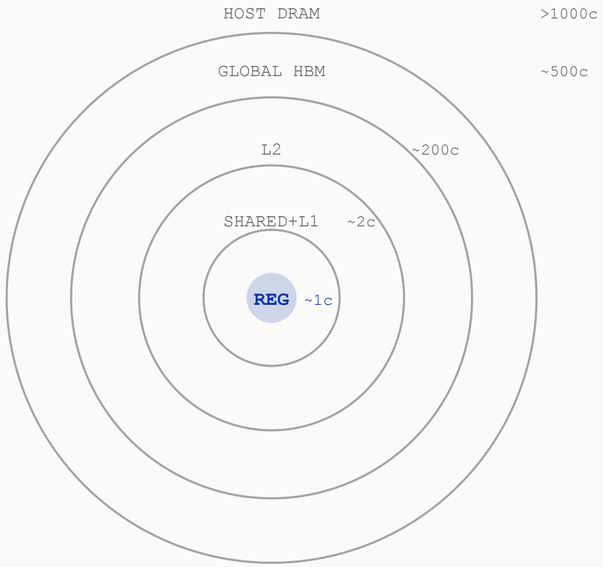
关键原则：同一 WARP 内的线程尽量走相同控制流路径

ARCHITECTURE PIPELINE & MEMORY HIERARCHY

# GPU 架构 & 内存层次

前端 Host CPU PCIe→ GigaThread Engine → Grid → Block 分配

SM 内 Warp Scheduler → Dispatch Unit → CUDA Core / Tensor / SFU → HBM




**01 · 寄存器 (REG)**

每线程私有，延迟 ~1 cycle。数量决定 Occupancy 上限。

**02 · 共享内存 + L1 (SMEM)**

Block 内共享，~2 cycles。与 L1 共享 64KB，可编程管理。

**03 · L2 + 全局内存 (HBM)**

所有 SM 共享。H100 HBM3 带宽 3 TB/s，延迟 ~400-600c。访存模式决定有效带宽。

**04 · 主机内存 (HOST)**

PCIe 总线访问，延迟最高。数据拷贝常成为性能瓶颈。

FROM STORE/COMPUTE SPLIT TO NEAR-MEMORY COMPUTING

# GPU 为什么不断设计更好的数据搬运?

硬件负责拉近空间距离，软件负责安排搬运时机。高性能 kernel 的本质，是让数据在正确的时间出现在正确的层级。

硬件层面 · 距离越来越近

**HOST DRAM**
5
最远：CPU 内存，经 PCIe / NVLink 进入设备。

**HBM**
4
封装内高带宽显存，容量大，但延迟仍高。

**L2**
3
全芯片共享，缓冲跨 SM 的数据复用。

**L1 / SMEM**
2
每个 SM 的近端存储，适合 tile 级复用。

**REG / CORE**
1
最近：寄存器贴着 CUDA Core / Tensor Core。

SPACE DISTANCE ↓ · DATA REUSE ↑

软件层面 · 指令级自由度

01 **CUDAMEMCPY + STREAM**
主机到设备的搬运异步化，让传输和计算重叠。

02 **COALESCED / VECTORIZED IO**
warp 内连续地址合并访问，向量化 load/store 减少无效内存事务。

03 **CP.ASYNC / TMA**
把 HBM → Shared Memory 的搬运提前排队，批量异步搬运形成流水。

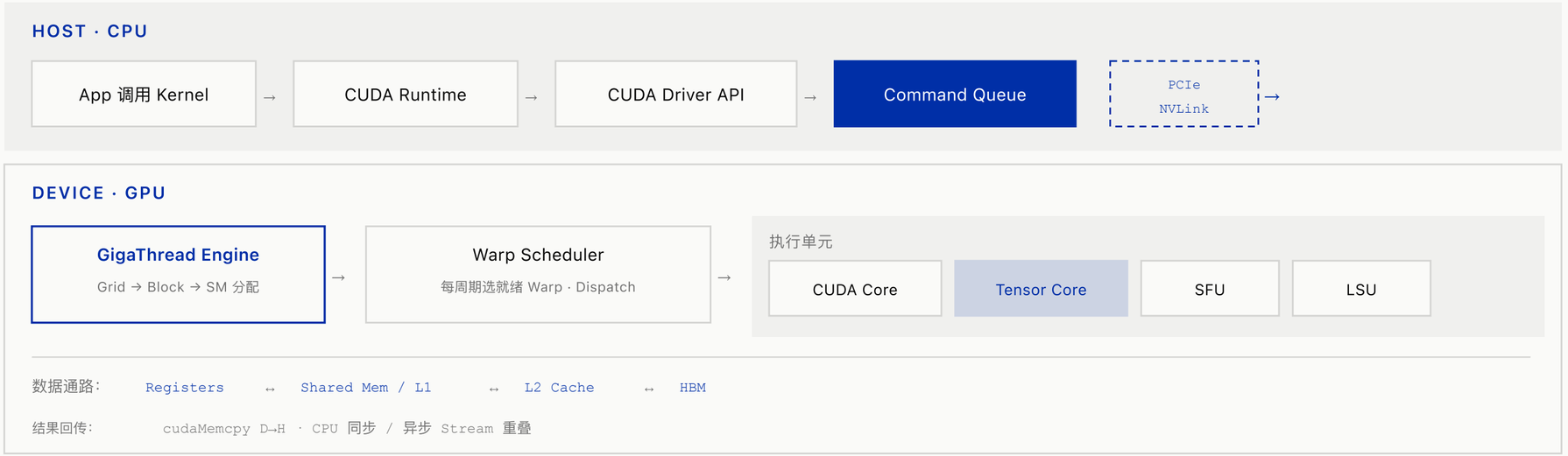
04 **WMMA / MMA + LDMATRIX**
程序员控制 tile、layout、fragment，让数据从 shared memory 精确喂给 Tensor Core。

核心判断

GPU 把搬运与矩阵计算的控制权暴露到 ISA / intrinsic 层，让程序员决定搬什么、何时搬、搬到哪。

HOST → DEVICE → HOST · KERNEL DISPATCH PIPELINE

# 算子下发全流程




**瓶颈** PCIe 传输延迟 > Kernel 计算 > Warp 调度。用 CUDA Stream 让计算与传输重叠。

PERFORMANCE PILLARS

# GPU 性能四要素

— 01 / PARALLEL

## 足够并行度

保证足够多的线程块占满所有 SM。Grid 规模太小会导致部分 SM 空闲。

— 02 / DIVERGE

## 减少分支发散

同一 warp 内尽量走同一路径。按 warp 边界组织条件逻辑，避免不规则分支。

— 03 / COALESCE

## 访存合并

连续地址的线程一起访问全局内存。非合并访问性能下降数倍到数十倍。

— 04 / REUSE

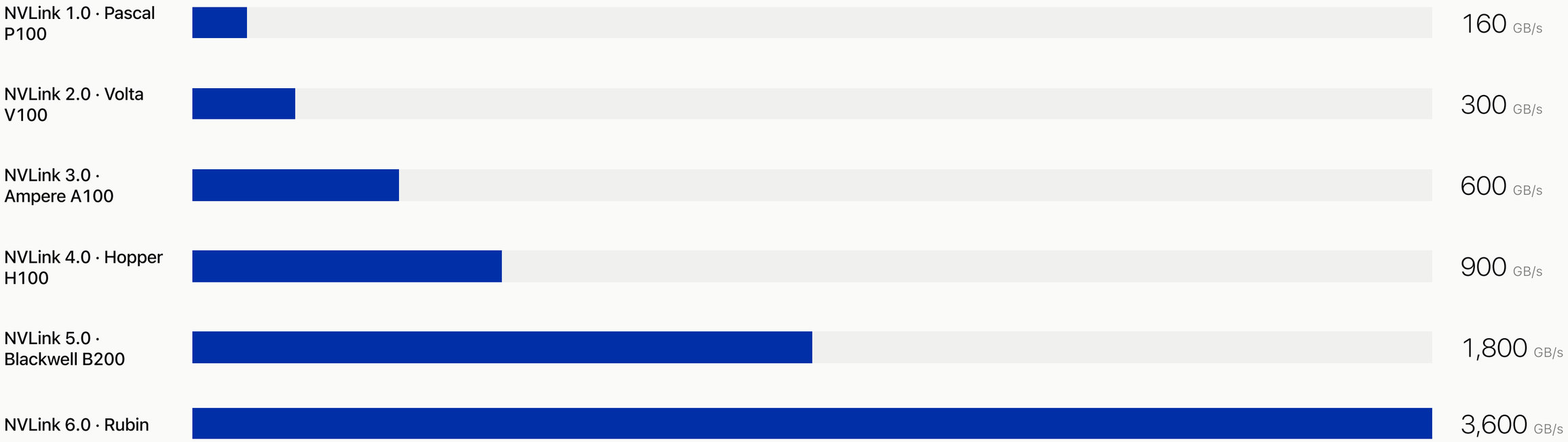
## 数据复用

频繁访问的数据优先放进共享内存或寄存器。用空间换访存次数。

NVLINK BANDWIDTH EVOLUTION

# 多节点互联：NVLink 带宽演进

从 160 GB/S 到 3600 GB/S · 十年增长 22.5 倍



ARCHITECTURE GENERATIONS · 2006 – 2026

# GPU 架构代际演进

| | | |
|---|---|---|
| 2006 | Tesla G80 | 统一着色器 + CUDA · 128 SP |
| 2010 | Fermi GF100 | 首个完整计算架构 · 512 CUDA Core |
| 2017 | Volta V100 | Tensor Core · 120 TFLOPS · NVLink 2.0 |
| 2024 | Blackwell B200 | 2080 亿晶体管 · FP4 · NVL72 机柜 |

晶体管

6.81亿→

2080亿 · 300×

CUDA CORE

128→

数万 · 持续演进

显存带宽

86GB/s→

8 TB/s · HBM

互联

PCIe→

NVLink 6.0

ASIC & DSA · DOMAIN-SPECIFIC ACCELERATORS

# ASIC 加速：专用硬件的力量

— 01 / GOOGLE

## TPU

Systolic Array 脉动阵列，大量乘加单元组成。专注矩阵乘法吞吐。Infeed → Compute → Outfeed 流水线。

— 02 / NVIDIA

## Tensor Core

Volta (2017) 首次引入。执行 4×4×4 矩阵 FMA (D = A × B + C)。V100 的 640 个 Tensor Core 提供 120 Tensor TFLOPS。H100 支持 FP8。

— 03 / AMD

## MI300X

CDNA 3 架构小芯片设计。192 GB HBM3（H100 的 2.4 倍），5.3 TB/s 带宽。ROCm 开源平台。大显存池在 LLM 推理中具优势。

— 04 / TREND

## DSA 趋势

领域专用加速器是 GPGPU 的重要补充。在 GPU 上集成专用单元，针对矩阵乘、稀疏、注意力等模式做定制加速。

THANK YOU

# 感谢聆听.

有问题欢迎继续交流。祝大家写出更高性能的 GPU kernel。





# Thank You.

TileLang Puzzle 开源训练营 · GPGPU 核心技术介绍