AI 芯片 – GPU 详解

GPU AI 编程本质

ZOMI

NVIDIA GPU架构发展

架构名称	Fermi	Kepler	Maxwell	Pascal	Volta	Turing	Ampere	Hopper
中文名字	费米	开普勒	麦克斯韦	帕斯卡	伏特	图灵	安培	赫柏
发布时间	2010	2012	2014	2016	2017	2018	2020	2022
核心参数	16个SM，每个SM包含32个CUDA Cores，一共512 CUDA Cores	15个SMX，每个SMX包括192个FP32+64个FP64 CUDA Cores	16个SM，每个SM包括4个处理块，每个处理块包括32个CUDA Cores +8个LD/ST Unit + 8 SFU	GP100有60个SM，每个SM包括64个CUDA Cores，32个DP Cores	80个SM，每个SM包括32个FP64 +64 Int32+64 FP32+8个Tensor Cores	102核心92个SM，SM重新设计，每个SM包含64个Int32+64个FP32+8个Tensor Cores	108个SM，每个SM包含64个FP32+64个INT32+32个FP64+4个Tensor Cores	132个SM，每个SM包含128个FP32+64个INT32+64个FP64+4个Tensor Cores
特点&优势	首个完整GPU计算架构，支持与共享存储结合的Cache层次GPU架构，支持ECC GPU架构	游戏性能大幅提升，首次支持GPU Direct技术	每组SM单元从192个减少到每组128个，每个SMM单元拥有更多逻辑控制电路	NVLink第一代，双向互联带宽160 GB/s，P100拥有56个SM HBM	NVLink2.0，Tensor Cores第一代，支持AI运算	Tensor Core2.0，RT Core第一代	Tensor Core3.0，RT Core2.0，NVLink3.0，结构稀疏性矩阵MIG1.0	Tensor Core4.0，NVlink4.0，结构稀疏性矩阵MIG2.0
纳米制程	40/28nm 30亿晶体管	28nm 71亿晶体管	28nm 80亿晶体管	16nm 153亿晶体管	12nm 211亿晶体管	12nm 186亿晶体管	7nm 283亿晶体管	4nm 800亿晶体管
代表型号	Quadro 7000	K80 K40M	M5000 M4000 GTX 9XX系列	P100 P6000 TTX1080	V100 TiTan V	T4，2080TI RTX 5000	A100 A30系列	H100

Why GPU

- 为什么 GPU 适用于 AI 计算？
- 为什么 AI 训练使用 GPU 而不是 CPU？

Talk Overview

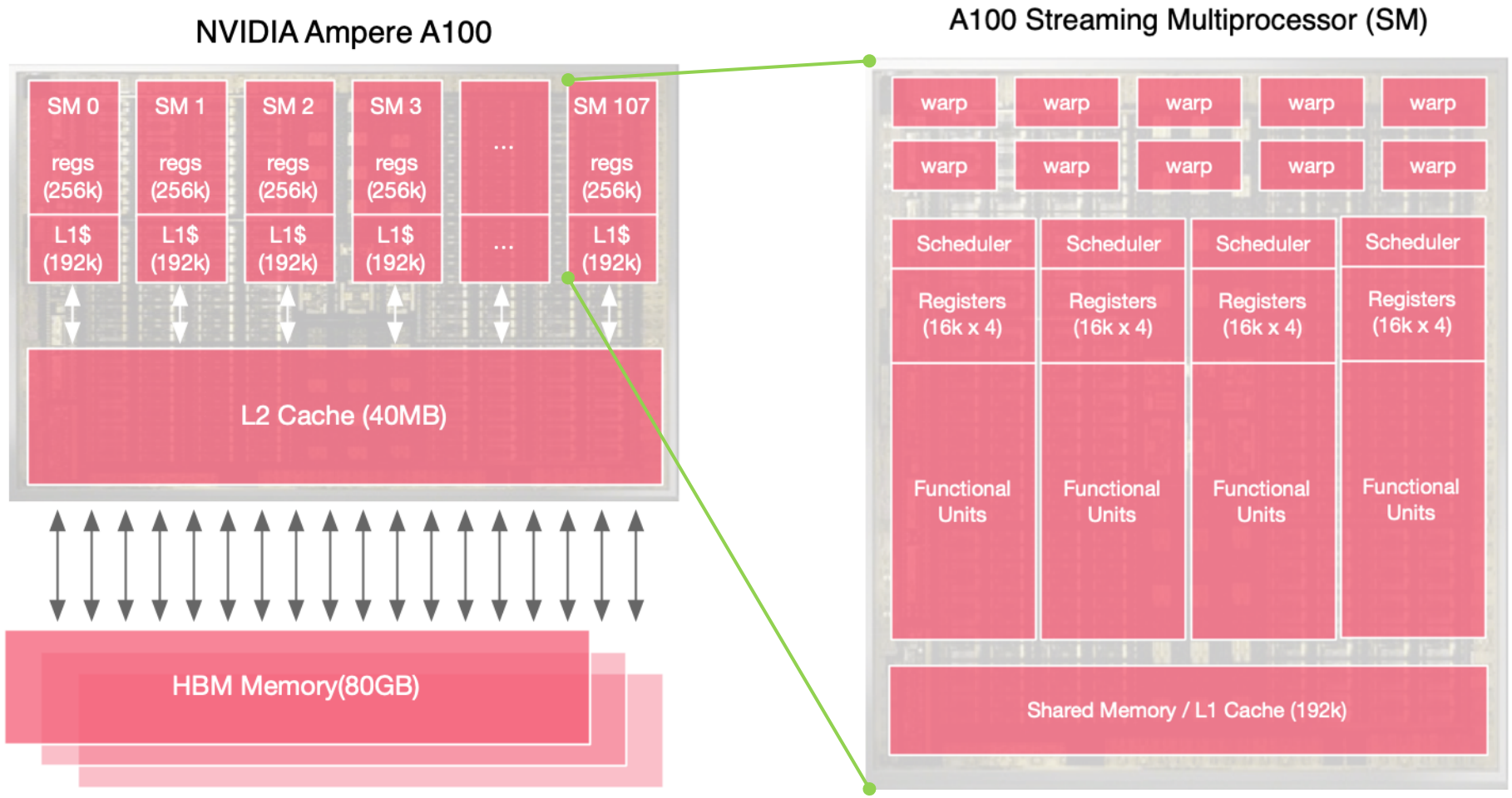
1. **硬件基础**
   - GPU 工作原理
   - GPU AI编程本质
2. **英伟达 GPU 架构**
   - GPU基础概念
   - 从 Fermi 到 Volta 架构
   - Turing 到 Hopper 架构
   - Tensor Code 和 NVLink 详解
3. **GPU 图形处理**
   - GPU 逻辑模块划分
   - 算法到 GPU 硬件
   - GPU 的软件栈
   - 图形流水线基础
   - 流水线不可编译单元
   - 光线跟踪流水线

Talk Overview

I. GPU AI编程本质

- Convolutional computation – 卷积计算
- GPU Thread hierarchical – GPU线程分级
- Relation of AI and Thread - AI计算模式和线程关系
- Algorithmic efficiency – 矩阵乘的算力利用率

GPU 线程机制



64 warps/SM

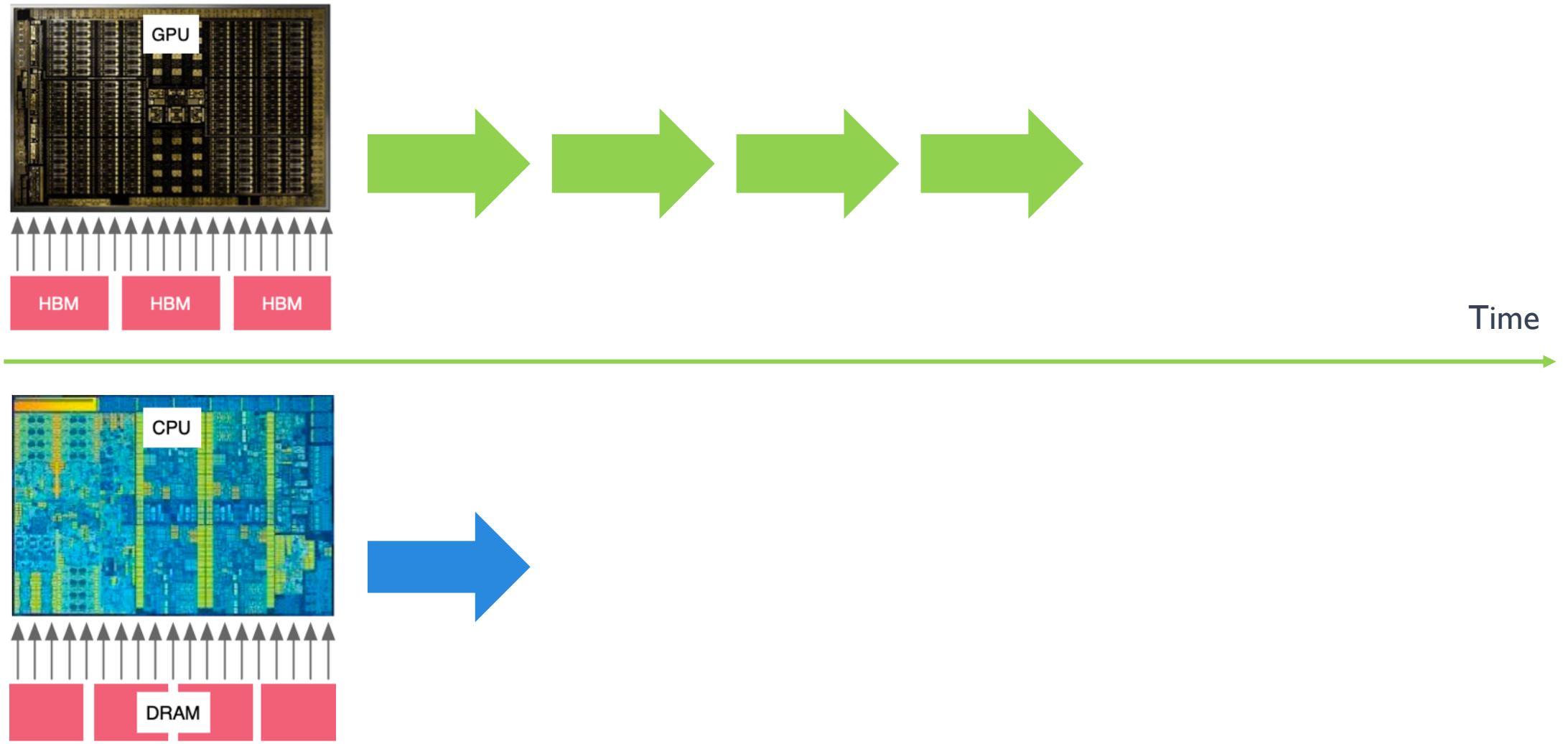
4x concurrent warp exec

64k 4-byte registers

192KB L1/shared memory (configurable split)

GPU runs threads in groups of 32 – each group is known as a warp

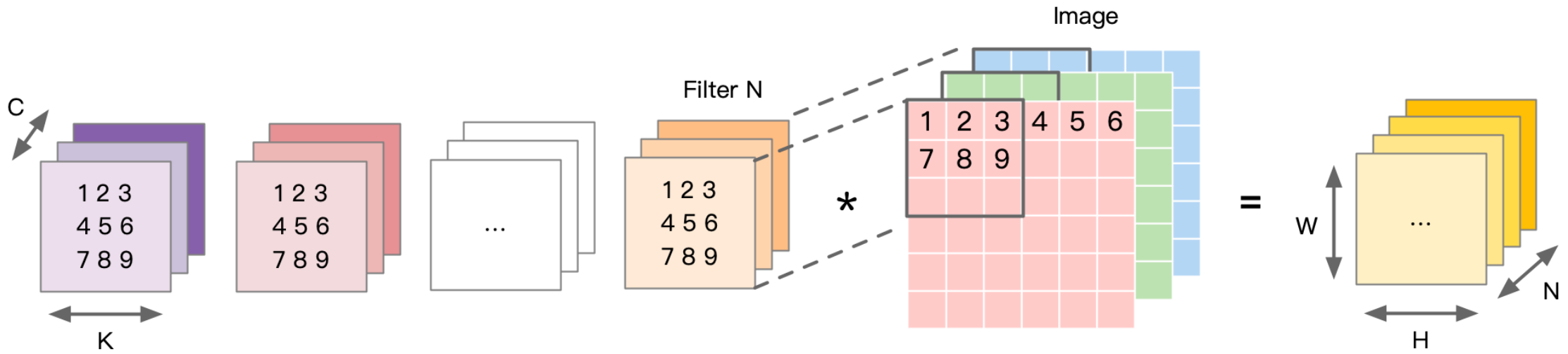
CPU/GPU 并行才是本质



卷积计算

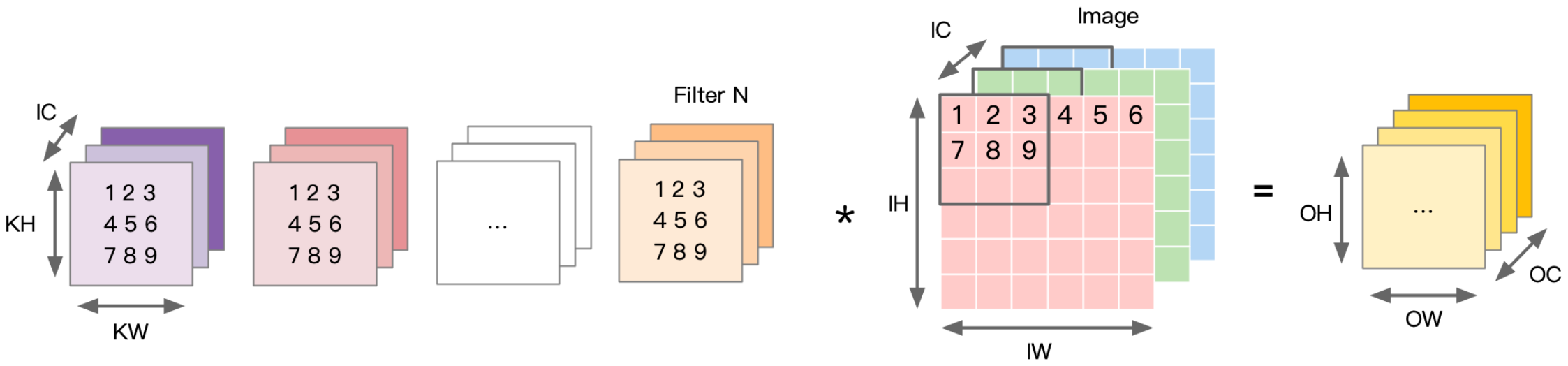
卷积通用过程

- 图像卷积正常的三通道卷积，输入维度为3维（H, W, 3），卷积核维度为（N, C, KH, KW），输出维度为（N, H, W）。卷积的一般计算方式为：



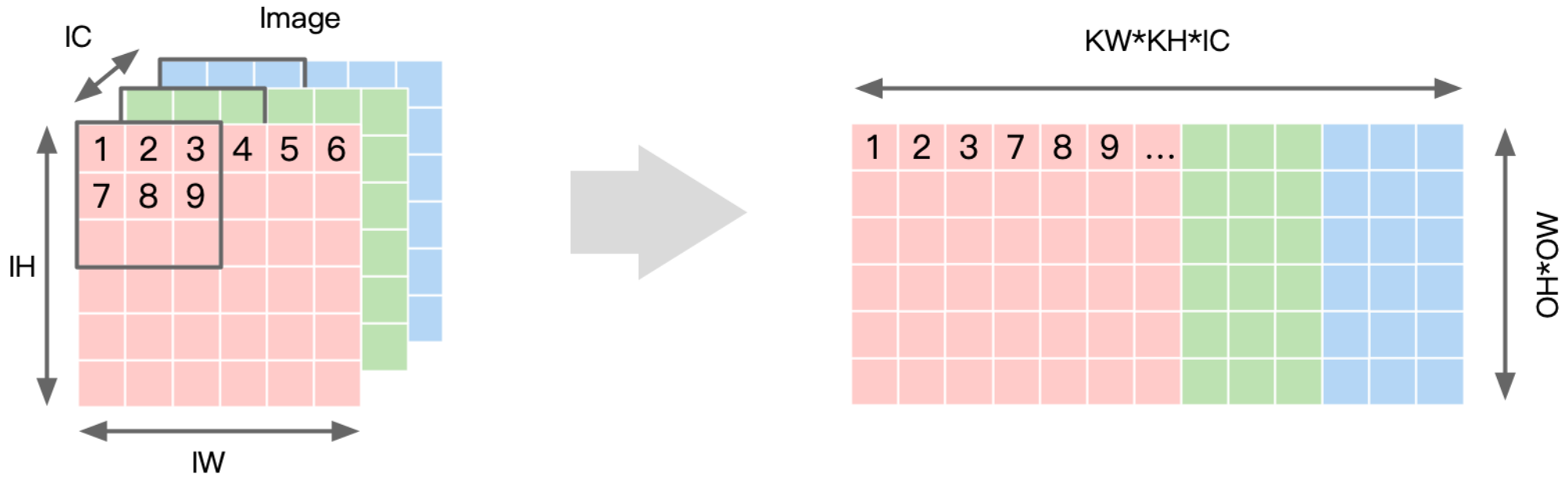
卷积通用过程

- 卷积默认采用数据排布方式为NHWC，输入维度为4维（N,IH,IW,IC），卷积核维度为（OC,KH,KW,IC），输出维度为（N,OH,OW,OC）。卷积的一般计算方式为：



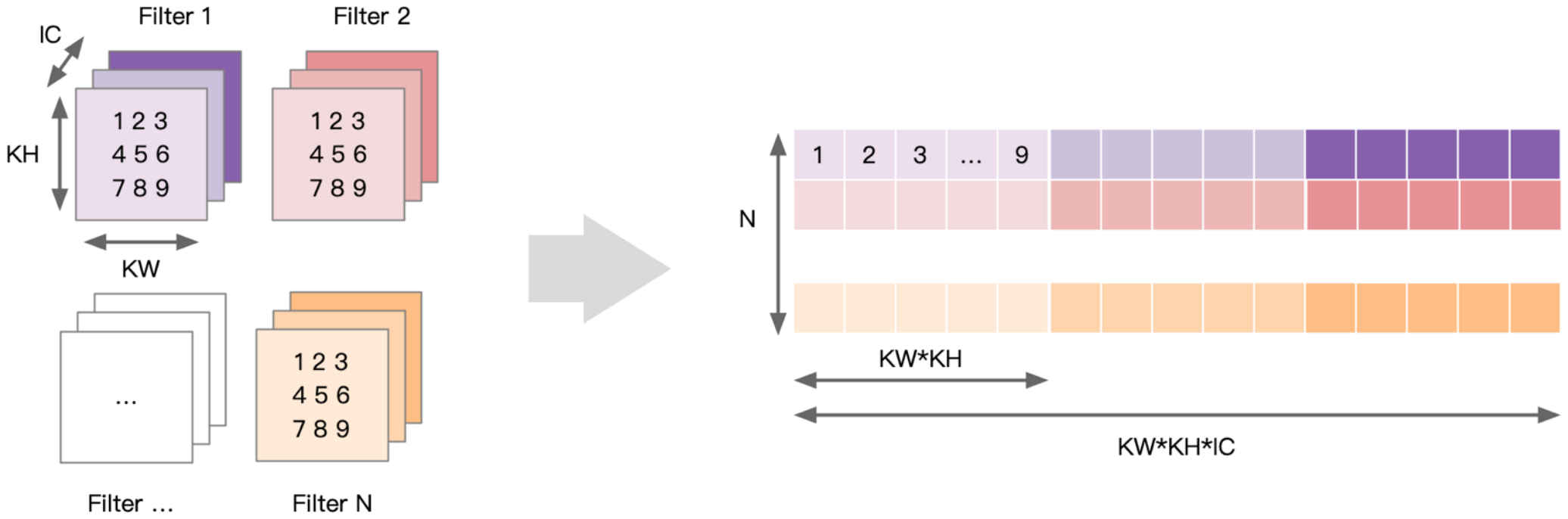
Img2col 算法过程

- 对 Input 进行重排，得到的矩阵见右侧，行数对应输出 OH*OW 个数；每个行向量里，先排列计算一个输出点所需要输入上第一个通道的 KH*KW 个数据，再按次序排列之后通道，直到通道 IC。



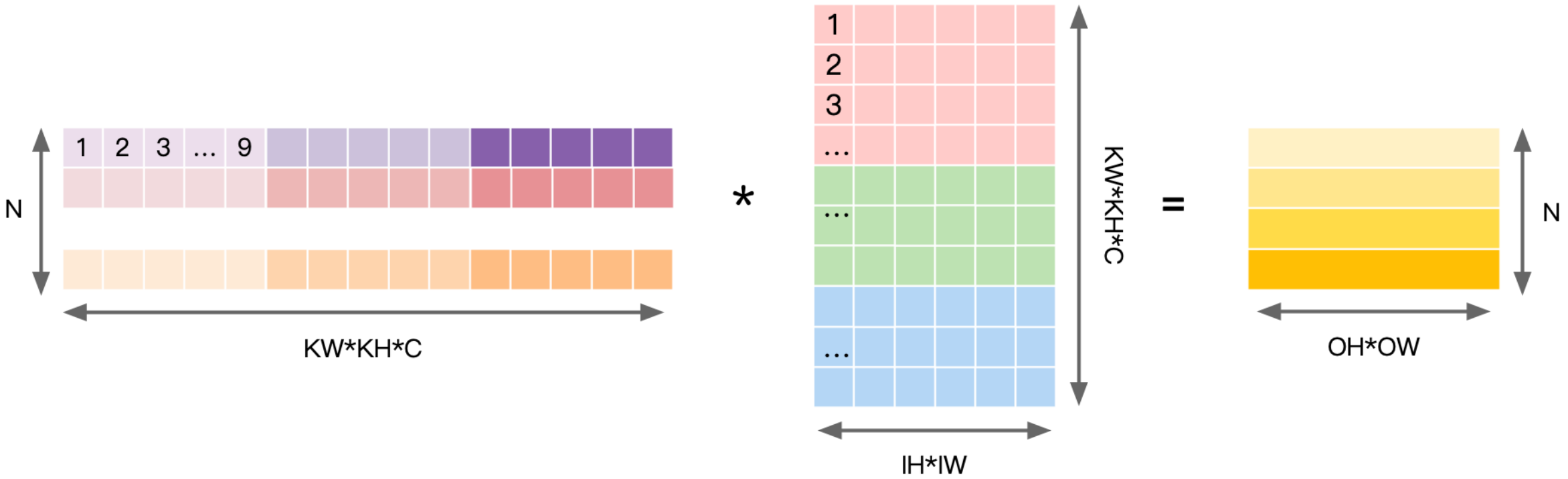
Img2col 算法过程

- 对权重数据进行重排，即以卷积的 stride 为步长展开后续卷积窗并存在 Matrix 下一列。将 N 个卷积核展开为权重矩阵的一行，因此共有 N 行，每个行向量上先排列第一个输入通道上 KH*KW 数据，再依次排列后面的通道直到 IC。



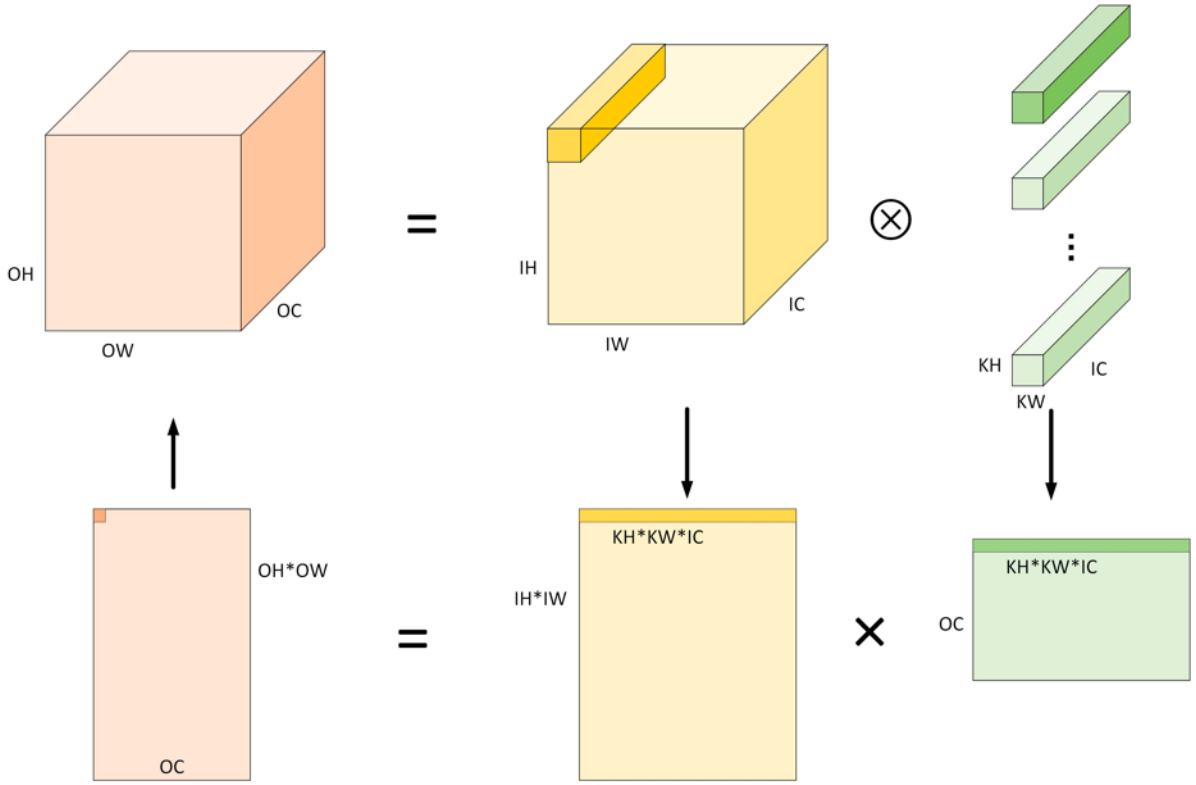
Img2col 算法过程

- 卷积操作转换为矩阵相乘，对 Kernel 和 Input 进行重新排列。将输入数据按照卷积窗进行展开并存储在矩阵的列中，多个输入通道的对应的窗展开之后将拼接成最终输出 Matrix 的一列：



Img2col 算法过程

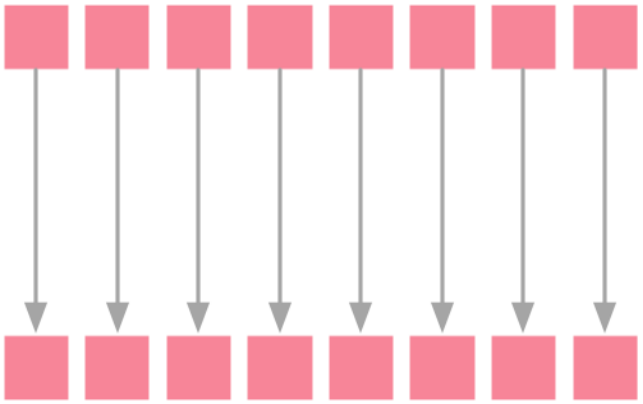
- 通过数据重排，完成 Im2col 的操作之后会得到一个输入矩阵，卷积的 Weights 也可以转换为一个矩阵，卷积的计算就可以转换为两个矩阵相乘的求解，得到最终的卷积计算结果。



GPU 线程分级

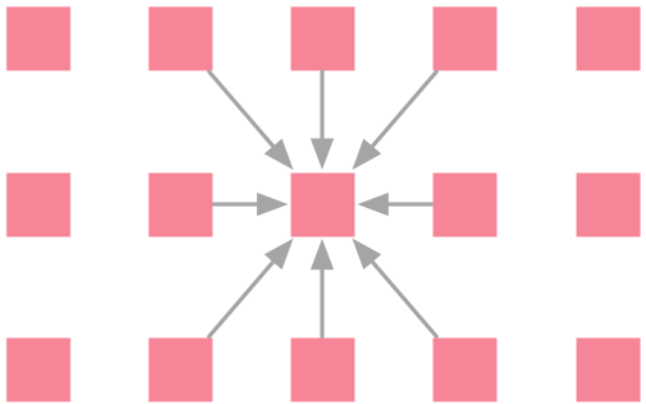
AI 计算模式与线程的关系

在AI计算模式中，不是所有计算都可以是线程独立的哦



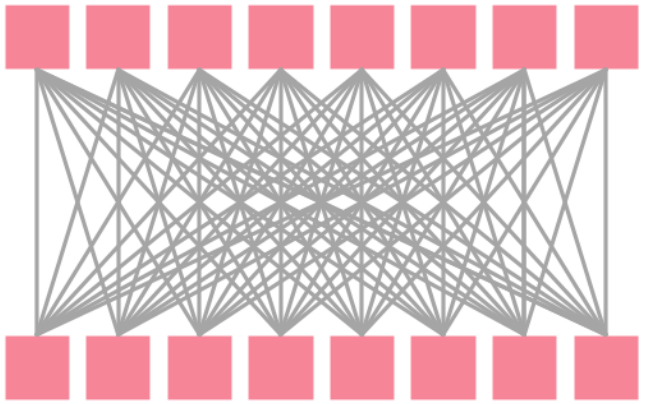
Element-wise

AX+Y



Local

Convolution

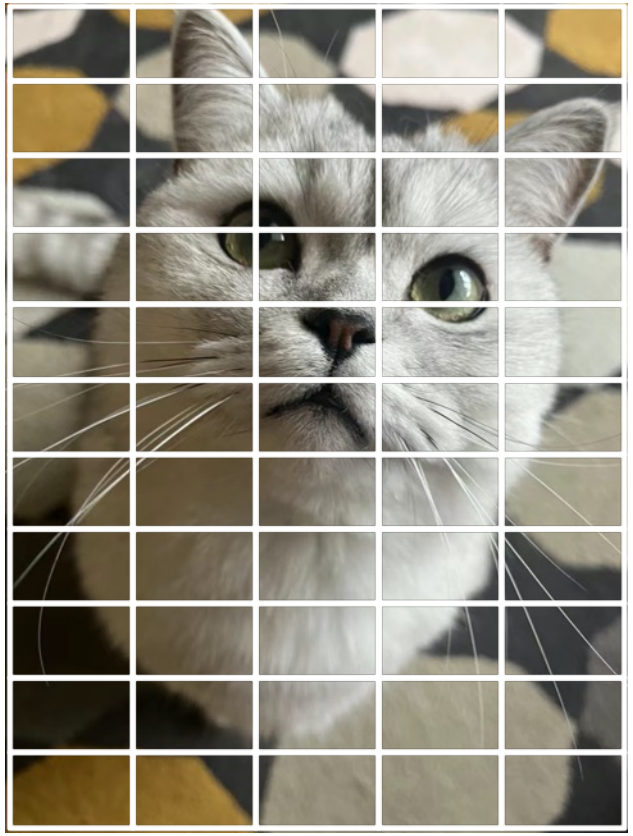


All to All

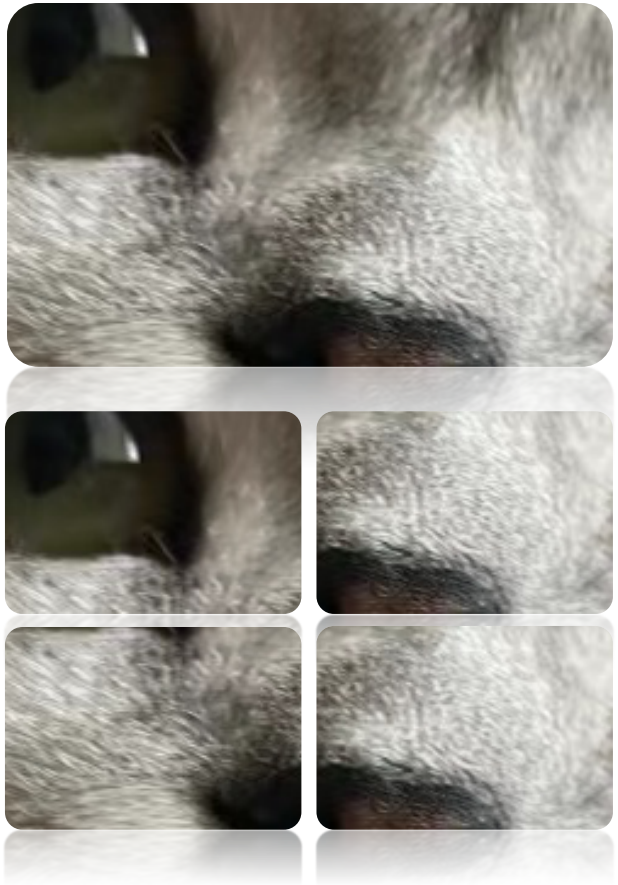
MatMul

线程分层执行

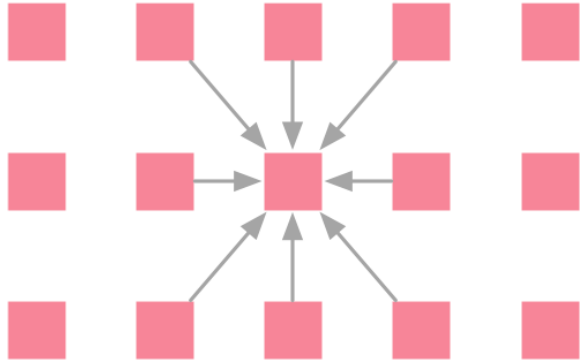
GPU 中 Blocks 是独立执行

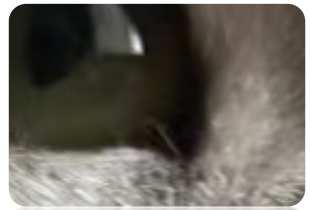


step1：用网格 Grid 进行覆盖



step2：Grid 中分块 Blocks

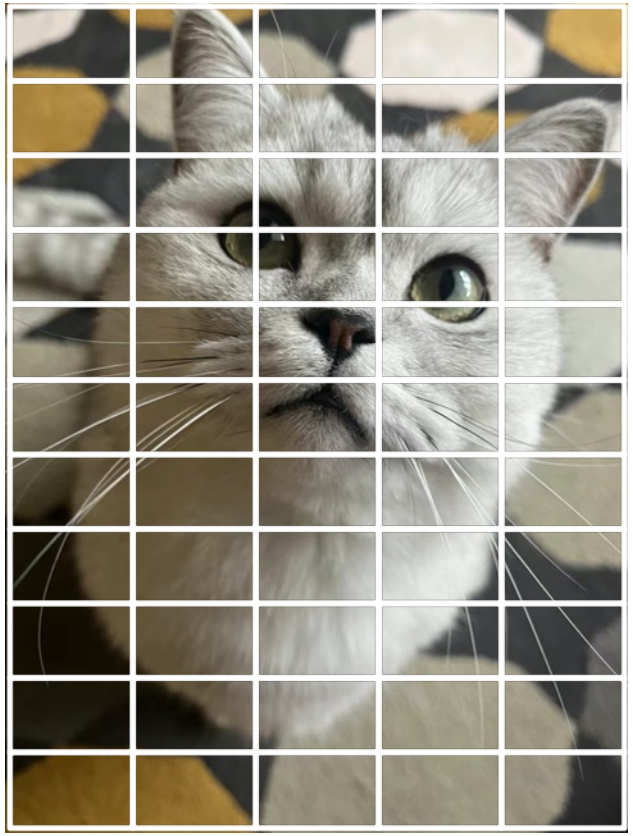




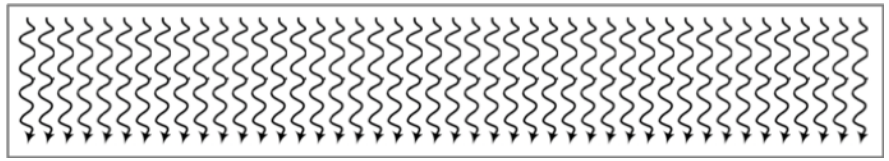
step3：Block 中线程 Threads

通过本地数据共享来计算

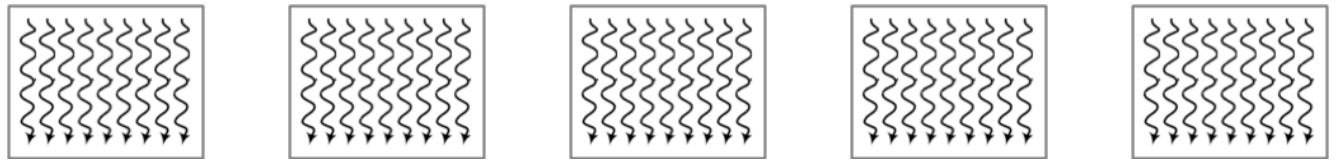
线程分层执行



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

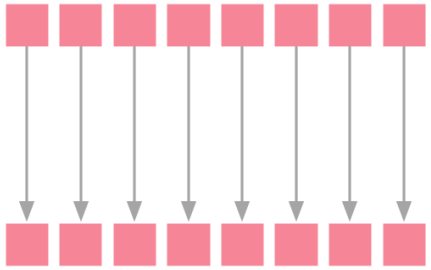
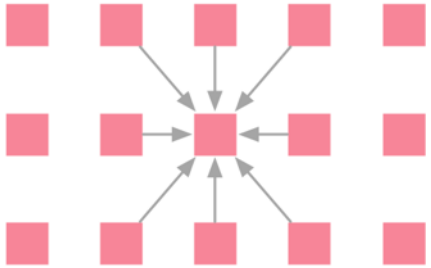
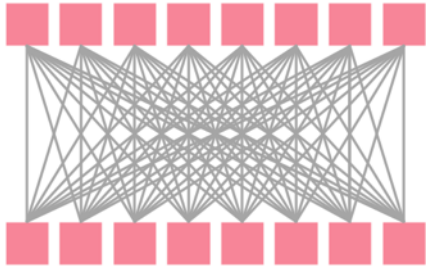


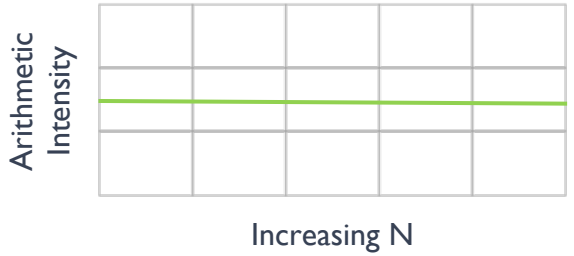
块 Block 中的线程数独立执行

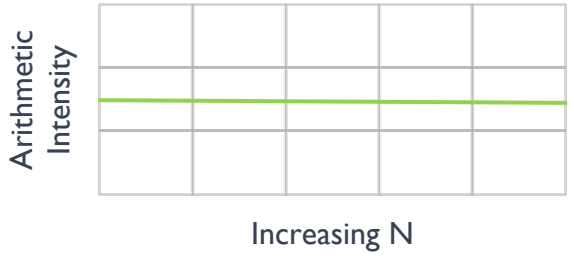
可以通过本地数据共享

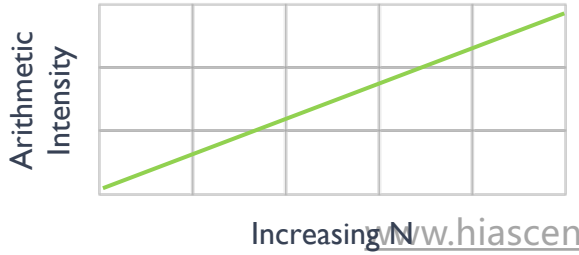
同步交换数据

并行提升整体计算强度

		Data Scales	Compute Scales	Intensity Scales
Element-wise		$O(N)$	$O(N)$	$O(1)$
Local		$O(N^2)$	$O(N^2)$	$O(1)$
All to All		$O(N)$	$O(N^2)$	$O(N)$

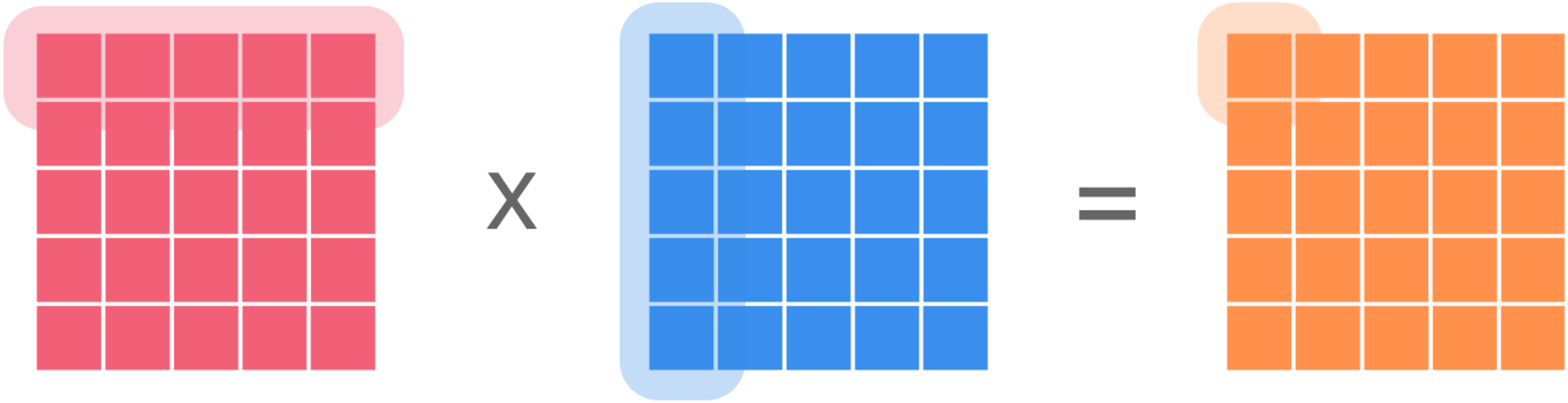




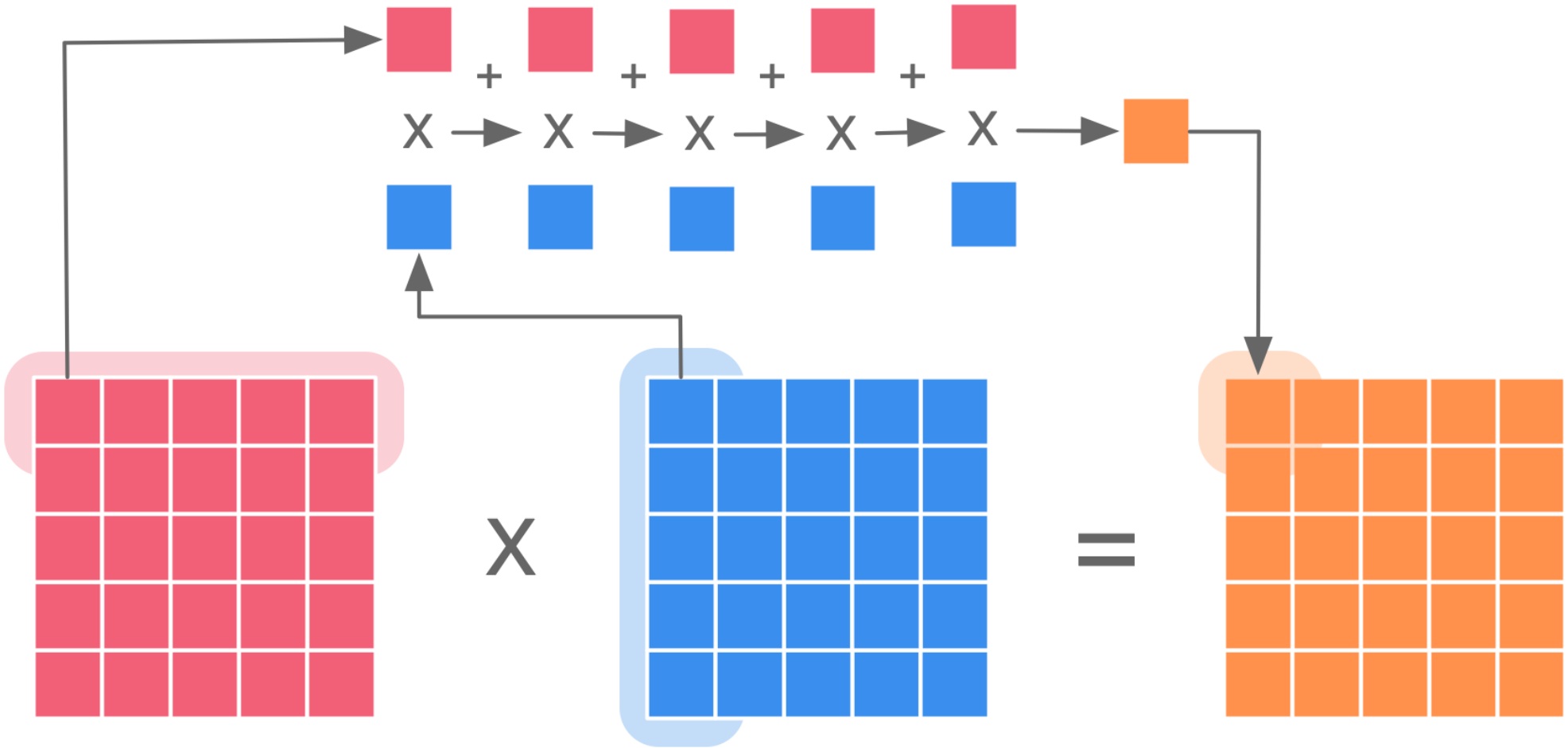


AI计算模式与线程关系

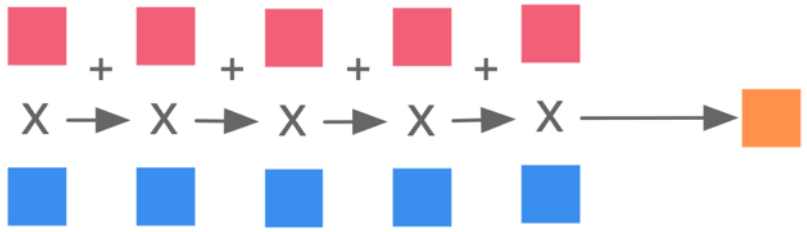
矩阵乘 MM 计算



矩阵乘 MM 计算



计算强度与矩阵乘的关系



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

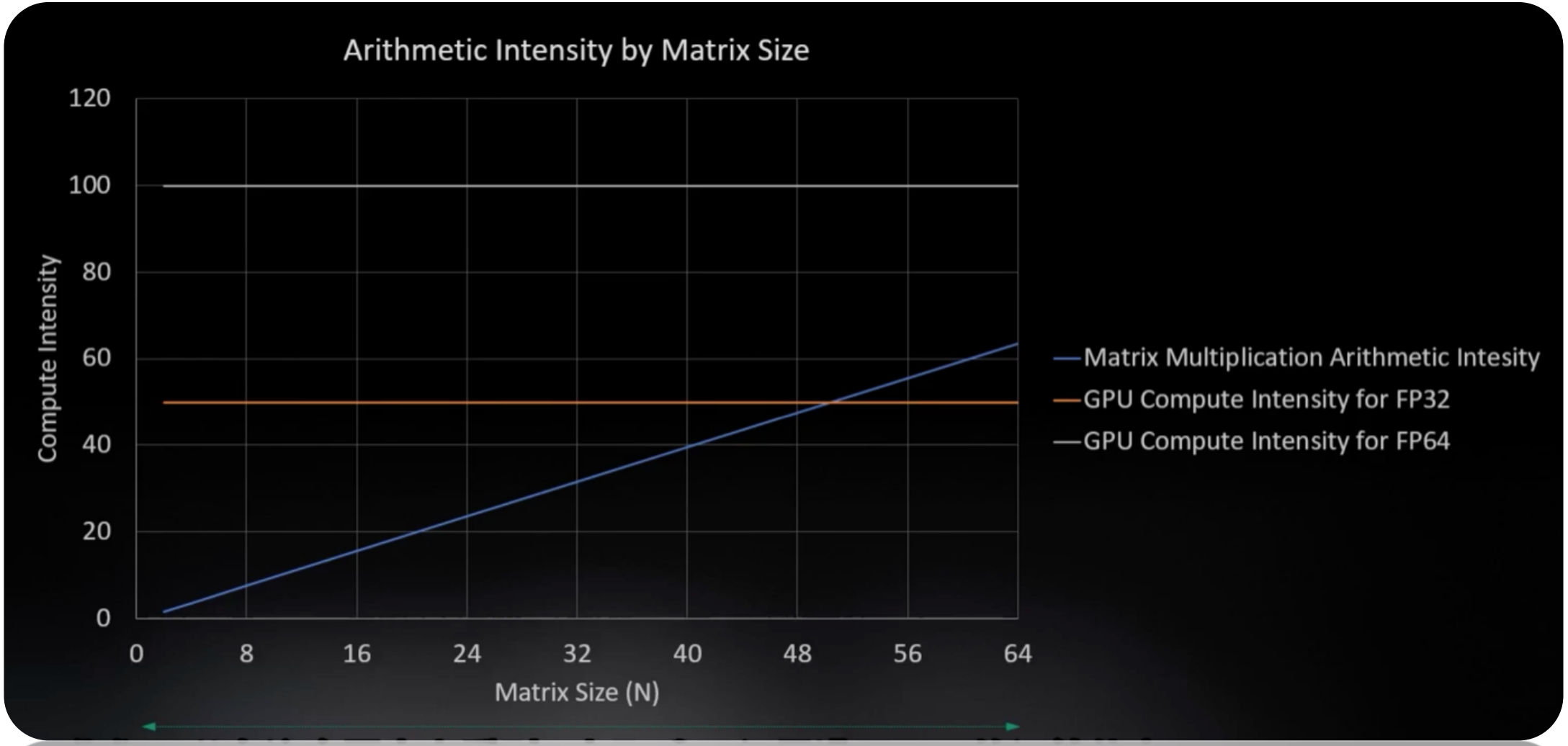
Arithmetic complexity is therefore: $N^2 \times (2N) = 2N^3 = N$

Number of data loads: $O(N)$

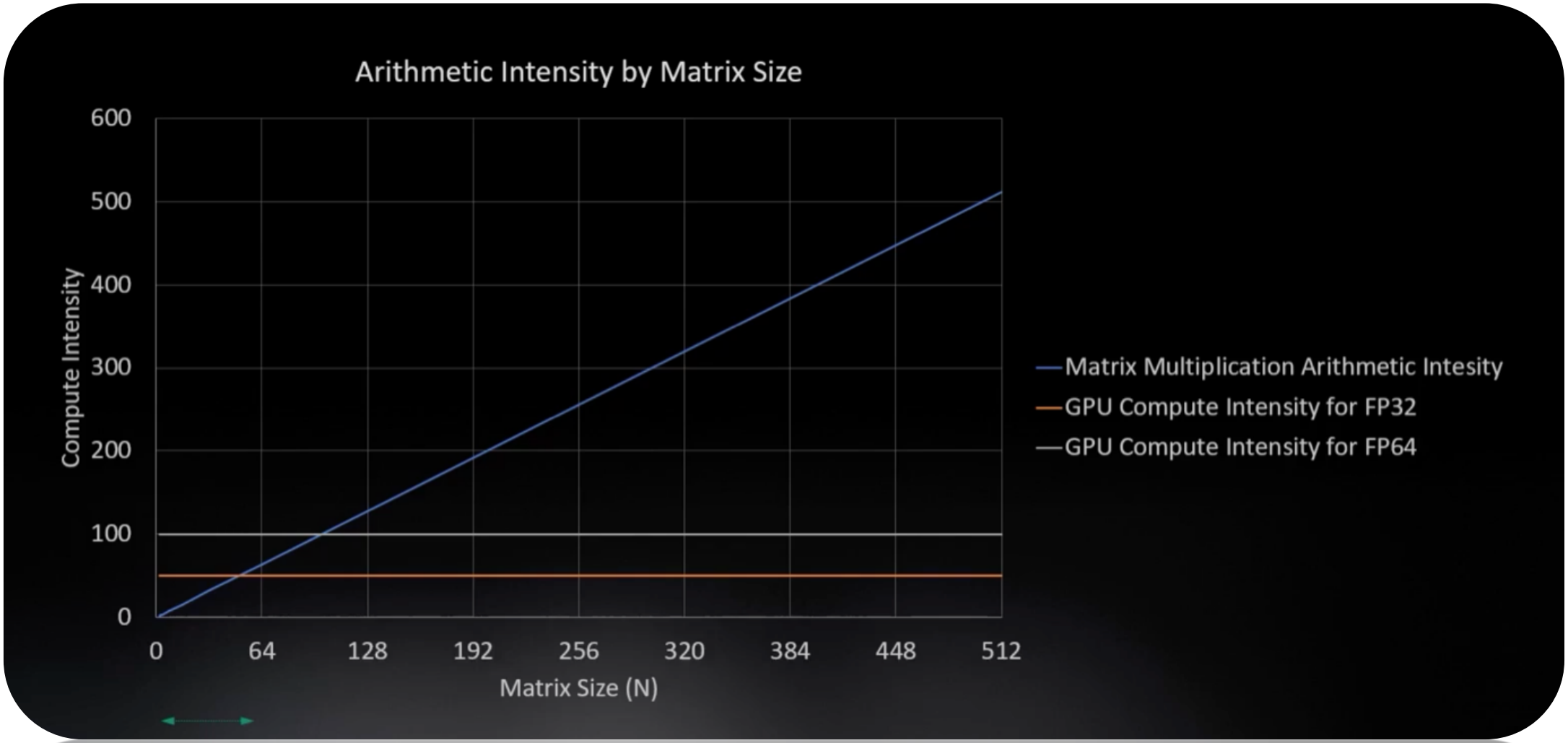
Arithmetic intensity scales as: $N^3 / N^2 = O(N)$

算术强度

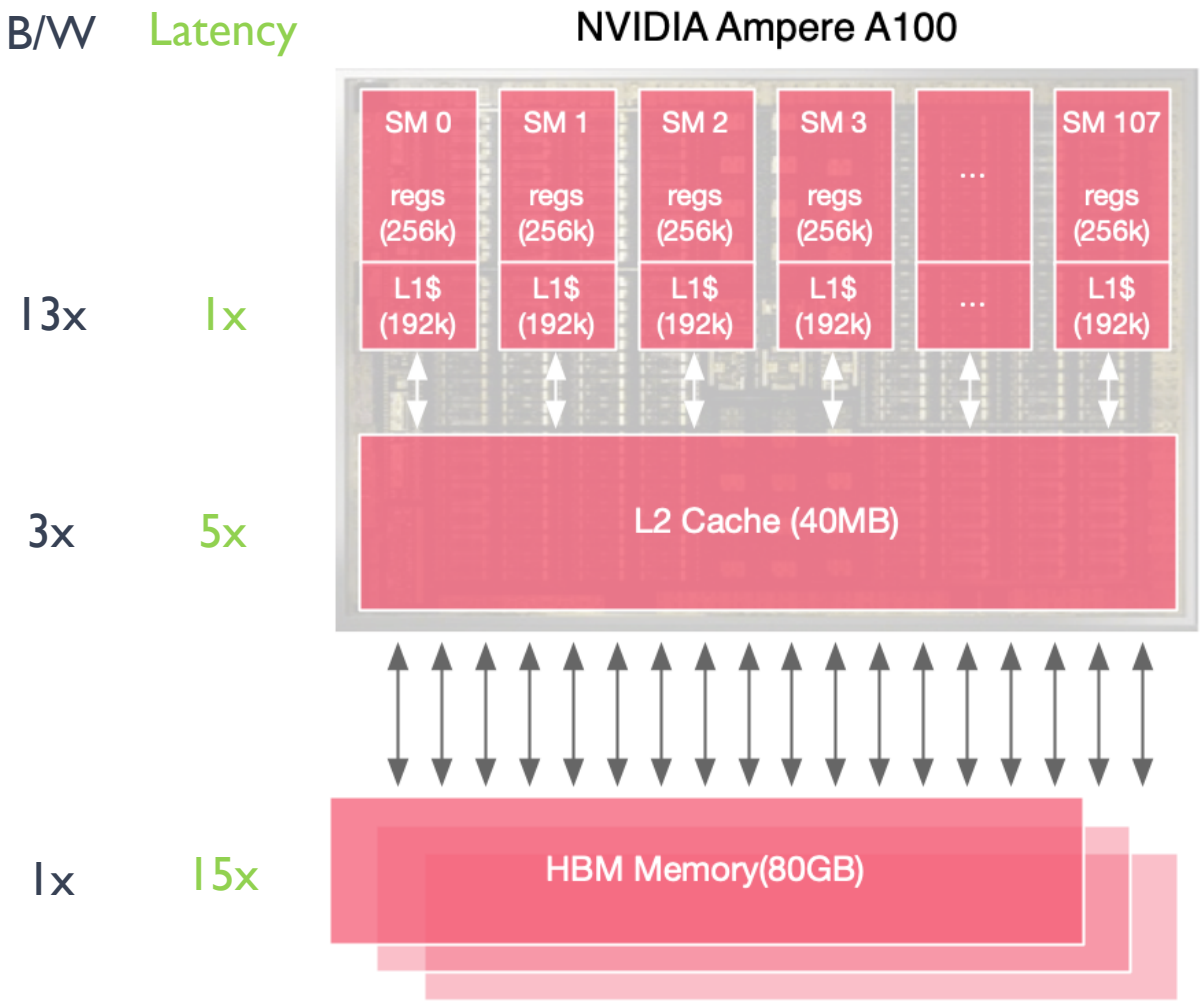
矩阵乘 MM 计算利用率



矩阵乘 MM 计算利用率

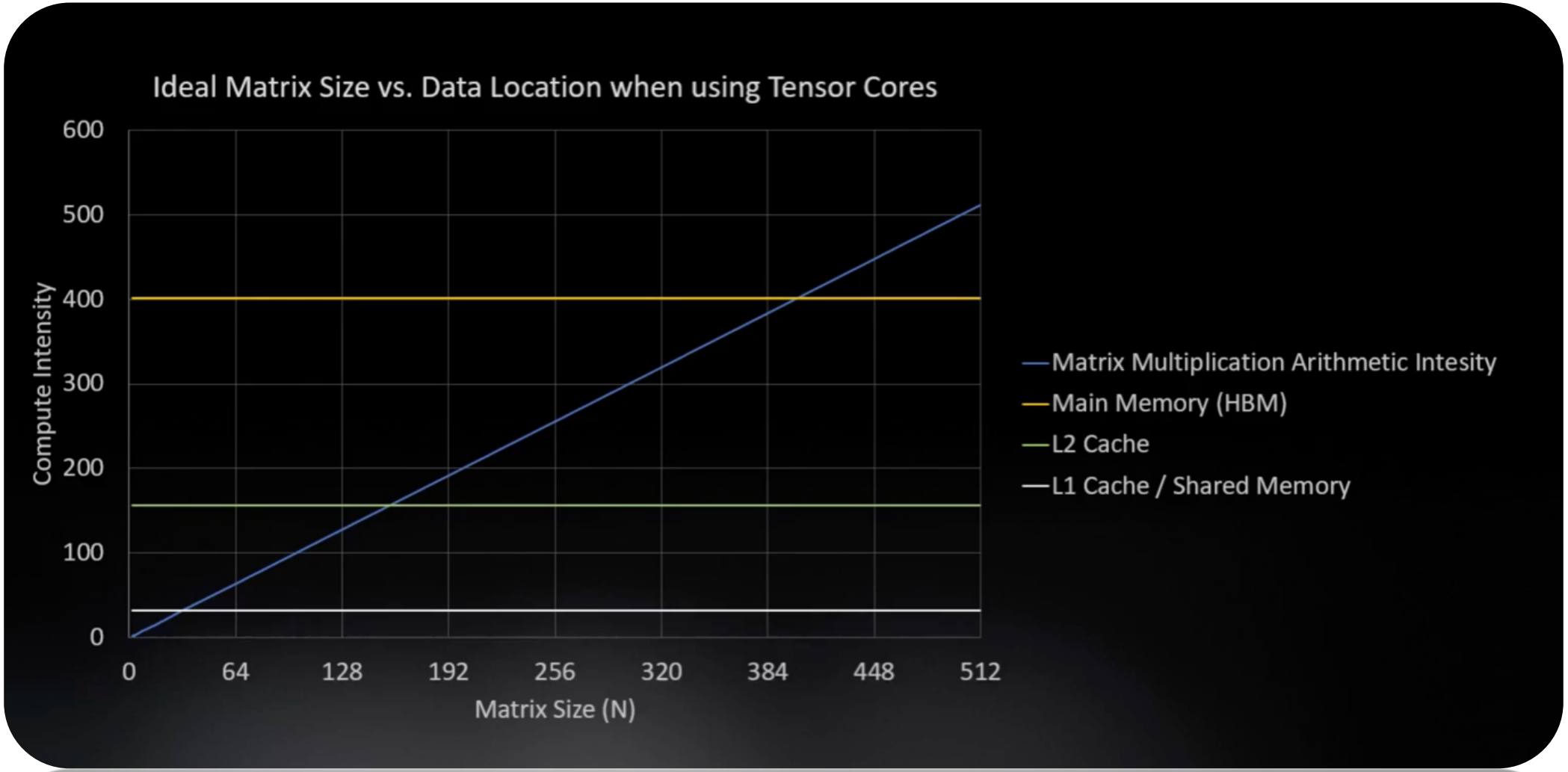


小矩阵计算



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

小矩阵计算



总结：为什么GPU适用于AI计算

Flops

你真的在乎算力？

当我们对计算本身有更深入的了解时候，

才会慢慢看到本质的问题：我的数据在哪里？

更应该关注

内存、带宽、时延

读取数据与计算的计算换算

How many operations must I do on some data to make it worth the cost of loading it?

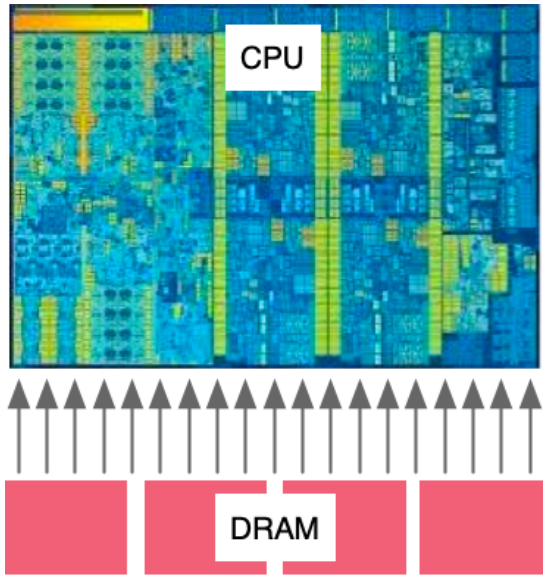
Required Compute Intensity = $\frac{FLOPs}{Data\ Rate}$ = 80

2000 GFLOPs FP64

200 GBytes / sec

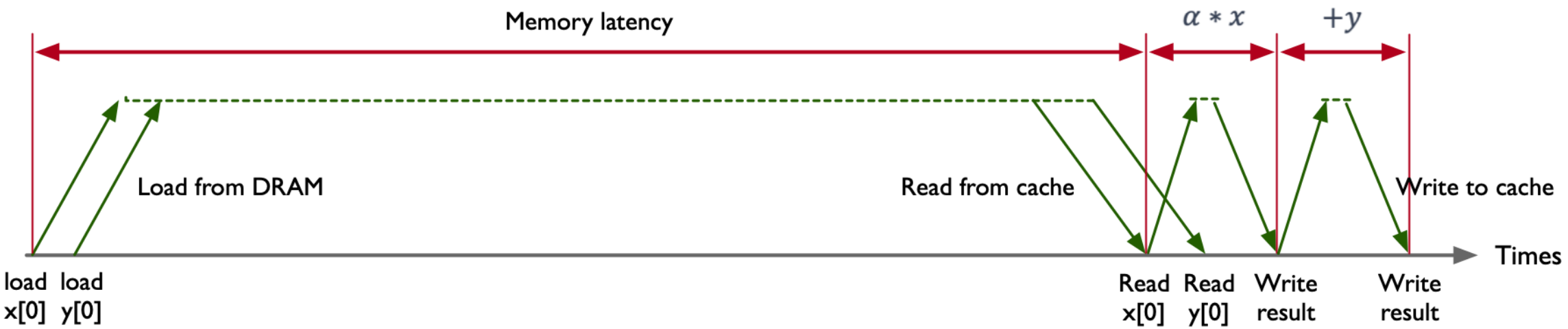
= 25 Giga-FP64 / sec

(FP64 = 8 bytes)



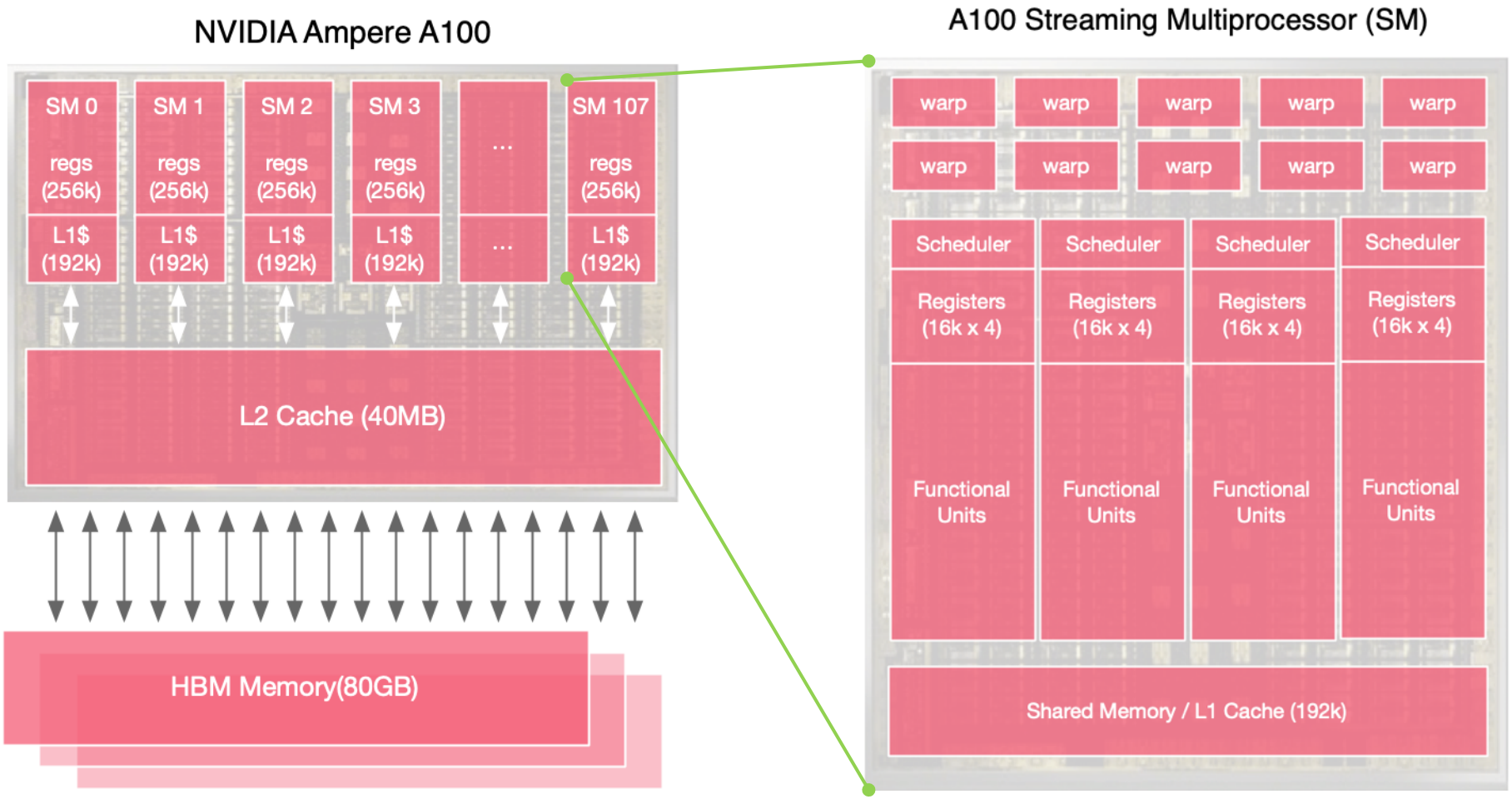
So for every number load from memory, Need to do 80 Operations on it to break even.

AX+Y 计算 DEMO



memory bus is idle 99.86% of the time

GPU 线程机制



64 warps/SM

4x concurrent warp exec

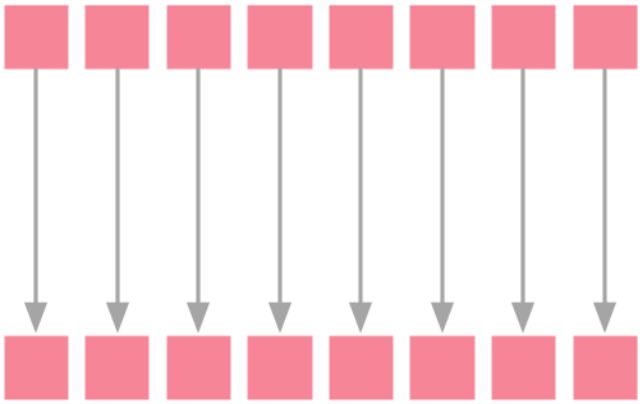
64k 4-byte registers

192KB L1/shared memory (configurable split)

GPU runs threads in groups of 32 – each group is known as a warp

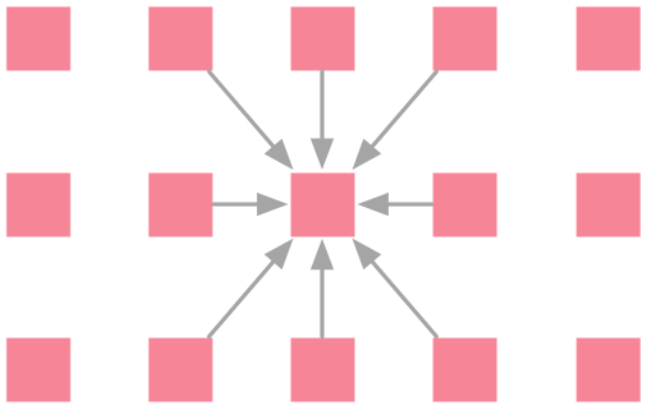
AI 计算模式与线程的关系

在AI计算模式中，不是所有计算都可以是线程独立的哦



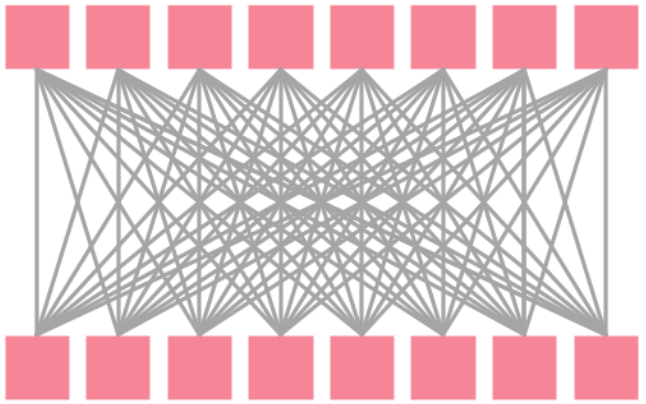
Element-wise

AX+Y



Local

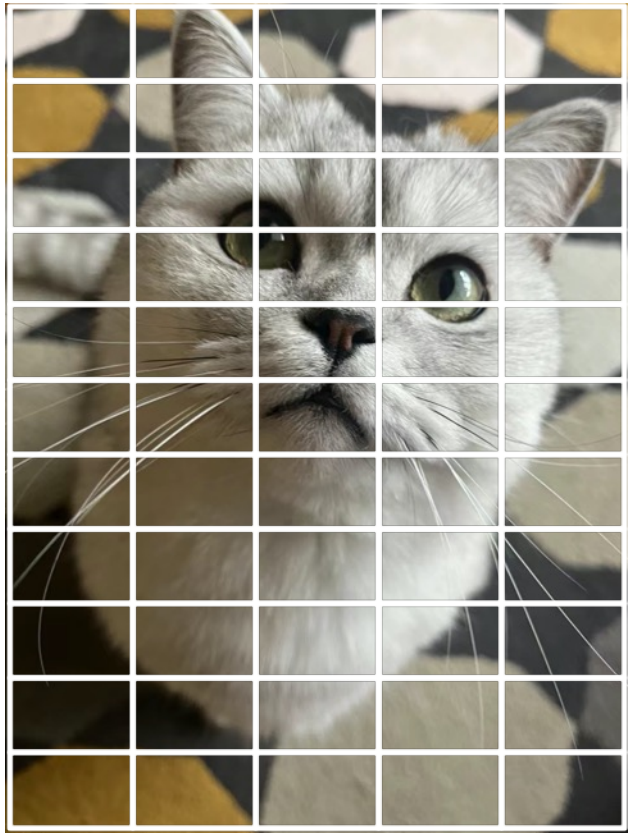
Convolution



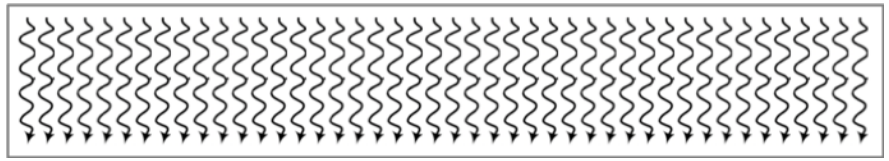
All to All

MatMul

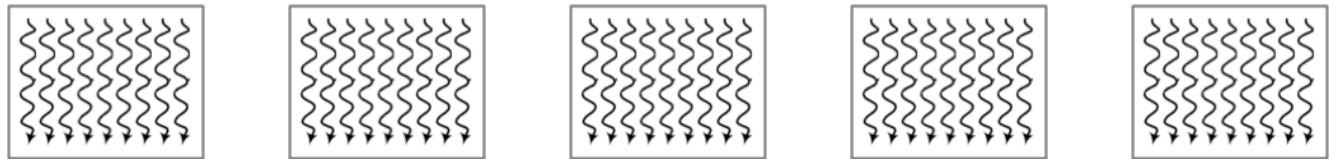
线程分层执行



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

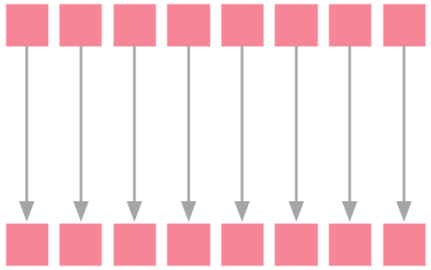
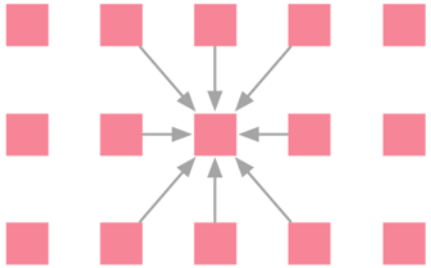
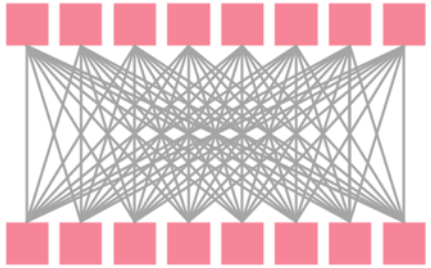


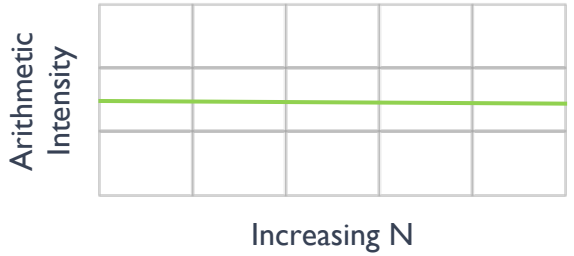
块 Block 中的线程数独立执行

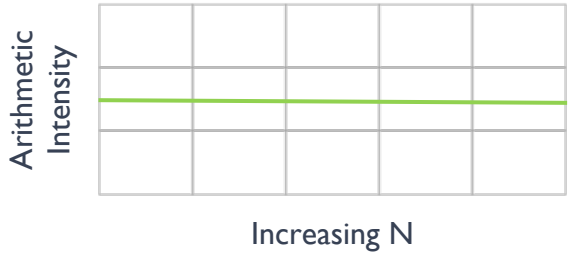
可以通过本地数据共享

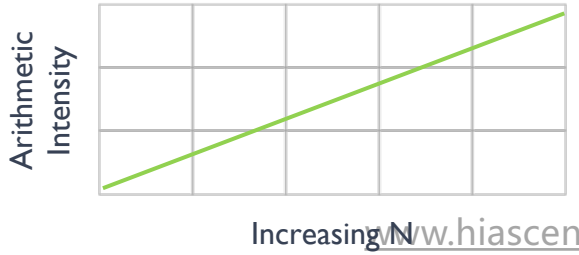
同步交换数据

并行提升整体计算强度

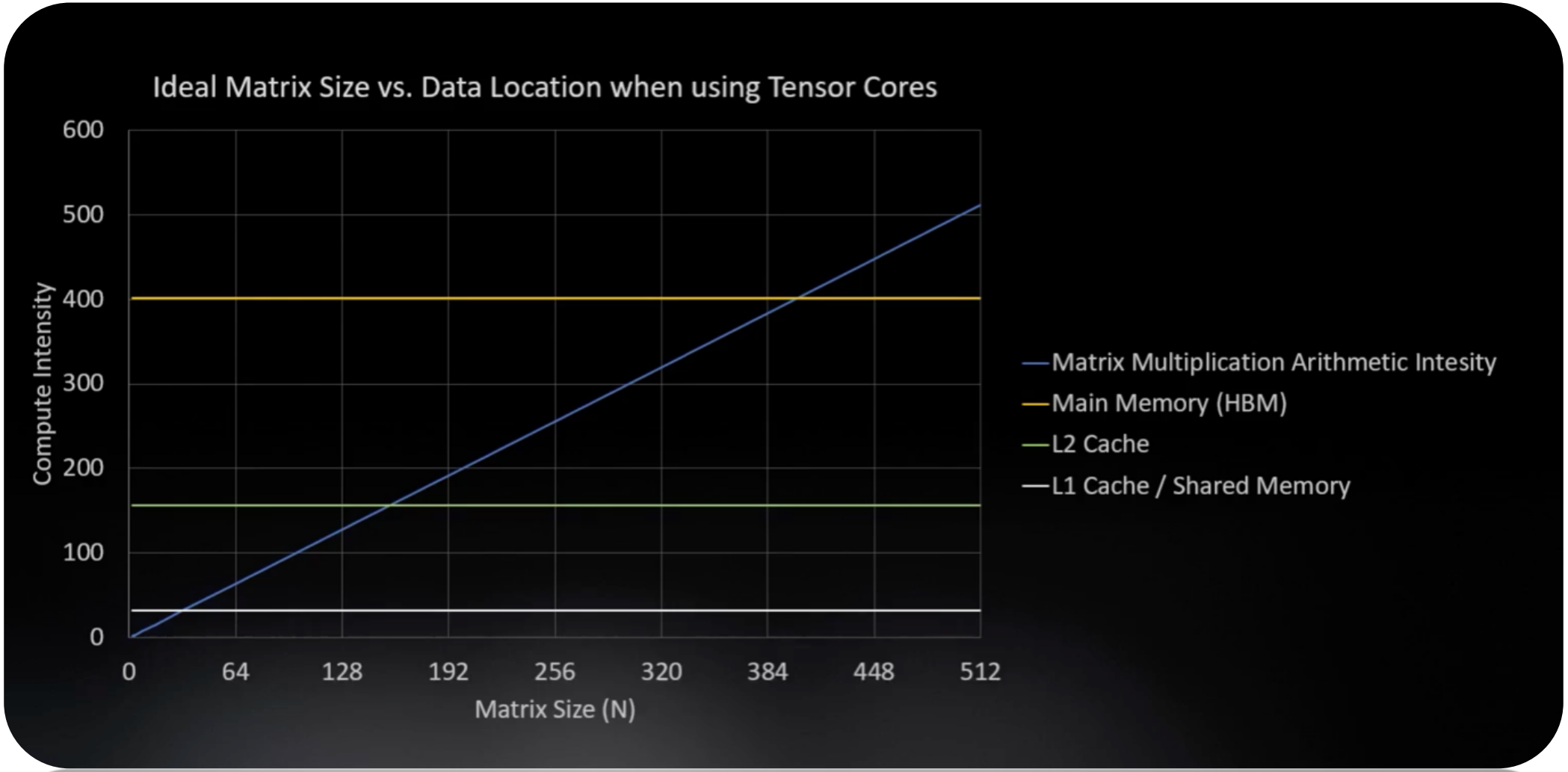
		Data Scales	Compute Scales	Intensity Scales
Element-wise		$O(N)$	$O(N)$	$O(1)$
Local		$O(N^2)$	$O(N^2)$	$O(1)$
All to All		$O(N)$	$O(N^2)$	$O(N)$







小矩阵计算



Why GPU

- 为什么 GPU 适用于 AI 计算？
- 为什么 AI 训练使用 GPU 而不是 CPU？

- 通过超配的线程 Threads 来掩盖时延，多级的缓存平衡计算和带宽的GAP，提出 Tensor Core来增加峰值算力

Reference

- https://www.techtarget.com/searchvirtualdesktop/definition/GPU-graphics-processing-unit
- https://www.techtarget.com/searchvirtualdesktop/definition/GPU-graphics-processing-unit#:~:text=GPUs%20work%20by%20using%20a,%2C%20high%2Dquality%20graphics%20rendering.
- https://computer.howstuffworks.com/graphics-card.htm
- https://www.cs.cmu.edu/afs/cs/academic/class/15462-f11/www/lec_slides/lec19.pdf
- https://www.intel.com/content/www/us/en/products/docs/processors/what-is-a-gpu.html
- https://www.investopedia.com/terms/g/graphics-processing-unit-gpu.asp
- https://www.youtube.com/watch?v=0_TN845dxUU
- https://www.youtube.com/watch?v=nEMzIwzmJT8
- https://www.heavy.ai/technical-glossary/cpu-vs-gpu
- https://www.youtube.com/watch?v=3110o0DYJXg
- https://www.youtube.com/watch?v=658n_Ym8dkk
- https://www.youtube.com/watch?v=5BiAlaFGCoE
- https://www.youtube.com/watch?v=bZdxcHEM-uc