



GPU Architecture for AI

Compute infrastructure | Streaming multiprocessors, warps, tensor cores, HBM and the cache hierarchy, on NVIDIA Hopper as the reference; roofline model and arithmetic intensity to tell memory-bound from compute-bound work

#GPU #Hopper #TensorCores #HBM #Roofline

A GPU in plain words

A large factory		An NVIDIA GPU
132 identical workshops	≈	132 streaming multiprocessors (H100 SXM)
Teams of 32 workers who do the same move together	≈	Warps of 32 threads (SIMT)
A specialised press that stamps a whole block per hit	≈	Tensor cores: a matrix tile per instruction
Each worker's own bench	≈	Registers
The workshop's shelf, shared by its teams	≈	Shared memory / L1 (per SM)
The central warehouse on site	≈	L2 cache (50 MB)
The trucks to the outside depot	≈	HBM, 3.35 TB/s on H100

WHERE THE ANALOGY BREAKS Factory workers wait idle when a truck is late; a GPU switches instantly to another ready warp, which is why it needs thousands of threads in flight to hide memory latency.

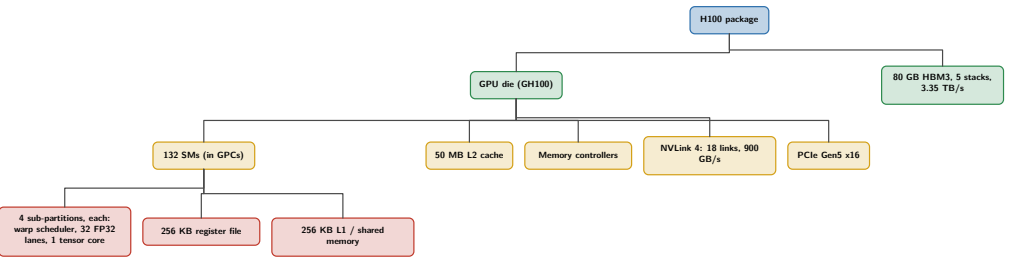
A GPU as a factory of identical workshops. Keep it in mind for the hierarchy below

Before GPU architecture, can you explain...

	Not yet	Partly	Yes
MUST KNOW Matrix multiplication and its cost 2-m-n-k FLOPs Learn: maths: matrices	☐	☐	☐
MUST KNOW Bytes per number: fp32 4, bf16 2, fp8 1 Learn: maths: numerical computing	☐	☐	☐
MUST KNOW Units: FLOP vs FLOP/s, GB/s Learn: this folder: 00-start-here	☐	☐	☐
HELPFUL CUDA programming model (grid, block, thread) Learn: programming: GPU programming basics	☒	☒	☒

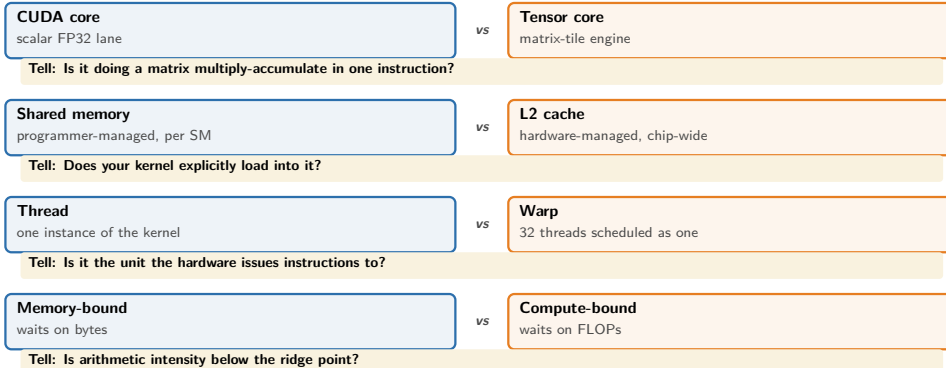
Check before reading

Inside the chip (NVIDIA H100 SXM as reference)



From package to lane. Numbers are for H100 SXM5 (Hopper, 2022); other generations differ in counts, not in the structure

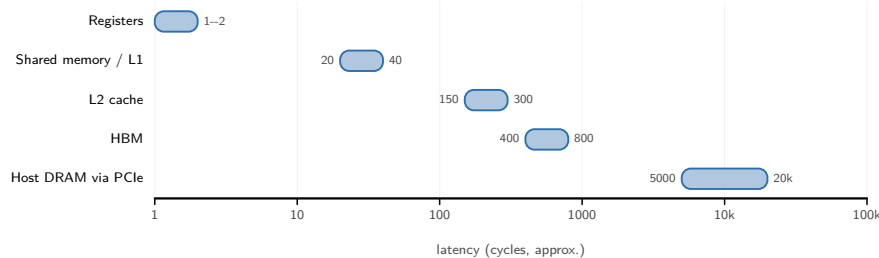
Part	H100 SXM figure	What it means for AI
SMs	132	work is split into thread blocks spread over SMs; too few blocks leave SMs idle
FP32 lanes (CUDA cores)	128 per SM, 16,896 total	element-wise ops, softmax, norms; minor share of AI FLOPs
Tensor cores (4th gen)	4 per SM, 528 total	matrix multiplies in FP64, TF32, BF16, FP16, FP8, INT8
Dense BF16 tensor peak	approx. 989 TFLOP/s	the number for planning; FP8 is approx. 2x
Registers	64K x 32-bit per SM	fastest storage; spills to memory kill speed
L1 / shared memory	256 KB per SM (up to 228 KB shared)	holds tiles reused by a thread block
L2 cache	50 MB	shared by all SMs; KV cache and weights do not fit
HBM3	80 GB at 3.35 TB/s	capacity = what fits; bandwidth = token speed
Max threads per SM	2,048 (64 warps)	occupancy target: enough warps to hide latency
Power (TDP)	up to 700 W	drives rack density and cooling (sheet 05)



Pairs that are often confused in GPU discussions

- **Hopper additions** FP8 tensor cores with the Transformer Engine; TMA (Tensor Memory Accelerator) copies tiles between HBM and shared memory asynchronously; thread-block clusters share memory across neighbouring SMs
- **Blackwell (B200, 2024-25)** two dies in one package joined by a 10 TB/s die-to-die link, acting as one GPU; FP4 and FP6 tensor formats with fine-grained (micro-block) scaling; NVLink 5 at 1.8 TB/s
- **Precision matters twice** halving bits doubles tensor-core throughput and halves bytes moved; it is the cheapest speed-up when accuracy allows

Memory hierarchy and latency



Approximate access latency per level in GPU clock cycles (log scale). Orders of magnitude, they vary by generation and access pattern

Each step down is roughly 10× slower and 10-1,000× larger. Fast kernels (FlashAttention, fused GEMMs) load a tile from HBM once into shared memory and registers and reuse it many times: that raises arithmetic intensity, the subject of the next section. **Latency is hidden, bandwidth is not:** with enough warps, a GPU overlaps waits, but it can never move more bytes per second than HBM allows.

Roofline and arithmetic intensity

Roofline: attainable speed of a kernel

FORMULA $P = \min(P_{\text{peak}}, I \times BW)$

WHERE

- P = attainable speed of the kernel (TFLOP/s)
- P_{peak} = peak compute of the GPU at that precision (dense) (TFLOP/s)
- I = arithmetic intensity = FLOPs ÷ bytes moved to and from HBM (FLOP/byte)
- BW = HBM bandwidth (TB/s)

WORKED

$P_{\text{peak}} = 989$
 $I = 64$
 $BW = 3.35$
 $P = \min(989, 64 \times 3.35)$
 $P = 214.4$ TFLOP/s

RESULT

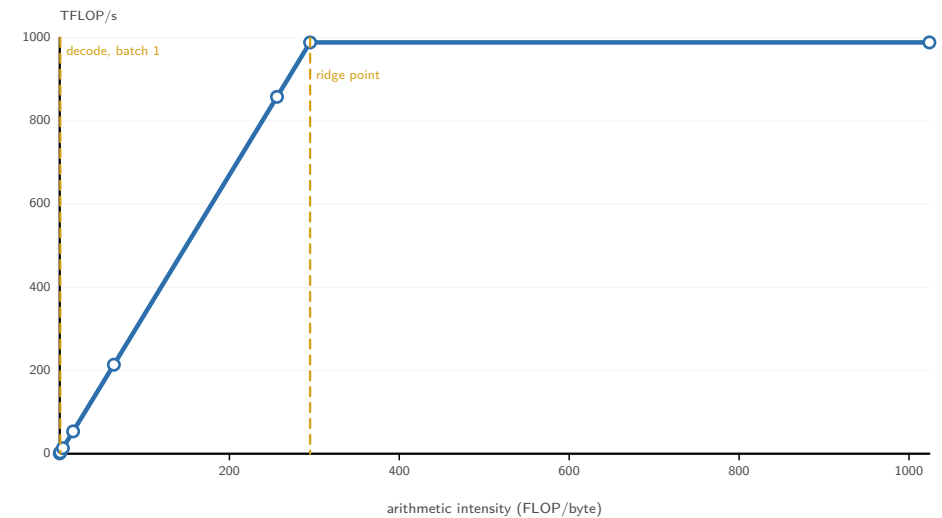
$P = 214.4$ TFLOP/s

READ IT

With $I = 64$ FLOP/byte an H100 reaches at most 214 TFLOP/s in BF16: memory-bound, 22 % of peak. The ridge point is $989 \div 3.35 \approx 295$ FLOP/byte.

MISTAKES

- Use dense peaks, not sparsity peaks
- Count bytes actually moved from HBM, not bytes touched in cache
- TB/s × FLOP/byte gives TFLOP/s: keep units consistent



The H100 BF16 roofline: attainable TFLOP/s = $\min(989, I \times 3.35)$ at chosen intensities. Everything left of the ridge (approx. 295 FLOP/byte) is memory-bound

Operation (BF16)	FLOPs	Bytes from HBM	Intensity	Bound by
Element-wise (ReLU, add) on n values	n	$4n$ (read $2n$ + write $2n$)	0.25	memory
Matrix-vector ($d \times d$ weights $\times$ vector)	$2d^2$	$\approx 2d^2$ (weights)	≈ 1	memory
Same weights, batch of B vectors	$2Bd^2$	$\approx 2d^2$	$\approx B$	memory if $B < 295$
Square matmul $n \times n \times n$	$2n^3$	$6n^2$ (A, B, C)	$n \div 3$	compute if $n > 900$
Softmax, LayerNorm (unfused)	few per value	several passes	< 1	memory: fuse them

PROBLEM

Generating one token multiplies each weight matrix by one vector per sequence. At what batch size does decoding stop being memory-bound on an H100?

GIVEN

- Weight matrix of $d \times d$ in BF16: bytes = $2d^2$, FLOPs per sequence = $2d^2$ (one multiply and one add per weight)
- B = batch size = sequences decoded together; weights are read once per step whatever B
- I = FLOPs $\div$ bytes; ridge point $P_{\text{peak}} \div \text{BW} = 989 \div 3.35 \approx 295$ FLOP/byte

1

Intensity at batch 1

$I = 2d^2 \div 2d^2 = 1$ FLOP/byte

Why: each weight is used once after being read

2

Attainable speed at batch 1

$\min(989, 1 \times 3.35) = 3.35$ TFLOP/s

Why: 0.3 % of peak: the tensor cores mostly wait

3

Intensity at batch B

$I = 2Bd^2 \div 2d^2 = B$ FLOP/byte

Why: the same weights serve B sequences

4

Batch to reach the ridge

$B \approx 295$

Why: above it decoding becomes compute-bound (KV-cache reads ignored here)

ANSWER

Batch 1 decoding runs at well under 1 % of peak; batching hundreds of sequences moves it towards the compute roof

CHECK

✓ This is why serving engines batch requests continuously and why quantised weights (fewer bytes) speed up decoding

Why LLM decoding is memory-bound and why batching fixes it

MYTH

✗ More CUDA cores is what makes a GPU fast for AI

→

✓ Tensor cores and HBM bandwidth dominate

Matrix multiplies run on tensor cores; most other work is limited by bandwidth.

✗ A kernel at 30 % of peak FLOP/s is badly written

→

✓ It may be at its memory roof

Compare with $\min(\text{peak}, I \times \text{BW})$ before optimising; then raise I by fusing or tiling.

✗ The L2 cache holds the model

→

✓ 50 MB is tiny next to GBs of weights

Weights and KV cache stream from HBM every step.

✗ Warps divergence does not matter on GPUs

→

✓ Divergent branches run serially

Keep branches uniform within a warp; tensor-core kernels avoid branching altogether.

Remember

01

SM $\rightarrow$ warp of 32 $\rightarrow$ tensor cores

the hierarchy every NVIDIA GPU shares

02

HBM bandwidth sets the memory roof

3.35 TB/s on H100

03

Ridge point = peak $\div$ bandwidth

approx. 295 FLOP/byte for H100 BF16

04

Decode at batch 1 has intensity ≈ 1

memory-bound; batch to fix

05

Fewer bits, more speed

FP8 / FP4 double throughput and halve bytes

Q1 How many threads are in a warp?
A 32.

Q2 SMs, L2 and HBM of an H100 SXM?
A 132 SMs, 50 MB L2, 80 GB HBM3 at 3.35 TB/s.

Q3 Write the roofline formula.
A $P = \min(P_{\text{peak}}, I \times \text{BW})$, with I = FLOPs per byte moved from HBM.

Tadata | tadata.fr | contact@tadata.codes

Q4 Ridge point of H100 in BF16?

A $989 \text{ TFLOP/s} \div 3.35 \text{ TB/s} \approx 295 \text{ FLOP/byte}$.

Q5 Arithmetic intensity of an element-wise op in BF16?

A About 0.25 FLOP/byte: 1 FLOP per 2 bytes read and 2 written.

Q6 Intensity of an $n \times n \times n$ matmul in BF16?

A $2n^3 \div 6n^2 = n/3 \text{ FLOP/byte}$.

Q7 What is warp divergence?

A Threads in a warp taking different branches, executed one after the other.

Q8 What did Blackwell add to tensor cores?

A FP4 and FP6 formats with micro-block scaling, in a two-die package.

Q9 Why do fused kernels like FlashAttention help?

A They reuse tiles in shared memory, cutting HBM traffic and raising arithmetic intensity.