FlashAttention & FlashAttention-3: IO-Aware Exact Attention from Algorithm to Hardware

Ziyang Tan

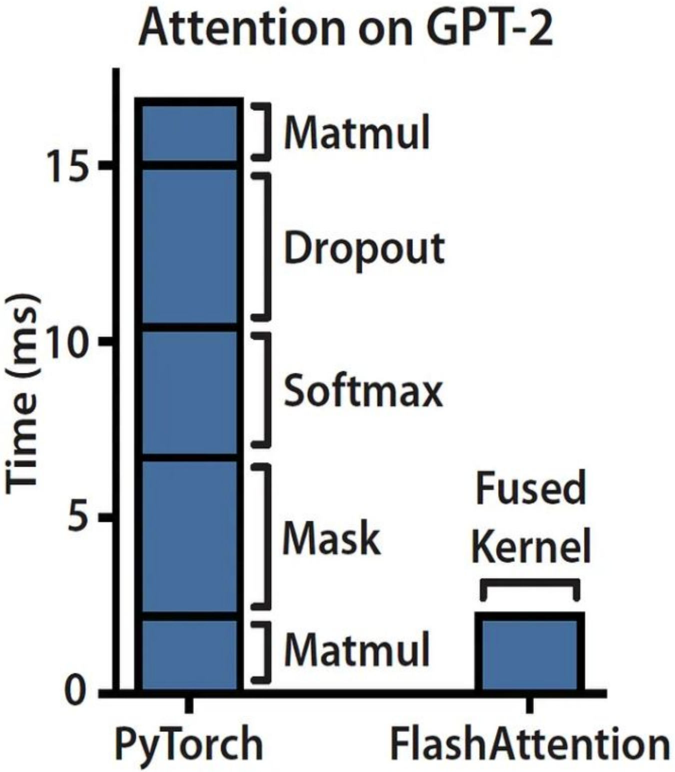
Outline

- ➢ Background: why standard attention is slow
- ➢ FA-1: idea → math → algorithm → IO complexity
- ➢ FA-3: Hopper-optimized pipeline & FP8
- ➢ Takeaways
- ➢ Q&A

Background

Compute-Bound vs Memory-Bound

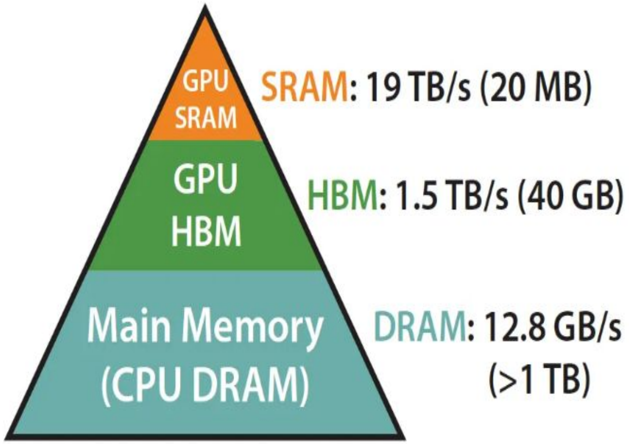
- Arithmetic Intensity: FLOPs per byte.
- Machine Balance: Peak FLOPs / Peak BW.
- **Memory-bound:** intensity < balance
  - Matmul (GEMM): high reuse ⇒ high intensity.
- **Compute-bound:** intensity > balance
  - Elementwise/Reductions (mask, softmax...):
  - low reuse ⇒ low intensity ⇒ memory-bound.



Dropout, softmax, masking — all elementwise ops, all memory bound. We can see they dominate the runtime.

Memory Hierarchy: Why IO-Aware Attention Wins

- Attention mixes memory-bound ops (mask, softmax).
- **Wall-clock time**:
  - Compute time + IO time (bottleneck)
  - dominated by HBM⇄SRAM traffic.
- Goal: minimize HBM round-trips, keep work on chip.



Memory Hierarchy with
Bandwidth & Memory Size

Standard Attention: Not IO-Aware (HBM treated as “free”)

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Step 1 — compute $S = QK^\top$

- **Reads:** Q Nd + K Nd → $2Nd$
- **Writes:** S N^2

Step 2 — compute $P = \mathrm{softmax}(S)$

- **Reads:** S N^2
- **Writes:** P N^2

Step 3 — compute $O = PV$

- **Reads:** P N^2 + V Nd → $N^2 + Nd$
- **Writes:** O Nd

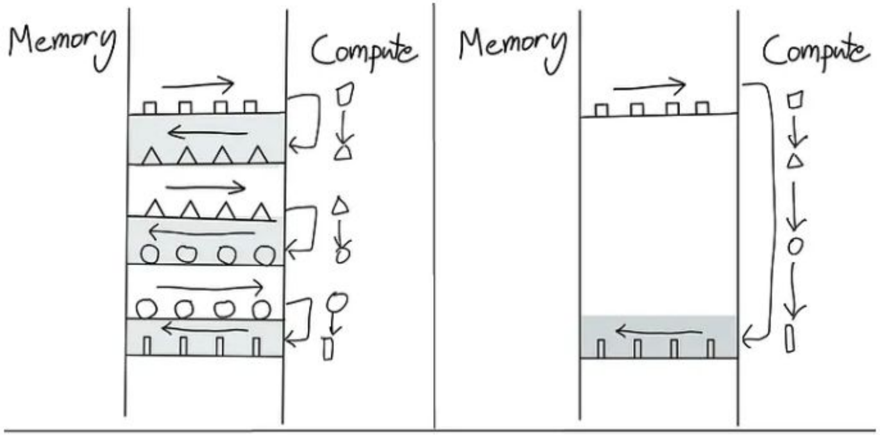
Totals (dominant terms):

- **Reads:** $3Nd + 2N^2$
- **Writes:** $Nd + 2N^2$
- **Grand HBM traffic:** $4N^2 + 4Nd$ elements → $O(N^2)$ dominated by materializing S and P.

FlashAttention-1

Operator Fusion & Materialization: Becoming IO-Aware

- **Kernel**: one GPU op, HBM load → on-chip compute → HBM store
- **Kernel Fusion**: combine multiple ops into one kernel
- **Materialization**: writing large intermediate tensors to HBM, then reading back later



Operator Fusion Simplified

https://horace.io/brrr_intro.html

FlashAttention: Tile.

- **Softmax**: compute the i-th output of a softmax

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- **Tiling**: compute attention by blocks

For numerical stability, the softmax of vector $x \in \mathbb{R}^B$ is computed as:

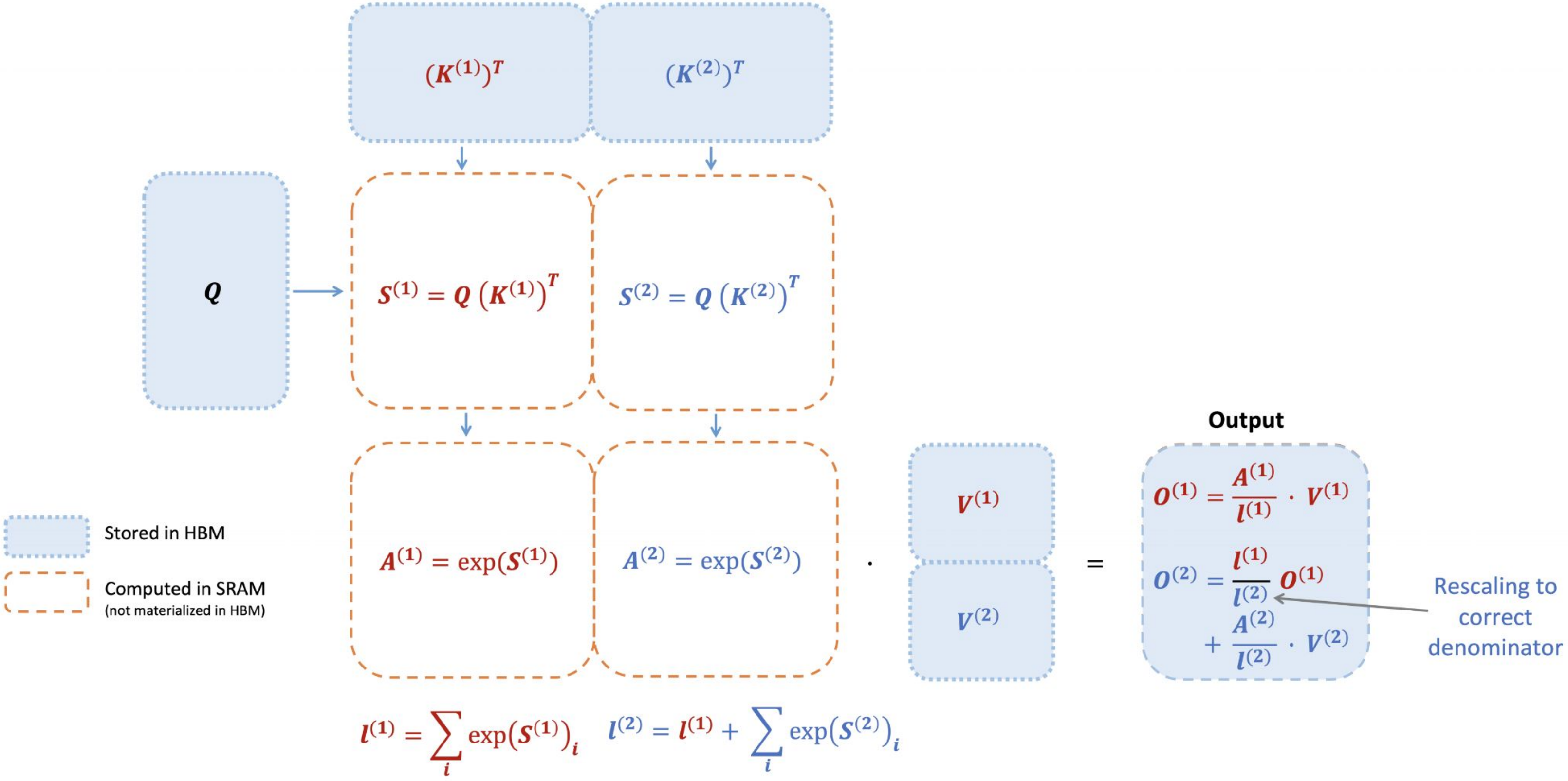
$$m(x) := \max_i \ x_i, \quad f(x) := \left[e^{x_1 - m(x)} \quad \ldots \quad e^{x_B - m(x)} \right], \quad \ell(x) := \sum_i f(x)_i, \quad \operatorname{softmax}(x) := \frac{f(x)}{\ell(x)}.$$

For vectors $x^{(1)}, x^{(2)} \in \mathbb{R}^B$, we can decompose the softmax of the concatenated $x = \left[x^{(1)} \ x^{(2)}\right] \in \mathbb{R}^{2B}$ as:

$$m(x) = m(\left[x^{(1)} \ x^{(2)}\right]) = \max(m(x^{(1)}), m(x^{(2)})), \quad f(x) = \left[e^{m(x^{(1)}) - m(x)} f(x^{(1)}) \quad e^{m(x^{(2)}) - m(x)} f(x^{(2)}) \right],$$

$$\ell(x) = \ell(\left[x^{(1)} \ x^{(2)}\right]) = e^{m(x^{(1)}) - m(x)} \ell(x^{(1)}) + e^{m(x^{(2)}) - m(x)} \ell(x^{(2)}), \quad \operatorname{softmax}(x) = \frac{f(x)}{\ell(x)}.$$

FlashAttention: Tile.



FlashAttention: Algorithm.

Algorithm 1 FlashAttention

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M.

Set block sizes $B_c = \lceil \frac{M}{4d} \rceil$, $B_r = \min\left(\lceil \frac{M}{4d} \rceil, d\right)$.

Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}$, $\ell = (0)_N \in \mathbb{R}^N$, $m = (-\infty)_N \in \mathbb{R}^N$ in HBM.

Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.

Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \dots, m_{T_r}$ of size B_r each.

for $1 \le j \le T_c$ **do**

  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.

  for $1 \le i \le T_r$ **do**

    Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.

    On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.

    On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.

    Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.

    Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.

  end for

end for

Return $\mathbf{O}$.

FlashAttention: recomputation

- **Activation/gradient checkpointing:** don't store all activations in fwd; recompute them in bwd → memory ↓, time ↑. Can choose the granularity (store every n layers) to trade memory for recompute.
- **FlashAttention's Twist:** recomputation adds FLOPs, but slashes HBM traffic
  - Goal: avoid storing $O(N^2)$ intermediates $S, P \in \mathbb{R}^{N\times N}$.
  - **Store only:**
    - Output $O \in \mathbb{R}^{N\times d}$
    - Per-row softmax stats $(m, \ell) \in \mathbb{R}^{N}$ *(row max & sum of exps)*
    - (Training) RNG state for dropout
  - **Recompute in backward:** From tiled $Q, K, V \in \mathbb{R}^{N\times d}$ kept **on-chip**, rebuild local S and P per tile using (m, ℓ), then compute dV, dQ, dK.
  - **Memory complexity:** $O(N)$ instead of $O(N^2)$.

FlashAttention: Complexity analysis

- **Setup / Notation:**
  - Sequence length N, head dim d, usable on-chip SRAM M (elements).
  - Tile sizes chosen to fit SRAM:
    $B_c = \frac{M}{4d}$ (**column tile** for K, V), $B_r = \min\left(\frac{M}{4d}, d\right)$ (**row tile** for Q).
  - Number of tiles: $T_c = \frac{N}{B_c} = \frac{4Nd}{M}, T_r = \frac{N}{B_r}$.
  - We count **HBM elements moved** (multiply by bytes/elt for traffic).
    Standard attention IO (for reference): $\Theta(Nd + N^2)$.

FlashAttention: Complexity analysis

One **sweep** = fix a K, V column tile j in SRAM and iterate over all Q row tiles i.

Per sweep HBM traffic:

1. **Load K_j, V_j once**: size $= (B_c d + B_c d) = 2B_c d = \frac{M}{2}$ → across all sweeps sums to $\Theta(Nd)$ (lower order).
2. Iterate all Q rows (total Nd elements):
   - **Read** Q: Nd
   - **Read & write** O partials: $2Nd$
   - **Read & write softmax stats** (m, ℓ): $2N$ (negligible vs Nd)

So, **per sweep** $\approx \Theta(Nd)$ elements (constants ignored).

Number of sweeps $= T_c = \frac{N}{B_c} = \frac{4Nd}{M}$.

$$\boxed{\text{HBM IO from } Q/O \text{ path}} = \Theta(Nd) \times \frac{4Nd}{M} = \Theta\left(\frac{N^2d^2}{M}\right)$$

Add the K/V reads over all sweeps: $\Theta(Nd)$ (lower order).

Final FA HBM IO:

$$\boxed{\Theta\left(\frac{N^2d^2}{M}\right) + \Theta(Nd)} \approx \Theta\left(\frac{N^2d^2}{M}\right)$$

FlashAttention-3

From FA-1/2 to FA-3 — Why change?

- **Utilization Gap:** FA2 achieves only 35–40% GPU utilization on H100, compared to 80–90% on A100 → clear room for improvement.

- **Asynchronous Scheduling:** Hopper allows overlap of Tensor Core GEMM and SFU/Softmax:

- **Low-Precision Support**: FP8 (Hopper), Higher FLOPS, Lower memory & bandwidth usage, Requires careful error control.

- FA3 is more like a **“manual” for exploiting Hopper hardware.**

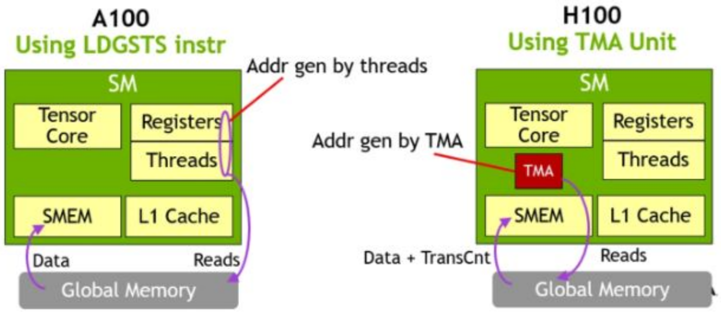
Hopper New Instructions

WGMMA (Warpgroup MMA)

- Async Tensor Core ops at warpgroup (128 threads) granularity
- Overlaps GEMM with CUDA Core/SFU work
- FA3: RS-GEMM (A in GMEM), SS-GEMM (A,B in SMEM)

TMA (Tensor Memory Accelerator)

- Async GMEM ↔ SMEM transfers (1D–5D tensors)
- Handles address, layout, swizzle; frees threads
- Supports multicast to multiple SMs



→ Key enablers of FA3's async compute & efficient data movement

Asynchrony Evolution to FA3

Pre-A100: Warp Specialization

- Producer warps load data, consumer warps compute.
- Warp scheduler hides latency via fast context switch.

A100: Multistage (cp.async)

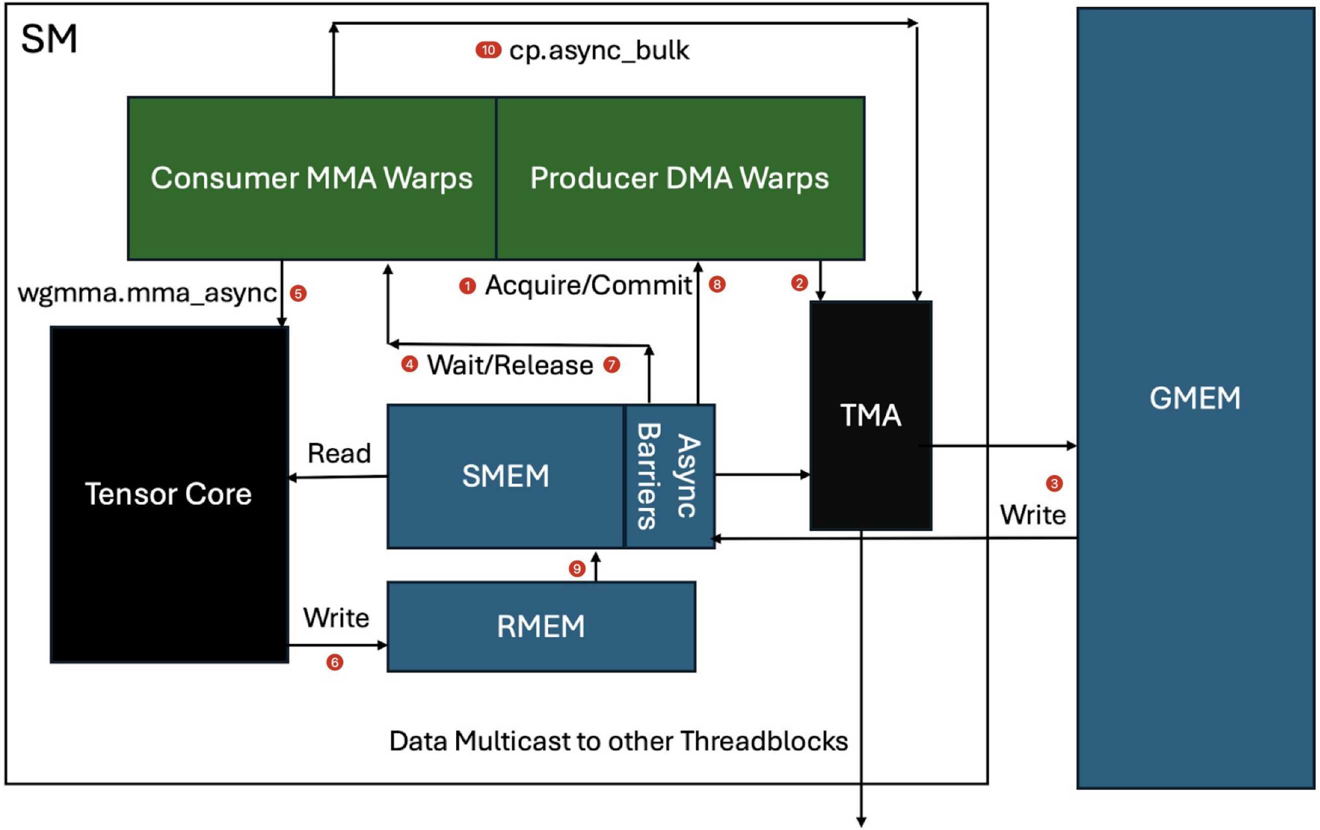
- Same warp overlaps load (N+1) with compute (N).
- Pipeline with double buffer → FA2 implementation.

H100: Warp Specialization + Intra-Warpgroup Overlap

- TMA handles async data movement (no warp overhead).
- WGMMA enables async Tensor Core ops across warpgroups.
- Register reallocation & lightweight producer → maximize compute.
- Overlap GEMM + Softmax inside warpgroups.

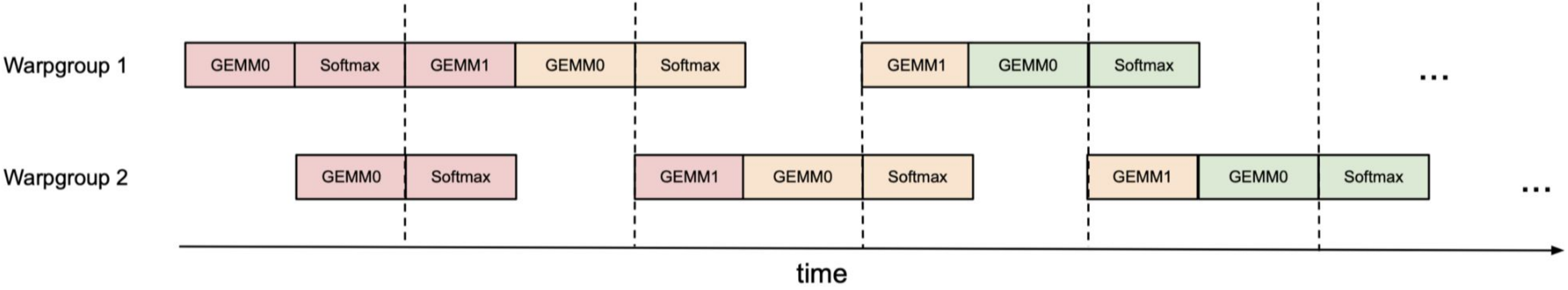
FA3 achieves deeper overlap of compute ↔ comm & compute ↔ compute.

Producer-Consumer Async



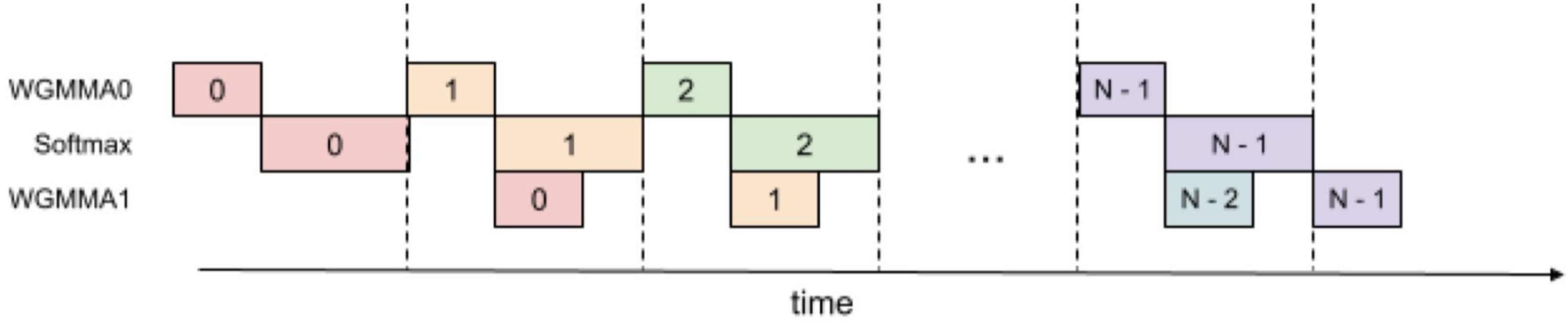
Ping-Pong Scheduling

- Occurs between two consumer warpgroups
- With WGMMA async execution, GEMM and Softmax can run concurrently
- Warpgroups alternate GEMM execution while overlapping with Softmax
- bar.sync at boundaries ensures correct data dependency
- Effect: higher Tensor Core utilization via inter-warpgroup pipelining



Intra-Warpgroup Overlap

- **Problem**: In attention inner loop, softmax depends on GEMM0 output, and GEMM1 depends on softmax result → serialized execution.
- **Idea**: Break dependencies by pipelining across iterations with extra register buffers.
- **Technique**: Overlap part of softmax instructions with subsequent GEMMs (see figure).
- **Effect**: Increases parallelism even within a single warpgroup, reducing stalls and boosting utilization



FP8 in FlashAttention-3

Efficiency: Layout Challenges

- FP8 WGMMA requires k-major layout for V → need transpose.
- Solution: in-kernel transpose using LDSM/STSM (efficient, register-friendly, avoids extra memory ops).
- FP32 accumulator layout ≠ FP8 operand layout → use byte permute + matching transpose to align.

Accuracy: Numerical Stability

- FP8 (e4m3): limited precision → higher quantization error.
- Block quantization: per-block scaling (naturally aligned with FA3 block ops).
- Incoherent processing: apply random orthogonal transform (Hadamard + ±1 diag) to spread outliers.
- Both techniques cut numerical error by up to 2.6×.

Takeaways

Takeaways

- Core idea: Treat attention as an IO problem → minimize HBM↔SRAM trips (not FLOPs).
- FA-1 (algorithmic):
  - Don't materialize S,P; tile Q/K/V and do incremental softmax with (m,ℓ).
  - Fuse mask+softmax+(dropout)+matmul-V in one kernel; recompute in backward.
  - HBM IO O(N^2*d^2/M); extra memory O(N) → faster + longer contexts.
- FA-3 (hardware-co-designed, Hopper):
  - Warp specialization (producer TMA vs consumer WGMMA) + ring SMEM buffer.
  - Overlap GEMM and softmax (ping-pong, 2-stage).
  - FP8 path: layout fixes + block quantization + incoherent processing.
  - Delivers ~1.5–2× throughput over FA-2 on H100 (FP16/BF16), higher with FP8.
- One-liner: Reduce memory traffic, not FLOPs.

Q&A

Thank You!

Innovation starts here