FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning

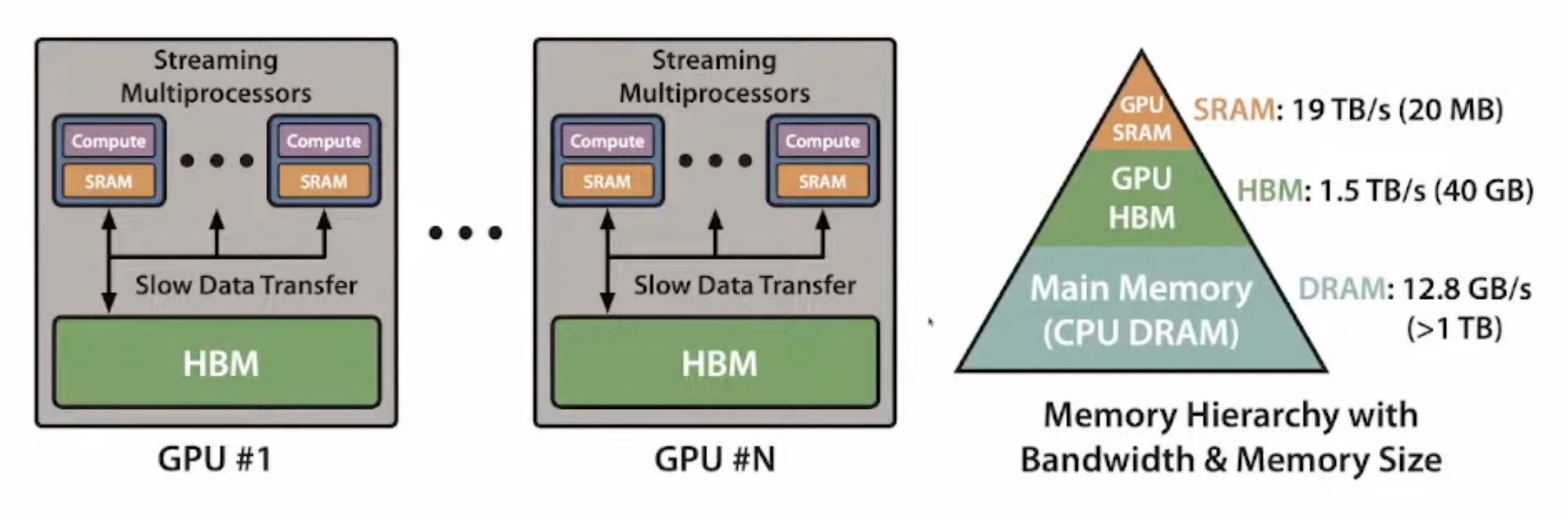
Tri Dao

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré

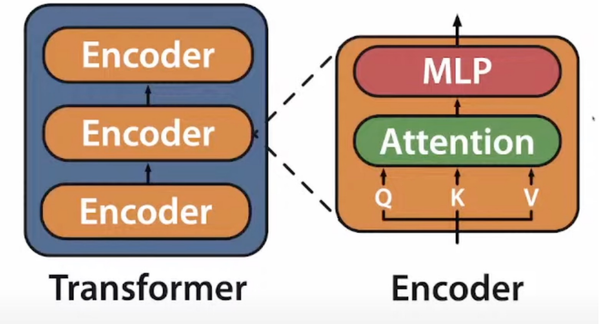
Yuhao Ge

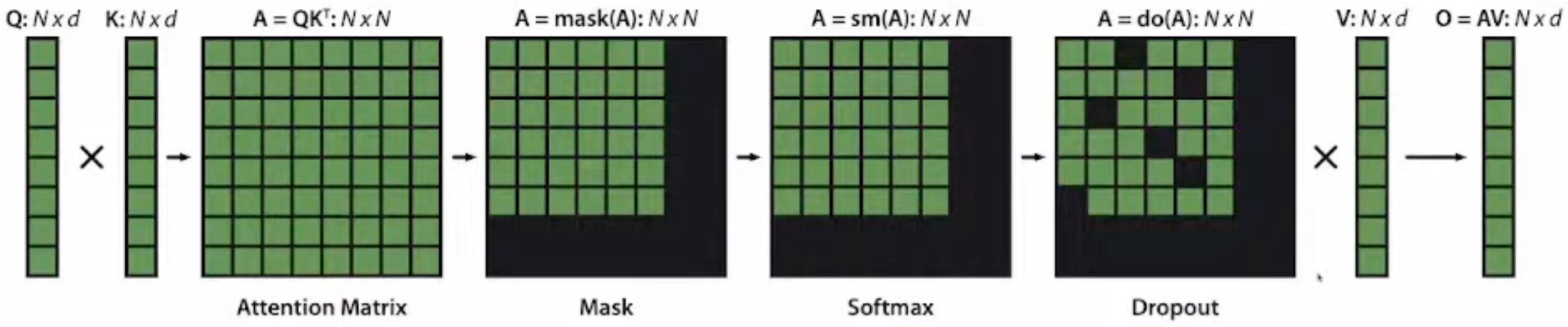
GPU Memory Hierarchy

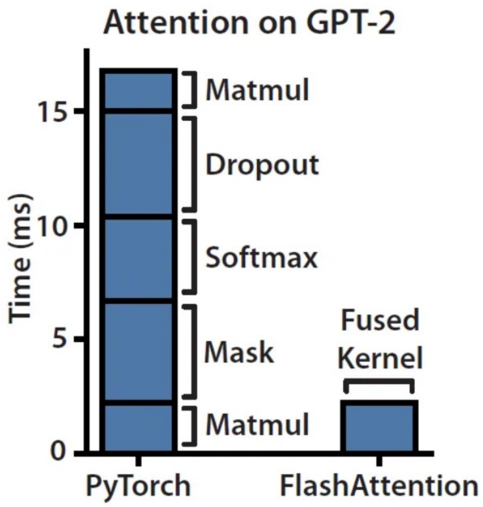


- A massive number of threads to execute an operation (kernel)
  - Load input from HBM to registers and SRAM
  - Computes
  - Load output to HBM

Attention is the Heart of Transformers







$$\mathbf{O} = \mathrm{Dropout}(\mathrm{Softmax}(\mathrm{Mask}(\mathbf{QK^T})))\mathbf{V}$$

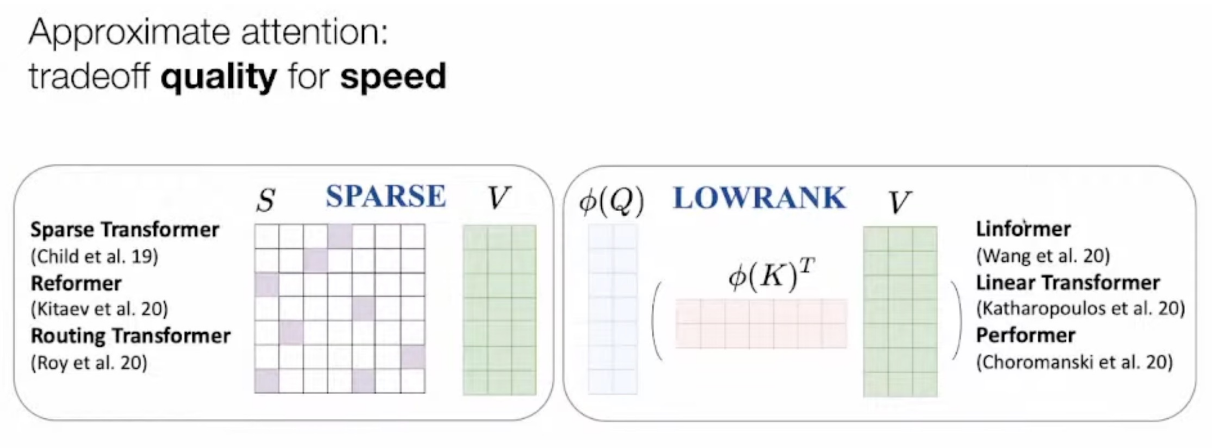
$$\mathbf{S} = \mathbf{QK}^\top \in \mathbb{R}^{N\times N}, \quad \mathbf{P} = \mathrm{softmax}(\mathbf{S}) \in \mathbb{R}^{N\times N}, \quad \mathbf{O} = \mathbf{PV} \in \mathbb{R}^{N\times d},$$

Algorithm 0 Standard Attention Implementation

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

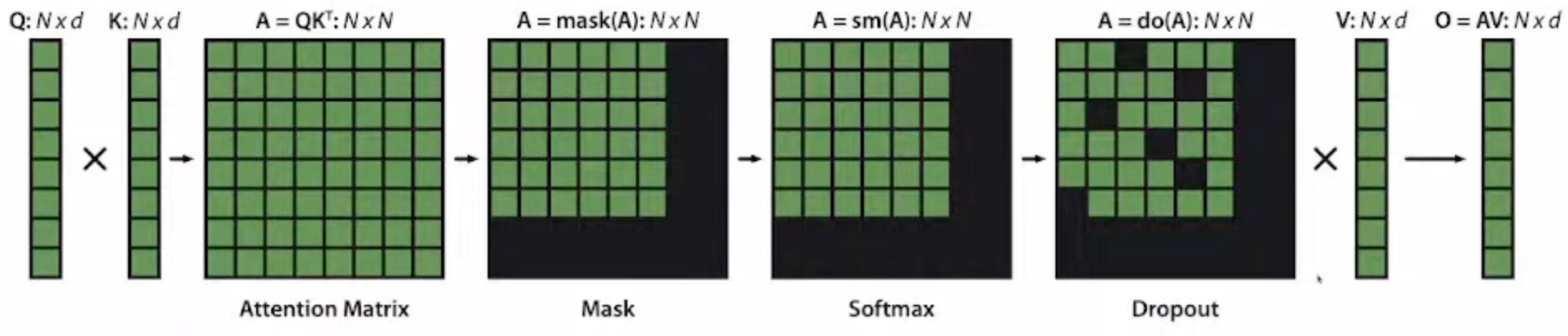
Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{QK}^\top$, write $\mathbf{S}$ to HBM.
Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O}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

Approximate Attention



FlashAttention is Exact Attention

FlashAttention



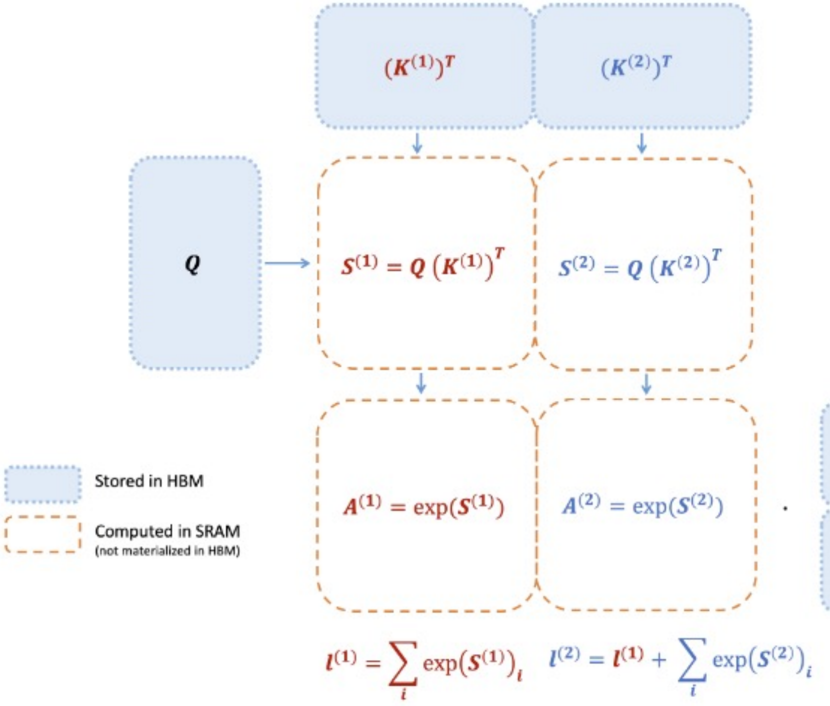
$$\mathbf{O} = \mathrm{Dropout}(\mathrm{Softmax}(\mathrm{Mask}(\mathbf{QK}^{\mathbf{T}})))\mathbf{V}$$

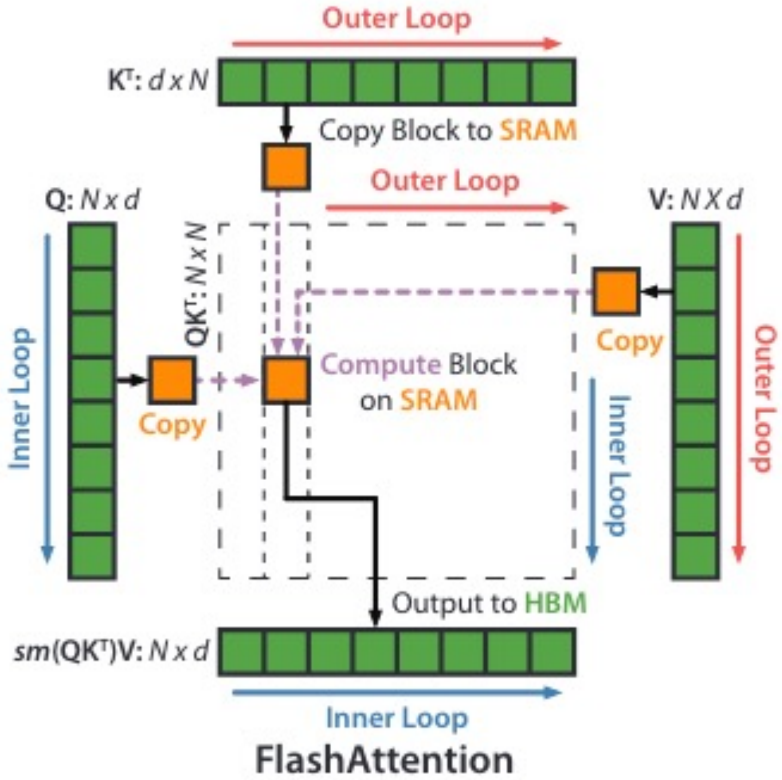
Challenge
- softmax normalization has row dependency
- Attention Matrix has quadratic memory consumption to the seq length

Purpose
- Compute softmax normalization without access to full input
- Backward without the large attention matrix from forward

Approach
- Tiling: Restructure algorithm to load block by block from HBM to SRAM
- Recomputation: Don't store attn. matrix from forward, but recompute

Tiling





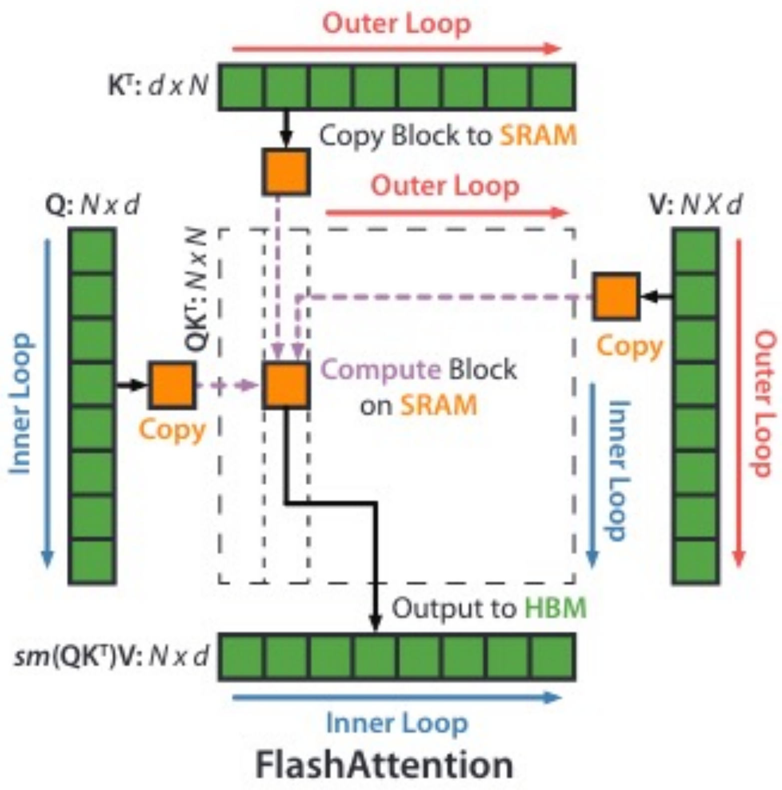
Online softmax instead computes “local” softmax with respect to each block and rescale to get the right output at the end

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax.

Recomputation

By storing softmax normalization factors from forward, quickly recompute attention in the backward from input in SRAM

Attention	Standard	FlashAttention
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3



Result

Faster training speed

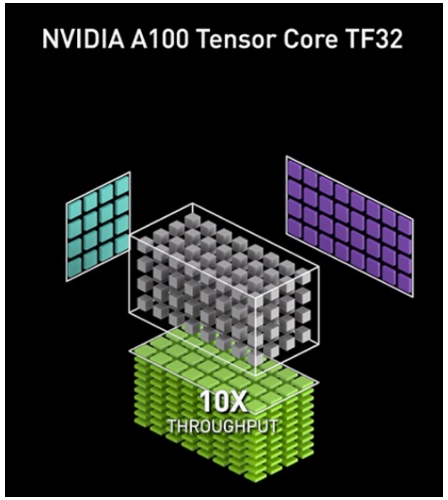
BERT Implementation	Training time (minutes)
Nvidia MLPerf 1.1 [58]	20.0 ± 1.5
FlashAttention (ours)	**17.4** ± 1.4

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [87]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	18.2	4.7 days (2.0×)
GPT-2 small - FlashAttention	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [87]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	14.3	11.5 days (1.8×)
GPT-2 medium - FlashAttention	14.3	**6.9 days (3.0×)**

Support longer sequence length

Model implementations	Context length	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Megatron-LM	1k	18.2	4.7 days (1.0×)
GPT-2 small - FlashAttention	1k	18.2	**2.7 days (1.7×)**
GPT-2 small - FlashAttention	2k	17.6	3.0 days (1.6×)
GPT-2 small - FlashAttention	4k	**17.5**	3.6 days (1.3×)

FlashAttention is still not as efficient as other primitives (GEMM)



Modern GPUs have specialized compute units (e.g., Tensor Cores on Nvidia GPUs) that makes matmul much faster.

Non-matmul FLOP is 16× more expensive than a matmul FLOP

- The forward pass only reaches 30-50% of the maximum throughput
- The backward pass only reaches 25-35% of maximum throughput

Optimized GEMM can reach up to 80-90% of the theoretical maximum device throughput

FlashAttention2

1. **Algorithm**: Tweak the algorithm from FlashAttention to reduce the number of non-matmul FLOPs.

2. **Parallelism**: Additionally parallelize over the sequence length

3. **Work** Partitioning: Decide how to partition the work between different warps.

Algorithm

FlashAttention

$$
\begin{aligned}
m^{(1)} &= \mathrm{rowmax}(\mathbf{S}^{(1)}) \in \mathbb{R}^{B_r} \\
\ell^{(1)} &= \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m^{(1)}}) \in \mathbb{R}^{B_r} \\
&\boxed{\tilde{\mathbf{P}}^{(1)} = \mathrm{diag}(\ell^{(1)})^{-1} e^{\mathbf{S}^{(1)}-m^{(1)}} \in \mathbb{R}^{B_r \times B_c}} \\
\mathbf{O}^{(1)} &= \tilde{\mathbf{P}}^{(1)}\mathbf{V}^{(1)} = \mathrm{diag}(\ell^{(1)})^{-1} e^{\mathbf{S}^{(1)}-m^{(1)}}\mathbf{V}^{(1)} \in \mathbb{R}^{B_r \times d} \\
m^{(2)} &= \max(m^{(1)}, \mathrm{rowmax}(\mathbf{S}^{(2)})) = m \\
\ell^{(2)} &= e^{m^{(1)}-m^{(2)}}\ell^{(1)} + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m^{(2)}}) = \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m}) + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m}) = \ell \\
\tilde{\mathbf{P}}^{(2)} &= \mathrm{diag}(\ell^{(2)})^{-1} e^{\mathbf{S}^{(2)}-m^{(2)}} \\
\mathbf{O}^{(2)} &= \mathrm{diag}(\ell^{(1)}/\ell^{(2)})^{-1}\mathbf{O}^{(1)} + \tilde{\mathbf{P}}^{(2)}\mathbf{V}^{(2)} = \mathrm{diag}(\ell^{(2)})^{-1} e^{s^{(1)}-m}\mathbf{V}^{(1)} + \mathrm{diag}(\ell^{(2)})^{-1} e^{s^{(2)}-m}\mathbf{V}^{(2)} = \mathbf{O}.
\end{aligned}
$$

FlashAttention2

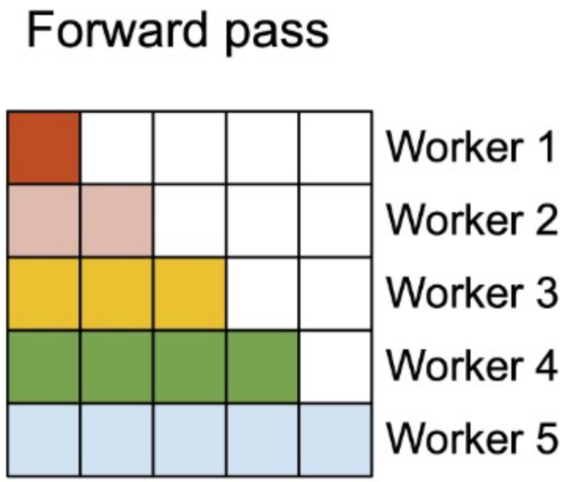
$$
\begin{aligned}
m^{(1)} &= \mathrm{rowmax}(\mathbf{S}^{(1)}) \in \mathbb{R}^{B_r} \\
\ell^{(1)} &= \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m^{(1)}}) \in \mathbb{R}^{B_r} \\
\tilde{\mathbf{O}}^{(1)} &= e^{\mathbf{S}^{(1)}-m^{(1)}}\mathbf{V}^{(1)} \in \mathbb{R}^{B_r \times d} \\
m^{(2)} &= \max(m^{(1)}, \mathrm{rowmax}(\mathbf{S}^{(2)})) = m \\
\ell^{(2)} &= e^{m^{(1)}-m^{(2)}}\ell^{(1)} + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m^{(2)}}) = \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m}) + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m}) = \ell \\
\tilde{\mathbf{P}}^{(2)} &= \mathrm{diag}(\ell^{(2)})^{-1} e^{\mathbf{S}^{(2)}-m^{(2)}} \\
\tilde{\mathbf{O}}^{(2)} &= \mathrm{diag}(e^{m^{(1)}-m^{(2)}})^{-1}\tilde{\mathbf{O}}^{(1)} + e^{\mathbf{S}^{(2)}-m^{(2)}}\mathbf{V}^{(2)} = e^{s^{(1)}-m}\mathbf{V}^{(1)} + e^{s^{(2)}-m}\mathbf{V}^{(2)} \\
\mathbf{O}^{(2)} &= \mathrm{diag}(\ell^{(2)})^{-1}\tilde{\mathbf{O}}^{(2)} = \mathbf{O}.
\end{aligned}
$$

Parallelism

- FlashAttention parallelizes over batch size and number of head
- There are BS * #head thread blocks, each running on a Streaming multiprocessor (SM)
- A100 -- 108 SMs

Parallelize over batch size and number of head → Parallelize over the b. h. and seq length dimension

- FlashAttention
  - For j-th (K, V), for i-th Q, computer using Kj, Vj, Qi in SRAM, and update Oi, li, mi in HBM
- FlashAttention2
  - Swap order of loop
  - Parallelize outer loop
  - Leads to improved occupancy



White square: causal mask for cases like auto-regressive language modeling

Work Partitioning Between Warps

We typically use 4 or 8 warps per thread block
- FlashAttention
  - Split K and V
  - All warps need to write intermediate results out to shared memory, synchronize, then add up
- FlashAttention2
  - Split Q
  - No need for communication between warps

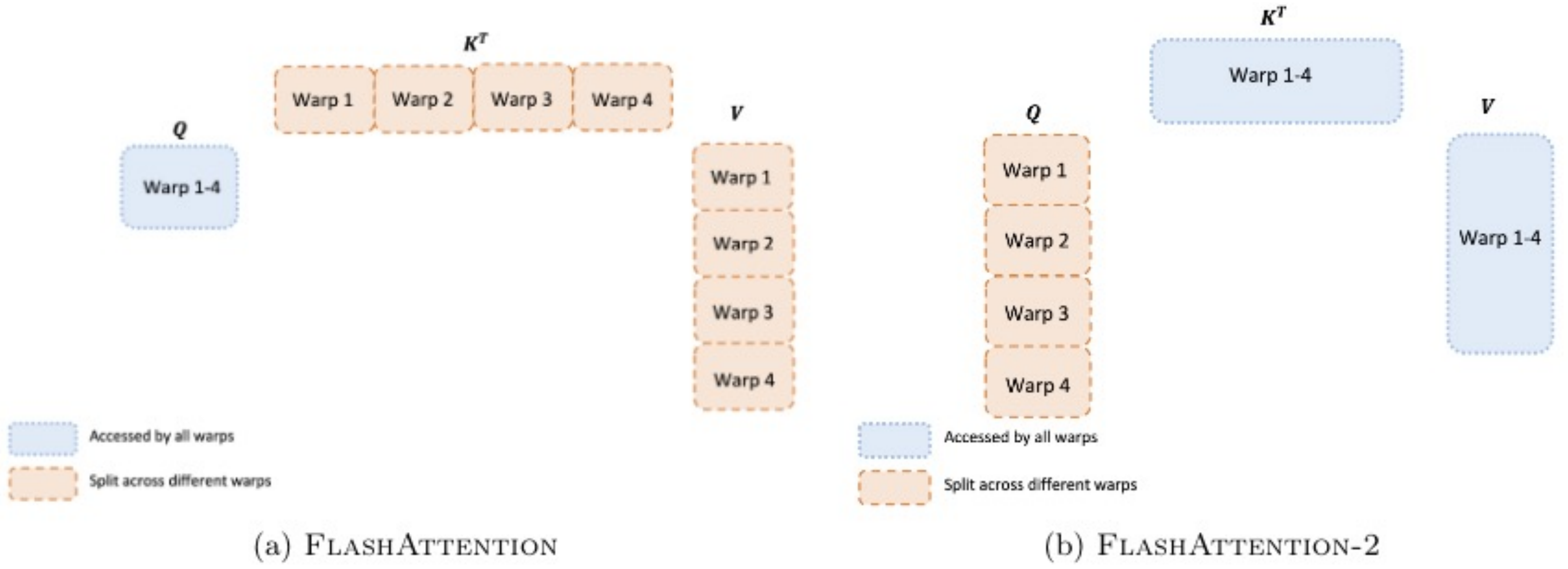
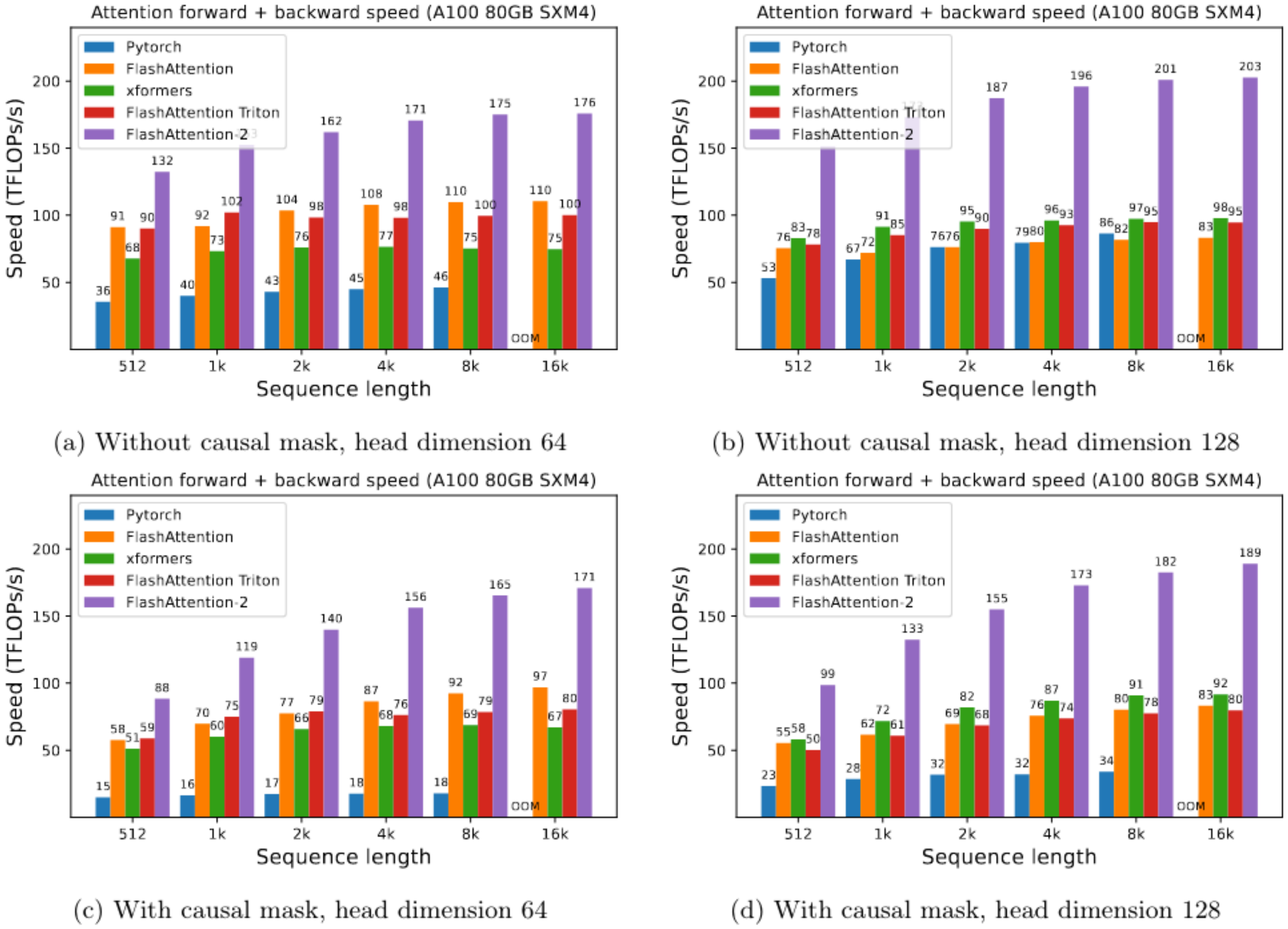


Figure 3: Work partitioning between different warps in the forward pass

Benchmarking attention.



(a) Without causal mask, head dimension 64

(b) Without causal mask, head dimension 128

(c) With causal mask, head dimension 64

(d) With causal mask, head dimension 128

Figure 4: Attention forward + backward speed on A100 GPU

2X faster than FlashAttention
1.3X faster than Triton
10X faster than Pytorch

Reaches up to 230 TFLOPs/s, 73% of the threoretical max on A100

End to end Performance

Table 1: Training speed (TFLOPs/s/GPU) of GPT-style models on 8×A100 GPUs. FlashAttention-2 reaches up to 225 TFLOPs/s (72% model FLOPs utilization). We compare against a baseline running without FlashAttention.

Model	Without FlashAttention	FlashAttention	FlashAttention-2
GPT3-1.3B 2k context	142 TFLOPs/s	189 TFLOPs/s	196 TFLOPs/s
GPT3-1.3B 8k context	72 TFLOPS/s	170 TFLOPs/s	220 TFLOPs/s
GPT3-2.7B 2k context	149 TFLOPs/s	189 TFLOPs/s	205 TFLOPs/s
GPT3-2.7B 8k context	80 TFLOPs/s	175 TFLOPs/s	225 TFLOPs/s

2.8X faster than baseline
1.3X faster than FlashAttention

Summary

- FlashAttention2 is 2x faster than FlashAttention
- FlashAttention2 will also speed up training, finetuning, and inference

- Future Directions
  - Device dependent, only applicable to Nvidia A100
  - Hand-writing CUDA implementation, specially designed for specific attention implementation
  - How the FlashAttention2 performs on sparse attention mechanisms
  - Auto-tuning mechanisms for selecting optimal block sizes and partitioning strategies could simplify the use of FlashAttention-2