UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

CS 498: Machine Learning System
Spring 2025

Minjia Zhang

The Grainger College of Engineering

Today

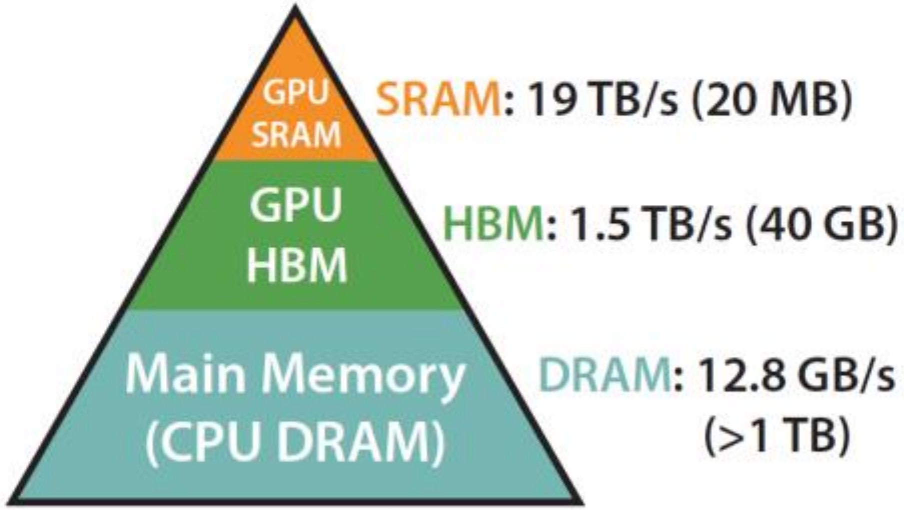
DL Inference

- FlashAttention

Revisit: GPU Memory Hierarchy

Memory is arranged hierarchically

- GPU SRAM is small, and supports the fastest access
- GPU HBM is larger but with much slower access
- CPU DRAM is huge, but the slowest of all



Memory Hierarchy with Bandwidth & Memory Size

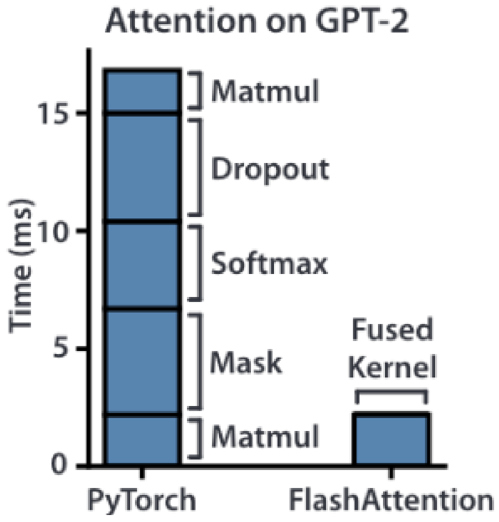
FlashAttention

Inference is usually memory-bound

- Matrix multiplication takes up 99% of the FLOPS
- But only takes up 61% of the runtime
- Lots of time is wasted moving data around on the GPU instead of doing computation

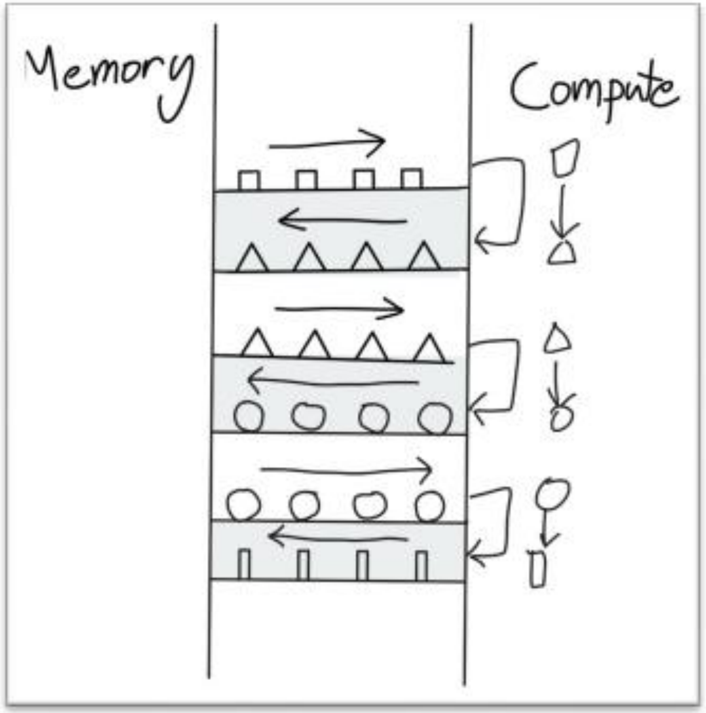
Table 1. Proportions for operator classes in PyTorch.

Operator class	% flop	% Runtime
△ Tensor contraction	99.80	61.0
□ Stat. normalization	0.17	25.5
○ Element-wise	0.03	13.5



Operator Fusion

Version A: Usually, we compute a neural network one operator at a time by moving operation input to GPU SRAM (fast/small), doing some computation, then returning the output to GPU HBM (slow/large)



Version B: Operator fusion instead moves the original input to GPU SRAM (fast/small), does a whole sequence of layer computations without ever touching HBM, and then returns the final layer output to GPU HBM (slow/large)

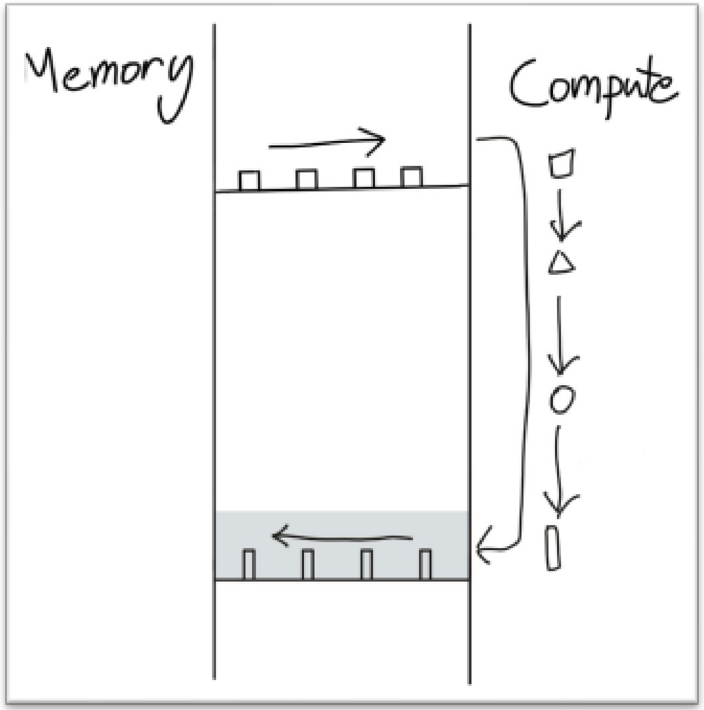


Figure from https://horace.io/brrr_intro.html

Operator Fusion

Version A: Usually, we compute a neural network one operator at a time by moving operation input to GPU SRAM (fast/small), doing some computation, then returning the output to GPU HBM (slow/large)

Version B: Operator fusion instead moves the original input to GPU SRAM (fast/small), does a whole sequence of layer computations without ever touching HBM, and then returns the final layer output to GPU HBM (slow/large)

Version A is how standard attention is implemented

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

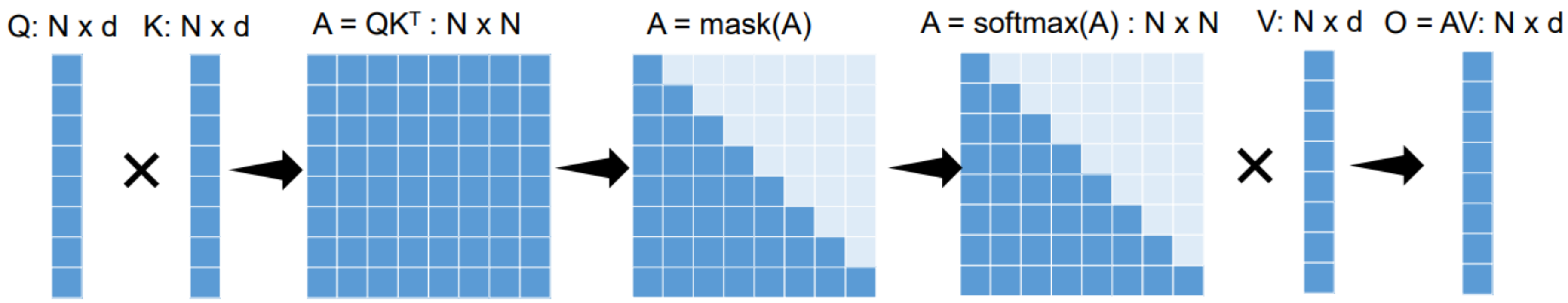
Algorithm 0 Standard Attention Implementation

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

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 $\mathbf{O}$.

Standard Attention

Attention: O = Softmax(QK^T) V



Version A is how standard attention is implemented

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

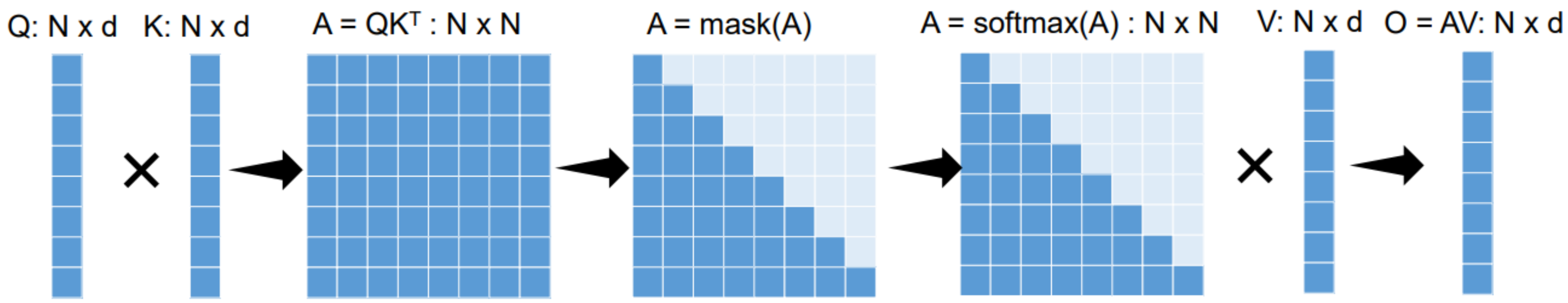
Algorithm 0 Standard Attention Implementation

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

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 $\mathbf{O}$.

Standard Attention

Attention: $O = \text{Softmax}(QK^T)\,V$



Challenges:

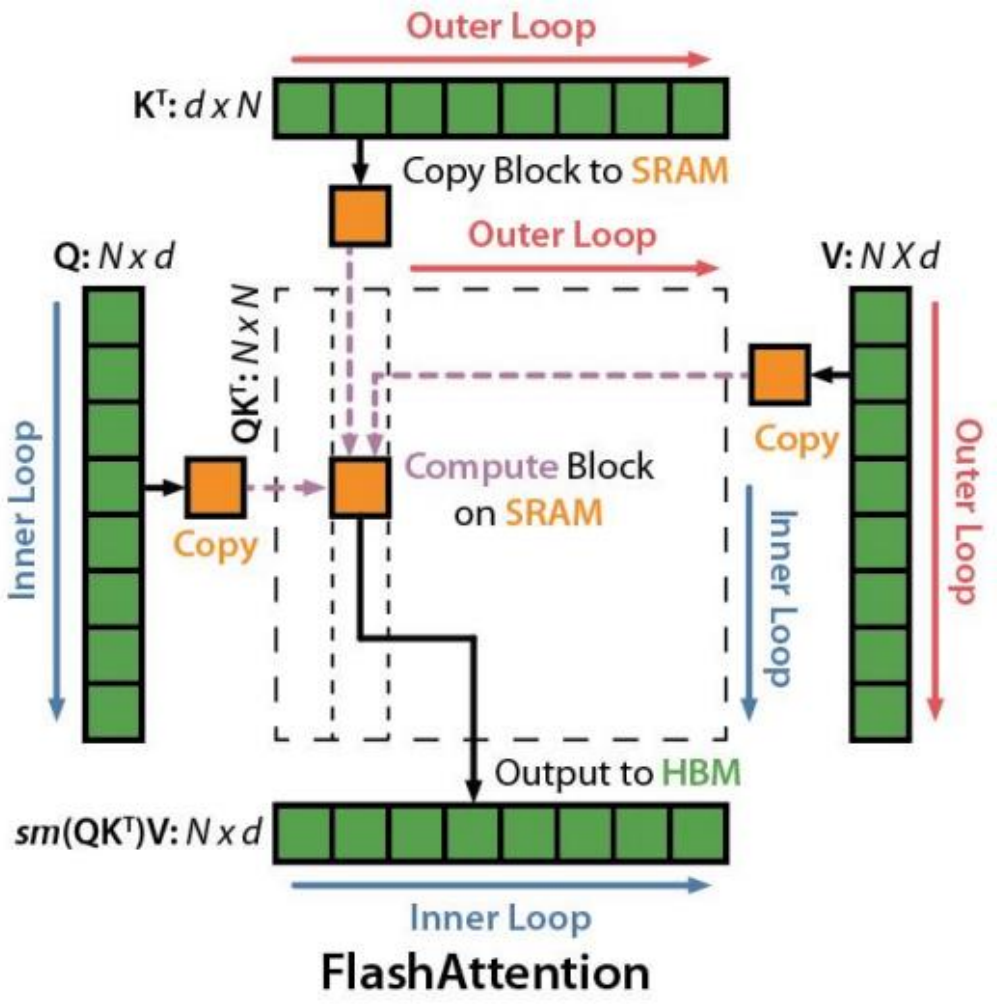
- Repeated reads/writes from GPU device memory
- Large intermediate results
- Cannot scale to long sequences due to O(N^2) intermediate results

FlashAttention

- Three key ideas are combined to obtain FlashAttention
  - Kernel fusion: One kernel that includes all operators during attention computation to avoid kernel launching overhead and intermediate data movement
  - Tiling: compute the attention weights block by block so that we don’t have to load everything into SRAM at once
  - Recomputation: don’t store the full attention matrix in forward, but just recompute the parts of it you need during the backward pass

Tiling: Decompose Large GeMM into Smaller Blocks

1. Load inputs by blocks from global to shared memory
2. On chip, compute attention output wrt the block
3. Update output in device memory by scaling



Tiling: Decompose Large GeMM into Smaller Blocks

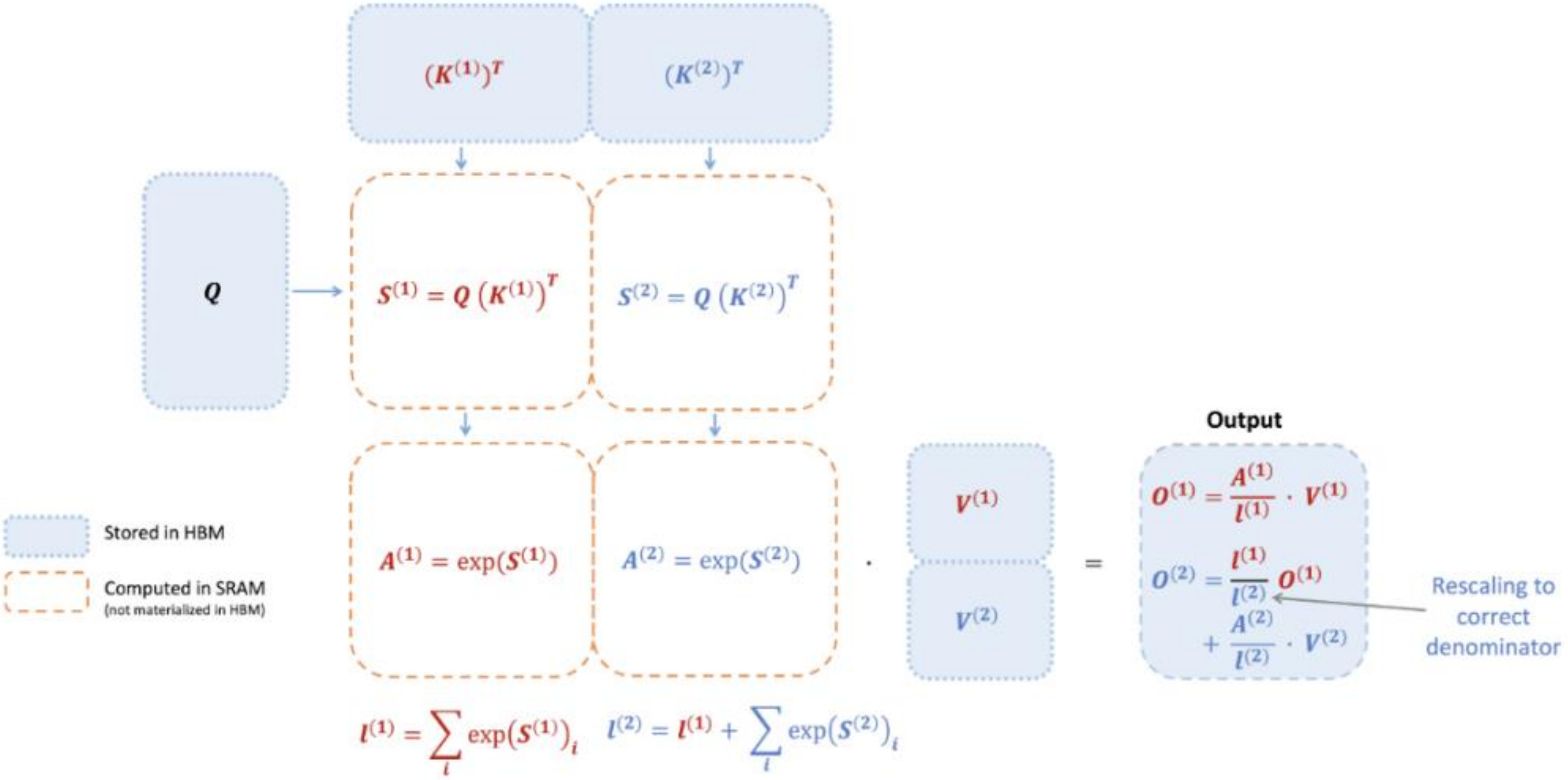
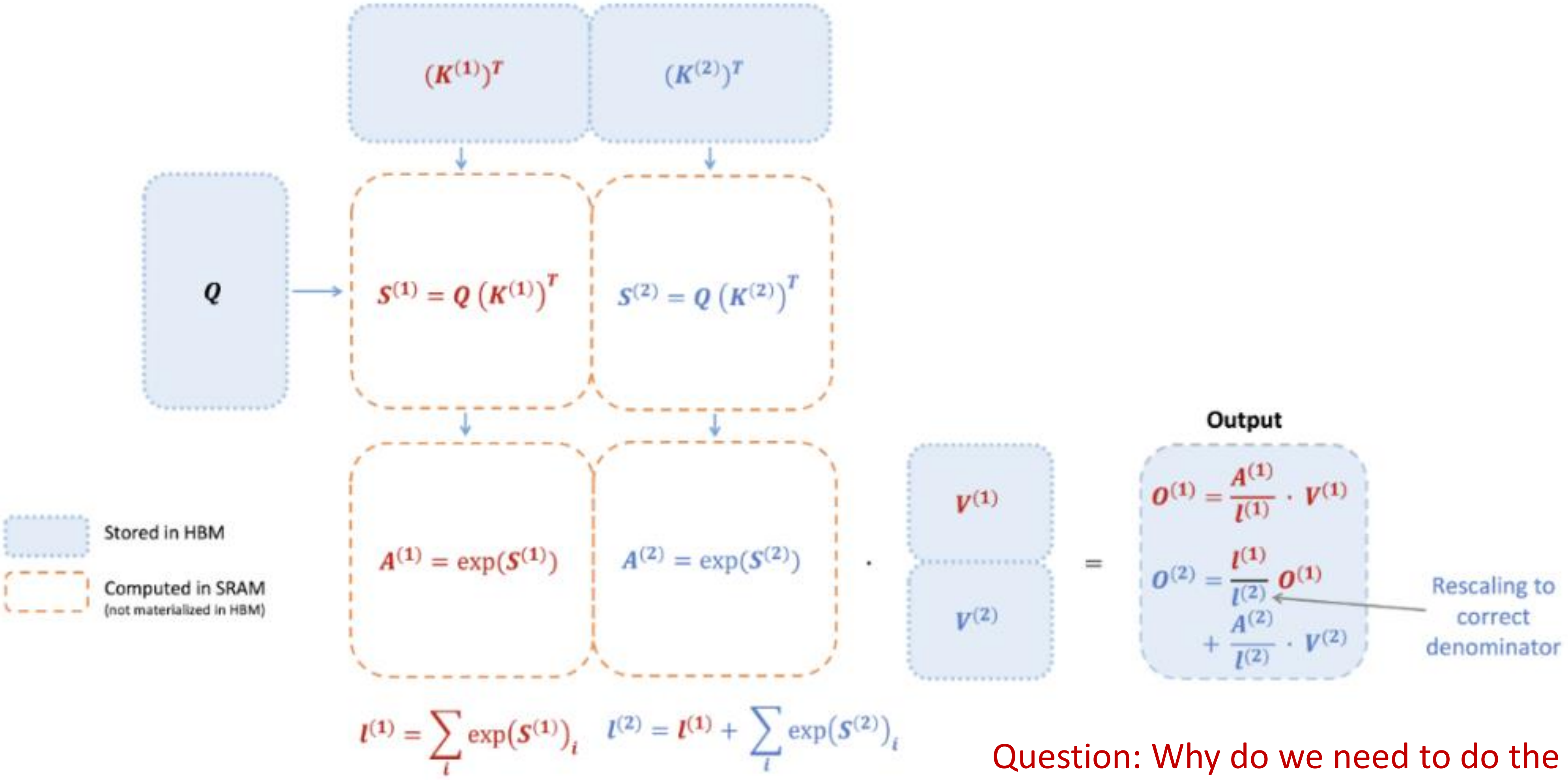


Figure from http://arxiv.org/abs/2307.08691

Tiling: Decompose Large GeMM into Smaller Blocks



Question: Why do we need to do the scaling? And how?

Figure from http://arxiv.org/abs/2307.08691

How to Implement Softmax?

For a vector $x \in \mathbb{R}^B$, softmax is computed as:

$$\text{softmax}(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Issue: Easily lead to numerical instability, e.g., overflow because of $\sum_j e^{x_j}$

Stable Softmax

1. Compute the maximum value in x:

$$m(x) := \max_i x_i$$

2. Compute the adjusted exponentials:

$$f(x) := \left[e^{x_1 - m(x)}, e^{x_2 - m(x)}, \ldots, e^{x_B - m(x)}\right]$$

3. Compute the normalization denominator:

$$\ell(x) := \sum_i f(x)_i$$

4. Compute the final softmax:

$$\text{softmax}(x) = \frac{f(x)}{\ell(x)}$$

FlashAttention Tiled Softmax: Tiling + Online Softmax

Let's say we have two blocks $x^{(1)}$ and $x^{(2)}$ each of size B. The concatenated vector is:

$$x = [x^{(1)}, x^{(2)}] \in \mathbb{R}^{2B}$$

1. Track the max value across blocks

$$m(x) = \max(m(x^{(1)}), m(x^{(2)}))$$

2. Compute adjusted exponentials:

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

3. Compute the normalization denominator (incrementally):

$$\ell(x) = e^{m(x^{(1)})-m(x)} \ell(x^{(1)}) + e^{m(x^{(2)})-m(x)} \ell(x^{(2)})$$

4. Compute the final softmax:

$$\text{softmax}(x) = \frac{f(x)}{\ell(x)}$$

FlashAttention Tiled Softmax: Tiling + Online Softmax

Let's say we have two blocks $x^{(1)}$ and $x^{(2)}$ each of size B. The concatenated vector is:

$$x = [x^{(1)}, x^{(2)}] \in \mathbb{R}^{2B}$$

1. Track the max value across blocks

$$m(x) = \max(m(x^{(1)}), m(x^{(2)}))$$

2. Compute adjusted exponentials:

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

3. Compute the normalization denominator (incrementally):

$$\ell(x) = e^{m(x^{(1)})-m(x)} \ell(x^{(1)}) + e^{m(x^{(2)})-m(x)} \ell(x^{(2)})$$

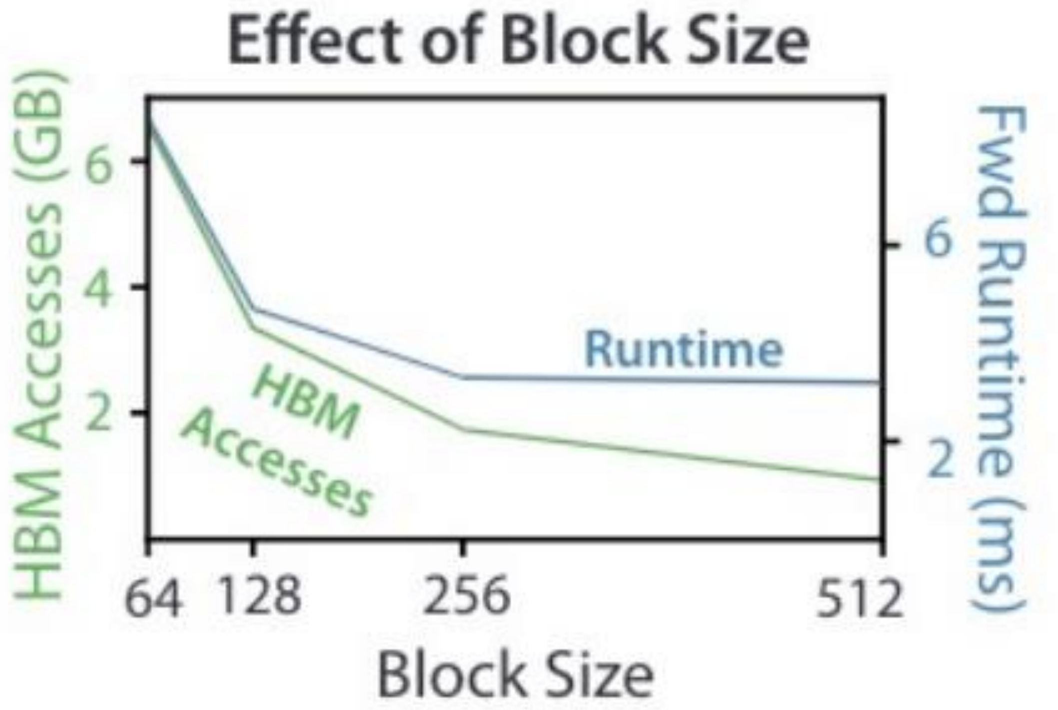
4. Compute the final softmax:

$$\mathrm{softmax}(x) = \frac{f(x)}{\ell(x)}$$

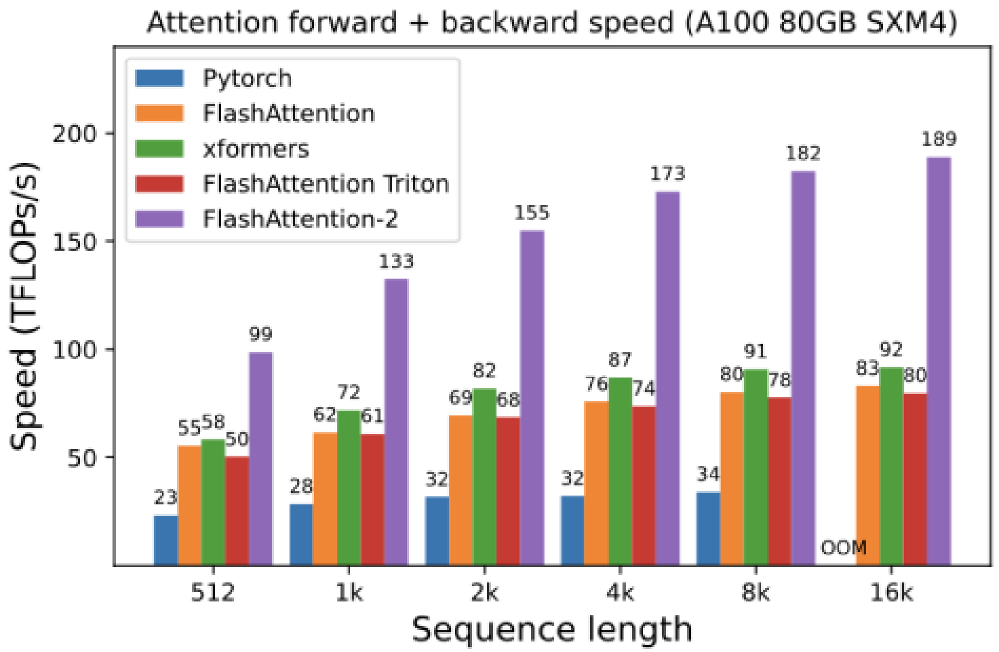
Only need to track intermediate statistics to compute softmax one block at a time

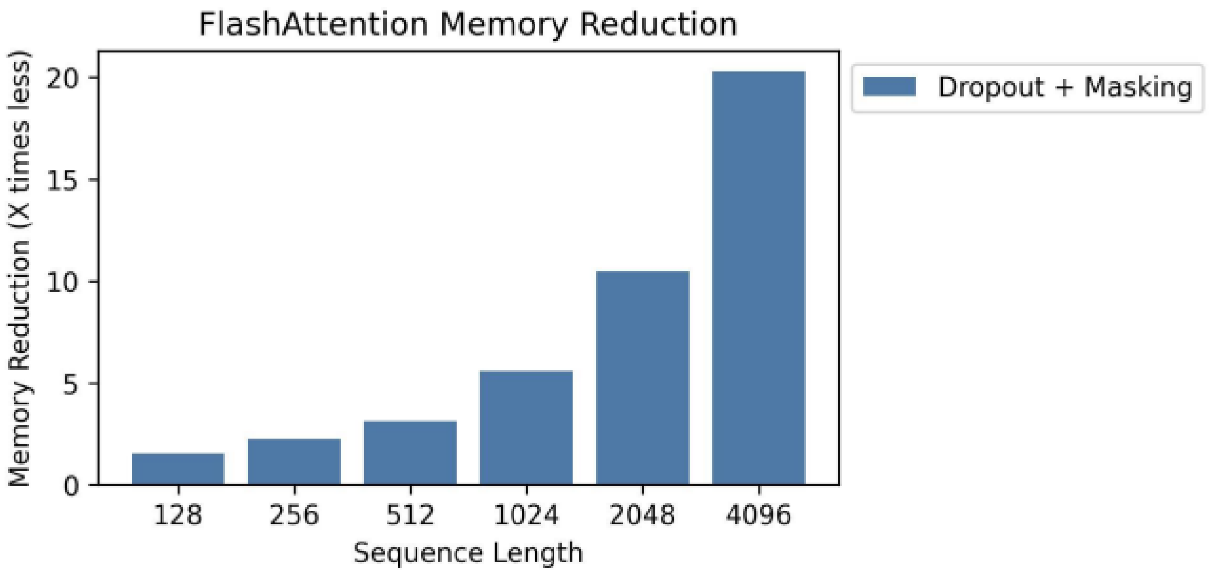
FlashAttention Results

The algorithm is performing exact attention, no reduction in perplexity or quality of the model



FlashAttention: 2-4x speedup, 10-20x memory reduction





Memory linear in sequence length

Questions?