10-423/10-623 Generative AI

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Efficient Attention (FlashAttention)

Matt Gormley & Henry Chai
Lecture 18
Nov. 4, 2024

Reminders

- **Homework 4: Visual Language Models**
  - **Out: Fri, Oct 25**
  - **Due: Tue, Nov 5 at 11:59pm**

FLASHATTENTION

Figure from https://slideslive.com/38988230/flashattention-fast-and-memoryefficient-exact-attention-with-ioawareness

FlashAttention

- One of the most impactful ideas in ML recently
- Even though many people probably don't even know they are using it!
- Introduced at HAET Workshop @ ICML July 2022
- Published @ NeurIPS Dec 2022

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

Tri Dao, Dan Fu ({trid, danfu}@cs.stanford.edu)
7/23/22 HAET Workshop @ ICML 2022

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Ruda, Christopher Ré. Flash Attention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *arXiv preprint arXiv:2205.14135.*
https://github.com/HazyResearch/flash-attention.

Figure from https://slideslive.com/38988230/flashattention-fast-and-memoryefficient-exact-attention-with-ioawareness

FlashAttention

- One of the most impactful ideas in ML recently
- Even though many people probably don't even know they are using it!
- Introduced at HAET Workshop @ ICML July 2022
- Published @ NeurIPS Dec 2022

Figure from https://awaisrauf.github.io/deepCuriosity/Attending-NeurIPS2023

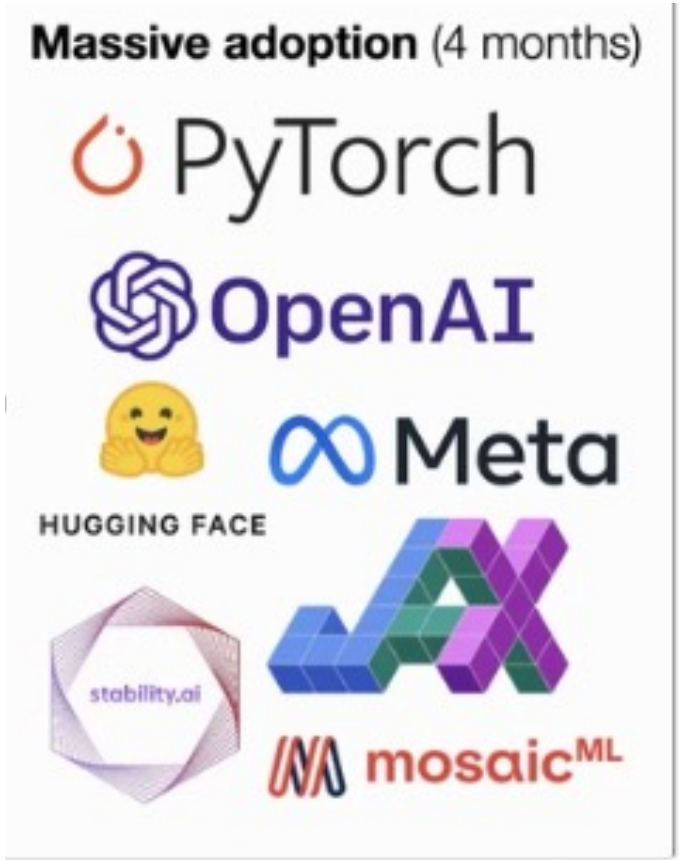
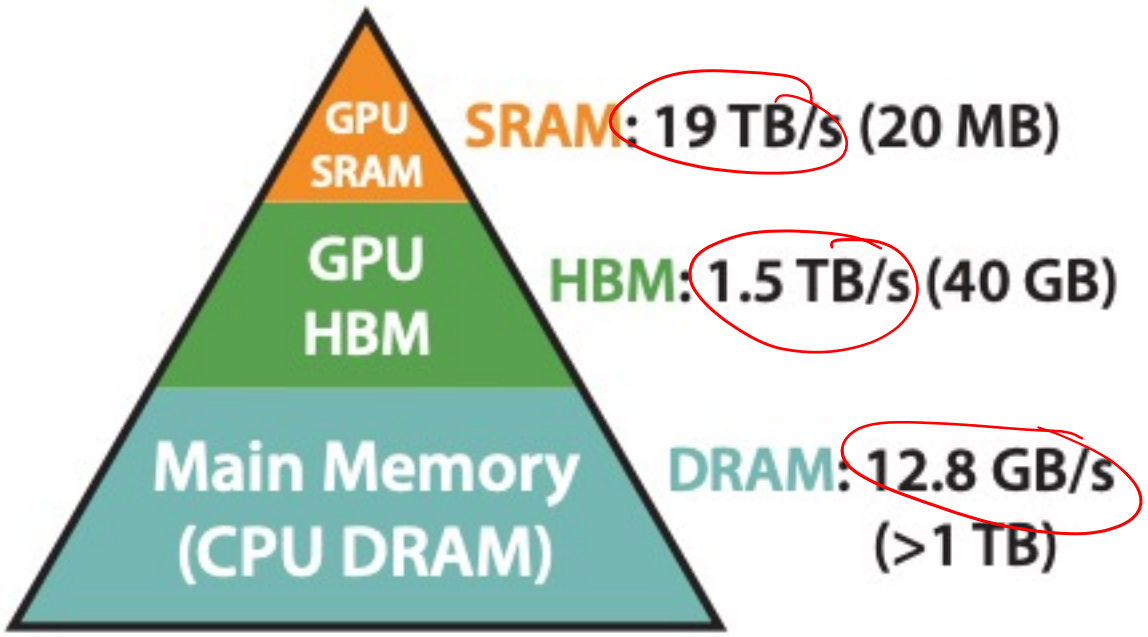


Figure from https://neurips.cc/virtual/2022/poster/54008

GPU Memory

Memory is arranged hierarchicaly

- 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

Figure from https://arxiv.org/pdf/2205.14135

GPU Memory and Transformers

Transformer training 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

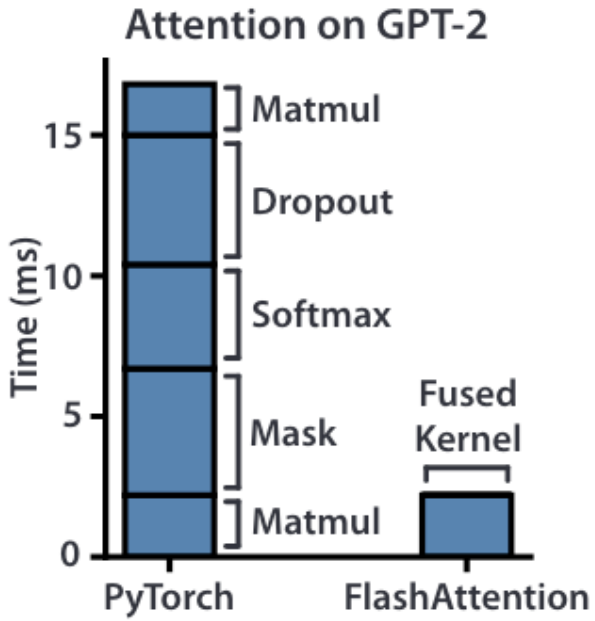


Figure from https://arxiv.org/pdf/2007.00072

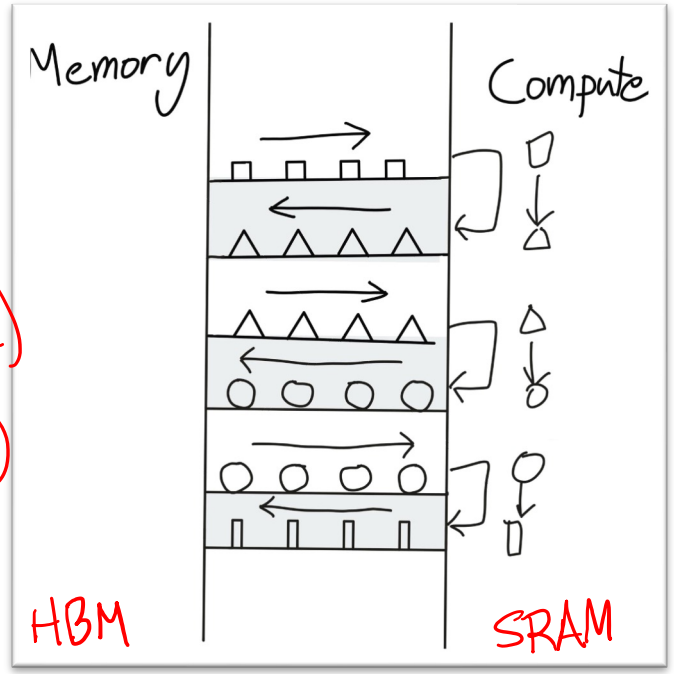
Figure from https://arxiv.org/pdf/2205.14135

Operator Fusion

Version A: Usually, we compute a neural network one layer one at a time by moving the layer 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)

$$x_1 = f_1(v)$$
$$x_2 = f_2(x_1)$$
$$x_3 = f_3(x_2)$$
$$x_4 = f_4(x_3)$$



$$x_4 = f_4(f_3(f_2(f_1(v))))$$

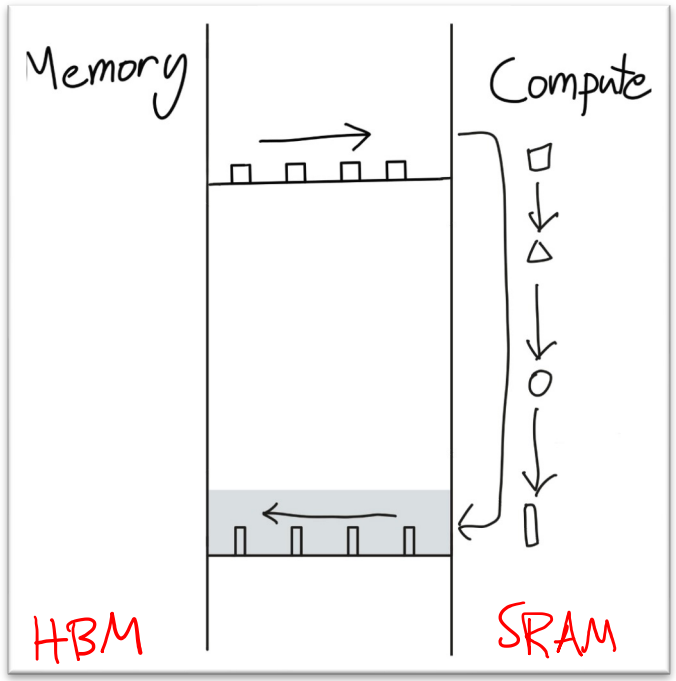


Figure from https://horace.io/brrr_intro.html

Operator Fusion

Version A: Usually, we compute a neural network one layer one at a time by moving the layer 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 exactly how standard attention is implemented

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

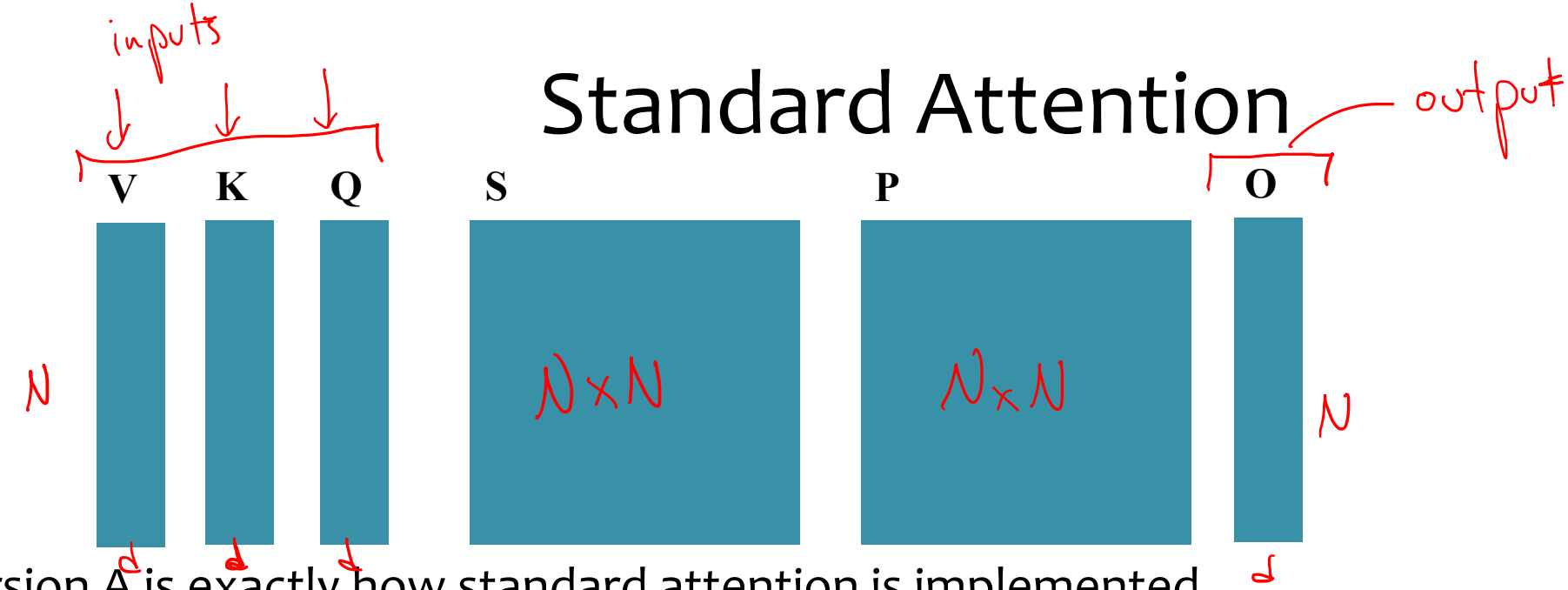
Algorithm 0 Standard Attention Implementation

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 $\mathbf{O}$.

Figure from https://arxiv.org/pdf/2205.14135

Standard Attention



Version A is exactly 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 $\mathbf{O}$.

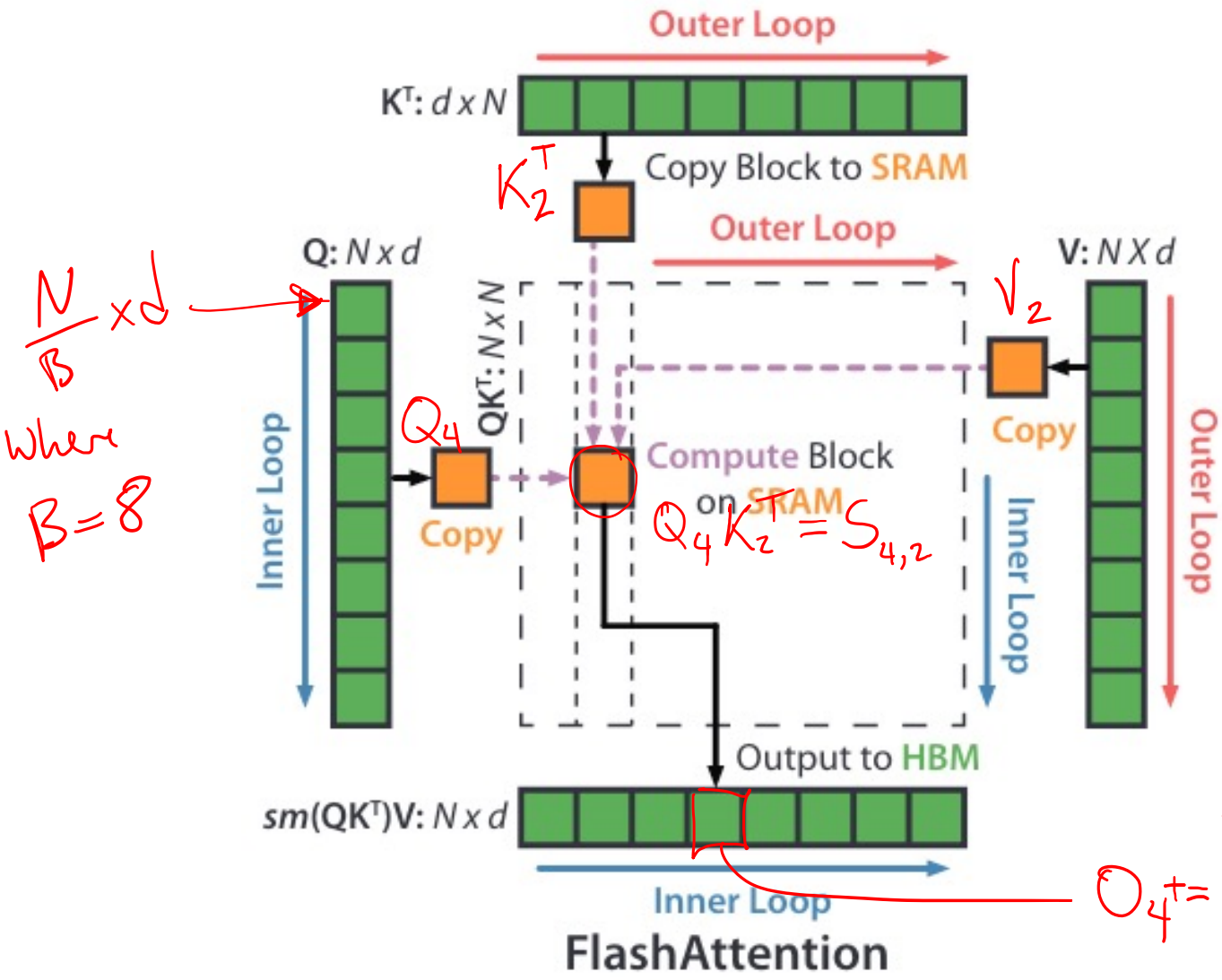
Figure from https://arxiv.org/pdf/2205.14135

FlashAttention

- Two key ideas are combined to obtain FlashAttention
- Both are well-established ideas, so the interesting part is how they are put together for attention
  1. Tiling: compute the attention weights block by block so that we don’t have to load everything into SRAM at once
  2. Recomputation: don’t ever store the full attention matrix, but just recompute the parts of it you need during the backward pass

FlashAttention: Tiling

$S = QK^T$ is divided into B^2 blocks



$\frac{N}{B} \times d$

where $B = 8$

$P_{ij} V_j$

$O_4 += \text{chentng\_softmax}(Q_4 K_2^T) V_2$

Figure from https://arxiv.org/pdf/2205.14135

FlashAttention

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 $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ 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 $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.

Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \ldots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \ldots, m_{T_r}$ 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 $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.

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

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

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

  end for

end for

Return $\mathbf{O}$.

FlashAttention: Tiling

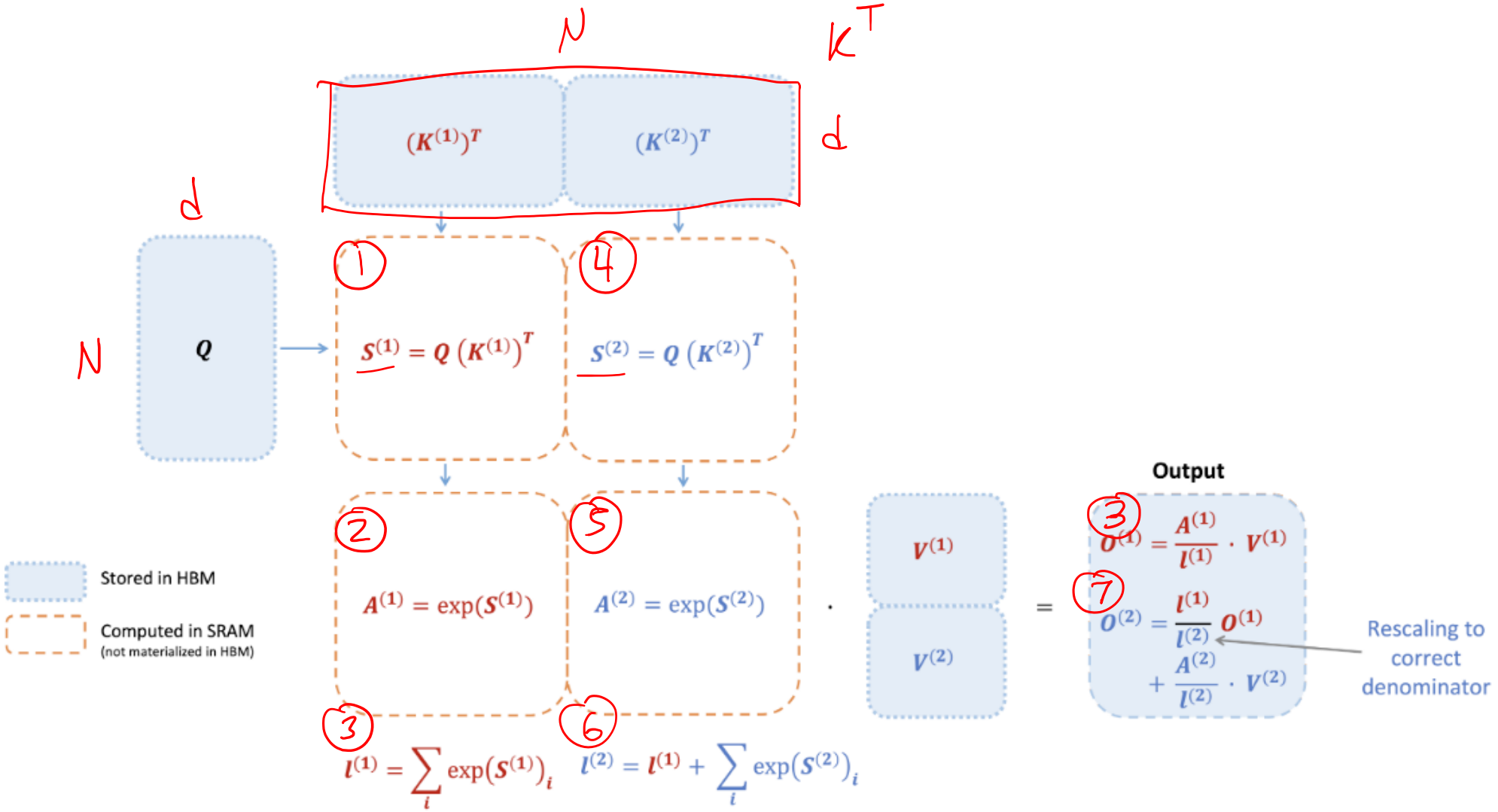
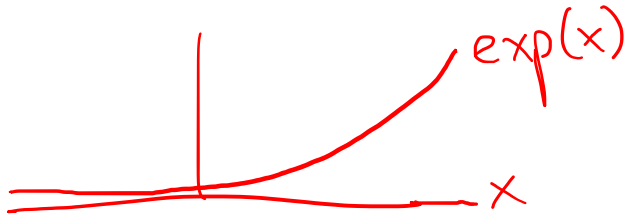


Figure from http://arxiv.org/abs/2307.08691



FlashAttention: Tiling

One of the key challenges is how to compute the softmax since it is inherently going to require working with multiple blocks

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

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

Online Softmax

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:

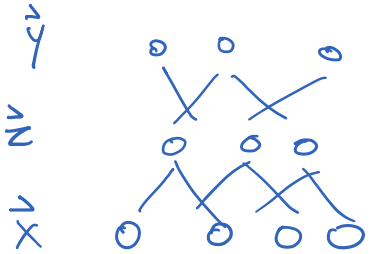
$$m(x) = m(\left[x^{(1)} \; x^{(2)} \right]) = \max(m(x^{(1)}), m(x^{(2)})), \quad f(x) = \left[e^{m(x^{(1)}) - m(x)} f(x^{(1)}) \quad e^{m(x^{(2)}) - m(x)} f(x^{(2)}) \right],$$

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

Therefore if we keep track of some extra statistics $(m(x), \ell(x))$, we can compute softmax one block at a time.

Figure from https://arxiv.org/pdf/2205.14135

Reconstruction for a Feed-Forward MLP



Forward

$z = \sigma(W_1 x + b_1)$

$y = \text{softmax}(W_2 z + b_2)$

Backward

$\partial J/\partial y = \ldots$

$\partial J/\partial z = \partial J/\partial y \cdot \partial y/\partial z$

$\partial J/\partial x = \partial J/\partial z \cdot \partial z/\partial x$

function of $z = z(1-z)\,W_1$

Forward

$z = \sigma(W_1 x + b_1)$

$y = \text{softmax}(W_2 z + b_2)$

del z

reconstruction

Backward

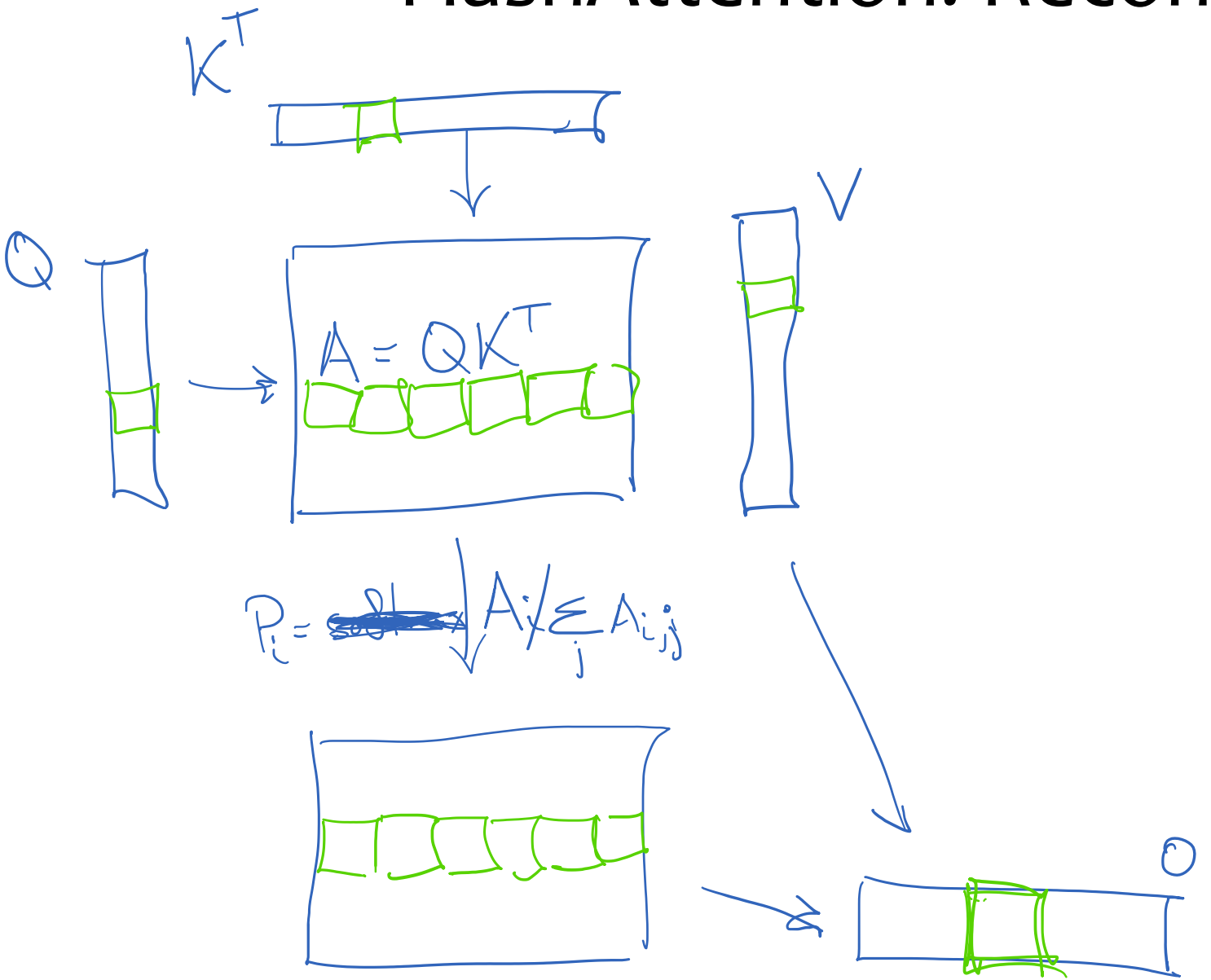
$\partial J/\partial y = \ldots$

$\partial J/\partial z = \ldots$

$z = \sigma(W_1 x + b_1)$

$\partial J/\partial x = \ldots$

FlashAttention: Reconstruction



FlashAttention

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 $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ 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 $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.

Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \ldots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \ldots, m_{T_r}$ 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 $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.

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

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

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

  end for

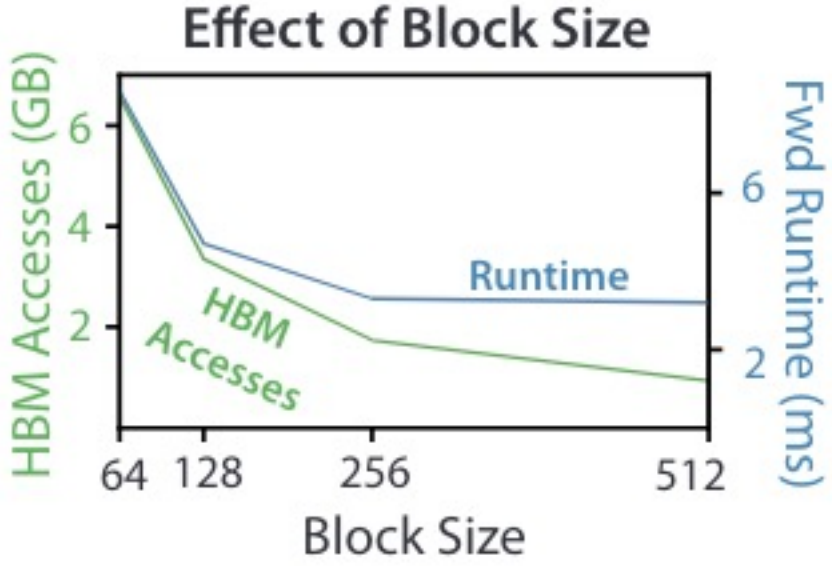
end for

Return $\mathbf{O}$.

FlashAttention: Results

- The algorithm is performing exact attention, so we see no reduction in perplexity or quality of the model
- The key metric is runtime

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



FlashAttention: Results

- The algorithm is performing exact attention, so we see no reduction in perplexity or quality of the model
- The key metric is runtime

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×)**