FlashAttention

cool paper

Attention

- Typical Attention Computing Flow:

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 $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.

Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.

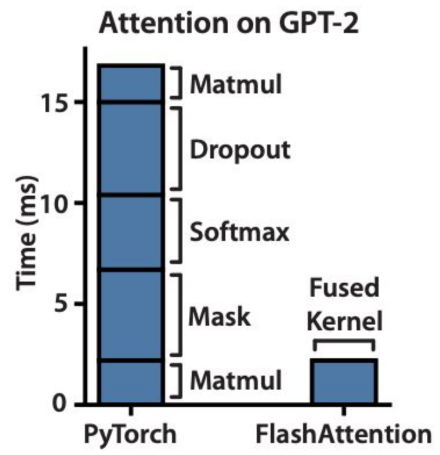
Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.

Return $\mathbf{O}$.

- TimeComplexity O(N^2 * d)
- Memory O(N^2)
- Usually N >> d (eg. N=4000, d=128)

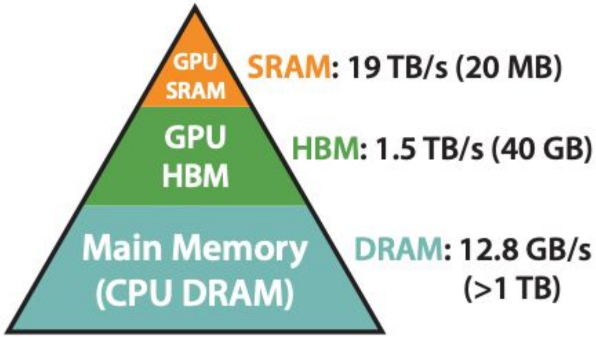
FlashAttention

- Fast and Memory Efficient Exact Attention with IO-Awareness
- Contribution
  - Exact: No approximation. (But it can be modified…)
  - Faster: BERT-large (15% quicker, E2E)
  - Use Less Memory: linear in sequence length



Background: GPU Memory Hierarchy

- A100
  - HBM(40-80G), 1.5TB/s bandwidth
  - GPU SRAM (192 KB per 108 streaming multiprocessors), 19TB/s
- Compute Bound vs Memory Bound
  - Compute Bound:
    - Matrix multiply with large inner dimension, and convolution with large number of channels
  - Memory Bound
    - elementwise (e.g., activation, dropout), and reduction (e.g., sum, softmax, batch norm, layer norm)



Memory Hierarchy with Bandwidth & Memory Size

Algorithm - Challenges

Algorithm 0 Standard Attention Implementation

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

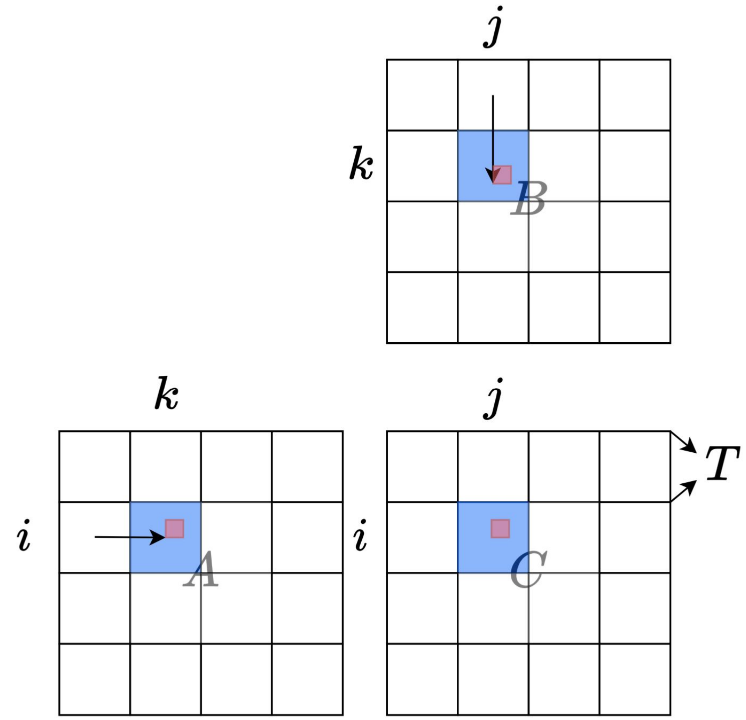
1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\boxed{\mathbf{S} = \mathbf{Q}\mathbf{K}^\top}$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\boxed{\mathbf{P} = \mathrm{softmax}(\mathbf{S})}$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks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

---

1. Matrix Multiplication - (Memory)
2. Softmax - (Memory, I/O)
3. Backward - (Memory)

Algorithm- Matrix Multiplication

- Tiling
- Only need 3T^2 elements stored on chips



Algorithm- Online Softmax

- Safe Softmax (3-pass):

$$\frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} = \frac{e^{x_i - m}}{\sum_{j=1}^{N} e^{x_j - m}}$$

- m_N must be computed after one pass.
  - … or not?

Algorithm 3-pass safe softmax

NOTATIONS

$\{m_i\}$: $\max_{j=1}^{i} \{x_j\}$, with initial value $m_0 = -\infty$.

$\{d_i\}$: $\sum_{j=1}^{i} e^{x_j - m_N}$, with initial value $d_0 = 0$, d_N is the denominator of safe softmax.

$\{a_i\}$: the final softmax value.

BODY

for $i \leftarrow 1, N$ **do**

$$m_i \leftarrow \max(m_{i-1}, x_i) \tag{7}$$

end

for $i \leftarrow 1, N$ **do**

$$d_i \leftarrow d_{i-1} + e^{x_i - m_N} \tag{8}$$

end

for $i \leftarrow 1, N$ **do**

$$a_i \leftarrow \frac{e^{x_i - m_N}}{d_N} \tag{9}$$

end

Algorithm- Online Softmax

- Safe Softmax (2-pass):
- Can we do 1 pass then?
  - Unfortunately, no.
  - … But we are not computing Softmax(Q* K^T) only, we want Softmax(Q* K^T) * V

$$
\begin{aligned}
d'_i &= \sum_{j=1}^{i} e^{x_j - m_i} \\
&= \left(\sum_{j=1}^{i-1} e^{x_j - m_i} \right) + e^{x_i - m_i} \\
&= \left(\sum_{j=1}^{i-1} e^{x_j - m_{i-1}} \right) e^{m_{i-1} - m_i} + e^{x_i - m_i} \\
&= d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}
\end{aligned}
$$

Algorithm- Online Softmax

- Magically, we can save one more pass.

$$\boldsymbol{o}_i := \sum_{j=1}^{i} \left(\frac{e^{x_j - m_N}}{\boxed{d'_N}} V[j,:] \right) \tag{13}$$

This still depends on m_N and d_N which cannot be determined until the previous loop completes. But we can play the "surrogate" trick in section 3 again, by creating a surrogate sequence $\boldsymbol{o}'$:

$$\boldsymbol{o}'_i := \left(\sum_{j=1}^{i} \frac{e^{x_j - m_i}}{d'_i} V[j,:] \right)$$

The n-th element of $\boldsymbol{o}$ and $\boldsymbol{o}'$ are the identical: $\boldsymbol{o}'_N = \boldsymbol{o}_N$, and we can find a recurrence relation between $\boldsymbol{o}'_i$ and $\boldsymbol{o}'_{i-1}$:

$$\begin{aligned}
\boldsymbol{o}'_i &= \sum_{j=1}^{i} \frac{e^{x_j - m_i}}{d'_i} V[j,:] \\
&= \left(\sum_{j=1}^{i-1} \frac{e^{x_j - m_i}}{d'_i} V[j,:] \right) + \frac{e^{x_i - m_i}}{d'_i} V[i,:] \\
&= \left(\sum_{j=1}^{i-1} \frac{e^{x_j - m_{i-1}}}{d'_{i-1}} \frac{e^{x_j - m_i}}{e^{x_j - m_{i-1}}} \frac{d'_{i-1}}{d'_i} V[j,:] \right) + \frac{e^{x_i - m_i}}{d'_i} V[i,:] \\
&= \left(\sum_{j=1}^{i-1} \frac{e^{x_j - m_{i-1}}}{d'_{i-1}} V[j,:] \right) \frac{d'_{i-1}}{d'_i} e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d'_i} V[i,:] \\
&= \boldsymbol{o}'_{i-1} \frac{d'_{i-1} e^{m_{i-1} - m_i}}{d'_i} + \frac{e^{x_i - m_i}}{d'_i} V[i,:] \qquad (14)
\end{aligned}$$

Algorithm - All Together

Algorithm 1 FlashAttention

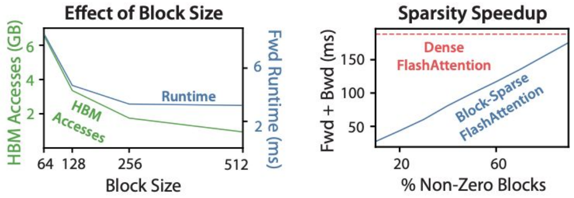
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M.

Set block sizes $B_c = \left\lceil \frac{M}{4d} \right\rceil, B_r = \min\left(\left\lceil \frac{M}{4d} \right\rceil, d\right)$.
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 $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (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 $\mathbf{O}$.

Algorithm

- O(N^2*d) FLOPs, O(N) extra memory
- HBM access?
    - Standard: O(Nd+N^2)
    - Flashattention: O(N^2*d^2/M)
    - In practice, d = 64-128, M=100k, d^2 << M

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



Algorithm - Backward?

- Recompute!
  - Q, K, V, O, l, m has already been stored.
  - For each block of dQ, dK and dV, recompute the block of S and P.

Algorithm 4 FLASHATTENTION Backward Pass

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{dO} \in \mathbb{R}^{N\times d}$ in HBM, vectors $\ell, m \in \mathbb{R}^N$ in HBM, on-chip SRAM of size M, softmax scaling constant $\tau \in \mathbb{R}$, masking function MASK, dropout probability p_{drop}, pseudo-random number generator state $\mathcal{R}$ from the forward pass.

Set the pseudo-random number generator state to $\mathcal{R}$.
Set block sizes $B_c = \left\lceil \frac{M}{4d} \right\rceil, B_r = \min\left(\left\lceil \frac{M}{4d} \right\rceil, d\right)$.
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 $\mathbf{dO}$ into T_r blocks $\mathbf{dO}_i, \dots, \mathbf{dO}_{T_r}$ 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.
Initialize $\mathbf{dQ} = (0)_{N\times d}$ in HBM and divide it into T_r blocks $\mathbf{dQ}_1, \dots, \mathbf{dQ}_{T_r}$ of size $B_r \times d$ each. Initialize $\mathbf{dK} = (0)_{N\times d}, \mathbf{dV} = (0)_{N\times d}$ in HBM and divide $\mathbf{dK}, \mathbf{dV}$ in to T_c blocks $\mathbf{dK}_1, \dots, \mathbf{dK}_{T_c}$ and $\mathbf{dV}_1, \dots, \mathbf{dV}_{T_c}$, of size $B_c \times d$ each.
for $1 \le j \le T_c$ **do**
  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
  Initialize $\tilde{\mathbf{dK}}_j = (0)_{B_c\times d}, \tilde{\mathbf{dV}}_j = (0)_{B_c\times d}$ on SRAM.
  for $1 \le i \le T_r$ **do**
    Load $\mathbf{Q}_i, \mathbf{O}_i, \mathbf{dO}_i, \mathbf{dQ}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
    On chip, compute $\mathbf{S}_{ij} = \tau \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r\times B_c}$.
    On chip, compute $\mathbf{S}_{ij}^{\text{masked}} = \text{MASK}(\mathbf{S}_{ij})$.
    On chip, compute $\mathbf{P}_{ij} = \text{diag}(l_i)^{-1}\exp(\mathbf{S}_{ij}^{\text{masked}} - m_i) \in \mathbb{R}^{B_r\times B_c}$.
    On chip, compute dropout mask $\mathbf{Z}_{ij} \in \mathbb{R}^{B_r\times B_c}$ where each entry has value $\frac{1}{1-p_{\text{drop}}}$ with probability $1 - p_{\text{drop}}$ and value 0 with probability p_{drop}.
    On chip, compute $\mathbf{P}_{ij}^{\text{dropped}} = \mathbf{P}_{ij} \circ \mathbf{Z}_{ij}$ (pointwise multiply).
    On chip, compute $\tilde{\mathbf{dV}}_j \leftarrow \tilde{\mathbf{dV}}_j + (\mathbf{P}_{ij}^{\text{dropped}})^\top \mathbf{dO}_i \in \mathbb{R}^{B_c\times d}$.
    On chip, compute $\mathbf{dP}_{ij}^{\text{dropped}} = \mathbf{dO}_i \mathbf{V}_j^\top \in \mathbb{R}^{B_r\times B_c}$.
    On chip, compute $\mathbf{dP}_{ij} = \mathbf{dP}_{ij}^{\text{dropped}} \circ \mathbf{Z}_{ij}$ (pointwise multiply).
    On chip, compute $D_i = \text{rowsum}(\mathbf{dO}_i \circ \mathbf{O}_i) \in \mathbb{R}^{B_r}$.
    On chip, compute $\mathbf{dS}_{ij} = \mathbf{P}_{ij} \circ (\mathbf{dP}_{ij} - D_i) \in \mathbb{R}^{B_r\times B_c}$.
    Write $\mathbf{dQ}_i \leftarrow \mathbf{dQ}_i + \tau \mathbf{dS}_{ij} \mathbf{K}_j \in \mathbb{R}^{B_r\times d}$ to HBM.
    On chip, compute $\tilde{\mathbf{dK}}_j \leftarrow \tilde{\mathbf{dK}}_j + \tau \mathbf{dS}_{ij}^\top \mathbf{Q}_i \in \mathbb{R}^{B_c\times d}$.
  end for
  Write $\mathbf{dK}_j \leftarrow \tilde{\mathbf{dK}}_j, \mathbf{dV}_j \leftarrow \tilde{\mathbf{dV}}_j$ to HBM.
end for
Return $\mathbf{dQ}, \mathbf{dK}, \mathbf{dV}$.

Block-Sparse FlashAttention

- Approximate attention
  - Masking specific block for S to reduce FLOPs
  - Θ(ND + N^2*d^2 * s/M) HBM accesses
  - s is the fraction of nonzero blocks in the block-sparsity mask, s can be sqrt(N)...

$$\mathbf{P} = \mathrm{softmax}(\mathbf{S} \odot \mathbb{1}_{\tilde{\mathbf{M}}}) \in \mathbb{R}^{N \times N},$$

Experiment

Table 2: GPT-2 small and medium using FlashAttention achieve up to 3× speed up compared to Huggingface implementation and up to 1.7× compared to Megatron-LM. Training time reported on 8×A100s GPUs.

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×)**

FlashAttention 2

- [Paper]
- Improvement
  - Better Iteration
  - Better usage of MatMul Acceleration
  - Better Parallelism

FlashAttention 2 - Fix

- O_i is the temporary rows of O for Q_i
- Why not lifting Q_i to outer loops?

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

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 2 - Online Softmax optimization

- Do not need to divide by L_i each time, only do it in the last time.

Algorithm 1 FlashAttention-2 forward pass

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

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 the output $\mathbf{O} \in \mathbb{R}^{N \times d}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, and divide the logsumexp L into T_r blocks $L_i, \ldots, L_{T_r}$ of size B_r each.

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

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

  On chip, initialize $\mathbf{O}_i^{(0)} = (0)_{B_r \times d} \in \mathbb{R}^{B_r \times d}, \ell_i^{(0)} = (0)_{B_r} \in \mathbb{R}^{B_r}, m_i^{(0)} = (-\infty)_{B_r} \in \mathbb{R}^{B_r}$.

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

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

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

    On chip, compute $m_i^{(j)} = \max(m_i^{(j-1)}, \mathrm{rowmax}(\mathbf{S}_i^{(j)})) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_i^{(j)} = \exp(\mathbf{S}_i^{(j)} - m_i^{(j)}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\ell_i^{(j)} = e^{m_i^{j-1} - m_i^{(j)}} \ell_i^{(j-1)} + \mathrm{rowsum}(\tilde{\mathbf{P}}_i^{(j)}) \in \mathbb{R}^{B_r}$.

    On chip, compute $\mathbf{O}_i^{(j)} = \mathrm{diag}(e^{m_i^{(j-1)} - m_i^{(j)}})^{-1} \mathbf{O}_i^{(j-1)} + \tilde{\mathbf{P}}_i^{(j)} \mathbf{V}_j$.

  end for

  On chip, compute $\mathbf{O}_i = \mathrm{diag}(\ell_i^{(T_c)})^{-1} \mathbf{O}_i^{(T_c)}$.

  On chip, compute $L_i = m_i^{(T_c)} + \log(\ell_i^{(T_c)})$.

  Write $\mathbf{O}_i$ to HBM as the i-th block of $\mathbf{O}$.

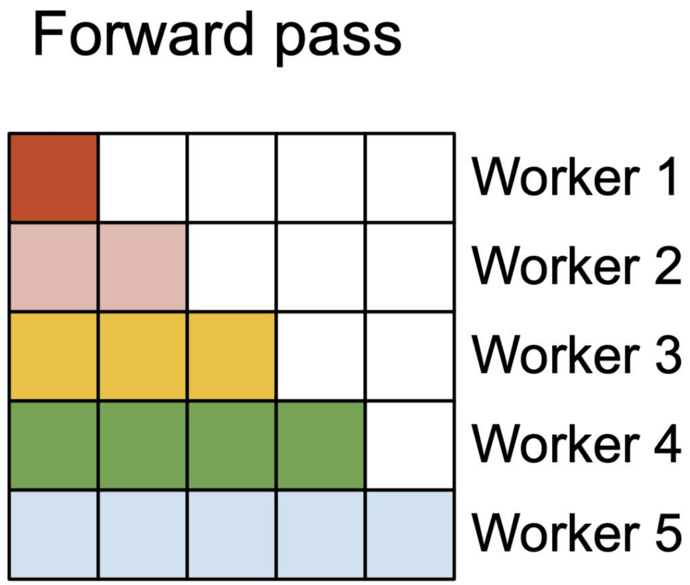
  Write L_i to HBM as the i-th block of L.

end for

Return the output $\mathbf{O}$ and the logsumexp L.

FlashAttention 2 - Parallelism

- FlashAttention 1: parallelizes over batch size and number of heads
- FlashAttention 2: Also over the length of input text.



FlashAttention 2 - Work Partitioning Between Warps

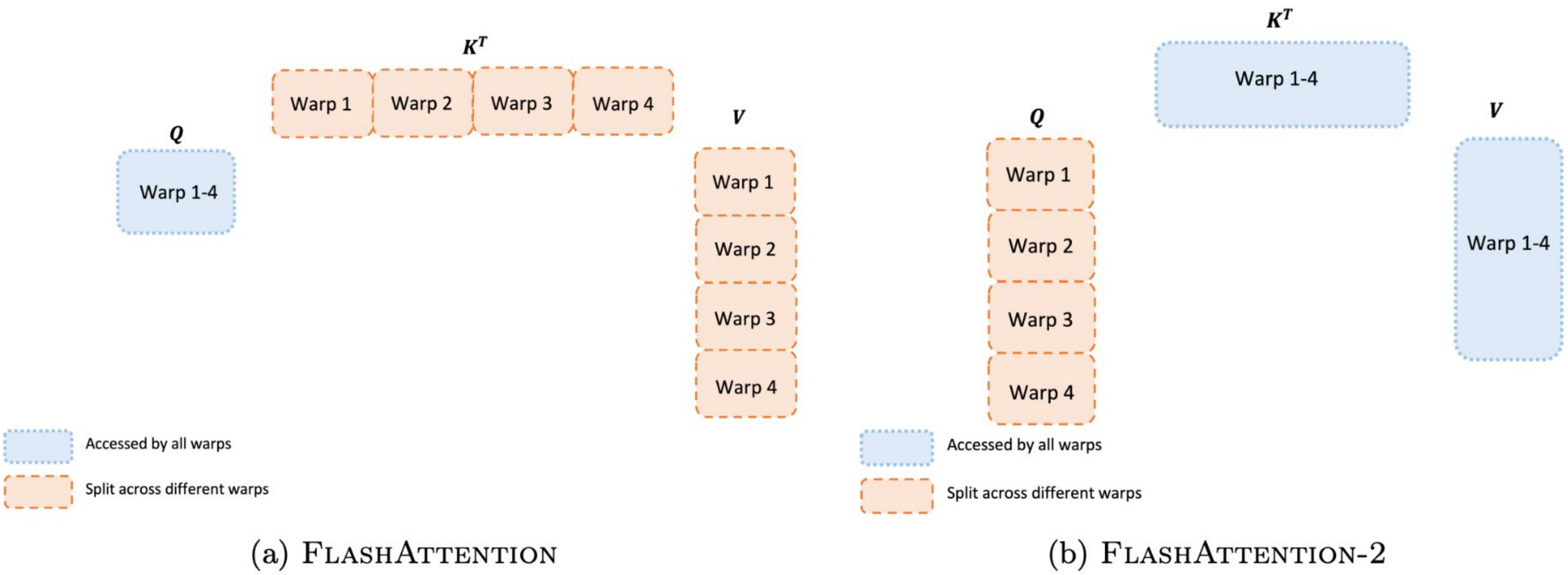
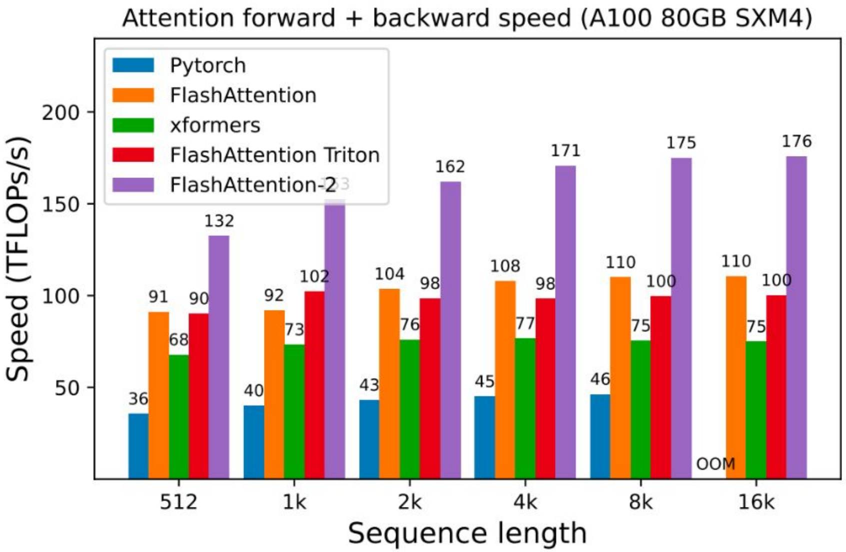
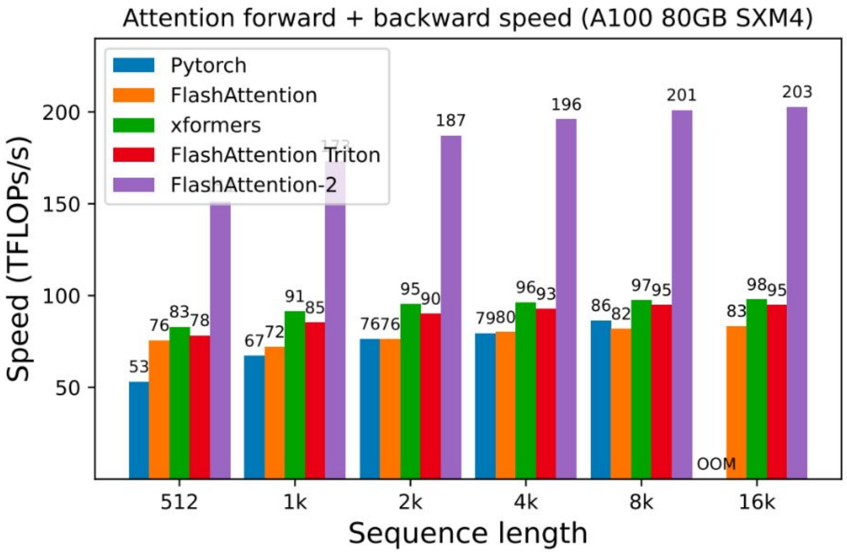


Figure 3: Work partitioning between different warps in the forward pass

FlashAttention 2 - Experiment





FlashAttention 3 (Flash-Decoding)

- Optimization on decoding/inference
- Why?
  - During inference, the query length is typically 1. So the partition of query length doesn’t work.
- Solution:
  - First, we split the keys/values in smaller chunks
  - We compute the attention of the query with each of these splits in parallel using FlashAttention. We also write 1 extra scalar per row and per split: the log-sum-exp of the attention values.
  - Finally, we compute the actual output by reducing over all the splits, using the log-sum-exp to scale the contribution of each split.
- Experiment
  - The up to 8x speedup end-to-end measured earlier is made possible because the attention itself is up to 50x faster than FlashAttention.

Reference

- FlashAttention
- FlashAttention V2
- https://medium.com/@sthanikamsanthosh1994/introduction-to-flash-attention-a-breakthrough-in-efficient-attention-mechanism-3eb47e8962c3
- https://huggingface.co/docs/text-generation-inference/en/conceptual/flash_attention
- From Online Softmax to FlashAttention
- FlashAttention V3