Fast and Memory-Efficient Exact Attention with IO-Awareness

NeurIPS 2022

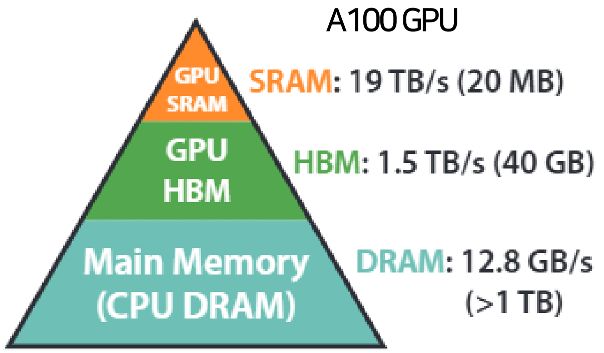
HUMANE Lab 김태균

2025.03.14

Introduction

- Sparse attention focuses on FLOP reduction but tends to ignore overheads from memory access (IO), resulting in lower wall-clock speed
- An IO-aware attention algorithm is needed to address this issue

=> Therefore, the goal is to reduce HBM access



Memory Hierarchy with Bandwidth & Memory Size

How to reduce HBM access?

1. Computing the softmax reduction
2. Not storing the large intermediate attention matrix for the backward pass

How to reduce HBM access?

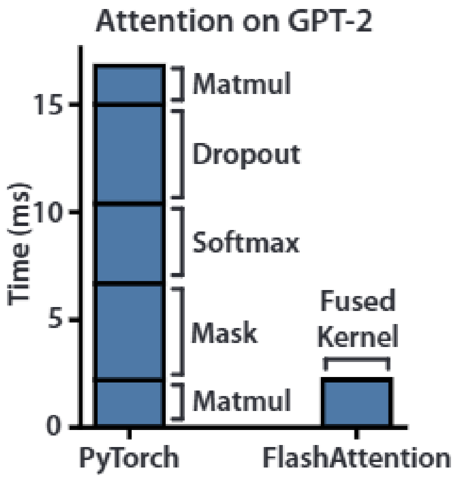
1. Computing the softmax reduction => Tiling
2. Not storing the large intermediate attention matrix for the backward pass => Recomputation

Background

- Operations
  - compute-bound
  - memory-bound

=> arithmetic intensity $\left(\frac{FLOPs}{Byte}\right)$

- Kernel fusion



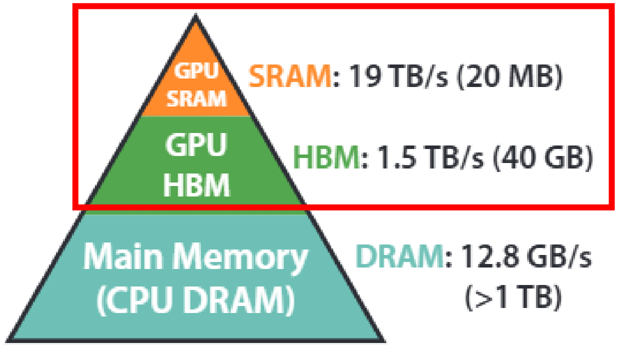
FlashAttention

- Goal : avoid r/w the attention matrix to and from HBM
  - Tiling
  - Recomputation

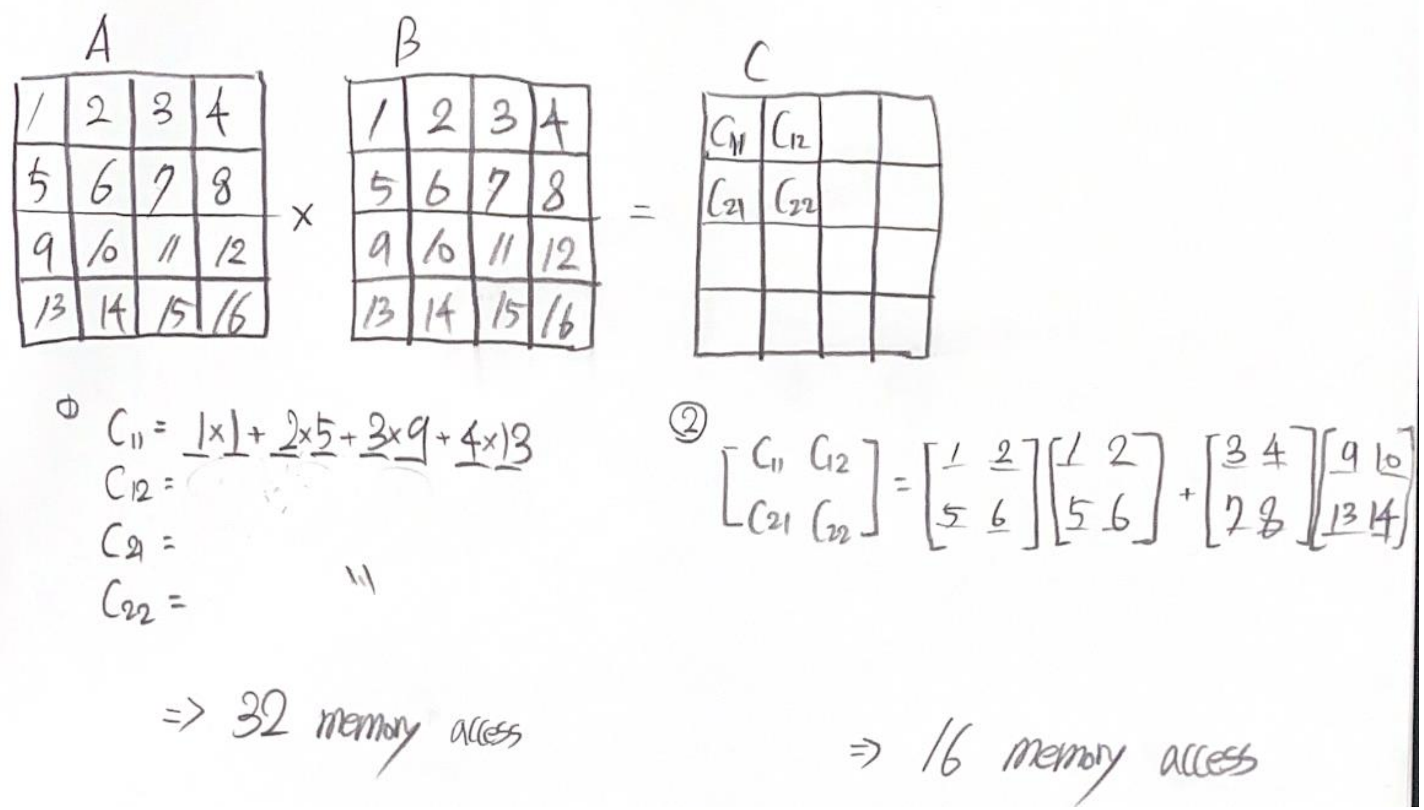
$$S = QK^T$$
$$P = softmax(S)$$
$$O = PV$$

Tiling

- A technique that divides the entire matrix into blocks for computation
  - memory hierarchy
  - matrix multiplication



Memory Hierarchy with
Bandwidth & Memory Size



Tiling

- A technique that divides the entire matrix into blocks for computation
  - memory hierarchy
  - matrix multiplication

$$\cancel{S = QK^T}$$
$$P = softmax(S)$$
$$O = PV$$

Tiling

- Safe softmax
  - subtract the maximum value to scale down
  - for numerical stability (to prevent overflow)

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{V} e^{x_j}}$$

Algorithm 1 Naive softmax

1: $d_0 \leftarrow 0$
2: **for** $j \leftarrow 1, V$ **do**
3: $\quad d_j \leftarrow d_{j-1} + e^{x_j}$
4: **end for**
5: **for** $i \leftarrow 1, V$ **do**
6: $\quad y_i \leftarrow \frac{e^{x_i}}{d_V}$
7: **end for**

memory access : 3V

$$y_i = \frac{e^{x_i - \max_{k=1}^{V} x_k}}{\sum_{j=1}^{V} e^{x_j - \max_{k=1}^{V} x_k}}$$

Algorithm 2 Safe softmax

1: $m_0 \leftarrow -\infty$
2: **for** $k \leftarrow 1, V$ **do**
3: $\quad m_k \leftarrow \max(m_{k-1}, x_k)$
4: **end for**
5: $d_0 \leftarrow 0$
6: **for** $j \leftarrow 1, V$ **do**
7: $\quad d_j \leftarrow d_{j-1} + e^{x_j - m_V}$
8: **end for**
9: **for** $i \leftarrow 1, V$ **do**
10: $\quad y_i \leftarrow \frac{e^{x_i - m_V}}{d_V}$
11: **end for**

memory access : 4V

Online normalizer calculation for softmax, arXiv preprint 2018

Tiling

- Online safe softmax
  - compute m online
  - reduce the the number of memory accesses by one

Algorithm 3 Safe softmax with online normalizer calculation

```
m_0 ← −∞
d_0 ← 0
for j ← 1, V do
    m_j ← max(m_{j−1}, x_j)
    d_j ← d_{j−1} × e^{m_{j−1} − m_j} + e^{x_j − m_j}
end for
for i ← 1, V do
    y_i ← e^{x_i − m_V} / d_V
end for
```

memory access : 3V

$d_1 = e^{x_1-m_1}$

$d_2 = e^{x_1-m_1} e^{m_1-m_2} + e^{x_2-m_2}$

$= e^{x_1-m_2} + e^{x_2-m_2}$

$d_3 = \left(e^{x_1-m_2} + e^{x_2-m_2}\right) e^{m_2-m_3} + e^{x_3-m_3}$

$= e^{x_1-m_3} + e^{x_2-m_3} + e^{x_3-m_3}$

$\vdots$

Can be proven by mathematical induction

Online normalizer calculation for softmax, arXiv preprint 2018

Tiling

- Online safe softmax
  - compute m online
  - reduce the number of memory accesses by one

Algorithm 3 Safe softmax with online normalizer calculation

1: $m_0 \leftarrow -\infty$
2: $d_0 \leftarrow 0$
3: **for** $j \leftarrow 1, V$ **do**
4: $\quad m_j \leftarrow \max(m_{j-1}, x_j)$
5: $\quad d_j \leftarrow d_{j-1} \times e^{m_{j-1}-m_j} + e^{x_j - m_j}$
6: **end for**
7: **for** $i \leftarrow 1, V$ **do**
8: $\quad y_i \leftarrow \frac{e^{x_i - m_V}}{d_V}$
9: **end for**

memory access : 3V

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

Online normalizer calculation for softmax, arXiv preprint 2018

Tiling

- Online safe softmax
  - compute m online
  - reduce the number of memory accesses by one

$$\cancel{S = QK^T}$$

$$\cancel{P = softmax(S)}$$

$$O = PV$$

Online normalizer calculation for softmax, arXiv preprint 2018

Tiling

$$\text{for } 1 \le i \le N$$
$$x_i = q K_i^T$$
$$m_i = \max(m_{i-1}, x_i)$$
$$d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$$
$$\text{for } 1 \le i \le N$$
$$a_i = e^{x_i - m_N} / d'_N$$
$$o_i = o_{i-1} + a_i V_i$$
$$\text{return } O_N$$

where N is the number of blocks

$$O_i = \sum_{j=1}^{i} \frac{e^{x_j - m_N}}{d'_N} V_j$$

trick

$$O'_i = \sum_{j=1}^{i} \frac{e^{x_j - m_i}}{d'_i} V_j$$
$$= \sum_{j=1}^{i-1} \frac{e^{x_j - m_i}}{d'_i} \cdot \left[\frac{d'_{i-1}}{d'_{i-1}} \cdot \frac{e^{-m_{i-1}}}{e^{m_{i-1}}} \right] V_j + \frac{e^{x_i - m_i}}{d'_i} V_i$$
$$= \left(\sum_{j=1}^{i-1} \frac{e^{x_j - m_{i-1}}}{d'_{i-1}} V_j \right) \cdot \frac{d'_{i-1}}{d'_i} \cdot e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d'_i} V_i$$
$$= O'_{i-1} \frac{d'_{i-1}}{d'_i} e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d'_i} V_i$$

remove dependency

Tiling

for $1 \le i \le N$

$$x_i = q K_i^T$$

$$m_i = \max(m_{i-1}, x_i)$$

$$d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$$

for $1 \le i \le N$

$$a_i = e^{x_i - m_N} / d'_N$$

$$o_i = o_{i-1} + a_i V_i$$

return o_N

fuse into a single loop

for $1 \le i \le N$

$$x_i = q K_i^T$$

$$m_i = \max(m_{i-1}, x_i)$$

$$d'_i = d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$$

$$o'_i = o'_{i-1} \frac{d'_{i-1}}{d'_i} e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d'_i} V_i$$

return o'_N

Tiling

for $1 \le i \le N$

$x_i = qK_i^T$

$m_i = \max(m_{i-1}, x_i)$

$d'_i = d'_{i-1} e^{m_{i-1}-m_i} + e^{x_i - m_i}$

for $1 \le i \le N$

$a_i = e^{x_i - m_N} / d'_N$

$O_i = O_{i-1} + a_i V_i$

return O_N

fuse into a single loop

$$\cancel{S = QK^T}$$

$$\cancel{P = softmax(S)}$$

$$\cancel{O = PV}$$

for $1 \le i \le N$

$x_i = qK_i^T$

$m_i = \max(m_{i-1}, x_i)$

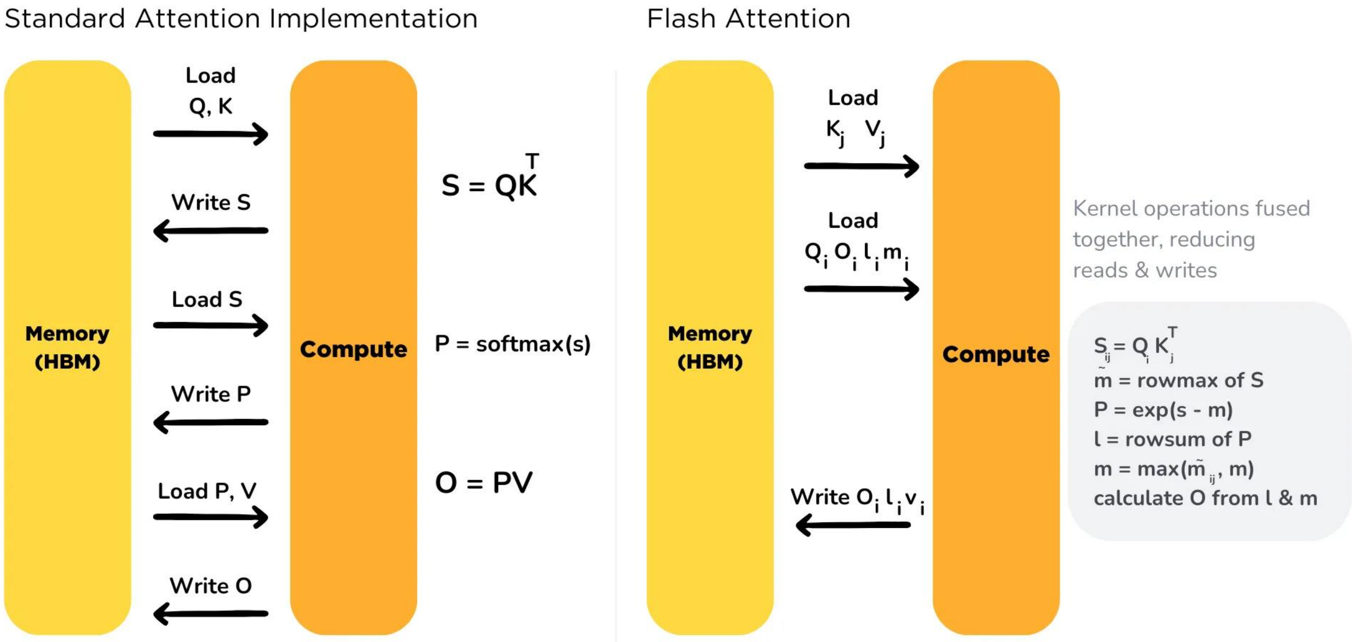
$d'_i = d'_{i-1} e^{m_{i-1}-m_i} + e^{x_i - m_i}$

$O'_i = O'_{i-1} \frac{d'_{i-1}}{d'_i} e^{m_{i-1}-m_i} + \frac{e^{x_i - m_i}}{d'_i} V_i$

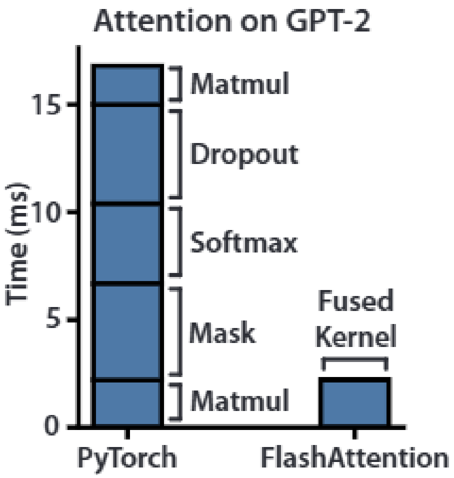
return O'_N

Recomputation

- A technique of recompute intermediate values (S, P) during the backward pass instead of storing them



Initialize O, l and m matrices with zeroes. m and l are used to calculate cumulative softmax. Divide Q, K, V into blocks (due to SRAM's memory limits) and iterate over them, for i is row & j is column.



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

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

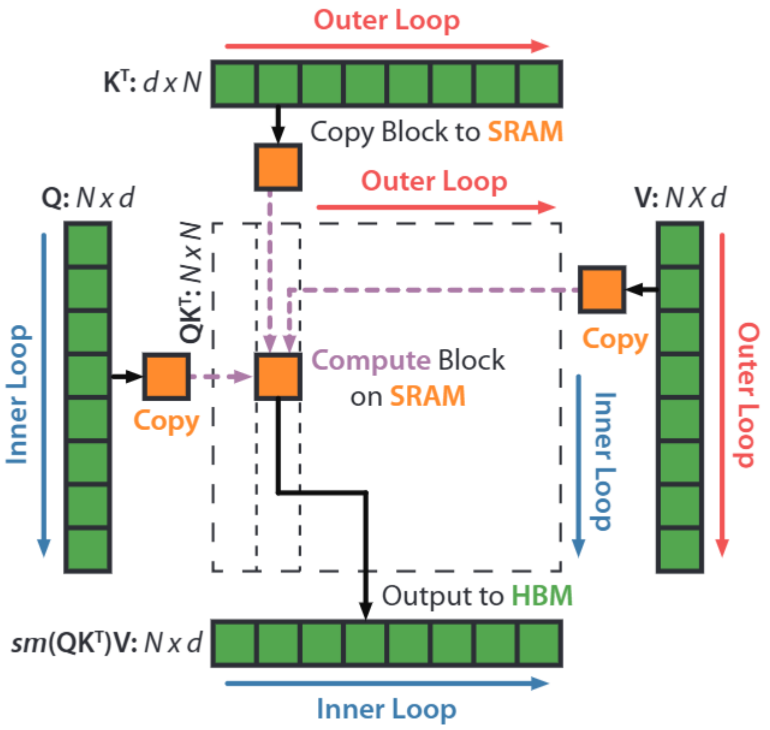
    Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\text{new}})^{-1} (\mathrm{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.

    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

Block-sparse FlashAttention

- Better IO complexity than FlashAttention by a factor proportional to the sparsity ratio
  - Given a predefined block sparsity mask M

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{N\times N}, \quad \mathbf{P} = \mathrm{softmax}(\mathbf{S} \odot \mathbb{1}_{\tilde{\mathbf{M}}}) \in \mathbb{R}^{N\times N}, \quad \mathbf{O} = \mathbf{P}\mathbf{V} \in \mathbb{R}^{N\times d}$$

Experiments

- Training time & model accuracy

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×)**

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×)

Experiments

- Attention runtime & memory benchmarks

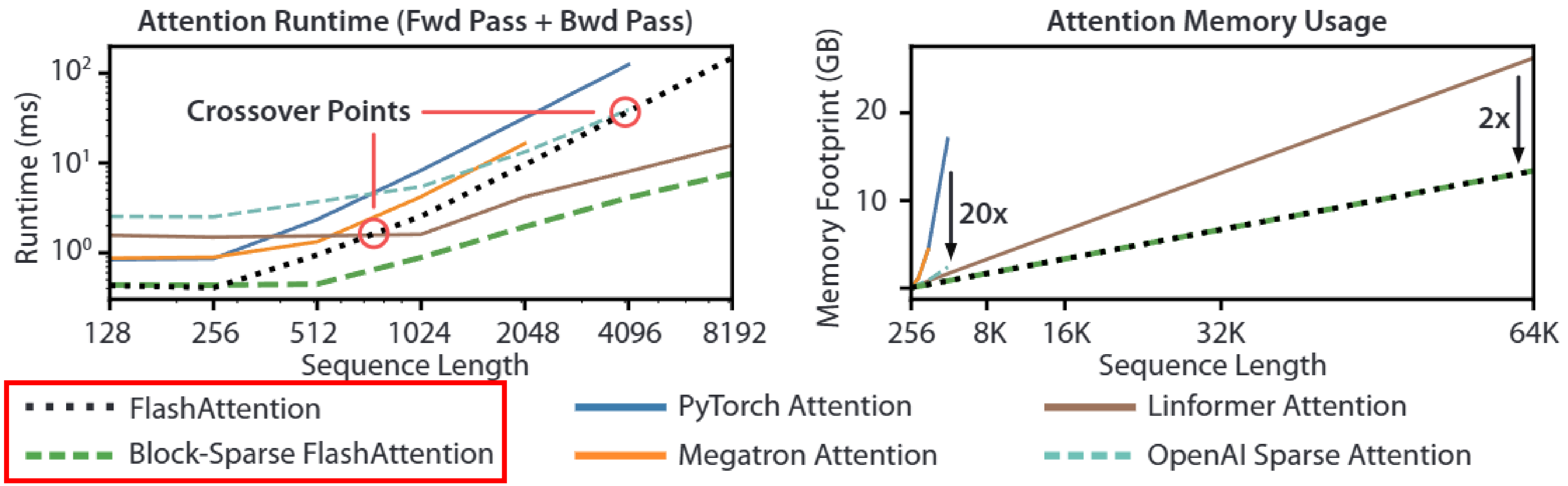


Figure 3: **Left:** runtime of forward pass + backward pass. **Right:** attention memory usage.

Conclusion

- Showing significant reduction in HBM accesses compared to standard attention