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

Tri Dao[†], Daniel Y. Fu[†], Stefano Ermon[†], Atri Rudra[‡], and Christopher Ré[†]

[†]Department of Computer Science, Stanford University
[‡]Department of Computer Science and Engineering, University at Buffalo, SUNY

{trid,danfu}@cs.stanford.edu, ermon@stanford.edu, atri@buffalo.edu,
chrismre@cs.stanford.edu

June 24, 2022

- 论文地址：https://arxiv.org/pdf/2205.14135.pdf
- 实现地址：https://github.com/HazyResearch/flash-attention

Content

Introduction

- Transformers have grown **larger and deeper**, but equipping them with **longer context** remains difficult since the self-attention module at their heart has time and memory complexity quadratic in sequence length.

- **Approximate attention methods**（sparse-approximation、low-rank approximation、and their combinations...）做法是：by trading off model quality to reduce the compute complexity aiming to reduce the compute and memory requirements of attention，效果是：reduce the compute requirements to linear or near-linear in sequence length，但是分析其缺陷在于they focus on **FLOP reduction** (which may not correlate with **wall-clock** speed) and **tend to ignore** overheads from **memory access** (IO).

Flops：每秒所执行的浮点数运算次数。

wall-clock time：进程从开始执行到完成所经历的完整的墙上时钟时间（wall clock）时间。

Motivation

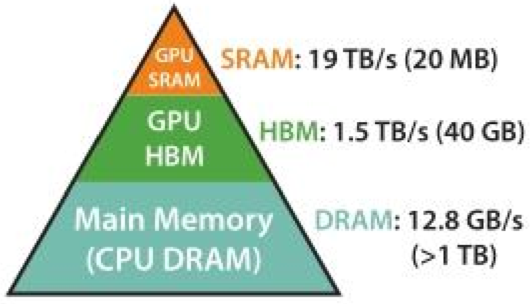
- **Transformer缺点：**Transformers are **slow** and **memory-hungry** on long sequences
- **原因分析：**since the time and memory complexity of self-attention are **quadratic**（二次方的）in sequence length.
- An important question is whether making **attention faster and more memory-efficient** can help Transformer models address their runtime and memory challenges for long sequences.
- 作者分析想要**wall-clock speedup**的关键原则在于：making attention algorithms **IO-aware**—accounting for reads and writes between levels of GPU memory。原因是：On modern GPUs, compute speed has **out-paced** memory speed, and most operations in Transformers are **bottlenecked by memory accesses**, when reading and writing data can account for a large portion of the runtime.

IO-aware：也即具有IO感知的。能考虑到I/O操作（在GPU指GPU内存级别的读写操作）

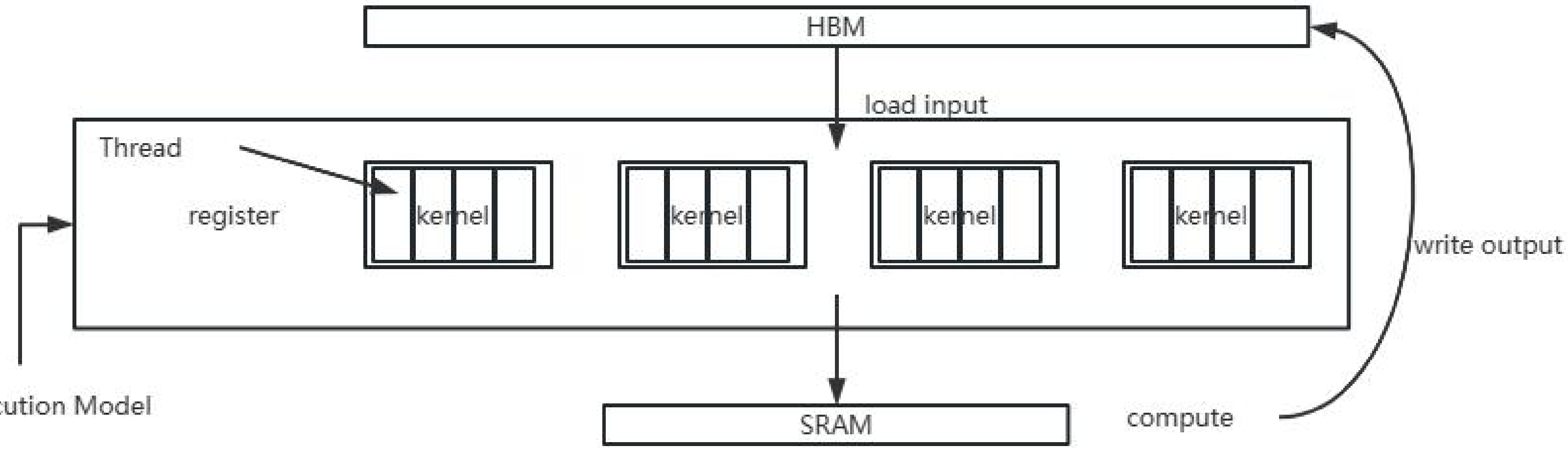
Background

- Hardware Performance
- Standard Attention Implementation

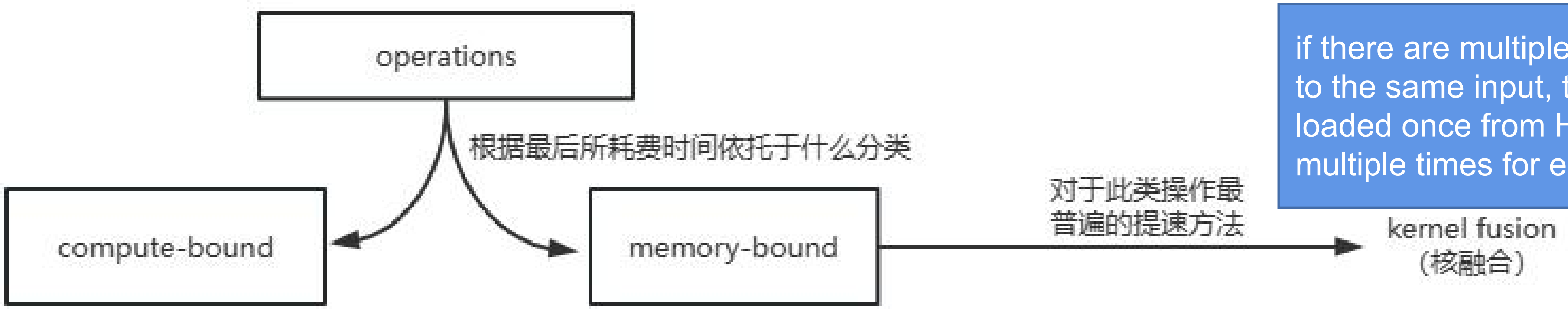
Hardware Performance



Memory Hierarchy with Bandwidth & Memory Size



GPU Execution Model



if there are multiple operations applied to the same input, the input can be loaded once from HBM, instead of multiple times for each operation.

matrix multiply with large inner dimension

elementwise (e.g., activation, dropout), and reduction (e.g., sum, softmax, batch norm, layer norm)

Standard Attention Implementation

Given input sequences $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ where N is the sequence length and d is the head dimension, we want to compute the attention output $\mathbf{O} \in \mathbb{R}^{N \times d}$:

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

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 $\mathbf{O}$.

Method

- Algorithm： Tiling&Recomputation
- Analysis： IO Complexity
- Extensions： Block-Sparse FlashAttention

Algorithm

FlashAttention旨在避免从高带宽显存HBM中读取和写入注意力矩阵（也即**减少HBM access**），这需要做到:

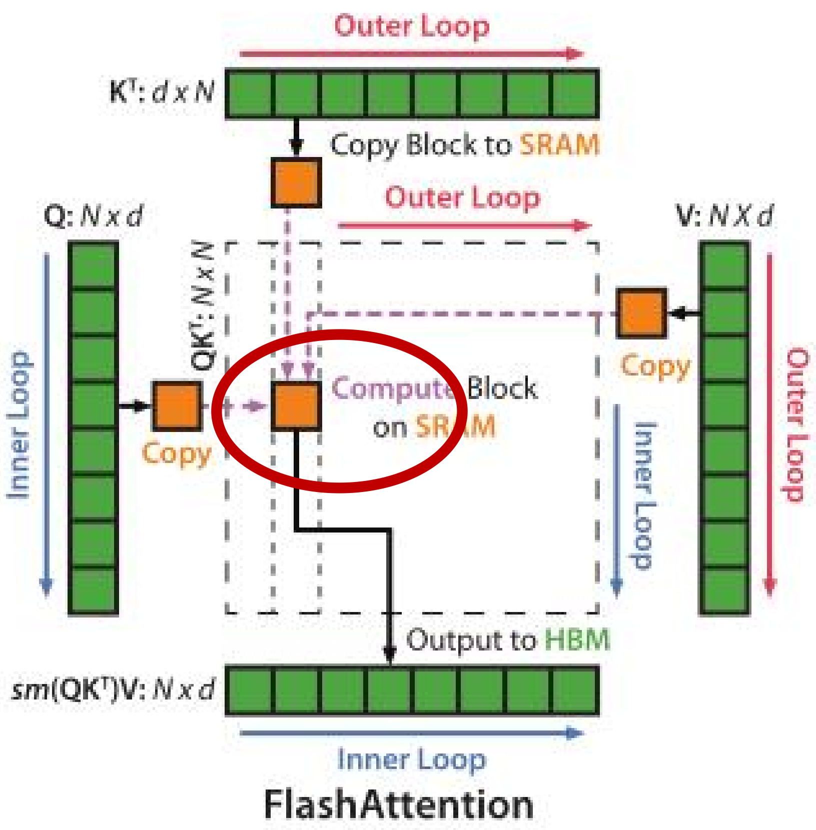
1. 在**不访问整个**输入的情况下计算softmax函数的缩减，而是使用spilt into block的技术**分块**计算。

2. **不存储**用于反向传播的大的**中间注意力矩阵。**

tiling技术：是减少GPU HBM访问次数的一种常见的优化策略。实现思路是将数据分片，然后将每个小分片缓存到访存速度更快的SRAM中，通过利用GPU上的SRAM来减少对HBM的访问，以提高核函数的执行效率。

重计算（Recomputation）：本质上是一种用时间换空间的策略，可以将它类比成一种 Tensor 缓存（Cache）策略，当显存空间不足时，可以选择把一些前向计算的结果清除；当需要再次用到这些计算结果时，再根据之前缓存的检查点（Checkpoint）去重新计算它们。

Tiling



针对上述第一点：

- Standard Attention中由于要计算softmax函数，而softmax函数都是按行计算的，按照这个逻辑，在和V做矩阵乘法之前，需要让Q、K的各个分块完成**整一行分块**的计算。这会导致程序运行的过程中出现对HBM中同一个位置的多次重复读取，**内存消耗非常大**，在得到softmax结果之后，再和矩阵V分块做矩阵乘。

- 如左图所示，Flash Attention中作者考虑到这一点，避免进行整行的读入写入来计算，将输入**分割成block块**，并在输入块上进行**多次传递**，从而实现**以增量的方式**执行softmax缩减。这样就大大**加快**了计算**运行的速度**。

Tiling

这里根据Standard Attention的softmax函数计算公式，推导分块后的使用**缩放分解**大的softmax函数的方法：

$$m(x) := \max_i \ x_i, \quad f(x) := \begin{bmatrix} e^{x_1 - m(x)} & \ldots & e^{x_B - m(x)} \end{bmatrix}, \quad \ell(x) := \sum_i f(x)_i, \quad \mathrm{softmax}(x) := \frac{f(x)}{\ell(x)}.$$

For vectors $x^{(1)}, x^{(2)} \in \mathbb{R}^B$, we can decompose the softmax of the concatenated $x = \begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix} \in \mathbb{R}^{2B}$ as:

$$m(x) = m(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = \max(m(x^{(1)}), m(x^{(2)})), \quad f(x) = \begin{bmatrix} e^{m(x^{(1)})-m(x)} f(x^{(1)}) & e^{m(x^{(2)})-m(x)} f(x^{(2)}) \end{bmatrix},$$

$$\ell(x) = \ell(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = e^{m(x^{(1)})-m(x)} \ell(x^{(1)}) + e^{m(x^{(2)})-m(x)} \ell(x^{(2)}), \quad \mathrm{softmax}(x) = \frac{f(x)}{\ell(x)}.$$

Tiling

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.
```

将注意力矩阵QKV划分为块

```
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 $\ell$ 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 $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
```

计算额外统计数据

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

组合两个结果

```
        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 $\mathbf{O}$.
```

Recomputation

针对上述第二点:

- Standard Attention中**存储**了每一步的**中间注意力矩阵**（S、P）。这两个矩阵由于更新QKV参数方面的需求在反向传播的时候是需要的，但由于这两个矩阵的长度与输入序列长度N相关（$O(N^2)$），序列长度越大，矩阵大小就越大，对时间和内存的耗费也就越大。

- FlashAttention并**不存储中间注意力矩阵**。而是**存储**前向传递输出O和**softmax归一化因子（m、� ）**，（$O(N)$）。通过这些可以很方便地在反向传递中根据存储在SRAM中的QKV分块快速地**重新计算（recoputation）**S、P这两个矩阵，这比从HBM上读取和写入完整的中间注意矩阵的标准方法更快，占用的内存也更少。

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

Analysis

We analyze its IO complexity,showing significant reduction in HBM accesses compared to standard attention.

- FlashAttention requires $\Theta(N^2 d^2 M^{-1})$ HBM accesses (d = head dimension , M = size of SRAM , N = seq len) as compared to $\Theta(Nd + N^2)$ of StandardAttention for typical values of d (64-128) and M (around 100KB), d^2 is many times smaller than M)

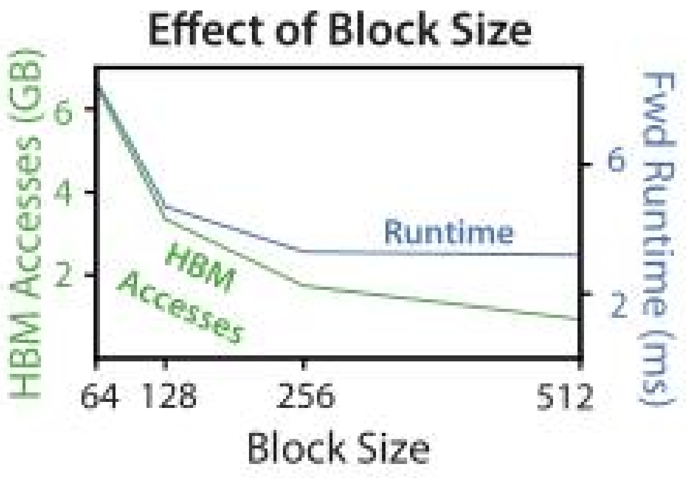
Theorem 2. *Let N be the sequence length, d be the head dimension, and M be size of SRAM with $d \leq M \leq Nd$. Standard attention (Algorithm 0) requires $\Theta(Nd + N^2)$ HBM accesses, while* FlashAttention *(Algorithm 1) requires $\Theta(N^2 d^2 M^{-1})$ HBM accesses.*

- We prove a lower-bound: one cannot asymptotically improve on the number of HBM accesses for all values of M (the SRAM size) when computing exact attention.

Proposition 3. *Let N be the sequence length, d be the head dimension, and M be size of SRAM with $d \leq M \leq Nd$. There does not exist an algorithm to compute exact attention with $o(N^2 d^2 M^{-1})$ HBM accesses for all M in the range $[d, Nd]$.*

Analysis

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



- 对比标准的Attention机制，FlashAttention虽然由于向后传播需要重新计算而导致了GFLOPs浮点数运算次数的增加，但是FlashAttention对HBM的读写和运行时间都有了显著的提高。

- 随着块的增大，HBM所读写的次数和运行时间都在不断地减少（由于对输入的块传递次数减少了）。而对于足够大的块大小，runtime主要受其他因素影响（例如：算术操作）。

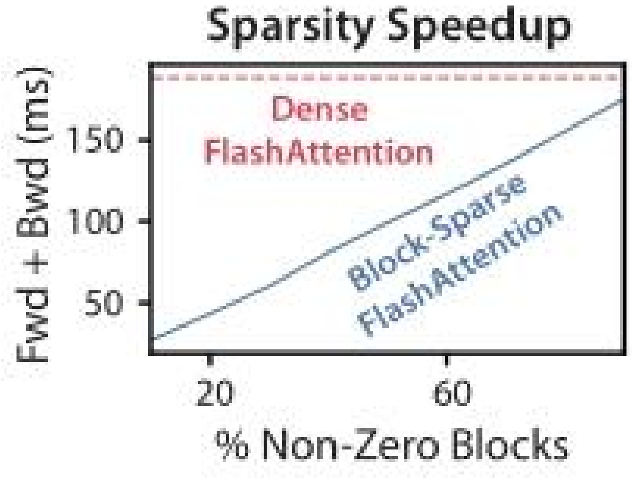
Extensions

- Propose **block-sparse FlashAttention**, whose IO complexity is smaller than FlashAttention by a factor **proportional to the sparsity.**

- Analyze the IO complexity of block-sparse FlashAttention.

Proposition 4. *Let N be the sequence length, d be the head dimension, and M be size of SRAM with $d \leq M \leq Nd$. Block-sparse* FlashAttention *(Algorithm $\boxed{5}$) requires $\Theta(Nd + N^2 d^2 M^{-1} s)$ HBM accesses where s is the fraction of nonzero blocks in the block-sparsity mask.*

- Validate that as the sparsity increases, the runtime of block-sparse FlashAttention improves proportionally.



Experiment

- Training Speed
- Quality
- Benchmarking Attention

Training Speed

- BERT

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

- GPT-2

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×)**

- Long-range Arena

Models	ListOps	Text	Retrieval	Image	Pathfinder	Avg	Speedup
Transformer	36.0	63.6	81.6	42.3	72.7	59.3	-
FlashAttention	37.6	63.9	81.4	43.5	72.7	59.8	2.4×
Block-sparse FlashAttention	37.0	63.0	81.3	43.6	73.3	59.6	**2.8**×
Linformer [84]	35.6	55.9	77.7	37.8	67.6	54.9	2.5×
Linear Attention [50]	38.8	63.2	80.7	42.6	72.5	59.6	2.3×
Performer [12]	36.8	63.6	82.2	42.1	69.9	58.9	1.8×
Local Attention [80]	36.1	60.2	76.7	40.6	66.6	56.0	1.7×
Reformer [51]	36.5	63.8	78.5	39.6	69.4	57.6	1.3×
Smyrf [19]	36.1	64.1	79.0	39.6	70.5	57.9	1.7×

Quality

- Language Modeling with Long Context

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×)

- Long Document Classification

	512	1024	2048	4096	8192	16384
MIMIC-III [47]	52.8	50.7	51.7	54.6	56.4	**57.1**
ECtHR [6]	72.2	74.3	77.1	78.6	**80.7**	79.2

- Path-X and Path-256

Model	Path-X	Path-256
Transformer	✗	✗
Linformer [84]	✗	✗
Linear Attention [50]	✗	✗
Performer [12]	✗	✗
Local Attention [80]	✗	✗
Reformer [51]	✗	✗
SMYRF [19]	✗	✗
FlashAttention	**61.4**	✗
Block-sparse FlashAttention	56.0	**63.1**

Benchmarking Attention

- Runtime(left) & Memory Footprint(right)

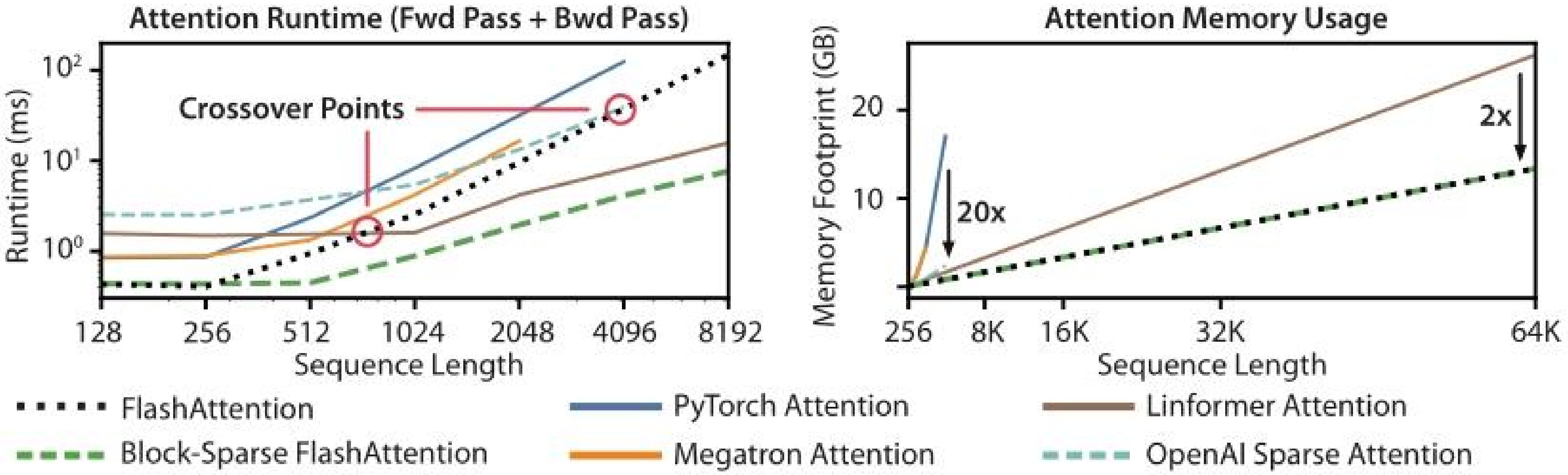


Figure 3: **Left:** runtime of forward pass + backward pass. **Right:** attention memory usage.

Limitations and Future Directions

- Compiling to CUDA
- IO-Aware Deep Learning
- Multi-GPU IO-Aware Methods

Thanks！