Self attention is quadratic memory and time complexity wrt sequence length

- Approximate attention methods attempt to address this by trading model quality to reduce compute complexity but do not achieve wall clock speedup
- Make attention algorithms IO- aware: accounting for reads and writes between levels of GPU memory
    - Tiling to reduce number of reads/ writes between GPU high bandwidth memory (slow) and GPU SRAM (fast)
    - Linear wrt sequence length
    - Enable longer contexts in Transformers -> better performance and speedup

Sparse approximation and low rank approximation reduce compute requirements to linear or near linear in sequence length but do not display wall clock speedup

- Focus on FLOP reduction (may not correlate with wall clock speed) and ignore overheads from memory access

Does not read and write large N x N attention matrix to HBM

- Computing softmax reduction without access to the whole input
- Not storing the large intermediate attention matrix for the backward

It does this by

- Split input into blocks and make several passes over input blocks, incrementally performing softmax reduction (tiling)
- Store softmax normalization factor to recompute attention on chip in backward pass
    - Faster than standard approach of reading intermediate attention matrix from HBM

Process

- Loops through blocks of K and V matrix and loads them into fast SRAM
- Loops over Q, loading them to SRAM and writing output to HBM

Tried this method on both normal transformers and block- sparse

Compute is increasingly being bottlenecked by memory (HBM) accesses, exploiting fast SRAM is important

Compute bound (convolution, matrix multiply), memory bound (activation, batch norm)

Kernel fusion: 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.

- Compilers can automatically fuse many operations but for model training the intermediate values still need to be written to HBM

Tiling: attention by blocks because x = [x1 x2]

- If we keep track of extra statics m(x), l(x) we can compute softmax one block at a time

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

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

Recomputation: storing output O and softmax normalization statistics (m(x), l(x)) we don't need to store S, P and could be computed instead (gradient checkpointing)

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 $\mathbf{P} = \text{softmax}(\mathbf{S})$, 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 $\mathbf{O}$.

We validate that the number of HBM accesses is the main determining factor of attention run-time. In Fig. 2 (left), we see that even though FLASHATTENTION has higher FLOP count compared to standard attention (due to recomputation in the backward pass), it has much fewer HBM accesses, resulting in much faster runtime. In Fig. 2 (middle), we vary the block size B_c of FLASHATTENTION, which results in different amounts of HBM accesses, and measure the runtime of the forward pass. As block size increases, the number of HBM accesses decreases (as we make fewer passes over the input), and runtime decreases. For large enough block size (beyond 256), the runtime is then bottlenecked by other factors (e.g., arithmetic operations). Moreover, larger block size will not fit into the small SRAM size.

Better than random performance on Path- X task (sequence length 16K)
Block sparse (BIG BIRD) can achieve good results on Path- 256 (Sequence 64K)

Why is it not quadratic time?

Original is quadratic to sequence length
In this work they claim linear to sequence length

- Is it because they make it linear wrt blocks so sequence length is negligible? No
- If M = Nd then it is technically linear to sequence length
    - Iterating through each of these is negligible because of SRAM
    - Wall clock time is faster because of SRAM but more FLOPs
- So lowkey a cheat because they use an intermediary