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

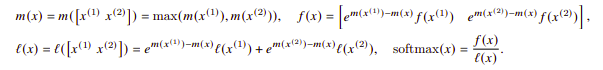
Compute bound (convolution, matrix multiply), memory bound (activation, batch norm)

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.

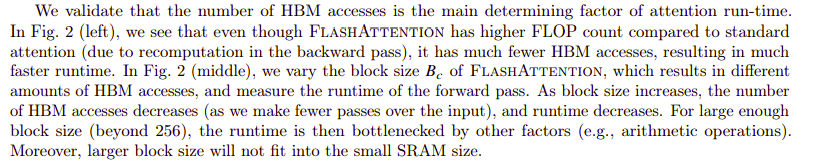
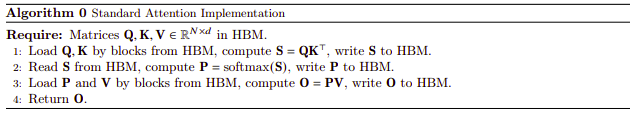
* Compilers can automatically fuse many operations but for model training the intermediate values still need to be written to HBM

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



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)



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