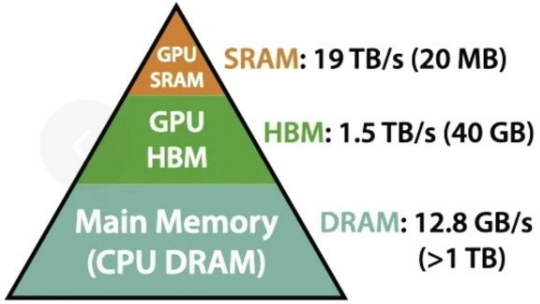
FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

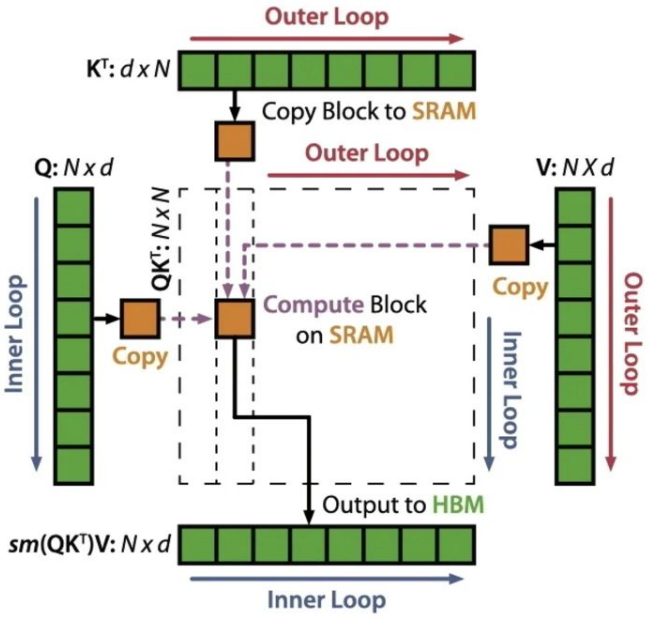
Tri Dao, Dan Fu ({trid, danfu}@cs.stanford.edu)
7/23/22 HAET Workshop @ ICML 2022

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Ruda, Christopher Ré. Flash Attention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *arXiv preprint arXiv:2205.14135.* https://github.com/HazyResearch/flash-attention.

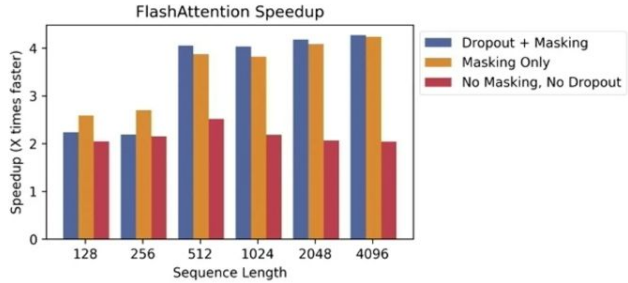
Attention: IO-Aware Tiling Yields Speed, Mem Savings, & Quality

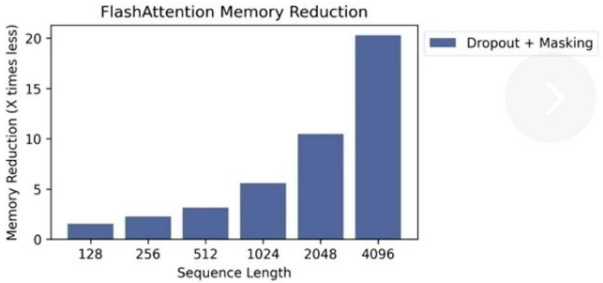


Memory Hierarchy with Bandwidth & Memory Size

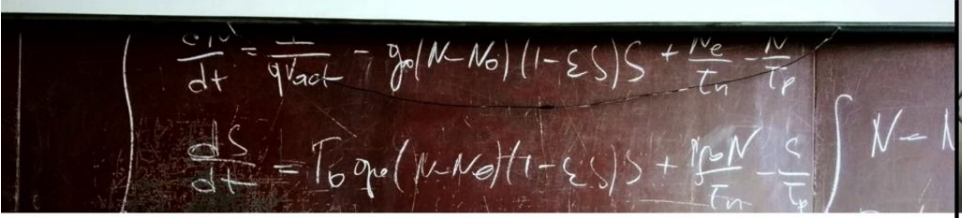


FlashAttention



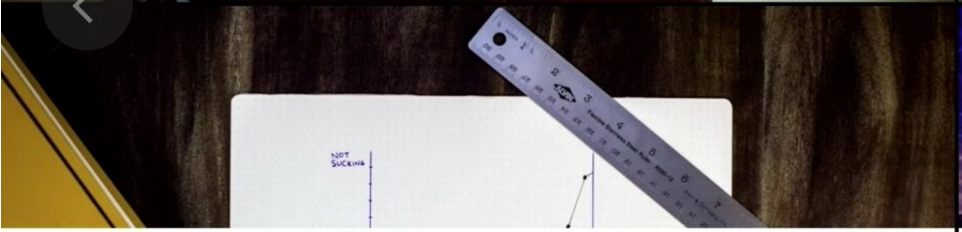


Model	Path-X	Path-256
Transformer	✗	✗
Linformer [81]	✗	✗
Linear Attention [48]	✗	✗
Performer [11]	✗	✗
Local Attention [77]	✗	✗
Reformer [49]	✗	✗
SMYRF [18]	✗	✗
FlashAttention	**61.4**	✗
Block-sparse FlashAttention	56.0	**63.1**



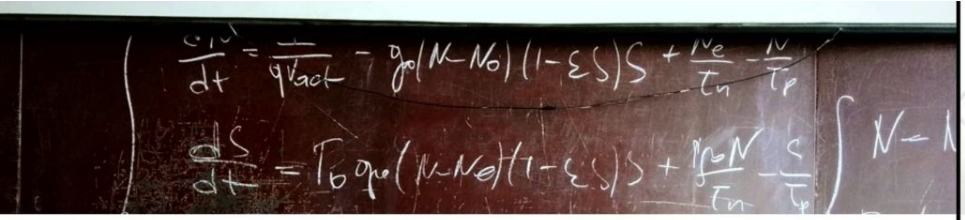
Background

FlashAttention



Empirical Validation

Future Directions



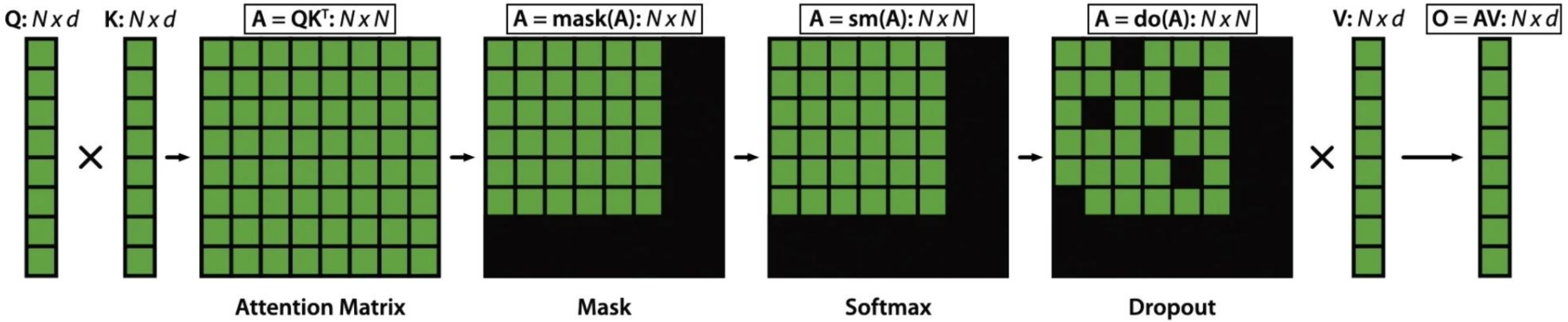
Background

FlashAttention

Empirical Validation

Future Directions

Background: Attention is Bottlenecked by Memory Reads/Writes



$$\mathbf{O} = \text{Dropout}(\text{Softmax}(\text{Mask}(\mathbf{QK}^T)))\mathbf{V}$$

Naive implementation requires repeated R/W from slow GPU HBM

DATA MOVEMENT IS ALL YOU NEED: A CASE STUDY ON OPTIMIZING TRANSFORMERS

Andrei Ivanov [*1] Nikoli Dryden [*1] Tal Ben-Nun [1] Shigang Li [1] Torsten Hoefler [1]

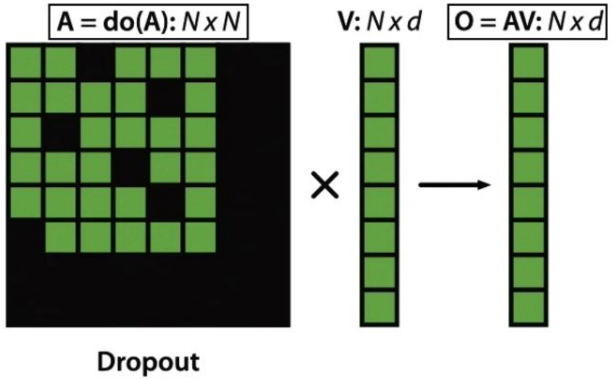
ABSTRACT

Transformers are one of the most important machine learning workloads today. Training one is a very compute-intensive task, often taking days or weeks, and significant attention has been given to optimizing transformers. Despite this, existing implementations do not efficiently utilize GPUs. We find that data movement is the key bottleneck when training. Due to Amdahl's Law and massive improvements in compute performance, training has now become memory-bound. Further, existing frameworks use suboptimal data layouts. Using these insights, we present a recipe for globally optimizing data movement in transformers. We reduce data movement by up to 22.91% and overall achieve a 1.30× performance improvement over state-of-the-art frameworks when training a BERT encoder layer and 1.19× for the entire BERT. Our approach is applicable more broadly to optimizing deep neural networks, and offers insight into how to tackle emerging performance bottlenecks.

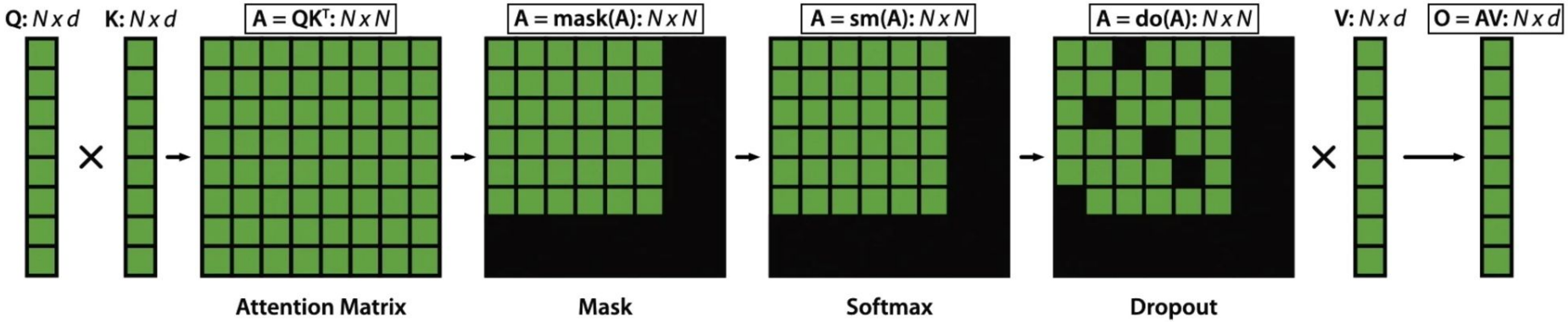
$$\mathbf{O} = \text{Dropout}(\text{Softmax}(\text{Mask}(\mathbf{QK}^T)))\mathbf{V}$$

Naive implementation requires repeated R/W from slow GPU HBM

lemory Reads/Writes

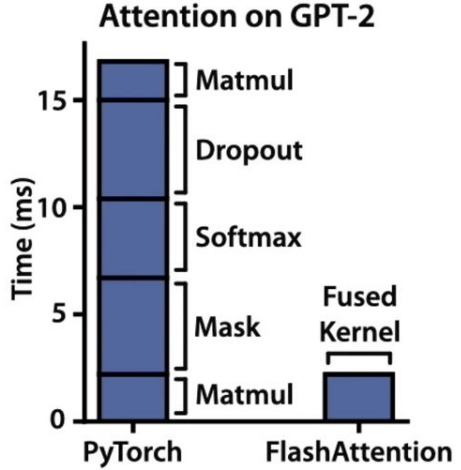


Background: Attention is Bottlenecked by Memory Reads/Writes

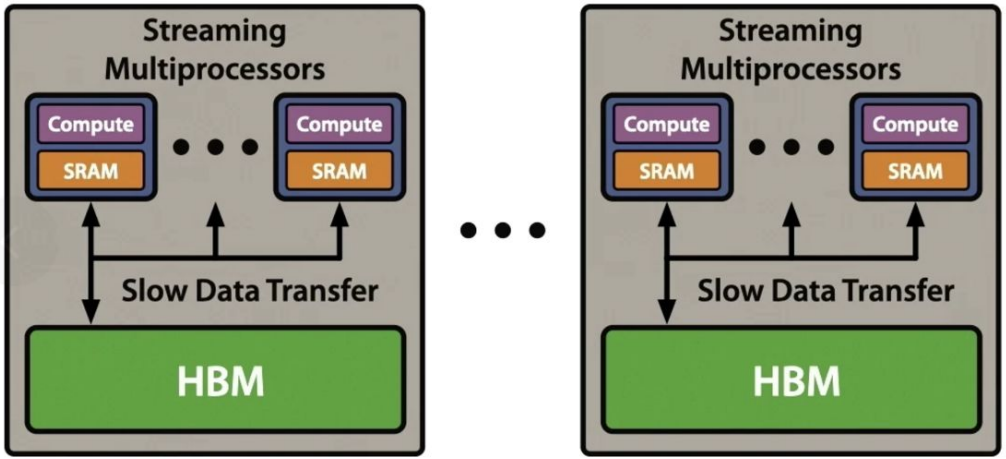


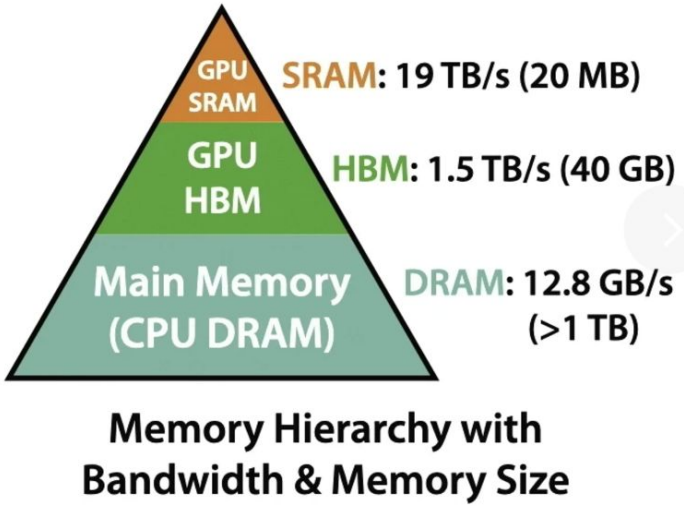
$$\mathbf{O} = \text{Dropout}(\text{Softmax}(\text{Mask}(\mathbf{QK}^T)))\mathbf{V}$$

Naive implementation requires repeated R/W from slow GPU HBM



Background: GPU Compute Model & Memory Hierarchy





Can we exploit the memory asymmetry to get speed up?
With IO-awareness (accounting for R/W to different levels of memory)

Blogpost: Horace He, Making Deep Learning Go Brrrr From First Principles.

Background

FlashAttention

Empirical Validation

Future Directions

How to Reduce HBM Reads/Writes: Compute by Blocks

Challenges: (1) compute softmax reduction without access to full input.
(2) backward without the large attention matrix from forward.

Tiling: Restructure algorithm to load block by block from HBM to SRAM to compute attention.

Recomputation: Don’t store attn. matrix from forward, recompute it in the backward.

Implementation: fused CUDA kernel for fine-grained control of memory accesses.

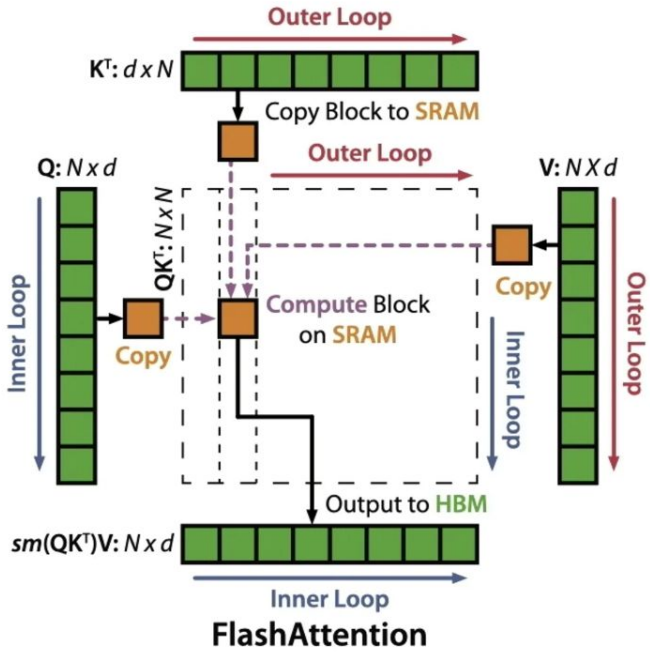
Tiling

Decomposing large softmax into smaller ones by scaling.

$$\text{softmax}([A_1, A_2]) = [\alpha\ \text{softmax}(A_1),\ \ \beta\ \text{softmax}(A_2)]\,.$$

$$\text{softmax}([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha\ \text{softmax}(A_1)\ V_1 + \beta\ \text{softmax}(A_1)\ V_2\,.$$

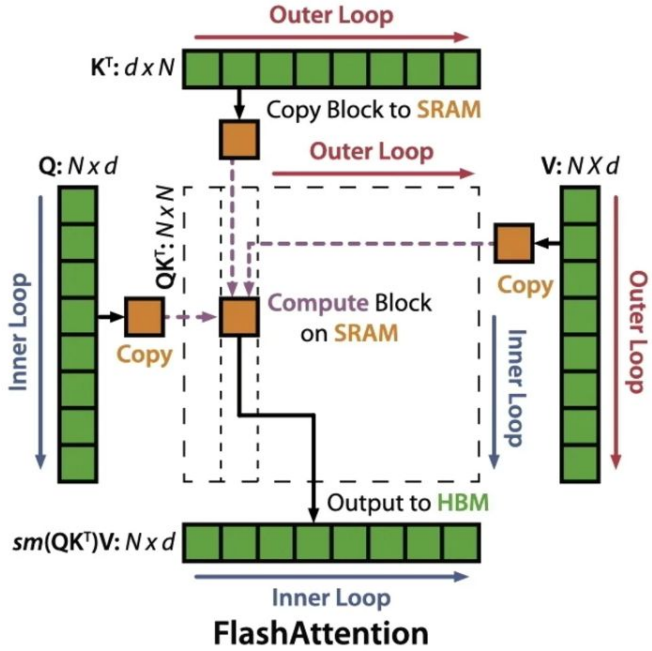
1. Load inputs by blocks from HBM to SRAM.
2. On chip, compute attention output wrt that block.
3. Update output in HBM by scaling.



Recomputation (Backward Pass)

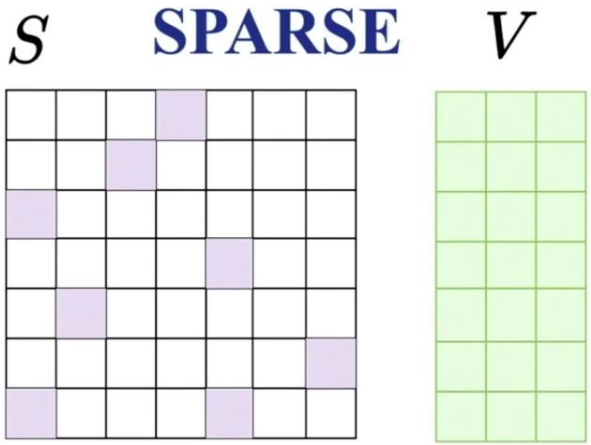
By storing softmax normalization factors from fwd (size N), quickly recompute attention in the bwd from inputs in SRAM.

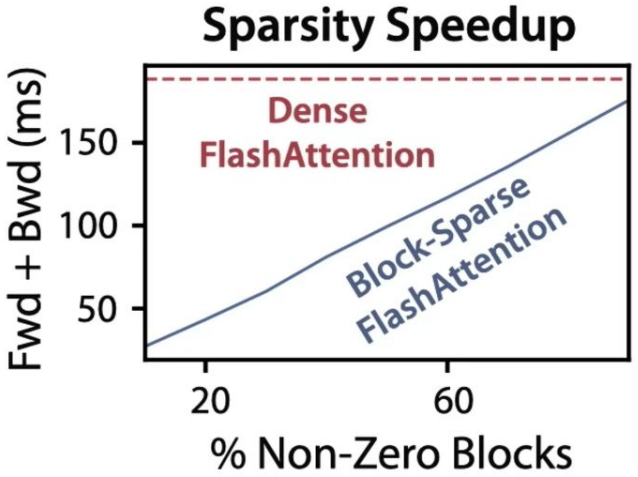
Attention	Standard	FLASHATTENTION
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3



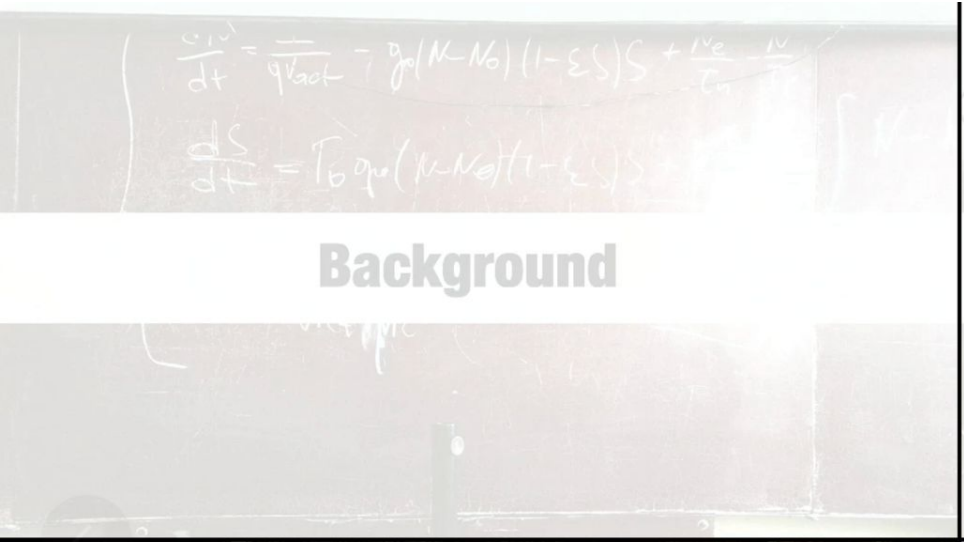
Speed up backward pass even with increased FLOPs.

Extension to Block-Sparse Attention



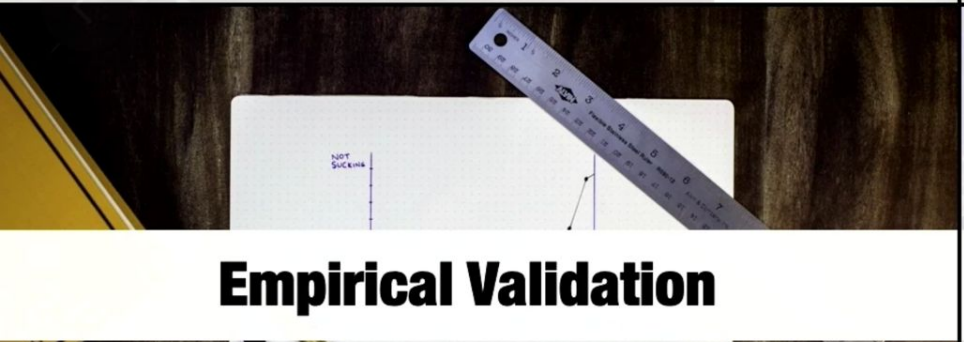


Just skip the zero blocks!



Background

FlashAttention

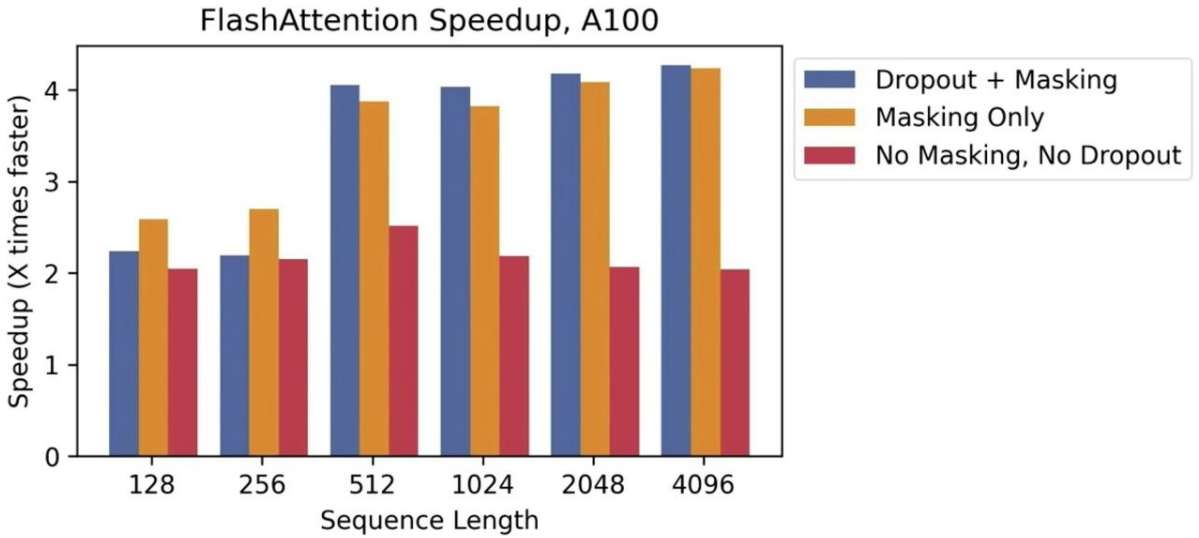


Empirical Validation

Future Directions

Attention Module: 2-4x speedup

Benchmarking **runtime** against sequence length



2-4x speedup over naive attention, esp. with dropout/masking

In paper: comparison with sparse/approximate attention

Faster Transformer Training: MLPerf Record for Training BERT

MLPerf: (highly optimized) standard benchmark for training speed

Time to hit an accuracy of 72.0% on MLM from a fixed checkpoint, averaged across 10 runs on 8xA100 GPUs

BERT Implementation	Training time (minutes)
Nvidia MLPerf 1.1 [56]	20.0 ± 1.5
FlashAttention (ours)	**17.4** ± 1.4

FlashAttention outperforms the previous MLPerf record by 15%

Mattson et al. *MLSys 2020*.

Faster Transformer Training: GPT-2

Training GPT-2 small and GPT-2 medium from scratch on OpenWebText, using 8xA100 GPUs

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [84]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [74]	18.2	4.7 days (2.0×)
GPT-2 small - FlashAttention	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [84]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [74]	14.3	11.5 days (1.8×)
GPT-2 medium - FlashAttention	14.3	**6.9 days (3.0×)**

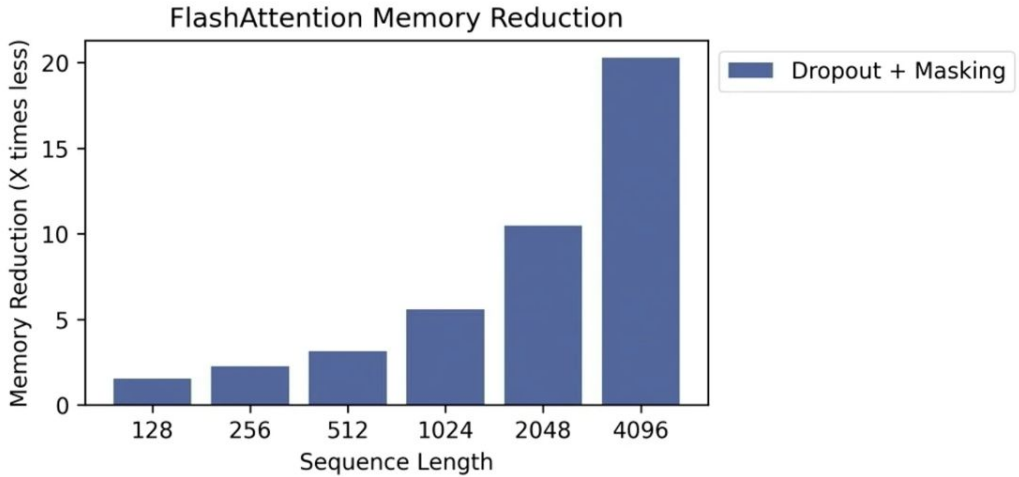
FlashAttention speeds up GPT-2 training by 2.0-3.5x

Longer sequence length: more speedup

Wolf et al. *EMNLP 2020*.
Shoeybi et al. *arXiv:1909.08053 2019*.

Attention Module: 10-20x memory reduction

Benchmarking **memory footprint** against sequence length



10-20x memory saving

Memory linear in sequence length — scaling up to 64K on one A100

Longer Sequences: GPT-2 with Long Context

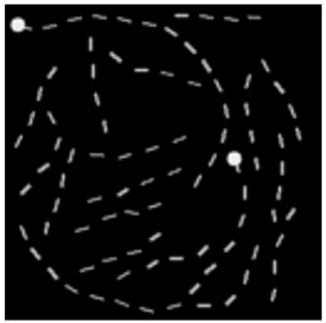
Training GPT-2 small with a longer context

Longer context: slower, but lower perplexity (better model)

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×)

FlashAttention with context length 4K is faster than Megatron-LM at 1K, and achieves lower perplexity

Longer Sequences: Path-X, Path-256



Path-X: Tell whether two dots are connected — designed to push Transformers (no patches)

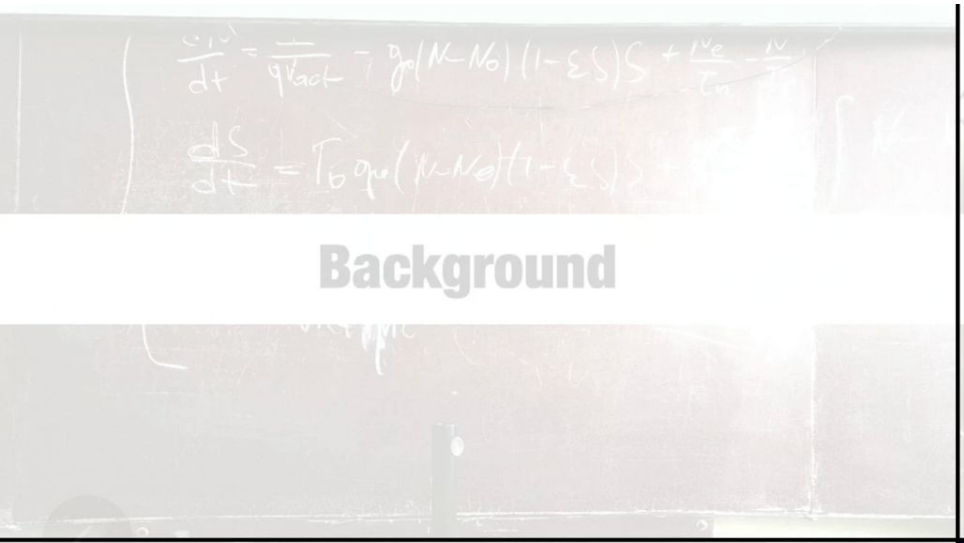
Requires sequence length 16K/64K for Path-X/Path-256

Model	Path-X	Path-256
Transformer	✗	✗
Linformer [81]	✗	✗
Linear Attention [48]	✗	✗
Performer [11]	✗	✗
Local Attention [77]	✗	✗
Reformer [49]	✗	✗
SMYRF [18]	✗	✗
FlashAttention	**61.4**	✗
Block-sparse FlashAttention	56.0	**63.1**

FlashAttention yields the first Transformer that can solve Path-X

Block-sparse FlashAttention enables Path-256

Tay et al. *ICLR 2020*.



Background

FlashAttention

Empirical Validation

Future Directions

Next Steps

- Getting it into more people's hands
- Technical work: extensions to support more GPUs — opportunity for compilers support for automatic fusion
- Bringing hardware-aware thinking to other ops

What can we do with longer sequences?

Summary

FlashAttention: **fast** and **memory-efficient** algorithm for exact attention

Key algorithmic ideas: tiling, recomputation

The upshot: faster model training, better models with longer sequences

Thank You!

Tri Dao

Dan Fu

Contact:
Tri Dao (trid@stanford.edu, @tri_dao)
Dan Fu (danfu@cs.stanford.edu, @realDanFu)

arXiv: https://arxiv.org/abs/2205.14135
Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Ruda, Christopher Ré. arXiv preprint arXiv:2204.07596.

Code: https://github.com/HazyResearch/flash-attention