FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness (2022)

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré

Flash-Decoding for long content inference (2023)

Tri Dao, Daniel Haziza, Fracisco Massa, Grigory Sizov

Presented by:

Raghav Sharma & Daniel Hocevar

March 21, 2025

UNIVERSITY OF TORONTO

DEFY GRAVITY

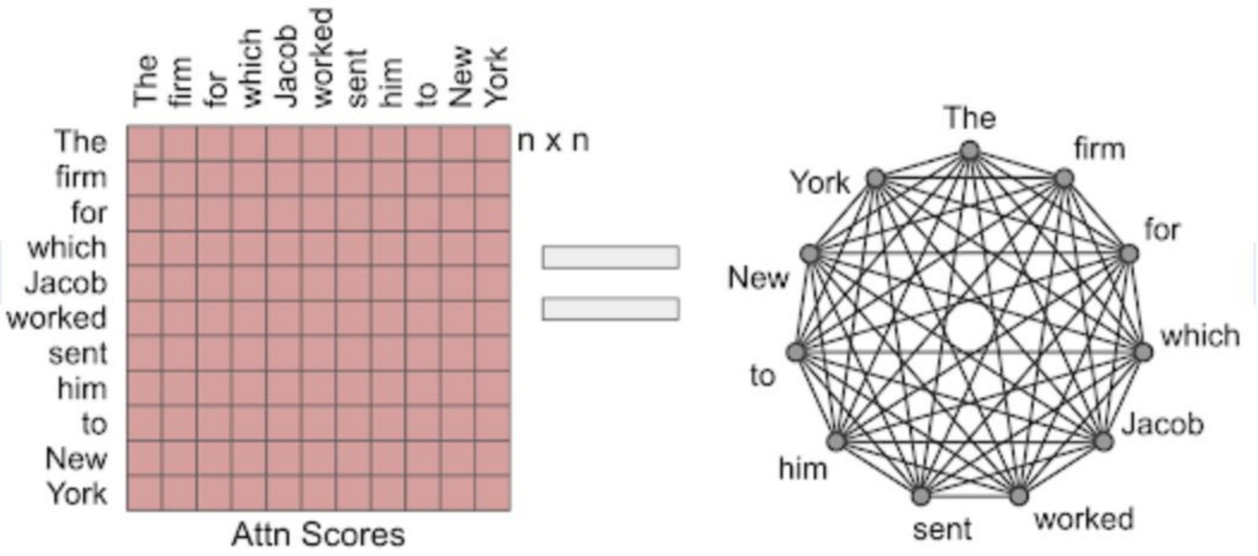
Why do we need flash attention?

Motivation

On long sequences, transformers are slow and require significant memory

The problem

Self attention has quadratic time/memory complexity



Self attention's quadratic complexity can be visualized using a fully connected graph

UNIVERSITY OF TORONTO

Image adapted from: Dubey, A. (2021, March 25). Constructing Transformers For Longer Sequences with Sparse Attention Methods. *Deep Mind.* https://research.google/blog/constructing-transformers-for-longer-sequences-with-sparse-attention-methods/

Making Attention More Efficient - Existing Approaches

Sparse approximation

Observe that the softmax in self-attention is dominated by the most similar query-key pairs:

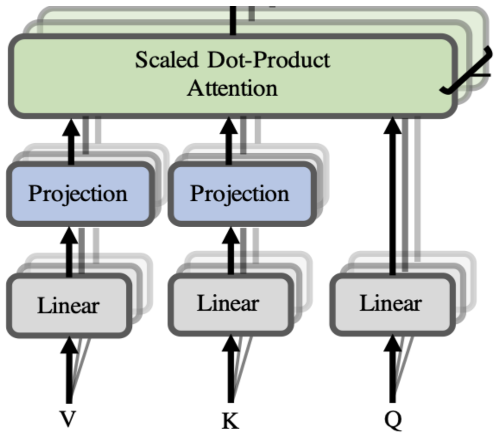
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Reformer (2020) groups similar queries/keys into buckets, and only attends within a bucket.

Low rank approximation

Linformer (2020) proved that components of soft-attention computation are low-rank.

Apply linear projection to compute low-rank representations for K and V to reduce computational requirements.



UNIVERSITY OF TORONTO

Making Attention IO-Aware: Drawing inspiration from other fields

IO-Aware Computing

Optimizing algorithms to run on specific types of hardware, accounting for their unique IO setup

- GPUs: accounting for read/writes to SRAM and HBM and accounting for their respective speeds

Examples

Database joins

- Optimizing communication between CPU registers, cache and disk storage

Image processing

- Halide (compiler for image processing) leverages IO-aware compute extensively

Linear algebra

- Limiting communications between slow and fast memory for matrix factorization

Preliminary: Different Types of Bounded Computation

Memory Bound Computation

Most of the time spent waiting for data to be read to/written from memory

Ex: Matrix multiplication for very large matrices

Compute Bound Computation

Most time spent waiting for calculations to be processed

Ex: Fibonacci calculation

Pseudocode: An Iterative Algorithm for Fibonacci Numbers.

```
procedure iterative fibonacci(n: nonnegative integer)
if n = 0 then
    return 0
else
   x := 0
   y := 1
   for i := 1 to n − 1
        z := x + y
        x := y
        y := z
return y
{output is the n^th Fibonacci number}
```

Attention computation is memory bounded

UNIVERSITY OF TORONTO

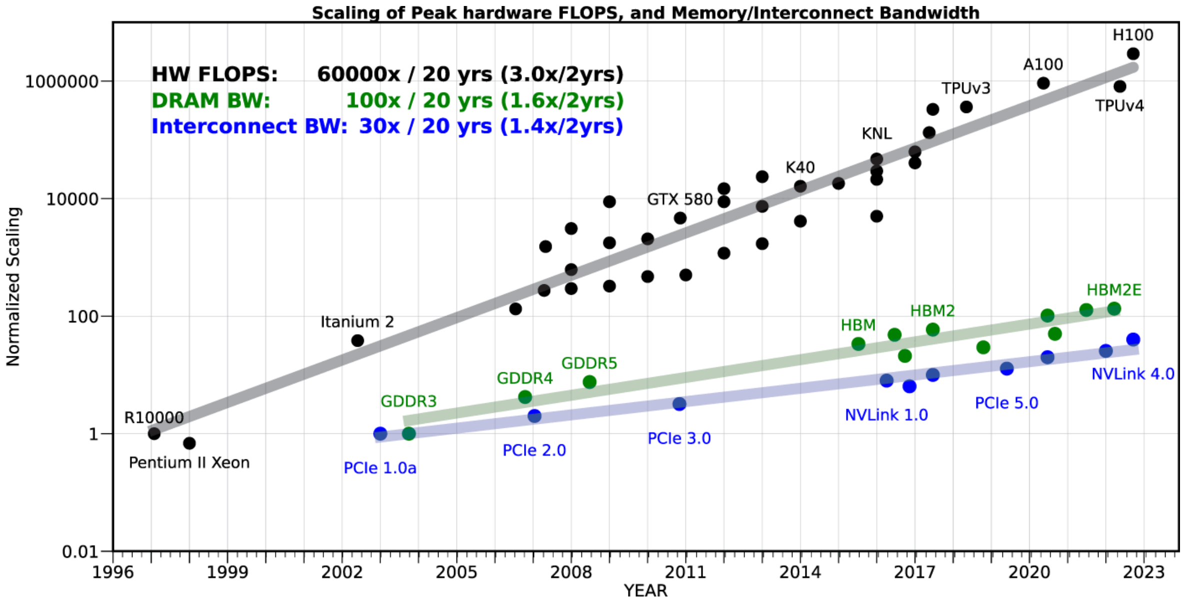
Preliminary: The Memory Wall Problem

Faster memory as a solution?

Unlikely anytime soon due to the memory wall problem

Memory Wall Problem

Rate of improvement in processor performance is outpacing the rate of improvement in memory performance



AI Hardware and Memory Wall Problem

Memory bounded computations will likely remain memory bounded

UNIVERSITY OF TORONTO

Figure from: Ghloami et al. (2024, March 21). AI and Memory Wall.

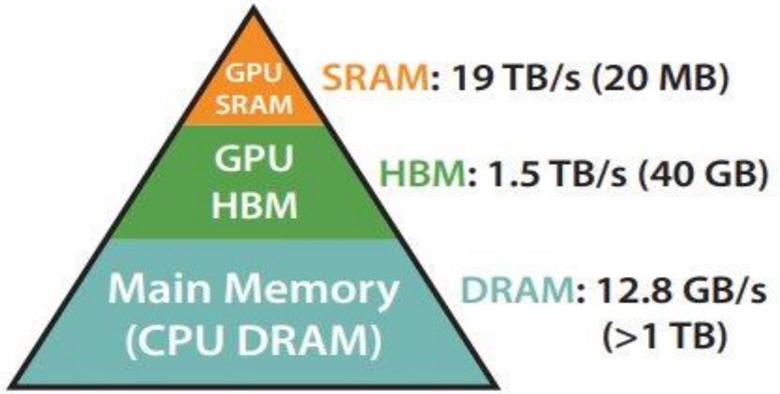
Goal: Reduce Memory Accesses in Attention Computation

FlashAttention Approach

Reorganize the Attention computation to access the slow memory as little as possible

Memory Hierarchy

In the GPU memory hierarchy, the HBM memory is the slow memory we want to avoid accessing as much as possible



Memory Hierarchy with Bandwidth & Memory Size

FlashAttention avoids this slow by memory by using two common optimization techniques:

1) **Tiling**
2) **Recomputation**

UNIVERSITY OF TORONTO

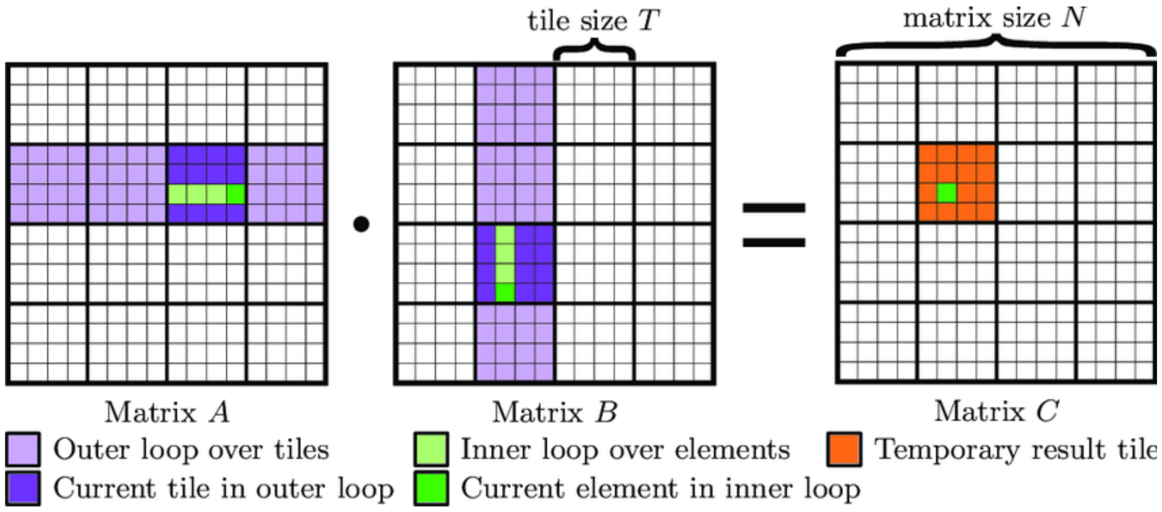
FlashAttention Optimization 1: Tiling

Challenge 1

Once the sequence length, N, is large, the corresponding **Q, K, V** matrices cannot fit into the small SRAM. Thus, you must constantly write/read them from the large HBM (quadratic accesses).

Solution

Break the **Q, K, V** matrices into blocks that fit into the small SRAM, so that each value in the matrices is read only once from the HBM (linear accesses).



Tiling for matrix multiplication

Tiling is a commonly used technique for matrix multiplication, so why hasn't this been done before?

FlashAttention Optimization 1: Tiling Continued

Challenge 2

Softmax must be applied row-wise and depends on the entire row. Can we still break up this computation into tiles?

Solution

You can compute the softmax for each block, and when adding it to the accumulated results of other blocks, scale it using **additionally computed statistics** (max and normalizer).

$$\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d} \text{ where } N \text{ is the sequence length and } d \text{ is the head dimension}$$

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

Attention Computation

UNIVERSITY OF TORONTO

FlashAttention Optimization 1: Tiling Extra Details

$$m(x) := \max_i \ x_i, \quad f(x) := \left[e^{x_1 - m(x)} \quad \ldots \quad e^{x_B - m(x)}\right], \quad \ell(x) := \sum_i f(x)_i, \quad \mathrm{softmax}(x) := \frac{f(x)}{\ell(x)}.$$

Typical softmax computation with numerical stability

$$\begin{aligned}
m(x) &= m\left(\left[x^{(1)} \ x^{(2)}\right]\right) = \max\left(m(x^{(1)}), m(x^{(2)})\right), \quad f(x) = \left[e^{m(x^{(1)}) - m(x)} f(x^{(1)}) \quad e^{m(x^{(2)}) - m(x)} f(x^{(2)})\right], \\
\ell(x) &= \ell\left(\left[x^{(1)} \ x^{(2)}\right]\right) = e^{m(x^{(1)}) - m(x)} \ell(x^{(1)}) + e^{m(x^{(2)}) - m(x)} \ell(x^{(2)}), \quad \mathrm{softmax}(x) = \frac{f(x)}{\ell(x)}.
\end{aligned}$$

Decomposing the softmax on a block level requires computing two extra statistics *m* and *l per block*

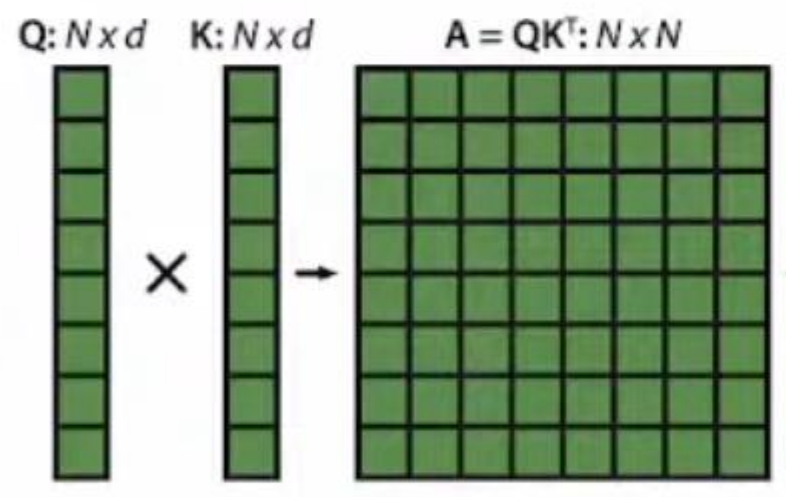
FlashAttention Optimization 2: Recomputation

Challenge

Avoid storing large **N by N** matrices, **S and P**, as intermediate values required for the backwards pass to compute gradients with respect to **Q, K, V**

Solution

Instead, by just storing the **N by d matrices, Q, K, V, O,** and the **N** extra block statistics (*m, l*), we can recompute the necessary intermediate values required for the gradient, with less space



Size visualization of intermediate values computed during attention

Fewer HBM accesses once again. But at the cost of more compute.

FlashAttention Optimization 2: Recomputation Extra Details

$\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ where N is the sequence length and d is the head dimension.

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

Attention Computation

$$dv_j = \sum_i \boxed{P_{ij}} do_i = \sum_i \boxed{\frac{e^{q_i^T k_j}}{L_i}} do_i$$

Derivative of the jth column of matrix V can be computed by recomputing parts of the intermediate matrix P using the block statistics *(m, l)*

UNIVERSITY OF TORONTO

FlashAttention Benefit 1: Faster Attention

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

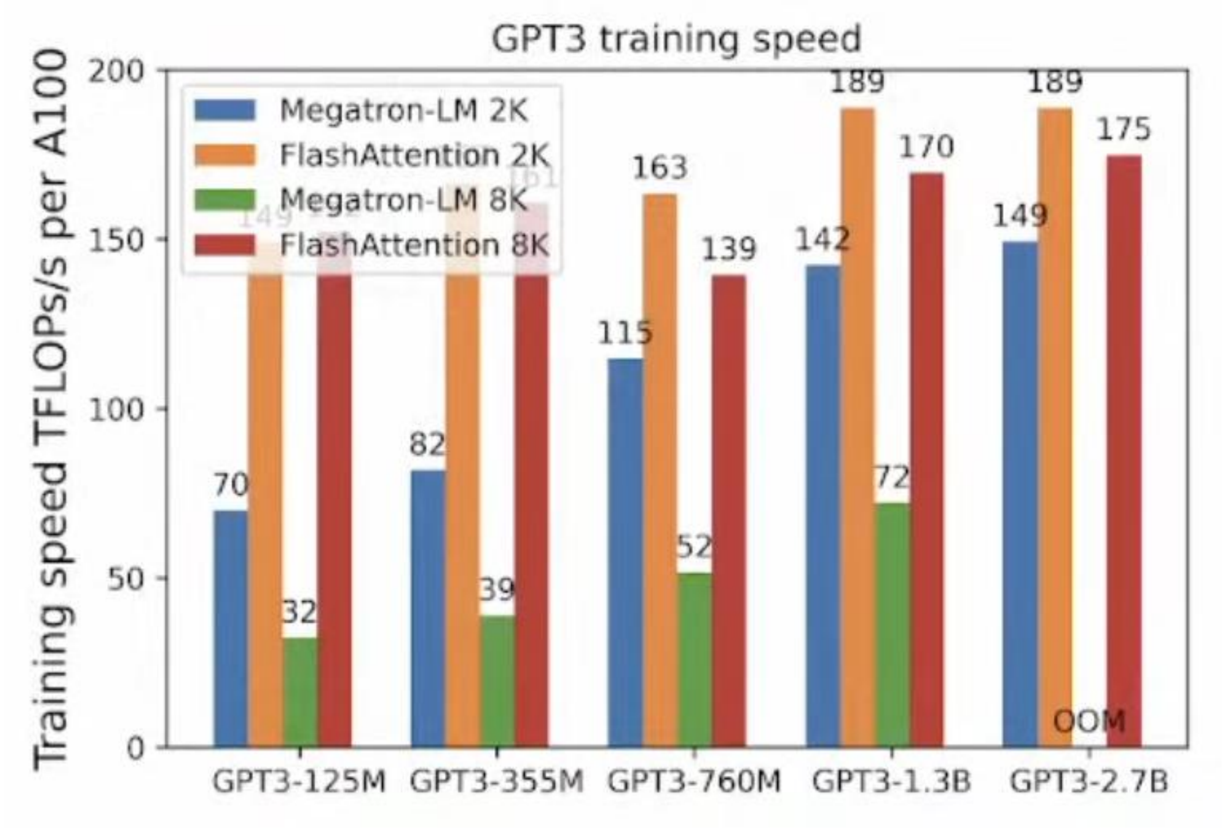
FLOPs versus HBM accesses of FlashAttention and the default PyTorch implementation at the time. Performance on an A100 GPU (seq. length 1024, head dim. 64, 16 heads, batch size 64)

Model implementations	Training time (speedup)
GPT-2 small - Huggingface [87]	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	4.7 days (2.0×)
GPT-2 small - FlashAttention	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [87]	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	11.5 days (1.8×)
GPT-2 medium - FlashAttention	**6.9 days (3.0×)**

Training time with different attention implementations. Training performed on 8 x A100 GPUs

UNIVERSITY OF TORONTO

FlashAttention Benefit 2: Supports Longer Sequences



Training GPT-3 with leading exact Attention implementations with sequence length 2K and 8K

FlashAttention Limitations

GPU Specific CUDA Kernels

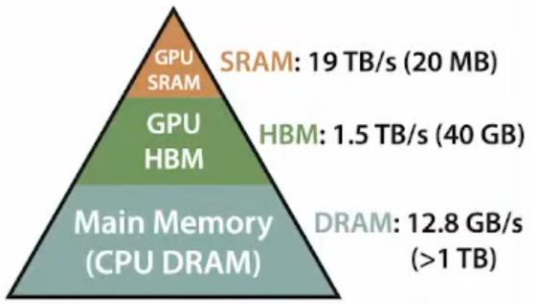
Each GPU has optimal different optimal block sizes based on its memory sizes

No support for Multi-GPU FlashAttention yet

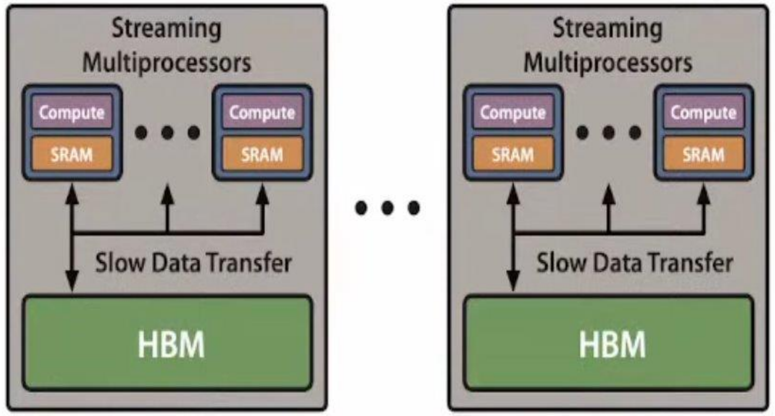
Need to account for an even slow layer of memory transfer – GPU to GPU data transfer, which makes Multi-GPU training not straightforward

FlashAttention is only helpful during training

During inference time, the user query batch size is commonly 1, so typically one only streaming multiprocessor can be used (A100 has 108)



Memory Hierarchy with Bandwidth & Memory Size



UNIVERSITY OF TORONTO

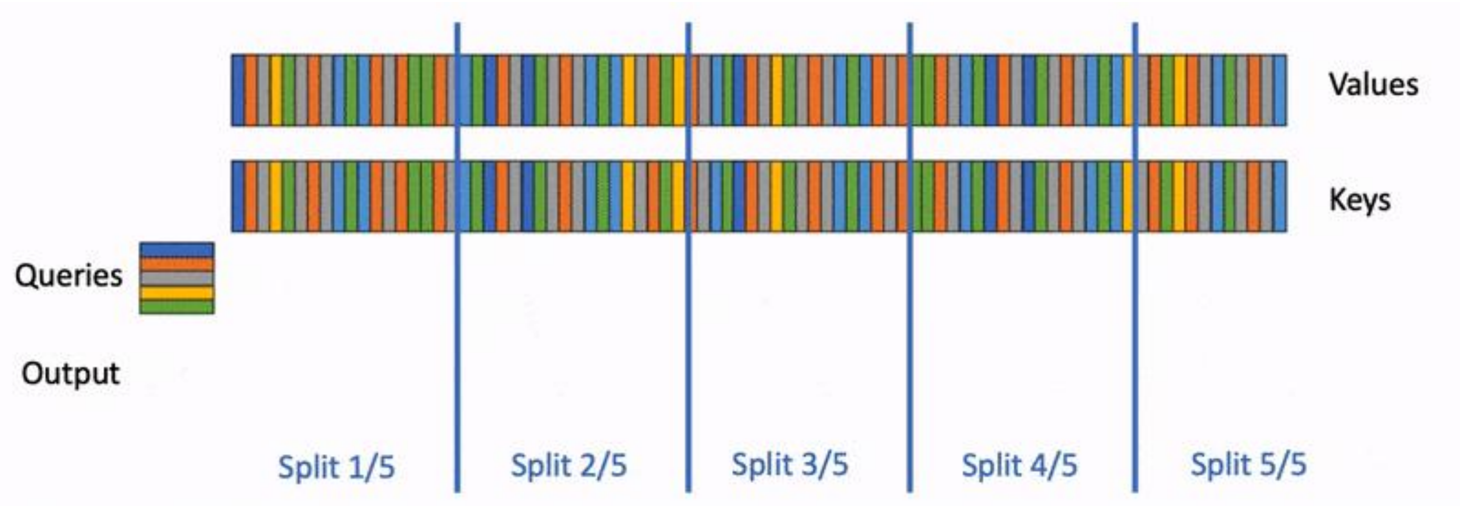
FlashDecoding: Speeding up inference

Problem

Bottlenecks during inference are different than training (only one query during inference)

Solution

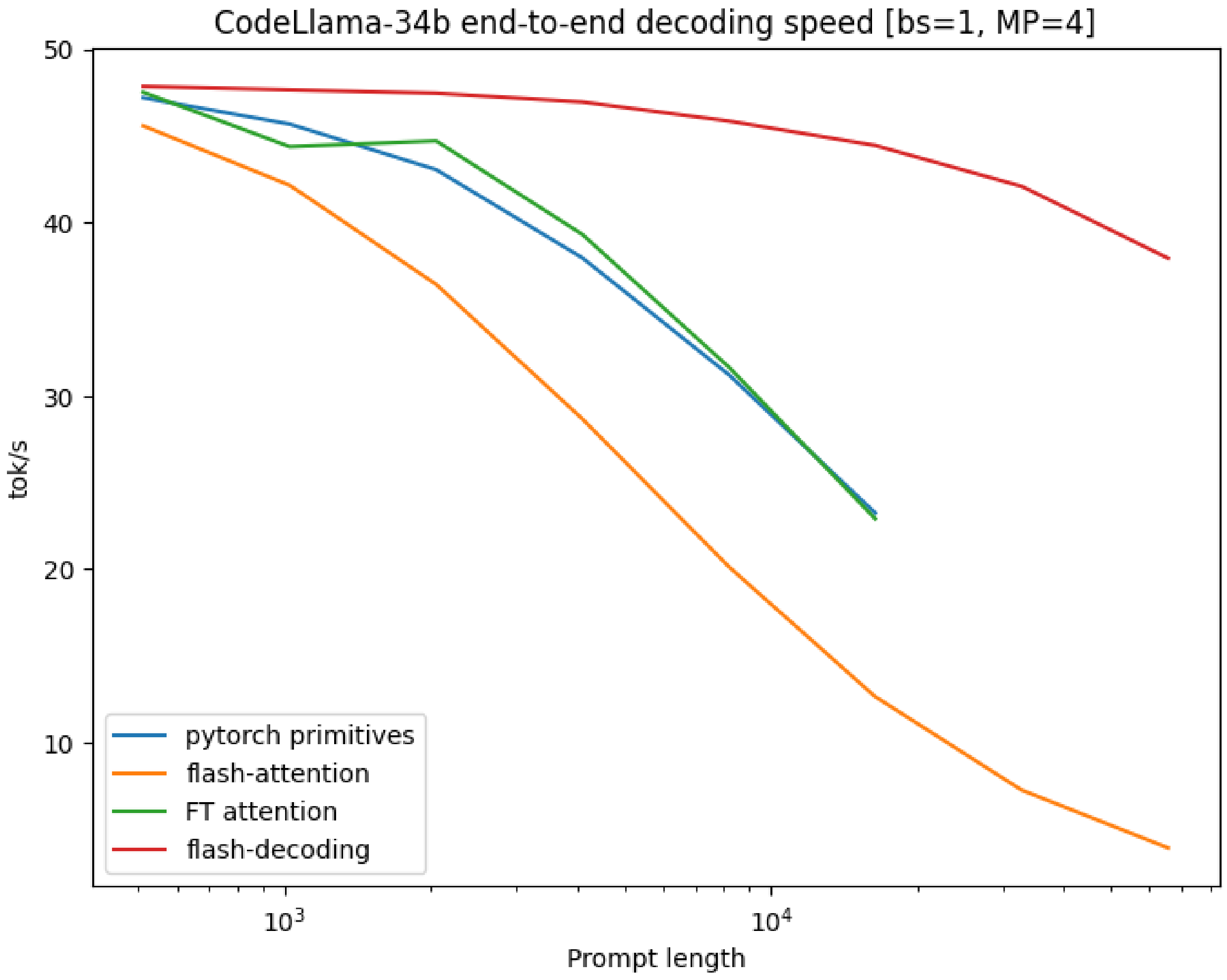
Parallelize across key/value sequence length:
- Split keys and values into smaller chunks
- For each split, compute attention using Flash Attention
- Compute output by reducing over all splits



FlashDecoding: Performance Analysis

Result

Flash decoding significantly improves the speed of generation – especially for longer sequences



Colab Notebook Walkthrough

Demonstration of Tiling on Matrix Multiplication

- Implemented Python GPU memory simulator
- Uses this simulator to profile the hbm accesses of 3 different matrix multiplication algorithms

[Colab Link]

Recap

Takeaways

- Attention in Transformers is memory bounded
- To speed it up, FlashAttention uses tiling and recomputation to reduce hbm accesses
- Tiling performs Attention computation in a block wise manner
- Recomputation avoids storing large intermediate matrices in the hbm during the backwards pass
- FlashAttention does not straightforwardly extend during inference time, due to a query batch size of 1
- FlashDecoding fixes this issue by parallelizing across the Key and Value matrices

Thank You!

UNIVERSITY OF
TORONTO