FlashAttention-2:
Faster Attention with Better Parallelism and Work Partitioning

By: Tri Dao

Presented by: Deema Alnuhait
Oct 4th

Motivation

- Longer sequence has proven better performance in LM and supports other modalities, code, audio.. etc.
- The main bottleneck for scaling to a longer sequence is the attention layer
- Scaling the sequence length implies a quadratic increase in runtime and memory

  - GPT4 >> 32k
  - MosiacML’s MPT >> 65k
  - Anthropic Claude >> 100k

Previous Work

- FlashAttention2 is built upon the previously widely adopted FlashAttention (referred to FlashAttention1 in this context)
- Main idea is to reorder the attention computation and leverage tiling recomputation which reduces memory usage from quadratic to linear in sequence length
- It yields to 2 to 4x time speedup over the optimized baselines and up to 10 to 20x memory utilization with no approximation
- With the increased sequence length FlashAttention underperform other primitives such as GEMM (General Matrix-multiply)
- Forward pass reaches 30-50% of theoretical maximum FLOPs/s
- Backward pass is even more challenging, reaching only 25-35% of the maximum throughput
- Optimized GEMM can reach up to 80-90% of the theoretical maximum device throughput

Background – Hardware Characteristics

- **Memory Hierarchy** comprises of high bandwidth memory (HBM) [Slow and Large], and on-chip SRAM (aka shared memory) [Fast and Small]
- **Execution Model** Kernal is the number of threads
- Threads are organized into blocks scheduled to run on streaming multiprocessors (SMs)
- In each thread block, threads are grouped into warps (32 threads)
- Threads within a warp threads can communicate faster
- Warps within a thread block can communicate by reading from / writing to shared memory
- Each kernel loads inputs from HBM to registers and SRAM, computes, then writes outputs to HBM.

Background – Standard Attention Implementation

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

The Softmax is applied row-wise.

Standard attention implementation

1. Calls GEMM (Matrix-Multiply) to compute **S**
2. Writes results to HBM
3. Load from HBM to compute the Softmax
4. Rewrite **P** to HBM
5. Calls GEMM to compute **O**

This frequent memory access cause s slow execution and requires $O(N^2)$ since **S** and **P** are materialize
P has to be saved for backward pass to compute the gradients

Background – Flash Attention or FA1

- Forward Pass: Utilize a classical technique of tiling to reduce memory IOs, by:

1. loading blocks of inputs from HBM to SRAM
2. computing attention for that block, then
3. updating the output without writing the large intermediate matrices **S** and **P** to HBM

- Using online Softmax to split the attention computation into blocks, rescale the output of each block to get non-approximate results.
- Reduces the number of memory's reads/writes

- FlashAttention yields 2-4× wall-clock speedup over-optimized baseline attention implementations.

Background – Standard Softmax

- Considering two blocks in S

$$
\begin{aligned}
m &= \max(\mathrm{rowmax}(\mathbf{S}^{(1)}), \mathrm{rowmax}(\mathbf{S}^{(2)})) \in \mathbb{R}^{B_r} \\
\ell &= \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m}) + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m}) \in \mathbb{R}^{B_r} \\
\mathbf{P} &= \begin{bmatrix} \mathbf{P}^{(1)} & \mathbf{P}^{(2)} \end{bmatrix} = \mathrm{diag}(\ell)^{-1} \begin{bmatrix} e^{\mathbf{S}^{(1)}-m} & e^{\mathbf{S}^{(2)}-m} \end{bmatrix} \in \mathbb{R}^{B_r \times 2B_c} \\
\mathbf{O} &= \begin{bmatrix} \mathbf{P}^{(1)} & \mathbf{P}^{(2)} \end{bmatrix} \begin{bmatrix} \mathbf{V}^{(1)} \\ \mathbf{V}^{(2)} \end{bmatrix} = \mathrm{diag}(\ell)^{-1} e^{\mathbf{S}^{(1)}-m} \mathbf{V}^{(1)} + e^{\mathbf{S}^{(2)}-m} \mathbf{V}^{(2)} \in \mathbb{R}^{B_r \times d}.
\end{aligned}
$$

-m for stabilization

Background – FA1

Online softmax instead computes “local” softmax with respect to each block and rescale to get the right output at the end

$$
\begin{aligned}
m^{(1)} &= \mathrm{rowmax}(\mathbf{S}^{(1)}) \in \mathbb{R}^{B_r} \\
\ell^{(1)} &= \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m^{(1)}}) \in \mathbb{R}^{B_r} \\
\tilde{\mathbf{P}}^{(1)} &= \mathrm{diag}(\ell^{(1)})^{-1} e^{\mathbf{S}^{(1)}-m^{(1)}} \in \mathbb{R}^{B_r \times B_c} \\
\mathbf{O}^{(1)} &= \tilde{\mathbf{P}}^{(1)}\mathbf{V}^{(1)} = \mathrm{diag}(\ell^{(1)})^{-1} e^{\mathbf{S}^{(1)}-m^{(1)}}\mathbf{V}^{(1)} \in \mathbb{R}^{B_r \times d} \\
m^{(2)} &= \max(m^{(1)}, \mathrm{rowmax}(\mathbf{S}^{(2)})) = m \\
\ell^{(2)} &= e^{m^{(1)}-m^{(2)}}\ell^{(1)} + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m^{(2)}}) = \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m}) + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m}) = \ell \\
\tilde{\mathbf{P}}^{(2)} &= \mathrm{diag}(\ell^{(2)})^{-1} e^{\mathbf{S}^{(2)}-m^{(2)}} \\
\mathbf{O}^{(2)} &= \mathrm{diag}(\ell^{(1)}/\ell^{(2)})^{-1}\mathbf{O}^{(1)} + \tilde{\mathbf{P}}^{(2)}\mathbf{V}^{(2)} = \mathrm{diag}(\ell^{(2)})^{-1} e^{s^{(1)}-m}\mathbf{V}^{(1)} + \mathrm{diag}(\ell^{(2)})^{-1} e^{s^{(2)}-m}\mathbf{V}^{(2)} = \mathbf{O}.
\end{aligned}
$$

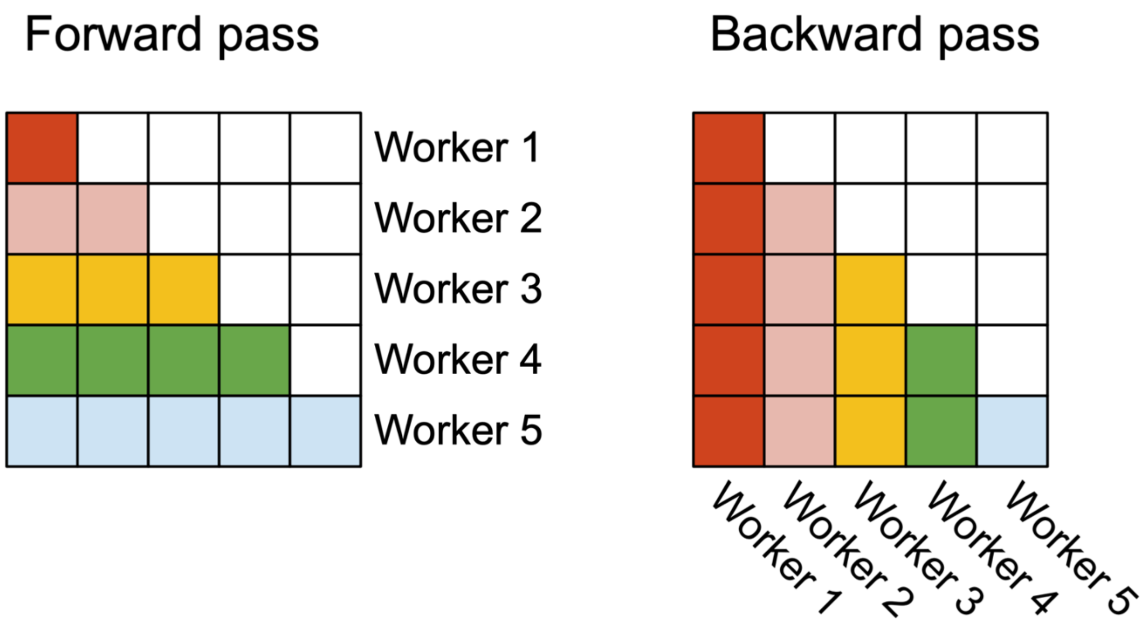
(1) Reduce Non-Matmul FLOPs Operations in FA2

$$
\begin{aligned}
m^{(1)} &= \mathrm{rowmax}(\mathbf{S}^{(1)}) \in \mathbb{R}^{B_r} \\
\ell^{(1)} &= \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m^{(1)}}) \in \mathbb{R}^{B_r} \\
\tilde{\mathbf{O}^{(1)}} &= e^{\mathbf{S}^{(1)}-m^{(1)}}\mathbf{V}^{(1)} \in \mathbb{R}^{B_r \times d} \\
m^{(2)} &= \max(m^{(1)}, \mathrm{rowmax}(\mathbf{S}^{(2)})) = m \\
\ell^{(2)} &= e^{m^{(1)}-m^{(2)}}\ell^{(1)} + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m^{(2)}}) = \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m}) + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m}) = \ell \\
\tilde{\mathbf{P}}^{(2)} &= \mathrm{diag}(\ell^{(2)})^{-1}e^{\mathbf{S}^{(2)}-m^{(2)}} \\
\tilde{\mathbf{O}}^{(2)} &= \mathrm{diag}(e^{m^{(1)}-m^{(2)}})\tilde{\mathbf{O}}^{(1)} + e^{\mathbf{S}^{(2)}-m^{(2)}}\mathbf{V}^{(2)} = e^{s^{(1)}-m}\mathbf{V}^{(1)} + e^{s^{(2)}-m}\mathbf{V}^{(2)} \\
\mathbf{O}^{(2)} &= \mathrm{diag}(\ell^{(2)})^{-1}\tilde{\mathbf{O}}^{(2)} = \mathbf{O}.
\end{aligned}
$$

(2) Parallelizing Attention Computation

- FA1 parallelizes over batch size (B) and the number of heads >> running many threads blocks together
- Each thread runs on a single streaming multiprocessor (SM) e.g., A100 has 108 SMs
- Unfortunately, with long sequence s, we cannot afford large batch size s or large number s of heads
- This leads to many idle SMs

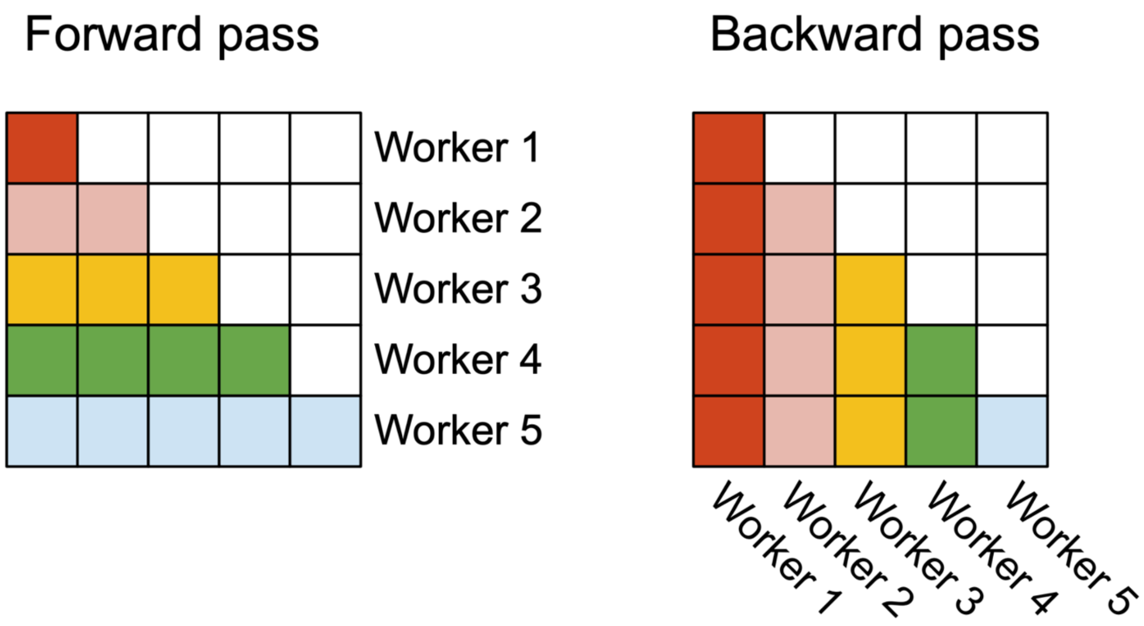
- FA2 parallelizes additionally over sequence length, so supports three parallelization



(2) Parallelizing Attention Computation

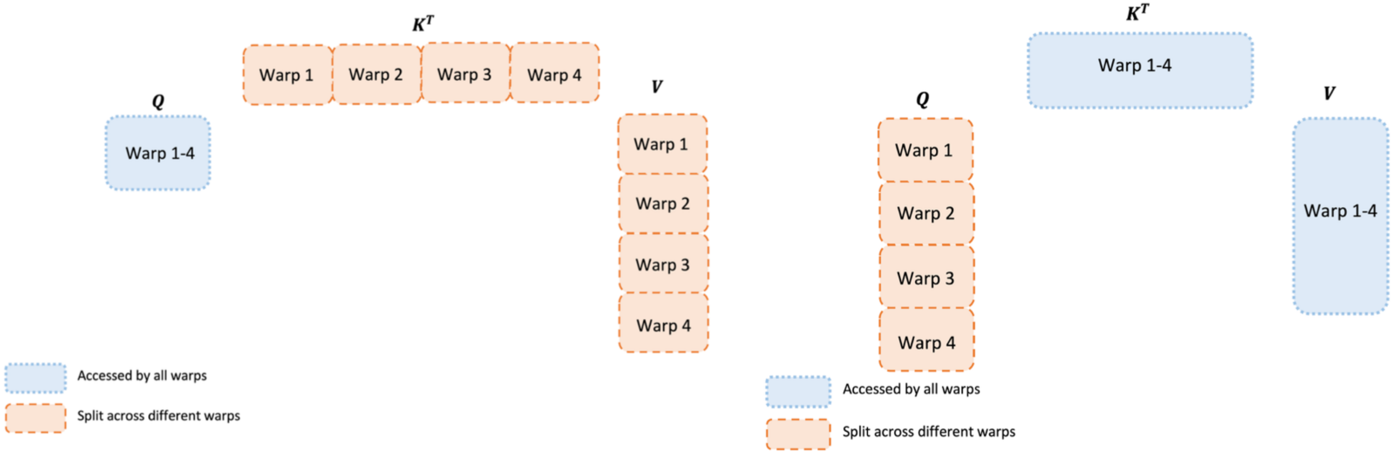
- FA1 works in two loops
- First over the j-th(k, v) blocks, and the second runs over the i-th Q
- They compute in SRAM (fast and small), and update O^i, l_i and m_i in HBM (large and slow)

- In FA2 It swaps the order of the loop
- Placing Q_i outside >> the loop goes through different blocks of the Q matrix
- Inner operates on K and V
- Swapping the order offers sequence length parallelization as the # Q == to the length of the sequence
- In autoregressive attention >> set the upper triangle to infinity and leave them uncomputed



(3) Work Partitioning between Warps

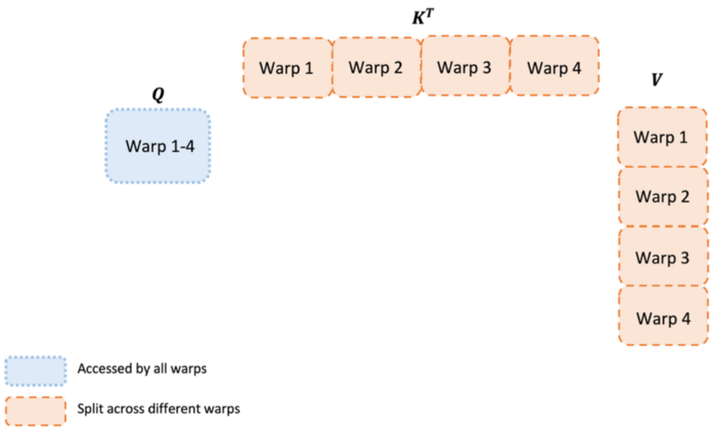
- Typically use 4 or 8 warps per thread block
- Every warps contains 32 threads
- In FA1 "Split-K" technique:
  - K and V are split across 4 warps, while Q is kept accessible by all warps
  - Each warp multiplies to get a slice of QK^T
  - Then multiply with a slice of V



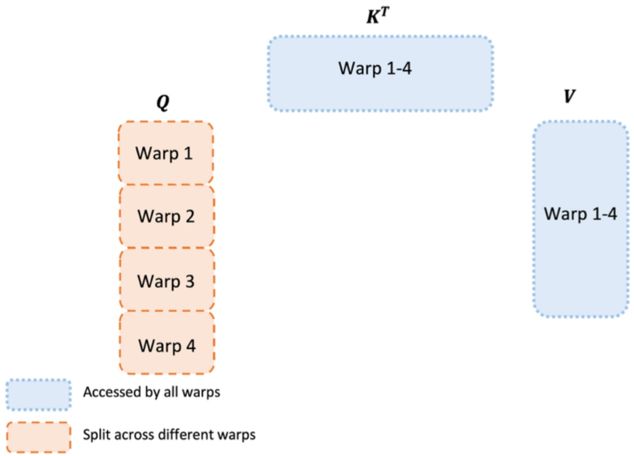
(a) FlashAttention (b) FlashAttention-2

(3) Work Partitioning between Warps

- In FA2:
  - we split Q across 4 warps
  - While keeping K and V accessible by all warps
  - After each warp computes a slice of QK^T, they only need to multiply with their shared slice of V, to get their corresponding slice of the output O
  - No need to communicate between warps
  - Reduction in shared memory's reads/writes >> speedup



(a) FlashAttention

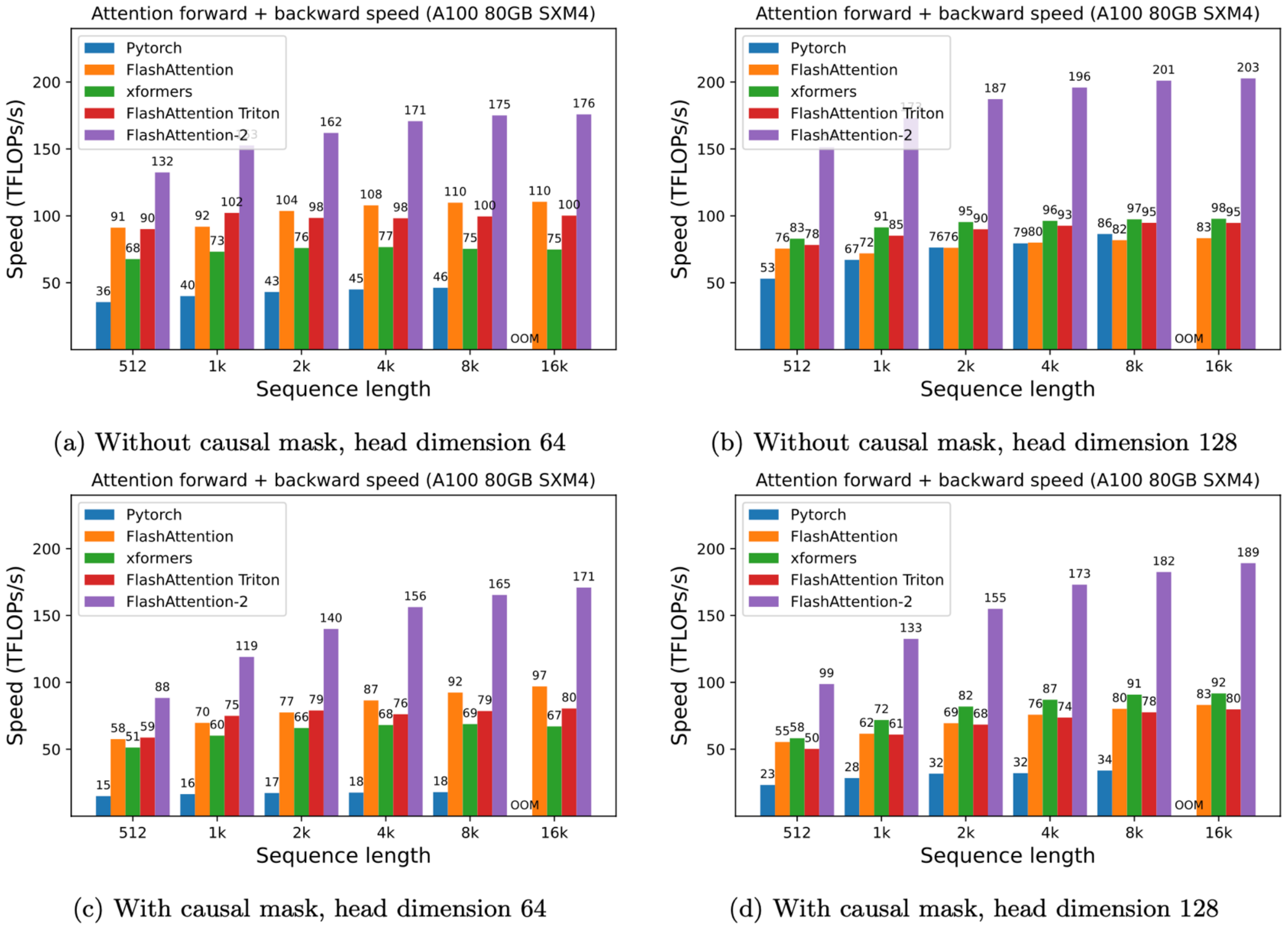


(b) FlashAttention-2

- The same applies to backward pass

Results

- A100
- Different Head Dimension [64, 128]
- W/O Causal mask



(a) Without causal mask, head dimension 64

(b) Without causal mask, head dimension 128

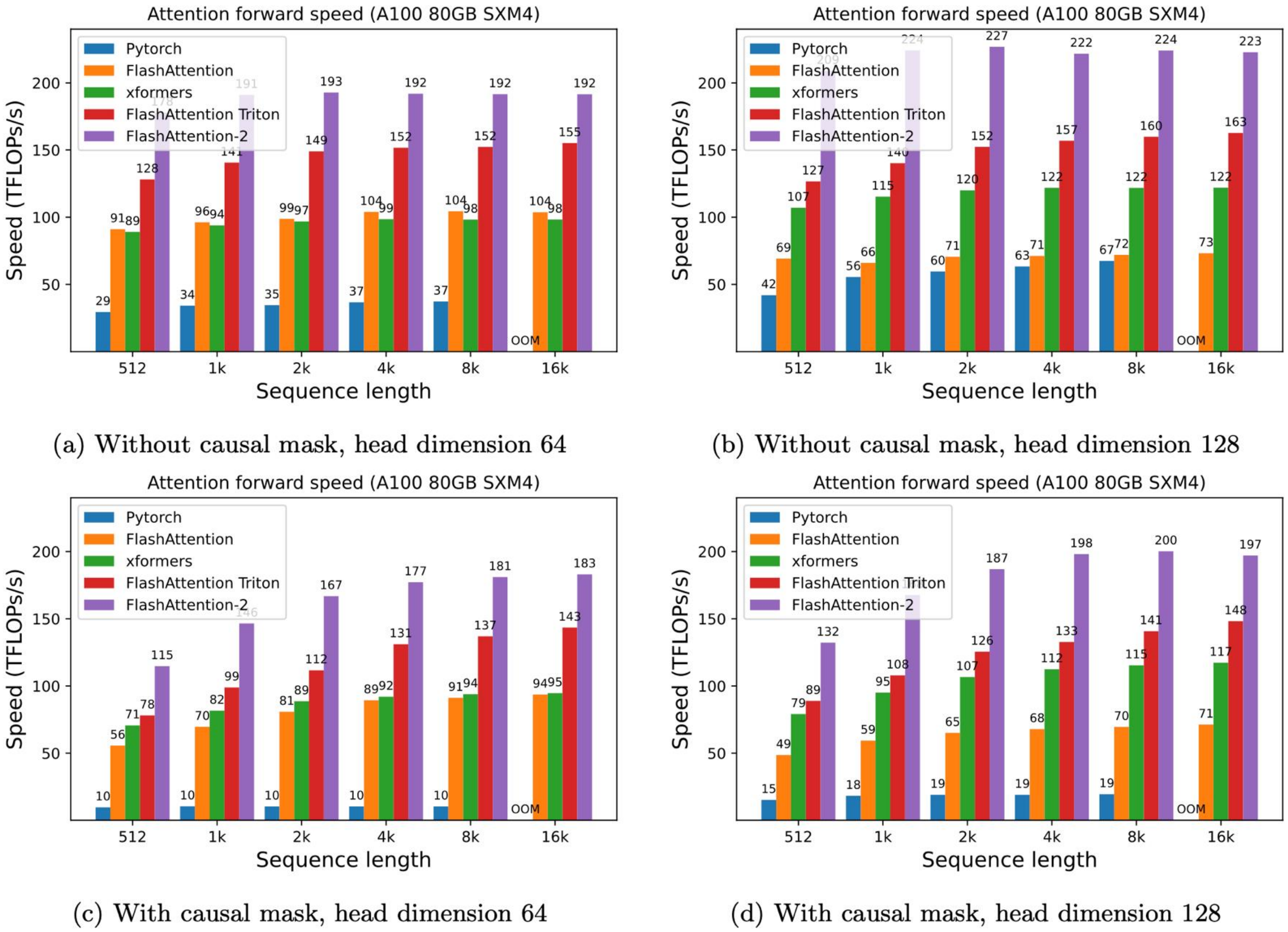
(c) With causal mask, head dimension 64

(d) With causal mask, head dimension 128

Figure 4: Attention forward + backward speed on A100 GPU

Results

- Forward Speed on A100
- Different Head Dimension [64, 128]
- W/O Causal mask



(a) Without causal mask, head dimension 64

(b) Without causal mask, head dimension 128

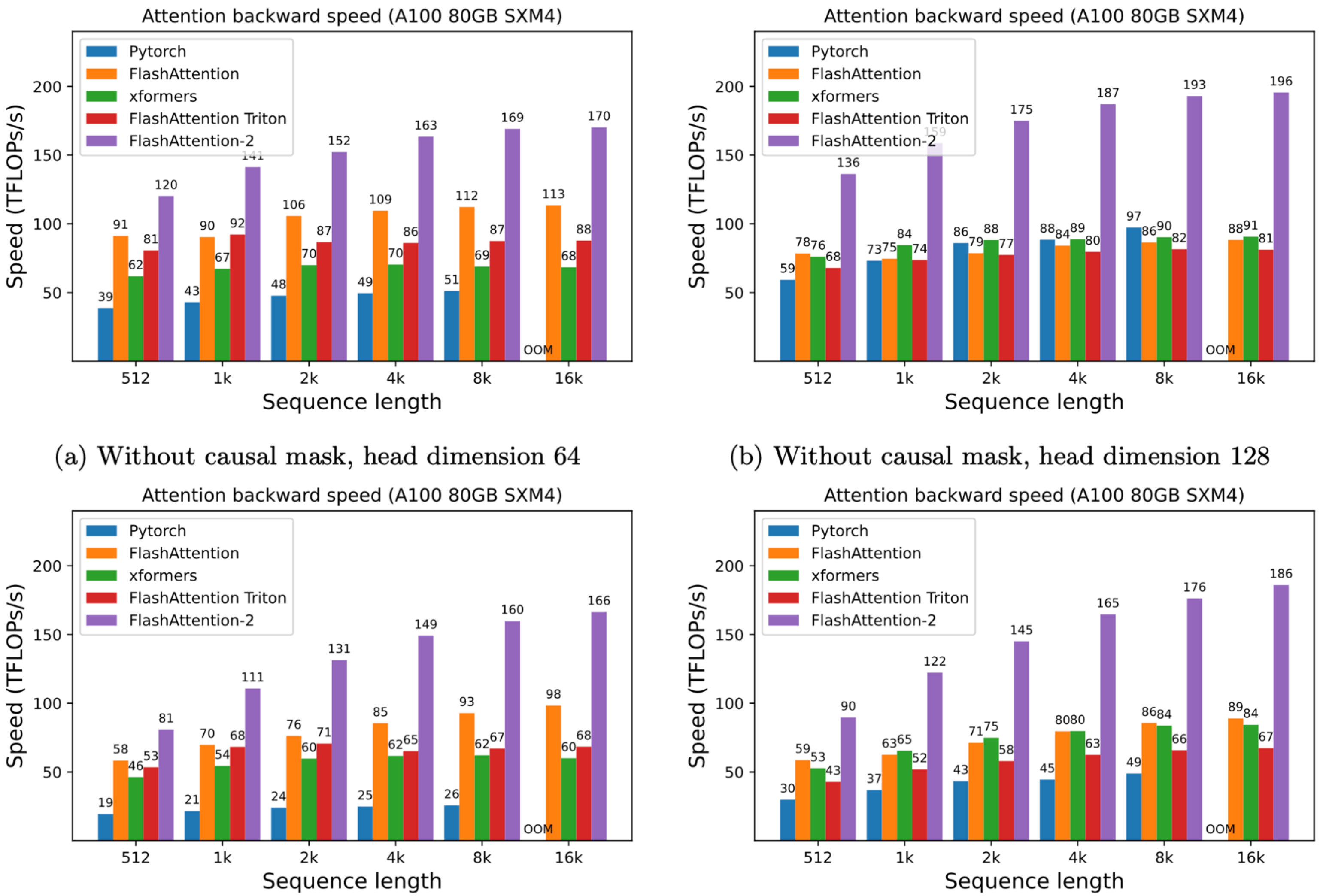
(c) With causal mask, head dimension 64

(d) With causal mask, head dimension 128

Figure 5: Attention forward speed on A100 GPU

Results

- Backward Speed on A100
- Different Head Dimension [64, 128]
- W/O Causal mask



(a) Without causal mask, head dimension 64

(b) Without causal mask, head dimension 128

(c) With causal mask, head dimension 64

(d) With causal mask, head dimension 128

Figure 6: Attention backward speed on A100 GPU

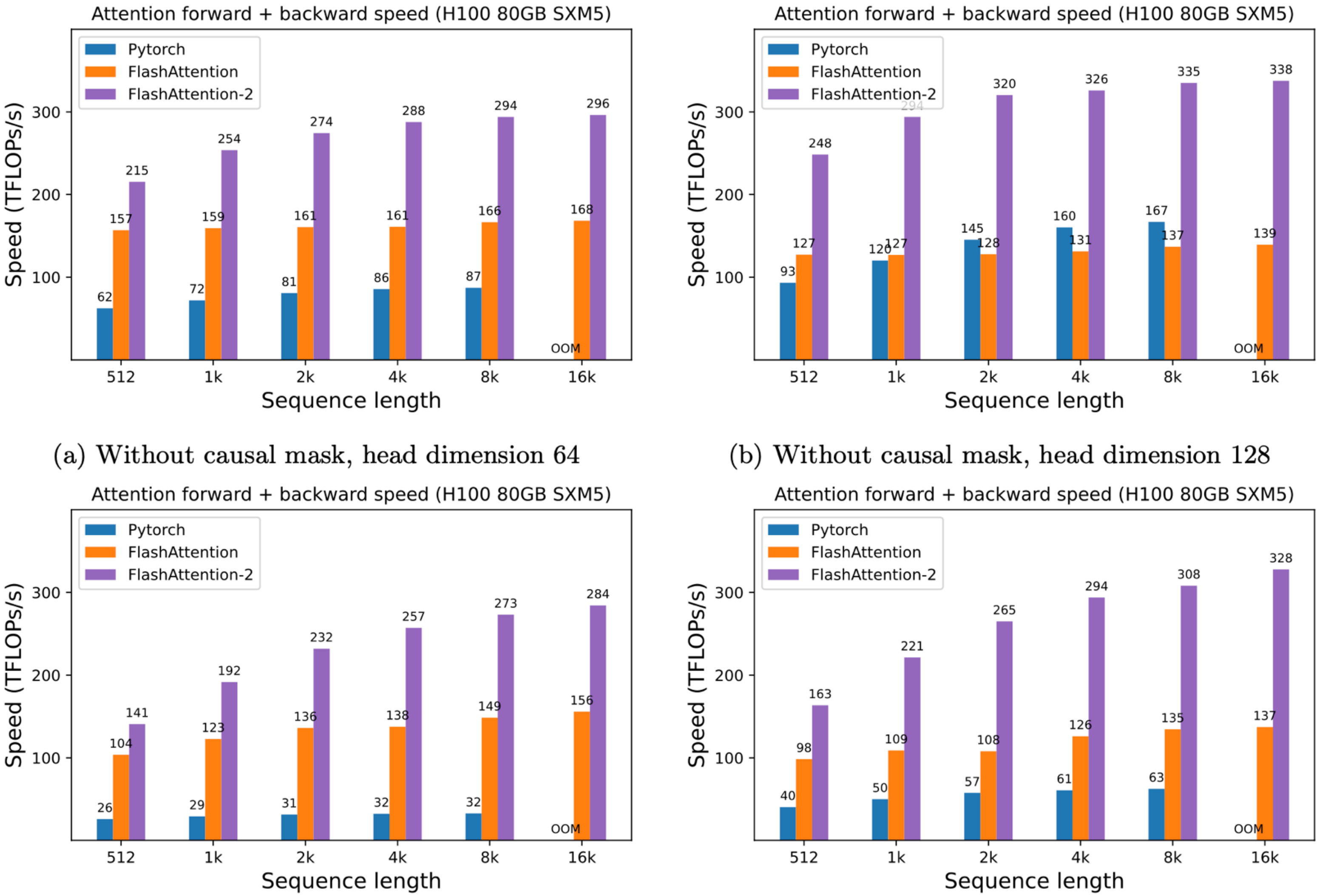
Results – Training Speed

Table 1: Training speed (TFLOPs/s/GPU) of GPT-style models on 8×A100 GPUs. FlashAttention-2 reaches up to 225 TFLOPs/s (72% model FLOPs utilization). We compare against a baseline running without FlashAttention.

Model	Without FlashAttention	FlashAttention	FlashAttention-2
GPT3-1.3B 2k context	142 TFLOPs/s	189 TFLOPs/s	196 TFLOPs/s
GPT3-1.3B 8k context	72 TFLOPS/s	170 TFLOPs/s	220 TFLOPs/s
GPT3-2.7B 2k context	149 TFLOPs/s	189 TFLOPs/s	205 TFLOPs/s
GPT3-2.7B 8k context	80 TFLOPs/s	175 TFLOPs/s	225 TFLOPs/s

H100 GPU

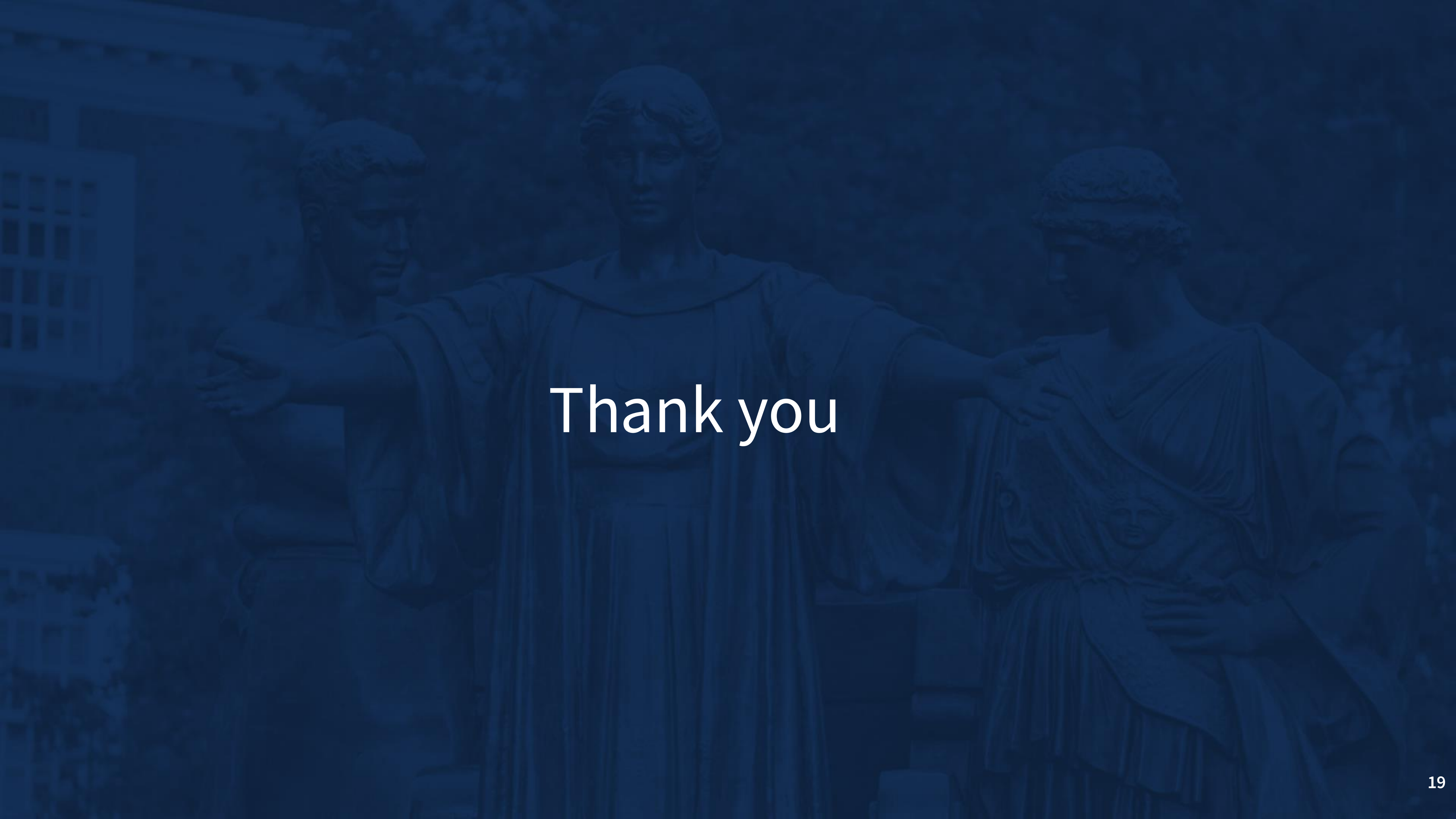
- Different Head Dimension [64, 128]
- W/O Causal mask



(a) Without causal mask, head dimension 64 (b) Without causal mask, head dimension 128

(c) With causal mask, head dimension 64 (d) With causal mask, head dimension 128

Figure 7: Attention forward + backward speed on H100 GPU



Thank you