LLM inference

DeepSpeed Inference, ByteTransformer, and FlexGen

Ying Cao

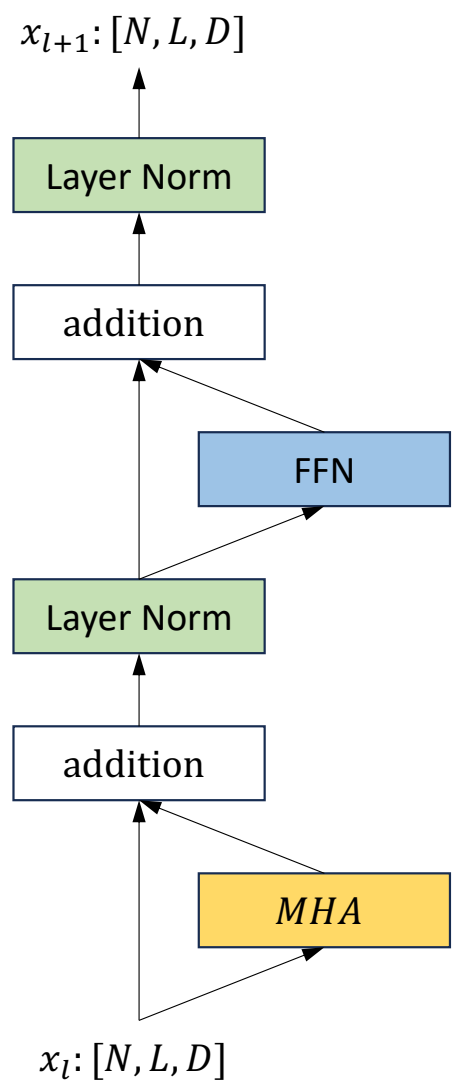
June 28, 2023

Table of contents

Background

The Transformer Architecture

The Transformer block[1].



- The transformer model consists of n_{layers} blocks that are stacked on top of each other.
- Learnable parameters in a Transformer block:

Components	Parameters	Shape
MHA	$W_Q^i, W_K^i, W_V^i, W_O^i$	$[d, d]$
MHA	$b_Q^i, b_K^i, b_V^i, b_O^i$	$[1, d]$
layer norm	$\gamma,\ \beta$	$[1, d]$
FFN	W_1^i	$[d, 4d]$
FFN	b_1^i	$[1, 4d]$
FFN	W_2^i	$[4d, d]$
FFN	b_2^i	$[1, d]$
In Total		$12d^2 + 13d$

[1] Ashish Vaswani et al. "Attention is All you Need". In: *NIPS*. 2017, pp. 5998–6008.

Estimating Storage Requirements for LLM Weights

LLMs are too large to fit in the memory of a single GPU device. Inference performance for large Transformer models can be limited by memory capacity.

- Storage required for LLM weights:

Model	#parameters[2]	n_{layers}	d	n_{heads}	d_{head}	#storage(GB)[3]
$\text{BERT}_{\text{base}}$	84.94 M	12	768	12	64	0.3164
$\text{BERT}_{\text{large}}$	302.00 M	24	1024	16	64	1.1250
GPT-3 small	84.94 M	12	768	12	64	0.3164
GPT-3 medium	302.00 M	24	1024	16	64	1.1250
GPT-3 large	679.50 M	24	1536	16	96	2.5313
GPT-3 13B	12.68 B	40	5140	40	128	47.2421
GPT-3 175B	173.95 B	96	12288	96	128	648.0006

- Other storage: Activations, K-V cache!

[2] #paramters $= n_{\text{layers}} * (12d^2 + 13d)$. **The parameters are counted in a manner that excludes word embedding and the last softmax parameters used to predict the following word.**

[3] #storage **is counted for single precision floating point number =** #parameters *** 4 / 1024 / 1024 / 1024**

LLM inference

Prefill Stage: prompts processing has large parallelism

1. cache K-V for prompts

$$\mathbf{x}_K^i = \mathbf{x}^i \cdot W_K^i$$
$$\mathbf{x}_V^i = \mathbf{x}^i \cdot W_V^i$$

2. predict next token

$$\mathbf{x}_Q^i = \mathbf{x}^i \cdot W_Q^i$$
$$\mathbf{x}_O^i = \text{layernorm}\left(\text{softmax}\left(\frac{\mathbf{x}_Q^i (\mathbf{x}_K^i)^T}{\sqrt{d}}\right)\mathbf{x}_V^i W_O^i + \mathbf{x}^i\right)$$
$$\mathbf{x}^{i+1} = \text{layernorm}\left(\text{relu}\left(\mathbf{x}_O^i W_1^i\right) W_2^1 + \mathbf{x}_O^i\right)$$

Decode Stage: next token generation has insufficient parallelism

1. update K-V for prompts

$$\mathbf{x}_K^i \leftarrow \text{concat}(\mathbf{x}_K^i, \mathbf{t}^i \cdot W_K^i)$$
$$\mathbf{x}_V^i \leftarrow \text{concat}(\mathbf{x}_V^i, \mathbf{t}^i \cdot W_V^i)$$

2. predict next token, the same as above, but each time a token

Two stages in LLM inference

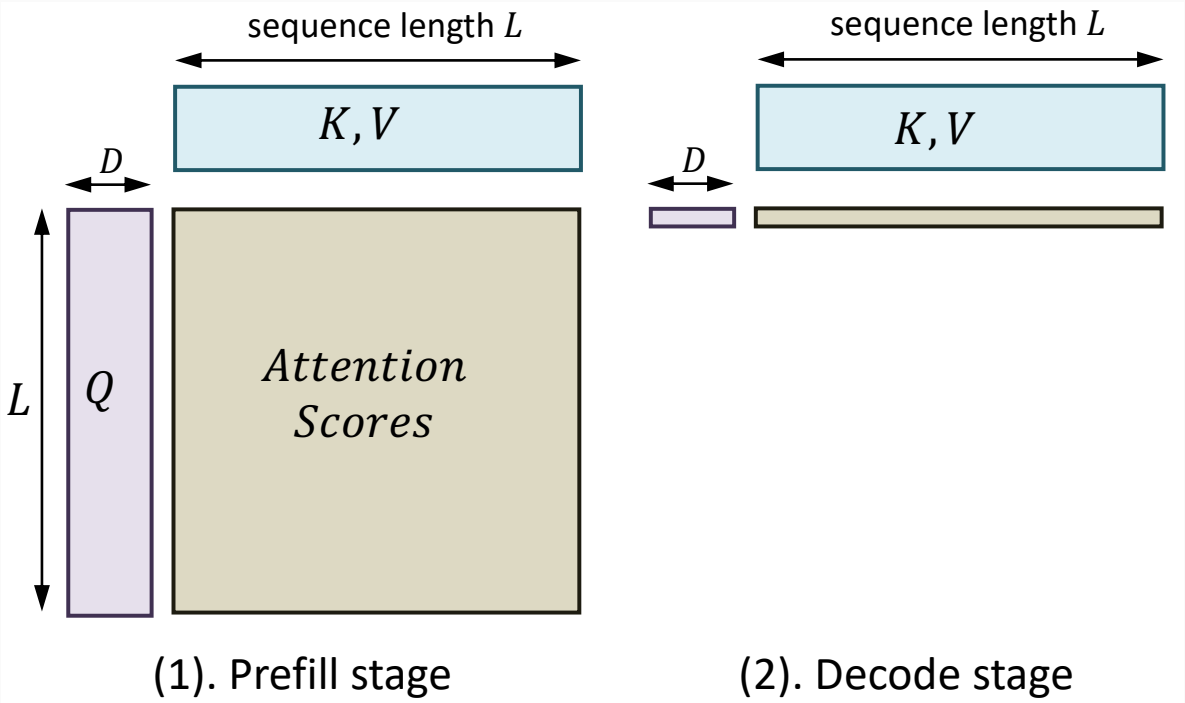
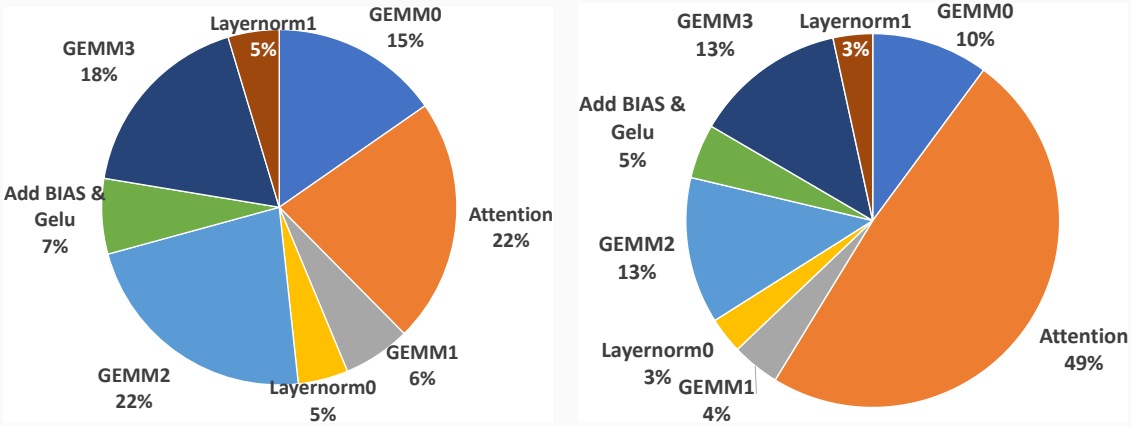


Figure 1: There is a significant change in batch size between the two stages of LLM inference.

Performance Breakdown of a Transformer Block



(a) Sequence lengths 256 (b) Sequence lengths 1024

Figure 2: A performance breakdown of a single Transformer block[4].

1. Transformer block is memory bounded.
2. The computational complexity of MHA is quadratic to sequence length.

[4] Yujia Zhai et al. "ByteTransformer: A High-Performance Transformer Boosted for Variable-Length Inputs". In: *CoRR* abs/2210.03052 (2022).

Distinction Between Transformer Training and Inference:

Small batch size.

1. Kernels are not well optimized for small batch size.
   The use of skinny matrix multiplication is a result of having a small batch size, where the batch size N is much smaller than the dimensionality d.

$$\mathbf{x}_Q^i = \mathbf{x}^i W_Q^i \qquad [N,d] = [N,d] \times [d,d]$$
$$\mathbf{y}_1^i = \mathbf{x}_O^i W_1^i \qquad [N,4d] = [N,d] \times [d,4d]$$
$$\mathbf{y}_2^i = \mathbf{y}_1^i W_2^i \qquad [N,d] = [N,4d] \times [4d,d]$$

2. Insufficient of parallelism and performance is limited by memory bandwidth in reading weights.

FlexGen, ByteTransformer, and DeepSpeed Inference

	FlexGen[5]	ByteTransformer[6]	DeepSpeed Inference[7]
Throughput-oriented	yes	no	no
Latency-oriented	no	yes	yes
SLA	not discussed	not discussed	not discussed
Model	OPT-175B	BERT-like	Dense Transformer MoE Transformer
Optimized Kernels	not discussed	yes for variable length	yes for small batch size
Kernel Fusion	not discussed	yes	yes
Multi-GPU	yes	no	yes

[5] Ying Sheng et al. "High-throughput Generative Inference of Large Language Models with a Single GPU". In: *CoRR* abs/2303.06865 (2023).

[6] Yujia Zhai et al. "ByteTransformer: A High-Performance Transformer Boosted for Variable-Length Inputs". In: *CoRR* abs/2210.03052 (2022).

[7] Reza Yazdani Aminabadi et al. "DeepSpeed- Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale". In: *SC*. IEEE, 2022, 46:1–46:15.

Deep Speed Inference

DeepSpeed Inference: Optimized Kernels

Challenges: small batch size leads to limited work in kernels

1. The overheads associated with kernel invocation become significant. → Kernel fusion
2. The performance is limited by the memory bandwidth utilization in reading weights. → Data reuse at shared memory or regiter level
3. Both CuTLASS and CuBLAS are not well-optimized for small batch size. → Optimize GEMM kernels for small matrix sizes

DeepSpeed Inference: SBI-GeMM

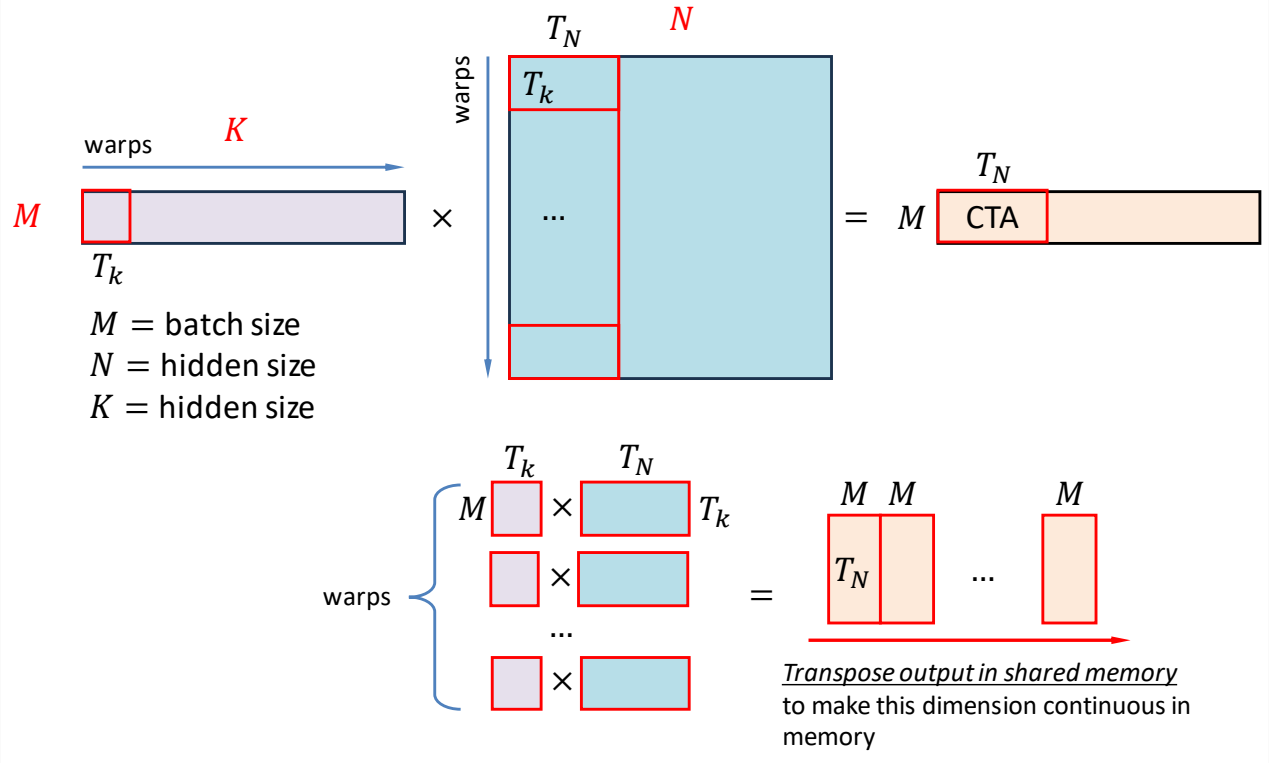


Figure 3: Rationale of the SBI-GeMM

1. tile strategies: tile dimensionality N first, if this does not produce enough parallelism, tile K then.
2. cooperative-group reduction: transpose output tile on the shared memory
3. leverage full cache-line.

DeepSpeed Inference: Fused Transformer Block

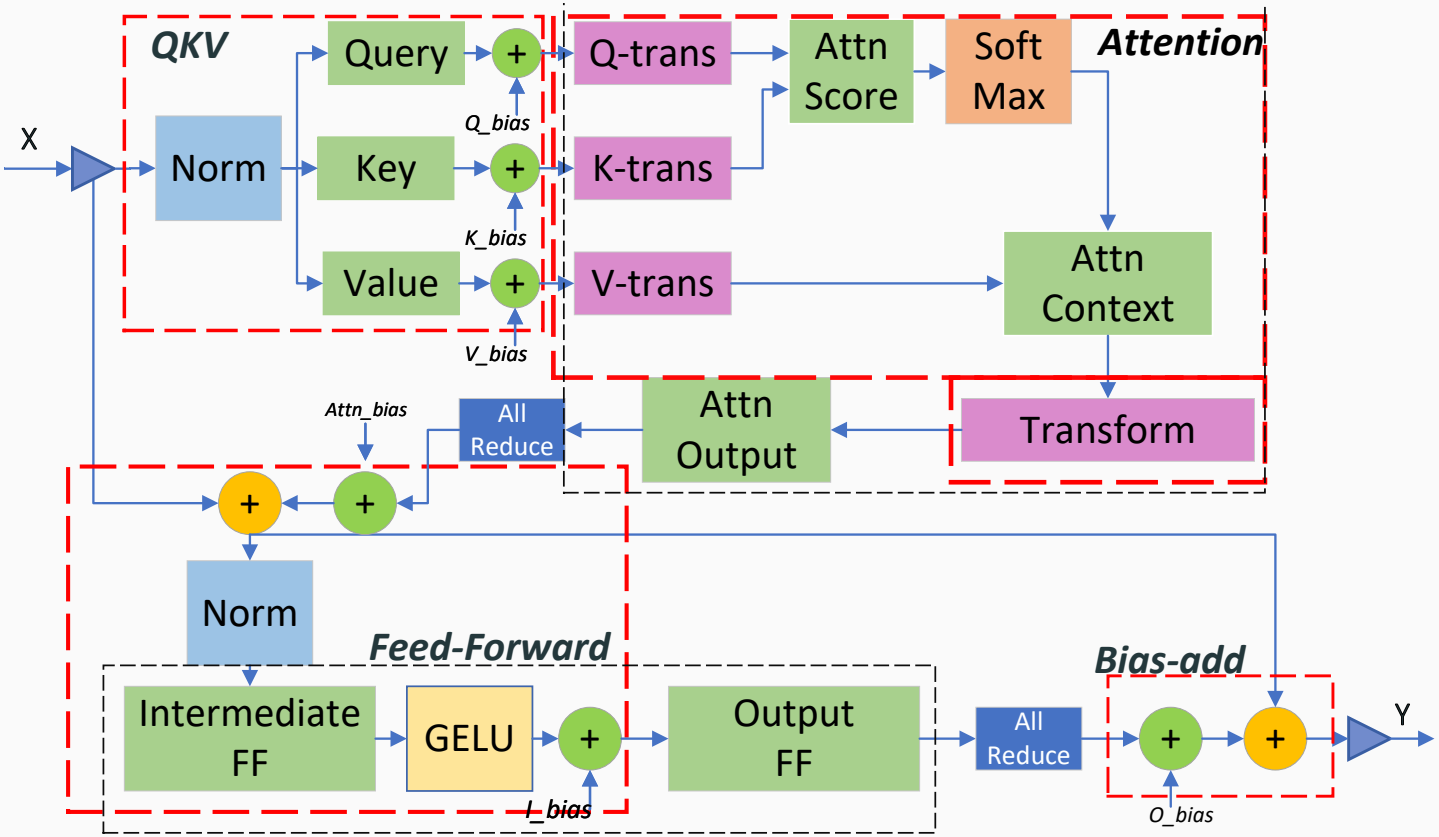


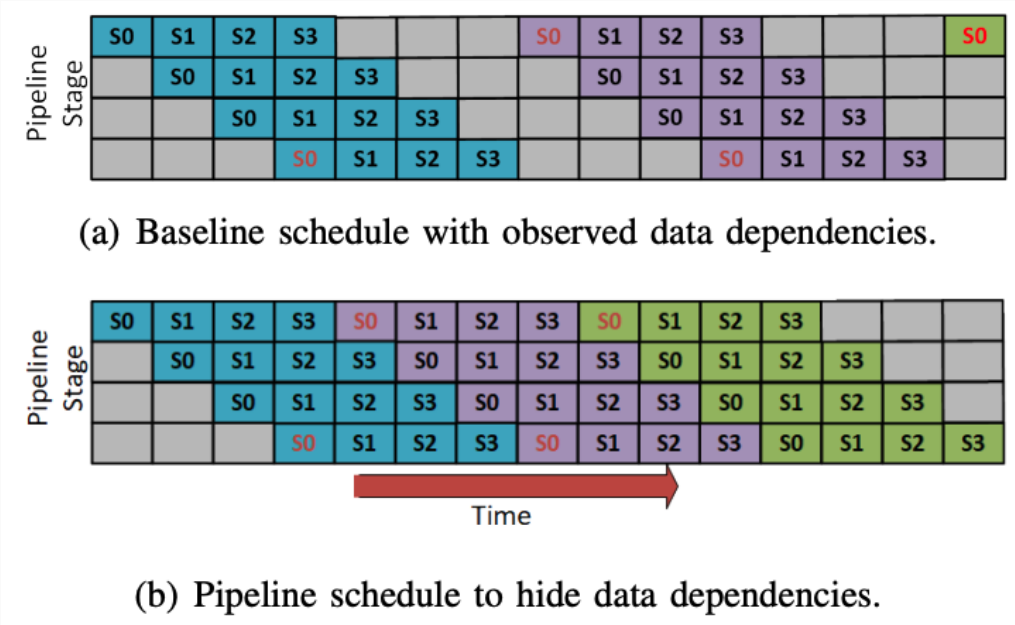
Figure 4: Fusion regions in DeepSpeed inference.

large-batch transformer: use CuBLAS for GEMM and keep GEMMs unfused.

DeepSpeed Inference: Inference on Many GPU System: I

Goals

1. reduce latency by leveraing aggregate memory bandwidth *across GPUs*. ← tensor parallelism in Megatron-LM
2. increase memory capacity by leveraging aggregate GPU memory *across multiple nodes*. ← optmized pipleine parallelism for inference



(a) Baseline schedule with observed data dependencies.

(b) Pipeline schedule to hide data dependencies.

Figure 5: A pipeline parallel schedule for generating the first three tokens for four sequences $S0$ to $S3$.

DeepSpeed Inference: Inference on Many GPU System: II

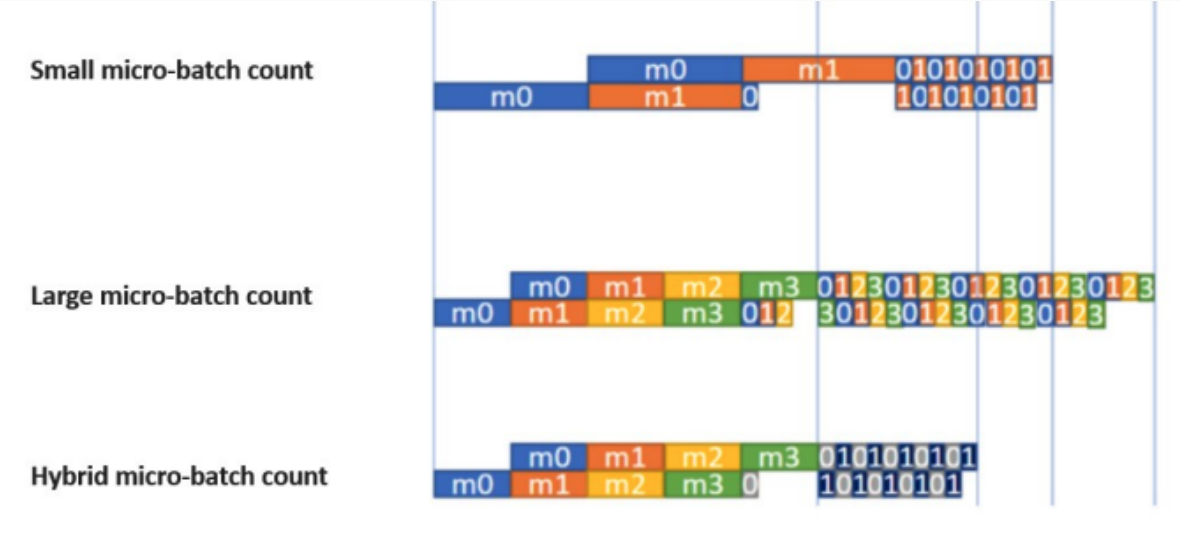


Figure 6: Different micro-batch for prompt-processing and token-generation.

- Suppose there are P pipeline stage, it is necessary to have P micro-batch to utilize all of the pipeline stages.
- The prompt processing stage has a large batch size that can saturate the GPU compute. macro-batches only affacts pipeline bubbles but not GPU execution time.
- The next token generation stage has a very small batch size, the performance is entirely bounded by reading weights. macro-batches number affacts execution time.
- **Solution**: large micro-batch count in prompt-processing and small micro-batch count in token generation.

ByteTransformer

Problem to Solve I: Oppornunities

To achieve low latency for an inference system that processes data online, what performance space can be leveraged?

1. the transformer block is memory bounded.
2. prompts have variable-length, padding them into a fixed length requires redundant computations.

Problem to Solve II: Variable-length challenges MHA

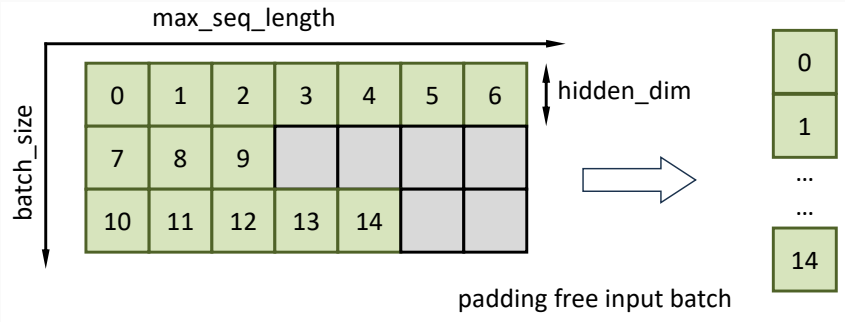


Figure 7: Padding free input batch.

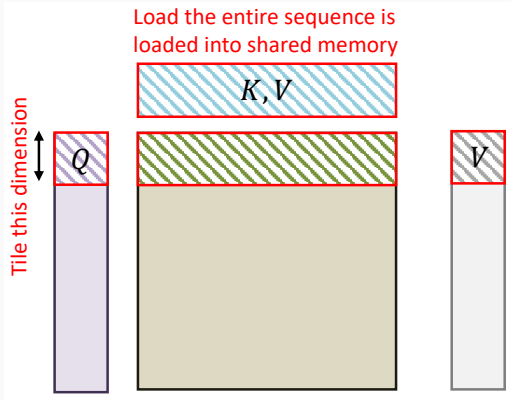


Figure 8: MHA with variable-length sequence batch.

Challenges: The computation of MHA is coupled with the sequence length dimension.

- To make it efficiently computed, batched GEMM is used. However, batched GEMM requires requires matrix multiplications to have a same shape.

Solution: Padding Free + Kernel Fusion

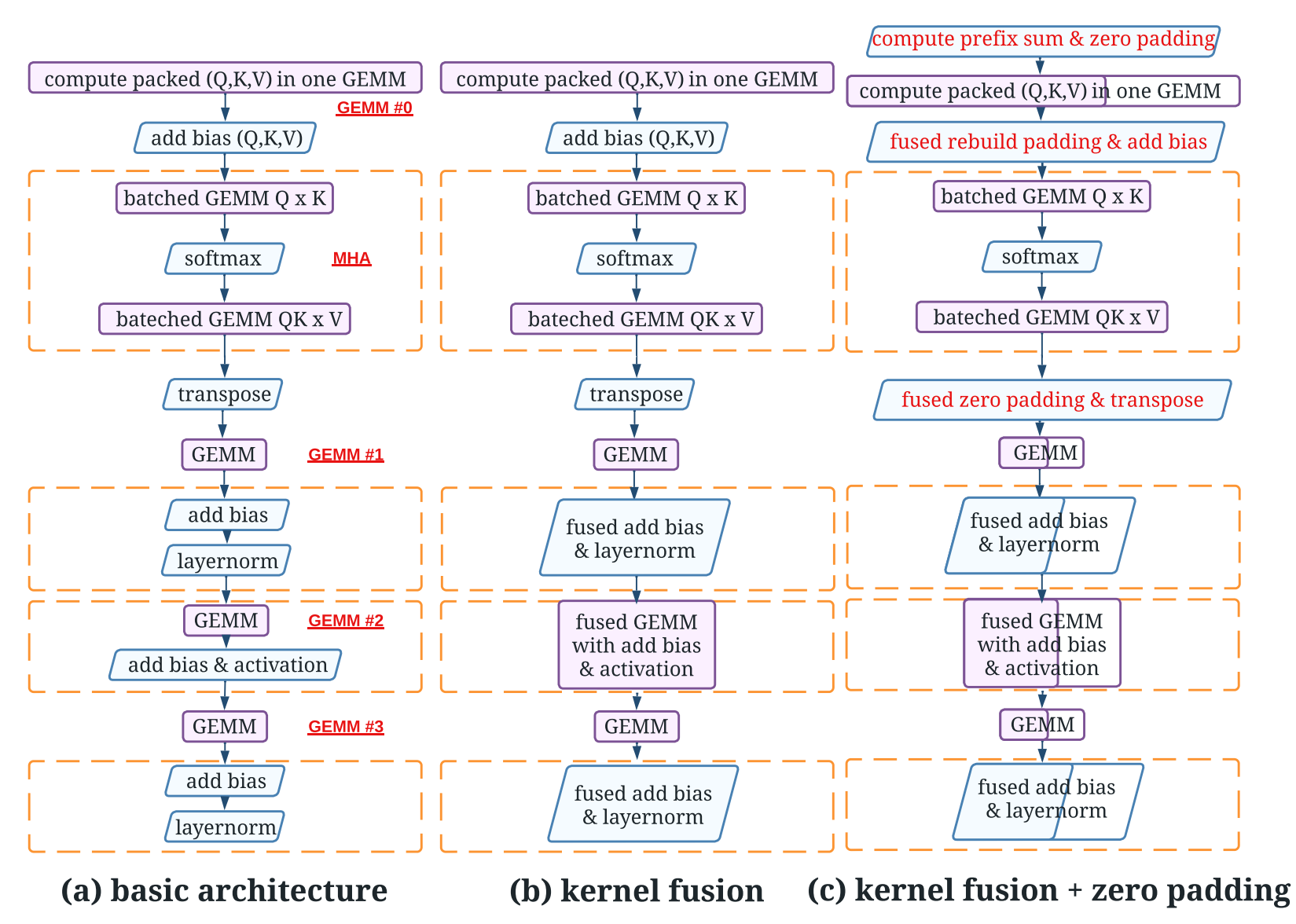


Figure 9: Compare ByteTransformer with other kernel fusion methods.

Solution: Padding and Padding Rebuild for MHA

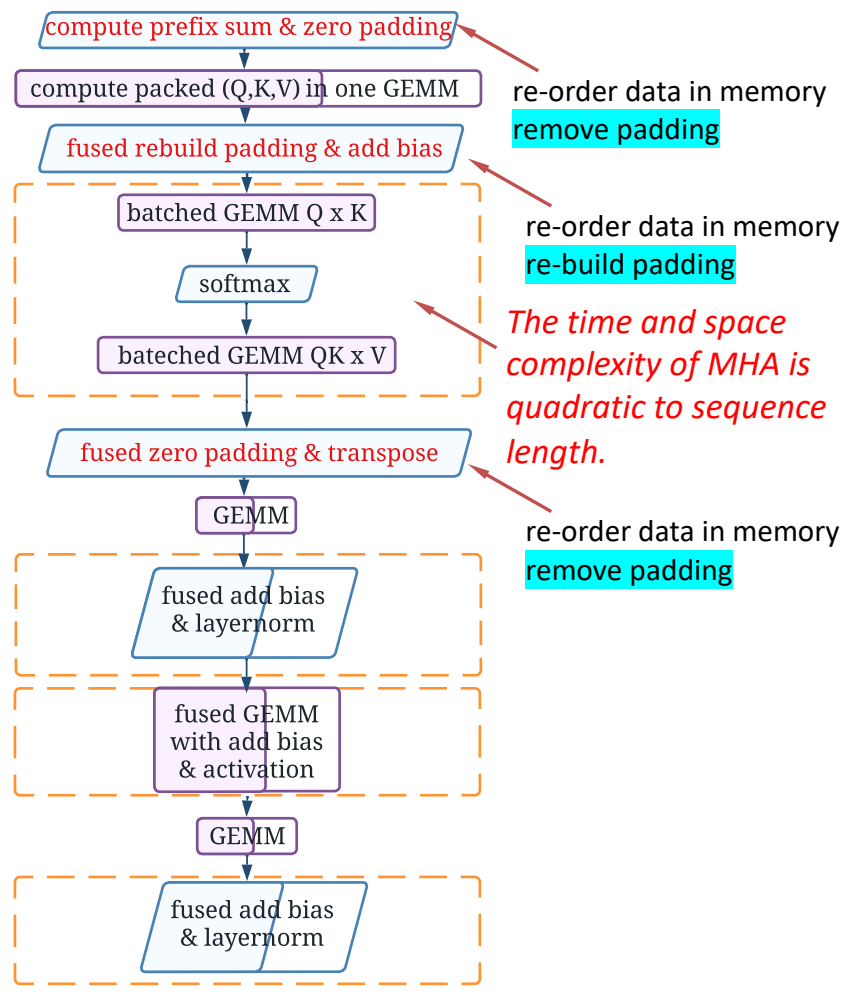


Figure 10: Overview of the ByteTransformer.

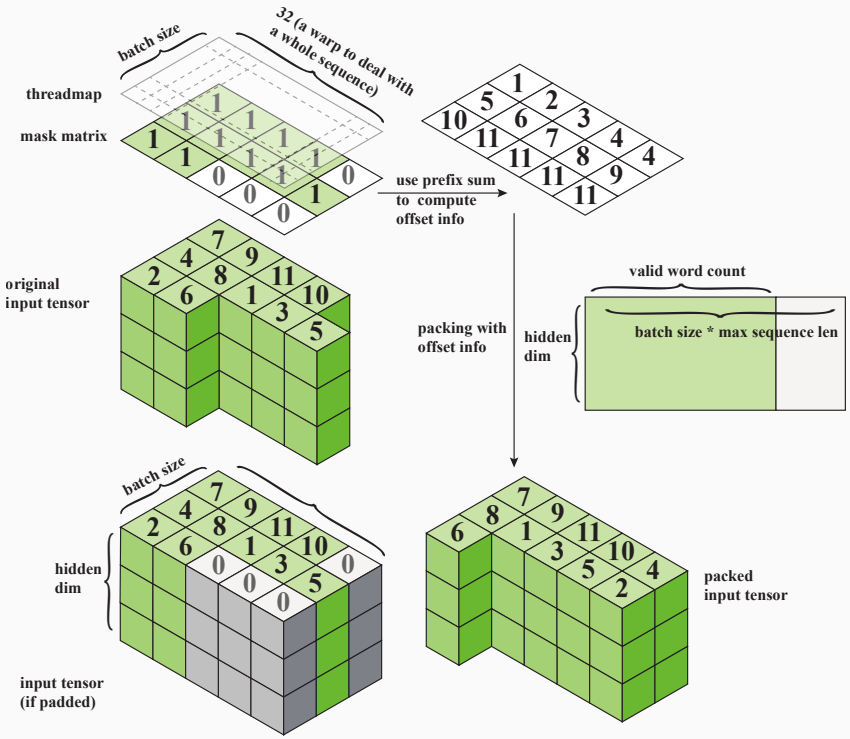


Figure 11: The padding-free operation.

Optimized MHA Kernel: Short Sequences (≤384 tokens)

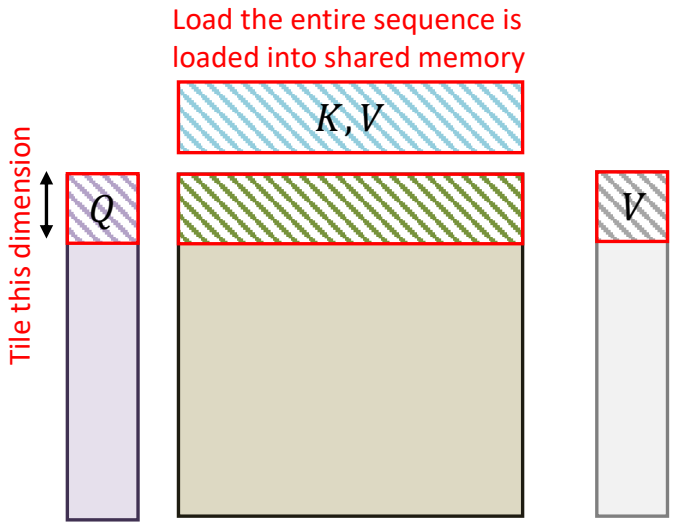


Figure 12: Workload assigned to a CTA.

- Tile Q length, and move the entire K, V sequence into shared memory.
- Keep the entire MHA computation on shared memory.

Optimized MHA Kernel: long Sequences (>384 tokens)

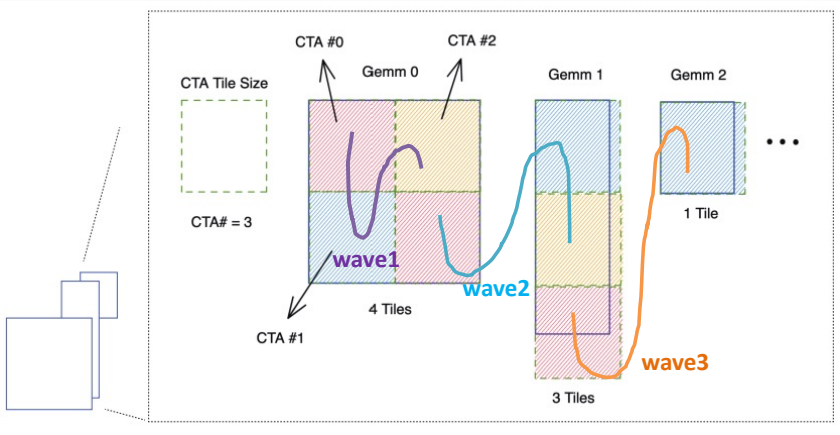


Figure 13: The grouped GEMM operation.

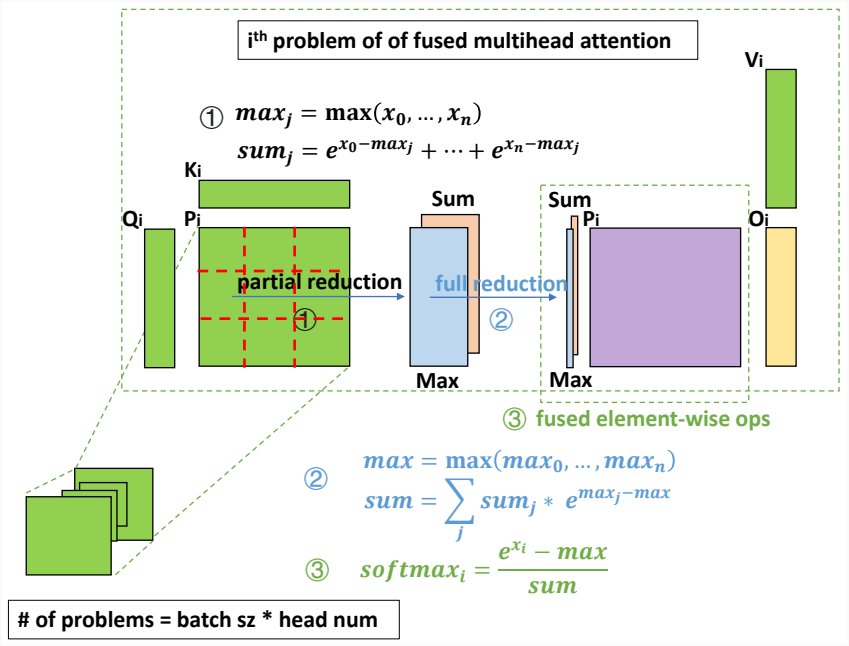
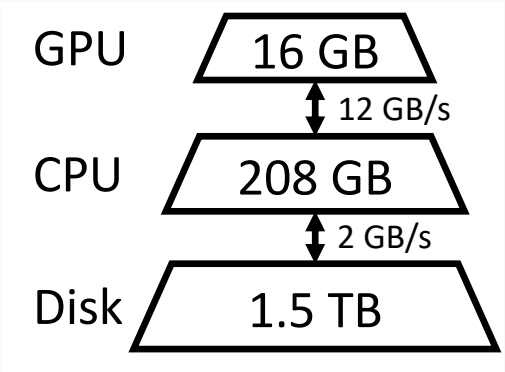


Figure 14: Grouped GEMM-based fused MHA.

FlexGen

FlexGen: Goal

- LLMs is too large to fit into the memory of a single GPU, design efficient offloading strategies for high-throughput (sacrifice latency) generative inference on a single commodity GPU.
- Run an LLM with limited GPU memory, offload it to secondary storage and perform computation part-by-part by partially loading it.



FlexGen: Problem Formulation

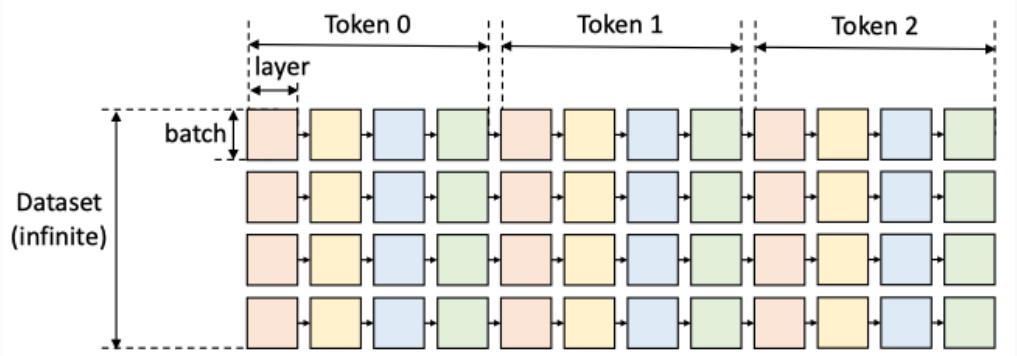


Figure 15: Computational graph of LLM inference.

The generative inference with offloading is a graph traversal problem. Figure out a valid path that traverses all squares, while subject to the constraints:

1. A square can only be computed if all squares to its left on the same row were computed.
2. To compute a square on a device, all its inputs must be loaded to the same device.
3. The activations should be stored until its right sibling is computed.
4. The KV cache should be stored until the rightmost square on the same.
5. At any time, the total size of tensors stored on a device cannot exceed its memory capacity.

Block Schedule: Throughput vs. Latency

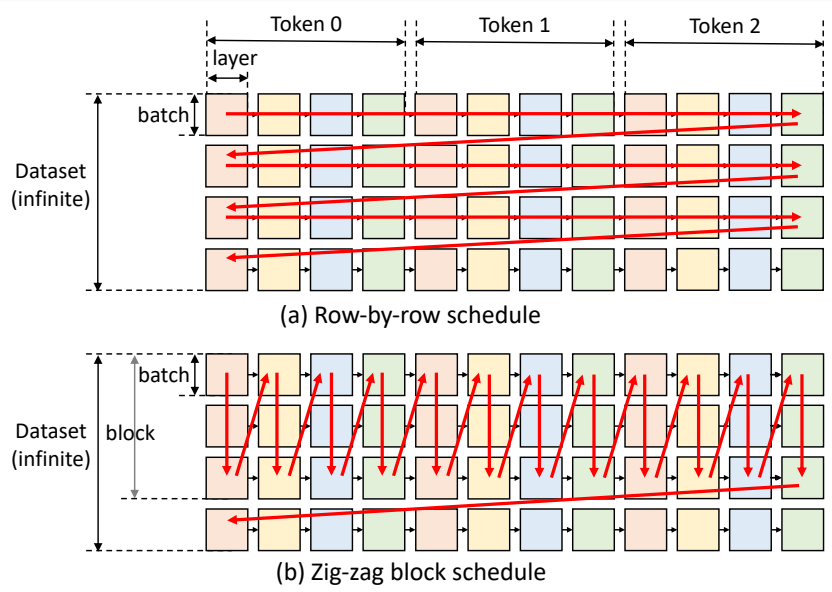


Figure 16: Virtualizing the block scheduling.

Algorithm 1 Block Schedule with Overlapping

```
for i = 1 to generation_length do
  for j = 1 to num_layers do
    // Compute a block with multiple GPU batches
    for k = 1 to num_GPU_batches do
      // Load the weight of the next layer
      load_weight(i, j + 1, k)
      // Store the cache and activation of the prev batch
      store_activation(i, j, k − 1)
      store_cache(i, j, k − 1)
      // Load the cache and activation of the next batch
      load_cache(i, j, k + 1)
      load_activation(i, j, k + 1)
      // Compute this batch
      compute(i, j, k)
      // Synchronize all devices
      synchronize()
    end for
  end for
end for
```

batch num

GPU batch size

Figure 17: The block scheduling algorithm.

The Search Space

1. **Block scheduling algorithm** searches two hyper parameters: batch size and batch nummbers.
2. **Tensor placement** searches how to store weights, activations, K-V caches on device.
   - wg, wc, wd define the percentages of weights stored on GPU, CPU, and disk
   - hg, hc, hd define the percentages of activations stored on GPU, CPU, and disk
   - cg, cc, cd define the percentages of k-V cache stored on GPU, CPU, and disk
3. **Computation delegation**: using CPU compute can still be beneficial in some cases.
   - The size of move KV cache is $b \times s \times d \times 4$;
   - The size of move activation is $b \times d$;
   - Compute attention scores on CPU reduce I/O by $s\times$, if the accociated KV cache is not stored on GPU, compute attention scores on CPU would be better.

Cost Model and Policy Search

Cost model: a mixture of latency and peak memory usage prediction

- Latency prediction
  1. I/O term is estimated by summing by all the I/O events, computation term is estimated by summing up all computation events. Assume perfect overlapping

$$T = T_{pre} \cdot l + T_{gen} \cdot (n-1) \cdot l$$

$$T_{pre} = \max(ctog^p, gtoc^p, dtoc^p, ctod^p, comp^p)$$

$$T_{gen} = \max(ctog^g, gtoc^g, dtoc^g, ctod^g, comp^g)$$

- The ILP to solve 11 variable:

$$\begin{aligned} \min_{p} \quad & T/bls \\ \text{s.t.} \quad & gpu\ peak\ memory < gpu\ mem\ capacity \\ & cpu\ peak\ memory < cpu\ mem\ capacity \\ & disk\ peak\ memory < disk\ mem\ capacity \\ & wg + wc + wd = 1 \\ & cg + cc + cd = 1 \\ & hg + hc + hd = 1 \end{aligned} \tag{1}$$

block size

placement

Figure 18: The ILP problem.