THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RELAXED SYSTEM LAB

MoE- & Long Sequence-Parallel Training

COMP4901Y

Binhang Yuan

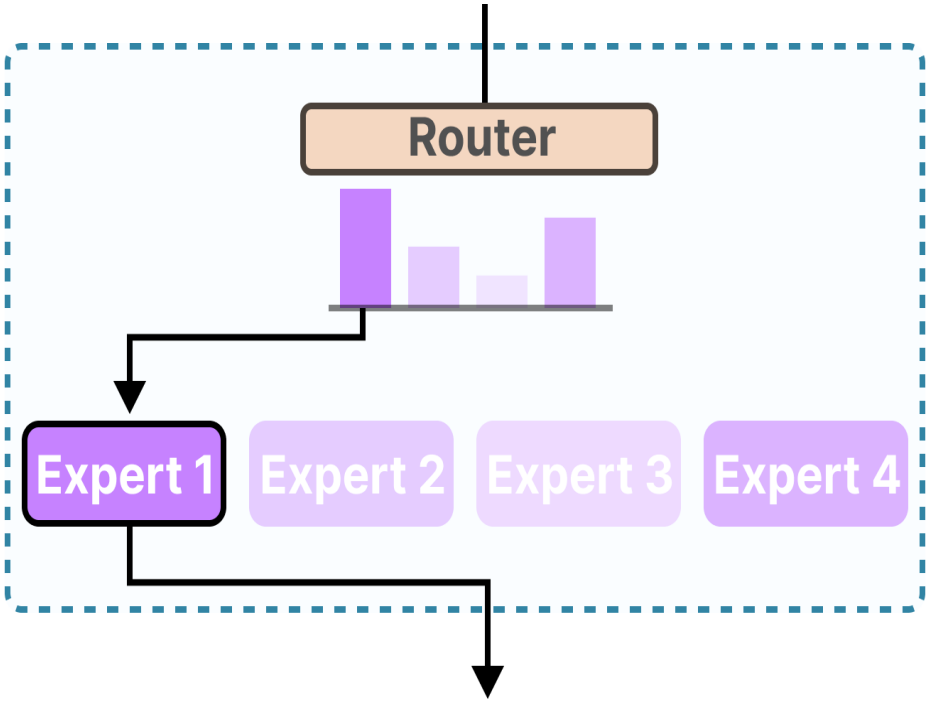
RELAXED
SYSTEM LAB

MoE Parallelism

Mixture of Expert

RELAXED SYSTEM LAB

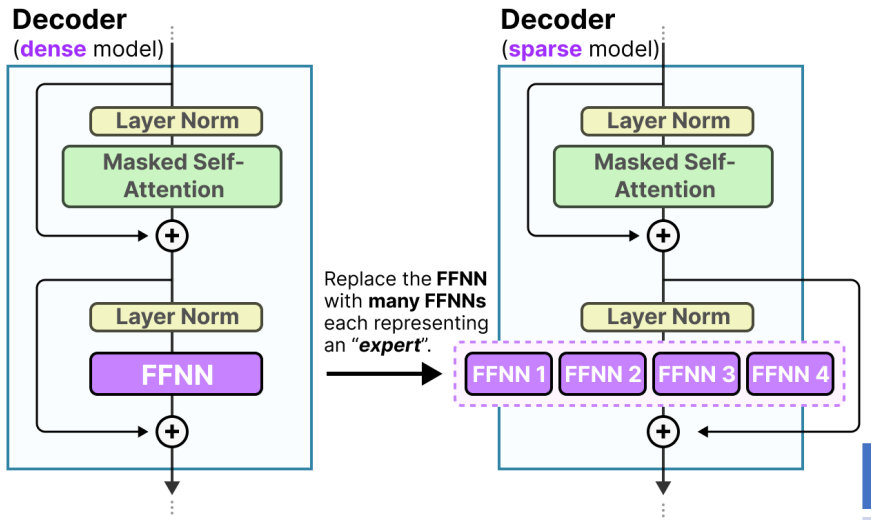
- Intuitive idea of MoE:
  - Divide a large model or layer into multiple smaller sub-networks, known as "experts," each specializing in different aspects of the input data or task.
- MoE components:
  - **Router** (Gating Network): This component determines which experts to activate for a given input.
  - **Experts**: individual neural sub-networks trained to handle specific subsets or patterns within the data.



- Benefit:
  - Unlike traditional dense models, where all parameters are activated for every input, MoE models use conditional computation to activate only a subset of experts relevant to a specific input.

Replace MLP with MoE in TransformerBlocks

RELAXED SYSTEM LAB



- B is the batch size;
- L is the sequence length;
- D is the model dimension;
- E is the number of experts;
- K is the number of activated experts;
- $H_E \ll D$ is the intermediate dimension of each expert.
- Noted that this formulation is only for explaining the computation, in practice we use the routing results as the input for each expert.

Dense MLP

Computation	Input	Output
$A = \text{Out}\, W^1$	$\text{Out} \in \mathbb{R}^{B\times L\times D}$, $W^1 \in \mathbb{R}^{D\times 4D}$	$A \in \mathbb{R}^{B\times L\times 4D}$
$A' = \text{relu}(A)$	$A \in \mathbb{R}^{B\times L\times 4D}$	$A' \in \mathbb{R}^{B\times L\times 4D}$
$X' = A'W^2$	$A' \in \mathbb{R}^{B\times L\times 4D}$, $W^2 \in \mathbb{R}^{4D\times D}$	$X' \in \mathbb{R}^{B\times L\times D}$

MoE

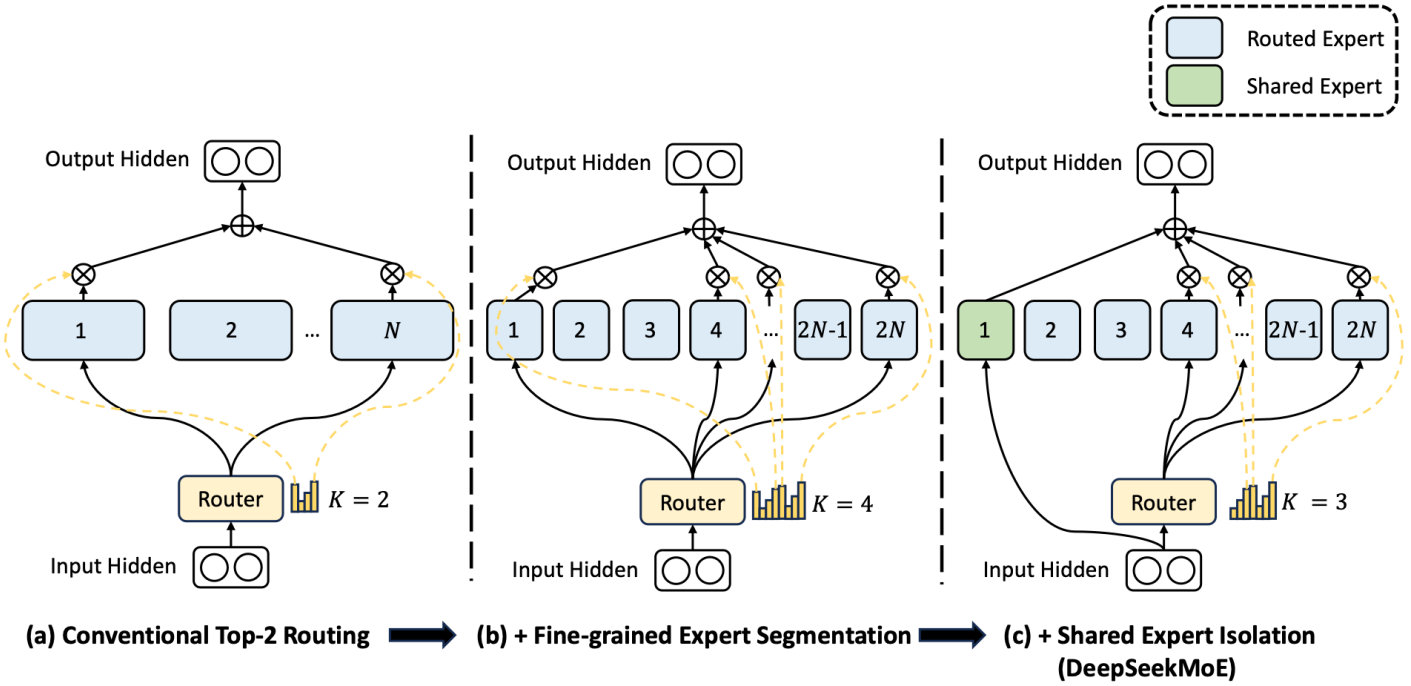
Computation	Input	Output
$S = \text{softmax}(\text{Out}\, W^G)$	$\text{Out} \in \mathbb{R}^{B\times L\times D}$, $W^G \in \mathbb{R}^{D\times E}$	$S \in \mathbb{R}^{B\times L\times E}$
$G_{i,j,k} = \begin{cases} S_{i,j,k} \text{ if } S_{i,j,k} \text{ in TopK}(S_{i,j,:}) \\ 0, \text{otherwise} \end{cases}$	$S \in \mathbb{R}^{B\times L\times E}$	$G \in \mathbb{R}^{B\times L\times E}$
$A_k = \text{Out}\, W_k^1, k = 1,2,\dots,E$	$\text{Out} \in \mathbb{R}^{B\times L\times D}$, $W_k^1 \in \mathbb{R}^{D\times H_E}$	$A_k \in \mathbb{R}^{B\times L\times H_E}$
$A_k{}' = \text{relu}(A_k), k = 1,2,\dots,E$	$A_k \in \mathbb{R}^{B\times L\times H_E}$	$A_k{}' \in \mathbb{R}^{B\times L\times H_E}$
$B_k = A_k' W^2, k = 1,2,\dots,E$	$A_k' \in \mathbb{R}^{B\times L\times H_E}$, $W_k^2 \in \mathbb{R}^{H_E\times D}$	$B_k \in \mathbb{R}^{B\times L\times D}$
$X' = \sum_{k=1}^{E} G_{:,:,k} * B_k$	$G \in \mathbb{R}^{B\times L\times E}$, $B_k \in \mathbb{R}^{B\times L\times D}$	$X' \in \mathbb{R}^{B\times L\times D}$

DeepSeek MoE

RELAXED SYSTEM LAB

- **Finely segmentation**: segmenting the experts into mN ones and activating mK from them is better than segmenting the experts into N ones and activating K from them, which allows for a more flexible combination of activated experts;
- **Shared expert**: isolating some experts as shared ones, aiming at capturing common knowledge and mitigating redundancy in routed experts.

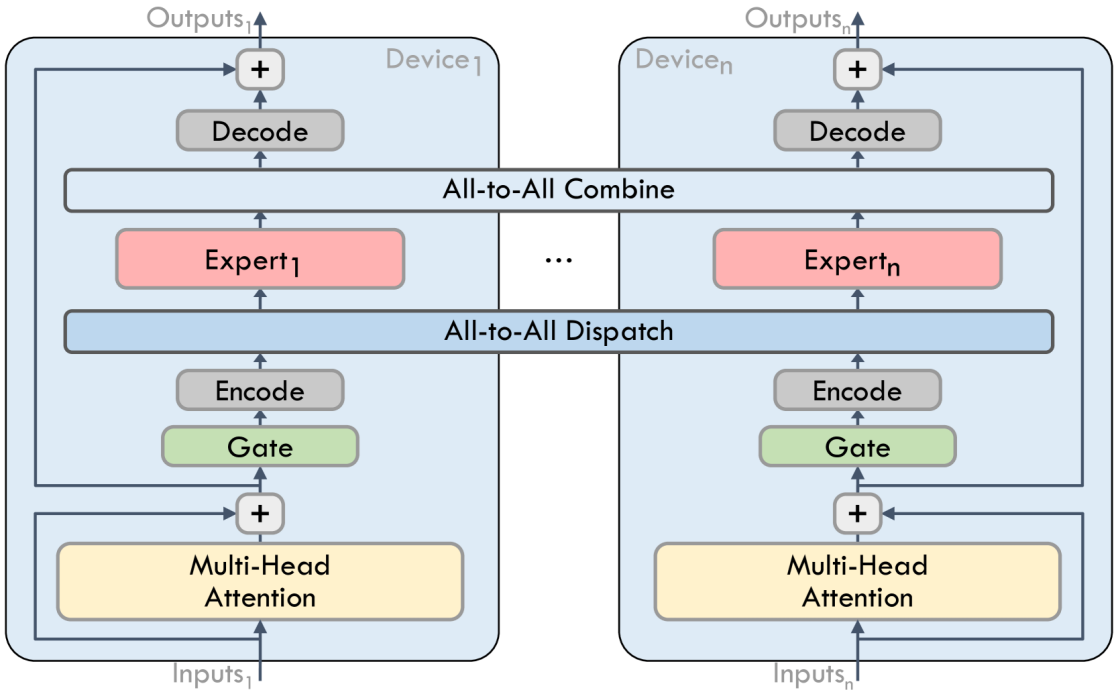
deepseek



MoE Parallelism

RELAXED SYSTEM LAB

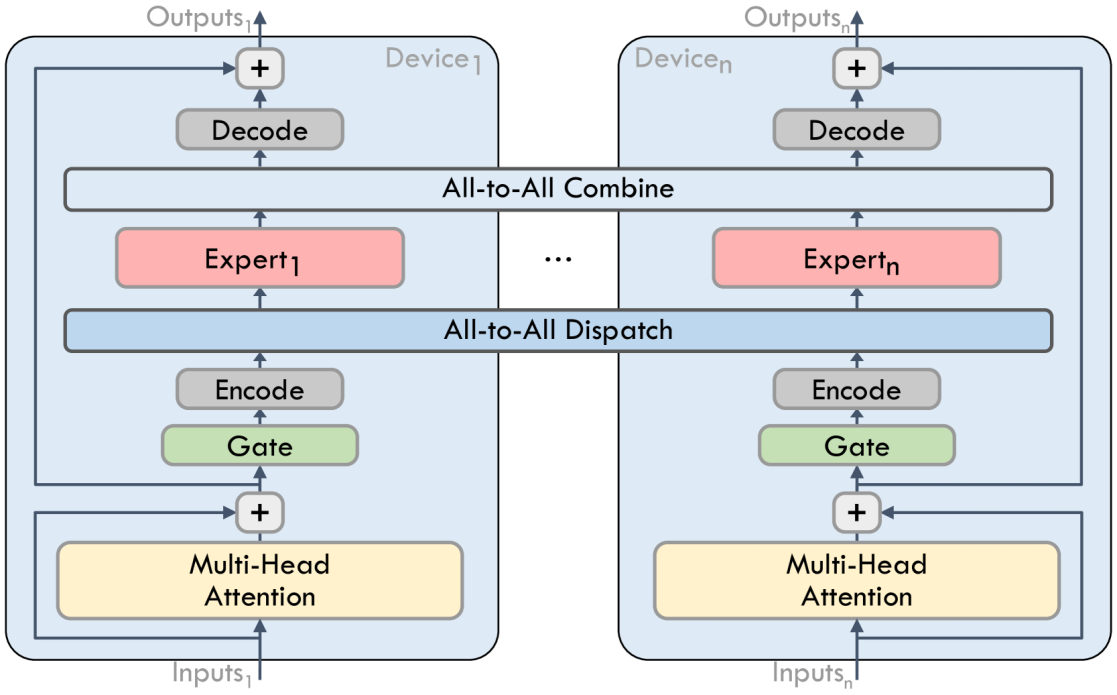
- How to parallelize the MoE training?
- Basic idea: extend standard data parallelism:
  - The experts (FFNs) are placed on different GPUs, e.g., if there is E expert and the expert parallel degree is d_{EP}, then each GPU handles $\frac{E}{d_{EP}}$ experts in parallel;
  - The rest parts of the model are duplicated on those GPUs running standard data parallelism.



MoE Parallelism

RELAXED SYSTEM LAB

- In the forward pass:
  - Two **AlltoAll** operations to communicate the activations:
    - One **AlltoAll** operation dispatches the routed activations in the current data batch to the corresponding experts;
    - One **AlltoAll** operation combines the computed expert output in the data batch that needs to be processed by the device.
  - Notice that there could be a little additional overhead to first share the shape of the dispatch tensors.



AlltoAll Communication

RELAXED SYSTEM LAB

torch.distributed.all_to_all(*output_tensor_list*, *input_tensor_list*, *group=None*, *async_op=False*) [SOURCE]

Scatters list of input tensors to all processes in a group and return gathered list of tensors in output list.

Complex tensors are supported.

Parameters

- **output_tensor_list** (*list*[*Tensor*]) – List of tensors to be gathered one per rank.
- **input_tensor_list** (*list*[*Tensor*]) – List of tensors to scatter one per rank.
- **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used.
- **async_op** (*bool, optional*) – Whether this op should be an async op.

Returns

Async work handle, if async_op is set to True. None, if not async_op or if not part of the group.

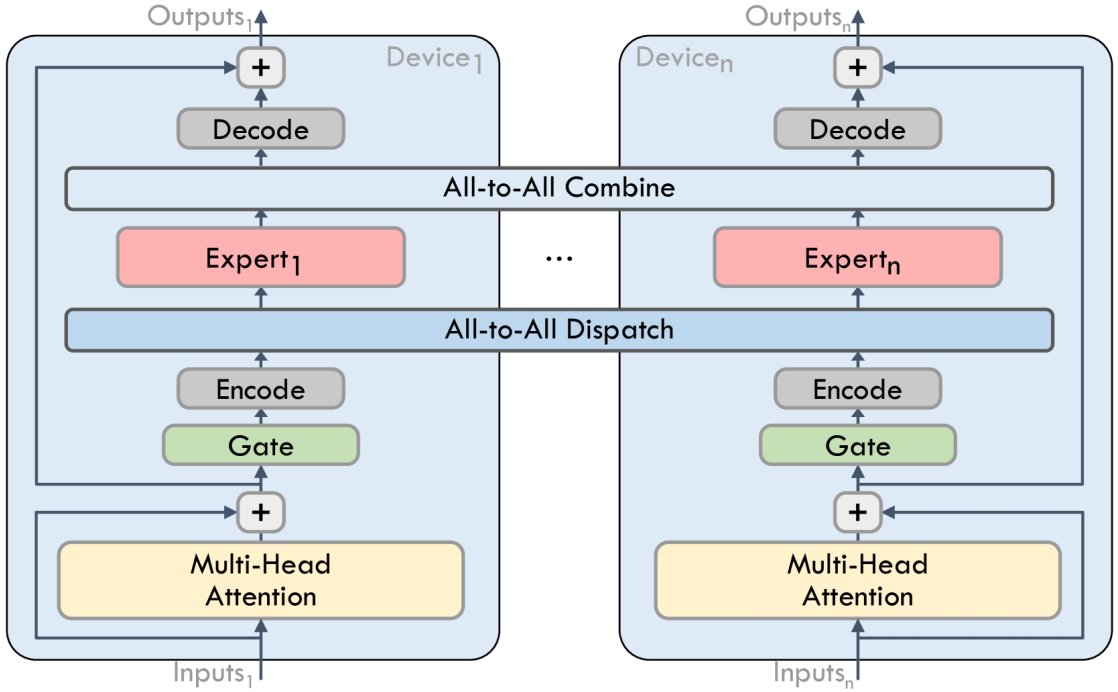
```
>>> input
tensor([0, 1, 2, 3, 4, 5])                                       # Rank 0
tensor([10, 11, 12, 13, 14, 15, 16, 17, 18])                     # Rank 1
tensor([20, 21, 22, 23, 24])                                     # Rank 2
tensor([30, 31, 32, 33, 34, 35, 36])                             # Rank 3
>>> input_splits
[2, 2, 1, 1]                                                     # Rank 0
[3, 2, 2, 2]                                                     # Rank 1
[2, 1, 1, 1]                                                     # Rank 2
[2, 2, 2, 1]                                                     # Rank 3
>>> output_splits
[2, 3, 2, 2]                                                     # Rank 0
[2, 2, 1, 2]                                                     # Rank 1
[1, 2, 1, 2]                                                     # Rank 2
[1, 2, 1, 1]                                                     # Rank 3
>>> input = list(input.split(input_splits))
>>> input
[tensor([0, 1]), tensor([2, 3]), tensor([4]), tensor([5])]                   # Rank 0
[tensor([10, 11, 12]), tensor([13, 14]), tensor([15, 16]), tensor([17, 18])] # Rank 1
[tensor([20, 21]), tensor([22]), tensor([23]), tensor([24])]                 # Rank 2
[tensor([30, 31]), tensor([32, 33]), tensor([34, 35]), tensor([36])]         # Rank 3
>>> output = ...
>>> dist.all_to_all(output, input)
>>> output
[tensor([0, 1]), tensor([10, 11, 12]), tensor([20, 21]), tensor([30, 31])]   # Rank 0
[tensor([2, 3]), tensor([13, 14]), tensor([22]), tensor([32, 33])]           # Rank 1
[tensor([4]), tensor([15, 16]), tensor([23]), tensor([34, 35])]              # Rank 2
[tensor([5]), tensor([17, 18]), tensor([24]), tensor([36])]                  # Rank 3
```

Each chunk is **NOT** necessary to have the same shape.

MoE Parallelism

RELAXED SYSTEM LAB

- In the backward pass:
  - Two **AlltoAll** operations to communicate the corresponding gradients of the activations back to the original device.
  - **AllReduce** operations to synchronize the gradients of non-expert model weights.



MoE Load Balance

RELAXED SYSTEM LAB

- Automatically learned routing strategies may encounter the issue of load imbalance:
  - Routing collapse: the model always selects only a few experts, preventing other experts from sufficient training.
  - Computation inefficiency: the computation load can vary in different devices, where the device handles the most tokens becomes the bottleneck.
- Solution: add some load balance loss during training (T is the total tokens in a batch, E is the number of experts, $S_{t,i}$ is the routing score for token t of expert i):

$$\mathcal{L}_{bal} = \sum_{i=1}^{E} f_i P_i$$

$$f_i = \frac{E}{KT} \sum_{t=1}^{T} \mathbb{I}(\text{Token } t \text{ selects Expert } i),$$

$$P_i = \sum_{t=1}^{T} S_{t,i}$$

Summary of MoE

RELAXED
SYSTEM LAB

- Most of the state-of-the-art model is based on MoE architectures:
  - E.g., Deepseek-V3, Mixtral, rumor about OpenAI GPT-4.
- MoE parallelism can be viewed as an extension of data parallelism:
  - Experts are distributed across different devices;
  - Non-expert parts are running standard data parallelism.
- Issue about load-balance.

Parallelism for Long Sequence Training

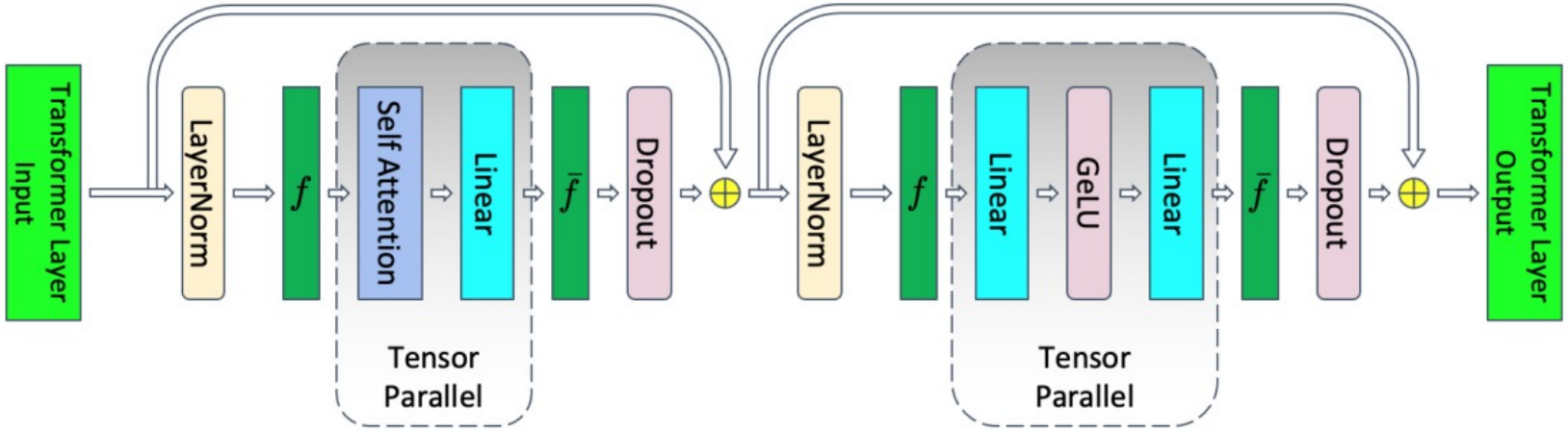
Issue about Long Context

RELAXED SYSTEM LAB

- The current LLM can be equipped with an extremely long context length, e.g., 128K, or 1M tokens;
- Even we have only one sample in the batch, there is heavy computations. We have to extend the current parallel strategies to:
  - Distributed the computation load;
  - Reduce the memory footprint.

Megatron Sequence Parallelism

RELAXED SYSTEM LAB

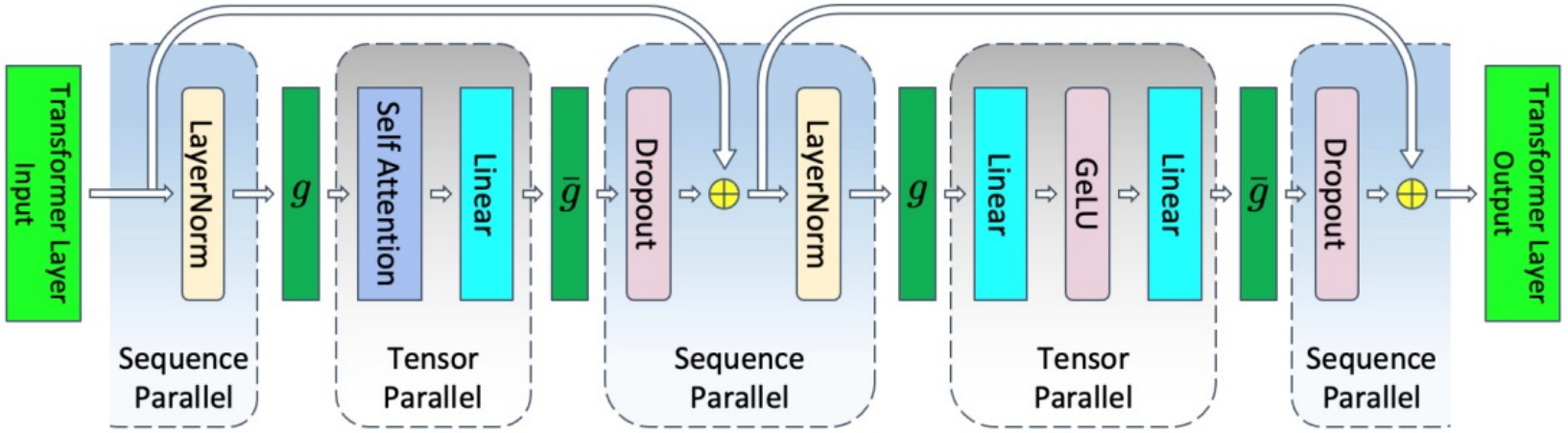


In standard tensor model parallelism:
f is no operation in the forward pass and **AllReduce** in the backward pass.
$\bar{f}$ is **AllReduce** in the forward pass and no operation in the backward pass.

- Key observation:
  - All the activations in the non-tensor parallel regions of a transformer layer are duplicated, e.g., after the **AllReduce** operation.
  - These operations are independent along the sequence dimension.
  - This characteristic allows us to partition these regions along the *sequence dimension*.

Megatron Sequence Parallelism

RELAXED SYSTEM LAB



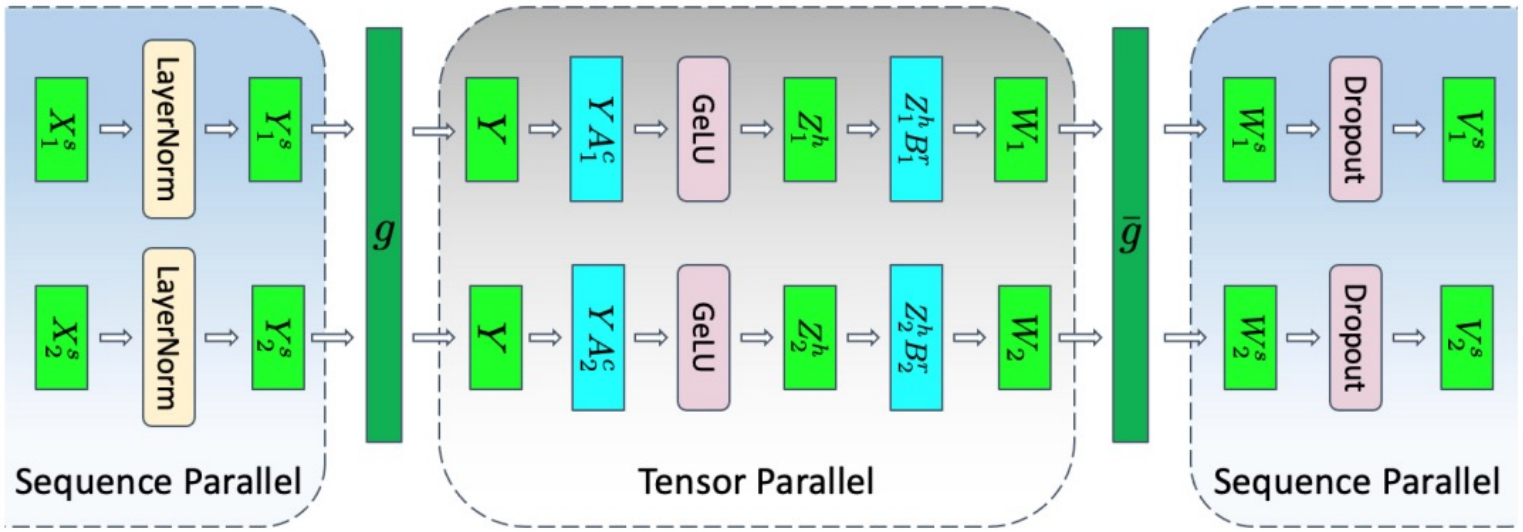
In tensor model parallelism combined with sequence parallelis:
g is **AllGather** operation in the forward pass and **ReduceScatter** in the backward pass.
$\bar{g}$ is **ReduceScatter** in the forward pass and **AllGather** operation in the backward pass.

- Sequence parallelism is an extension of the previous tensor model parallelism:
- Partitioning along the sequence dimension reduces the memory required for the activations. This extra level of parallelism introduces new collective communication paradigm.

Megatron Sequence Parallelism

RELAXED SYSTEM LAB

- This is the concrete partition of the MLP with a parallel degree as 2.



g is **AllGather** operation in the forward pass and **ReduceScatter** in the backward pass.
$\bar{g}$ is **ReduceScatter** in the forward pass and **AllGather** operation in the backward pass.

Deepspeed-Ulysses

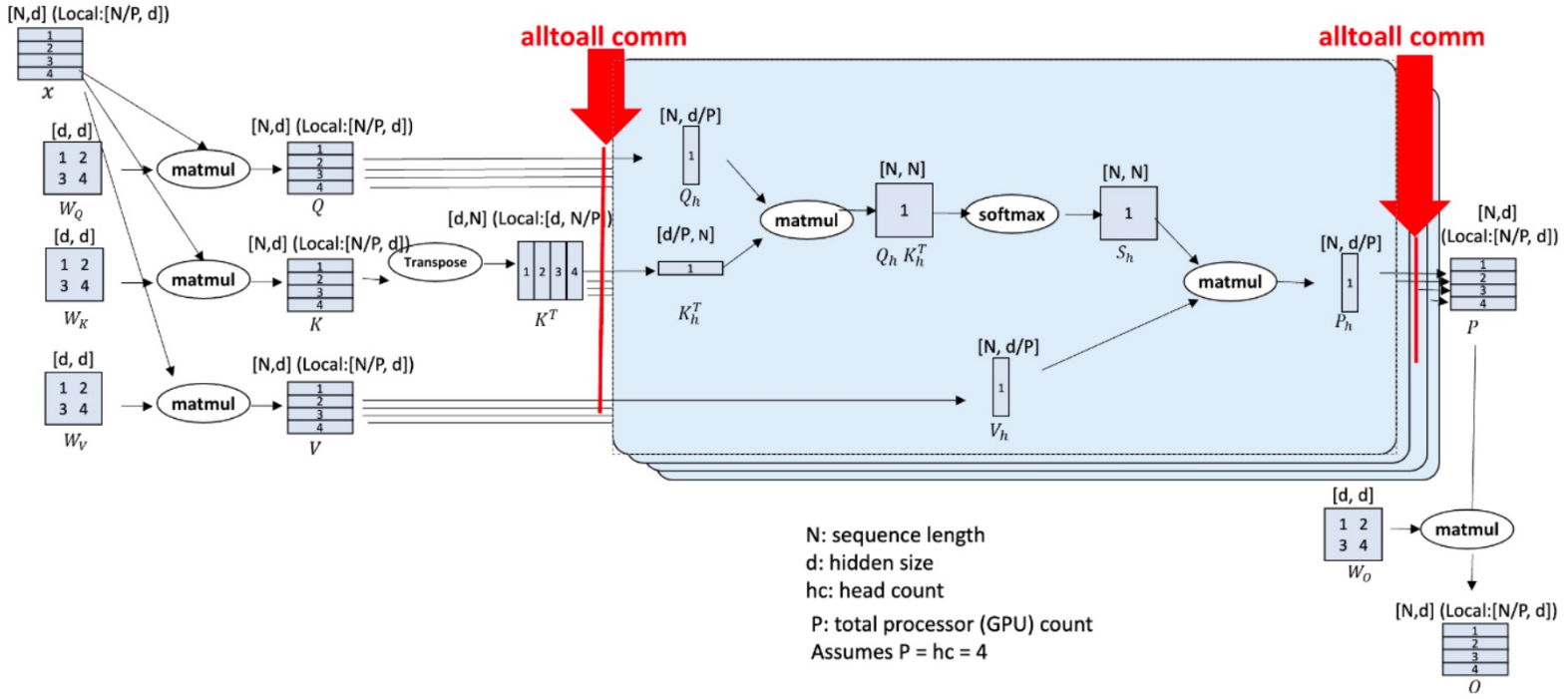
RELAXED SYSTEM LAB

- Limitation of Megatron sequence parallelism: In Megatron sequence parallelism, it has to be combined with tensor model parallelism.
- Deepspeed proposed another way of distributing the long sequence parallel training:
  - A modification of data parallelism: instead of partition the batch by data samples, Ulysses partitions the batch by sequence dimension.
  - Model parameters are duplicated among all devices.
  - Activation tensors along the sequence dimension across multiple GPUs.
  - Everything except attention computation is straight-forward. The core question is about *how to accomplish the attention computation if we split the sequence?*

Deepspeed-Ulysses

RELAXED SYSTEM LAB

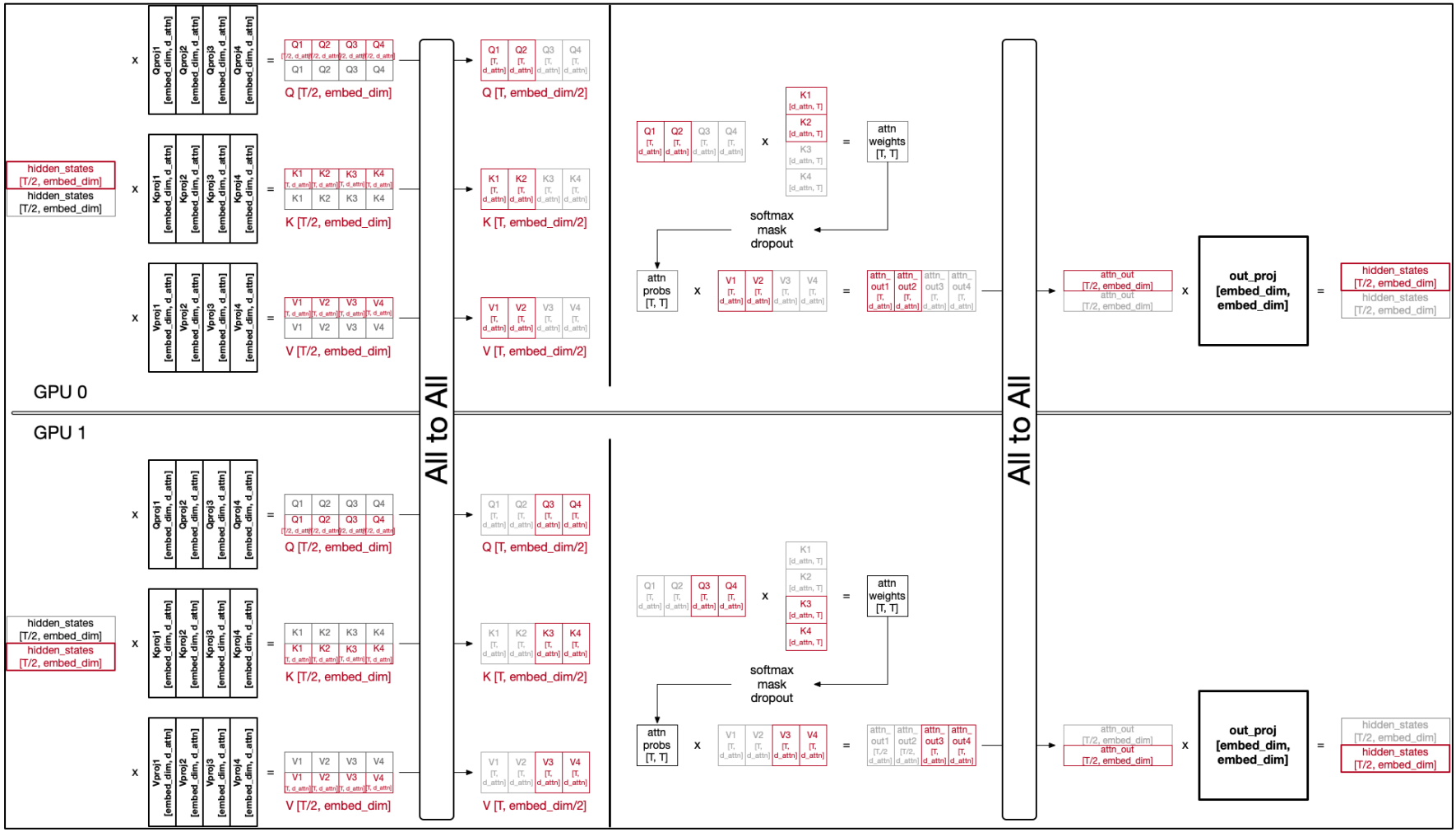
- **Core idea：**
  - Partition the computation of attention heads among different GPUs;
  - Use one All-to-All to shuffle the input Query, Key, Value;
  - Use one All-to-All to collect the corresponding output.



Deepspeed-Ulysses

RELAXED SYSTEM LAB

- **Core idea:**
  - Partition the computation of attention heads among different GPUs;
  - Use one All-to-All to shuffle the input Query, Key, Value;
  - Use one All-to-All to collect the corresponding output.



https://insujang.github.io/2024-09-20/introducing-context-parallelism

Deepspeed-Ulysses Communication

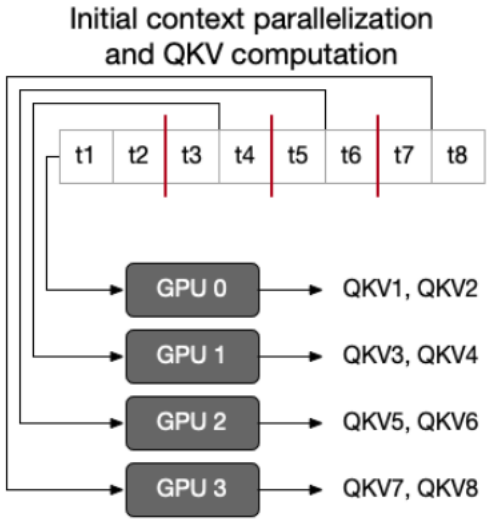
RELAXED SYSTEM LAB

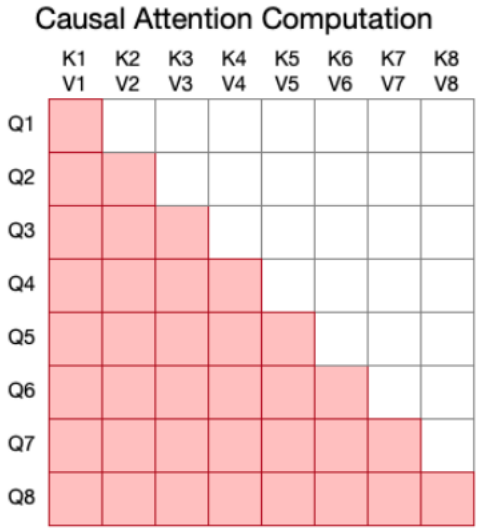
- Forward pass:
  - Before computing the attention mechanism, one **AlltoAll** operation redistributes the partitioned queries, keys, and values such that each GPU receives the full sequence data for the device's subset of attention heads.
  - After computing the attention mechanism, one **AlltoAll** operation to recover the partition of activations along the sequence dimension.
- Backward pass:
  - Two corresponding **AlltoAll** operations to send back the corresponding gradients of the activations.
  - Standard **AllReduce** operations for all gradient synchronization of the model parameters in data parallelism.

Ring Attention/Megatron Context Parallelism

RELAXED SYSTEM LAB

- Limitation of Deepspeed-Ulysses: the parallel degree is limited by the number of attention heads, we cannot add more devices to further scaling out the computation.
- Ring attention computation for the attention mechanism:
  - Split the data by sequence dimension;
  - Iteratively accumulate local attention results by iterating Q and K/V as a nested loop;
  - Each GPU holds a portion of queries, and sends its keys and values to the next GPU and receives them from the previous GPU (ring).

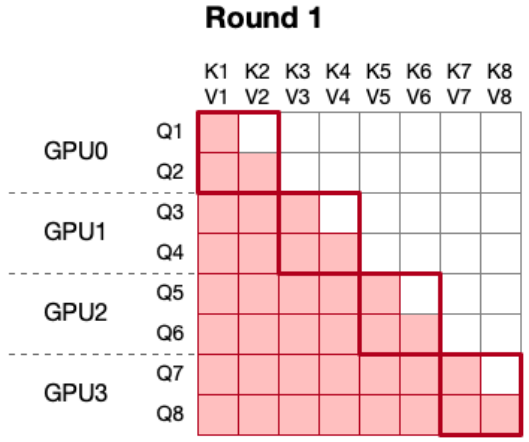


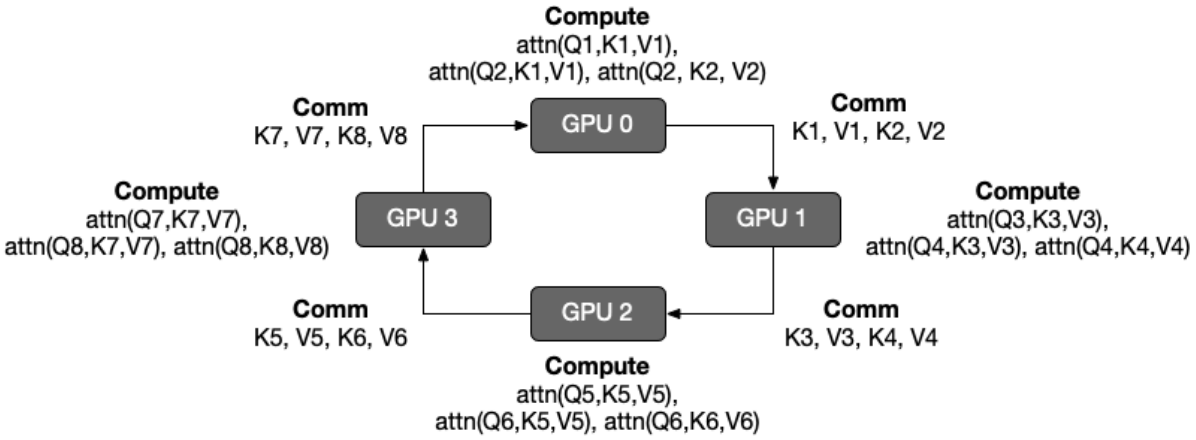


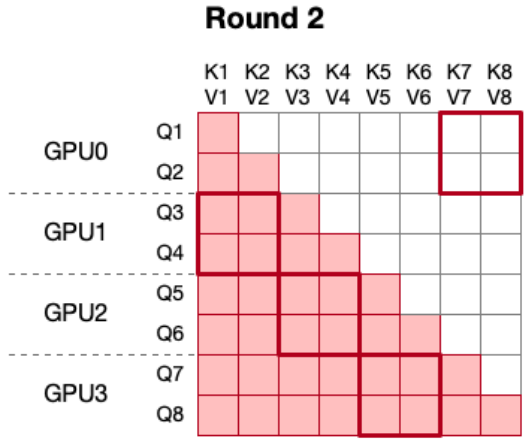
https://insujang.github.io/2024-09-20/introducing-context-parallelism

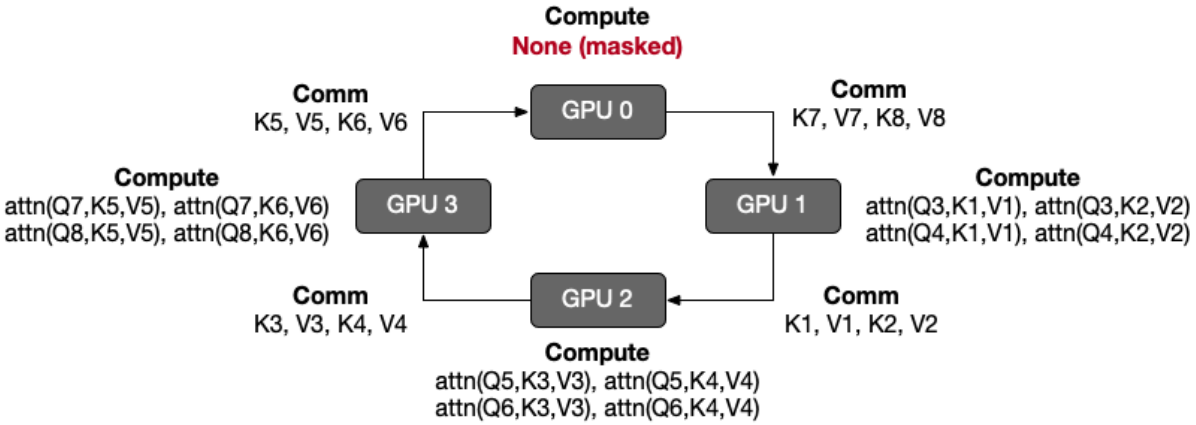
Ring Attention

RELAXED SYSTEM LAB





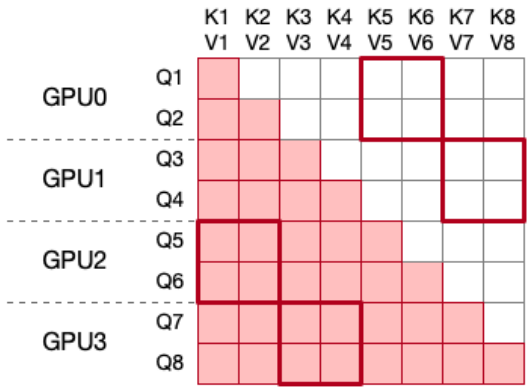


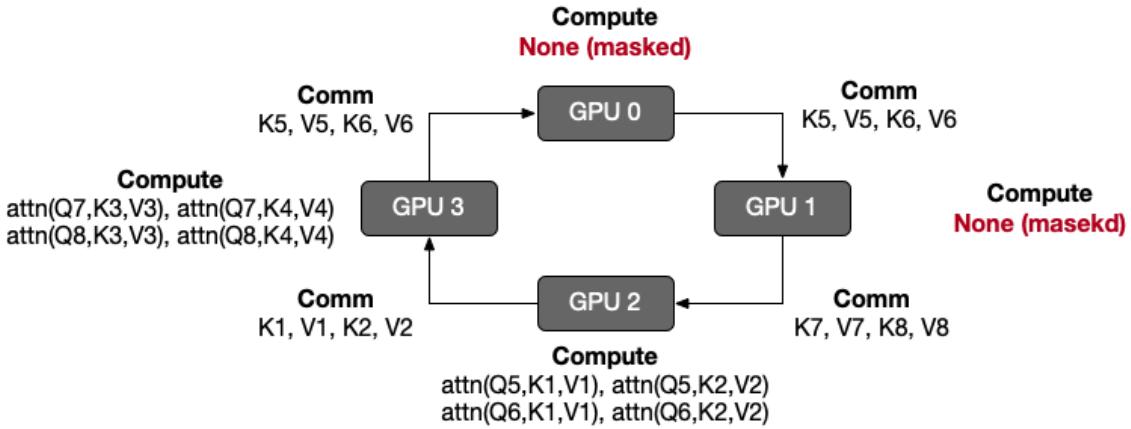


Ring Attention

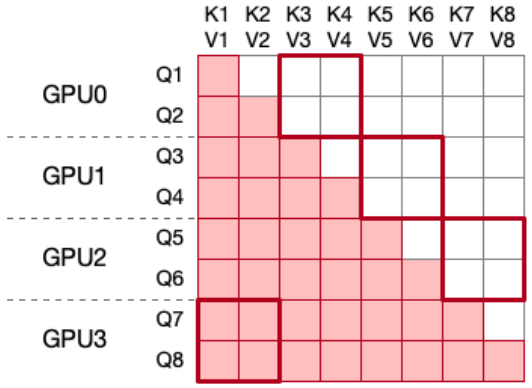
RELAXED SYSTEM LAB

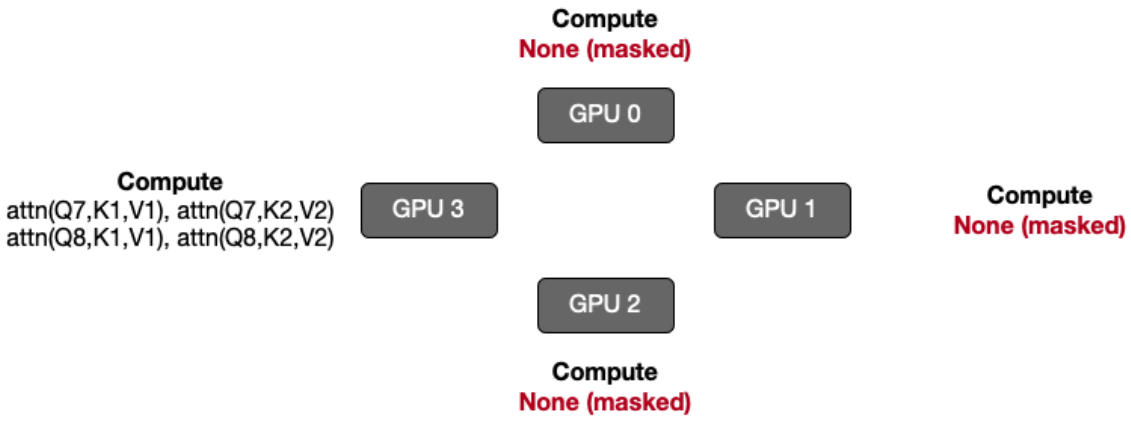
Round 3





Round 4

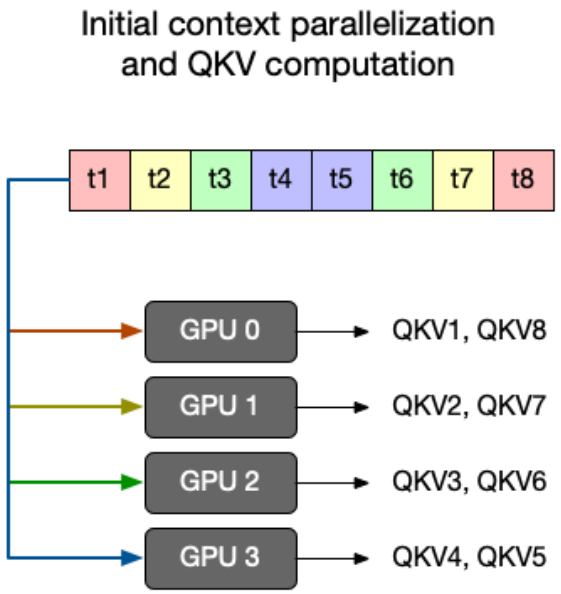


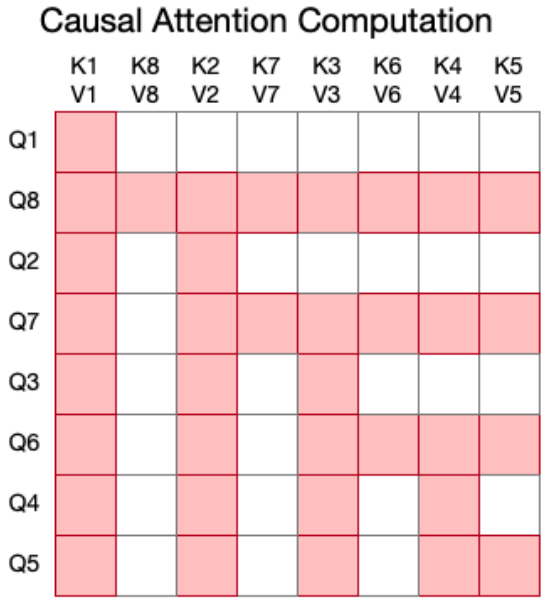


Megatron Context Parallelism

RELAXED SYSTEM LAB

- A trivial issue to be fixed in Ring Attention: the load-balance in each iteration:
- Megatron context parallelism implementation: an optimization not to compute unnecessary masked parts to balance the workflow.

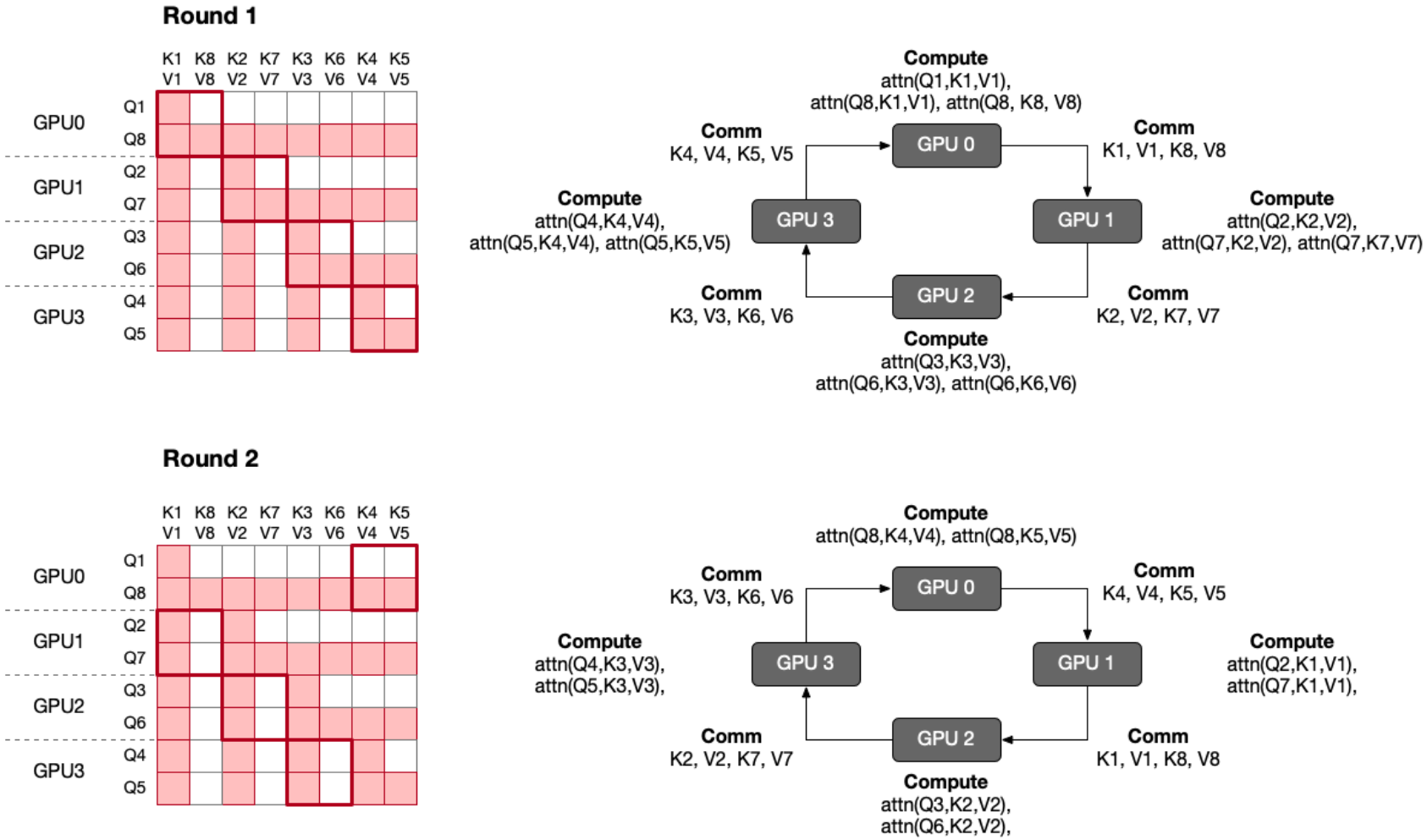




https://insujang.github.io/2024-09-20/introducing-context-parallelism

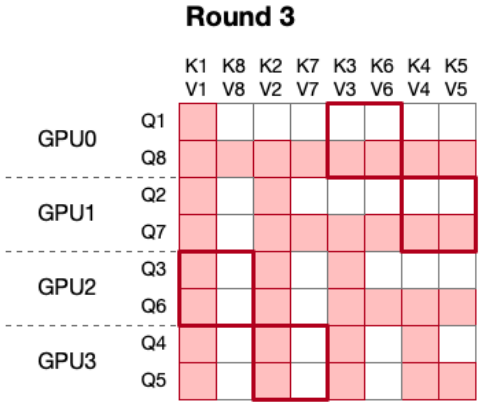
Megatron Context Parallelism

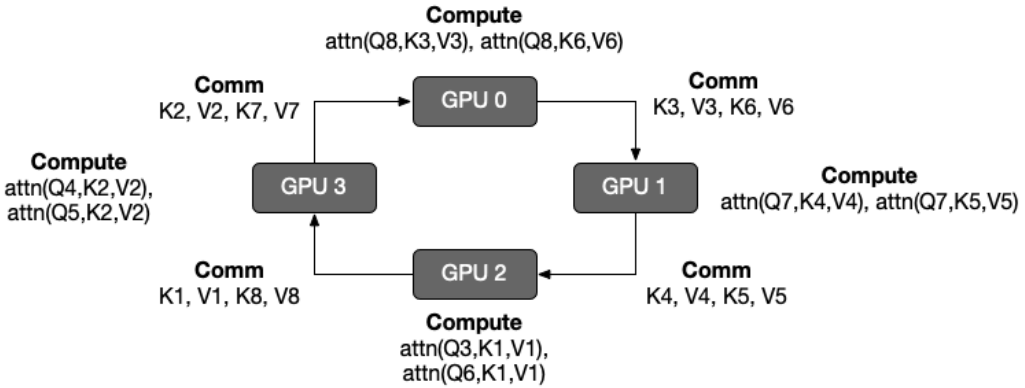
RELAXED SYSTEM LAB

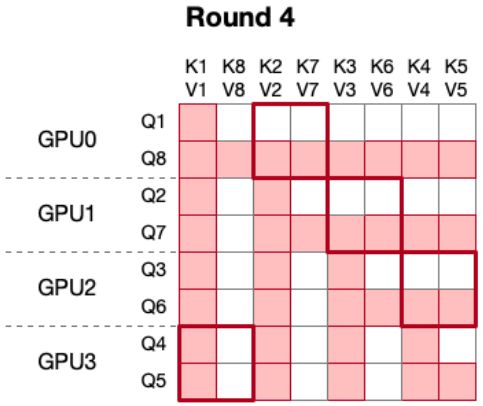


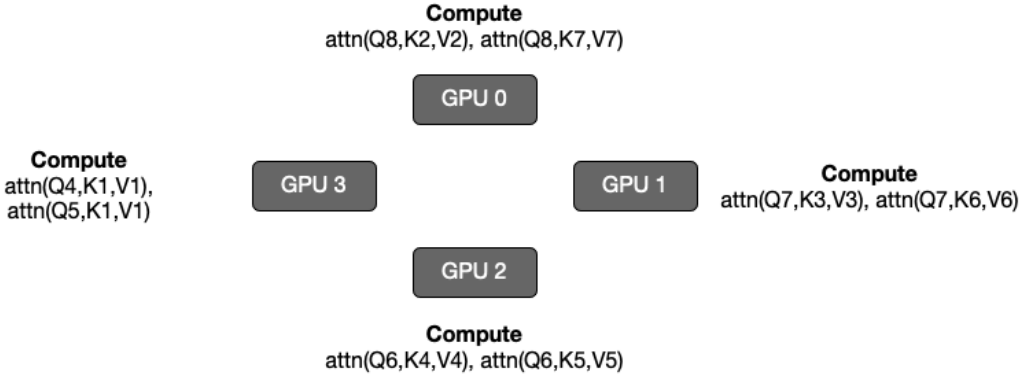
Megatron Context Parallelism

RELAXED SYSTEM LAB









References

- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mixture-of-experts
- https://huggingface.co/blog/moe
- https://arxiv.org/abs/2401.06066
- https://arxiv.org/pdf/2205.05198
- https://arxiv.org/pdf/2309.14509
- https://insujang.github.io/2024-09-20/introducing-context-parallelism/