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

Tensor Model- & Optimizer-Parallel Training

COMP4901Y

Binhang Yuan

Recall Data & Pipeline Parallelism

RELAXED SYSTEM LAB

- Data Parallelism:
  - **Memory issue**: each device needs to maintain a complete copy of the model (parameters, gradients, and optimizer status).
  - **Statistical efficiency**: if the global batch size is too large, it may affect the convergence rate
- Pipeline Parallelism:
  - **Bubble overhead:** the pipeline parallelism efficiency decreases as the number of stages increases.

Tensor Model Parallelism

TransformerBlocks$(x \in R^{B \times L \times D}) \to x' \in \mathbb{R}^{B \times L \times D}$

RELAXED SYSTEM LAB

- B is the batch size;
- L is the sequence length;
- D is the model dimension;
- Multi-head attention:
  $D = n_H \times H$
- H is the head dimension;
- n_h is the number of heads.

Computation	Input	Output
$Q = XW^Q$	$X \in \mathbb{R}^{B \times L \times D}$, $W^Q \in \mathbb{R}^{D \times D}$	$Q \in \mathbb{R}^{B \times L \times D}$
$K = XW^K$	$X \in \mathbb{R}^{B \times L \times D}$, $W^K \in \mathbb{R}^{D \times D}$	$K \in \mathbb{R}^{B \times L \times D}$
$V = XW^V$	$X \in \mathbb{R}^{B \times L \times D}$, $W^V \in \mathbb{R}^{D \times D}$	$V \in \mathbb{R}^{B \times L \times D}$
$[Q_1, Q_2 \dots, Q_{n_h}] = \text{Partion}_{-1}(Q)$	$Q \in \mathbb{R}^{B \times L \times D}$	$Q_i \in \mathbb{R}^{B \times L \times H}, i = 1, \dots n_h$
$[K_1, K_2 \dots, K_{n_h}] = \text{Partion}_{-1}(K)$	$K \in \mathbb{R}^{B \times L \times D}$	$K_i \in \mathbb{R}^{B \times L \times H}, i = 1, \dots n_h$
$[V_1, V_2 \dots, V_{n_h}] = \text{Partion}_{-1}(V)$	$V \in \mathbb{R}^{B \times L \times D}$	$V_i \in \mathbb{R}^{B \times L \times H}, i = 1, \dots n_h$
$\text{Score}_\text{i} = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{D}}), i = 1, \dots n_h$	$Q_i, K_i \in \mathbb{R}^{B \times L \times H}$	$\text{score}_\text{i} \in \mathbb{R}^{B \times L \times L}$
$Z_i = \text{score}_i V_i, i = 1, \dots n_h$	$\text{score}_\text{i} \in \mathbb{R}^{B \times L \times L}, V_i \in \mathbb{R}^{B \times L \times H}$	$Z_i \in \mathbb{R}^{B \times L \times H}$
$Z = \text{Merge}_{-1}([Z_1, Z_2 \dots, Z_{n_h}])$	$Z_i \in \mathbb{R}^{B \times L \times H}, i = 1, \dots n_h$	$Z \in \mathbb{R}^{B \times L \times D}$
$\text{Out} = ZW^O$	$Z \in \mathbb{R}^{B \times L \times D}$, $W^O \in \mathbb{R}^{D \times D}$	$\text{Out} \in \mathbb{R}^{B \times L \times D}$
$A = \text{Out}\, W^1$	$\text{Out} \in \mathbb{R}^{B \times L \times D}$, $W^1 \in \mathbb{R}^{D \times 4D}$	$A \in \mathbb{R}^{B \times L \times 4D}$
$A' = \text{relu}(A)$	$A \in \mathbb{R}^{B \times L \times 4D}$	$A' \in \mathbb{R}^{B \times L \times 4D}$
$X' = A'W^2$	$A' \in \mathbb{R}^{B \times L \times 4D}$, $W^2 \in \mathbb{R}^{4D \times D}$	$X' \in \mathbb{R}^{B \times L \times D}$

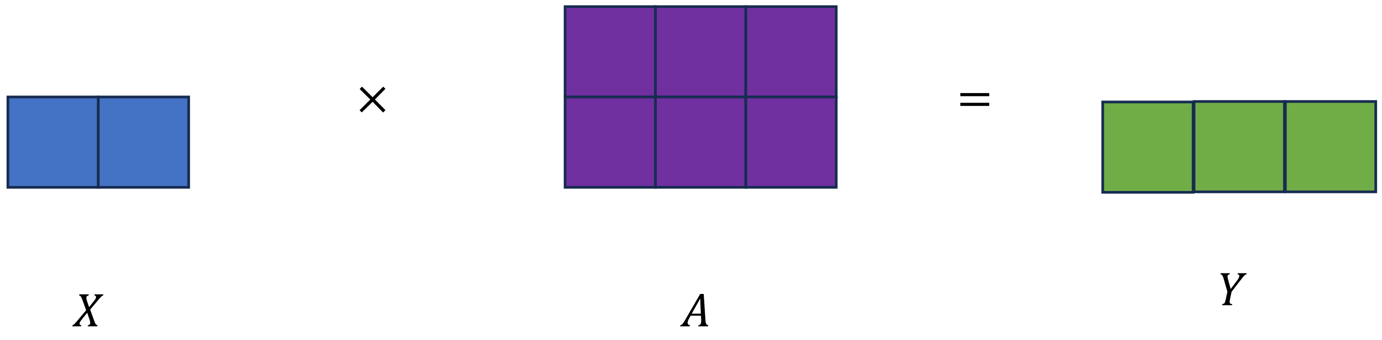
Tensor Model Parallelism

RELAXED SYSTEM LAB

- High-level idea:
  - The tensor is split up into multiple chunks;
  - Instead of having the whole tensor reside on a single GPU, each shard of the tensor resides on its designated GPU.
  - Each shard is processed separately and in parallel on different GPUs.
  - The results are synchronized at the end of the step.

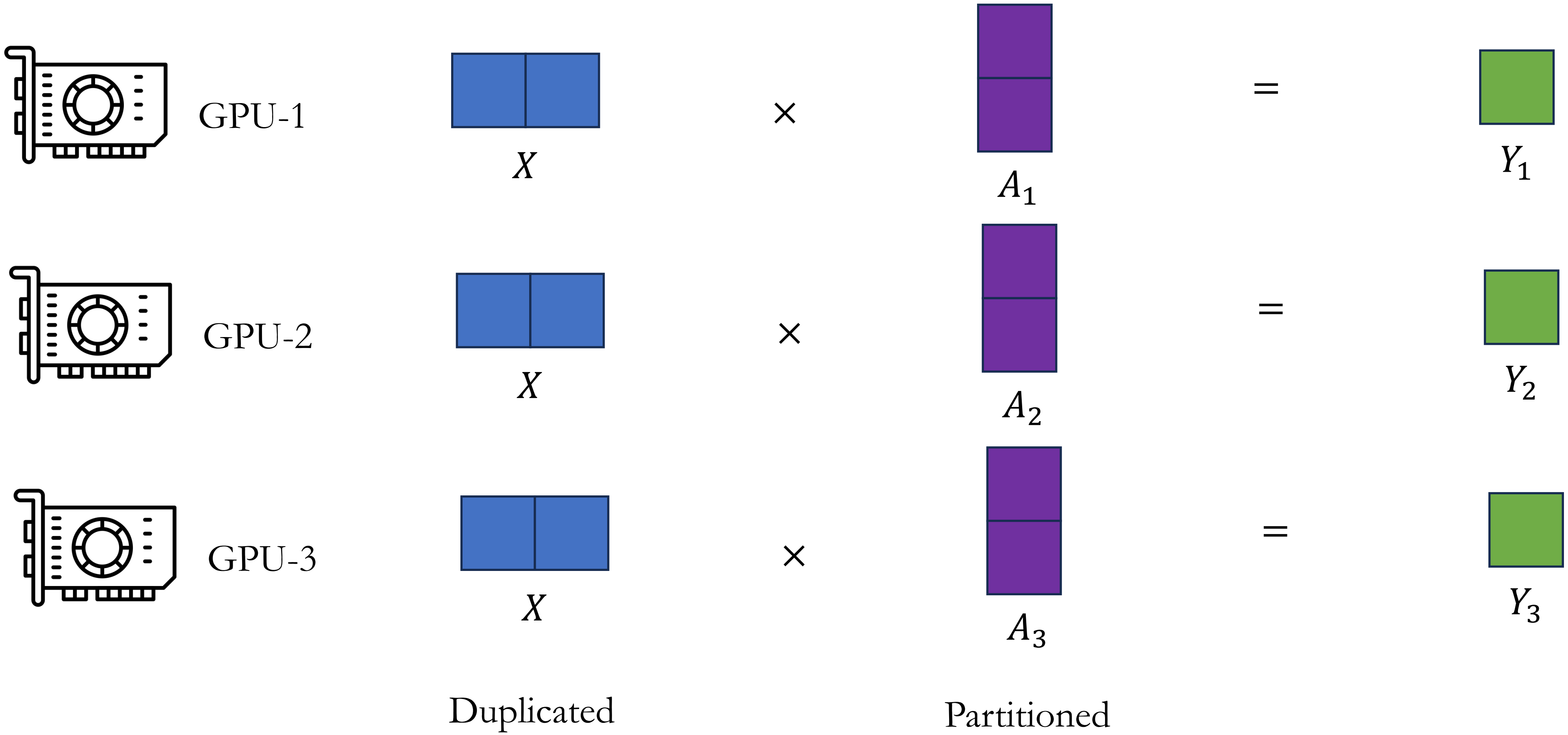
Partition the Matrix Multiplication

- Distribute the matrix multiplication:



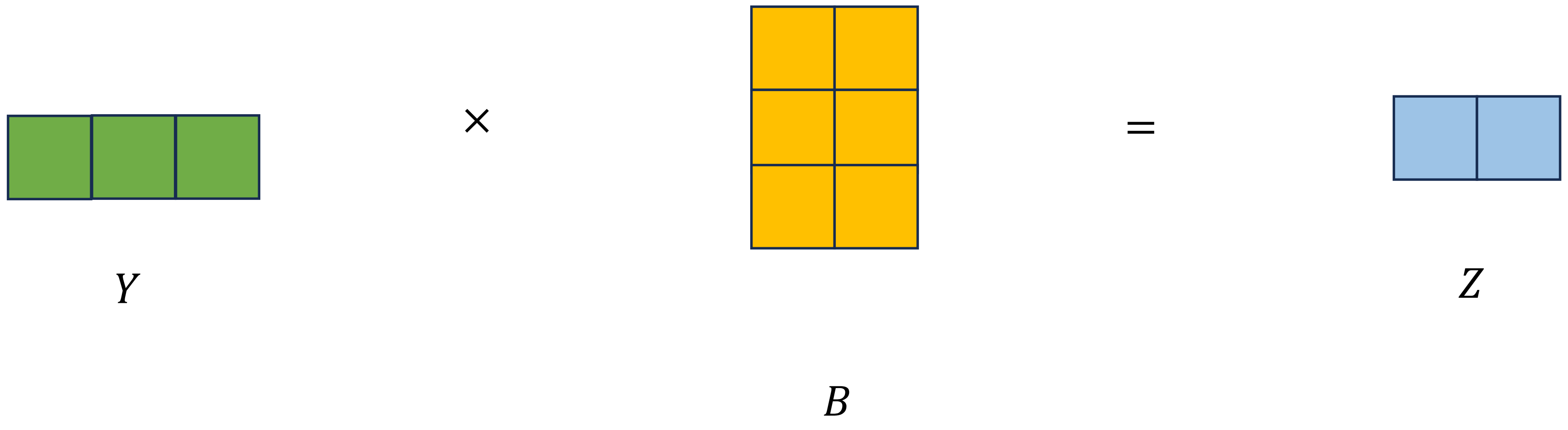
Partition the Matrix Multiplication

- Distribute the matrix multiplication:



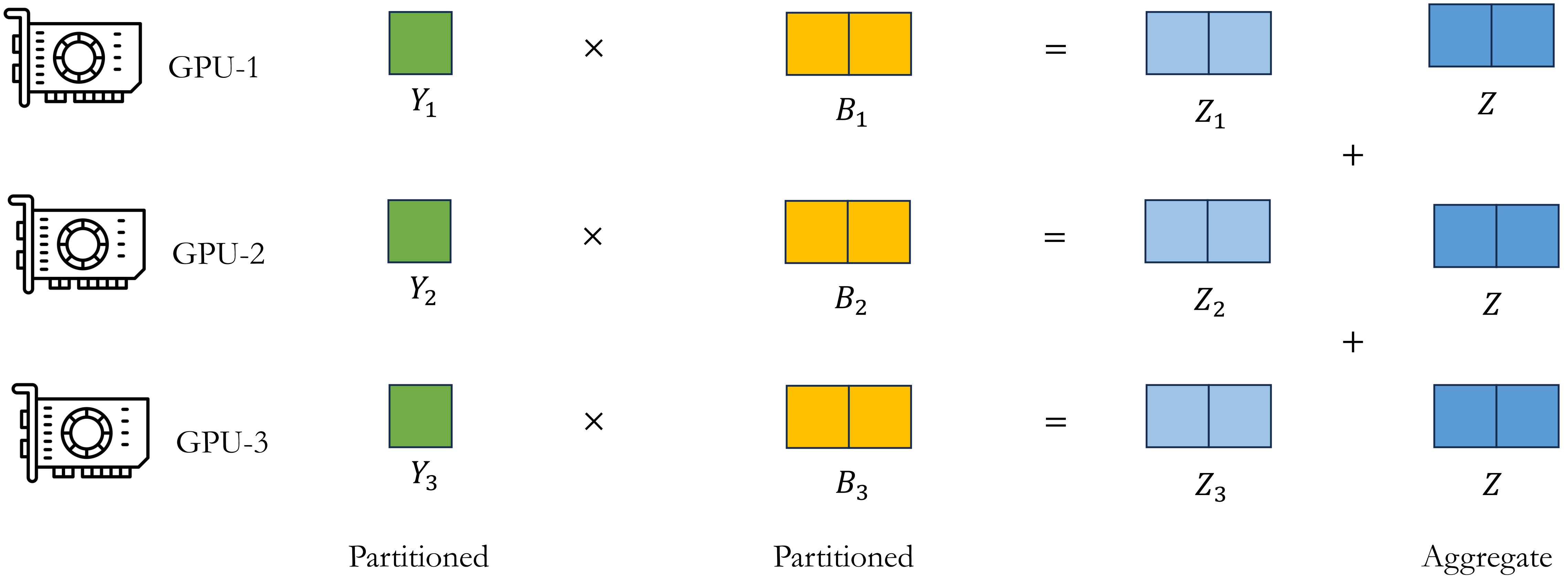
Partition the Matrix Multiplication

- Distribute the matrix multiplication:



Partition the Matrix Multiplication

- Distribute the matrix multiplication:

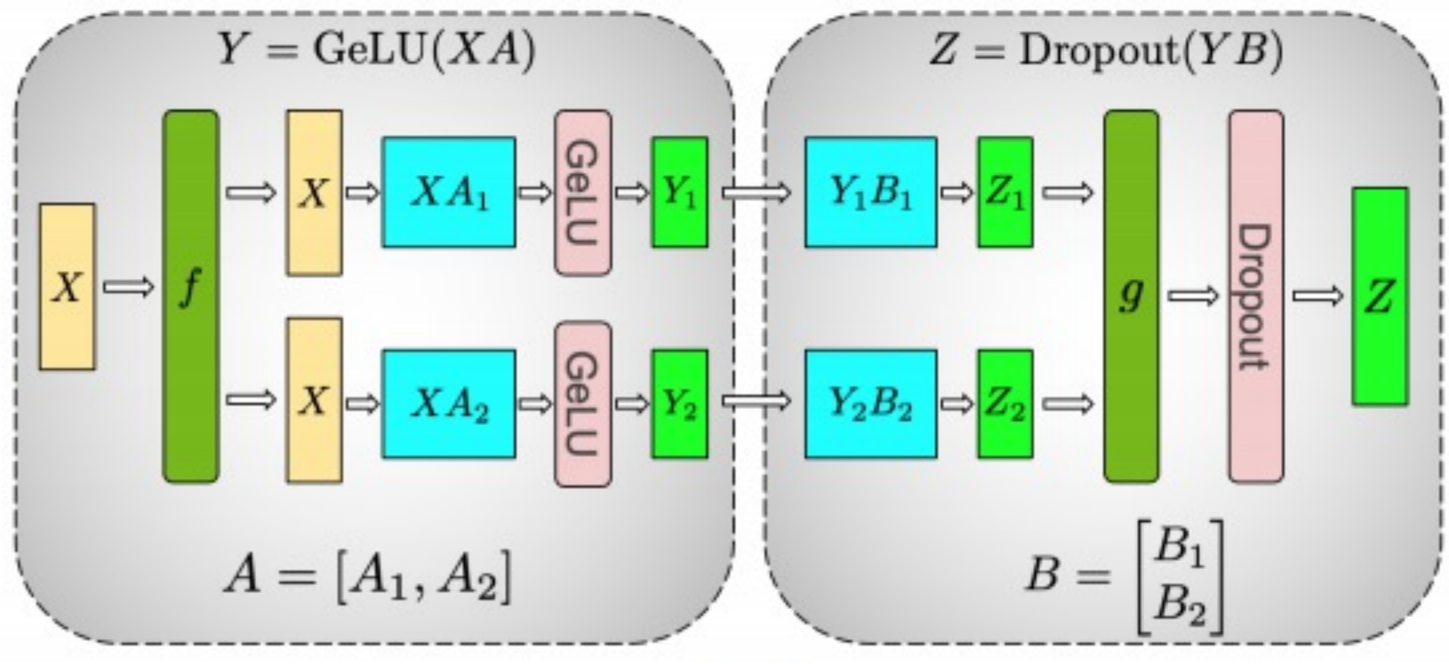


Tensor Model Parallelism

RELAXED SYSTEM LAB

- Split the first weight matrix col-wisely;
- Split the second weight matrix row-wisely;
- Duplicate the input on each GPU;
- Apply the computation as we illustrated above;
- Aggregate the outputs after the local computation.

MLP in Tensor Model Parallelism

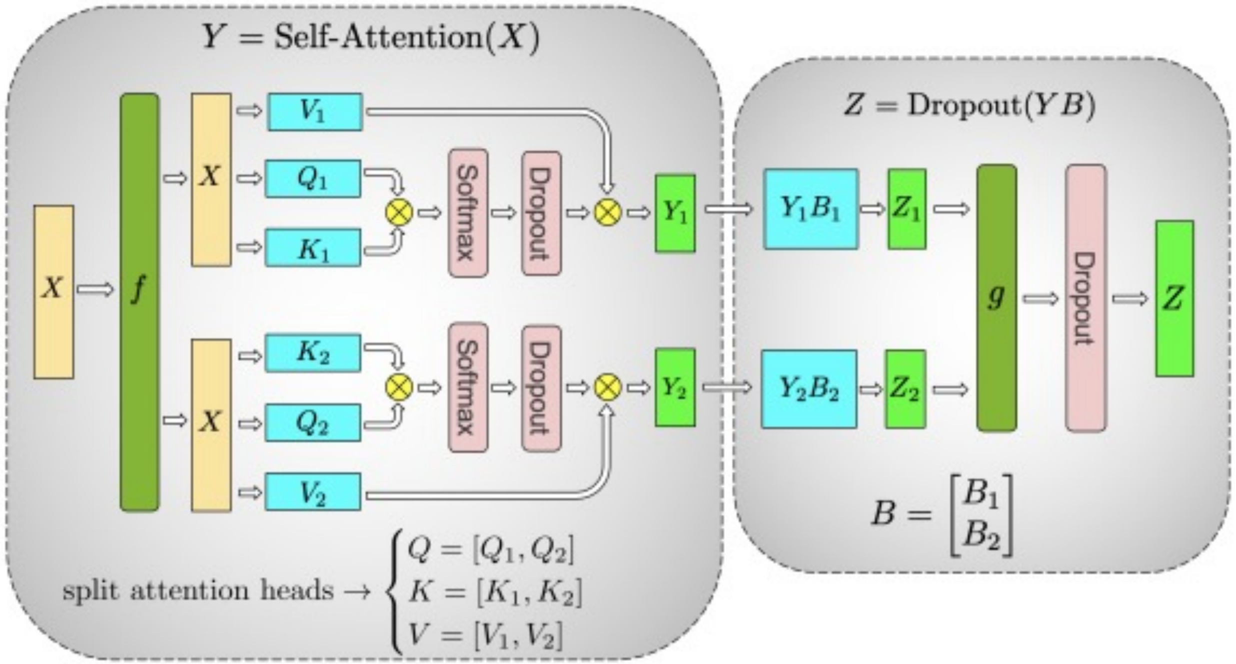


(a) MLP

- f is the identity operator in the forward pass and the **AllReduce** operator in the backward pass.
- g is the **AllReduce** operator in the forward pass and the identity operator in the backward pass.

Multi-Head Attention in Tensor Model Parallelism

RELAXED SYSTEM LAB



(b) Self-Attention

- f is the identity operator in the forward pass and the AllReduce operator in the backward pass.
- g is the AllReduce operator in the forward pass and the identity operator in the backward pass.

TransformerBlocks in Tensor Model Parallelism

- B is the batch size;
- L is the sequence length;
- D is the model dimension;
- Multi-head attention:
  $D = n_H \times H$
- H is the head dimension;
- n_H is the number of heads.
- d_{tp} is the tensor parallel degree: $d_{tp} \leq n_H$.

Computation	Input	Output
$Q = XW^Q$	$X \in \mathbb{R}^{B\times L\times D}$, $W^Q \in \mathbb{R}^{D\times\frac{D}{d_{tp}}}$	$Q \in \mathbb{R}^{B\times L\times\frac{D}{d_{tp}}}$
$K = XW^K$	$X \in \mathbb{R}^{B\times L\times D}$, $W^K \in \mathbb{R}^{D\times\frac{D}{d_{tp}}}$	$K \in \mathbb{R}^{B\times L\times\frac{D}{d_{tp}}}$
$V = XW^V$	$X \in \mathbb{R}^{B\times L\times D}$, $W^V \in \mathbb{R}^{D\times\frac{D}{d_{tp}}}$	$V \in \mathbb{R}^{B\times L\times\frac{D}{d_{tp}}}$
$\left[Q_1, Q_2 \dots, Q_{\frac{n_H}{d_{tp}}}\right] = \text{Partion}_{-1}(Q)$	$Q \in \mathbb{R}^{B\times L\times\frac{D}{d_{tp}}}$	$Q_i \in \mathbb{R}^{B\times L\times H}, i = 1, \dots \frac{n_H}{d_{tp}}$
$\left[K_1, K_2 \dots, K_{\frac{n_H}{d_{tp}}}\right] = \text{Partion}_{-1}(K)$	$K \in \mathbb{R}^{B\times L\times\frac{D}{d_{tp}}}$	$K_i \in \mathbb{R}^{B\times L\times H}, i = 1, \dots \frac{n_H}{d_{tp}}$
$\left[V_1, V_2 \dots, V_{\frac{n_H}{d_{tp}}}\right] = \text{Partion}_{-1}(V)$	$V \in \mathbb{R}^{B\times L\times\frac{D}{d_{tp}}}$	$V_i \in \mathbb{R}^{B\times L\times H}, i = 1, \dots \frac{n_H}{d_{tp}}$
$\text{Score}_i = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{D}}), i = 1, \dots \frac{n_H}{d_{tp}}$	$Q_i, K_i \in \mathbb{R}^{B\times L\times H}$	$\text{score}_i \in \mathbb{R}^{B\times L\times L}$
$Z_i = \text{score}_i V_i, i = 1, \dots \frac{n_H}{d_{tp}}$	$\text{score}_i \in \mathbb{R}^{B\times L\times L}$, $V_i \in \mathbb{R}^{B\times L\times H}$	$Z_i \in \mathbb{R}^{B\times L\times H}$
$Z = \text{Merge}_{-1}\left(\left[Z_1, Z_2 \dots, Z_{\frac{n_H}{d_{tp}}}\right]\right)$	$Z_i \in \mathbb{R}^{B\times L\times H}, i = 1, \dots \frac{n_H}{d_{tp}}$	$Z \in \mathbb{R}^{B\times L\times\frac{D}{d_{tp}}}$
$\text{Out} = ZW^O$	$Z \in \mathbb{R}^{B\times L\times\frac{D}{d_{tp}}}$, $W^O \in \mathbb{R}^{\frac{D}{d_{tp}}\times D}$	$\text{Out} \in \mathbb{R}^{B\times L\times D}$
AllReduce(Out)	$\text{Out} \in \mathbb{R}^{B\times L\times D}$	$\text{Out} \in \mathbb{R}^{B\times L\times D}$

TransformerBlocks in Tensor Model Parallelism

- B is the batch size;
- L is the sequence length;
- D is the model dimension;
- Multi-head attention:
  $D = n_H \times H$
- H is the head dimension;
- n_H is the number of heads.
- d_{tp} is the tensor parallel degree: $d_{tp} \leq n_H$.

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

RELAXED SYSTEM LAB

RELAXED
SYSTEM LAB

Tensor Model Parallelism in Megatron-LM

https://github.com/NVIDIA/Megatron-LM

Entrance of the Training Scripts

```
model = GPTModel(
    config=config,
    transformer_layer_spec=transformer_layer_spec,
    vocab_size=args.padded_vocab_size,
    max_sequence_length=args.max_position_embeddings,
    pre_process=pre_process,
    post_process=post_process,
    fp16_lm_cross_entropy=args.fp16_lm_cross_entropy,
    parallel_output=True,
    share_embeddings_and_output_weights=not args.untie_embeddings_and_output_weights,
    position_embedding_type=args.position_embedding_type,
    rotary_percent=args.rotary_percent,
)
```

https://github.com/NVIDIA/Megatron-LM/blob/main/pretrain_gpt.py#L60

Launch Scripts

```
GPT_ARGS="
    --tensor-model-parallel-size 2 \
    --pipeline-model-parallel-size 2 \
    --sequence-parallel \
    --num-layers 24 \
    --hidden-size 1024 \
    --num-attention-heads 16 \
    --seq-length 1024 \
    --max-position-embeddings 1024 \
    --micro-batch-size 4 \
    --global-batch-size 16 \
    --lr 0.00015 \
    --train-iters 500000 \
    --lr-decay-iters 320000 \
    --lr-decay-style cosine \
    --min-lr 1.0e-5 \
    --weight-decay 1e-2 \
    --lr-warmup-fraction .01 \
    --clip-grad 1.0 \
    --fp16
"
```

https://github.com/NVIDIA/Megatron-LM/blob/main/examples/pretrain_gpt_distributed_with_mp.sh#L28

Parallel Strategies

- Data Parallelism:
  - **Memory issue**: each device needs to maintain a complete copy of the model (parameters, gradients, and optimizer status).
  - **Statistical efficiency**: if the global batch size is too large, it may affect the convergence rate
- Pipeline Parallelism:
  - **Bubble overhead:** the pipeline parallelism efficiency decreases as the number of stages increases.
- Tensor model parallelism:
  - **Limited** to transformer architectures.
  - **Communication intensive:** each TransformerBlock requests two **AllReduce**s in the forward pass and two **AllReduce**s in the backward pass.

Zero Redundancy Optimizer (ZeRO)

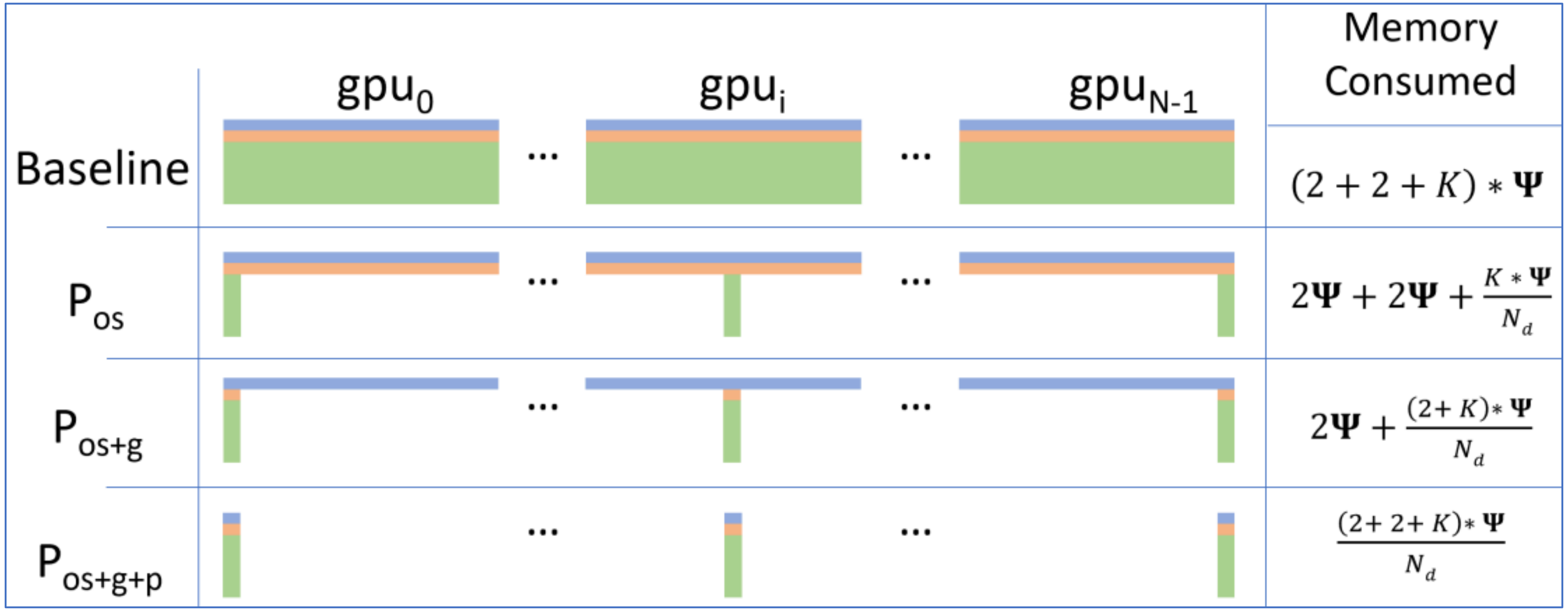
RELAXED SYSTEM LAB

Zero Redundancy Optimizer (ZeRO)

RELAXED SYSTEM LAB

- **<u>Core design idea</u>**:
  - Reduce the memory footprint per device for data-parallel training.
- Optimize the memory footprint:
  - Model parameters;
  - Gradients;
  - Optimizer status.

Zero Redundancy Optimizer (ZeRO)



- ψ is the total number of parameters;
- K denotes the memory multiplier of optimizer states;
- N_d denotes the parallel degree.

Zero Redundancy Optimizer (ZeRO)

RELAXED SYSTEM LAB

- **<u>ZeRO Stage-1</u> P_{os}:**
  - The optimizer states are partitioned across the processes, so that each process updates only its partition.
  - Same communication volume as data parallelism.

Zero Redundancy Optimizer (ZeRO)

RELAXED SYSTEM LAB

- **<u>ZeRO Stage-2</u> P_{os+g}:**
  - The reduced gradients for updating the model weights are also partitioned such that each process retains only the gradients corresponding to its portion of the optimizer states.
  - Same communication volume as data parallelism.

Zero Redundancy Optimizer (ZeRO)

- **ZeRO Stage-3** $\mathbf{P_{os+g+p}}$:
  - The model parameters are partitioned across the processes. ZeRO-3 will automatically collect and partition them during the forward and backward passes.
  - 50% increase in communication volume (when compared with data parallelism).

ZeRO Animation Illustration

RELAXED SYSTEM LAB

ZeRO 4-way data parallel training

Using:

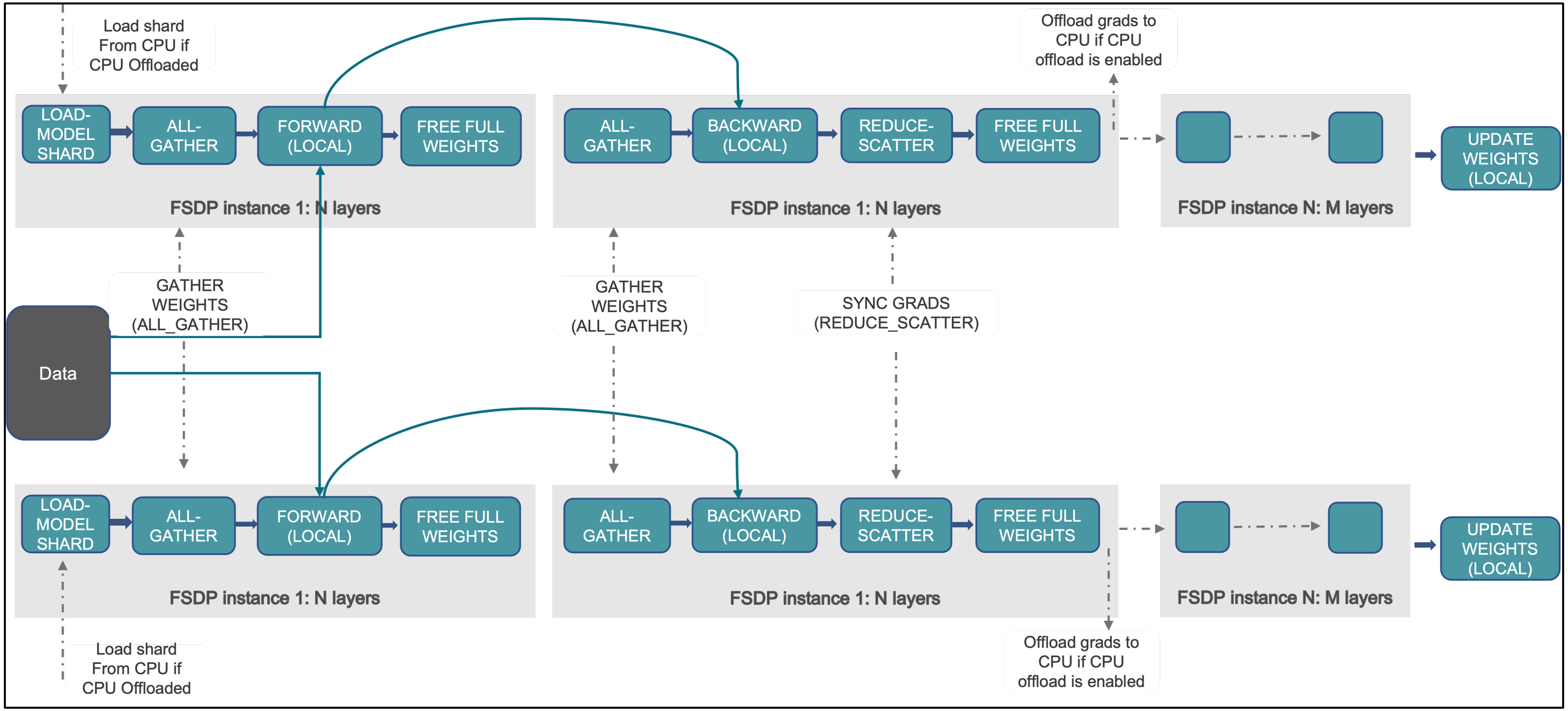
- P_{os} (Optimizer state)
- P_g (Gradient)
- P_p (Parameters)

https://www.microsoft.com/en-us/research/blog/ZeRO-deepspeed-new-system-optimizations-enable-training-models-with-over-100-billion-parameters/

FSDP in PyTorch

- FullyShardedDataParallel (FSDP) is the corresponding implementation of ZeRO-S3 in PyTroch:
  - FSDP is a type of data parallelism that shards model parameters, optimizer states and gradients across DDP ranks.
  - When training with FSDP, the GPU memory footprint is smaller than when training with DDP across all workers.
  - Come with the cost of increased communication volume.
  - The communication overhead is reduced by internal optimizations like overlapping communication and computation.

FSDP in PyTorch



FSDP in PyTorch

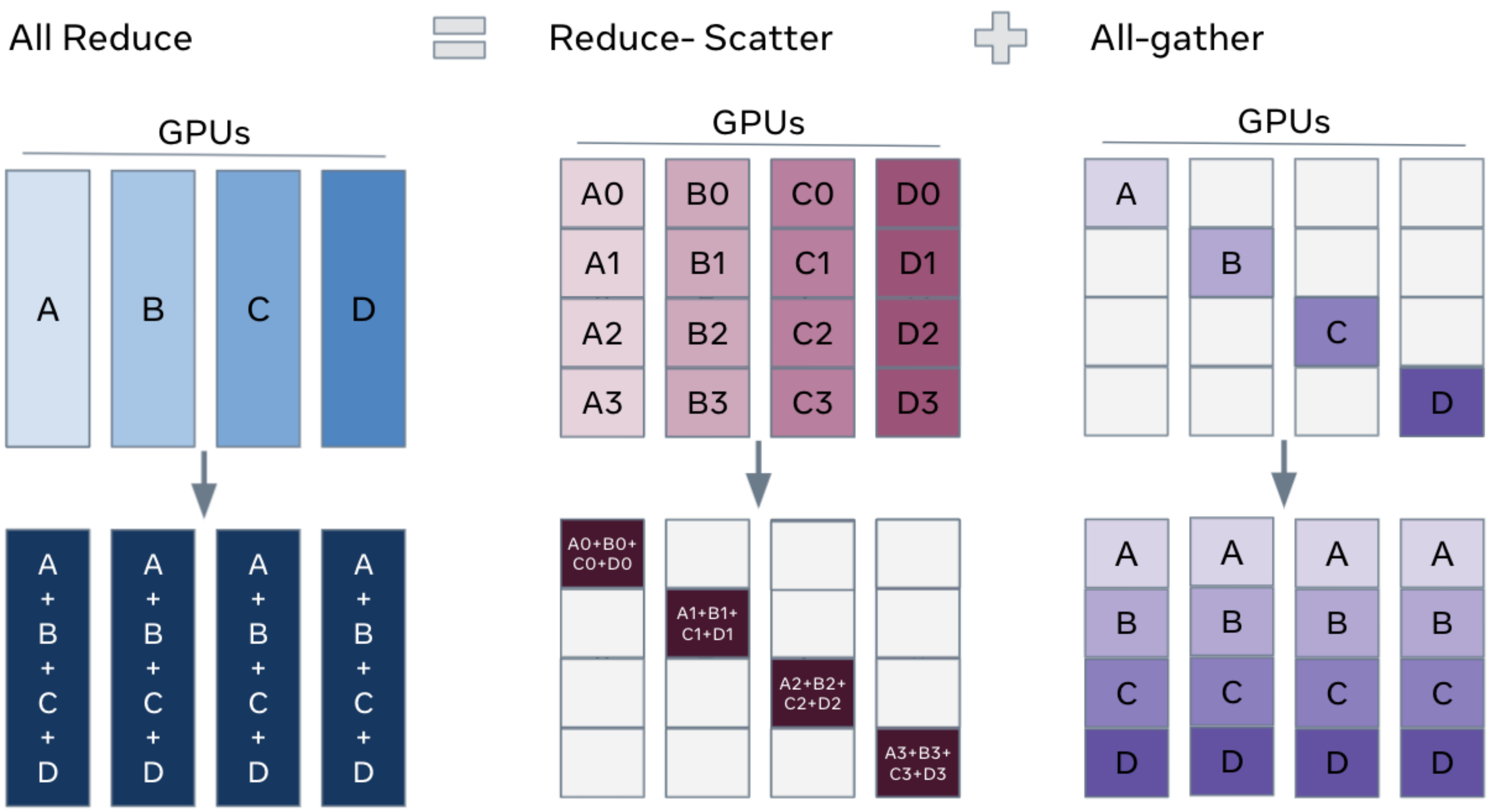
RELAXED SYSTEM LAB

- In construction:
  - Shard model parameters and each rank only keeps its own shard.
- In forward pass:
  - Run **AllGather** to collect all shards from all ranks to recover the full parameter in this FSDP unit;
  - Run forward computation;
  - Discard parameter shards it has just collected.
- In backward pass:
  - Run **AllGather** to collect all shards from all ranks to recover the full parameter in this FSDP unit;
  - Run backward computation.
  - Run **ReduceScatter** to sync gradients;
  - Discard parameters.

FSDP in PyTorch

- FSDP decomposes the DDP gradient **AllReduce** into **ReduceScatter** and **AllGather**.
- During the backward pass, FSDP reduces and scatters gradients, ensuring that each rank possesses a shard of the gradients.
- Then FSDP updates the corresponding shard of the parameters in the optimizer step.
- In the subsequent forward pass, FSDP performs an **AllGather** operation to collect and combine the updated parameter shards.

FSDP in PyTorch



PyTorch FSDP Practice

FSDP API

RELAXED SYSTEM LAB

FULLYSHARDEDDATAPARALLEL

CLASS torch.distributed.fsdp.FullyShardedDataParallel(*module, process_group=None, sharding_strategy=None, cpu_offload=None, auto_wrap_policy=None, backward_prefetch=BackwardPrefetch.BACKWARD_PRE, mixed_precision=None, ignored_modules=None, param_init_fn=None, device_id=None, sync_module_states=False, forward_prefetch=False, limit_all_gathers=True, use_orig_params=False, ignored_states=None, device_mesh=None*) [SOURCE]

A wrapper for sharding module parameters across data parallel workers.

This is inspired by Xu et al. as well as the ZeRO Stage 3 from DeepSpeed. FullyShardedDataParallel is commonly shortened to FSDP.

Example:

```
>>> import torch
>>> from torch.distributed.fsdp import FullyShardedDataParallel as FSDP
>>> torch.cuda.set_device(device_id)
>>> sharded_module = FSDP(my_module)
>>> optim = torch.optim.Adam(sharded_module.parameters(), lr=0.0001)
>>> x = sharded_module(x, y=3, z=torch.Tensor([1]))
>>> loss = x.sum()
>>> loss.backward()
>>> optim.step()
```

https://pytorch.org/docs/stable/fsdp.html

Using FSDP

Use FSDP API

```
from torch.distributed.fsdp import (
   FullyShardedDataParallel,
   CPUOffload,
)
from torch.distributed.fsdp.wrap import (
   default_auto_wrap_policy,
)
import torch.nn as nn

class model(nn.Module):
   def __init__(self):
       super().__init__()
       self.layer1 = nn.Linear(8, 4)
       self.layer2 = nn.Linear(4, 16)
       self.layer3 = nn.Linear(16, 4)

#model = DistributedDataParallel(model())

fsdp_model = FullyShardedDataParallel(
   model(),
   fsdp_auto_wrap_policy=default_auto_wrap_policy,
   cpu_offload=CPUOffload(offload_params=True),
)
```

References

- https://huggingface.co/transformers/v4.9.2/parallelism.html
- https://arxiv.org/abs/2104.04473
- https://github.com/NVIDIA/Megatron-LM/tree/main
- https://arxiv.org/pdf/1910.02054.pdf
- https://deepspeed.readthedocs.io/en/latest/zero3.html
- https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html