THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY | DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

RELAXED SYSTEM LAB

Optimizer Parallel Training

COMP4901Y

Binhang Yuan

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.

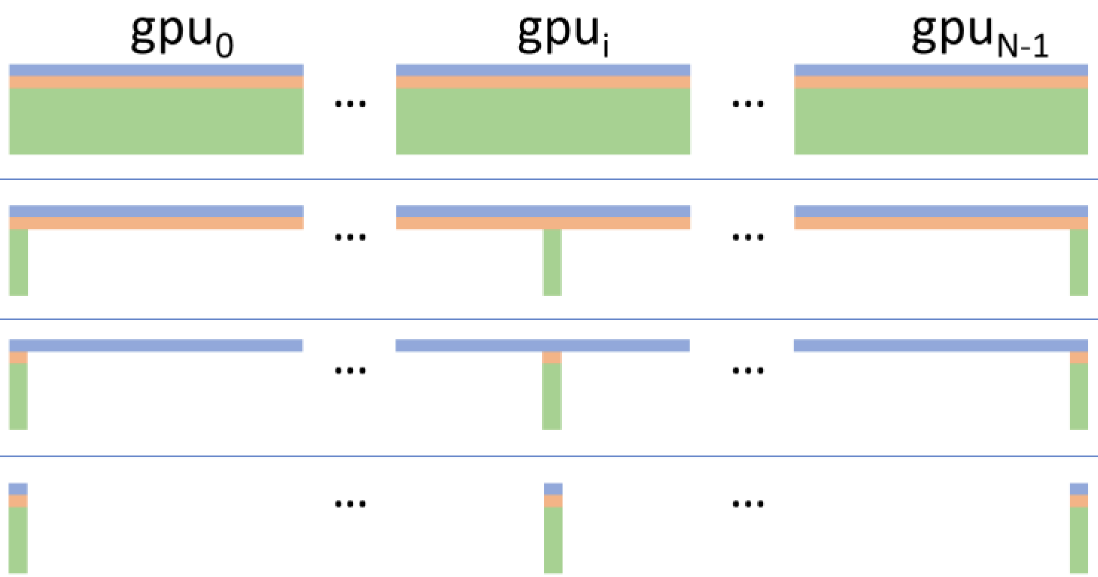
Zero Redundancy Optimizer (ZeRO)

RELAXED SYSTEM LAB

Zero Redundancy Optimizer (ZeRO)

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

Zero Redundancy Optimizer (ZeRO)



	Memory Consumed
Baseline	$(2 + 2 + K) * \Psi$
P_{os}	$2\Psi + 2\Psi + \frac{K * \Psi}{N_d}$
P_{os+g}	$2\Psi + \frac{(2+K) * \Psi}{N_d}$
P_{os+g+p}	$\frac{(2+2+K) * \Psi}{N_d}$

- 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

- **ZeRO Stage-1** $\mathbf{P_{os}}$:
  - 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

- **ZeRO Stage-2** $\mathbf{P_{os+g}}$:
  - 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)

RELAXED SYSTEM LAB

- **<u>ZeRO Stage-3</u> $\mathbf{P_{os+g+p}}$:**
  - 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

ZeRO 4-way data parallel training

Using:

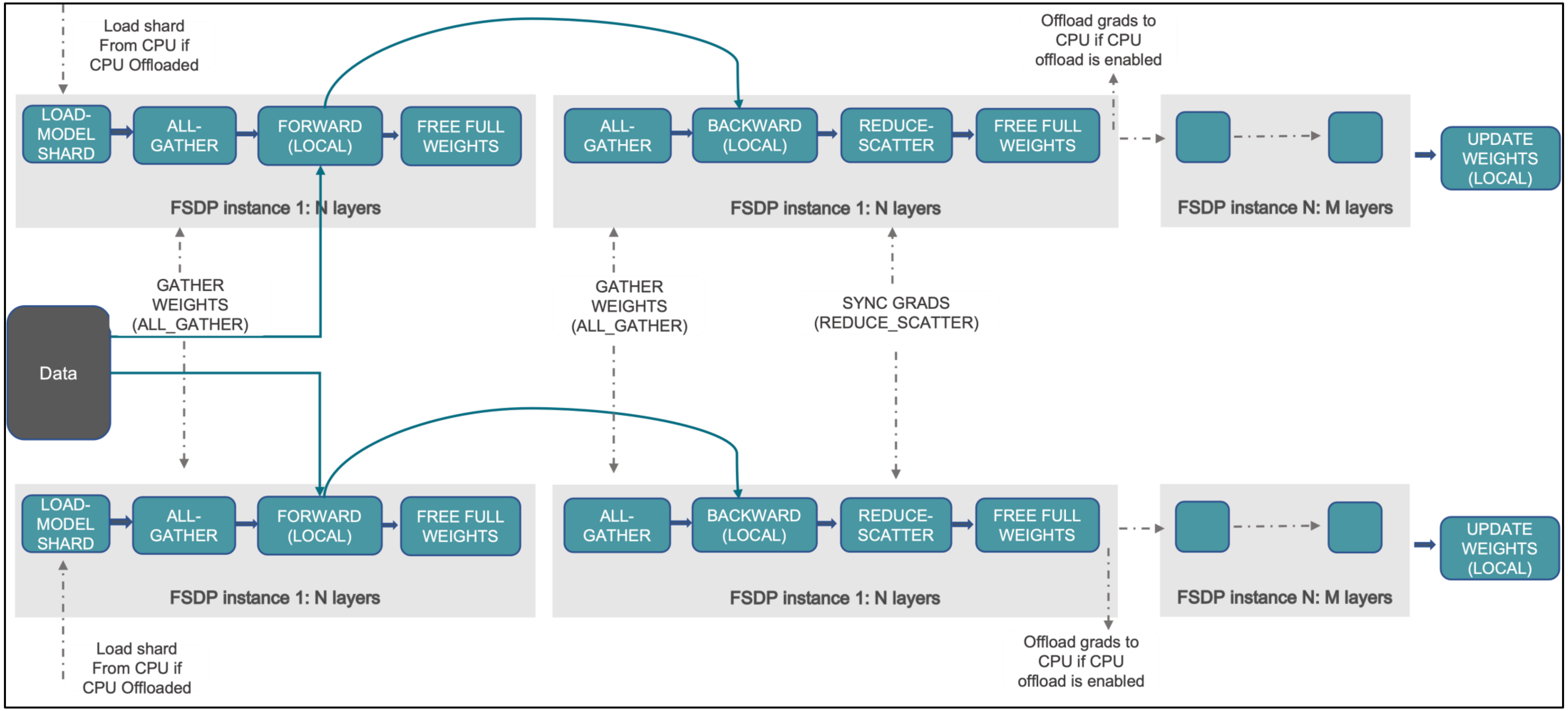
- P_{os} (Optimizer state)
- P_g (Gradient)
- P_p (Parameters)

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

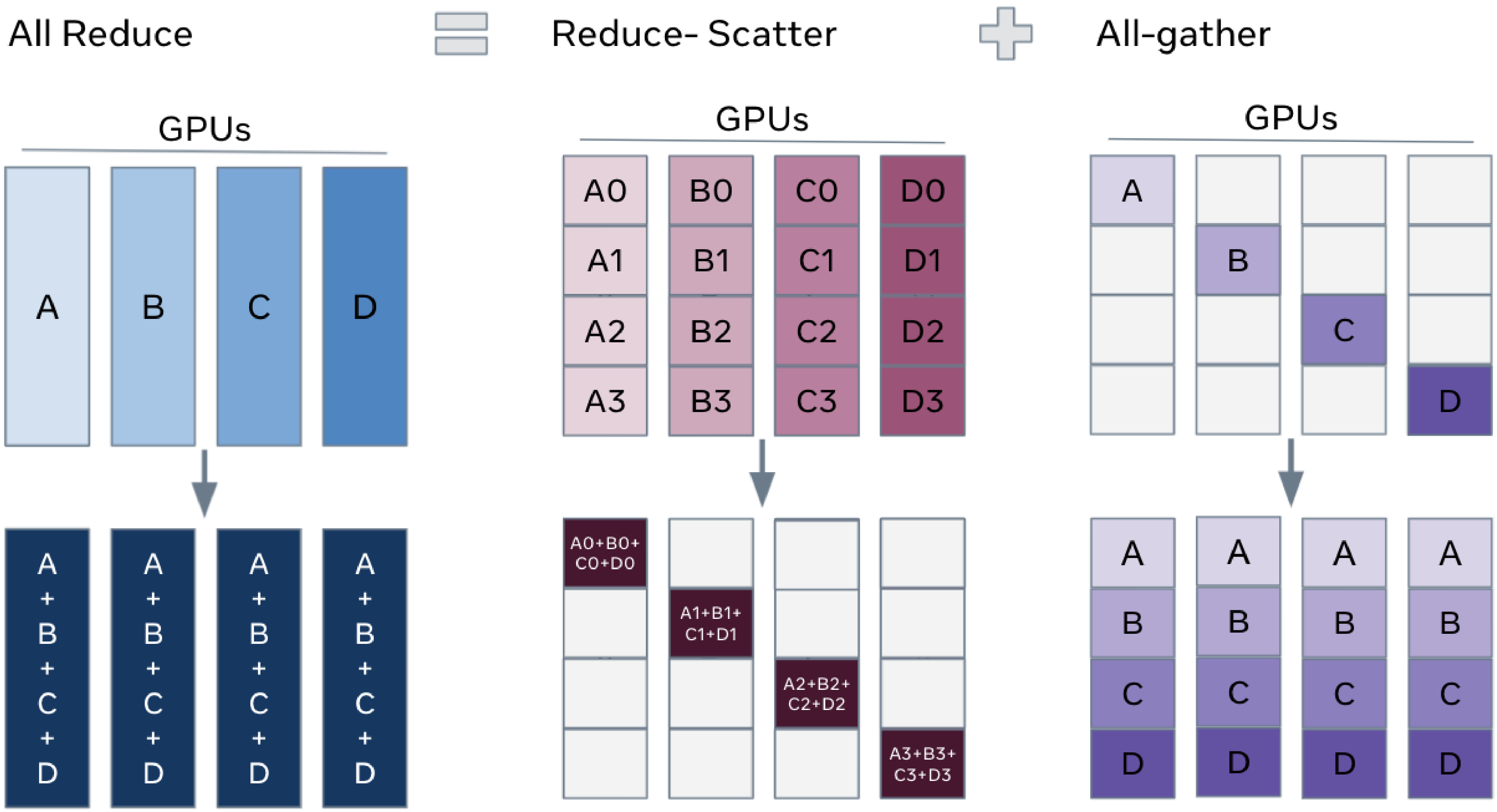
- In construction:
  - Shard model parameters and each rank only keeps its own shard.
- In forward pass:
  - Run **AllGather** to collect all shards from all ranks to recover the full parameter in this FSDP unit;
  - Run forward computation;
  - Discard parameter shards it has just collected.
- In backward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

RELAXED SYSTEM LAB

- 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

RELAXED SYSTEM LAB

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://arxiv.org/pdf/1910.02054.pdf
- https://deepspeed.readthedocs.io/en/latest/zero3.html
- https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html