Fully Sharded Data Parallel

Presented By
Ahmed Taha

Agenda

- Definitions, Terminology & Terminology
- How DDP works?
- How FSDP works?
- Reasons to use FSDP
- Reasons not to use FSDP

What is in the GPU memory?

- Parameters
- Gradients
- Optimizer State

>>>> Intermediate features/activation are omitted throughout this presentation.

What is in the GPU memory?

- Parameters
- Gradients
- Optimizer State

CLASS torch.optim.Adam(*params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False, *, foreach=None, maximize=False, capturable=False, differentiable=False, fused=None*) [SOURCE]

Implements Adam algorithm.

$$\begin{aligned}
&\textbf{input} : \gamma \text{ (lr)}, \beta_1, \beta_2 \text{ (betas)}, \theta_0 \text{ (params)}, f(\theta) \text{ (objective)} \\
&\qquad\qquad \lambda \text{ (weight decay)}, \; \textit{amsgrad}, \; \textit{maximize} \\
&\textbf{initialize} : m_0 \leftarrow 0 \text{ (first moment)}, v_0 \leftarrow 0 \text{ (second moment)}, \; \widehat{v_0}^{max} \leftarrow 0
\end{aligned}$$

[Adam — PyTorch 2.0 documentation](#)

Adam Recap

momentum $\longrightarrow m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t \longleftarrow$ Gradient

variance $\longrightarrow v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

weights $\longrightarrow \quad w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t$

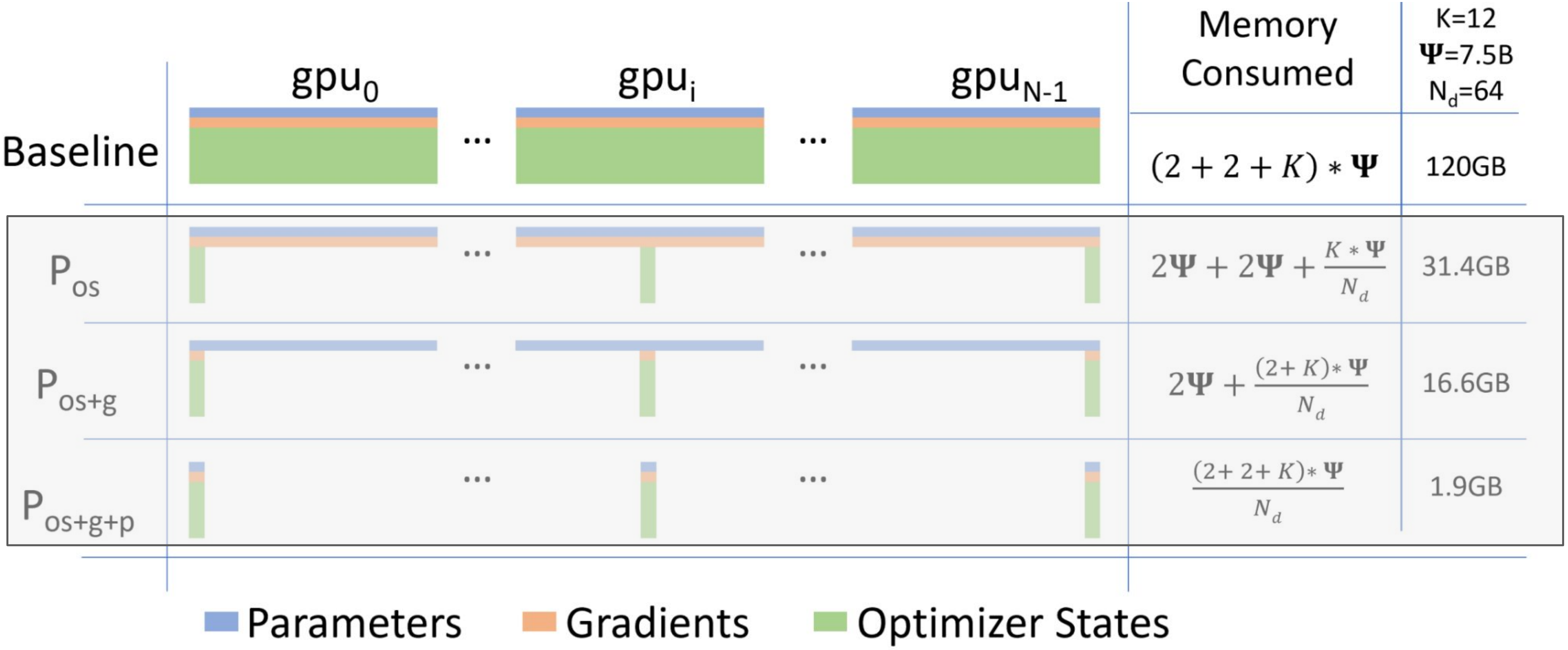
What is in the GPU memory?

Assuming FP16 and x parameters (*all* trainable)

- Parameters => $2x$ *(*2 bytes for float16*)*
- Gradients => $2x$
- Optimizer State [Adam] => $12x$
    - Parameter copy $4x$ (4 bytes for float32)
    - Momentum $4x$
    - Variance $4x$
    - All stored in float32!!

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models https://arxiv.org/abs/1910.02054

What is in the GPU memory?



ZeRO: Memory Optimizations Toward Training Trillion Parameter Models https://arxiv.org/abs/1910.02054

Agenda

- Definitions, Terminology & Terminology
- **How DDP works?**
- How FSDP works?
- Reasons to use FSDP
- Reasons not to use FSDP

How DDP Works?

We should learn about *NCCL* first!

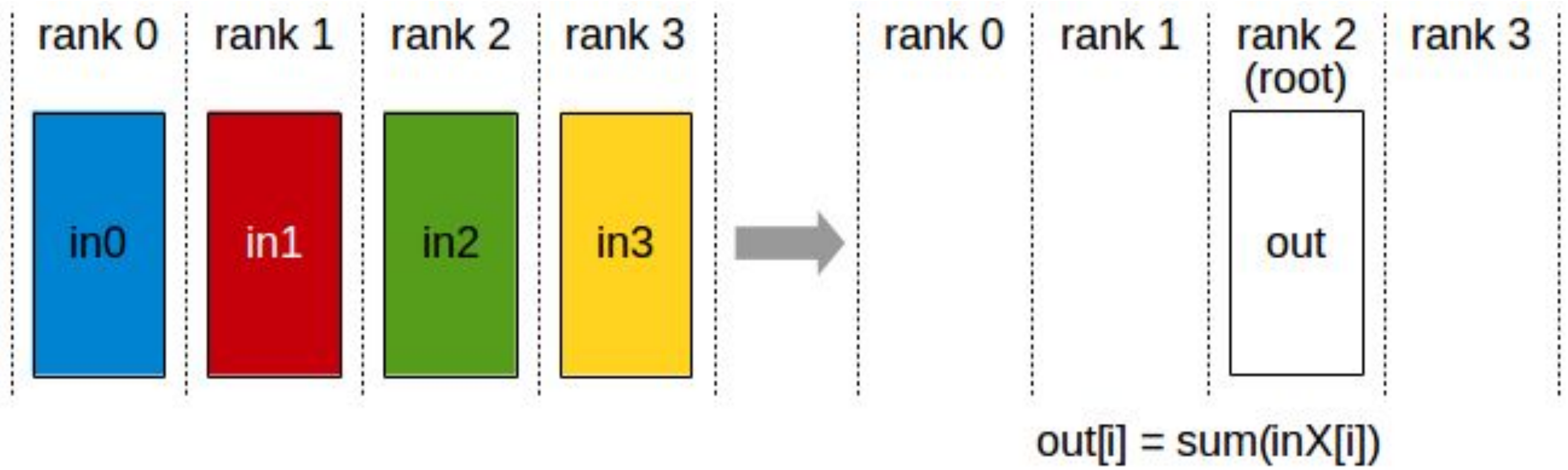
Don’t worry, it is simple :)

What is NCCL?

NVIDIA NCCL

The NVIDIA Collective Communication Library (NCCL) implements multi-GPU and multi-node communication primitives optimized for NVIDIA GPUs and Networking. NCCL provides routines such as all-gather, all-reduce, broadcast, reduce, reduce-scatter as well as point-to-point send and receive that are optimized to achieve high bandwidth and low latency over PCIe and NVLink high-speed interconnects within a node and over NVIDIA Mellanox Network across nodes.

NCCL - Reduce



https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html#reduce

NCCL - Reduce

ncclReduce

ncclResult_t **ncclReduce**(const void* *sendbuff*, void* *recvbuff*, size_t *count*, ncclDataType_t *datatype*, ncclRedOp_t *op*, int *root*, ncclComm_t *comm*, cudaStream_t *stream*)

ncclRedOp_t

`ncclRedOp_t`

Defines the reduction operation.

`ncclSum`

Perform a sum (+) operation

`ncclProd`

Perform a product (*) operation

`ncclMin`

Perform a min operation

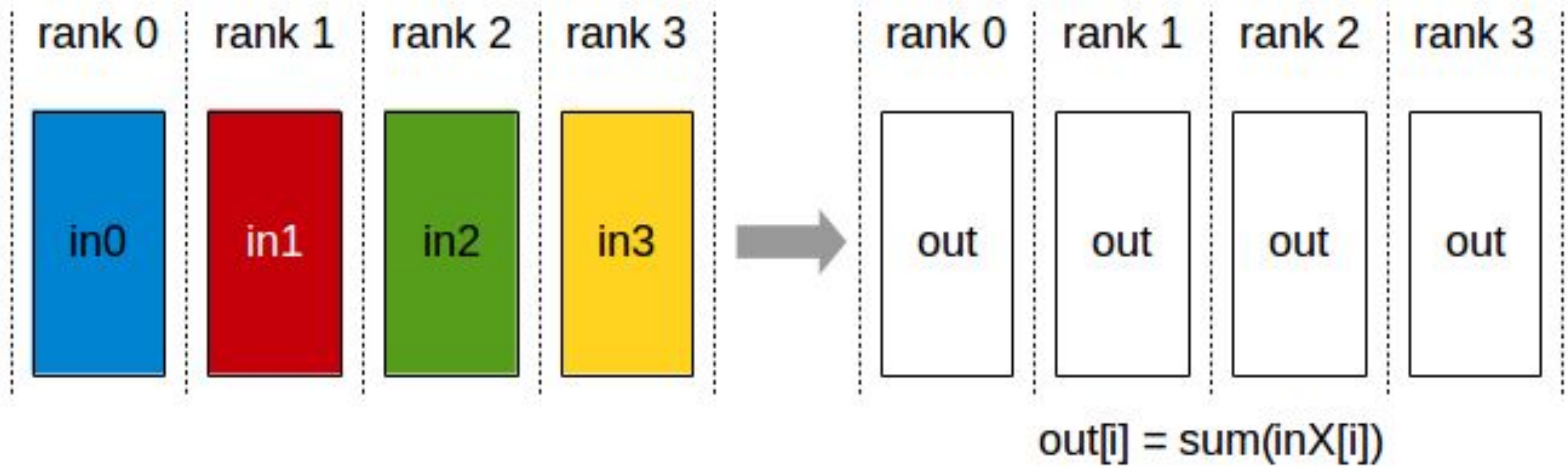
`ncclMax`

Perform a max operation

`ncclAvg`

Perform an average operation, i.e. a sum across all ranks, divided by the number of ranks.

NCCL - AllReduce

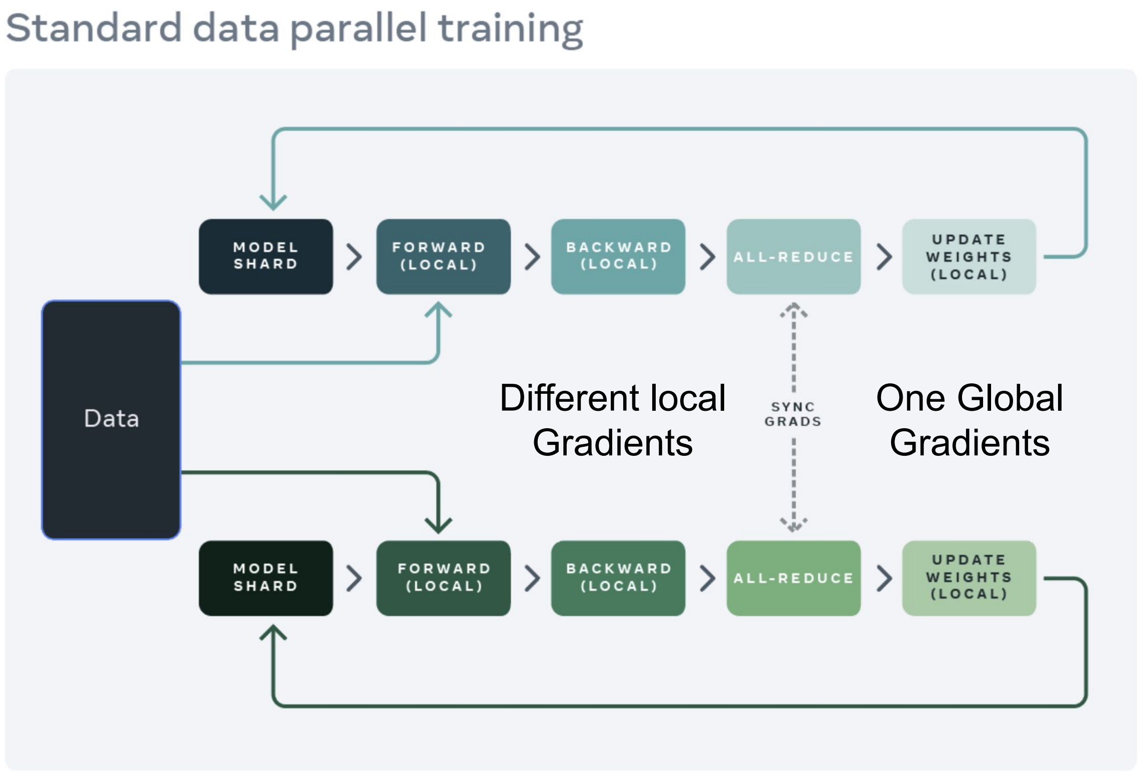


https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html#allreduce

How DDP Works?

- For every GPU (node)
  - FeedForward locally
  - Backward & compute gradient locally
  - AllReduce(gradient) – across nodes
  - Update optimizer states and weights locally

How DDP Works?



https://engineering.fb.com/2021/07/15/open-source/fsdp/

Agenda

- Definitions, Terminology & Terminology
- How DDP works?
- **How FSDP works?**
- Reasons to use FSDP
- Reasons not to use FSDP

What is *NOT* FSDP?

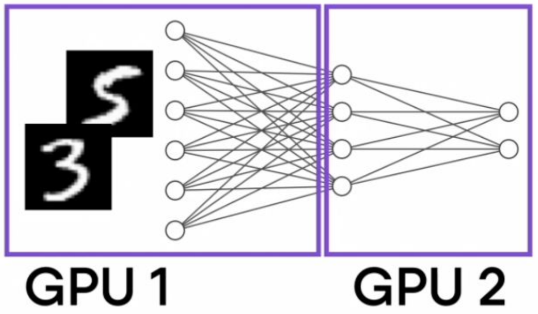
- Model Parallelism
- Tensor Parallelism
- Pipeline Parallelism

What is *NOT* FSDP?

- Model Parallelism

```python
class model_parallel(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers_1 = nn.Sequential(...)
        self.layers_2 = nn.Sequential(...)
        # ...
        self.layers_1.cuda(0)
        self.layers_2.cuda(1)
        # ...

    def forward(x):
        x = x.cuda(0)
        x = self.layers_1(x)
        x = x.cuda(1)
        x = self.layers_2(x)
        x = ...
    return x
```

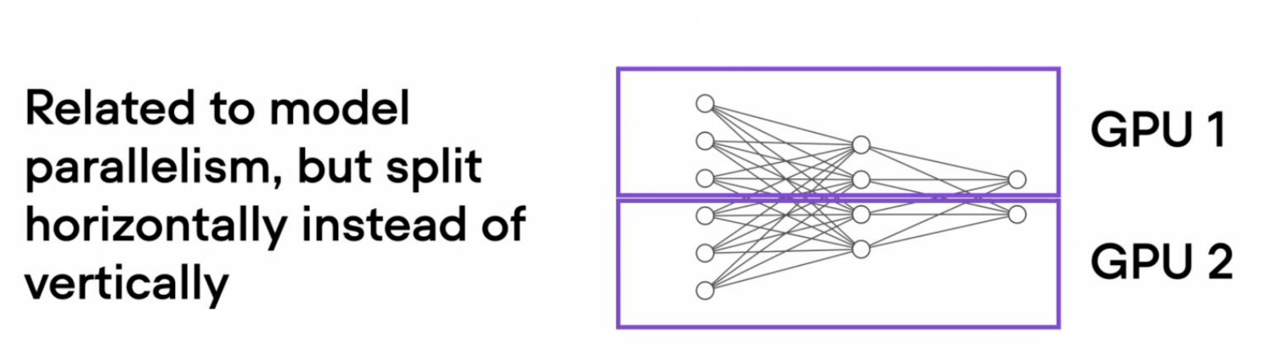


in PyTorch (not recommended)

https://lightning.ai/courses/deep-learning-fundamentals/9.0-overview-techniques-for-speeding-up-model-training/unit-9.2-multi-gpu-training-strategies/

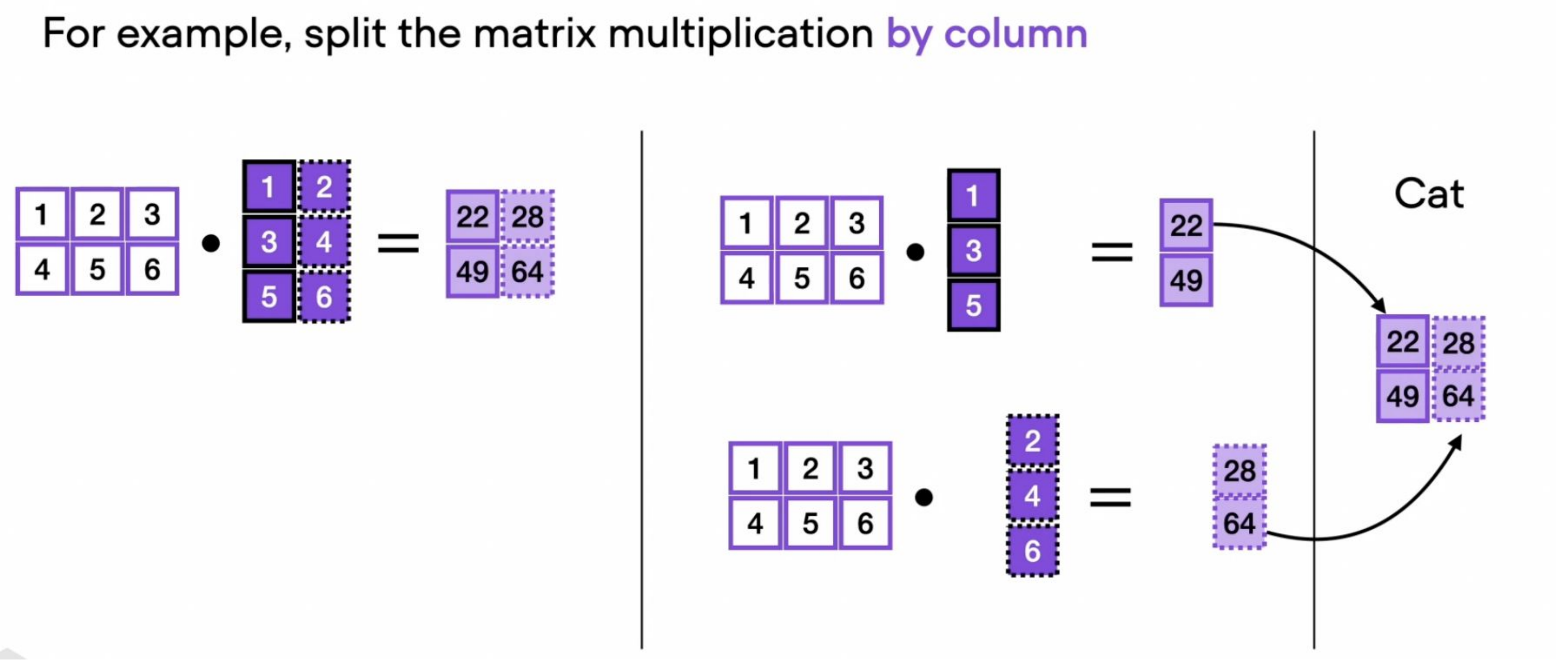
What is *NOT* FSDP?

- Tensor Parallelism



https://lightning.ai/courses/deep-learning-fundamentals/9.0-overview-techniques-for-speeding-up-model-training/unit-9.2-multi-gpu-training-strategies/

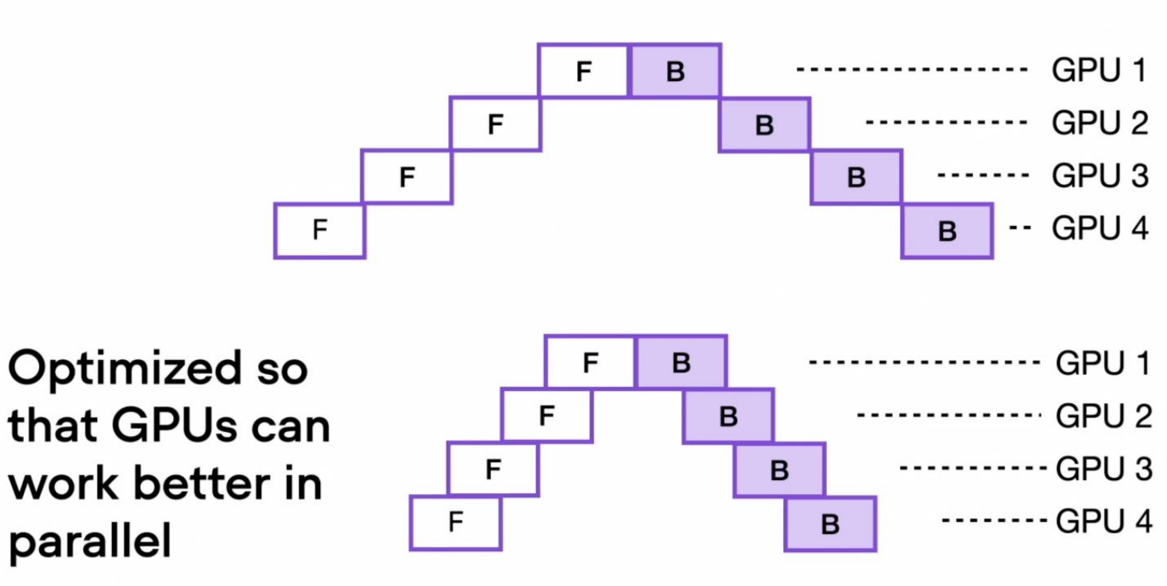
What is *NOT* FSDP?



https://lightning.ai/courses/deep-learning-fundamentals/9.0-overview-techniques-for-speeding-up-model-training/unit-9.2-multi-gpu-training-strategies/

What is *NOT* FSDP?

- Pipeline Parallelism [Mix Data and Model Parallelism]



https://lightning.ai/courses/deep-learning-fundamentals/9.0-overview-techniques-for-speeding-up-model-training/unit-9.2-multi-gpu-training-strategies/

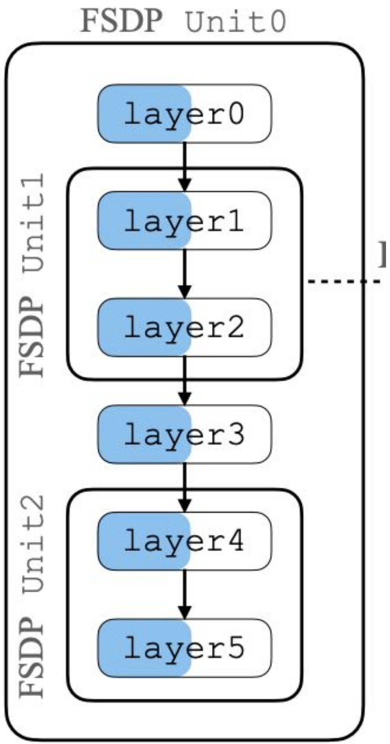
How FSDP Works?

More Terminology!!

1. FSDP Unit [Vertically “Splitting”]
2. Sharding [Horizontally “Splitting”]
3. All-Gather
4. Reduce-Scatter

FSDP Unit [Vertical “Splitting”]

- A unit to split the model. E.g.,
  - A layer
  - A stage
  - A group of layers (nn.Module)



Sharding [Horizontal "Splitting"]

- Sharding:
  - Store FSDP unit (e.g., a single layer/stage) on *FlatParameter*
  - Split *FlatParameter* on multiple nodes

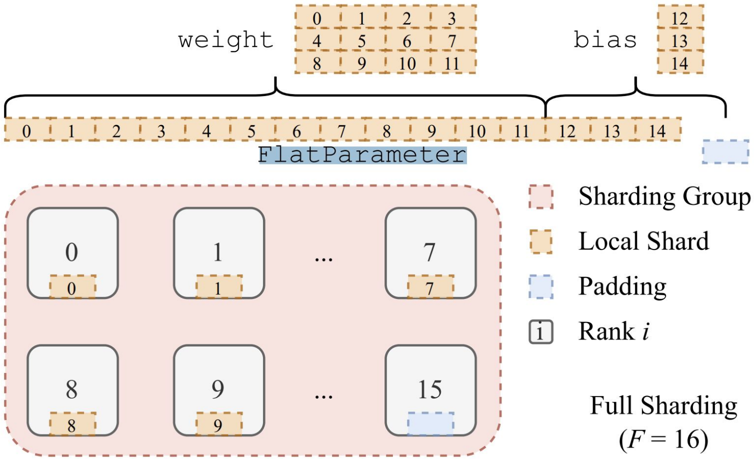
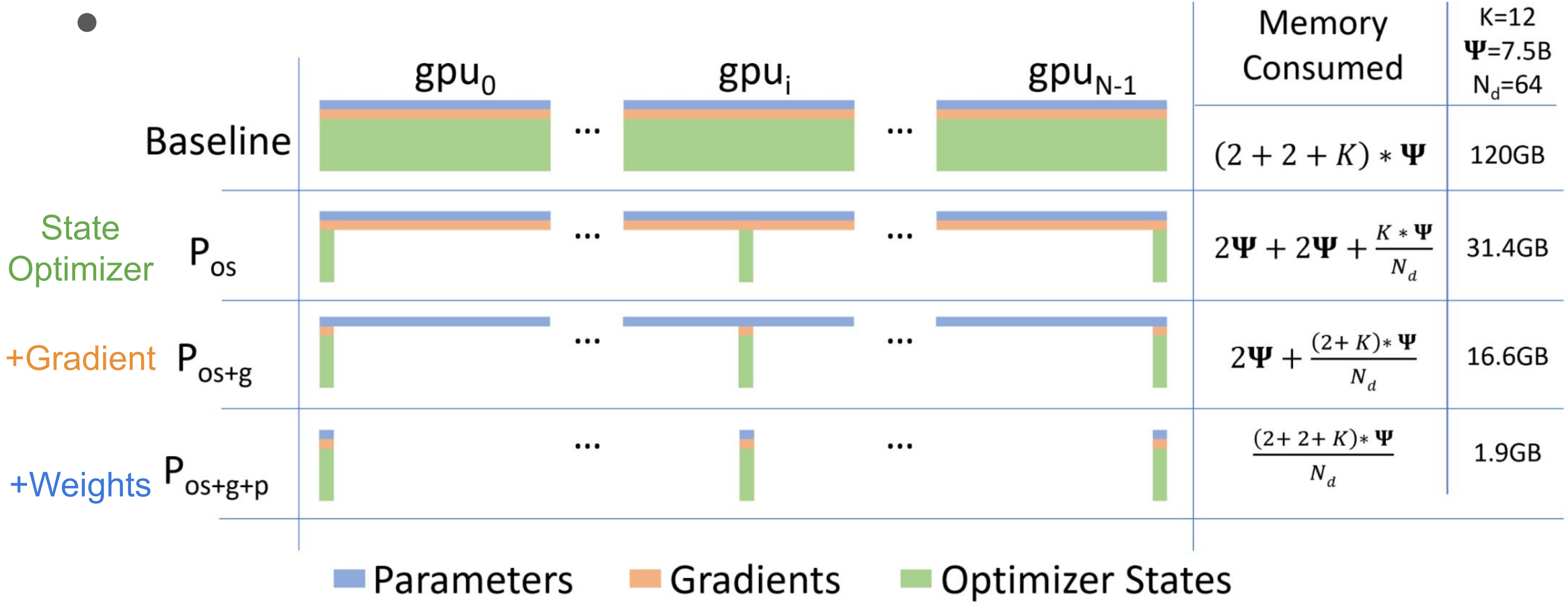


Figure 3: Full Sharding Across 16 GPUs

How FSDP Works? Sharding Strategy

- 

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models https://arxiv.org/abs/1910.02054

How FSDP Works? Sharding Strategy

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=False*, *use_orig_params=False*, *ignored_parameters=None*) [SOURCE]

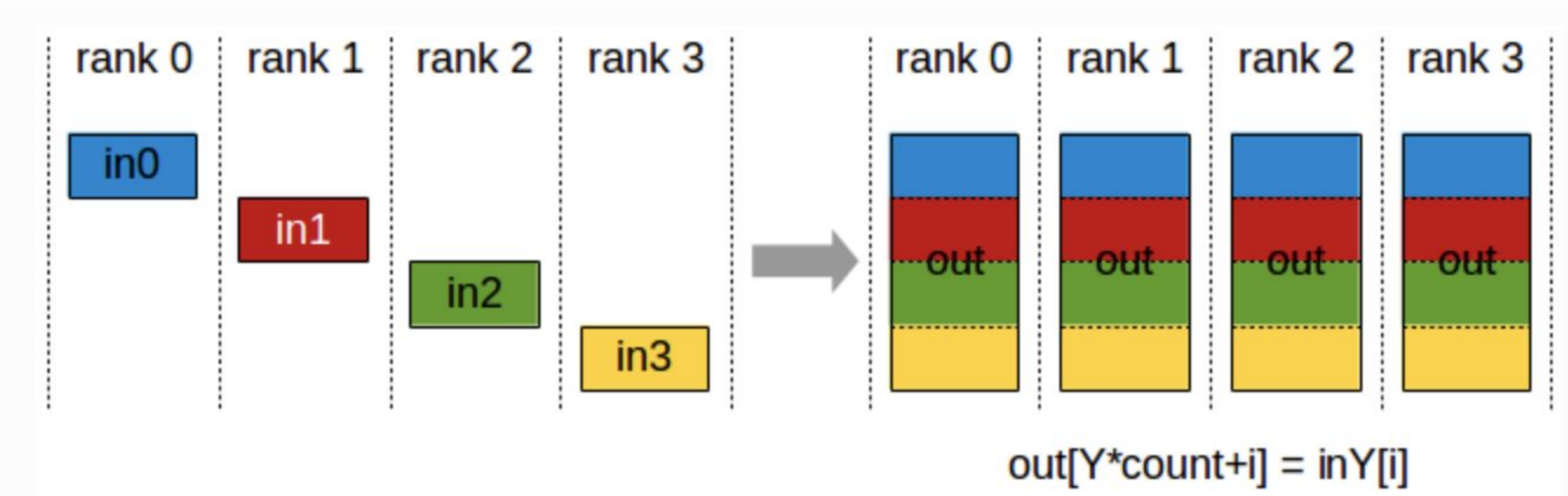
https://www.youtube.com/watch?v=a3iW6Cggccw

How FSDP Works?

More Terminology!!

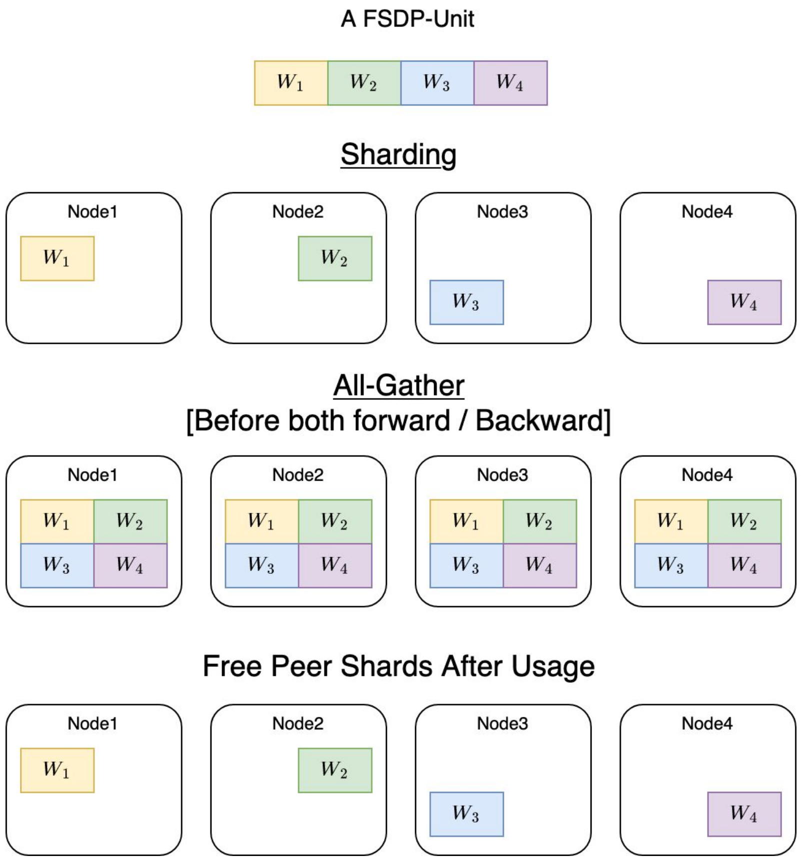
1. FSDP Unit
2. Sharding
3. **All-Gather [Both Forward + Backward]**
4. Reduce-Scatter

How FSDP Works? All-Gather



https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html#allgather

How FSDP Works? All-Gather



How FSDP Works? All-Gather

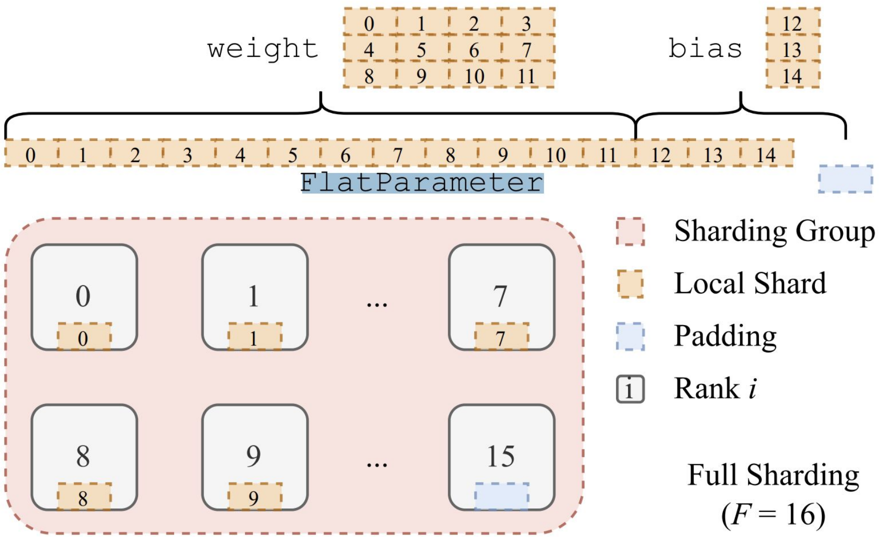
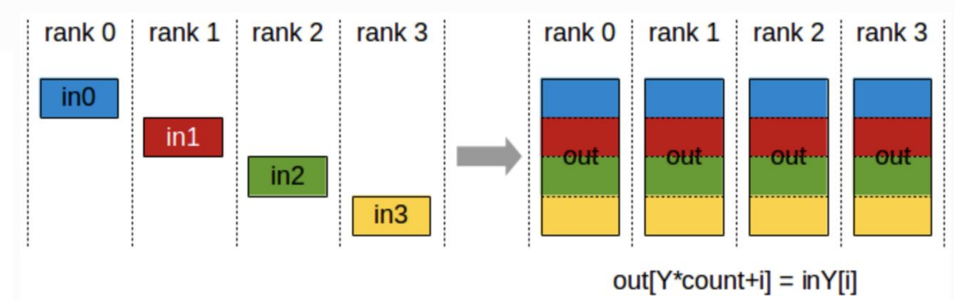


Figure 3: Full Sharding Across 16 GPUs



https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html#allgather

How FSDP Works? All-Gather

- We split our FSDP-Unit parameters across GPUs
- We need all-gather *per* FSDP-unit
  - Forward
  - Backward

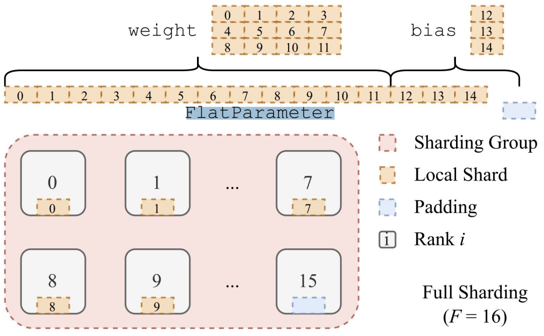
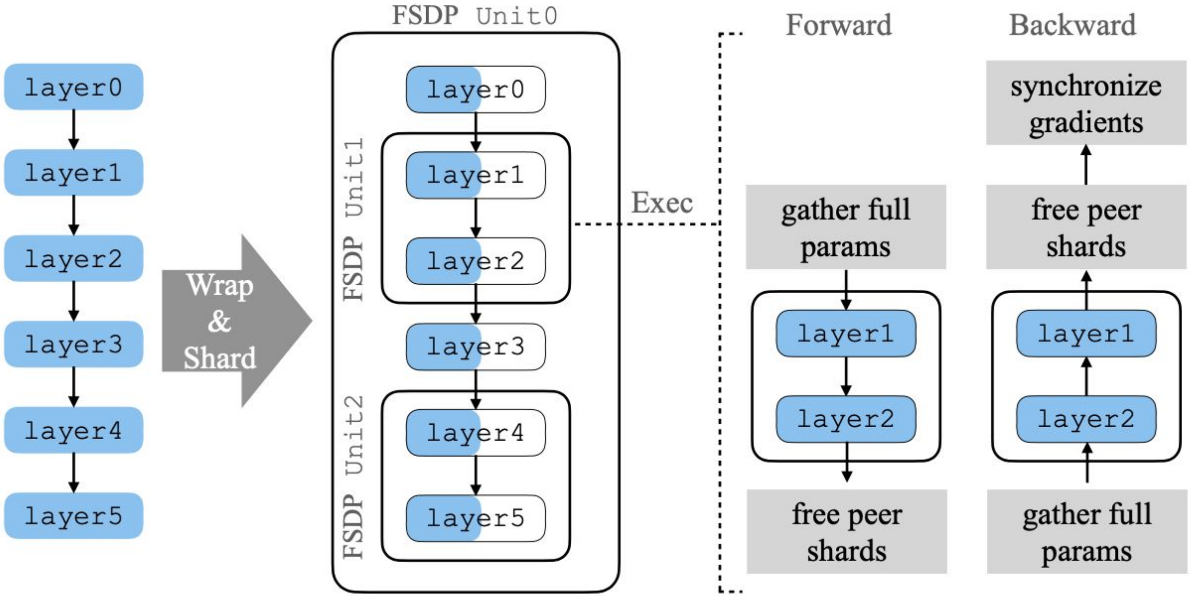


Figure 3: Full Sharding Across 16 GPUs

and gradients sharded. The memory requirements for FSDP are proportional to the size of the sharded model plus the size of the largest fully-materialized FSDP unit.

PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel https://arxiv.org/abs/2304.11277

How FSDP Works? All-Gather



PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel https://arxiv.org/abs/2304.11277

How FSDP Works? All-Gather

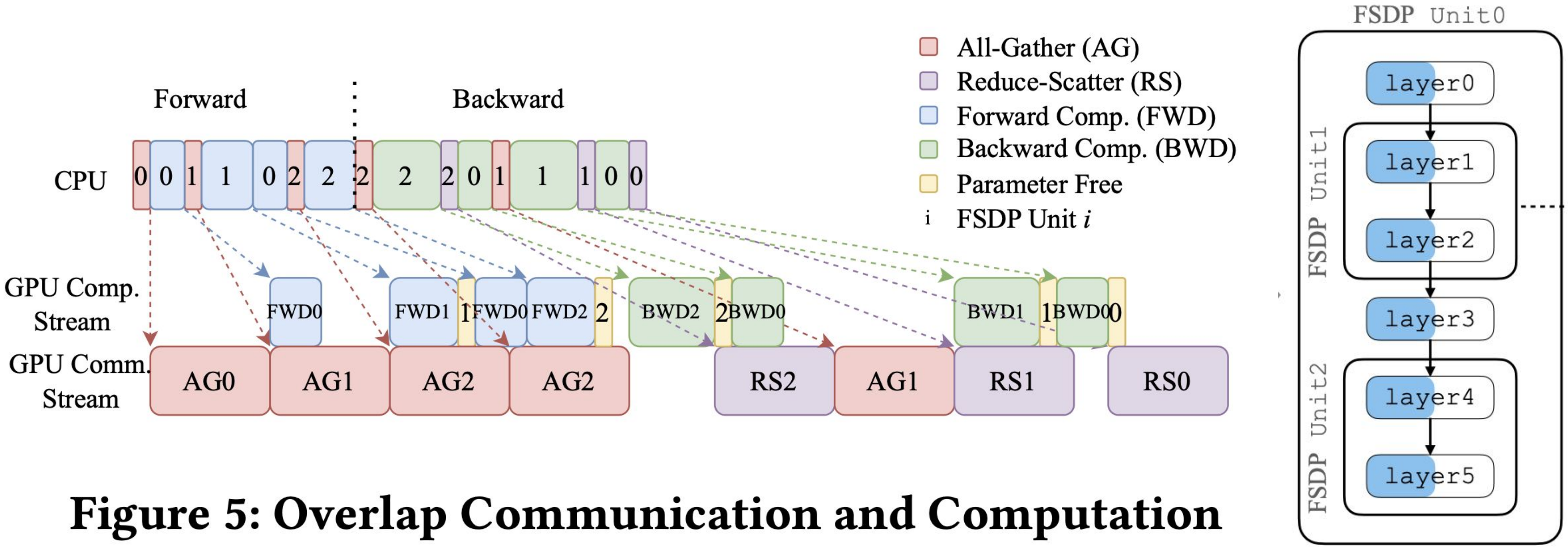


Figure 5: Overlap Communication and Computation

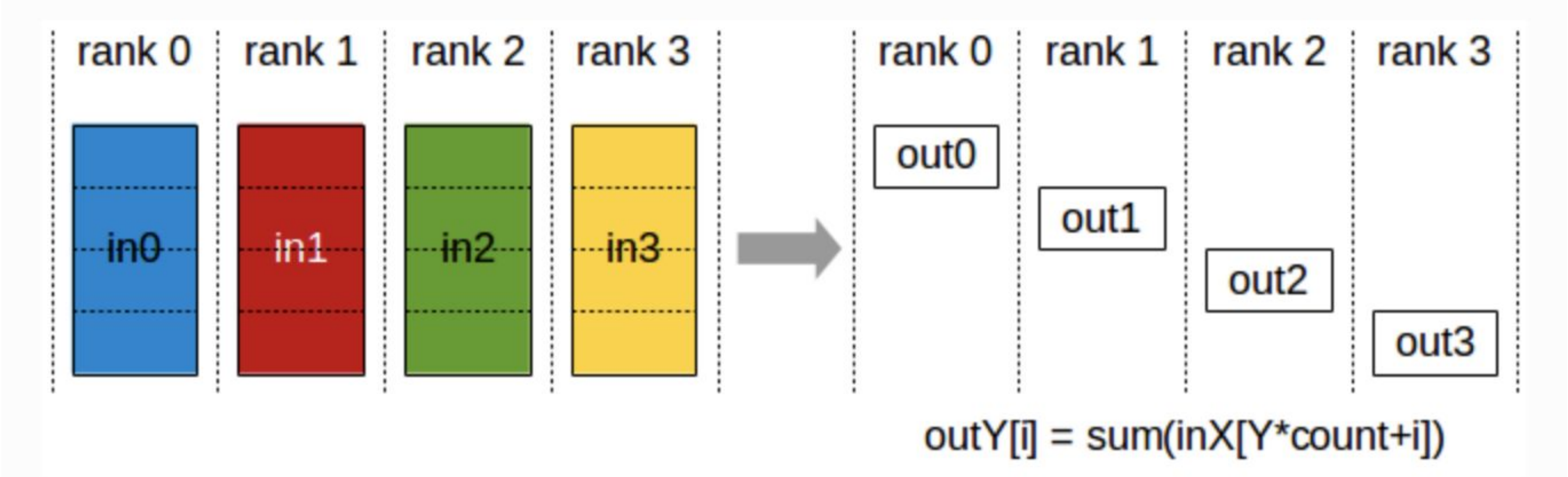
PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel https://arxiv.org/abs/2304.11277

How FSDP Works?

More Terminology!!

1. FSDP Unit
2. Sharding
3. All-Gather [Both Forward + Backward]
4. **Reduce-Scatter**

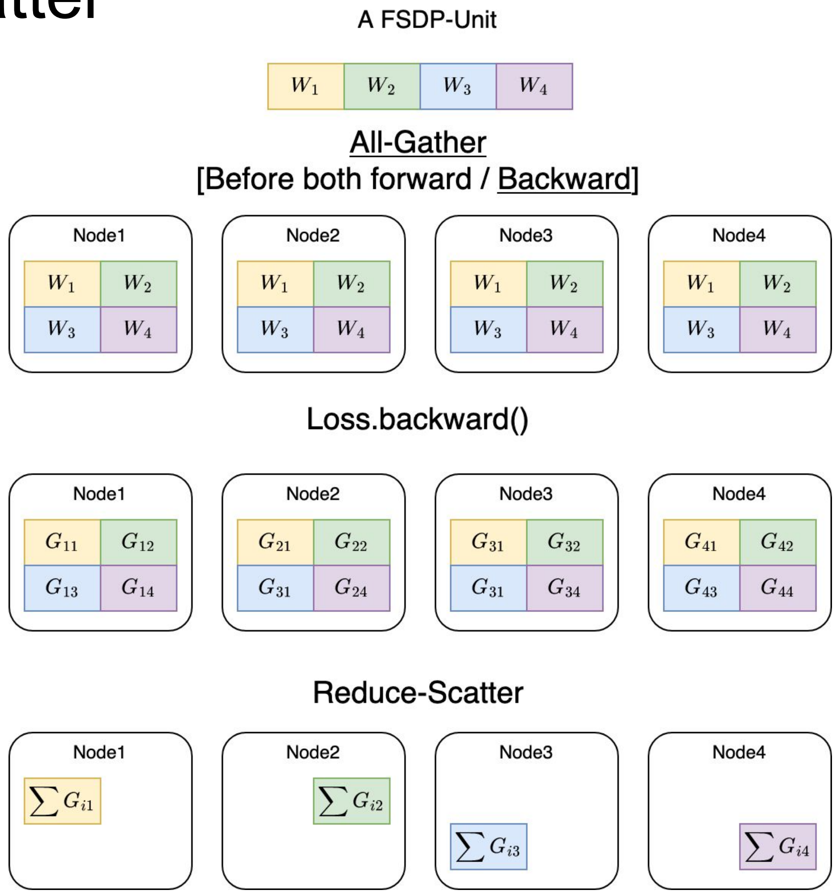
How FSDP Works? Reduce-Scatter



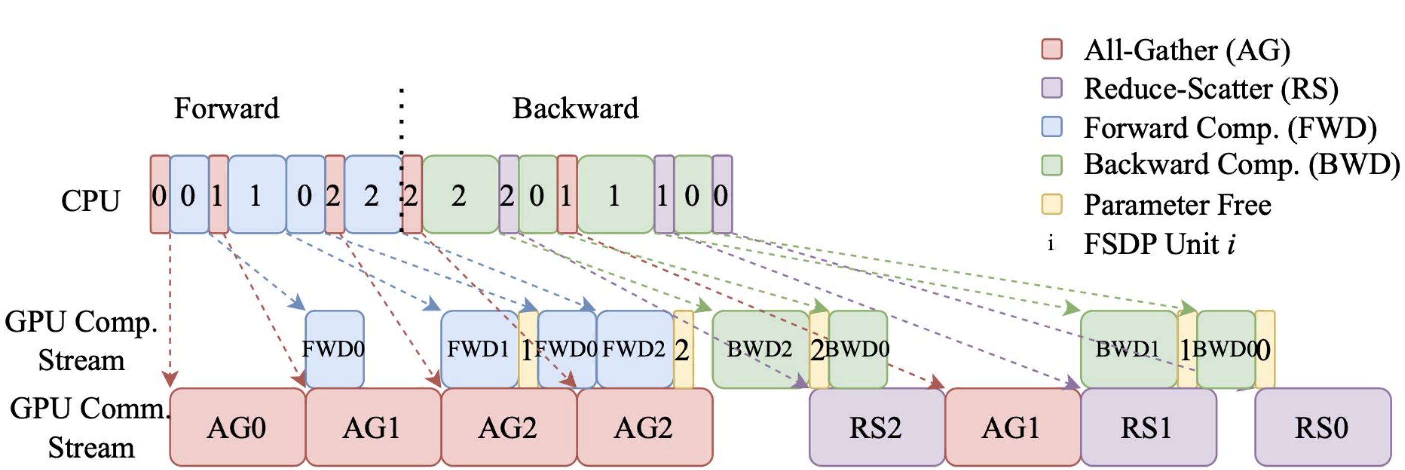
https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html#reducescatter

How FSDP Works? Reduce-Scatter

- After all-gather
  - all nodes have the same W_i
  - But different nodes have different G_i
  - Why?



How FSDP Works? Reduce-Scatter



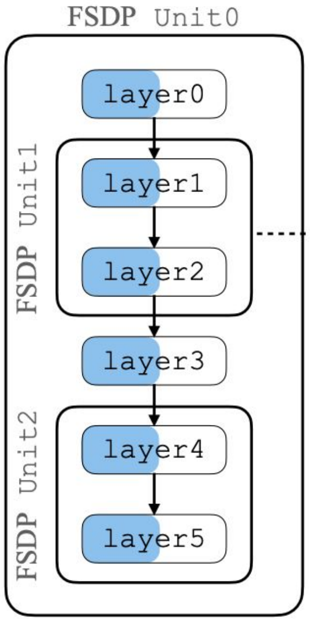


Figure 5: Overlap Communication and Computation

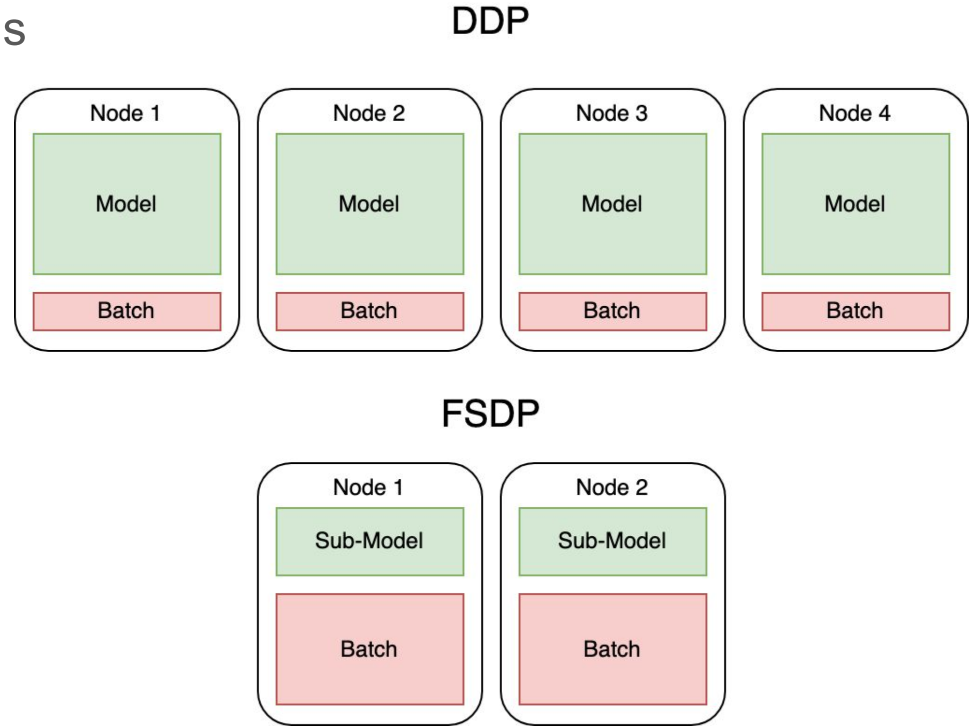
PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel https://arxiv.org/abs/2304.11277

Agenda

- Definitions, Terminology & Terminology
- How DDP works?
- How FSDP works?
- **Reasons to use FSDP**
- **Reasons not to use FSDP**

Reasons to use FSDP

- Train Billion-size Models
- More communication between GPUs
- Trade memory for time



Reasons to use FSDP

- It is simple :)
- Two code changes
  - Wrap our network using FSDP instead as DDP
  - Loading/saving checkpoints

ddp_trainer

```
ddp_model = DistributedDataParallel(
    model,
    device_ids=[comm.get_local_rank()],
    find_unused_parameters=True
)
```

fsdp_trainer

```
>>> module = MyModule(device="meta")
>>> def my_init_fn(module):
>>>     # responsible for initializing a module, such as with
reset_parameters
>>>     ...
>>> fsdp_model = FSDP(module, param_init_fn=my_init_fn,
auto_wrap_policy=size_based_auto_wrap_policy)
```

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

Agenda

- Definitions, Terminology & Terminology
- How DDP works?
- How FSDP works?
- Reasons to use FSDP
- **Reasons not to use FSDP**

Reasons not to use FSDP

- FSDP is developed for **billion-size** models
  - For models < 100 million parameter, consider activation-checkpointing or reversible layers
- FSDP is recently – mid 2023 – supported in PyTorch
  - Mixed Precision requires PyTorch 1.13
  - bfloat – which is highly recommended – requires Ampere GPUs (A100, A6000)
  - Float16 requires **Sharded**GradScaler
- We are expected to incur more communication cost across GPUs.

```
# Recommend using BFloat16 if possible.
# FP16 runs 4% slower vs Bfloat16, all things equal, likely due to cost of rescaling.
# Rescaler has to play guessing game of how much to rescale,
# bad guesses mean that mini-batch is tossed due to having NAN values (inefficient)
```

https://www.youtube.com/watch?v=-caN92JtKqA&list=PL_IsbAsL_o2BT6aerEKgIoufVD_fodnuT&index=4

Reasons not to use FSDP

- *Built-in wrapping policy for creating FSDP units*
- *Generic but less efficient (communication wise)*

```
>>> module = MyModule(device="meta")
>>> def my_init_fn(module):
>>>     # responsible for initializing a module, such as with
reset_parameters
>>>     ...
>>> fsdp_model = FSDP(module, param_init_fn=my_init_fn,
auto_wrap_policy=size_based_auto_wrap_policy)
```

- *Custom wrapping policy for creating FSDP units*
- *Arch-Specific but more efficient*

```
transformer_auto_wrapper_policy = functools.partial(
        transformer_auto_wrap_policy,
        transformer_layer_cls={
            T5Block,  # < ---- Your Transformer layer class
        },
    )
```

https://www.youtube.com/watch?v=HQeKwCsnH4k&list=PL_lsbAsL_o2BT6aerEKgIoufVD_fodnuT&index=2

Reasons not to use FSDP

- Partial fine-tuning is not trivial (needs non-pytorch code)

https://www.youtube.com/watch?v=BCGP9UtWWzk&list=PL_IsbAsL_o2BT6aerEKgIoufVD_fodnuT&index=10

Resources

- PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel
- ZeRO: Memory Optimizations Toward Training Trillion Parameter Models
- Multi-GPU Training Strategies
- Nvidia NCCL Operations
- PyTorch FSDP Tutorials
- Fully Sharded Data Parallel: faster AI training with fewer GPUs