Finetuning with Accelerate FSDP

Riccardo Scheda
r.scheda@cineca.it

January 2026

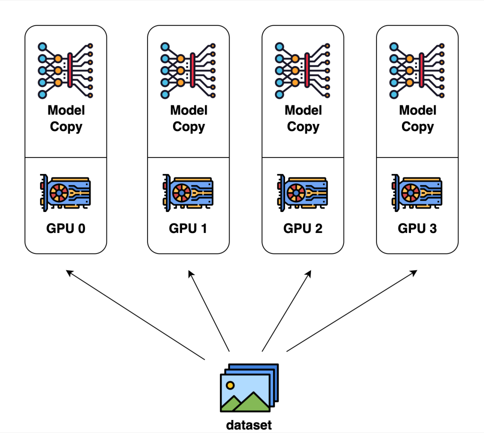
WHY PARALLEL TOOLS IN AI?

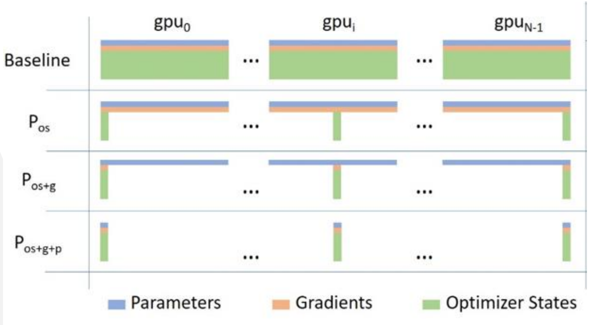
DIFFERENT TYPES OF PARALLELISM

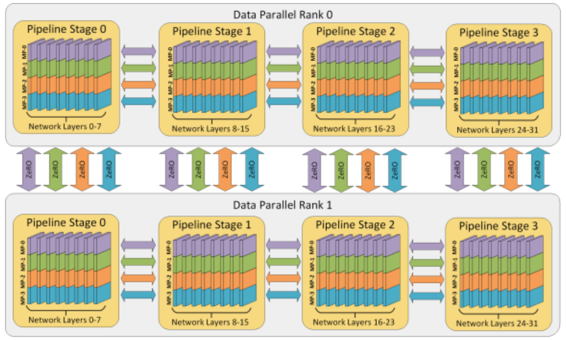
Data Parallelism

Model Sharding

Model Parallelism





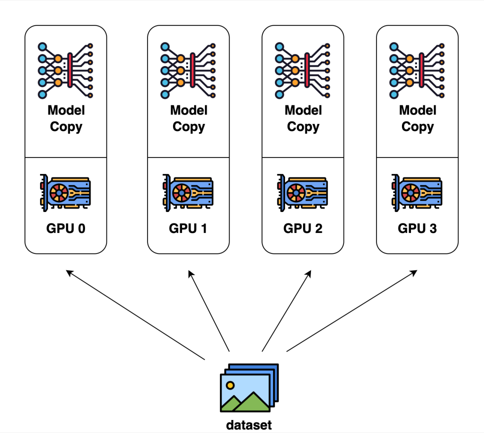


https://colossalai.org/docs/concepts/paradigms_of_parallelism/

https://www.deepspeed.ai/tutorials/pipeline/

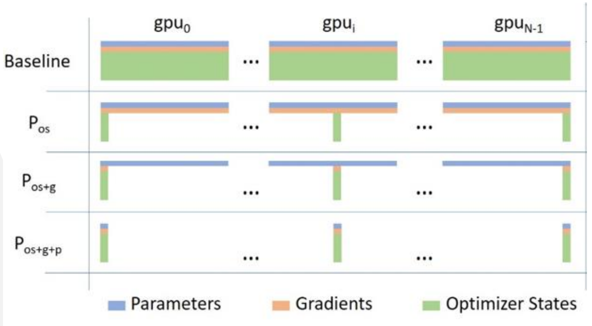
DIFFERENT TYPES OF PARALLELISM

Data Parallelism

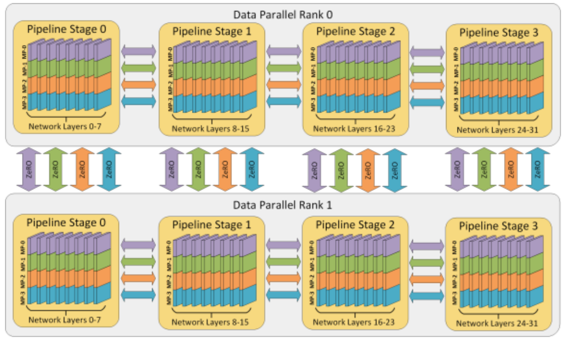


https://colossalai.org/docs/concepts/paradigms_of_parallelism/

Model Sharding

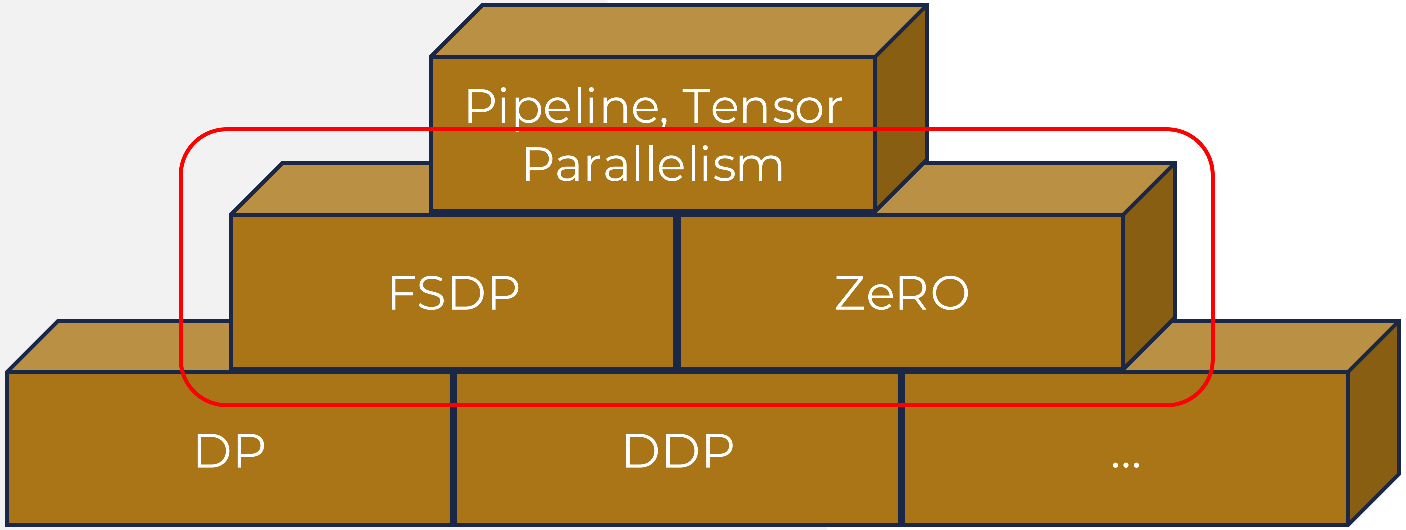


Model Parallelism



https://www.deepspeed.ai/tutorials/pipeline/

DIFFERENT TYPES OF PARALLELISM



Model Parallelism

Model Sharding

Data Parallelism

The Ultra-Playbook Cheatsheet

Step 1: Fit model into memory

GPU rich case: 🤑

- **Small models (<10B):** use a single parallelism technique, e.g. TP or ZeRO-3/DP with Full Recompute across 8 GPUs.
- **Large models (10B+):** requires more than 8 GPUs, you have several options:
  - Combining Tensor Parallelism (TP=8) with Pipeline Parallelism
  - Combining Tensor Parallelism (TP=8) with Data Parallelism (ZeRO-3)
  - Using only ZeRO-3 (i.e. only pure Data Parallelism)
- **512+ GPU scale**: pure DP/ZeRO-3 becomes inefficient due to communication cost - better to then combine DP with either TP or PP
- **1024+ GPU scale**, a recommended setup can be TP=8 with DP (ZeRO-2) and PP
- Special cases: for **long context** consider CP and for **MoE** arch use EP

GPU poor case 😭:

- **Reduce memory:** use full activation checkpointing and/or gradient accumulation

Step 2: Satisfy target global batch size

Experiments tell us which batch size is ideal for training (4-40M tokens). So we either have to increase or decrease the batch size based on step 1 to meet it.

Increase Global Batch Size:

- Scale up DP or CP or gradient accumulation steps

Decrease Global Batch Size:

- Reduce DP or CP in favor of other parallelization strategies

Step 3: Optimizing Training Throughput

There is no general recipe for the best configuration so at this point we should experiment:

- **Scale up TP** up to the node size to reduce other parallel strategies
- **Increase DP** with ZeRO-3 while keeping target GBS
- **Use PP** if communication becomes a bottleneck for DP
- Play with **micro batch size** to balance max GBS, model size, compute/comms

Parallelization Strategies

Strategy	Batch Size	Memory Reduction	Compute Reduction	Communication	Compute/Communication Overlap
Data Parallelism	gbs scales with DP	can reduce mbs by increasing dp → reduce activations	can reduce mbs by increasing dp	bwd: allreduce grads_bf16	overlapped with microbatch's backward: `(DP-1) * num_params * peak_flops / (2 * peak_bw * num_tokens * DP)`
DP+ZeRO-1	gbs scales with DP	model_fp32/dp optimstates/dp	can reduce mbs by increasing dp	bwd: allreduce grads_bf16 step_end: allgather model_fp32	Same as above
DP+ZeRO-2	gbs scales with DP	model_fp32/dp grads_fp32/dp optimstates/dp	can reduce mbs by increasing dp	bwd: reduce-scatter grads_bf16 step_end: allgather model_fp32	overlapped with microbatch's backward: `(DP-1) * num_params * peak_flops / (4 * peak_bw * num_tokens * DP)`
DP+ZeRO-3 (FSDP)	gbs scales with DP	model_bf16/dp model_fp32/dp grads_fp32/dp optimstates/dp	can reduce mbs by increasing dp	(x num_layers) fwd: allgather model_fp32 bwd: allgather model_fp32 bwd: reduce-scatter grads_fp32	overlapped with next layer's fwd/bwd: `(DP-1) * peak_flops / (2 * seq * mbs * peak_bw)`
Tensor Parallelism	No effect	model_bf16/tp model_fp32/tp grads_fp32/tp optimstates/tp activs/tp	model_bf16/tp	(x 4 x num_layers) fwd: allgather activs_bf16 bwd: reduce-scatter grads_bf16	overlapped with next TP region (attn/MLP/layernorm): `(TP-1) * peak_flops / (24 * hidden_size * peak_bw)`
Pipeline Parallelism (1f1b)	prefers large gas to reduce bubble	model_bf16/pp model_fp32/pp grads_fp32/pp optimstates/pp	model_bf16/pp	(x gas) fwd: recv activs_bf16 fwd: send activs_bf16 bwd: recv grads_bf16 bwd: send grads_bf16	overlapped with next microbatch's fwd/bwd: `PP * peak_flops / (32 * hidden_size * num_layers * peak_bw)`
Context Parallelism	prefers large seq for better overlap	activations/cp	seq/cp	(x cp-1 x num_layers) fwd: send/recv activs_bf16 bwd: send/recv grads_bf16	Overlap with attention computation(ring attention): `(CP-1) * B * L/CP * H_kv × (num_k + num_v)`
Expert Parallelism	Batch size scales with EP	experts/ep	experts/ep	(x num_layers) fwd: all2all activs_bf16 bwd: all2all grads_bf16	overlapped with MoE block `(EP-1) * peak_flops / (12 * num_experts * hidden_size * peak_bw)`

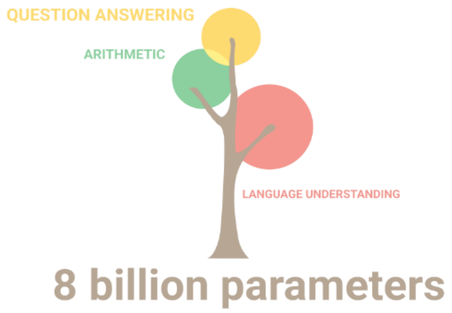
Click HERE
for a close up

MODEL SHARDING

WHY MODEL SHARDING?

Full-Precision : fp32 (4 bytes)		
Params	7B x 4	28 GB
Activation	7B x 4	28 GB
Gradients	7B x 4	28 GB
Optimizer	7B x 4 x 2	56 GB
TOT		**140 GB**

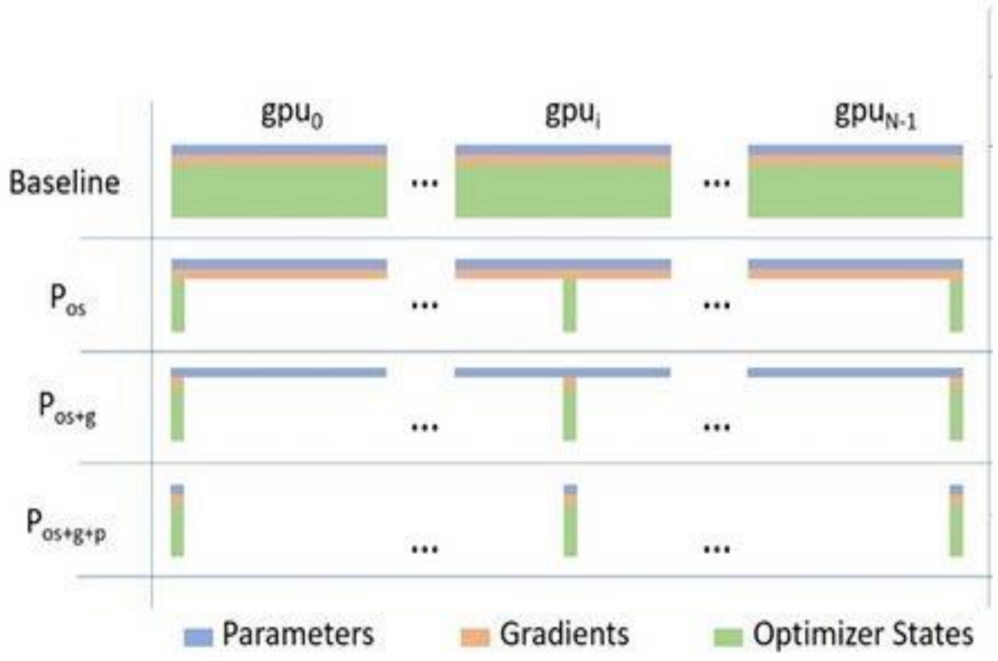
Half-Precision : fp16 or bf16 (2 bytes)		
Params	7B x 2	14 GB
Activation	7B x 2	14 GB
Gradients	7B x 2	14 GB
Optimizer	7B x 2 x 2	28 GB
TOT		**70 GB**



In Leonardo we have A100 GPUs with 64 GB each! → OoM error!

SHARDING-HOW IT WORKS

- Each device holds a **"shard"** of the model **parameters**, **optimizer states** and **gradients across GPUs**
- Data are **split** on multiple GPUs (single or multi-node)
- Enables **larger** models **training** (the model Does not need to fit in a single gpu)
- Replace **All Reduce** communication With **Reduce Scatter+ All Gather**

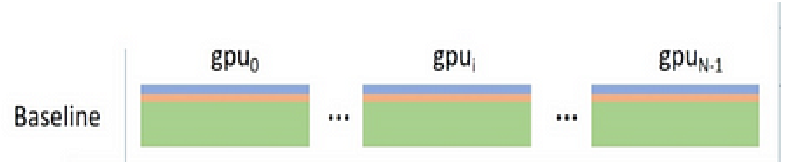


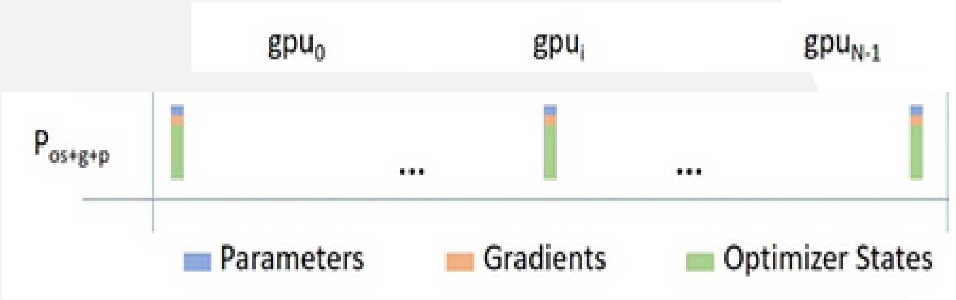
[1] https://medium.com/@pranay.janupalli/understanding-model-sharding-and-model-parallelism-scaling-large-language-models-dee6144d0591#:~:text=%E2%80%94%20Model%20Sharding:%20Usually%20involves%20a,types%20of%20operations%20across%20devices
[2] https://huggingface.co/docs/accelerate/usage_guides/deepspeed

SHARDING-HOW IT WORKS

DDP training :

- each worker (GPU) owns a replica of the entire model.
- it uses all-reduce to sum up only the gradients over different workers, (but the model weights and optimizer states are replicated across all workers)





Model Sharding, across DDP ranks, shard either:

- model parameters,
- optimizer states
- gradients

[1] https://medium.com/@pranay.janupalli/understanding-model-sharding-and-model-parallelism-scaling-large-language-models-dee6144d0591#:~:text=%E2%80%94%20Model%20Sharding:%20Usually%20involves%20a,types%20of%20operations%20across%20devices
[2] https://huggingface.co/docs/accelerate/usage_guides/deepspeed

SHARDING-HOW IT WORKS

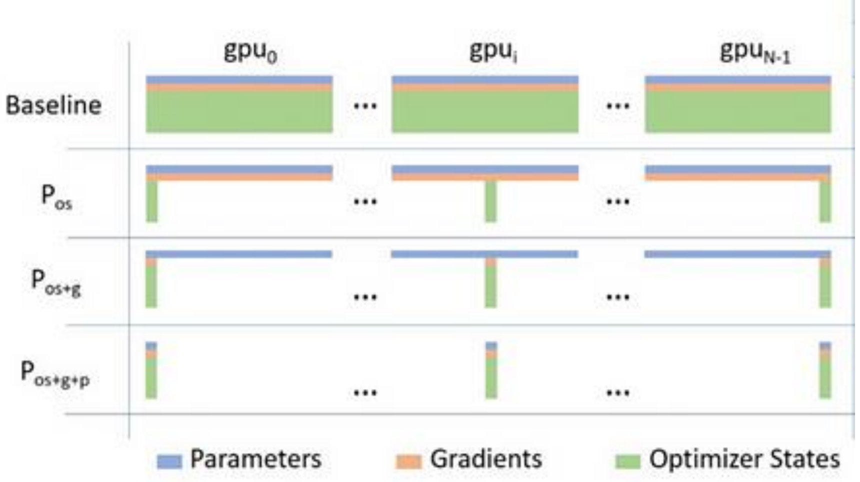
1

ZeRO (DeepSpeed)

Zero Redundancy Optimizer

Stage 1
Stage 2
Stage 3

https://www.deepspeed.ai/tutorials/zero/
https://huggingface.co/docs/accelerate/usage_guides/deepspeed

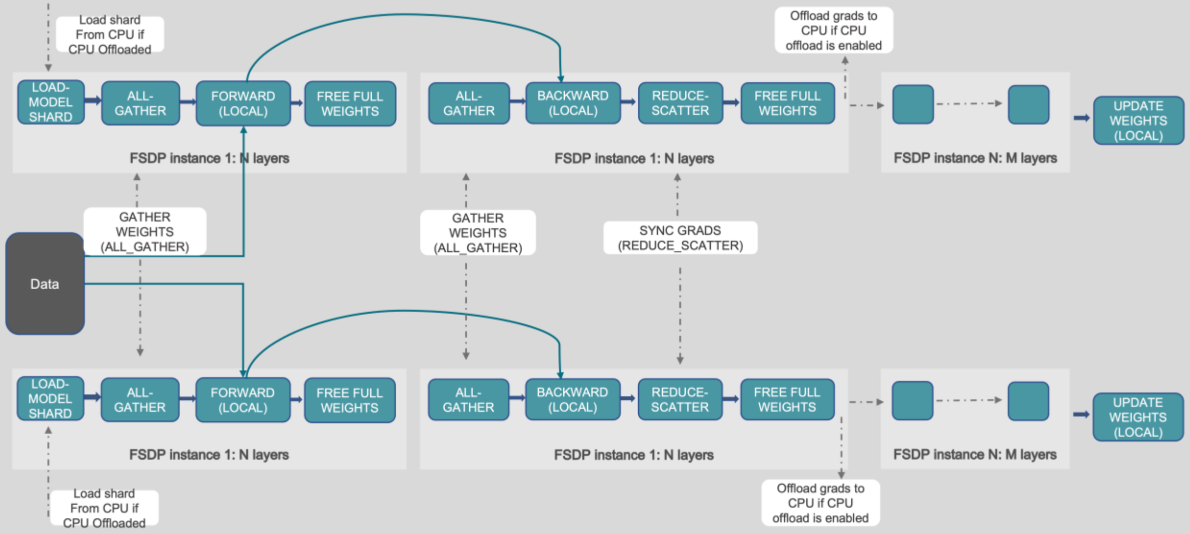


2

FSDP (PyTorch)

Fully Sharded Data Parallelism

https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html#how-fsdp-works
https://huggingface.co/docs/accelerate/usage_guides/fsdp



FSDP and ZeRO Stage 3

PROs & CONs

PROs :
GPU memory footprint is smaller than when training with DDP across all workers.
Allow the training of some very large models by allowing larger models or batch sizes to fit on device.

CONs :
cost of increased communication volume.

DEEPSPEED ZeRO

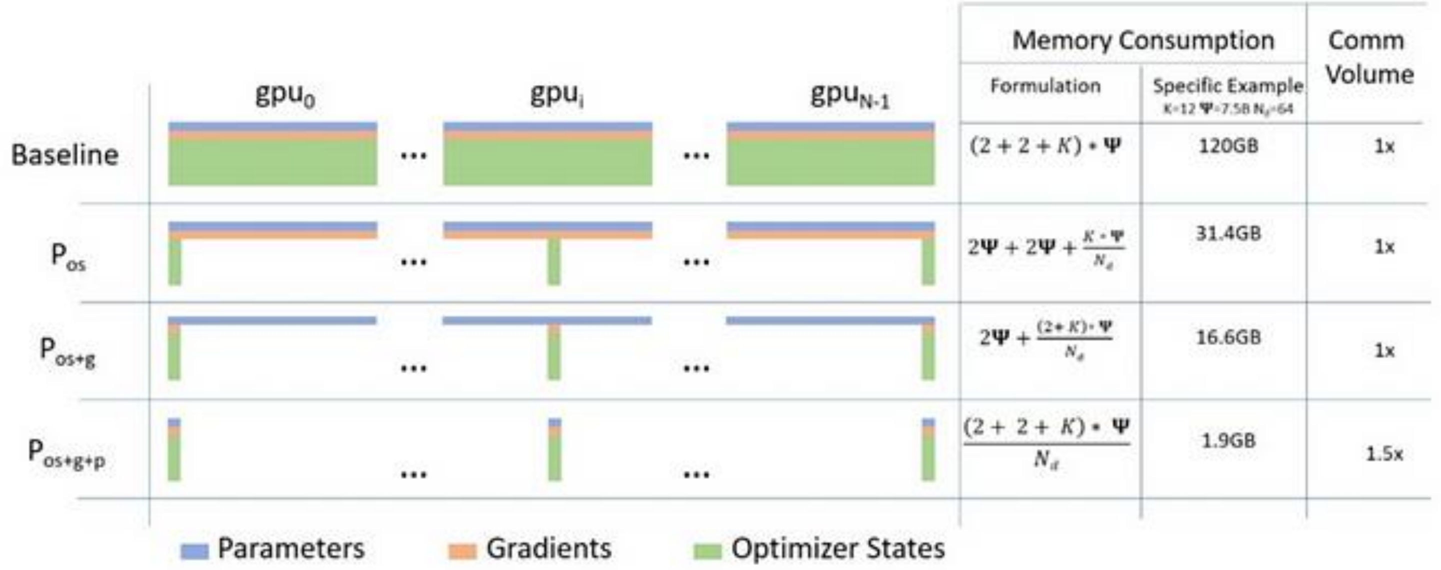


Figure 1: Memory savings and communication volume for the three stages of ZeRO compared with standard data parallel baseline. In the memory consumption formula, Ψ refers to the number of parameters in a model and K is the optimizer specific constant term. As a specific example, we show the memory consumption for a 7.5B parameter model using Adam optimizer where K=12 on 64 GPUs. We also show the communication volume of ZeRO relative to the baseline.

DEEPSPEED ZeRO

- **Stage 1**: partition of the **optimizer states.** Each process updates only its partition

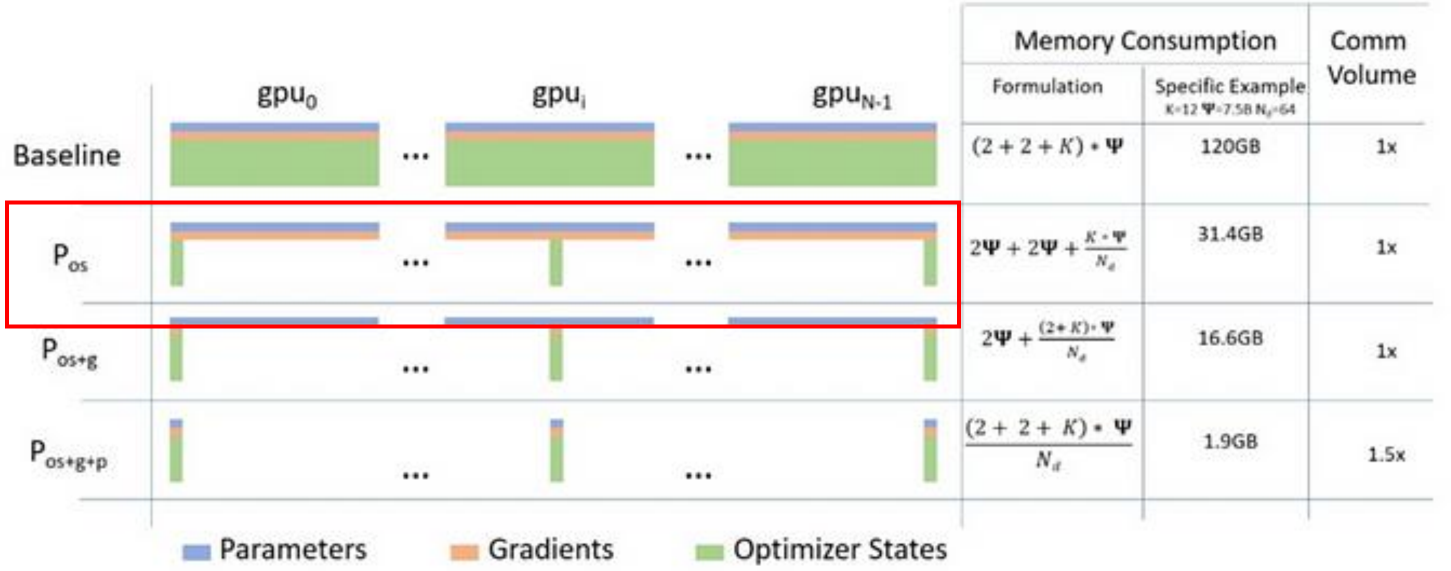


Figure 1: Memory savings and communication volume for the three stages of ZeRO compared with standard data parallel baseline. In the memory consumption formula, Ψ refers to the number of parameters in a model and K is the optimizer specific constant term. As a specific example, we show the memory consumption for a 7.5B parameter model using Adam optimizer where K=12 on 64 GPUs. We also show the communication volume of ZeRO relative to the baseline.

DEEPSPEED ZeRO

- **Stage 1**: partition of the **optimizer states.** Each process updates only its partition

- **Stage 2**: Partition also of the **gradients**. Each process retains only the gradients corresponding to its portion of the optimizer states.

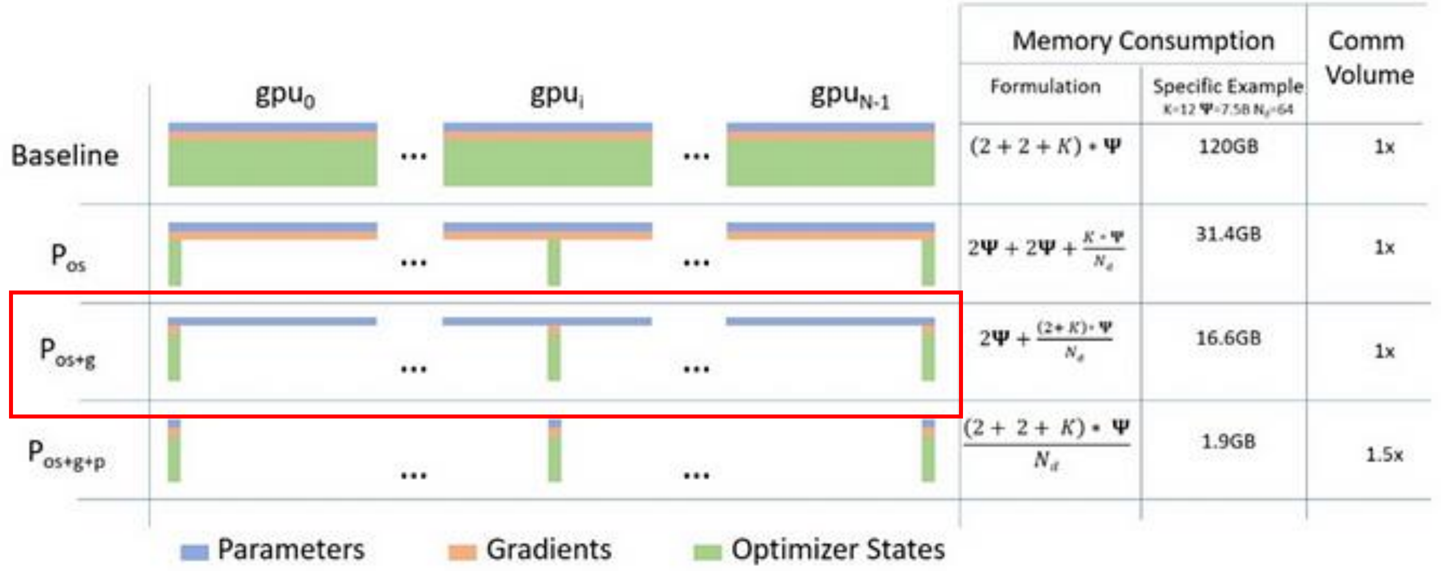


Figure 1: Memory savings and communication volume for the three stages of ZeRO compared with standard data parallel baseline. In the memory consumption formula, Ψ refers to the number of parameters in a model and K is the optimizer specific constant term. As a specific example, we show the memory consumption for a 7.5B parameter model using Adam optimizer where K=12 on 64 GPUs. We also show the communication volume of ZeRO relative to the baseline.

DEEPSPEED ZeRO

- **Stage 1**: partition of the **optimizer states.** Each process updates only its partition

- **Stage 2**: Partition also of the **gradients**. Each process retains only the gradients corresponding to its portion of the optimizer states.

- **Stage 3**: Partition of the **model parameters.** ZeRO-3 automatically collects and partitions them during forward and backward passes.

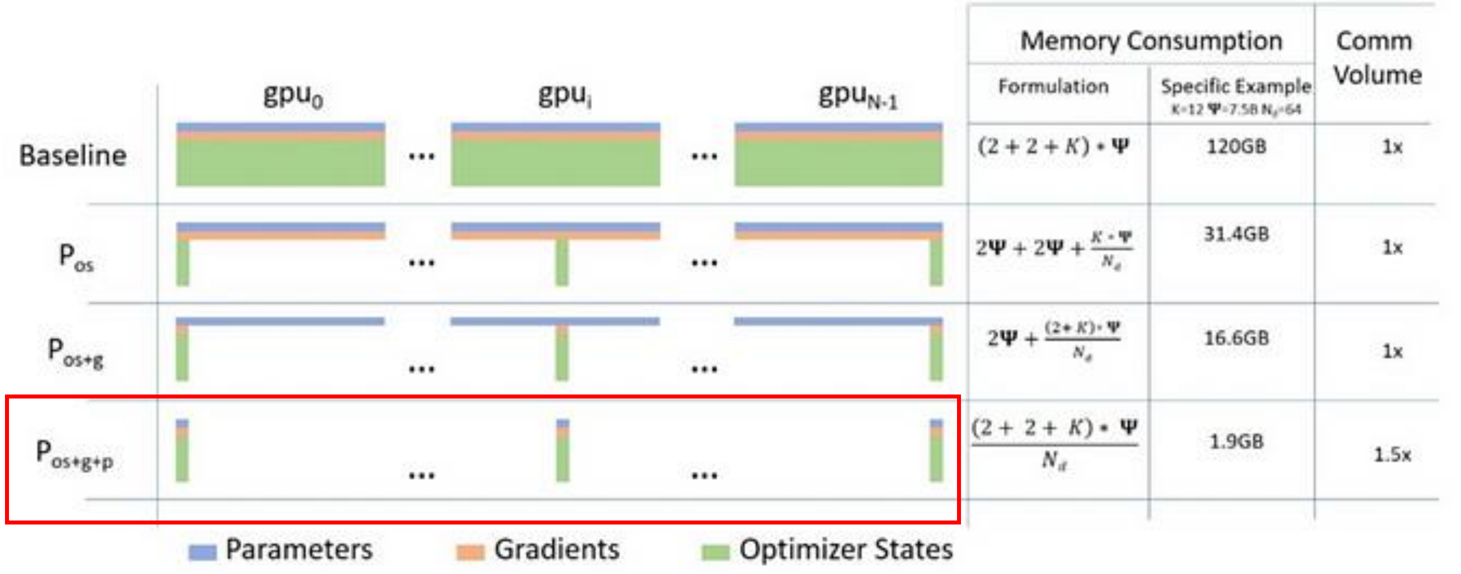


Figure 1: Memory savings and communication volume for the three stages of ZeRO compared with standard data parallel baseline. In the memory consumption formula, Ψ refers to the number of parameters in a model and *K* is the optimizer specific constant term. As a specific example, we show the memory consumption for a 7.5B parameter model using Adam optimizer where *K*=12 on 64 GPUs. We also show the communication volume of ZeRO relative to the baseline.

DEEPSPEED ZeRO

- **Stage 1**: partition of the **optimizer states.** Each process updates only its partition
- **Stage 2**: Partition also of the **gradients**. Each process retains only the gradients corresponding to its portion of the optimizer states.
- **Stage 3**: Partition of the **model parameters.** ZeRO-3 automatically collects and partitions them during forward and backward passes.

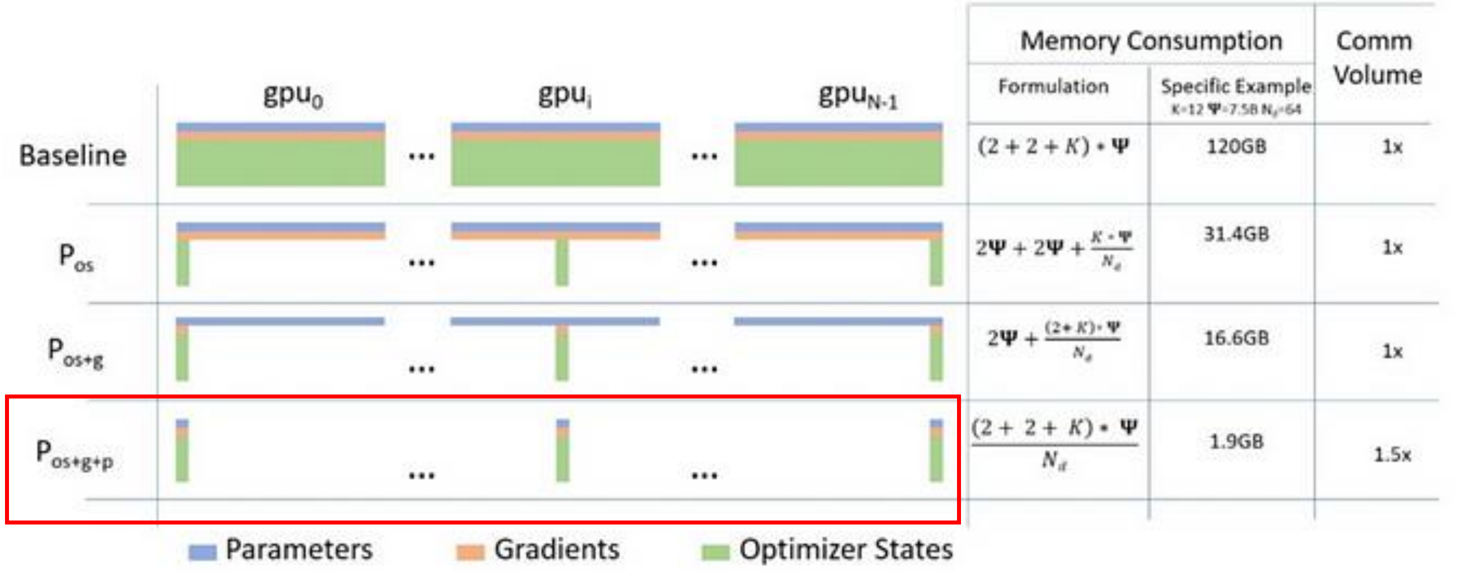


Figure 1: Memory savings and communication volume for the three stages of ZeRO compared with standard data parallel baseline. In the memory consumption formula, Ψ refers to the number of parameters in a model and *K* is the optimizer specific constant term. As a specific example, we show the memory consumption for a 7.5B parameter model using Adam optimizer where *K*=12 on 64 GPUs. We also show the communication volume of ZeRO relative to the baseline.

Same as PyTorch FSDP!!

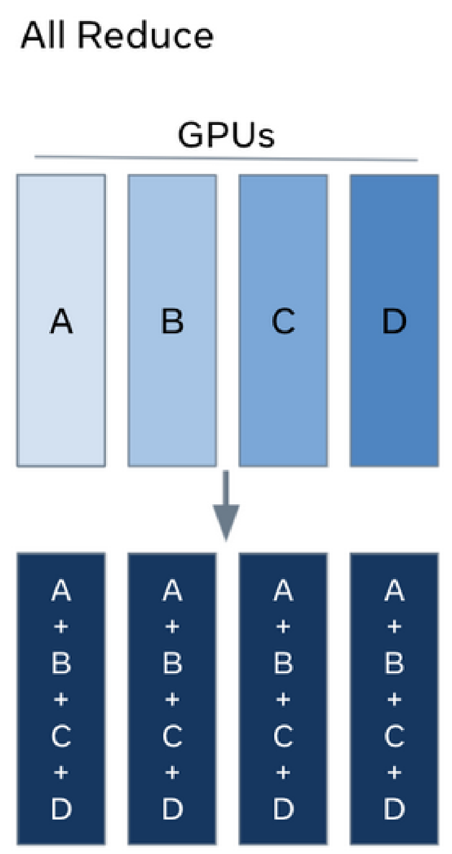
Fully Sharded Data Parallel (FSDP)

COMMUNICATIONS

[1] https://docs.pytorch.org/tutorials/intermediate/FSDP_tutorial.html

COMMUNICATIONS

- **All-Reduce**:
  all ranks receive the result

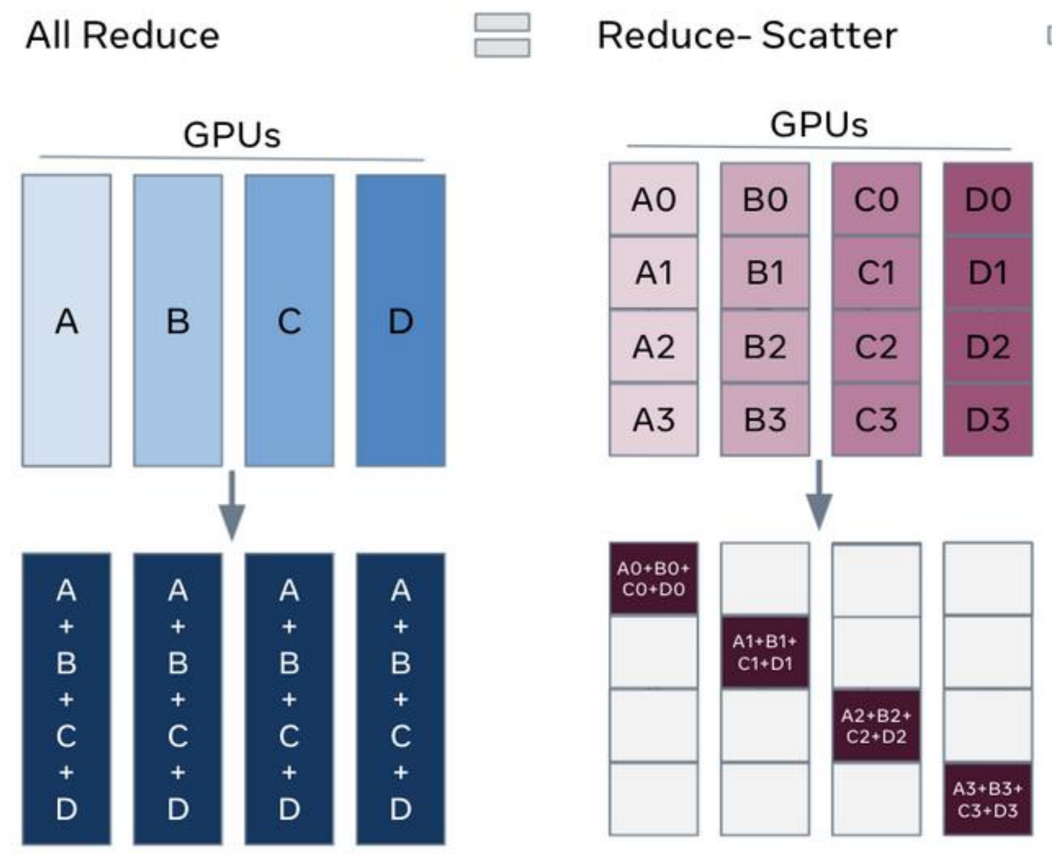


[1] https://docs.pytorch.org/tutorials/intermediate/FSDP_tutorial.html

COMMUNICATIONS

- **All-Reduce**:
  all ranks receive the result

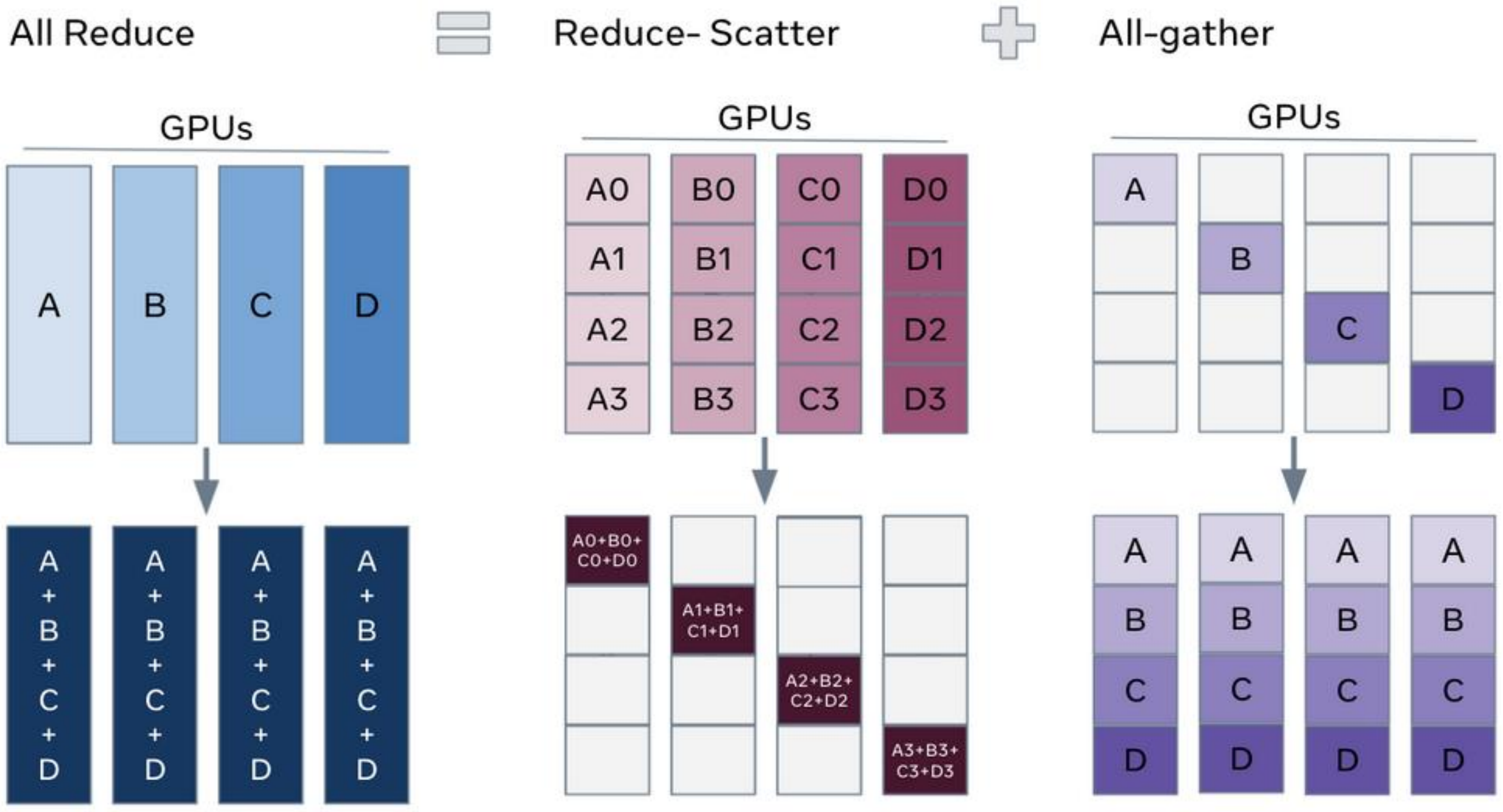
- **Reduce-Scatter**:
  one rank receive
  the result



[1]
https://docs.pytorch.org/tutorials/intermediate/FSDP_tutorial.html

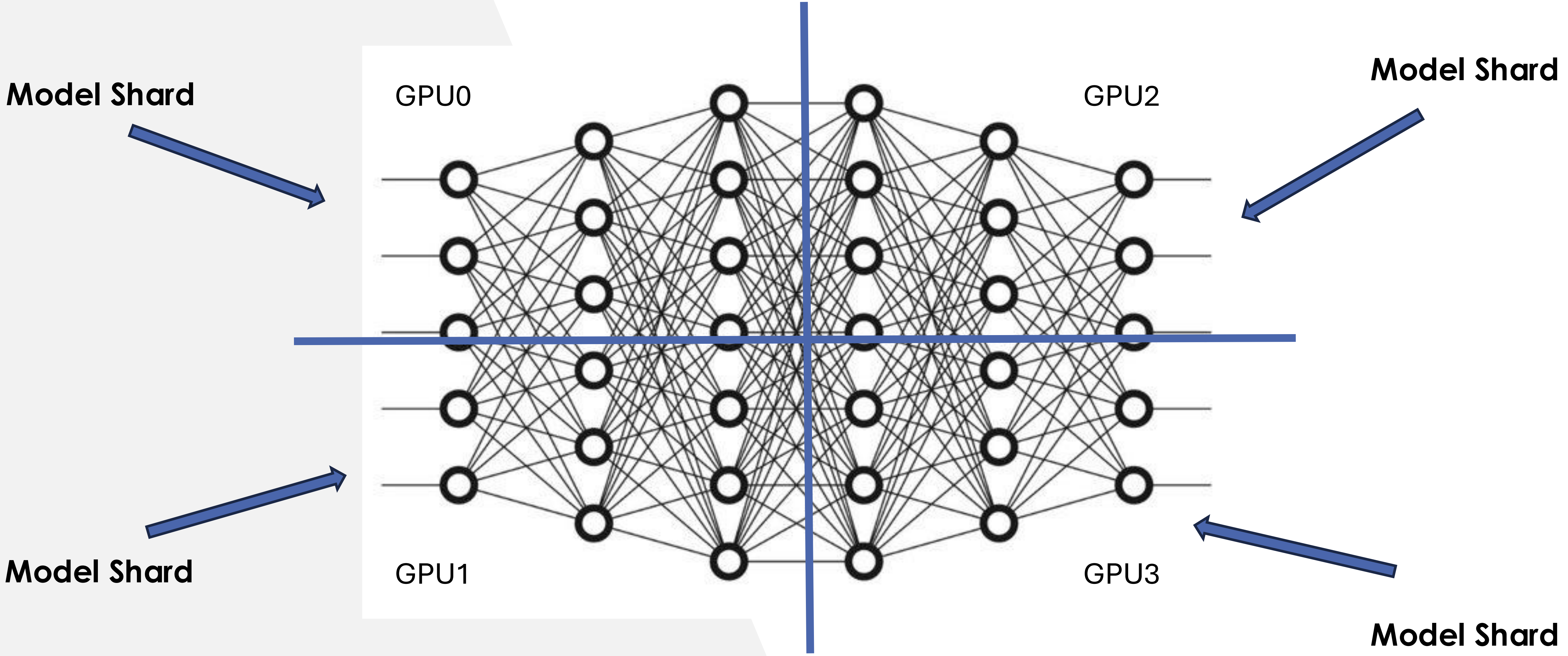
COMMUNICATIONS

- **All-Reduce**:
  all ranks receive the result
- **Reduce-Scatter**:
  one rank receive
  the result
- **All-Gather**:
  each rank receives
  the contributions of other ranks
  (**but not the final result**)



[1] https://docs.pytorch.org/tutorials/intermediate/FSDP_tutorial.html

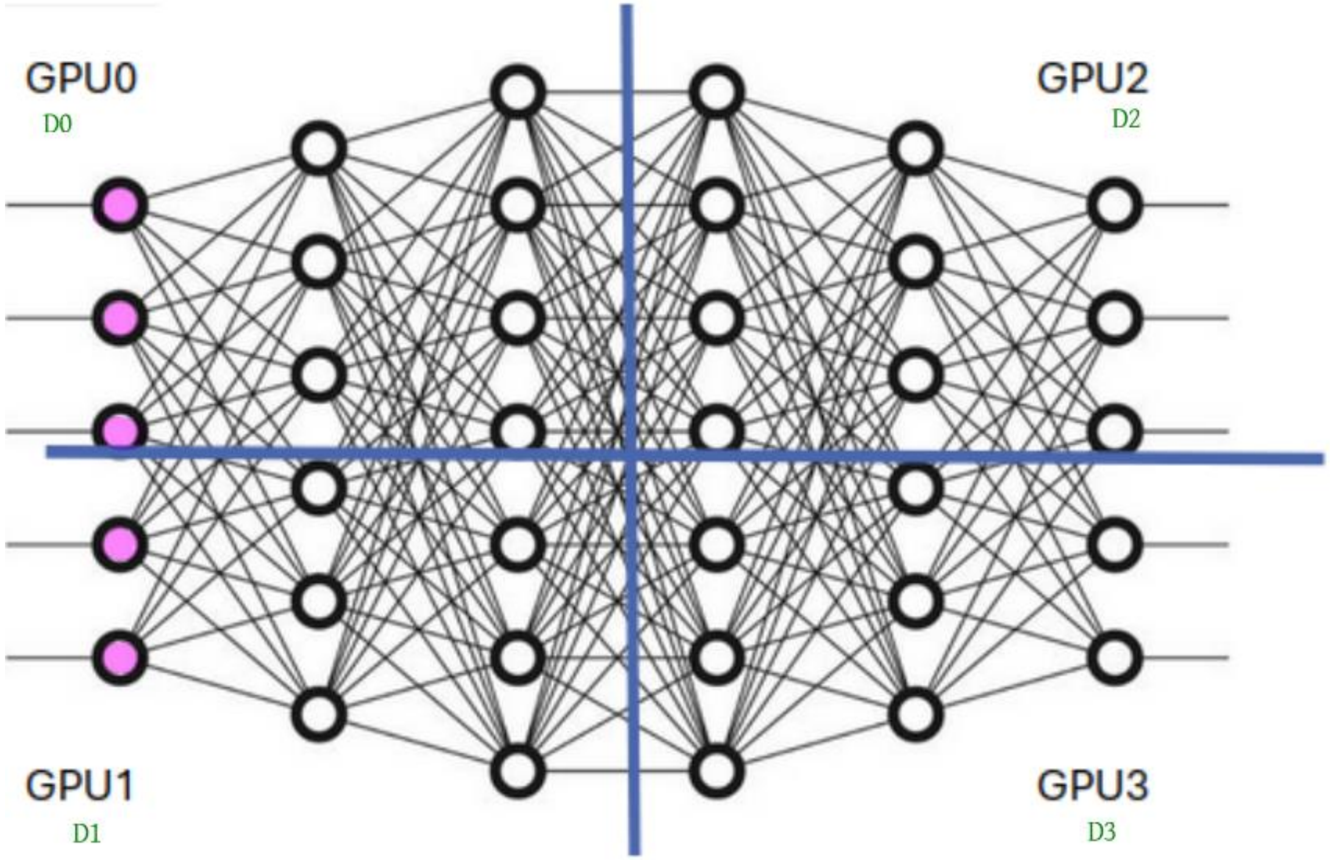
HOW TO SPLIT A MODEL WITH Fully Sharded Data Parallel (FSDP)



HOW TO SPLIT A MODEL WITH FSDP

AllGather Layer 0 (L0):

GPU0 : L0(D0)
GPU1: L0(D1)
GPU2: L0(D2)
GPU3: L0(D3)



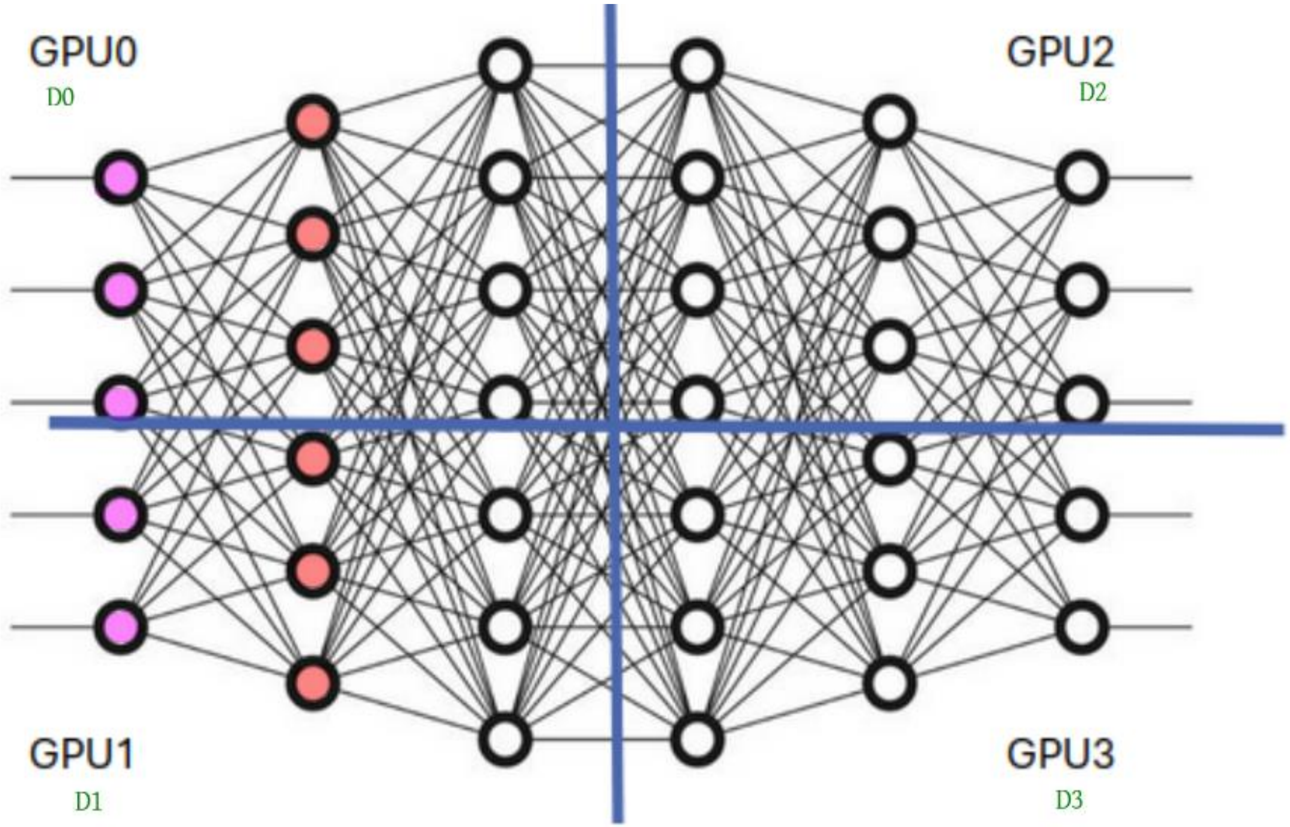
HOW TO SPLIT A MODEL WITH FSDP

AllGather Layer 0 (L0):

GPU0 : L0(D0)
GPU1: L0(D1)
GPU2: L0(D2)
GPU3: L0(D3)

AllGather Layer 1 (L1):

GPU0: L1(L0(D0))
GPU1: L1(L0(D1))
GPU2: L1(L0(D2))
GPU3: L1(L0(D3))



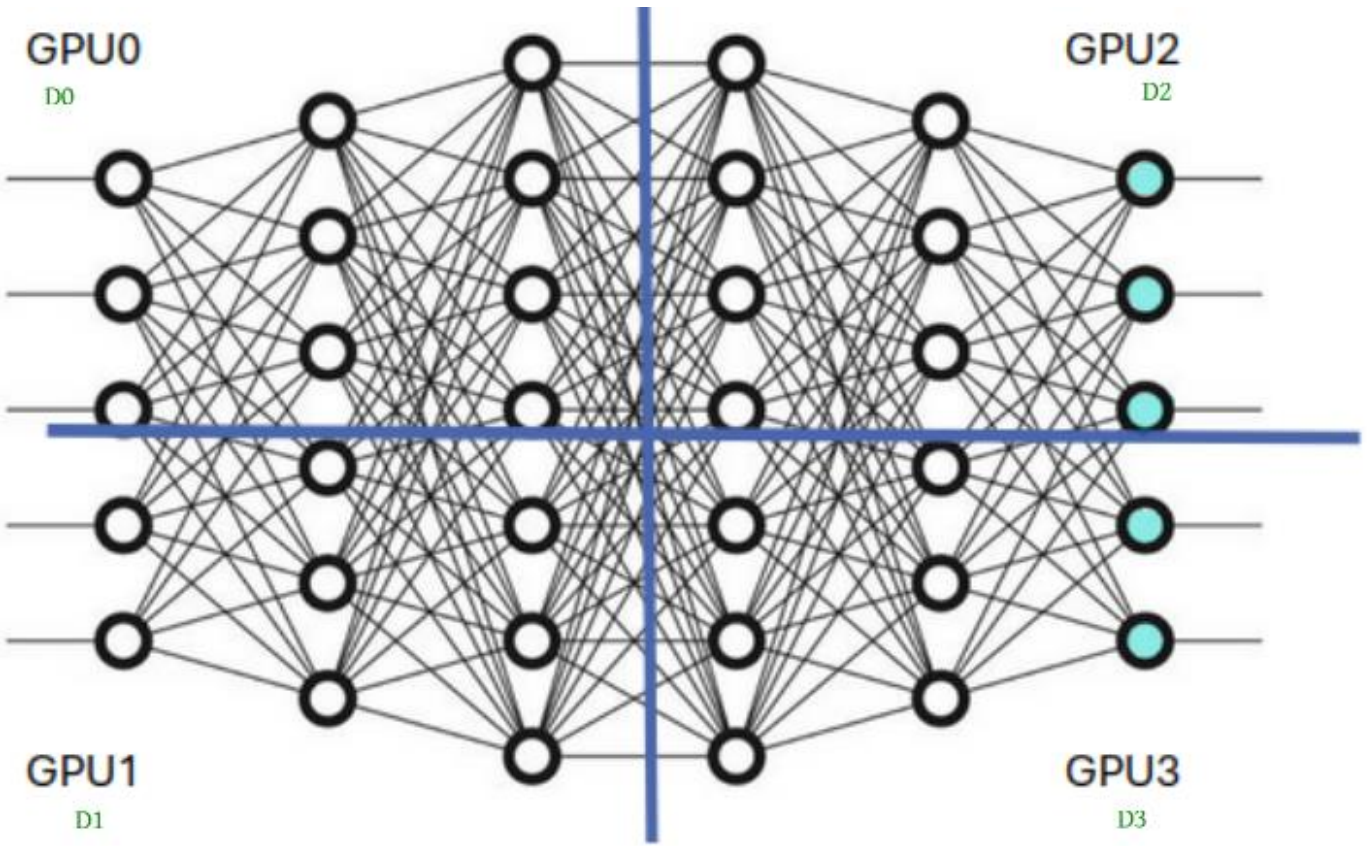
HOW TO SPLIT A MODEL WITH FSDP

AllGather Layer N (LN):

GPU0: LN(...(L0(D0))
GPU1: LN(...(L0(D1))
GPU2: LN(...(L0(D2))
GPU3: LN(...(L0(D3))

All Gather + Reduce Scatter:

$$\left.\begin{array}{ll} \text{GPU0:} & \dfrac{\partial \mathcal{L}(D_0)}{\partial W_{LN}} \\[2ex] \text{GPU1:} & \dfrac{\partial \mathcal{L}(D_1)}{\partial W_{LN}} \\[2ex] \text{GPU2:} & \dfrac{\partial \mathcal{L}(D_2)}{\partial W_{LN}} \\[2ex] \text{GPU3:} & \dfrac{\partial \mathcal{L}(D_3)}{\partial W_{LN}} \end{array}\right] \rightarrow \sum_{i=0}^{3} \frac{\partial \mathcal{L}(D_i)}{\partial W_{LN}}$$

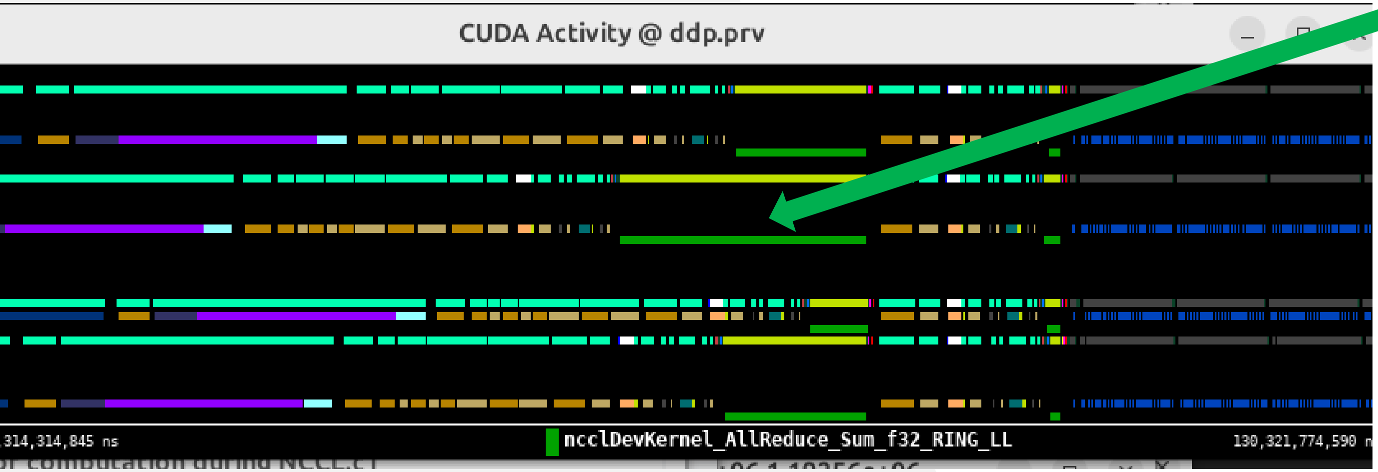


FSDP

DDP vs FSDP traces

DDP

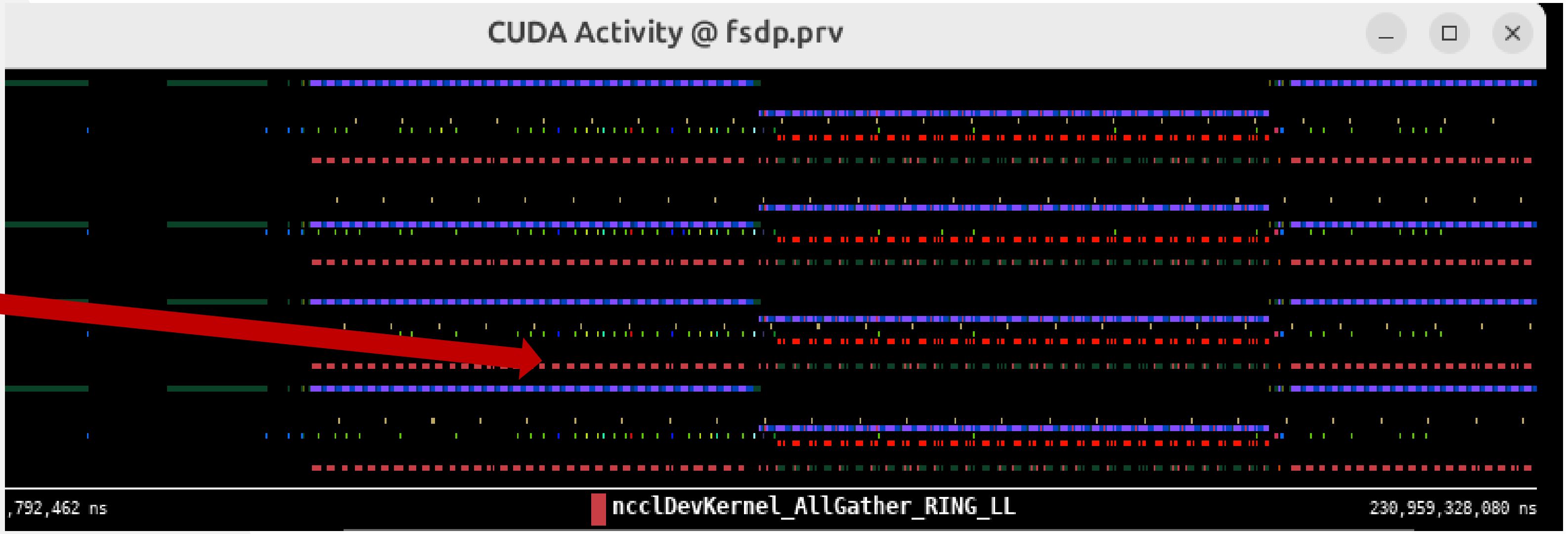
Bwd: All_reduce



FSDP

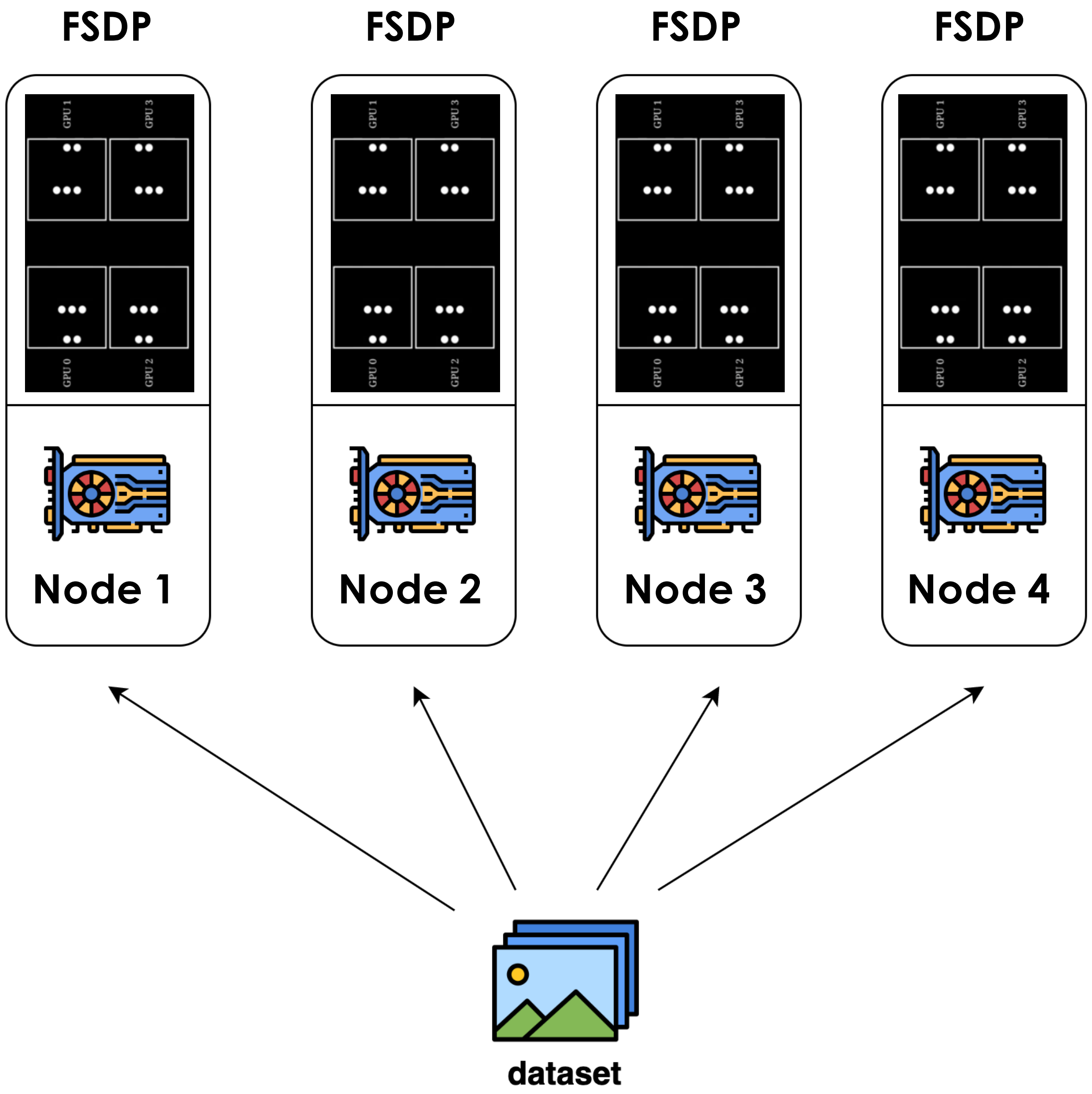
Fwd: All_Gather

Bwd: All_Gather + Reduce Scatter



Hybrid Sharded Data Parallel (HSDP)

- **HYBRID SHARD**: Apply FULL SHARD **within one node**, and **replicate** parameters across nodes.



Pro:

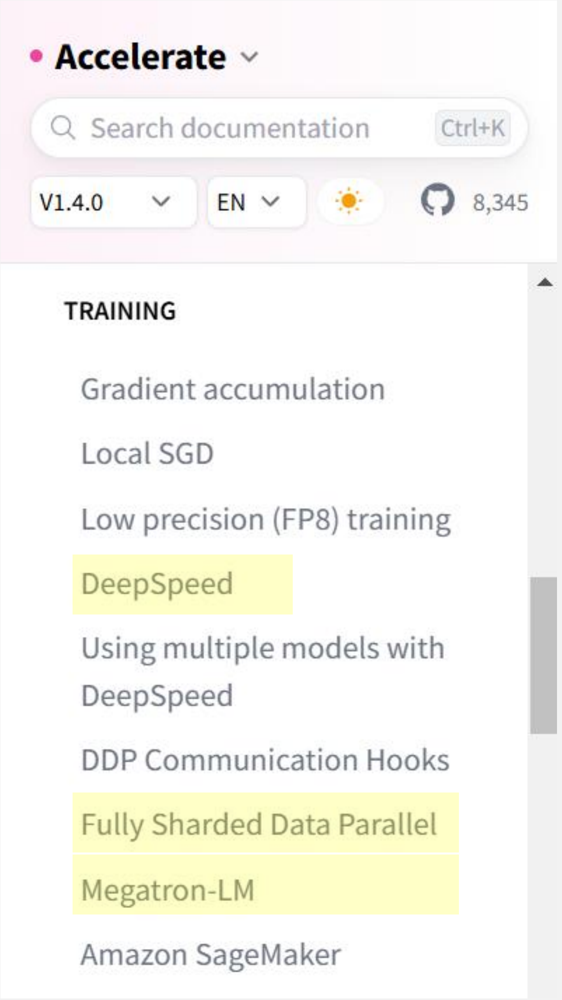
- **Reduced** communication volume

Cons:

- **More** GPU memory needed w.r.t. FSDP

Huggingface Accelerate

ACCELERATE by Hugging Face



Accelerate is an HF library that enables running the same PyTorch code across any distributed configuration by adding just few lines of code.

"In short, training and inference at scale made simple, efficient and adaptable."

https://huggingface.co/docs/accelerate/index

Accelerate config

- Distributed training across multiple nodes with a simple configuration files and few changes in the python script

- Several level of parallelism: data parallelism, model sharding, hybrid sharding. Handles both multi CPUs and multi GPUs using
Only a config file!

On your machine just run **accelerate config** and answer the questions asked.
This will generate automatically a config file.

ACCELERATE

Example of **config.yaml** files

Activate the python env:

`source /leonardo_work/tra26_castiel2/rscheda0/my_env/bin/activate`

Create a .yaml config file:

`accelerate config`

```yaml
compute_environment: LOCAL_MACHINE
debug: false
distributed_type: FSDP
downcast_bf16: 'no'
enable_cpu_affinity: false
fsdp_config:
  fsdp_auto_wrap_policy: TRANSFORMER_BASED_WRAP
  fsdp_backward_prefetch: BACKWARD_PRE
  fsdp_cpu_ram_efficient_loading: false
  fsdp_forward_prefetch: true
  fsdp_offload_params: false
  fsdp_sharding_strategy: FULL_SHARD
  fsdp_state_dict_type: SHARDED_STATE_DICT
  fsdp_sync_module_states: false
  fsdp_use_orig_params: true
machine_rank: 0
main_training_function: main
mixed_precision: bf16
num_machines: 1
num_processes: 4
rdzv_backend: static
same_network: true
tpu_env: []
tpu_use_cluster: false
tpu_use_sudo: false
use_cpu: false
```

ACCELERATE

HARDWARE SELECTION

CPU: Whether or not to force the training on the CPU.
GPU: Whether or not this should launch a distributed GPU training
TPU: Whether or not this should launch a TPU training
IPEX: Whether or not this should launch an Intel Pytorch Extension (IPEX) training.

RESOURCES SELECTION

MIXED PRECISION: {no,fp16,bf16,fp8} (str) — Whether or not to use mixed precision training. Choose between FP16 and BF16 (bfloat16) training. BF16 training is only supported on Nvidia Ampere GPUs and PyTorch 1.10 or later.
NUM_PROCESSES: (int) — Total number of processes to be launched in parallel.
NUM_MACHINES: (int) Total number of nodes used
NUM_CPUS_THREADS_PER_PROCESS: (int) **-** Number of CPUs threads per process.

[1] https://huggingface.co/docs/accelerate/package_reference/cli

ACCELERATE

TYPE OF PARALLELISM

USE_DEEPSPEED: Whether or not to use DeepSpeed for training.
USE_FSDP: Whether or not to use FSDP for training
USE_MEGATRON_LM: Whether or not to use Megatron-LM for training

FSDP SHARDING STRATEGY

FULL_SHARD (ZeRO Stage-3)
SHARD_GRAD_OP (ZeRO Stage-2)
NO_SHARD (ZeRO Stage-0. Nosharding)

[1] https://huggingface.co/docs/accelerate/package_reference/cli

ACCELERATE

To launch the training with the new Accelerate configuration performing distributed training it will be enough to run

```
accelerate launch --config_file config_accelerate.yaml my_script.py <py_args>
```

In SLURM it becomes

```
accelerate launch \
    --main_process_ip "$MASTER_ADDR" \
    --main_process_port $MASTER_PORT \
    --machine_rank $SLURM_PROCID \
    --rdzv_backend c10d \
    --config_file config_accelerate.yaml
    my_script.py <py_args>
```

ACCELERATE – LLama 3.1 Fine-Tuning

How to launch

```
LAUNCHER="accelerate launch \
   --main_process_ip "$MASTER_ADDR" \
    --main_process_port $MASTER_PORT \
    --machine_rank $SLURM_PROCID \
    --rdzv_backend c10d \
"
echo LAUNCHER=$LAUNCHER

module load cuda/12.1

source env/bin/activate

#srun $LAUNCHER peft_finetuning_python2.py config_accelerate.yaml --sample_packing False
srun $LAUNCHER --config_file config_accelerate.yaml peft_finetuning_python2.py
```

To run the job:
`sbatch job.sh`

Results

NUM NODES	NUM GPUS	TRAINING TIME	SPEED UP
1	1	3670	1
1	2	1821 s	2.01

ACCELERATE – EXERCISE

- **pull the repository!!** `git pull`

  Go to the location:
  `cd castiel-multi-gpu-ai/content/it/accelerate_fsdp`

- Run the job.sh on 4,8 gpus, adjusting the also the number of nodes. **Keep in mind that you need to change also the configuration file!**

- Look at the GPU %. Is there any difference with 4,8 gpus? Is the code using all the GPUs

- Compute the speedup

Results

NUM NODES	NUM GPUS	TRAINING TIME	SPEED UP
1	1	3670	1
1	2	1821 s	2.01
1	4	?	?
2	8	?	?

Grazie