DDP & AllReduce

Distributed Data Parallel

2025

Распределенный блок

- GPU clusters
- **DDP & AllReduce <——— you are here**
- FSDP
- TP & CP & EP & PP …

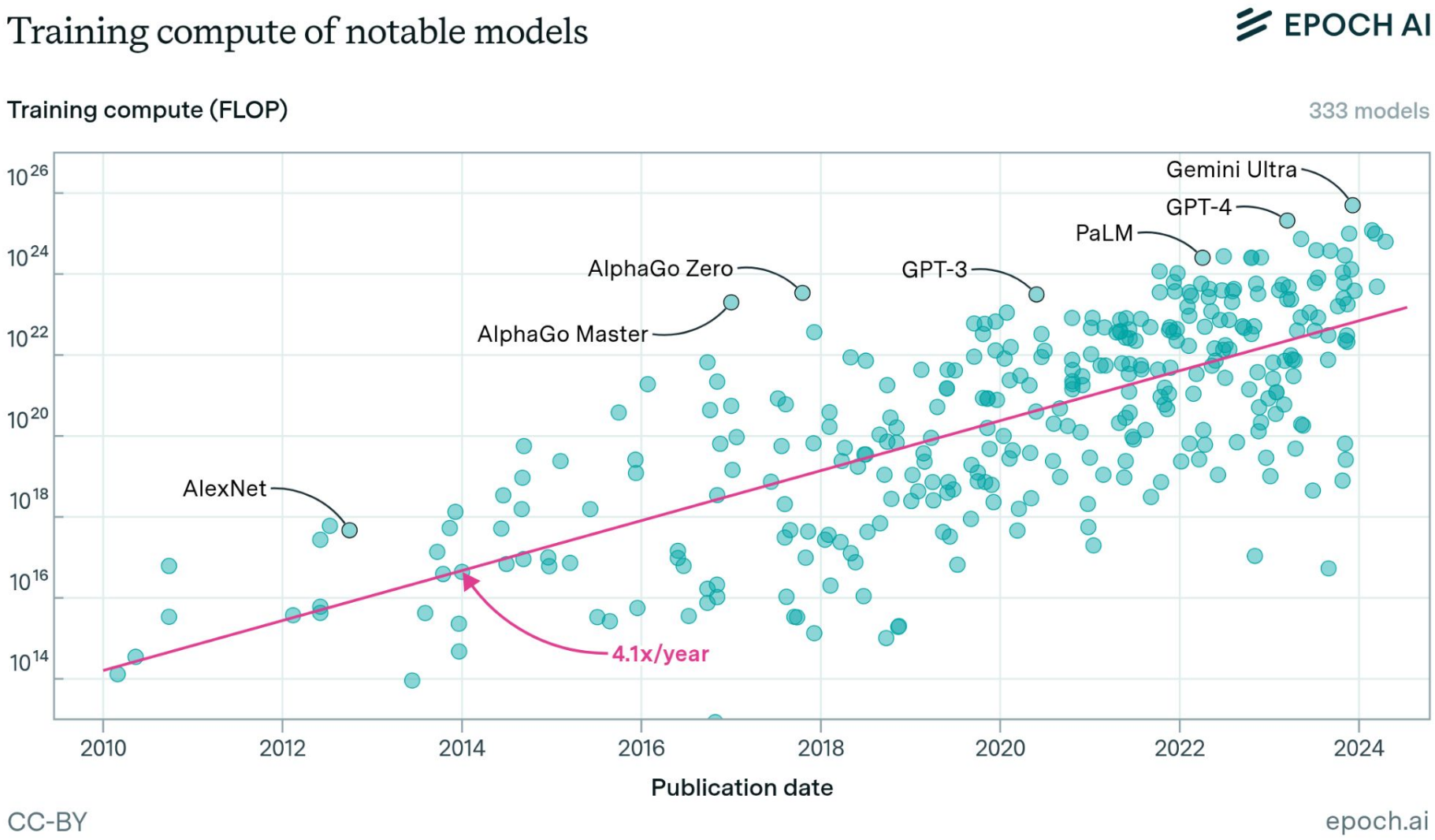
Сегодняшняя лекция

- Distributed Training
- Data Parallel
- AllReduce
- Overlapping with optim step
- Unsynced GD

Сегодняшний семинар

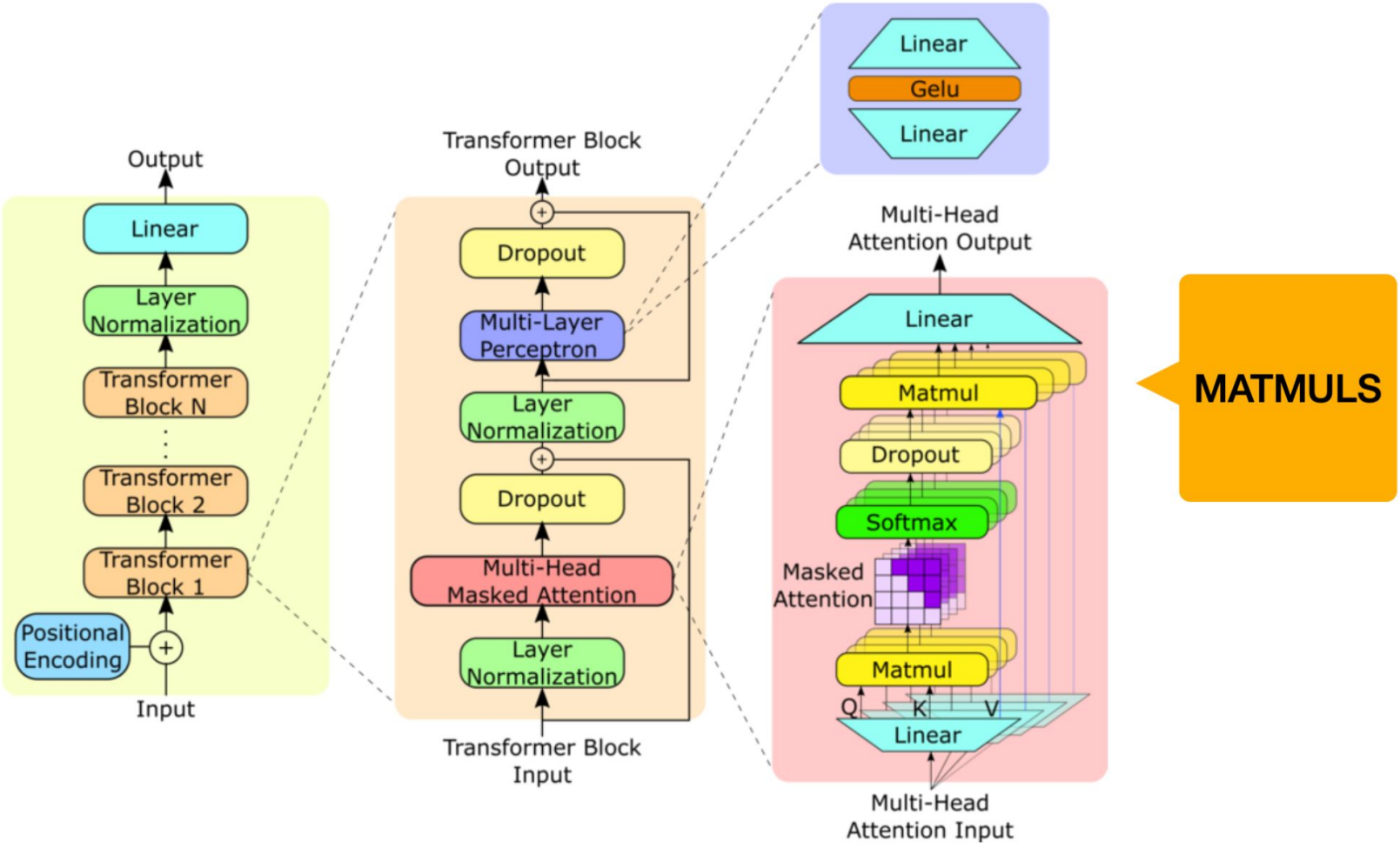
- nccl tests
- ddp training
- optimizer
- memory + perf profiles

Distributed Training



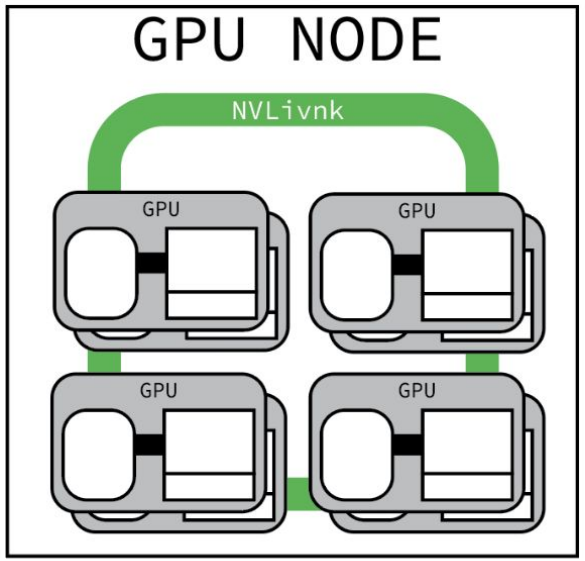
Source: https://epoch.ai/blog/training-compute-of-frontier-ai-models-grows-by-4-5x-per-year

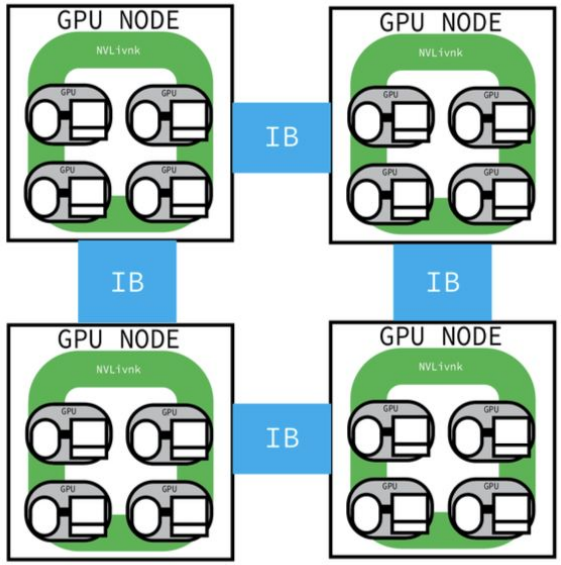
Recap: transformer



Recap: GPU clusters

- GPU объединяются в сервера a.k.a “ноды” с быстрой сетью
- Ноды объединяются в кластера





Source: https://arxiv.org/pdf/2407.20018v1

Parallelisms

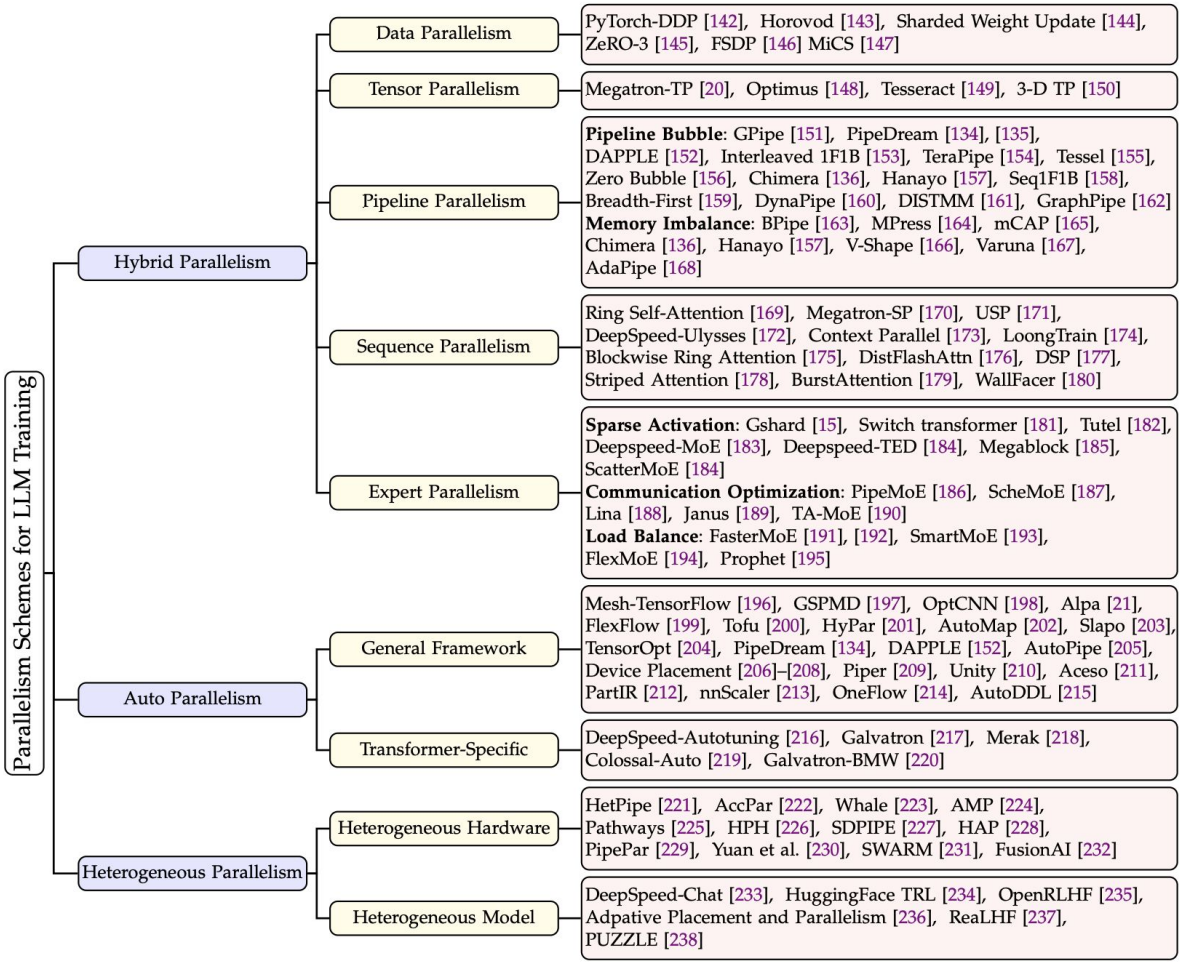
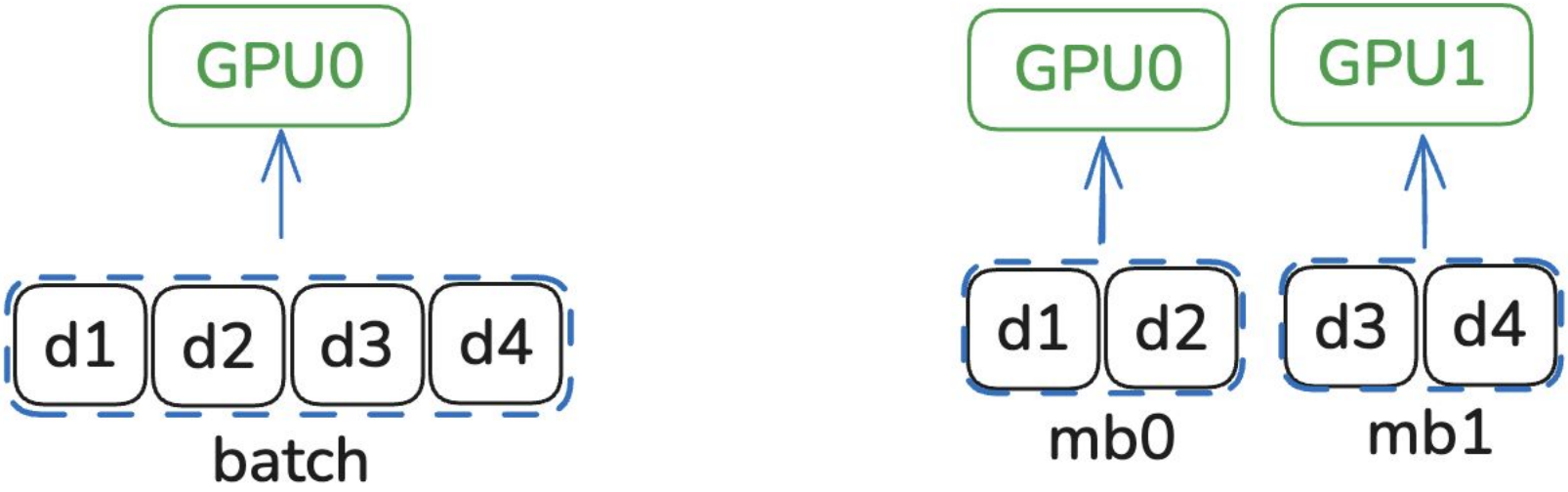


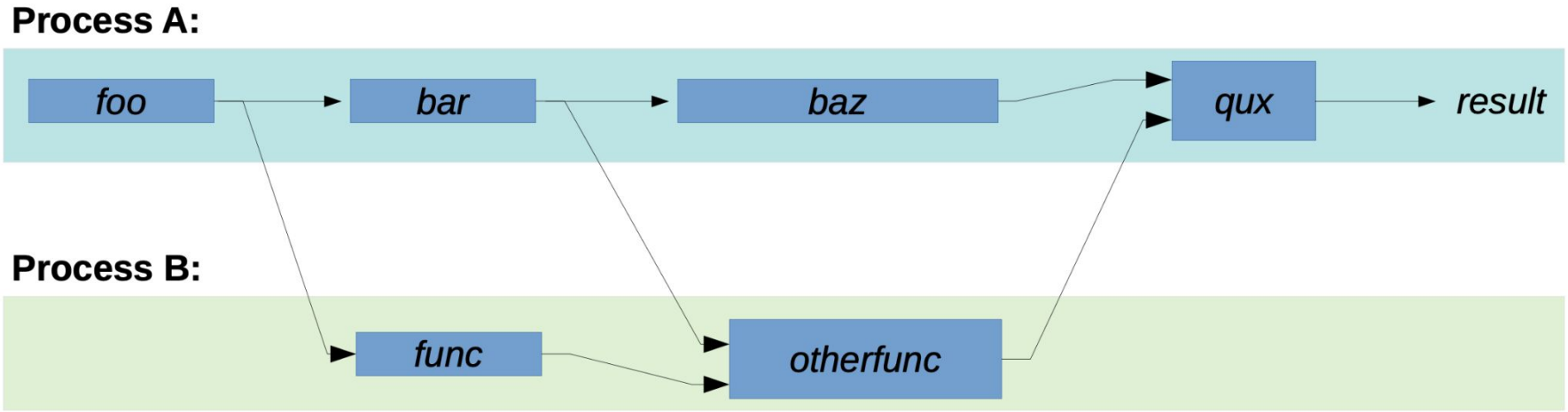
Fig. 7: Studies on parallelism schemes for distributed LLM training.

Source: https://arxiv.org/pdf/2407.20018v1



Data Parallel

Channel / Pipe



Channel (pipe):

- Communication in O(message size)
- Asynchronous read/write

Source: https://github.com/mryab/efficient-dl-systems/tree/main/week04_data_parallel

*Слайды EffDL

A meme stolen from somewhere

AllReduce NCCL

AllReduce

torch.distributed.all_reduce(*tensor, op=<RedOpType.SUM: 0>, group=None, async_op=False*) [source]

Reduces the tensor data across all machines in a way that all get the final result.

After the call `tensor` is going to be bitwise identical in all processes.

Complex tensors are supported.

Parameters:

- **tensor** (*Tensor*) – Input and output of the collective. The function operates in-place.
- **op** (*optional*) – One of the values from `torch.distributed.ReduceOp` enum. Specifies an operation used for element-wise reductions.
- **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

Source: https://docs.pytorch.org/docs/stable/distributed.html#torch.distributed.all_reduce

AllReduce

Examples

```
>>> # All tensors below are of torch.int64 type.
>>> # We have 2 process groups, 2 ranks.
>>> device = torch.device(f"cuda:{rank}")
>>> tensor = torch.arange(2, dtype=torch.int64, device=device) + 1 + 2 * rank
>>> tensor
tensor([1, 2], device='cuda:0') # Rank 0
tensor([3, 4], device='cuda:1') # Rank 1
>>> dist.all_reduce(tensor, op=ReduceOp.SUM)
>>> tensor
tensor([4, 6], device='cuda:0') # Rank 0
tensor([4, 6], device='cuda:1') # Rank 1
```

```
>>> # All tensors below are of torch.cfloat type.
>>> # We have 2 process groups, 2 ranks.
>>> tensor = torch.tensor(
...     [1 + 1j, 2 + 2j], dtype=torch.cfloat, device=device
... ) + 2 * rank * (1 + 1j)
>>> tensor
tensor([1.+1.j, 2.+2.j], device='cuda:0') # Rank 0
tensor([3.+3.j, 4.+4.j], device='cuda:1') # Rank 1
>>> dist.all_reduce(tensor, op=ReduceOp.SUM)
>>> tensor
tensor([4.+4.j, 6.+6.j], device='cuda:0') # Rank 0
tensor([4.+4.j, 6.+6.j], device='cuda:1') # Rank 1
```

Source: https://docs.pytorch.org/docs/stable/distributed.html#torch.distributed.all_reduce

AllReduce

TABLE V
STEPS IN ONE LOOP ITERATION OF NCCL RING ALLREDUCE

Step Index	NCCL Primitive
0	`send`
1 to $k-2$	`recvReduceSend`
$k-1$	`recvReduceCopySend`
k to $2k-3$	`recvCopySend`
$2k-2$	`recv`

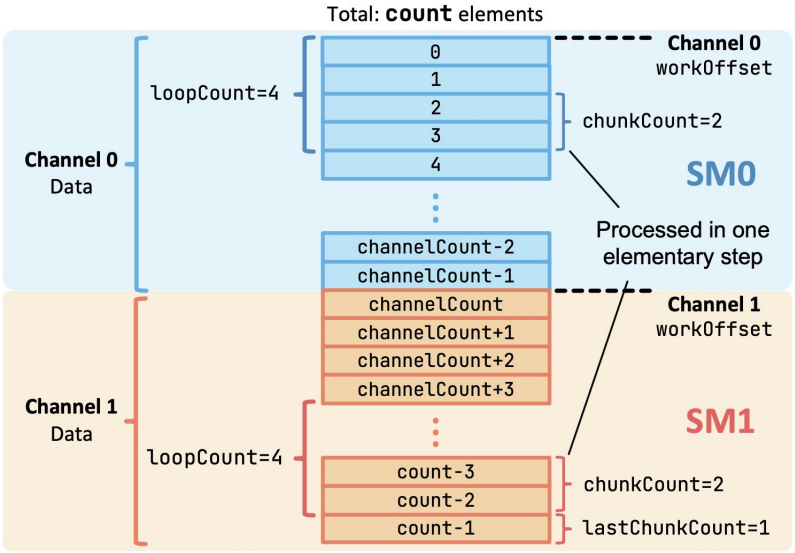
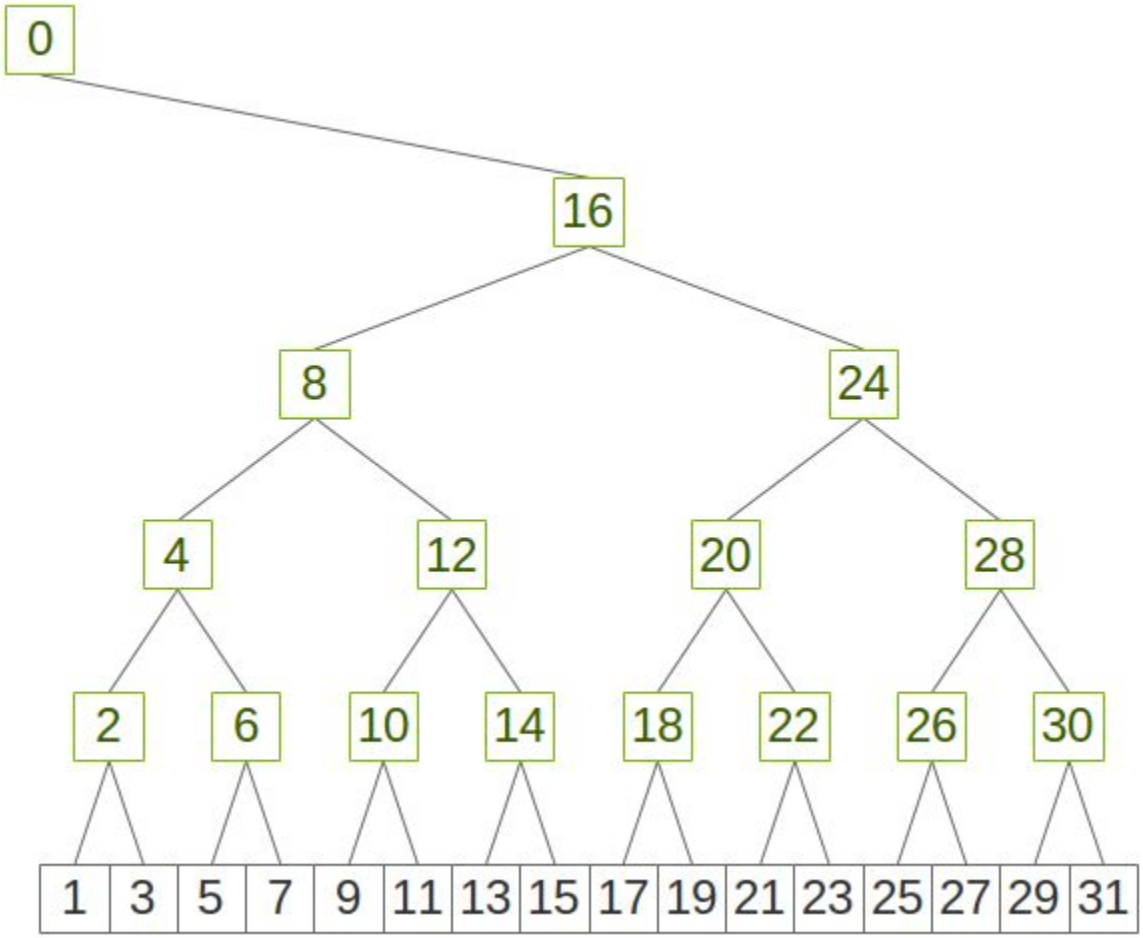


Fig. 3. Visualization of NCCL's data partitioning strategy across communication channels and loop iterations.

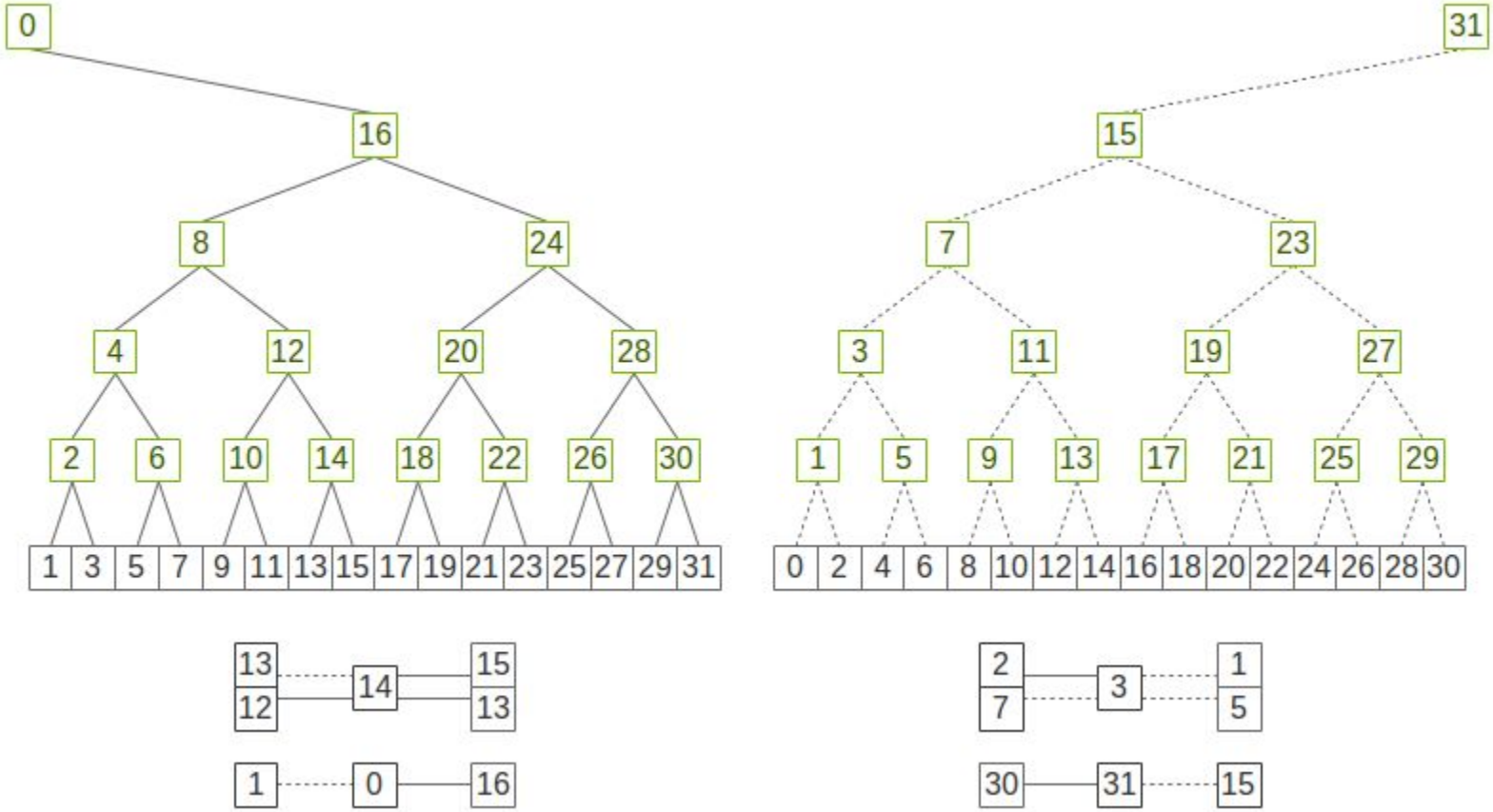
Source: https://arxiv.org/pdf/2507.04786v1

Tree AllReduce



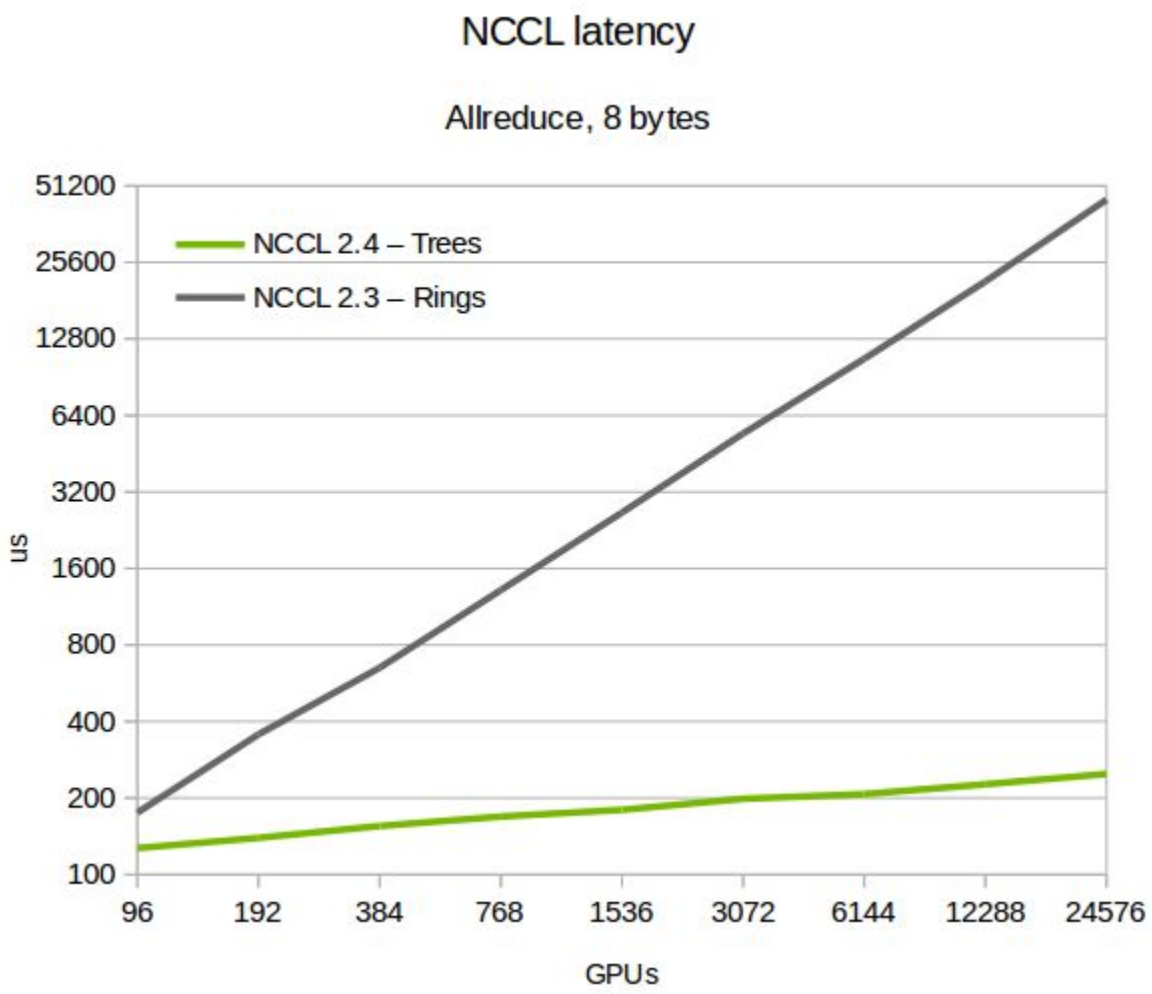
Source: https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4

Tree AllReduce



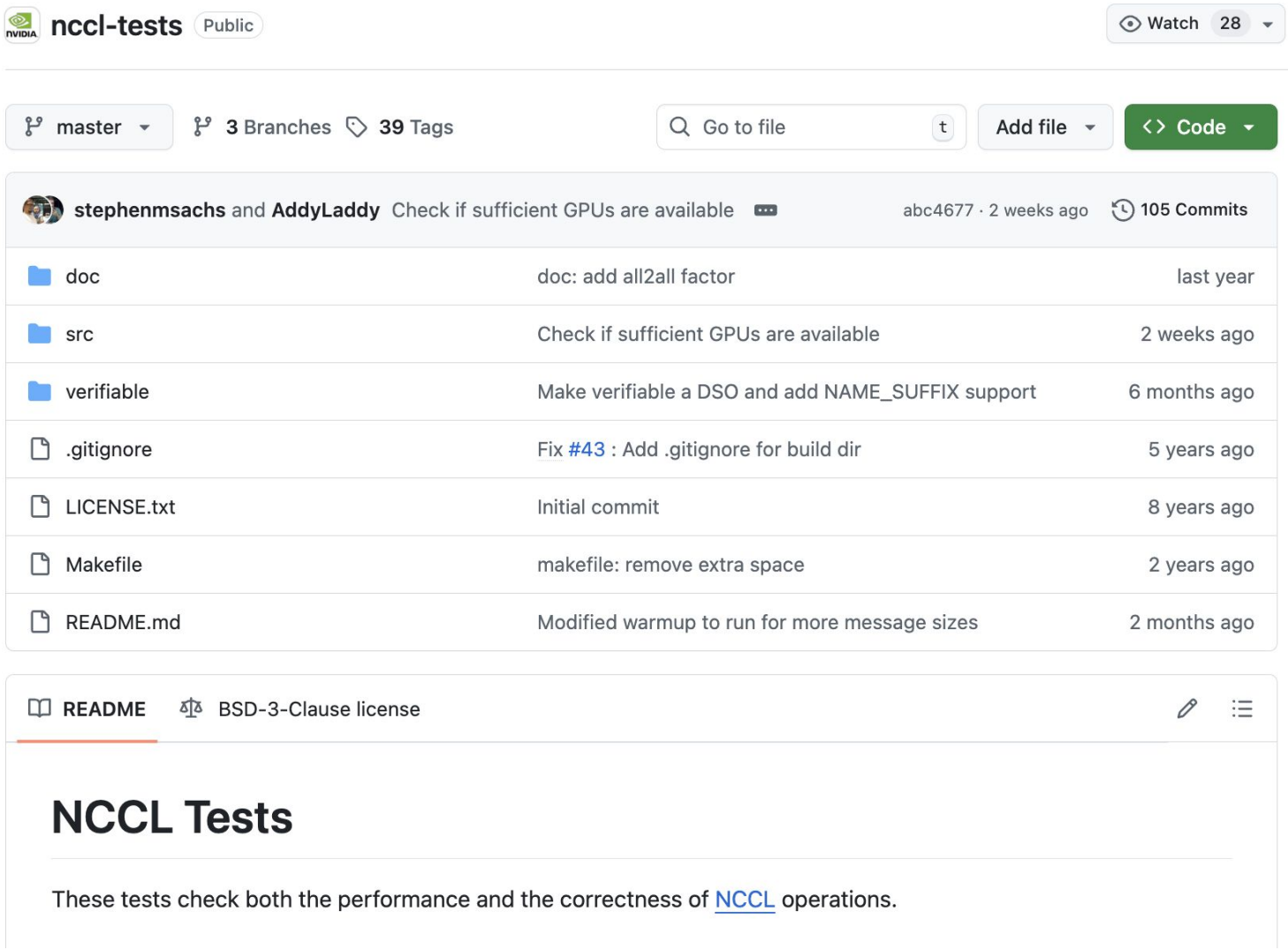
Source: https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4

Ring vs Tree AllReduce



Source: https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4

NCCL tests



nccl-tests Public

Watch 28

master · 3 Branches · 39 Tags · Go to file t · Add file · Code

stephenmsachs and AddyLaddy Check if sufficient GPUs are available · abc4677 · 2 weeks ago · 105 Commits

doc	doc: add all2all factor	last year
src	Check if sufficient GPUs are available	2 weeks ago
verifiable	Make verifiable a DSO and add NAME_SUFFIX support	6 months ago
.gitignore	Fix #43 : Add .gitignore for build dir	5 years ago
LICENSE.txt	Initial commit	8 years ago
Makefile	makefile: remove extra space	2 years ago
README.md	Modified warmup to run for more message sizes	2 months ago

README · BSD-3-Clause license

NCCL Tests

These tests check both the performance and the correctness of NCCL operations.

Source: https://github.com/NVIDIA/nccl-tests

NCCL benchmarking

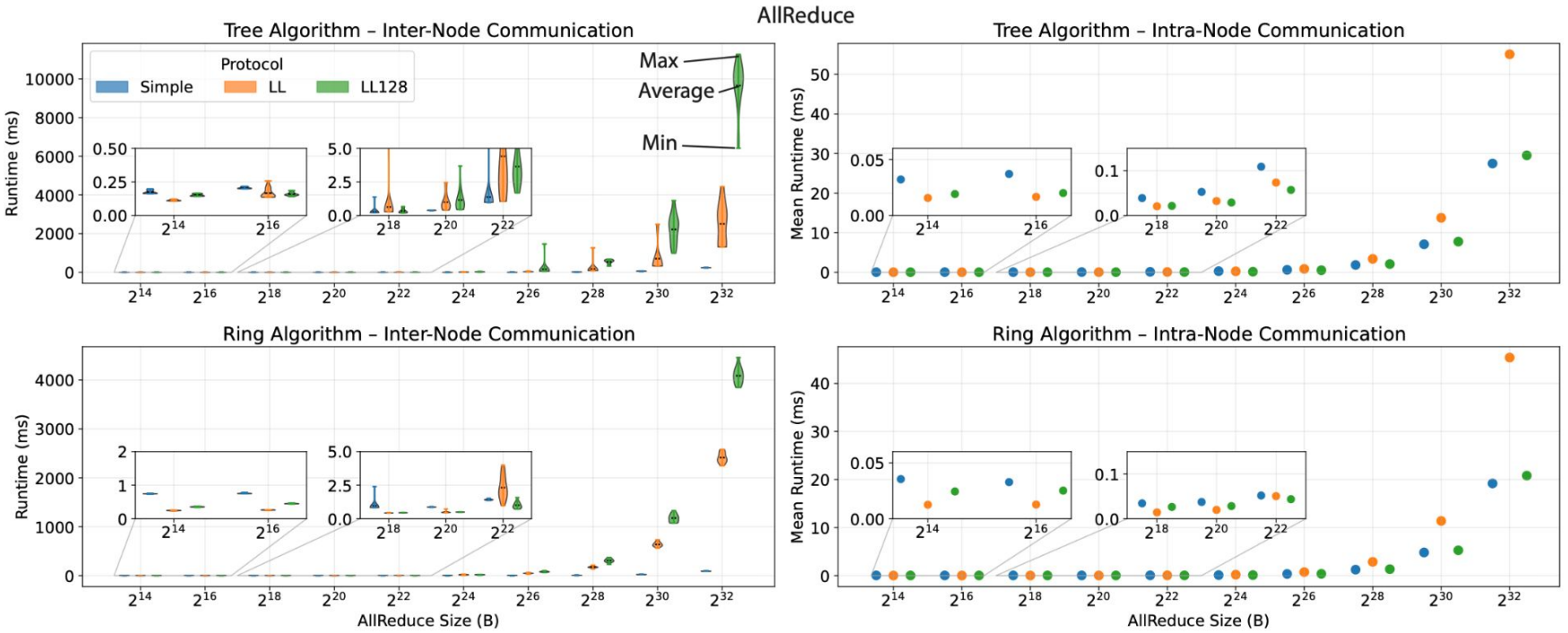


Fig. 6. Runtime comparison of protocols for Ring and Tree AllReduce when running inter- and intra-node. Each data point consists of 20 runs with a warm-up phase. For intra-node communication we report only the median value for readability as the variance is very low.

Source: https://arxiv.org/html/2507.04786v1

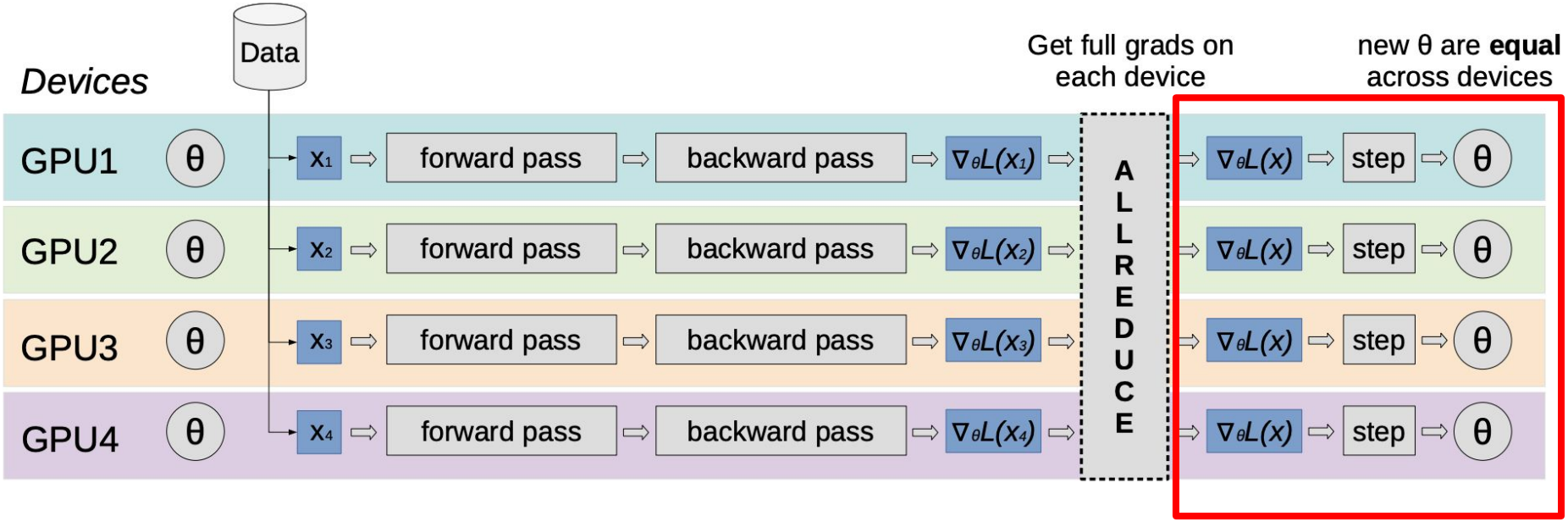
AllReduce calculations

```
(main) root@C.27273295:/workspace/nccl-tests$ ./build/all_reduce_perf  -g 2
# nccl-tests version 2.17.4 nccl-headers=21701 nccl-library=21701
# Collective test starting: all_reduce_perf
# nThread 1 nGpus 2 minBytes 33554432 maxBytes 33554432 step: 1048576(bytes) warmup iters: 1 iters: 20 agg iters: 1
validation: 1 graph: 0
#
# Using devices
#  Rank  0 Group  0 Pid  12457 on bd14101a12c9 device  0 [0000:41:00] Tesla V100-PCIE-16GB
#  Rank  1 Group  0 Pid  12457 on bd14101a12c9 device  1 [0000:61:00] Tesla V100-PCIE-16GB
#
#                                                              out-of-place                       in-place
#       size         count      type   redop    root     time   algbw   busbw  #wrong     time   algbw   busbw  #wro
ng
#        (B)    (elements)                               (us)  (GB/s)  (GB/s)            (us)  (GB/s)  (GB/s)
    33554432       8388608     float     sum      -1  2943.25   11.40   11.40       0  2953.54   11.36   11.36
 0
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 11.3806
#
# Collective test concluded: all_reduce_perf

(main) root@C.27273295:/workspace/nccl-tests$
```

2xV100, PCIe, 11.36 GB/s

Optimizers


Devices
Data
Get full grads on each device
new θ are **equal** across devices
GPU1
GPU2
GPU3
GPU4
θ
x_1
x_2
x_3
x_4
forward pass
backward pass
$\nabla_\theta L(x_1)$
$\nabla_\theta L(x_2)$
$\nabla_\theta L(x_3)$
$\nabla_\theta L(x_4)$
ALLREDUCE
$\nabla_\theta L(x)$
step

Optimizers

$$v_{k+1} = \beta_1 v_k + (1 - \beta_1)\nabla f(x_k)$$
$$G_{k+1} = \beta_2 G_k + (1 - \beta_2)(\nabla f(x_k))^2$$
$$x_{k+1} = x_k - \frac{\alpha}{\sqrt{G_{k+1} + \varepsilon}} v_{k+1}.$$

Adam Optimizer

Optimizers

$$v_{k+1} = \beta_1 v_k + (1 - \beta_1) \nabla f(x_k)$$

$$G_{k+1} = \beta_2 G_k + (1 - \beta_2)(\nabla f(x_k))^2$$

$$x_{k+1} = x_k - \frac{\alpha}{\sqrt{G_{k+1} + \varepsilon}} v_{k+1}.$$

Adam Optimizer

Optimizers

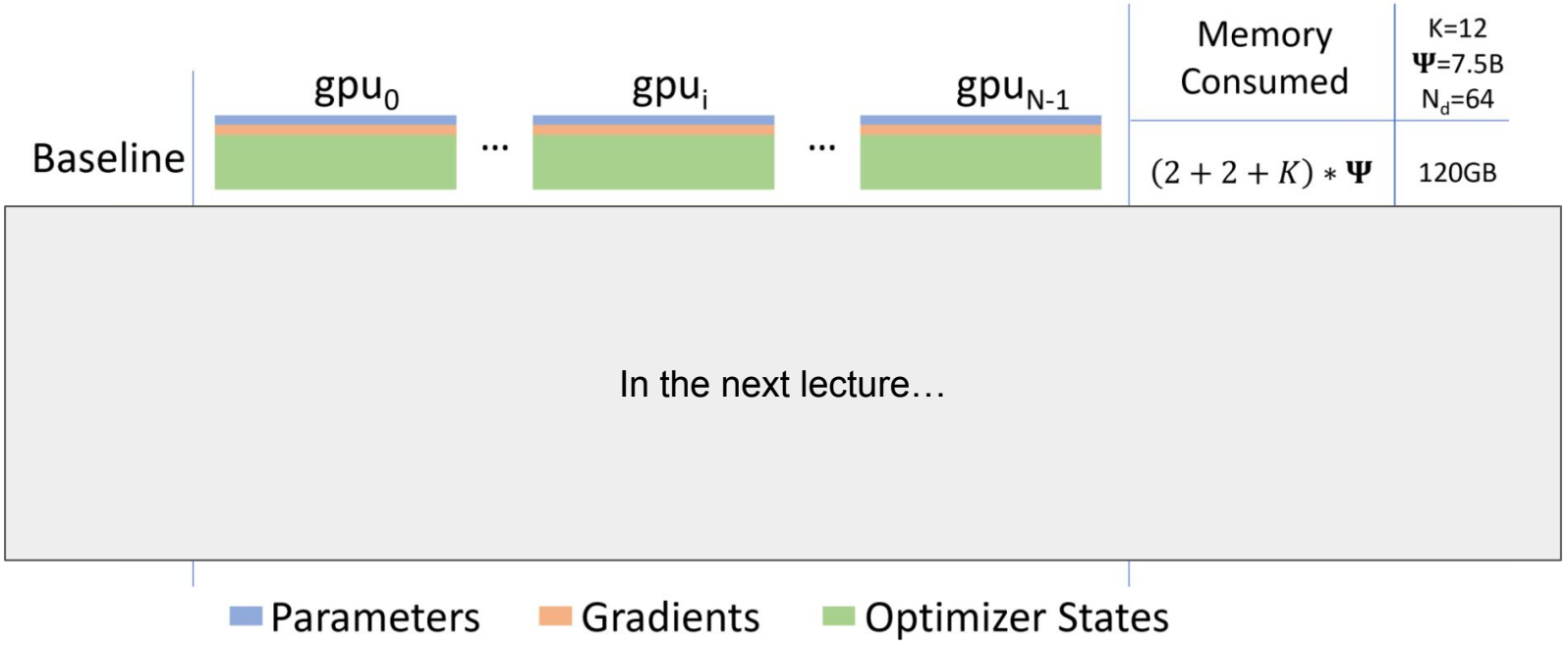
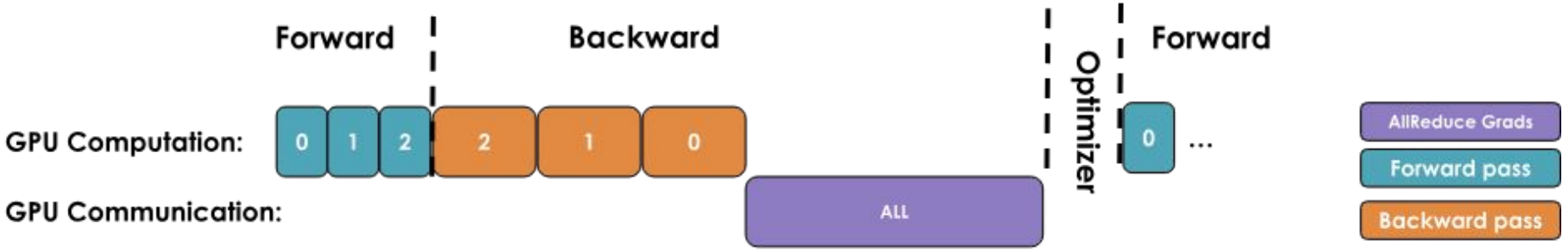


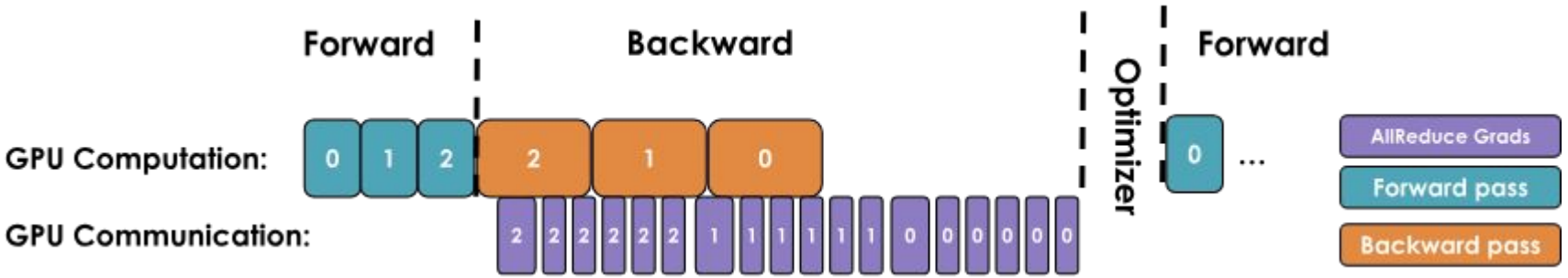
Figure 1: Comparing the per-device memory consumption of model states, with three stages of *ZeRO*-DP optimizations. Ψ denotes model size (number of parameters), K denotes the memory multiplier of optimizer states, and N_d denotes DP degree. In the example, we assume a model size of $\Psi = 7.5B$ and DP of $N_d = 64$ with $K = 12$ based on mixed-precision training with Adam optimizer.

AllReduce + Optimizer step



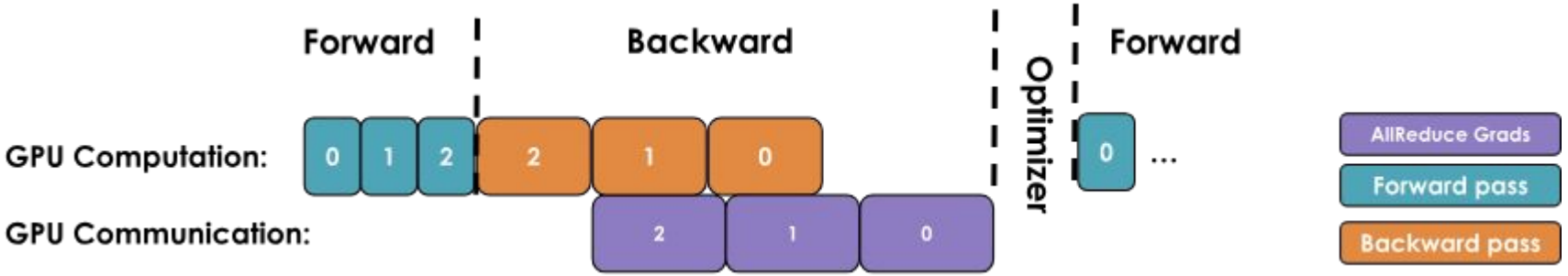
Source: https://huggingface.co/spaces/nanotron/ultrascale-playbook

AllReduce + Optimizer step



Source: https://huggingface.co/spaces/nanotron/ultrascale-playbook

AllReduce + Optimizer step



Source: https://huggingface.co/spaces/nanotron/ultrascale-playbook

AllReduce + Optimizer step

```python
class DataParallelNaive(nn.Module):
    def __init__(self, module):
        ...
        self.register_backward_hook(self._allreduce_grads)

    def forward(self, *inputs, **kwargs):
        return self.module(*inputs, **kwargs)

    def register_backward_hook(self, hook):
        for p in self.module.parameters():
            if p.requires_grad is True:
                p.register_hook(hook)

    def _allreduce_grads(self, grad):
        dist.all_reduce(grad, op=dist.ReduceOp.SUM, group=pgm.process_group_manager.cp_dp_group)
        grad /= pgm.process_group_manager.cp_dp_world_size
        return grad
```

Source: https://huggingface.co/spaces/nanotron/ultrascale-playbook

Pytorch DDP

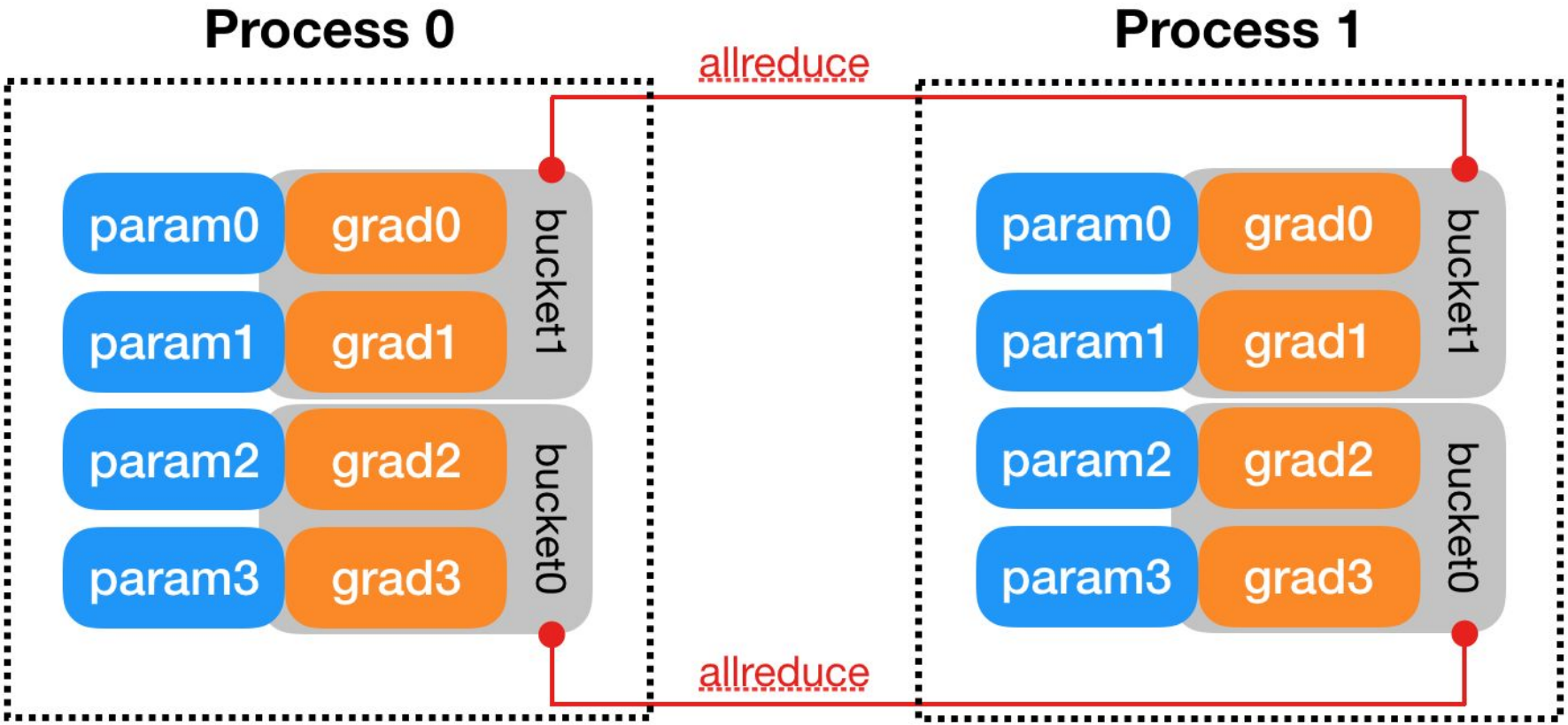
```
from torch.nn.parallel import DistributedDataParallel as DDP


def example(rank, world_size):
    # create default process group
    dist.init_process_group("gloo", rank=rank, world_size=world_size)
    # create local model
    model = nn.Linear(10, 10).to(rank)
    # construct DDP model
    ddp_model = DDP(model, device_ids=[rank])
    # define loss function and optimizer
    loss_fn = nn.MSELoss()
    optimizer = optim.SGD(ddp_model.parameters(), lr=0.001)

    # forward pass
    outputs = ddp_model(torch.randn(20, 10).to(rank))
    labels = torch.randn(20, 10).to(rank)
    # backward pass
    loss_fn(outputs, labels).backward()
    # update parameters
    optimizer.step()
```

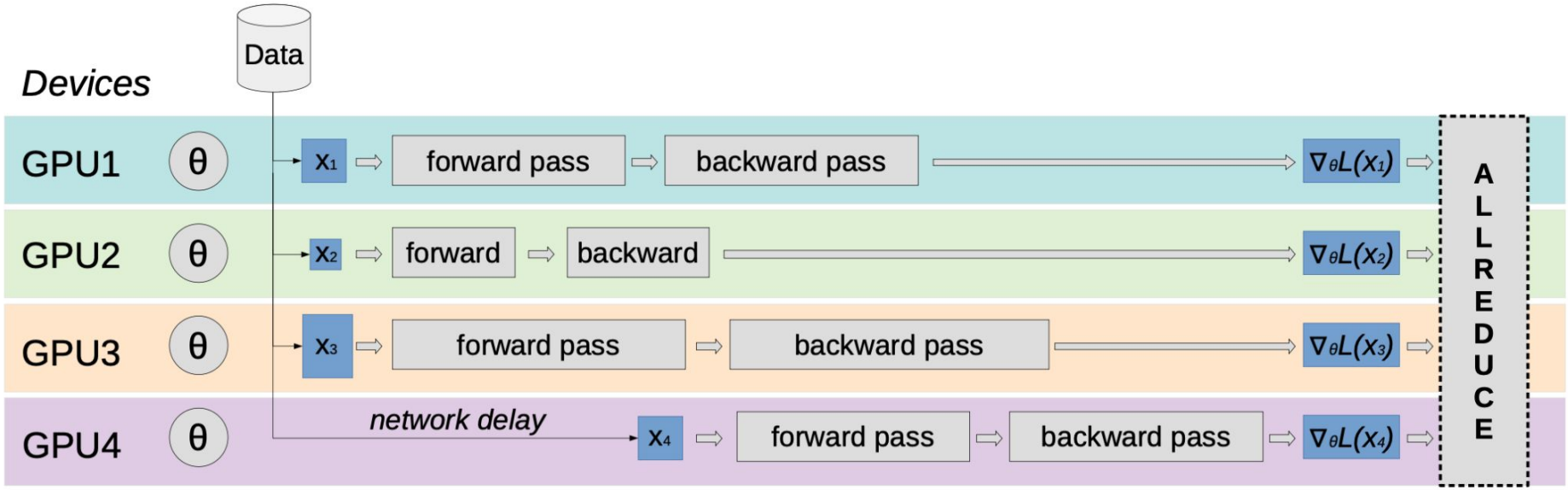
Source: https://docs.pytorch.org/docs/2.9/notes/ddp.html

Pytorch DDP



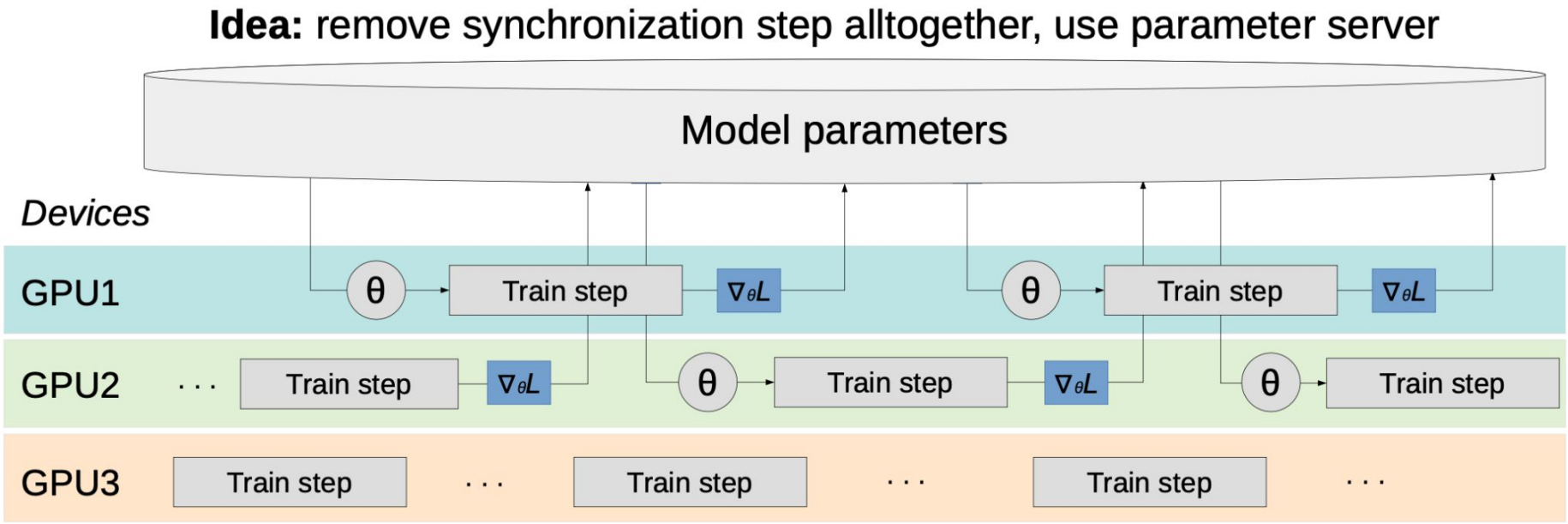
Source: https://docs.pytorch.org/docs/2.9/notes/ddp.html

Heterogeneity issue



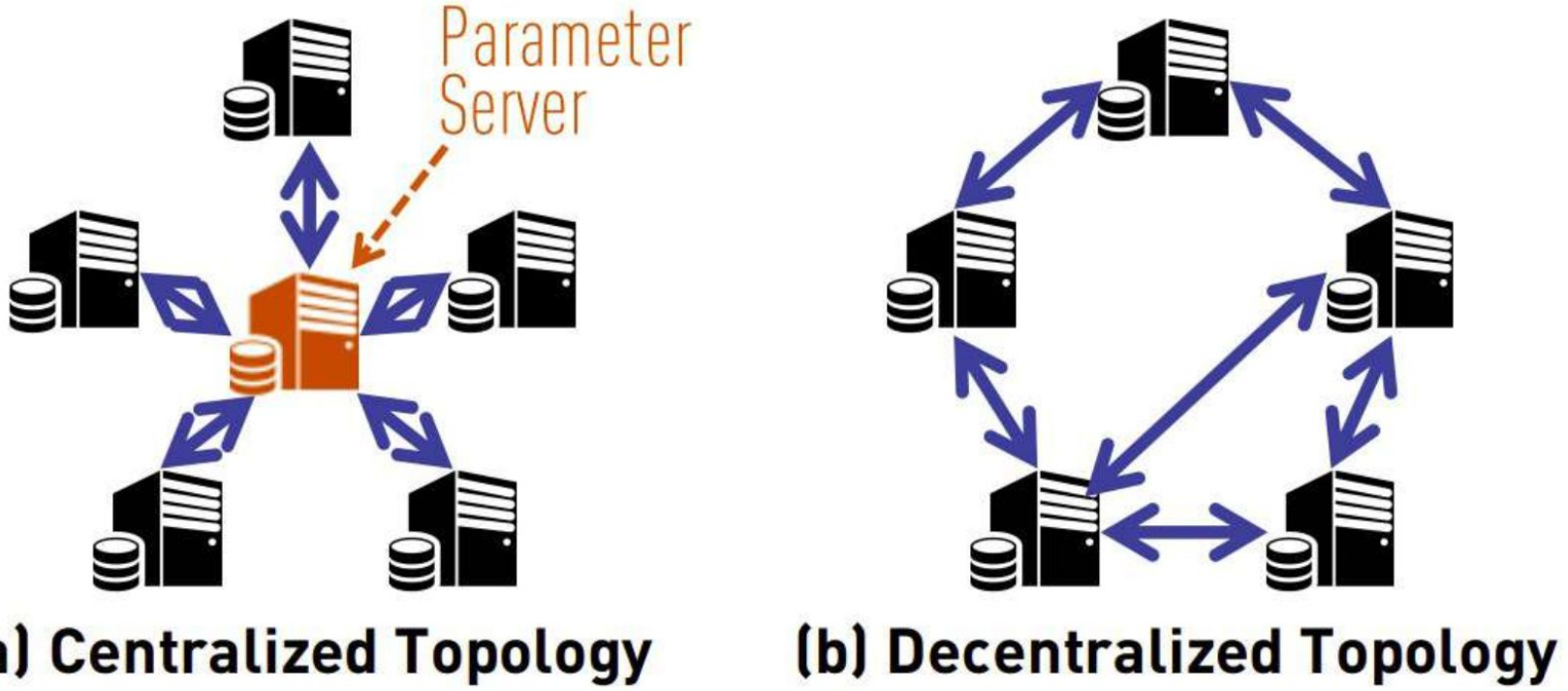
Source: https://github.com/mryab/efficient-dl-systems/tree/main/week04_data_parallel

Heterogeneity issue



Source: https://github.com/mryab/efficient-dl-systems/tree/main/week04_data_parallel

Decentralized Training with Gossip



Source: https://github.com/mryab/efficient-dl-systems/tree/main/week04_data_parallel

Q&A

Семинар

AllReduce calculations

```
(main) root@C.27273295:/workspace/nccl-tests$ ./build/all_reduce_perf  -g 2
# nccl-tests version 2.17.4 nccl-headers=21701 nccl-library=21701
# Collective test starting: all_reduce_perf
# nThread 1 nGpus 2 minBytes 33554432 maxBytes 33554432 step: 1048576(bytes) warmup iters: 1 iters: 20 agg iters: 1
validation: 1 graph: 0
#
# Using devices
#  Rank  0 Group  0 Pid  12457 on bd14101a12c9 device  0 [0000:41:00] Tesla V100-PCIE-16GB
#  Rank  1 Group  0 Pid  12457 on bd14101a12c9 device  1 [0000:61:00] Tesla V100-PCIE-16GB
#
#                                                              out-of-place                       in-place
#       size         count      type   redop    root     time   algbw   busbw  #wrong     time   algbw   busbw  #wro
ng
#        (B)    (elements)                               (us)  (GB/s)  (GB/s)            (us)  (GB/s)  (GB/s)
    33554432       8388608     float     sum      -1  2943.25   11.40   11.40       0  2953.54   11.36   11.36
 0
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 11.3806
#
# Collective test concluded: all_reduce_perf

(main) root@C.27273295:/workspace/nccl-tests$
```

2xV100, PCIe, 11.36 GB/s