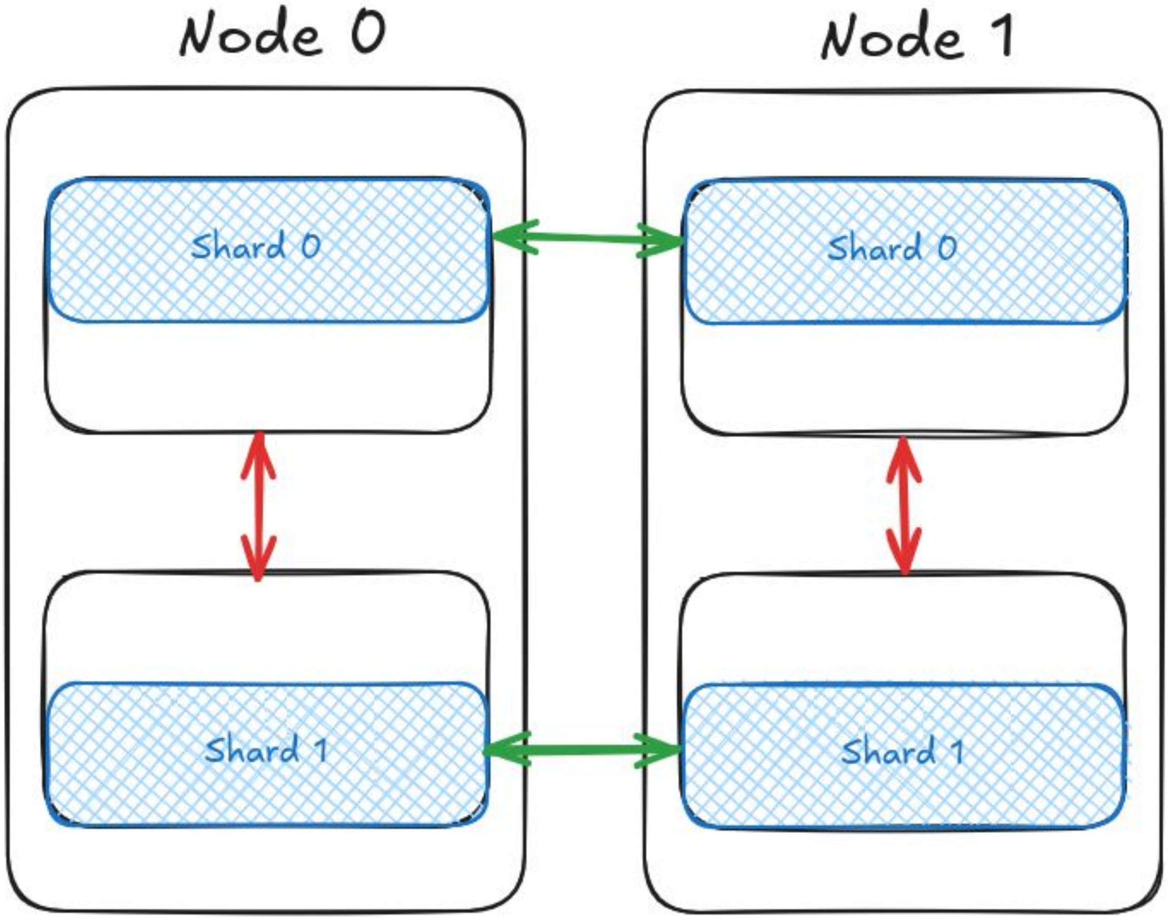
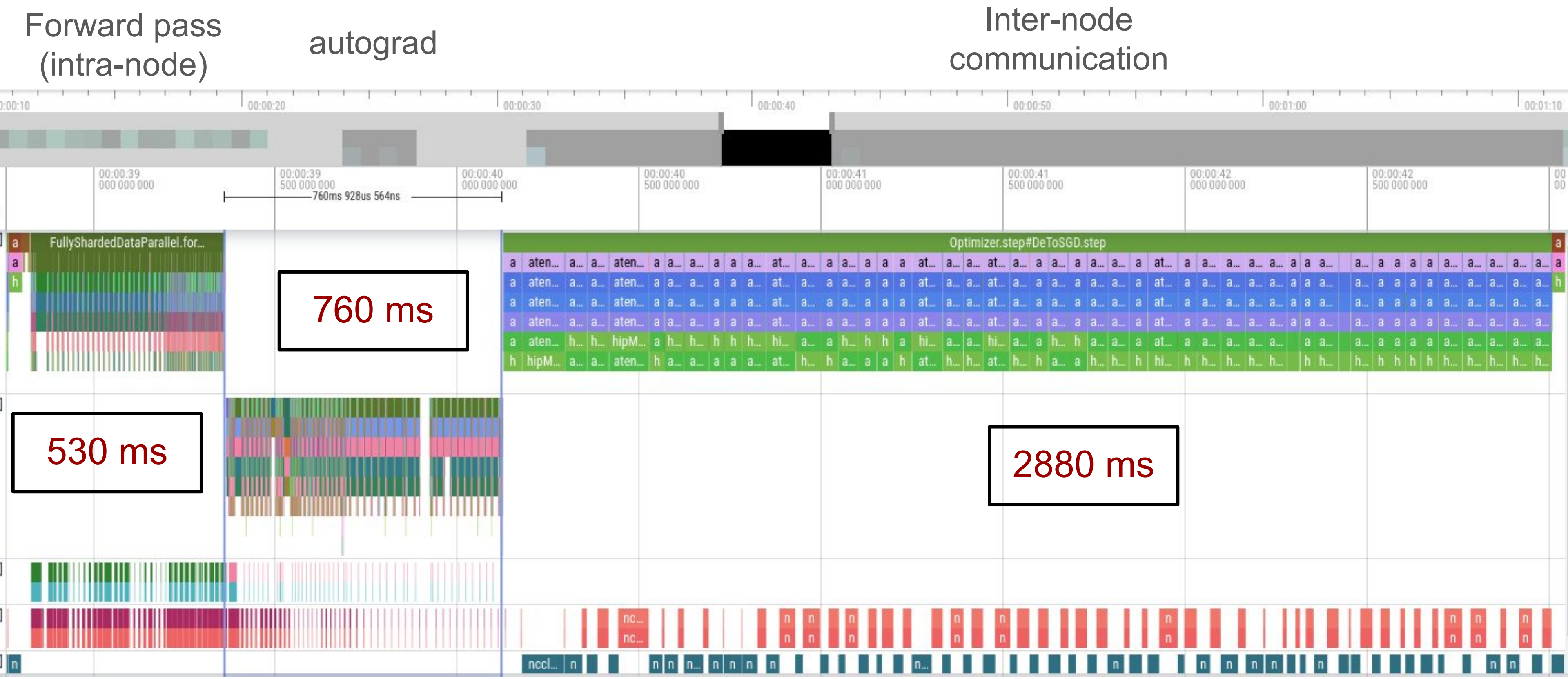
DeToNATION

LUMI Hackathon
12-16th May, 2025

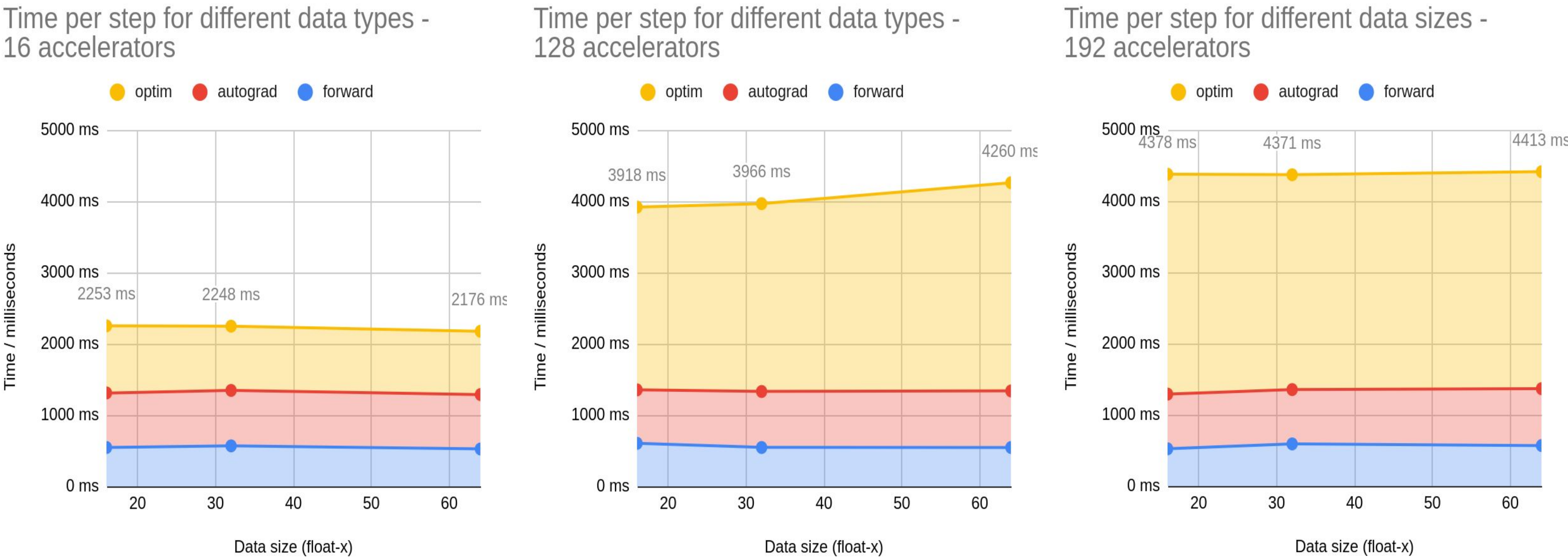
Mogens Henrik From & Jacob Nielsen
University of Southern Denmark (SDU) · Ordbogen A/S


Node 0
Node 1
Shard 0
Shard 0
Shard 1
Shard 1

Communication bound



Communication bound: Latency bound?



Sign before vs after communication

- *Sign* parameters gives us a ternary system: { -1, 0, 1 }
  - can be represented in 2 bits → packing 4 (5) values into an *int8* structure.
- This reduces the communication requirements in *all_reduce* or *all_gather*
- However, the training behaviour is different, yielding worse performance.

*Sign **after** communication:*

```
0: Epoch 1 training loss  : 0.5599
0: Epoch 1 validation Loss: 0.4720
0: Epoch 2 training loss  : 0.4575
0: Epoch 2 validation Loss: 0.4283
0: Epoch 3 training loss  : 0.4237
0: Epoch 3 validation Loss: 0.4010
```

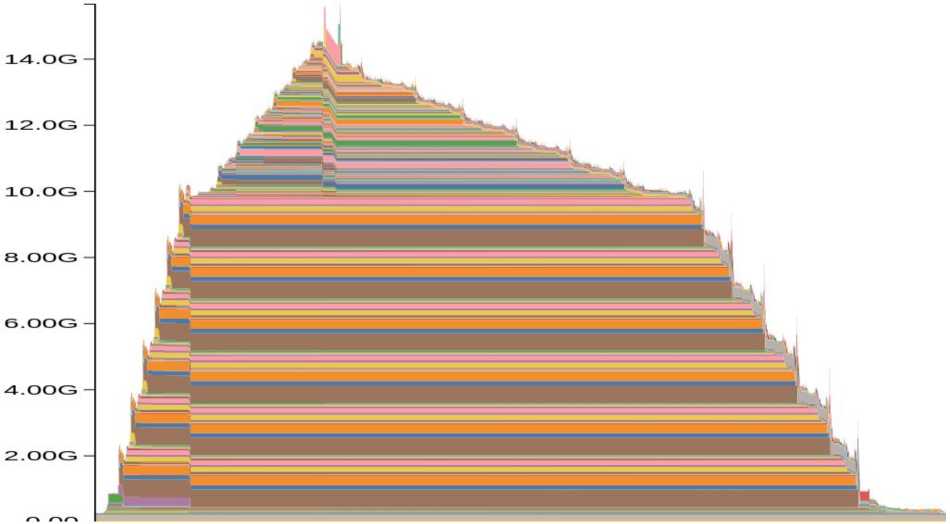
*Sign **before** communication:*

```
0: Epoch 1 training loss  : 1.0179
0: Epoch 1 validation Loss: 0.7328
0: Epoch 2 training loss  : 0.7114
0: Epoch 2 validation Loss: 0.6753
0: Epoch 3 training loss  : 0.6736
0: Epoch 3 validation Loss: 0.6508
```

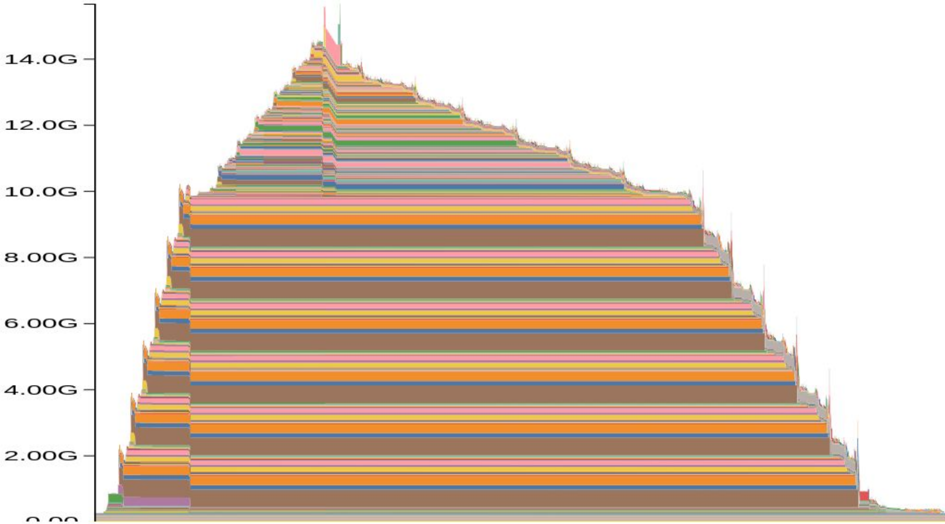
The memory issue

```
torch.cuda.memory._record_memory_history()
step()
torch.cuda.memory._dump_snapshot()
```

Sharding to 2 GPUs



Sharding to 8 GPUs



The memory issue

- We noticed that the memory reduction on each GPU was not as expected when sharding
- In short: PyTorch's CUDA Caching Allocator

1 GPU:

FSDP peak memory Peak memory: **0.242 GB**
FSDP peak memory Peak max memory: 0.484 GB

2 GPUs:

FSDP peak memory Peak memory: **0.122 GB**
FSDP peak memory Peak max memory: 0.606 GB

NCCL Variables

- The small details, with the big consequences.
- Small scale experiments, Nodes 2x8, Batch size 64.
- *NCCL_MIN_NCHANNELS / NCCL_MAX_NCHANNELS*
  - Default NCCL Auto: 1.86 s/it
  - *min. 16, max. 32* 1.30 s/it
  - *min. 32, max. 32* 1.03 - ~1.20 s/it
- NCCL_NET_GDR_LEVEL=PHB
  - Use GPU Direct RDMA when GPU and NIC are on the same NUMA node.
  - No direct change, probably already used automatically
- NCCL_ALGO
  - Still pending
  - Here we want to test on larger set of nodes

Tooling

- Helping us doing sanity checks on the allocated nodes and their connections with HPC Affinity Tracker (HPCAT)

- RCCL Tests making sanity checks together with HPCAT results.

- Using NCCL Debug outputs in trying to analyse the actual communication

Thanks for all guidance!