Distributed Training

MIPT
18.03.2021
Oleh Shliazhko

https://github.com/king-menin/mipt-nlp2021

Today

- Why distributed training?
- Process Communication 101
- Pytorch Distributed Data Parallel
- Practice: Pytorch DDP
- Beyond DDP: Deepspeed and Model Parallel

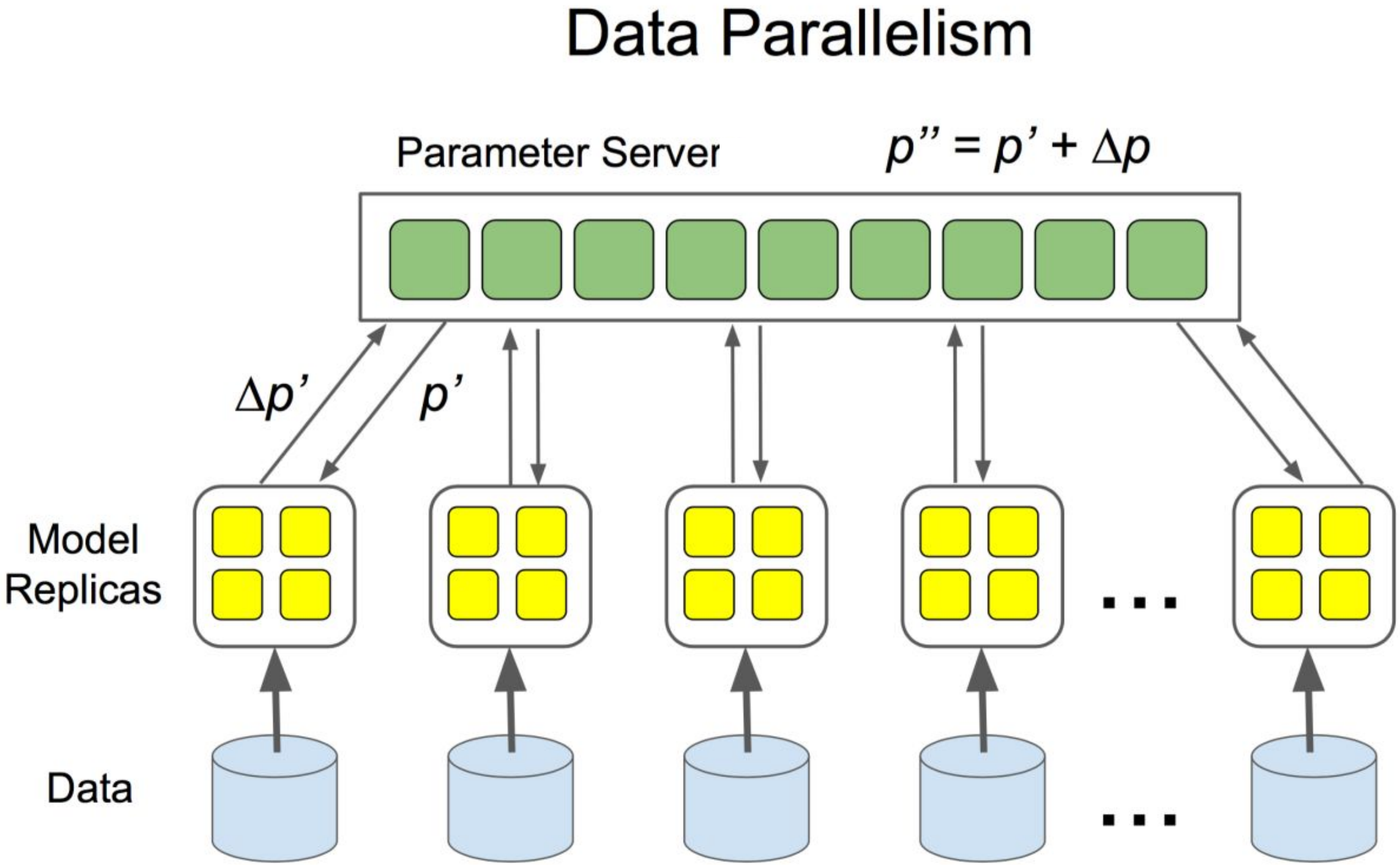
Why distributed training?

- Training speed - samples/second

- Bottleneck - batch size, due to GPU Memory

- Solution
  - split batch between GPUs
  - copy model to all GPUs
  - update master parameters from all gradients
  - copy updated parameters back to GPUs

Why distributed training?

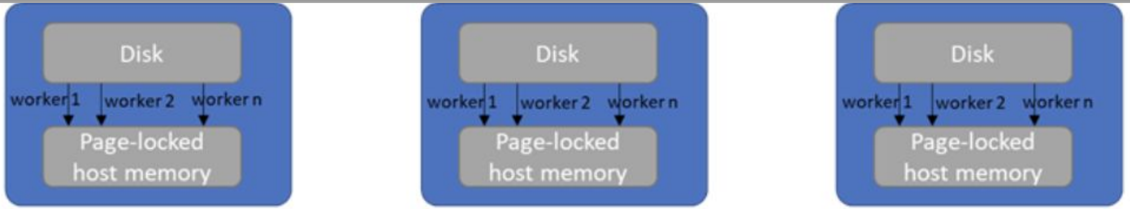


Why distributed training?

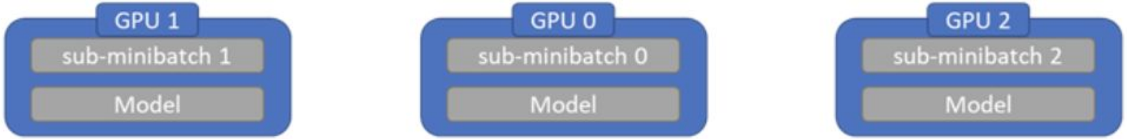
- Single node case - master copy of weights in RAM
- Bottleneck - number of GPUs on a single node
- Solution:
  - GET MOAR NODES
  - split batch between GPUs on many servers
  - sync weight updates for all models in some way

Why distributed training?

1. Load data from disk into page-locked memory on the host. Use multiple worker processes to parallelize data load. Distributed minibatch sampler ensures that each process loads non-overlapping data
h

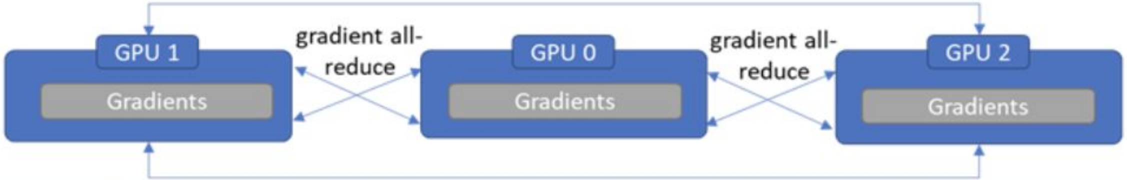


2. Transfer minibatch data from page-locked memory to each GPU concurrently. No data broadcast is needed. Each GPU has an identical copy of the model and no model broadcast is needed either



3. Run forward pass on each GPU, compute output

4. Compute loss, run backward pass to compute gradients. Perform gradient all-reduce in parallel with gradient computation



5. Update Model parameters. Because each GPU started with an identical copy of the model and gradients were all-reduced, weights updates on all GPUs are identical.

Why distributed training?

Could we get rid of single parameter server?

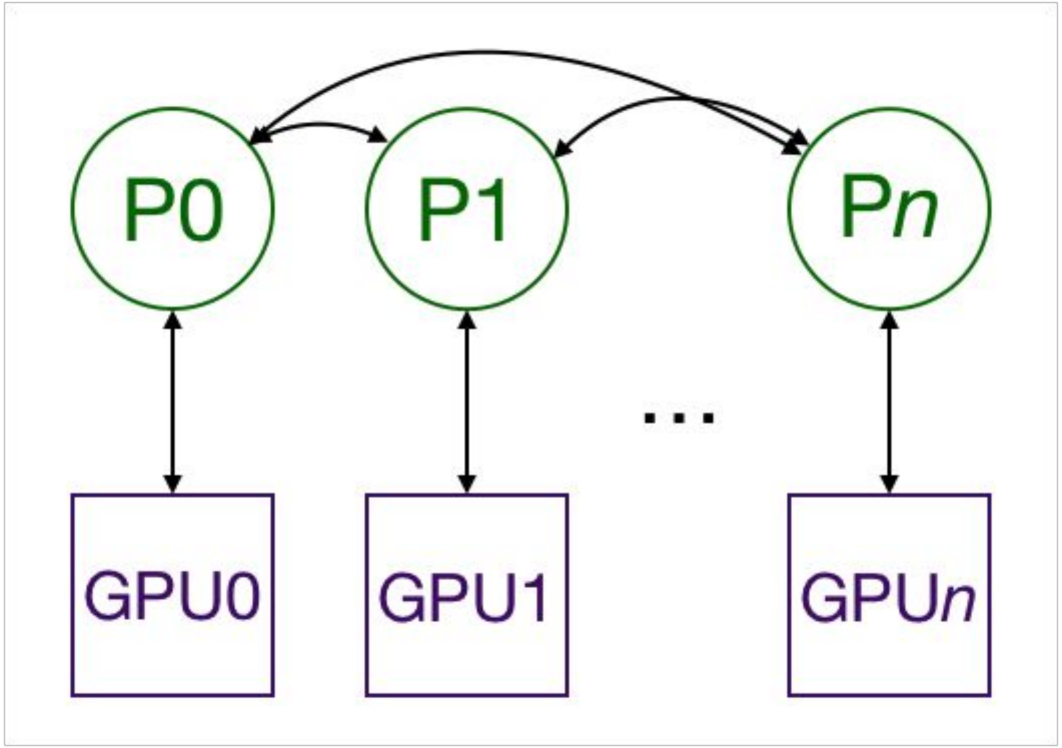
Let's try put all gradients on all workers.

Pros:

- No single point of failure
- Nice identical code in each process (SIMD)

Cons:

- A lot of networking



Why distributed training?

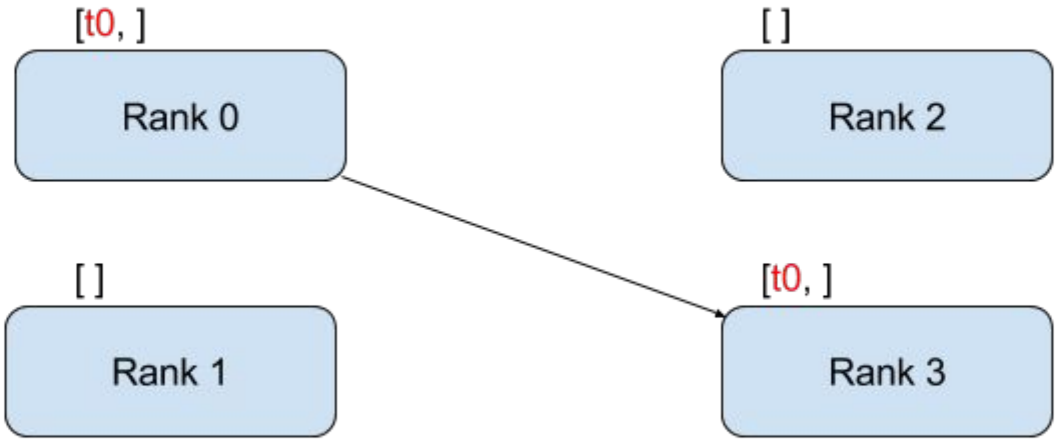
Single node algorithm

1. Get batch from dataloader
2. Forward pass
3. Backward pass
4. **Get gradients from all GPUS**
5. Compute gradients average
6. Update local weights

Process Communication 101

Point-to-Point Communication

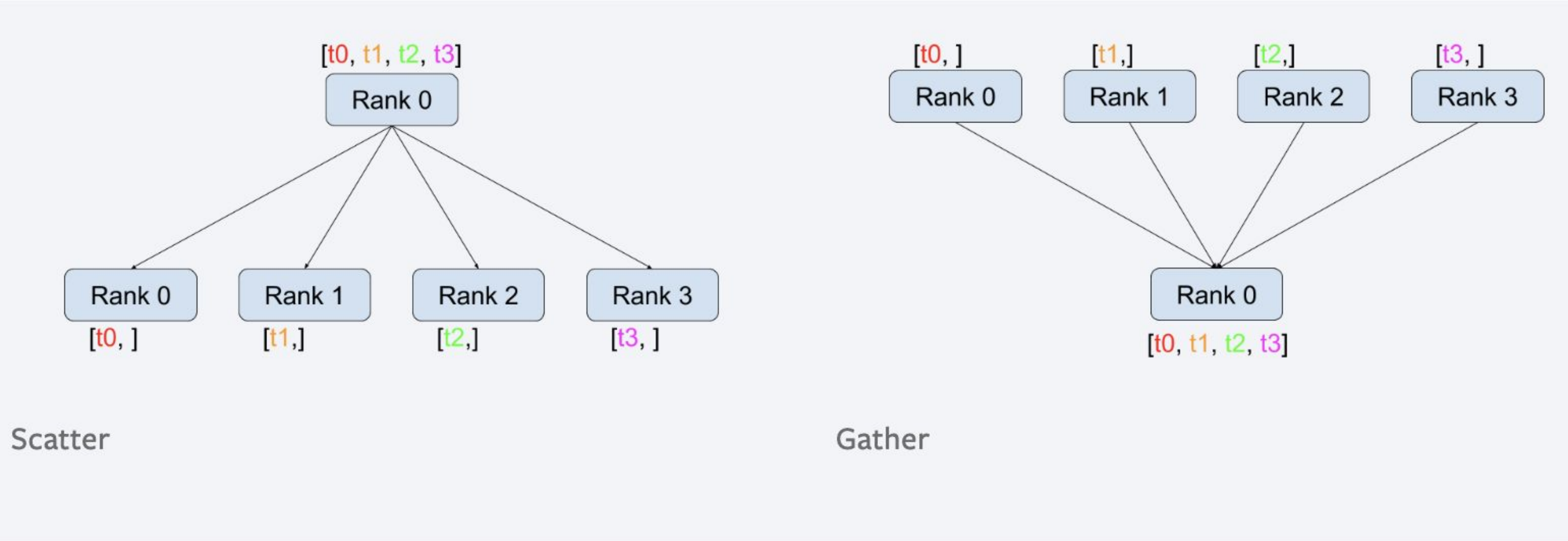
Source: send(tensor, dest_rank)
Destination: recv(tensor, src_rank)



https://pytorch.org/tutorials/intermediate/dist_tuto.html

Process Communication 101

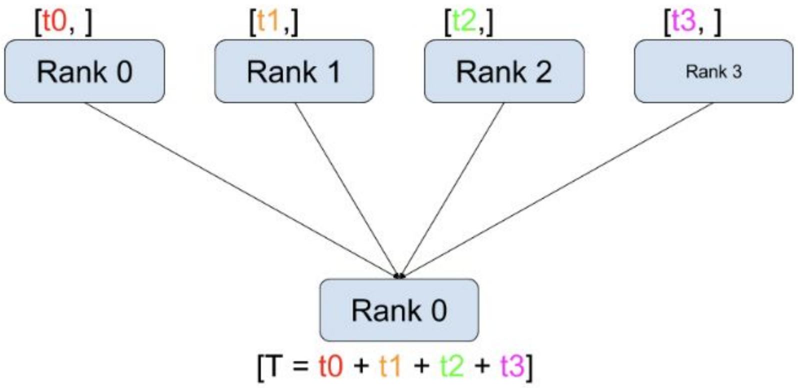
Collective Communication



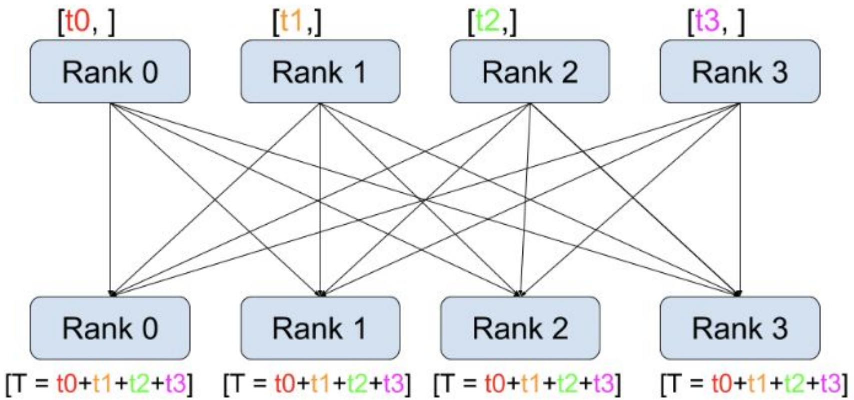
https://pytorch.org/tutorials/intermediate/dist_tuto.html

Process Communication 101

Collective Communication



Reduce

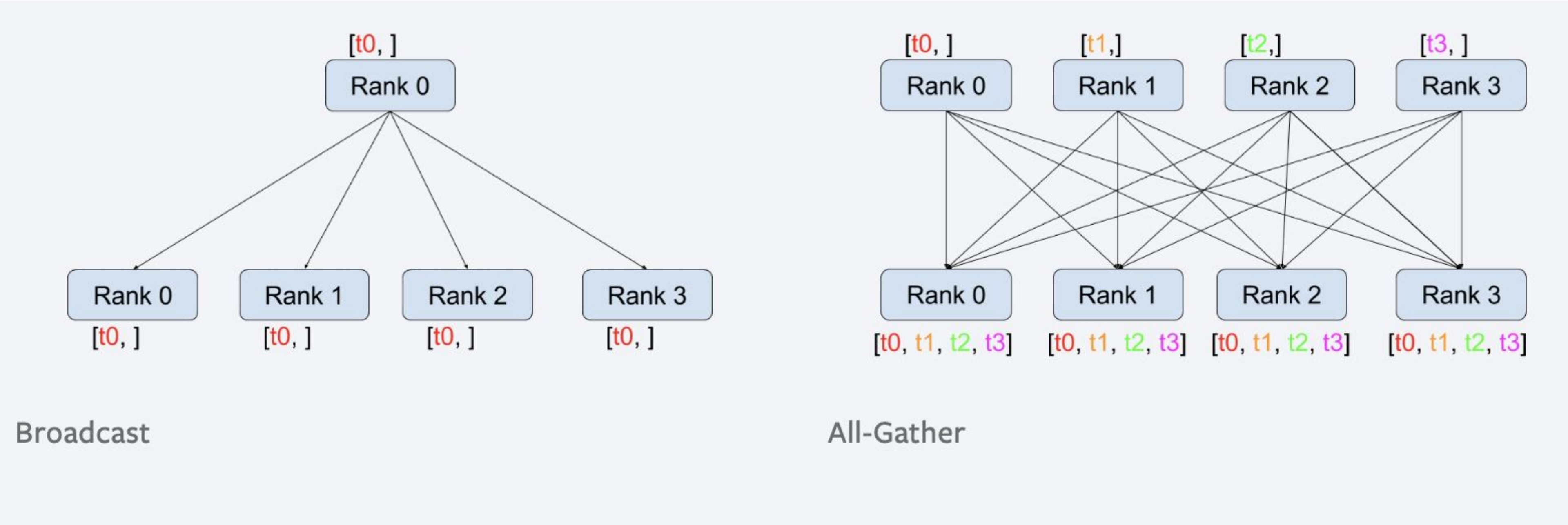


All-Reduce

https://pytorch.org/tutorials/intermediate/dist_tuto.html

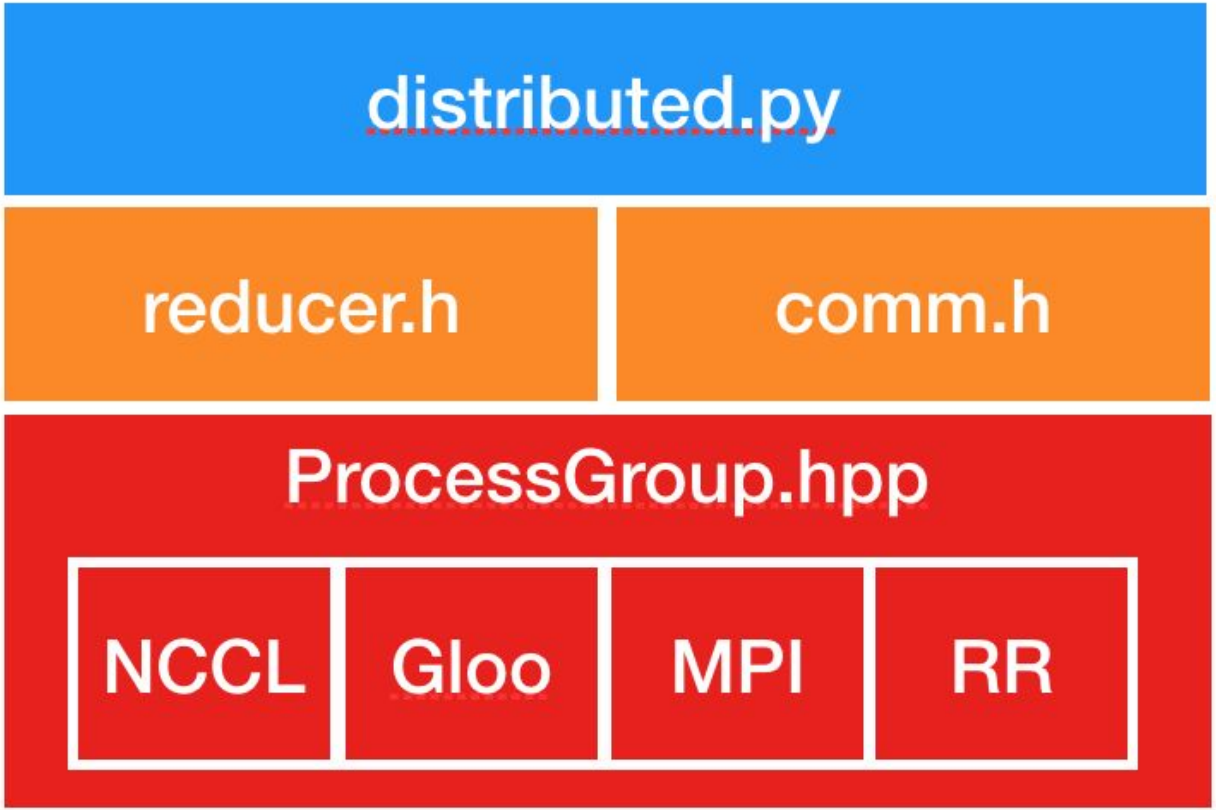
Process Communication 101

Collective Communication



https://pytorch.org/tutorials/intermediate/dist_tuto.html

Pytorch Distributed Data Parallel



Backend-agnostic API

Broadcast/Gather/Reduce operations

Convenient Data Parallel Wrapper

Pytorch Distributed Data Parallel

Glossary:

- **world_size** - total number of GPUs. Also a number of processes, assuming 1 process per GPU
- **global_rank** - global process id, unsigned int from [0, world_size)
- **local_rank** - local process id for current node, unsigned int from [0, number_of_GPUs_per_node)

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

dist.init_process_group(backend='nccl', init_method='env://')
torch.cuda.set_device(local_rank)

#  Split  dataset into world_size parts and read part number global_rank
dataloader = get_dataset_part(global_rank, world_size)

model = DDP(model, device_ids=[local_rank])
```

Practice: Pytorch DDP

Beyond DDP: Deepspeed & Model Parallel
GOTO: train_large_scale_gpt3

Questions