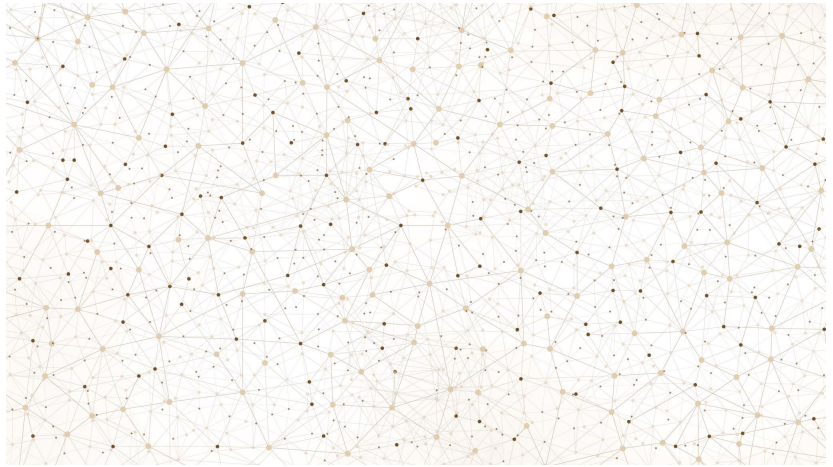
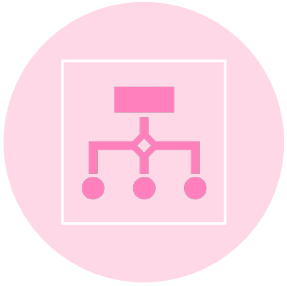
INTRODUCTION TO HOROVOD

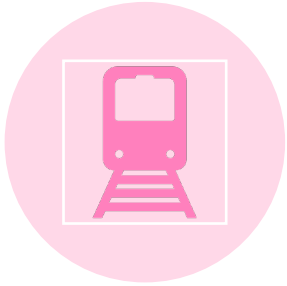


BY KEVIN FOTSO

What is Horovod? (1)



A distributed deep learning training framework for Tensorflow, Keras, Pytorch, and Apache MXNet.



Originally developed by Uber to train ML models in hours and mins instead of days and weeks.

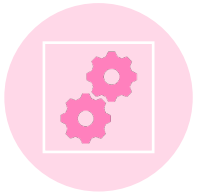
What is Horovod? (2)

WITH HOROVOD, ML ALGORITHM CAN BE SCALED TO RUN ON 100 OF GPUS WITH JUST FEW LINE OF PYTHON CODE.

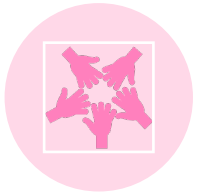
PORTABLE, RELATIVELY EASY AND FAST.

IT IS HOSTED UNDER THE LINUX FOUNDATION

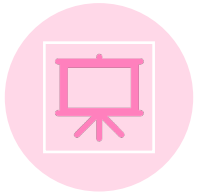
Context (1) – Tensorflow distributed



There are other tools to achieve distributed ML parallelism.



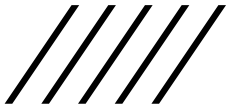
Tensorflow distributed (tf.distribute.Strategy).



Specific to Tensorflow only.

Supports parallelization on GPUs and TPUs.

More used on the cloud and less on HPC.



Source: https://www.tensorflow.org/guide/distributed_training

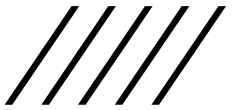
Context (2) – Pytorch distributed

- **Pytorch Distributed framework**
- Developed by the Facebook AI team
- Lots of tutorial on how to use it on HPC
- Creates replicas of the model with each working on a batch

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Import torch and torch DDP modules

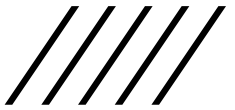


Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Local model is created

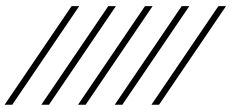


Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Converted into distributed training model

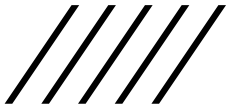


Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Optimizer

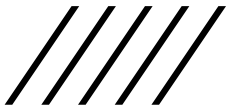


Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Backward pass of the model

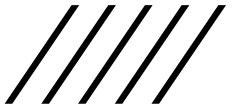


Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Relatively non intrusive overall!!!



Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

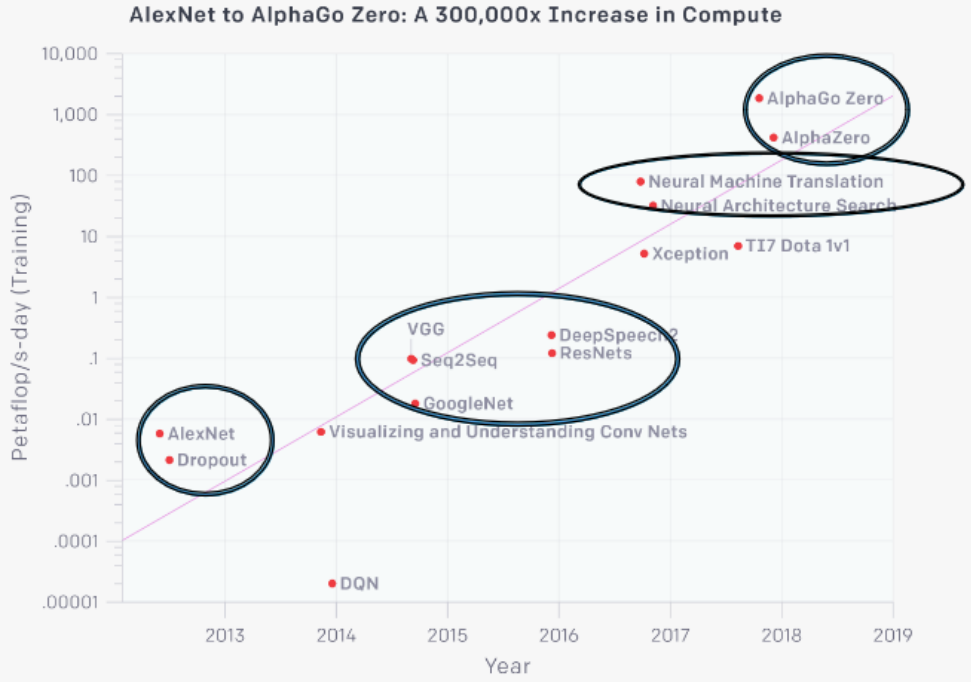
Context (3) - LBANN

- Livermore Big Artificial Neural Network toolkit (LBANN)
- Can be installed with spack
- Provides a Pytorch frontend
- Targets HPC systems with homogeneous compute nodes
- GPU accelerators.
- Relies only on MPI for communication.

Need for ML distributed training(1)

Need for distributed (parallel) training on HPC

"Since 2012, the amount of compute used in the largest AI training runs has been increasing exponentially with a 3.5 month doubling time (by comparison, Moore's Law had an 18 month doubling period)." https://openai.com/blog/ai-and-compute/



Eras:

- Before 2012 …
- 2012 – 2014: single to couple GPUs
- 2014 – 2016: 10 – 100 GPUs
- 2016 – 2017: large batch size training, architecture search, special hardware (etc, TPU)

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

Need for ML distributed training(2)

Distributed deep learning for ResNet-50

TRAINING TIME AND TOP-1 VALIDATION ACCURACY WITH RESNET-50 ON IMAGENET

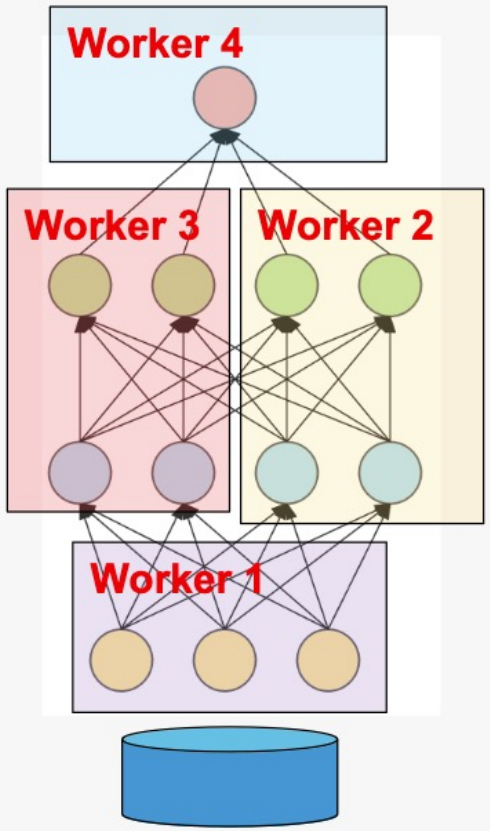
		Batch Size	Processor	DL Library	Time	Accuracy
He et al. [1]	2016	256	Tesla P100 × 8	Caffe	29 hours	75.3 %
Goyal et al. [2]		8,192	Tesla P100 × 256	Caffe2	1 hour	76.3 %
Smith et al. [3]		8,192 → 16,384	full TPU Pod	TensorFlow	30 mins	76.1 %
Akiba et al. [4]		32,768	Tesla P100 × 1,024	Chainer	15 mins	74.9 %
Jia et al. [5]		65,536	Tesla P40 × 2,048	TensorFlow	6.6 mins	75.8 %
Ying et al. [6]		65,536	TPU v3 × 1,024	TensorFlow	1.8 mins	75.2 %
Mikami et al. [7]		55,296	Tesla V100 × 3,456	NNL	2.0 mins	75.29 %
Yamazaki et al	2019	**81,920**	**Tesla V100 × 2,048**	**MXNet**	**1.2 mins**	**75.08%**

Quoted from Masafumi Yamazaki, arXiv:1903.12650

Source: ATPESC 2023, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-8-Talk-7-Zheng-Distributed-Deep-Learning.pdf

Parallelization schemes (1)

Parallelization schemes – Model Parallelism (MP)



Model parallelism

```python
import torch
import torch.nn as nn
import torch.optim as optim


class ToyModel(nn.Module):
    def __init__(self):
        super(ToyModel, self).__init__()
        self.net1 = torch.nn.Linear(10, 10).to('cuda:0')
        self.relu = torch.nn.ReLU()
        self.net2 = torch.nn.Linear(10, 5).to('cuda:1')

    def forward(self, x):
        x = self.relu(self.net1(x.to('cuda:0')))
        return self.net2(x.to('cuda:1'))
```

```python
model = ToyModel()
loss_fn = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.001)

optimizer.zero_grad()
outputs = model(torch.randn(20, 10))
labels = torch.randn(20, 5).to('cuda:1')
loss_fn(outputs, labels).backward()
optimizer.step()
```

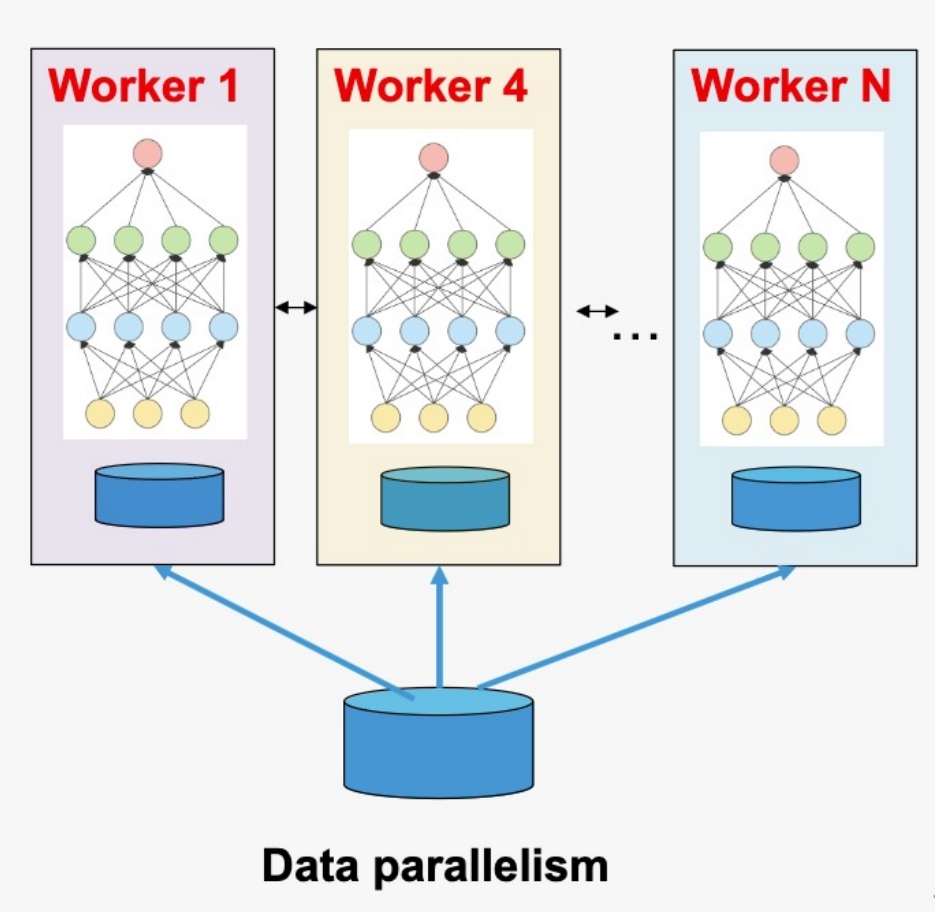
PyTorch multiple GPU model parallelism within a node

Worker 1

Worker 2

Source: ATPESC 2023, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-8-Talk-7-Zheng-Distributed-Deep-Learnng.pdf

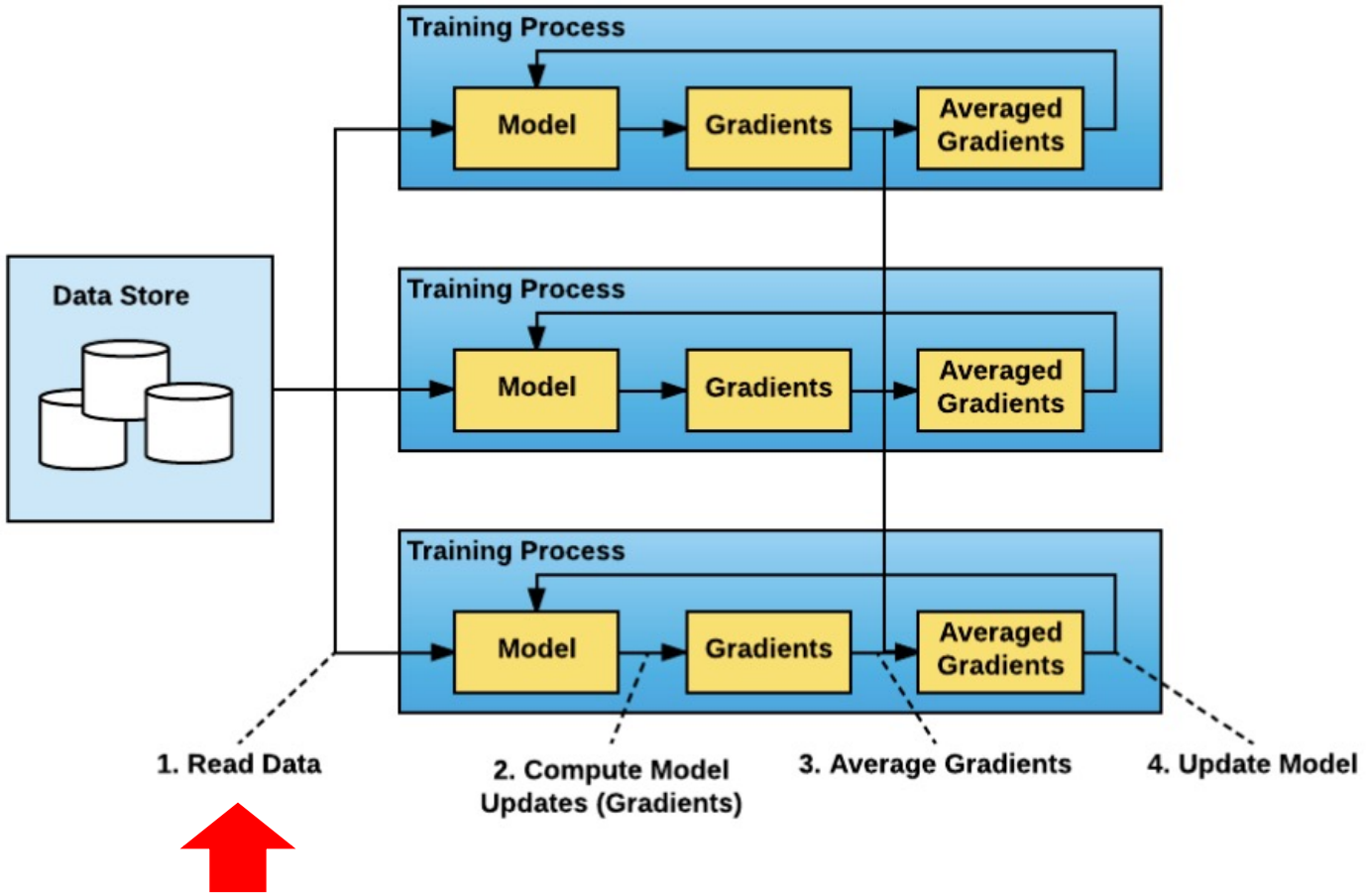
Parallelization schemes (2)



- Data parallelism
- Model is replicated on each worker
- Each worker processes a subset of the minibatch
- Weights are synchronized before updating the model
- Widely supported.

Source: ATPESC 2023, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-8-Talk-7-Zheng-Distributed-Deep-Learnng.pdf

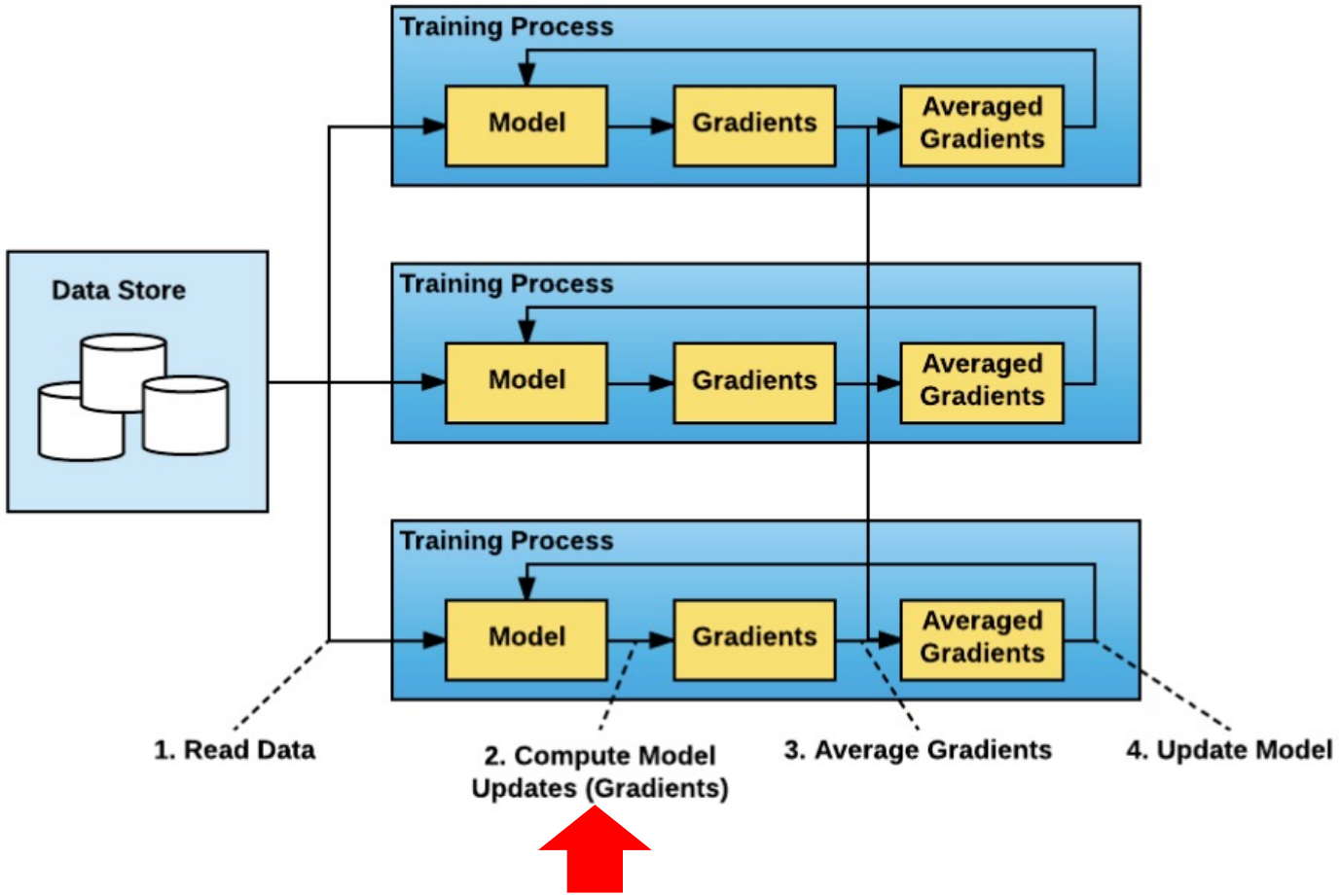
Overview of the Horovod training model



Reads chunk of data (batch)

Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

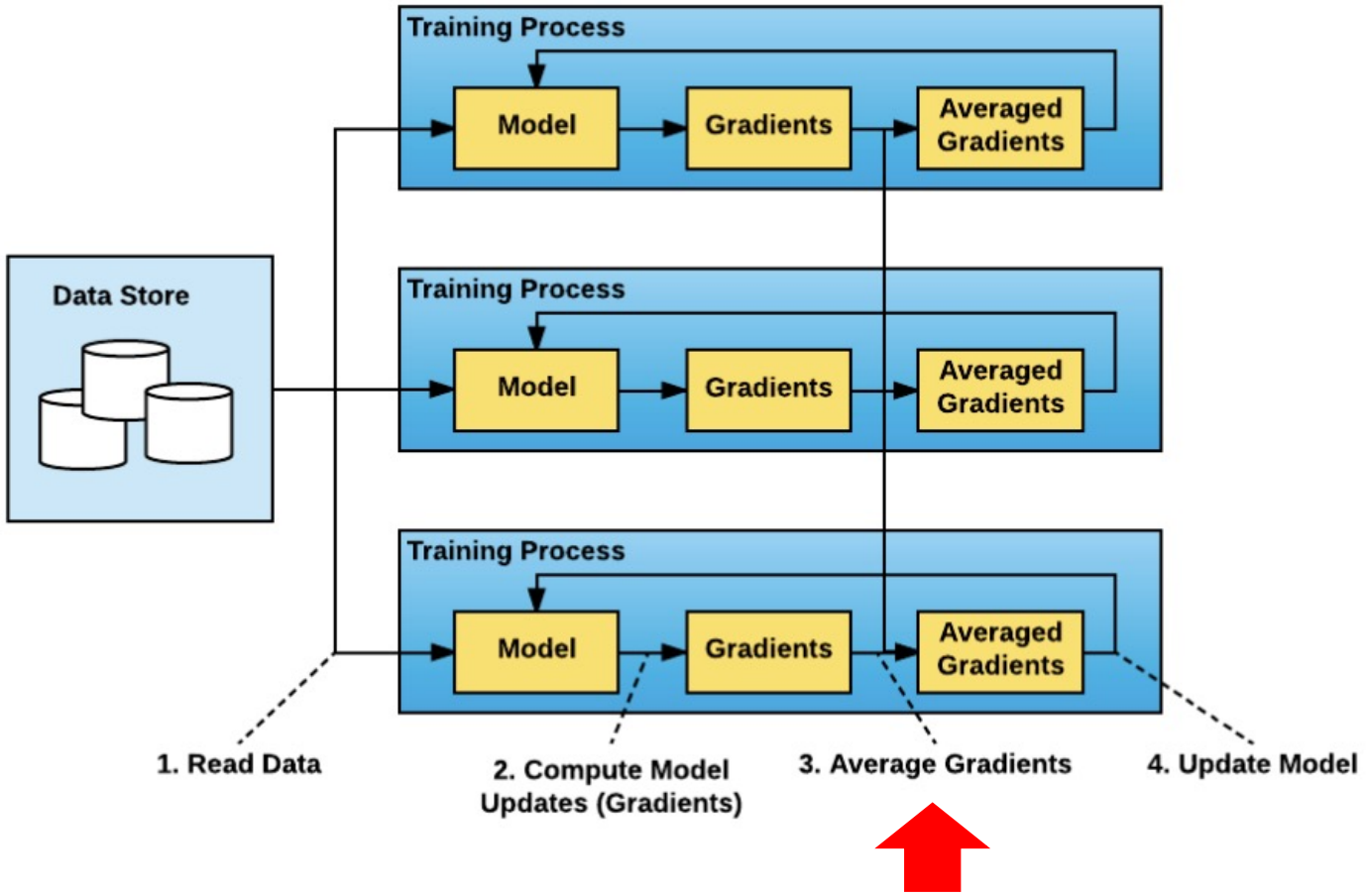
Overview of the Horovod training model (1)



Gradients of different batches calculated separately

Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

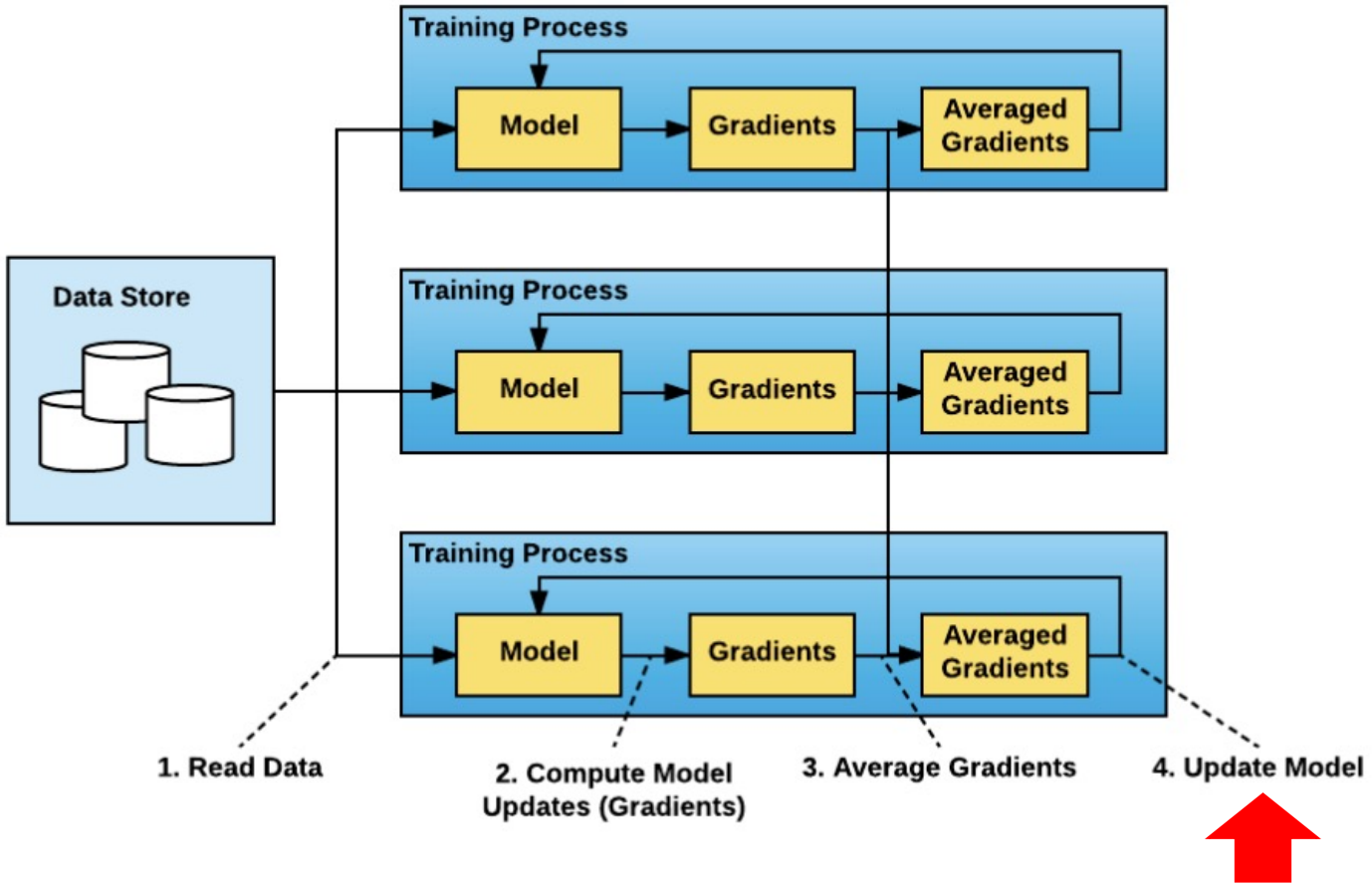
Overview of the Horovod training model(1)



Average across nodes to apply consistent updates to model copy in each node

Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

Overview of the Horovod training model (1)

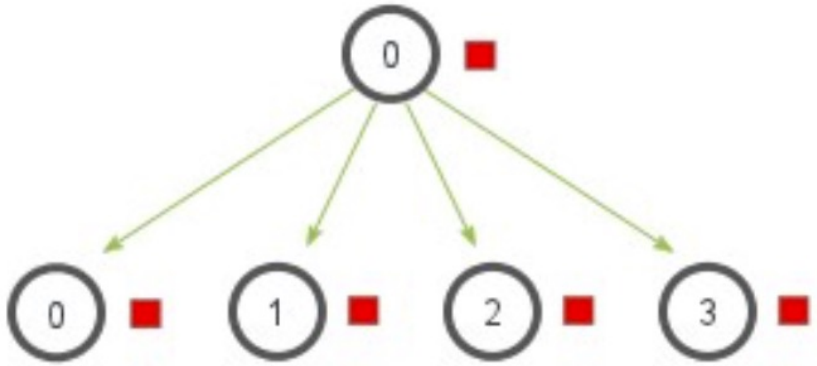


Update the model and repeat

Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

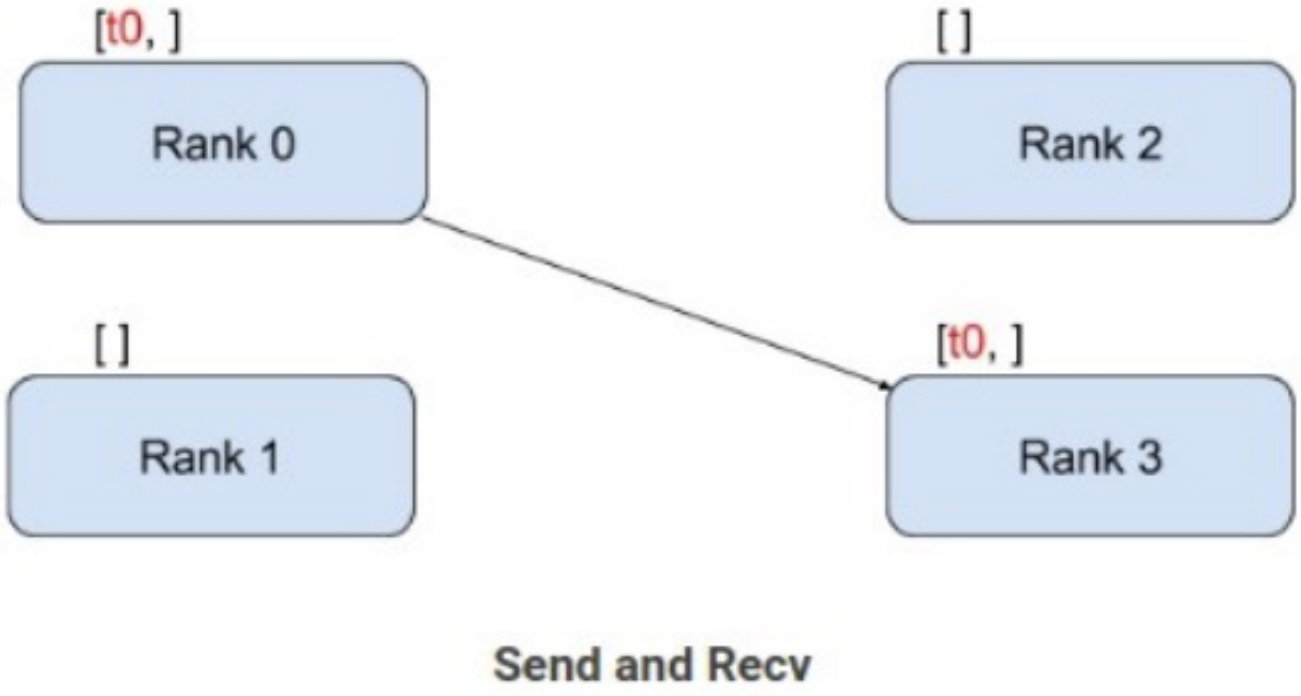
Distributed computing basics (1)

- A rank is an integer that serves as an identifier of a node from 0 to N-1
- A rank of 0 identifies the “Master Node” and any other ranks are the workers.
- Basic operation of distributed computing.



Point-to-point communication

- Transfer data from one rank to another

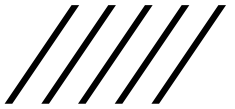


Source: Pytorch tutorial WRITING DISTRIBUTED APPLICATIONS WITH PYTORCH

Point-to-point communication

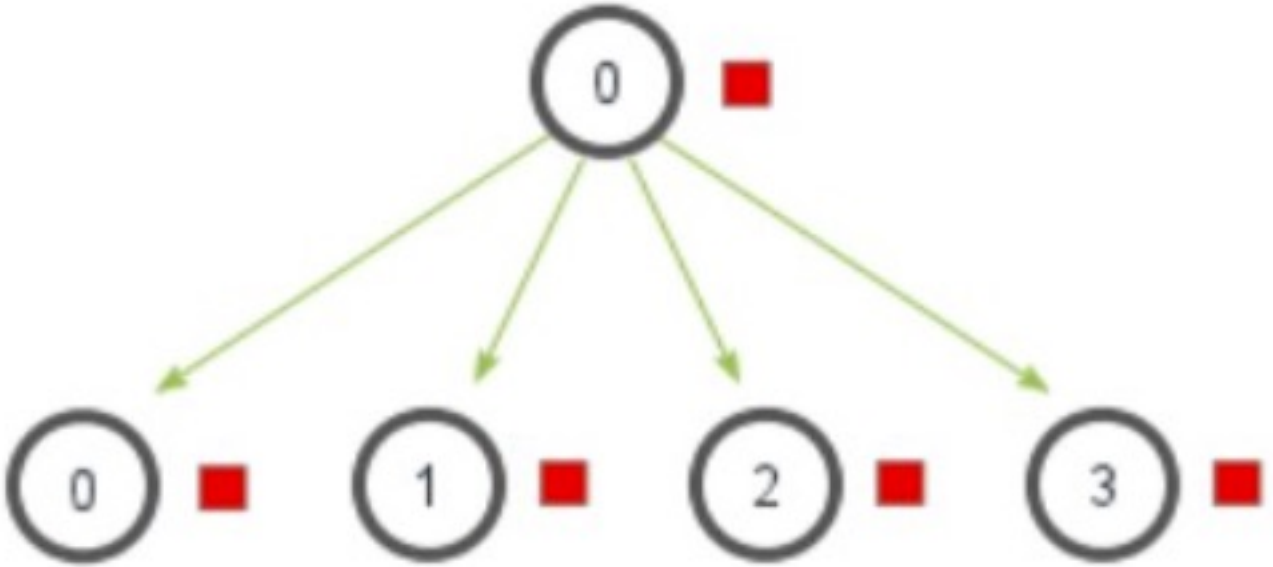
```python
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = {'a': 7, 'b': 3.14}
    comm.send(data, dest=1)
elif rank == 1:
    data = comm.recv(source=0)
    print('On process 1, data is ',data)
```

Rank 0 is sending data to rank 1



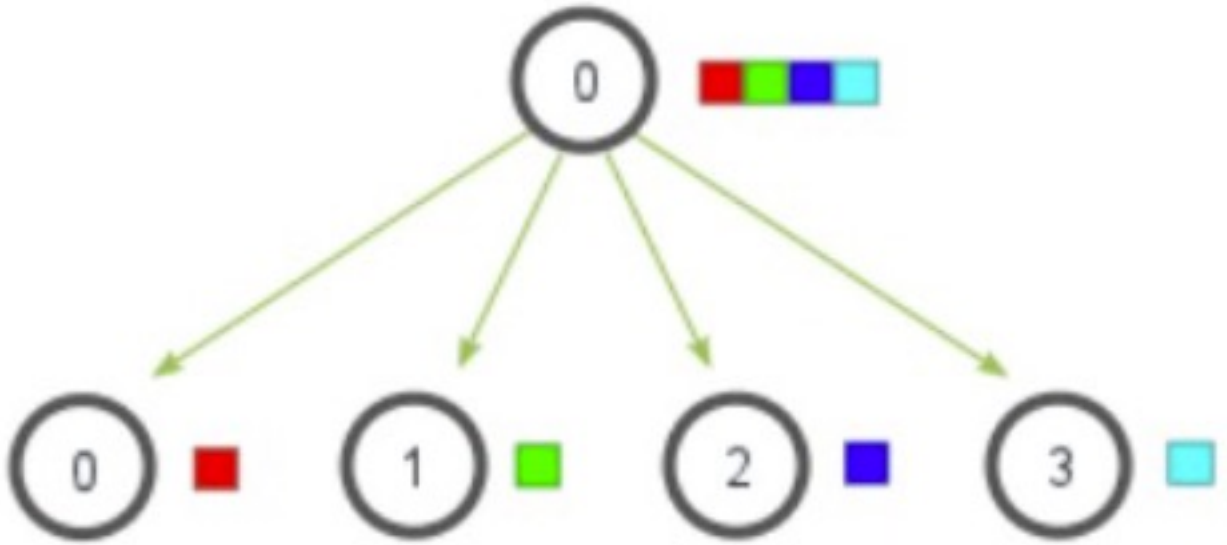
Recap- Collective communication (1)

- Broadcasting: Takes 1 chunk of variable and send an exact copy to all processes



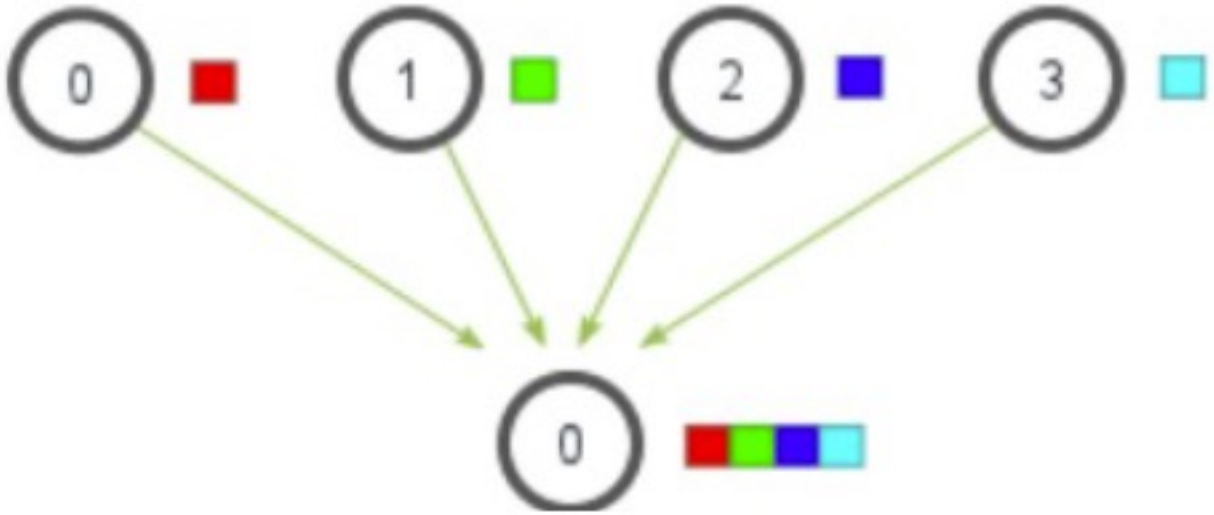
Collective communication (2)

- Scattering: Takes an array and distributes section across ranks



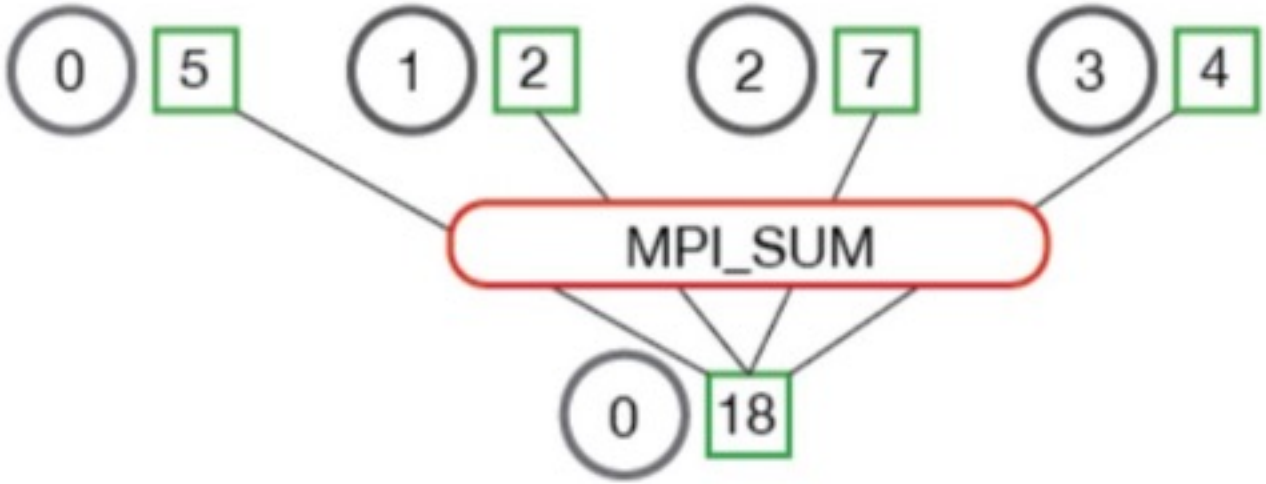
Collective communication (3)

- Gathering: Is the reverse of scattering



Collective communication (4)

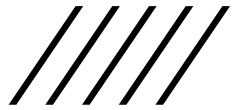
- Reduce: Takes values from an array on each processes and reduces them to a single result root process.



Snippet of Horovod basic code

```
import tensorflow as tf
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes
# during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing
# when done or an error occurs.
```

Import horovod & tensorflow

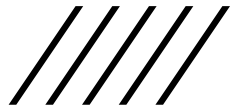


Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

Snippet of Horovod basic code

```
import tensorflow as tf
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes
# during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing
# when done or an error occurs.
```

Wraps any regular Tensorflow opt with Horovod optimizer which will take care of averaging the gradients automatically

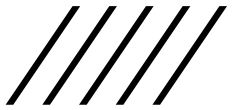


Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

Snippet of Horovod basic code

```
import tensorflow as tf
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes
# during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing
# when done or an error occurs.
```

Broadcast variable across all ranks to ensure consistent initialization



Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

Snippet of Horovod basic code

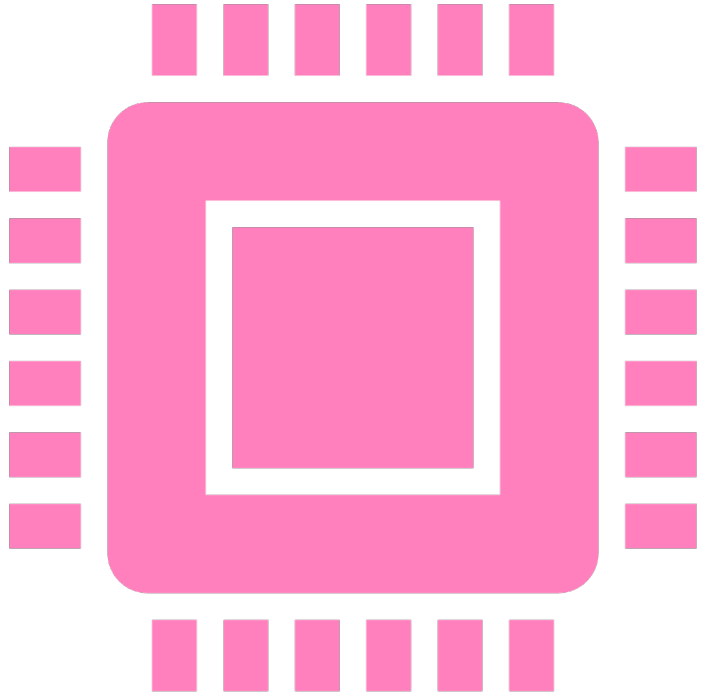
```
with tf.train.MonitoredTrainingSession(checkpoint_dir="/tmp/train_logs",
                                       config=config,
                                       hooks=hooks) as mon_sess:
 while not mon_sess.should_stop():
   # Perform synchronous training.
   mon_sess.run(train_op)
```

Monitoring session for automatic session init, checkpoint accommodation & closing

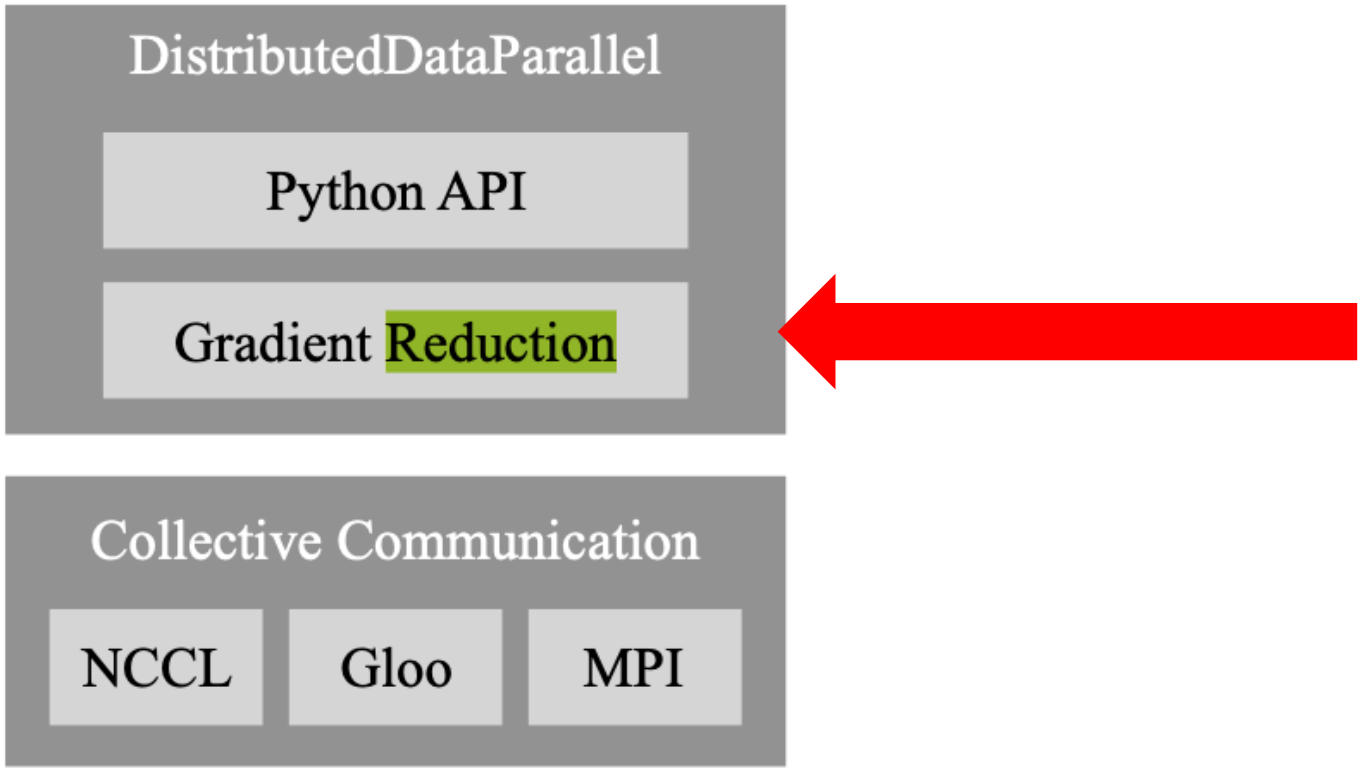
Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

Collective communication with Horovod

- Message Passing Interface (MPI) for CPUs and GPUs in general.
- NVIDIA Collective Communication Library (NCCL) with multi-node & multi GPU.
- GLOO developed by Facebook AI
- Allow to average the gradients automatically (ring-allreduce operation)



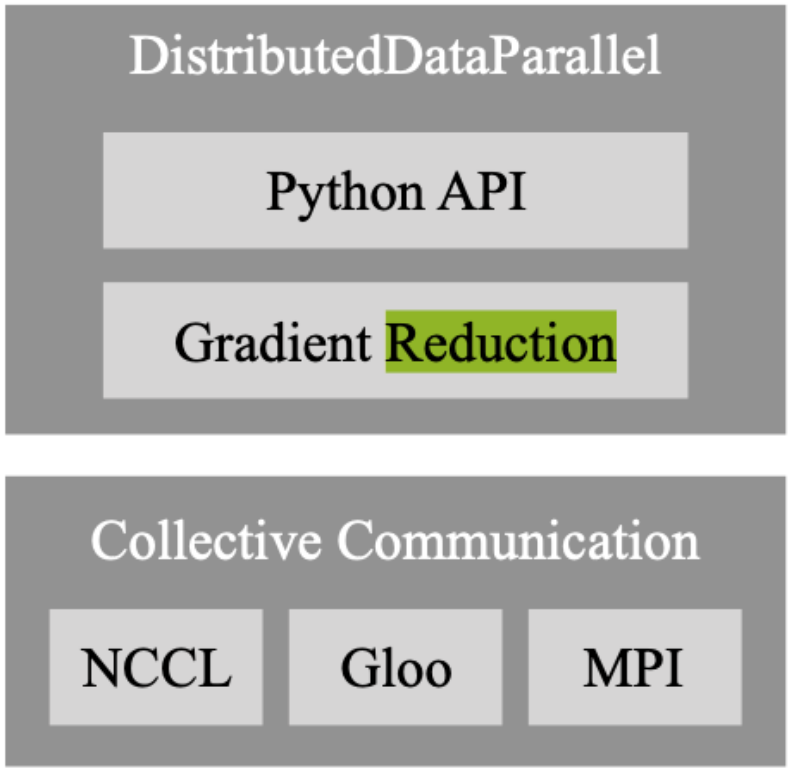
Collective communication with Pytorch distributed



AllReduce to calculate average gradients on each parameter across all processes -> gradient tensor

Source: "Pytorch Distributed: Experiences on Accelerating Data Parallel Training

Collective communication with Pytorch distributed

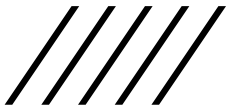


Communication backends

Source: "Pytorch Distributed: Experiences on Accelerating Data Parallel Training

Install Horovod on Alpine

- Install the Deep learning Framework (Tensorflow, Pytorch)
- Load Openmpi
- Install `gxx_linux-64`
- Use spack to install nccl (`cuda_arch=80`)
- Follow this guide: https://horovod.readthedocs.io/en/latest/gpus.html



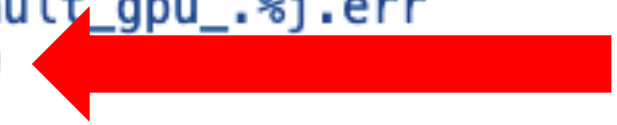
Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

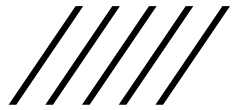
Job name, output and error

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```



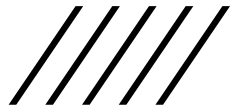
NVIDIA partition name



Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

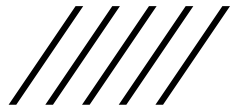
We ask for 2 nodes



Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

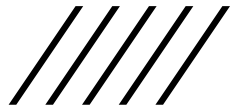
We ask for 30 CPU cores total.
NOT GPU cores !!!



Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

We ask for 2 GPUs per node



Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

Anschutz users account

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00
```

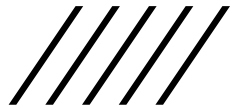
Walltime

```
ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

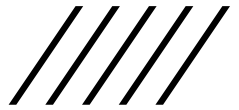
Load gcc to load OpenMPI



Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

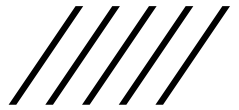
Load cuda to use nsys



Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

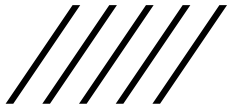
Load anaconda



Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

Activate environment



Horovod on Alpine (Slurm script)

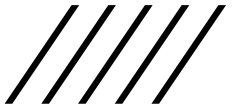
NCCL library export

```
# We export the NCCL library
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/projects/kefo9343/software/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/nccl-2.18.3-1-2pmue

# We want the XLA flags and tensorrt for optimization
export XLA_FLAGS=--xla_gpu_cuda_data_dir /projects/kfotso@xsede.org/software/anaconda/envs/prs_gpu

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/python3.8/site-packages/tensorrt


# If I want to run on multiple nodes, I will need to know their hostnames
scontrol show hostname > $SLURM_SUBMIT_DIR/nodelist.txt
node1=`sed -n 1p $SLURM_SUBMIT_DIR/nodelist.txt`

# An example in case you want to add a new node, corresponding to the line number
node2=`sed -n 2p $SLURM_SUBMIT_DIR/nodelist.txt`
```

Horovod on Alpine (Slurm script)

```
# We export the NCCL library
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/projects/kefo9343/software/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/nccl-2.18.3-1-2pmuetqeeecftln5wn6jqy6xcvyzak6d/lib

# We want the XLA flags and tensorrt for optimization
export XLA_FLAGS=--xla_gpu_cuda_data_dir /projects/kfotso@xsede.org/software/anaconda/envs/prs_gpu

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/python3.8/site-packages/tensorrt


# If I want to run on multiple nodes, I will need to know their hostnames
scontrol show hostname > $SLURM_SUBMIT_DIR/nodelist.txt
node1=`sed -n 1p $SLURM_SUBMIT_DIR/nodelist.txt`

# An example in case you want to add a new node, corresponding to the line number
node2=`sed -n 2p $SLURM_SUBMIT_DIR/nodelist.txt`
```

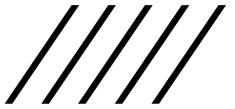
XLA flags and tensorRT for optimization



Horovod on Alpine (Slurm script)

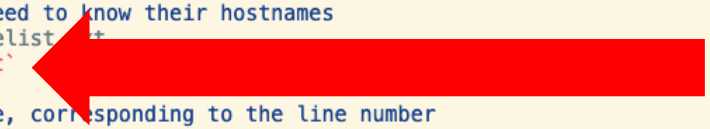
```
# We export the NCCL library
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/projects/kefo9343/software/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/nccl-2.18.3-1-2pmuetqeeecftln5wn6jqy6xcvyzak6d/lib

# We want the XLA flags and tensorrt for optimization
export XLA_FLAGS=--xla_gpu_cuda_data_dir /projects/kfotso@xsede.org/software/anaconda/envs/prs_gpu

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/python3.8/site-packages/tensorrt


# If I want to run on multiple nodes, I will need to know their hostnames
scontrol show hostname > $SLURM_SUBMIT_DIR/nodelist.txt
node1=`sed -n 1p $SLURM_SUBMIT_DIR/nodelist.txt`

# An example in case you want to add a new node, corresponding to the line number
node2=`sed -n 2p $SLURM_SUBMIT_DIR/nodelist.txt`
```

We create a nodelist with scontrol that we store in nodelist.txt



Horovod on Alpine (Slurm script)

```
# We export the NCCL library
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/projects/kefo9343/software/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/nccl-2.18.3-1-2pmuetqeeecftln5wn6jqy6xcvyzak6d/lib

# We want the XLA flags and tensorrt for optimization
export XLA_FLAGS=--xla_gpu_cuda_data_dir /projects/kfotso@xsede.org/software/anaconda/envs/prs_gpu

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/python3.8/site-packages/tensorrt


# If I want to run on multiple nodes, I will need to know their hostnames
scontrol show hostname > $SLURM_SUBMIT_DIR/nodelist.txt
node1=`sed -n 1p $SLURM_SUBMIT_DIR/nodelist.txt`

# An example in case you want to add a new node, corresponding to the line number
node2=`sed -n 2p $SLURM_SUBMIT_DIR/nodelist.txt`
```

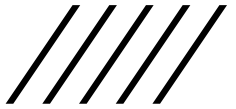


We assign the content of nodelist.txt into variables

Horovod on Alpine (Slurm script)

```
# We test it with 1 GPU
"Testing it with 1 GPU"
nsys profile --trace=cuda,nvtx,osrt,cudnn,cublas \
        --capture-range=cudaProfilerApi \
        --gpu-metrics-device=all \
        --output=horovod_profile \
        horovodrun -np 1 -H localhost:1 horovod_housing_tf.py
```

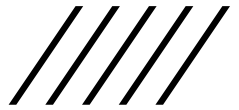
We use nsys for profiling



Horovod on Alpine (Slurm script)

```
# We test it with 1 GPU
"Testing it with 1 GPU"
nsys profile --trace=cuda,nvtx,osrt,cudnn,cublas \
        --capture-range=cudaProfilerApi \
        --gpu-metrics-device=all \
        --output=horovod_profile \
        horovodrun -np 1 -H localhost:1 horovod_housing_tf.py
```

More information on the parameters here: https://alcf.anl.gov/sites/default/files/2022-07/Nsight%20Systems%20-%20DL%20Profiling%20Argonne%20National%20Labs%202022-06-30.pdf



Horovod on Alpine (Slurm script)

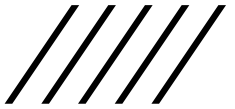
```
# We test it with 1 GPU
"Testing it with 1 GPU"
nsys profile --trace=cuda,nvtx,osrt,cudnn,cublas \
        --capture-range=cudaProfilerApi \
        --gpu-metrics-device=all \
        --output=horovod_profile \
        horovodrun -np 1 -H localhost:1 horovod_housing_tf.py
```

We start 1 major process for the GPU; useful for debugging

Horovod on Alpine (Slurm script)

```
# We test it with 1 GPU
"Testing it with 1 GPU"
nsys profile --trace=cuda,nvtx,osrt,cudnn,cublas \
        --capture-range=cudaProfilerApi \
        --gpu-metrics-device=all \
        --output=horovod_profile \
        horovodrun -np 1 -H localhost:1 horovod_housing_tf.py
```

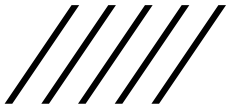
Name of the node
I am running on



Horovod on Alpine (Slurm script)

```
# We test it with 2 GPUs
echo "Testing it with 2 GPUs"
nsys profile --trace=cuda,nvtx,osrt,cudnn,cublas \
        --capture-range=cudaProfilerApi \
        --gpu-metrics-device=all \
        --output=horovod_profile \
        horovodrun -np 2 -H localhost:2 horovod_housing_tf.py
```

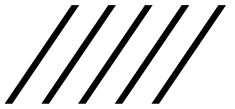
Here we use 2 GPUs



Horovod on Alpine (Slurm script)

```
# We test it with 3 GPUs
echo "Testing it with 3 GPUs"
nsys profile --trace=cuda,nvtx,osrt,cudnn,cublas \
        --capture-range=cudaProfilerApi \
        --gpu-metrics-device=all \
        --output=horovod_profile
        horovodrun -np 3 -H $node1:2,$node2:1 \
        horovod_housing_tf.py
```

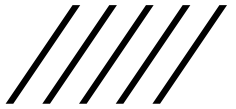
We use 2 GPUs on node 1 and 1 on node 2



Horovod on Alpine (Slurm script)

```
# We test it with 4 GPUs
echo "Testing it with 4 GPUs"
nsys profile --trace=cuda,nvtx,osrt,cudnn,cublas \
        --capture-range=cudaProfilerApi \
        --gpu-metrics-device=all \
        --output=horovod_profile \
        horovodrun -np 4 -H $node1:2,$node2:2
        horovod_housing_tf.py
```

We use 2 GPUs on node 1 and 2 on node 2



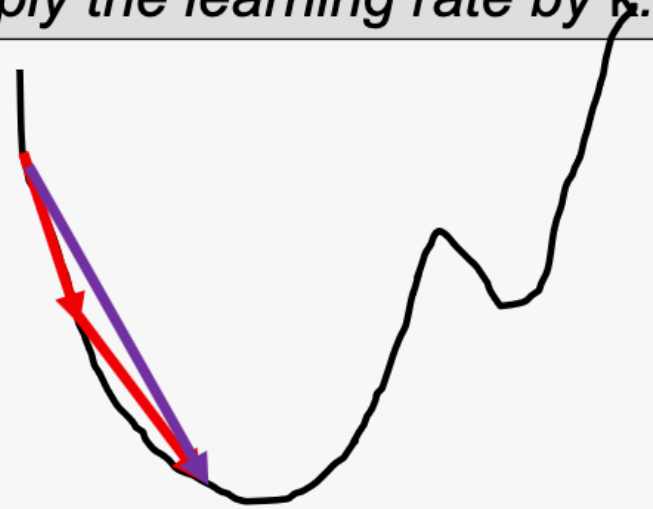
Horovod on Alpine (Basic Neural Net)

- California housing dataset
- Target variable is the median house value for California districts
- Simple regression neural network
- Code adapted from the Hands On Machine learning book chapter 10.

Important point(1)

Linear rate scaling rule

When the minibatch size is multiplied by k, *multiply the learning rate by* k.



Mini-batch stochastic Gradient Descent

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

Learning rate, lr Mini-batch

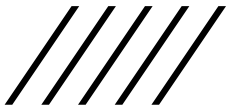
$$lr_{scale} = lr * nprocs$$

Typical practice / suggestion:

- keep local batch size per worker, i.e., increase the global batch size linearly
- Increase the learning rate proportionally

Actuality: need to adjust the batch size and learning rate at different scale

Source: ATPESC 2023, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-8-Talk-7-Zheng-Distributed-Deep-Learnng.pdf



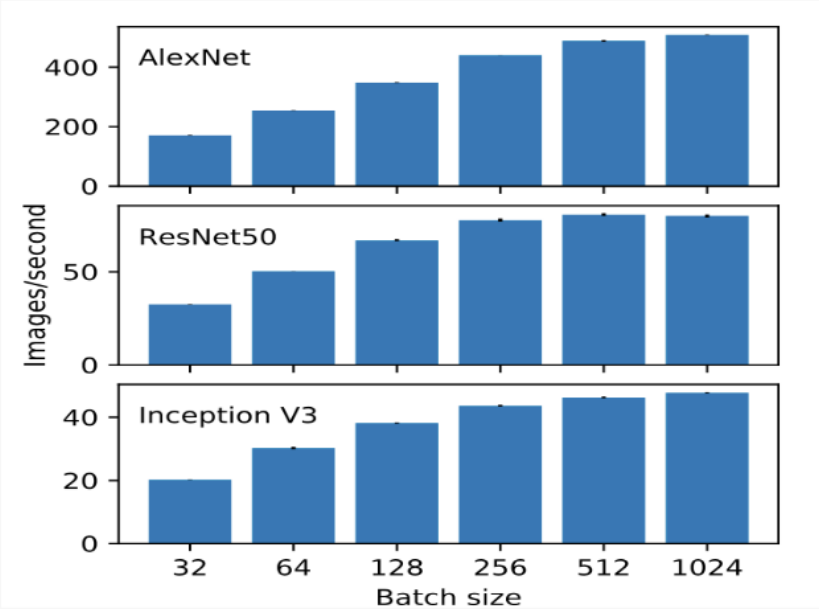
Important point(2)

Large minibatch training

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

Minibatch

- Option 1. Keeping the same global minibatch size with each worker processing B/N batch (strong scaling)
- Option 2. Increasing the global minibatch size by N times, so that each worker processes batches of size B (week scaling)



Per node throughput of different local batch size

H. Zheng, https://www.alcf.anl.gov/files/Zheng_SDL_ML_Frameworks_1.pdf

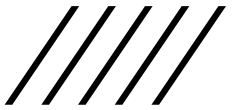
1. Decrease of local batch size reduces the per node throughput;
2. Increase of global minibatch size reduces the number of updates on each epoch (n=X/B); thus it increases the compute/communication ratio

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

Horovod on Alpine (Python script)

```
# This horovod test comes from the original chapter 10 of Hands on Machine learning
# https://github.com/ageron/handson-ml3/blob/main/10_neural_nets_with_keras.ipynb
print(4)
# Get the current timestamp
timestamp1 = time.time()

# Horovod: initialize Horovod.
hvd.init()

# Selecting the GPU devices on the node
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
    tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
```

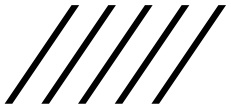
Timestamp to measure the duration



Horovod on Alpine (Python script)

```
# This horovod test comes from the original chapter 10 of Hands on Machine learning
# https://github.com/ageron/handson-ml3/blob/main/10_neural_nets_with_keras.ipynb
print(4)
# Get the current timestamp
timestamp1 = time.time()

# Horovod: initialize Horovod.
hvd.init()

# Selecting the GPU devices on the node
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
    tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
```

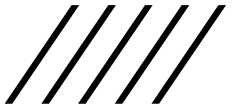
Initialize Horovod
and set multi GPUs



Horovod on Alpine (Python script)

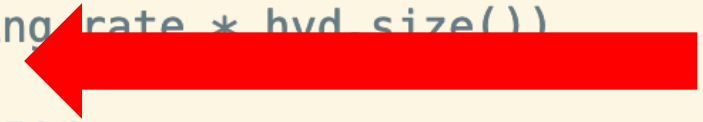
```
tf.random.set_seed(42)
norm_layer = tf.keras.layers.Normalization(input_shape=X_train.shape[1:])
model = tf.keras.Sequential([
    norm_layer,
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(1)
])
```

Simple Neural Net

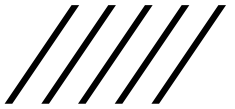


Horovod on Alpine (Python script)

```
# Tune the learning rate
learning_rate =  1e-3

#K.set_value(model.optimizer.learning_rate, learning_rate)
# Horovod: adjust learning rate based on number of GPUs.
opt = tf.keras.optimizers.Adadelta(learning_rate * hvd.size())

# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)# Horovod: add Horovod Distributed Optimizer.

model.compile(loss="mse", optimizer=opt, metrics=["RootMeanSquaredError"])
norm_layer.adapt(X_train)

# Defining batch size and epochs as a function of horovod
batch_size = 8
```



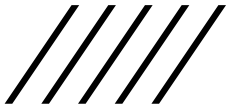
Learning rate is scaled by the number of GPUs



Horovod on Alpine (Python script)

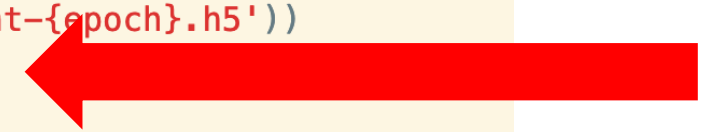
```
total_epochs_ = int(math.ceil(6.0 / hvd.size()))


callbacks = [
    # Horovod: broadcast initial variable states from rank 0 to all other processes.
    # This is necessary to ensure consistent initialization of all workers when
    # training is started with random weights or restored from a checkpoint.
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
]

# Horovod: save checkpoints only on worker 0 to prevent other workers from corrupting them.
if hvd.rank() == 0:
    callbacks.append(tf.keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))


# Model fitting across multiple GPUs
model.fit(X_train,
          y_train,
          epoch=total_epochs_,
          verbose=1,
          callbacks=callbacks,
          validation_data=(X_valid, y_valid))
```

We scale the number of epoch by the number of GPUs

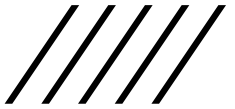


Horovod on Alpine (Python script)

```
total_epochs_ = int(math.ceil(6.0 / hvd.size()))


callbacks = [
    # Horovod: broadcast initial variable states from rank 0 to all other processes.
    # This is necessary to ensure consistent initialization of all workers when
    # training is started with random weights or restored from a checkpoint.
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
]

# Horovod: save checkpoints only on worker 0 to prevent other workers from corrupting them.
if hvd.rank() == 0:
    callbacks.append(tf.keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))


# Model fitting across multiple GPUs
model.fit(X_train,
          y_train,
          epoch=total_epochs_,
          verbose=1,
          callbacks=callbacks,
          validation_data=(X_valid, y_valid))
```



checkpointing

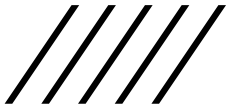


Run slurm script

- sbatch **horovod_mpi.sh**
- Check the nodes where the job is running

```
e.org@login-ci2 (                                    ]$ squeue --me
 JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
481362     aa100 horovod_ kfotso@x  R       2:34      2 c3gpu-a9-u31-1,c3gpu-a
479946  acompile acompile kfotso@x  R    2:03:48      1 c3cpu-c9-u5-2
481778 atesting_ sinterac kfotso@x  R       7:56      1 c3gpu-a9-u29-1
e.org@login-ci2                                     ]$ ssh c3gpu-a9-u29-1^C
```

We have a GPU node that we can ssh to: c3gpu-a9-u29-1



GPU monitoring

- ssh to a that node on a different screen and run:

nvidia-smi -l

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100 80G...  On   | 00000000:25:00.0 Off |                    0 |
| N/A   33C    P0    80W / 300W |   1869MiB / 81920MiB |     26%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100 80G...  On   | 00000000:81:00.0 Off |                    0 |
| N/A   32C    P0    81W / 300W |   1869MiB / 81920MiB |     25%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    569301      C   python                           1866MiB |
|    1   N/A  N/A    569302      C   python                           1866MiB |
+-----------------------------------------------------------------------------+
```

We can confirm that we are using 2 GPUs!!!

GPU monitoring

- ssh to a that node on a different screen and run:

nvidia-smi -l

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100 80G...  On   | 00000000:25:00.0 Off |                    0 |
| N/A   33C    P0    80W / 300W |   1869MiB / 81920MiB |     26%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100 80G...  On   | 00000000:81:00.0 Off |                    0 |
| N/A   32C    P0    81W / 300W |   1869MiB / 81920MiB |     25%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    569301      C   python                           1866MiB |
|    1   N/A  N/A    569302      C   python                           1866MiB |
+-----------------------------------------------------------------------------+
```

Percentage GPU usability. That is the portion of time 1 or more kernels were used

GPU monitoring

- ssh to a that node on a different screen and run:

nvidia-smi -l

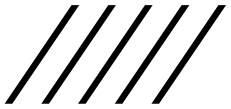
```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100 80G...  On   | 00000000:25:00.0 Off |                    0 |
| N/A   33C    P0    80W / 300W |   1869MiB / 81920MiB |     26%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100 80G...  On   | 00000000:81:00.0 Off |                    0 |
| N/A   32C    P0    81W / 300W |   1869MiB / 81920MiB |     25%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    569301      C   python                           1866MiB |
|    1   N/A  N/A    569302      C   python                           1866MiB |
+-----------------------------------------------------------------------------+
```

How much memory is used.

GPU monitoring

- nvidia-smi has more parameters that can be used. Cheatsheet here: https://www.docs.arc.vt.edu/usage/gpumon.html

```
[kfotso@xsede.org@c3gpu-a9-u29-1 ~]$ nvidia-smi --query-gpu=timestamp,name,pci.bus_id,driver_version,temperatur
e.gpu,utilization.gpu,utilization.memory --format=csv -l 5
timestamp, name, pci.bus_id, driver_version, temperature.gpu, utilization.gpu [%], utilization.memory [%]
2023/10/23 21:42:45.228, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 32, 0 %, 0 %
2023/10/23 21:42:45.230, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 31, 0 %, 0 %
2023/10/23 21:42:50.233, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 33, 7 %, 0 %
2023/10/23 21:42:50.234, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 32, 5 %, 0 %
2023/10/23 21:42:55.236, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 34, 25 %, 2 %
2023/10/23 21:42:55.237, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 32, 23 %, 1 %
2023/10/23 21:43:00.238, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 34, 25 %, 2 %
2023/10/23 21:43:00.240, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 33, 25 %, 1 %
2023/10/23 21:43:05.241, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 34, 25 %, 2 %
2023/10/23 21:43:05.243, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 33, 24 %, 1 %
2023/10/23 21:43:10.244, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 35, 26 %, 2 %
2023/10/23 21:43:10.246, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 33, 25 %, 2 %
2023/10/23 21:43:15.247, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 35, 26 %, 2 %
2023/10/23 21:43:15.248, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 33, 26 %, 2 %
```



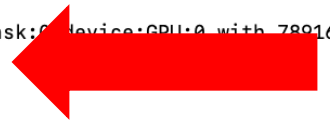
Slurm job (2 GPUs)

```
tput=horovod_profile horovodrun -np 2 -H localhost:2 horovod_cnn_mnist.py
GPU 0: General Metrics for NVIDIA GA100 (any frequency)
GPU 1: General Metrics for NVIDIA GA100 (any frequency)
2023-10-23 21:47:46.810582: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critic
al operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,0]<stderr>:2023-10-23 21:47:52.457092: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,0]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stderr>:2023-10-23 21:47:52.485600: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,1]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stdout>:4
[1,0]<stdout>:4
[1,0]<stderr>:2023-10-23 21:47:57.693575: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 0, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:25:00.0, compute capability: 8.0
[1,1]<stderr>:2023-10-23 21:47:57.708825: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 1, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:81:00.0, compute capability: 8.0
[1,0]<stdout>:Epoch 1/3
[1,1]<stdout>:Epoch 1/3
[1,0]<stderr>:2023-10-23 21:48:01.104191: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,1]<stderr>:2023-10-23 21:48:01.117719: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,0]<stderr>:2023-10-23 21:48:01.793663: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.813954: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.846349: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f6a85723010 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,1]<stderr>:2023-10-23 21:48:01.846381: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,0]<stderr>:2023-10-23 21:48:01.854430: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f0deed4b800 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,0]<stderr>:2023-10-23 21:48:01.854486: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,1]<stderr>:2023-10-23 21:48:01.857037: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,0]<stderr>:2023-10-23 21:48:01.861230: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,1]<stderr>:2023-10-23 21:48:02.027518: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
[1,0]<stderr>:2023-10-23 21:48:02.034153: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
```

The profiling sees the 2 GPUs.

Slurm job (2 GPUs)

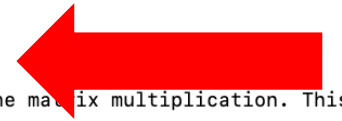
```
tput=horovod_profile horovodrun -np 2 -H localhost:2 horovod_cnn_mnist.py
GPU 0: General Metrics for NVIDIA GA100 (any frequency)
GPU 1: General Metrics for NVIDIA GA100 (any frequency)
2023-10-23 21:47:46.810582: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critic
al operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,0]<stderr>:2023-10-23 21:47:52.457092: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,0]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stderr>:2023-10-23 21:47:52.485600: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,1]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stdout>:4
[1,0]<stdout>:4
[1,0]<stderr>:2023-10-23 21:47:57.693575: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 0, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:25:00.0, compute capability: 8.0
[1,1]<stderr>:2023-10-23 21:47:57.708825: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 1, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:81:00.0, compute capability: 8.0
[1,0]<stdout>:Epoch 1/3
[1,1]<stdout>:Epoch 1/3
[1,0]<stderr>:2023-10-23 21:48:01.104191: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,1]<stderr>:2023-10-23 21:48:01.117719: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,0]<stderr>:2023-10-23 21:48:01.793663: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.813954: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.846349: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f6a85723010 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,1]<stderr>:2023-10-23 21:48:01.846381: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,0]<stderr>:2023-10-23 21:48:01.854430: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f0deed4b800 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,0]<stderr>:2023-10-23 21:48:01.854486: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,1]<stderr>:2023-10-23 21:48:01.857037: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,0]<stderr>:2023-10-23 21:48:01.861230: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,1]<stderr>:2023-10-23 21:48:02.027518: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
[1,0]<stderr>:2023-10-23 21:48:02.034153: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
```



2 GPU got assigned

Slurm job (2 GPUs)

```
tput=horovod_profile horovodrun -np 2 -H localhost:2 horovod_cnn_mnist.py
GPU 0: General Metrics for NVIDIA GA100 (any frequency)
GPU 1: General Metrics for NVIDIA GA100 (any frequency)
2023-10-23 21:47:46.810582: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critic
al operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,0]<stderr>:2023-10-23 21:47:52.457092: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,0]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stderr>:2023-10-23 21:47:52.485600: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,1]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stdout>:4
[1,0]<stdout>:4
[1,0]<stderr>:2023-10-23 21:47:57.693575: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 0, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:25:00.0, compute capability: 8.0
[1,1]<stderr>:2023-10-23 21:47:57.708825: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 1, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:81:00.0, compute capability: 8.0
[1,0]<stdout>:Epoch 1/3
[1,1]<stdout>:Epoch 1/3
[1,0]<stderr>:2023-10-23 21:48:01.104191: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,1]<stderr>:2023-10-23 21:48:01.117719: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,0]<stderr>:2023-10-23 21:48:01.793663: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.813954: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.846349: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f6a85723010 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,1]<stderr>:2023-10-23 21:48:01.846381: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,0]<stderr>:2023-10-23 21:48:01.854430: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f0deed4b800 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,0]<stderr>:2023-10-23 21:48:01.854486: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,1]<stderr>:2023-10-23 21:48:01.857037: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,0]<stderr>:2023-10-23 21:48:01.861230: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,1]<stderr>:2023-10-23 21:48:02.027518: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
[1,0]<stderr>:2023-10-23 21:48:02.034153: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
```



CUDNN gets loaded for each

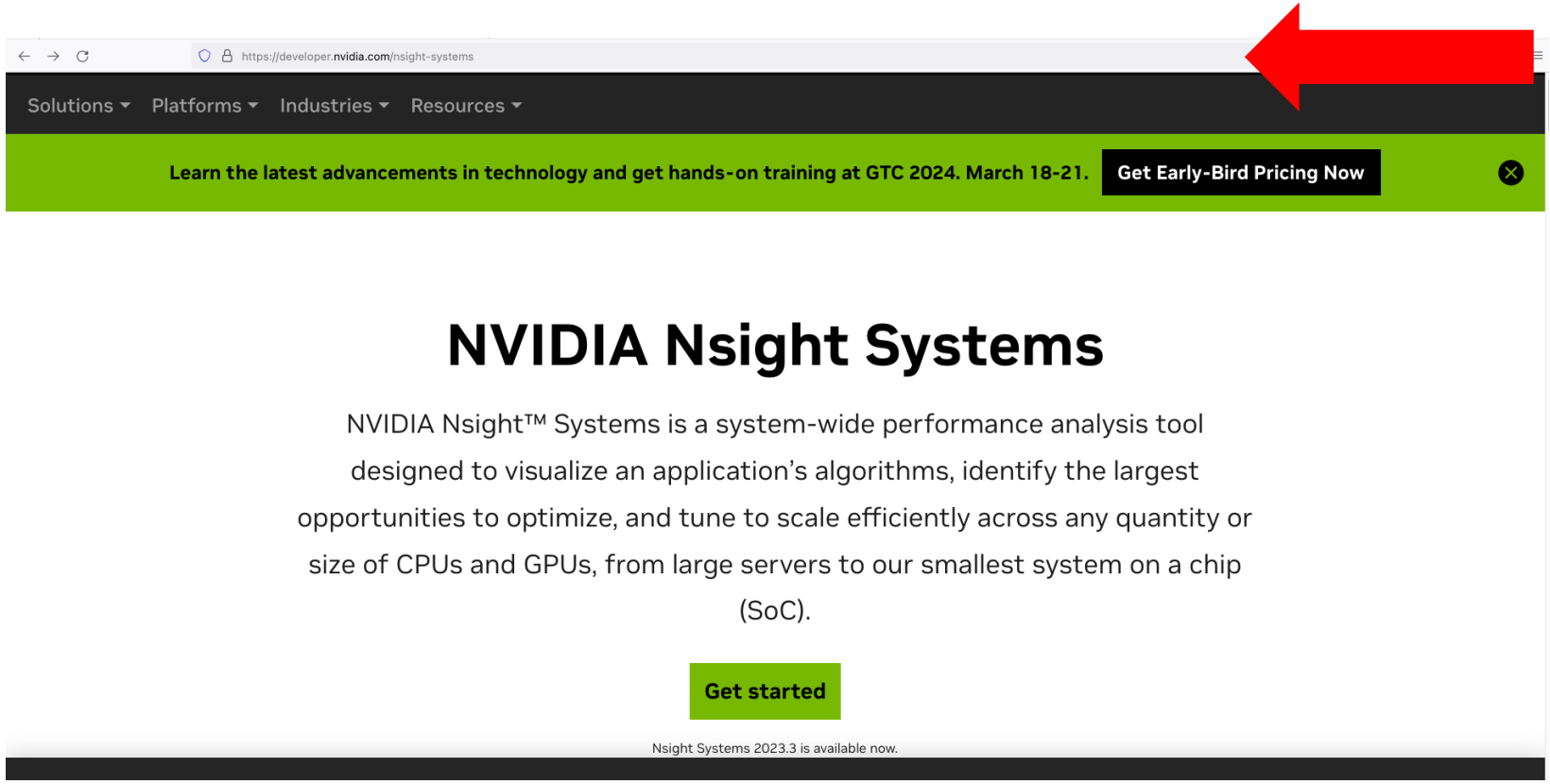
Slurm job (2 GPUs)

```
tput=horovod_profile horovodrun -np 2 -H localhost:2 horovod_cnn_mnist.py
GPU 0: General Metrics for NVIDIA GA100 (any frequency)
GPU 1: General Metrics for NVIDIA GA100 (any frequency)
2023-10-23 21:47:46.810582: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critic
al operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,0]<stderr>:2023-10-23 21:47:52.457092: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,0]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stderr>:2023-10-23 21:47:52.485600: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,1]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stdout>:4
[1,0]<stdout>:4
[1,0]<stderr>:2023-10-23 21:47:57.693575: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 0, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:25:00.0, compute capability: 8.0
[1,1]<stderr>:2023-10-23 21:47:57.708825: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 1, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:81:00.0, compute capability: 8.0
[1,0]<stdout>:Epoch 1/3
[1,1]<stdout>:Epoch 1/3
[1,0]<stderr>:2023-10-23 21:48:01.104191: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,1]<stderr>:2023-10-23 21:48:01.117719: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,0]<stderr>:2023-10-23 21:48:01.793663: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.813954: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.846349: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f6a85723010 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,1]<stderr>:2023-10-23 21:48:01.846381: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute C
[1,0]<stderr>:2023-10-23 21:48:01.854430: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f0deed4b800 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,0]<stderr>:2023-10-23 21:48:01.854486: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,1]<stderr>:2023-10-23 21:48:01.857037: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,0]<stderr>:2023-10-23 21:48:01.861230: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,1]<stderr>:2023-10-23 21:48:02.027518: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
[1,0]<stderr>:2023-10-23 21:48:02.034153: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
```

XLA opt is initialize
and the stream is
executed

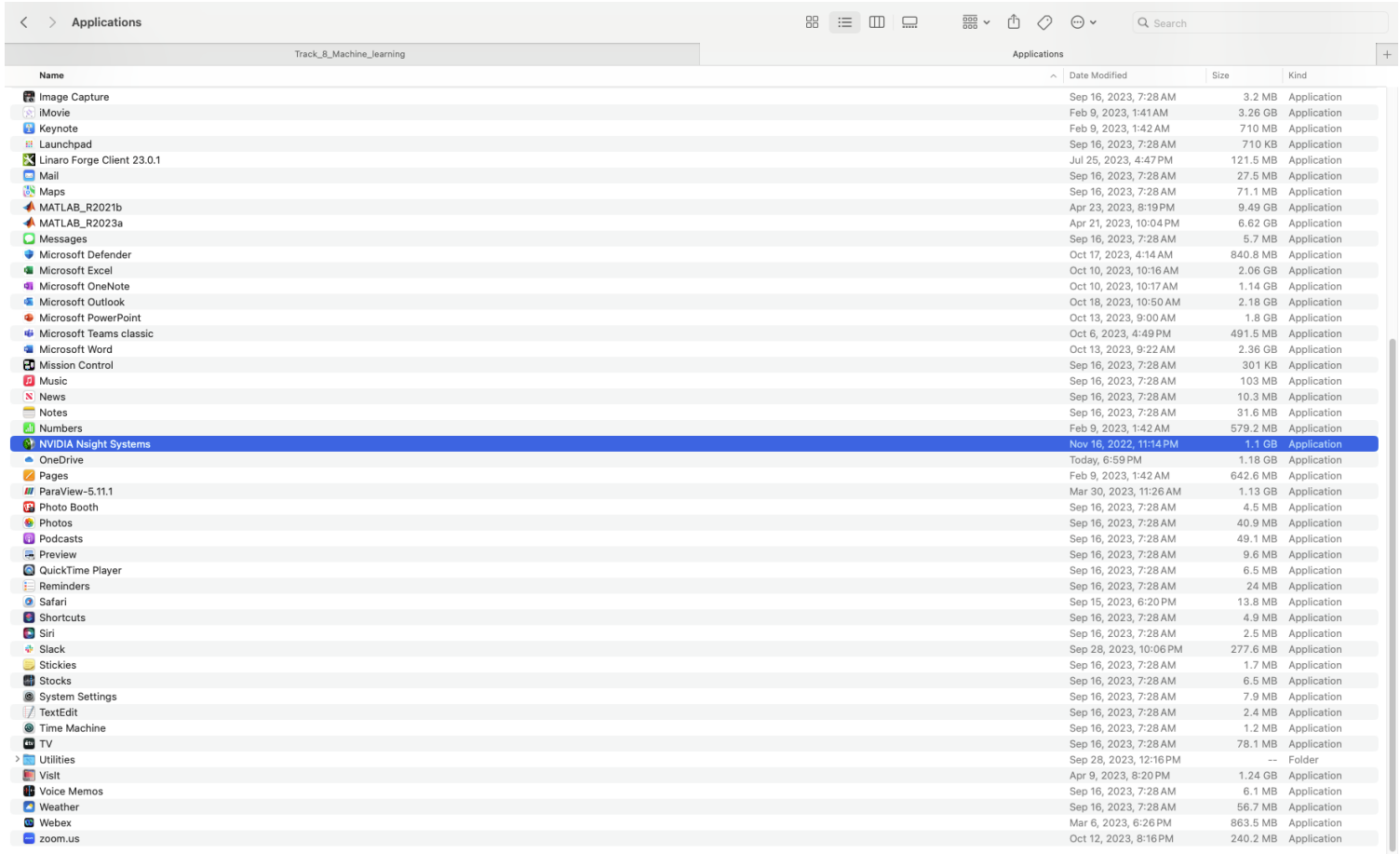
NVIDIA Nsight

- Download the NVIDIA Nsight system client tool to your local computer



https://developer.nvidia.com/nsight-systems

NVIDIA Nsight (2)

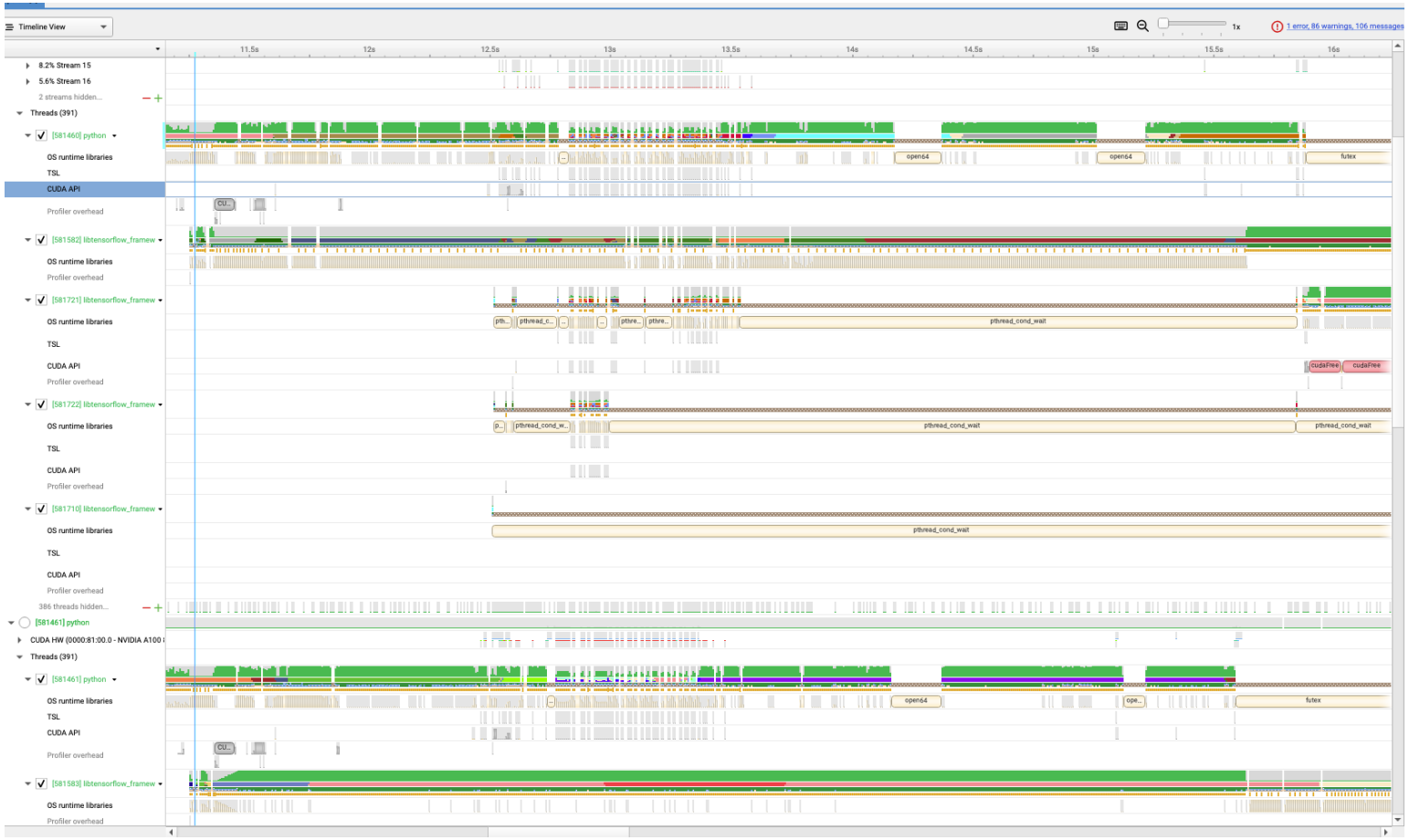


NVIDIA Nsight (3)

Click on open

NVIDIA Nsight (3)

NVIDIA Nsight (4)



NVIDIA Nsight (5)

File View Tools Help

Project Explorer

- Project 1
- Project 2
- rdf_doconcurrent_multicore
- rdf_doconcurrent_gpu
- minicfdstdpar_profile.qdrep
- minicfdstdpar_profile.qdrep
- nsys-report-2eed
- nsys-report-d898
- report1

report1

Diagnostics Summary

Messages

	Source	Process ID	Time	Description
ⓘ	Daemon		-00:00.017	Intel(c) Last Branch Record (LBR) backtrace sampling is not supported on this target.
ⓘ	Daemon		-00:00.017	Dwarf backtraces collected.
ⓘ	Daemon		-00:00.017	Hardware event 'CPU Cycles', with sampling period 1000000, used to trigger process-tree CPU IP sample collection.
ⓘ	Daemon		-00:00.000	1 CPU IP samples collected for every CPU IP backtrace collected.
ⓘ	Analysis		00:00.000	Profiling has started.
ⓘ	Daemon	581320	00:00.000	Process was launched by the profiler, see /scratch/alpine/kfotso@xsede.org/nvidia/nsight_systems/quadd_session_10581301/streams/pid_581320_stdout.log and stderr.log for program output
ⓘ	Injection	581334	00:00.328	Common injection library initialized successfully.
ⓘ	Injection	581334	00:00.363	OS runtime libraries injection initialized successfully.
ⓘ	Injection	581334	00:00.374	OpenGL injection initialized successfully.
ⓘ	Injection	581408	00:05.549	Common injection library initialized successfully.
ⓘ	Injection	581409	00:05.552	Common injection library initialized successfully.
ⓘ	Injection	581408	00:05.582	OS runtime libraries injection initialized successfully.
ⓘ	Injection	581408	00:05.593	OpenGL injection initialized successfully.
ⓘ	Injection	581409	00:05.735	OS runtime libraries injection initialized successfully.
ⓘ	Injection	581409	00:05.746	OpenGL injection initialized successfully.
ⓘ	Injection	581418	00:05.917	Common injection library initialized successfully.
ⓘ	Injection	581418	00:05.995	OS runtime libraries injection initialized successfully.
ⓘ	Injection	581418	00:06.006	OpenGL injection initialized successfully.
ⓘ	Injection	581424	00:06.224	Common injection library initialized successfully.
ⓘ	Injection	581424	00:06.259	OS runtime libraries injection initialized successfully.
ⓘ	Injection	581424	00:06.283	OpenGL injection initialized successfully.
ⓘ	Injection	581320	00:06.353	Common injection library initialized successfully.
ⓘ	Injection	581320	00:06.391	OS runtime libraries injection initialized successfully.
ⓘ	Injection	581320	00:06.415	OpenGL injection initialized successfully.
ⓘ	Injection	581460	00:06.559	Common injection library initialized successfully.
ⓘ	Injection	581461	00:06.568	Common injection library initialized successfully.
ⓘ	Injection	581460	00:06.594	OS runtime libraries injection initialized successfully.
ⓘ	Injection	581460	00:06.605	OpenGL injection initialized successfully.
ⓘ	Injection	581461	00:06.605	OS runtime libraries injection initialized successfully.
ⓘ	Injection	581461	00:06.616	OpenGL injection initialized successfully.
ⓘ	Injection	581474	00:06.773	Common injection library initialized successfully.

It list the events that happen

Horvod code snipet Pytorch

```
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
                    transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

Transforms to preprocess data

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

Horvod code snipet Pytorch

```
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
                    transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

Loading the data in a distributed fashion

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

Horvod code snipet Pytorch

```
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
                    transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

Broadcast params

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

Horvod code snipet Pytorch

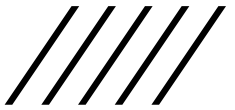
```
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
                    transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

Wrap optimizer in hvd optimizer

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

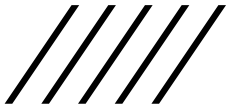
To be done next week

- Workshop slides will be uploaded to the Github page
- Workshop ppt and videos will be shared with users
- Installation recipe will be uploaded on the Github page



Announcements for next semester

- Part 2 of the Horovod workshop will be given sometimes in January/February next year.
- It will include how to do profiling directly with Tensorflow/Tensorflow board.
- It will include an attempt to use Horovod with AMD GPUs.
- It will include GridSearch params optimization distribution across multiple nodes/GPUs.



Questions?