Large Scale Distributed Deep Learning on Supercomputers

NRIS

Norwegian research infrastructure services

Hicham Agueny, PhD
Scientific Computing Group
Info.Tech. department, UiB/NRIS

Why distributed training ?

- Memory limitations presents a challenge when large models (or datasets) exceed a single GPU memory capacity
- Constraints of single GPU memory restrict smaller batch sizes affecting both performance and convergence
- Training deep learning models on massive datasets remains a challenge and necessitates the utilization of **distributed training frameworks** optimized for large **High-Performance Computing (HPC) systems**.

Motivation

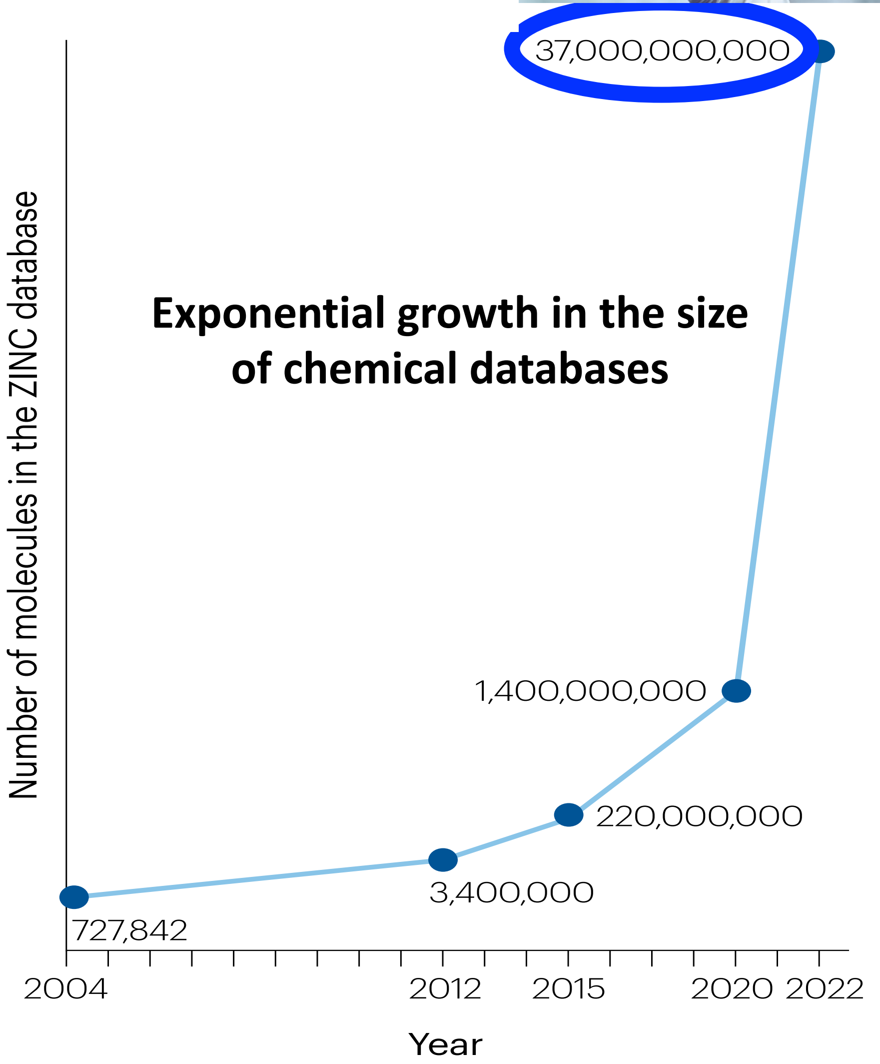
nature reviews drug discovery

Perspective | Published: 08 December 2023

Integrating QSAR modelling and deep learning in drug discovery: the emergence of deep QSAR

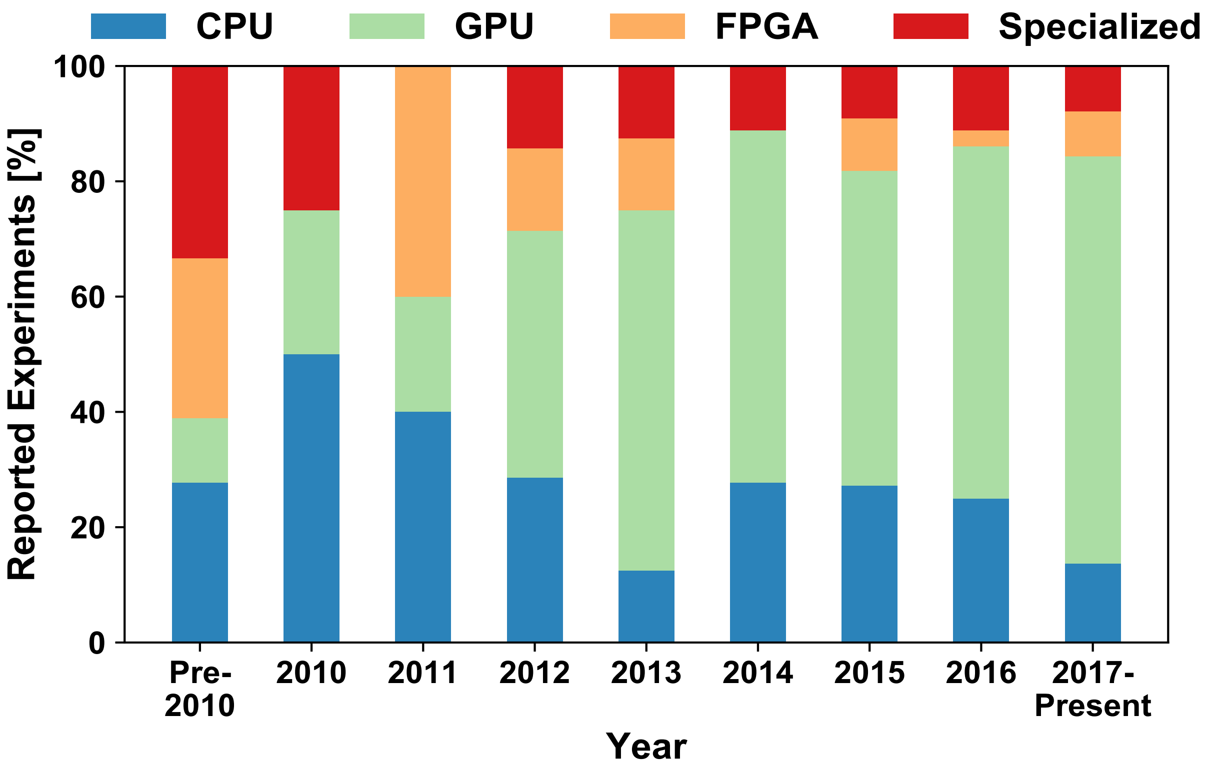
to screen 40 billion molecules (combining ZINC15 and Enamine REAL Space databases) against SARS-CoV-2 M^{pro} (ref. 95). The consecutive deep docking runs with the five programmes took approximately 90 days of computing on 250 GPUs and 640 CPU cores and reduced the

with GPUs, and the resulting GPU-AutoDock method was used on the 27,000 GPUs of the Summit supercomputer to process the Enamine REAL library against SARS-CoV-2 M^{pro} in 1 day[110]. In another large-scale

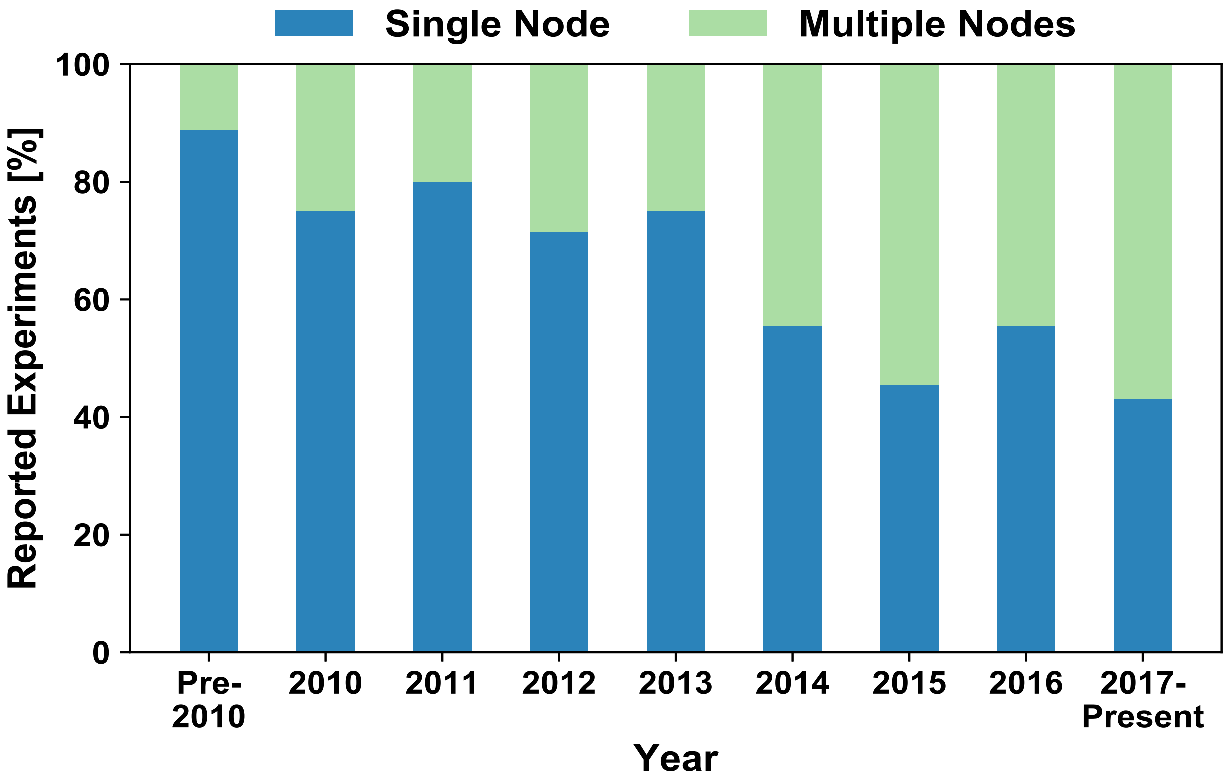


Survey: Hardware architectures for Machine Learning

Out of the 252 reviewed papers, 159 papers present empirical results and provide details about their hardware setup.



(a) Hardware Architectures



(b) Training with Single vs. Multiple Nodes

Tal Ben-Nun and Torsten Hoefler. 2019. ACM Comput. Surv. 52, 4, Article 65 (August 2019), 43 pages

Learning Outcomes

- Get an overview of the architecture of compute nodes in LUMI-G system.
- Understand conceptual difference between model parallelism and data parallelism.
- Understand conceptual difference between data parallelism in a centralised and

  a decentralised architecture in Deep Neural Network.
- Gain insight into the concept of Horovod for distributed deep learning.
- Implement Horovod-TensorFlow through a small example.

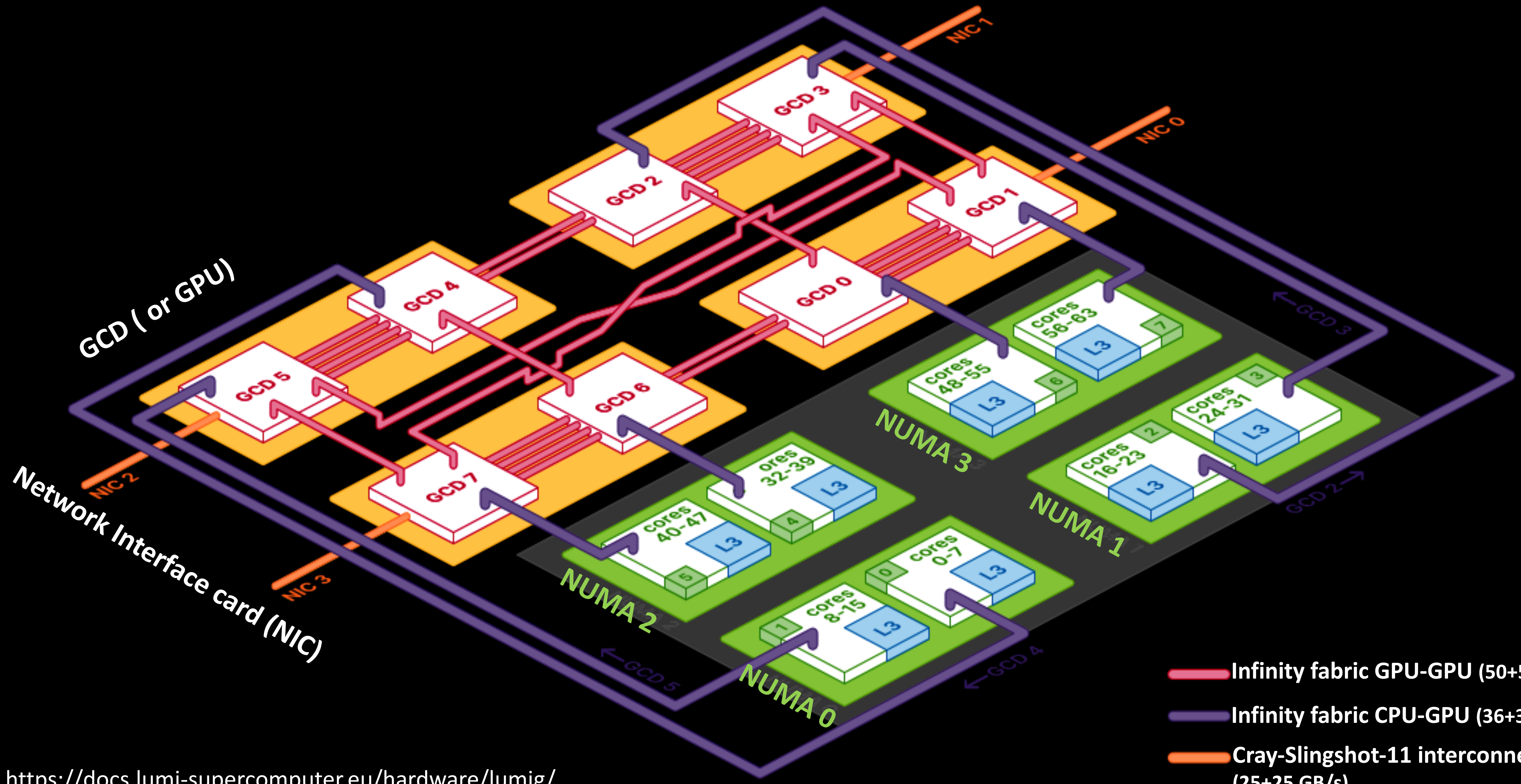
Supercomputer LUMI

LUMI-G

- 2928 nodes
- 1 AMD EPYC 7A53 64-Core CPU
- 4 AMD MI250X GPUs
    - 2 Graphics Compute Dies (GCDs) per GPU
    - 128 GB HBM2e per GPU
- HPE Slingshot interconnect
- Each GPU node features four 200 Gbit/s network interconnect cards, i.e. has 800 Gbit/s injection bandwidth.
- 512 GB DDR4 memory

Taken from LUMI Hackathon Introduction 22/11/2023

Architecture of a LUMI-G Compute node



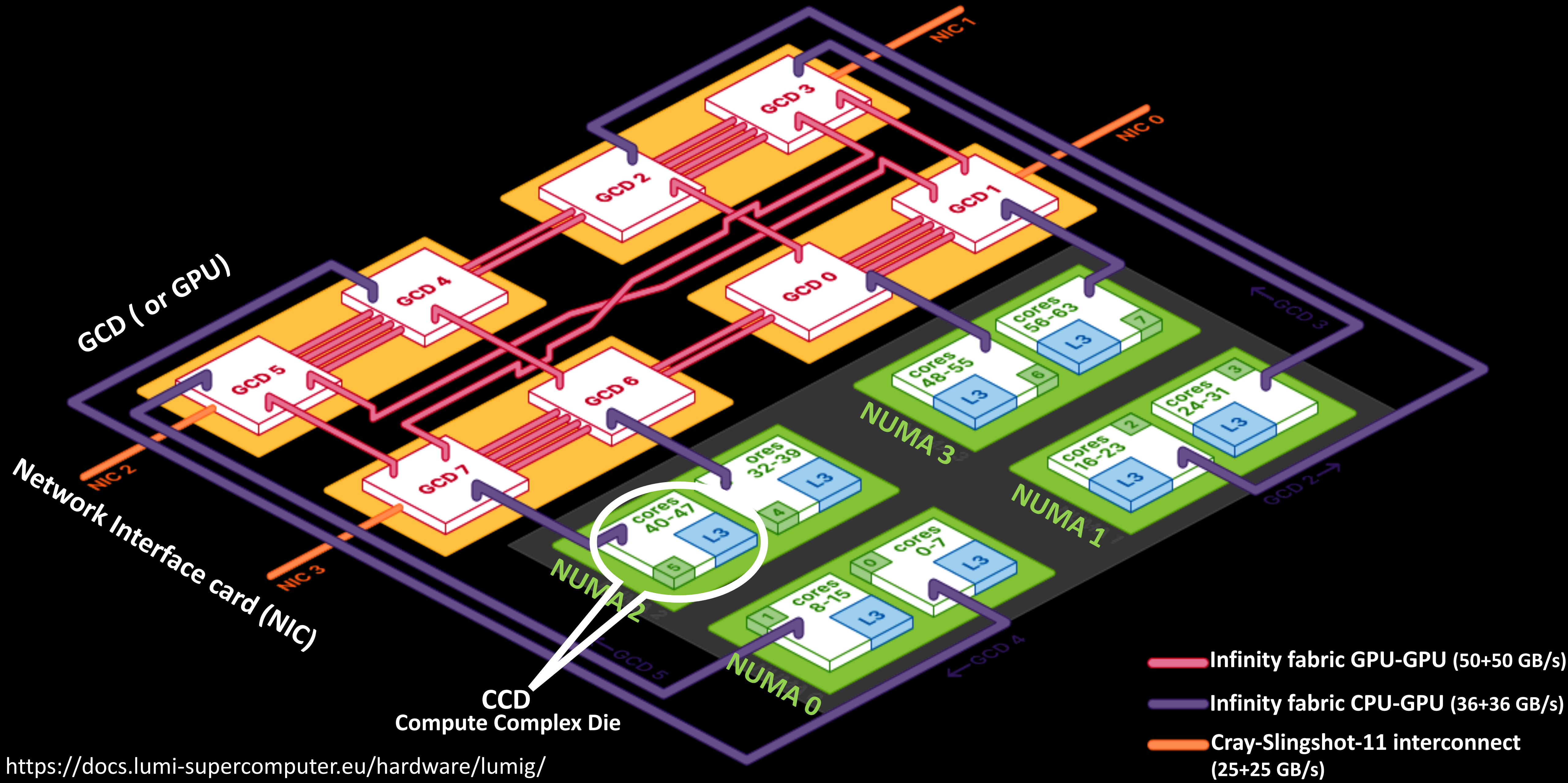
Infinity fabric GPU-GPU (50+50 GB/s)

Infinity fabric CPU-GPU (36+36 GB/s)

Cray-Slingshot-11 interconnect (25+25 GB/s)

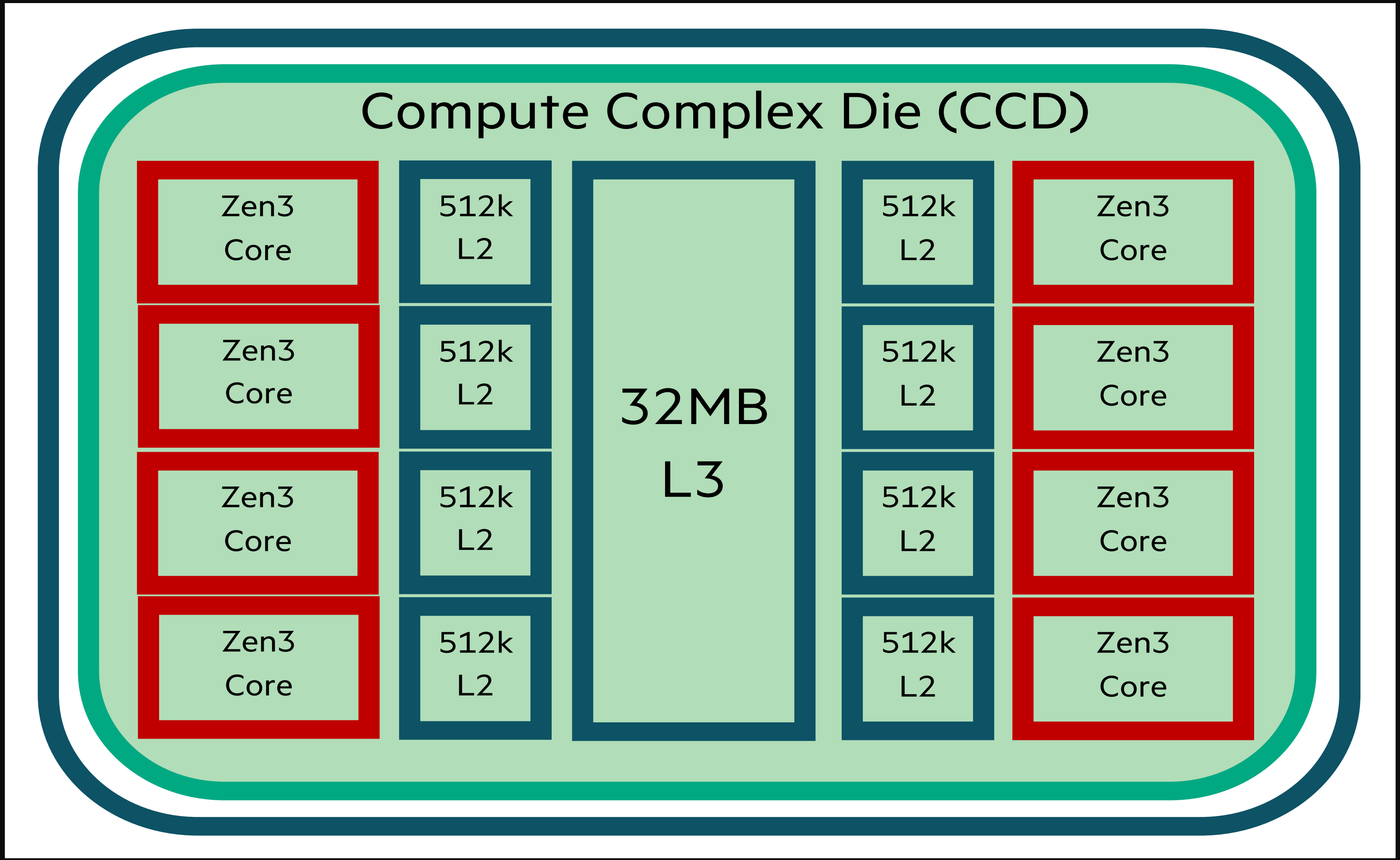
https://docs.lumi-supercomputer.eu/hardware/lumig/

Architecture of a LUMI-G Compute node



https://docs.lumi-supercomputer.eu/hardware/lumig/

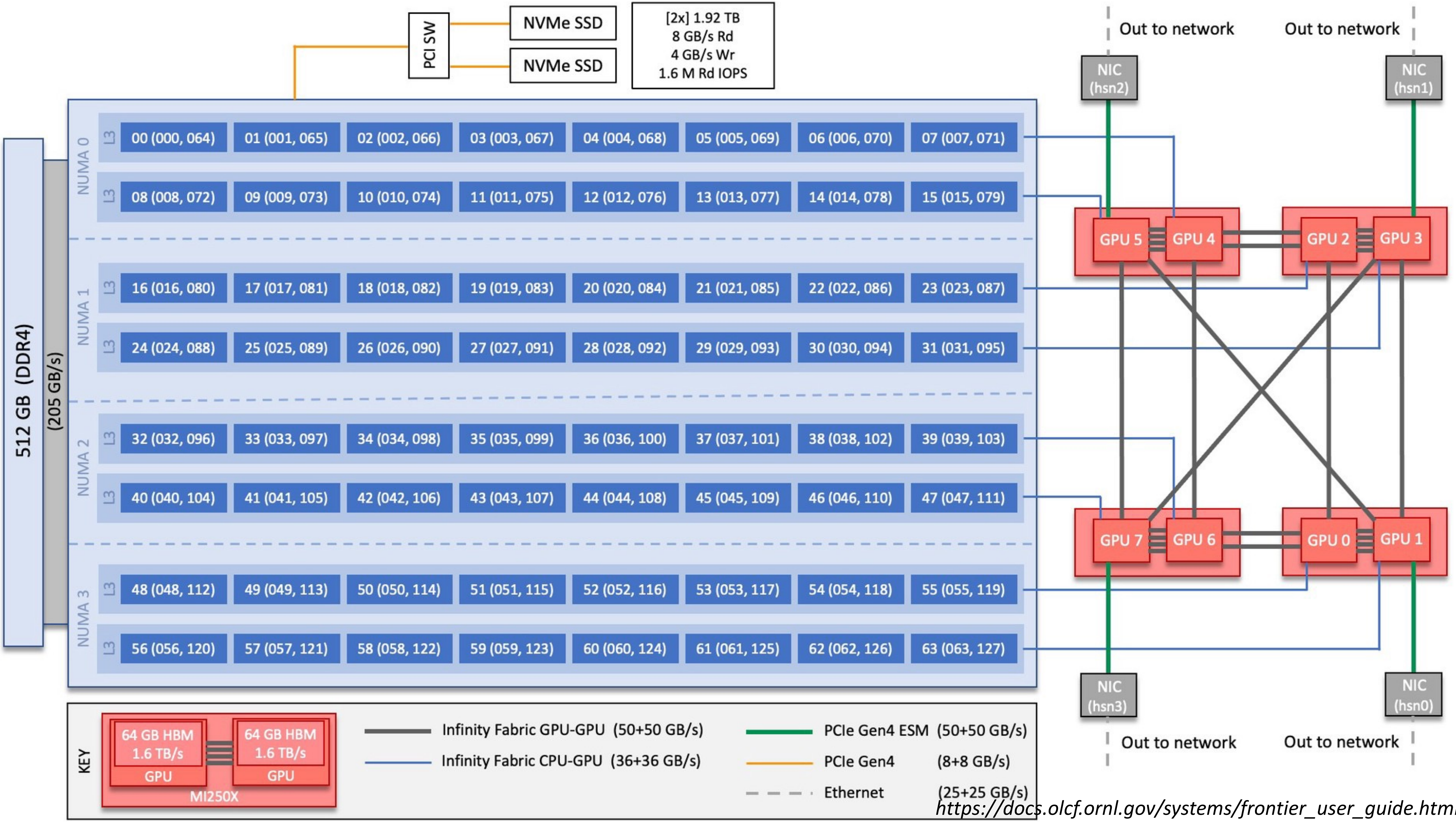
Compute Complex Die (CCD): AMD EPYC Zen3 Trento Architecture



Compute Complex Dies
Host cores & L2/L3 cache
- L1 cache 32 kB/core
- L2 cache 512 kB/core
- L3 cache 32 MB/8-cores

Infinity fabric CPU-GPU (36+36 GB/s)
Cray-Slingshot-11 interconnect
(25+25 GB/s)

Taken from LUMI Hackathon Introduction 22/11/2023

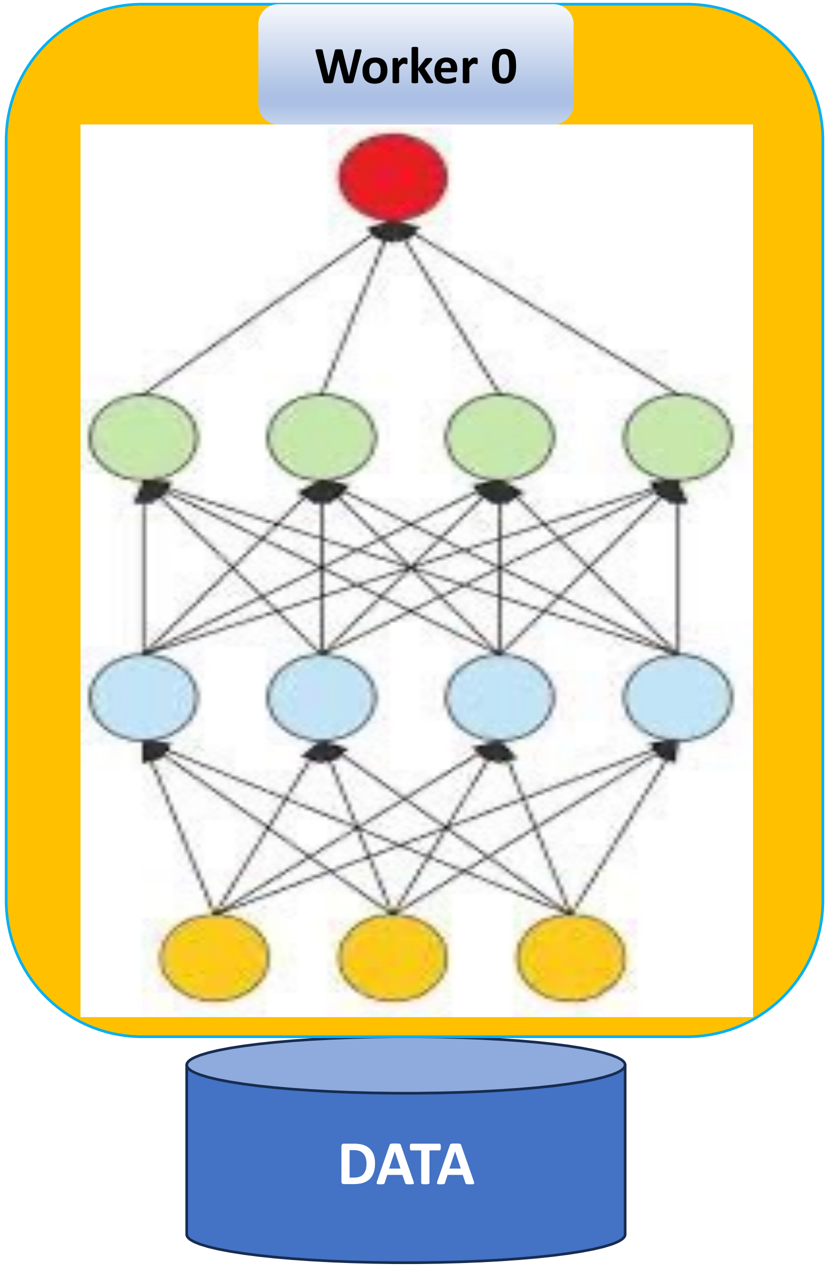


https://docs.olcf.ornl.gov/systems/frontier_user_guide.html

Concept of Distributed DNN Training

- **Model Parallelism**
- **Data Parallelism**
  - **Centralized DNN**
  - **Decentralized DNN**

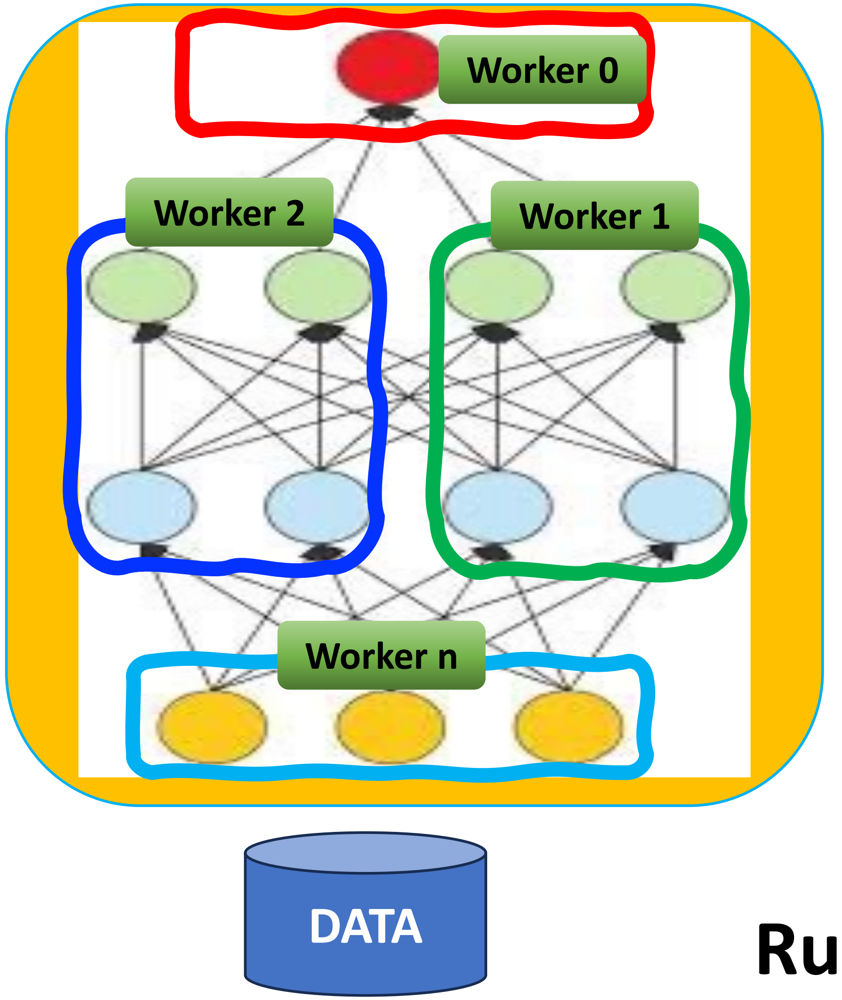
DNN Training on a single worker



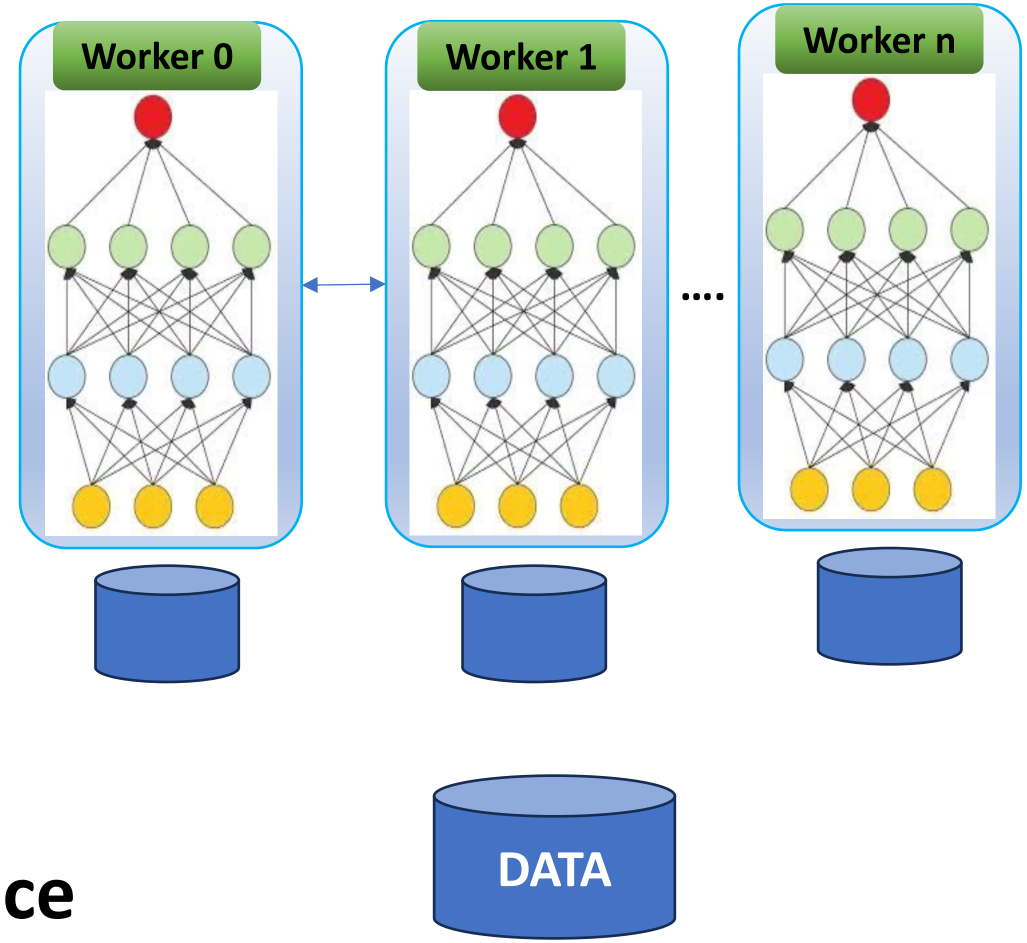
Distributed DNN Training: Parallelism Schemes

Parallelism in DNN: Training **large DNN models** or **large dataset** on **multiple Workers** in a shared or distributed environment.

Model Parallelism



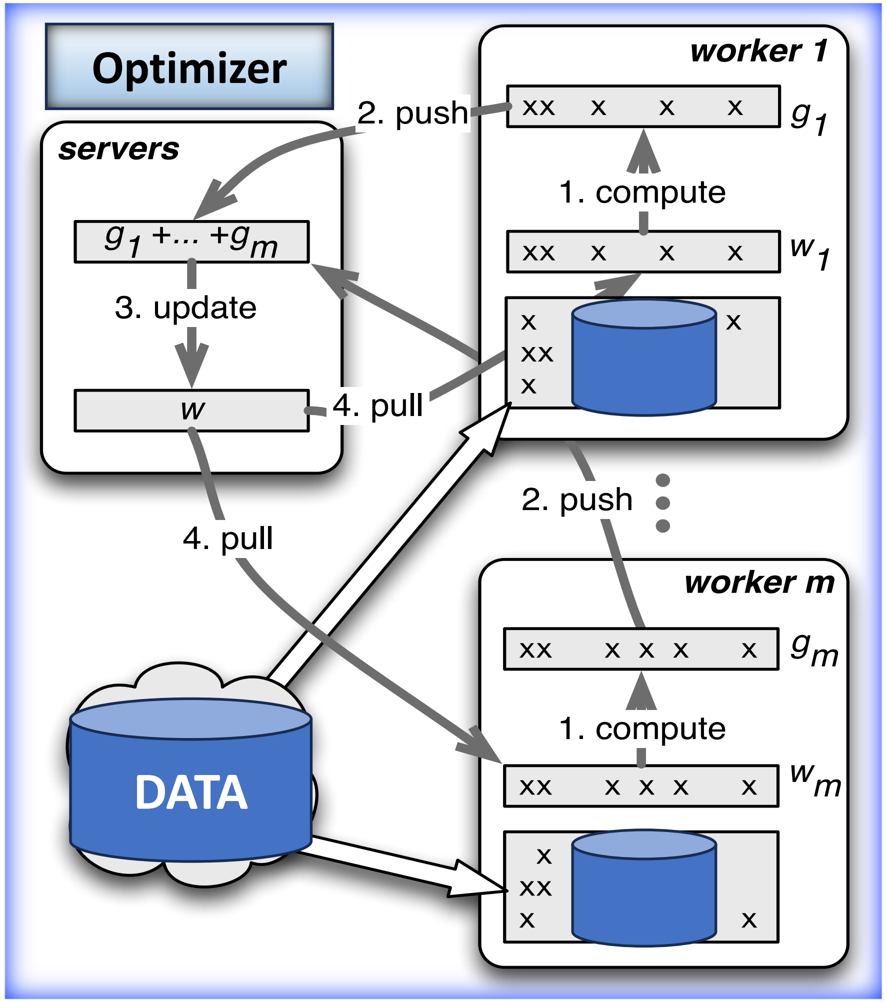
Data Parallelism



Simplicity
&
Scalability
&
Runtime Performance

Centralized Distributed DNN Training:

$$\nabla_{\mathbf{w}} f(\mathbf{w}; X) = \frac{1}{N}\left(\frac{1}{B}\sum_{i=1}^{B}\nabla_{\mathbf{w}}\ell(\mathbf{w},\mathbf{x}_i) + \frac{1}{B}\sum_{i=B+1}^{B\times 2}\nabla_{\mathbf{w}}\ell(\mathbf{w},\mathbf{x}_i) + \ldots + \frac{1}{B}\sum_{i=B\times(N-1)+1}^{B\times N}\nabla_{\mathbf{w}}\ell(\mathbf{w},\mathbf{x}_i)\right)$$

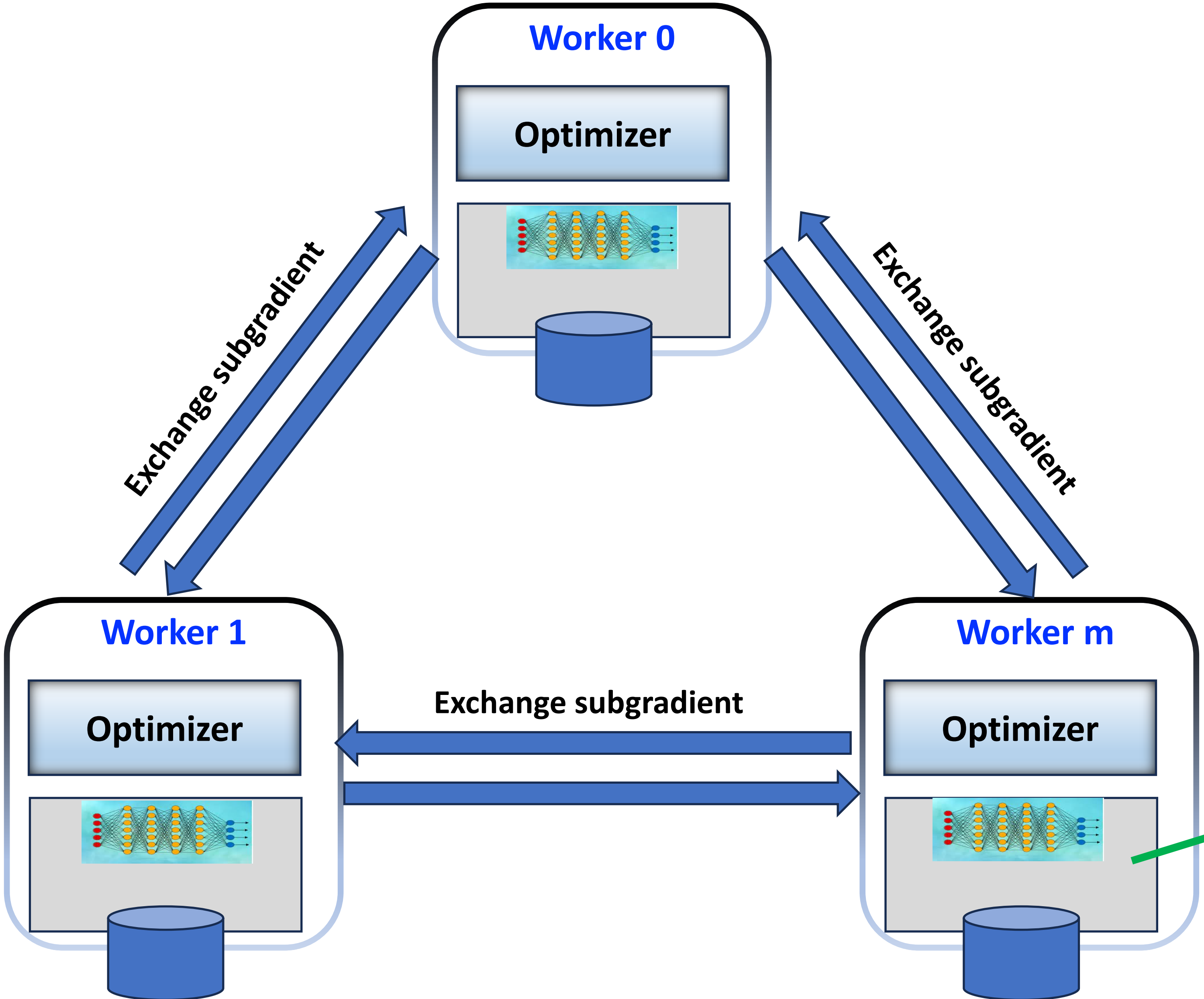


- Parameter servers collect subgradiants, compute gradiant and update weights
- Each worker pulls weights from server computes subgradient and sends its value back to the server

- No direct communication between workers
- All workers directly communicate with servers
- Overhead communication when increasing nbr of workers. **The scaling is poor.**

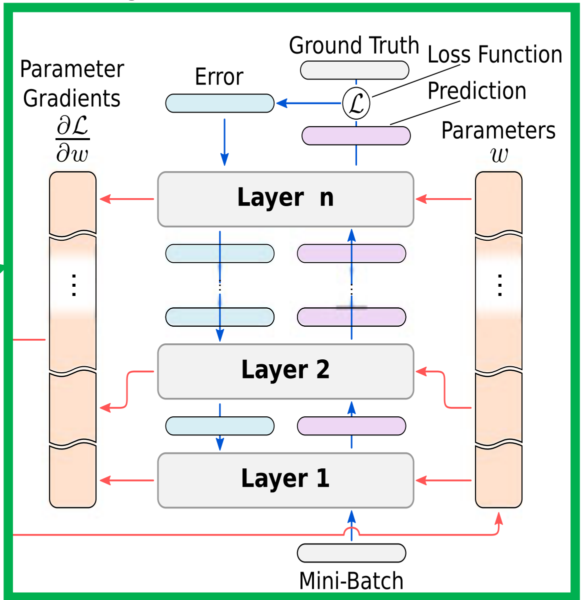
M. Li et al. (Baidu, Google) Scaling distributed machine learning with the parameter server, Proc. 11th USENIX Conf. Oper. Syst. Design Implement., 2014, pp. 583–598.

Decentralized **Distributed DNN Training:**



- No parameter servers
- Each worker computes (sub)gradient and exchange its value with the neighbooring workers (forming a Ring)
- Each worker computes their own weights
- Each worker don’t exchange weights with other workers

M. Langer et al. IEEE TPDS 31:12, 2020



Overview of distributed DL frameworks

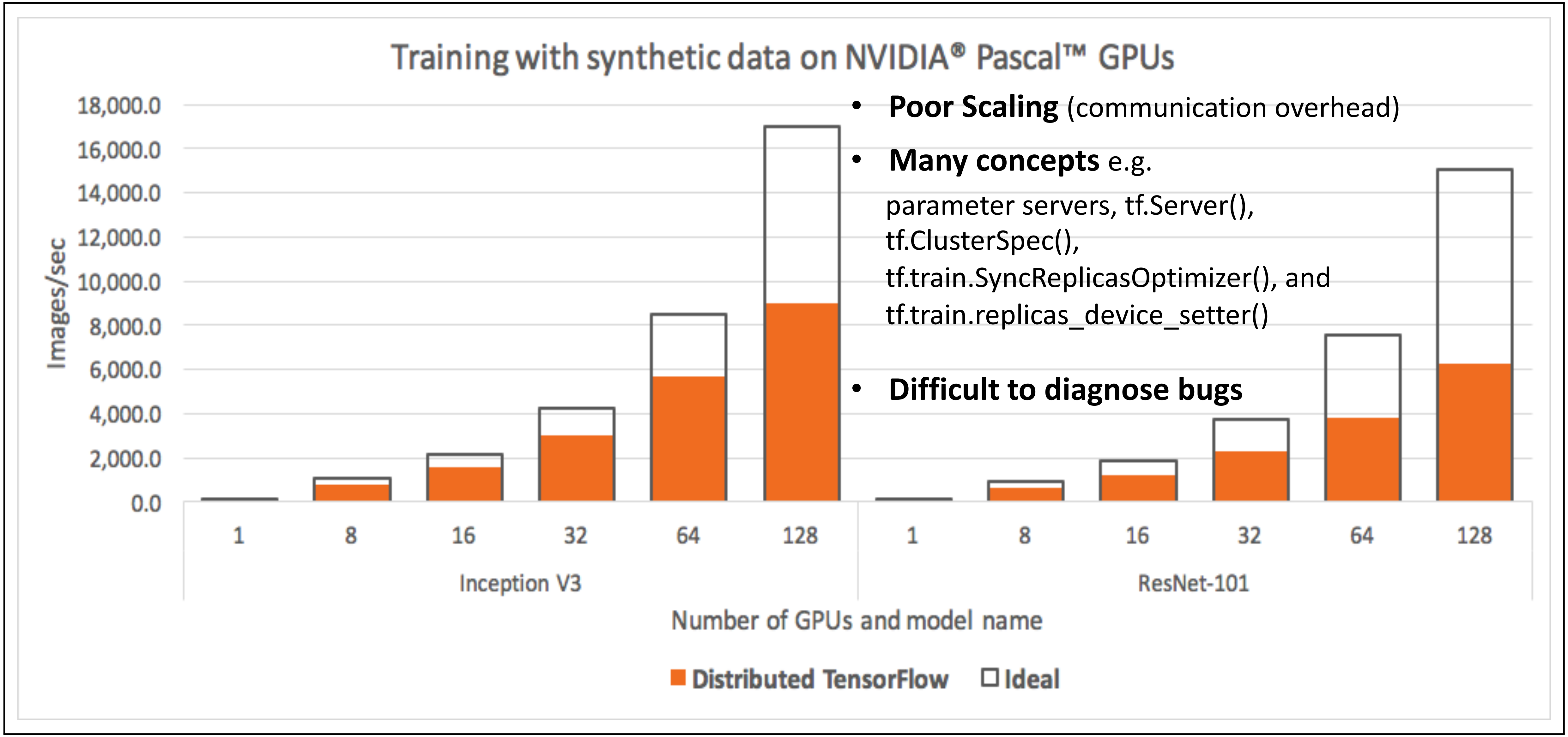
Framework	Parallelism	Communication
DistBelief [18]	Model + Data	Asynchronous
FireCaffe [21]	Data	Synchronous
Horovod [5]	~~Model~~ + Data	Synchronous
MXNet [23]	Model + Data	Bounded Asynchronous
Petuum [19]	Model + Data	Bounded Asynchronous
TensorFlow [22]	model + Data	Bounded Asynchronous
PyTorch-DDP [6]	Model + Data	Synchronous
DeepSpeed [7]	Model + Data	Synchronous

Bounded asynchronous is a hybrid of synchronous and asynchronous communication

Aach *et al.* "Large scale performance analysis of distributed deep learning frameworks for convolutional neural networks" *Journal of Big Data 10:96 (2023)*

Standard distributed TensorFlow

Scaling performance



- **Poor Scaling** (communication overhead)
- **Many concepts** e.g.
  parameter servers, tf.Server(), tf.ClusterSpec(), tf.train.SyncReplicasOptimizer(), and tf.train.replicas_device_setter()
- **Difficult to diagnose bugs**

Alexander Sergeev, Mike Del Balso https://arxiv.org/abs/1802.05799

Distributed DNN Training with Horovod

- Concept of Horovod
- Implementation of Horovod with TensorFlow
- Example of MNIST dataset training

Distributed DNN Training with Horovod

What is Horovod ?

- Horovod is an open Source library built for distributed training on multiple GPUs and across multiple nodes.
- Horovod is designed to integrate existing DL frameworks: TensorFlow, Keras, PyTorch, Apache MXNet.
- Horovod is built based on communication libraries e.g. MPI (Message Passing Interface), NCCL, Gloo.

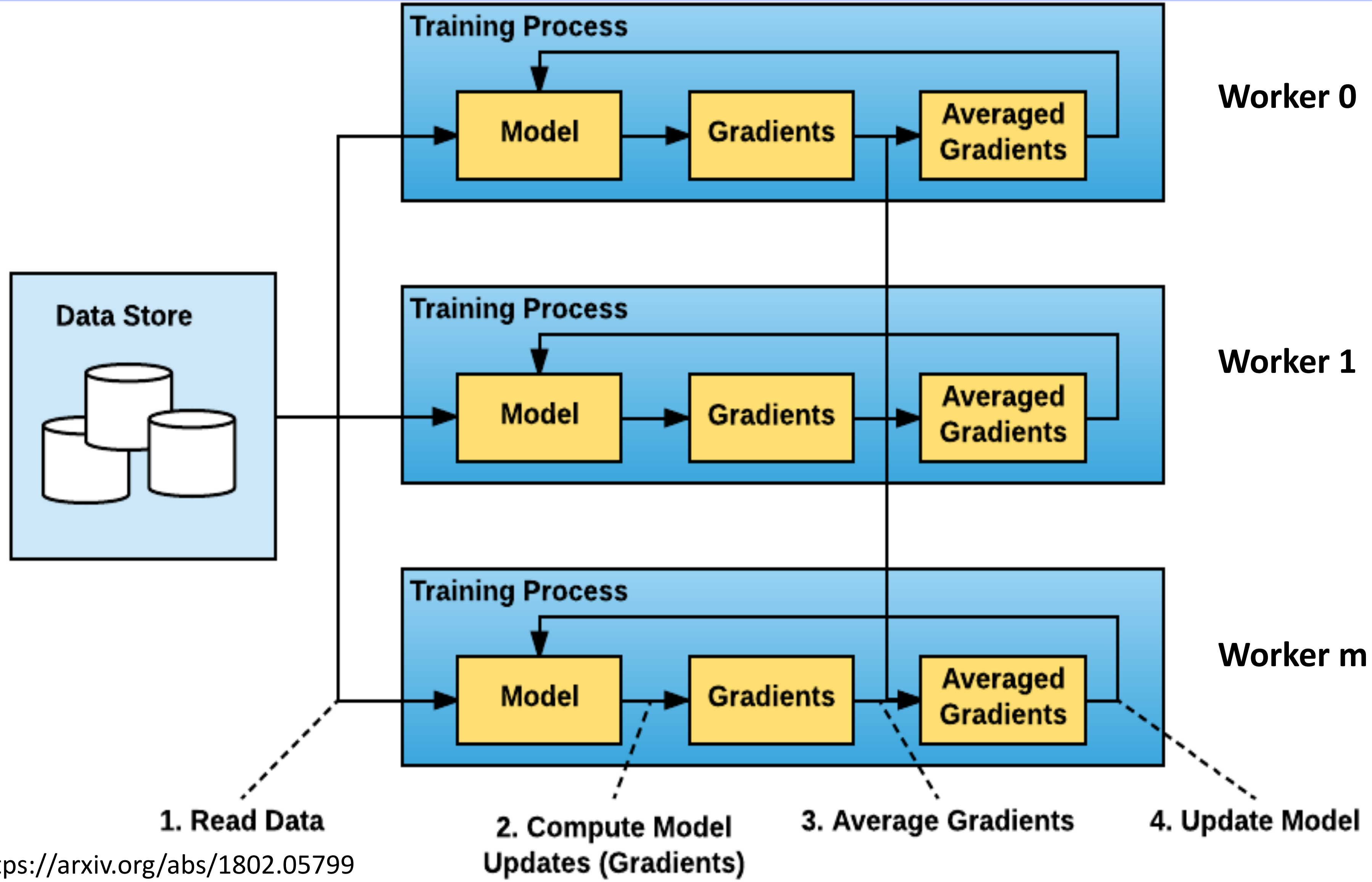
Concept of Horovod:

Key points of Horovod:

- Decentralised data parallelism scheme
- Adjusting learning rate technique **Facebook: https://arxiv.org/abs/1706.02677 (2017)**
- Optimal bandwith ring-allreduce **https://www.sciencedirect.com/science/article/pii/S0743731508001767 (2009)**
- Ring-allreduce algorithm **Baidu: https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/ (2017)**

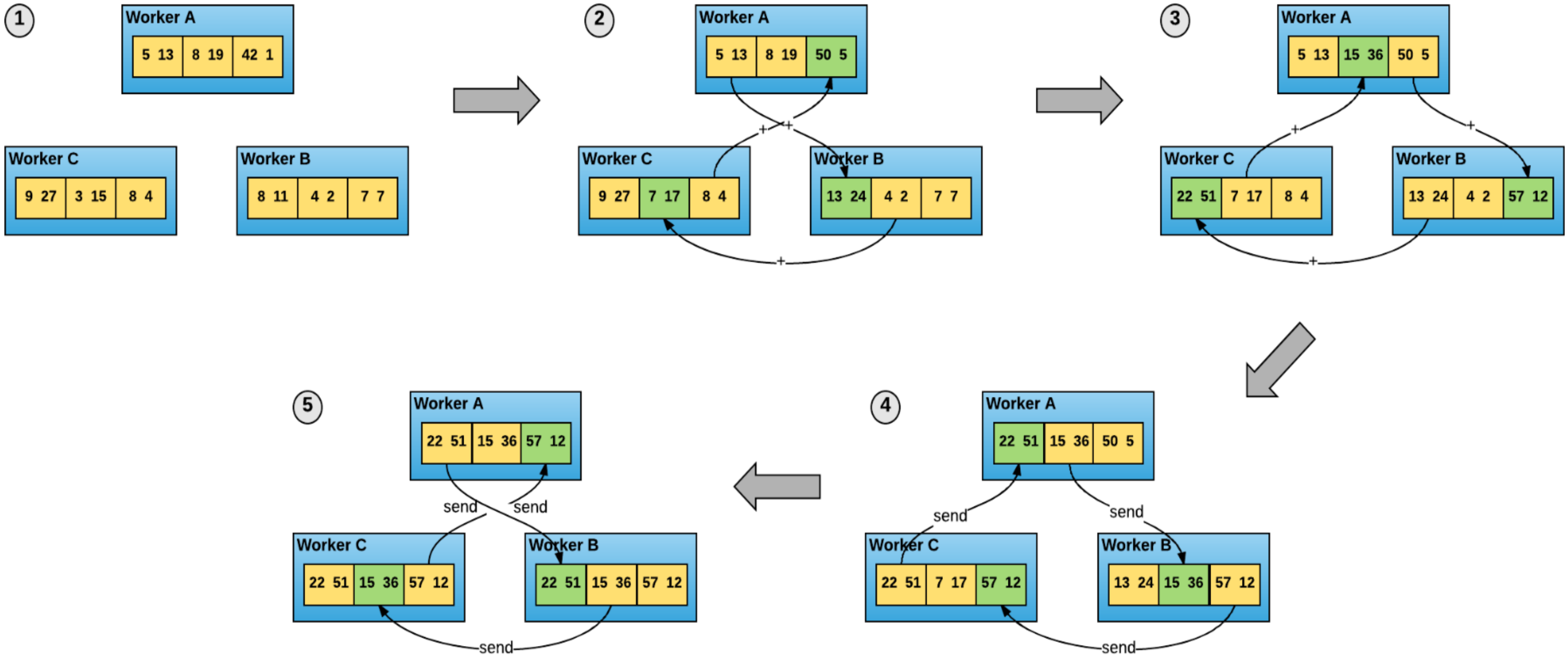
Concept of Horovod: Data parallelism

$$\nabla_{\mathbf{w}} f(\mathbf{w}; X) = \frac{1}{N}\left(\frac{1}{B}\sum_{i=1}^{B} \nabla_{\mathbf{w}}\ell(\mathbf{w}, \mathbf{x}_i) + \frac{1}{B}\sum_{i=B+1}^{B\times 2} \nabla_{\mathbf{w}}\ell(\mathbf{w}, \mathbf{x}_i) + \ldots + \frac{1}{B}\sum_{i=B\times(N-1)+1}^{B\times N} \nabla_{\mathbf{w}}\ell(\mathbf{w}, \mathbf{x}_i) \right)$$



A. Sergeev, M. Del Balso https://arxiv.org/abs/1802.05799

Concept of Horovod: ring-allreduce algorithm

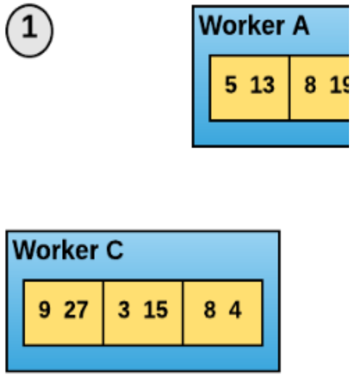


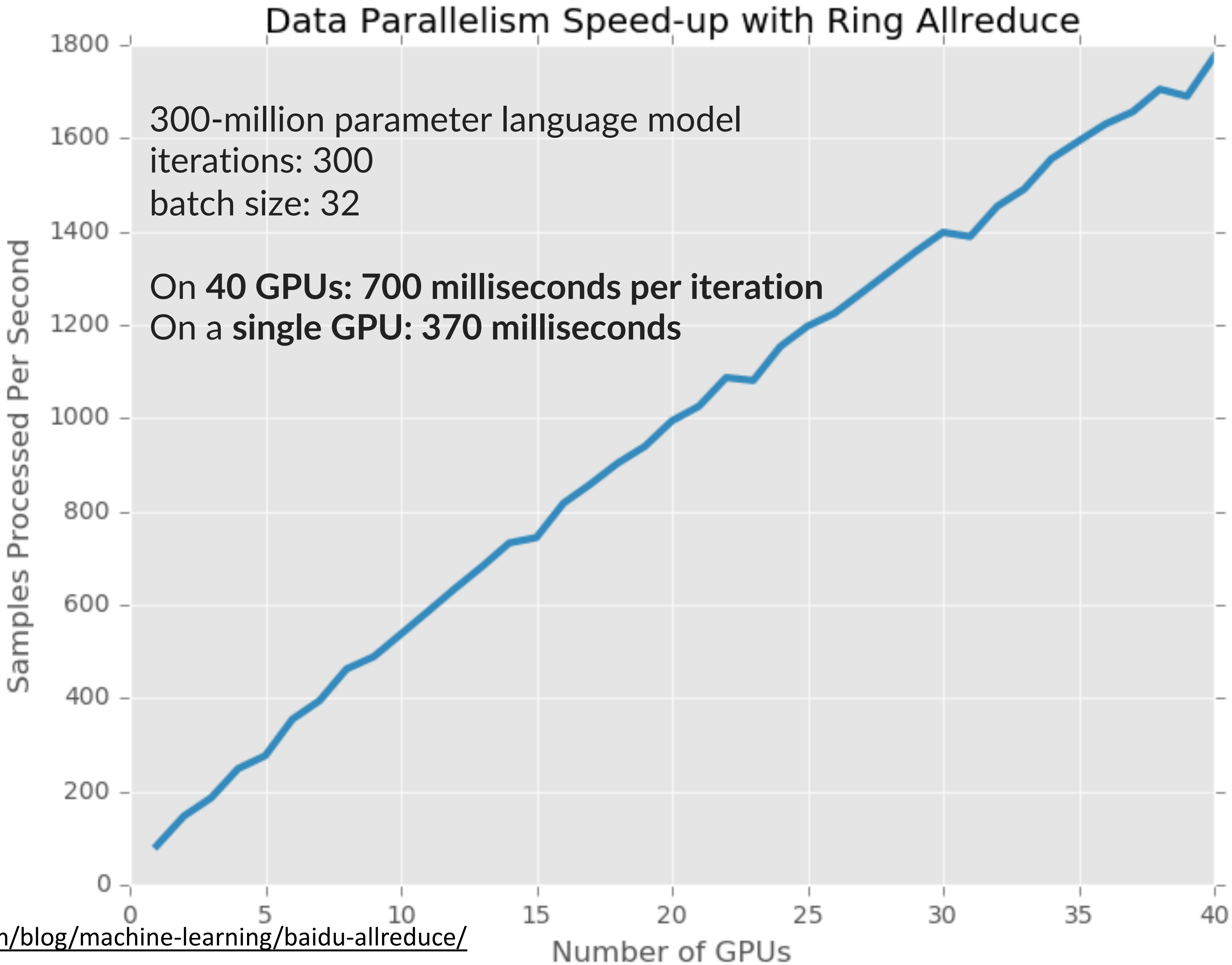
Overlaping between communication (data transfer) and computation (backpropagation)

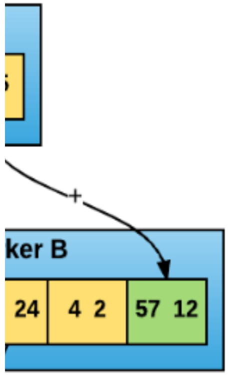
P. Patarasuk & X. Yuan J. Parallel Distrib. Comput. 69, 117–124 (2009)

A. Sergeev, M. Del Balso https://arxiv.org/abs/1802.05799 (2018)

Concept of Horovod: ring-allreduce algorithm

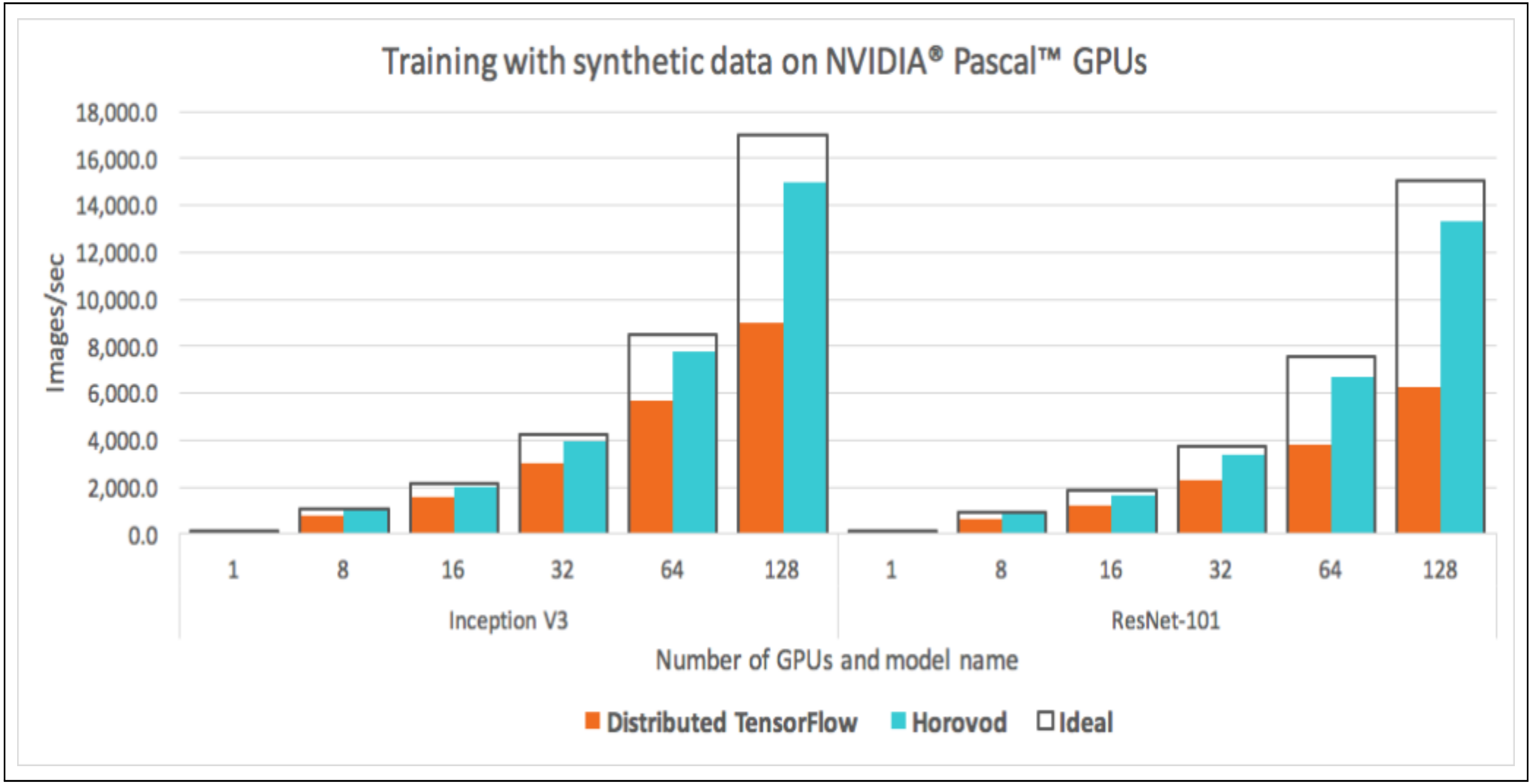






https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Horovod benchmarks



A. Sergeev, M. Del Balso https://arxiv.org/abs/1802.05799

Implementation

Implemention of Horovod with TensorFlow

0- Import Horovod

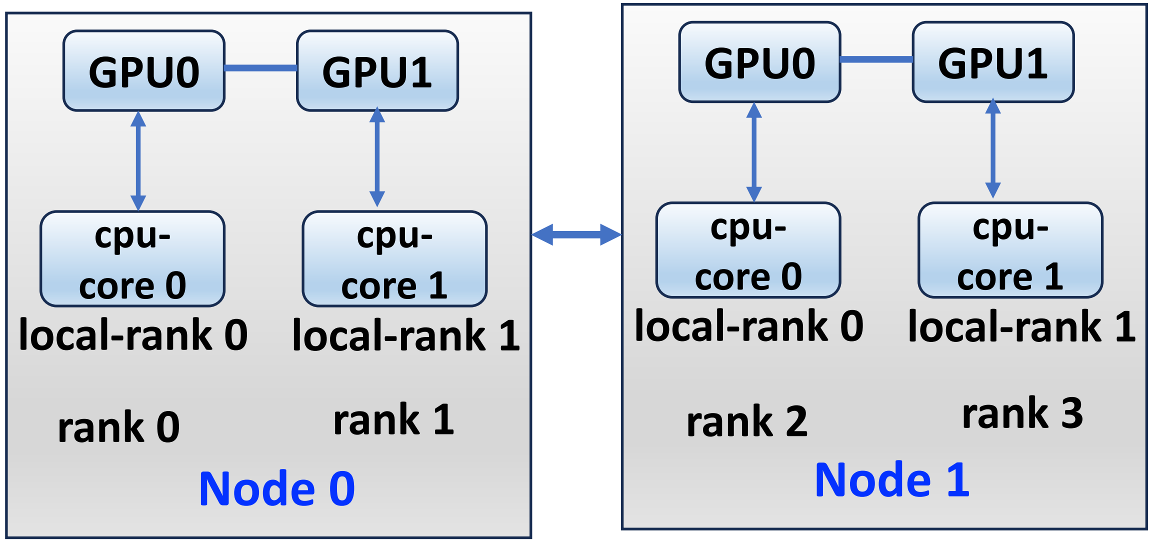
```
import horovod.tensorflow as hvd
```

1- Initialize Horovod

```
hvd.init()
```

2- Assign each GPU to a single process (local rank)

```
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')
```



3- Scale learning rate

```
learning_rate = learning_rate * hvd.size()
```

Effective batch size = **batch size x Nbr of devices**

An increase in learning rate compensates the increased batch size.

4-Apply Horovod distributed optimizer to the original optimizer

```
Opt = hvd.DistributedOptimizer(Opt)
      Or
hvd.DistributedGradientTape if using
tf.GradientTape
```

5-Broadcast initial variables from rank==0 to all processes

```
hvd.broadcast_variables
```

This after initializing models and optimizers.

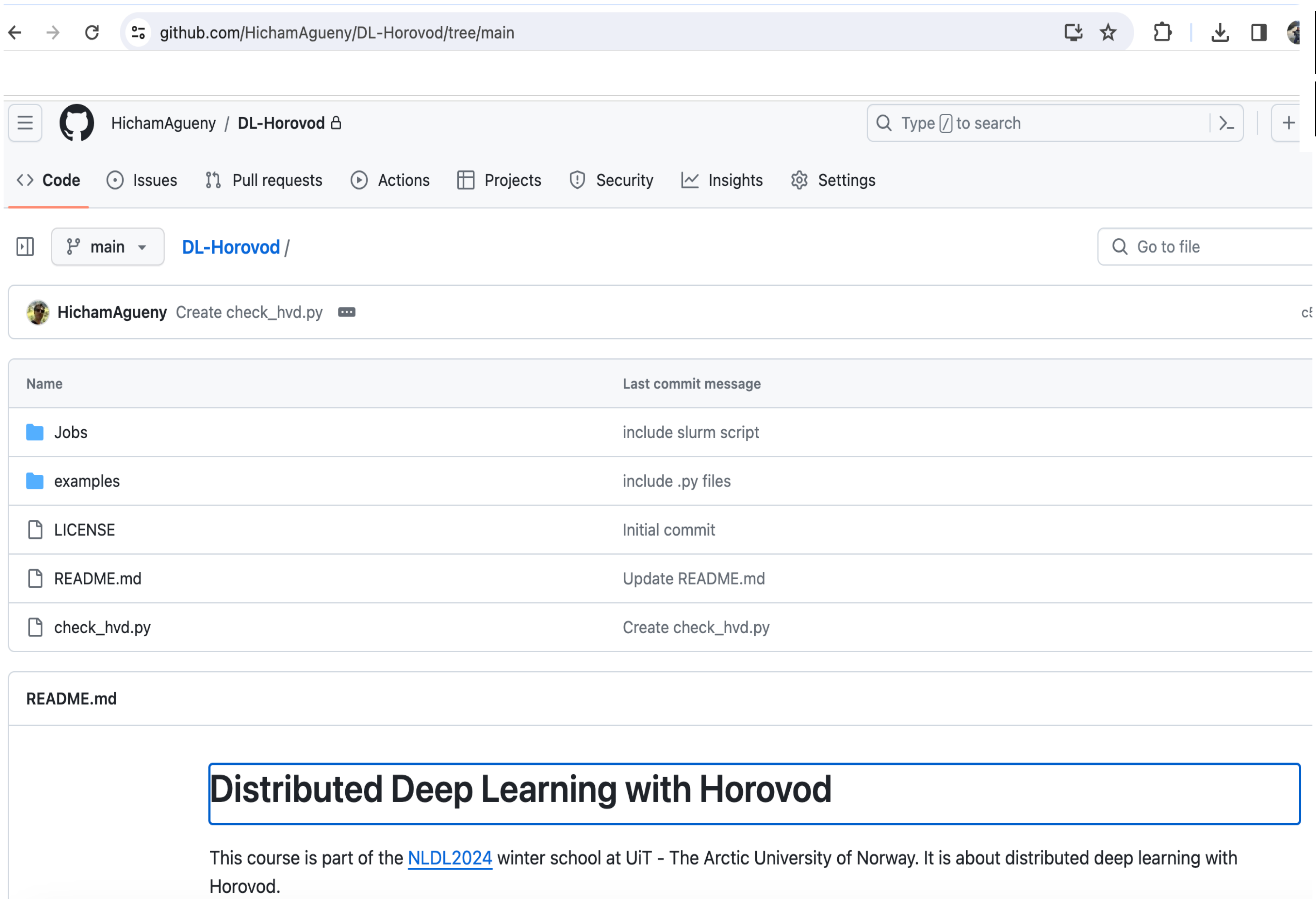
6-Save checkpoints on rank==0

```
checkpoint.save() when hvd.rank() == 0
```

https://horovod.readthedocs.io/en/stable/tensorflow.html

Tutorial

GitHub repo: $ git clone https://github.com/HichamAgueny/DL-Horovod.git



Tutorial: MNIST dataset training

Single-GPU training

```
def train(learning_rate,batch_size,epochs):
    # Import tensorflow modules
    import tensorflow as tf
    from tensorflow import keras
```

Distributed with Horovod

```
def train_hvd(learning_rate,batch_size,epochs):
    # Import tensorflow modules
    import tensorflow as tf
    from tensorflow import keras
    import horovod.tensorflow.keras as hvd

   # Initialize Horovod
    hvd.init()

    # Assign each GPU to each local rank
    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')
```

Tutorial: MNIST dataset training

Single-GPU training

```
def train(learning_rate,batch_size,epochs):
    …
    …….
    # Prepare dataset
    # Here the default is rank=0, size=1
    (x_train, y_train), (x_test, y_test) = get_dataset()


    # Initialize DNN model
    model = get_model()

    # Specify the optimizer:
    #
    optimizer = keras.optimizers.Adadelta(
                     learning_rate)
```

Distributed with Horovod

```
def train_hvd(learning_rate,batch_size,epochs):
    …
    …….
    # Prepare dataset with the use of Horovod rank and size
    # the data is partitioned according to the nbr of processes
    (x_train, y_train), (x_test, y_test) = get_dataset(
                                              hvd.rank(), hvd.size())

    # Initialize DNN model
    model = get_model()

    # Specify the optimizer:
    # Scale the learning rate with the total number of GPUs
    optimizer = keras.optimizers.Adadelta(
                     learning_rate=learning_rate * hvd.size())

    # Use the Horovod Distributed Optimizer
    optimizer = hvd.DistributedOptimizer(optimizer)
```

Tutorial: MNIST dataset training

Single-GPU training

```
def train(learning_rate,batch_size,epochs):
    …
    …….
    # Compile the model
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy’])
```

Distributed with Horovod

```
def train_hvd(learning_rate,batch_size,epochs):
    …
    …….
    # Compile the model
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy’])

   # Create a callback to broadcast
    callbacks = [
     #Broadcast the initial variable from rank 0 to all ranks.
        hvd.callbacks.BroadcastGlobalVariablesCallback(0),
     #Average metrics at the end of every epoch.
        hvd.callbacks.MetricAverageCallback(),
      #Scale the learning rate `lr = lr * hvd.size()`.
      #warmup_epochs could be adjusted.
        hvd.callbacks.LearningRateWarmupCallback(
              lr=1e-3*hvd.size(), warmup_epochs=3, verbose=1),
    ]
```

Tutorial: MNIST dataset training

Single-GPU training

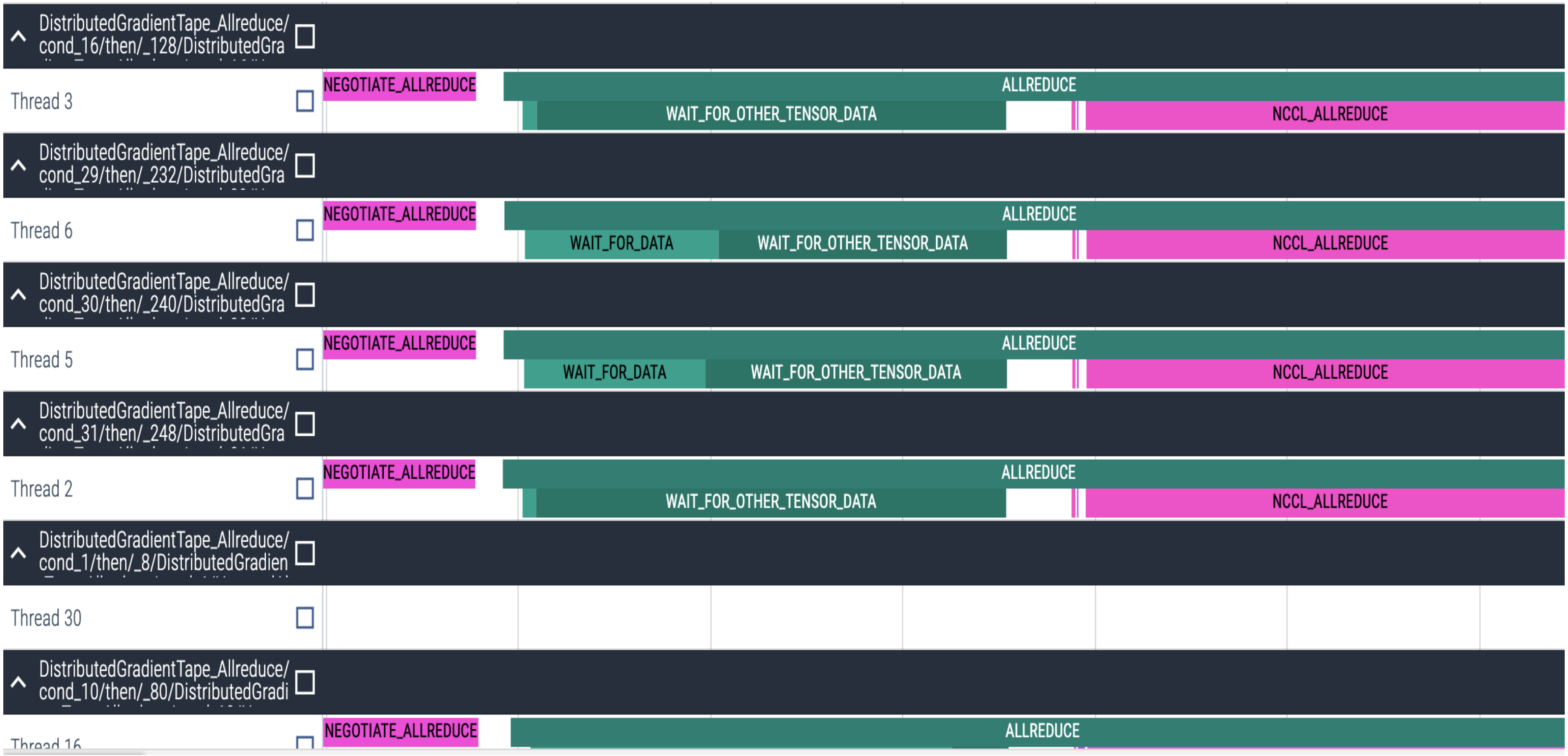
```
def train(learning_rate,batch_size,epochs):
    …
    …….
    #save model checkpoints during training

    callbacks = tf.keras.callbacks.ModelCheckpoint(
                                     checkpoint_file,
                                      monitor='val_loss',
                                      mode='min',
                                      save_best_only=True)

    # Train the model
    model.fit(x_train,
              y_train,
              batch_size=batch_size,
              callbacks=callbacks,
              epochs=epochs,
              verbose=2,
              validation_data=(x_test, y_test))
```

Distributed with Horovod

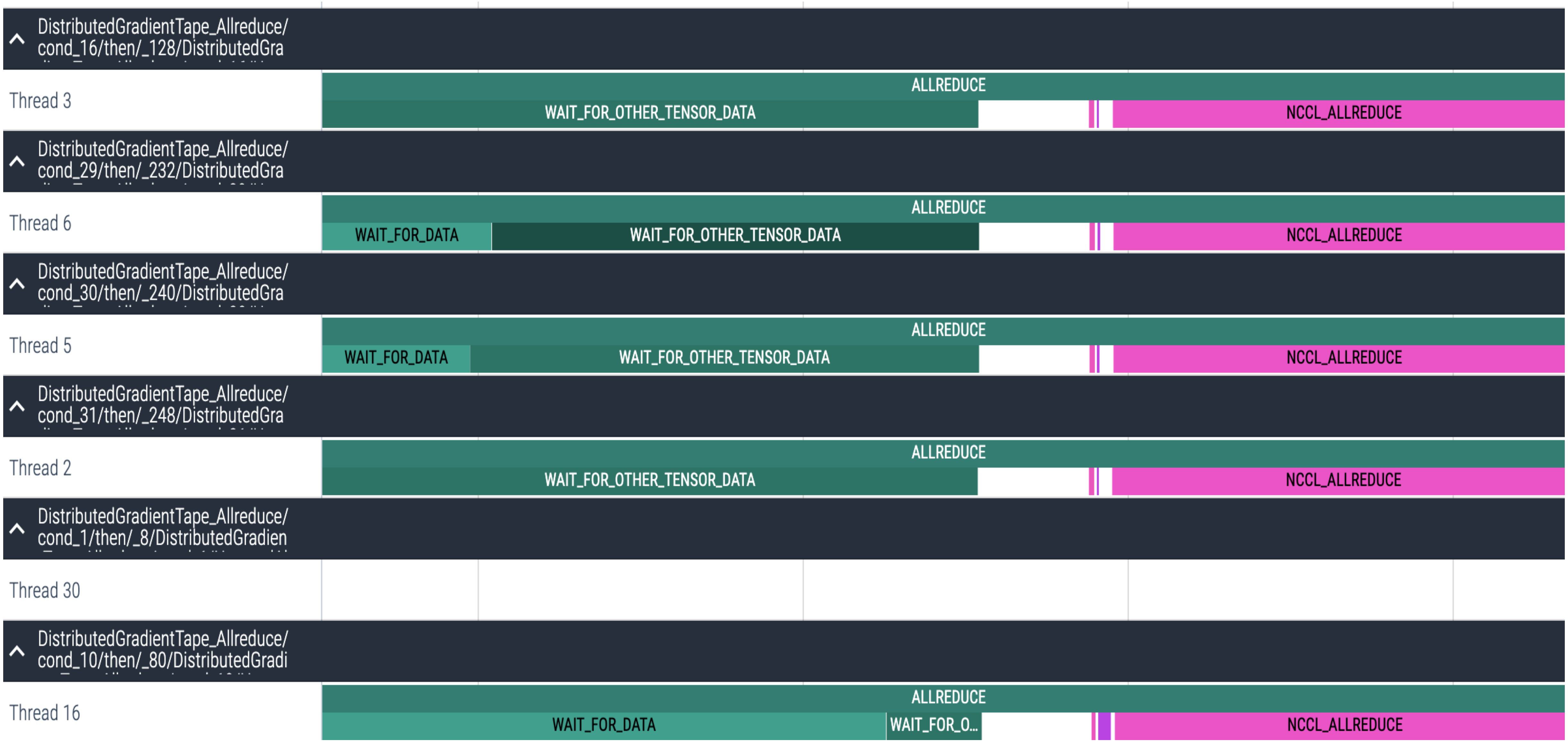
```
def train_hvd(learning_rate,batch_size,epochs):
    …
    …….
    # Save checkpoints during training only on worker 0
    if hvd.rank() == 0:
        callbacks.append(
            keras.callbacks.ModelCheckpoint(checkpoint_file,
                                            monitor='val_loss',
                                            mode='min',
                                            save_best_only=True))

    # Train the model
    model.fit(x_train,
              y_train,
              batch_size=batch_size,
              callbacks=callbacks,
              epochs=epochs,
              verbose=2,
              validation_data=(x_test, y_test))
```

Horovod timeline for Profiling

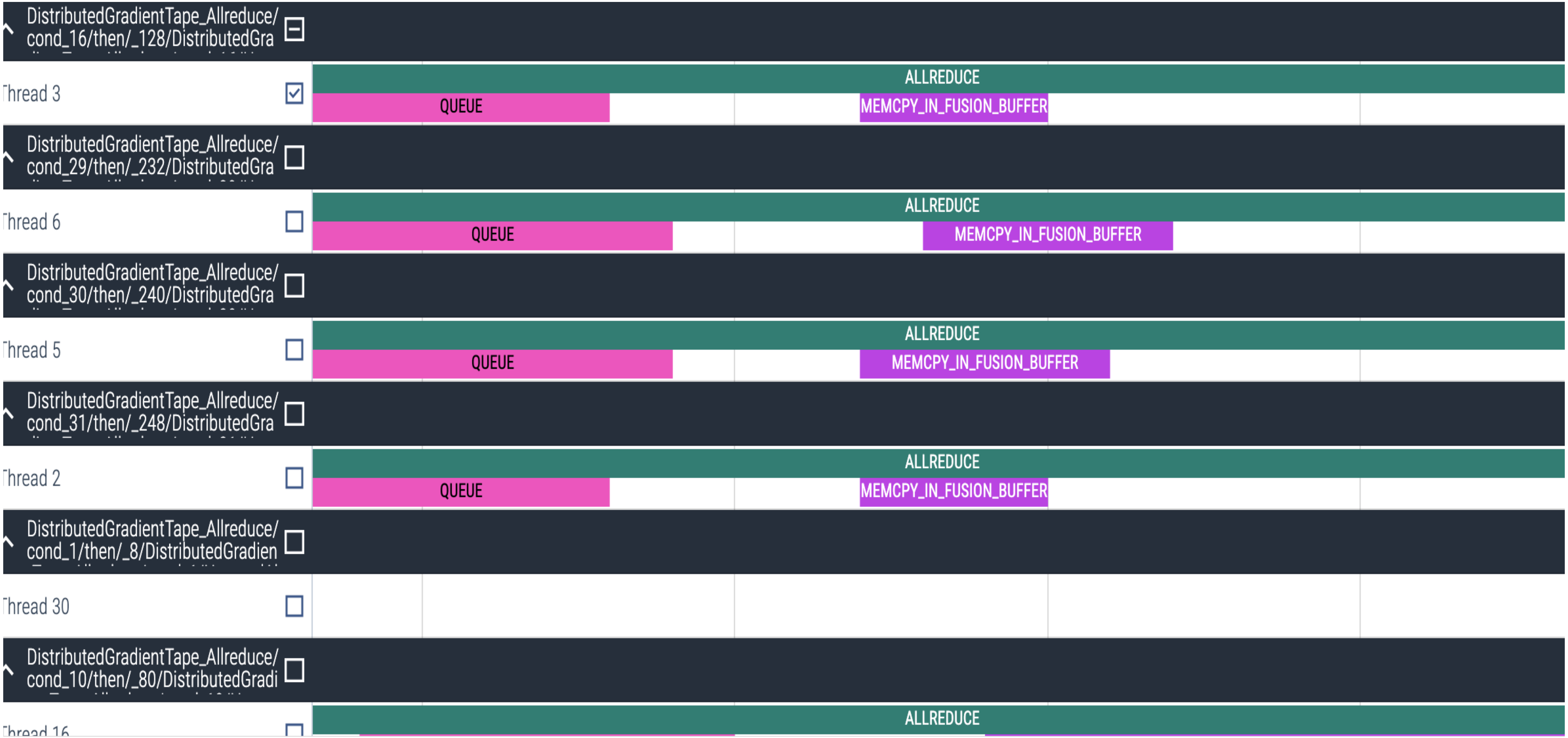
Horovod timeline for Profiling



Horovod timeline for Profiling



Horovod timeline for Profiling



GPU-Binding (Efficient data transfer)

```
hiagueny@uan01:~> salloc -A project_465000485 -t 00:05:00 -p standard-g -N 1 --gpus 8
salloc: Pending job allocation 3636016
salloc: job 3636016 queued and waiting for resources
salloc: job 3636016 has been allocated resources
salloc: Granted job allocation 3636016
hiagueny@uan01:~> srun rocm-smi --showtoponuma


======================= ROCm System Management Interface =======================
================================== Numa Nodes ==================================
GPU[0]          : (Topology) Numa Node: 3
GPU[0]          : (Topology) Numa Affinity: 3
GPU[1]          : (Topology) Numa Node: 3
GPU[1]          : (Topology) Numa Affinity: 3
GPU[2]          : (Topology) Numa Node: 1
GPU[2]          : (Topology) Numa Affinity: 1
GPU[3]          : (Topology) Numa Node: 1
GPU[3]          : (Topology) Numa Affinity: 1
GPU[4]          : (Topology) Numa Node: 0
GPU[4]          : (Topology) Numa Affinity: 0
GPU[5]          : (Topology) Numa Node: 0
GPU[5]          : (Topology) Numa Affinity: 0
GPU[6]          : (Topology) Numa Node: 2
GPU[6]          : (Topology) Numa Affinity: 2
GPU[7]          : (Topology) Numa Node: 2
GPU[7]          : (Topology) Numa Affinity: 2
============================= End of ROCm SMI Log ==============================
hiagueny@uan01:~> srun lscpu | grep NUMA
NUMA node(s):                   4
NUMA node0 CPU(s):              0-15,64-79
NUMA node1 CPU(s):              16-31,80-95
NUMA node2 CPU(s):              32-47,96-111
NUMA node3 CPU(s):              48-63,112-127
```

Binding option: CPU-GPU affinity

```
hiagueny@uan01:~> salloc -A project_465000485 -t 00:05:00 -p standard-g -N 1 --gpus 8
salloc: Pending job allocation 3636016
salloc: job 3636016 queued and waiting for resources
salloc: job 3636016 has been allocated resources
salloc: Granted job allocation 3636016
hiagueny@uan01:~> srun rocm-smi --showtoponuma


======================= ROCm System Management Interface =========
==================================== Numa Nodes ====================
GPU[0]          : (Topology) Numa Node: 3
GPU[0]          : (Topology) Numa Affinity: 3
GPU[1]          : (Topology) Numa Node: 3
GPU[1]          : (Topology) Numa Affinity: 3
GPU[2]          : (Topology) Numa Node: 1
GPU[2]          : (Topology) Numa Affinity: 1
GPU[3]          : (Topology) Numa Node: 1
GPU[3]          : (Topology) Numa Affinity: 1
GPU[4]          : (Topology) Numa Node: 0
GPU[4]          : (Topology) Numa Affinity: 0
GPU[5]          : (Topology) Numa Node: 0
GPU[5]          : (Topology) Numa Affinity: 0
GPU[6]          : (Topology) Numa Node: 2
GPU[6]          : (Topology) Numa Affinity: 2
GPU[7]          : (Topology) Numa Node: 2
GPU[7]          : (Topology) Numa Affinity: 2
============================= End of ROCm SMI Log ================
hiagueny@uan01:~> srun lscpu | grep NUMA
NUMA node(s):                     4
NUMA node0 CPU(s):                0-15,64-79
NUMA node1 CPU(s):                16-31,80-95
NUMA node2 CPU(s):                32-47,96-111
NUMA node3 CPU(s):                48-63,112-127
```

NUMA node 3

NUMA node 1

NUMA node 0

NUMA node 2

#!/bin/bash

....

#SBATCH --gpus=8

#SBATCH --exclusive

**srun --cpu-bind=map_cpu: 49,57, 17,25, 1,9, 33,41 **

./application

Or

MASK="0x${fe}000000000000,0x${fe}00000000000000,
0x${fe}0000,0x${fe}000000,0x${fe},0x${fe}00,
0x${fe}00000000,0x${fe}0000000000"

srun --cpu-bind=mask_cpu:$MYMASKS \
./application

See here for more details

https://github.com/HichamAgueny/DL-Horovod/tree/main/Jobs

Conclusion

- Overview of the compute nodes architecture in LUMI-G.
- Model parallelism vs data parallelism
- Centralized vs decentralized distributed training strategy
- Horovod for distributed training
    - Simple to implement
    - Suitable for large scale distributed training
    - Works with multiple ML frameworks
    - Ring allreduce algorithm
- Horovod timeline profiling

GitHub Repo