Accelerating Neural Network Training with Distributed Asynchronous and Selective Optimization (DASO)

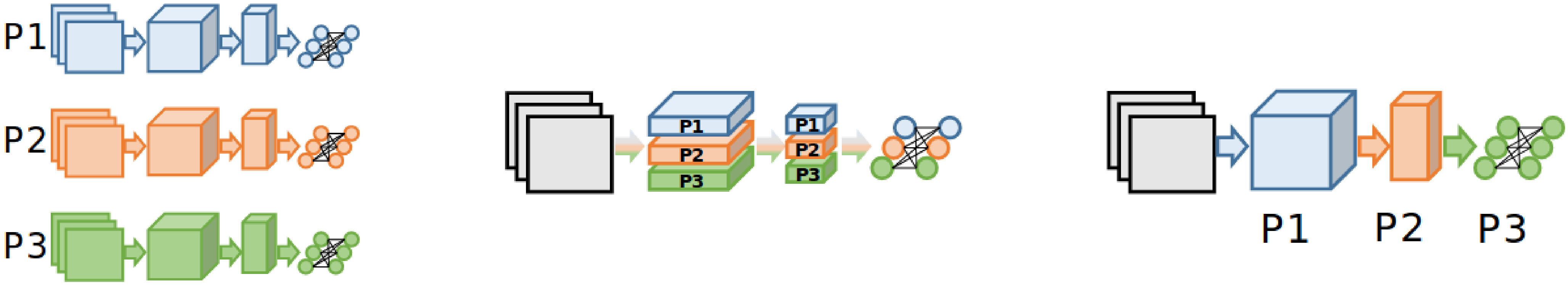
Daniel Coquelin[1], Charlotte Debus[1], Markus Götz[1], Fabrice von der Lehr[2], James Kahn[1], Martin Siggel[2], Achim Streit[1]

1: KIT: SCC
2: DLR: SC

Parallel Neural Networks

Training Networks Faster

- **Data parallel: networks mirrored on all processes**
- Model parallelism: network layers are divided across processes
- Pipelining: network is divided between processes by layer

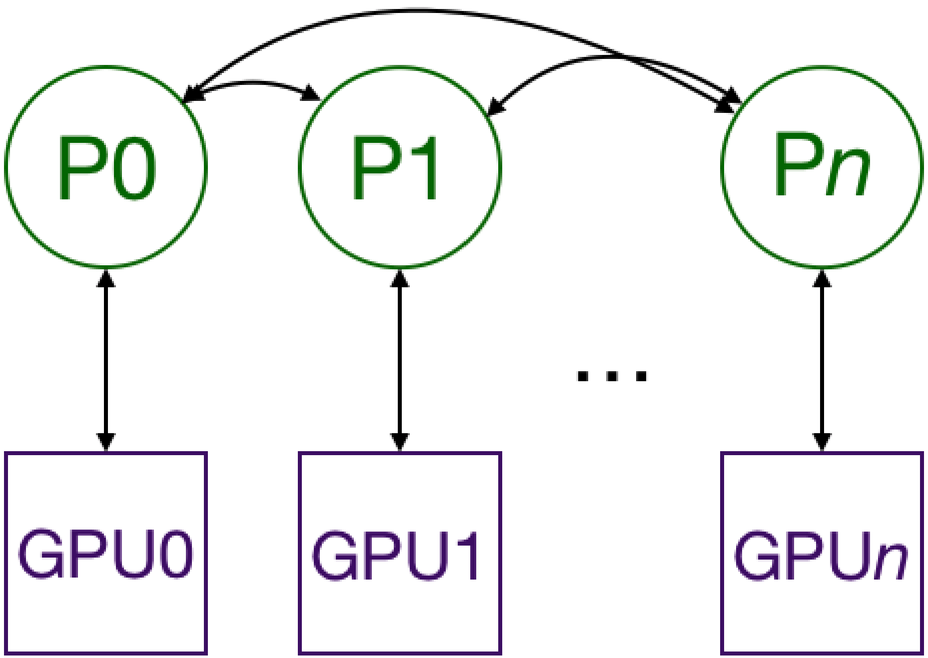


(a) Data Parallelism (b) Model Parallelism (c) Layer Pipelining

Data Parallel Neural Networks

Optimizing -- Focusing on SGD

- In DPNNs model parameters must be synchronized
  - Often done after each batch
  - All processes talk to each other
- Synchronous vs Asynchronous methods
- Regardless of method: this is one of the most prominent training bottlenecks!

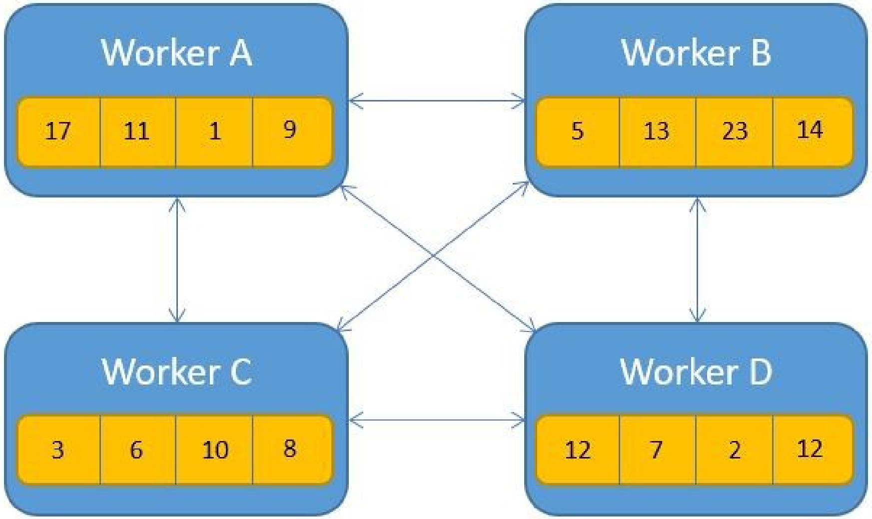


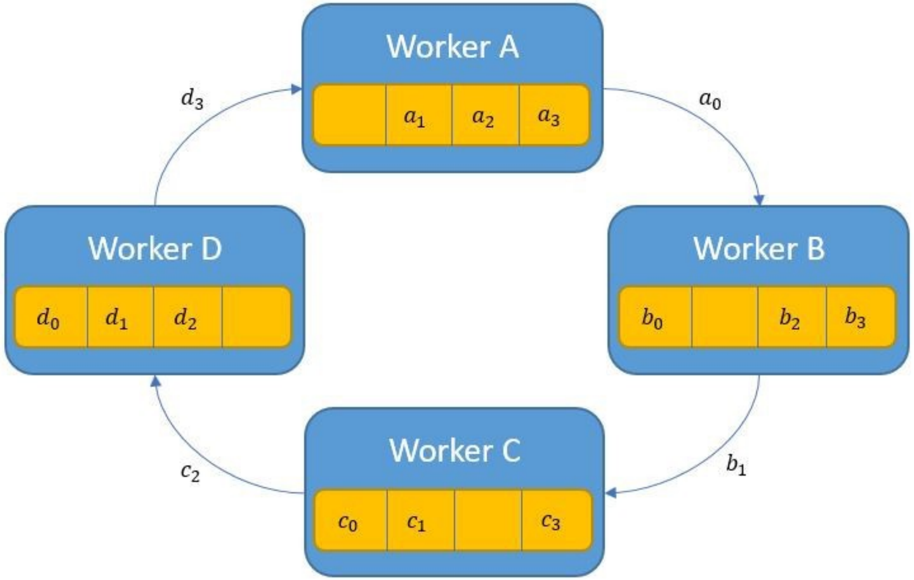
Optimizing DPNN Synchronization

Methods for Reducing Communication Time

- Tensor Fusion
- Compression

- Modified Allreduce logic
- Sending data during the backward step

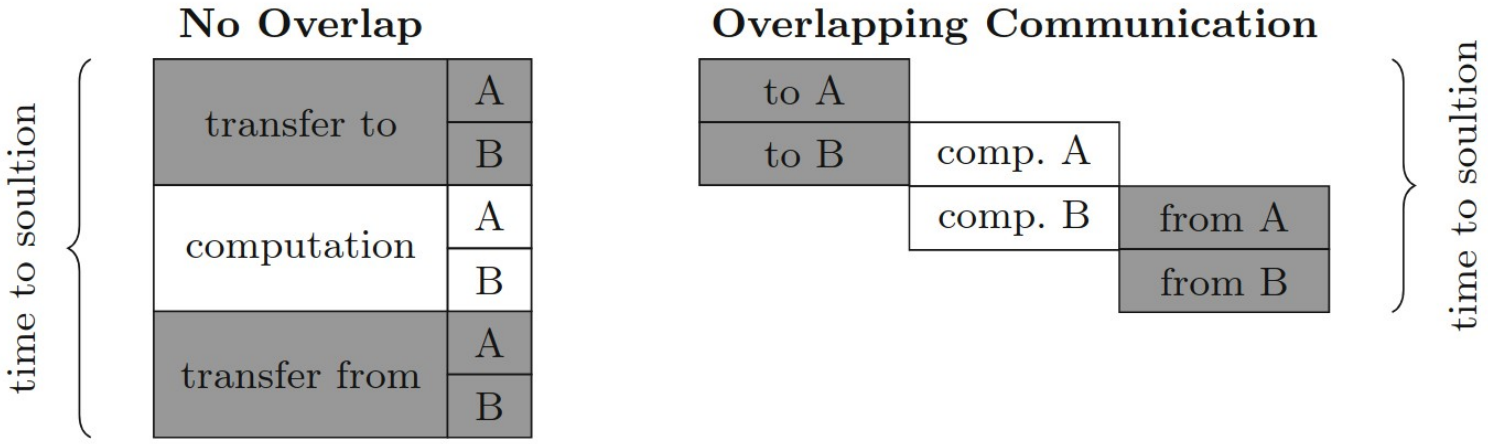




Old-School Distributed Computing

Hiding Communication Behind Computation

- Long studied, highly desired, arduous to do well
- When done well -> much faster execution without loss of accuracy



HeAT – The Helmholtz Analytics Toolkit

A Distributed and Accelerated Tensor Framework

- Distributed tensor framework
  - NumPy-like, Python interface
- Accelerated and distributed processing
  - **GPU Computing**
  - **Multiple cluster nodes via MPI**
- Seamlessly use GPUs and CPUs on both common clusters, personal workstations, and HPC systems
- Algorithms specifically tailored to distributed data
- High-level algorithms
  - Sklearn-like machine-learning
  - **PyTorch-style Neural Networks**

Distributed Asynchronous and Selective Optimization - DASO

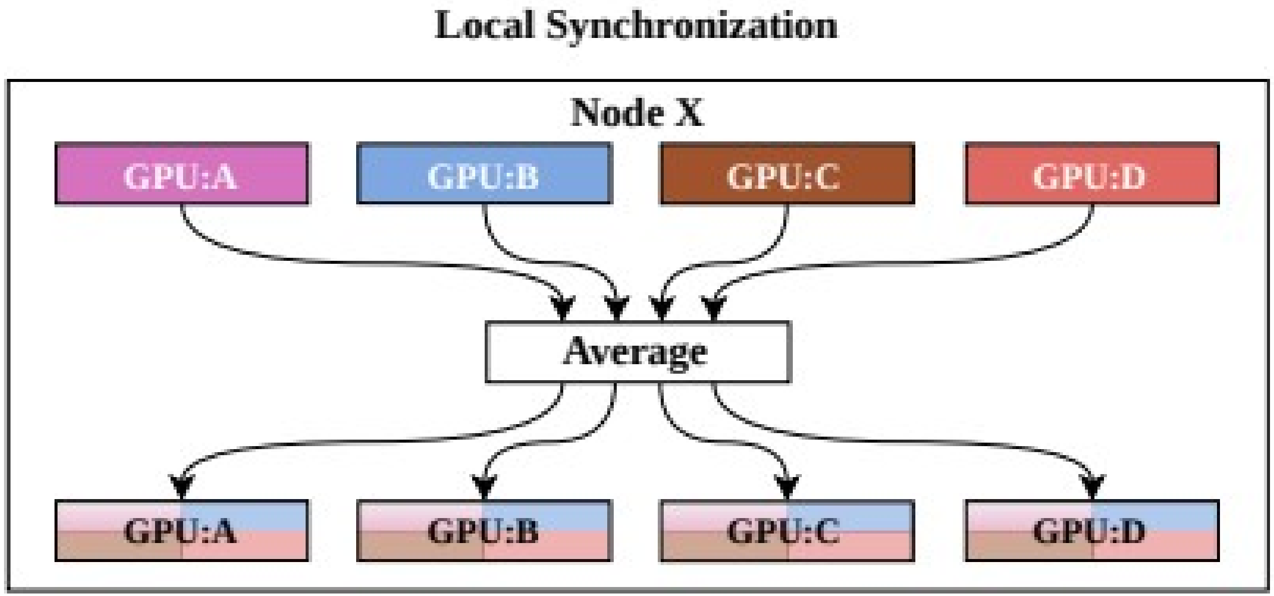
Motivation

- Better utilize cluster architecture
- Reduce communication overhead
- Increase speed with selective global updates
- Divide the global synchronization into three steps:
  1) Local Synchronization
  2) Global Synchronization
  3) Local Update

Distributed Asynchronous and Selective Optimization - DASO

Local Synchronization

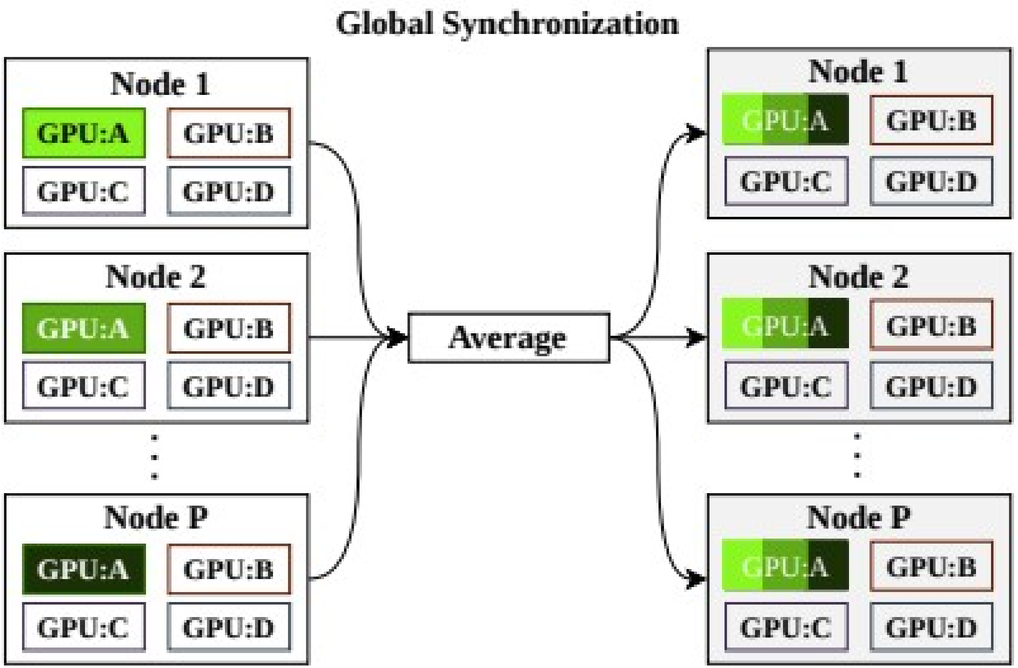
- Traditional synchronization of gradients
- torch.nn.DistributedDataParallel



Distributed Asynchronous and Selective Optimization - DASO

Global Synchronization

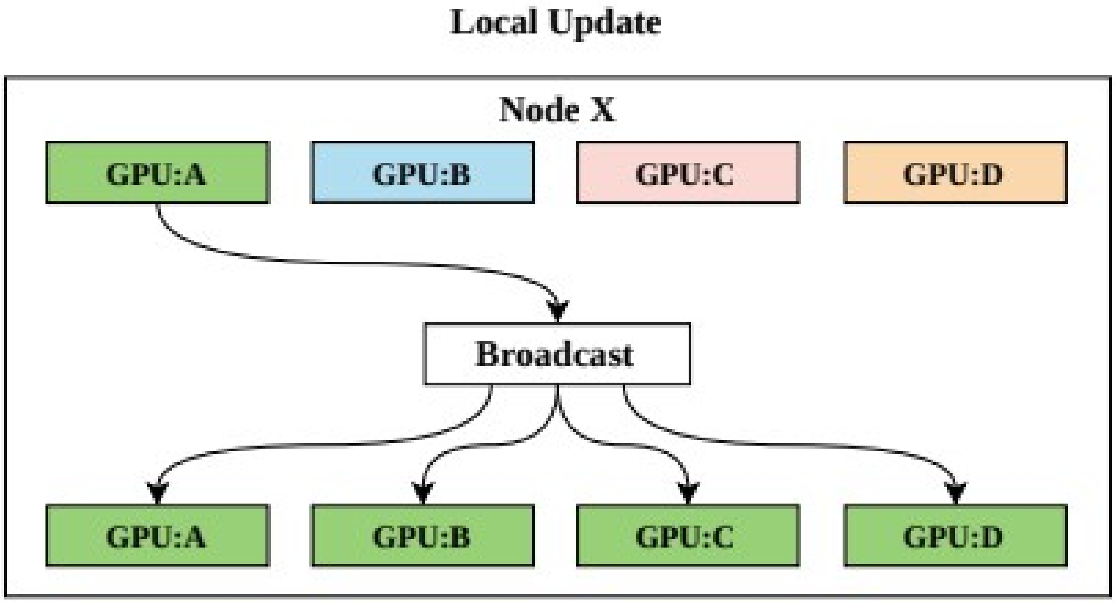
- One GPU/node communicates parameters with the other **nodes**
- MPI Groups
- Average operation only occurs within this group



Distributed Asynchronous and Selective Optimization - DASO

Local Update

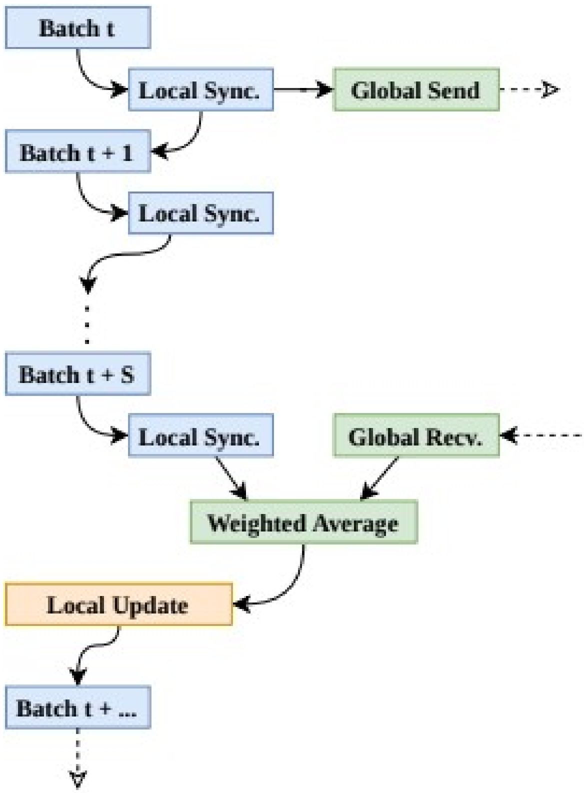
- After averaging, the MPI Group member sends it to overwrite the network parameters on the other GPUs



Distributed Asynchronous and Selective Optimization - DASO

Synchronization Order in Practice

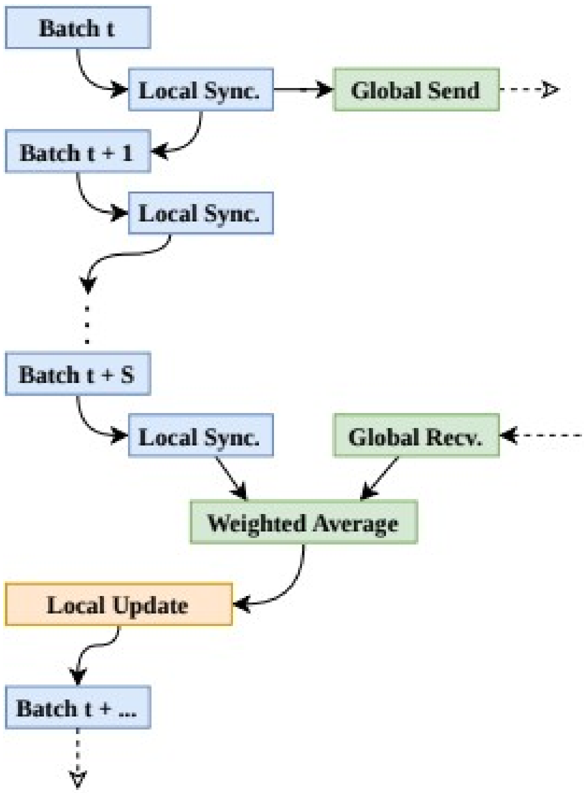
- Warm-up : traditional DPNN training
- Cycling
  - delay between sending and receiving global parameters
  - weighted average for folding in global parameters to updated model states
- Cool-down: traditional DPNN training



Distributed Asynchronous and Selective Optimization - DASO

Cycling Phase

- After parameters are sent, GPUs local to one node continue to train
- After `S` batches, the global parameters are received
- Weighted average to merge local paramters
  - local parameters are doubly weighted
- The number of batches, `S`, between sending and receiving parameters cycles by factors of two
  - i.e. 4, 2, 1, 4, 2, 1, ...



Distributed Asynchronous and Selective Optimization - DASO

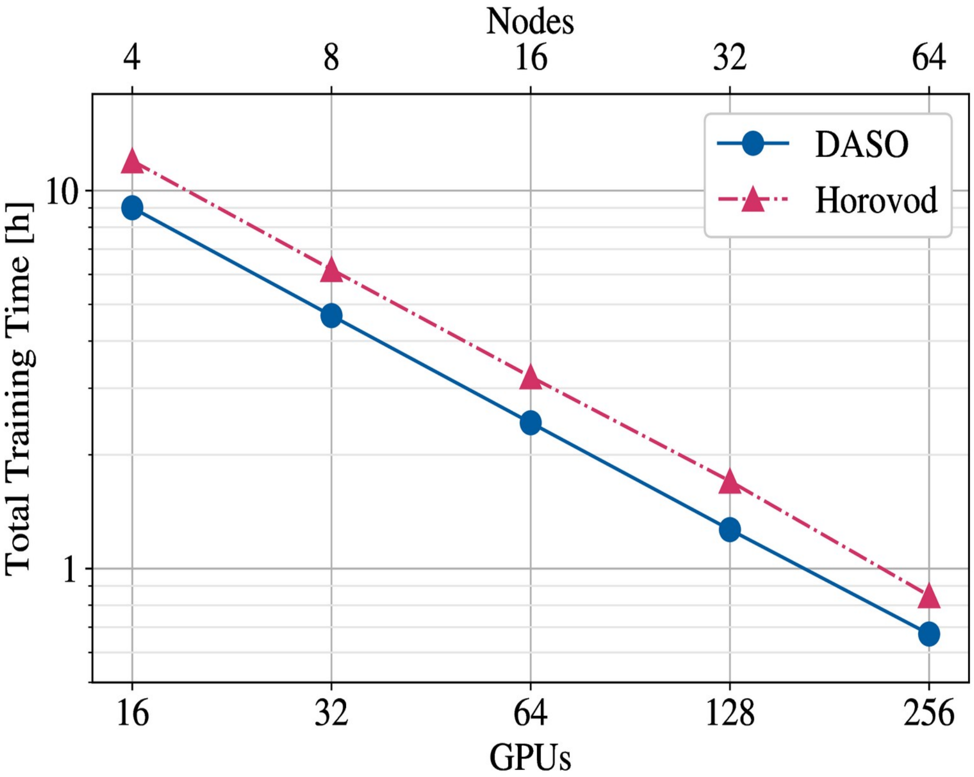
Benefits

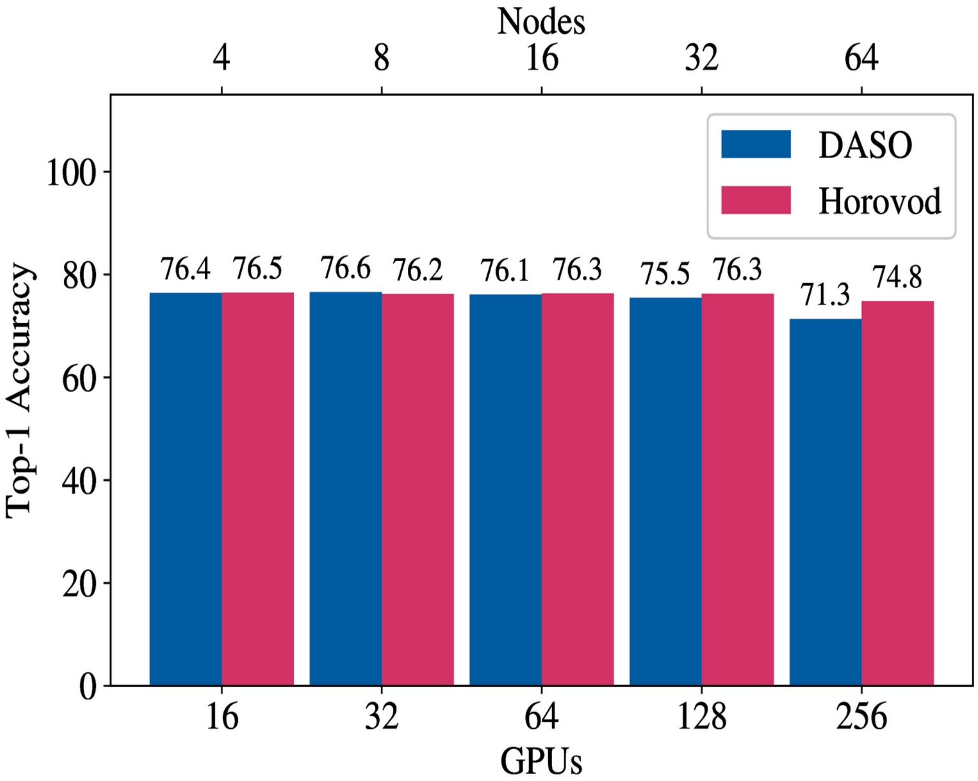
- Fully utilize computing clusters
- Maintain accuracy at large node counts
- Easy to use / implement

```python
import heat as ht
import torch
...
# create PyTorch distributed group
world_size = ht.MPI_WORLD.size
rank = ht.MPI_WORLD.rank
local_rank = rank % num_local_gpus
torch.distributed.init_process_group(
    backend="nccl",
    rank=local_rank,
    world_size=world_size
)
...
# the DASO optimizer is created
daso_optimizer = ht.optim.DASO(
    local_optimizer=optimizer,
    total_epochs=num_epochs
)
...
# the hierarchical network is created
ht_model = ht.nn.DataParallelMultiGPU(
    net,
    daso_optimizer
)
```

DASO vs Horovod – ImageNet Training with ResNet-50

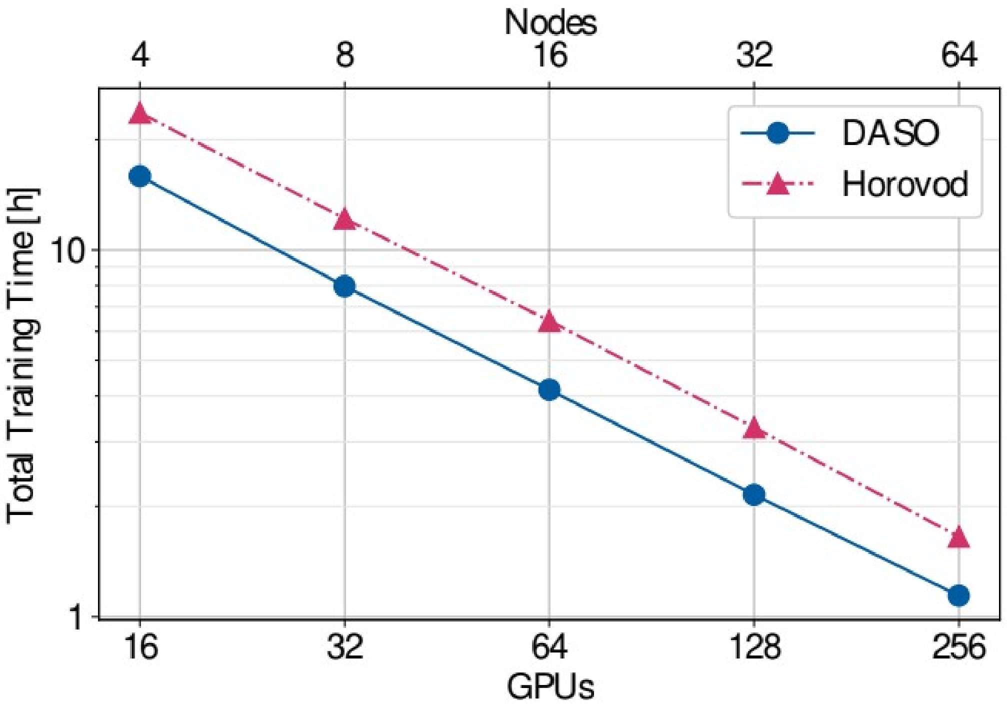
DASO - Distributed Asynchronous and Selective Optimization

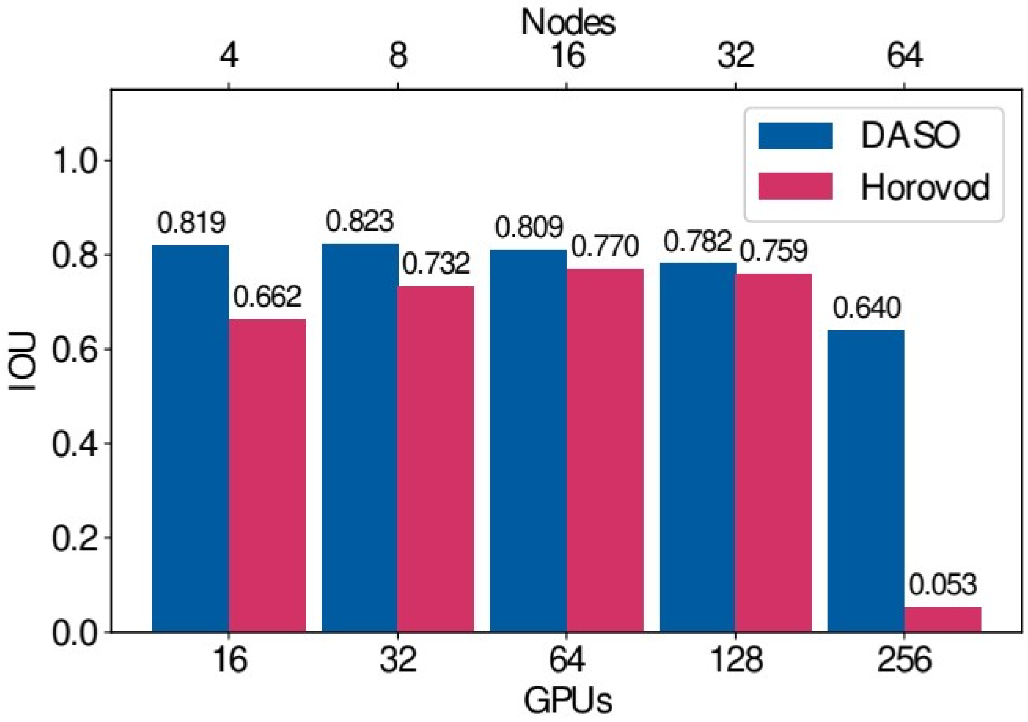




DASO vs Horovod – State-Of-The-Art Model on CityScapes

DASO - Distributed Asynchronous and Selective Optimization





DASO

PyTorch-Style + HPC

- Currently, can you:
  - Train a PyTorch NN with a HeAT dataset? ✓
  - Use PyTorch functions within training? ✓
  - Use the PyTorch Dataloader? ✓
  - Train with a dataset which does not fit into the available memory? ✓
  - Train a network faster than Horovod? ✓

Bottom Line:
DASO trains a network up to 34% faster than Horovod without losing accuracy.

Come and Feel the HeAT!

- Open source software with the liberal MIT license

- Install it with the PIP package:

```
pip install heat
```

  Or join us on GitHub:

```
git clone https://github.com/helmholtz-analytics/heat
```

Acknowledgements

- All collaborators, admins and assistants. In particular:
- Achim Basermann, Philipp Bekemeyer, Lena Blind, Benjamin Bourgart, Claudia Comito, Daniel Coquelin, Charlotte Debus, Michael Denker, Philipp Glock, Klaus Görgen, Björn Hagemeier, Stefan Kollet, Philipp Knechtges, Kai Krajsek, Jakob Ohm, Melven Röhrig-Zöllner, Simon Schmitz, Alexander, Schug, Martin Siggel, Luca Spataro, Achim Streit, Alexandre Strube, Michael Tarnawa, Arthur Voronin, Marie Weiel-Potyagaylo

- This work is supported by the Helmholtz Association Initiative and Networking Fund (INF) under project number ZT-I-0003
- This work was funded by Helmholtz Association’s Initiative and Networking Fund through Helmholtz AI

- Thank you to everyone for participating in this project and for bringing HeAT to life!

References

- Jonas Hahnfeld, Tim Cramer, Michael Klemm, Christian Terboven, and Matthias S. Müller. A pattern for overlapping communication and computation with openmp In Bronis R. de Supinski, Stephen L.Olivier, Christian Terboven, Barbara M.Chapman, and Matthias S. Mller, editors, Scaling OpenMP for Exascale Performance and Portability, pages 325−337, Cham, 2017. Springer International Publishing.
- https://yangkky.github.io/2019/07/08/distributed-pytorch-tutorial.html
- Ben-Nun, T. and Hoefler, T. Demystifying Parallel and Distributed Deep Learning: An In-depth Concurrency Analysis. ACM Computing Surveys (CSUR), 52(4):1–43, 2019. doi: 10.1145/3320060.
- https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da

Available Features and Ongoing Projects

Simple, 2 process example

- **Available Currently:**
- Mean, Std, Var, Average
- Reshape, flatten, ravel, flip
- Complex numbers
- Matrix multiplication
- Histograms
- K-means + friends
- Spectral clustering
- LASSO
- Data Parallel Neural Networks
- And many more!

- **Ongoing Projects**
- ASSET
- mpi4torch
- SVD
- DPNN improvements and extenstions