TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Center for Information Services and High Performance Computing (ZIH)

Zwischenpräsentation Bachelorarbeit

Parallelization of GPUs based on Horovod – A Scaling and Performance Analysis based on an Application from Material Sciences

Paul Orlob

Structure

1) Introduction
2) Model- & Data Parallelism
3) Horovod
4) Example from Material Sciences
5) Speedup & Efficiency
6) Performance Analysis
7) Optimization
8) Next Steps

Motivation

- **Problem**: increasingly large networks + large amounts of data
  - Increased training time
  - Especially problematic when tuning hyperparameters
- **Solution**: GPU-Parallelization, multiple GPUs and Nodes used for training

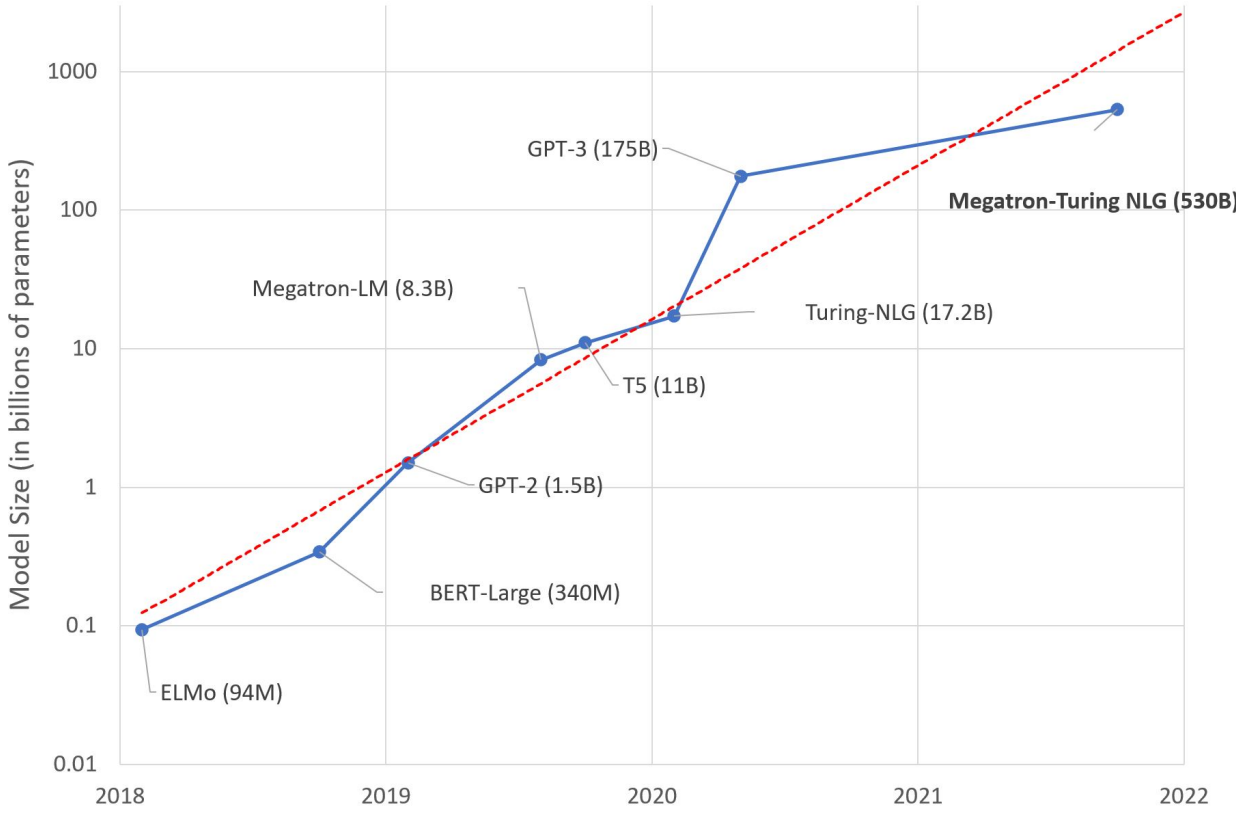


Image: A. Alvi, P. Kharya. „Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, the World's Largest and Most Powerful Generative Language Model". microsoft.com. 2020 [Online] Available: https://www.microsoft.com/en-us/research/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/

Data Parallelism [1, 2, 4]

- **N** workers
- Each Worker has a copy of the network
- Split data into **N** partitions
- Parallel computation of forward- and backward pass
- → Synchronization of **N** groups of parameter-gradients at the end of each backward pass
- → # of synchronizations = # of batches

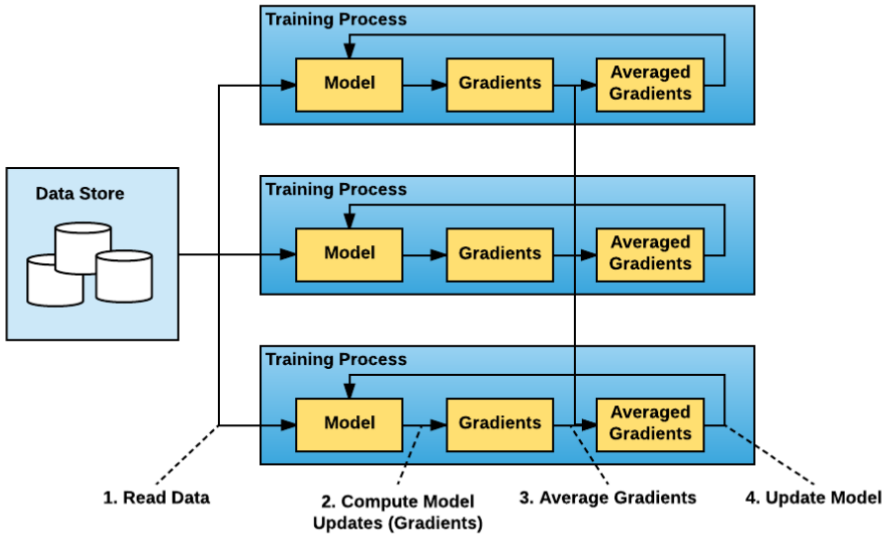


Figure 1: Data parallel training [3]

Model Parallelism [1, 2, 4]

- **N** workers
- Each Worker has a copy of all data
- Split network into **M** partitions
- Parallel computation of (some) network layers
- → Synchronization every time results are needed by another worker
  - Layers can only be computed sequentially

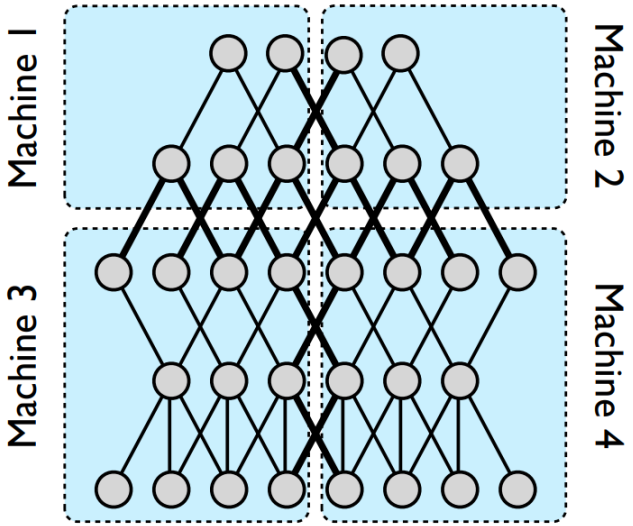


Image : J. Dean et al., “Large scale distributed deep networks,” Advances in neural information processing systems, vol. 25, 2012.

Data vs. Model Parallelism [1,2,3,4]

Data Parallelism	Model Parallelism
Efficient if amount of computation per parameter is high	Efficient if amount of computation per neuron output is high
Number of synchronizations dependant on number of batches	Number of synchronizations dependant on network topology
Reduces size of data per worker	Reduces size of network per worker
Parallel, except gradient synchronization	Parallelism highly dependant on network topology, interconnected sequential layers can not be computed in parallel

Horovod [3]

- Library created by **Uber**, published **2017**, **Apache 2.0** license
- **Motivation:**
  - TensorFlow Distributed used parameter servers (at time of creation) [7]
  - Complicated software development: Number of Parameter Servers; Boilerplate Code
- Supports major DL frameworks (Tensorflow, Keras, PyTorch, MXNet)
- Supports Data Parallelism out-of-the-box
- Uses „ring-allreduce“ algorithm, bandwith-optimal [5, 6]
- „ring-allreduce“ implemented in *MPI* & *NCCL* (NVIDIA Collective Communication Library)
- → **88% efficient** scaling when training Inception V3, ResNet-101

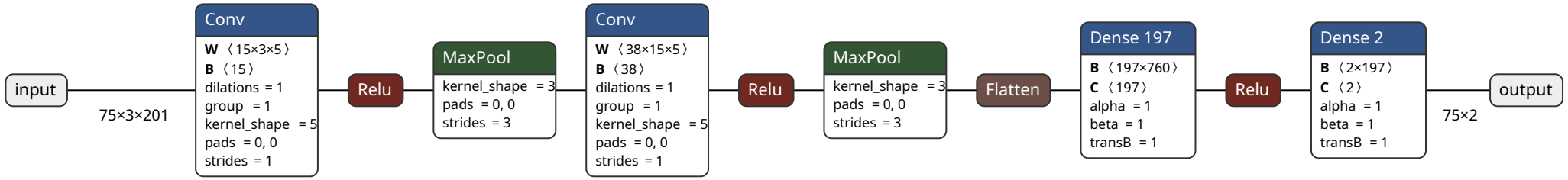
Example from Material-Sciences: Dataset [8]

- Prediction of material parameters from strain-stress curves

- 1M labelled datapoints → 75% training, 25% validation
- 1 point → (200 points {strain, stress, flag}; 2 parameters)

Example from Material-Sciences: Network [8]

- Using Hyperparameteroptimization:
  - Batch Size 75, Learning Rate: $4.7331*10^{-4}$
  - 153,441 parameters → 0.61MB parameter size
- → „large“ dataset, „small“ network → data parallelism preffered



Topology generated using: L. Roeder. „Netron“ netron.app (accessed Aug. 1, 2022)

Training on Taurus

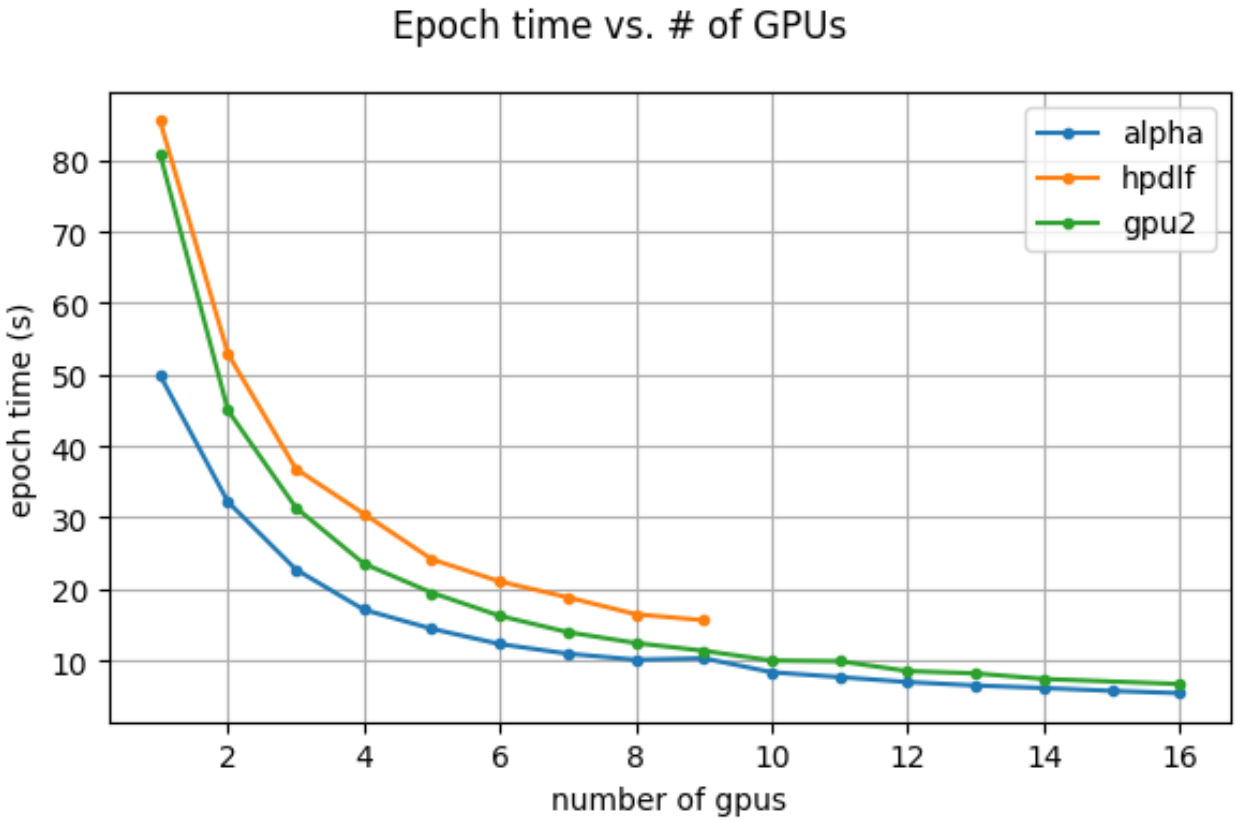
- Partitions: Alpha, GPU2, HPDLF
- Using *NCCL* and *MPI*
- **NCCL Advantages:** [9]
  - *Intra-Node*: uses aggregated NVLinks, PCIe and shared-memory
  - *Inter-Node*: uses aggregated network interfaces (TCP und RDMA)
  - Automatic topology detection
  - Compatible with *MPI*

Computing Resources

- Runs executed in SLURM *exclusive* mode
- **#CPUs** = (maxCPUs / maxGPUs) * #GPUs
- **Memory**: all available Memory on the Node
- **#GPUs**: variable
- Default: **NCCL**

Speedup und Efficiency, sequential Distribution of GPUs

- **Sequential distribution**: All GPUs of a node are used, before allocating another node

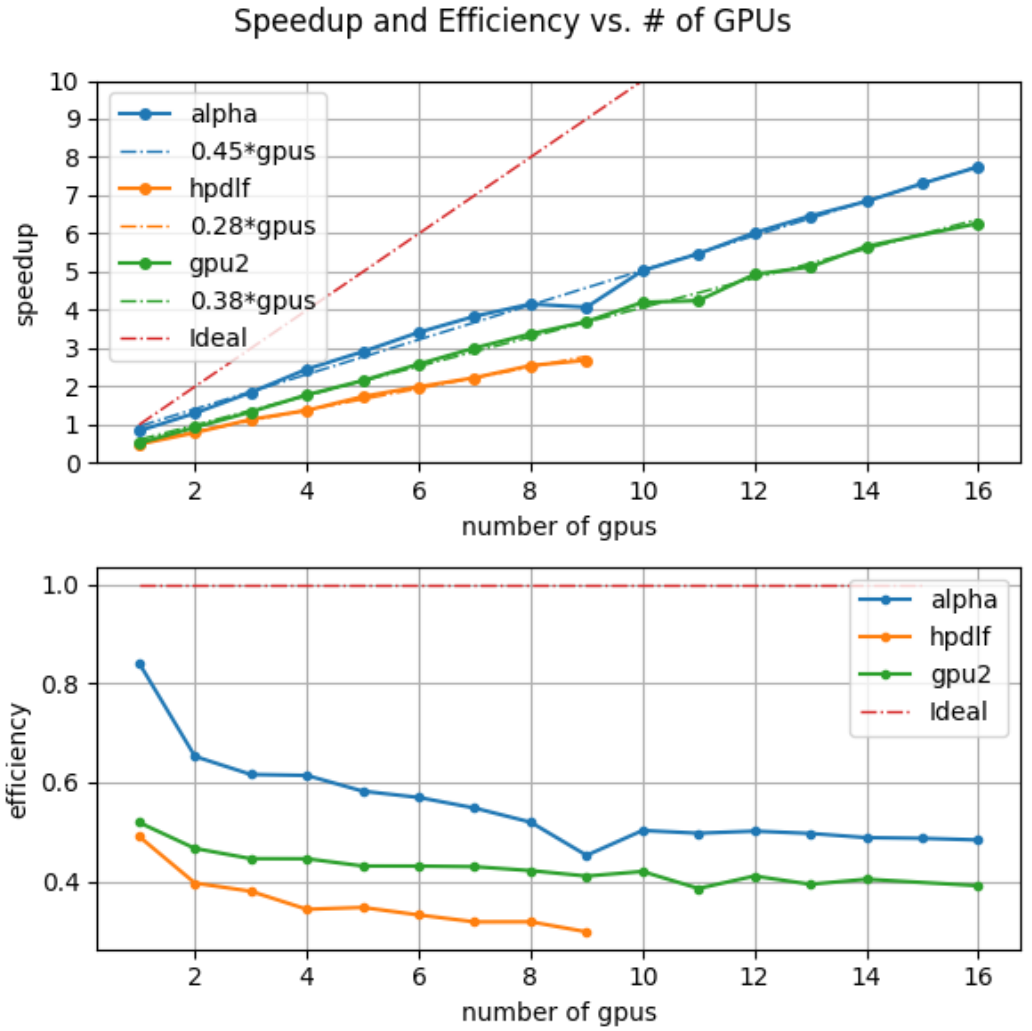


Speedup and Efficiency, sequential

- **Baseline**: Alpha, 1 GPU, without Horovod

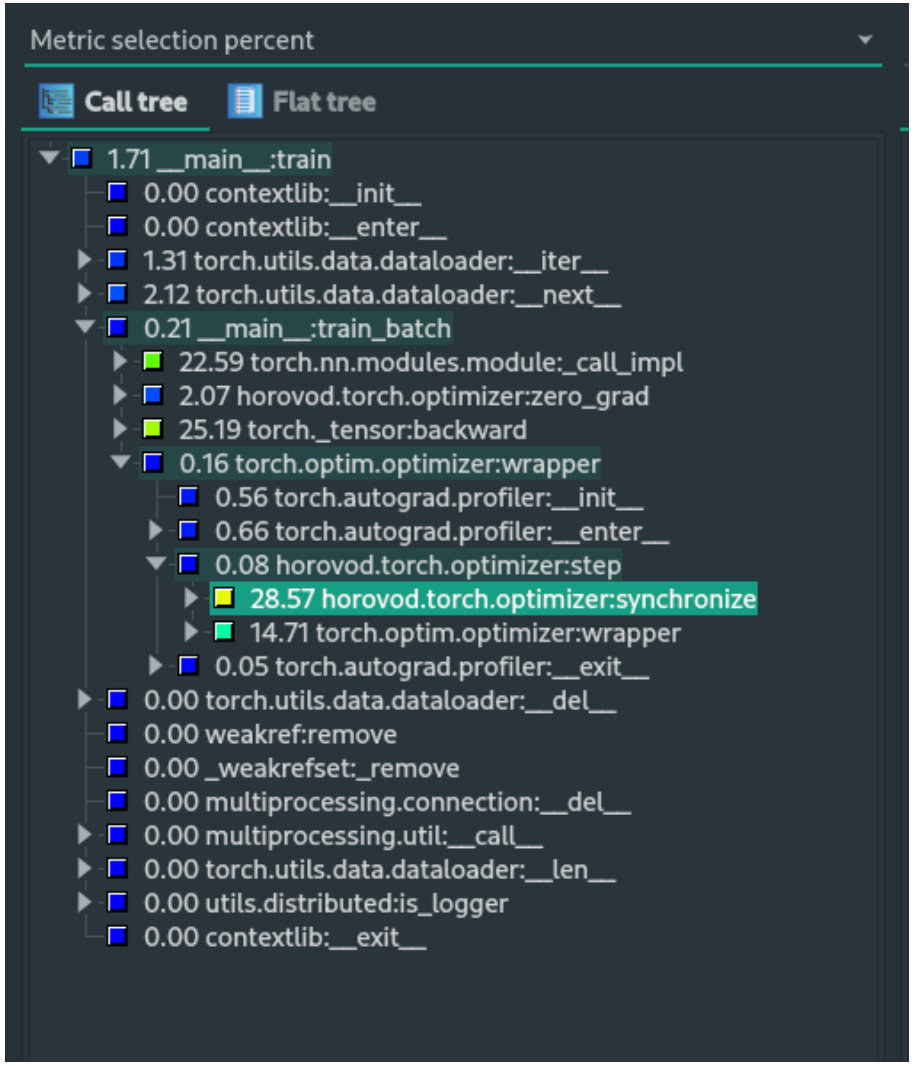
- Speedup ~linear, Speedup per GPU:
  - **Alpha**: 0.45 (MPI: 0.44)
  - **HPDLF**: 0.28 (MPI: 0.26)
  - **GPU2**: 0.38

- MPI scales marginally worse than NCCL
- Horovod on 1 GPU imposes overhead, ~17%
- *Why does scaling not come close to the promised 88%?*



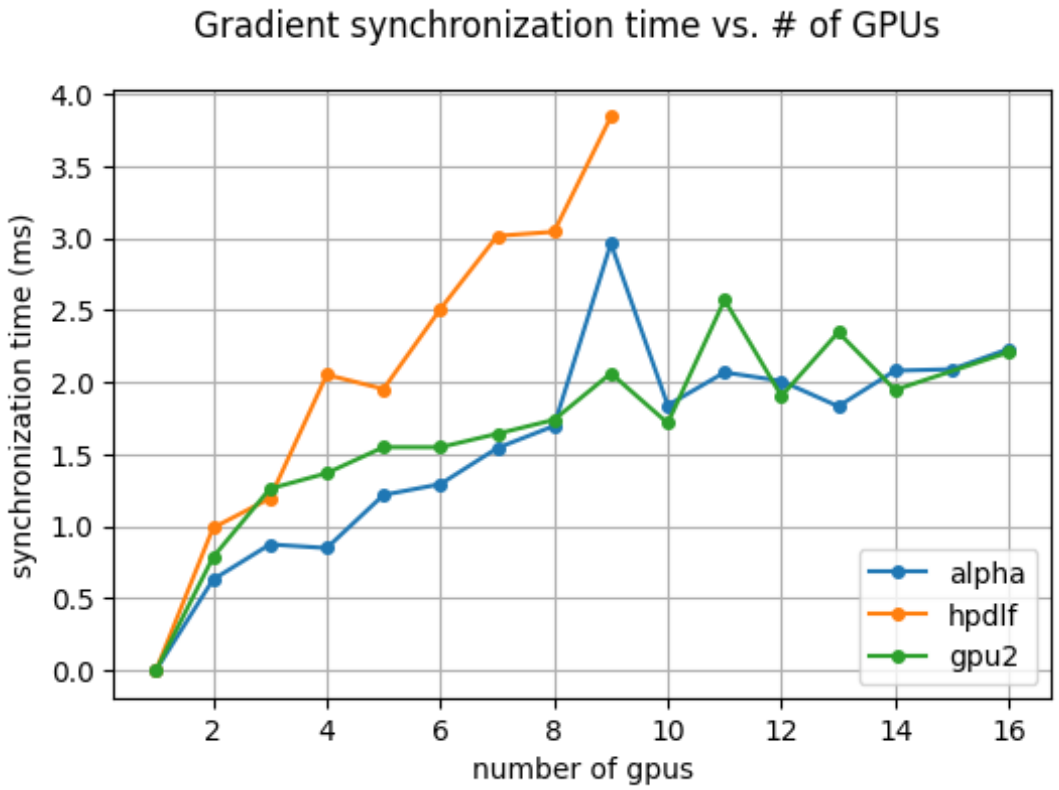
Profiling

- 2 Nodes, 6 GPUs, HPDLF
- **28%** of training runtime used by gradient synchronization
- Loading data only takes 2% → data already in RAM



Horovod Communication Overhead

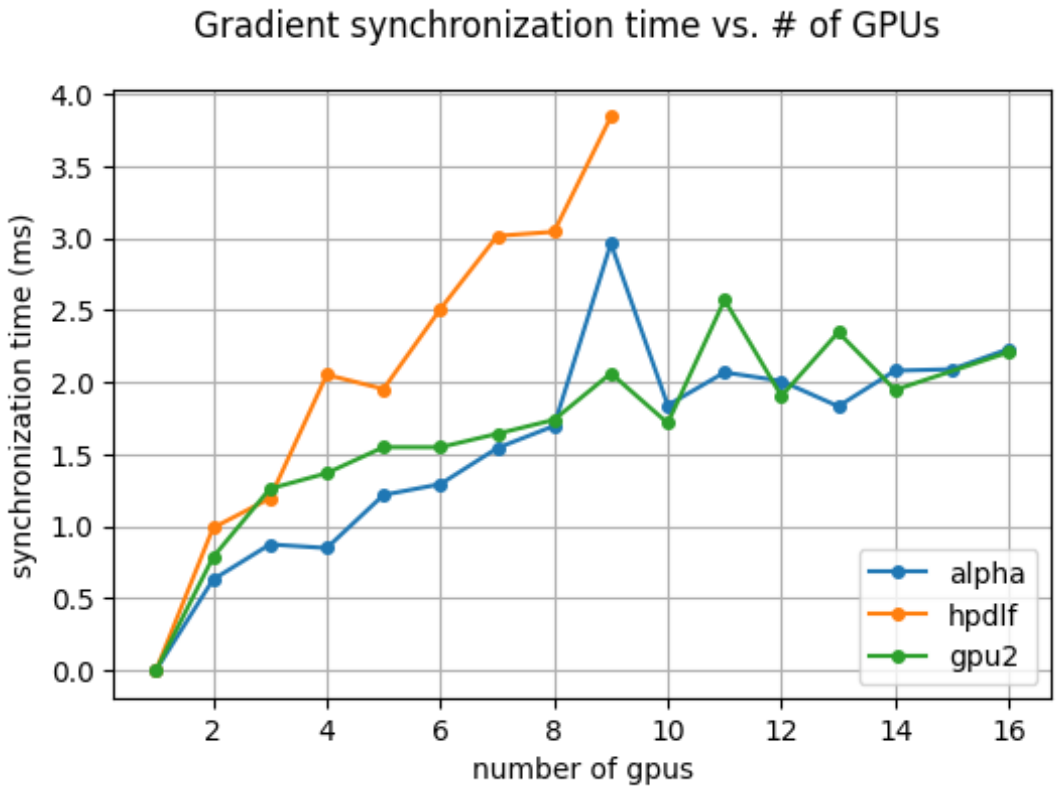
- Synchronization of gradients before updating parameters
- „synchronization time“ = time for synchronizing gradients



Observations:
- Time increases w.r.t. number of gpus
- Steeper gradient for HPDLF
- Curve flattens for GPU2 und Alpha

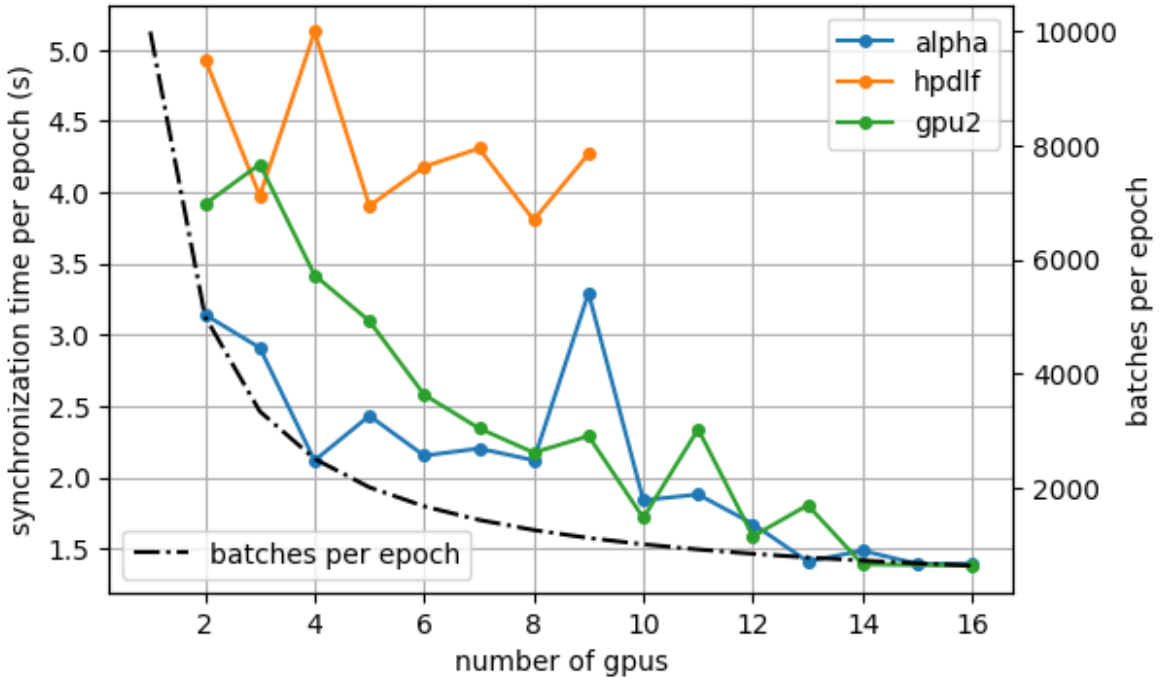
Horovod Communication Overhead

- Number of **Synchronizations** per epoch equal to **number of batches** per epoch
  - Small batch size, large dataset → many synchronizations
  - 10 000 batches per epoch for full dataset
  - On 2 workers:
    - Data partitioned into 2
    - 5 000 batches * 1ms = 5s per epoch

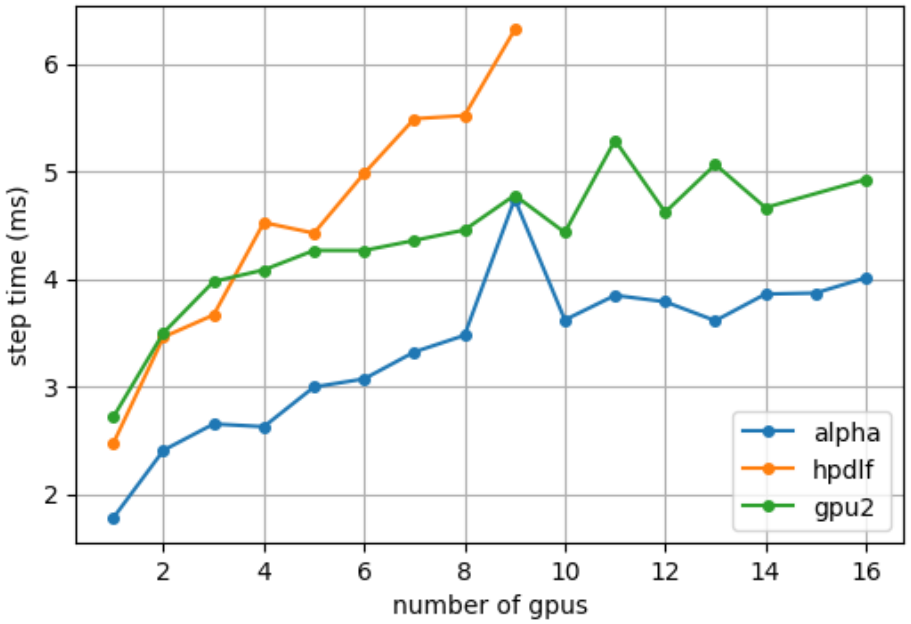


Horovod Communication Overhead

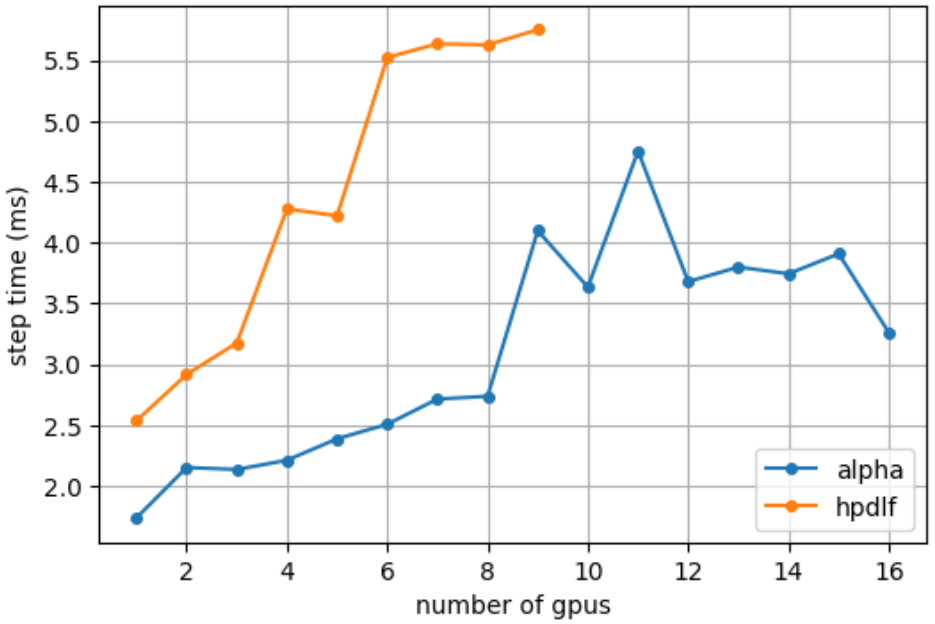
- Number of **Synchronizations** per epoch equal to **number of batches** per epoch
  - Number of Batches per epoch descreases due to data split
  - Less synchronizations
  - Number of synchronizations decreases faster than duration per synchronization increases → overall decrease



NCCL



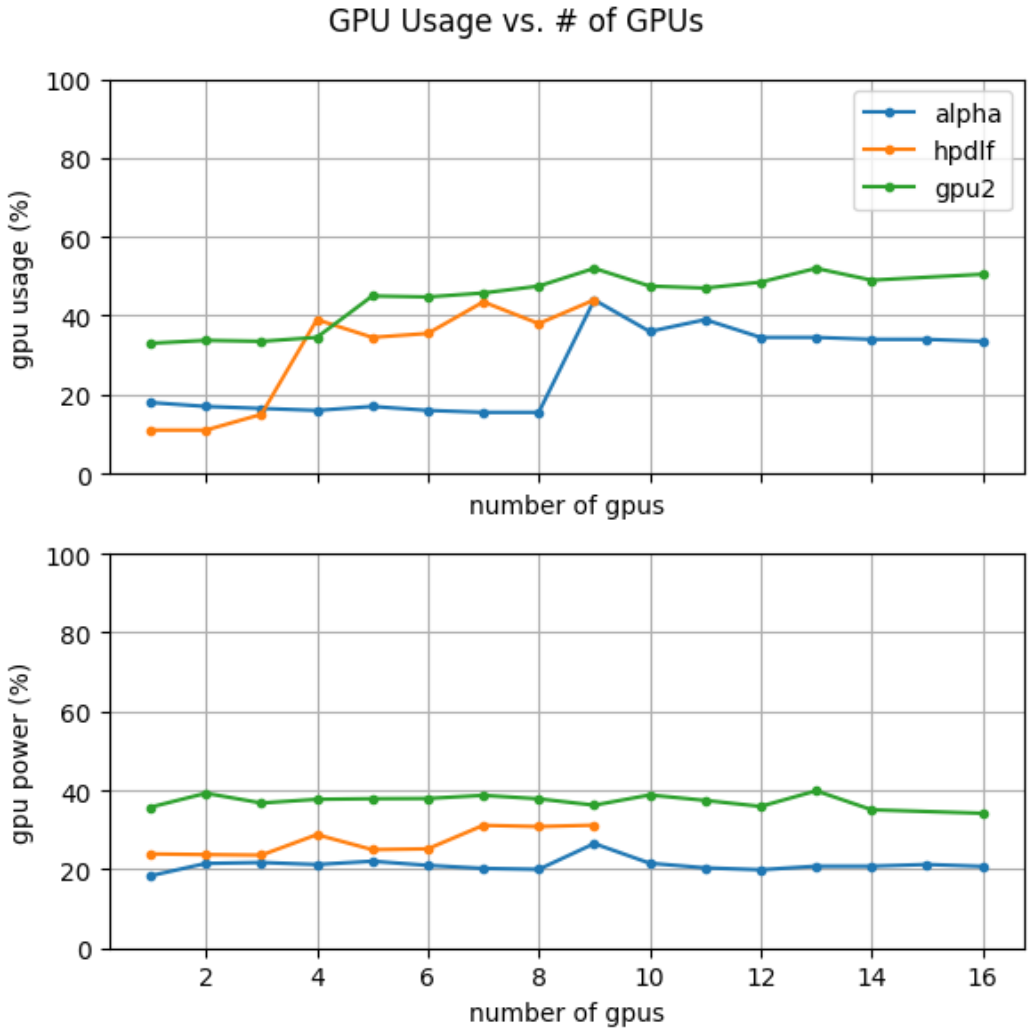
MPI



- MPI more efficient intra-node
- NCCL more efficient inter-node
- Counterintuitive, NCCL should use intra-GPU links (NVLink) → MPI is more efficient for small tensors [11]

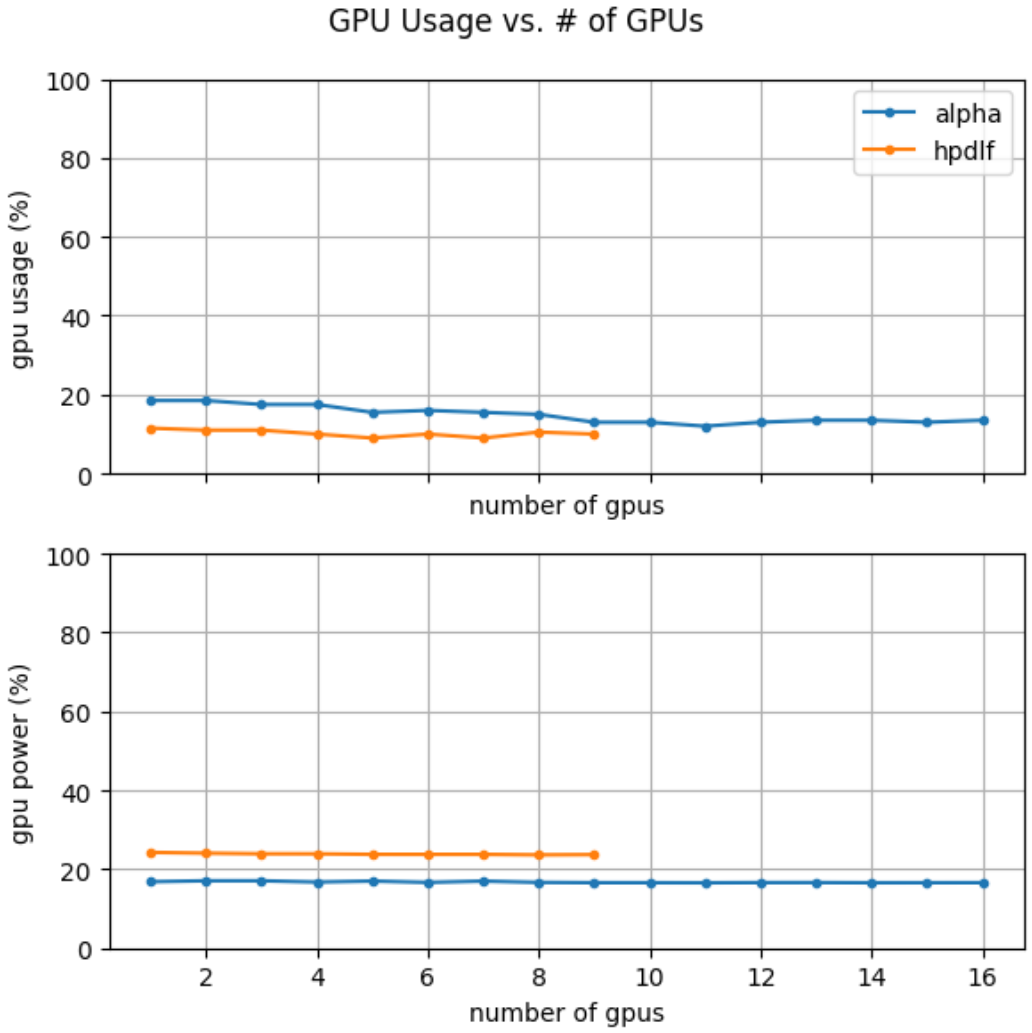
System Usage

- Usage suddenly increases for Alpha/HPDLF when **first**, additional Node is allocated
  - No impact on speedup
  - NCCL?
- Low overall GPU usage due to:
  - small Batch Size
  - small Netzwerk
  - → Overhead CPU-GPU communication



Comparison with MPI

- No sudden increase in GPU Usage
- Similar low GPU usage



Optimizations

- **Increasing Batch Size**

  + Less synchronizations per Epoch

  + Higher GPU usage

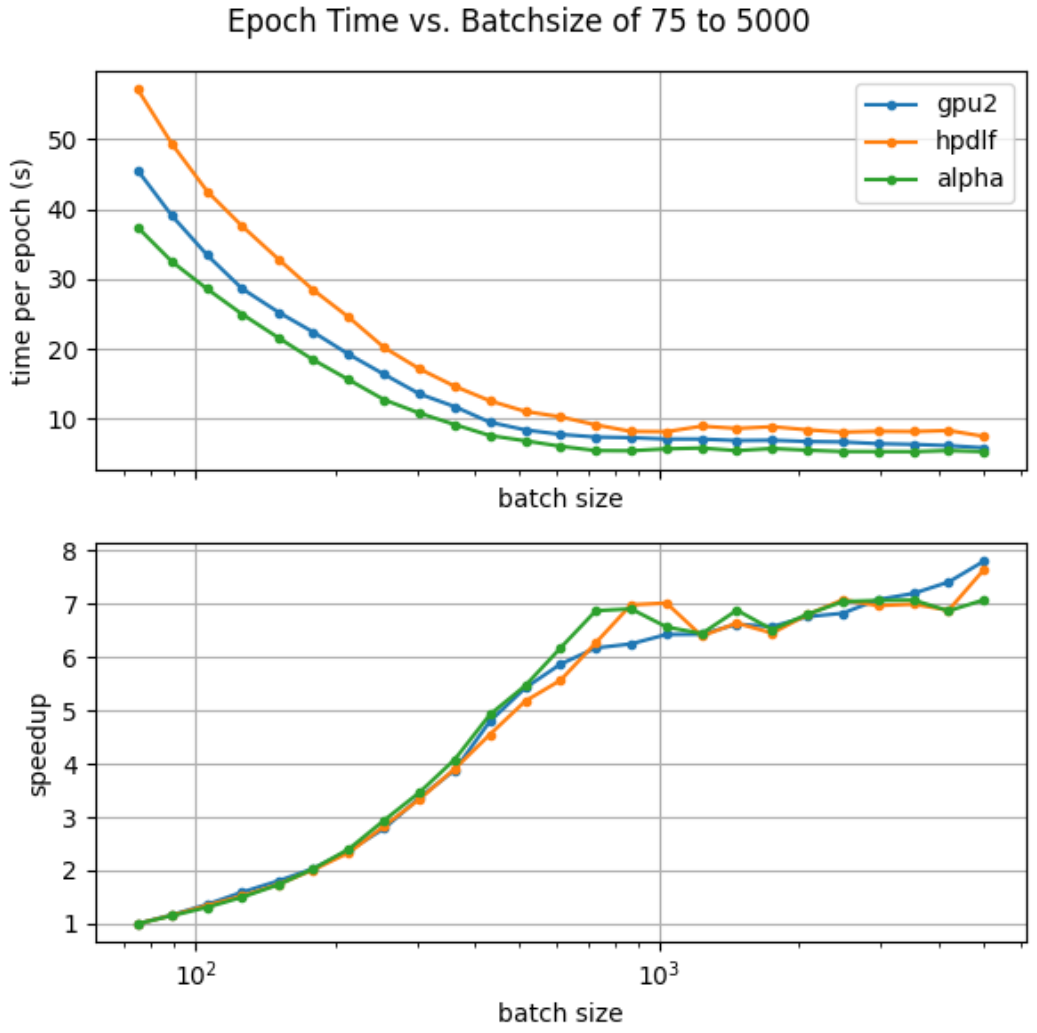
  - Worse accuracy with found hyperparameters (new HP search)

  - Possibly generalization-gap with large Batch-Sizes [10]

- **Move dataset into GPU-Memory**
  - Dataset only ~4GB, enough memory available on each partition
  - Decrease in data-loading time

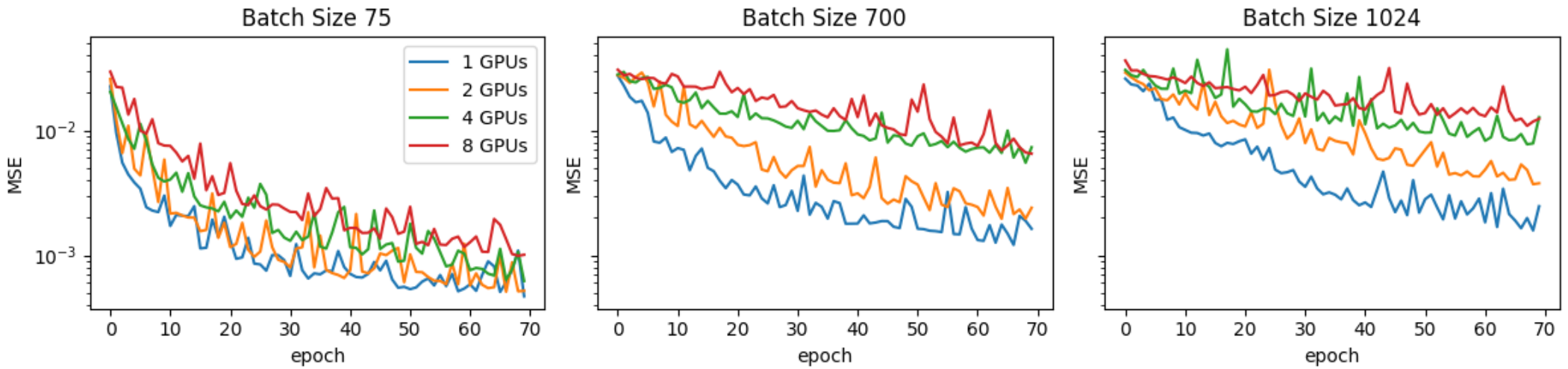
Optimizations – Batch Size

- **Test on one GPU**
  - 7x Speedup for Batch Size 1000



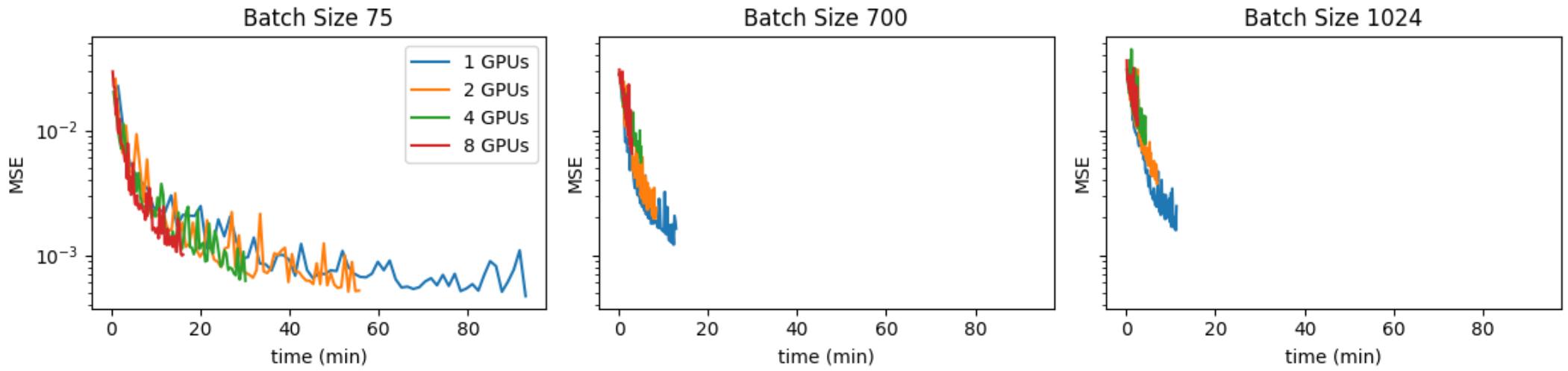
Optimizations – Batch Size

- Worse Accuracy with same number of epochs
- effective_batch_size = batch_size * n_workers
- Less gradient updates per epoch



Optimizations – Batch Size

- Accuracy w.r.t. Time
  - Don’t use epochs but training time as metric for ending training
  - Increase LR by sqrt(k), when increasing Batch Size by k [1]



Next Steps

- Train for the same time with increased batch size
- Scale learning rate
- Optimize HPs for larger batches

Thank you for your time.
Any questions?

Literaturverzeichnis

[1] A. Krizhevsky, “One weird trick for parallelizing convolutional neural networks,” arXiv e-prints, p. earXiv:1404.5997, Apr. 2014.

[2] T. Ben-Nun and T. Hoefler, Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. arXiv, 2018. doi: 10.48550/ARXIV.1802.09941.

[3] A. Sergeev and M. Del Balso, Horovod: fast and easy distributed deep learning in TensorFlow. arXiv, 2018. doi: 10.48550/ARXIV.1802.05799.

[4] J. Dean et al., “Large scale distributed deep networks,” Advances in neural information processing systems, vol. 25, 2012.

[5] P. Patarasuk and X. Yuan, “Bandwidth Optimal All-Reduce Algorithms for Clusters of Workstations,” J. Parallel Distrib. Comput., vol. 69, no. 2, pp. 117–124, Feb. 2009, doi: 10.1016/j.jpdc.2008.09.002.

[6] A. Gibiansky, "Bringing HPC Techniques to Deep Learning - Andrew Gibiansky", andrew.gibiansky.com, 2017. [Online]. Available: https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/.

[7] Martín Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016, arXiv:1603.04467.

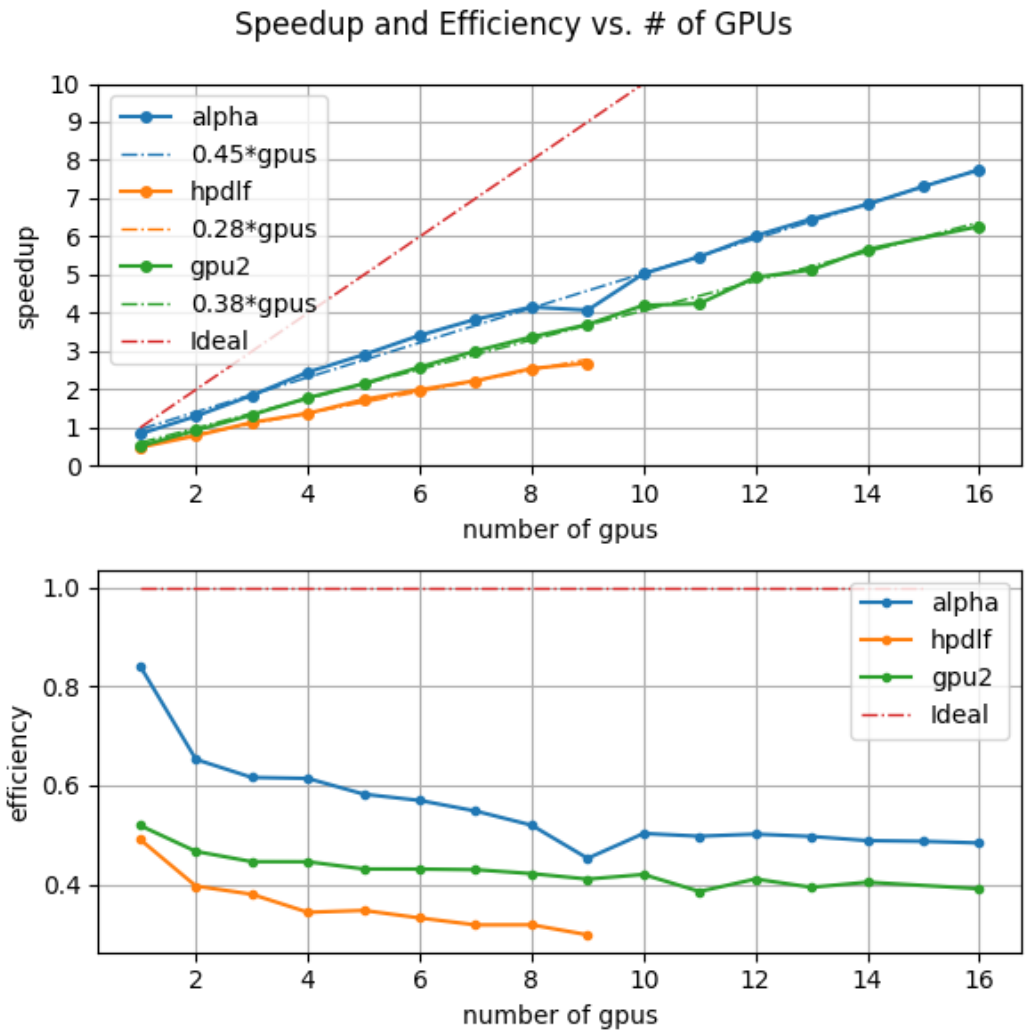
[8] P. Winkler, N. Koch, A. Hornig, and J. Gerritzen, “OmniOpt – A Tool for Hyperparameter Optimization on HPC,” in High Performance Computing, 2021, pp. 285–296.

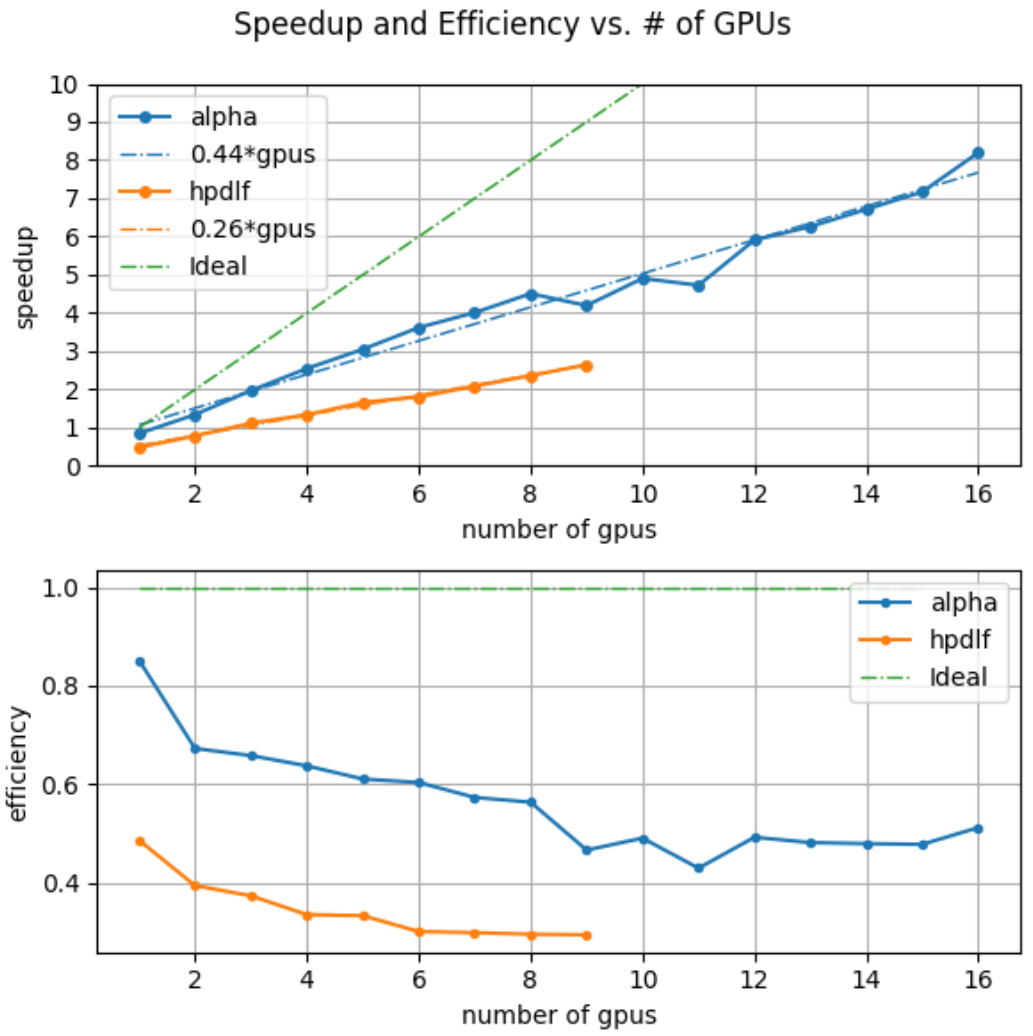
[9] S. Jeaugey, „Scaling Deep Learning Training with NCCL“, developer.nvidia.com, 2018. [Online]. Available: https://developer.nvidia.com/blog/scaling-deep-learning-training-nccl

[10] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima,” CoRR, vol. abs/1609.04836, 2016, [Online]. Available: http://arxiv.org/abs/1609.04836

[11] E. Hölzl. „Communication Backends, Raw performance benchmarking“, mlbench.github.io, 2020. [Online]. Available: https://mlbench.github.io/2020/09/08/communication-backend-comparison/

Speedup und Efficiency, MPI





Gradient Synchronization Time NCCL vs MPI