TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Center for Information Services and High Performance Computing (ZIH)

Zwischenpräsentation Bachelorarbeit

GPU-Parallelisierung auf Basis von Horovod - eine Skalierungs- und Performance Analyse anhand eines Beispiels aus den Materialwissenschaften

Paul Orlob

ZIH
Center for Information Services & High Performance Computing

Gliederung

1) Einleitung
2) Model- & Data Parallelism
3) Horovod
4) Beispiel Materialwissenschaften
5) Speedup & Efficiency
6) Performance Analyse
7) Optimierungen
8) Nächste Schritte

Motivation

- **Problem**: Immer größere Netzwerke + Große Datenmengen → Lange Trainingszeiten
  - Besonders problematisch bei Hyperparameteroptimierung
- **Lösung**: GPU-Parallelisierung, Nutzung mehrerer GPUs für Training

Data Parallelism [1, 2, 4]

- **N** workers
- Jeder Worker besitzt Kopie des Netzwerkes
- Aufteilen der Trainingsdaten in **N** Partitionen
- Parallele Berechnung des Forward- und Backward Pass
- → Synchronisation der **N** Parameter-Gradienten am Ende des Backward Pass eines Batches
- → Anzahl der Synchronisationen = Anzahl der Batches

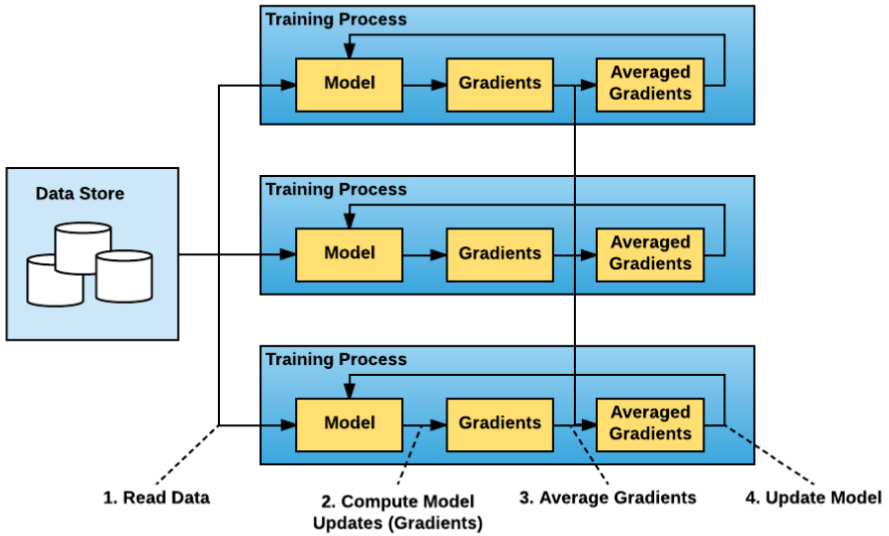


Figure 1: Data parallel training [3]

Model Parallelism [1, 2, 4]

- **N** workers
- Jeder Worker besitzt alle Trainingsdaten
- Aufteilen des Netzwerkes in **M** Partitionen
- Parallele Berechnung von Netzwerkschichten
- → Synchronisation wenn Ergebnisse von anderen Workern benötigt werden

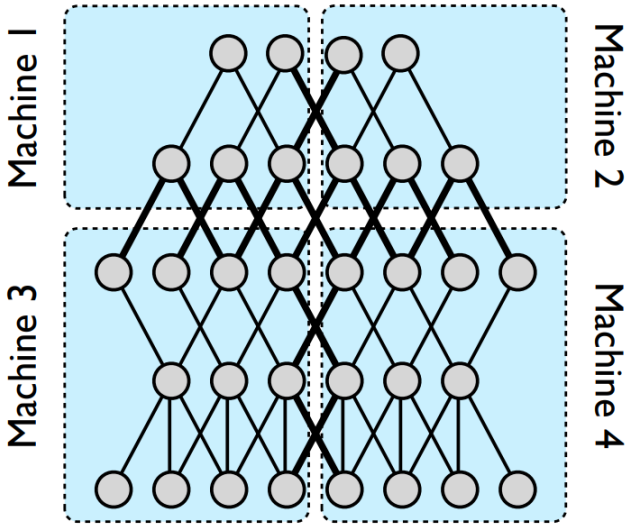


Image : J. Dean et al., “Large scale distributed deep networks,” Advances in neural information processing systems, vol. 25, 2012.

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

Data Parallelism	Model Parallelism
Besser wenn viel Rechenleistung pro Parameter	Besser wenn viel Rechenleistung pro Neuron Output
Anzahl der Synchronisationen abhängig von Anzahl der Batches	Anzahl der Synchronisationen abhängig von Netzwerk Topologie
-	Verringert Größe des Netzwerkes auf GPU
Bis auf Synchronisation der Gradienten vollständig parallel	Nicht vollständig parallel, da oft auf Ergebnis des vorherigen Layers (Workers) gewartet werden muss

Horovod [3]

- Library von **Uber**, **2017** veröffentlicht, **Apache 2.0** Lizenz
- Unterstützt Tensorflow, Keras, PyTorch, MXNet
- Unterstützt Data Parallelism
- **Motivation:**
  - TensorFlow Distributed verwendet Parameter Server [7]
  - Entwicklung schwierig: Anzahl der Parameter Server; Boilerplate Code
- Nutzt Algorithmus „ring-allreduce“, optimale Nutzung des Netzwerks bei ausreichender Buffergröße [5, 6]
- „ring-allreduce“ implementiert in MPI und NCCL (NVIDIA Collective Communication Library)
- → 88% effizente Skalierung des Trainings von Inception V3, ResNet-101

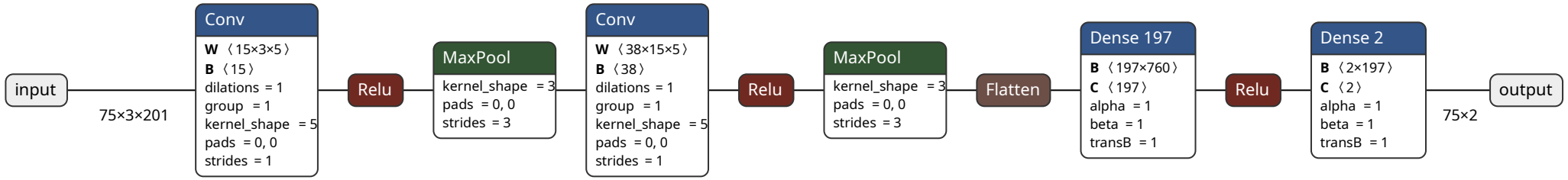
Beispiel Materialwissenschaften: Daten [8]

- Vorhersage von Materialparametern aus Spannungs-Dehnungs-Diagramm

- 1 Mio. gelabelte Datenpunkte → 75% Training, 25% Test
- 1 Datenpunkt → (200 Punkte {Strain, Stress, Flag}; 2 Materialparameter)

Beispiel Materialwissenschaften: Netzwerk [8]

- Mit Hilfe von Hyperparameteroptimierung optimiert:
  - Batch Size 75, Learning Rate: $4.7331*10^{-4}$
  - 153,441 Parameter → 0.61MB Parameter Größe
- → viele Daten, „kleines“ Netzwerk → Data Parallelism vorteilhaft



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

Training auf Taurus

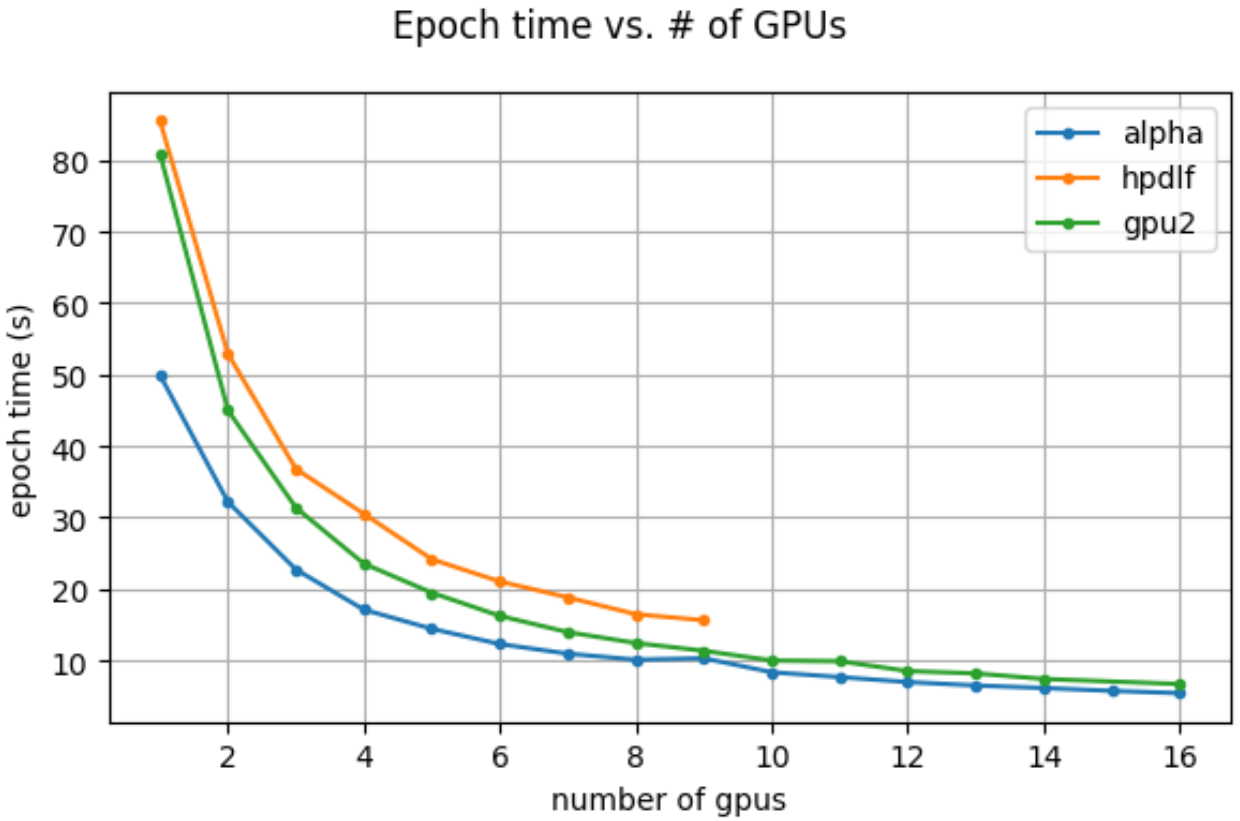
- Auf Partitionen: Alpha, GPU2, HPDLF
- Mit NCCL und NCCL
- **Vorteile NCCL: [9]**
  - *Intra-Node*: Nutzung von aggregierten NVLinks, PCIe und shared-memory
  - *Inter-Node*: Nutzung von aggregierten Netzwerk Interfaces (TCP und RDMA)
  - Automatische Topologie-Erkennung
  - MPI kompatibel

Hardware Zuweisung

- Runs im *exclusive* SLURM-Modus ausgeführt
- **#CPUs** = maxCPUs / #GPUs
- **Memory**: gesamten Memory eines Nodes zugeteilt
- **#GPUs**: variabel
- Standard: **NCCL**

Speedup und Efficiency, sequentielle Verteilung

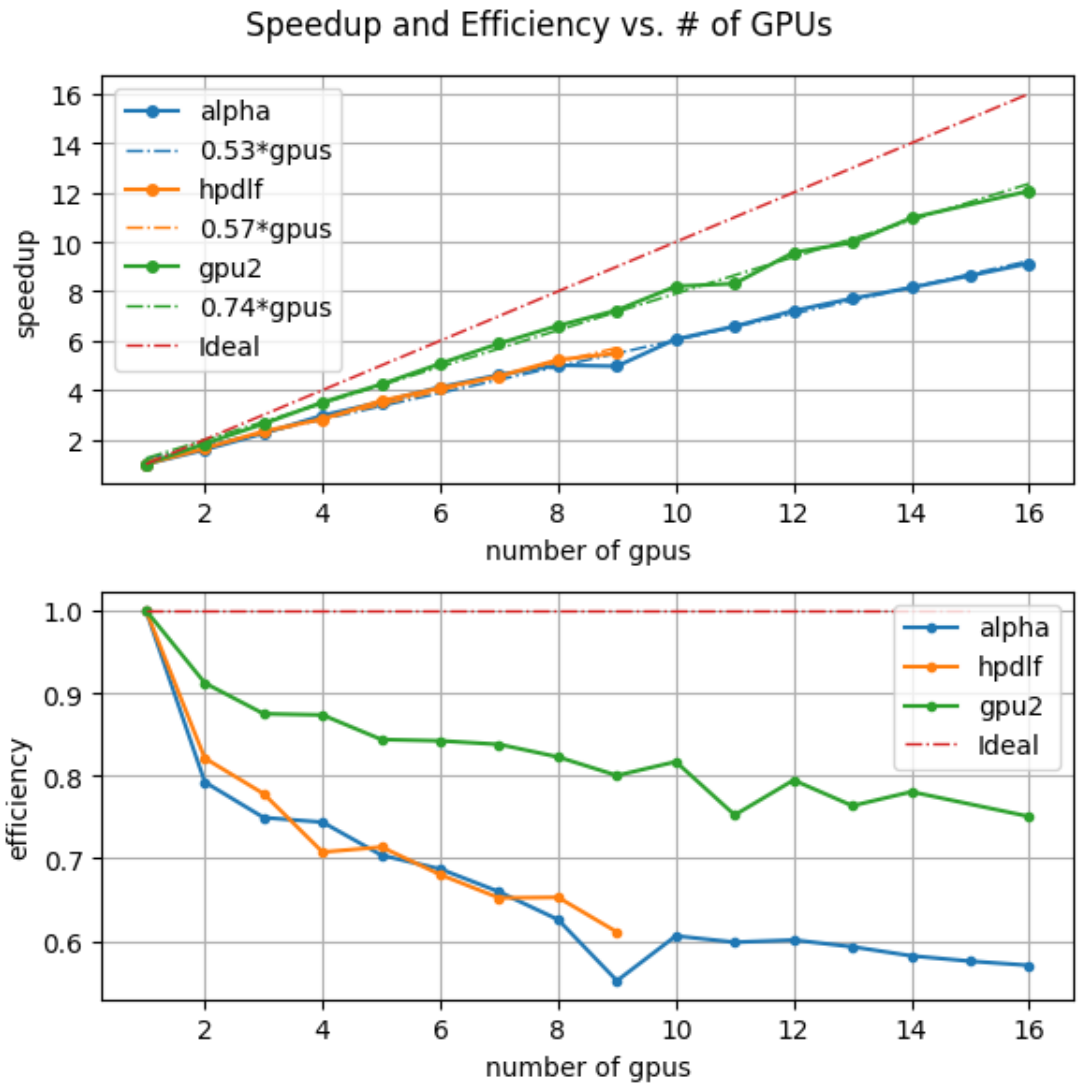
- **Sequentielle Verteilung**: Alle GPUs eines Nodes werden genutzt, bevor nächster Node verwendet wird



Speedup und Efficiency, sequentiell

- Speedup/Efficiency berechnet **pro Partition,** Baseline ist Run mit einer GPU auf der Partition

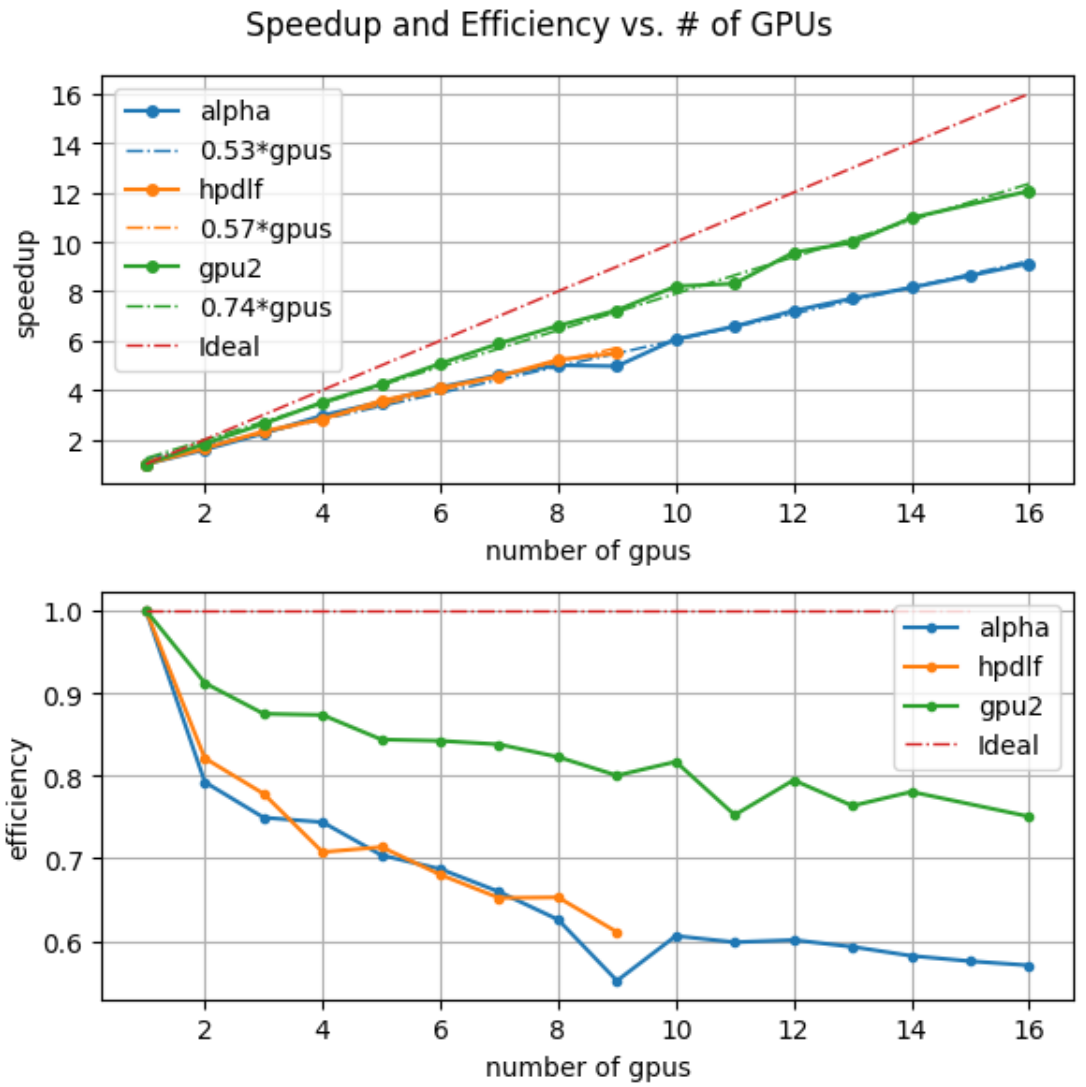
- Speedup ~linear, Speedup pro GPU:
  - **Alpha**: 0.53
  - **HPDLF**: 0.57
  - **GPU2**: 0.74



Speedup und Efficiency

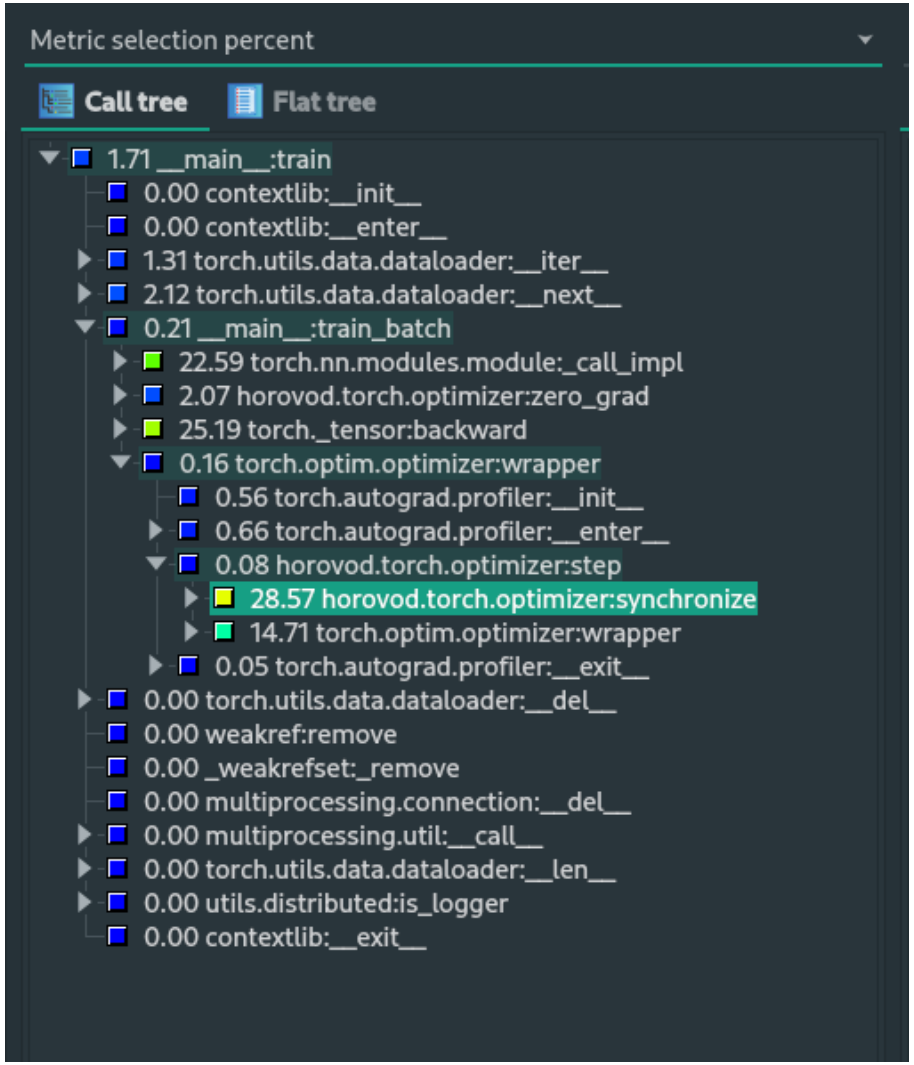
- Speedup/Efficiency berechnet **pro Partition,** Baseline ist Run mit einer GPU auf der Partition
- Speedup ~linear, Speedup pro GPU:
  - **Alpha**: 0.53 (MPI: 0.52)
  - **HPDLF**: 0.57 (MPI: 0.55)
  - **GPU2**: 0.74
- Geringer Unterschied NCCL vs MPI, geringe Anzahl der synchronisierten Gradienten

Offene Fragen:
- Warum bessere Skalierung auf GPU2 als auf Alpha/HPDLF?



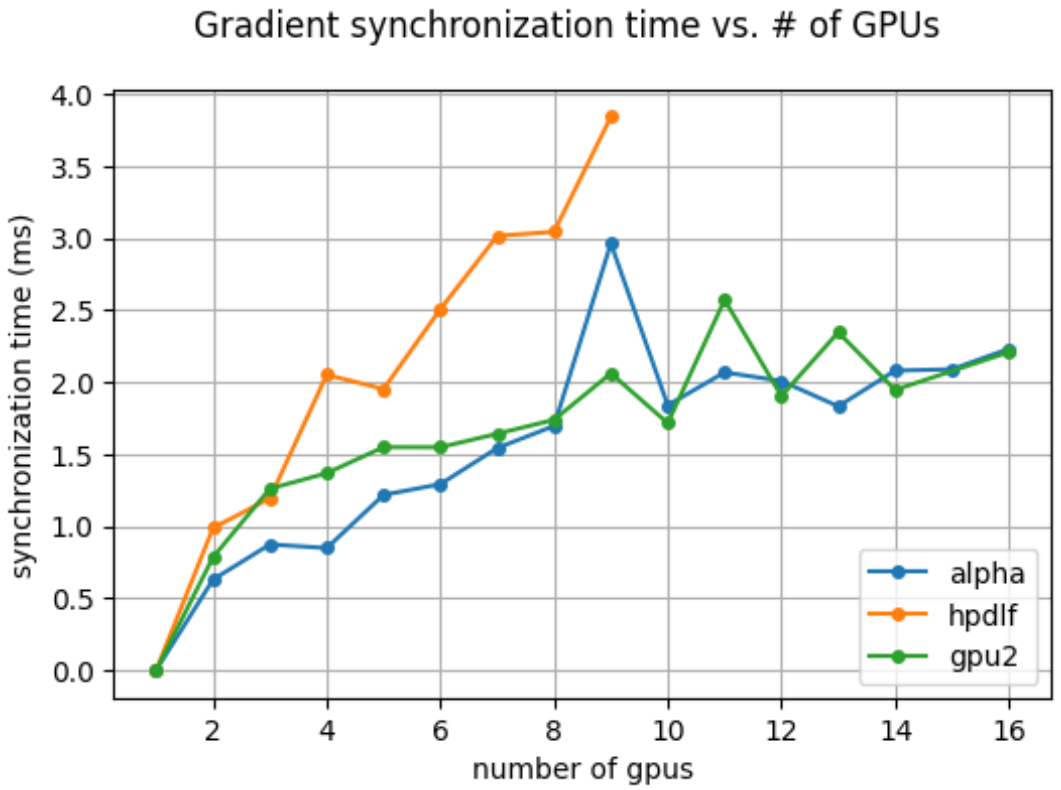
Profiling

- 2 Nodes, 6 GPUs, HPDLF
- **28%** der Zeit einer Epoche Synchronisation
- Laden der Daten nur 2% → Daten in RAM



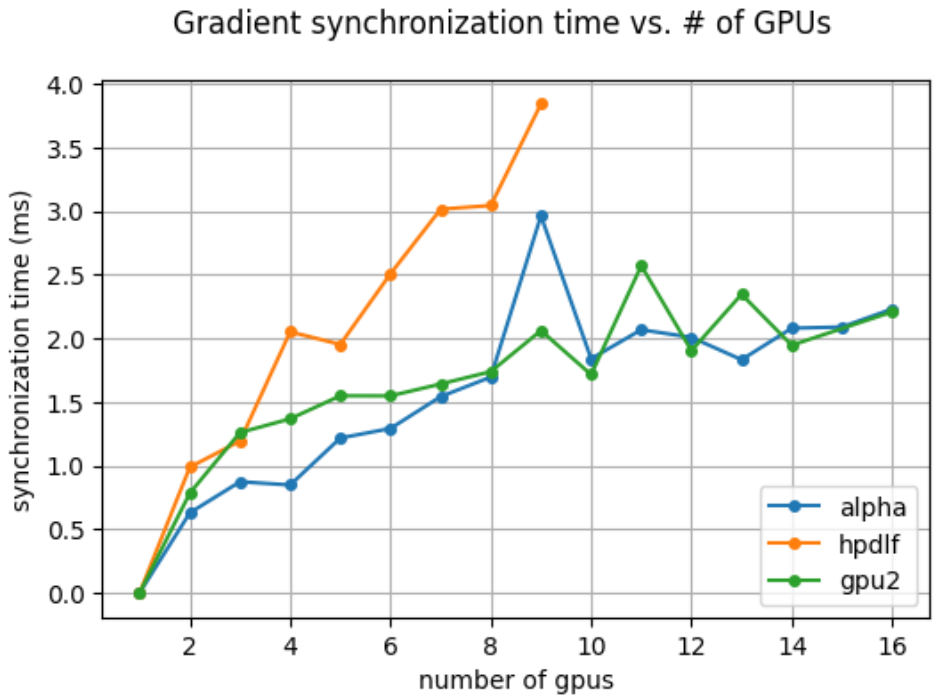
Horovod Communication Overhead

- Synchronisierung der Gradienten vor aktualisieren der Parameter des Netzwerkes
- „synchronization time“ ist Zeit für Synchronisation der Netzwerk Gradienten
- Im Moment noch etwas ungenau, da Parameter Update enthalten

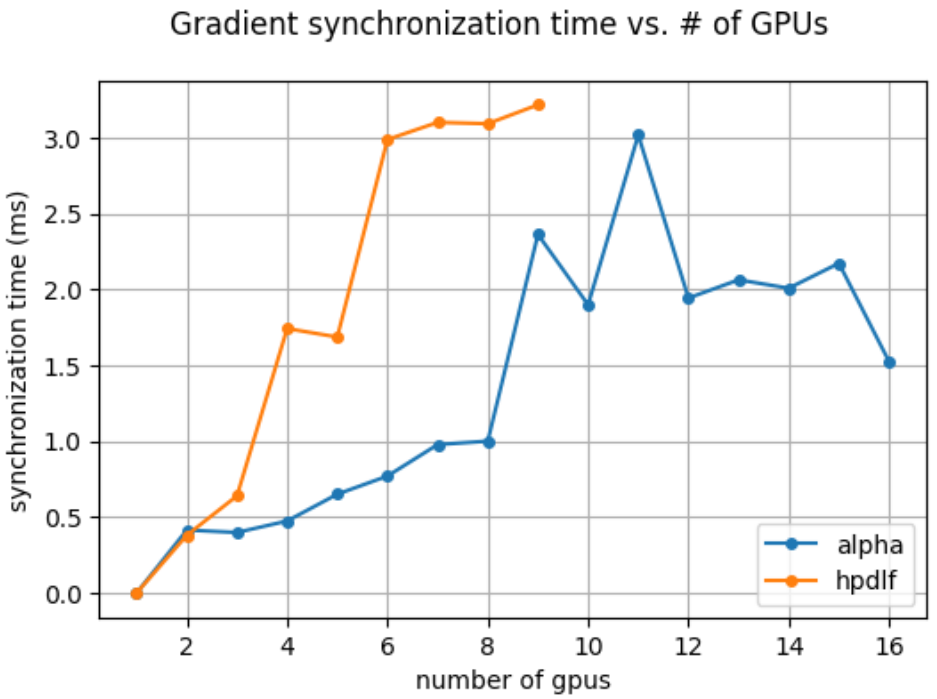


Beobachtung:
- Zeit steigt mit Anzahl der GPUs
- Steilerer Anstieg bei HPDLF
- Kurve flacht ab bei GPU2 und Alpha

NCCL



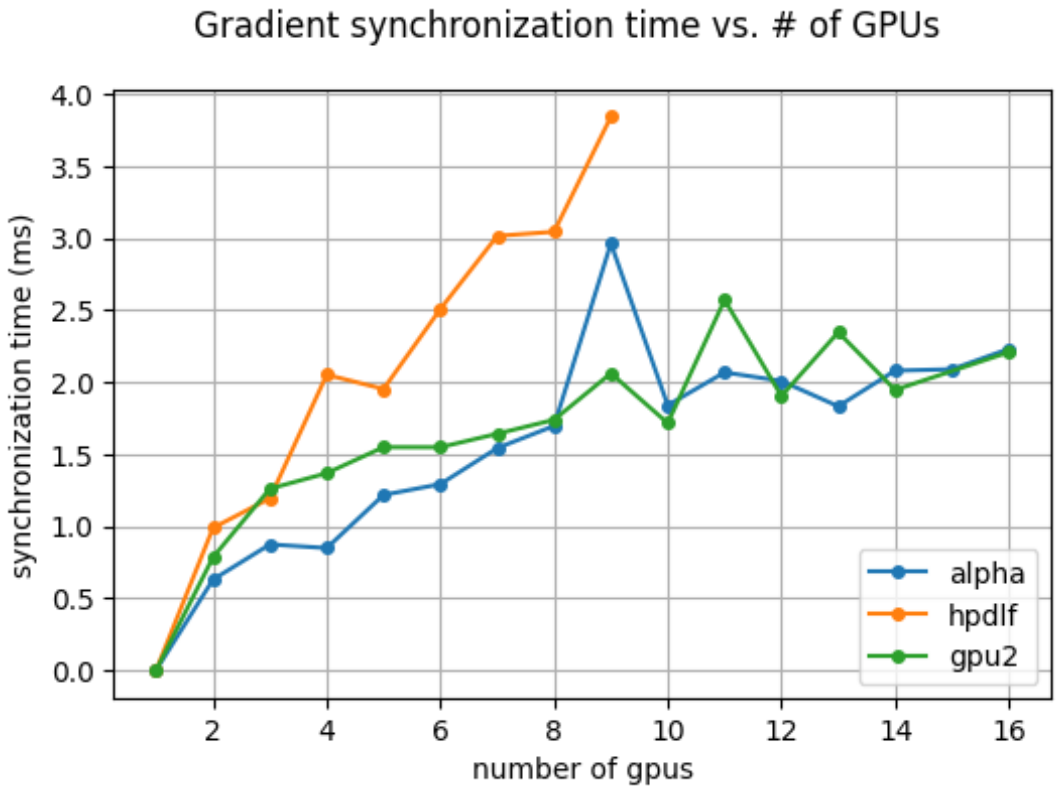
MPI



- MPI effizienter auf einem Node
- NCCL effizienter über mehrere Nodes

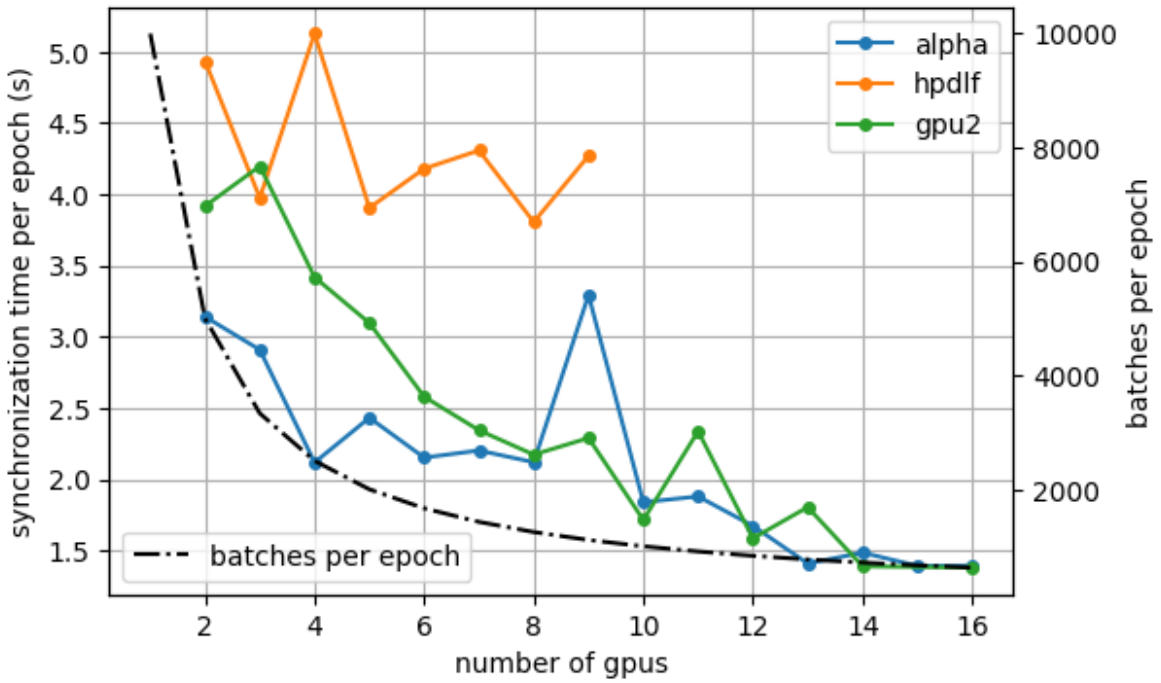
Horovod Communication Overhead

- Anzahl der **Synchronisierungen** pro Epoche entspricht **Anzahl der Batches** pro Epoche
  - Kleine Batch Size, viele Daten → viele Synchronisierungen
  - 750 000 / 75 = 10 000 Synchronisierungen



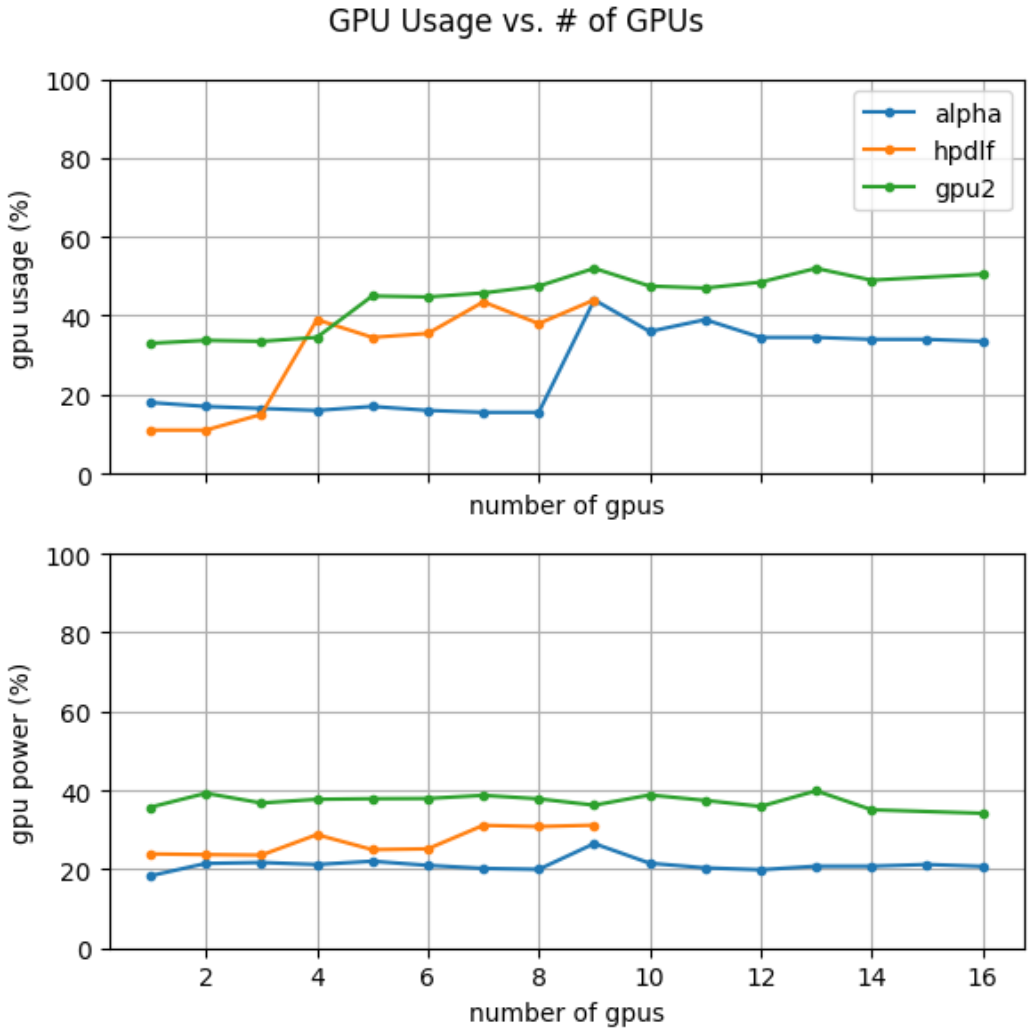
Horovod Communication Overhead

- Anzahl der **Synchronisierungen** pro Epoche entspricht **Anzahl der Batches** pro Epoche
  - Anzahl der Batches pro Epoche sinkt, da Daten aufgeteilt werden
  - Geringere Kosten für Synchronisation



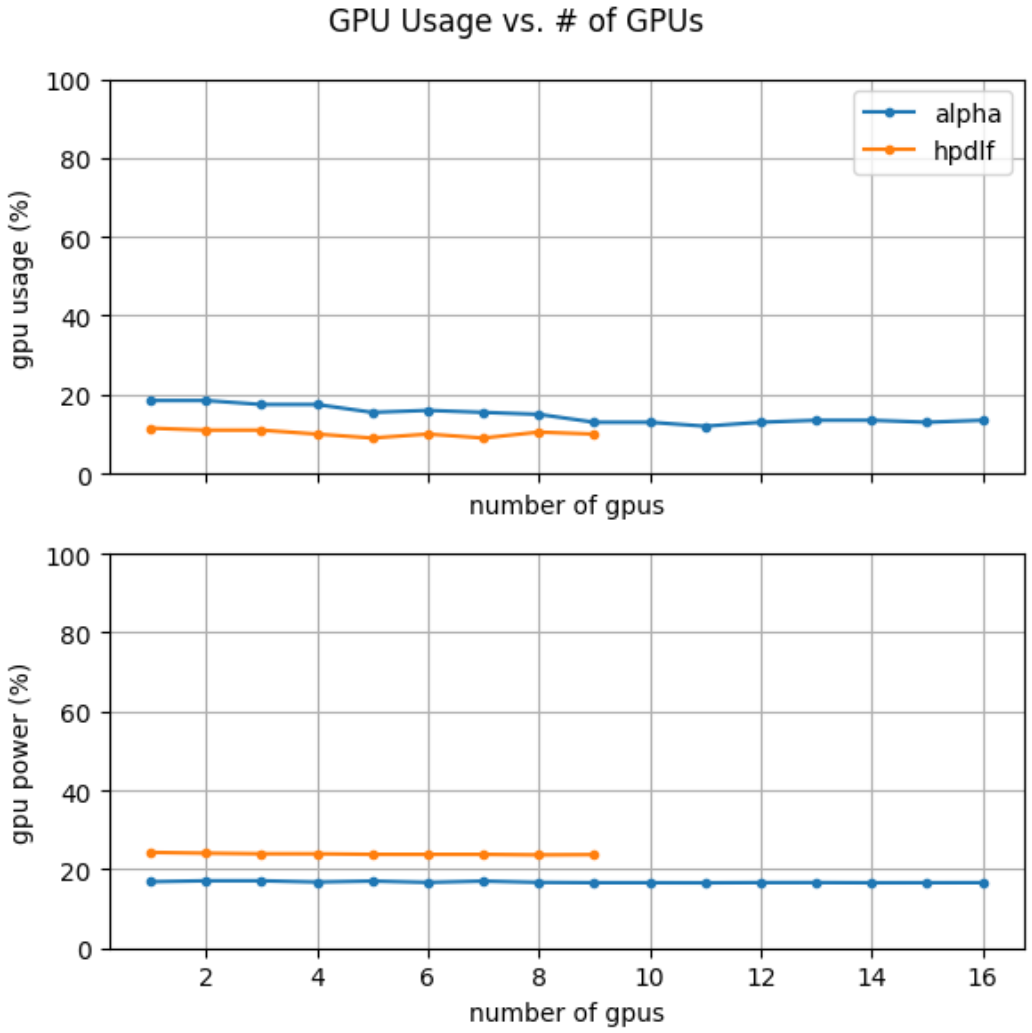
Auslastung

- Auslastung steigt bei Alpha/HPDLF sprunghaft an wenn **erster**, zusätzlicher Node genutzt wird
  - Keine Auswirkung auf Speedup
  - NCCL?
- GPU Auslastung generell sehr gering
  - Geringe Batch Size
  - Kleines Netzwerk
  - → Overhead CPU-GPU Kommunikation



Auslastung Vergleich mit MPI

- Bei Nutzung von MPI kein Sprung zu beobachten
- Ebenfalls kein Einfluss auf Performance



Optimierungen

- **Erhöhen der Batch Size**

  + Weniger Synchonisierungen pro Epoche

  + Bessere GPU Auslastung

  - Schlechtere Accuracy mit vorhandenen Hyperparametern (neue HP-Suche)

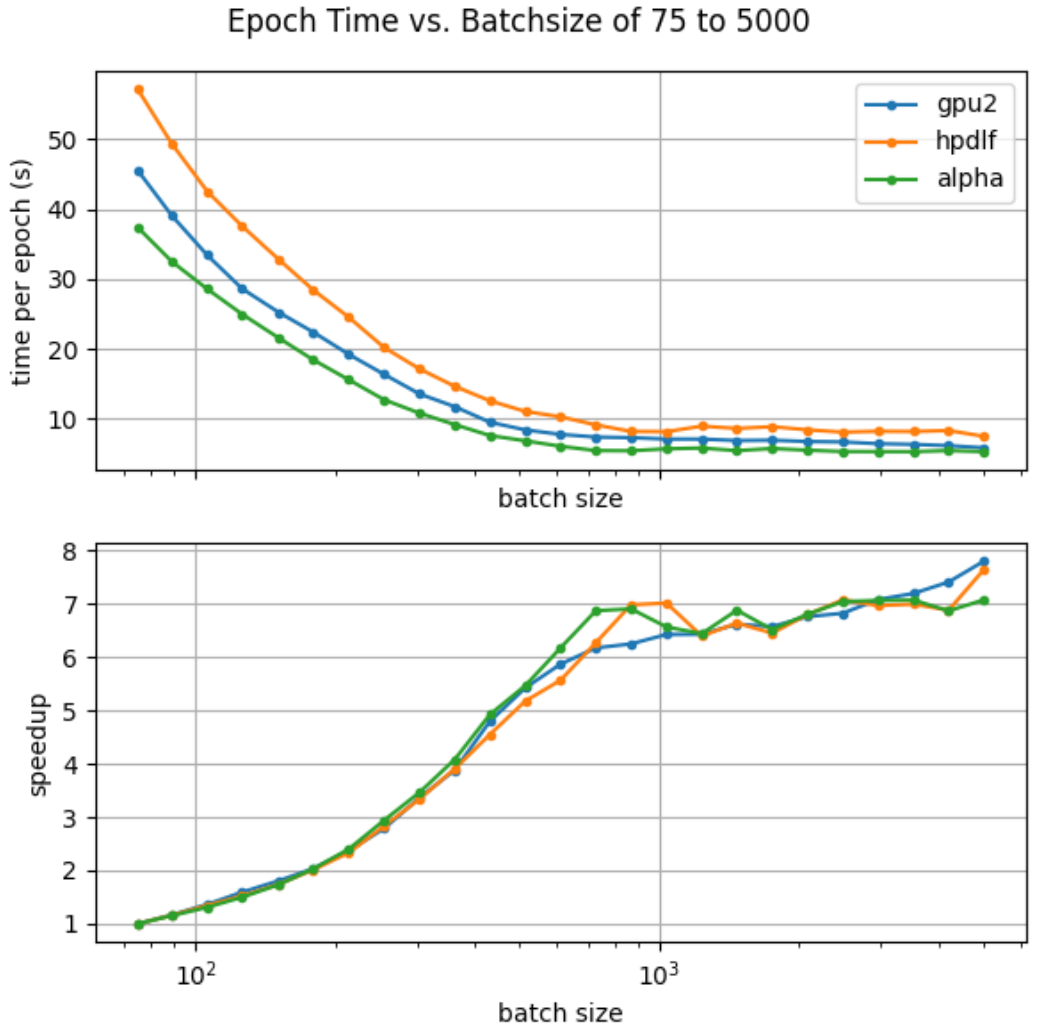
  - Möglicherweise Generalization-Gap mit hohen Batch-Sizes [10]

- **Gesamten Datensatz in GPU-Memory**

  - Ausreichend Platz, Datensatz ~4GB groß

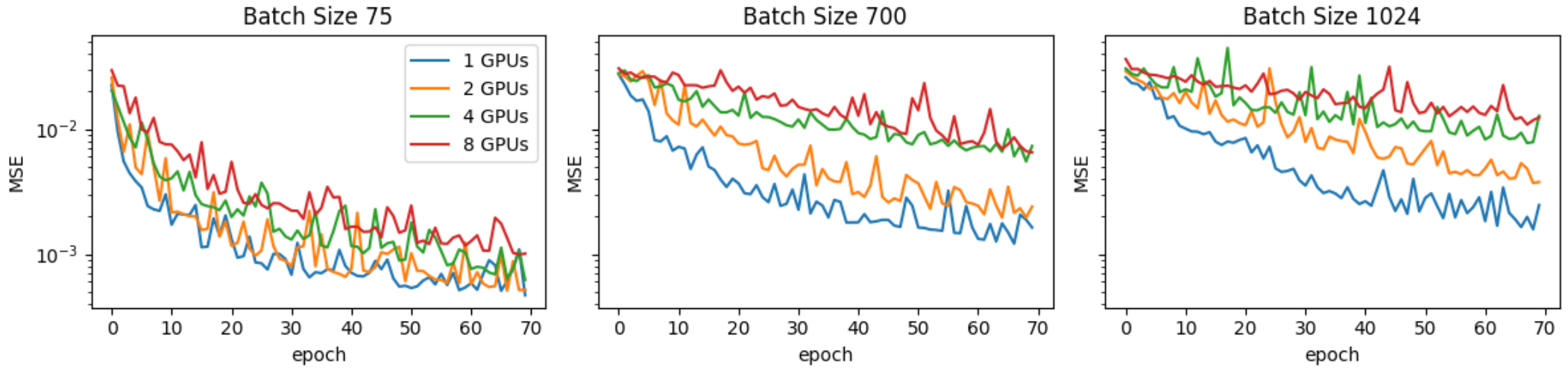
Optimierungen – Batch Size

- **Test auf einer GPU**
  - 7x Speedup bei Batch Size 1000



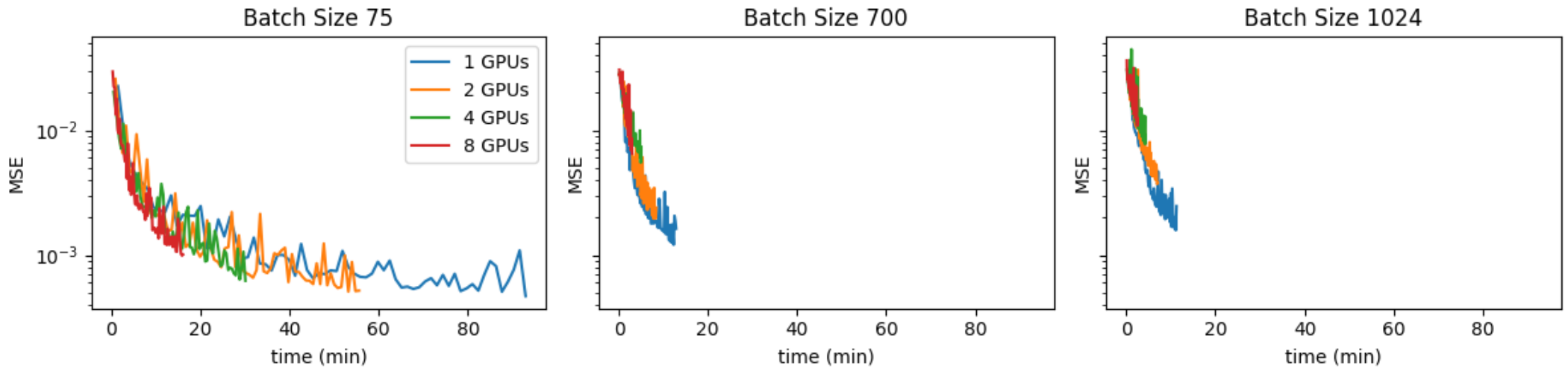
Optimierungen – Batch Size

- Schlechtere Accuracy nach gleicher Anzahl Epochen



Optimierungen – Batch Size

- Accuracy vs. Time
  - Ähnliche Kurven?
  - Wenn Batch Size um k erhöht wird LR um sqrt(k) erhöhen [1]



Nächste Schritte

- Batch Size erhöhen, gleiche Zeit trainieren → neue Daten für Scaling
- Learning Rate linear skalieren
- Ggfs. Hyperparameter optimieren
- Ursache für bessere Skalierung auf GPU2 (Batch Size?, Auslastung GPUs?)
- (Daten in GPU Memory)

Vielen Dank
Fragen?

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/.

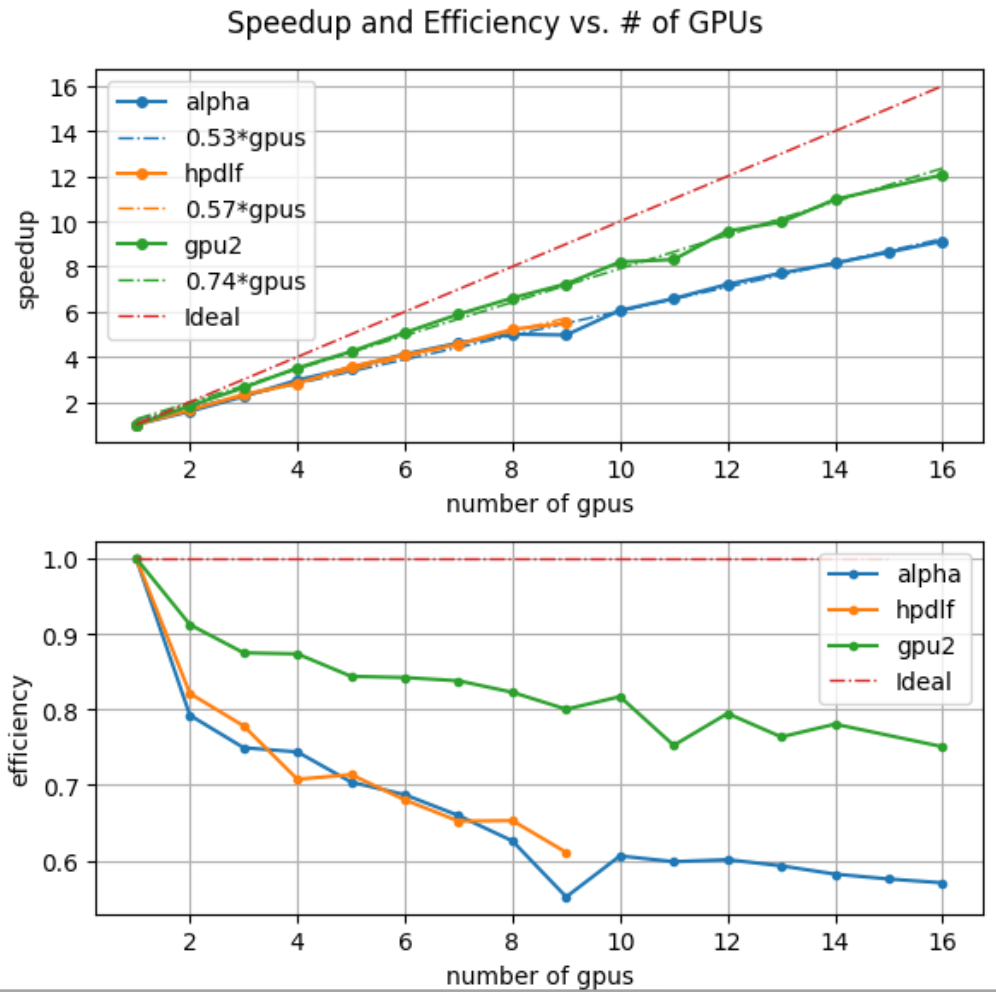
[7] Martín Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016, arXiv:1603.04467.

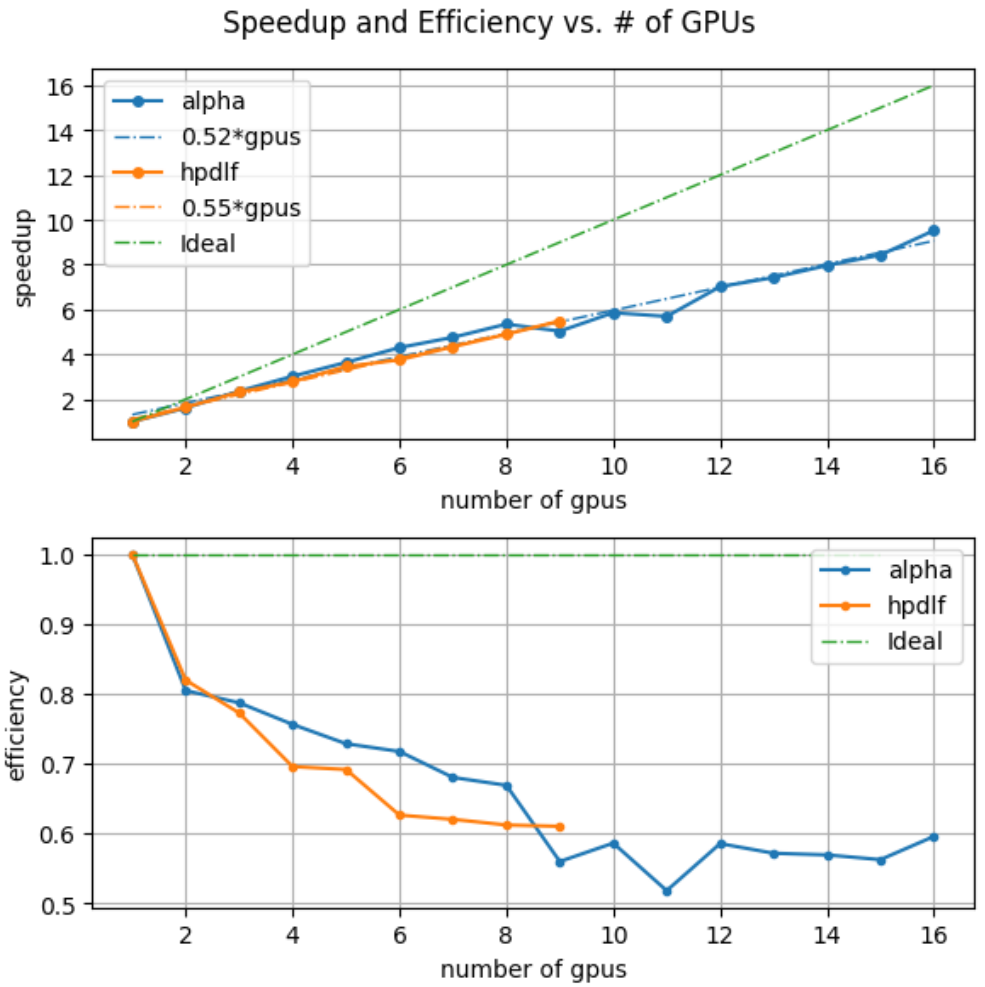
[8] P. Winkler, N. Koch, A. Hornig, and J. Gerritzen, “OmniOpt – A Tool for Hyperparameter Optimization on HPC,” in High Performance Computing, 2021, pp. 285–296.

[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

Speedup und Efficiency, MPI





Gradient Synchronization Time NCCL vs MPI