Towards Scalable Machine Learning

Janis Keuper

itwm.fraunhofer.de/ml

Competence Center High Performance Computing
Fraunhofer ITWM, Kaiserslautern, Germany

Fraunhofer Center Machnine Larning

Outline

I Introduction / Definitions

II Is Machine Learning a HPC Problem?

III Case Study: Scaling the Training of Deep Neural Networks

IV Towards Scalable ML Solutions [current Projects]

V A look at the (near) future of ML Problems and HPC

Introduction

Machine Learning @CC-HPC

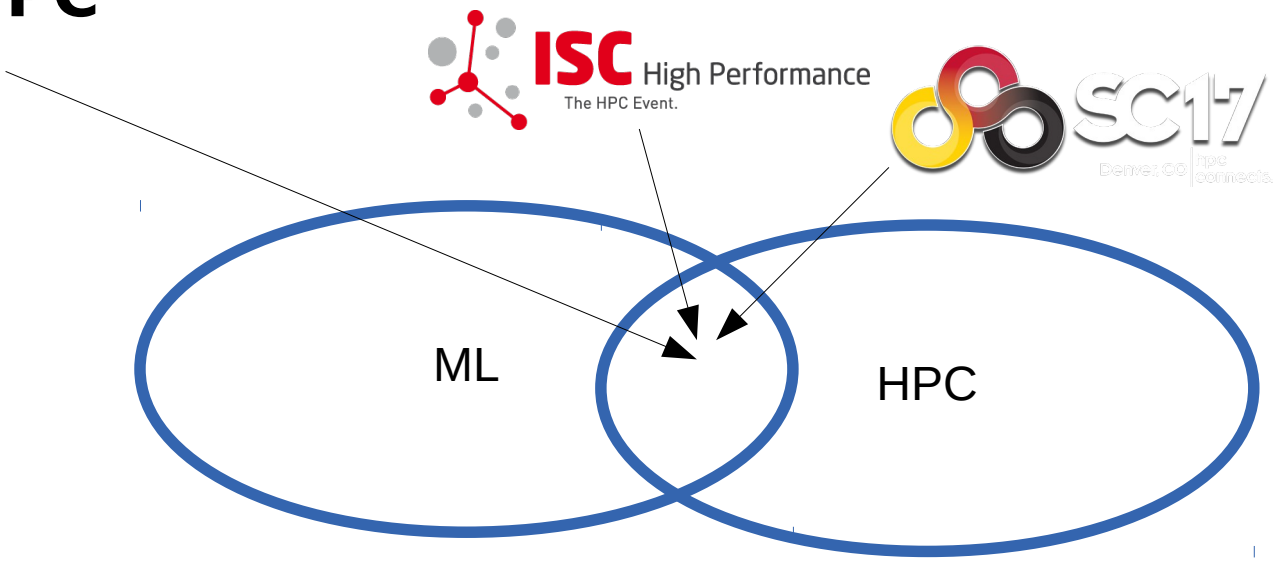
Scalable distributed ML Algorithms
- Distributed Optimization Methods
- Communication Protocols

Distributed DL Frameworks

“Automatic” ML
- DL Meta-Parameter Learning
- DL Topology Learning

HPC-Systems for Scalable ML
- Distributed I/O
- Novel ML Hardware
- Low Cost ML Systems

DL Methods:
- Semi- and Unsupervised DL
- Generative Models
- ND CNNs



Industry Applications
- DL Software optimization for Hardware / Clusters
- DL for Seismic Analysis
- DL Chemical Reaction Prediction
- DL for autonomous driving

Setting the Stage | Definitions

Scalable ML

- Large model size
  (implies large data as well)
- Extreme compute effort
- Goals:
  - Larger models
  - (linear) strong an weak scaling through (distributed) parallelization

→ HPC

vs

Large Scale ML

- Very large data sets
  (online stream)
- “normal” model size and compute effort
  (traditional ML methods)
- Goals:
  - Make training feasible
  - Often online training

→ Big Data

Scaling DNNs

Simple strategy in DL
if it does not work: scale it!

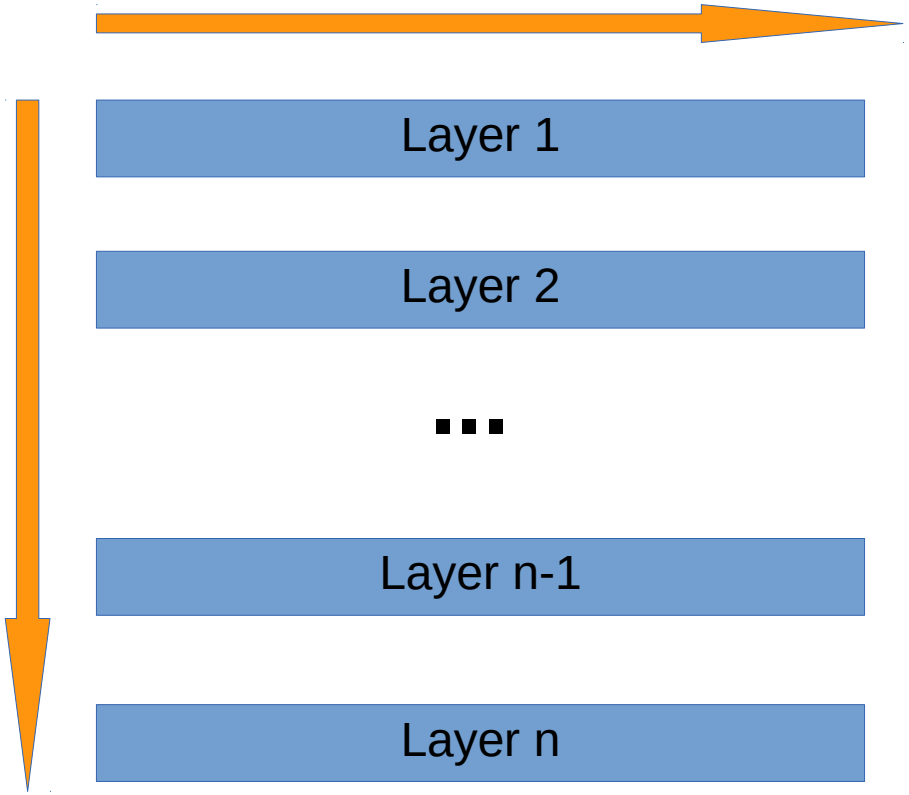
Scaling in two dimensions:

1. Add more layers = more matrix mult
more convolutions

2. Add more units = larger matrix mult
more convolutions

Don't forget: in both cases

MORE DATA! → more iterations



Scaling DNNs

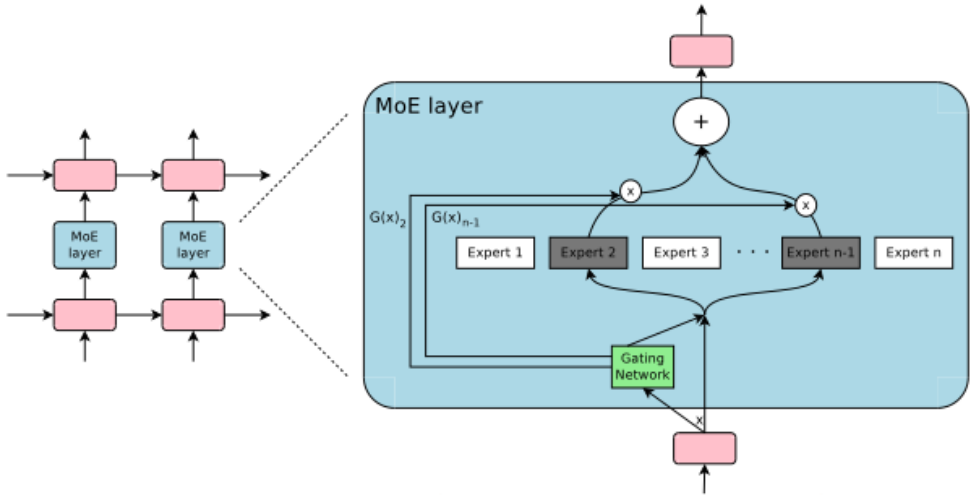
OUTRAGEOUSLY LARGE NEURAL NETWORKS: THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER

Noam Shazeer[1], Azalia Mirhoseini*†[1], Krzysztof Maziarz*[2], Andy Davis[1], Quoc Le[1], Geoffrey Hinton[1] and Jeff Dean[1]

[1]Google Brain, {noam,azalia,andydavis,qvl,geoffhinton,jeff}@google.com
[2]Jagiellonian University, Cracow, krzysztof.maziarz@student.uj.edu.pl

ABSTRACT

The capacity of a neural network to absorb information is limited by its number of parameters. Conditional computation, where parts of the network are active on a per-example basis, has been proposed in theory as a way of dramatically increasing model capacity without a proportional increase in computation. In practice, however, there are significant algorithmic and performance challenges. In this work, we address these challenges and finally realize the promise of conditional computation, achieving greater than 1000x improvements in model capacity with only minor losses in computational efficiency on modern GPU clusters. We introduce a Sparsely-Gated Mixture-of-Experts layer (MoE), consisting of up to thousands of feed-forward sub-networks. A trainable gating network determines a sparse combination of these experts to use for each example. We apply the MoE to the tasks of language modeling and machine translation, where model capacity is critical for absorbing the vast quantities of knowledge available in the training corpora. We present model architectures in which a MoE with up to 137 billion parameters is applied convolutionally between stacked LSTM layers. On large language modeling and machine translation benchmarks, these models achieve significantly better results than state-of-the-art at lower computational cost.



Network of Networks

137 billion free parameters !

Fraunhofer

Is Scalable ML a HPC Problem?

- In terms of compute needed (YES)
- Typical HPC Problem setting: is communication bound
  = non trivial parallelization (YES)
- I/O bound (New to HPC)

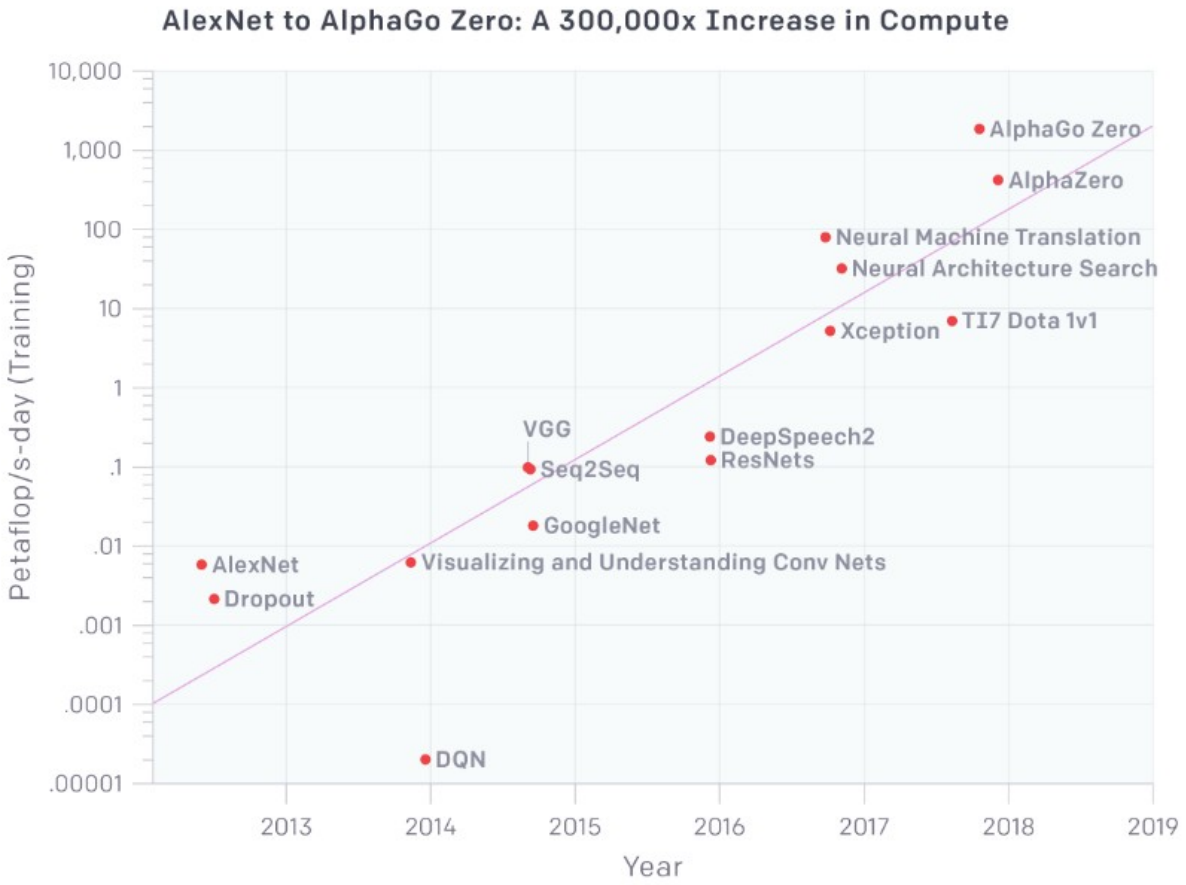
Success in Deep Learning is driven by compute power:

→ # FLOP needed to compute leading model is

~ doubling every 3.5 months !

→ increase since 2012: factor

~300.000 !



https://blog.openai.com/ai-and-compute/

Fraunhofer

Impact on HPC (Systems)

- New HPC Systems
  - Like ONCL “Summit”
  - Power 9
  - ~30k NVIDIA Volta GPUs
  - New storage hierarchies
- New Users (=new demands)
- Still limited resources

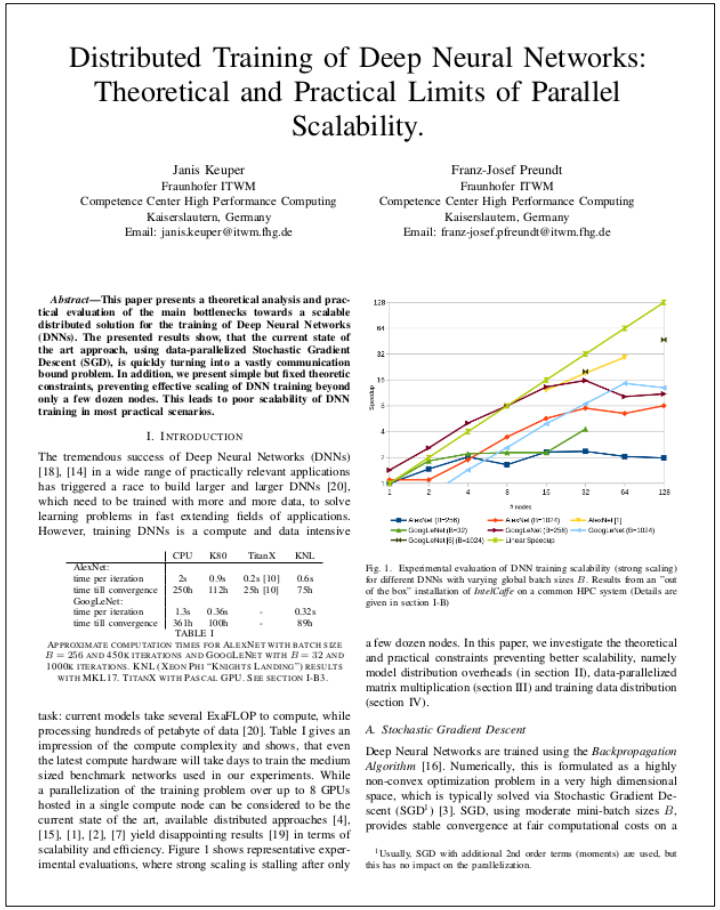
https://www.nextplatform.com/2018/03/28/a-first-look-at-summit-supercomputer-application-performance/

III Case Study: Training DNNs

I Overview: distributed parallel training of DNNs

II Limits of Scalability

- Limitation I: Communication Bounds
- Limitation II: Skinny Matrix Multiplication
- Limitation III: Data I/O



Distributed Training of Deep Neural Networks: Theoretical and Practical Limits of Parallel Scalability.

Janis Keuper
Fraunhofer ITWM
Competence Center High Performance Computing
Kaiserslautern, Germany
Email: janis.keuper@itwm.fhg.de

Franz-Josef Preundt
Fraunhofer ITWM
Competence Center High Performance Computing
Kaiserslautern, Germany
Email: franz-josef.pfreundt@itwm.fhg.de

***Abstract*—This paper presents a theoretical analysis and practical evaluation of the main bottlenecks towards a scalable distributed solution for the training of Deep Neural Networks (DNNs). The presented results show, that the current state of the art approach, using data-parallelized Stochastic Gradient Descent (SGD), is quickly turning into a vastly communication bound problem. In addition, we present simple but fixed theoretic constraints, preventing effective scaling of DNN training beyond only a few dozen nodes. This leads to poor scalability of DNN training in most practical scenarios.**

I. Introduction

The tremendous success of Deep Neural Networks (DNNs) [18], [14] in a wide range of practically relevant applications has triggered a race to build larger and larger DNNs [20], which need to be trained with more and more data, to solve learning problems in fast extending fields of applications. However, training DNNs is a compute and data intensive task: current models take several ExaFLOP to compute, while processing hundreds of petabyte of data [20]. Table I gives an impression of the compute complexity and shows, that even the latest compute hardware will take days to train the medium sized benchmark networks used in our experiments. While a parallelization of the training problem over up to 8 GPUs hosted in a single compute node can be considered to be the current state of art, available distributed approaches [4], [15], [1], [2], [7] yield disappointing results [19] in terms of scalability and efficiency. Figure 1 shows representative experimental evaluations, where strong scaling is stalling after only a few dozen nodes. In this paper, we investigate the theoretical and practical constraints preventing better scalability, namely model distribution overheads (in section II), data-parallelized matrix multiplication (section III) and training data distribution (section IV).

	CPU	K80	TitanX	KNL
AlexNet:				
time per iteration	2s	0.9s	0.2s [10]	0.6s
time till convergence	250h	112h	25h [10]	75h
GoogLeNet:				
time per iteration	1.3s	0.36s	-	0.32s
time till convergence	36 1h	100h	-	89h

TABLE I
Approximate computation times for AlexNet with batch size $B = 256$ and 450K iterations and GoogLeNet with $B = 32$ and 1000K iterations. KNL (Xeon Phi "Knights Landing") results with MKL17. TitanX with Pascal GPU. See section I-B3.

Fig. 1. Experimental evaluation of DNN training scalability (strong scaling) for different DNNs with varying global batch sizes B. Results from an "out of the box" installation of *IntelCaffe* on a common HPC system (Details are given in section I-B)

A. Stochastic Gradient Descent

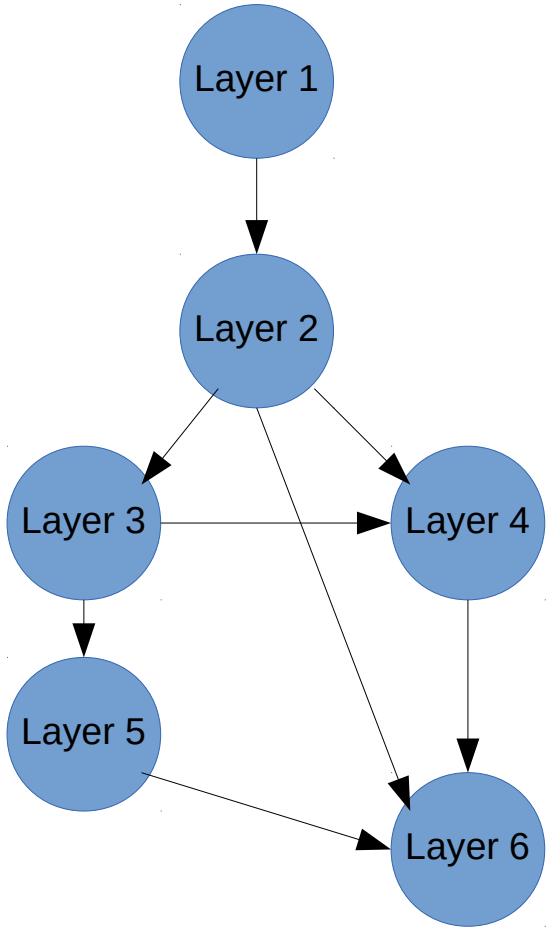
Deep Neural Networks are trained using the *Backpropagation Algorithm* [16]. Numerically, this is formulated as a highly non-convex optimization problem in a very high dimensional space, which is typically solved via Stochastic Gradient Descent (SGD[1]) [3]. SGD, using moderate mini-batch sizes B, provides stable convergence at fair computational costs on a

[1]Usually, SGD with additional 2nd order terms (moments) are used, but this has no impact on the parallelization.

Based on our paper from SC 16

Deep Neural Networks

In a Nutshell



At an abstract level, DNNs are:

- directed (acyclic) graphs
- ***Nodes*** are compute entities (=Layers)
- ***Edges*** define the data flow through the graph

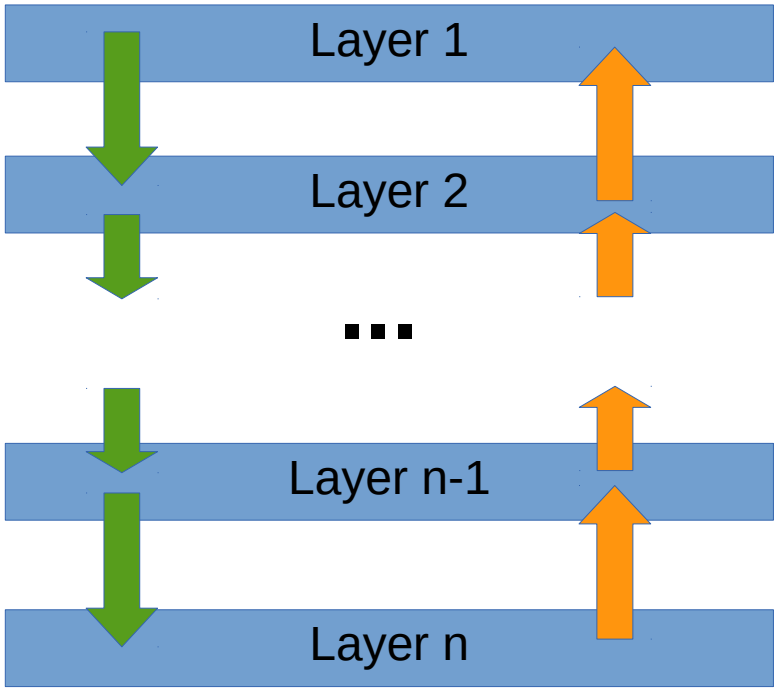
Inference / Training

Forward feed of data through the network

Deep Neural Networks

In a Nutshell

Common intuition



At an abstract level, DNNs are:

- directed (acyclic) graphs
- ***Nodes*** are compute entities (=Layers)
- ***Edges*** define the data flow through the graph

Inference / Training

Forward feed of data through the network

Training Deep Neural Networks

The Underlying Optimization Problem

Computed via **Back Propagation** Algorithm:

1. feed forward and compute activation
2. error by layer
3. compute derivative by layer

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2}\|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

Minimize Loss-Function via gradient descent (high dimensional and NON CONVEX!)

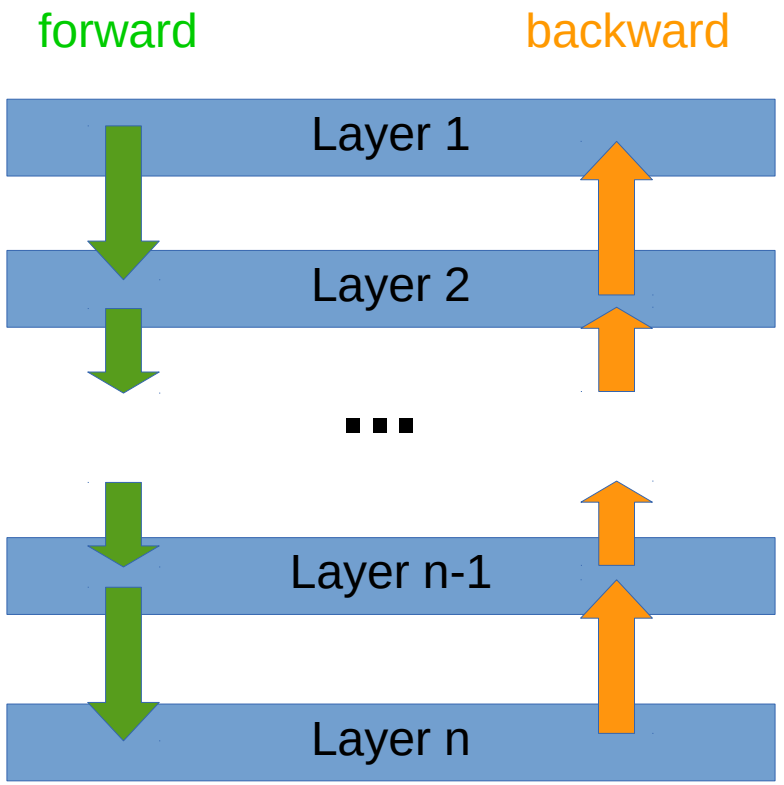
$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

Optimization Problem

By Stochastic Gradient Descent (SGD)

1. Initialize weights W at random
2. Take small random subset X (=**batch**) of the train data
3. Run X through network (forward feed)
4. Compute Loss
5. Compute Gradient
6. Propagate backwards through the network
7. Update W

Repeat 2-8 until convergence



Parallelization

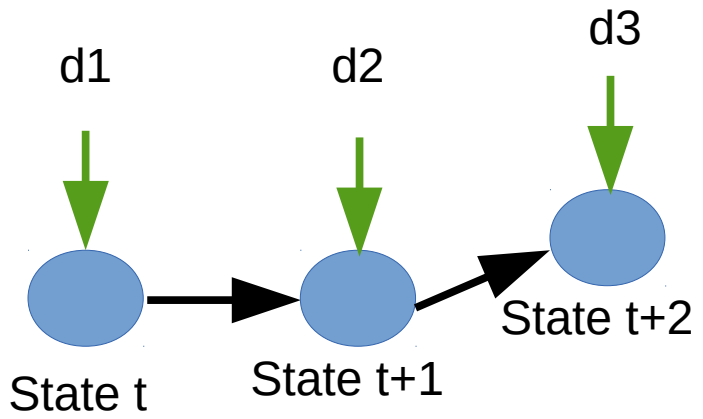
Common approaches to parallelize SGD for DL

Parallelization of SGD is very hard: it is an inherently sequential algorithm

1. Start at some state **t** (point in a billion dimensional space)
2. Introduce **t** to data batch **d1**
3. Compute an update (based on the objective function)
4. Apply the update →**t+1**

How to gain Speedup ?

Make faster updates
Make larger updates

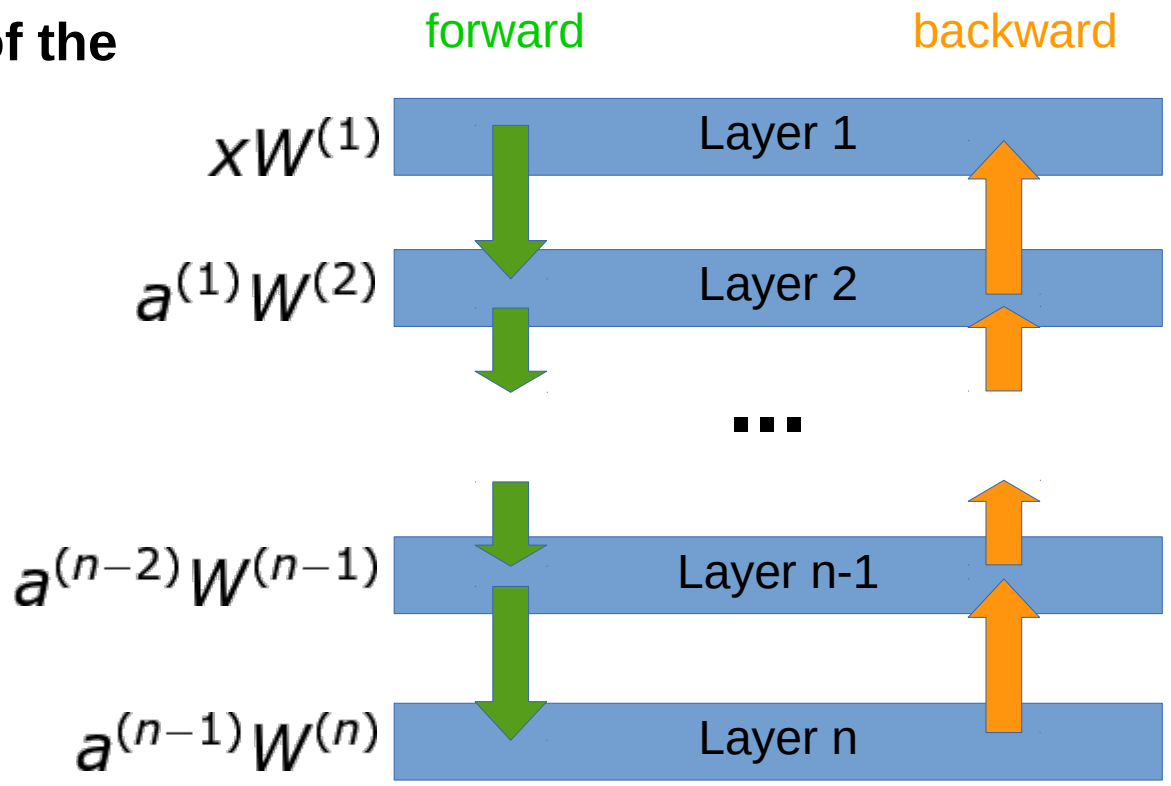


Parallelization

Common approaches to parallelize SGD for DL

Internal parallelization = parallel execution of the layer operation

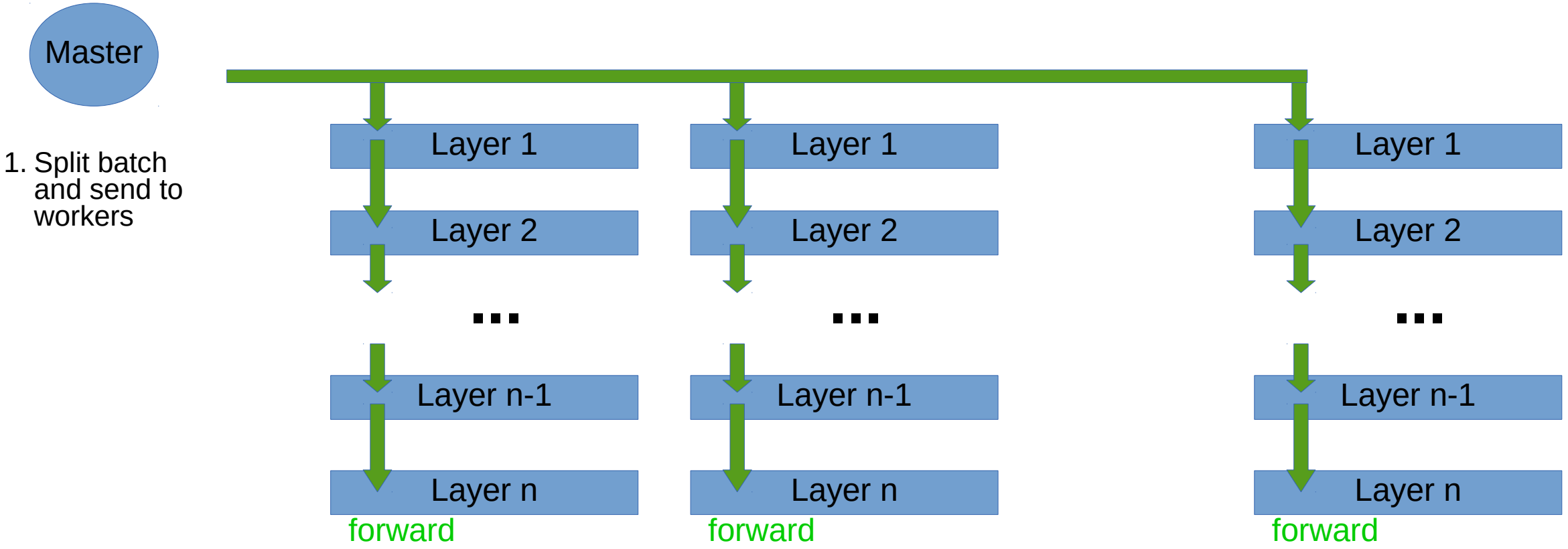
- Mostly **dense matrix multiplication**:
  - standard blas sgemm
  - MKL, Open-Blas
  - **CuBlas** on GPUs
- Task parallelization for special Layers
  - Cuda-CNN for fast convolutions



Parallelization

Common approaches to parallelize SGD for DL

External: data parallelization over the data batch

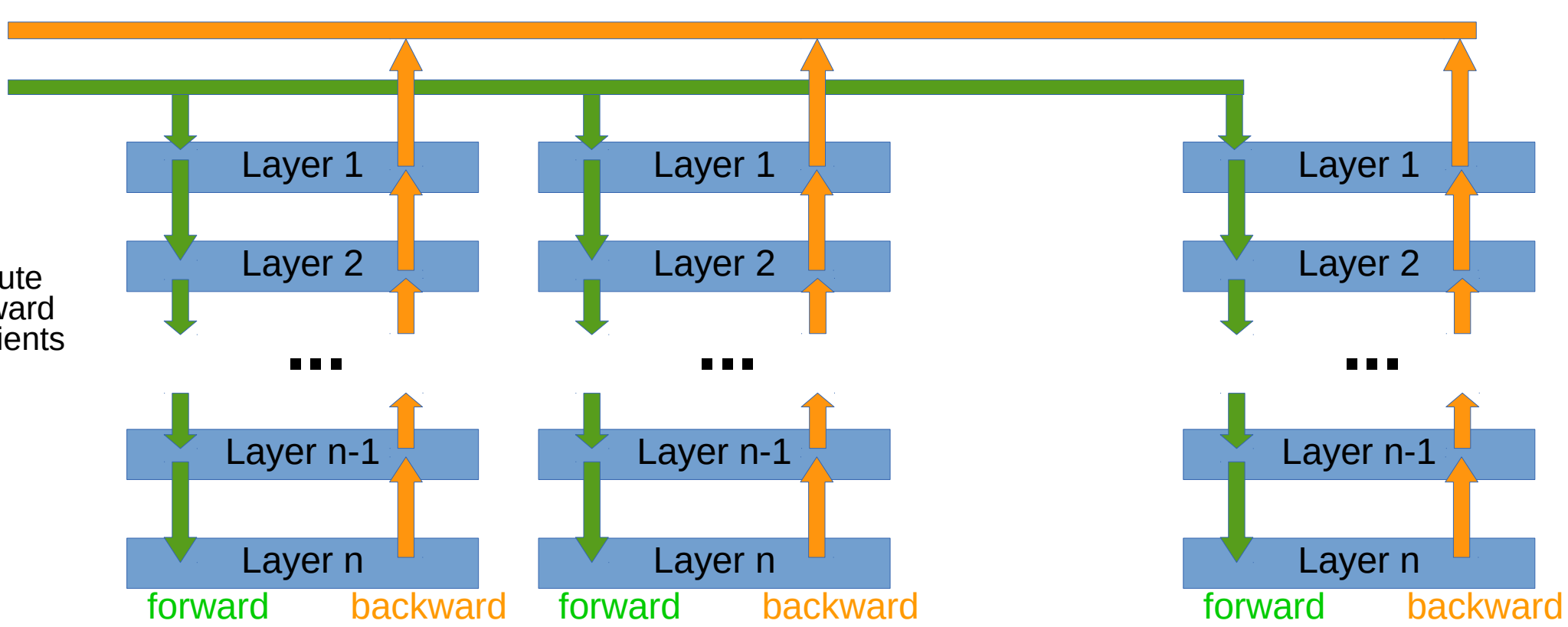


Parallelization

Common approaches to parallelize SGD for DL

External: data parallelization over the data batch

1. Split batch and send to workers
2. Workers compute forward+backward and send gradients to master



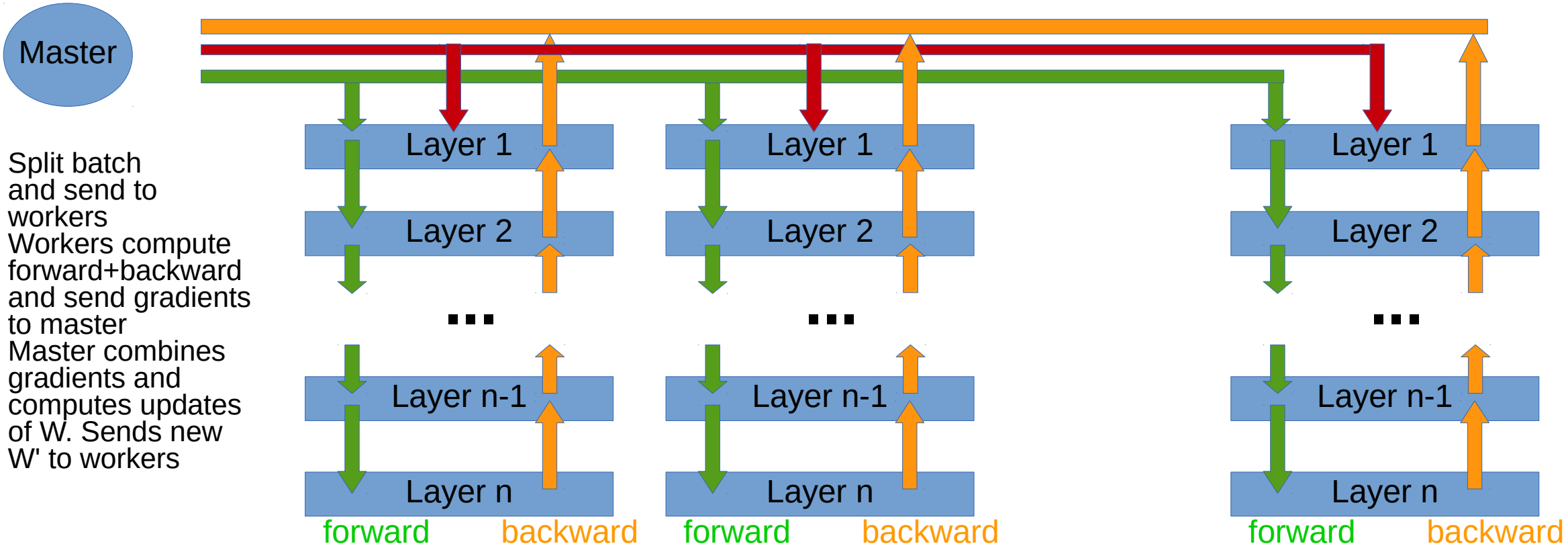
Fraunhofer

Parallelization

Common approaches to parallelize SGD for DL

External: data parallelization over the data batch

1. Split batch and send to workers
2. Workers compute forward+backward and send gradients to master
3. Master combines gradients and computes updates of W. Sends new W' to workers

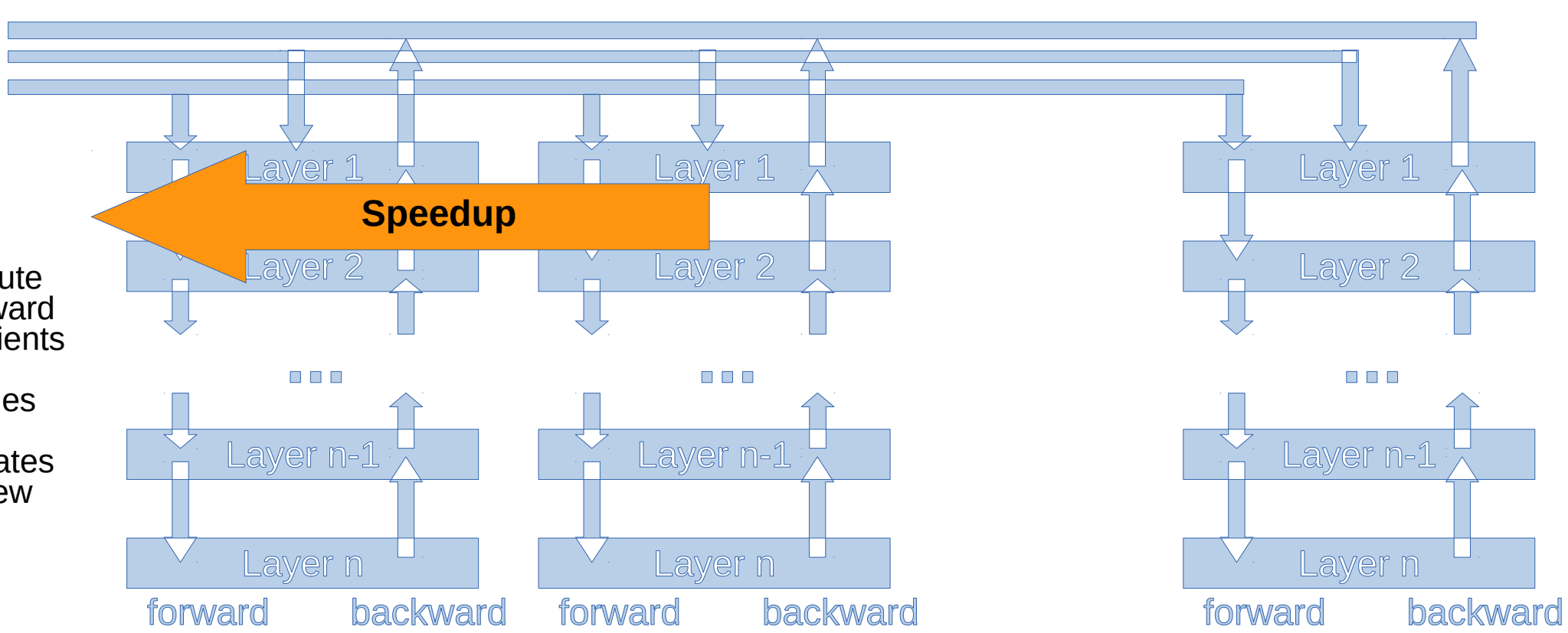


Parallelization

Common approaches to parallelize SGD for DL

External: data parallelization over the data batch

1. **Split batch and send to workers**
2. Workers compute forward+backward and send gradients to master
3. Master combines gradients and computes updates of W. Sends new W' to workers

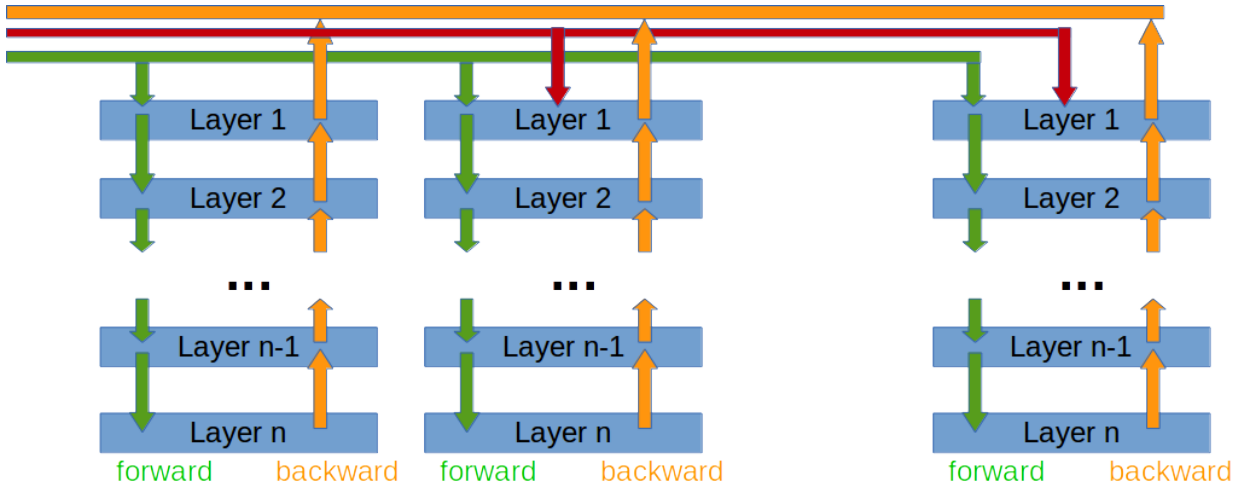


Limitation I

Distributed SGD is heavily Communication Bound

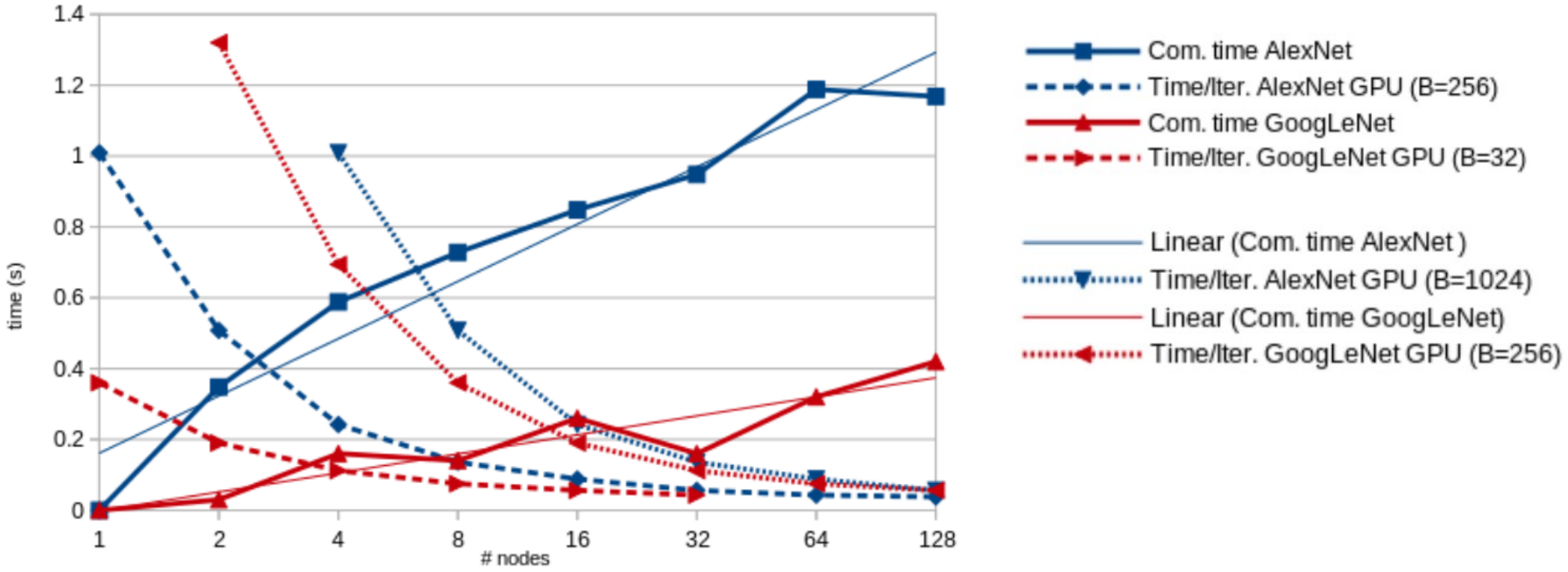
Gradients have the same size as the model

- Model size can be hundreds of MB
- Iteration time (GPU) <1s



Communication Bound

Experimental Evaluation



Solving the Communication Bottleneck

Solutions in Hardware: example NVLink

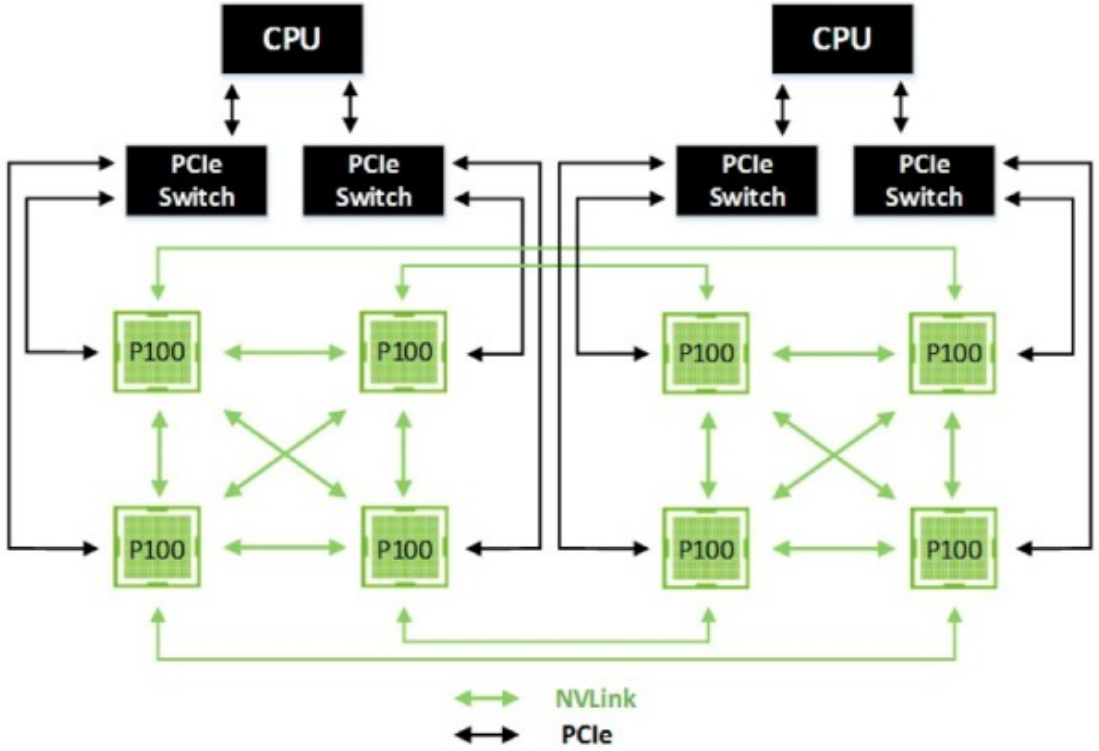


Diagram DGX-1 by NVIDIA

NVIDIA DGX-1 / HGX-1

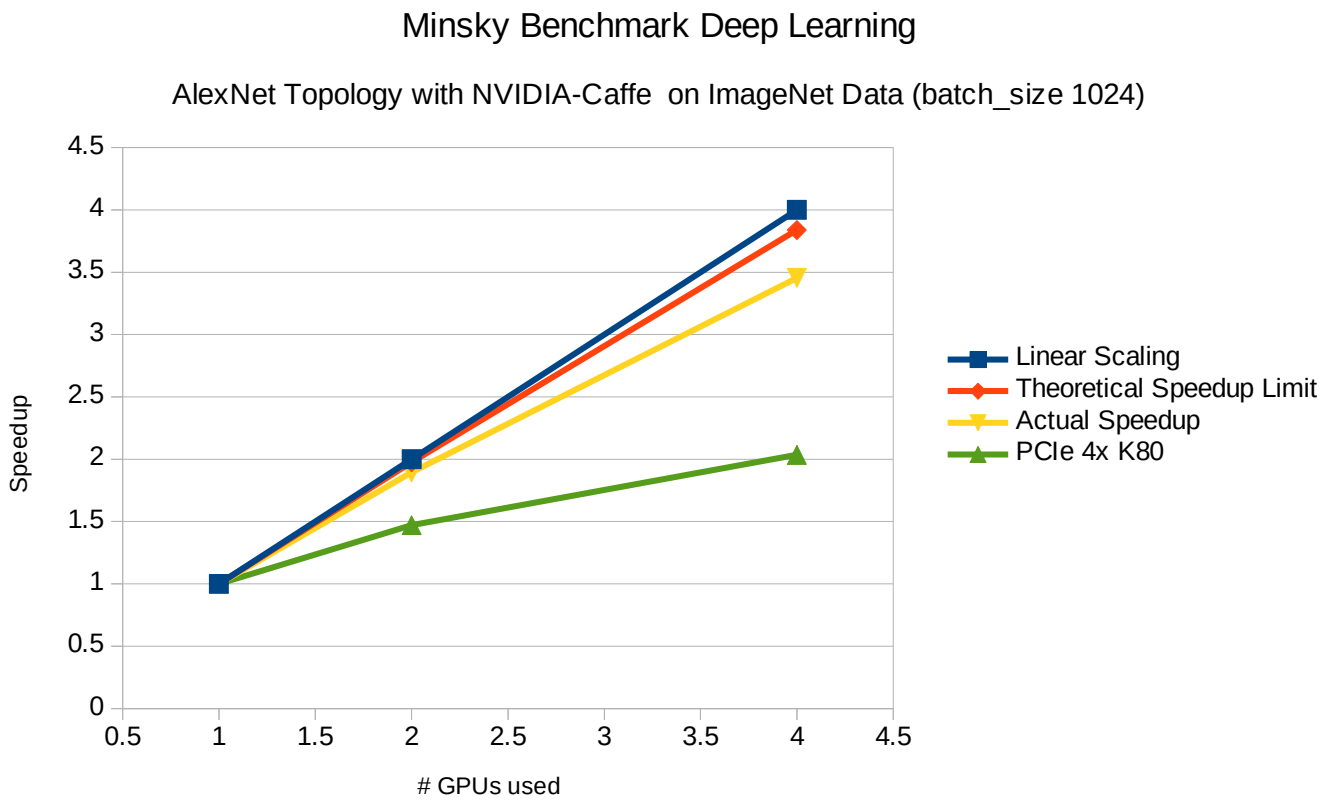
- 8x P100
- DGX-1: NVLink between all GPUs

NVLink spec: ~40GB/s (single direction)
NVLink II (Volta): ~75GB/s (single direction)

PCIe v3 16x: ~15GB/s

Solving the Communication Bottleneck

Solutions in Hardware: NVLink Benchmark



IBM Minsky

- 4x P100
- 2x10 core Power8 (160 hw threads)
- NVLink between all components

NVLink spec: ~40GB/s (single direction)

Limitation I

Distributed SGD is heavily Communication Bound

How to solve this in distributed environments?

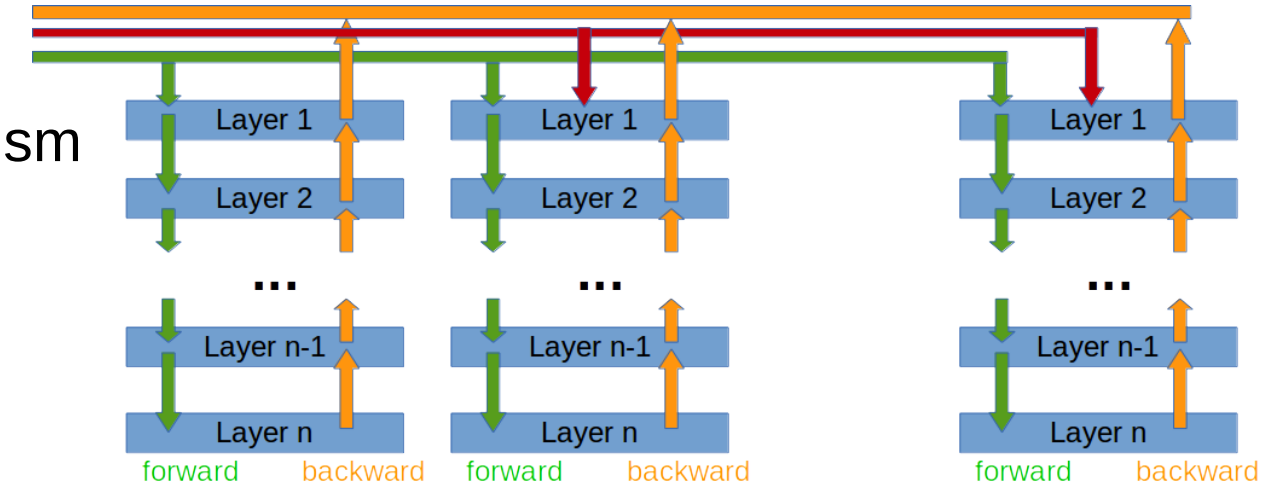
Limitation II

Mathematical Problems – aka the “Large Batch Size Problem”

Recall:

speedup comes from pure data-parallelism

→ splitting the batch over the workers



“Small Batch Size Problem”

Data Parallelization over the Batch Size

Problem: Batch size decreasing with distributed scaling

Hard Theoretic Limit: b > 0

→ **GoogLeNet: No Scaling beyond 32 Nodes**
→ **AlexNet: Limit at 256 Nodes**

External Parallelization hurts the internal (BLAS / cuBlas) parallelization even earlier.

In a nutshell: for skinny matrices there is simply not enough work for efficient internal parallelization over many threads.

“Small Batch Size Problem”

Data Parallelization over the Batch Size

Computing Fully Connected Layers:

Single dense Matrix Multiplication

Layer	# operations	matrix sizes
Fully Connected	1	$b \times I * I \times O$
Convolutional	b	$C \times I * I \times Z$
Softmax	b	$I \times 1 * 1 \times 1$

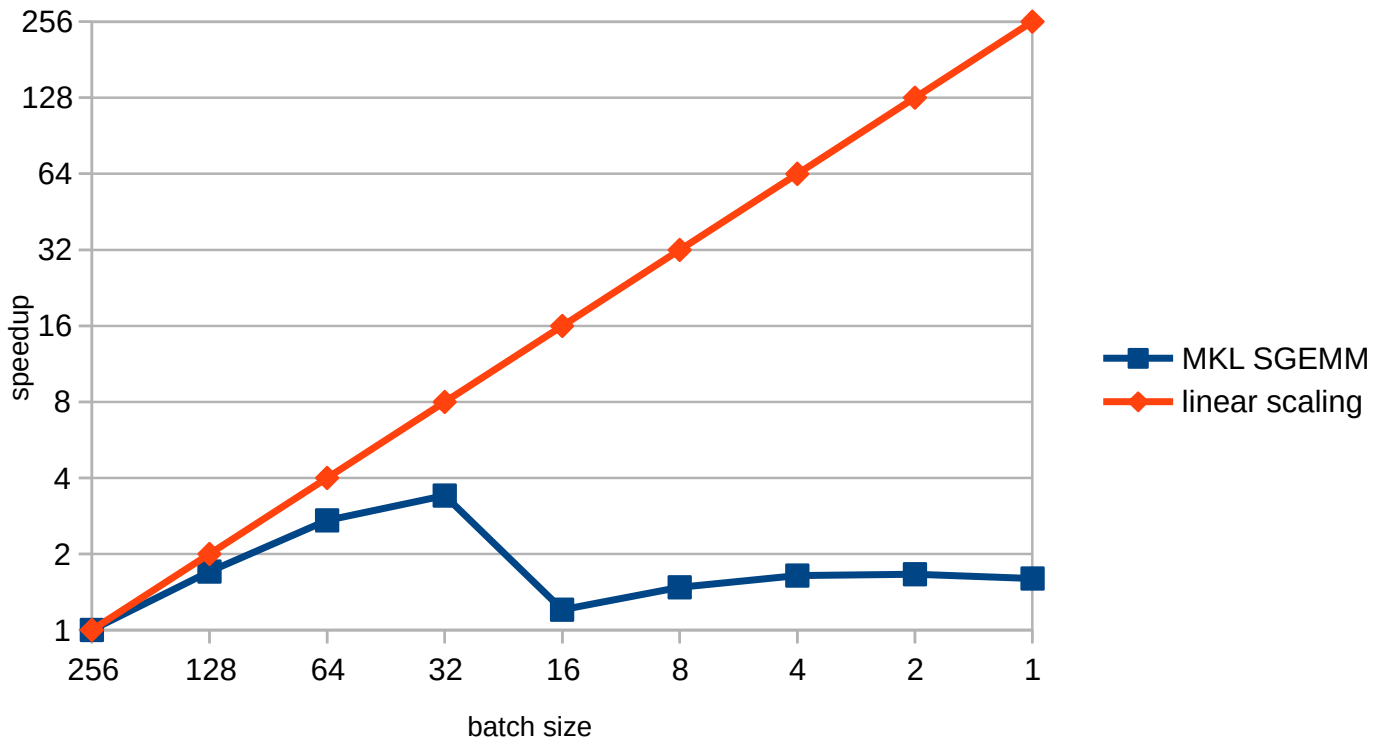
Definitions:

I: Input size from top layer
O: Output size of this layer
b: local Batch size (train or validation)
C: Number of filters
c: Number of input channels (RBG image: $c = 3$)
P: Patch size (i.e. pixel)
k: kernel size
Z: Effective size after kernel application.

For convolution $Z := \left(\sqrt{P} - \lfloor (k/2) \right)^2$

TABLE III

SIZE AND NUMBER OF OF THE MATRIX MULTIPLICATIONS (SGEMM) PER FORWARD PASS FOR SELECTED LAYERS.



Fraunhofer

Experimental Evaluation

Increasing the Batch Size

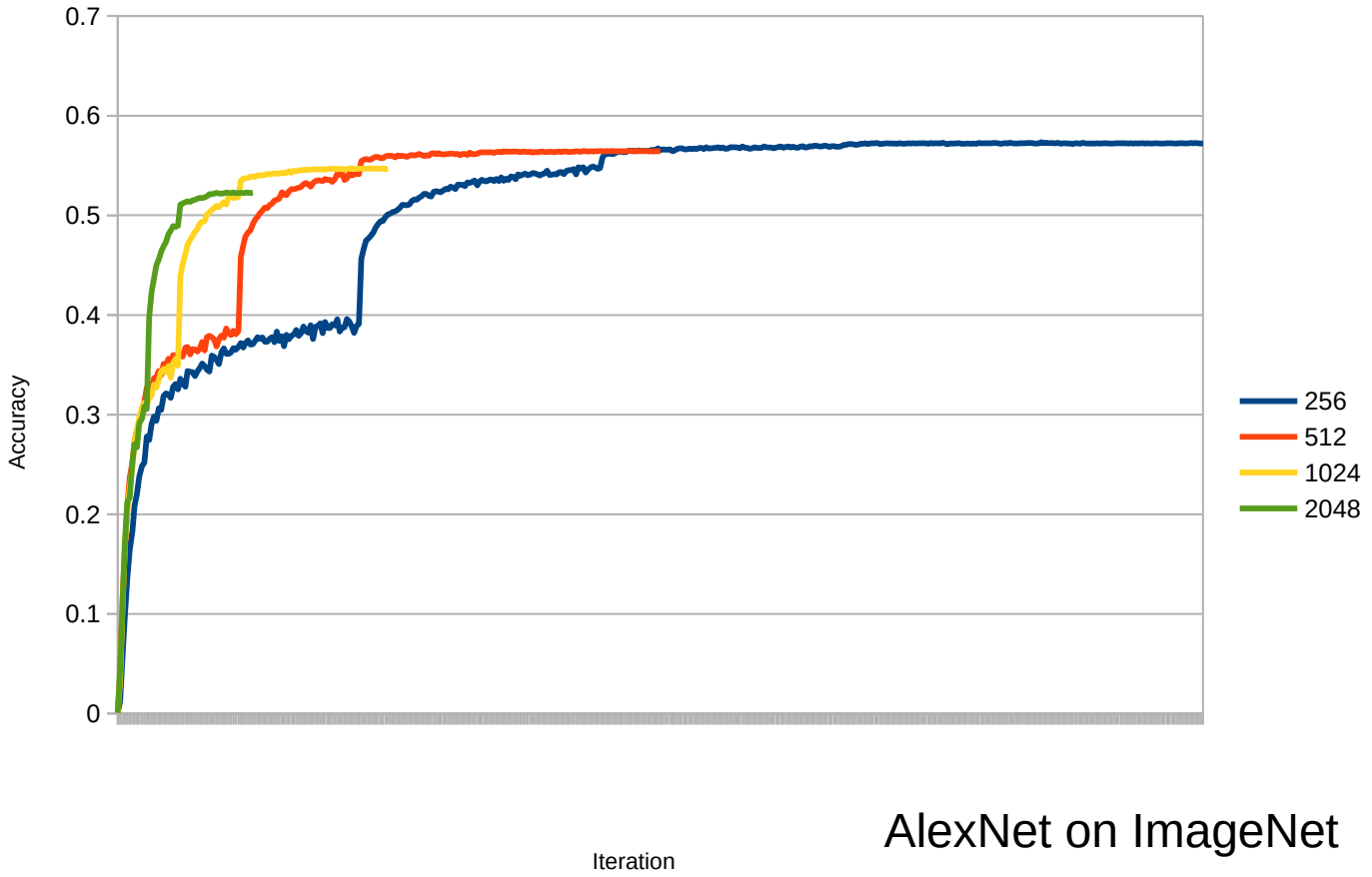
Solution proposed in literature:

Increase Batch size

But:

Linear speedup against original Problem only if we can reduce the number iterations accordingly

This leads to loss of accuracy



AlexNet on ImageNet

The “large batch” Problem

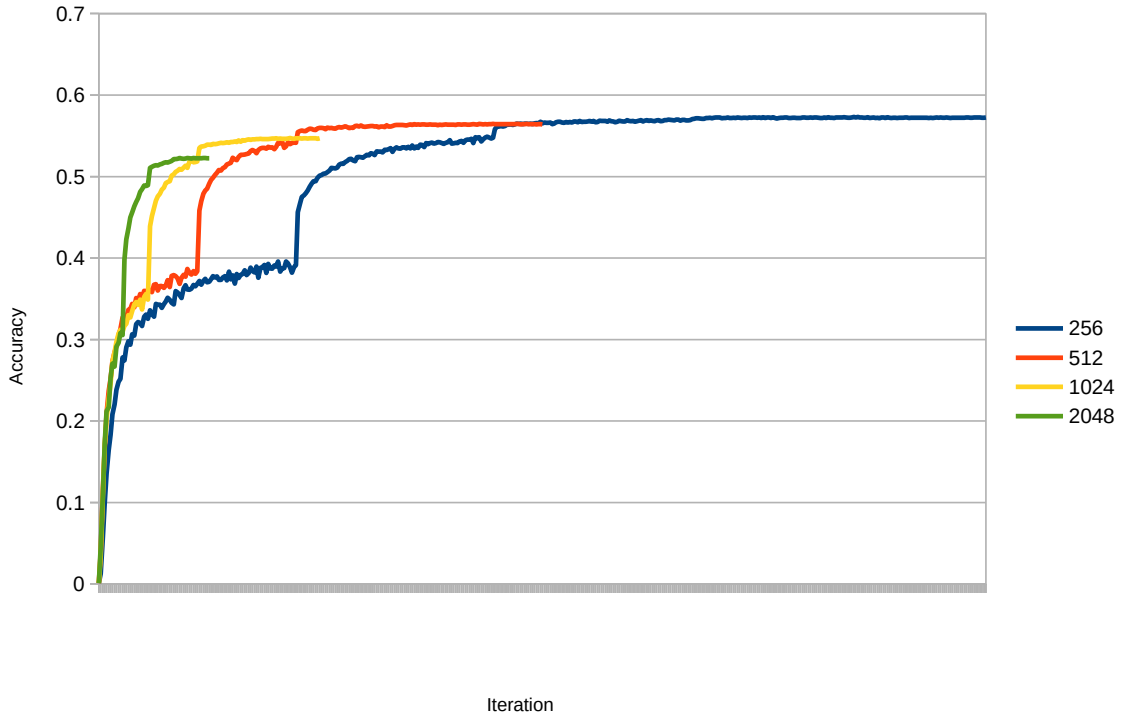
Central Questions

Why is this happening?

Is it dependent on the Topologie / other parameters?

How can it be solved?

→ large batch size SGD would solve most scalability problems!

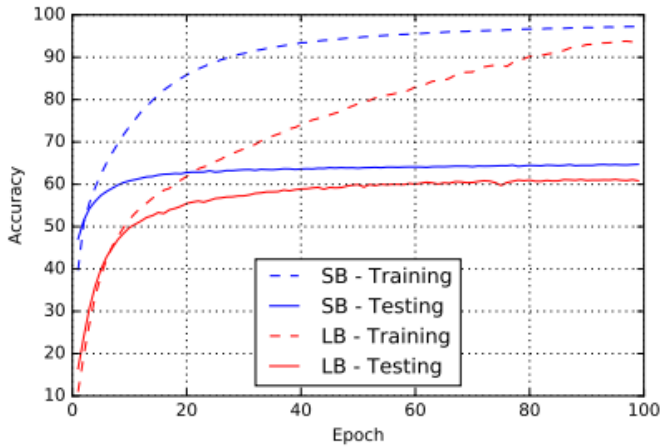


The “large batch” Problem

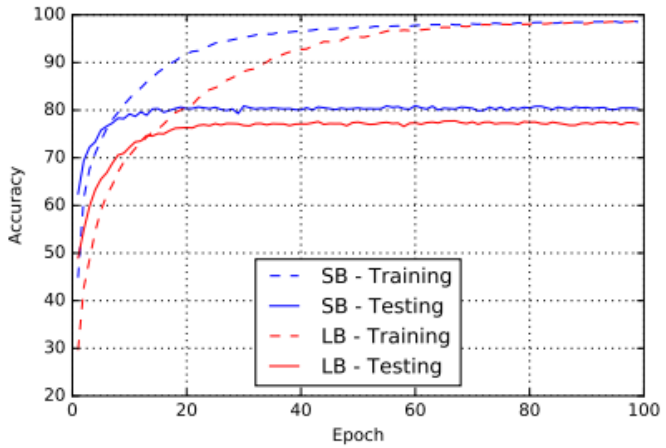
What is causing this effect? [theoretical and not so theoretical explanations]

→ **The “bad minimum”**

→ **gradient variance / coupling of learning rate and batch size**



(a) Network F_2



(b) Network C_1

Figure 2: Training and testing accuracy for SB and LB methods as a function of epochs.

The “bad minimum”

Theory: larger batch causes degrease in gradient variance, causing convergence to local minima...

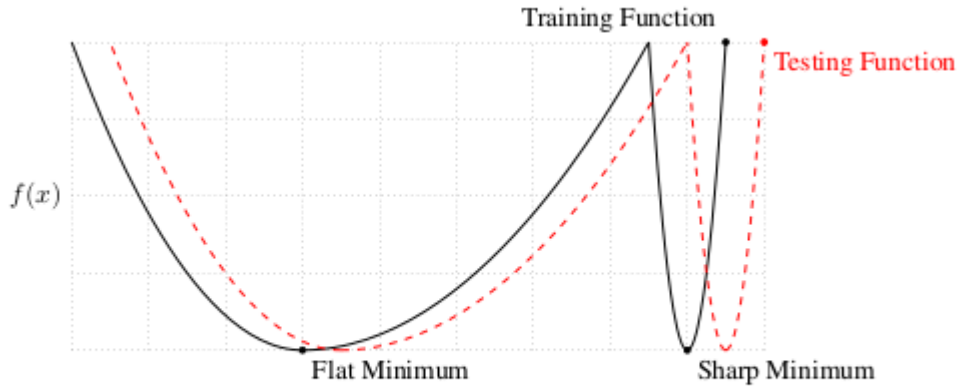


Figure 1: A Conceptual Sketch of Flat and Sharp Minimizers (Y-axis indicates value of the loss function and X-axis indicates the weights)

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima

Nitish Shirish Keskar*
Northwestern University
Evanston, IL 60208
keskar.nitish@u.northwestern.edu

Dheevatsa Mudigere
Intel Corporation
Bangalore, India
dheevatsa.mudigere@intel.com

Jorge Nocedal
Northwestern University
Evanston, IL 60208
j-nocedal@northwestern.edu

Mikhail Smelyanskiy
Intel Corporation
Santa Clara, CA 95054
mikhail.smelyanskiy@intel.com

Ping Tak Peter Tang
Intel Corporation
Santa Clara, CA 95054
peter.tang@intel.com

Abstract

The stochastic gradient descent method and its variants are algorithms of choice for many Deep Learning tasks. These methods operate in a small-batch regime wherein a fraction of the training data, usually 32–512 data points, is sampled to compute an approximation to the gradient. It has been observed in practice that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize. There have been some attempts to investigate the cause for this generalization drop in the large-batch regime, however the precise answer for this phenomenon is, hitherto unknown. In this paper, we present ample numerical evidence that supports the view that large-batch methods tend to converge to sharp minimizers of the training and testing functions – and that sharp minima lead to poorer generalization. In contrast, small-batch methods consistently converge to flat minimizers, and our experiments support a commonly held view that this is due to the inherent noise in the gradient estimation. We also discuss several empirical strategies that help large-batch methods eliminate the generalization gap and conclude with a set of future research ideas and open questions.

→ empirical evaluation shows high correlation of sharp minima and weak generalization

Limitation II

Mathematical Problems – aka the “Large Batch Size Problem”

Problem not fully understood

No general solutions (yet) !

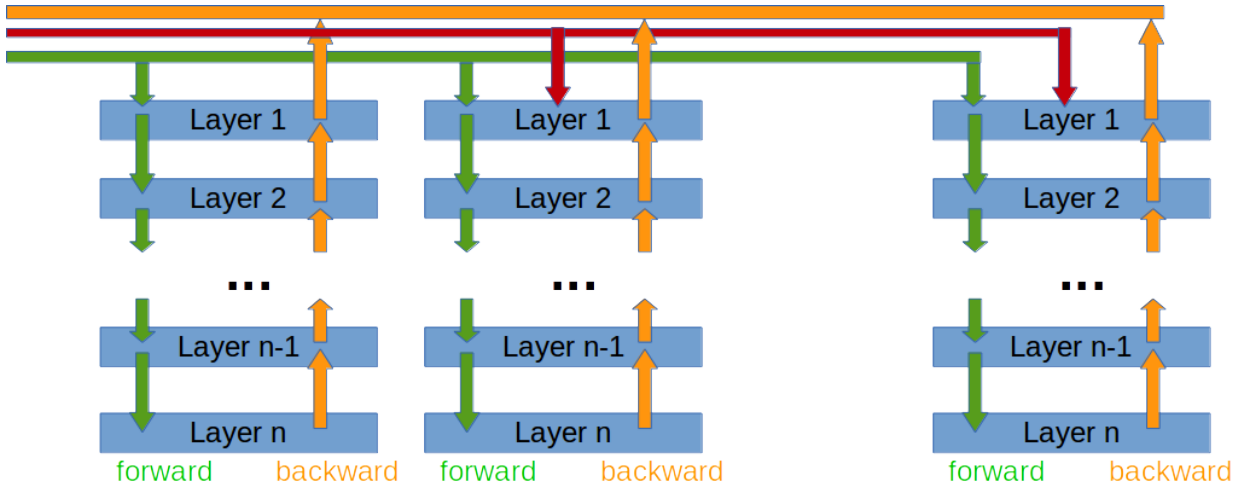
Do we need novel optimization methods ?!

Limitation III

Data I/O

Hugh amounts of training data need to Be streamed to the GPUs

- Usually 50x – 100x the training data set
- Random sampling (!)
- Latency + Bandwidth competing with optimization communication



Distributed I/O

Distributed File Systems are another Bottleneck !

- Network bandwidth is already exceeded by the SGD communication
- Worst possible file access pattern:
  - **Access many small files at random**

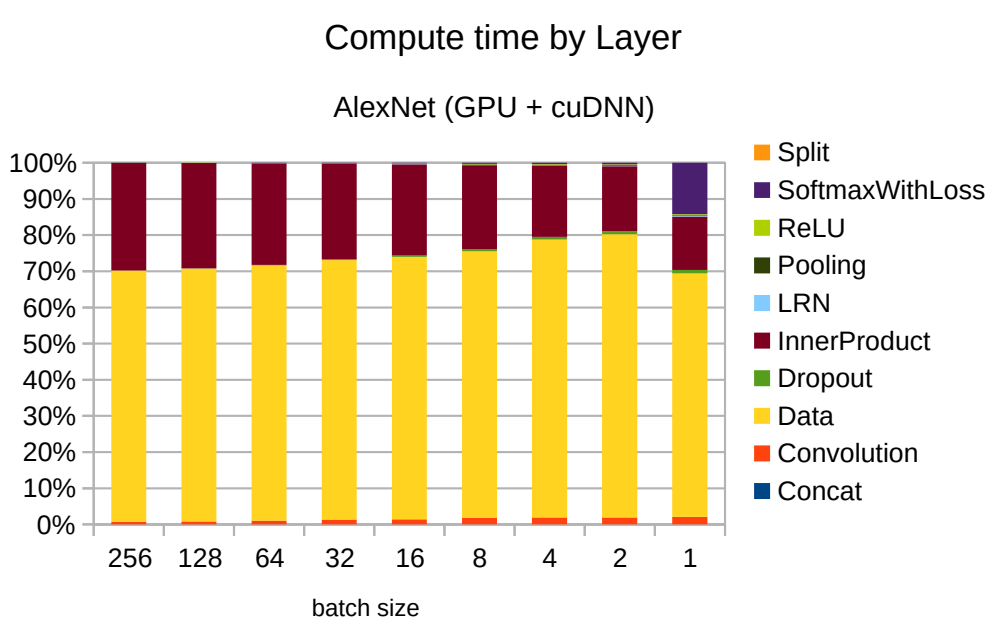
This problem already has effects on local multi-GPU computations

E.g. on DG-X1 or Minsky, single SSD (~0.5 GB/s) to slow to feed >= 4 GPUs

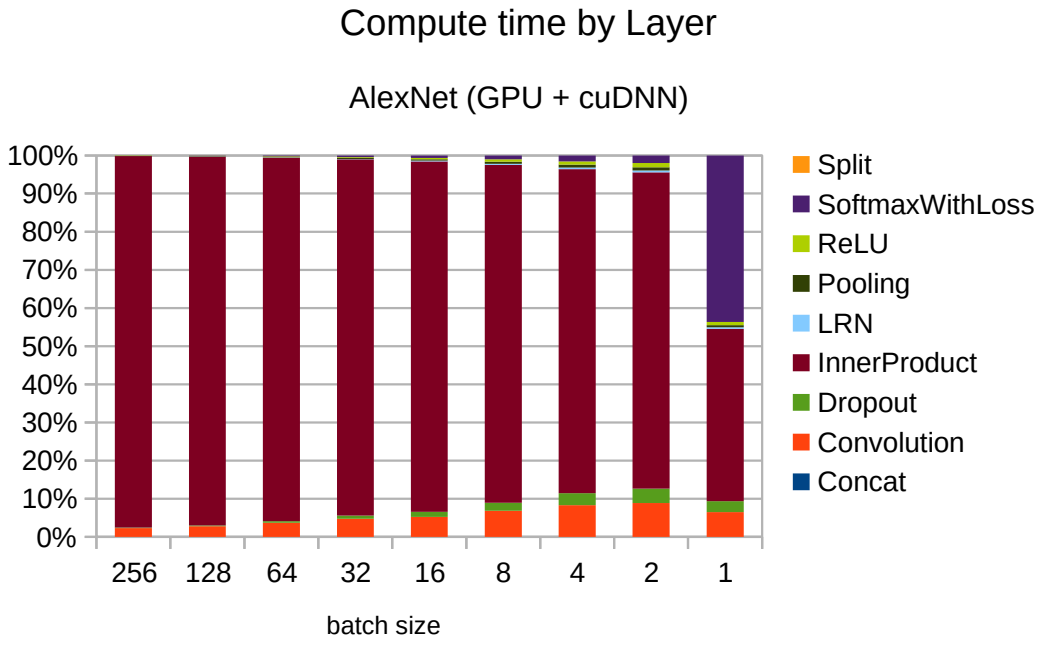
-> solution: Raid 0 with 4 SSDs

Distributed I/O

Distributed File Systems are another Bottleneck !



Results shown for SINGLE node access to a Lustre working directory (HPC Cluster, FDR-Infiniband)



Results shown for SINGLE node Data on local SSD.

IV Towards Scalable Solutions

Distributed Parallel Deep Leaning with HPC tools + Mathematics

CaffeGPI:
Distributed Synchronous SGD Parallelization With asynchronous communication overlay

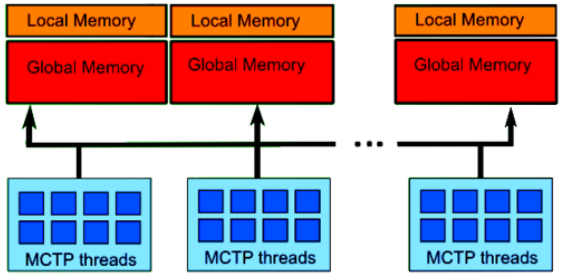
EPiGRAM2

Better scalability using asynchronous PGAS programming of optimization algorithms with GPI-2.

Direct RDMA access to main and GPU memory instead of message passing

Optimized data-layers for distributed File systems

Global Address Space Programming Interface
GASPI

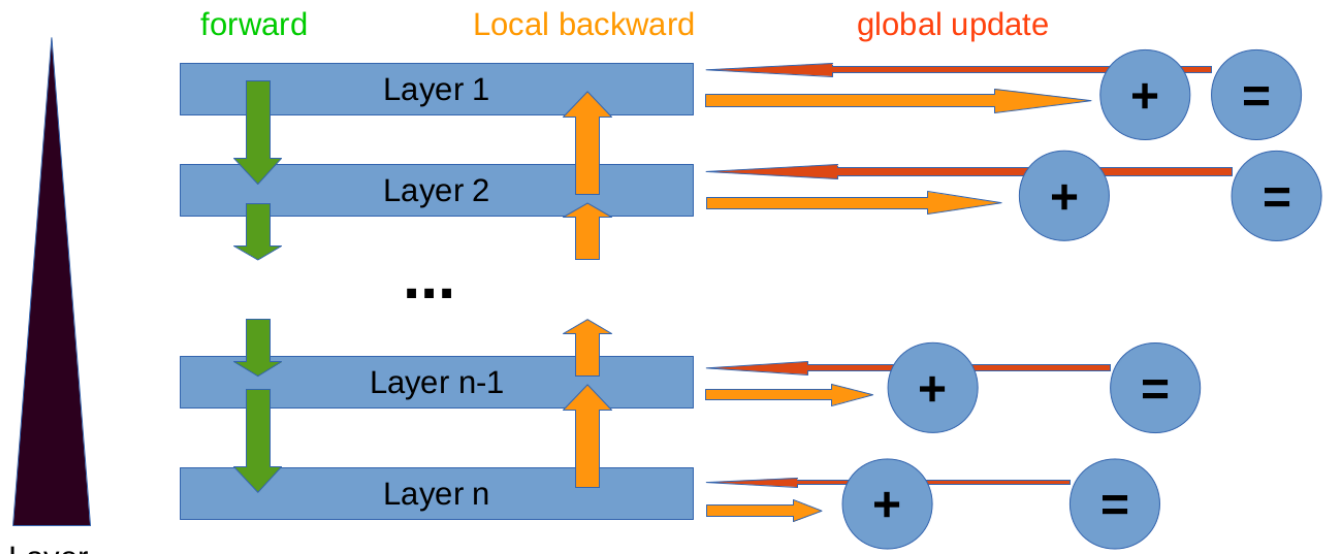


BeeGFS®

Fraunhofer

CC-HPC Current Projects

CaffeGPI: Approaching DNN Communication



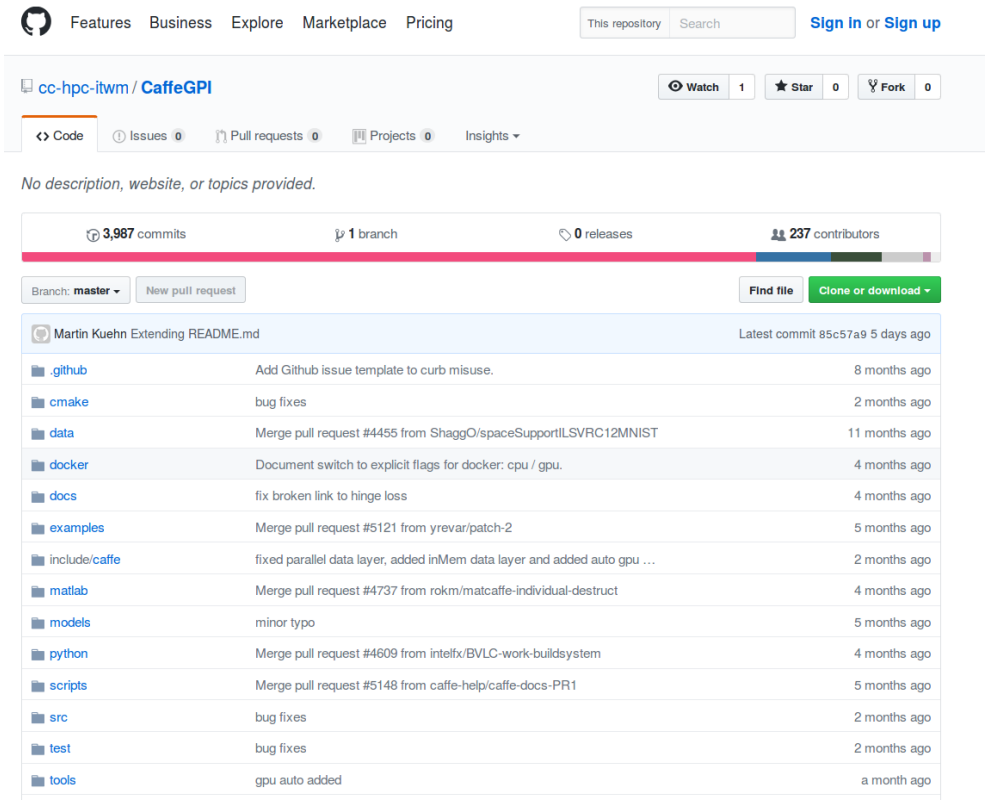
- Communication Reduction Tree
- Communication Quantization
- Communication Overlay
- Direct RDMA GPU → GPU
  - Based on GPI

Global Address Space Programming Interface
GASPI

- Optimized distributed data-layer

BeeGFS®

CaffeGPI:
Open Source



https://github.com/cc-hpc-itwm/CaffeGPI

Using GPI-2 for Distributed Memory Parallelization of the Caffe Toolbox to Speed up Deep Neural Network Training

Martin Kuehn, Janis Keuper and Franz-Josef Pfreundt
Competence Center High Performance Computing
Fraunhofer Institute for Industrial Mathematics
Fraunhofer-Platz 1, D-67663 Kaiserslautern, Germany

May 31, 2017

Abstract

Deep Neural Network (DNN) are currently of great interest in research and application. The training of these networks is a compute intensive and time consuming task. To reduce training times to a bearable amount at reasonable cost we extend the popular Caffe toolbox for DNN with an efficient distributed memory communication pattern. To achieve good scalability we emphasize the overlap of computation and communication and prefer fine granular synchronization patterns over global barriers. To implement these communication patterns we rely on the the "Global address space Programming Interface" version 2 (GPI-2) communication library. This interface provides a light-weight set of asynchronous one-sided communication primitives supplemented by non-blocking fine granular data synchronization mechanisms. Therefore, CaffeGPI is the name of our parallel version of Caffe. First benchmarks demonstrate better scaling behavior compared with other extensions, e.g., the Intel™Caffe. Even within a single symmetric multiprocessing machine with four graphics processing units, the CaffeGPI scales better than the standard Caffe toolbox. These first results demonstrate that the use of standard High Performance Computing (HPC) hardware is a valid cost saving approach to train large DDNs. I/O is an other bottleneck to work with DDNs in a standard parallel HPC setting, which we will consider in more detail in a forthcoming paper.

1 Introduction

Deep Neural Network (DNN) architectures have improved considerably the accuracy in data classification opening the door for a plethora of use cases in image classification, speech recognition or semantic text understanding. However, the training of DNNs is a very compute intensive task. So, the raising interest in these architectures created a tremendous demand for compute resources which is further intensified by a race to greater sizes of DNNs.

Another important factor is the time necessary for training DNNs. To train a popular architecture like, e.g., GoogLeNet can easily take several days on a Graphics Processing Unit (GPU). To make things worse the training usually is an iterative process of trials and modifications in the DNN architecture. So, keeping training times tolerably is a key requirement to actually apply DNNs in research and industry.

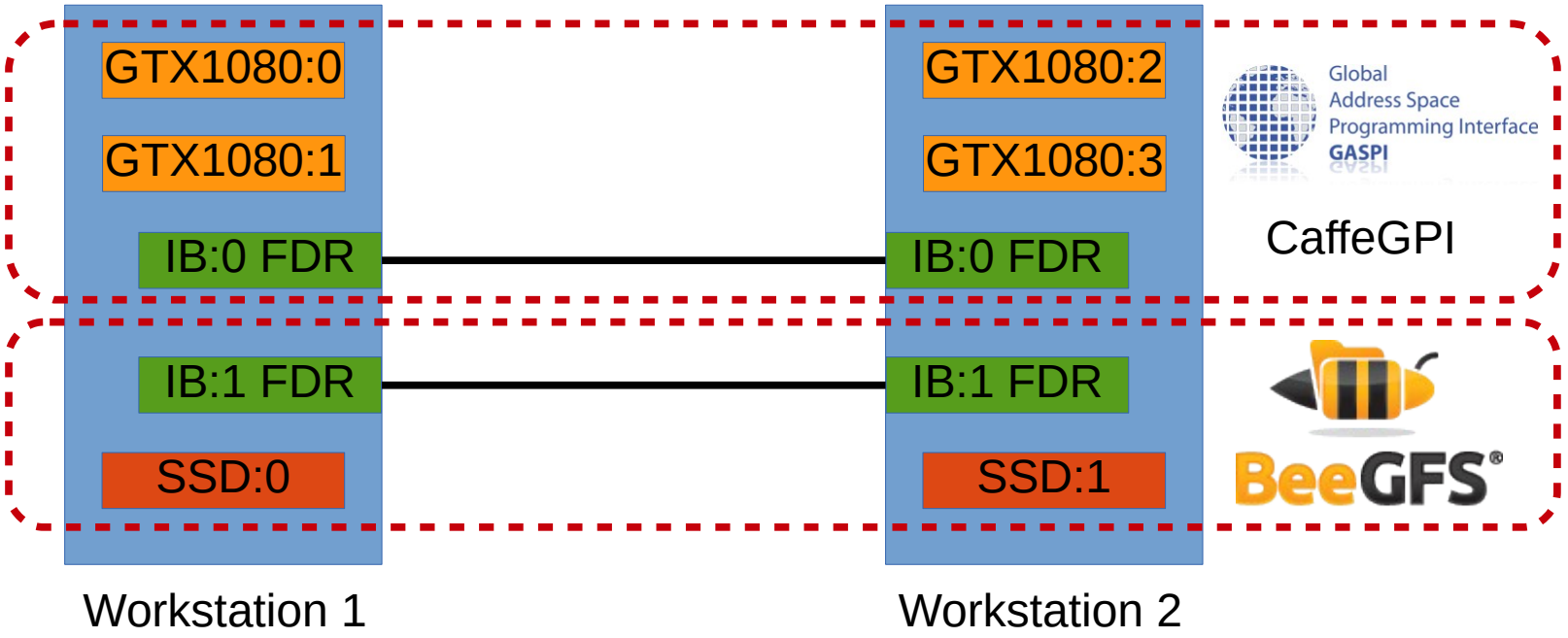
In response to this challenge, hardware vendors brought to market special hardware, e.g., the DGX-1 sold by NVIDIA or the S822LC ("Minsky") sold by IBM. They try to integrate as much compute power in terms of floating point operations per second (FLOP/s) as possible in a single compute node. While this special hardware comes also with a special price it is also not as flexible to apply to other problems in computer science. On the other hand, there already exists a plethora of compute systems in the world used for High Performance Computing

https://arxiv.org/abs/1706.00095

Fraunhofer

Projects: Low Cost Deep Learning Setup:
Build on CaffeGPI

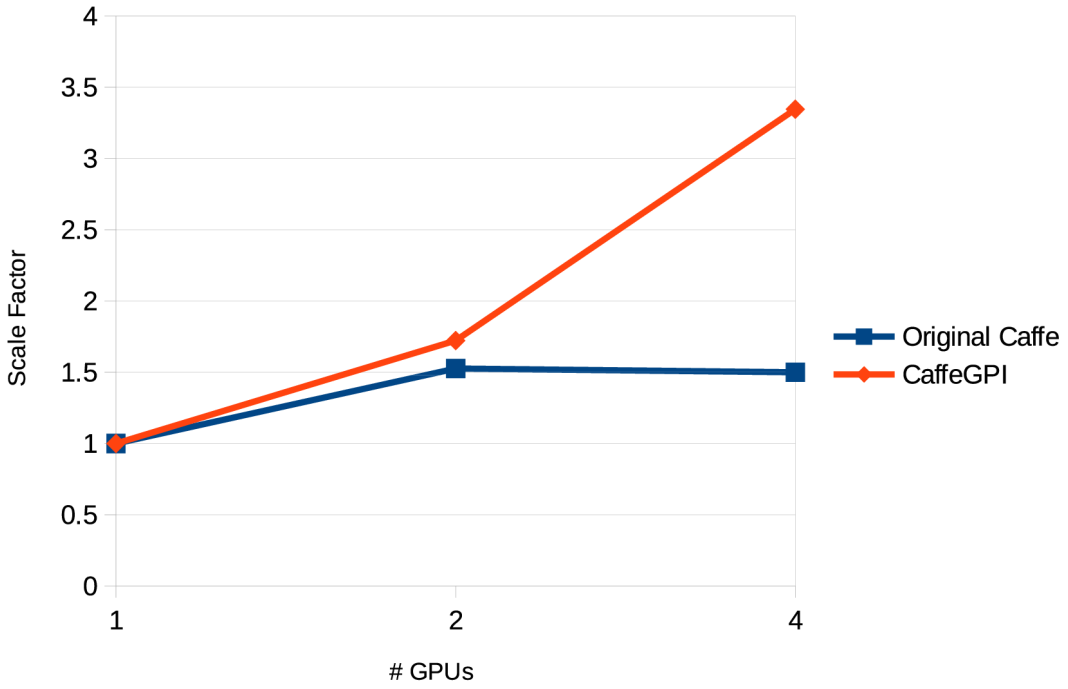
Price: <8000 EUR standard components



Specs:

- 32 GB GPU Mem
- 64 GB PGAS Mem
- 2TB BeeGFS for Train Data
- GPU interconnect: PCIe

CaffeGPI:
Benchmarks: Single Node

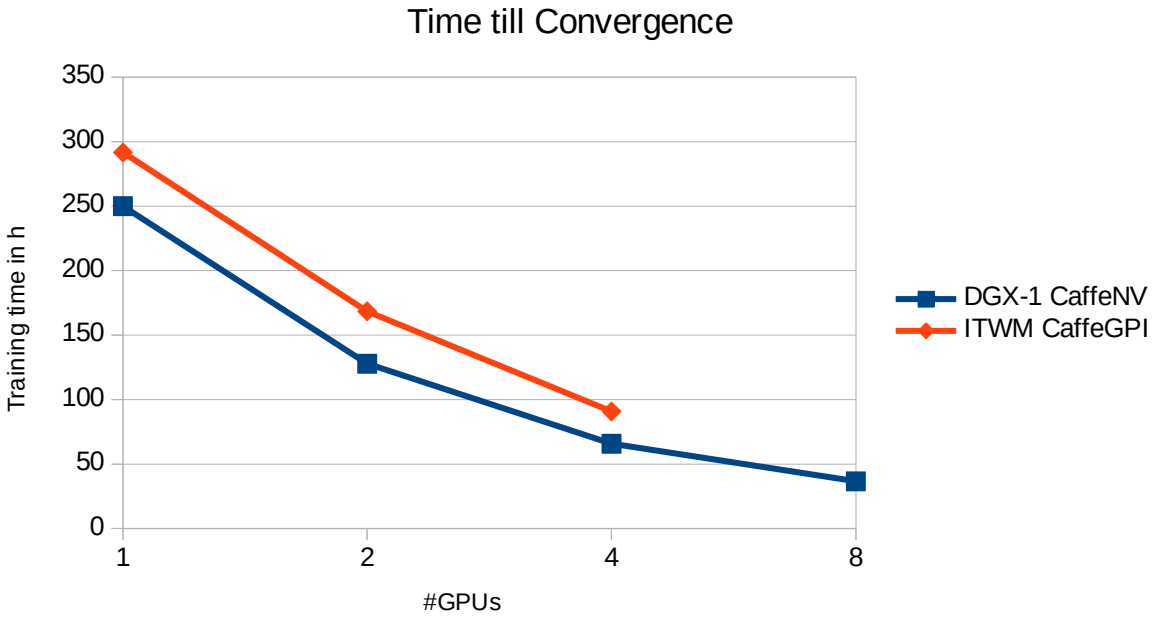


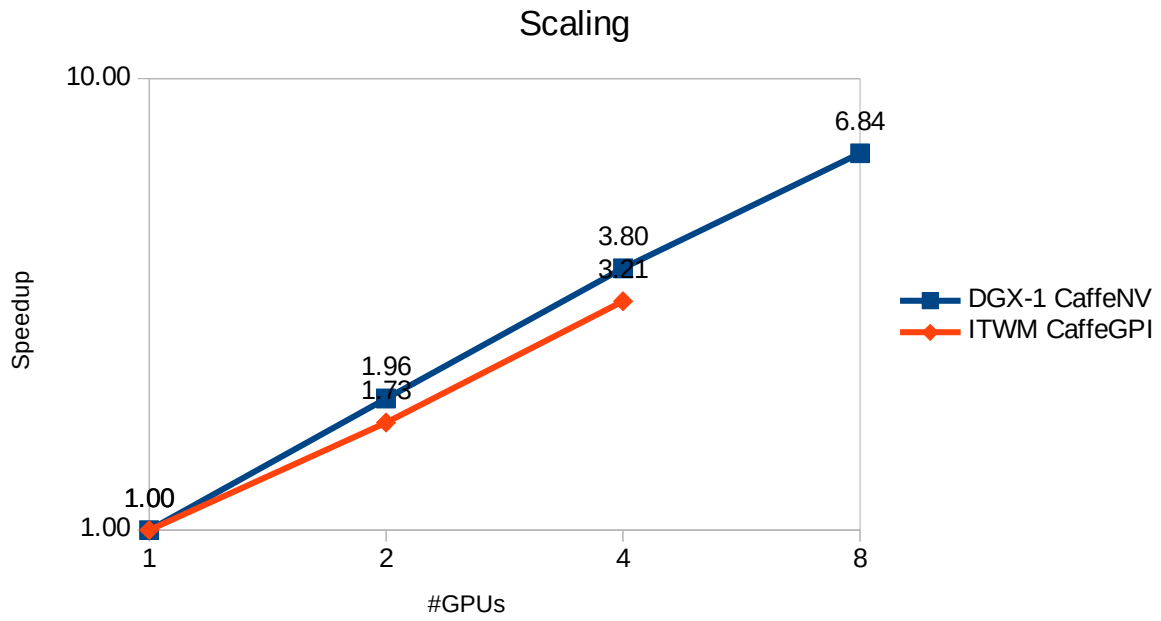
Specs:

4x K80 GPU (PCIe Interconnect)
Cuda 8
CuDNN 5.1

Topology: AlexNet
Batch Size (global) (1024)

Low Cost Deep Learning Setup:
Build on CaffeGPI

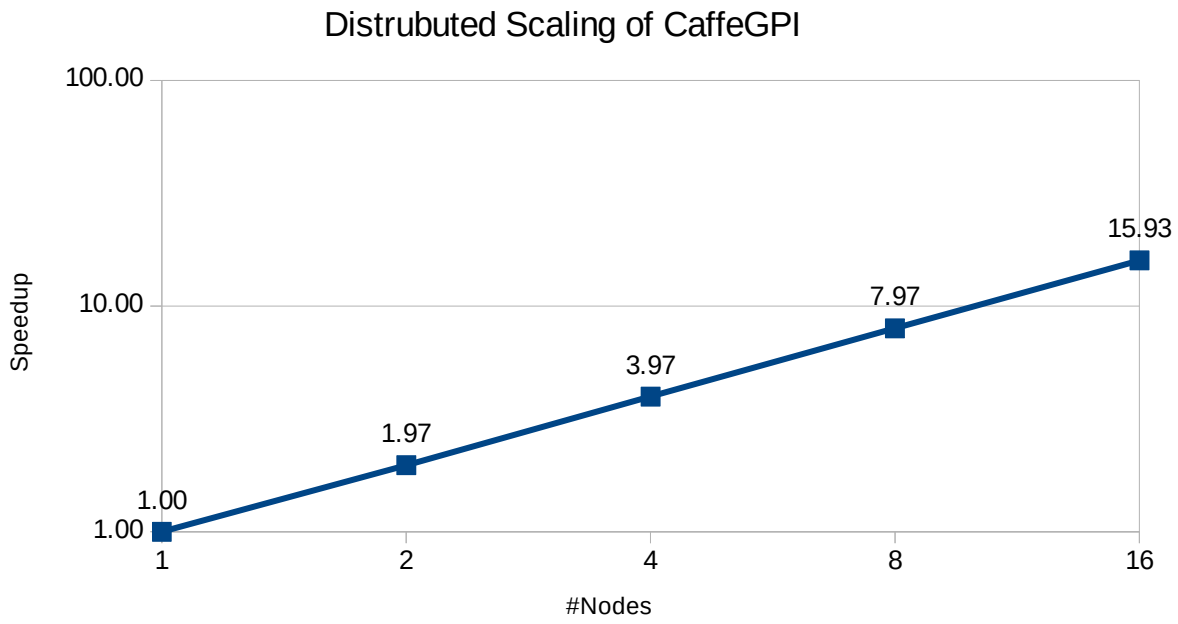




Specs: Topology: GoogLeNet, Cuda 8, cuDNN 5.1, CaffeNV 16.4, Batch Size/Node: 64

Fraunhofer

CaffeGPI:
Benchmarks



Specs:

1x K80 GPU (per node)
Cuda 8
CuDNN 5.1

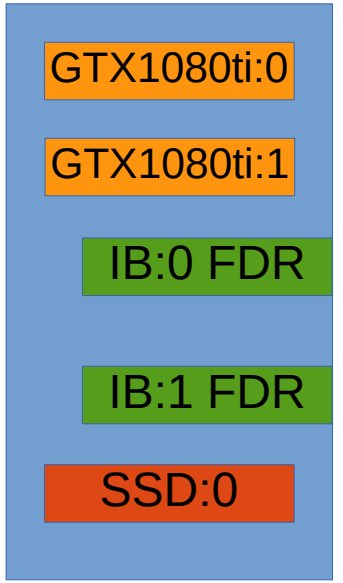
Topology: GloogleNet
Batch Size: 64 per node

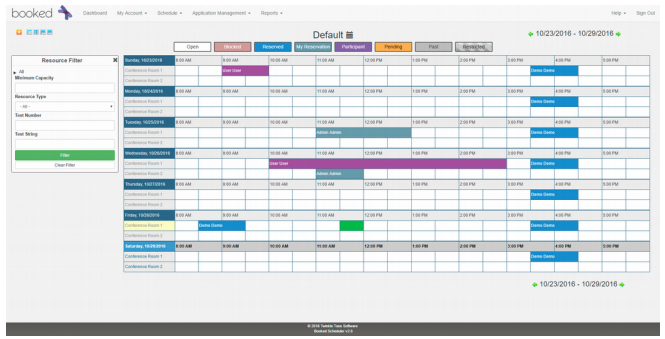
New Project: Low Cost Deep Learning Cluster

Bundesministerium für Bildung und Forschung

GPU Cluster for Fraunhofer Consortium @ITWM

- **Low cost Hardware**
  - Consumer GPUs
  - Novel AMD architecture
  - Hosting Cost per GPU ~ 1.25 k EUR. Compared to DGX-1 ~ 10k
- **Fast Interconnect and Data I/O**
  - Parallel FS with local NVMe
- **Open Source Multi-User Management**
  - Reservation system
  - Scheduling
  - Custom Containers
  - Web Interface



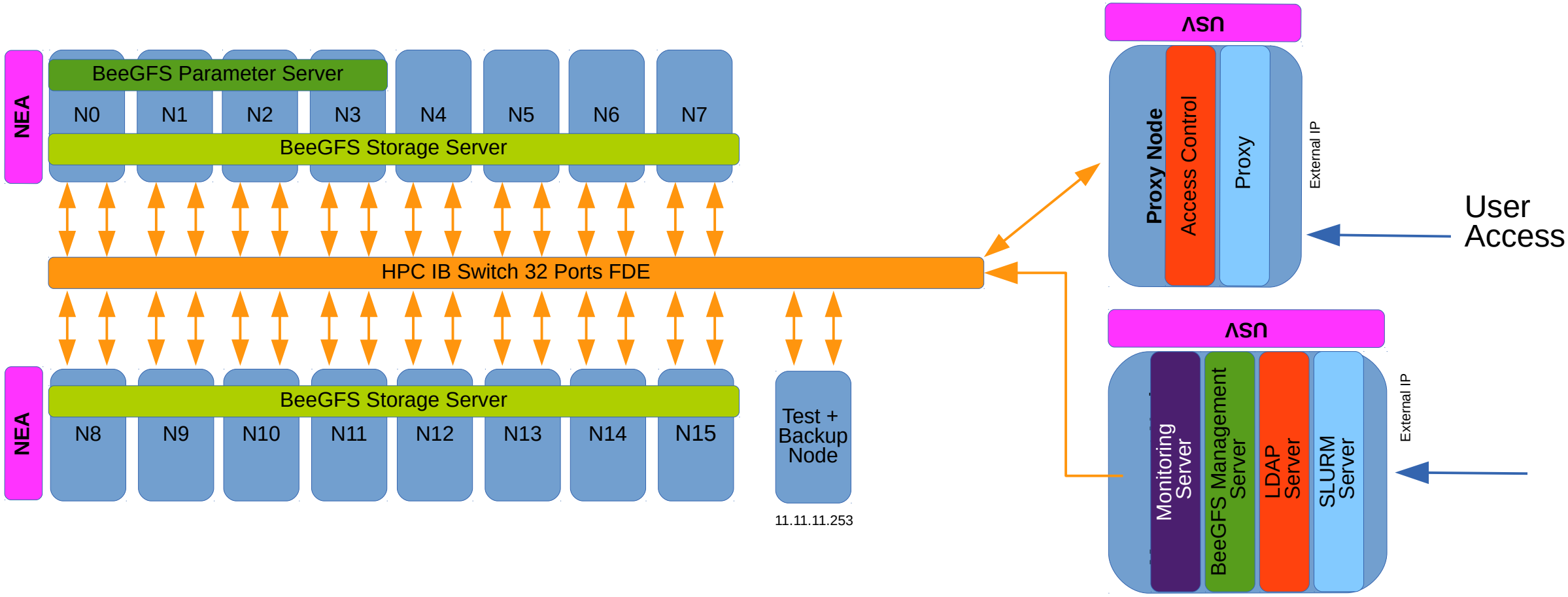


Fraunhofer

Low Cost Deep Learning Setup

Currently building: Prototype 16-Node System

Bundesministerium für Bildung und Forschung



Fraunhofer

Project Carme

An open source software stack for multi-user GPU clusters

Bundesministerium für Bildung und Forschung

Carme (/ˈkɑːrmiː/ KAR-mee; Greek: Κάρμη) is a **Jupiter** moon, also giving the name for a **Cluster** of Jupiter moons (the carme group).

Or in our case:

an open source frame work to mange resources for multiple users running **Jupyter** notebooks on a **Cluster** of compute nodes.

Fraunhofer

Project Carme

An open source software stack for multi-user GPU clusters

Common problems in GPU-Cluster oppration:

- **Interactive, secure multi user environment**
  - ML and Data Science users want interactive GUI access to compute resources
- **Resource Management**
  - How to assign (GPU) resources to competing users?
    - User management
    - Accounting
    - Job scheduling
    - Resource reservation
- **Data I/O**
  - Get user data to compute nodes (I/O Bottleneck)
- **Maintenance**
  - Meet (fast changing and diverse) software demands of users

Project Carme

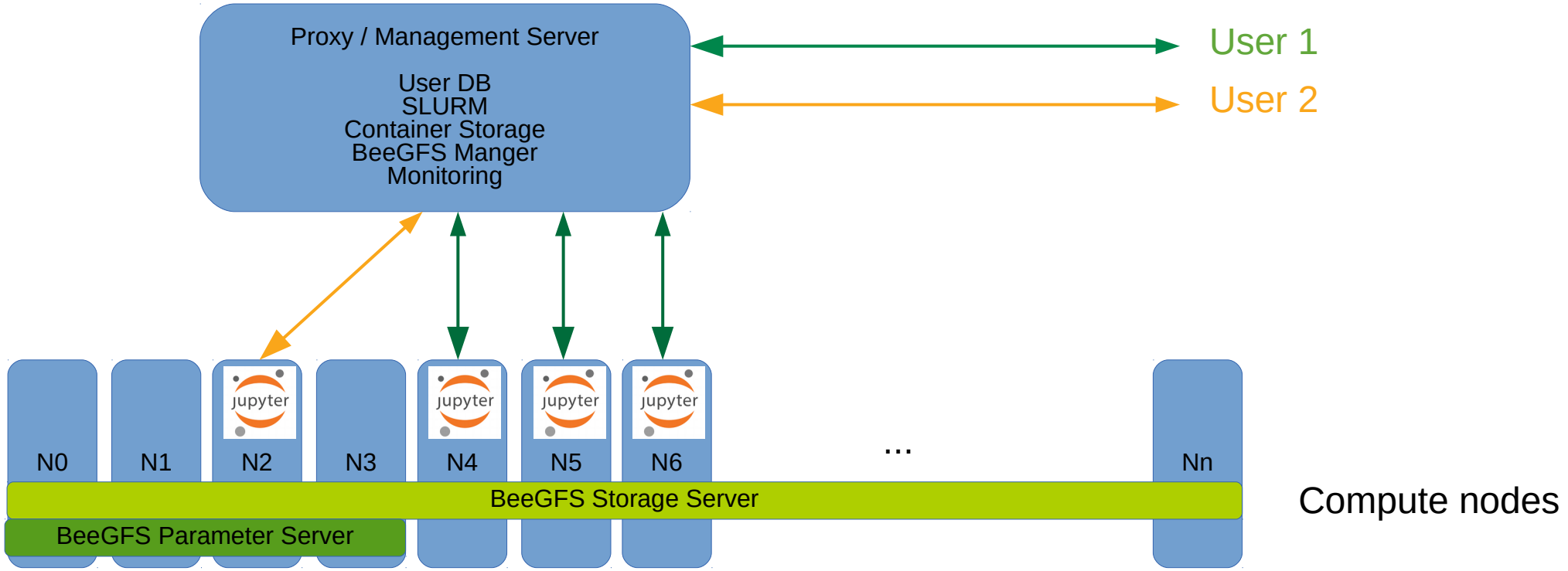
An open source software stack for multi-user GPU clusters

***Carme* core idea:**

- **Combine established open source ML and DS tools with HPC back-ends**
  - Use containers
    - (for now) Docker
  - Use Jupyter Notebooks as main web based GUI-Frontend
    - All web front-end (OS independent, no installation on user side needed)
  - Use HPC job management and scheduler
    - SLURM
  - Use HPC data I/O technology
    - ITWM's BeeGFS
  - Use HPC maintenance and monitoring tools

Project Carme

An open source software stack for multi-user GPU clusters

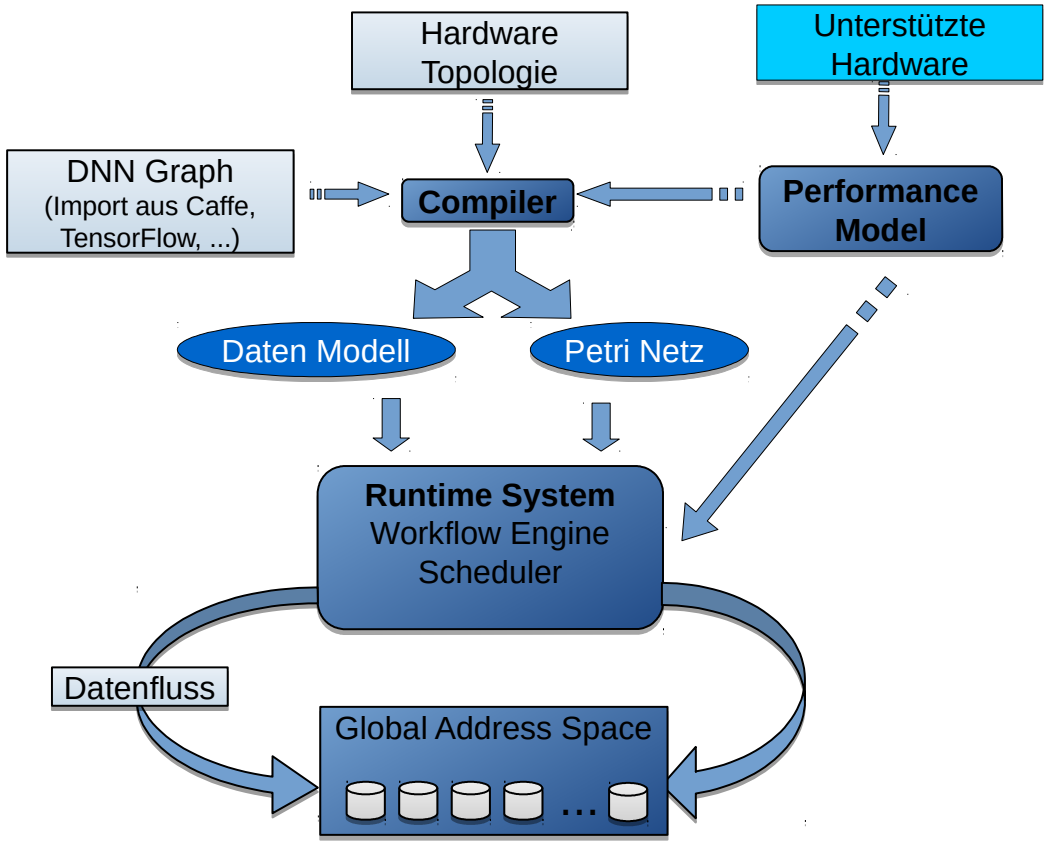


Fraunhofer

HP-DLF

High Performance Deep Learning Framework

- Scalable
- Transparent
- generic
- Auto-parallel
- Elastic
- Automatic data flow
- Automatic Hardware selection
- Portable
- Monitoring
- Simulation
- **New optimization methods**



Bundesministerium für Bildung und Forschung

Fraunhofer ITWM

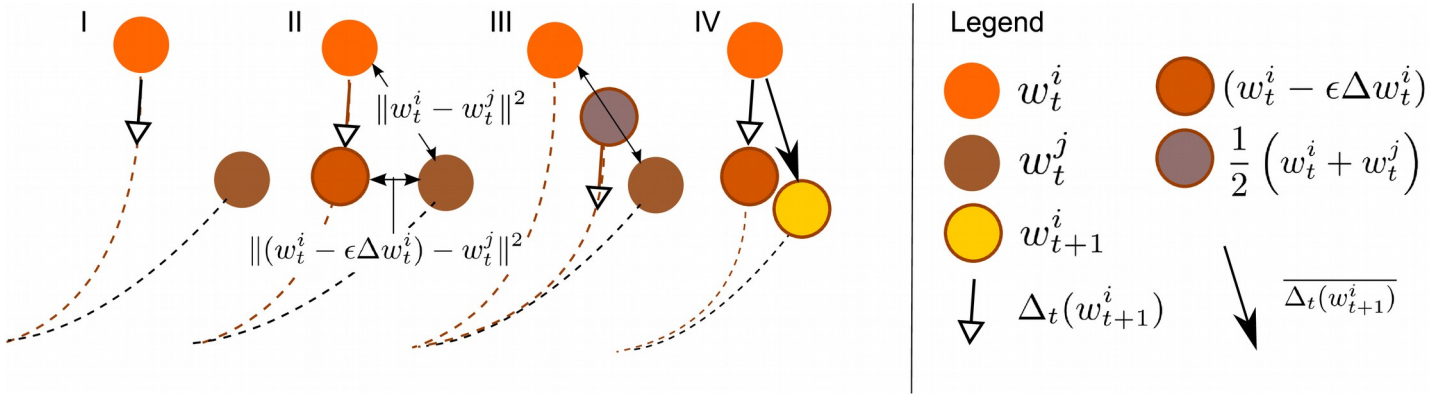
TECHNISCHE UNIVERSITÄT DRESDEN

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

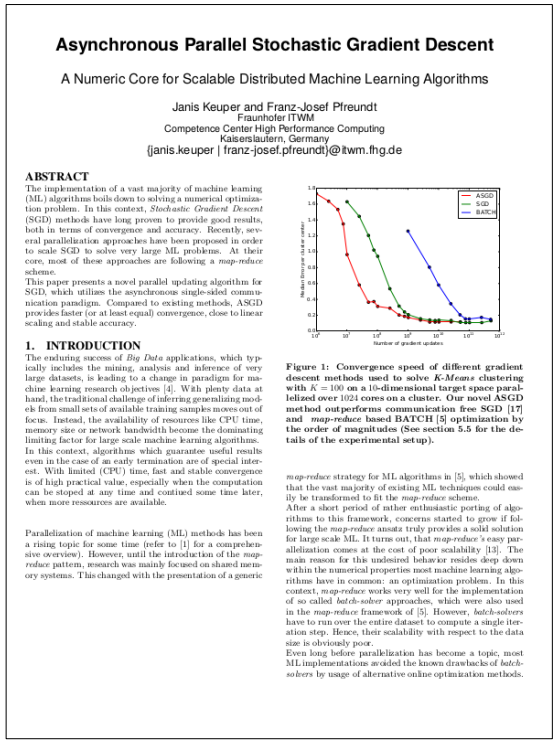
DFKI

Optimization for HP-DLF

Our asynchronous optimization algorithm



- Sparse communication for multi model optimization
- Lower demands on the communication bandwidth
- Superior convergence



Asynchronous Parallel Stochastic Gradient Descent

A Numeric Core for Scalable Distributed Machine Learning Algorithms

Janis Keuper and Franz-Josef Pfreundt
Fraunhofer ITWM
Competence Center High Performance Computing
Kaiserslautern, Germany
{janis.keuper | franz-josef.pfreundt}@itwm.fhg.de

ABSTRACT

The implementation of a vast majority of machine learning (ML) algorithms boils down to solving a numerical optimization problem. In this context, *Stochastic Gradient Descent* (SGD) methods have long proven to provide good results, both in terms of convergence and accuracy. Recently, several parallelization approaches have been proposed in order to scale SGD to solve very large ML problems. At their core, most of these approaches are following a *map-reduce* scheme.
This paper presents a novel parallel updating algorithm for SGD, which utilizes the asynchronous single-sided communication paradigm. Compared to existing methods, ASGD provides faster (or at least equal) convergence, close to linear scaling and stable accuracy.

1. INTRODUCTION

The enduring success of *Big Data* applications, which typically includes the mining, analysis and inference of very large datasets, is leading to a change in paradigm for machine learning research objectives [4]. With plenty data at hand, the traditional challenge of inferring generalizing models from small sets of available training samples moves out of focus. Instead, the availability of resources like CPU time, memory size or network bandwidth become the dominating limiting factor for large scale machine learning algorithms.
In this context, algorithms which guarantee useful results even in the case of an early termination are of special interest. With limited (CPU) time, fast and stable convergence is of high practical value, especially when the computation can be stoped at any time and continued some time later, when more ressources are available.

Parallelization of machine learning (ML) methods has been a rising topic for some time (refer to [1] for a comprehensive overview). However, until the introduction of the *map-reduce* pattern, research was mainly focused on shared memory systems. This changed with the presentation of a generic *map-reduce* strategy for ML algorithms in [5], which showed that the vast majority of existing ML techniques could easily be transformed to fit the *map-reduce* scheme.
After a short period of rather enthusiastic porting of algorithms to this framework, concerns started to grow if following the *map-reduce* ansatz truly provides a solid solution for large scale ML. It turns out, that *map-reduce*'s easy parallelization comes at the cost of poor scalability [13]. The main reason for this undesired behavior resides deep down within the numerical properties most machine learning algorithms have in common: an optimization problem. In this context, *map-reduce* works very well for the implementation of so called *batch-solver* approaches, which were also used in the *map-reduce* framework of [5]. However, *batch-solvers* have to run over the entire dataset to compute a single iteration step. Hence, their scalability with respect to the data size is obviously poor.
Even long before parallelization has become a topic, most ML implementations avoided the known drawbacks of *batch-solvers* by usage of alternative online optimization methods.

Figure 1: Convergence speed of different gradient descent methods used to solve *K-Means* clustering with $K = 100$ on a 10-dimensional target space parallelized over 1024 cores on a cluster. Our novel ASGD method outperforms communication free SGD [17] and *map-reduce* based BATCH [5] optimization by the order of magnitudes (See section 5.5 for the details of the experimental setup).

Fraunhofer

V A Look a the near Future

For HPC:

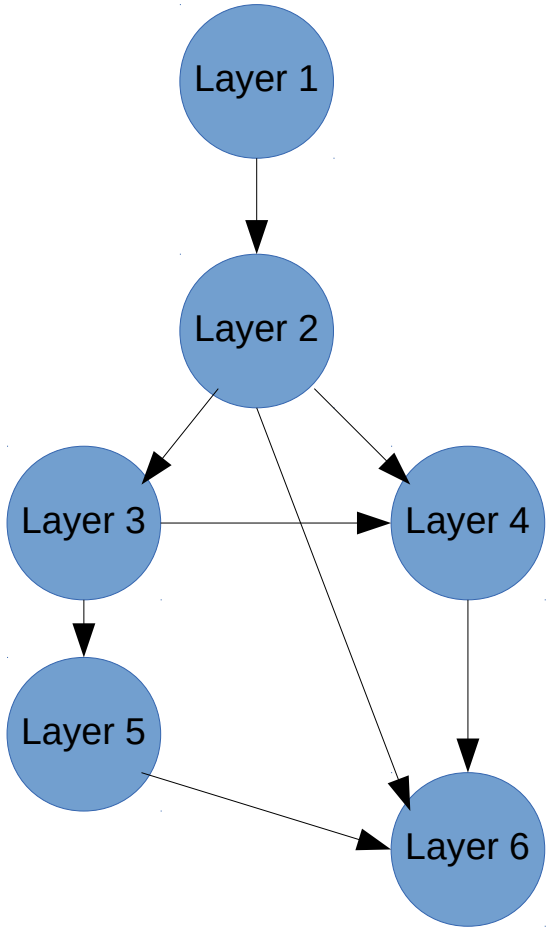
- **New Hardware Accelerators**
- **New Interconnects**
- **New Architectures**

From ML

- **Models are still growing!**
- **Learning to learn**

Automatic Design

Of Deep Neural Networks



Basically, a graph optimization problem :

- ***Select Node Types***
  - And their Meta-Parameters
- ***Connect Edges*** to define the data flow through the graph

Optimization target: minimize test error

Problems:

- **Huge and difficult search space**
- **Each iteration requires training of a DNN**

Automatic Design

Current Approaches: Reinforcement Learning

Model	Error rate	# params ($\times 10^6$)
Maxout [7]	9.38	–
Network in Network [19]	8.81	–
VGG [27] [1]	7.94	15.2
ResNet [10]	6.61	1.7
MetaQNN [1] [2]	9.09	3.7
Neural Architecture Search [39]	3.84	32.0
CGP-CNN (ConvSet)	6.75	1.52
CGP-CNN (ResSet)	5.98	1.68

Evaluation on CIFAR-10:
Better than “State of the Art” (hand designed) performance

- After ~12500 iterations
- Compute time: ~ 10000 GPU days

Under review as a conference paper at ICLR 2017

NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING

Barret Zoph*, Quoc V. Le
Google Brain
{barretzoph,qvl}@google.com

ABSTRACT

Neural networks are powerful and flexible models that work well for many difficult learning tasks in image, speech and natural language understanding. Despite their success, neural networks are still hard to design. In this paper, we use a recurrent network to generate the model descriptions of neural networks and train this RNN with reinforcement learning to maximize the expected accuracy of the generated architectures on a validation set. On the CIFAR-10 dataset, our method, starting from scratch, can design a novel network architecture that rivals the best human-invented architecture in terms of test set accuracy. Our CIFAR-10 model achieves a test error rate of 3.84, which is only 0.1 percent worse and 1.2x faster than the current state-of-the-art model. On the Penn Treebank dataset, our model can compose a novel recurrent cell that outperforms the widely-used LSTM cell, and other state-of-the-art baselines. Our cell achieves a test set perplexity of 62.4 on the Penn Treebank, which is 3.6 perplexity better than the previous state-of-the-art.

1 INTRODUCTION

The last few years have seen much success of deep neural networks in many challenging applications, such as speech recognition (Hinton et al., 2012), image recognition (LeCun et al., 1998; Krizhevsky et al., 2012) and machine translation (Sutskever et al., 2014; Bahdanau et al., 2015; Wu et al., 2016). Along with this success is a paradigm shift from feature designing to architecture designing, i.e., from SIFT (Lowe, 1999), and HOG (Dalal & Triggs, 2005), to AlexNet (Krizhevsky et al., 2012), VGGNet (Simonyan & Zisserman, 2014), GoogleNet (Szegedy et al., 2015), and ResNet (He et al., 2016a). Although it has become easier, designing architectures still requires a lot of expert knowledge and takes ample time.

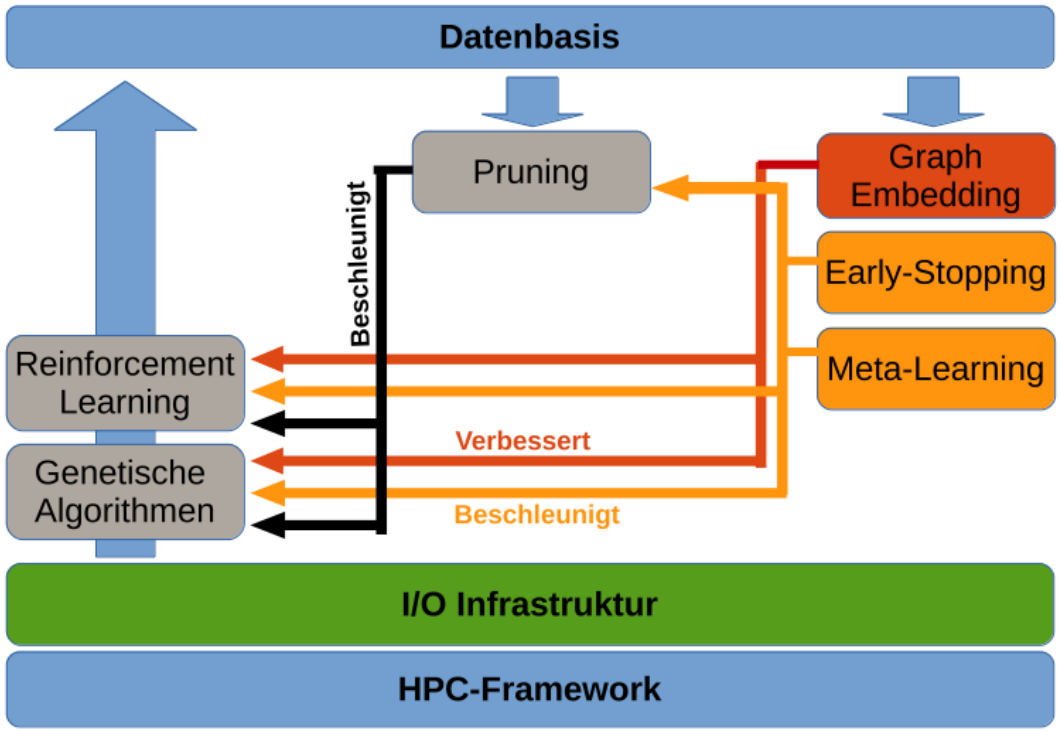
Figure 1: An overview of Neural Architecture Search.

This paper presents Neural Architecture Search, a gradient-based method for finding good architectures (see Figure 1). Our work is based on the observation that the structure and connectivity of a neural network can be typically specified by a variable-length string. It is therefore possible to use

*Work done as a member of the Google Brain Residency program (g.co/brainresidency).

DeToL

Deep Topology Learning



- Genetic Algorithms
- Reinforcement Learning
- Early Stopping
- Graph Embedding
- Meta-Learning
- Pruning

On application size problems

Data basis generartion

Bundesministerium für Bildung und Forschung

UNIVERSITÄT MANNHEIM

UNI FREIBURG

lrz Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften

PSIORI

$\Pr(E|H_p) \times \Pr(H_p)$

Fraunhofer

Discussion


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS