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

Janis Keuper

Itwm.fraunhofer.de/ml

Competence Center High Performance Computing
Fraunhofer ITWM, Kaiserslautern, Germany

Outline

Training Deep Neural Networks

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!)

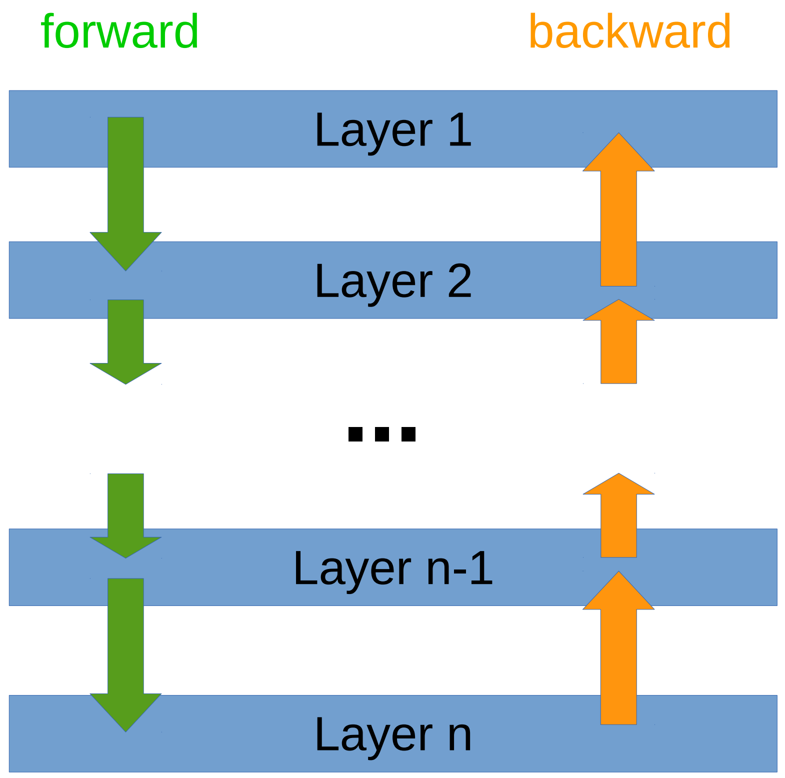
$$J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2$$

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

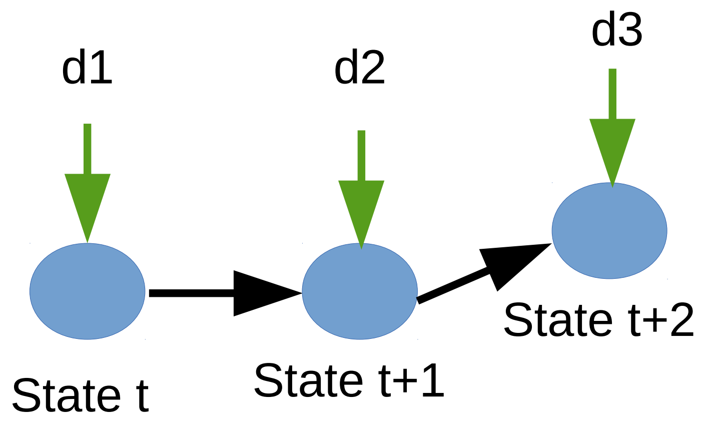
Common approaches to parallelize SGD for DL

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



Fraunhofer

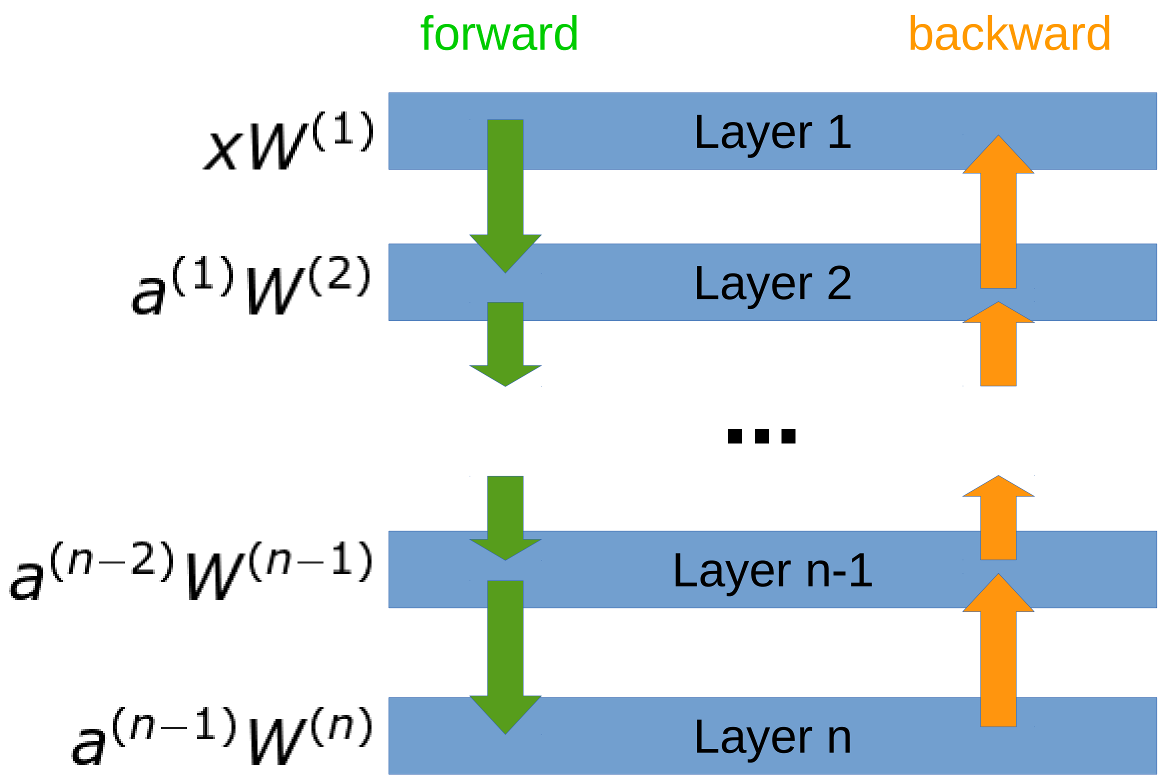
Parallelization

Common approaches to parallelize SGD for DL

Internal parallelization

- Dense matrix multiplication:
  - standard blas sgemm
  - MKL, Open-Blas
  - **CuBlas**

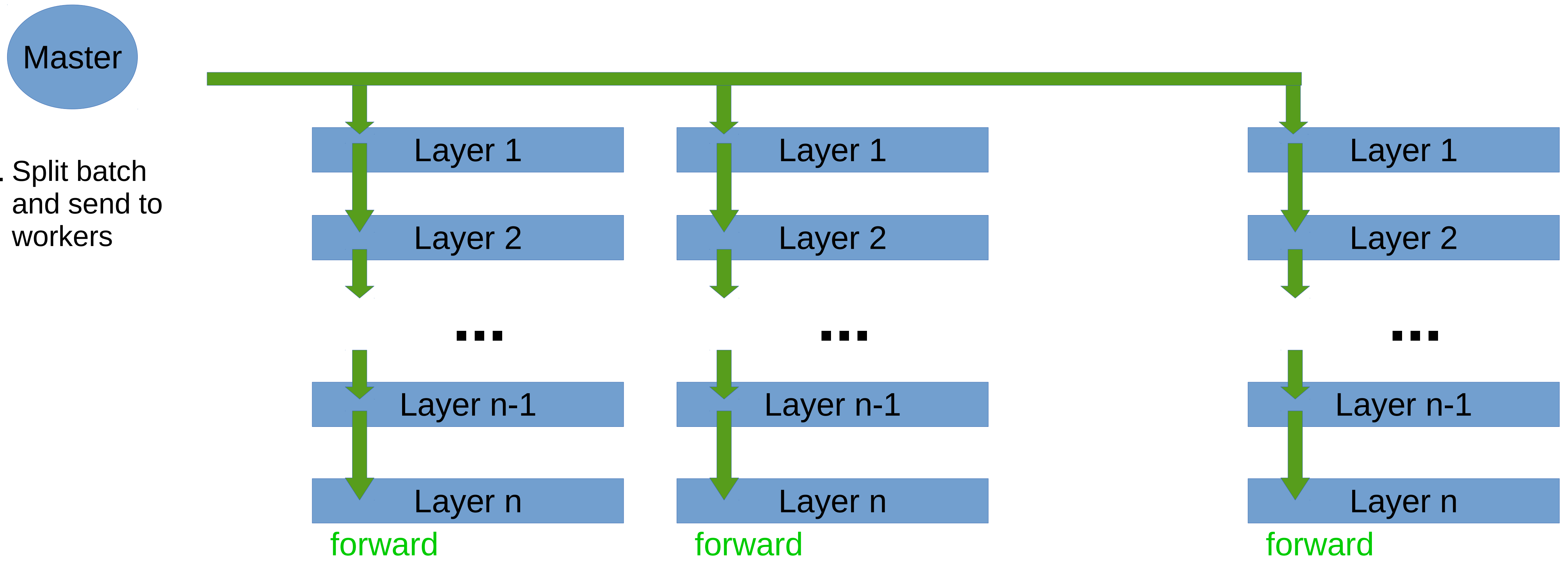
- Task parallelization for special Layers
  - Cuda-CNN for fast convolutions



Parallelization

Common approaches to parallelize SGD for DL

External: data parallelization over the data batch

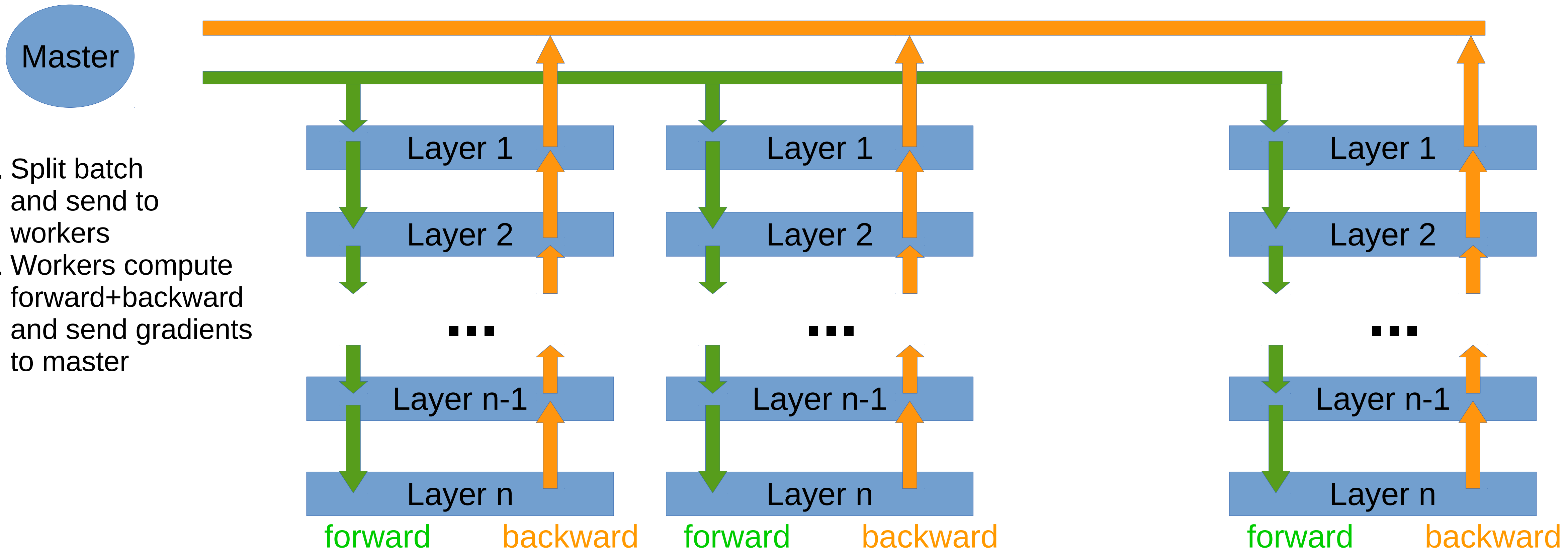


1. Split batch and send to workers

Parallelization

Common approaches to parallelize SGD for DL

External: data parallelization over the data batch

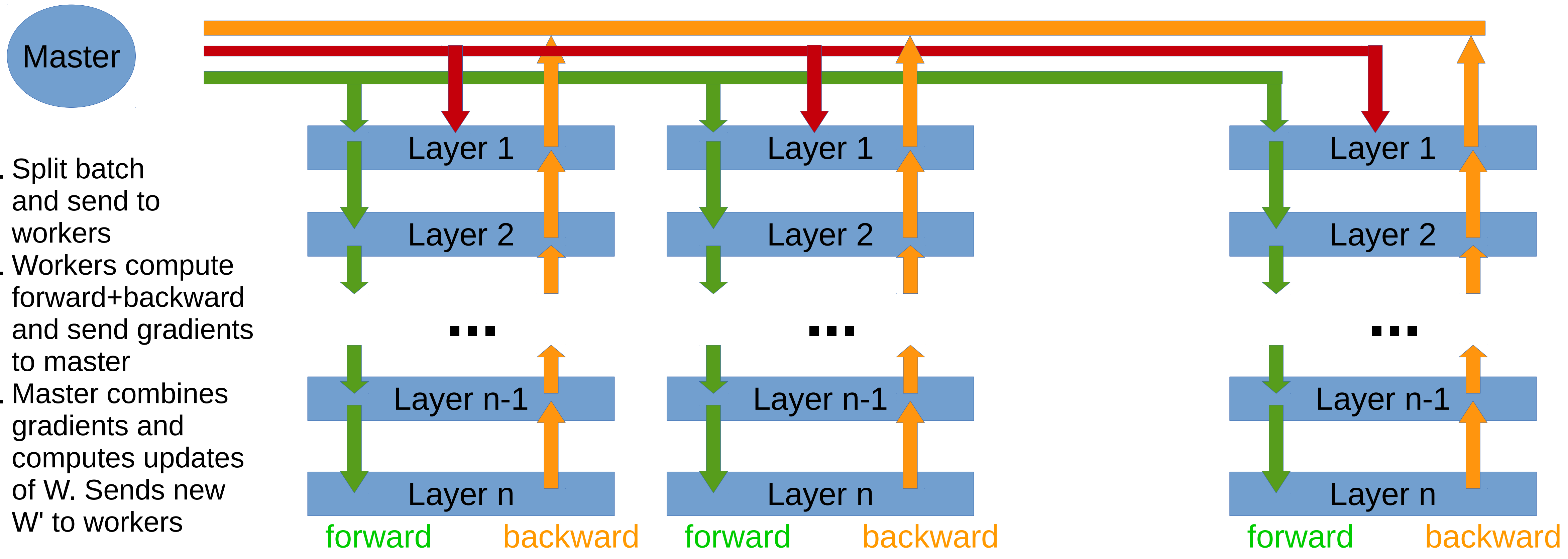


1. Split batch and send to workers
2. Workers compute forward+backward and send gradients to master

Parallelization

Common approaches to parallelize SGD for DL

External: data parallelization over the data batch

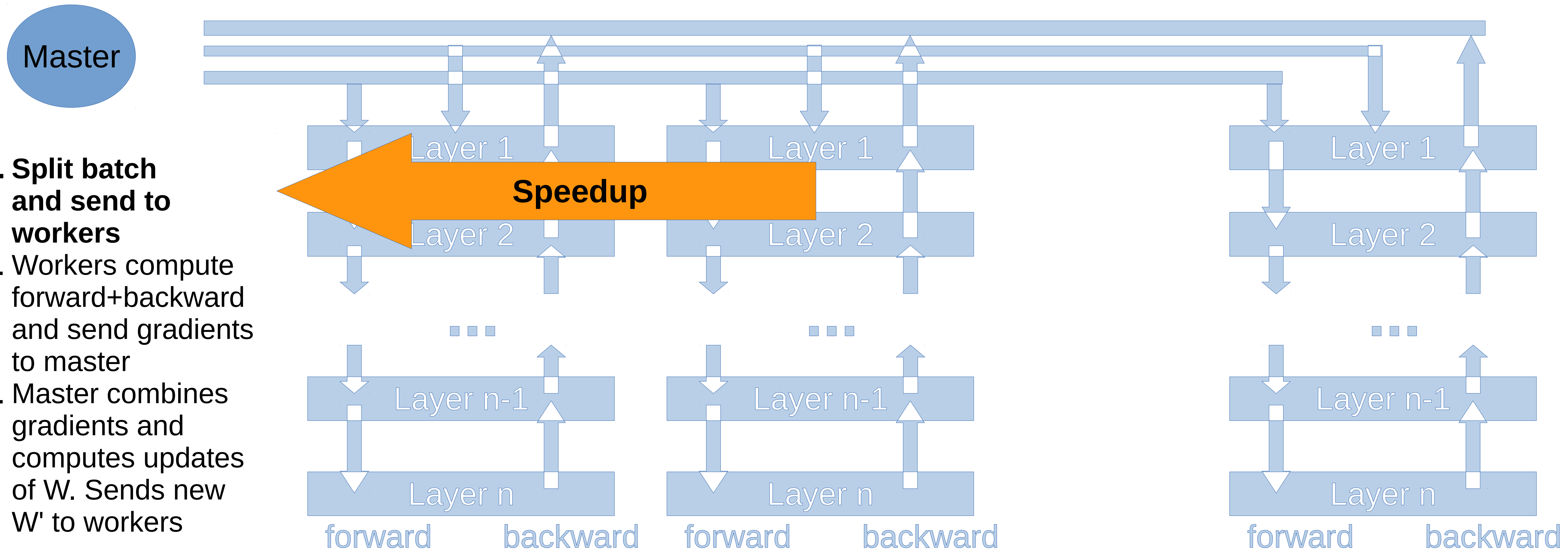


1. Split batch and send to workers
2. Workers compute forward+backward and send gradients to master
3. Master combines gradients and computes updates of W. Sends new W' to workers

Parallelization

Common approaches to parallelize SGD for DL

External: data parallelization over the data batch



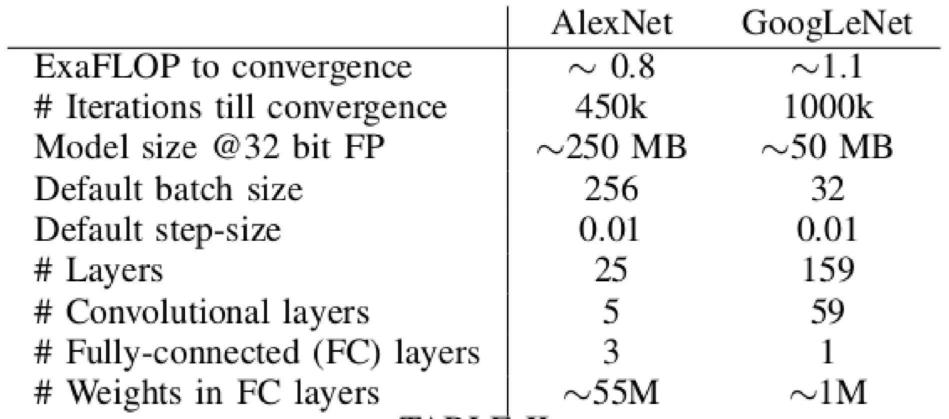
1. **Split batch and send to workers**
2. Workers compute forward+backward and send gradients to master
3. Master combines gradients and computes updates of W. Sends new W' to workers

Fraunhofer

Experimental Evaluation

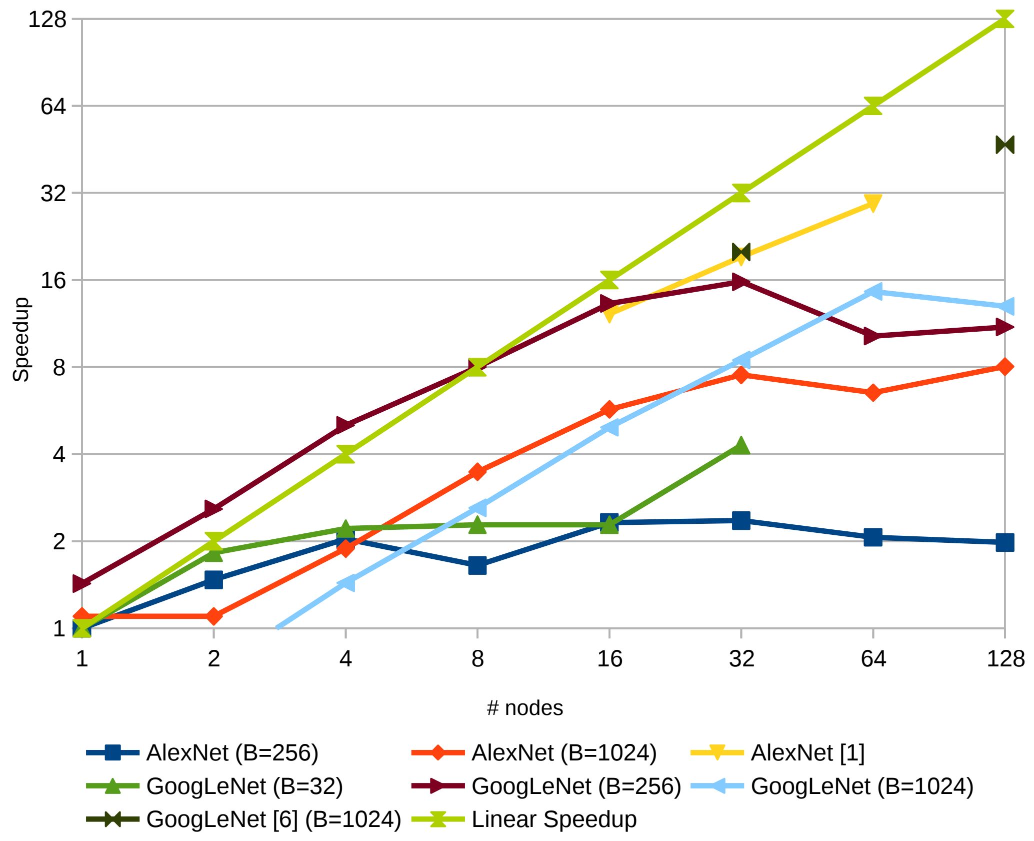
Scaling Distributed Parallel Synchronous SGD Training

Experimental Setup:
HPC Cluster with FDR Infiniband Interconnect
K80 GPUs / Xeon E5 CPUs
Intel Caffe Distribution



	AlexNet	GoogLeNet
ExaFLOP to convergence	$\sim$ 0.8	$\sim$1.1
# Iterations till convergence	450k	1000k
Model size @32 bit FP	$\sim$250 MB	$\sim$50 MB
Default batch size	256	32
Default step-size	0.01	0.01
# Layers	25	159
# Convolutional layers	5	59
# Fully-connected (FC) layers	3	1
# Weights in FC layers	$\sim$55M	$\sim$1M

TABLE II
PROPERTIES OF THE DEEP NEURAL NETWORKS USED FOR THE FOLLOWING BENCHMARKS.

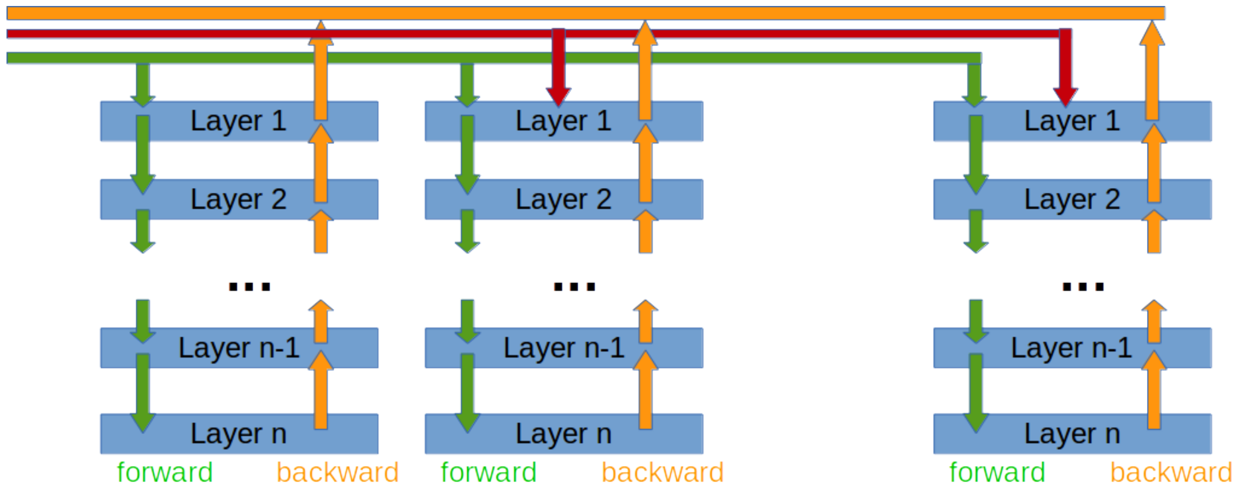


Limitation I

Distributed SGD is heavily Communication Bound

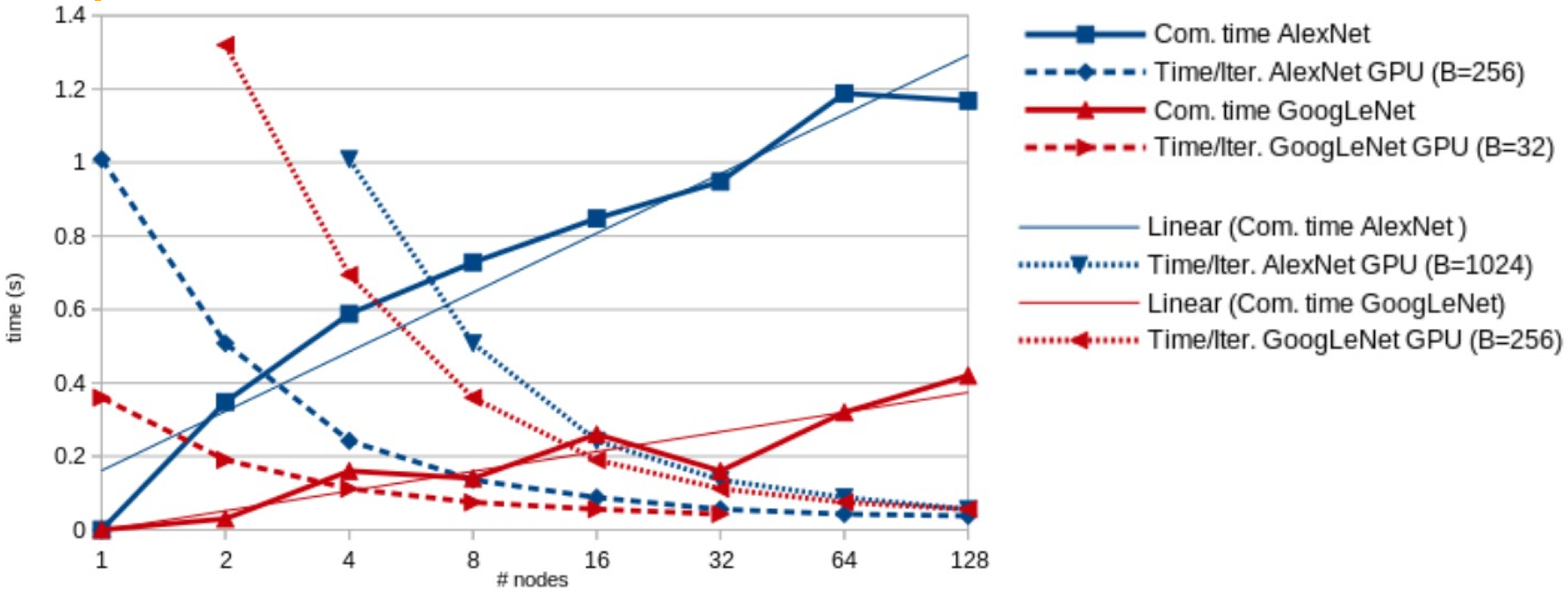
Gradients have the same size as the model

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



Communication Bound

Experimental Evaluation



Fraunhofer

Communication Bottleneck

Possible Solutions

- Network Design
  - Avoid fully connected Layers for smaller models (see GoogLeNet vs AlexNet)

- Reduce Model Size
  - Reduce Floating Point precision (8 Bit is enough)

- Reduce / Avoid Communication
  - Sparse Updates
  - Compression
  - Asynchronous Updates

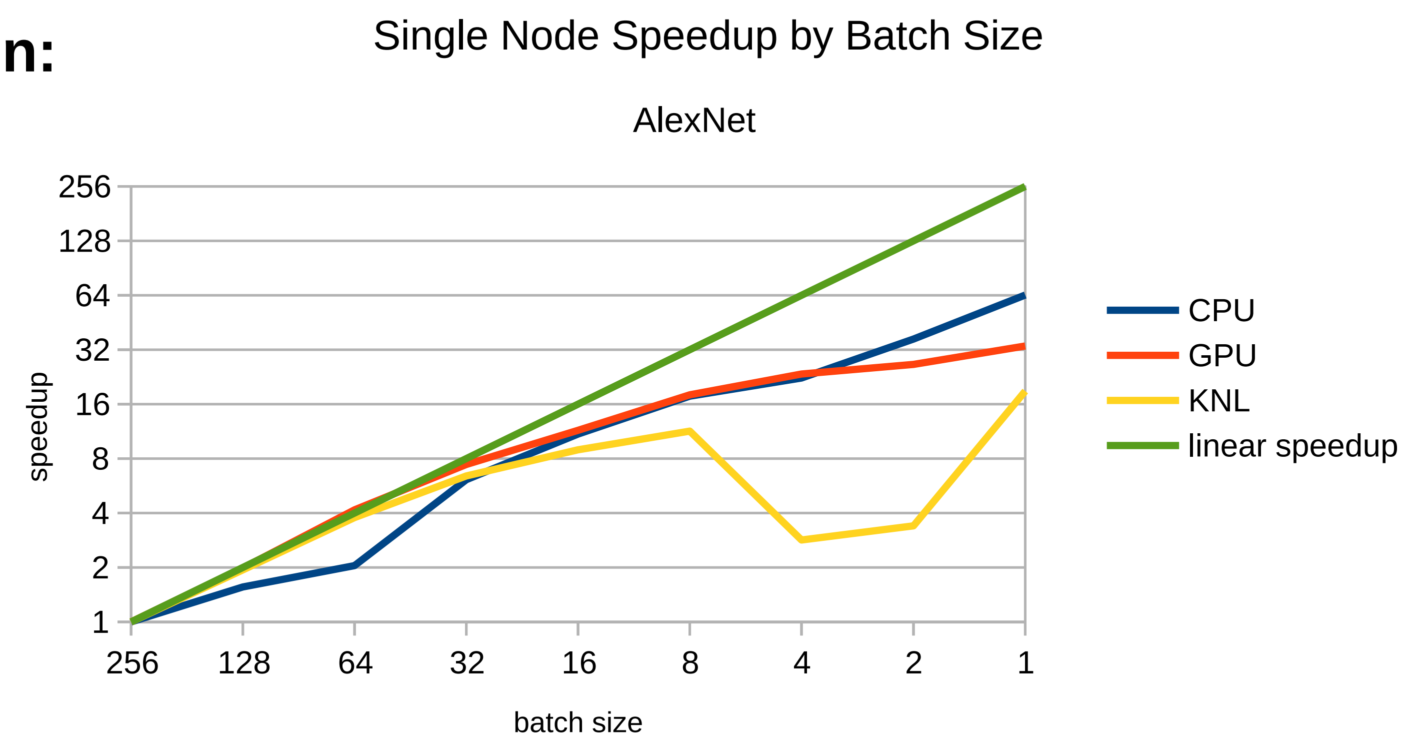
Janis Keuper and Franz-Josef Pfreundt. 2015. **Asynchronous parallel stochastic gradient descent: a numeric core for scalable distributed machine learning algorithms.** In Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments (MLHPC '15). ACM, New York, NY, USA, , Article 1 , 11 pages. DOI=http://dx.doi.org/10.1145/2834892.2834893

Experimental Evaluation

Assuming free Communication

Simulating free communication:

Single Node Optimization with reduced Bach size

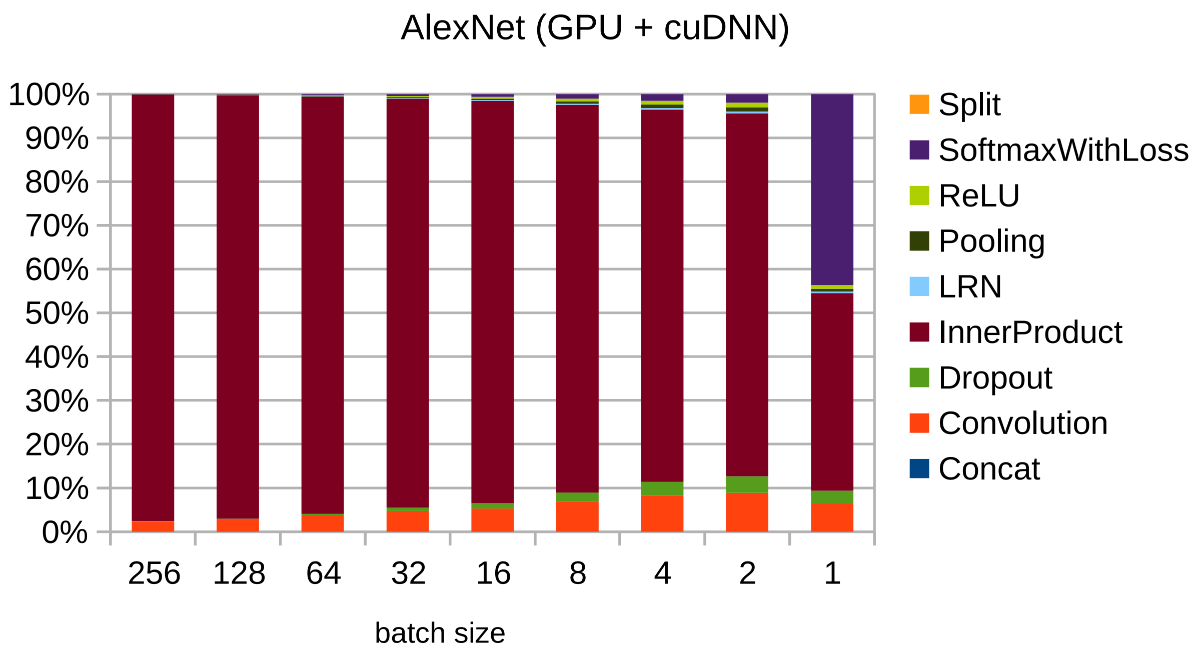


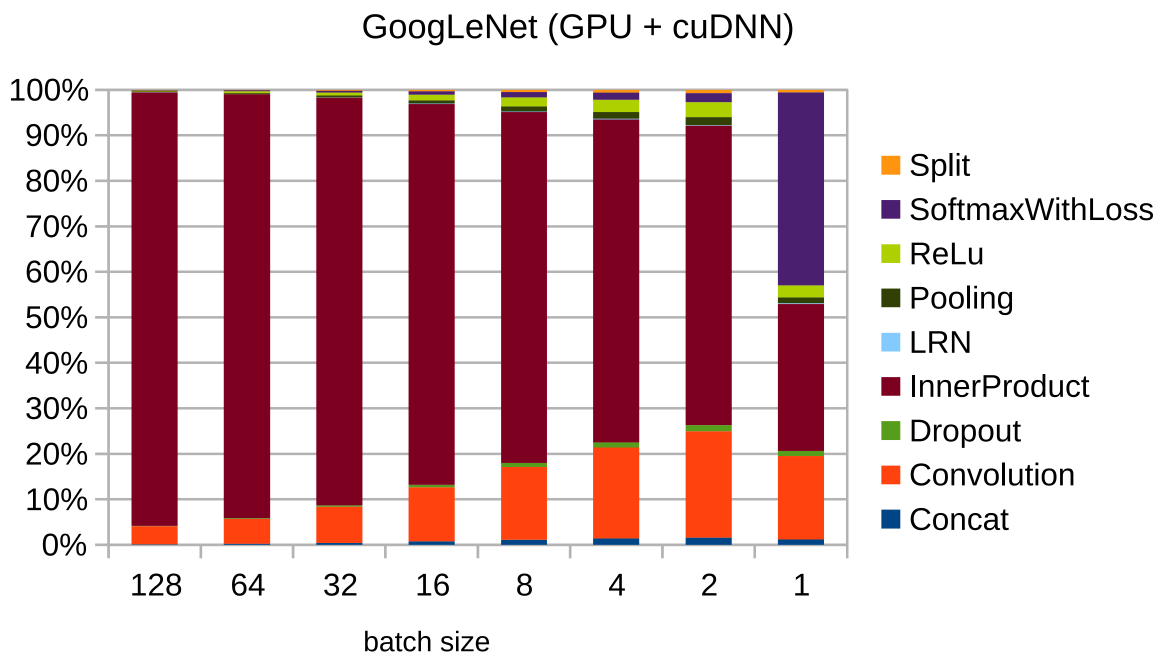
Fraunhofer

Experimental Evaluation

Compute Times Layer by Layer (assuming free Communication)

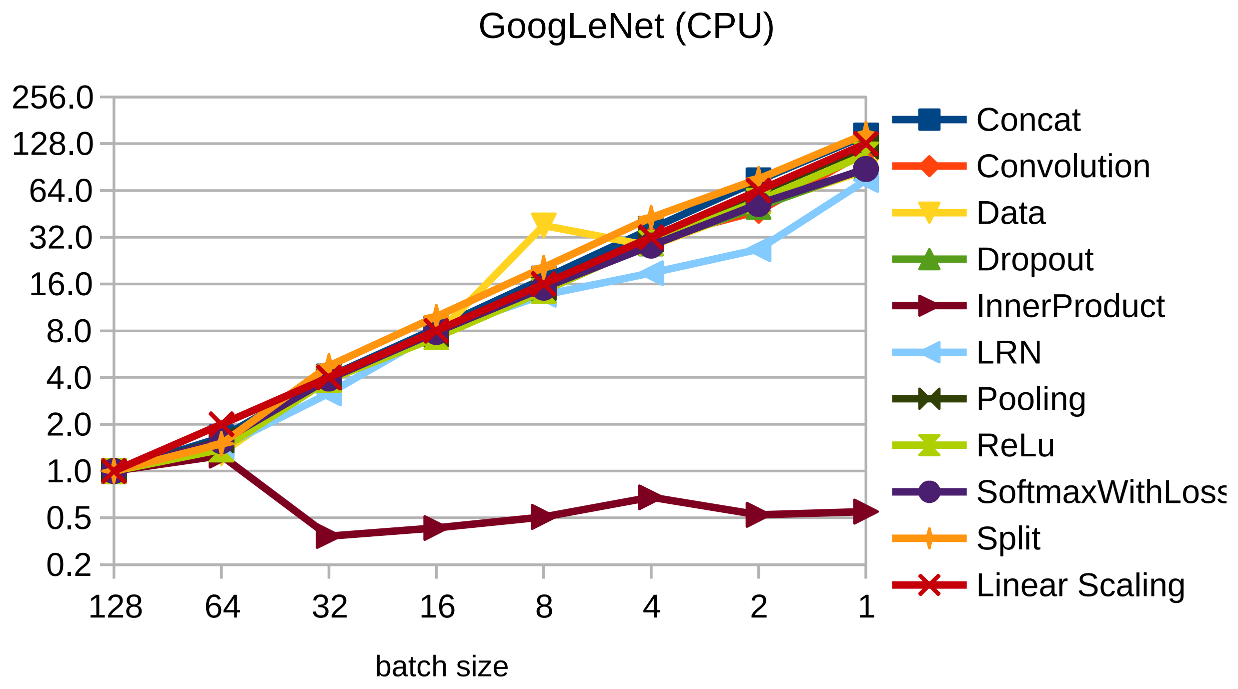
Compute time by Layer

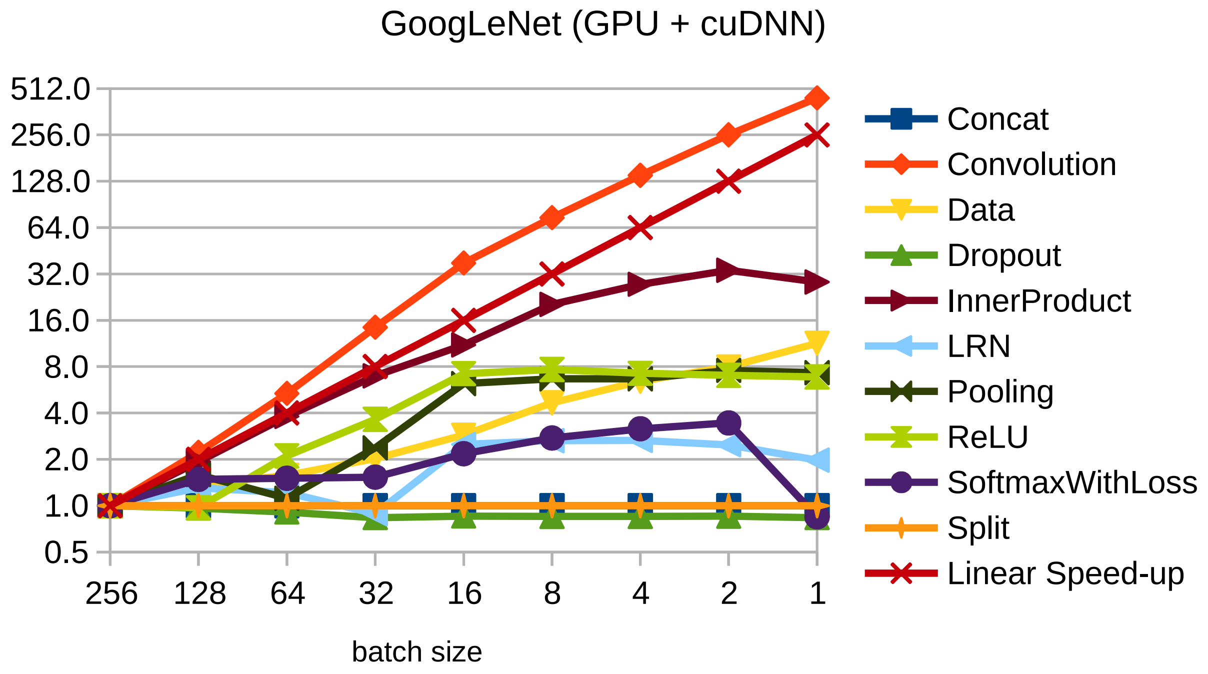




Experimental Evaluation

Speedup Layer by Layer (assuming free Communication)





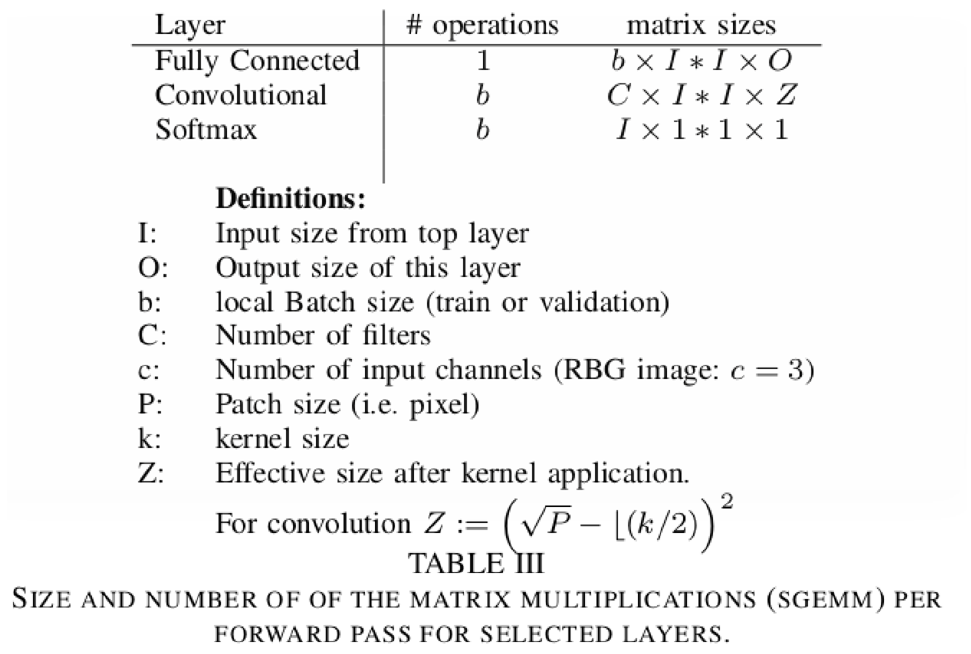
Fraunhofer

Experimental Evaluation

Speedup Matrix Multiplication

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	$I \times 1 * 1 \times 1$

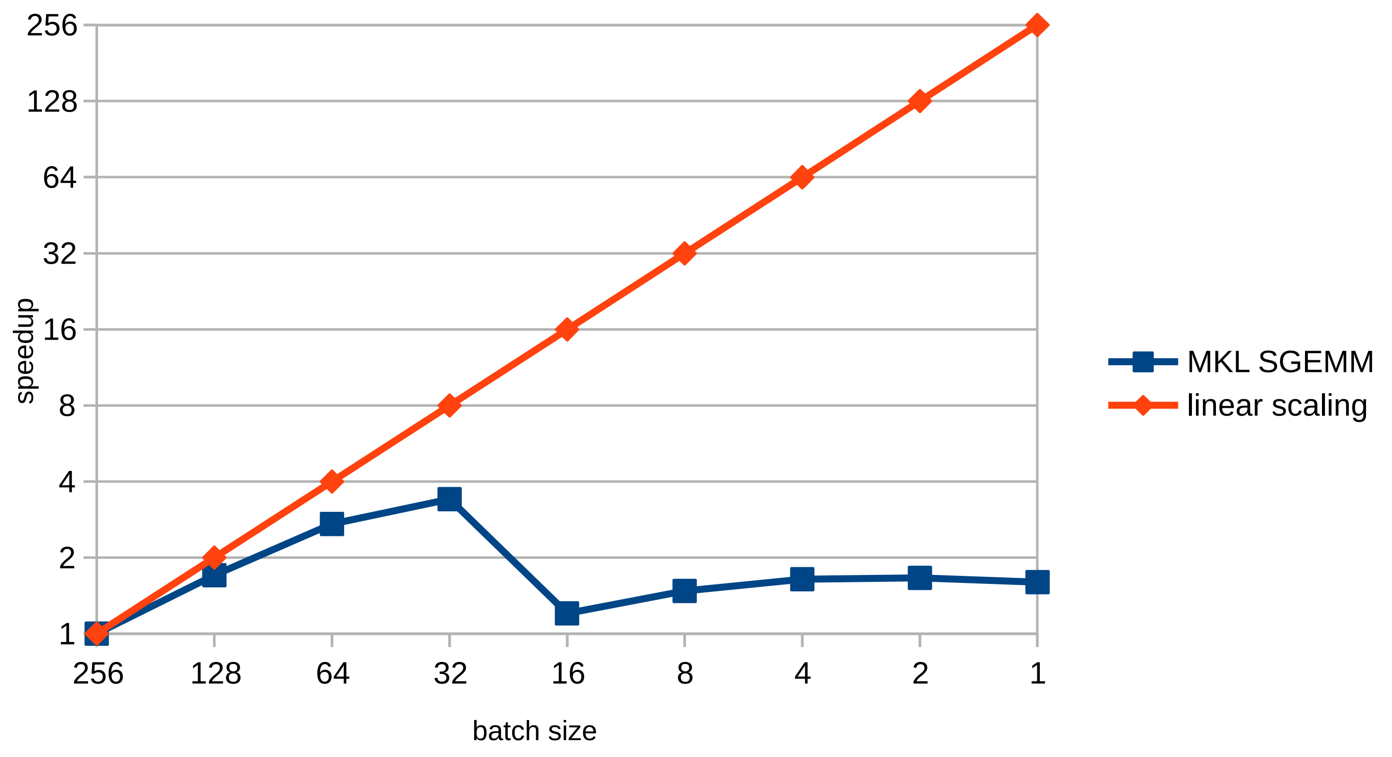
Definitions:

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.



Theoretical Limits

Parallelizing “Skinny” Matrix Multiplication

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.

Experimental Evaluation

Increasing the Batch Size

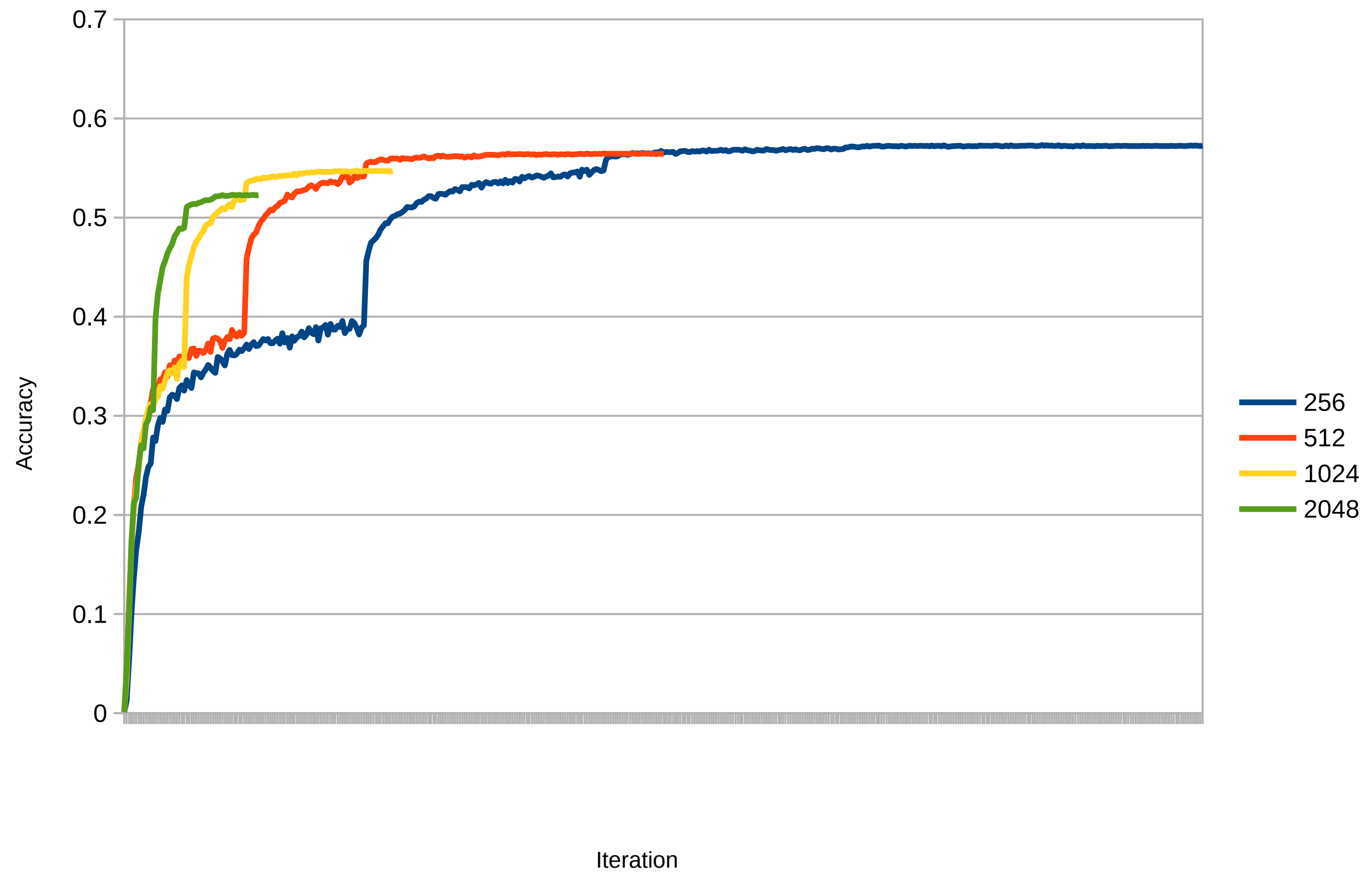
Solution proposed in literature:

Increase Batch size

But:

Linear speedup against original Problem only if we increase step size

This leads to loss of accuracy



Theoretical Limits

Increasing the Batch Size

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.** arXiv preprint arXiv:1609.04836, 2016.

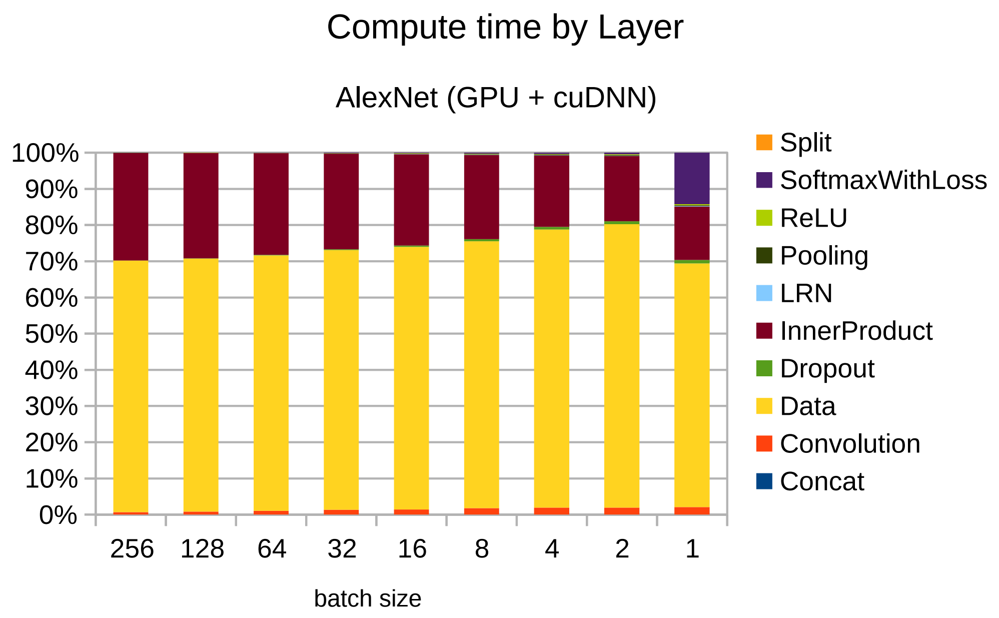
Theoretical analysis: larger Batch sizes will lead to worse generalization properties.

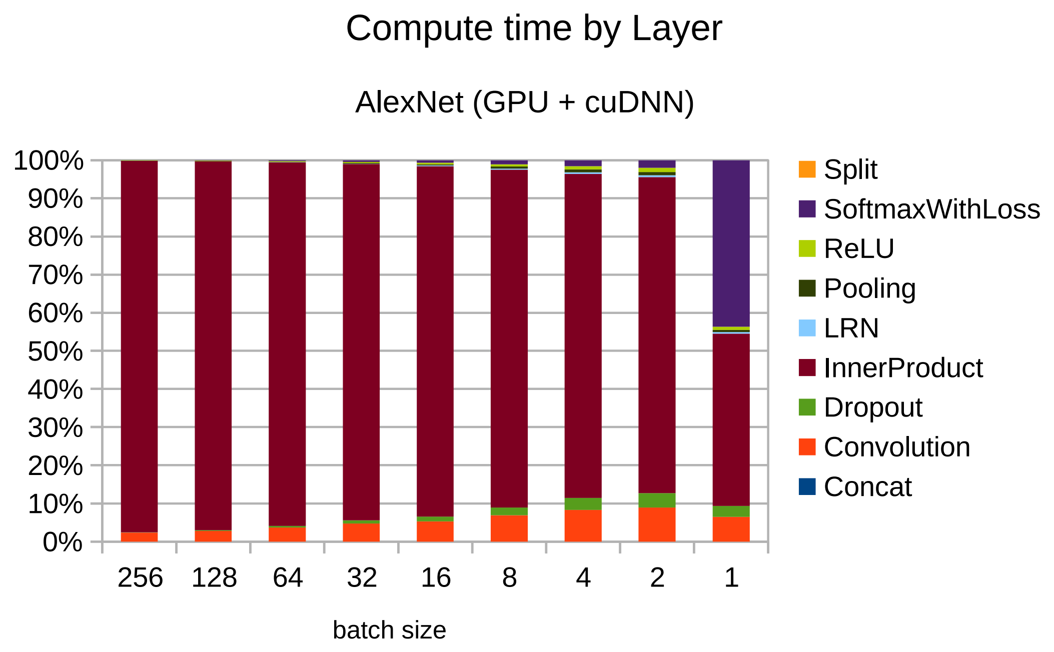
In a Nutshell: reduced noise causes overfitting to sharp saddle points.

Fraunhofer

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.

Fraunhofer

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**

Distributed I/O

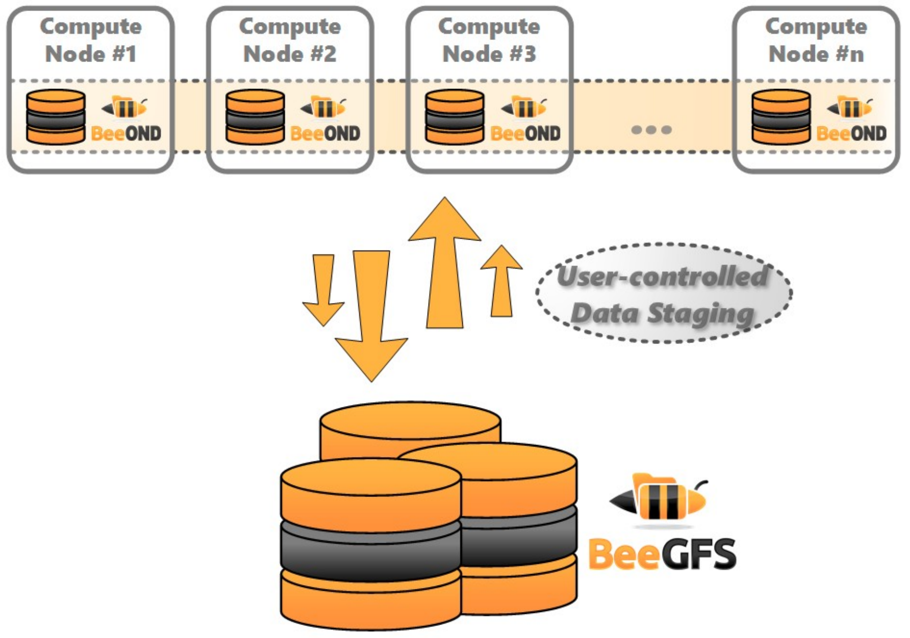
Possible Solution

BeeGFS®

Visit us at Booth 1243

Current Project based on our **BeeGFS BeeOND**:

- Build temp file system over SSDs on compute nodes
  - Keep data close
  - Every compute node is a meta-data server
- Combine all files in one large binary with fixed offsets
  - BeeGFS can handle simultaneous access



Conclusions

The main problem with training DNNs via distributed SGDs is that the computation load per iteration is to low.

This problem will further increase with faster compute units (GPUs).

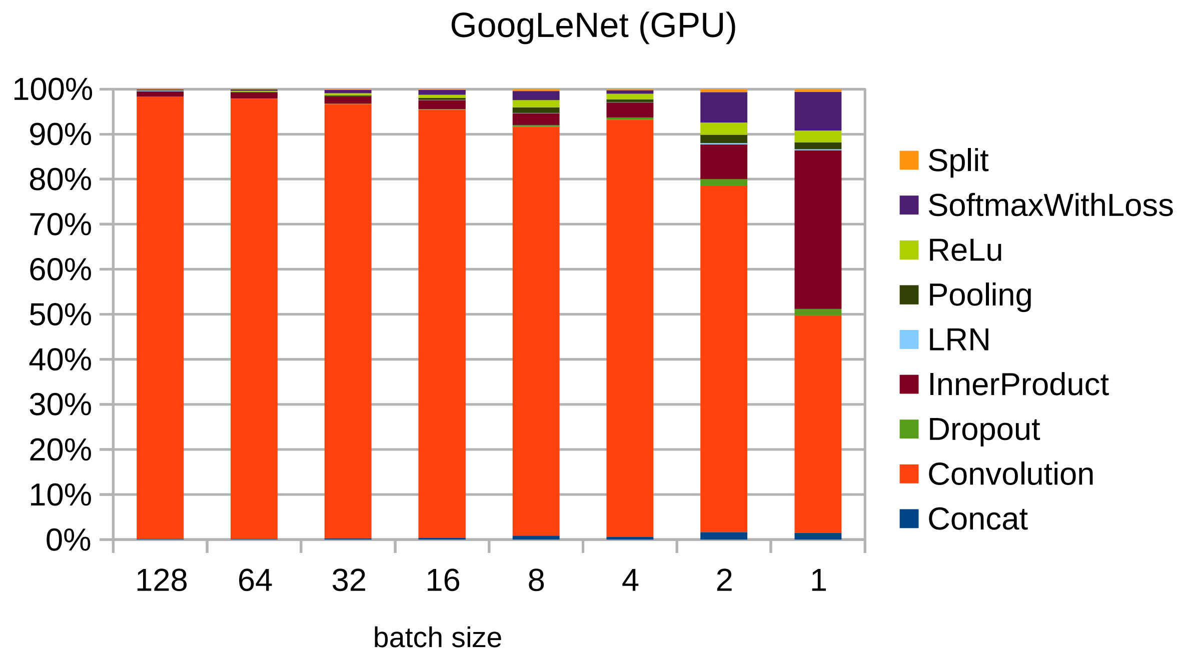
Possible solutions:

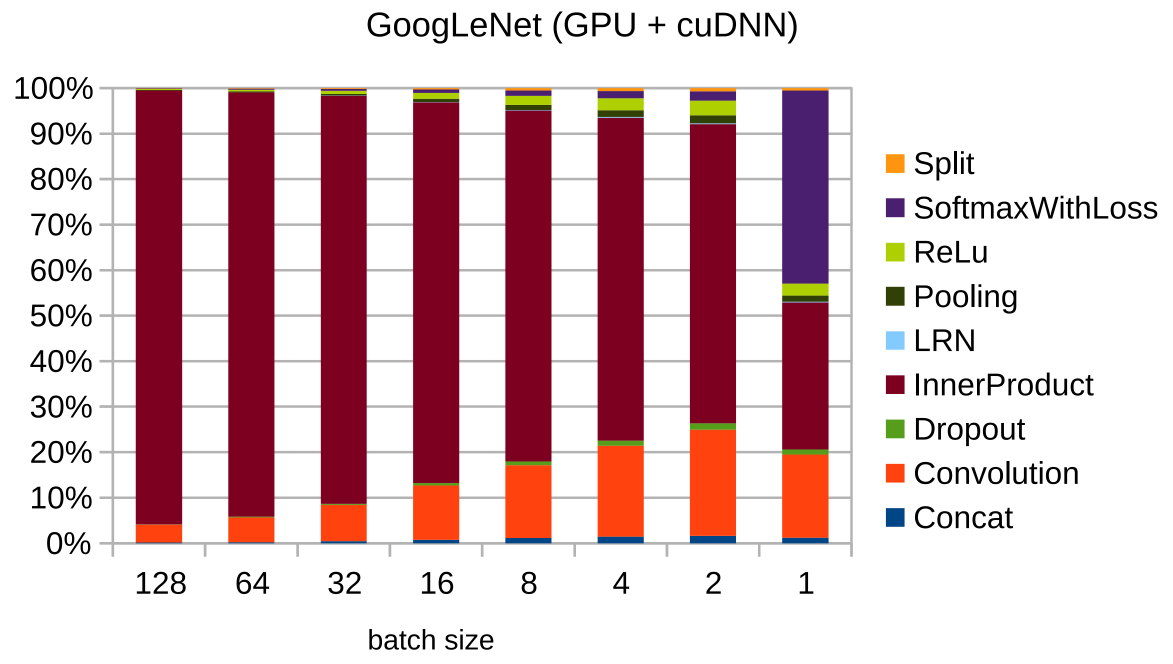
Change Network to handle the overfitting problem for large Batch sizes (?)

Alternative optimization methods (SGD is not the only way).

Appendix

Effect of optimized Convolution Functions (cuDNN + MKL17)

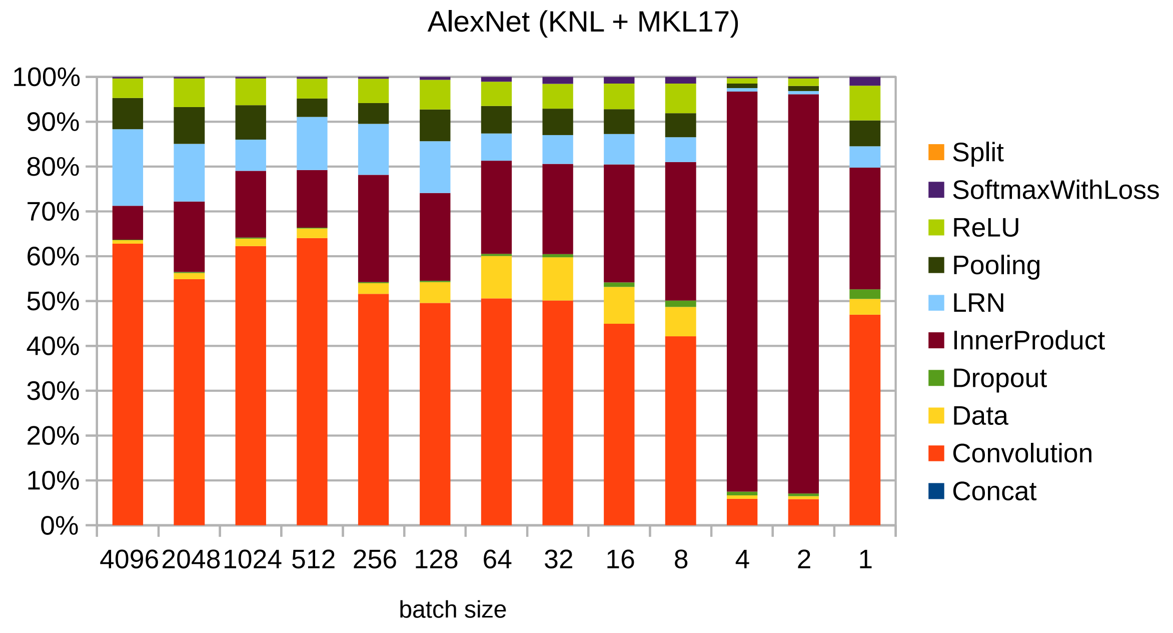


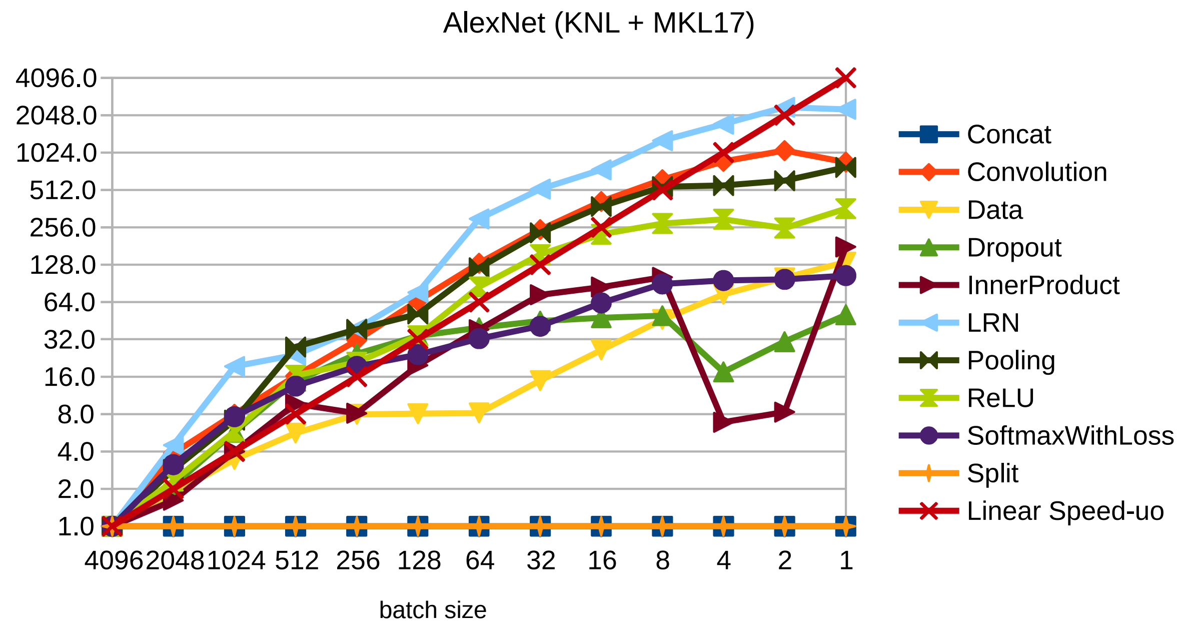


Fraunhofer

Appendix

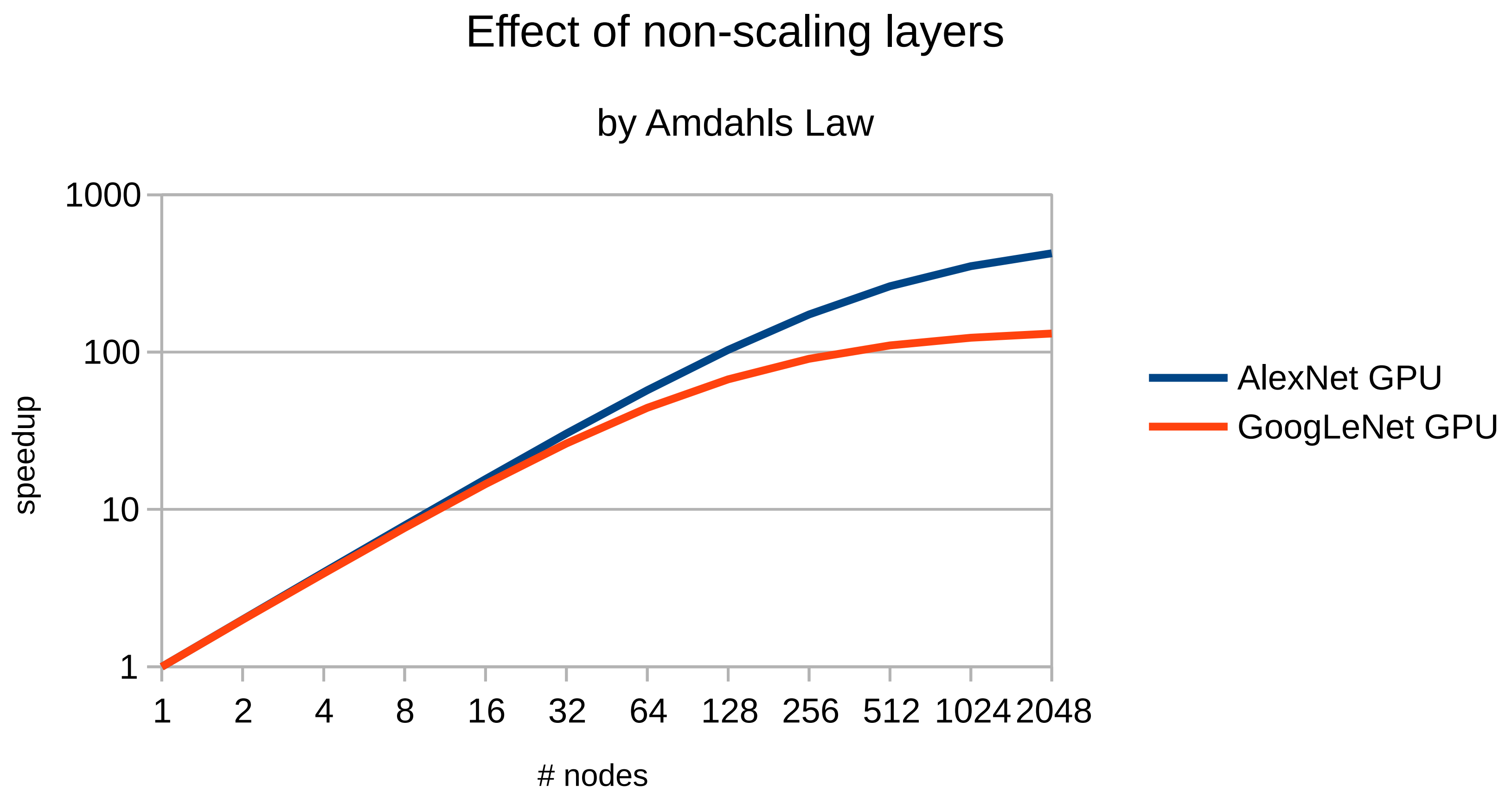
KNL





Appendix

Amdahls Law: Non-Scaling Layers


Effect of non-scaling layers
by Amdahls Law
speedup
# nodes
AlexNet GPU
GoogLeNet GPU