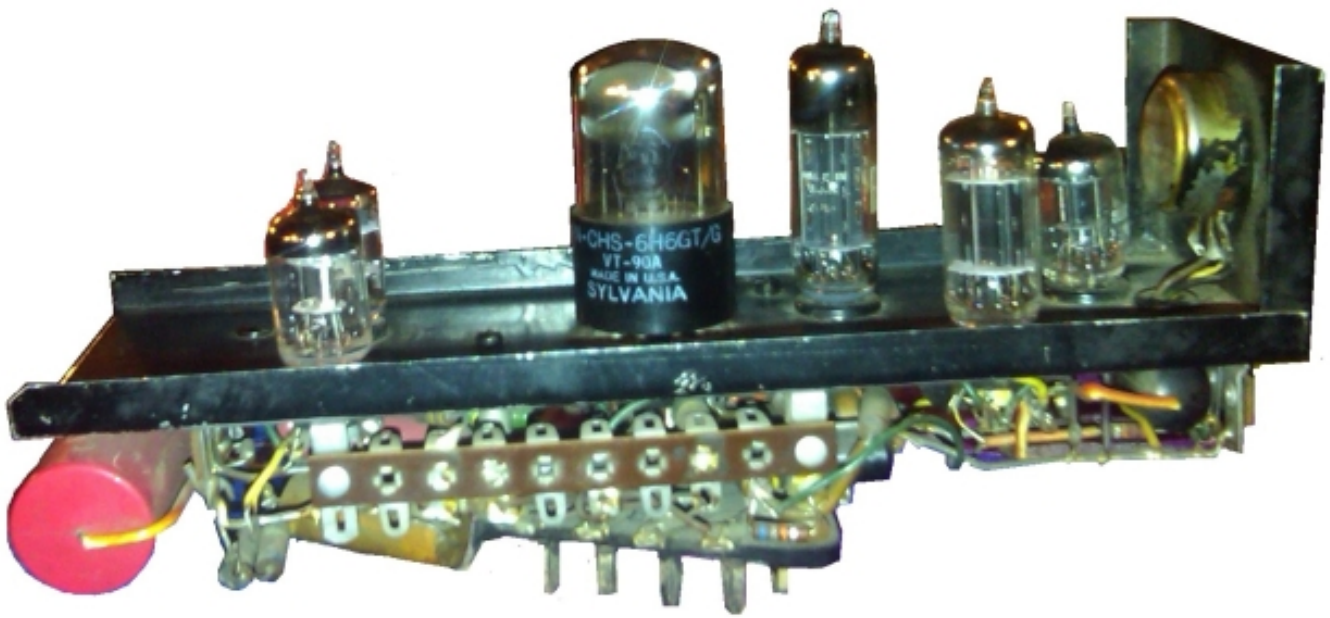
SCALING DEEP LEARNING

Bryan Catanzaro, 13 February 2017

SYSTEMS FOR AI



SNARC: 1951, 40 neurons
Stochastic Neural Analog Reinforcement Computer
(Marvin Minsky)

OVERVIEW

What is neural network training, computationally?

What limits scalability of neural network training?

- Today's frameworks and training approaches are limited

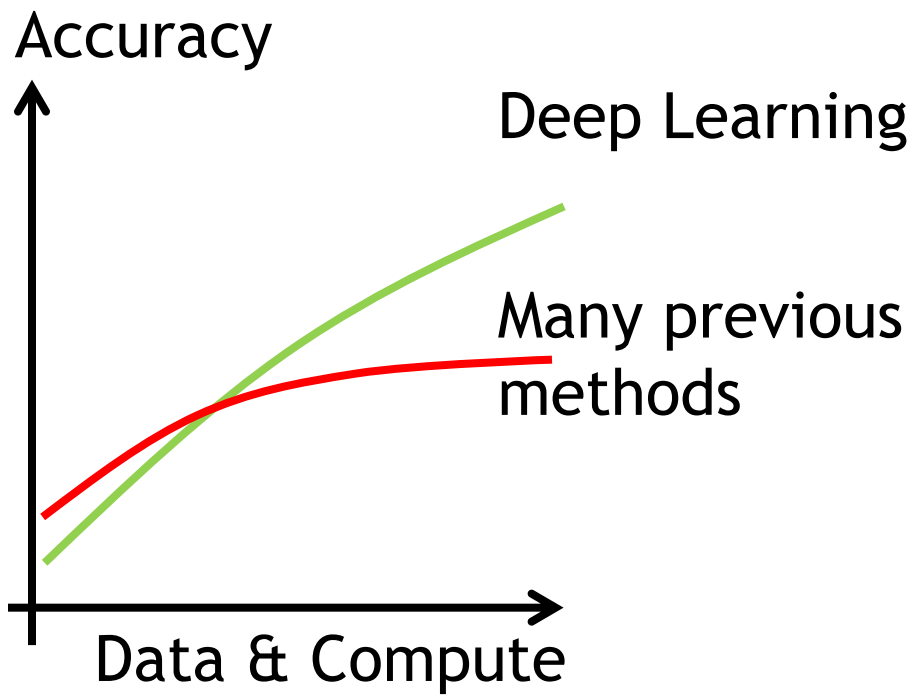
What can we do to overcome those limits?

- Example: Persistent RNN

What is the future of systems for training neural networks?

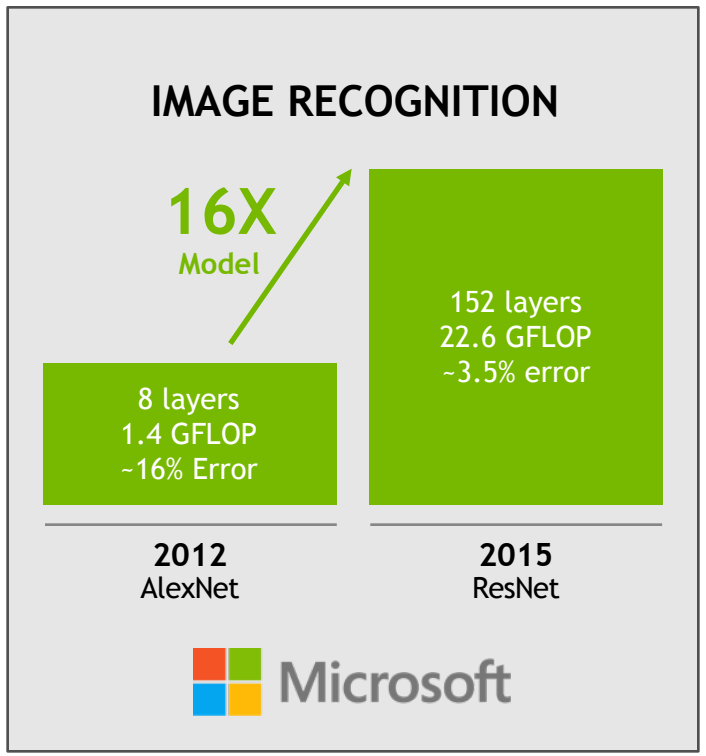
WHY NEURAL NETWORKS?

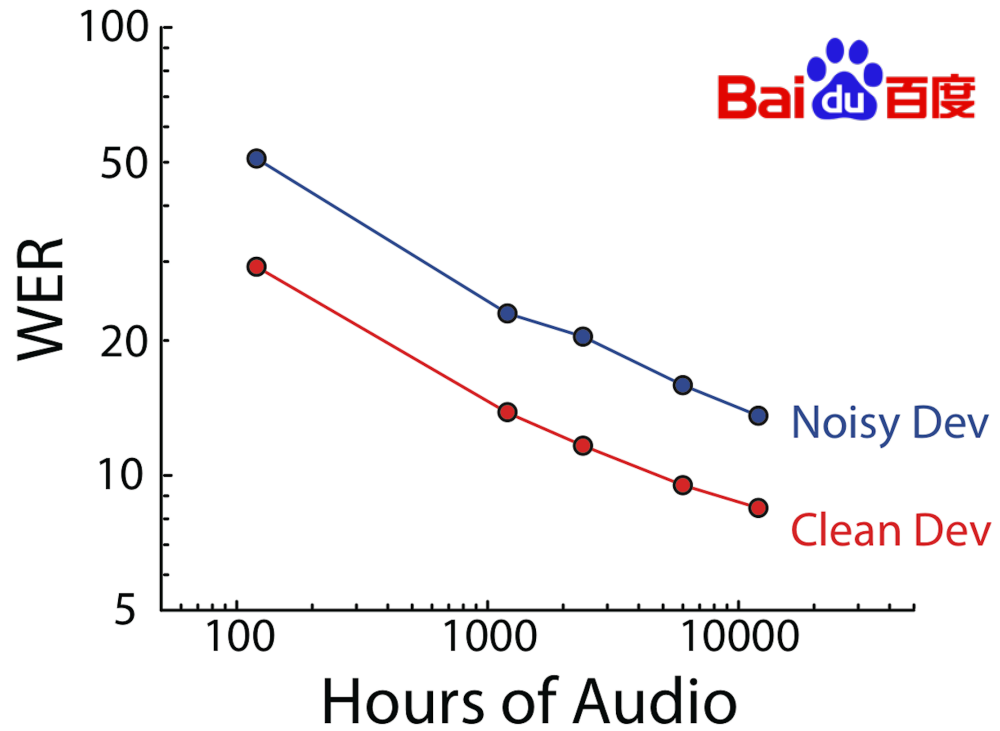
1. Neural networks benefit from large datasets
2. They're relatively simple, so frameworks & libraries help
3. They fit modern computing hardware



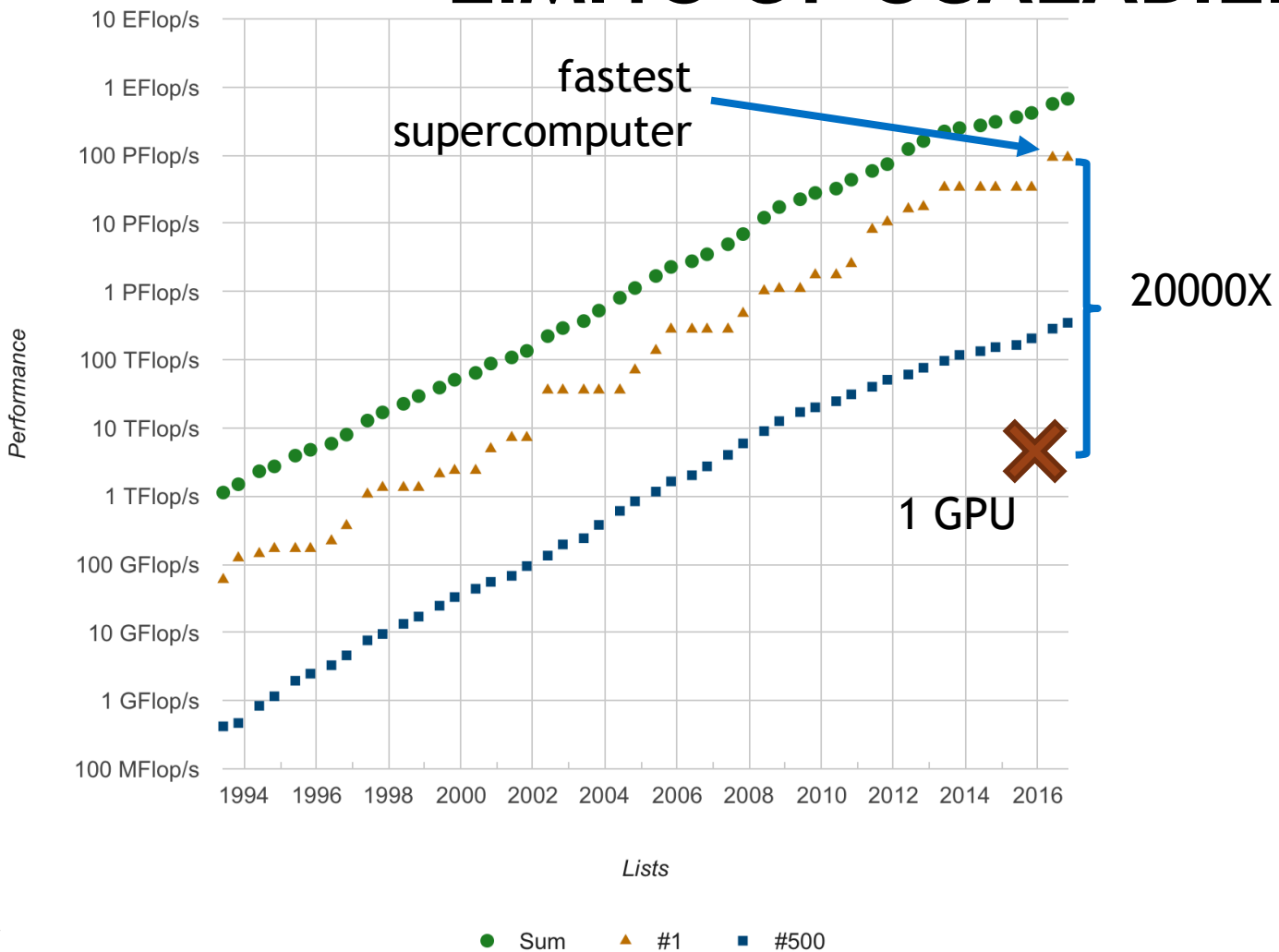
SCALE MATTERS

More data, more compute: More AI





LIMITS OF SCALABILITY



AI: most important problem

How can we use our best computers for it?

Current best practices use ~128 GPUs

Research problem: how can we use 20000?

DEEP NEURAL NETWORKS

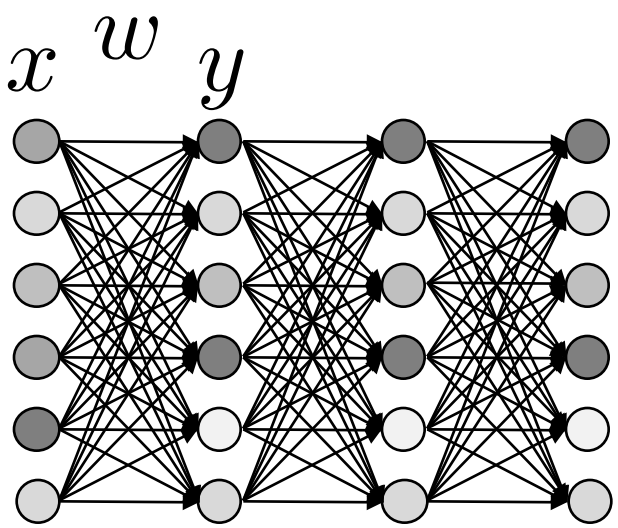
Simple function approximators

$$y_j = f\left(\sum_i w_{ij} x_i\right)$$

One layer

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

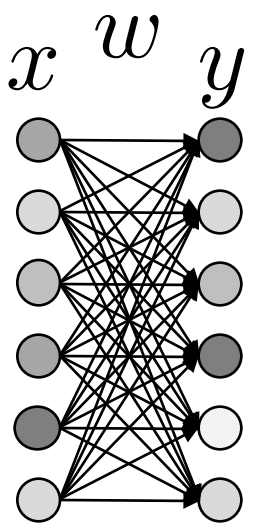
nonlinearity



Deep Neural Network

TRAINING NEURAL NETWORKS

$$y_j = f\left(\sum_i w_{ij} x_i\right)$$



Computation dominated by dot products

Multiple inputs, multiple outputs, batch means it is compute bound

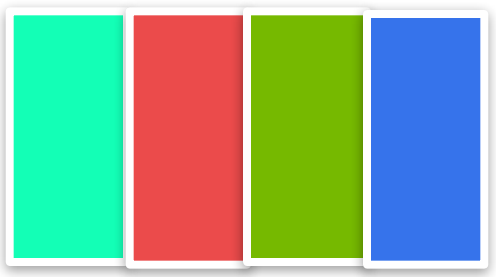
Stochastic Gradient Descent used to train models

Train 1 model: Tens of Exaflops

PARALLEL NEURAL NETWORK TRAINING

Two main strategies

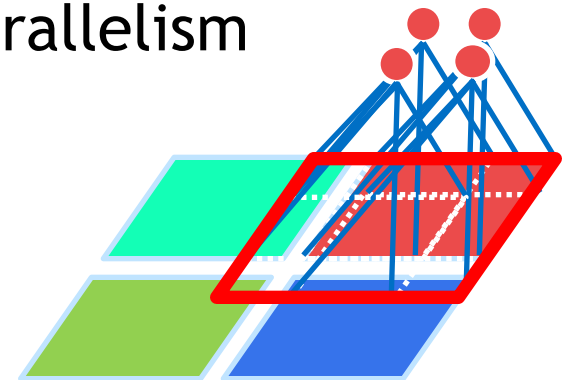
- Data parallelism



Activations (minibatch)

- Transfers gradients
- Multiple models

- Model Parallelism



Neurons

- Transfers partial activations
- Distributed model

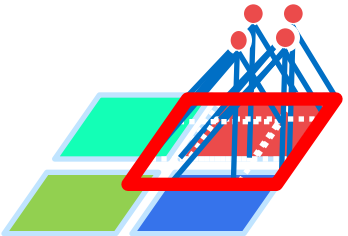
PARALLEL NEURAL NETWORK TRAINING

Strengths and weaknesses

- Data parallelism

- Strengths:
  - Software simplicity
  - Large, decoupled transfers
  - ”Relaxed” versions
- Weaknesses:
  - Limited by optimization algorithm and hardware

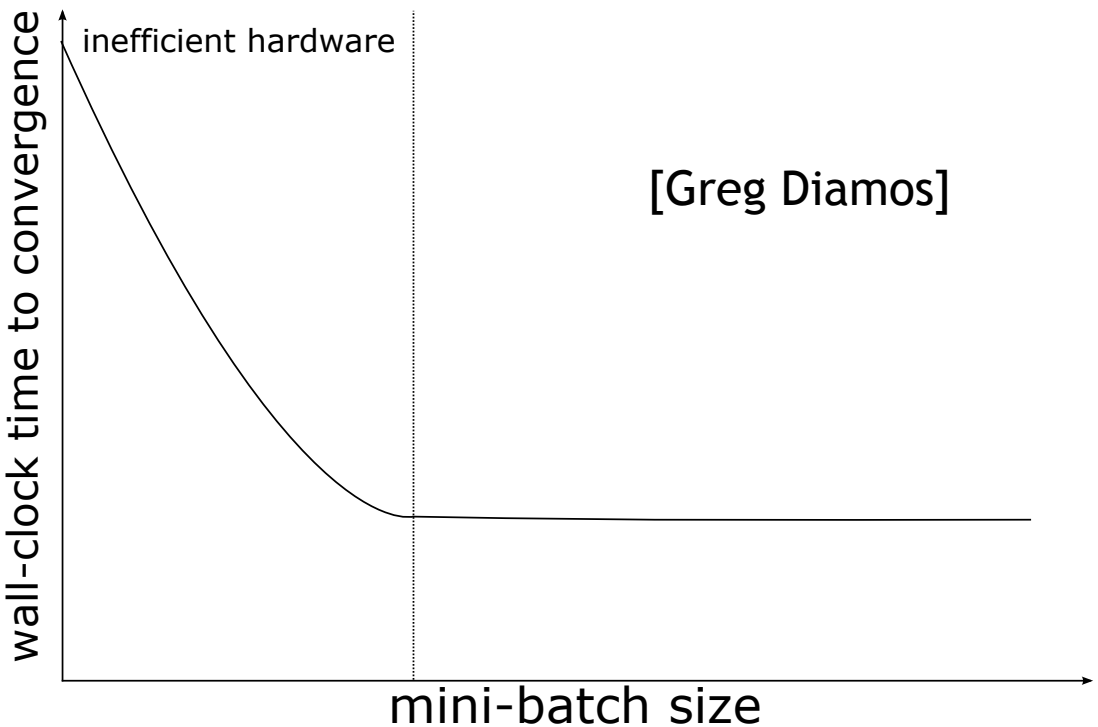
- Model parallelism



- Strengths:
  - Orthogonal to optimization
- Weaknesses:
  - Difficult to incorporate in a framework
  - Small, synchronous transfers
  - Limited by model size & structure

LIMITS OF DATA PARALLELISM

Processors become inefficient as minibatch decreases



[Greg Diamos]

Primary reason: less parameter reuse makes computation bandwidth bound

Secondary reason: SIMD parallelism underutilized

ROOFLINE MODEL

Given:

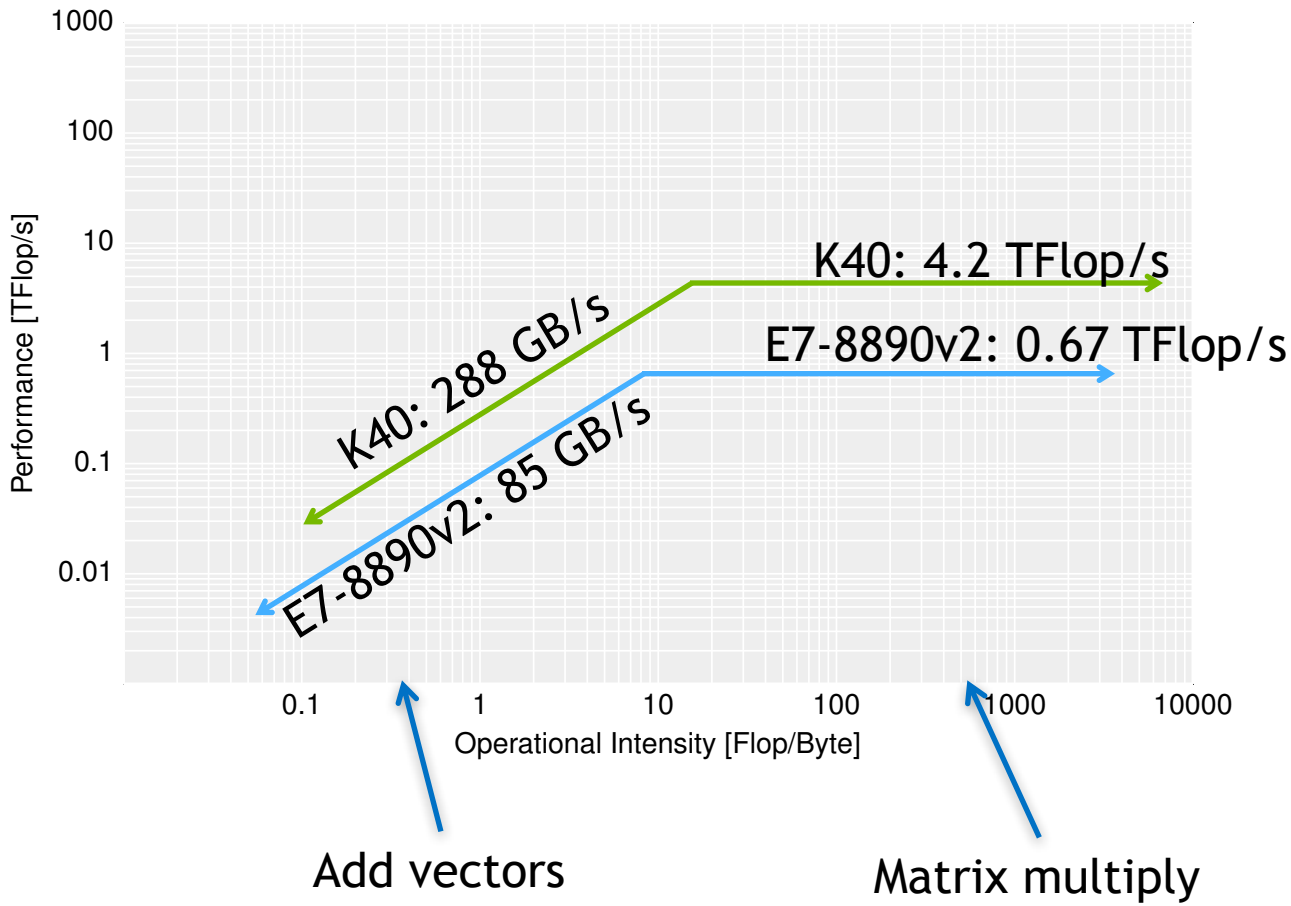
Processor BW

Processor Flop/s

You can find the Speed of Light

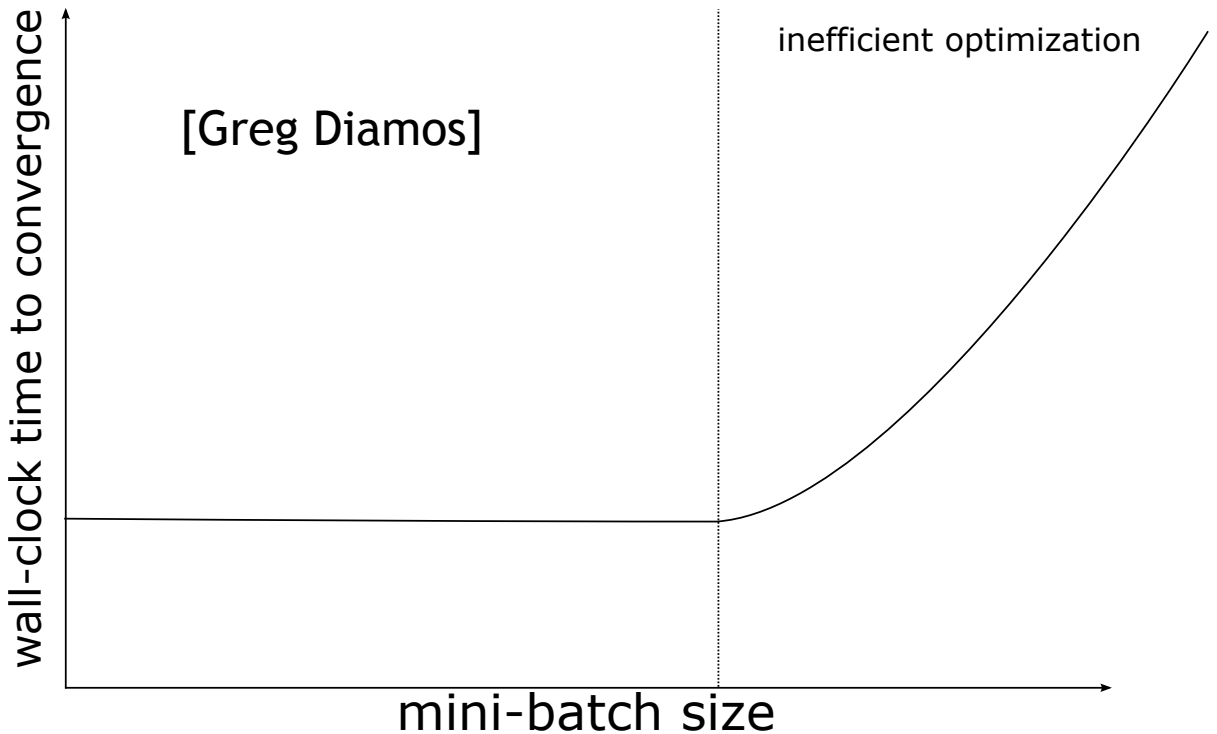
Approach the roofline

Such simple bounds are a powerful tool



LIMITS OF DATA PARALLELISM

Optimization algorithms fail as minibatch increases

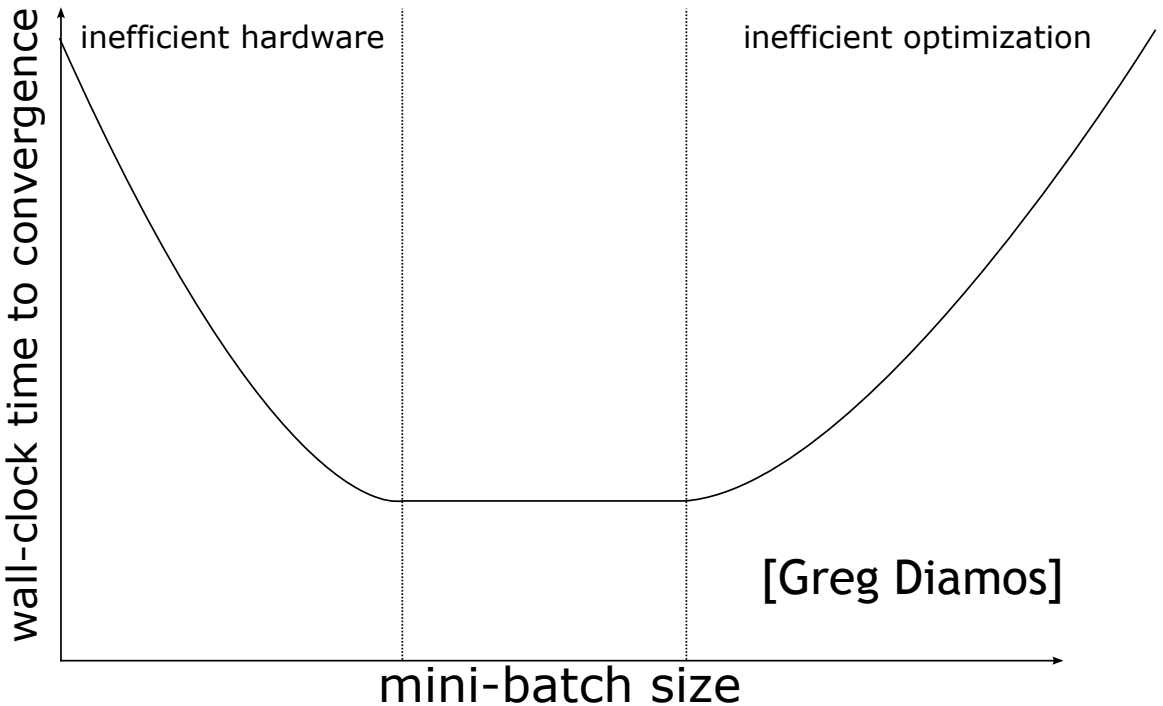


Progress towards objective per SGD step not linear as a function of computational work

Even worse: Generalization and accuracy empirically suffer as minibatch becomes too large

LIMITS OF DATA PARALLELISM

The elusive optimum



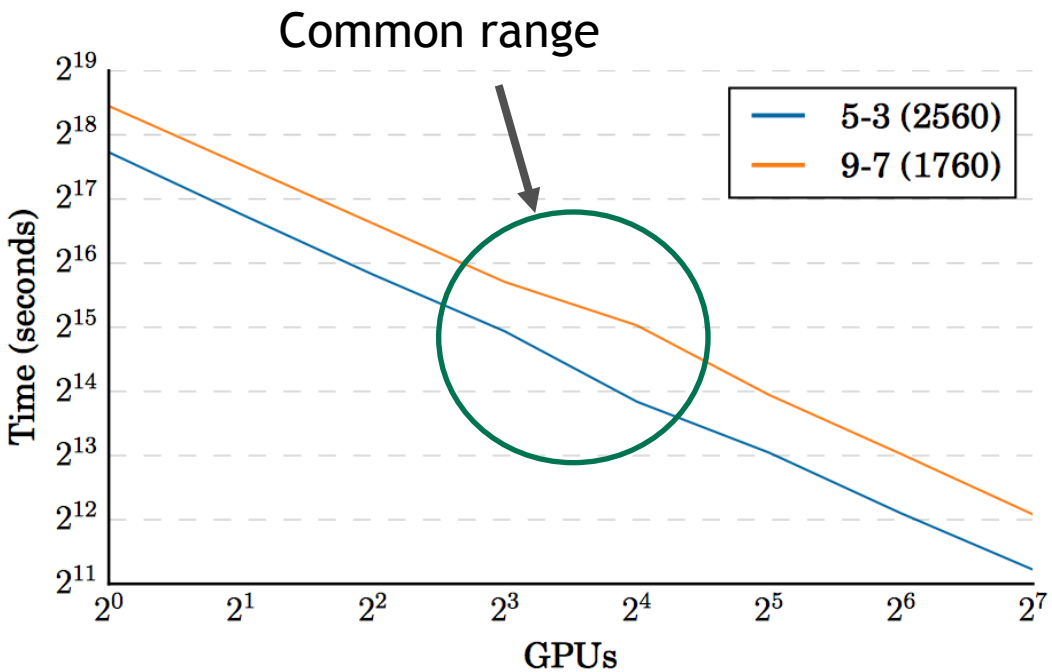
Some amount of data parallelism is optimum

This amount depends to change based on:

- Model
- Dataset
- Optimization algorithm

Generally we run at 512-2048 samples

LIMITS OF DATA PARALLELISM



Time to train one epoch
[Deep Speech 2]

Synchronous data parallelism:
- typically 8-32 GPUs
- 512-2048 samples
- 64 samples/GPU

This will vary based on:

- model
- dataset
- optimization algorithm
- processor type
- …

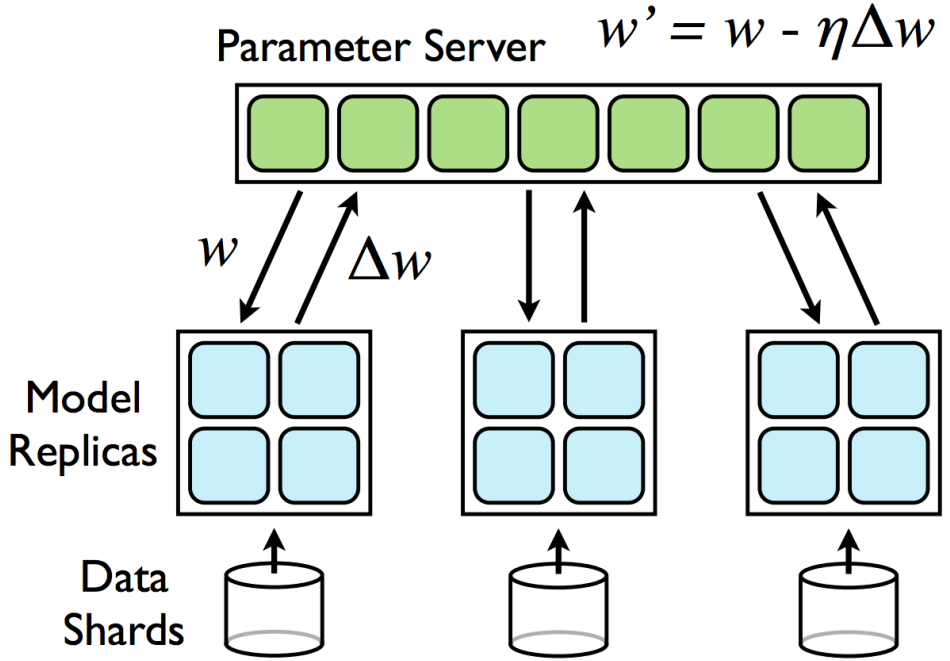
SYNCHRONOUS VERSUS ASYNCHRONOUS SGD

Synchronous SGD:

Gather all gradients together from all processors, then take a step

Asynchronous SGD:

Let model replicas learn in a decoupled fashion, exchange gradients more loosely



Asynchronous SGD with Parameter Server

ASYNCHRONOUS SGD TYPES

Parameter Server [Dean, NIPS 2012]

Asynchronously update parameters

Elastic Averaging SGD [Zhang, NIPS 2015]

Communicate periodically

Hogwild [Recht, NIPS 2011]

Don’t synchronize, just overwrite parameters opportunistically.

1-bit SGD [Seide, Interspeech 2014]

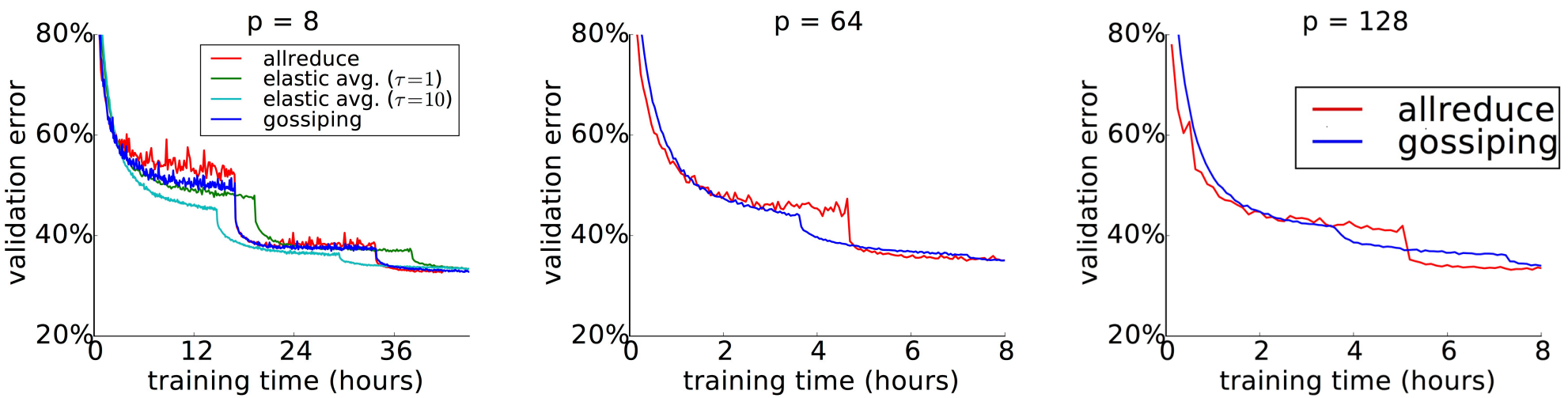
Send quantized gradients around, keep residuals

WHY I LIKE SYNCHRONOUS SGD

Simple, Reproducible, Good Convergence

Human aspect of debugging the training process requires reproducibility

Synchronous methods get the best accuracy in my experience



[Keutzer, NIPS Workshop, 2016]

ALL REDUCE (COLLECTIVE)

Adding gradients from all processes

Big messages (from big gradients)

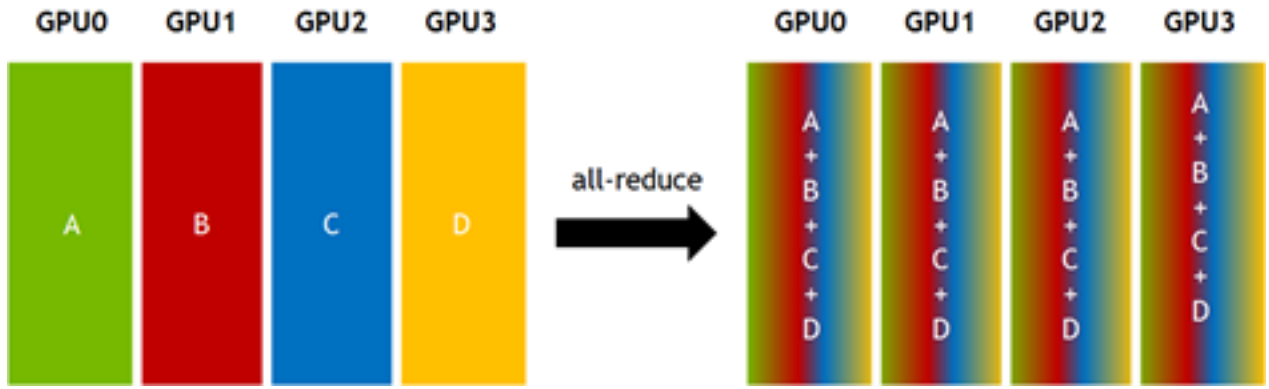
Ring algorithm

Works well on most interconnect topologies

Processes only interact with a right and left neighbor

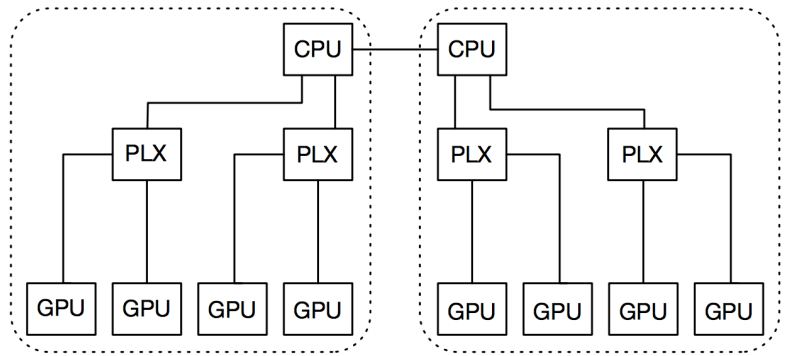
Is interconnect bandwidth bound

Other algorithms can be used as well



HARDWARE TOPOLOGY

Mapping processes onto the machine



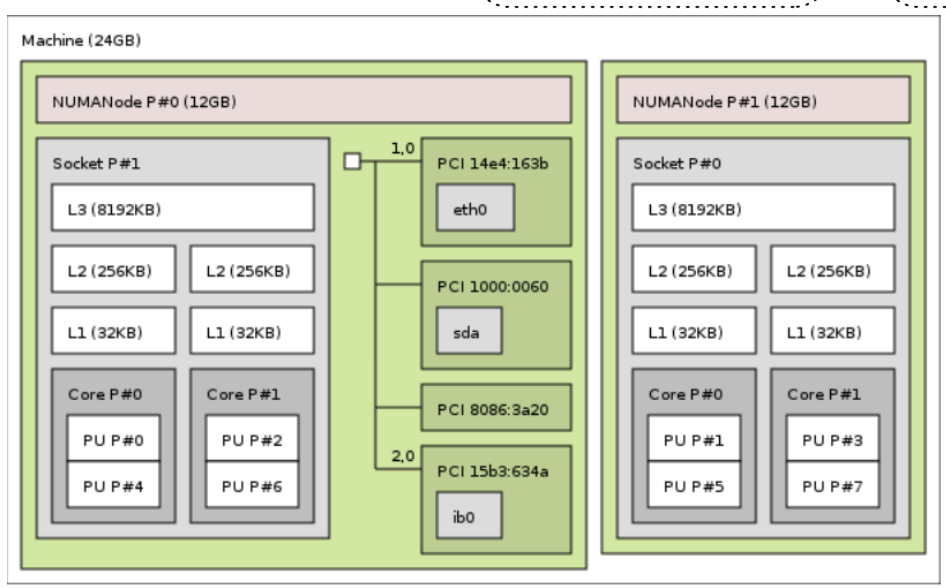
Interconnect & memory comes in hierarchies

To get best performance, you need to map processes to hierarchy

This doesn't happen magically

Defaults may be very bad

`hwloc` is good software for this



COMMUNICATION LIBRARIES

NCCL, MPI

NCCL: Optimized intra-node communication

Library with sophisticated topology aware collective algorithms

MPI: Library for inter-node communication

CUDA-aware MPI means you can run MPI programs using GPUs

Scalable, distributed code in a familiar environment for HPC

MODEL PARALLELISM

At the moment, still exotic

Orthogonal to data parallelism (algorithmically)

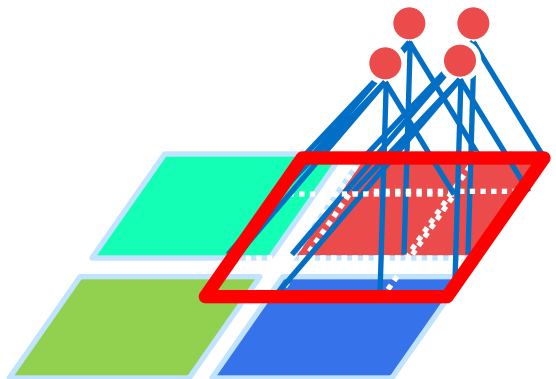
But more challenging to implement

- Current DL frameworks don't support it

Distributed tensor frameworks could help

- But likely to work for some models and not others

Scalability limited by model size and structure



Neurons

PERSISTENT RNNS

Make a bandwidth bound problem a compute bound problem

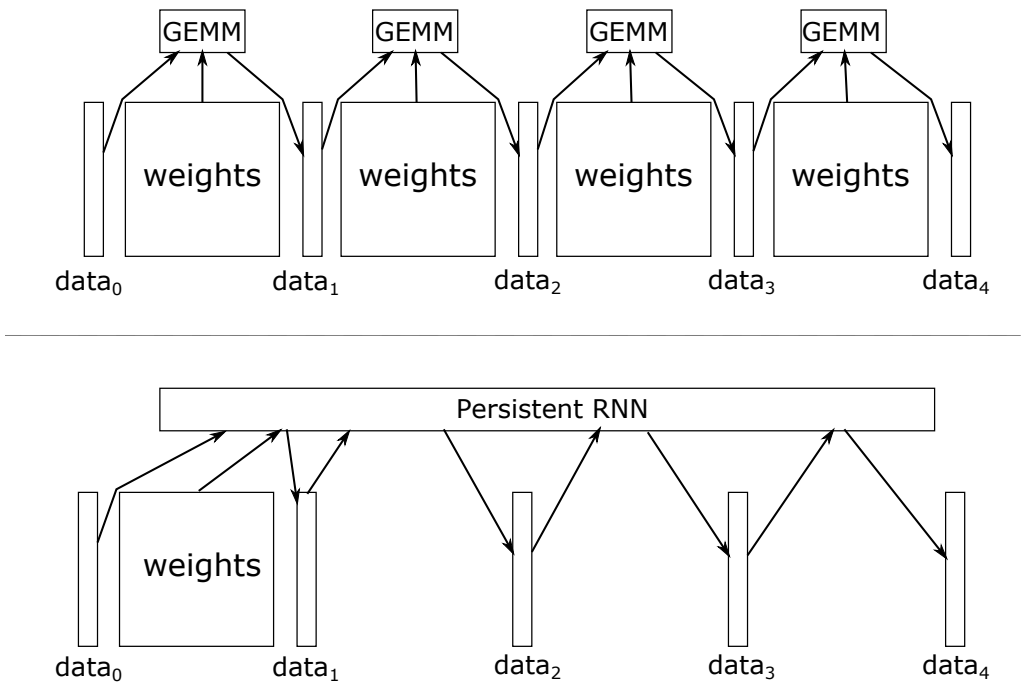
Traditional RNN implementations reload weights every timestep

This dramatically lowers the arithmetic intensity

Making us dependent on large batches

Limiting scale

Could we cache the weights on chip?

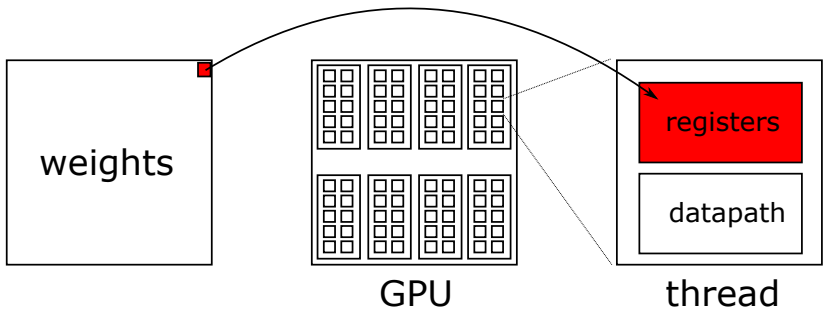


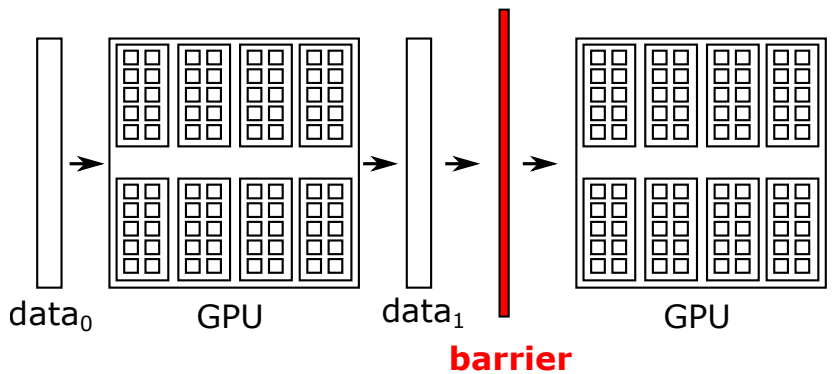
[Diamos, ICML 2016]

PERSISTENT RNNS

Implementation

[Diamos, ICML 2016]





Largest, highest bandwidth storage on GPU is register file

Up to 6 MB on Maxwell Titan X

Limited RNN size:
1152 neurons in 6 MB

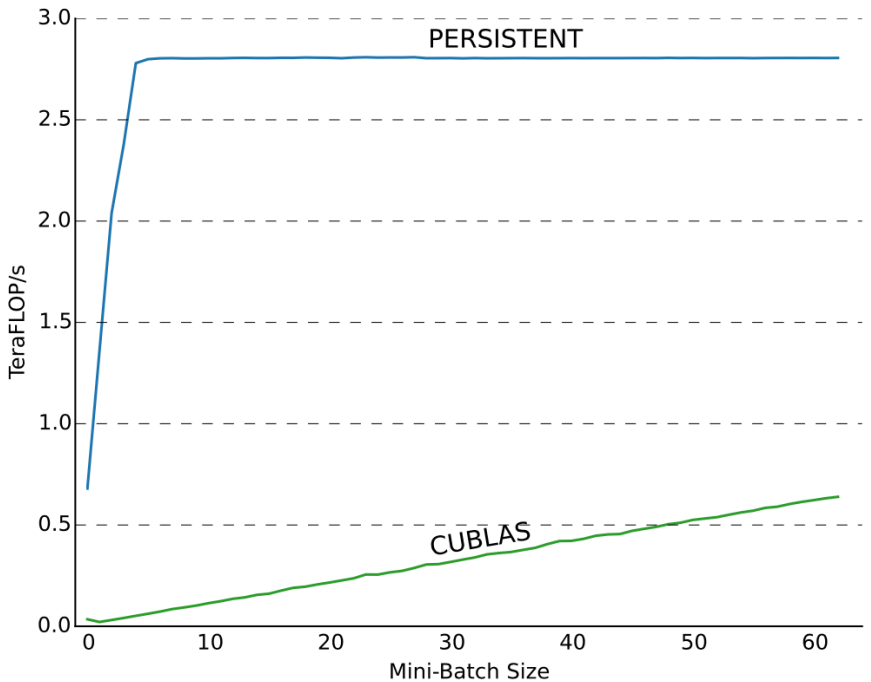
Implementing the RNN requires a global barrier

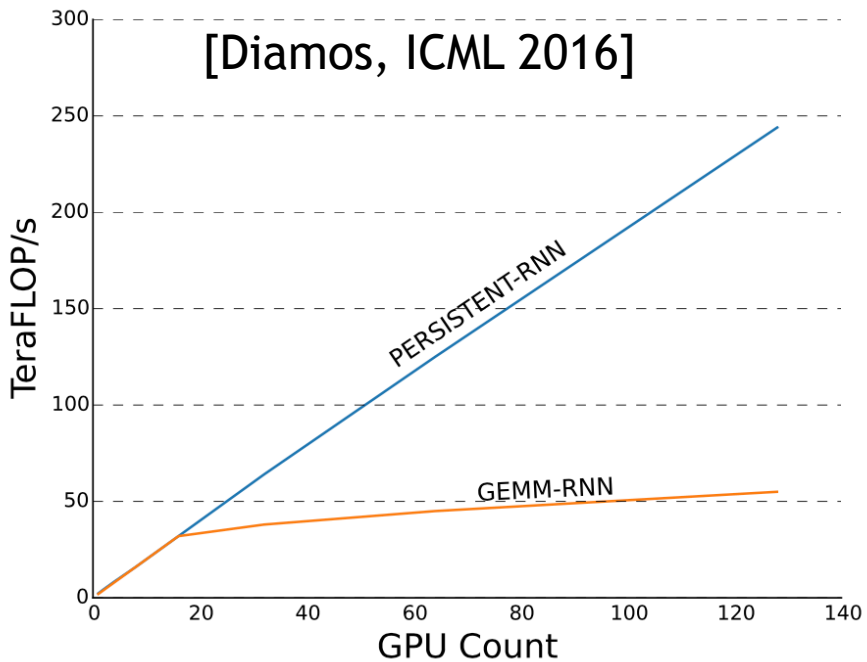
This is somewhat difficult to achieve on GPUs, but it is possible

http://github.com/baidu-research/persistent-rnn

PERSISTENT RNNS

Awesome for limited size Vanilla RNNs





Only requires a MB of 4 per processor to saturate

Now in CUDNN

Strong scaling (fixed MB of 512)

DISTRIBUTED PERSISTENT RNNS

Model Parallelism FTW?

To overcome capacity limitations, distribute the model!

This requires a global barrier between processors, not just on each processor

Possible on GPUs with NVLINK

Implemented with atomic operations and memory fences

This could overcome size limitations for RNNs (LSTMs, GRUs, etc.)

Could compose with data parallelism to scale RNN training to hundreds of GPUs

LIBRARIES

Optimized Kernels

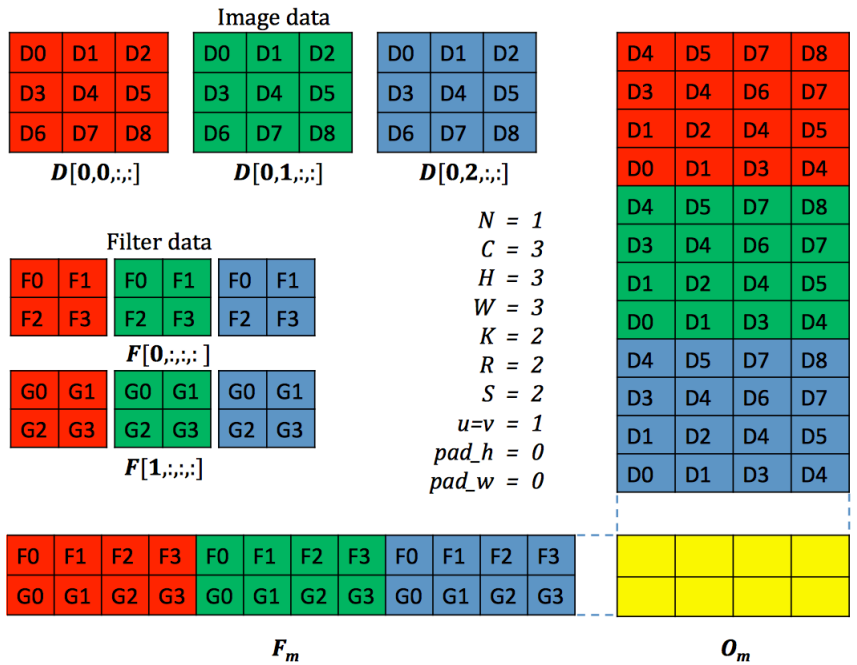
CUBLAS: Linear algebra

CUDNN: Neural network kernels

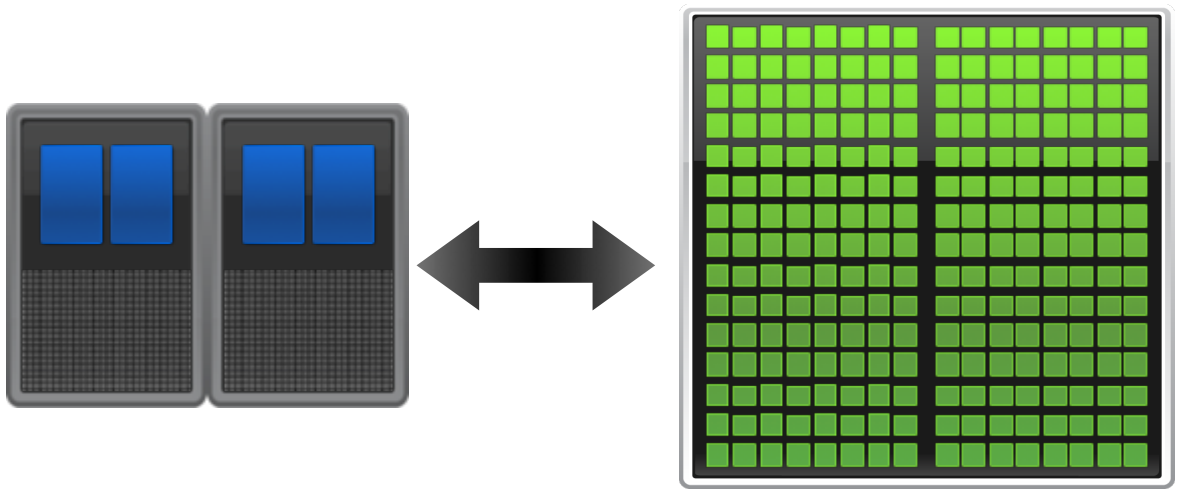
Convolutions (direct, Winograd, FFT)

Can achieve > Speed of Light!

Recurrent Neural Networks



HETEROGENEOUS COMPUTING



Latency Optimized CPU

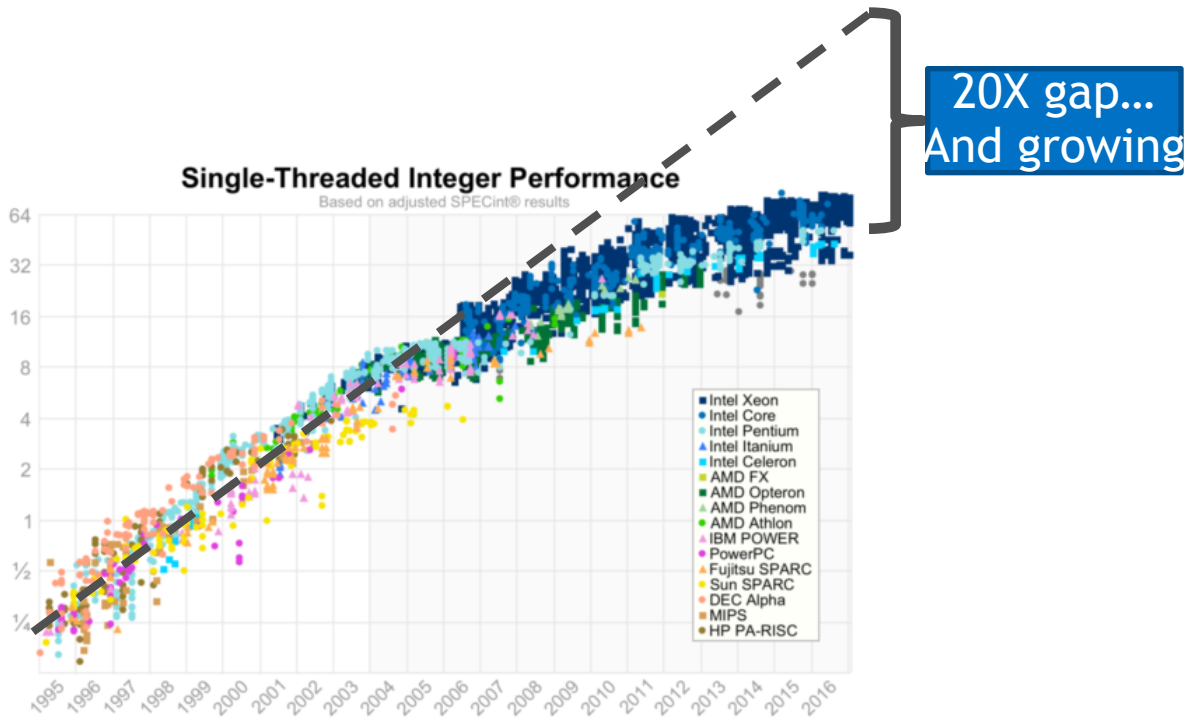
Fast Serial Processing

Throughput Optimized GPU

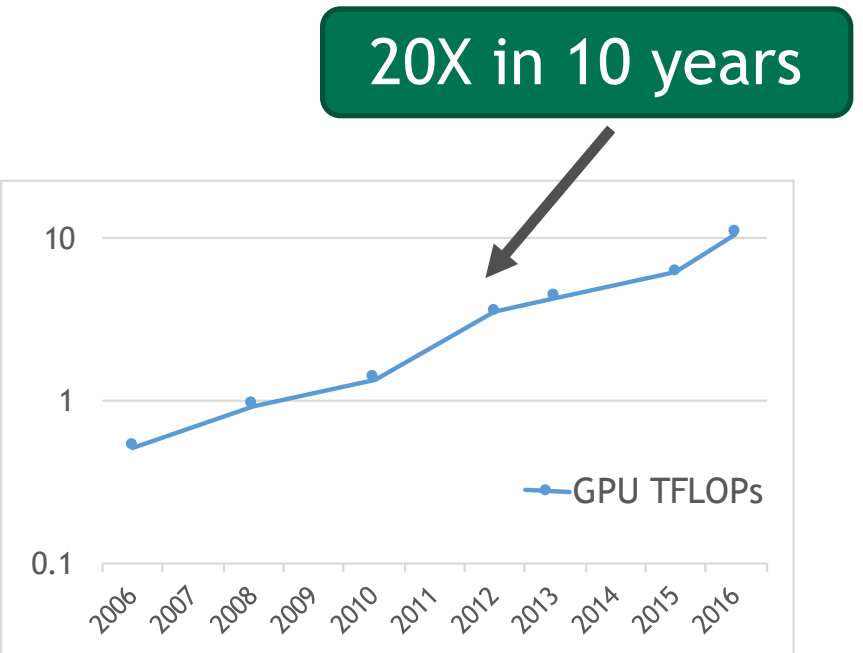
Scalable Parallel Processing

LAWS OF PHYSICS

Successful AI will use Throughput Computing



Latency Optimized Performance



Throughput Optimized Performance

ACCELERATED COMPUTING

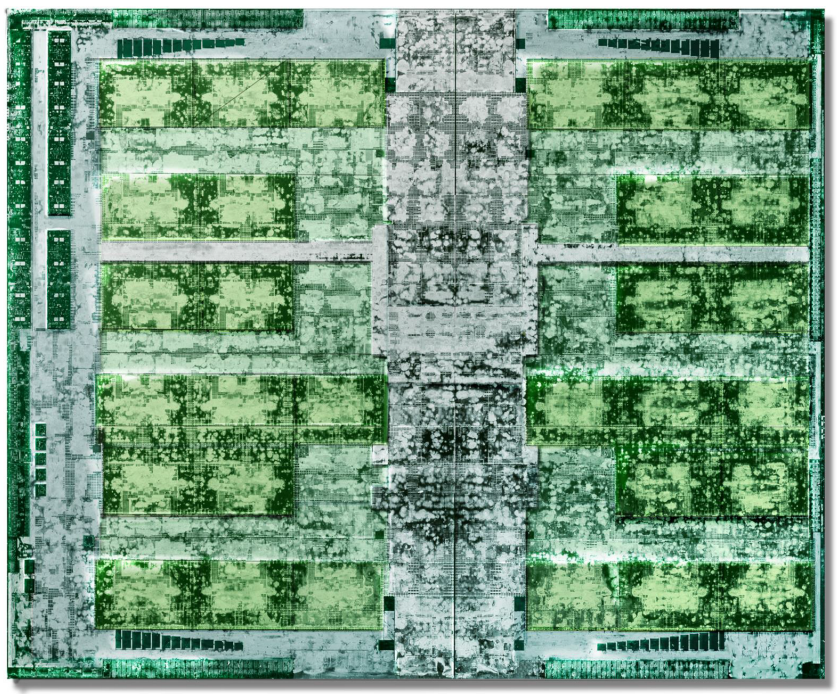
GPUs have always contained specialized hardware

Find economically important problem
that needs compute

Make hardware for it to take it to speed of light

GPUs will have more specialization for AI

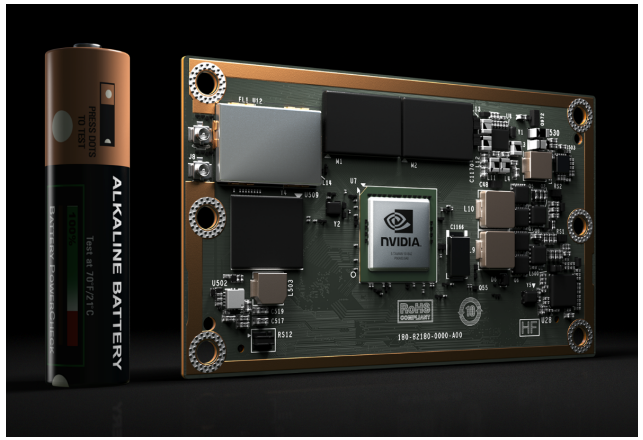
We see that happening with arithmetic for DL



GP100 Die

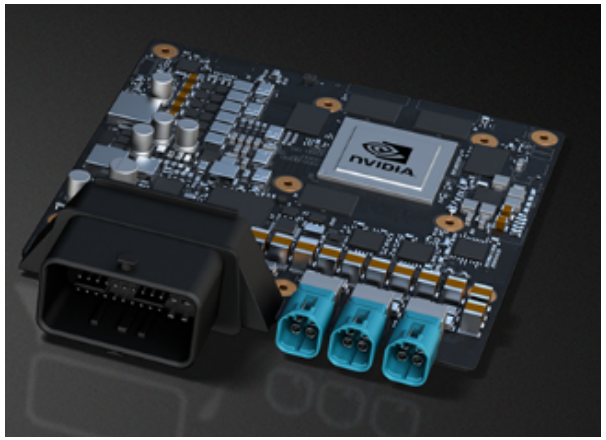
HARDWARE PLATFORMS

Scaling from 1 – 3000 Watts: Power limited in all cases



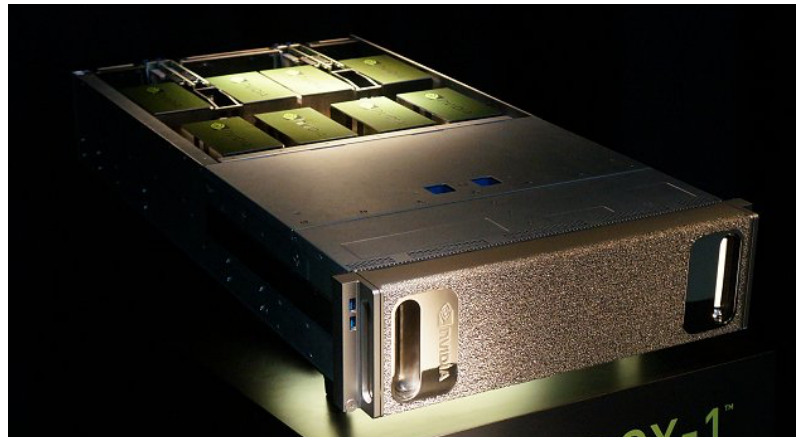
Jetson

Embedded



Drive PX

Automotive



DGX

Data Center

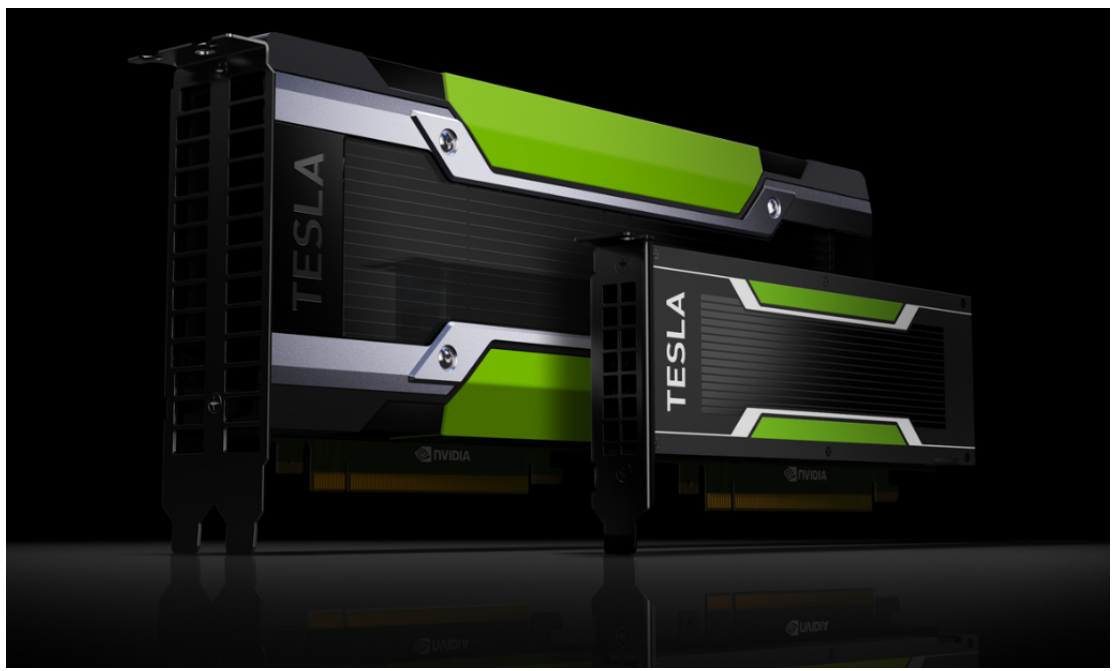
ARITHMETIC

Mixed precision for training

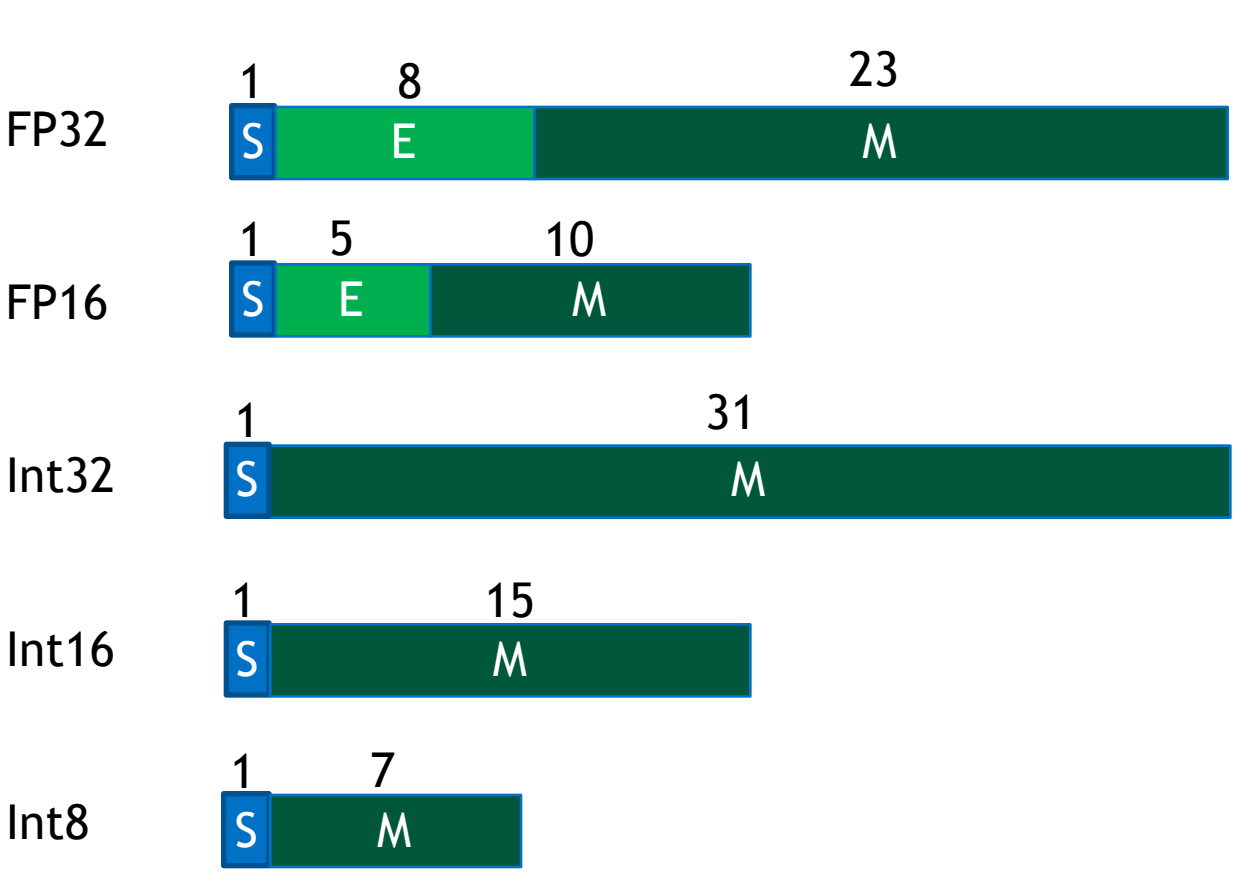
FP32 + FP16

Lower precision integer for inference

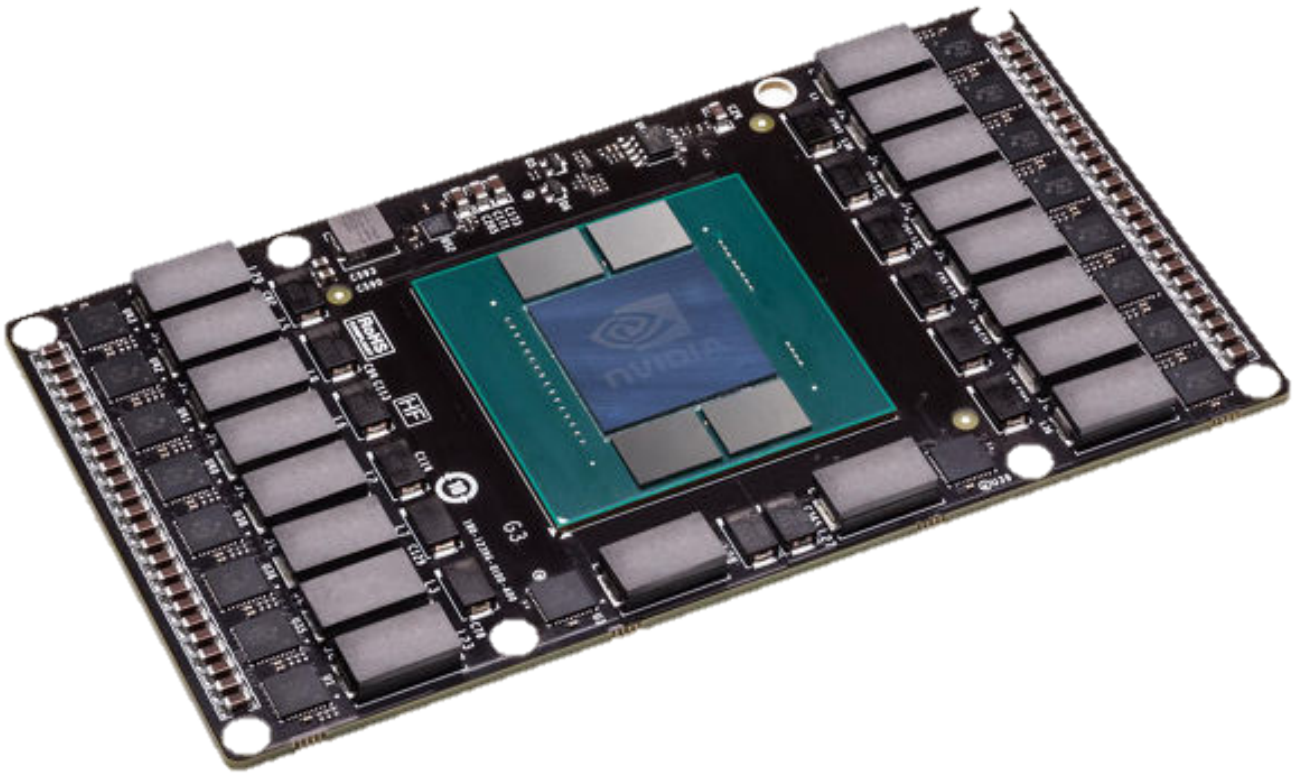
Int8



NUMBER REPRESENTATION



PASCAL GP100



10 TeraFLOPS FP32

20 TeraFLOPS FP16

16GB HBM – 750GB/s

300W TDP

67GFLOPS/W (FP16)

16nm process

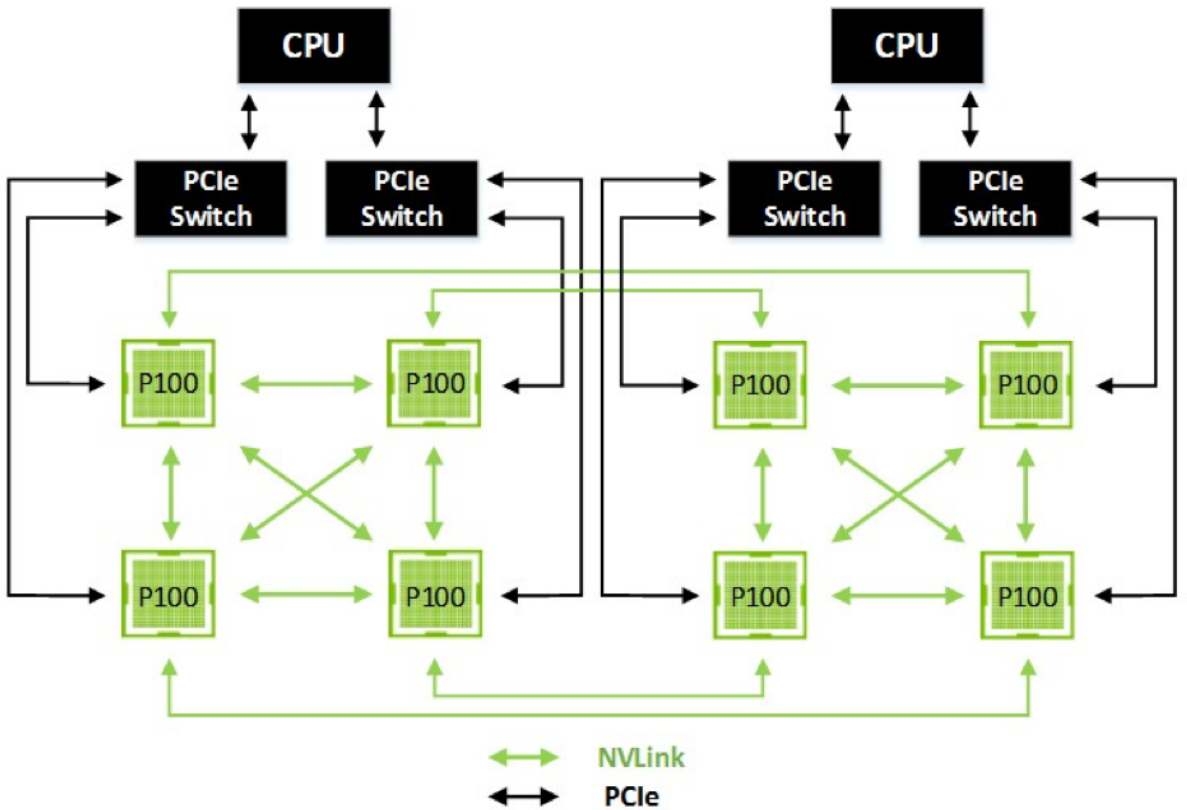
160GB/s NV Link

COMMUNICATION LINKS

Enabling Scalability

NVLINK:

- 5-12X PCIe bandwidth
- Supports remote atomics
- Supports high-bandwidth access to CPU memory (IBM Power)



DGX-1

SCALING DEEP LEARNING

Systems for AI have renewed importance

Scaling enables new experiments

Bigger models

Bigger datasets

Lots of work going on across the industry
to make the next generation of scaling possible

Bryan Catanzaro
@ctnzr


NVIDIA