

PRACTICAL DEEP LEARNING

한재근 과장 | jahan@nvidia.com

유현곤 부장 | hryu@nvidia.com / 양한별 과장 | hanbyuly@nvidia.com



AGENDA

Power of small filters

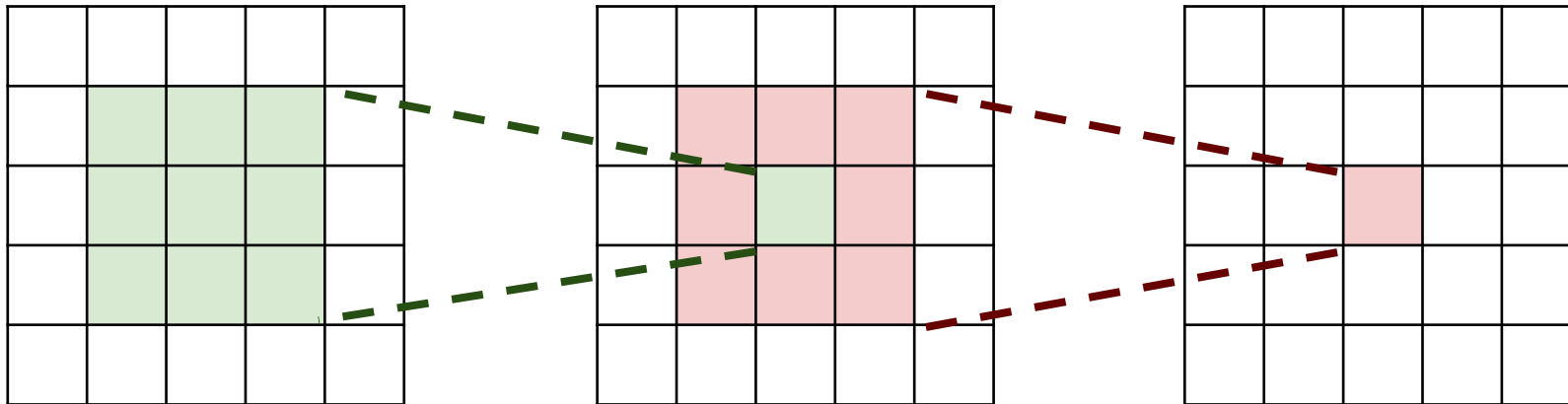
GPU Computing & Parallelism

Floating Point Precision

Power of Small Filters

Stacking Convolution Layers

Suppose we stack two 3x3 conv layers (stride 1)
Each neuron sees 3x3 region of previous activation map



Input

First Conv

Second Conv

Stacking one more layer

Three 3 x 3 conv gives similar
Representational power as a single
7 x 7 convolution

1x(7x7) convolution vs. 3x(3x3) convolution

one CONV with 7 x 7 filters

Number of weights:

$$= C \times (7 \times 7 \times C) = 49 C^2$$

Number of multiply-adds:

$$= (H \times W \times C) \times (7 \times 7 \times C)$$

$$= 49 HWC^2$$

More weight
More compute

three CONV with 3 x 3 filters

Number of weights:

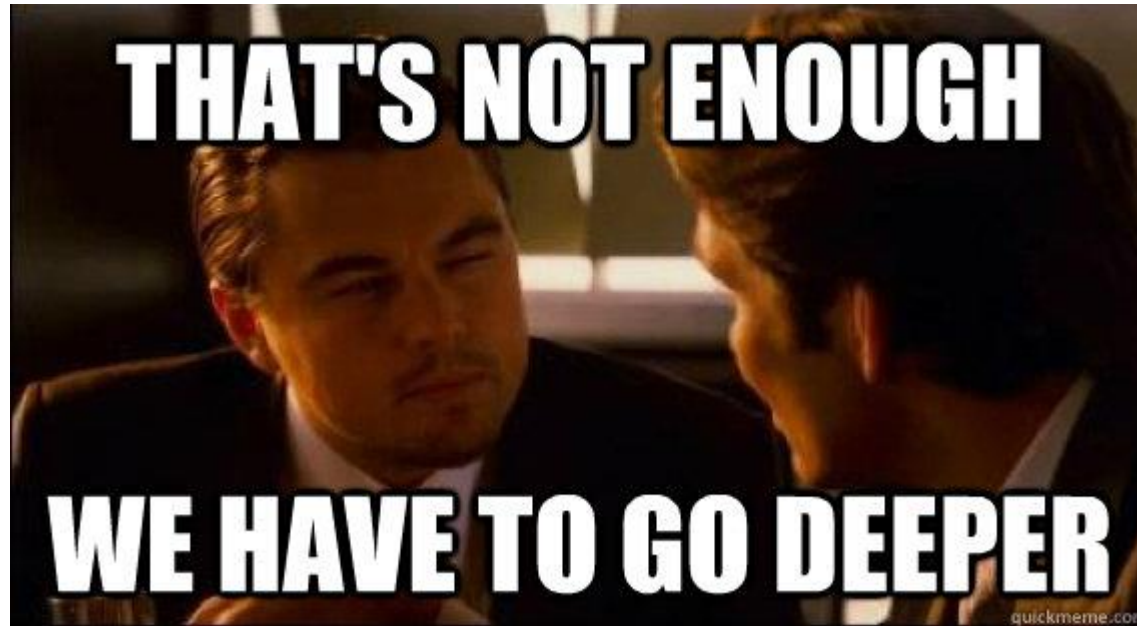
$$= 3 \times C \times (3 \times 3 \times C) = 27 C^2$$

Number of multiply-adds:

$$= 3 \times (H \times W \times C) \times (3 \times 3 \times C)$$

$$= 27 HWC^2$$

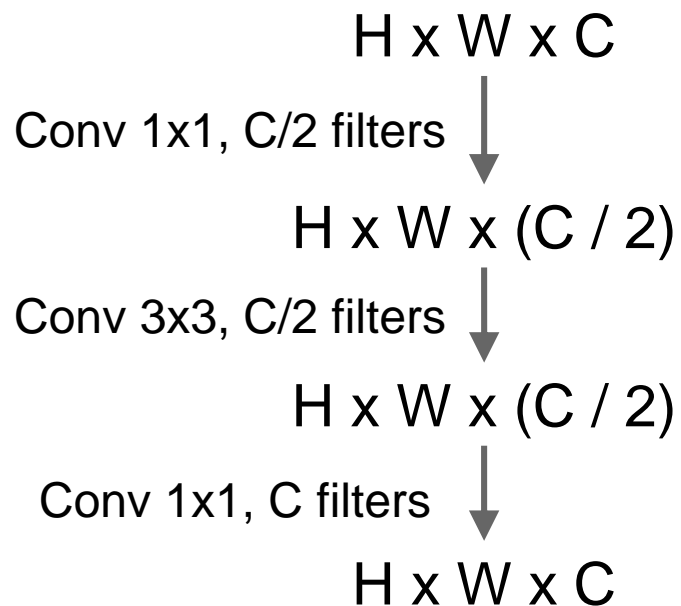
Less weight
Less compute



Convolution filter smaller than 3×3 ?

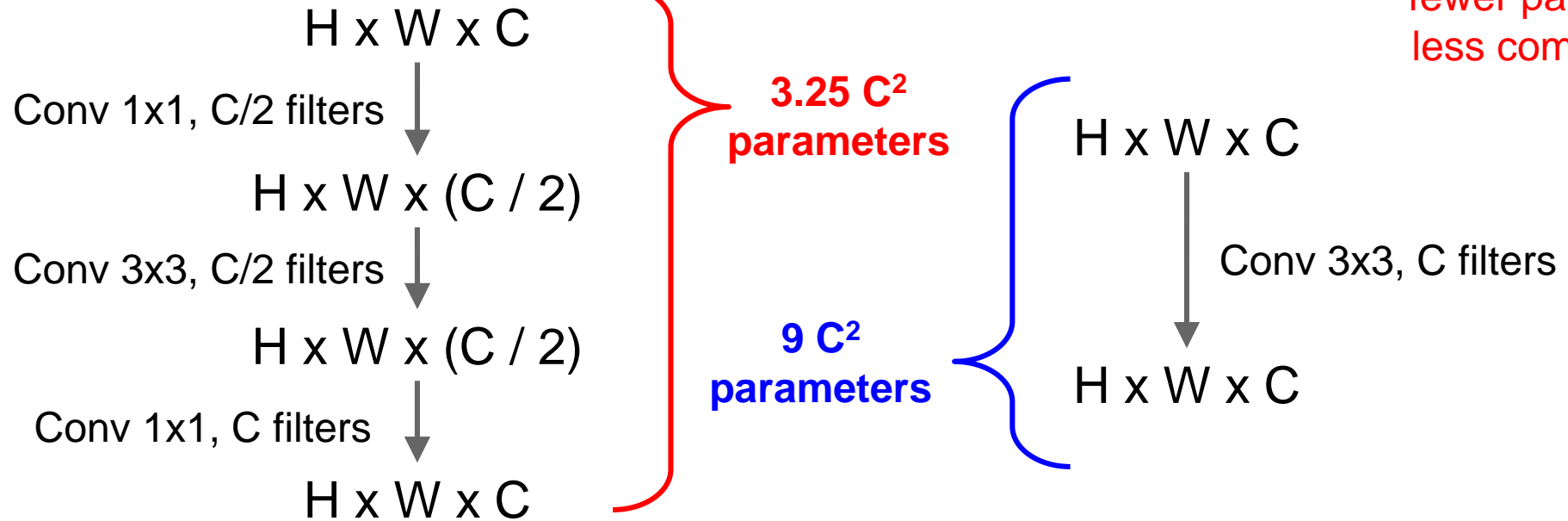
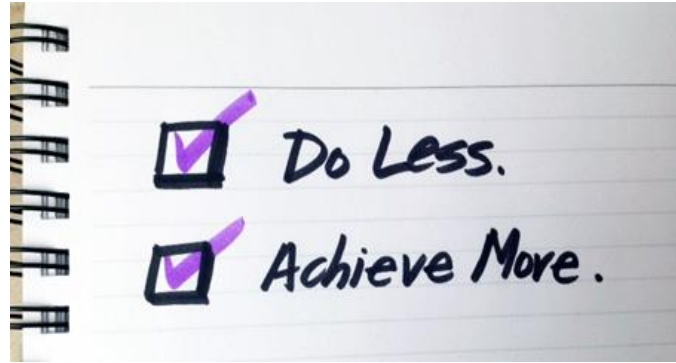
1x1 Convolution

Bottleneck Layer



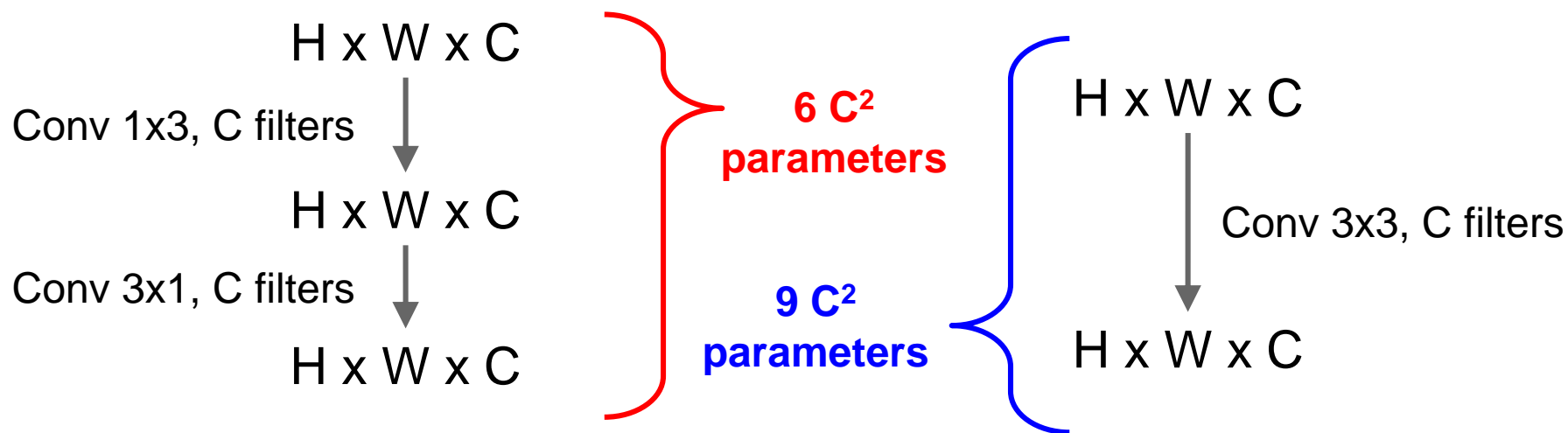
1. “bottleneck” 1 x 1 conv to reduce dimension
2. 3 x 3 conv at reduced dimension
3. Restore dimension with another 1 x 1 conv

[Seen in Lin et al, “Network in Network”, GoogLeNet, ResNet]



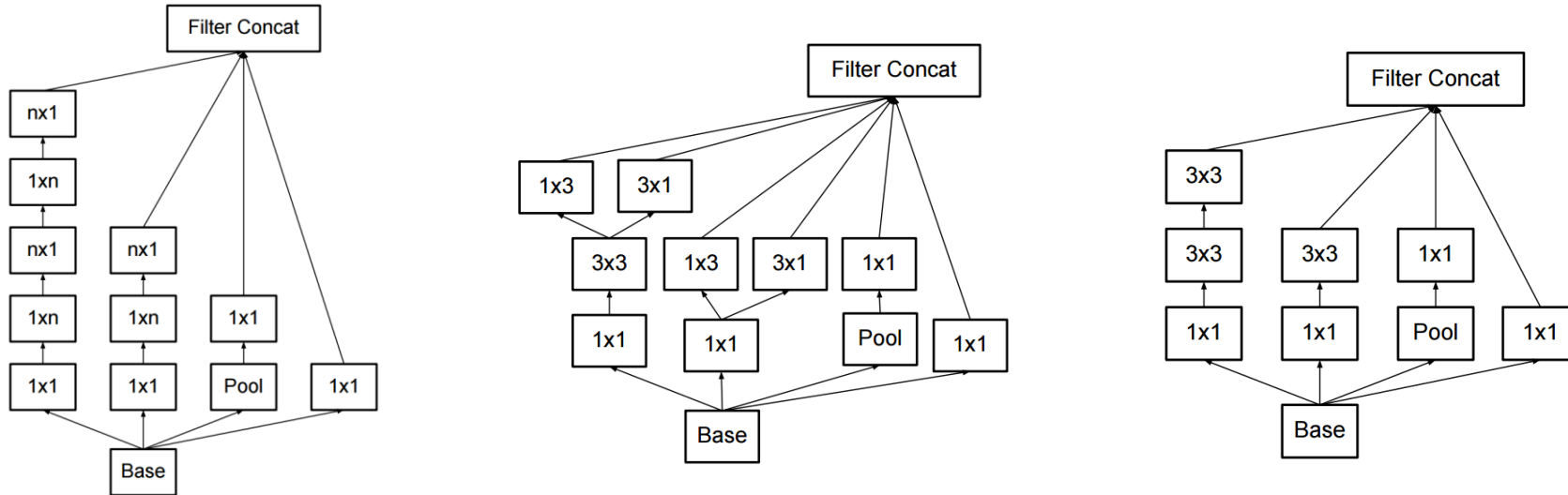
Breaking 3x3 filter

Divide & Conquer



Recent GoogLeNet

New More Inception Layers

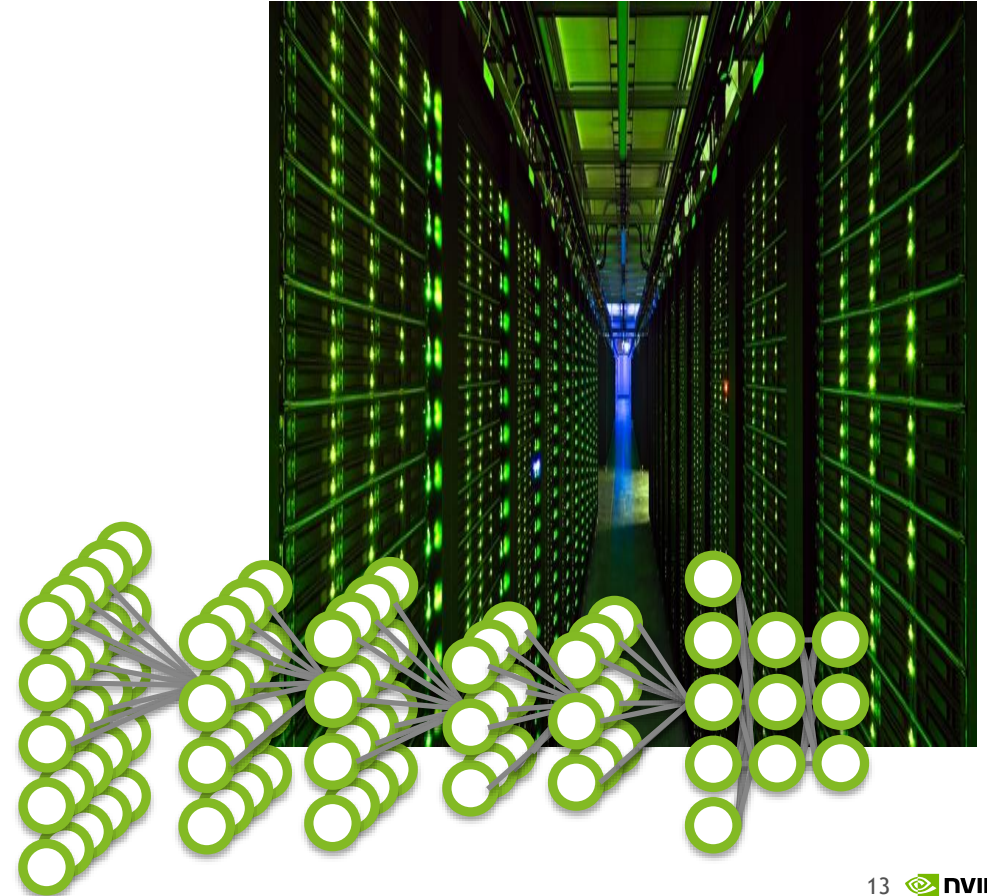


Szegedy et al, "Rethinking the Inception Architecture for Computer Vision"

GPU Computing

GPU for Computing

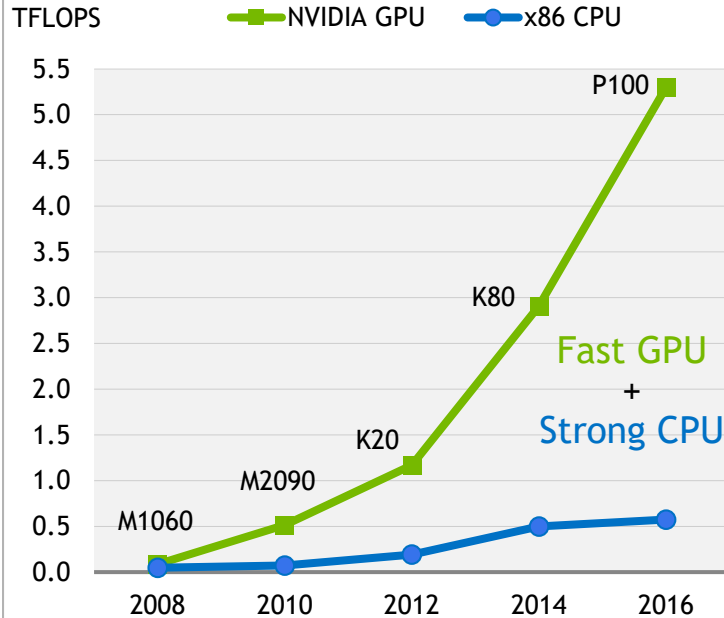
A good friend for Computing



TESLA: GPU Accelerated computing platform

Focused on Co-Design for Accelerated Data Center

Fast GPU Engineered for High Throughput



Productive Programming Model & Tools



Expert Co-Design

APPLICATION

MIDDLEWARE

SYS SW

LARGE SYSTEMS

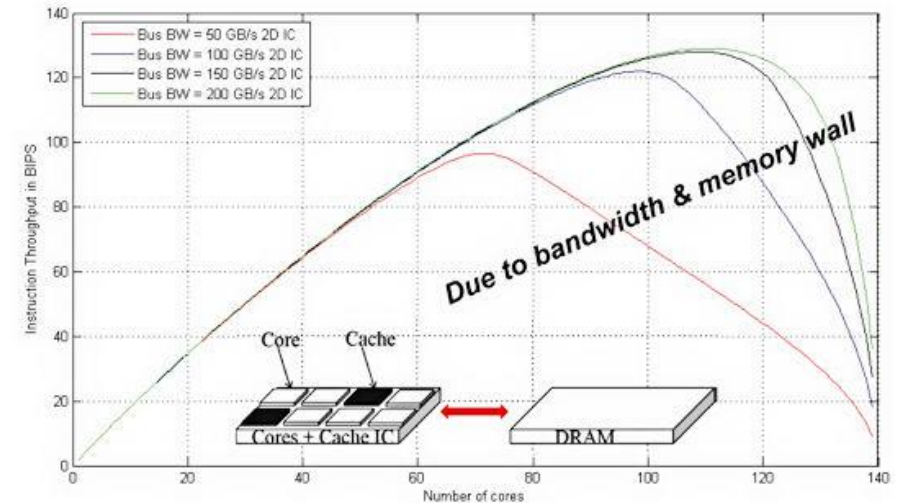
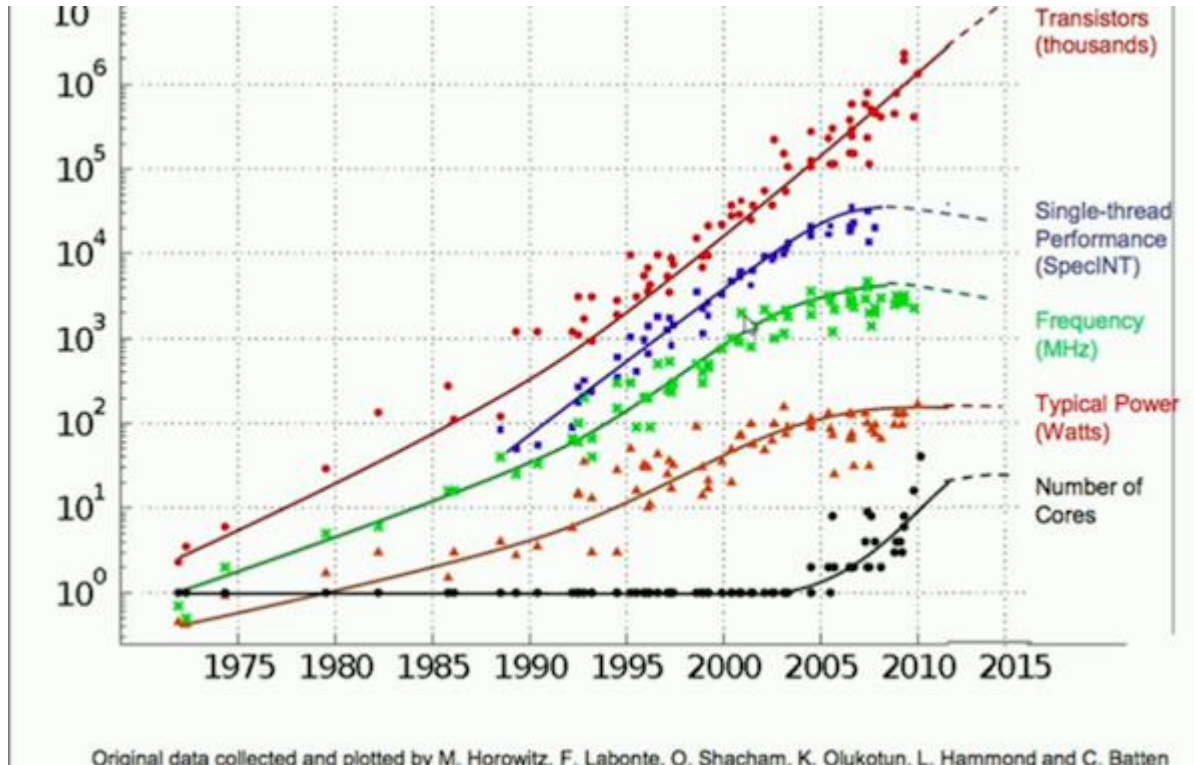
PROCESSOR

Accessibility



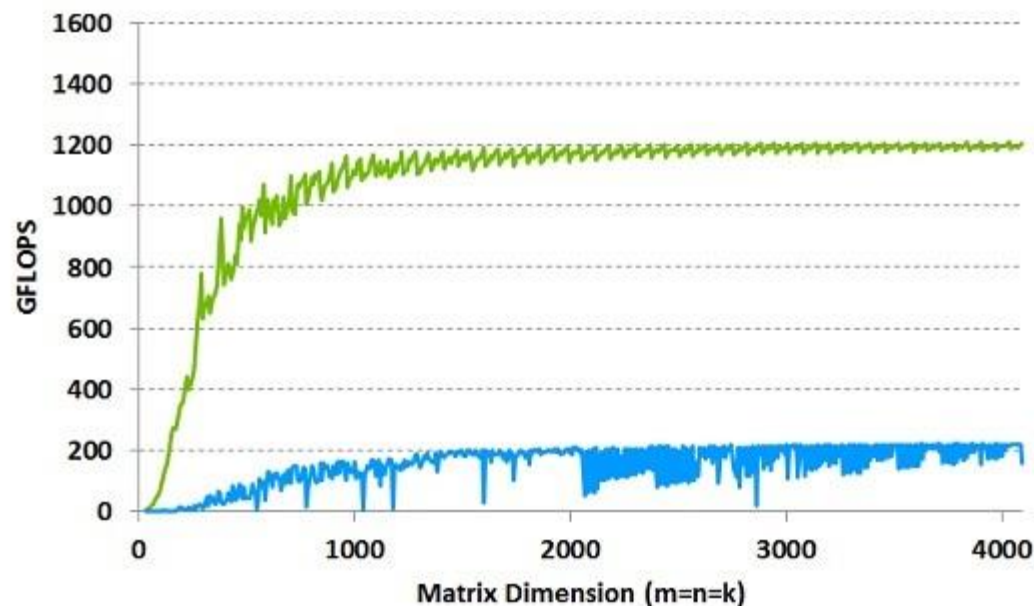
No More Free Lunch

So many things to consider for High Performance



GPU are really good at Matrix Multiplication

Which Neural Network Does



GPU: NVIDIA Tesla K40
with cuBLAS

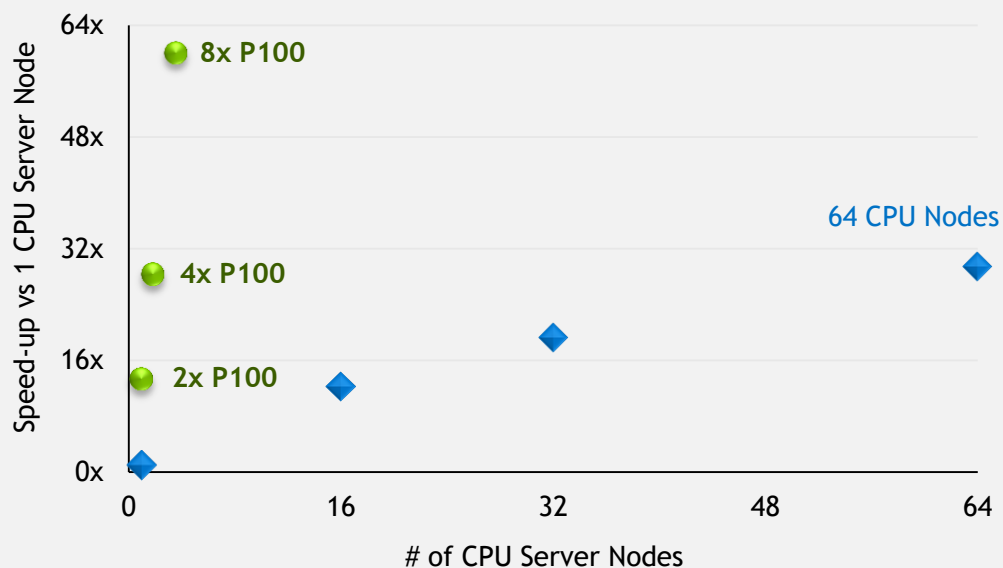
CPU: Intel E5-2697 v2
12 core @ 2.7 Ghz
with MKL

EXTRAORDINARY STRONG SCALING

One Strong Node Faster Than Lots of Weak Nodes

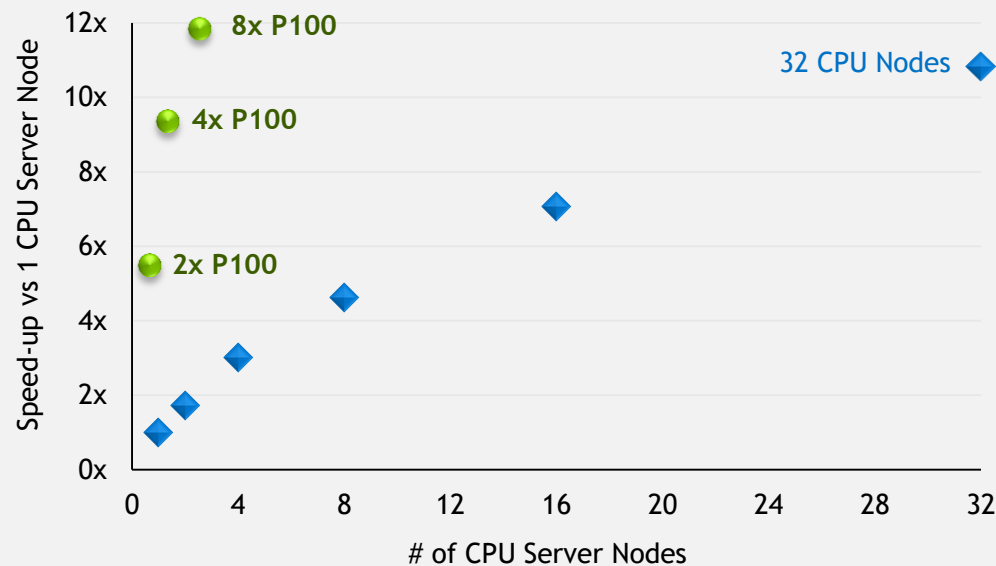
CAFFE ALEXNET PERFORMANCE

Single P100 NVLink-enabled Node vs Lots of Weak Nodes



VASP PERFORMANCE

Single P100 PCIe Node vs Lots of Weak Nodes

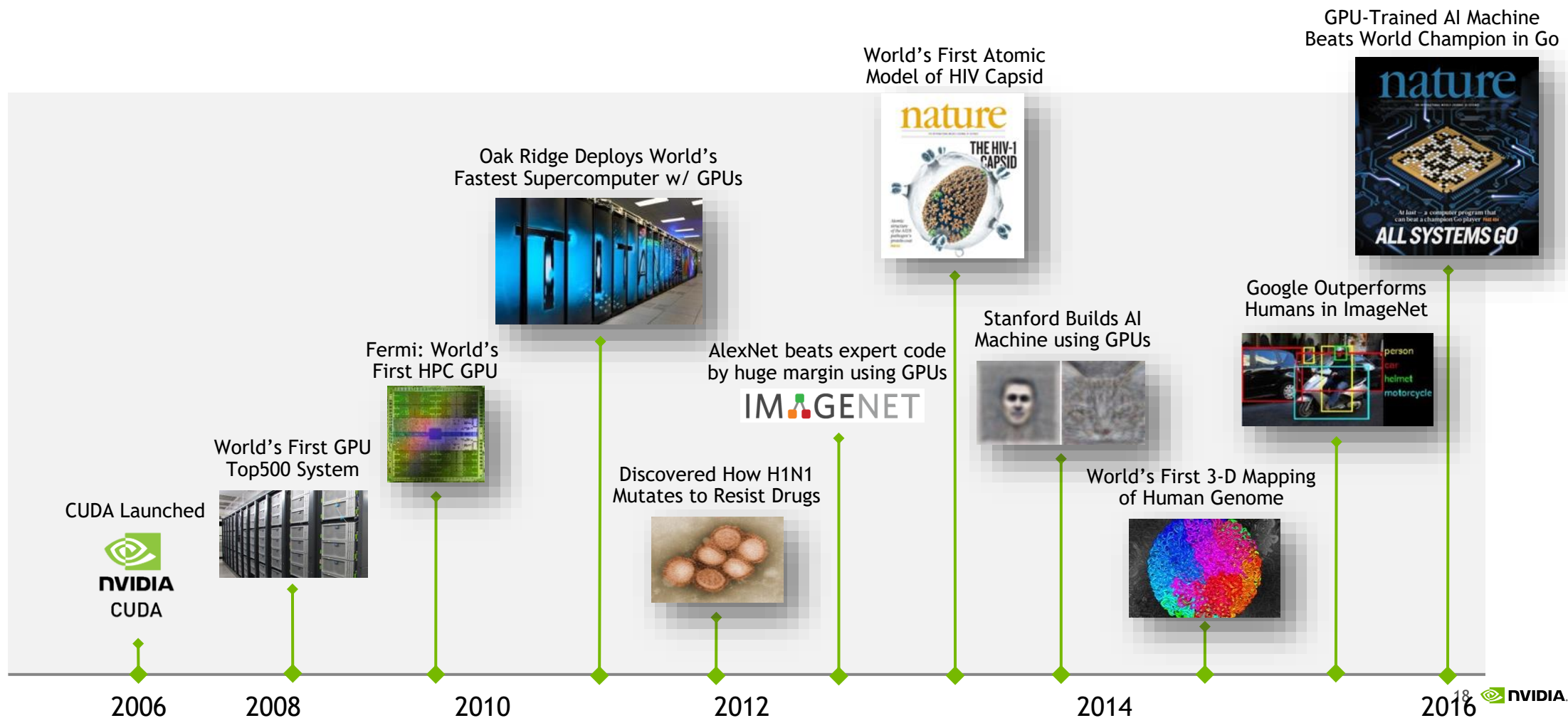


CPU: Dual Socket Intel E5-2680v3 12 cores, 128 GB DDR4 per node, FDR IB

VASP 5.4.1_05Feb16, Si-Huge Dataset. 16, 32 Nodes are estimated based on same scaling from 4 to 8 nodes

Caffe AlexNet scaling data: <https://software.intel.com/en-us/articles/caffe-training-on-multi-node-distributed-memory-systems-based-on-intel-xeon-processor-e5>

TEN YEARS OF GPU COMPUTING



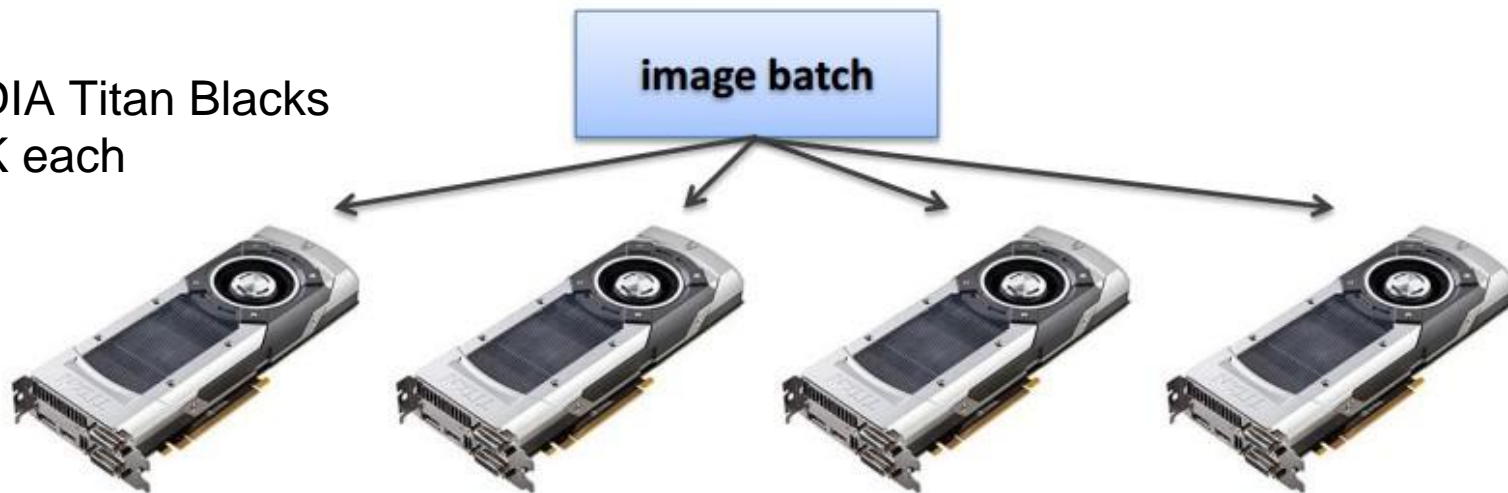
Need More computing Power?

Even with GPUs, training can be slow

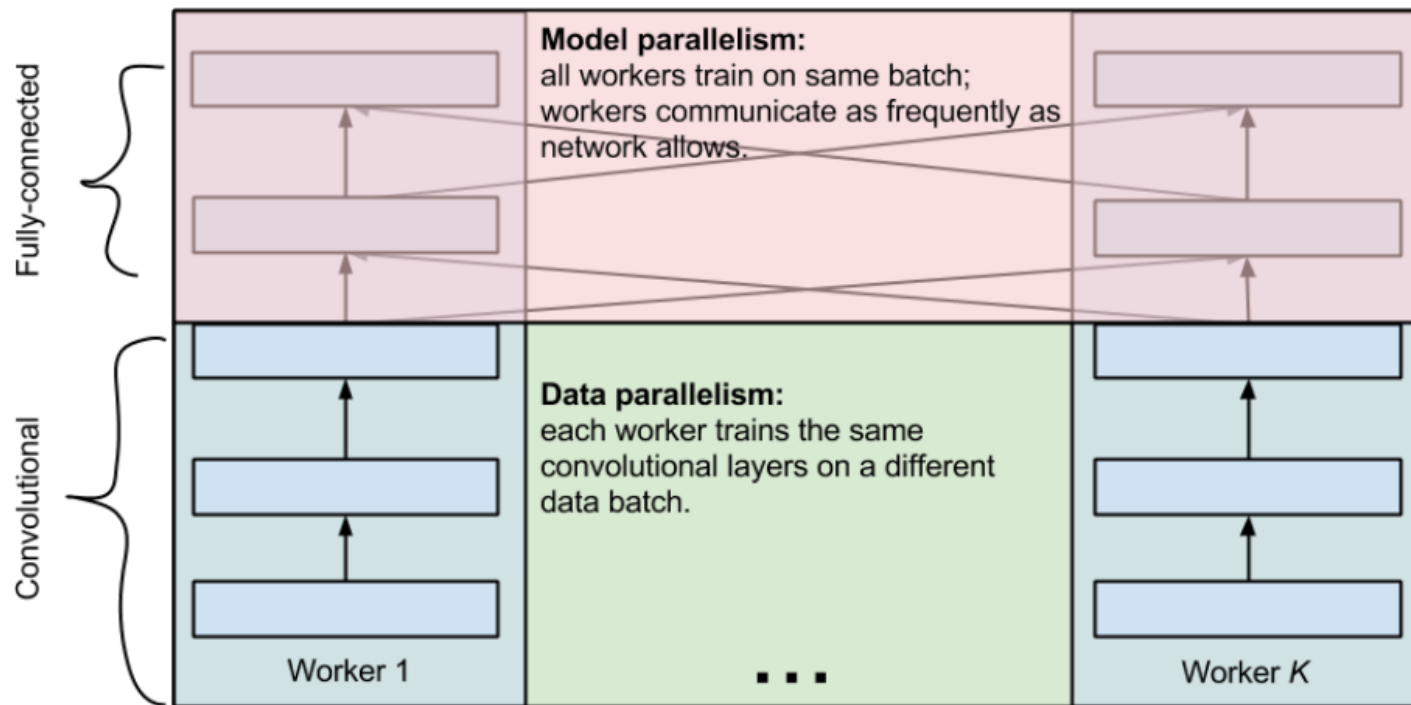
VGG: ~2-3 weeks training with 4 GPUs

ResNet 101: 2-3 weeks with 4 GPUs

NVIDIA Titan Blacks
~\$1K each



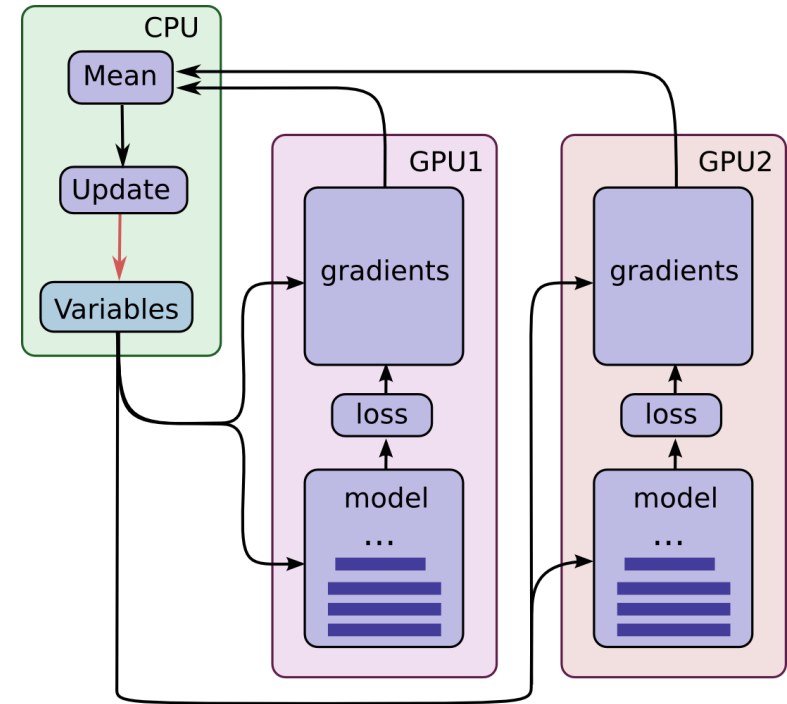
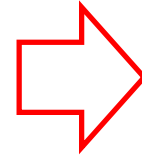
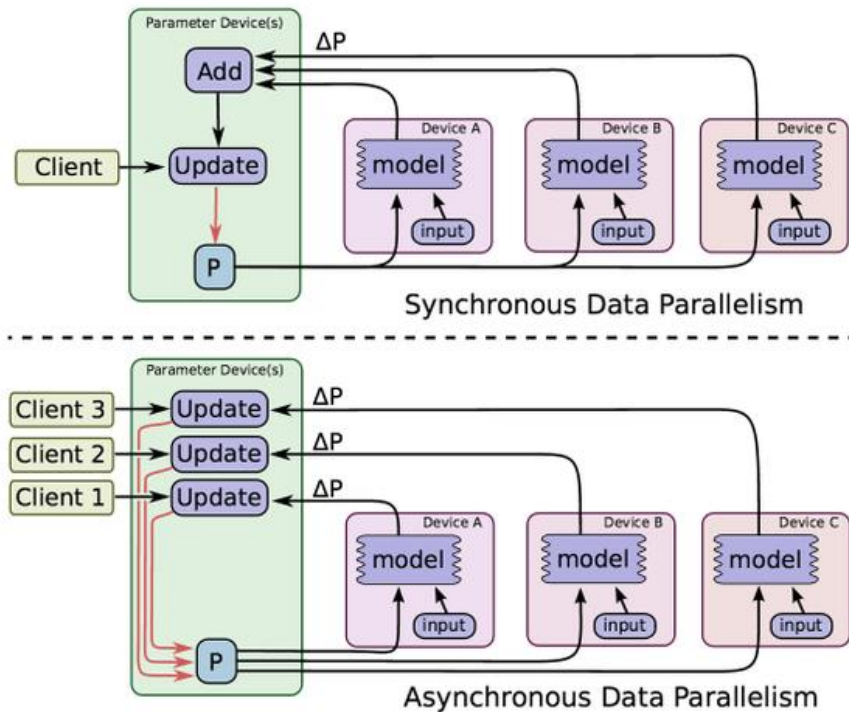
MultiGPU Training



Alex Krizhevsky, “One weird trick for parallelizing convolutional neural networks”

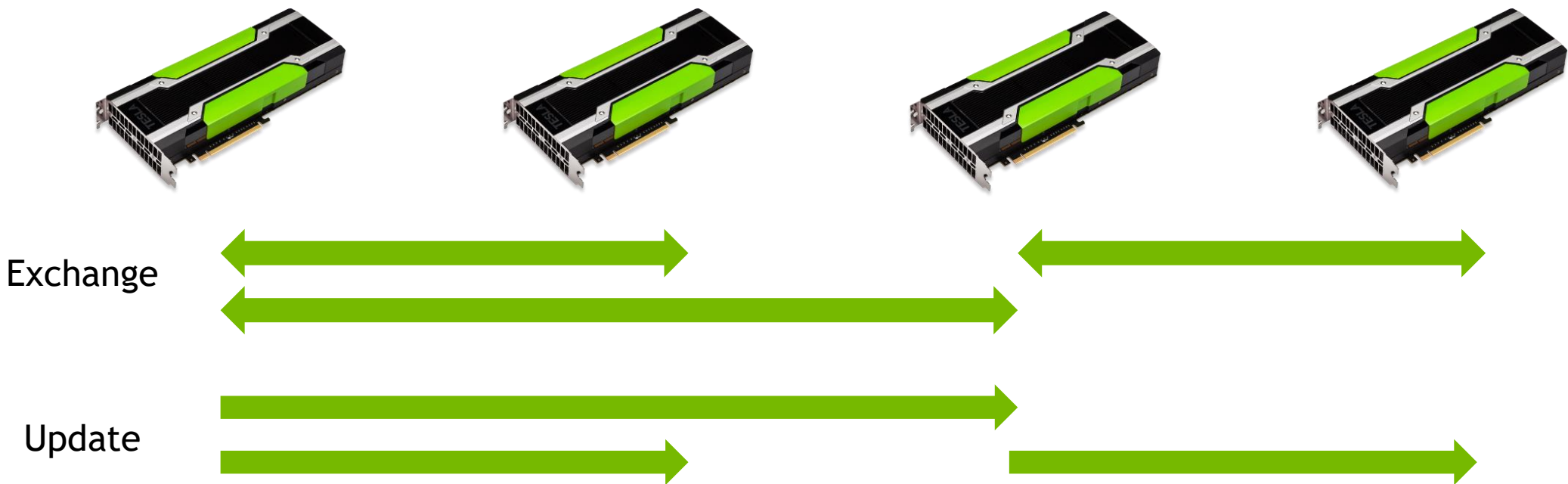
Tensorflow Approach

Synchronous Data Parallelism



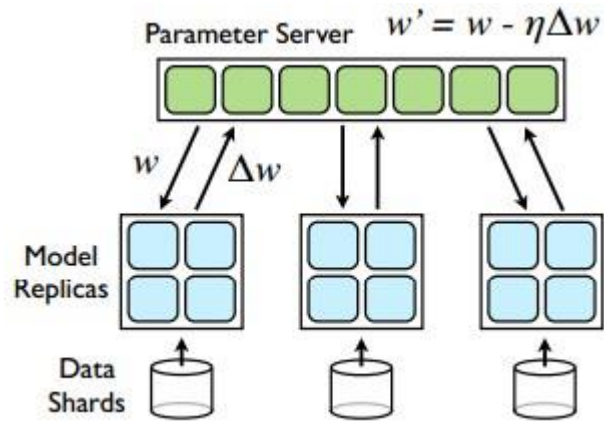
Caffe's Approach

Tree Reduction

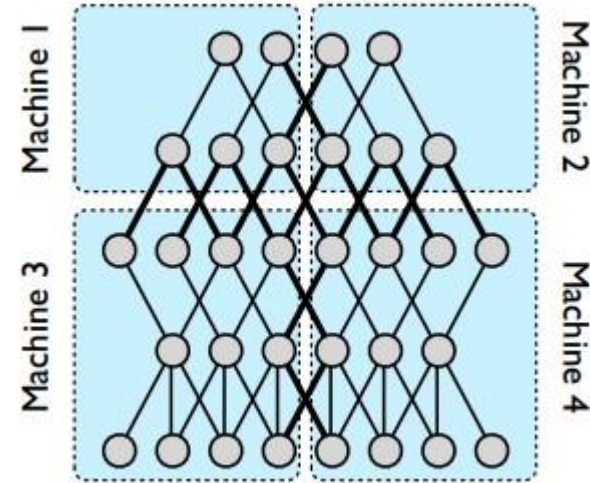


The current implementation uses a tree reduction strategy. e.g. if there are 4 GPUs in the system, 0:1, 2:3 will exchange gradients, then 0:2 (top of the tree) will exchange gradients, 0 will calculate updated model, 0->2, and then 0->1, 2->3.

Distributed CPU Training



Data parallelism



Model parallelism

[Large Scale Distributed Deep Networks, Jeff Dean et al., 2013]

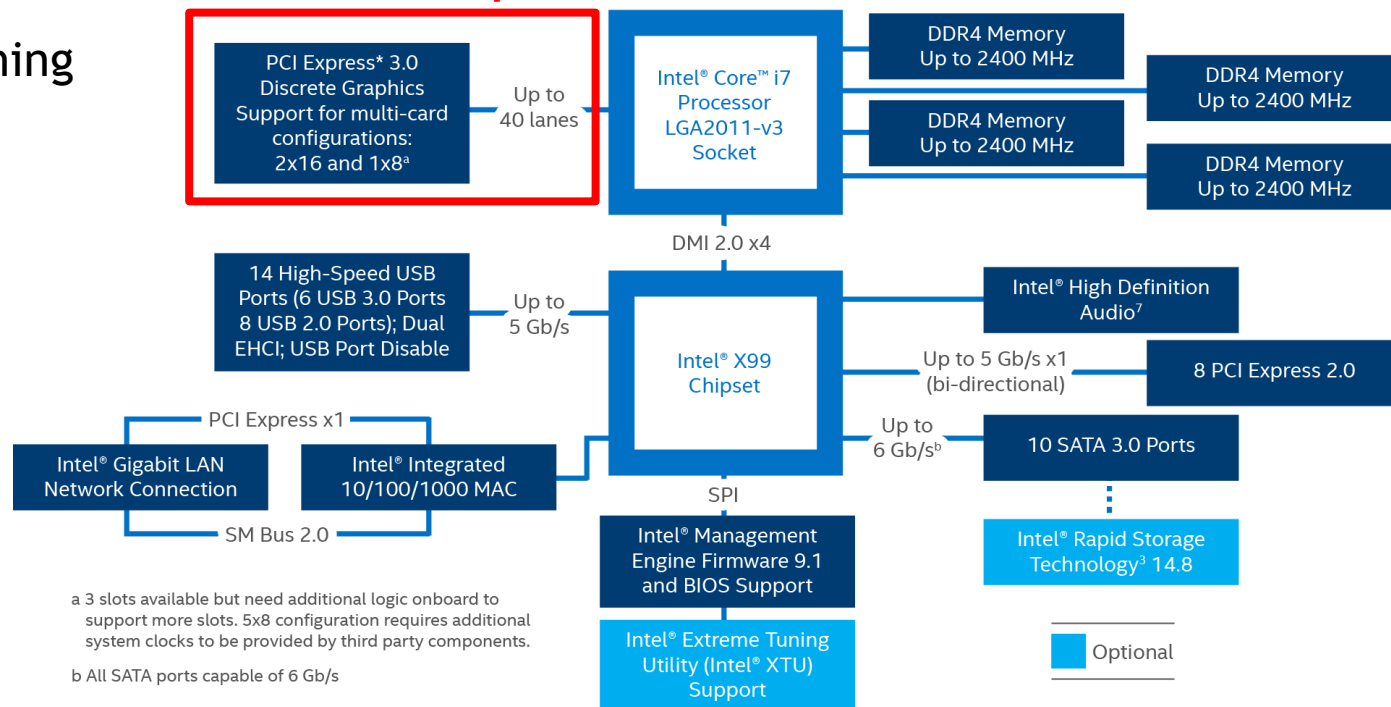
Bottlenecks



CPU - GPU communication bottleneck

CPU: data augment and prefetching
with multi-threads
GPU: Training

GPU communicate path

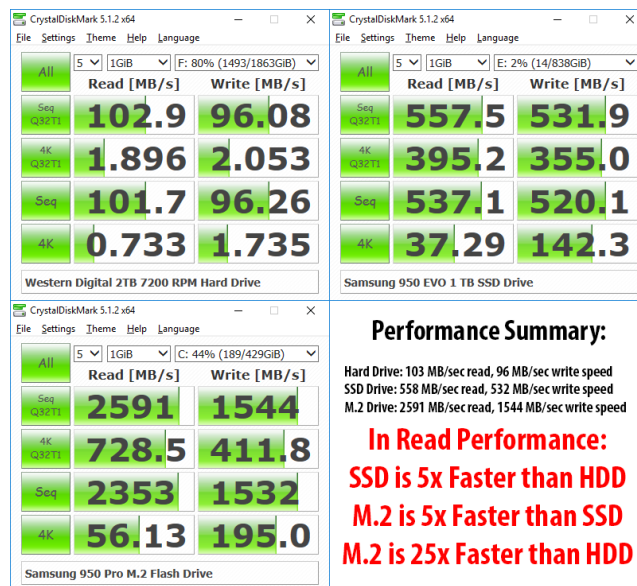




Storage bottleneck



- HDD is slow to read
- Pre-processed images stored contiguously in files, read as raw byte stream from SSD disk



Why do I Need a DB?

- Latency Matters.
- Sequential Read: ~150 MB/s
- Random Read: ~6 MB/s
- AlexNet needs: ~40 MB/s* (200 images/s)
- 8-GPU AlexNet needs: ~320 MB/s

*: Based on 256x256 uncompressed color image, and 5ms/image training speed.
Data is on HDD - SSD is another story

ImageNet Recipes from a Practical View, Yangqing Jia

GPU memory bottleneck

Titan X: 12 GB

Tesla P100: 16 GB

Tesla P40: 24GB

e.g.

AlexNet: ~3GB needed with batch size 256

Floating Point Precision

Floating Precision

32 bit “single” precision is typically used for CNNs for performance

Prediction and statues:

16 bit “half” precision will be the new standard
GPU already support FP16 on memory
Hardware support is ready for P100
Frameworks need to be ready for HW support

AlexNet (One Weird Trick paper) - Input 128x3x224x224

Library	Class	Time (ms)	forward (ms)	backward (ms)
CuDNN[R4]-fp16 (Torch)	cudnn.SpatialConvolution	71	25	46
Nervana-neon-fp16	ConvLayer	78	25	52
CuDNN[R4]-fp32 (Torch)	cudnn.SpatialConvolution	81	27	53
TensorFlow	conv2d	81	26	55
Nervana-neon-fp32	ConvLayer	87	28	58
fbfft (Torch)	fbnn.SpatialConvolution	104	31	72
Chainer	Convolution2D	177	40	136
cudaconvnet2*	ConvLayer	177	42	135
CuDNN[R2] *	cudnn.SpatialConvolution	231	70	161
Caffe (native)	ConvolutionLayer	324	121	203
Torch-7 (native)	SpatialConvolutionMM	342	132	210
CL-nn (Torch)	SpatialConvolutionMM	963	388	574
Caffe-CLGreenTea	ConvolutionLayer	1442	210	1232

Benchmarks on Titan X, from

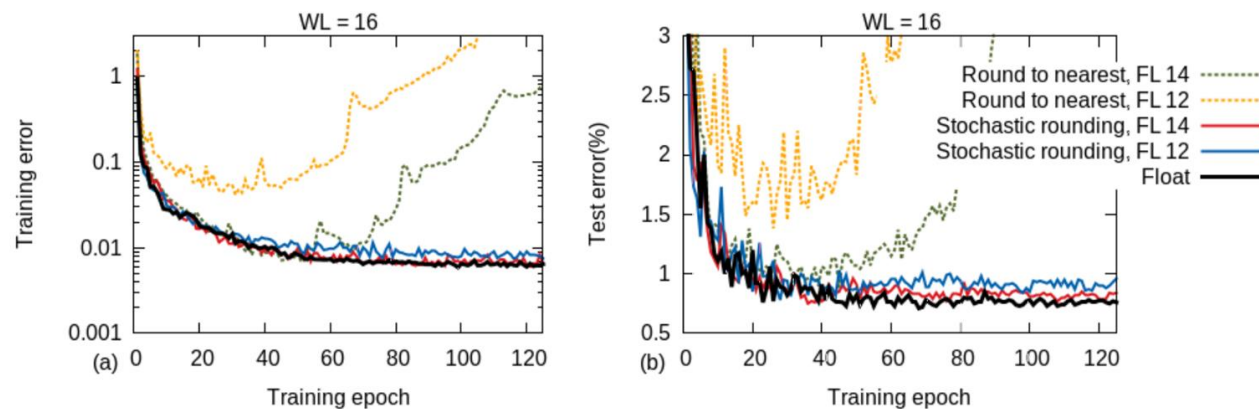
<https://github.com/soumith/convnet-benchmarks>

FP16 Training?

How low can we go?

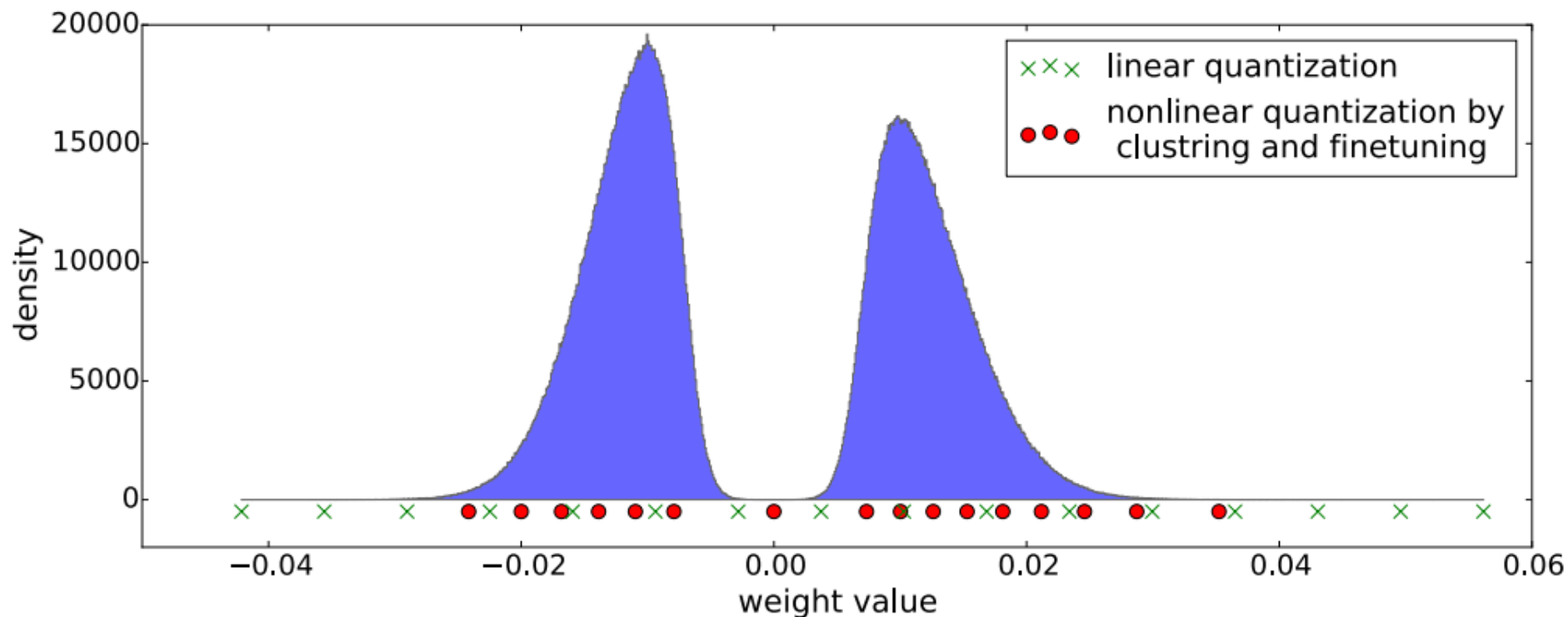
Gupta et al, 2015:

Train with **16-bit fixed point** with stochastic rounding



Gupta et al, "Deep Learning with Limited Numerical Precision", ICML 2015

Trained Quantization

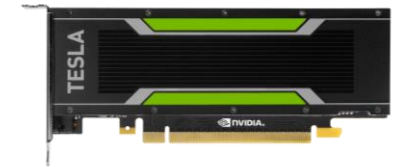
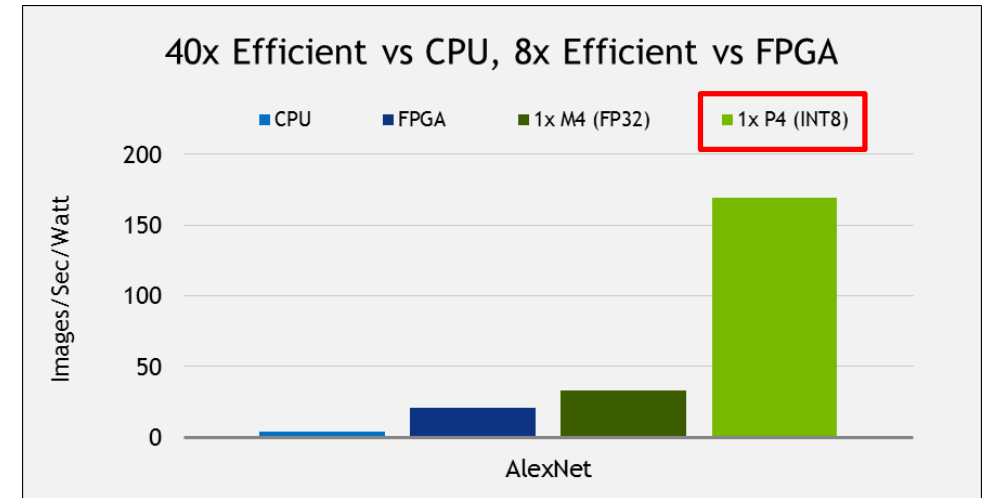
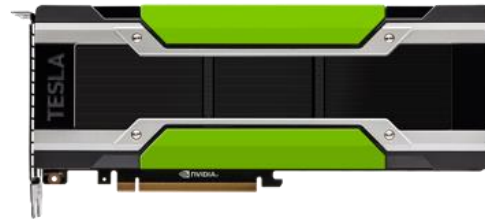
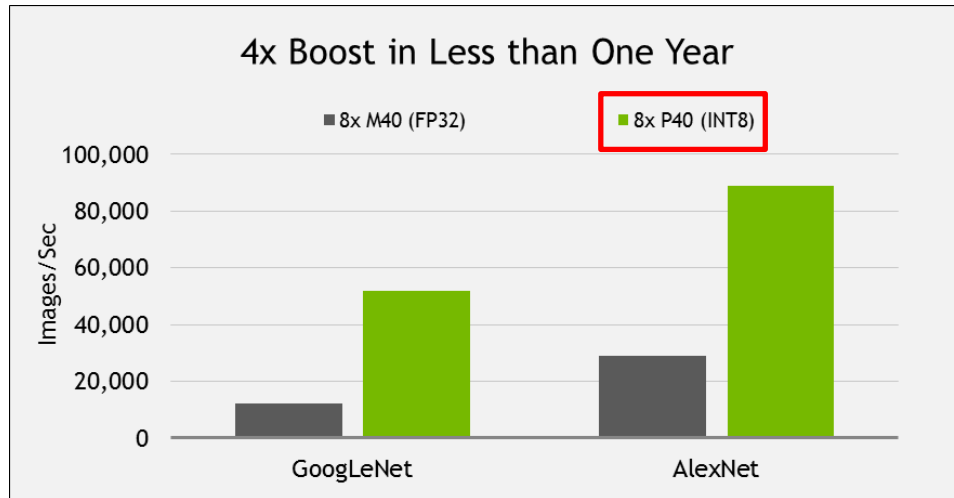


Precision

		Range	Accuracy
FP32	<div> <div>1</div> <div>8</div> <div>23</div> <div>S</div> <div>E</div> <div>M</div> </div>	$10^{-38} - 10^{38}$.000006%
FP16	<div> <div>1</div> <div>5</div> <div>10</div> <div>S</div> <div>E</div> <div>M</div> </div>	$6 \times 10^{-8} - 6 \times 10^4$.05%*
Int32	<div> <div>1</div> <div>31</div> <div>S</div> <div>M</div> </div>	$0 - 2 \times 10^9$	$\frac{1}{2}$
Int16	<div> <div>1</div> <div>15</div> <div>S</div> <div>M</div> </div>	$0 - 6 \times 10^4$	$\frac{1}{2}$
Int8	<div> <div>1</div> <div>7</div> <div>S</div> <div>M</div> </div>	$0 - 127$	$\frac{1}{2}$
Binary	<div> <div>1</div> <div>M</div> </div>	$0-1$	$\frac{1}{2}$

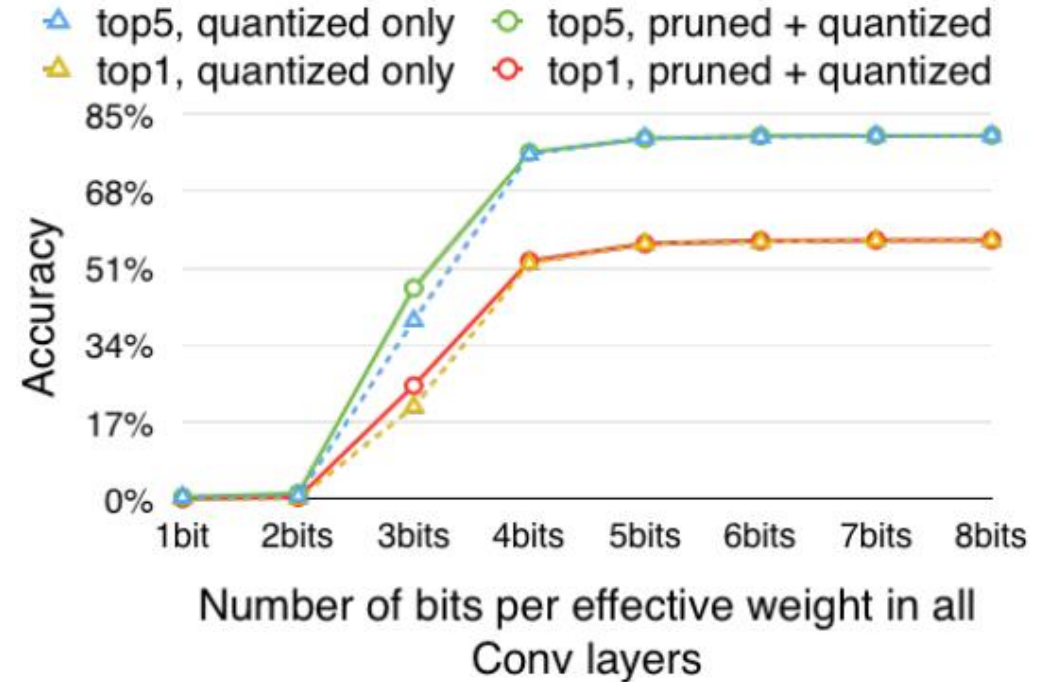
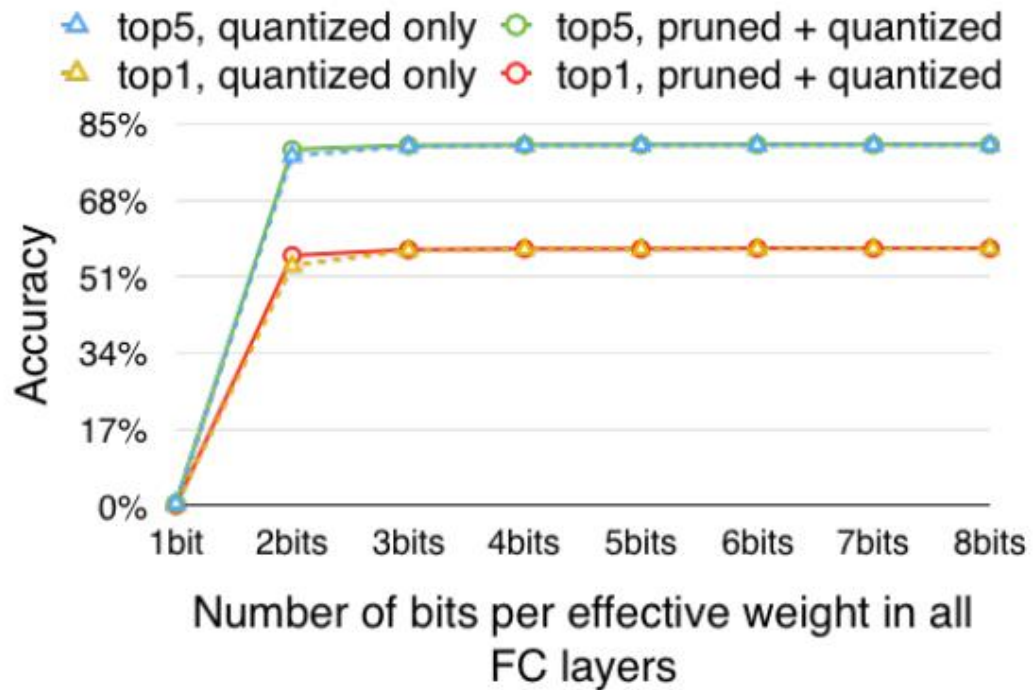
INT8 Inference

NVIDIA readied such Hardware Accelerator



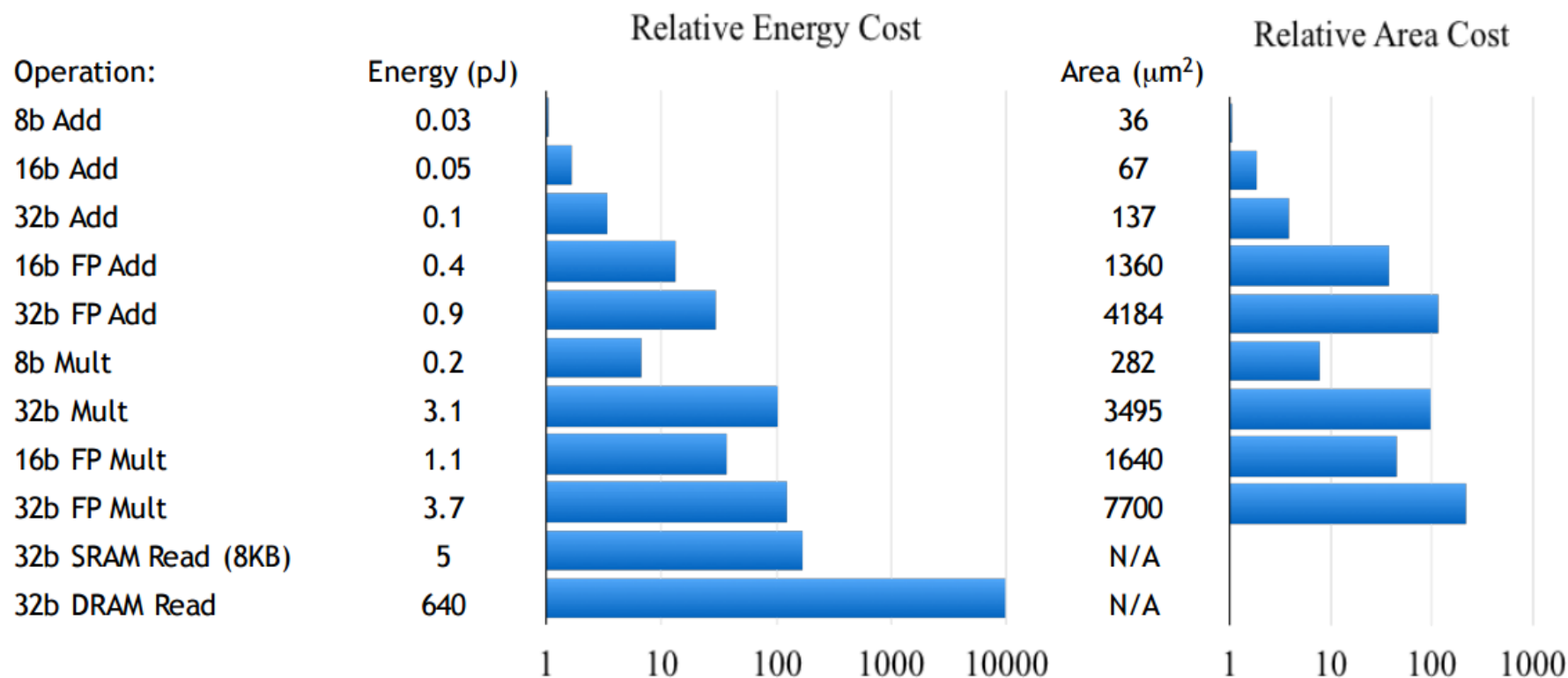
2-6 bits/Weight Sufficient

Inference for ANN/CNN



Lower Computing Cost

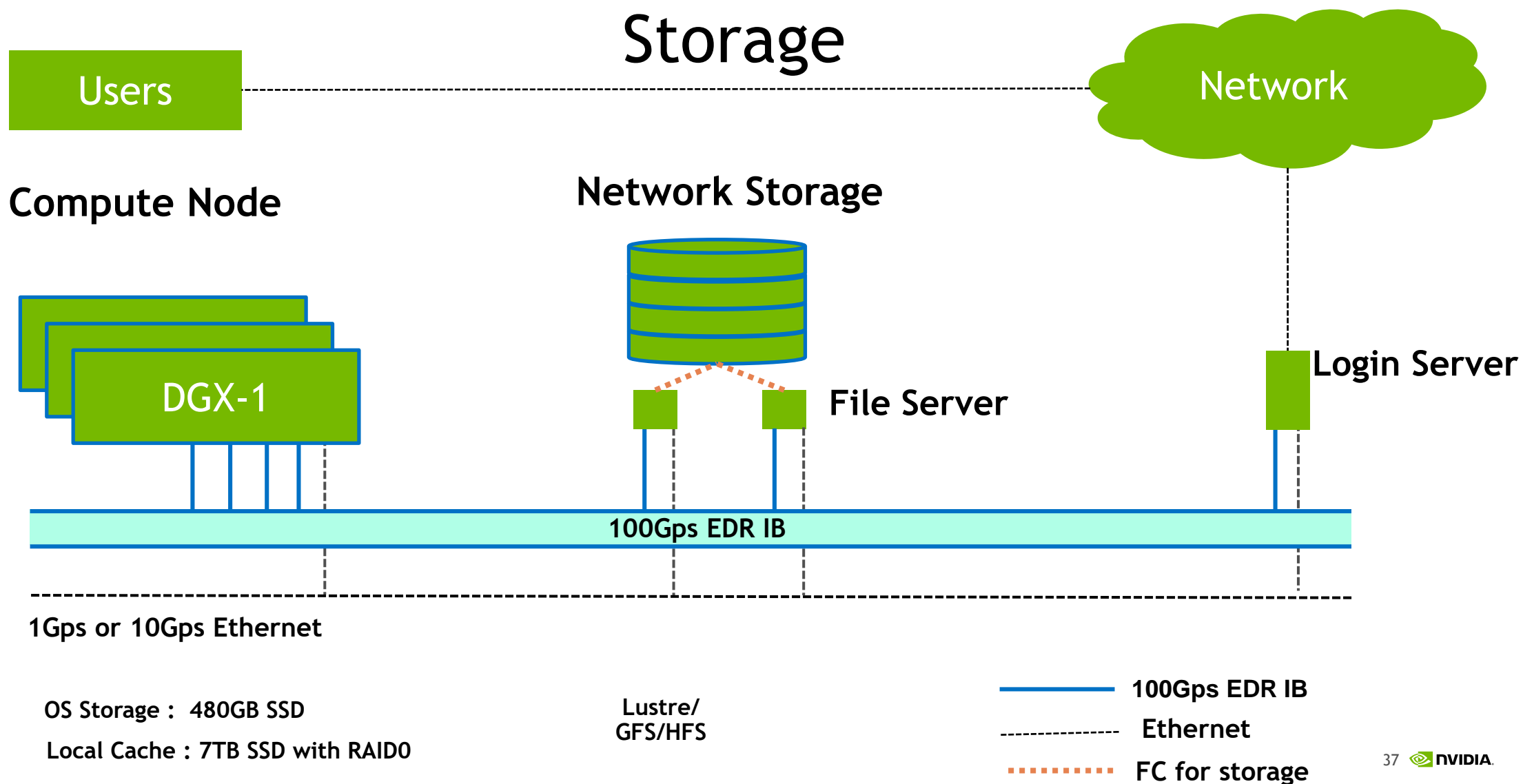
In energy and area



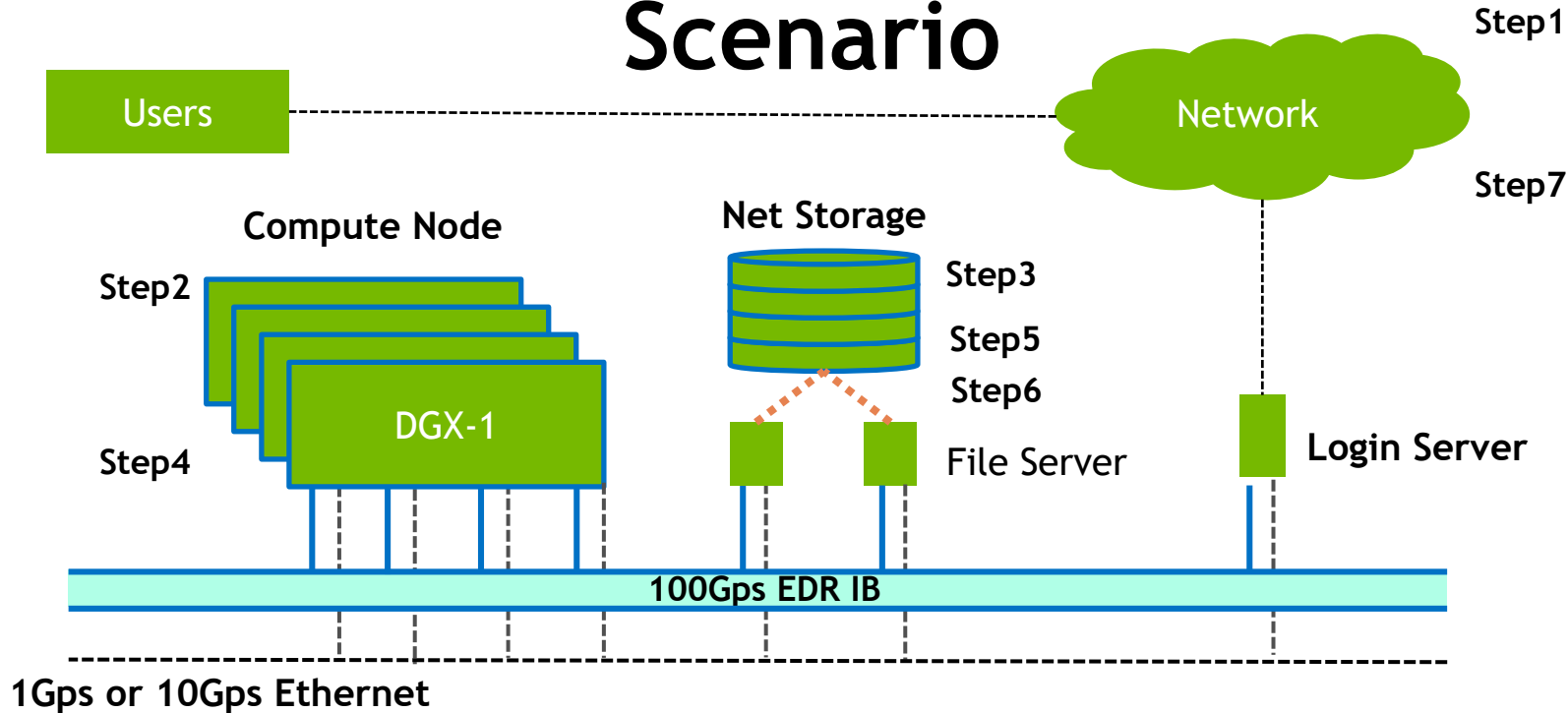
Energy numbers are from Mark Horowitz “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014 Area numbers are from synthesized result using Design Compiler under TSMC 45nm tech node. FP units used DesignWare Library.

Deep Learning R&D Infra

Storage



Scenario



- Step1. login server
- Step2. assign dedicate DGX-1 in scheduler (slurm/lsf/pbs/sge)
- Step3. pull 3TB dataset from Network File System to local cache
- Step4. training with DGX-1
- Step5. result param in local cache
- Step6. push result to Network Storage
- Step7. release node & logout

Caffe/DIGITS Interactive Mode

Step1 Login Node

```
ssh hryu@psgcluster.nvidia.com
```

Step2 assign empty DGX-1 node

```
srun --comment=docker --pty bash -i
```

Step3 load DL framework

```
docker pull nvidia/caffe or nvidia/digits
```

```
nvidia-docker run --rm -p 5000:34448 -v $(pwd):/data -v $(pwd):/jobs nvidia/digits
```

```
[hryu@ivb104 imagenet]$ docker images
REPOSITORY          TAG                 IMAGE ID
nvidia/digits        latest             15c2e99ab
[hryu@ivb104 imagenet]$ module unload digits/digi
[hryu@ivb104 imagenet]$ nvidia-docker run --rm -p
2016-11-18 05:44:06 [9] [INFO] Starting gunicorn
libdc1394 error: Failed to initialize libdc1394
2016-11-18 05:44:07 [9] [DEBUG] Arbiter booted
2016-11-18 05:44:07 [9] [INFO] Listening at: http
2016-11-18 05:44:07 [9] [INFO] Using worker: sock
2016-11-18 05:44:07 [117] [INFO] Booting worker v
2016-11-18 05:44:07 [INFO ] Loaded 0 jobs.
```

New Image Classification

10.31.229.80:5000/datasets/images/classification/new

DIGITS New Dataset hryu (Logout) Info

New Image Classification Dataset

Image Type ?

Color

Image size (Width x Height) ?

256 x 256

Resize Transformation ?

Squash

See example

Use Image Folder Use Text Files

Training Images ?

/

Minimum samples per class ?

2

Maximum samples per class ?

% for validation ?

25

% for testing ?

0

☐ Separate validation image folder

Tensorflow Interactive Mode

Step1 Login Node

```
ssh hryu@psgcluster.nvidia.com
```

Step2 assign empty DGX-1 node

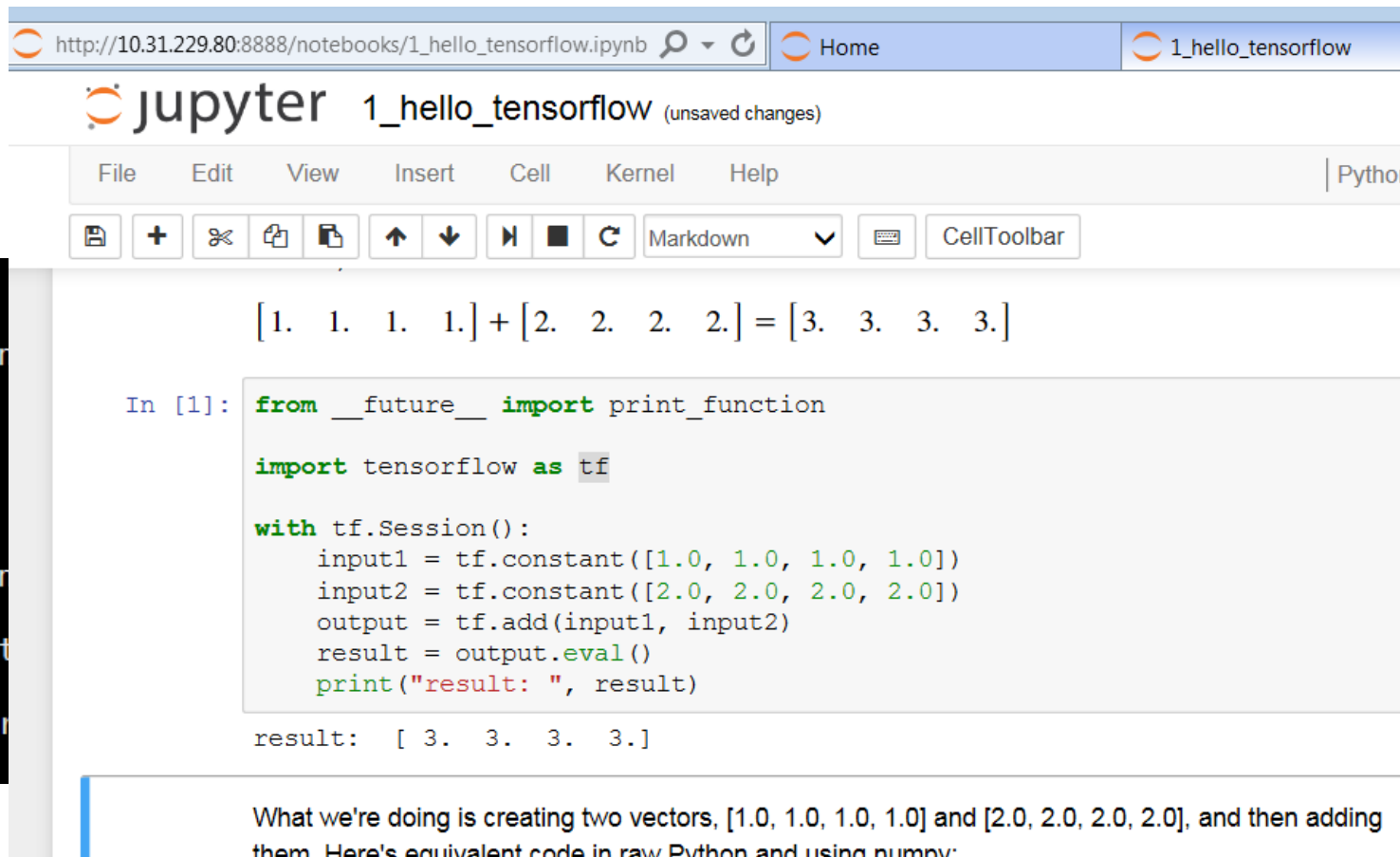
```
srun --comment=docker --pty bash -i
```

Step3 load DL framework

```
docker pull gcr.io/tensorflow/tensorflow:latest-gpu
```

```
nvidia-docker run -it -p 8888:8888 -p 6006:6006 gcr.io/tensorflow/tensorflow:latest-gpu
```

```
[hryu@ivb104 imagenet]$ nvidia-docker-  
-gpu  
[I 06:03:59.701 NotebookApp] Writing r  
tebook_cookie_secret  
[W 06:03:59.716 NotebookApp] WARNING:  
ryption. This is not recommended.  
[W 06:03:59.716 NotebookApp] WARNING:  
thentication. This is highly insecure  
[I 06:03:59.732 NotebookApp] Serving r  
[I 06:03:59.733 NotebookApp] 0 active  
[I 06:03:59.733 NotebookApp] The Jupyter  
888/  
[I 06:03:59.733 NotebookApp] Use Contr  
firmation).
```



The screenshot shows a Jupyter Notebook interface in a web browser. The address bar displays the URL `http://10.31.229.80:8888/notebooks/1_hello_tensorflow.ipynb`. The notebook title is `1_hello_tensorflow` with a note about unsaved changes. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help) and a toolbar with icons for saving, adding cells, and other functions. A code cell is active, containing the following Python code:

```
In [1]: from __future__ import print_function  
  
import tensorflow as tf  
  
with tf.Session():  
    input1 = tf.constant([1.0, 1.0, 1.0, 1.0])  
    input2 = tf.constant([2.0, 2.0, 2.0, 2.0])  
    output = tf.add(input1, input2)  
    result = output.eval()  
    print("result: ", result)
```

The output of the cell is displayed below the code:

```
result: [ 3.  3.  3.  3.]
```

Below the output, there is a text box with the following explanation:

What we're doing is creating two vectors, `[1.0, 1.0, 1.0, 1.0]` and `[2.0, 2.0, 2.0, 2.0]`, and then adding them. Here's equivalent code in raw Python and using `numpy`:

Assign GPUs for Docker

Use Environment Variables NV_GPU

`NV_GPU='{GPU Index}' nvidia-docker run {docker options}`

```
[~]$ nvidia-smi
Tue Nov 15 06:08:48 2016
```

NVIDIA-SMI 361.77				Driver Version: 361.77			
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr. ECC	
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	
0	Tesla P100-SXM2...	On	0000:06:00.0	Off			0
N/A	36C	P0	31W / 300W	0MiB / 16280MiB	0%	Default	
1	Tesla P100-SXM2...	On	0000:07:00.0	Off			0
N/A	35C	P0	31W / 300W	0MiB / 16280MiB	0%	Default	
2	Tesla P100-SXM2...	On	0000:0A:00.0	Off			0
N/A	34C	P0	31W / 300W	0MiB / 16280MiB	0%	Default	
3	Tesla P100-SXM2...	On	0000:0B:00.0	Off			0
N/A	34C	P0	31W / 300W	0MiB / 16280MiB	0%	Default	
4	Tesla P100-SXM2...	On	0000:85:00.0	Off			0
N/A	37C	P0	30W / 300W	0MiB / 16280MiB	0%	Default	
5	Tesla P100-SXM2...	On	0000:86:00.0	Off			0
N/A	37C	P0	31W / 300W	0MiB / 16280MiB	0%	Default	
6	Tesla P100-SXM2...	On	0000:89:00.0	Off			0
N/A	38C	P0	31W / 300W	0MiB / 16280MiB	0%	Default	
7	Tesla P100-SXM2...	On	0000:8A:00.0	Off			0
N/A	33C	P0	31W / 300W	0MiB / 16280MiB	0%	Default	

`$NV_GPU='1,3' nvidia-docker run`

GPUs in Docker

Only 2 GPUs within Docker

`$NV_GPU='1,3' nvidia-docker run`

2 GPUs is assigned in Docker

Scheduler (slurm/LSF/PBS/SGE)
manage resources

NV_GPU environment
allocate resource to NV_GPU variables

NVDocker
load NV_GPU variables

```
[[~]$ NV_GPU='1,3' nvidia-docker run --rm compute.nvidia.com/nvidia/cuda nvidia-smi
Tue Nov 15 14:06:28 2016
```

NVIDIA-SMI 361.77				Driver Version: 361.77			
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute	M.
0	Tesla P100-SXM2...	Off	0000:07:00.0	Off			0
N/A	35C	P0	31W / 300W	0MiB / 16280MiB	0%	Default	
1	Tesla P100-SXM2...	Off	0000:0B:00.0	Off			0
N/A	34C	P0	32W / 300W	0MiB / 16280MiB	0%	Default	

```
Processes:
GPU      PID  Type  Process name                      GPU Memory
Usage
=====
No running processes found
```

