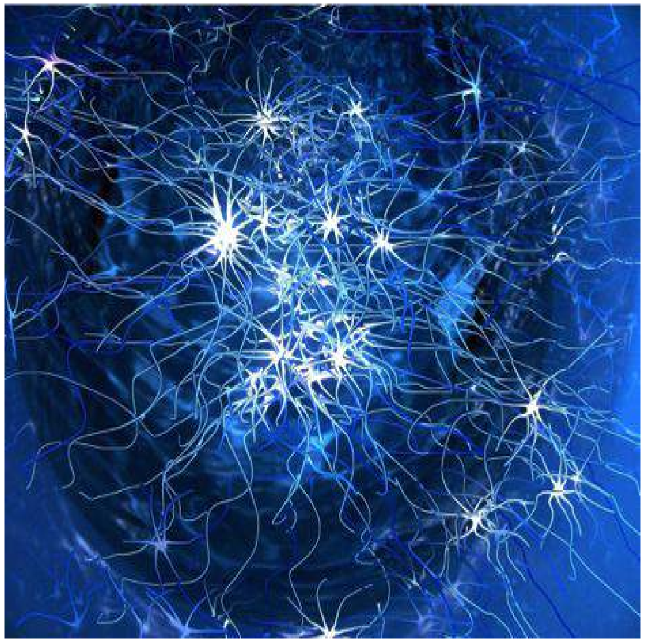
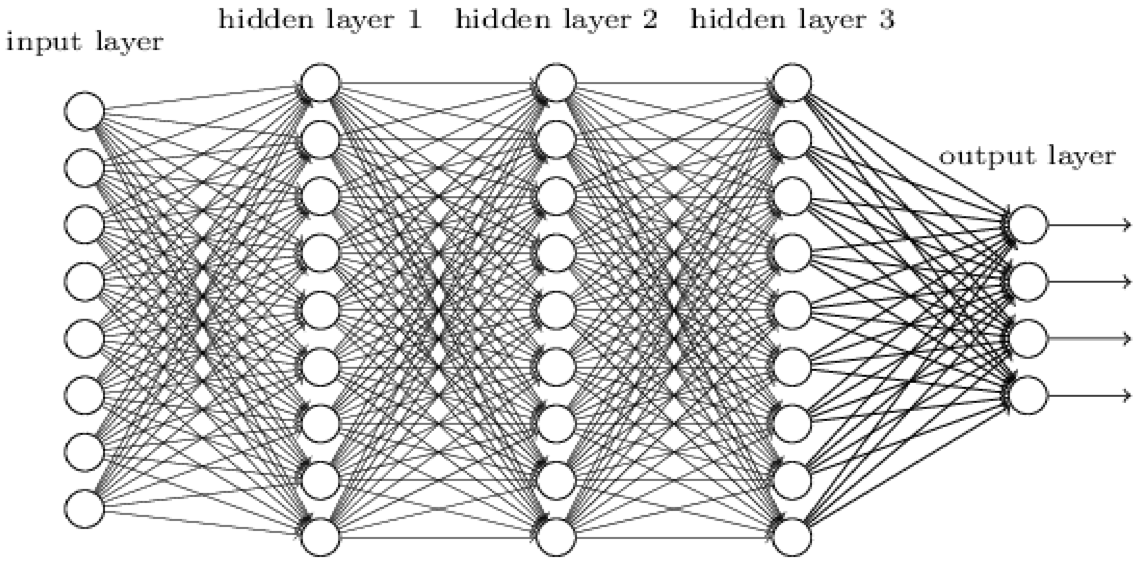




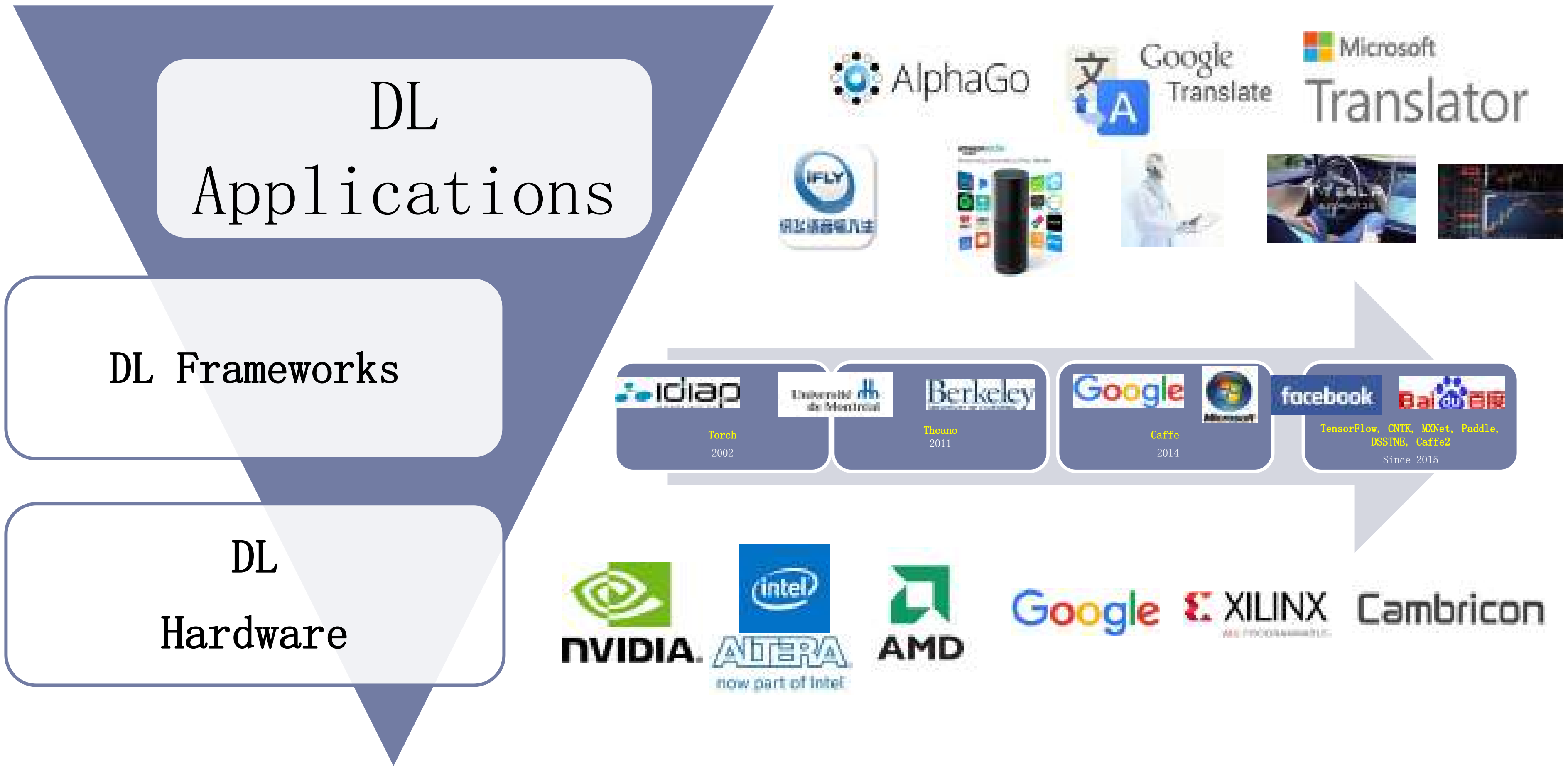
Performance Modeling and Evaluation of Distributed Deep Learning Frameworks on GPUs

Dr. Xiaowen Chu 褚晓文
Department of Computer Science, Hong Kong Baptist University 香港浸会大学计算机科学系

Outline

- Evolution of CPUs and GPUs in the Last Decade
- Understanding the GPU Performance
- Efficient Convolutions in Deep Learning
- Analysis of Distributed Training of Deep Neural Networks
- Benchmarking Results on Distributed DL Frameworks

Ecosystem of Deep Learning (DL)

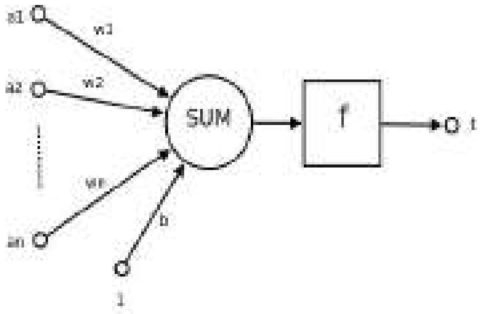


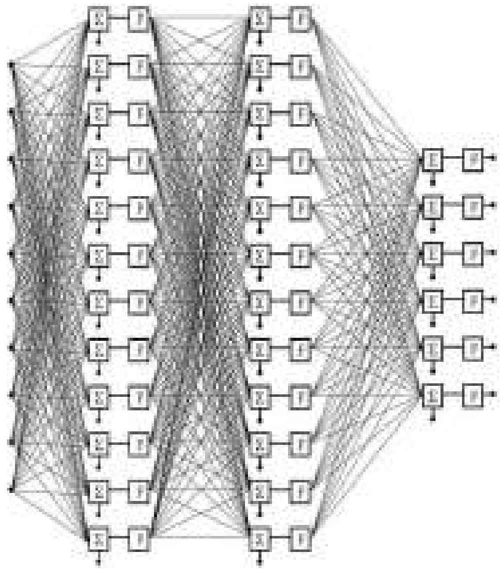
Cornerstone of Deep Learning: Computing

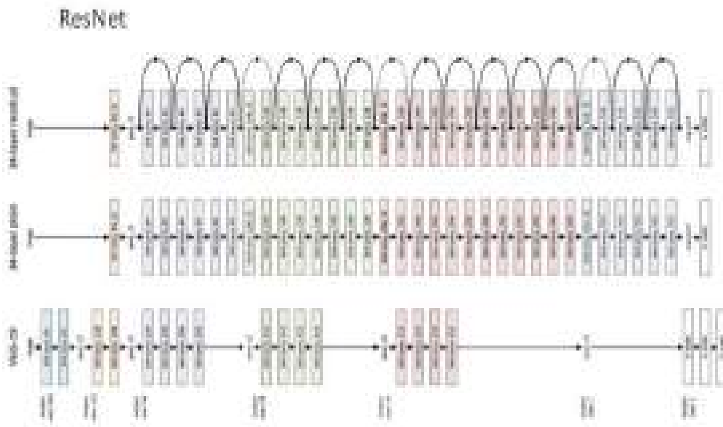
- Using Artificial Neural Networks as an example

A fully-connected neural networkDeep neural networks

A single neuron

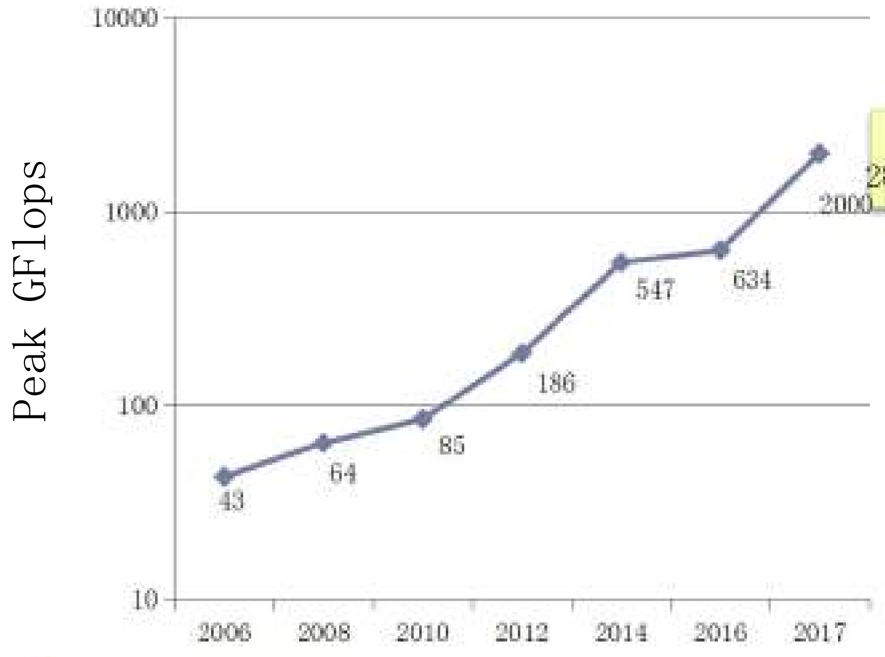






The Last Decade of Intel CPUs

- 50x of peak performance boost:
  - From ~40GFlops at 2006 to ~2000GFlops at 2017 (FP32)



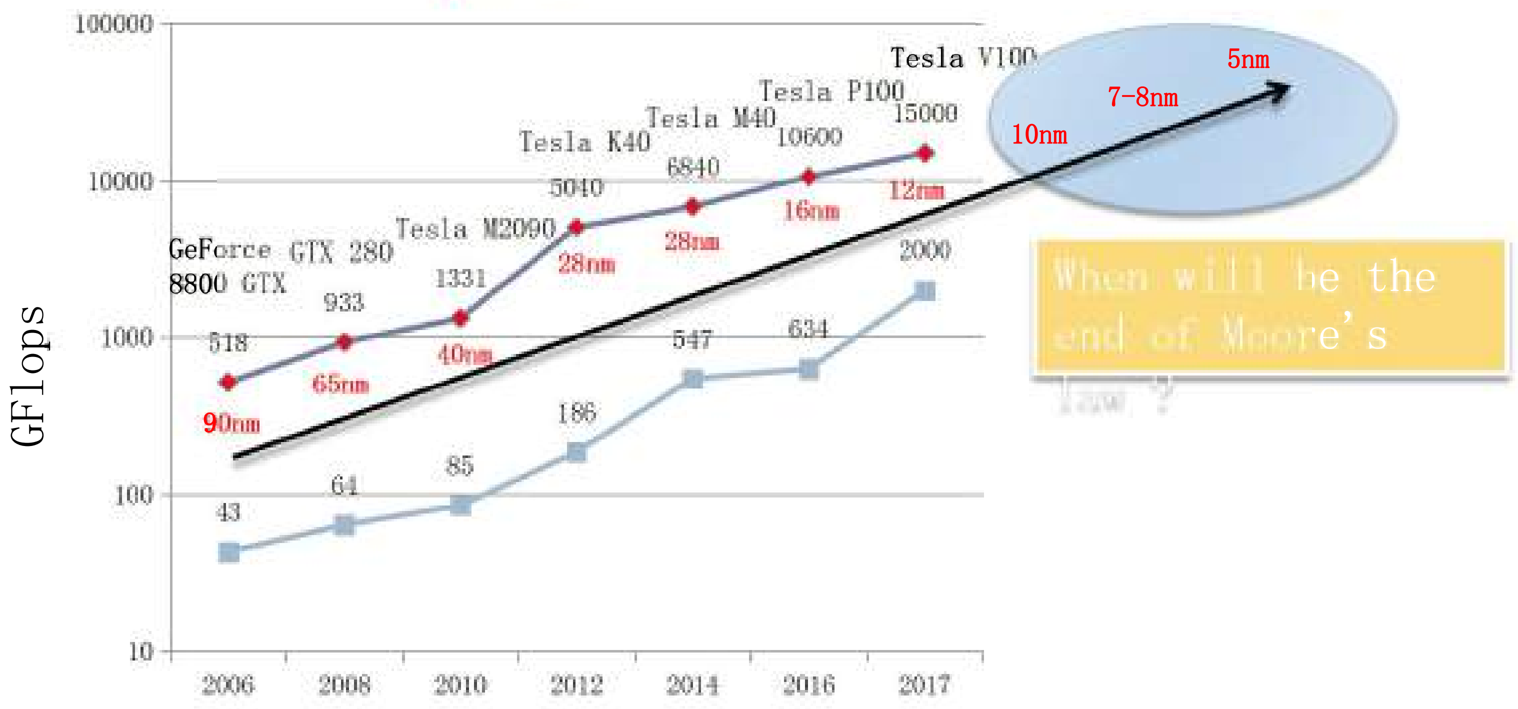
Xeon Platinum 8180
28-cores, 2.5GHz, AVX-512

# of cores: 7x
SIMD: 8x

Xeon X5355
4-cores, 2.66GHz, SSE3

The Last Decade of Nvidia GPUs

- 30x of peak performance boost, 17x more energy efficient:
  - From ~500GFlops at 2006 to ~15TFlops at 2017 (FP32)
  - From 128 cores to 5376 cores



The Last Decade of Nvidia GPUs

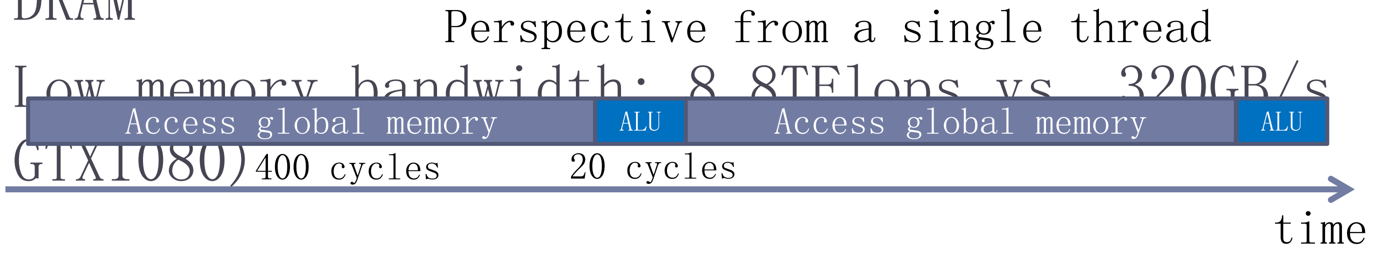
2006 2008 2011 2013 2015 2016 2017

Generation	G80	GT200	Fermi	Kepler	Maxwell	Pascal		Volta
Example	GeForce 8800 GTX	GTX 280	Tesla M2090	Tesla K40	Tesla M40	Tesla P100	GTX 1080	Tesla V100
FP32 Cores	128	240	512	2880	3072	3584	2560	5376
Peak FP32 GFLOPs	518	933	1331	5040	6840	10600	8873	15000
Memory	GDDR3	GDDR3	384-bit GDDR5	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2	256-bit GDDR5X	4096-bit HBM2
Memory Bandwidth (GB/sec)	57.6	141.7	177	288	288	549/732	320	900
TDP	155W	236W	225W	235W	250W	300W	180W	250W

Memory bandwidth: only ~15x

Challenge in GPU Computing

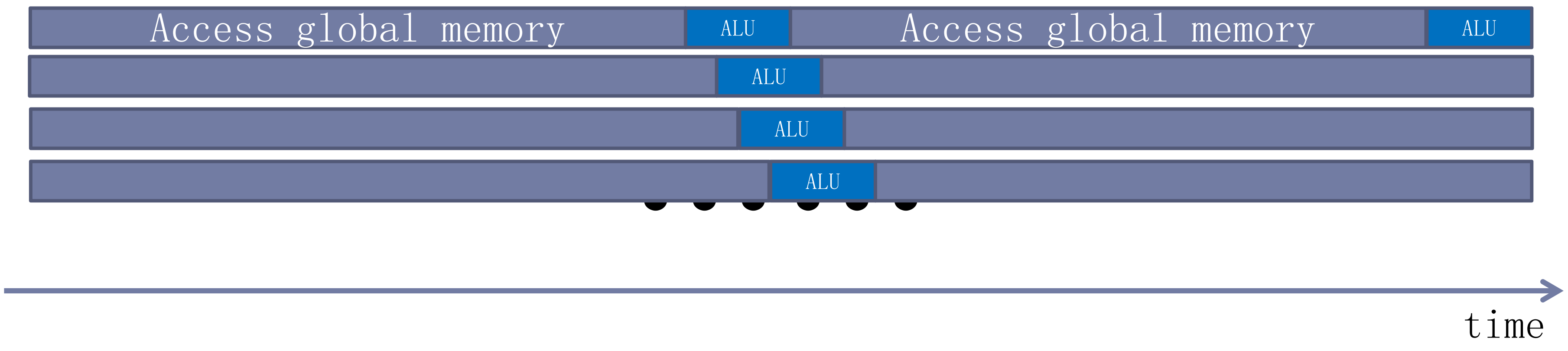
- Big gap between processing capacity and memory access
  - Computing is fast: each core (ALU) can finish one or two operations per cycle
    - 1000 cores x 1GHz = 1TFlops
    - But, one arithmetic operation requires two reads and one write
- Memory as a bottleneck
  - Long latency: hundreds of cycles to fetch data from DRAM
  - Low memory bandwidth: 8.8TFlops vs. 320GB/s (Nvidia GTX1080)



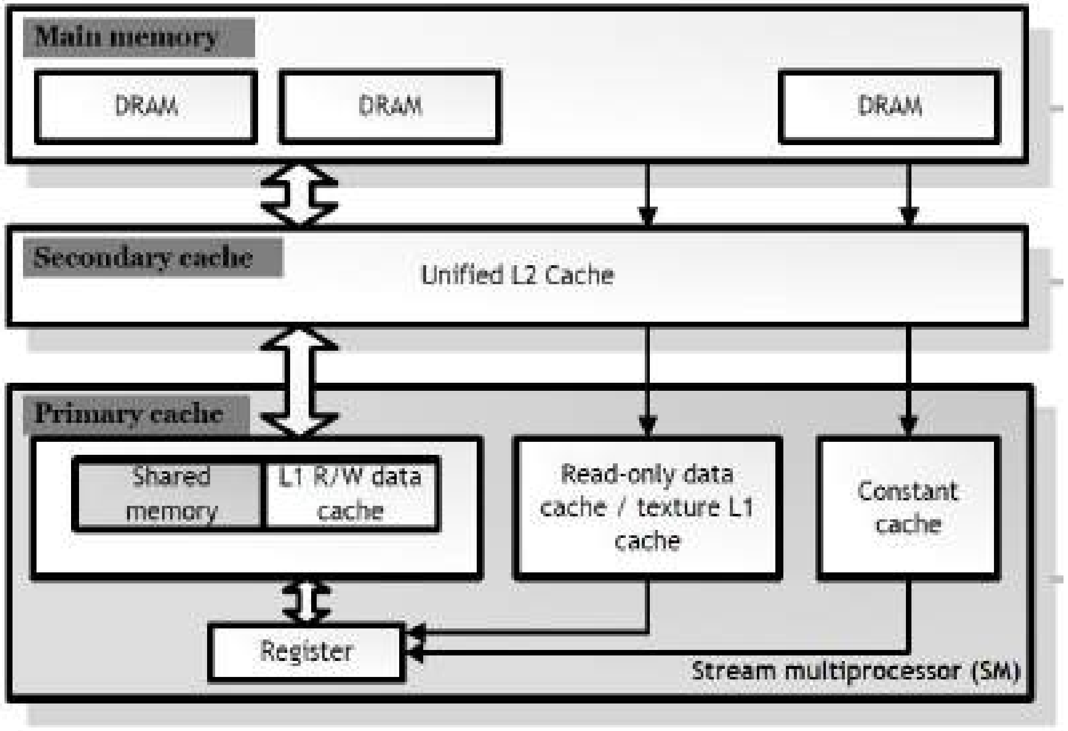
Solution 1: Multithreading

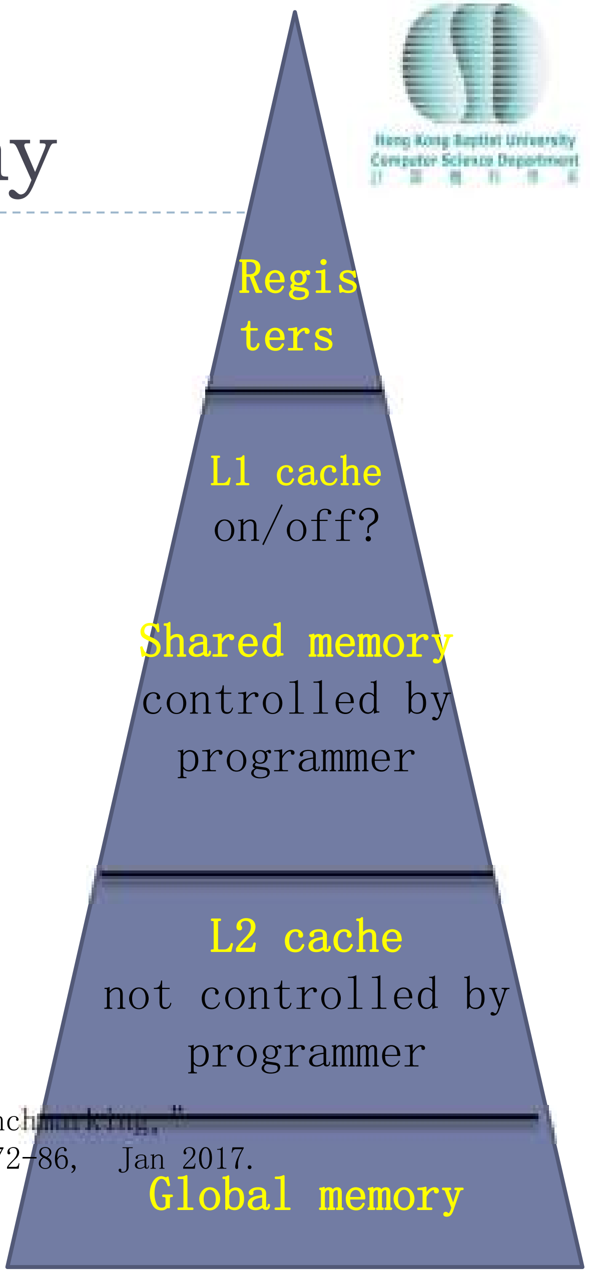
- Use multithreading to make the cores busy
  - At least hundreds of thousands of threads

Perspective from multiple threads (for each multistage ALU pipel



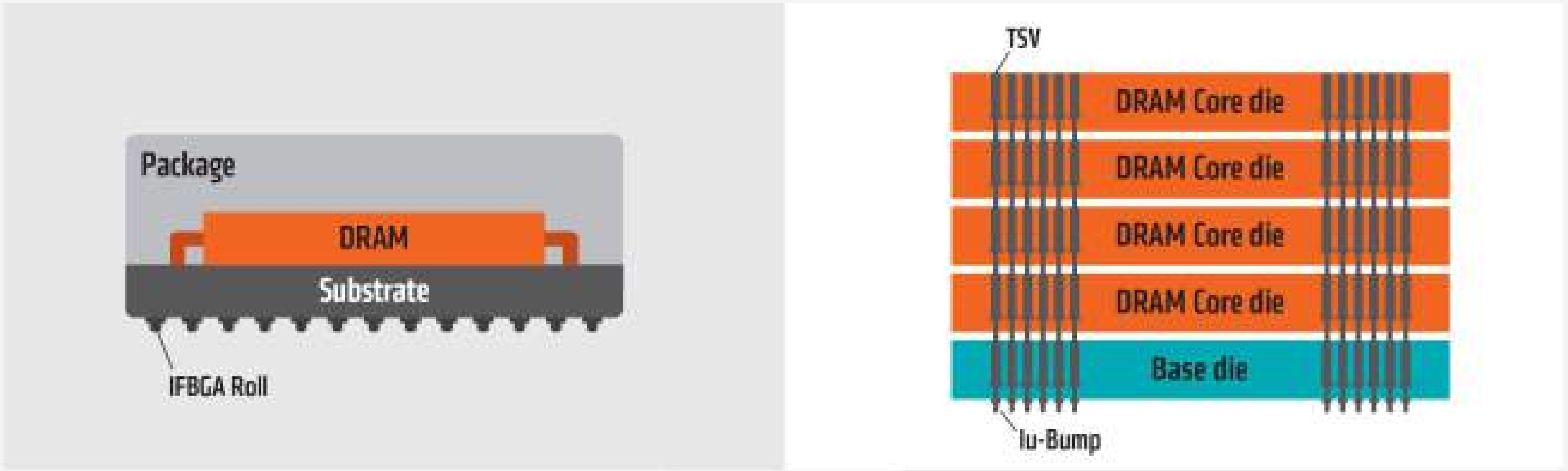
Solution 2: Memory Hierarchy

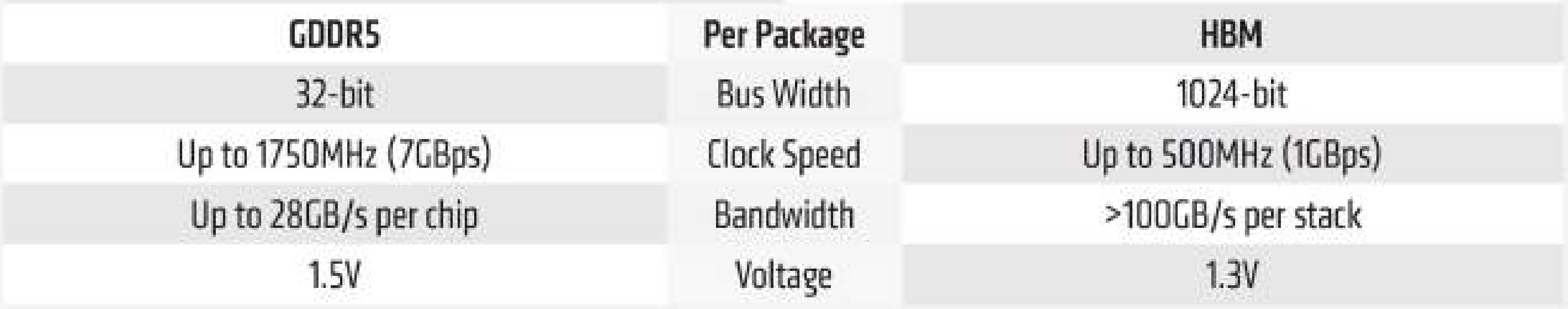



Registers
L1 cache on/off?
Shared memory controlled by programmer
L2 cache not controlled by programmer
Global memory

[Ref] X. Mei and X.-W. Chu, "Dissecting GPU Memory Hierarchy through Microbenchmarking," IEEE Transactions on Parallel and Distributed Systems, Vol. 28. No. 1, pages 72-86, Jan 2017.

Solution 3:
High Bandwidth Memory (HBM)





GDDR5	Per Package	HBM
32-bit	Bus Width	1024-bit
Up to 1750MHz (7GBps)	Clock Speed	Up to 500MHz (1GBps)
Up to 28GB/s per chip	Bandwidth	>100GB/s per stack
1.5V	Voltage	1.3V

[REF] https://www.amd.com/Documents/High-Bandwidth-Memory-HBM.pdf

Myth of GPU Performance

- A general question: what speedup can be achieved by GPUs (vs. CPU)?

Could be between 0.5x – 300x

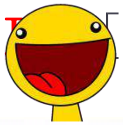
- The speedup depends on many factors:
  - What kind of CPU and GPU?
  - The nature of the application
    - Amdahl's law
    - Compute-bounded or bandwidth-bounded
  - How do you design the parallel algorithm?
  - How do you optimize the CPU/GPU code?

Hardware and software are equally important!

Roofline Model: Preliminaries

- An insightful visual performance model that considers two major performance constraints:
  - aggregated computational power of ALUs
  - memory bandwidth

- Operational intensity (OI): number of operations per byte of DRAM traffic. Assume single-precision operations,
  - Dot product: $z = \boldsymbol{xy}$, OI = 1/4
  - Dense Matrix-vector multiply: $\boldsymbol{y} = \mathbf{A}\boldsymbol{x}$, OI = 1/2
  - Dense Matrix-matrix multiply: $\mathrm{C}_{n\mathrm{x}n} = \mathrm{A}_{n\mathrm{x}n}\mathrm{B}_{n\mathrm{x}n}$, OI [1/4, n/6]

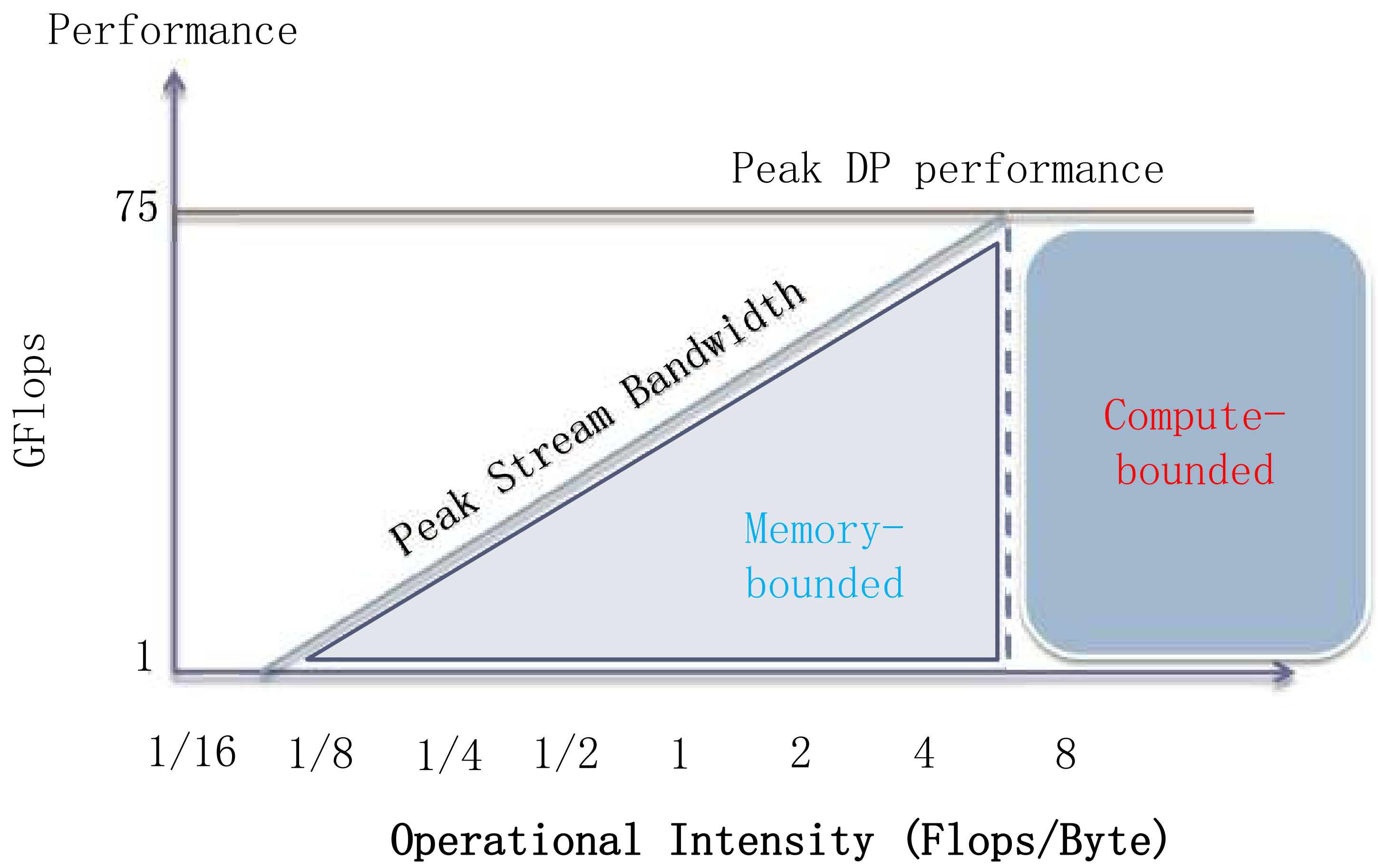


Given 8.8TFlops and 320GB/s, we need OI = 28 to fully utilize the GPU cores.

[Ref] S. Williams, et al. "Roofline: An Insightful Visual Performance Model for Multicore Architectures," Communications of the ACM, Vol. 52, No. 4, Apr 2009.

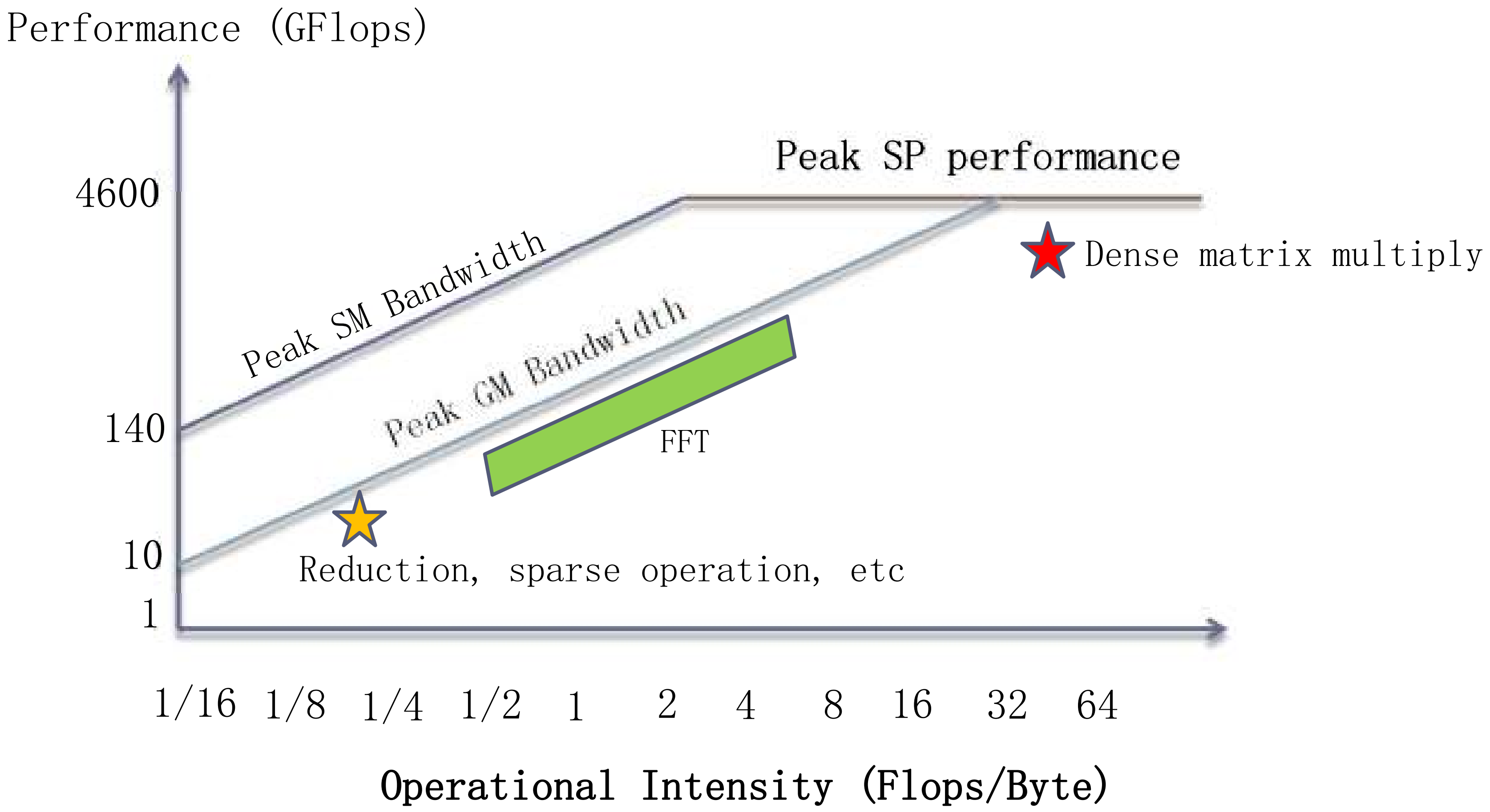
A Simple Roofline Model: PERF = MIN(FP_PERF, DRAM_BW × OI)

Intel Xeon E5345
FP_PERF = 75GFlops, DRAM_BW = 10GB/s



An Example of GPU Roofline Model

NVIDIA GTX980
FP_PERF = 4600GFlops, DRAM_BW = 156GB/s



Deep Learning Software Tools

	Torch	Caffe	CNTK	MXNet	TensorFlow	PaddlePaddle	PyTorch	Caffe2
First Release	2002	2014	2015	2015	2015	2016	2016	2017
Authors	S. Chintala et al.	Yangqing Jia, UC Berkeley	Microsoft	DMLC	Google	Baidu	Facebook	Yangqing Jia, Facebook
Github Followers	7496	21542	13248	12263	80253	5906	9746	6404
Major Version Updates	Github commits	Sep 14, 1.0rc Feb 15, 1.0rc2 Jan 16, 1.0rc3 Jan 17, 1.0rc4 Feb 17, 1.0rc5 Apr 17, 1.0rel	Dec 15, 1.0 Apr 16, 1.1 Jun 16, 1.5 Jul 16, 1.6 Sep 16, 1.7 Jun 17, 2.0 Jul 17, 2.1 Sep 15, 2.2 Nov 22, 2.3	Nov 15, 0.1 May 16, 0.7 Dec 16, 0.8 Jan 17, 0.9.3 May 17, 0.10 Sep 6, 0.11 Oct 31, 0.12	Nov 15, 0.5 … Jun 16, 0.9 Sep 16, 0.10 Nov 16, 0.11 Dec 16, 0.12 Feb 17, 1.0 Apr 17, 1.1 Jun 17, 1.2 Aug 17, 1.3 Oct 12, 1.4	Aug 16, 0.8b Dec 16, 0.9 May 17, 0.10	Sep 16, alpha Feb 17, 0.1.7 Feb 17, 0.1.8 Mar 17, 0.1.10 Apr 17, 0.1.11 May 17, 0.1.12 Aug 17, 0.2.0 Dec 5, 0.3.0	Apr 17, 0.6 Apr 17, 0.7 Jul 17, 0.8 Aug 17, 0.8.1

Challenges for Deep Learning Practitioners

- How to choose hardware?
  - Performance
  - Cost
  - Reliability
- How to choose software?
  - Performance
  - Usability
  - Community support
- How to optimize performance?

We try to address the performance aspect.

- We have been working in GPU computing since 2008.
- We worked on a distributed deep learning project during 2014-15.

Our Benchmarking Project http://dlbench.comp.hkbu.edu.hk/

- Started at May 2016
  - Core group members: SHI Shaohuai, WANG Qiang, XU Pengfei, WEI Lai
- Objectives
  - To evaluate the running time performance of **different deep learning software** on **different hardware platforms**
  - To identify performance bottleneck and propose solutions
- Features
  - Reproducible results: open-source at https://github.com/hclhkbu/dlbench
  - To be as fair as possible: reviewed by the community
  - Long-term: keep updating the software version and new hardware

Our Work in the 1.5 Year

Tested Software

- Caffe, Caffe-MPI
- CNTK
- TensorFlow
- Torch
- MXNet

Tested Hardware:

- CPUs: Intel CPU i7-3820, Intel Xeon E5-2630v4
- GPUs: Nvidia GTX980, GTX1080, GTX Titan X Pascal, Tesla K80, Tesla P40, Tesla P100
- A node with multiple GPUs
- Small GPU clusters

Tested Networks

- Fully-connected neural networks
- CNN: AlexNet, GoogLeNet, ResNet
- RNN: LSTM

Tested Data Sets

- Synthetic Data
- MNIST
- CIFAR10
- ImageNet

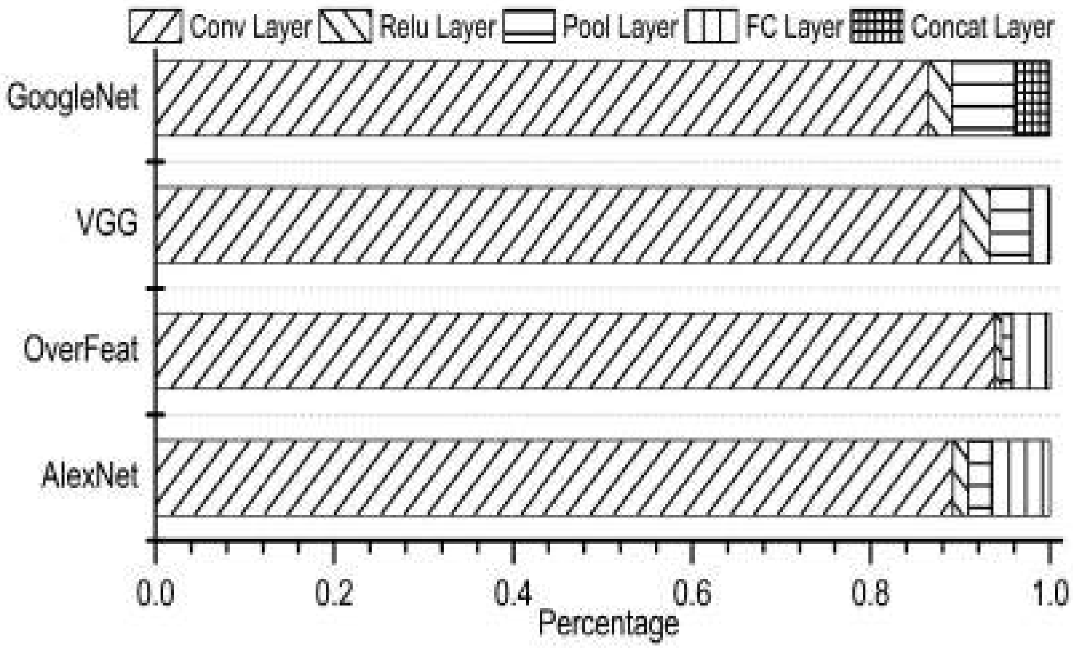
Key Operations in Deep Neural Networks

- Convolution layer: convolution operation
  - Three popular implementations
- Fully-connected l multiplication

[Ref 1] S. Chetlur, et al, cuDNN: Efficient Primitives for Deep Learning, arXiv 2014

[Ref 2] N. Vasilache, et al, Fast Convolutional Nets With fbfft: A GPU Performance Evaluation, ICLR 2015

[Ref 3] A. Lavin and S. Gray, Fast Algorithms for Convolutional Neural Networks, CVPR 2016



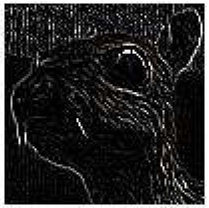
Understanding Convolution

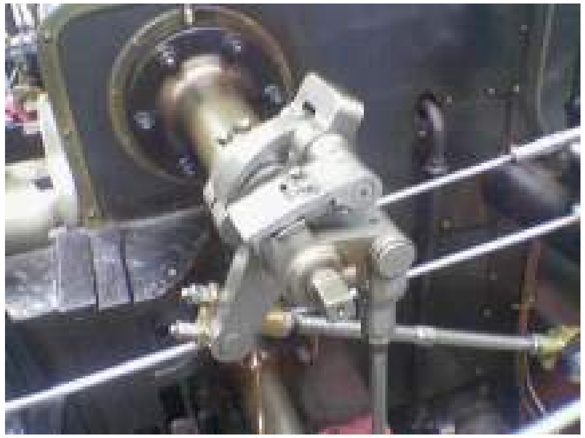
Input image

Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

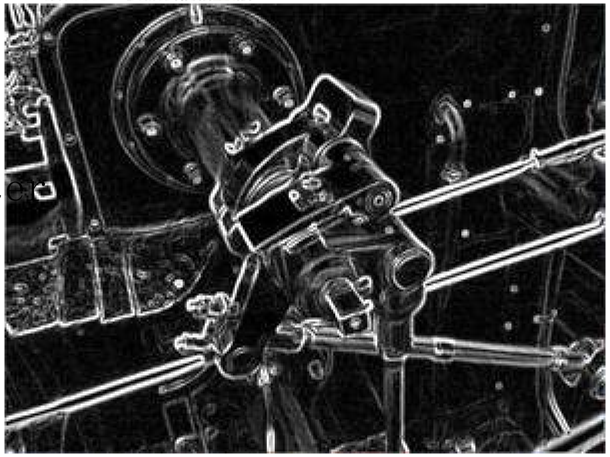
Feature map





Sobel filter

$\longrightarrow$



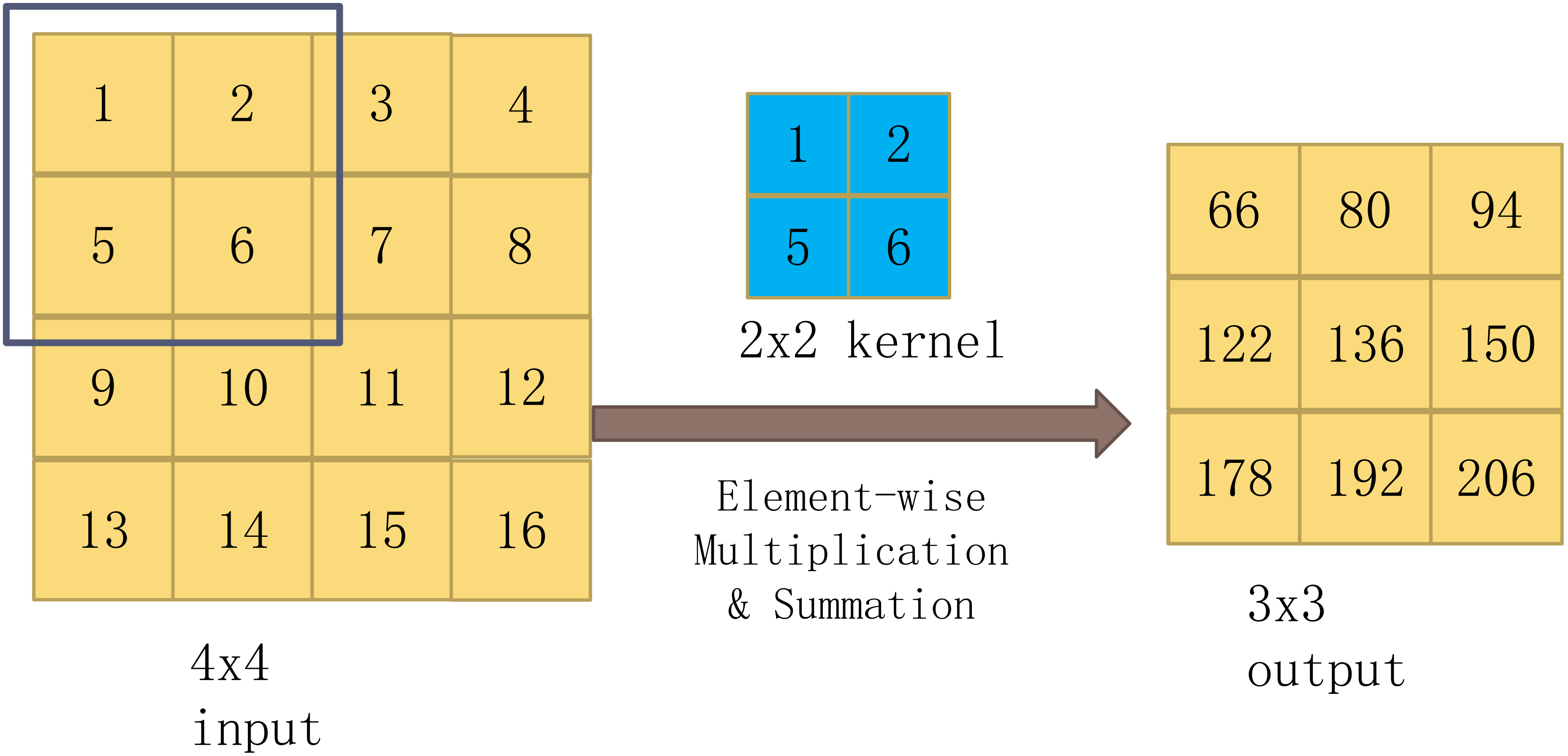
But, the design of convolution kernels (feature engineering) is very dif

⇩

Convolutional neural networks: learn a set of kernels automatically from tr

Image source: wikipedia.org

An Example of 2D Convolution



Three Implementations of Convolution

- Conventional matrix-multiplication based
  - Well-known in signal processing
  - Simple to understand and implement
- Discrete Fourier Transform
  - **Convolution theorem**: a convolution of two discrete signals can be performed by multiplication in the frequency domain
  - Rely on efficient implementations of FFT
  - Very efficient for medium and large kernels
- Winograd’ s minimal filtering algorithm
  - Originally proposed in 1960’ s – 1980’ s, applied to DNN in 2016
  - Very efficient for small kernels
  - FPGA solutions are available now

Matrix-Multiplication based Convolution

$$\begin{bmatrix}1&2\\3&4\end{bmatrix} \otimes \begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} = \begin{bmatrix}37&47\\67&77\end{bmatrix}$$

2x2 kernel

3x3 input

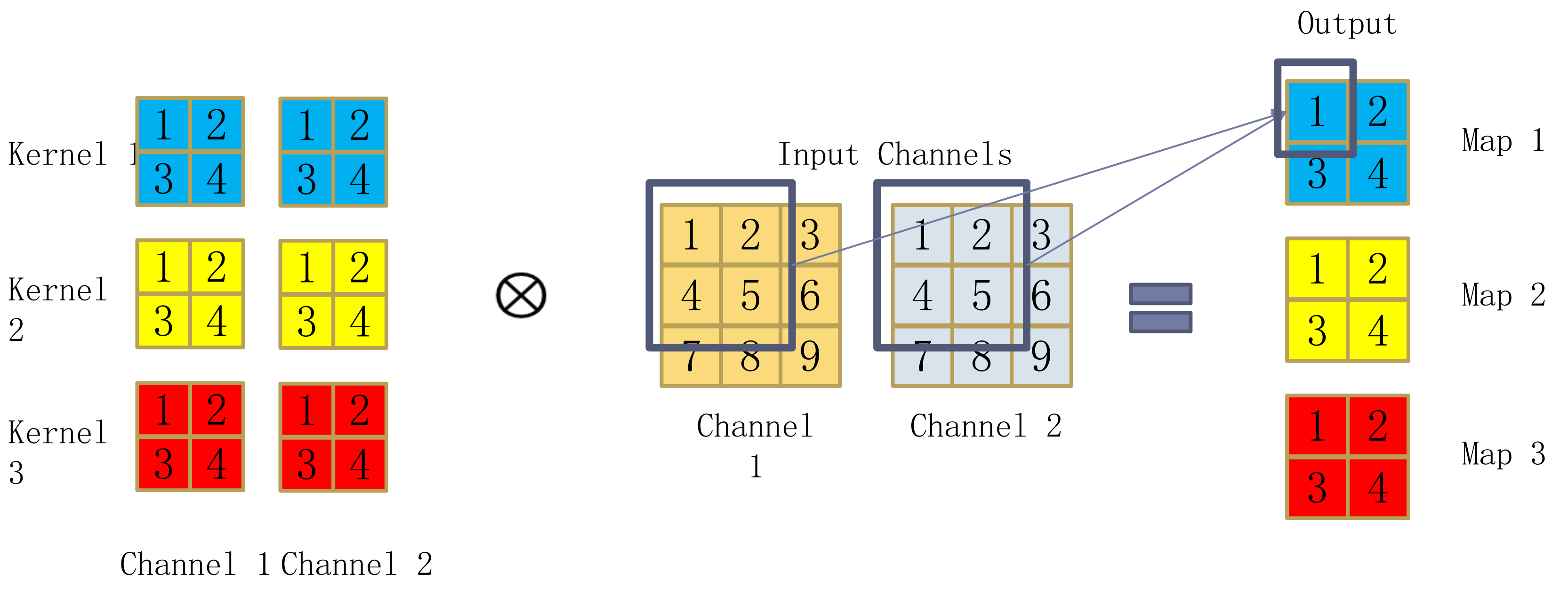
2x2 output

Re-arrange data structure with necessary data repetition.

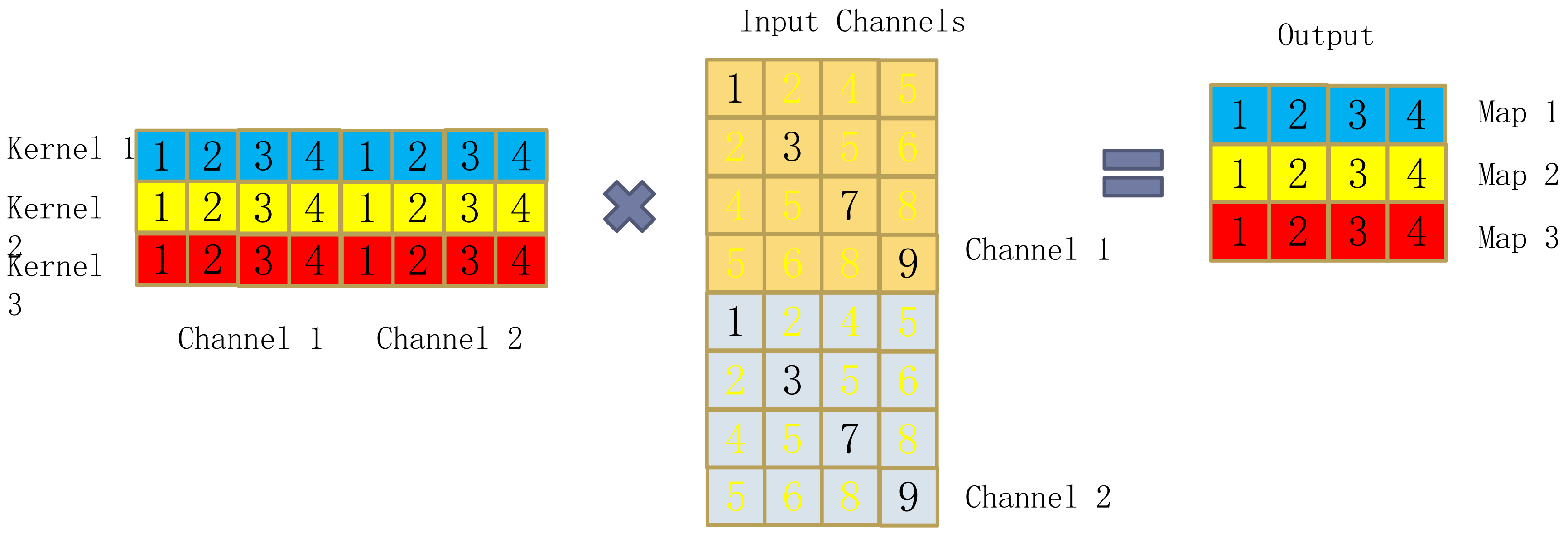
$$\begin{bmatrix}1&2&3&4\end{bmatrix} \times \begin{bmatrix}1&2&4&5\\2&3&5&6\\4&5&7&8\\5&6&8&9\end{bmatrix} = \begin{bmatrix}37&47&67&77\end{bmatrix}$$

Toeplitz Matrix

Matrix-Multiplication based Convolution

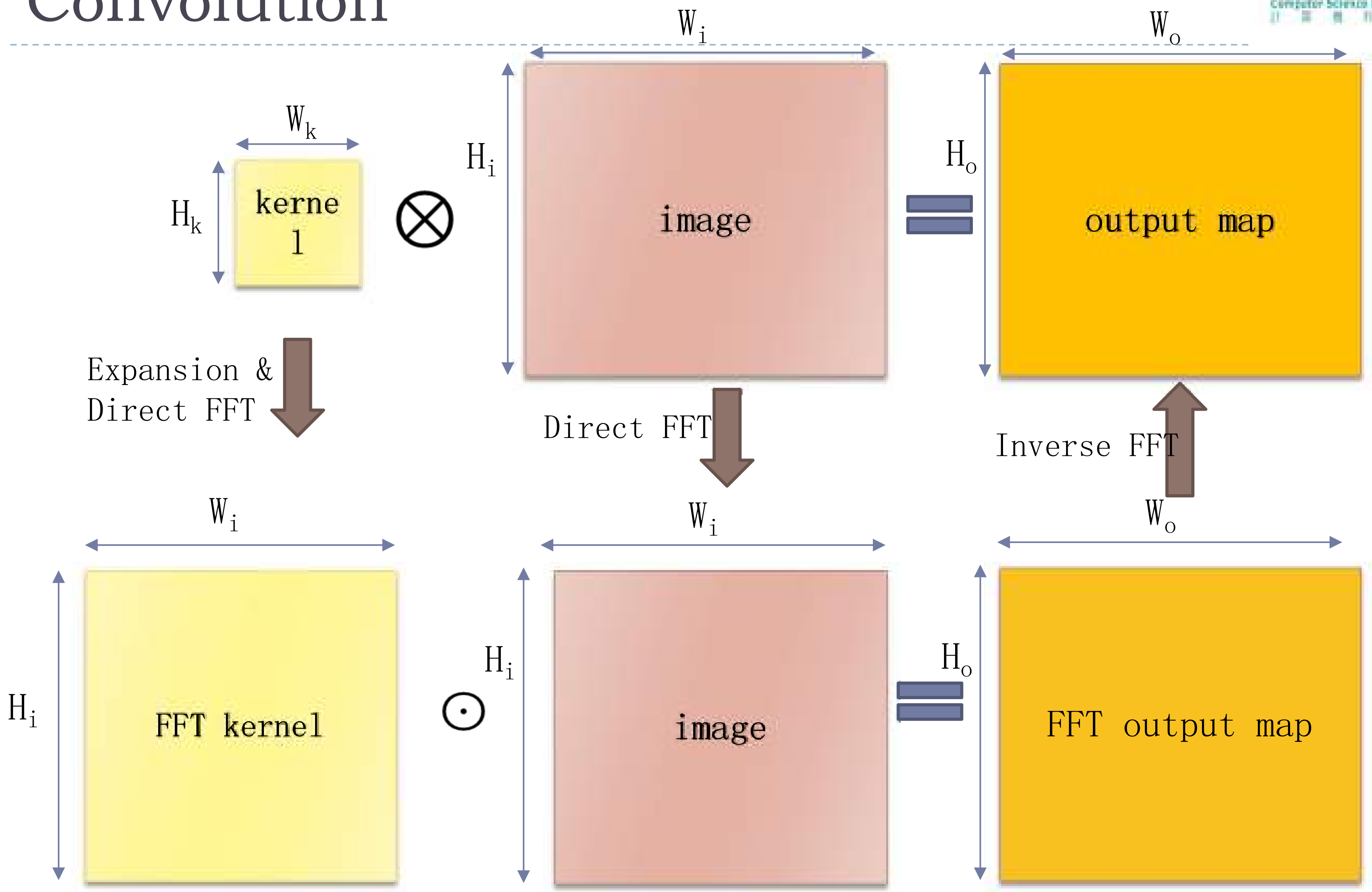


Matrix-Multiplication based Convolution



- The computational complexity is the same as straightforward convolution
- Benefit from existing high-performance GEMM library
  - Traditional CPU (multi-threading): OpenBLAS, Intel MKL, etc
  - GPU : cuBLAS, cuDNN, etc

Fast Fourier Transform (FFT) based Convolution



Fast Fourier Transform (FFT) based Convolution

- Generally,
  - $H_o = H_i - H_k + 1$
  - $W_o = W_i - W_k + 1$

- H_i: height of input image
- W_i: width of input image
- H_k: height of kernel
- W_k: width of kernel
- H_o: height of output image
- W_o: width of output image

- Computational Complexity
  - Straightforward implementation: $O(H_o \times W_o \times H_k \times W_k)$
  - FFT-based: $O(H_i \times W_i \times log_2(H_i \times W_i))$
- Typically, the input image is much larger than the kernel
  - Kernel size increases -> computational efficiency grows!

[REF]
1. Podlozhnyuk, Victor. "FFT-based 2D convolution." NVIDIA white paper (2007).
2. Mathieu, Michael, Mikael Henaff, and Yann LeCun. "Fast training of convolutional networks through FFTs." arXiv:1312.5851 (2013).
3. N. Vasilache, et al, Fast Convolutional Nets With fbfft: A GPU Performance Evaluation, ICLR 2015.

Winograd-based Convolution

- Reduce the number of multiplication operations
- E.g., in 1D convolution

In 2D convolution F(2x2, 3x3):
36 muls → 16 muls

$$\begin{bmatrix} d_0 & d_1 & d_2 & d_3 \end{bmatrix} \otimes \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{matrix} d_0 g_0 + d_1 g_1 + d_2 g_2 \\ d_1 g_0 + d_2 g_1 + d_3 g_2 \end{matrix}$$

6 multiplications + 4 additions

$$F(2,3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix}$$

$$m_1 = (d_0 - d_2) g_0 \qquad m_2 = (d_1 + d_2) \frac{g_0 + g_1 + g_2}{2}$$

$$m_4 = (d_1 - d_3) g_2 \qquad m_3 = (d_2 - d_1) \frac{g_0 - g_1 + g_2}{2}$$

4 multiplications + 12 additions

[REF] Lavin, Andrew, and Scott Gray. "Fast algorithms for convolutional neural networks." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

Importance of Performance Modeling

- to understand the overall performance
- to identify the performance bottleneck
- to optimize the software
  - which platform or library?
- to optimize the hardware
  - CPU, GPU, storage, network

Mini-batch SGD

- 1) Read a mini-batch of data from disk to memory: t_{io}
- 2) Transfer the data from the CPU memory to the GPU memory: t_{h2d}
- 3) Feed forward operation: t_f
- 4) Backward propagation: t_b
- 5) Update model parameters: t_u

$$\boxed{t_{iter} = t_{io} + t_{h2d} + t_f + t_b + t_u}$$

Multiple GPUs: Synchronous SGD

Algorithm 1 S-SGD

```
1: procedure S-SGD((parameters, data, $N$))
2:     for each worker $i \in \{1, 2, ..., N\}$ do
3:         $\nabla g_i \leftarrow SGD(parameters, \frac{data}{N})$
4:     Aggregate from all workers: $\nabla g \leftarrow \frac{1}{N}\sum_{i=1}^{N} \nabla g_i$
5:     Return $\nabla g$
```

$$t_{iter} = t_{io} + \boxed{t_{h2d} + t_f + t_b + t_u} + t_{comm}$$

$$= t_{io} + t_{gpu} + t_{comm}$$

Optimization Opportunities

- I/O time t_{io}:
  - t_{io} depends on training data size, storage system (HDD, SSD, RAID, NFS), data format (compressed or not?), CPU
  - Use multi-threading to overlap t_{io} with the previous t_{iter}
- Communication time t_{comm}:
  - In the backword propagation, the gradient communications of layer i can be overlapped with

Since the GPU speed grows much faster than IO and network, $\mathbf{t_{io}}$ and $\mathbf{t_{comm}}$ become more important in performance optimization.

Speedup Analysis

- If communication time can be hidden by computing time

$$\text{SPEEDUP} = \frac{N_g(t_{io\_1}+t_{gpu})}{t_{io\_n_g}+t_{h2d}+t_f+t_b+t_{comm}^{(1)}+t_u^{(1)}}$$

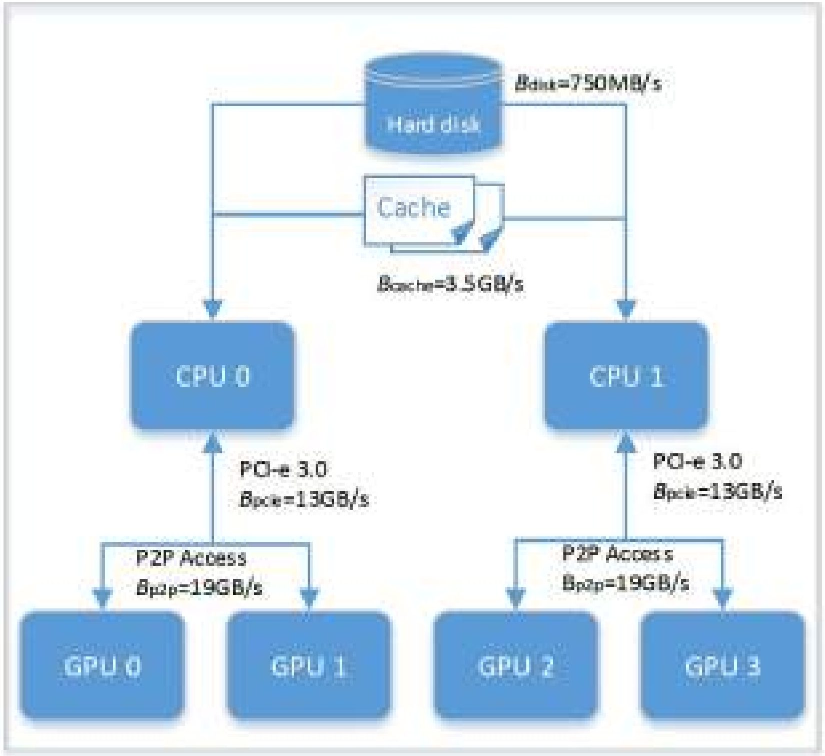
- Otherwise:

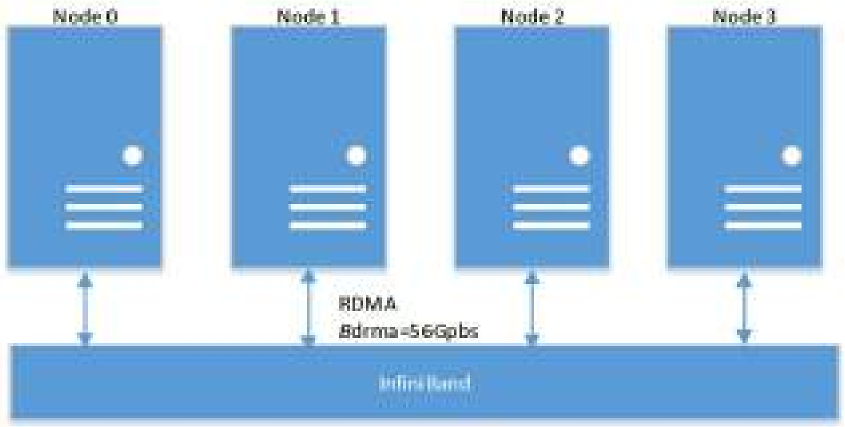
$$\frac{N_g(t_{io\_1}+t_{gpu})}{t_{io\_n_g}+t_{h2d}+t_f+\sum_{i=C}^{L} t_{comm}^{(i)}+\sum_{i=1}^{C-1} t_b^{(i)}+t_{comm}^{(1)}+t_u^{(1)}}$$

Different Optimization Strategies

- CNTK 2.0
  - Use non-pageable memory to reduce t_{h2d}
  - Use Nvidia NCCL for all-reduce communication
  - Backward propagation is not pipelined with gradient aggregation
- MXNet
  - Backward propagation is pipelined with gradient aggregation
- TensorFlow
  - Supports both PS mode and all-reduce mode
- Caffe-MPI
  - Use non-pageable memory to reduce t_{h2d}
  - Use Nvidia NCCL 2.0 for all-reduce communication
  - Backward propagation is pipelined with gradient aggregation

Our Testbed





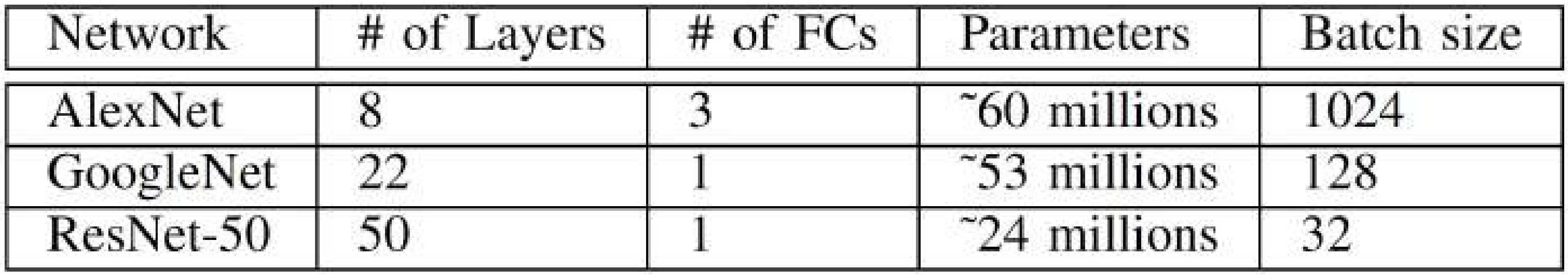
The cluster has 4 nodes

Each node has 4 Nvidia Tesla P40 cards

DL Frameworks and Networks

DL Software	Version	cuDNN
Caffe-MPI	2.0	v6
CNTK	2.0	v6
MXNet	0.10.0	v6
TensorFlow	1.2.1	v6

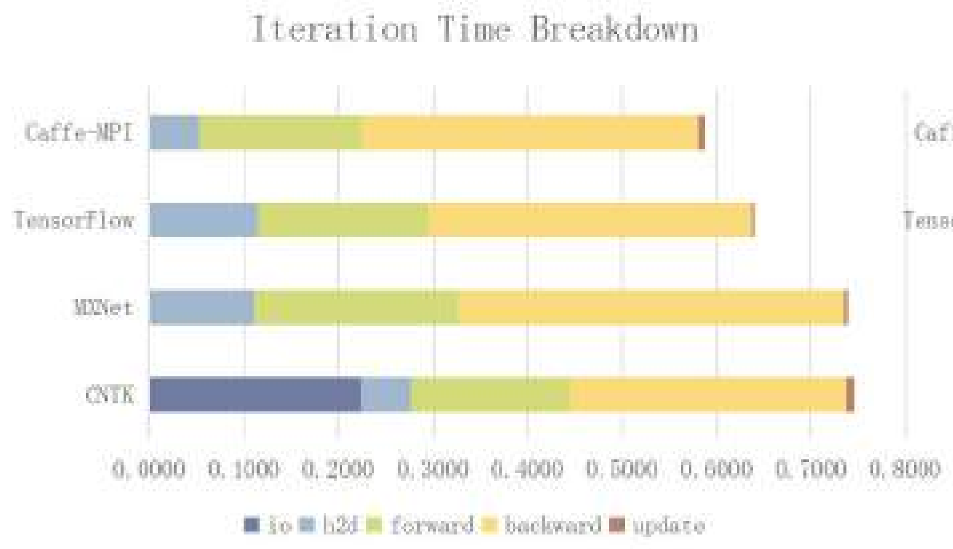
IMAGENET



Network	# of Layers	# of FCs	Parameters	Batch size
AlexNet	8	3	~60 millions	1024
GoogleNet	22	1	~53 millions	128
ResNet-50	50	1	~24 millions	32

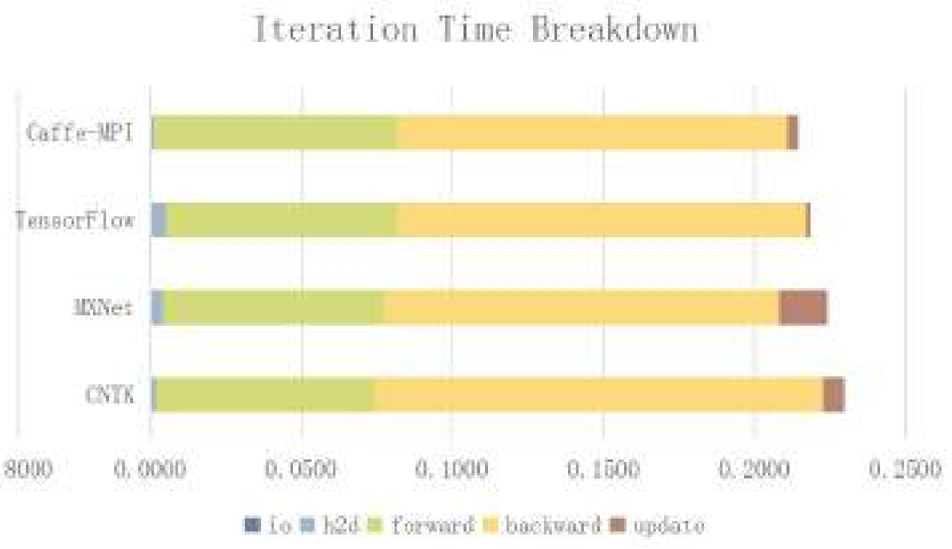
Single GPU Performance

AlexNet



Batch size = 1024

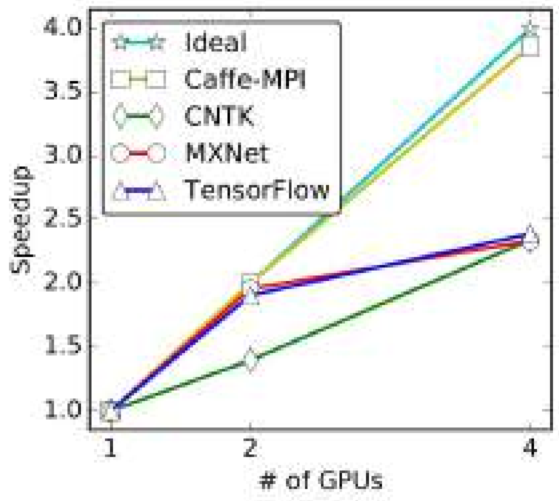
ResNet-50



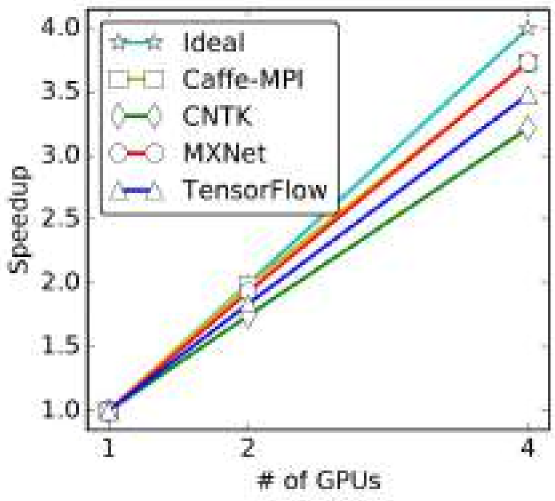
Batch size = 32

Multiple-GPU Benchmarking Results

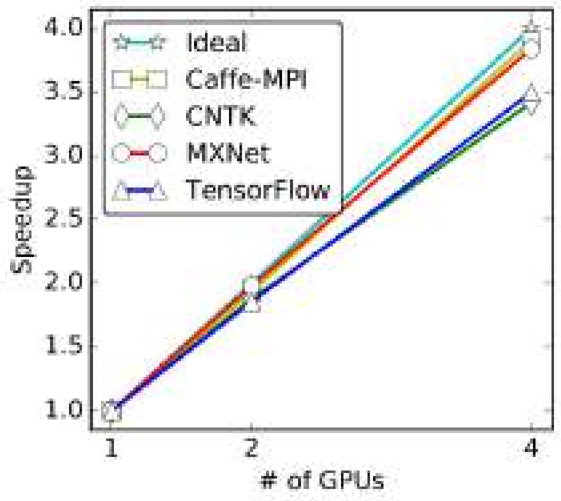
Batch size = 1024/2048/4096Batch size = 128/256/512   Batch size = 32/64/128



(a) AlexNet



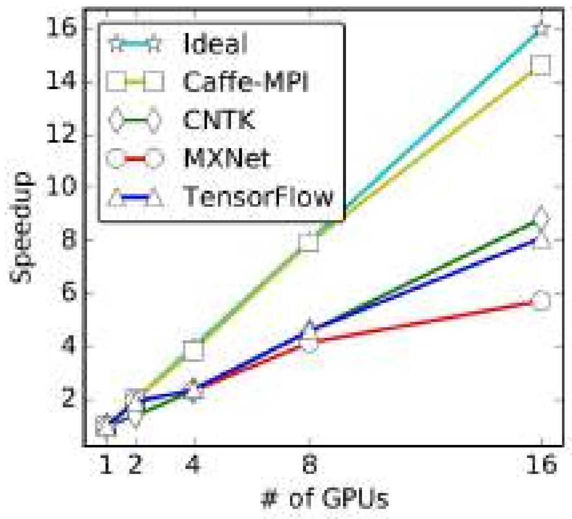
(b) GoogleNet



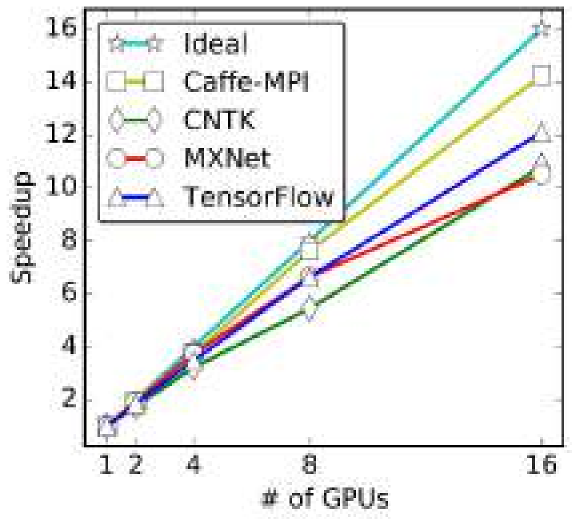
(c) ResNet-50

In our test, the batch size is proportional to the number of GPUs.
So when using more GPUs, the IO performance becomes a bottleneck on Alex
PCIe communication time is less than 0.1 second, and can overlap with co

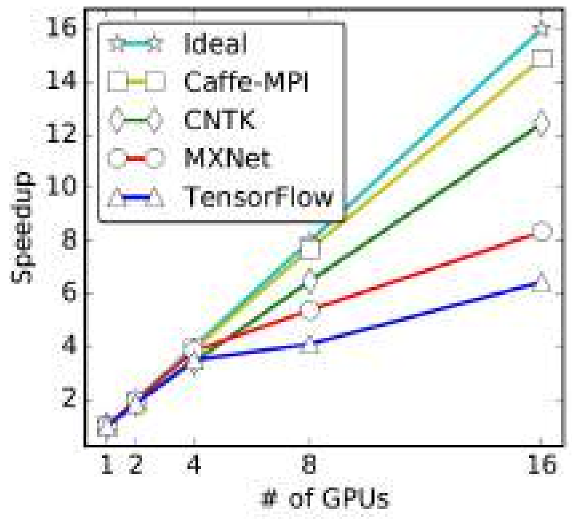
Cluster Benchmarking Results



(a) AlexNet



(b) GoogleNet



(c) ResNet-50

In a GPU cluster, the network communication may significantly affect the

It is important to overlap network communications with gradient calculati

Conclusion

- For single GPU, it is important to understand the performance of different implementations of convolution
- For multiple GPUs within a single node, synchronous SGD in general works well with 4 GPU cards
  - Scalability could be limited by disk I/O performance
- For GPU cluster, the scalability can be limited by network performance
  - Try to overlap the communication time with computing time layer by layer
  - Try Nvidia NCCL 2.0

[Ref] S. Shi and X.-W. Chu, “Performance Modeling and Evaluation of Distributed Deep Learning Frameworks on GPUs,” arXiv:1711.05979.

Acknowledgement