SAPPORO JAPAN MAY 26-29 2019

ISCAS 2019

IEEE International Symposium on Circuits and Systems

IEEE CAS

ISCAS 2019 Tutorial

Machine & Deep Learning for Edge-Cloud Computing Systems

Baoxin Li & Fengbo Ren

Computer Science & Engineering

Arizona State University

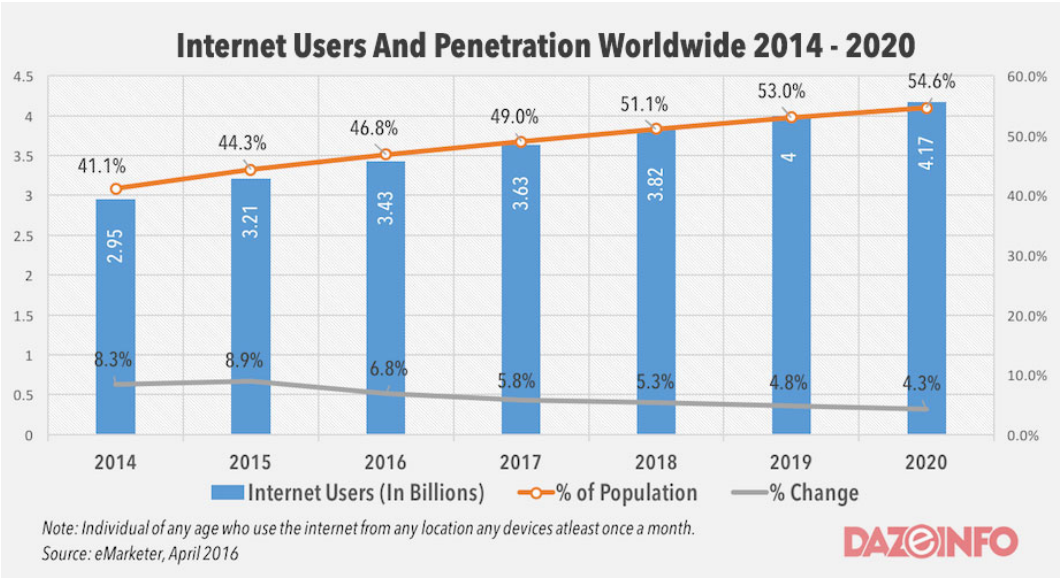
Module 3:
Enabling Deep Learning for Edge Computing

- The Era of IoT & Edge Computing
- FPGAs for Edge Computing
- The Edge Deployment Challenge of Deep Learning
- Techniques for Reducing Computational Complexity of Deep Neural Networks (DNNs)

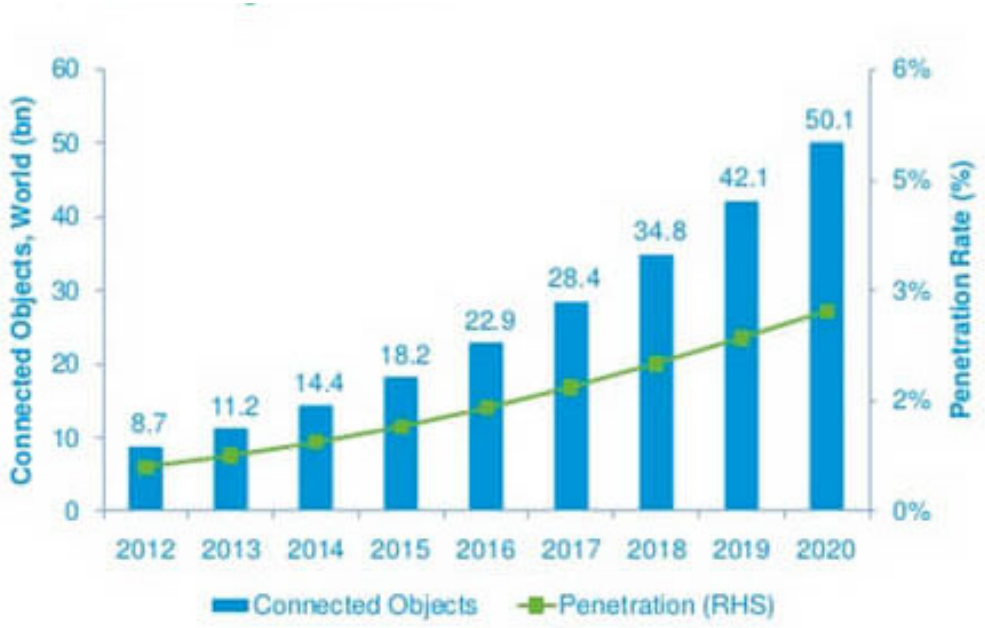
The Era of IoT & Edge Computing

Today's Internet is Internet-of-things (IoT)

Connected Human Users



Connected Things

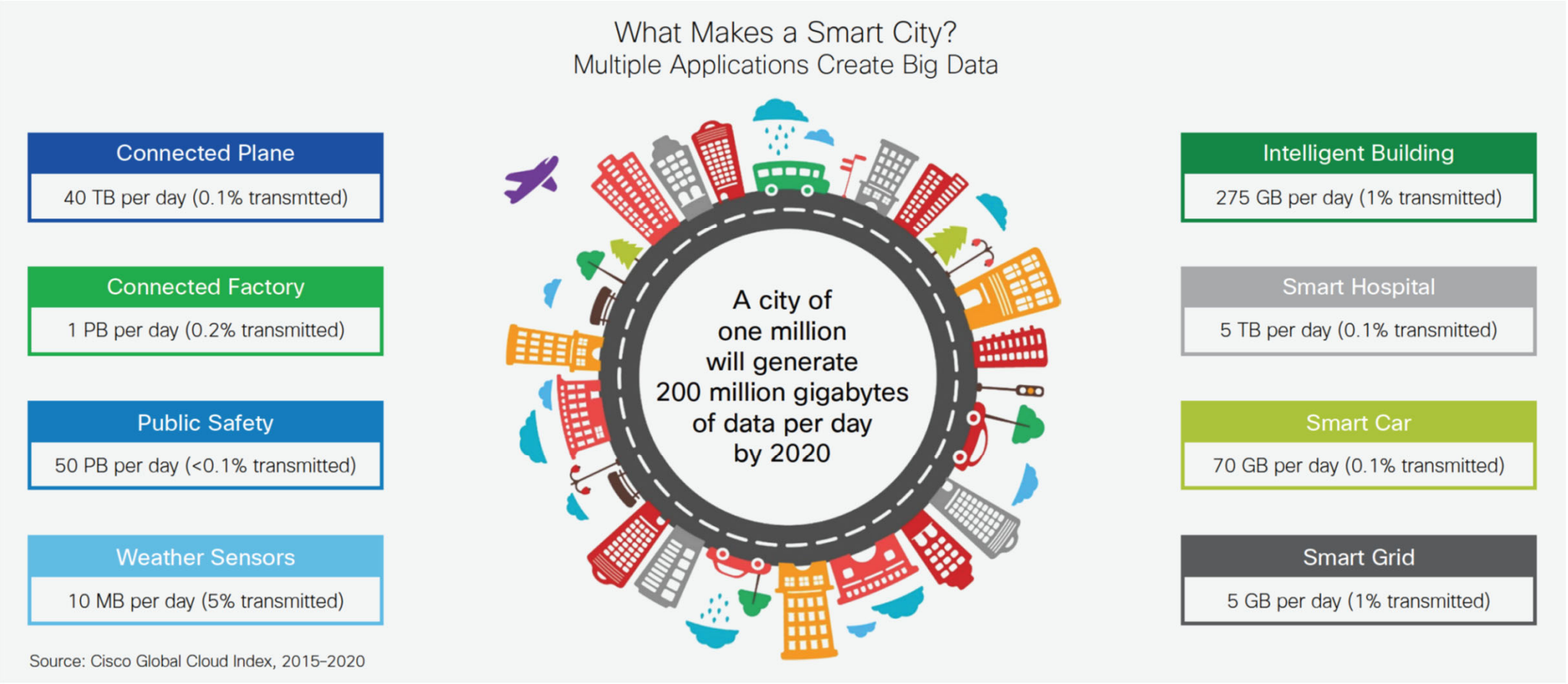


4.17 Billion by 2020 ⟷ 50.1 Billion by 2020

12x more things than human users

Data Explosion in The IoT Era

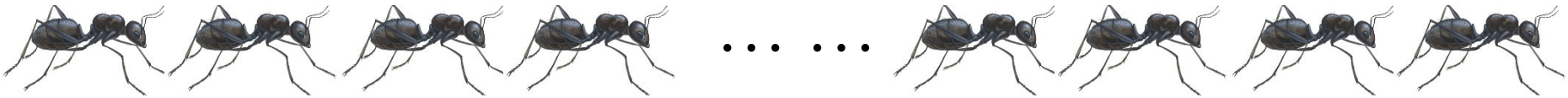
By 2020, IoT will create **600 Zetta Bytes** of data per year



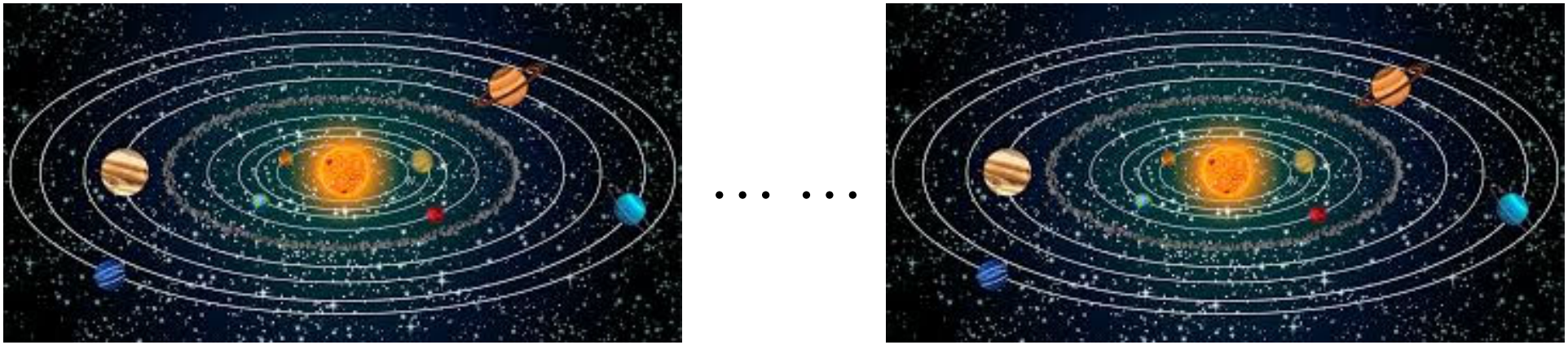
"Cisco global cloud index: Forecast and methodology 2015–2020." *White paper* (2016).

How Big is 600 ZB of Data?

600 Zetta ants in a row



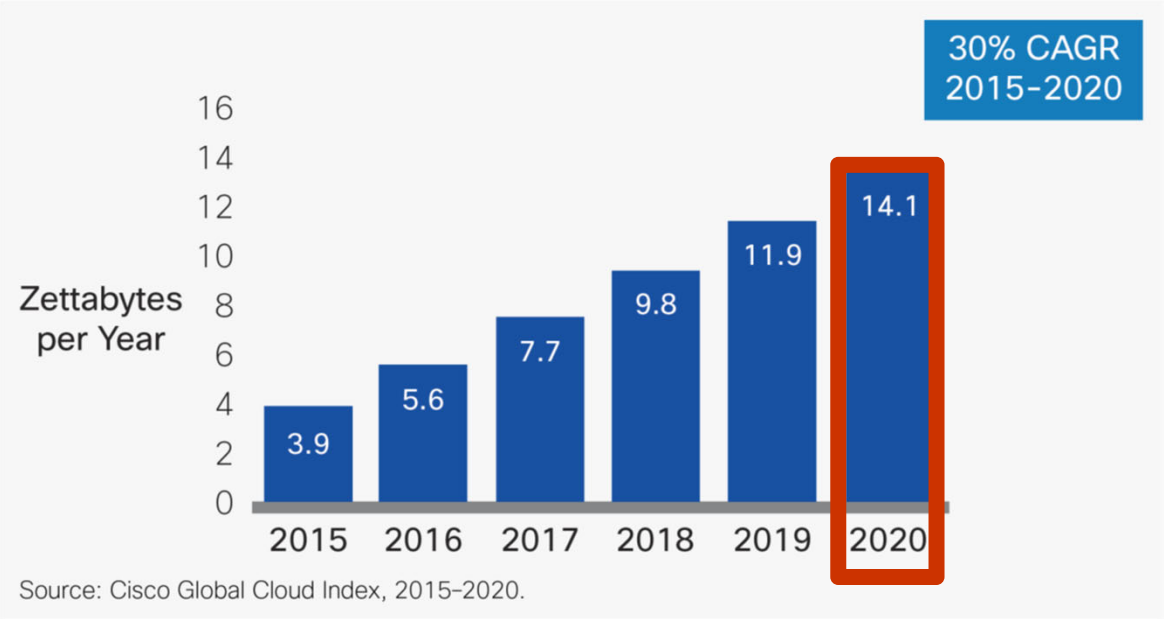
=

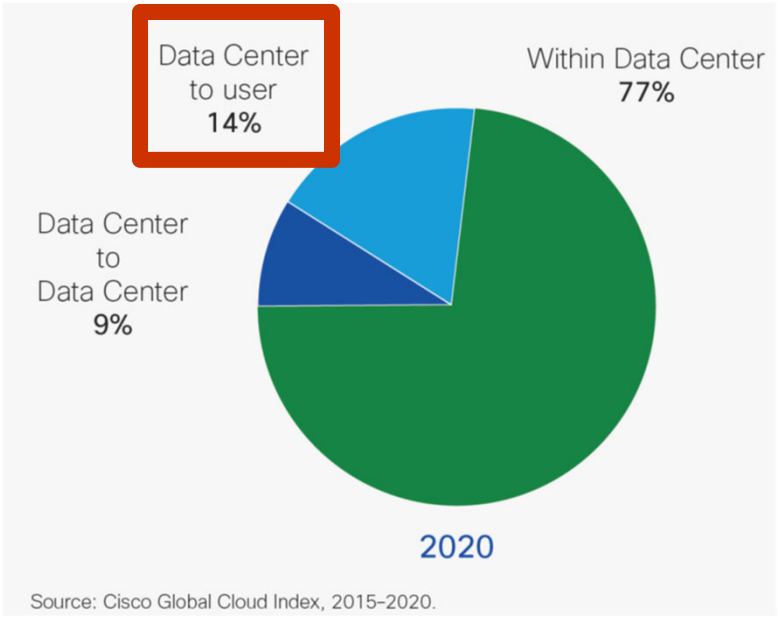


12 Million solar systems in a row

380,521 light years

Cloud Data Center IP Traffic Growth



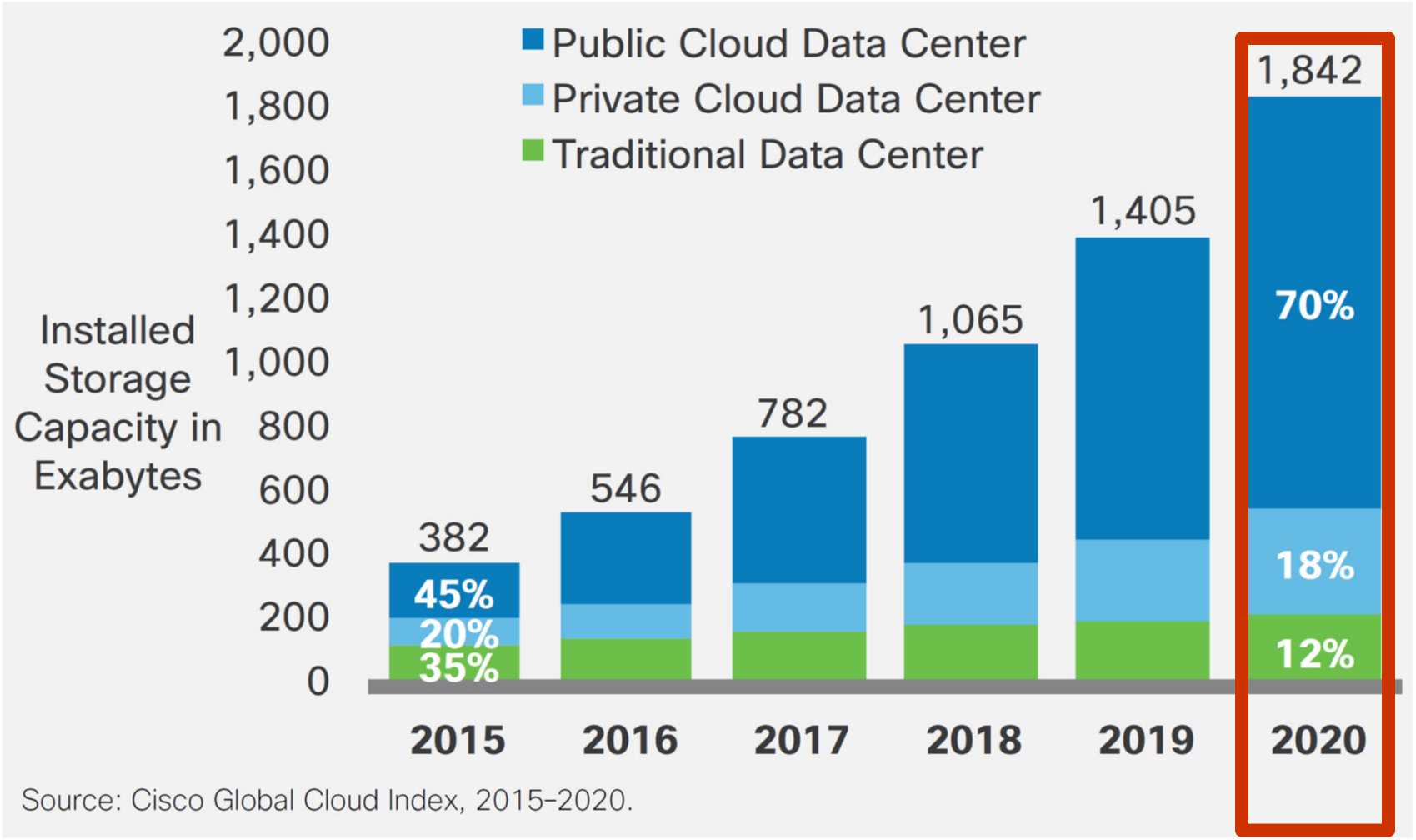


Cloud-User IP Traffic Bandwidth by 2020: **14.1 ZB / yr.**

2 ZB / yr. (14%) is for data center to end users

"Cisco global cloud index: Forecast and methodology 2015–2020." *White paper* (2016).

Global Data Center Storage Capacity Growth



Global Data Center Storage Capacity by 2020: **1.8 ZB**

1.62 ZB (88%) is for Cloud Data Centers

"Cisco global cloud index: Forecast and methodology 2015–2020." *White paper* (2016).

By 2020, Cloud Will Fall Short by 2 Orders of Magnitude in Handling IoT Data

600 ZB

IoT data generated /yr.

2 ZB

End-user-to-cloud IP traffic bandwidth / yr.

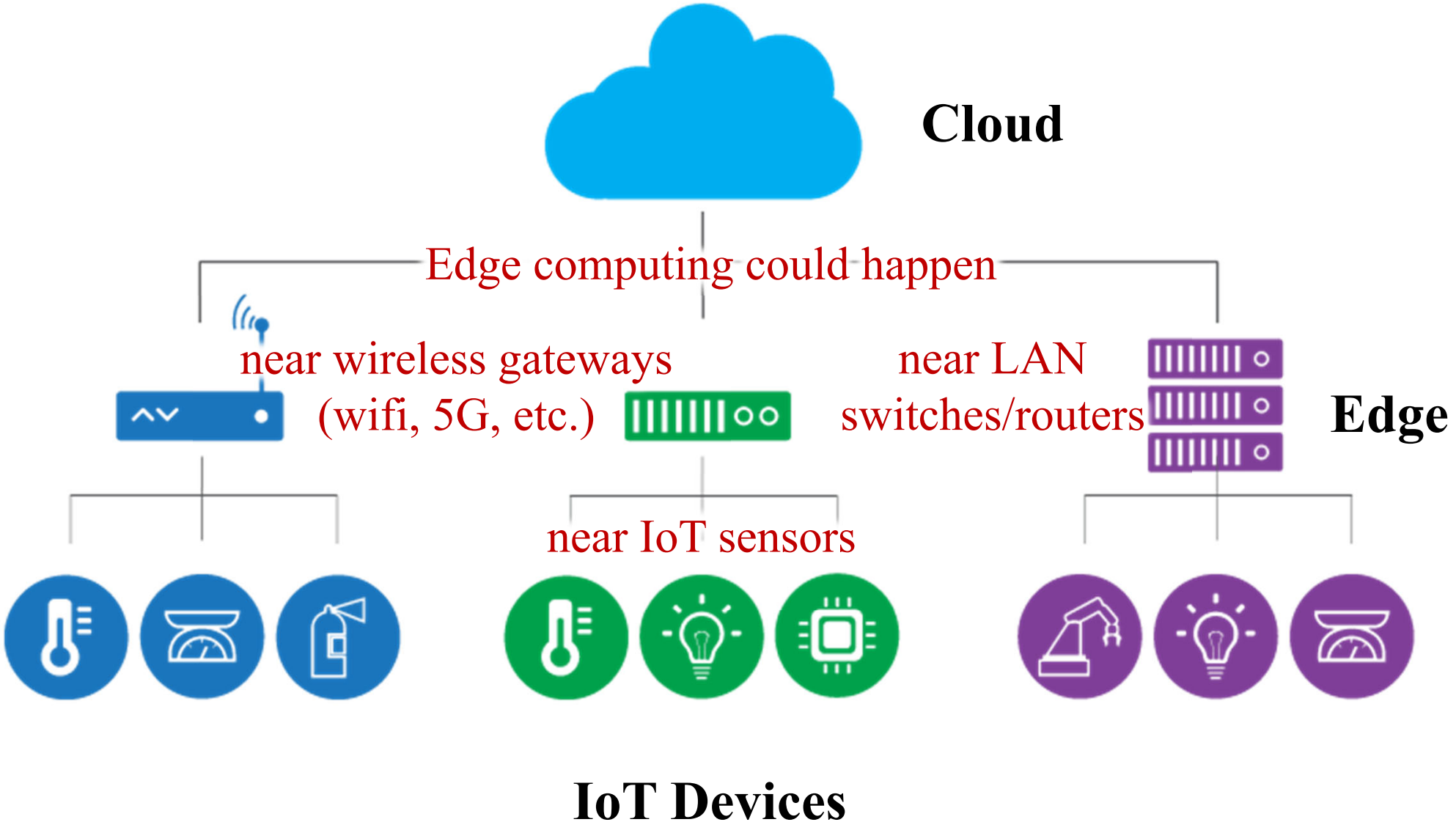
1.62 ZB

Cloud data center storage capacity

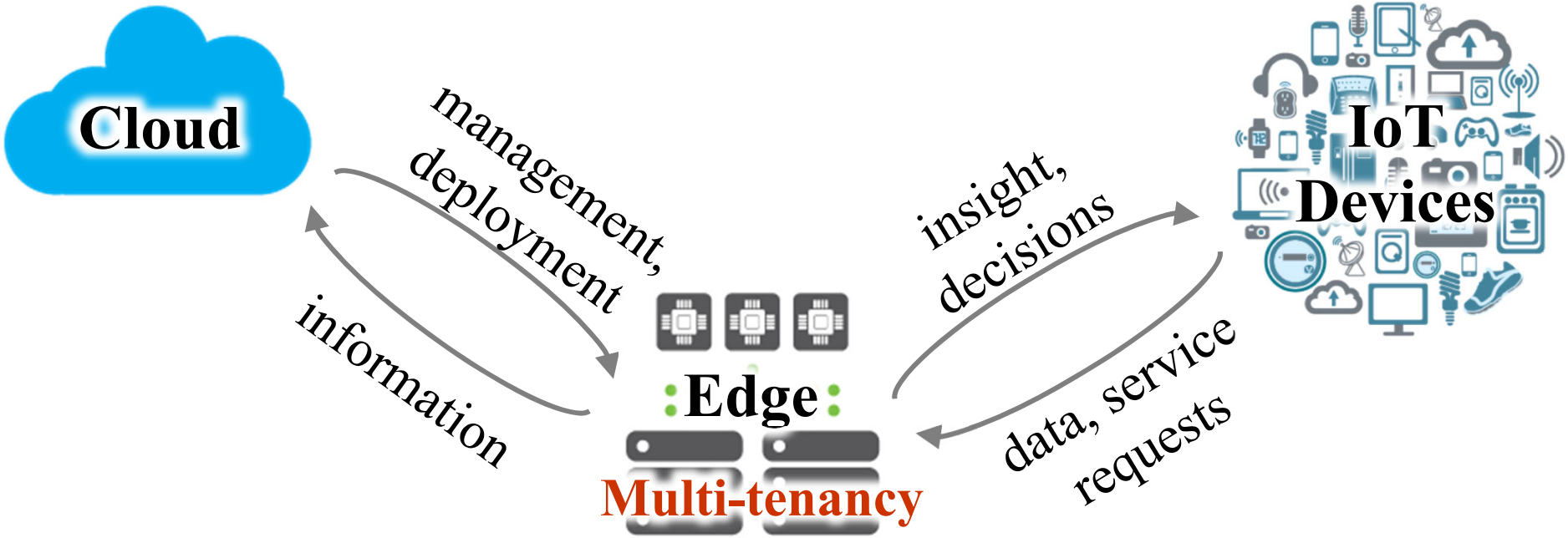
Edge computing is needed to bridge this gap!

"Cisco global cloud index: Forecast and methodology 2015–2020." *White paper* (2016).

Edge Computing: Distributed Computing Paradigm, Brings Computation & Storage Closer to Data Source



Edge Computing Benefits



- Faster response time
- Reliable operations even with limited connectivity
- Security and compliance
- Cost-effective compared to on-sensor/embedded computing
- Interoperability between legacy machines and modern IoT platforms

Cloud and Edge Workloads Have Distinct Characteristics

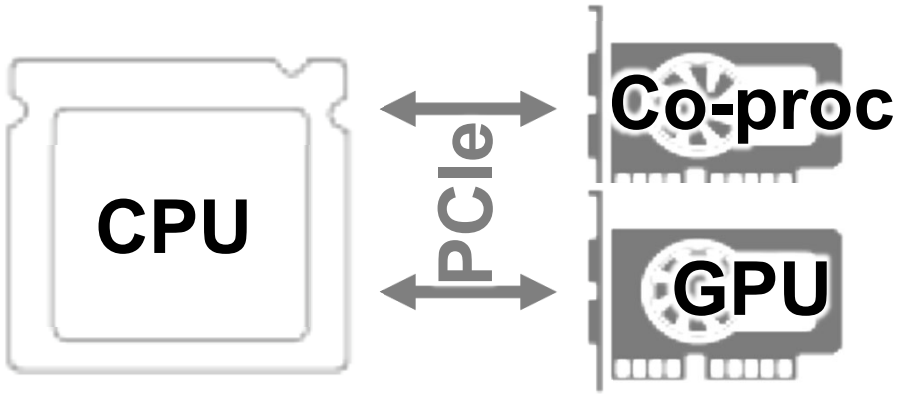
	Cloud	Edge
Main User	Human Users	IoT Devices
Time/latency-sensitive?	Usually Not	Usually Yes
Cluster Mode	Centralized	Geographically Distributed
Data Movement Pattern	Batches of Static Data	Streams of Dynamic Data
Data Access Interface	Deep Memory Hierarchy	Input/output (I/O) Channels

- Edge deals with frequent/periodic service requests from IoT devices
- Edge handles time-sensitive workloads in geographically distributed fashion
- Edge applications process streaming data from I/Os

Existing Solution (Cloudlets) Are Insufficient

CPU/GPU Architecture

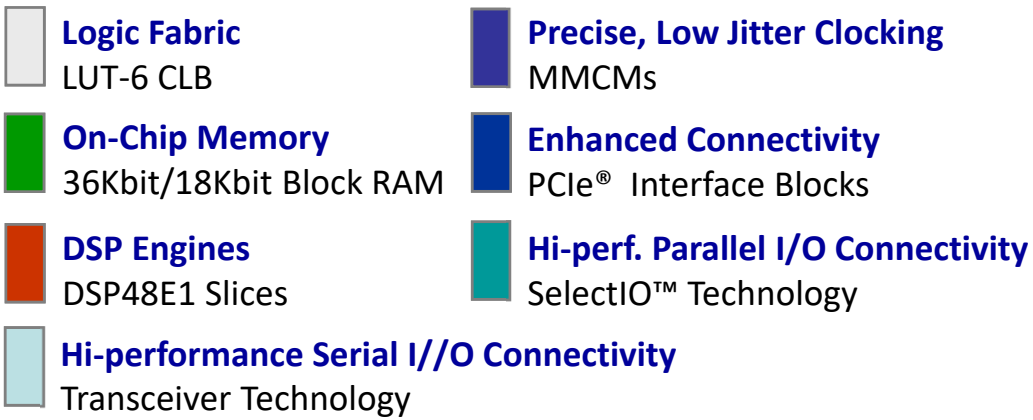
- Optimized for **batch** processing of **memory** data
- Limited performance for processing **streaming** data from **I/Os**
- Only exploit spatial parallelism but **not the temporal (pipeline) parallelism** needed for streaming processing
- Power-hungry, **low energy efficiency**



FPGAs for Edge Computing

What is an FPGA?

- An FPGA is a farm of configurable hardware resources whose functionality and interconnection can be redefined at run-time by programming the FPGA configuration memory.
- A state-of-the-art FPGA carries an enormous amount of fine-grained logic, computation, memory, and I/O resources, as well as coarse-grained functional blocks.
- Upon the configuration of these resources, an FPGA can implement any *custom* hardware architecture to accelerate any algorithm, achieving both performance and efficiency gains.



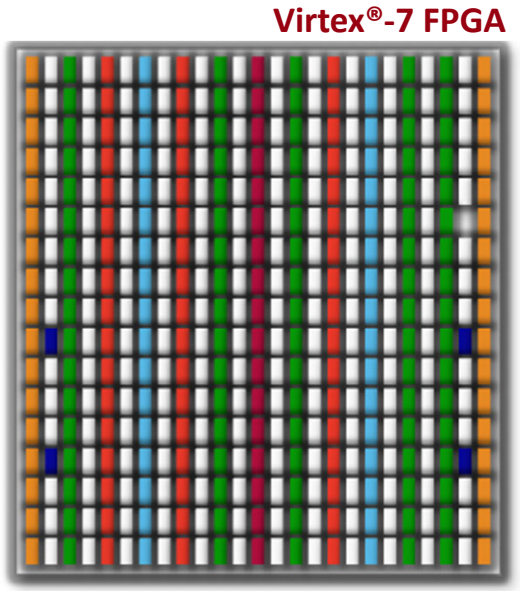
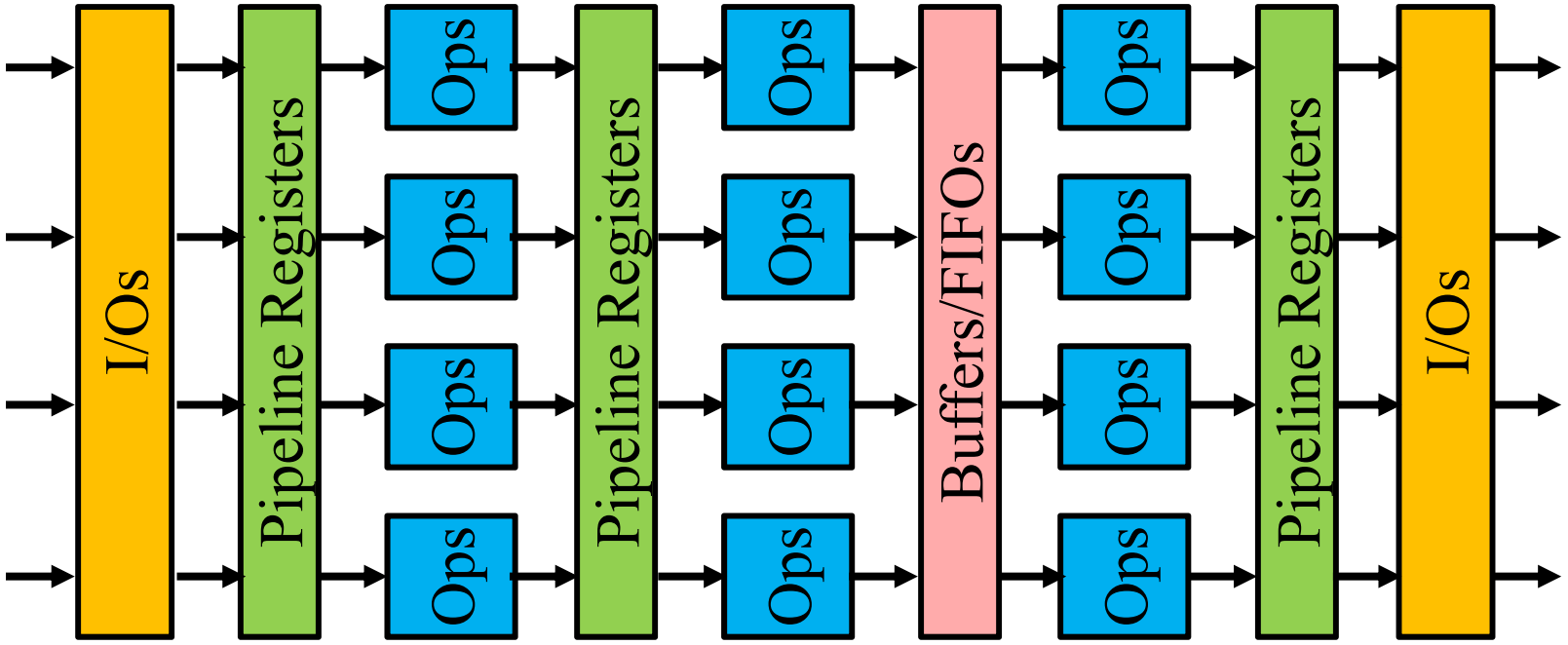


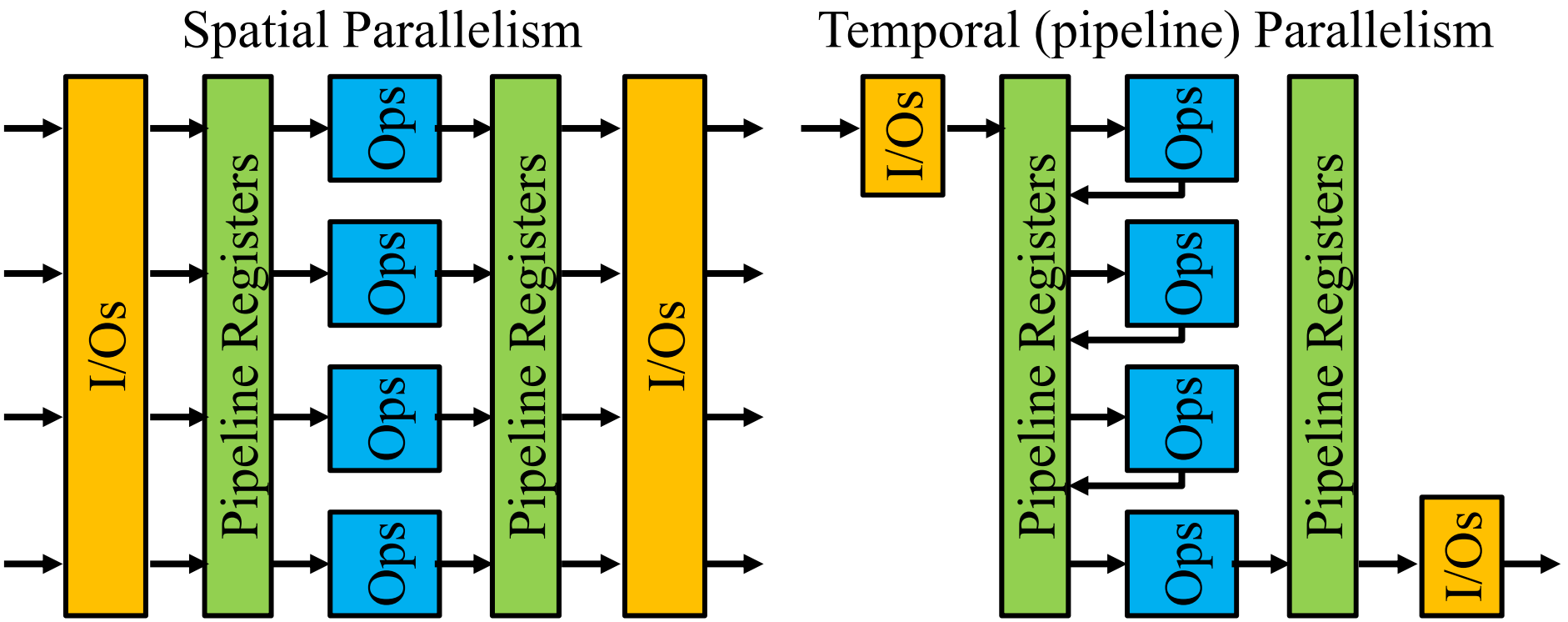
Image courtesy of Xilinx Inc.

FPGAs Are Inherently Efficient for Processing Streaming Data from I/Os



- With abundant register, memory, and configurable I/O resources, a streaming architecture can be implemented on an FPGA to process data streams directly from I/Os in a pipelined fashion
- The pipeline registers allow efficient data movement among processing elements without involving memory access, resulting in significantly improved throughput and reduced latency

FPGAs Have Architecture Adaptability—Can Exploit Both Spatial and Temporal Parallelism



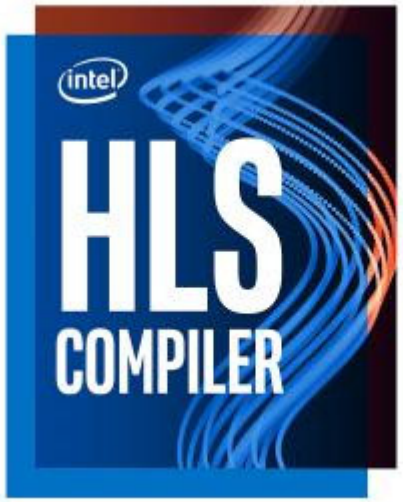
- FPGAs can adapt their architecture to best fit any algorithm characteristics due to their hardware flexibility
- The ability to exploit both spatial and temporal (pipeline) parallelism allows FPGAs to provide consistently high throughput for accelerating both high-concurrency and high-dependency algorithms, keeping the promise to efficiently serve a broader range of IoT/edge applications.

FPGAs Can Have Higher Energy Efficiency Than CPUs (Up to 100sX) and GPUs (Up to 10sX)

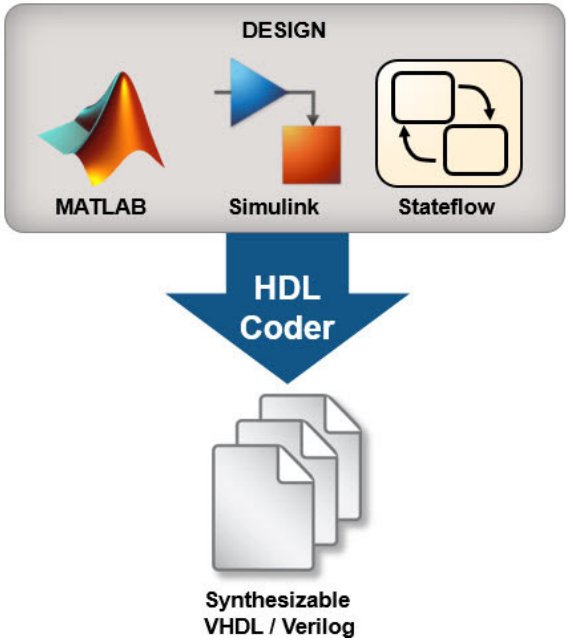
Normalized Energy Efficiency					
FPGA	CPU	GPU	Algorithm	Application Field	Paper Title
10	1	0.3	RNN	NLP	Accelerating Recurrent Neural Networks in Analytics Servers: Comparison of FPGA, CPU, GPU, and ASIC
80	1	16	NN	Computer Vision	Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC
2.7	N/A	1	CNN	Computer Vision	Embedded FPGA PlatformsOptimizing CNN-based Object Detection Algorithms on Embedded FPGA Platforms
4.9	N/A	1	CNN	Computer Vision	Embedded FPGA PlatformsOptimizing CNN-based Object Detection Algorithms on Embedded FPGA Platforms
197	1	14	LTSM	NLP	ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA
4.4	1	1.16	sparse matrix mul	linear algebra	A High Memory Bandwidth FPGA Accelerator for Sparse Matrix-Vector Multiplication
21	1	11	CNN	Computer Vision	Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster
4	N/A	1	CNN	Computer Vision	Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?
11.7	N/A	1	3D ultrasound computer tomography	Computer Vision	Evaluation of performance and architectural efficiency of FPGAs and GPUs in the 40 and 28 nm generations for algorithms in 3D ultrasound computer tomography
24	1	N/A	CNN	Computer Vision	Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks
50	1	1.3	sparse matrix mul	linear algebra	A Scalable Sparse Matrix-Vector Multiplication Kernel for Energy-Efficient Sparse-Blas on FPGAs
74	1	15	CNN	Computer Vision	From High-Level Deep Neural Models to FPGAs
6	N/A	1	3D ultrasound computer tomography	Computer Vision	Comparison of Processing Performance and Architectural Efficiency Metrics for FPGAs and GPUs in 3D Ultrasound Computer Tomography
11.6	N/A	1	Smith-Waterman	Genome sequencing	FPGA Based OpenCL Acceleration of Genome Sequencing Software
5.18	1	4.6	linear algebra	linear algebra	Evaluating and Optimizing OpenCL Kernels for High Performance Computing with FPGAs
2.6	1	N/A	Apache Spark	Genome sequencing	When Apache Spark Meets FPGAs: A Case Study for Next-Generation DNA Sequencing Acceleration
0.827	1	0.665		Genome sequencing	Energy efficiency of sequence alignment tools—Software and hardware perspectives
18.1	1	6.2	AES		Parallel AES Encryption Engines Many-Core Processor Arrays
270	1	N/A	Smith-Waterman	Genome sequencing	CMOST: A System-Level FPGA Compilation Framework
220	1	N/A	MPEG	image processing	CMOST: A System-Level FPGA Compilation Framework
287	1	9.2	Random Number Generators		A Comparison of CPUs, GPUs, FPGAs, and Massively Parallel Processor Arrays for Random Number Generation
4.6	N/A	1	optical flow	Computer Vision	A Comparison of FPGA and GPU for Real-Time Phase-Based Optical Flow, Stereo, and Local Image Features
3.7	1	2	option pricing	finance	Comparing Performance and Energy Efficiency of FPGAs and GPUs for High Productivity Computing
336	1	21	Quasi-Monte Carlo	finance	High-Performance Quasi-Monte Carlo Financial Simulation: FPGA vs. GPP vs. GPU
25	1	2.8		astronomical system	A comparative study on ASIC, FPGAs, GPUs and general purpose processors in the O(N2) gravitational N-body simulation
6.8	N/A	1	FIR	video processing	Separable FIR Filtering in FPGA and GPU Implementations: Energy, Performance, and Accuracy Considerations
1.73	N/A	1	option pricing	finance	An Energy Efficient FPGA Accelerator for Monte Carlo Option Pricing with the Heston Model
1	N/A	2.4	option pricing	finance	Energy-Efficient FPGA Implementation for Binomial Option Pricing Using OpenCL
3.9	N/A	1	sparse matrix mul	linear algebra	Communication Optimization of Iterative Sparse MatrixVector Multiply on GPUs and FPGAs

- Means improved thermal stability and reduced cooling/energy costs, critical to edge computing considering the limited form factor of edge devices.

FPGAs Now Support High-Level Programming Languages, C/C++/OpenCL/MATLAB and More



MATLAB HDL Coder



SDAccel™
Environment

FPGAs Are Becoming Increasingly More Accessible to Application Developers

- Intel FPGA SKD for OpenCL Provides
  - An offline compiler that generates the kernel hardware with interface logic and builds the kernel image needed for FPGA programming
  - A software-based emulator for symbolic debugging and functional verification
  - A run-time environment (firmware, software, device driver) to control and monitor the kernel deployment and execution, and transfer data between the host and FPGA device

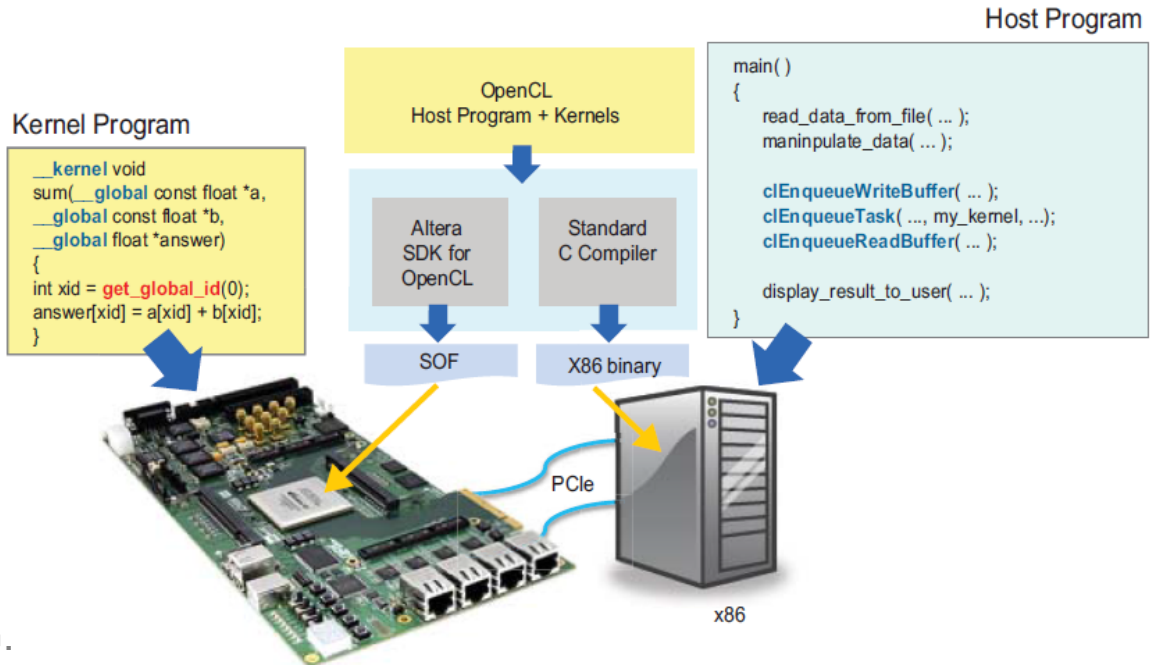


Image courtesy of Intel PSG.

Board Support Package (BSP): Turn An FPGA Board into A PCIe Computing Card

- A BSP must be designed at the HDL level and mapped onto the FPGA to interact with the OpenCL runtime, which contains
  - A static region that contains validated glue logics connected by an on-chip network that manages the board-level resources (DDR, Various I/Os, etc.) and interfaces them with user-defined kernels
  - A partial reconfigurable region that can be dynamically programmed to implement any user-defined kernels at run time.

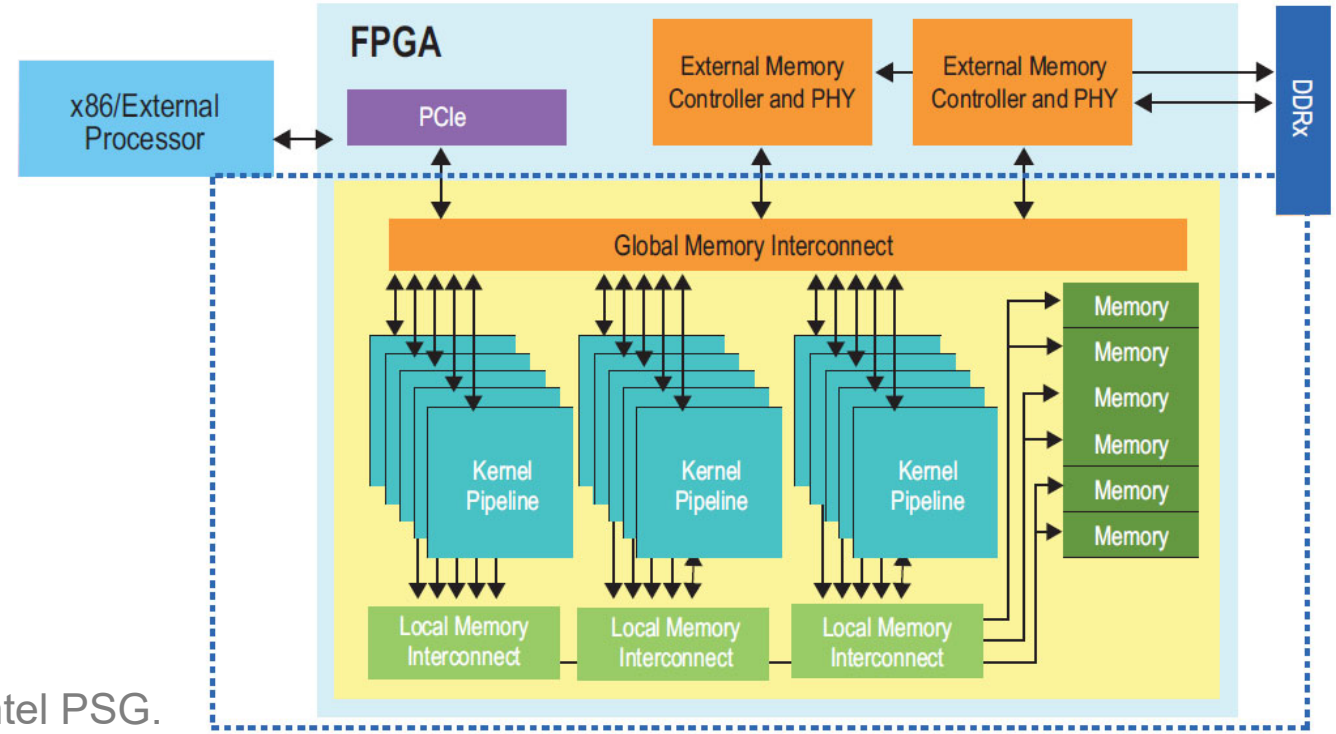


Image courtesy of Intel PSG.

FPGAs Have Now Become the 3rd General-purpose Computing Device after CPU, GPU

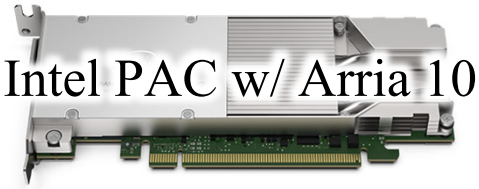
- Some OpenCL-supported FPGA card vendors and off-the-shelf products

BittWare a molex company

Nallatech a molex company

REFLEX CES Custom Embedded Systems

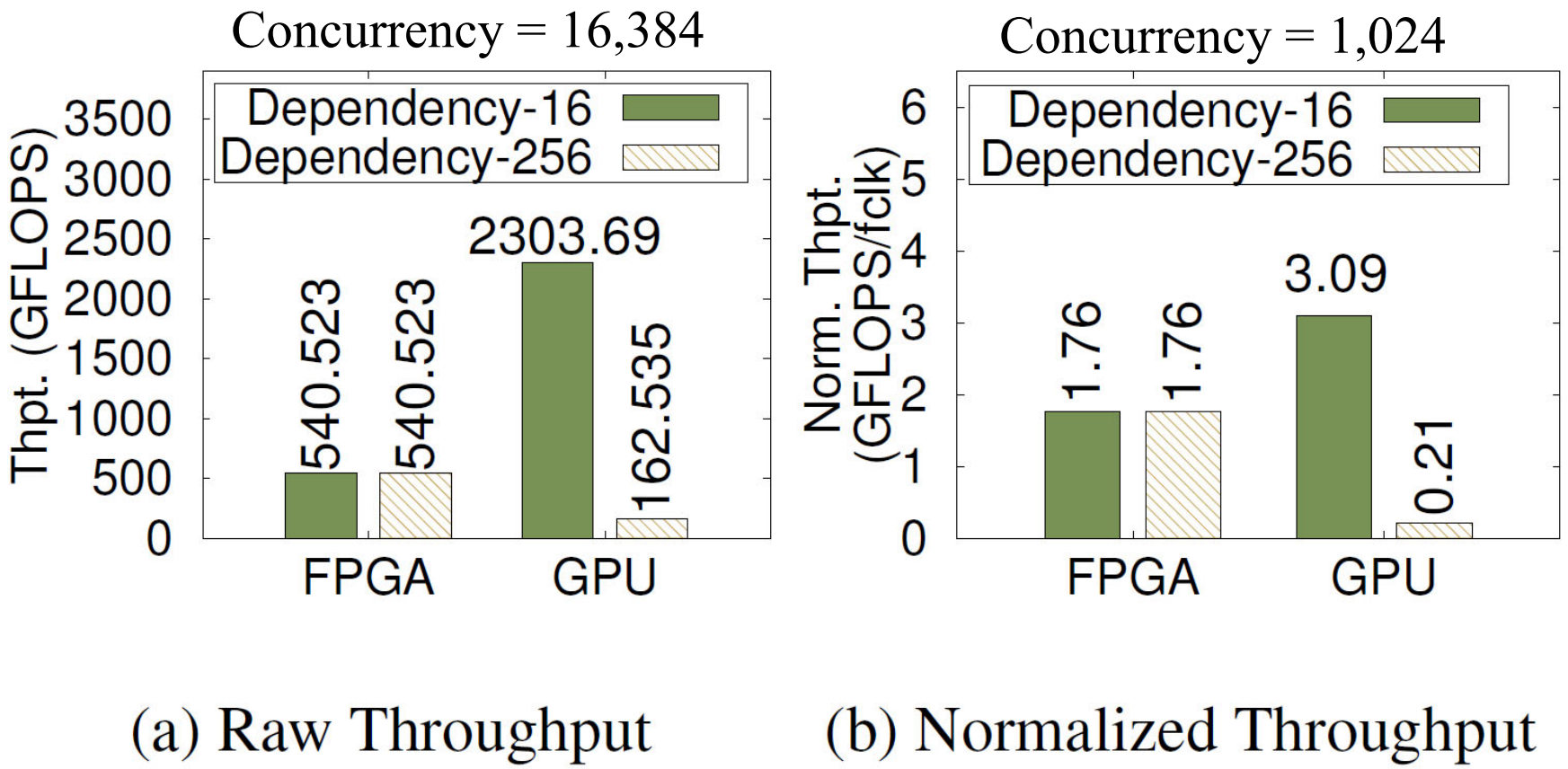
Full Height PCIe Form Factors



Low-profile PCIe Form Factors

FPGAs Are Suitable for Edge Computing: Performance Invariance to Algorithm Characteristics

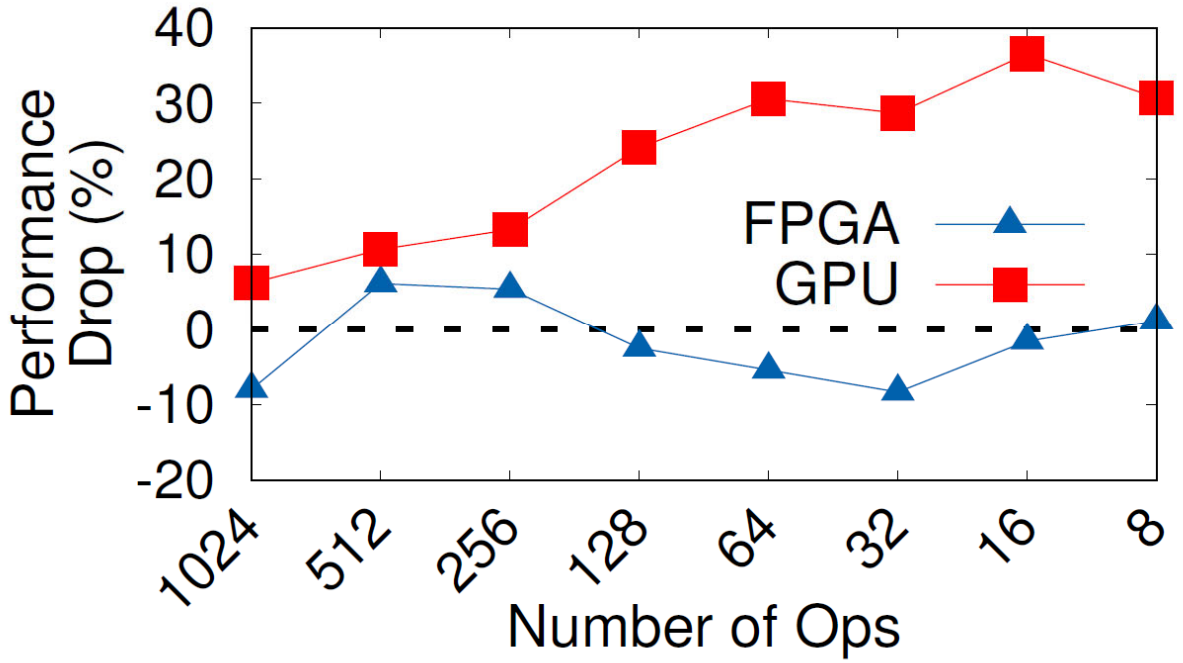
- FPGAs' architectural adaptiveness can guarantee a stably high processing throughput for various edge workloads



S. Biookaghazadeh, M. Zhao, and F. Ren, "Are fpgas suitable for edge computing?." *USENIX Workshop on Hot Topics in Edge Computing* 2018.

FPGAs Are Suitable for Edge Computing: Good at Handling Cond. Dependency (if/else/case)

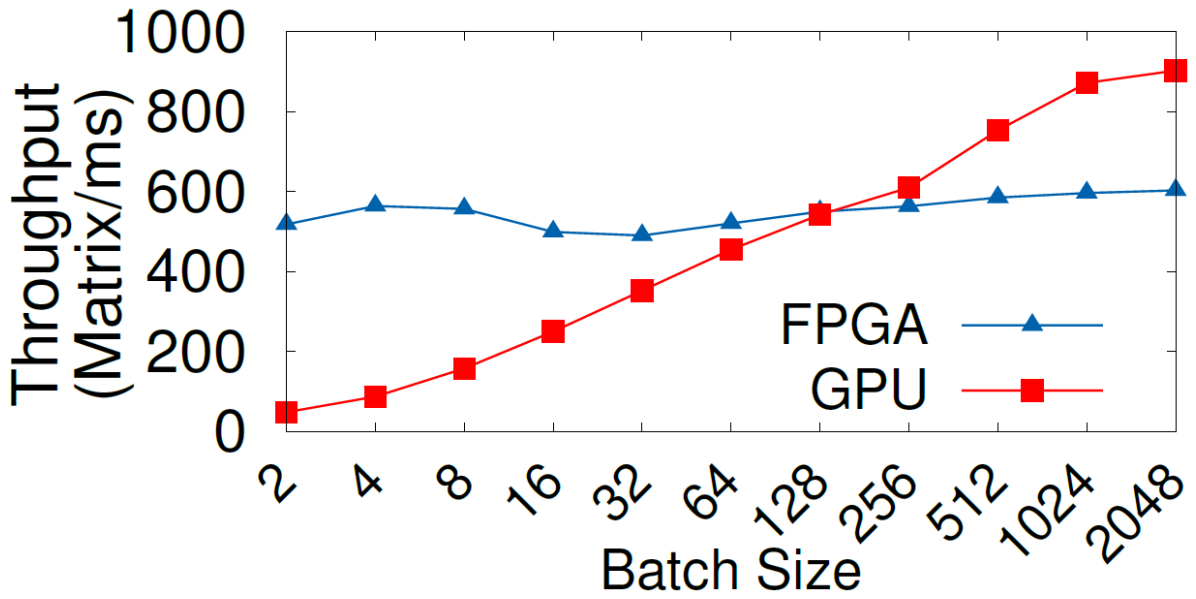
- FPGAs' fine-grained logic can efficiently implement if/else/case as customized logic pipelined with or controlling the data path
- Give edge applications more flexibility in handling different conditions



S. Biookaghazadeh, M. Zhao, and F. Ren, "Are fpgas suitable for edge computing?." *USENIX Workshop on Hot Topics in Edge Computing* 2018.

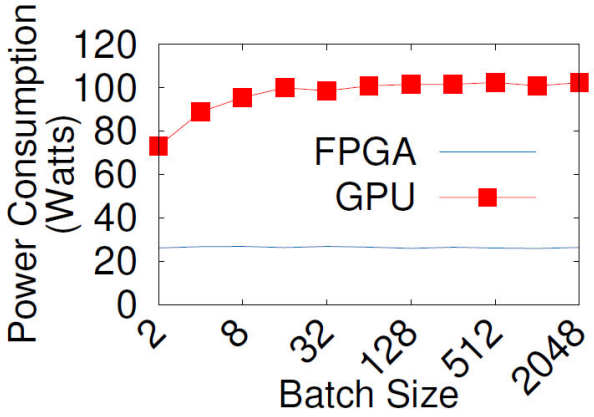
FPGAs Are Suitable for Edge Computing: Performance Invariance to Data Batch Size

- Edge applications process dynamic data from I/Os that have very small batch size
- FPGAs can exploit temporal parallelism to efficiently process I/O data in a pipelined fashion, providing a consistent throughput even at small batch sizes

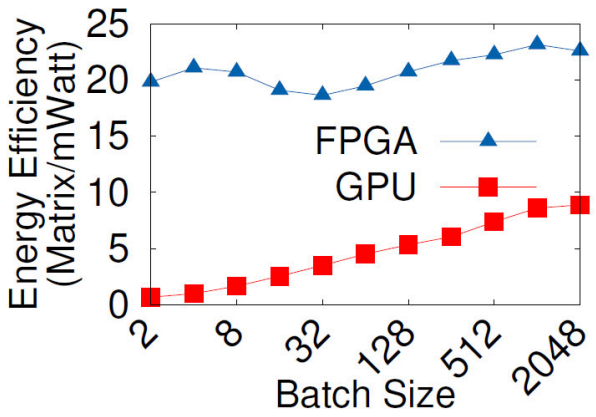


S. Biookaghazadeh, M. Zhao, and F. Ren, "Are fpgas suitable for edge computing?." *USENIX Workshop on Hot Topics in Edge Computing* 2018.

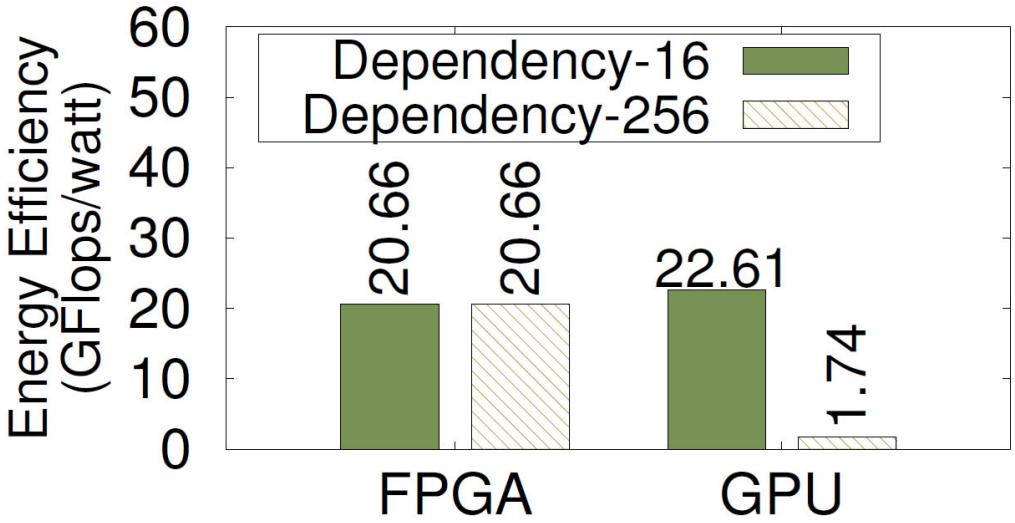
FPGAs Are Suitable for Edge Computing: Consistently High Energy Efficiency



(a)



(b)



(c)

S. Biookaghazadeh, M. Zhao, and F. Ren, "Are fpgas suitable for edge computing?." *USENIX Workshop on Hot Topics in Edge Computing* 2018.

The Edge Deployment Challenge of Deep Learning

DNNs Have High Computational Complexity (CC)
DNN Training Rarely Considers HW Constraints

DSP*

3~6 orders of magnitude more computational complexity

DNN*

- 2D-DWT -> 0.1 MFLOPs
- 2D-DCT -> 0.2 MFLOPs
- FFT -> 3.9 MFLOPs

- AlexNet -> 0.7 GFLOPs
- ResNet-152 -> 11 GFLOPs
- RetinaNet -> 12 GFLOPs
- FasterRCNN -> 446 GFLOPs

$$y = Ax$$

$x = s_1 \cdot s_2 \ldots s_k$, S is sparse

$$y = f_k(\ldots f_2(f_1(W, X)))$$

$$1D: \sim 2nlog_2n$$
$$2D: \sim 2n^2log_2n$$

$$2D: \sim kcs^2n'^2,$$
$k = \#\ of\ layers,$
$c = featureMapCount$
$s = filterSize$
kcs^2 often $\gg 10^4$

*input image size: 224*224

DNNs Have Large Model Size
Off-chip Memory-bounded Computation

Comparison of model size

	Algorhthm	Weights
DSP	2D-DWT	256 KB
	2D-DCT	128 KB
Deep learning	AlexNet	233 MB
	ResNet50	98 MB
	RetinaNet	120 MB
	FasterRCNN	5500 MB

Comparison of CPU/FPGA/GPU cache/on-chip memory size

	device	Cache/on-chip memory
CPU	Intel Xeon E5-2630	16.7 MB
FPGA	Arria 10 GX 1150	65.7 MB
GPU	NVIDIA Tesla T4	4 MB

High CC and Limited Sp.-temp. Utilization Challenge for Multi-tenancy Edge Computing

Comparison of FLOPs

	Algorhthm	FLOPs
DSP	2D-DWT	0.1 M
	2D-DCT	0.2 M
Deep learning	AlexNet	0.7 G
	ResNet50	4 G
	RetinaNet	12 G
	FasterRCNN	446 G

$$s = \frac{GOPs}{freq * P * runtime} * 100\%$$

- **Occupation Rate:** spatial utilization of cores
- **Stall Rate:** Temporal utilization of cores

Comparison of spatial-temporal utilization of CPU/FPGA/GPU

	Device	Peak throughput	Spatial-temporal utilization (s)*
CPU	Intel Xeon E5-2630	0.3 TFLOPs	10% - 18%
FPGA[1]	Arria10	1.5 TFLOPs	63% - 83%
GPU [2]	Nvidia Tesla T4	14 TFLOPs	6% - 33%

* Rough estimation from literature

= about 2 FPS for Faster RCNN on 224*224 image with batch size of 1
How about larger images?
How about multiple app.?

[1] Wang, Dong, Ke Xu, and Diankun Jiang. "PipeCNN: An OpenCL-based open-source FPGA accelerator for convolution neural networks." 2017 International Conference on Field Programmable Technology (ICFPT). IEEE, 2017.
[2] https://developer.nvidia.com/deep-learning-performance-training-inference

Techniques for Reducing Computational Complexity of Deep Neural Networks (DNNs)

Techniques for Reducing Computational Complexity of DNNs

- Reduce total FLOPs
  - Pruning
  - Computational transformation (CNN-specific)
  - Simplified building blocks (CNN-specific)

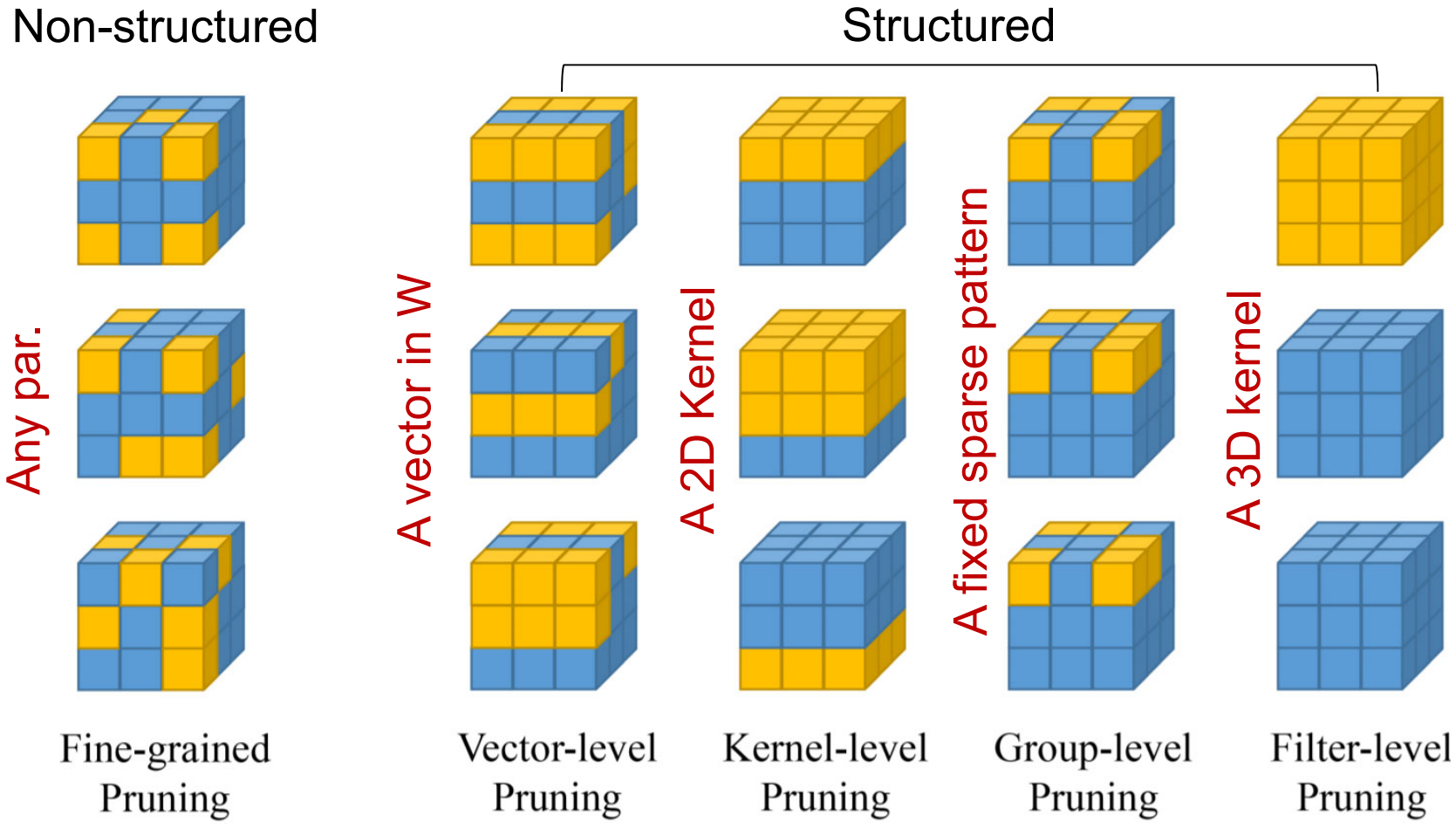
- Reduce numerical precision
  - Quantization
  - Ternary/Binary Nets

Pruning Key Concept

- Many parameters in deep networks are unimportant or unnecessary
- Identify and prune out non-critical (redundant) weights and/or activations
- Expand the sparsity of the weight matrix will result in
  - Smaller memory footprint since the parameters can be stored in the compressed sparse row (CSR) or compressed sparse column (CSC) format
  - Reduced computational complexity

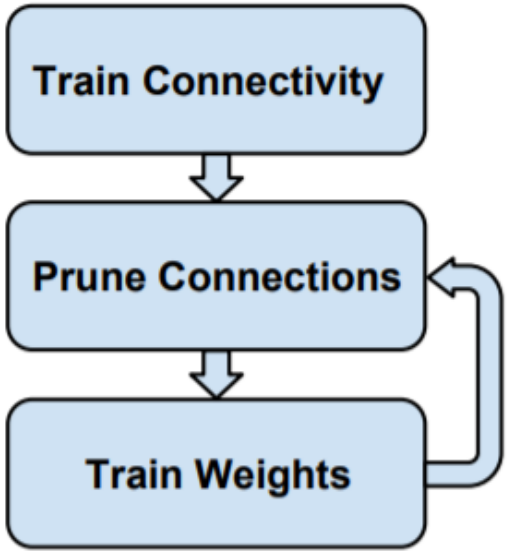
Pruning Can be Done at Different Levels of Granularity

- Depending on the pattern and granularity, pruning and the remaining network can be structured or non-structured



Cheng, Jian, et al. "Recent advances in efficient computation of deep convolutional neural networks." *Frontiers of Information Technology & Electronic Engineering* 19.1 (2018): 64-77.

Pruning Approach



- Start with fully/partially trained network
- Analyze the criticality of a or a group of parameters to final accuracy and prune.
  - Calculate the approximate 2nd derivatives of the loss w.r.t. the praters (non-structured) [B. Hassibi, et al., NIPS'93]
  - Remove weight connections in small magnitude (non-structured), combined with quantization and Huffman encoding [H. Song, et al., ICLR'16]
  - Select filter combination that minimizes the reconstruction errors of the next layer's feature map [J.-H. Luo, et al., ICCV'17]
  - Introduce filter selection weight, find the optimal filter selection weight vector that minimize the feature map errors [K. He, et al., ICCV'17]
  - Use the scaling factor of the batch normalization layer to decide filter importance [Z. Liu et al., ICCV'17]
- Fine-tune the remaining weights through training to compensate for the accuracy drop

Non-structured Pruning Results

	Dataset	CNN arch.	Top-1 accuracy/%	Top-5 accuracy/%	Accuracy drop/%	Model compression	FLOPs reduction
[1]	ImageNet	AlexNet	57.23		0.01	9x	3x
[1]	ImageNet	VGG-16	68.66		0.16	13x	5x
[2]	ImageNet	AlexNet	-	79.56	0.87	11x	6x
[2]	ImageNet	GoogLeNet	-	87.28	0.98	3x	2x
[2]	ImageNet	SqueezeNet	-	80.47	0.14	2x	1x
[3]	ImageNet	AlexNet	56.91	-	0.39	18x	-

- Up to 18x model size reduction
- Up to 6x FLOPs reduction
- Well-bounded accuracy (<1%) drop due to fine-grained pruning
- Must adopt sparse matrix representation format.
- Limited speedup on GPUs due to indexing overhead and irregular memory access.

[1] Han, Song, et al. "Learning both weights and connections for efficient neural network." *Advances in neural information processing systems*. 2015.
[2] Yang, T. J., Chen, Y. H., & Sze, V. (2017). Designing energy-efficient convolutional neural networks using energy-aware pruning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5687-5695).
[3] Guo, Yiwen, Anbang Yao, and Yurong Chen. "Dynamic network surgery for efficient dnns." Advances In Neural Information Processing Systems. 2016.

Structured Pruning Results

	Dataset	CNN arch.	accuracy/%	Pruning Granularity	Accuracy drop/%	Model compression	FLOPs reduction	Speedup ratio
[1]	ImageNet	ResNet-50	72.04 (Top-1)	filter level	0.84	1.5x	1.6x	-
[1]	ImageNet	ResNet-50	68.42 (Top-1)	filter level	4.46	3.0x	3.5x	-
[1]	ImageNet	VGG-16	-	filter level	1.00	17x	3.3x	-
[2]	ImageNet	ResNet-50	90.8 (Top-5)	filter level	1.4	-	-	2x
[2]	ImageNet	VGG-16	89.6 (Top-5)	filter level	0.3	-	-	5x

- Up to 17x model size reduction
- Up to 3.5x FLOPs reduction
- Can lead to more accuracy drop due to coarse-grained pruning
- Vector-, kernel-, and filter-level pruning requires fewer indices and are much more friendly to memory access
- Up to 5x speed up on GPUs
- FLOPs saving of filter-level pruning should translate into speed-up accordingly on FPGAs and GPUs

[1] Luo, Jian-Hao, Jianxin Wu, and Weiyao Lin. "Thinet: A filter level pruning method for deep neural network compression." *Proceedings of the IEEE international conference on computer vision*. 2017.
[2] He, Yihui, Xiangyu Zhang, and Jian Sun. "Channel pruning for accelerating very deep neural networks." Proceedings of the IEEE International Conference on Computer Vision. 2017.

Impact on FPGA Implementation
3x Faster and 11.5x More E.-Efficient Than GPU

20x model size reduction

Method			Non-structured pruning
Network			LSTM
Dataset			TIMIT
Accuracy (%)			89.3
Device			Xilinx XCKU060
Weight bit-width			16
Act. bit-width			16
Weight on chip			No
Frequency (MHz)			200
Throughput (GOP/s)			282
FPGA DSP peak (GLOP/s)*			1327
Actural throughput/Device peak			0.2x
Latency (ms)			0.0827
Power (W)			41
Energy efficiency (GOP/s/W)			7
Accelation ratio	CPU (core i7 5930k)	FPGA/CPU (actual)	43x
	GPU (Pascal Titan X)	FPGA/GPU (peak)	0.12x
		FPGA/GPU (actual)	3x

10x

2x

* FPGA DSP peak performance is scaled down by actual frequency

- Non-structured pruning is much more friendly to FPGA than GPU. FPGAs can implement the indexing scheme into memory controller & computation pipeline with customized logic.

Han, Song, et al. "Ese: Efficient speech recognition engine with sparse lstm on fpga." 2017 ACM/SIGDA International Symposium on FPGA.

Techniques for Reducing Computational Complexity of DNNs

- Reduce total FLOPs
  - Pruning
  - Computational transformation (CNN-specific)
  - Simplified building blocks (CNN-specific)

- Reduce numerical precision
  - Quantization
  - Ternary/Binary Nets

Computational Transformation Key Concept

- Computational complexity of CNNs is dominated by Conv layers
- The key to accelerate CNN is to accelerate Conv computation
- Convolution has been widely used in image processing (filtering, pattern matching, etc.), there exist rich research on the efficient computation methods of Conv
- Apply existing fast algorithms of Conv to accelerate CNN

Table 1 Computation and parameters for state-of-the-art convolution neural networks

Method	Parameter			Computation		
	Size (M)	Conv (%)	Fc (%)	FLOPs (G)	Conv (%)	Fc (%)
AlexNet	61.0	3.8	96.2	0.72	91.9	8.1
VGG-S	103.0	6.3	93.7	2.60	96.3	3.7
VGG16	138.0	10.6	89.4	15.50	99.2	0.8
NIN	7.6	100.0	0	1.10	100.0	0
GoogLeNet	6.9	85.1	14.9	1.60	99.9	0.1
ResNet-18	5.6	100.0	0	1.80	100.0	0
ResNet-50	12.2	100.0	0	3.80	100.0	0
ResNet-101	21.2	100.0	0	7.60	100.0	0

FLOPs: floating-point operations; Conv: convolutional layers; Fc: fully-connected layers

Image Courtesy of Cheng, Jian, et al.

Winograd: Minimal Filter Algorithm, Can Apply to Multi-dim. Convolution with Stride Size of 1

1. Each channel of input feature map (FM) is divided into multiple small tiles x of size $(m + r - 1) \times (m + r - 1)$ with an overlap of $r - 1$ elements

2. Each FM tile x is processed as $V = B^T x B$

3. Each filter w of size $r \times r$ is pre-processed as $U = G^T w G$

4. For the 2-D convolution of each FM tile and filter combination in a channel, denoted as $F(m \times m, r \times r)$, compute the element-wise matrix multiplication $M = U \odot V$

5. Results are summed over of all channels $M' = \sum_{i=1}^{C} Y_i$
6. Apply inverse transform $Y = A^T M A$ to get final convolution results

Winograd breaks down a large-size matrix-matrix mult. of Conv into a large batch of smaller-size matrix-matrix mult.

Lavin, Andrew, and Scott Gray. "Fast algorithms for convolutional neural networks. CVPR 2016.

Winograd Benefits: Reduce Multiplications for $F(m \times m, r \times r)$ from m^2r^2 to $(m + r - 1)^2$

Winograd Examples

F(m*m, r*r)	Multiplications		
	m =2, r =3	m =4, r =3	m =6, r =3
Standard algorithm	36	144	324
Winograd algorithm	16	36	64
Arithmetic complexity reduction	2.25x	4x	5.06x

- Significantly reduce mult. at the cost of increased additions (reduced total FLOPs).
- Benefits to dedicated HW implementation (FPGAs, ASICs)
  - Additions are much cheaper in dedicated HW
  - A , G, B are constant transformation matrices with very low precision
  - The mult. in $U = G^T wG$, $M = U \odot V$, and Y $= A^T MA$ can be efficiently implemented by lookup tables and shift registers
  - The large batch of smaller-size matrix-matrix mult. can be computed in a highly parallel fashion.

Huang, Y., et al. "A High-efficiency FPGA-based Accelerator for Convolutional Neural Networks using Winograd Algorithm." Journal of Physics: Conference Series. Vol. 1026. No. 1. IOP Publishing, 2018.

Winograd Can Improve GPU Performance Especially for Small Batch Size

N	cuDNN		F(2x2,3x3)		Speedup
	msec	TFLOPS	msec	TFLOPS	
1	12.52	3.12	5.55	7.03	2.26X
2	20.36	3.83	9.89	7.89	2.06X
4	104.70	1.49	17.72	8.81	5.91X
8	241.21	1.29	33.11	9.43	7.28X
16	203.09	3.07	65.79	9.49	3.09X
32	237.05	5.27	132.36	9.43	1.79X
64	394.05	6.34	266.48	9.37	1.48X

Table 5. cuDNN versus $F(2 \times 2, 3 \times 3)$ performance on VGG Network E with fp32 data. Throughput is measured in Effective TFLOPS, the ratio of direct algorithm GFLOPs to run time.

- 2.26x speedup for batch size = 1 for no accuracy drop
- The smaller-size matrix-matrix mult. can exploit L2 cache more efficiently and reduce DRAM access
- The large batch matrix-matrix mult. can be highly parallelized in GPU execution
- Leads to higher GPU utilization especially at small batch sizes

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

Fast Fourier Transform (FFT): Fast Algorithm for Discrete Fourier Transform (DFT)

- DFT: A linear mapping that Transform a signal into frequency domain

$$\text{DFT: } X = Wx$$

The transformation matrix W can be defined as $W = \left(\frac{\omega^{jk}}{\sqrt{N}}\right)_{j,k=0,\ldots,N-1}$, or equivalently:

$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix},$$

Symmetric

Structurally Sparse

$$\text{FFT: } X = W_1 W_2 \cdots W_k x$$

Dual of The Convolution Theorem

$$x \cdot y \quad \overset{\text{IFFT}}{\underset{\text{FFT}}{\leftrightarrow}} \quad \frac{1}{N} X \otimes Y$$

- Convolution in one domain equals point-wise multiplication in the other domain
- Convolution and point-wise multiplication are inter-changeable through domain transformation

FFT: Transform 2D/3D Conv into Element-wise Matrix Multiplication in Frequency Domain

$$\mathbf{conv2D}(X[c], \Theta[n, c]) = \mathrm{IFFT}\Big(\mathrm{FFT}(X[c]) \odot \mathrm{FFT}(\Theta[n, c])\Big)$$

- Reduce the computational complexity to $\mathcal{O}(W^2 log_2 W)$
- Can further reduce the computational complexity to $\mathcal{O}(W log_2 r)$ using the overlap-and-add method when input feature map size is much larger than the filter size
- FFT based approach is more competitive for large kernel size ($r > 5$)
- For small kernel sizes (e.g. 1×1, 3×3), Winogram based approach is typically more effective.
- For small kernel sizes, FFT based approach must use large tile size (e.g. 64×64) to further reduce the multiplication complexity to the Winograd level, which will require larger memory workspace to hold transformed data

Abdelouahab, Kamel, et al. "Accelerating CNN inference on FPGAs: A Survey." arXiv preprint arXiv:1806.01683 (2018).

Impact on FPGA Implementation
Winograd Is More Effective Than FFT

Table 3: FPGA-Based CNN accelerators employing computational transform to accelerate conv layers

		Network	Network Workload		Bitwidth	Desc.	Device	Freq	Through	Power	LUT	DSP	Memory	Normalized Throughput#
			Comp. (GOP)	Param. (M)				(MHz)	(GOPs)	(W)	(K)		(MB)	
Winograd	[31]	AlexNet-C	1.3	2.3	Float 32	OpenCL	Virtex7 VX690T	200	46		505	3683	56.3	
	[30]	AlexNet-C	1.3	2.3	Float16	OpenCL	Arria10 GX1150	303	1382	44.3	246	1576	49.7	**2.89**
	[70]	VGG16-C	30.7	14.7	Fixed 16	HLS	Zynq ZU9EG	200	3045	23.6	600	2520	32.8	**3.02**
		AlexNet-C	1.3	2.3					855					
FFT	[32]	AlexNet-C	1.3	2.3	Float 32		Stratix5 QPI	200	83	13.2	201	224	4.0	**1.85**
		VGG19-C	30.6	14.7					123					**2.75**
GEMM	[28]	AlexNet-C	1.3	2.3	Fixed 16	OpenCL	Stratix5 GXA7	194	66	33.9	228	256	37.9	**0.66**
	[66]	VGG16-F	31.1	138.0	Fixed 16	HLS	Kintex KU060	200	365	25.0	150	1058	14.1	**0.86**
							Virtex7 VX960T	150	354	26.0	351	2833	22.5	
	[61]	VGG16-F	31.1	138.0	Fixed 16	OpenCL	Arria10 GX1150	370	866	41.7	437	1320	25.0	
					Float 32	OpenCL		385	1790	37.5		2756	29.0	

For "fair" comparison, the throughput is normalized by frequency and DSP parallelism (assumed doubled for Fixed-16 to cancel out the impact of quantization).

- For AlexNet, Winograd and FFT improves the FPGA performance by 4.3x and 2.7x, respectively, comparing to general matrix multiplication (GEMM)
- For VGG, the improvement is 3.5x and 3.1x, respectively, over GEMM
- Winograd based implementation is 1.1-1.6x faster than FFT based implementation

Abdelouahab, Kamel, et al. "Accelerating CNN inference on FPGAs: A Survey." arXiv preprint arXiv:1806.01683 (2018).

Techniques for Reducing Computational Complexity of DNNs

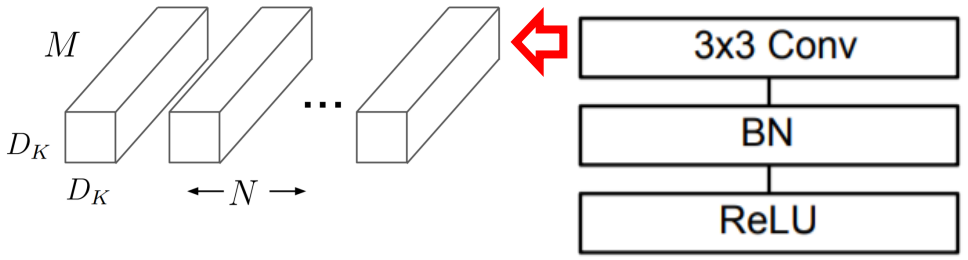
- Reduce total FLOPs
  - Pruning
  - Winograd (CNN-specific)
  - Simplified building blocks (CNN-specific)

- Reduce numerical precision
  - Quantization
  - Ternary/Binary Nets

MobileNet: Factorize A Convolution into 2 Stages Depth-wise and Point-wise Convolution

- Depth-wise convolution applies a single filter per each input channel
- Point-wise (1x1) convolution creates a linear combination of the depth-wise convolution outputs
- Filtering of each input channel and channel combining are performed separately

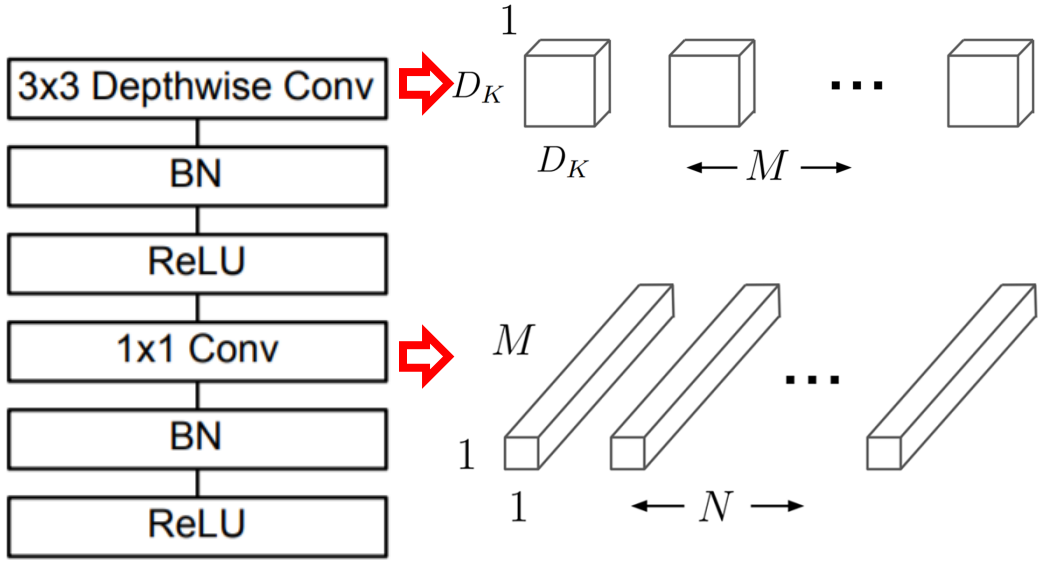
Conventional Conv Layer



$$\frac{D_K \times D_K \times M \times D_F * D_F + M \times N \times D_F \times D_F}{D_K \times D_K \times M \times N \times D_F \times D_F}$$

$$\text{FLOPs Saving} = \frac{1}{N} + \frac{1}{D_K^2}$$

Depth Separable Convolution



Howard, A. et al. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

MobileNet – Results

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Comparison with conventional Conv

- 9x FLOPs saving
- 7x Memory saving
- <1% accuracy drop

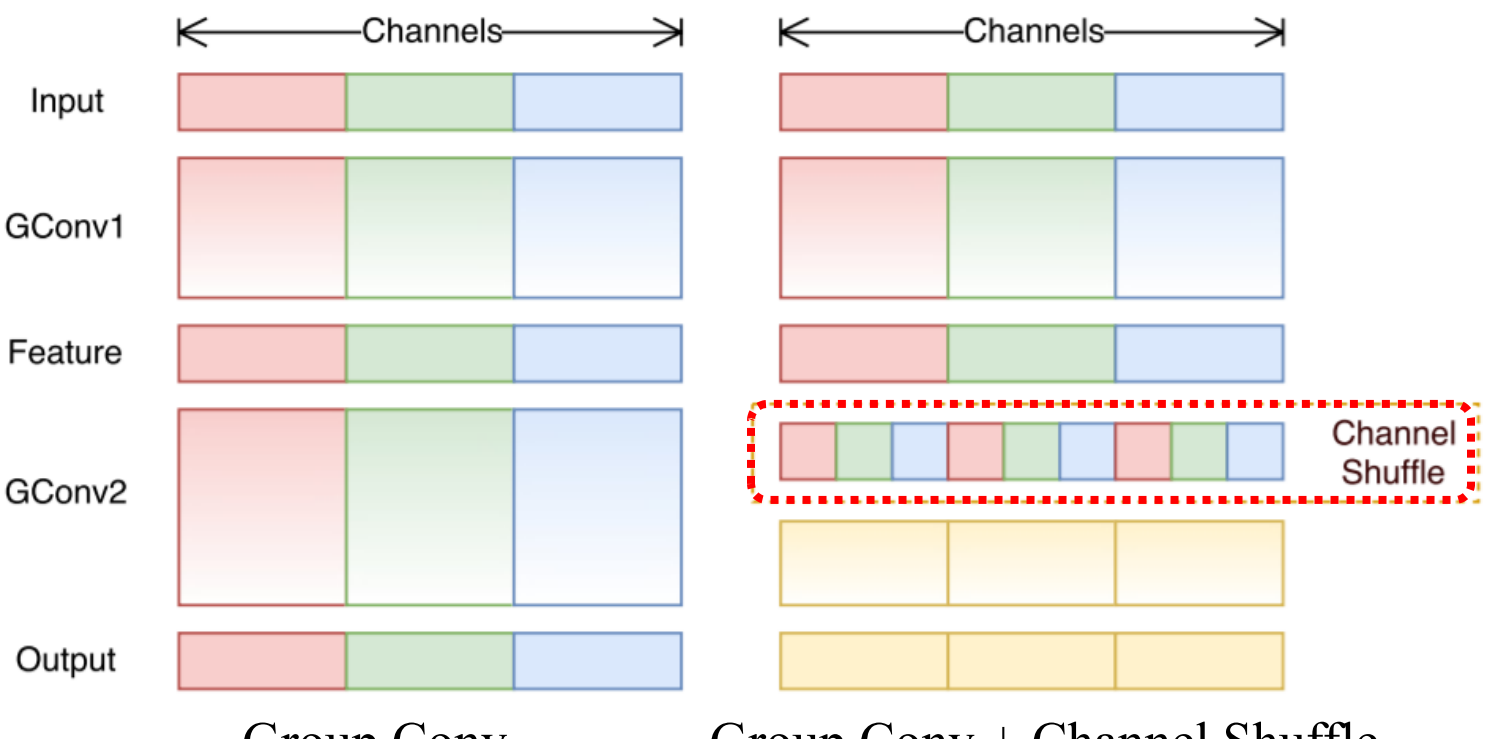
Comparison with other networks

- 3-27x FLOPs saving
- 1.6-33x Memory saving
- <1% accuracy drop

Howard, A. et al. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

ShuffleNet: Point-wise Group Conv + Channel Shuffle for Low Complexity and High Accuracy

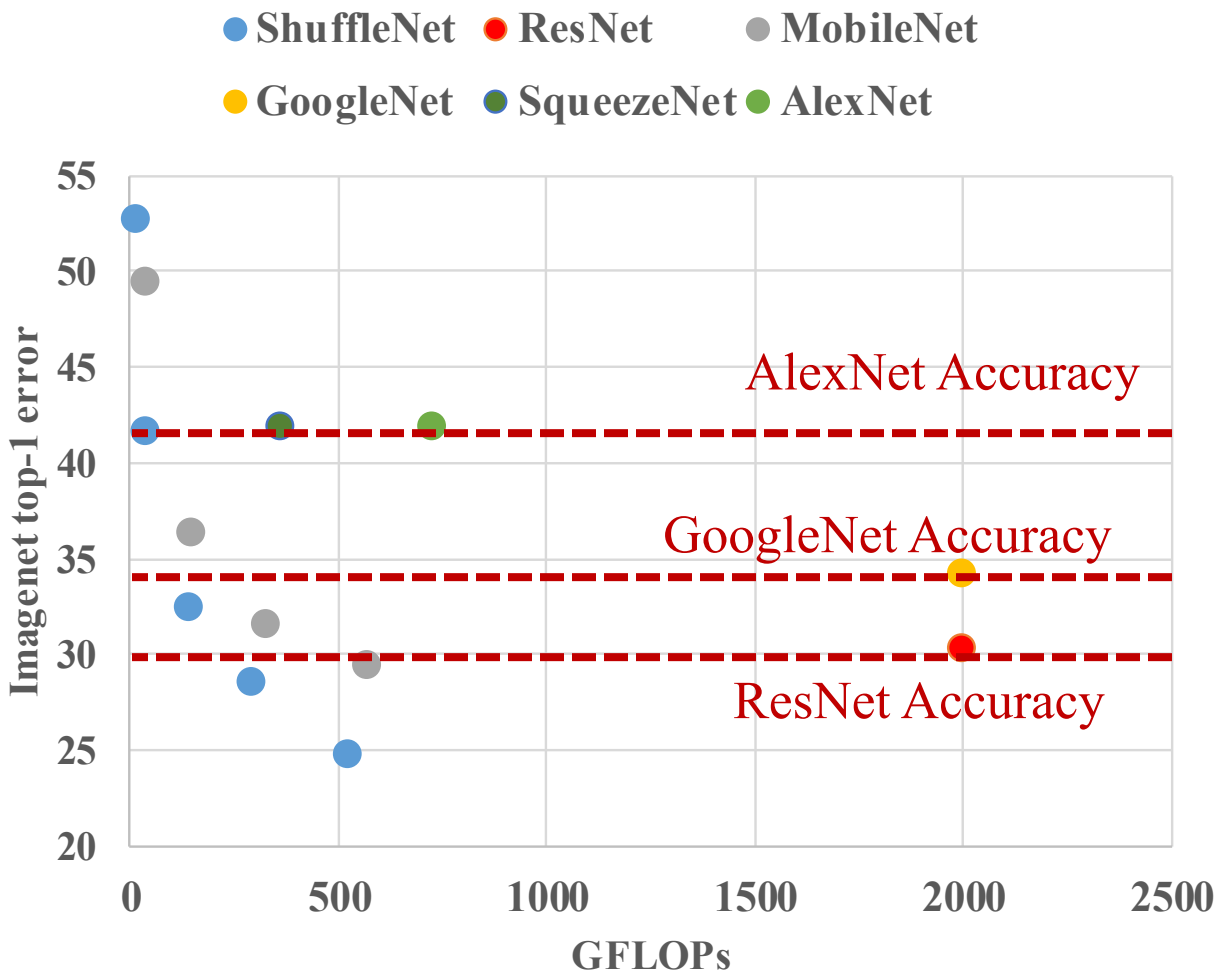
- Build on top of the bottleneck unit proposed by ResNet
- Group convolution: divide input channels and filters into G groups, cross talk is only allowed within the same group. Large FLOPs reduction with compromised accuracy
- Channel shuffle: shuffle the channel-specific feature maps in each unit to allow cross talk among groups. Enhance accuracy w/o increasing FLOPs



Group Conv | Group Conv + Channel Shuffle | Shuffle unit

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.

Comparison of ShuffleNet and MobileNet



FLOPs reduction
(within 2% accuracy drop)

- ShuffleNet: 8-14x
- MobileNet: 4-14x

- ShuffleNet has a better accuracy FLOPs tradeoff especially in the high-accuracy region
- Within 2% drop from ResNet accuracy, ShuffleNet and MobileNet achieves 8x and 4x FLOPs reduction, respectively.

Impact on FPGA Implementation

Evaluation on Stratix V 5GSD8 FPGA

	Efficient CNN Models		
	ResNet-50	MobileNet V1	MobileNet V2
# Ops (GOP)	7.74	1.14 **6.8x**	0.611 **12.7x**
# Param. (M)	25.5	4.21	3.47
Clock Freq. (MHz)	200	200	200
Bit width	16 bit	16 bit	16 bit
DSP usage	1680	1664	1856
Latency (ms)	7.95	0.884 **9.0x**	1.02 **7.8x**
Throughput (GOPS)	973.2	1287.2	592
Top-1 Accuracy	93.5	88.3	87.5

- MobileNet achieves 9x faster inference on FPGA than ResNet-50 with a accuracy drop of 5%
- MobileNet's building block is high structural and FPGA friendly, the FLOPs reduction is well-reflected by FPGA inference speed-up.

Zhao, Ruizhe, et al. "Towards efficient convolutional neural network for domain-specific applications on FPGA." 2018 28th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2018.

Techniques for Reducing Computational Complexity of DNNs

- Reduce total FLOPs
  - Pruning
  - Winograd (CNN-specific)
  - Simplified building blocks (CNN-specific)

- Reduce numerical precision
  - Quantization
  - Ternary/Binary Nets

Quantization Is Pervasive
Widely Used in DSP HW Design

- Floating-point and fixed-point data format is efficient in representing a large and limited dynamic range, respectively
  - General-purpose hardware adopts floating-point data formats
  - Application-specific and dedicated hardware adopts fixed-point data formats

- Minimizing numerical precision is critical to HW design
  - HW cost and power efficiency (add/mult complexity and memory footprint, power) is proportional to the word-length of a fixed-point data format.

- The minimum word-length needed for HW design depends on
  - Dynamic range of input data/signal
  - Operations type and iteration bound in the processing pipeline
  - Error tolerance or precision requirement of the final results

Good Reasons to Quantize A DNN

- A real-valued network is typically numerically redundant
  - DNNs are application-specific and deals with input data/signal with a limited dynamic range and have deterministic operators and iteration bound as well as error tolerance to quantization noise

- Can be combined with FLOPs reduction techniques and broadly applicable to a wide range of DNN models

- Smaller model size and memory footprint, better cache reuse
- Increase the parallelism of computation and inference speed
  - GPUs and FPGAs can trade off parallelism with numerical precision
  - GPUs: double, single, half, INT8, INT4, bit-wise
  - FPGA supports all numerical precision due to hardware reconfigurability, and is highly efficient for single and all fixed-point, and bit-wise arithmetic.

Quantization Key Concepts

- Analyze the statistics of model parameters to determine either the optimized wordlength or quantizers (given pre-defined word-length) that minimizes the accuracy drop of the network
  - A quantizer is defined by its scale Δ and zero-point, can be asymmetric or symmetric depending on where the zero-point is
  - Optional fine-turning can be followed to compensate for accuracy drop

- Weights and activations must be analyzed separately and can use different word-length and/or quantizers
  - Activation quantization analysis require forward propagation with enough validation data (weight quantization analysis does not)

- Quantization can be done at different granularities
  - E.g. in CNNs, weights in convolutional layers, weights in fully connected layers, and activations can use different word-length and/or quantizers [P. Gysel, et al. ICLR 2016]

Quantization Key Concepts

- Weight/activation quantization of CNNs can be further optimized at a finer granularity
  - To optimize the quantizer parameters, Δ and zero-point for each unit

- Per-layer quantization: the entire 4D tensor of convolutional kernels and the activations in one convolutional layer use the same quantizer, those of different convolutional layers use different quantizers.

- Per-channel quantization: each 3D convolutional kernel in each convolutional layer (responsible for producing one output feature map) uses a different quantizer
  - Per-channel activation quantization is typically not considered as it would complicate the computation of convolutions for the next layer.

Post-Training Quantization Approach: Fine-grained Quantization w/ Quantizer Opt.

1. Start with a fully trained model

2. Define a desired word-length of weights and activation (8 bit is generally a good number to start with)

3. Perform per-channel quantization on weights by optimizing the the quantizer (Δ and zero-point) per channel in each layer with asymmetric range to minimize the quantization errors

4. Perform per-layer quantization on activations by forward propagation with validation data and quantized weights and optimizing the quantizer of activations per layer to minimize the accuracy drop

Krishnamoorthi, Raghuraman. "Quantizing deep convolutional networks for efficient inference: A whitepaper." *arXiv preprint arXiv:1806.08342* (2018).

Post-Training Quantization Results on ImageNet (8-bit, 4x Model Size Saving)

Weight Quantization Schemes

Network	Asymmetric, per-layer	Symmetric, per-channel	Asymmetric, per-channel	Activation Only Per-layer	Floating Point
Mobilenet-v1_1_224	0.001	0.591	0.703	0.708	0.709
Mobilenet-v2_1_224	0.001	0.698	0.697	0.7	0.719
Nasnet-Mobile	0.722	0.721	0.74	0.74	0.74
Mobilenet-v2_1.4_224	0.004	0.74	0.74	0.742	0.749
Inception-v3	0.78	0.78	0.78	0.78	0.78
Resnet-v1_50	0.75	0.751	0.751	0.751	0.752
Resnet-v2_50	0.75	0.75	0.75	0.75	0.756
Resnet-v1_152	0.766	0.762	0.767	0.761	0.768
Resnet-v2_152	0.761	0.76	0.76	0.76	0.778

- Weight quantization is more sensitive to accuracy drop
- Asymmetric per-channel weight quantization + per-layer activation quantization produces the best accuracy
- 8-bit word-length lead to 0-2.2% accuracy drop
- Fine-tuning based on the quantized data format can be applied to compensate for the accuracy drop (similar to in-training quantization)

Krishnamoorthi, Raghuraman. "Quantizing deep convolutional networks for efficient inference: A whitepaper." *arXiv preprint arXiv:1806.08342* (2018).

In-training Quantization Approach

- Train a model from scratch with quantization effect being modeled
- Maintain weights and activation as floating-point numbers and quantize them in a simulated fashion (similar to post-training) for both forward and backward propagation to model the effect of quantization
  - The gradient value uses full-precision and is calculated with respect to the simulated quantized parameters
- Maintain full-precision shadow weights and update them with gradients so that small gradient values can be added up instead of underflowed
  - The simulated quantized weights are sampled from the full-precision shadow weights after each update

Krishnamoorthi, Raghuraman. "Quantizing deep convolutional networks for efficient inference: A whitepaper." *arXiv preprint arXiv:1806.08342* (2018).

In-training v.s. Post-training Quantization Results on ImageNet (8-bit, 4x Model Size Saving)

	Post-training quantization		In-training quantization		
Network	Asymmetric, per-layer (Post Training Quantization)	Symmetric , per-channel (Post Training Quantization)	Asymmetric, per-layer (Quantiza-tion Aware Training)	Symmetric, per-channel (Quantiza-tion Aware Training)	Floating Point
Mobilenet-v1_1_224	0.001	0.591	0.70	0.707	0.709
Mobilenet-v2_1_224	0.001	0.698	0.709	0.711	0.719
Nasnet-Mobile	0.722	0.721	0.73	0.73	0.74
Mobilenet-v2_1.4_224	0.004	0.74	0.735	0.745	0.749
Inception-v3	0.78	0.78	0.78	0.78	0.78
Resnet-v1_50	0.75	0.751	0.75	0.75	0.752
Resnet-v2_50	0.75	0.75	0.75	0.75	0.756
Resnet-v1_152	0.766	0.762	0.765	0.762	0.768
Resnet-v2_152	0.761	0.76	0.76	0.76	0.778

- In-training quantization provides the best accuracy and allows for simpler quantization schemes
- Asymmetric per-layer and symmetric per-channel can bound the accuracy drop to ≤1.8% and ≤1%, respectively, better than post-training w/ asymmetric per-channel

Krishnamoorthi, Raghuraman. "Quantizing deep convolutional networks for efficient inference: A whitepaper." *arXiv preprint arXiv:1806.08342* (2018).

Impact on FPGA Implementation

- Reduced numerical precision increases the computation parallelism due to HW flexibility of FPGA. Each 27-bit mult. In Stratix V can be used as two 18-bit mult.

Reference		1	2
Method		Quantization + Winograd	Quantization + SVD FC
Network		VGG-16	VGG-16
Dataset		CIFAR-10	ImageNet
Accuracy (%)		93.5	1.1% drop
Device		Altera Stratix V	Zynq XC7Z045
Weight bit-width		16	16
Act. bit-width		16	16
Frequency (MHz)		200	150
Throughput (GOP/s)		2561	136
Normalized Throughput (GOP/s)#		732	-
FPGA 27-bit DSP peak (FLOP/s)*		392.6	201
Actural thpt/Device peak		1.9x	0.7x
Latency (ms)		10.3	-
Power (W)		-	9.63
Energy efficiency (GOP/s/W)		-	14.22
Accelation ratio	**FPGA/CPU (actual)**	6x (i5-4590)	-

* FPGA DSP peak performance is scaled down by actual frequency

\# Normalized by the accelerator factor of Wingrad (3.5x) to cancel out the impact of Winograd method.

[1] Zhao, Ruizhe, et al. "Towards efficient convolutional neural network for domain-specific applications on FPGA." 2018 28th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2018.

[2] Qiu, Jiantao, et al. "Going deeper with embedded fpga platform for convolutional neural network." Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2016.

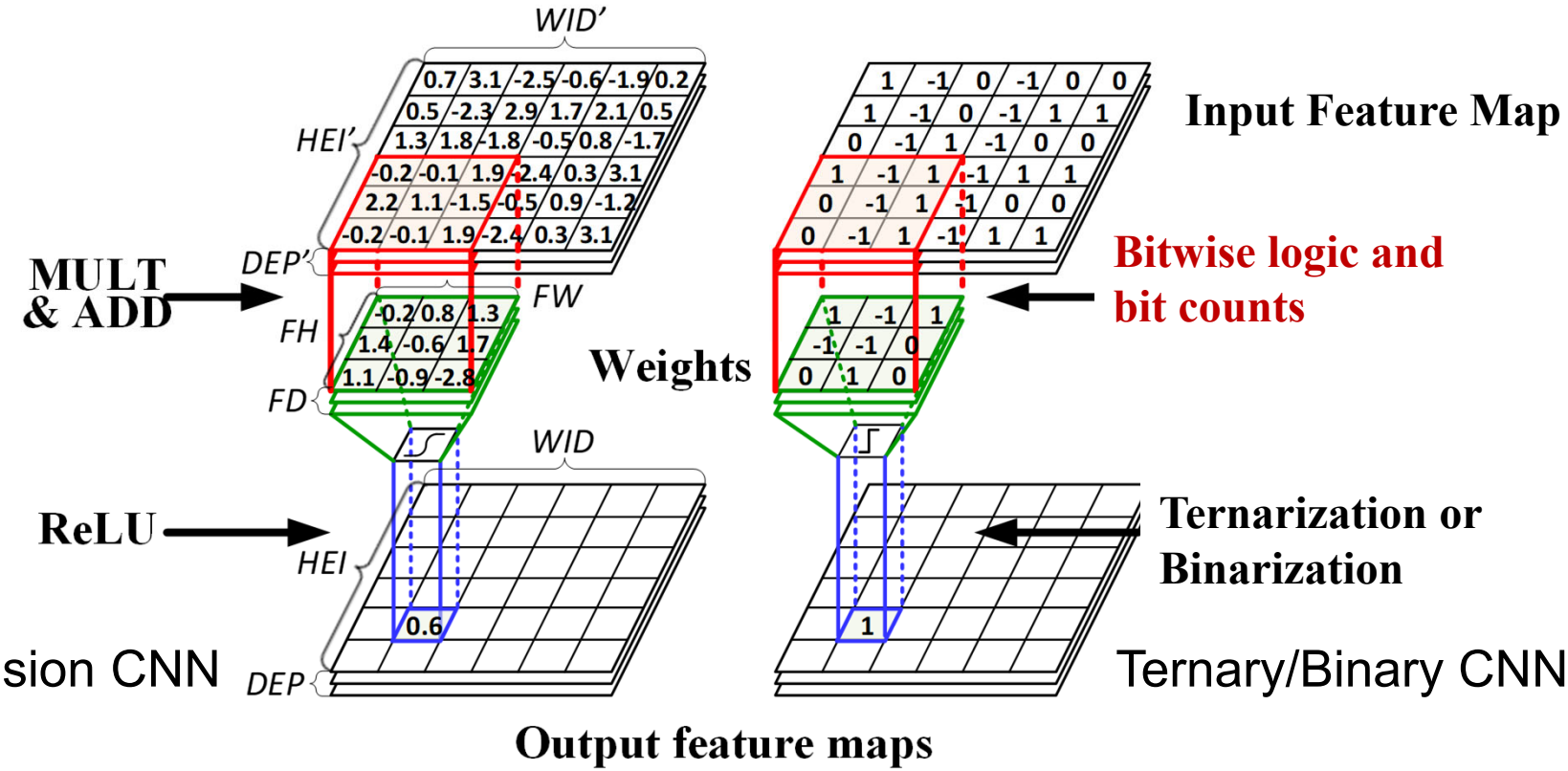
Techniques for Reducing Computational Complexity of DNNs

- Reduce total FLOPs
  - Pruning
  - Winograd (CNN-specific)
  - Simplified building blocks (CNN-specific)

- Reduce numerical precision
  - Quantization
  - Ternary/Binary Nets

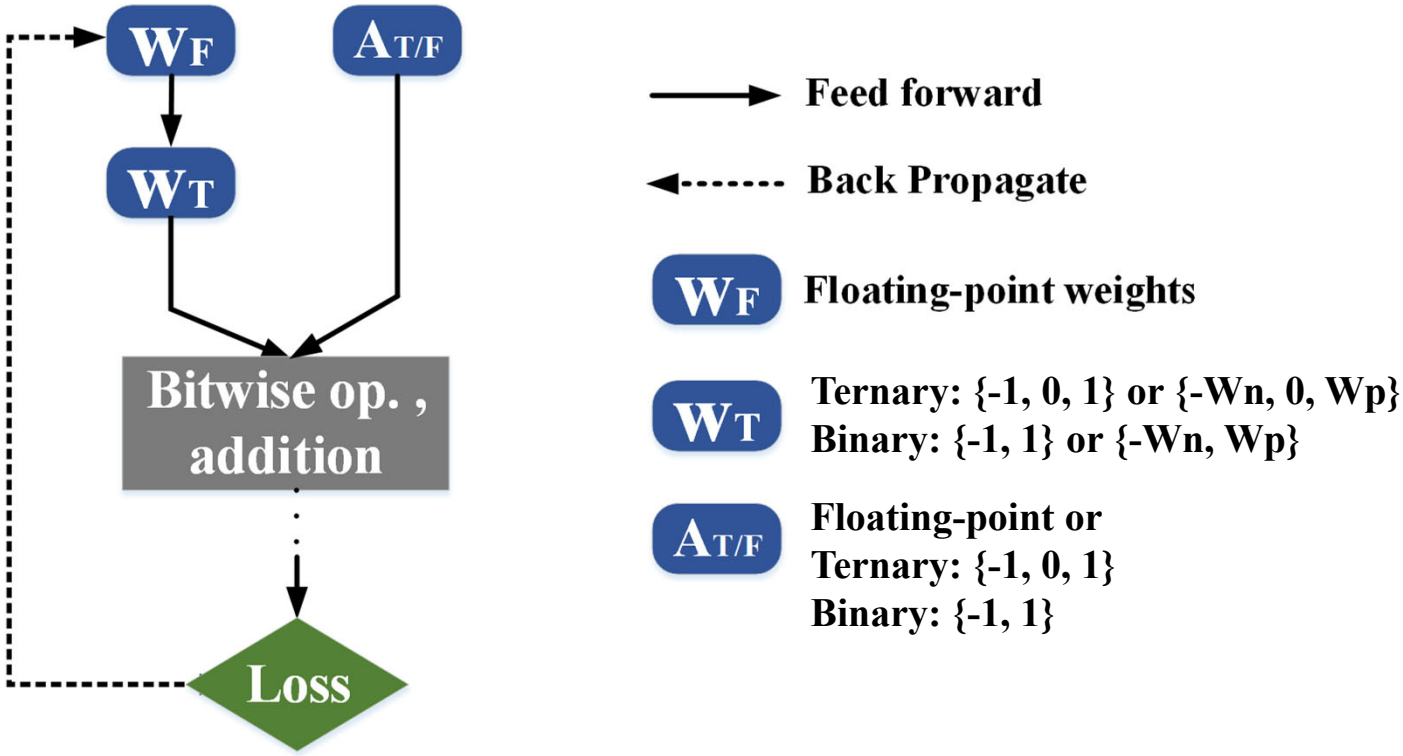
Ternary/Binary Nets: Ternary/Binary Weights and/or Activations

- A ternary/binary weight/activation just need 2/1 bit for representation
  - Many binary nets keep 1st-layer weights floating-point for accuracy
- Convolution becomes bitwise logic + short addition or bit count
- Significant reduction of computational complexity and model size
- Large impact on FPGA/ASIC performance (throughput, power, E. Eff.)
  - Enable massive parallelism and on-chip weight storage



T/BNN Training: In-training Quantization w/ Pre-defined or Learned Ternary/Binary Quantizers

- A ternary/binary quantizer is defined by {-Wn, 0, Wp}/{Wn, Wp} and can be symmetric or asymmetric
- The quantizers can be pre-defined as Wn=Wp=1 or optimized by learning
- The granularity of weight quantization can be global, per-layer, per-kernel and that of weight quantization is typically global or per-layer



Ternary Net Results

Method	Activations	Weights	Main Operations	Performance			
				MNIST LeNet-5	CIFAR-10 VGG-7	ImageNet AlexNet	ImageNet ResNet-18
Baseline	float	float	MUL, ADD	99.48	92.88	57.2/80.2	69.6/89.2
TWN[1]	float	{-Wn,0,+Wp}	ADD	99.38	92.56	54.5/76.8	65.3/86.2
TNN[2]	{-1,0,1}	{-1,0,1}	AND, bitcount	98.33	87.89	-	-
GXNOR[3]	{-1,0,1}	{-1,0,1}	AND, bitcount	99.32	92.5	-	-

- Pre-defined ternary quantizer can bound accuracy drop to 0.16% and 0.38% on MNIST and CIFAR-10, respectively.
- Learned ternary weights with floating activation can further enhance the accuracy and achieve 16x model size reduction with $\leq$ 4.3% accuracy drop on ImageNet.

[1] Li, Fengfu, Bo Zhang, and Bin Liu. "Ternary weight networks." NIPS (2016).
[2] Alemdar, Hande, et al. "Ternary neural networks for resource-efficient AI applications." 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017.
[3] Deng, Lei, et al. "Gated xnor networks: Deep neural networks with ternary weights and activations under a unified discretization framework." arXiv preprint arXiv:1705.09283 4.9 (2017).

Ternary Net's Impact on FPGA Implementation

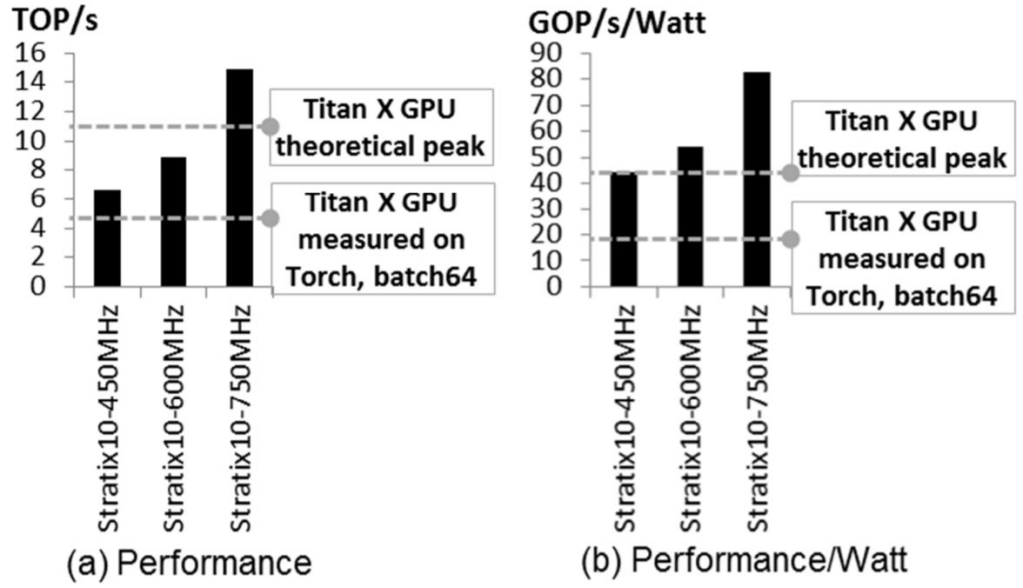


Figure 13. Ternary ResNet-50 results for ImageNet problem size, on Titan X GPU and Stratix 10 FPGA.

- FPGA implementation achieves 1.4x higher throughput and 2.4x better energy efficiency than GPU for <1% accuracy drop.
- For Ternary convolutions, bit-wise logics are mapped onto LUTs, but the floating-point accumulation are still mapped onto DSP slices.
- The computation parallelism is much improved but still limited by the amount of DSP resources available on an FPGA.

Nurvitadhi, E., et al. (2017, February). Can FPGAs beat GPUs in accelerating next-generation deep neural networks?. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays(pp. 5-14). ACM.

Binary Net Results

Method	Activations	Weights	Main operations	Performance			
				MNIST LeNet-5	CIFAR-10 VGG	ImageNet AlexNet	ImageNet ResNet-18
Baseline	float	float	MUL, ADD	99.48	92.88	57.2/80.2	69.6/89.2
[1]BWN	float	{-Wn,+Wp}	ADD	99.38	92.58	56.8/79.4	60.8/83
[2]BC	float	{-1,+1}	ADD	98.82	91.76	35.5/61.0	-
[3]BNN	{-1,+1}	{-1,+1}	XNOR, bitcount	98.6	89.86	27.9/50.42	-
[4]XNOR	{-Wn,+Wp}	{-Wn,+Wp}	XNOR, bitcount, ADD	99.21	90.02	44.2/69.2	55.9/79.1
[5]DoReFa	{0,1}	{0,1}	XNOR, bitcount	-	-	47.7/-	-
[6]TBN	{-1,0,+1}	{-Wn,+Wp}	AND, XNOR, bitcount	98.38	90.85	55.6/79.0	58.2/81.0
[7]LQ-Net	2 bits	{-Wn,+Wp}	AND, XNOR, bitcount	-	-	-	62.6/84.3

- About 32x model size reduction from single-precision to 1 bit.
- Learned binary weights and activations can bound accuracy drop to 0.3% and 2.8% on MNIST and CIFAR-10, respectively. Use ternary activations can further improve the accuracy.
- Learned binary weights with learned ternary activations can bound accuracy drop to 7% on ImageNet.

[1] Rastegari, Mohammad, et al. CVPR, 2016. [2] Courbariaux, Matthieu, et al. Advances in neural information processing systems. 2015. [3] Courbariaux, Matthieu, et al., arXiv preprint arXiv:1602.02830 (2016). [4] Rastegari, Mohammad, et al., ECCV, 2016. [5] Zhou, Shuchang, et al., arXiv preprint arXiv:1606.06160 (2016). [6] Wan, Diwen, et al., ECCV, 2018. [7] Zhang, Dongqing, et al., ECCV, 2018.

Impact on FPGA Implementation

	1	2	3
Network	AlexNet	AlexNet	AlexNet
Dataset	CIFAR-10	CIFAR-10	CIFAR-10
Accuracy (%)	88%	89%	81%
Device	Xilinx Virtex-7	Zynq 7Z020	Xilinx Virtex-7
Weight bit-width	1	1	1
Act. bit-width	1	1	1
Frequency	90	143	2000
Throughput (GOP/s)	7663	207.8	11600
Device peak (GOP/s)*	337	92	748
Throughput/Device peak(%)	23x	1.6x	16x
Latency (ms)	-	5.94	0.55
Power (W)	8.2	4.7	-
Energy efficiency (GOP/s/W)	935 GOPS/s	35.8 imgs/s/W	-

* FPGA DSP peak performance is scaled down by actual frequency

- Binary convolutions can be fully mapped onto LUTs and enable massive computation parallelism on FPGA
- Can easily achieve a super high throughput that is >20x higher than the peak full-precision DSP throughput

[1] Li, Yixing, et al. "A GPU-outperforming FPGA accelerator architecture for binary convolutional neural networks." ACM Journal on Emerging Technologies in Computing Systems (JETC) 14.2 (2018): 18.
[2] Zhao, Ritchie, et al. "Accelerating binarized convolutional neural networks with software-programmable fpgas." Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017.
[3] Umuroglu, Yaman, et al. "Finn: A framework for fast, scalable binarized neural network inference." Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017.

Zoom Into [1] From The Previous Table Binary Net FPGA Implementation Details

- A binary convolution is XNOR + bit-count
- All convolution operations can be fully mapped onto LUT resource—the key to enable massive parallelism on FPGAs
- To also utilize the available DSP recourses, convolution can be folded proportionally (based on a throughput constraint)—the key to maximize FPGA resource utilization

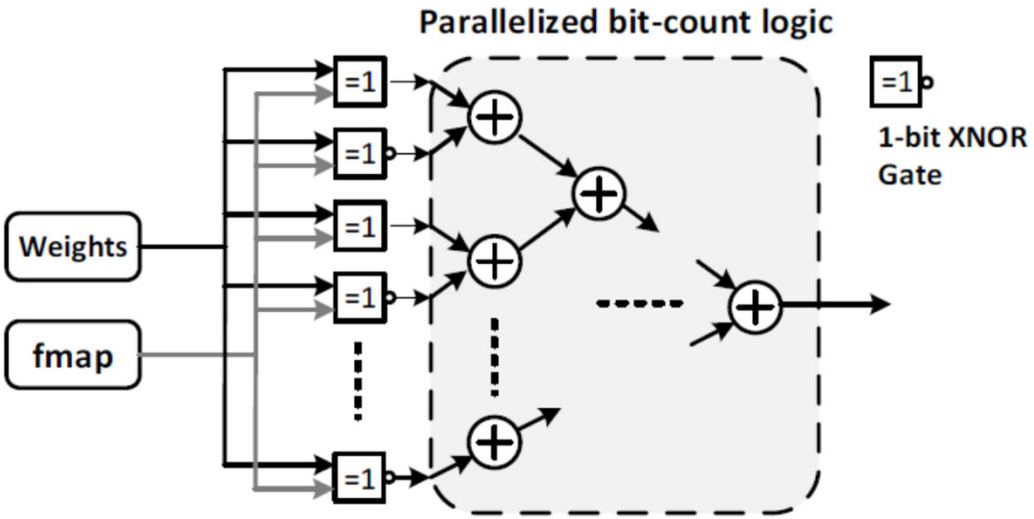
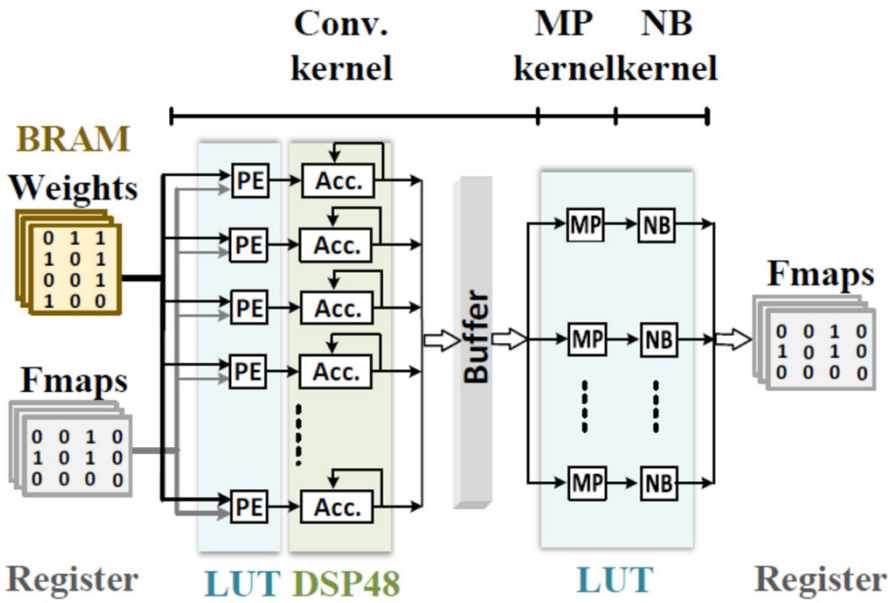


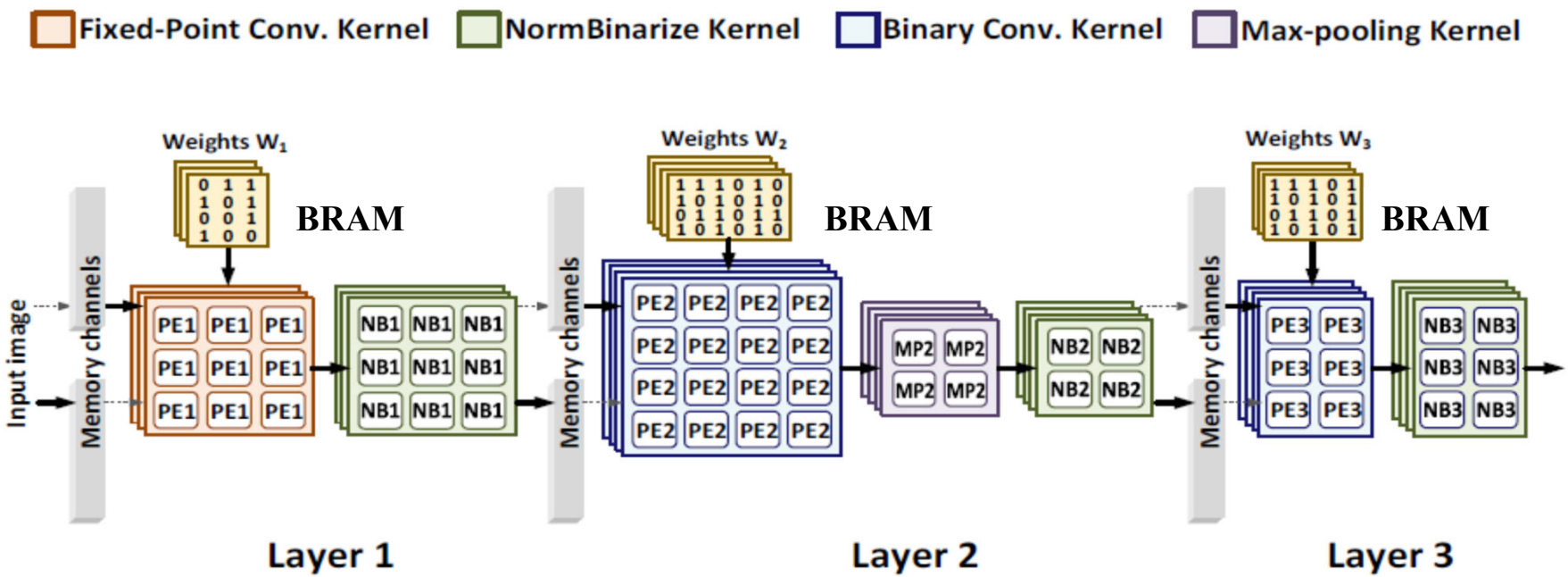
Fig. 5. Processing element (PE).



Li, Yixing, et al. "A GPU-outperforming FPGA accelerator architecture for binary convolutional neural networks." ACM Journal on Emerging Technologies in Computing Systems (JETC) 14.2 (2018): 18.

Zoom Into [1] From The Previous Table Binary Net FPGA Implementation Details

- Huge amount of LUT resources allow us to explore both spatial and temporal (pipeline) parallelism
  - All layers are mapped onto FPGA with optimized folding factors
  - Double buffering memory channel is used between adjacent layers to create a deep pipelined streaming architecture
- All binary weights are stored in BRAM on-chip
  - Avoid large power consumption and bandwidth bottleneck of DRAM



Li, Yixing, et al. "A GPU-outperforming FPGA accelerator architecture for binary convolutional neural networks." ACM Journal on Emerging Technologies in Computing Systems (JETC) 14.2 (2018): 18.

Zoom Into [1] From The Previous Table Binary Net FPGA Implementation Details

- For a deep pipelined streaming architecture, the throughput is determined by the slowest layer
  - The optimized architecture should have equal processing latency/throughput for all layers.

$$throughput = \frac{max(C_1, C_2, C_3 \ldots, C_k)}{freq},$$

- The optimal folding factor and maximum parallelism that balance the throughput across all layers are chosen

Table 3. Optimized parameters for each layer

Layer	UF	P	$Cycle_{conv}$	$Cycle_{est}$	$Cycle_r$
Conv 1	27	32	3538944	4096	5233
Conv 2	384	32	150994944	12288	12386
Conv 3	384	16	75497472	12288	12296
Conv 4	768	16	150994944	12288	13329
Conv 5	768	8	75497472	12288	12386
Conv 6	1536	8	150994944	12288	14473

Zoom Into [1] From The Previous Table Binary Net FPGA Implementation Details

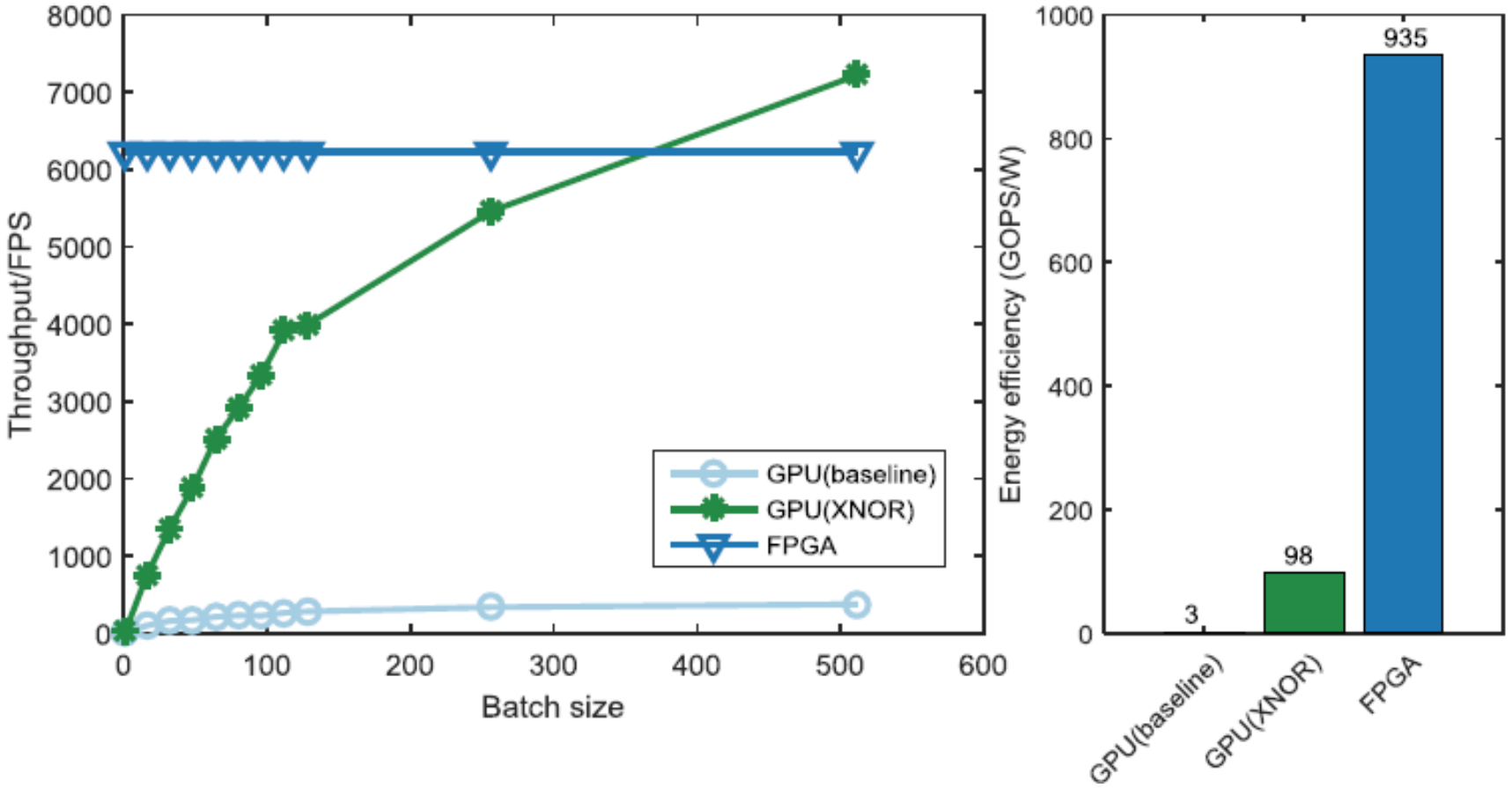
- The computation parallelism is bounded by LUT (100,000s) not DSP (1,000s) as the binary convolution is largely mapped onto LUTs
- This is the key to enable massive parallelism of computation on FPGAs
  - Each 6-input LUT can map 2.5 XNORs, peak device binary conv parallelism is about 433,200×2.5≈1,000,000

Table 4. FPGA resource utilization summary

Resource	LUTs	BRAMs	Registers	DSP
Used	342126	1007	70769	1096
Available	433200	2060	607200	2800
Utilization/%	78.98	48.88	14.30	39.14

Zoom Into [1] From The Previous Table Binary Net FPGA Implementation Results

- 8.3x faster and 75x more energy-efficient than a Titan X GPU for processing small batch size of 1



Throughput and energy efficiency comparison with GPU implementations

Li, Yixing, et al. "A GPU-outperforming FPGA accelerator architecture for binary convolutional neural networks." ACM Journal on Emerging Technologies in Computing Systems (JETC) 14.2 (2018): 18.

Hardware-friendliness Comparison

Reference			1	2	3	1	4	5	6
Method			MobileNet	TernaryNet	Pruning	Quantization	Quantization	Winograd	BinaryNet
Network			mobilenet	ResNet-50	LSTM	VGG-16	VGG-16	VGG	AlexNet
Dataset			CIFAR-10	ImageNet	TIMIT	CIFAR-10	ImageNet	ImageNet	CIFAR-10
Accuracy (%)			88.3	~1% drop	89.3	93.5	1.1% drop	-	87.80%
Device			Altera Stratix V	Altera Stratix 10	Xilinx XCKU060	Altera Stratix V	Zynq XC7Z045	Xilinx ZCU102	Xilinx Virtex-7
Weight bit-width			16	2	16	16	16	16	1
Act. bit-width			16	6	16	16	16	16	1
Weight on chip			No	No	No	No	No	No	Yes
Frequency (MHz)			200	450	200	200	150	200	90
Throughput (GOP/s)			1287	6500	282	2561	136	2479	7663
FPGA DSP peak (GOP/s)*			2500	5000	1327	826	201	1496	337
Actual throughput/Device peak			1.5x	1.3x	0.2x	3.1x	0.7x	1.7x	23x
Latency (ms)			0.8	-	0.0827	10.3	-	-	-
Power (W)			-	-	41	-	9.63	23.6	8.2
Energy efficiency (GOP/s/W)			-	40	32	-	14.22	105.4	935
Accelation ratio	**CPU**	**FPGA/CPU (actual)**	-	-	43x (core i7 5930k)	6x (i5-4590)	-	-	-
	GPU	**GPU device**	-	Titax X	Pascal Titan X	-	-	Titan X	Titan X
		FPGA/GPU (peak)	-	1.4x	0.12x	-	-	0.2x	0.048x
		FPGA/GPU (actual)	-	1.5x	3x	-	-	0.6x	10x

* FPGA DSP peak performance is scaled down by actual frequency

- Quantization are used in combination with all other methods
- Winograd is equally friendly to GPU and FPGA
- Non-structural pruning is more FPGA friendly than GPU.
- BinaryNet is extremely FPGA friendly—enable massive parallelism on FPGAs.

Reference of Previous Page

[1] Zhao, Ruizhe, et al. "Towards efficient convolutional neural network for domain-specific applications on FPGA." 2018 28th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2018.

[2] Nurvitadhi, E., et al. (2017, February). Can FPGAs beat GPUs in accelerating next-generation deep neural networks?. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays(pp. 5-14). ACM.

[3] Han, Song, et al. "Ese: Efficient speech recognition engine with sparse lstm on fpga." Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017.

[4] Qiu, Jiantao, et al. "Going deeper with embedded fpga platform for convolutional neural network." Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2016.

[5] Liang, Yun, et al. "Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2019).

[6] Li, Yixing, et al. "A GPU-outperforming FPGA accelerator architecture for binary convolutional neural networks." ACM Journal on Emerging Technologies in Computing Systems (JETC) 14.2 (2018): 18.

Module 3 Conclusion

- Cloud is insufficient for supporting the fast-growing IoT.
- Edge computing will become main-stream and pervasive in the era of IoT.
- FPGAs are highly suitable for and will play an important role in edge computing.
  - Due to architectural adaptiveness to algorithm characteristics, capability to explore both spatial and temporal parallelism, performance invariance to small batch size, high throughput and energy efficiency for I/O data processing.
- The high computational complexity of DNN models is a big challenge for edge deployment.

Module 3 Conclusion

- The use of a combination of techniques for reducing computational complexity of DNN models can significantly improve the inference performance on edge.
- Non-structural pruning is more FPGA friendly, while structural pruning are friendly to both FPGAs and GPUs.
- Winograd and simplified building blocks reduce computations in a highly structural way thus are equally friendly to GPUs and FPGAs.
- Post-training or In-training quantization with 8 bits should be applied in combination with other methods whenever applicable.

Module 3 Conclusion

- Binary Nets are extremely FPGA friendly.
- Binary Nets enable massive computation parallelism and on-chip weight storage, which can provide GPU-outperforming performance in terms of both throughput and energy efficiency.
- Binary Nets should be your top choice for edge deployment on FPGAs as along as a satisfactory accuracy can be achieved for your target application.

Acknowledgement

- Thanks to Ms. Yixing Li for her kind assistance in creating the slides of Module 3!

Thank you for your attention!

Questions?