Basic Design Approaches to Accelerating Deep Neural Networks

Rangharajan Venkatesan
NVIDIA Corporation

Live Q&A Session: Feb. 13, 2020, 7:00-7:20am, PST

Contact Info
email: rangharajanv@nvidia.com

Self-Introduction

- B.Tech in Electronics and Communication Engineering from IIT Roorkee in 2009
- Ph.D. in Electrical and Computer Engineering from Purdue University in 2014
- Joined NVIDIA
  - Research Scientist: 2014-2016
  - Senior Research Scientist: 2016-Present

- Research Interests
  - Machine Learning Accelerators
  - High-Level Synthesis
  - Low Power VLSI design
  - SoC Design methodologies

About this Tutorial

- Overview of design approaches to improve hardware efficiency
- Focus on inference
  - Most of the techniques are generic and applicable to training as well
- Does cover ..
  - Key metrics
  - Design considerations
  - Hardware optimizations
  - Hardware/software co-design techniques
- Does not cover ..
  - Algorithms and model design
  - Detailed software stack

Outline

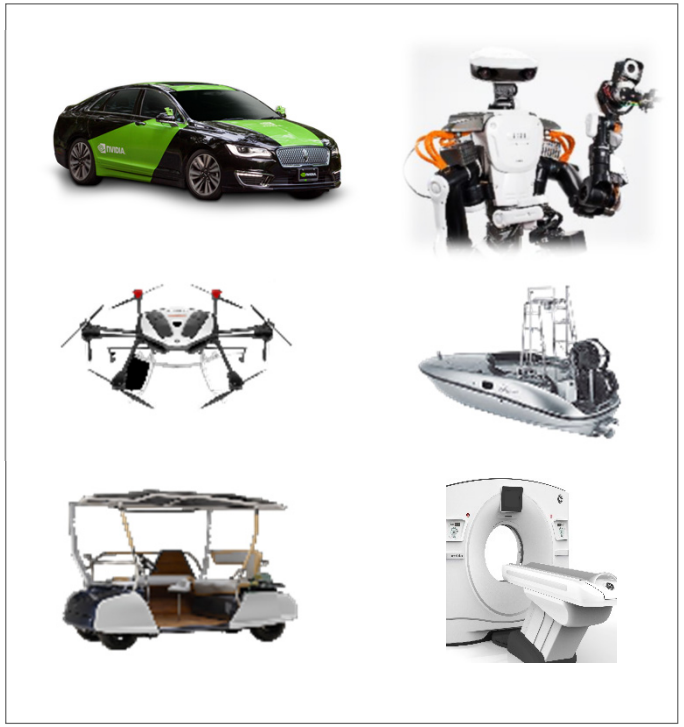
- Motivation for Hardware Acceleration
- Deep Neural Networks (DNN): Basics
- Accelerator Architecture Design
  - Compute: Efficient MAC Datapath
  - Storage: Memory Hierarchy
  - Communication: Interconnect Topologies
- Compiler Mapping Flow
  - Data reuse, Parallelism, Pipelining
- Neural Network Optimizations
  - Quantization
  - Sparsity

Outline

- Motivation for Hardware Acceleration
- Deep Neural Networks (DNN): Basics
- Accelerator Architecture Design
  - Compute: Efficient MAC Datapath
  - Storage: Memory Hierarchy
  - Communication: Interconnect Topologies
- Compiler Mapping Flow
  - Data reuse, Parallelism, Pipelining
- Neural Network Optimizations
  - Quantization
  - Sparsity

Vast World of DNN Applications

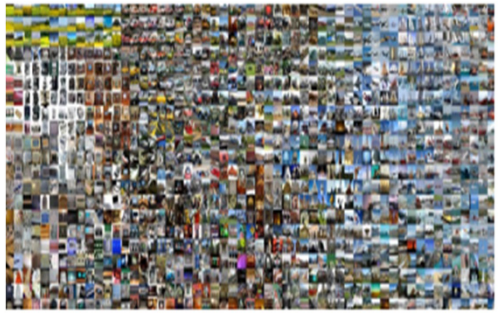
DATA CENTER



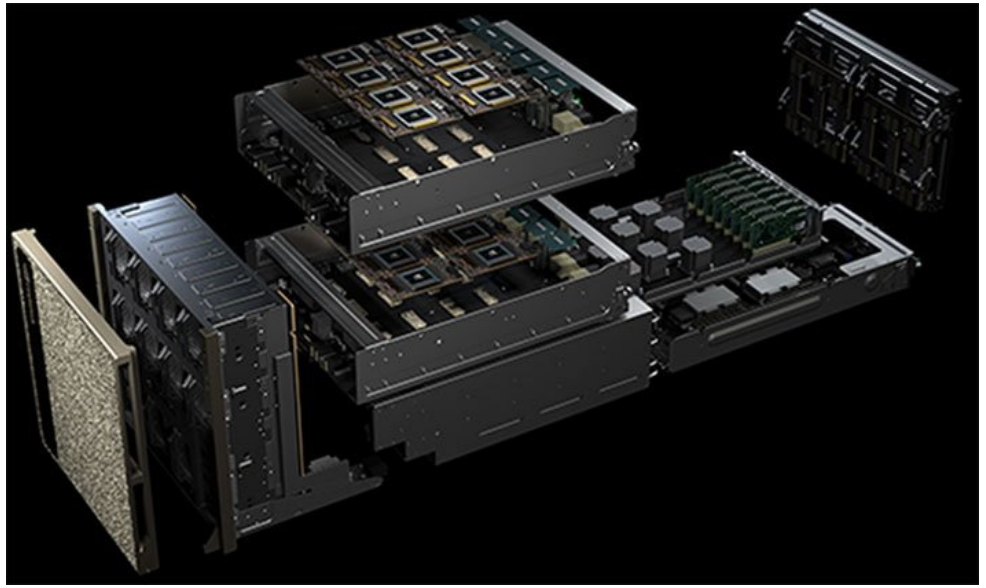
EMBEDDED COMPUTERS

EMBEDDED DEVICES

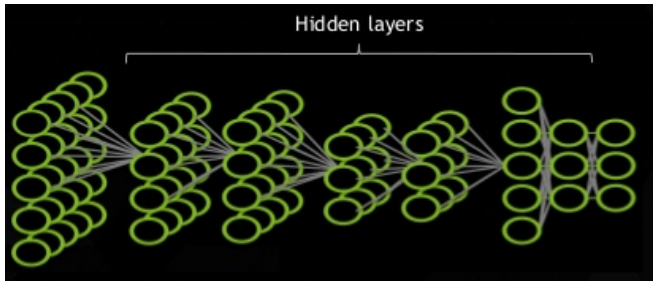
Hardware-enabled AI Revolution



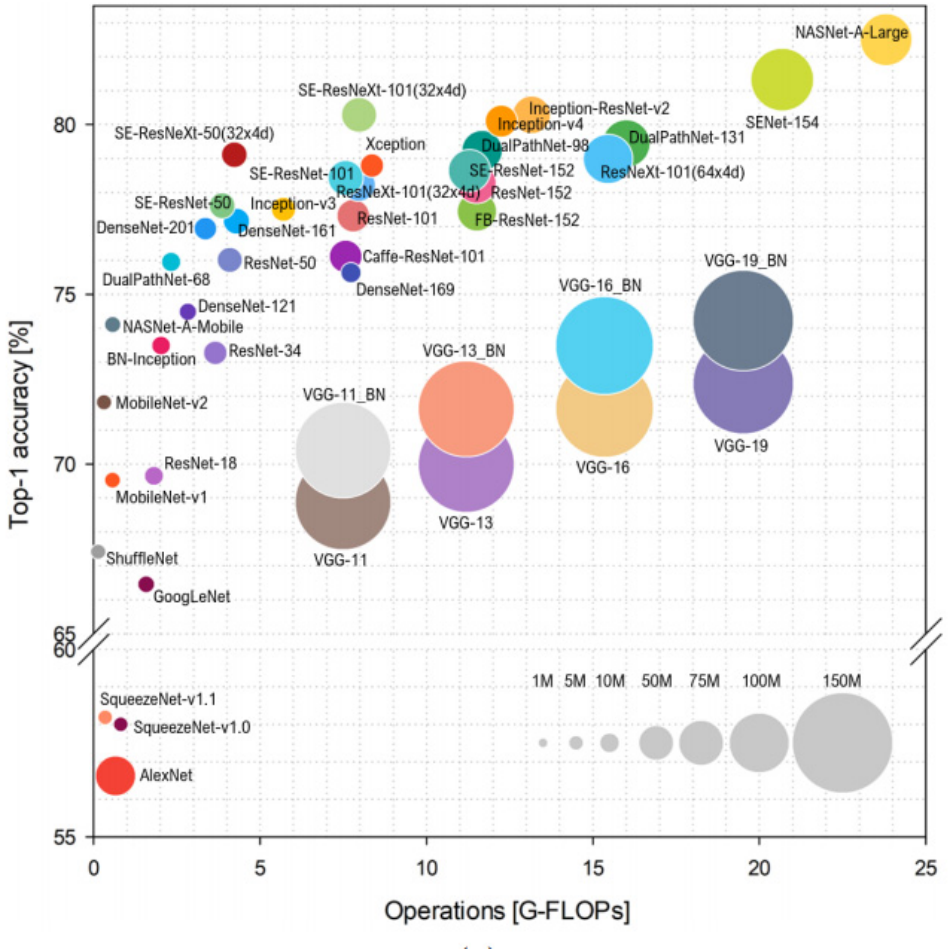
Data

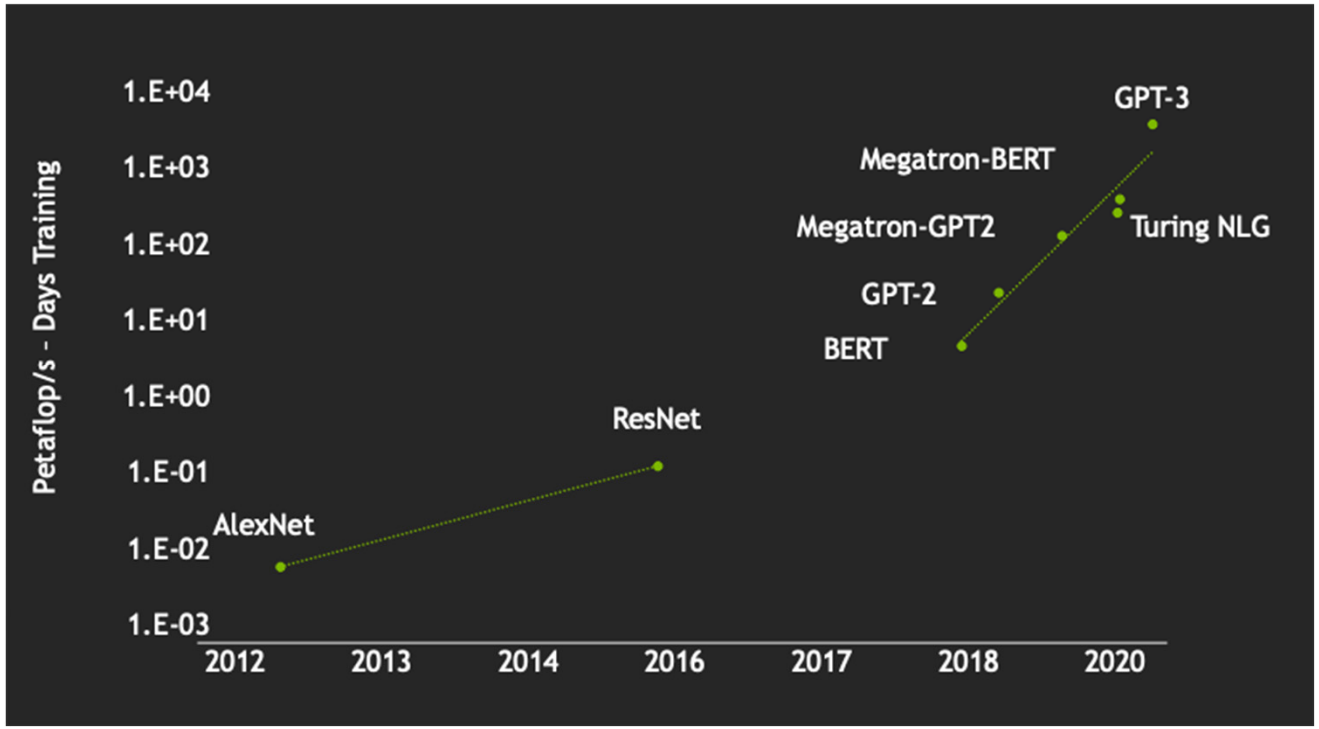


Efficient Compute



Growth in Application Complexity

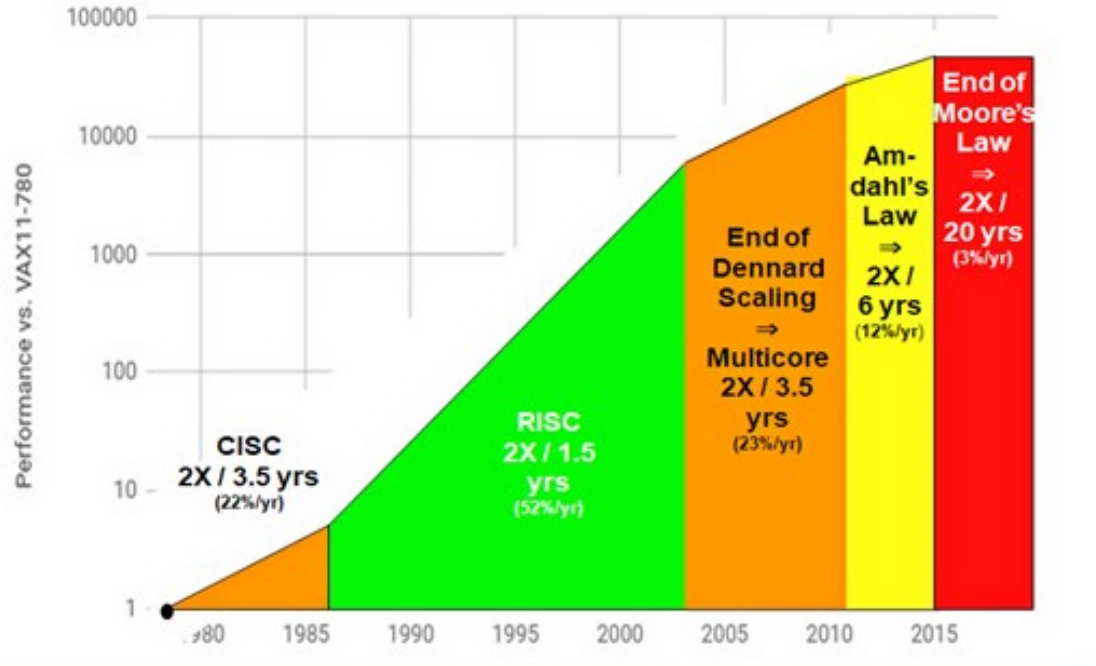




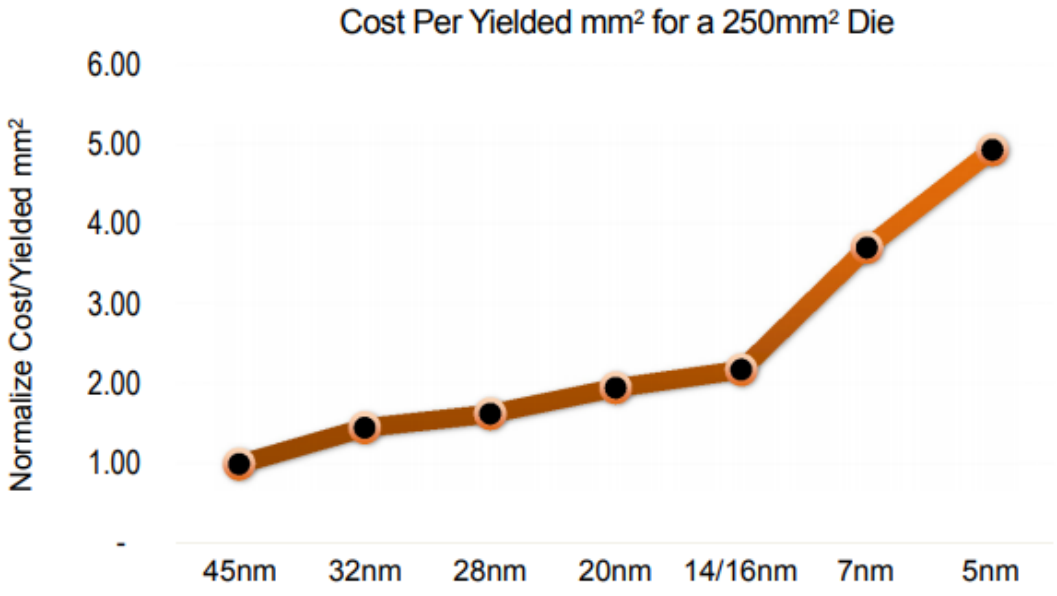
Bianco et al., *IEEE Access, 2018.*

Ack: Bill Dally, *GTC China, 2020.*

Hardware Performance Challenges



End of Moore's Law



Increasing cost

John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018

S. Naffziger et al., *ISSCC 2020*

Energy Efficiency Challenges

Super-human performance at low energy efficiency

1920 CPUs, 280 GPUs, **Power: ~300,000 W**

Google AlphaGo vs. Lee Sedol

Ack: Anand Raghunathan, *Purdue University*

Ref: "Showdown". *The Economist*, 19 Nov. 2016

Solution: Hardware Specialization

Reconfigurable FPGAs

✓Leverage reconfigurability of FPGA to accelerate a specific neural network

Accelerators

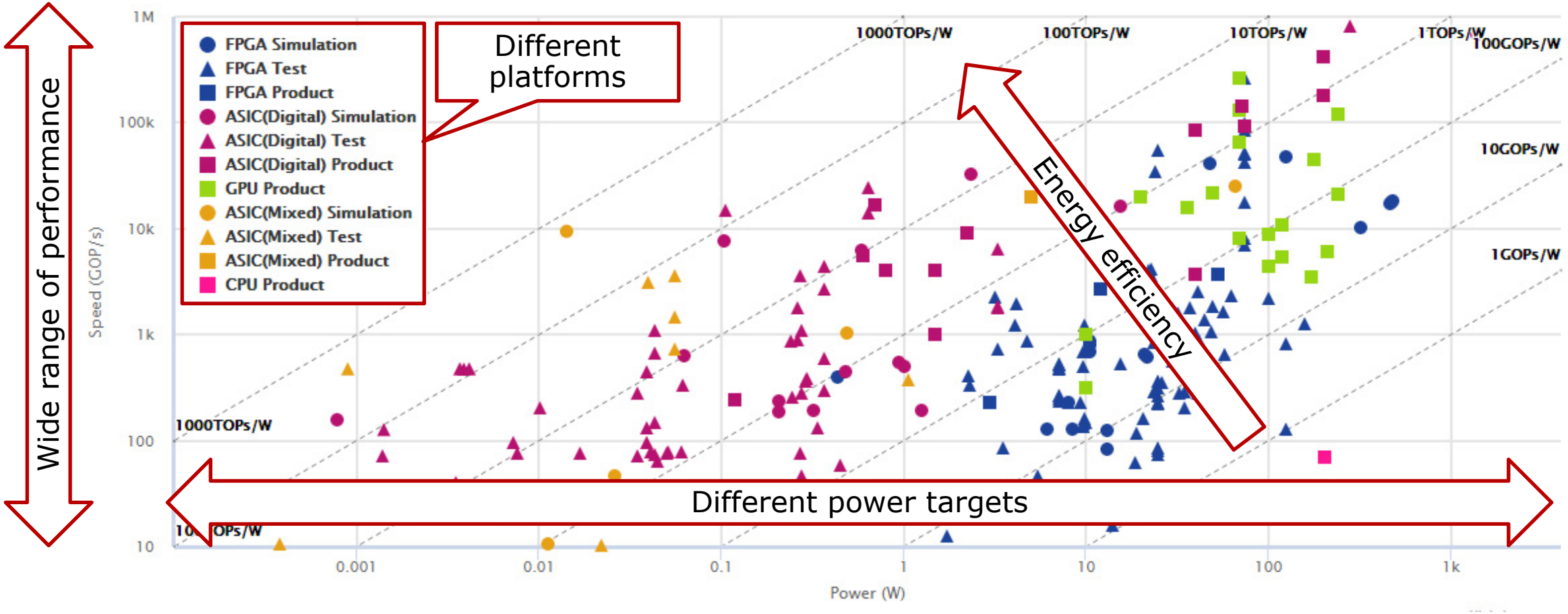
Programmable Processors

✓Programmable hardware with support for scalar and vector math functions

Fixed Function Accelerators

✓Customized hardware accelerators to support a class of neural networks

Many DNN Accelerators Exist!!

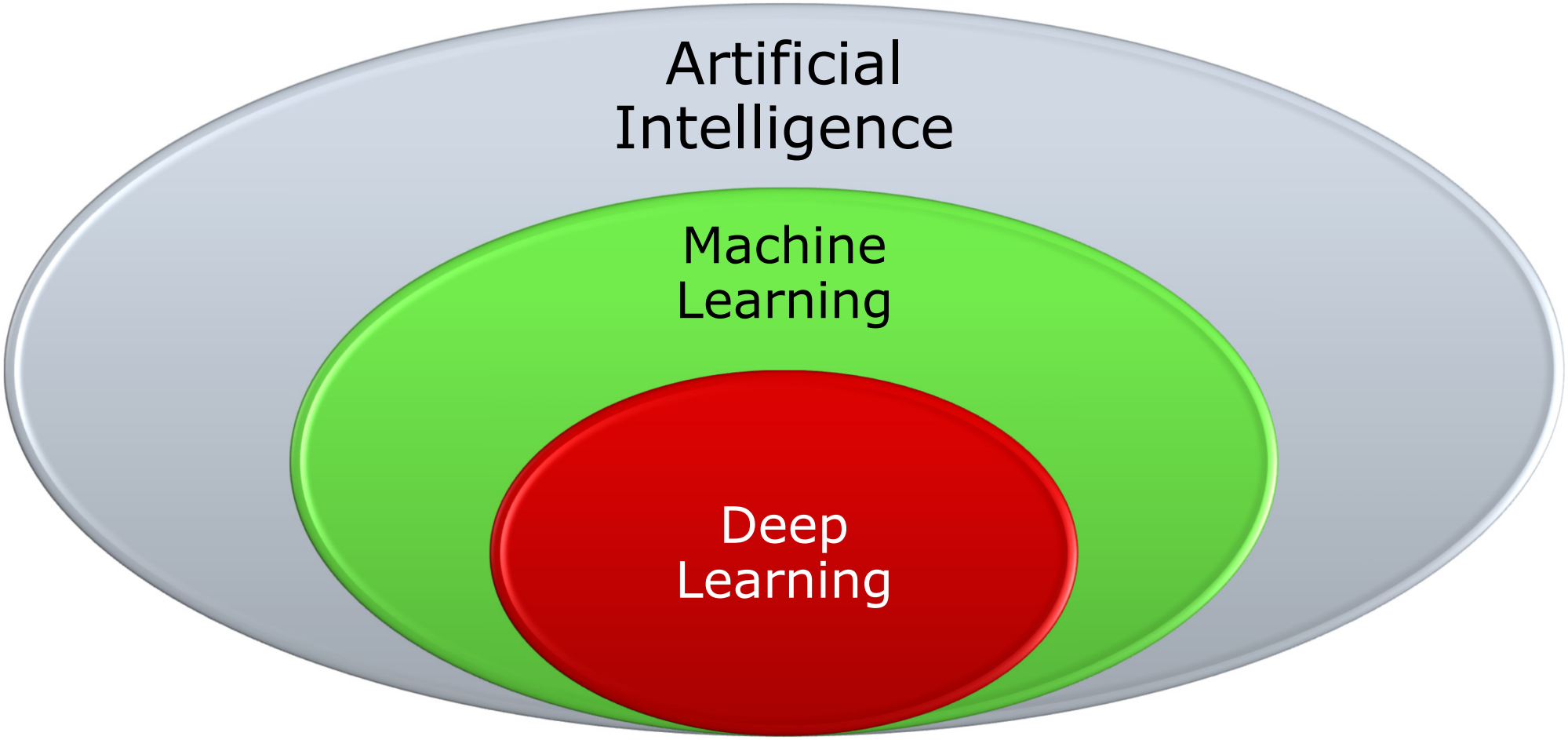


Source: https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/

Outline

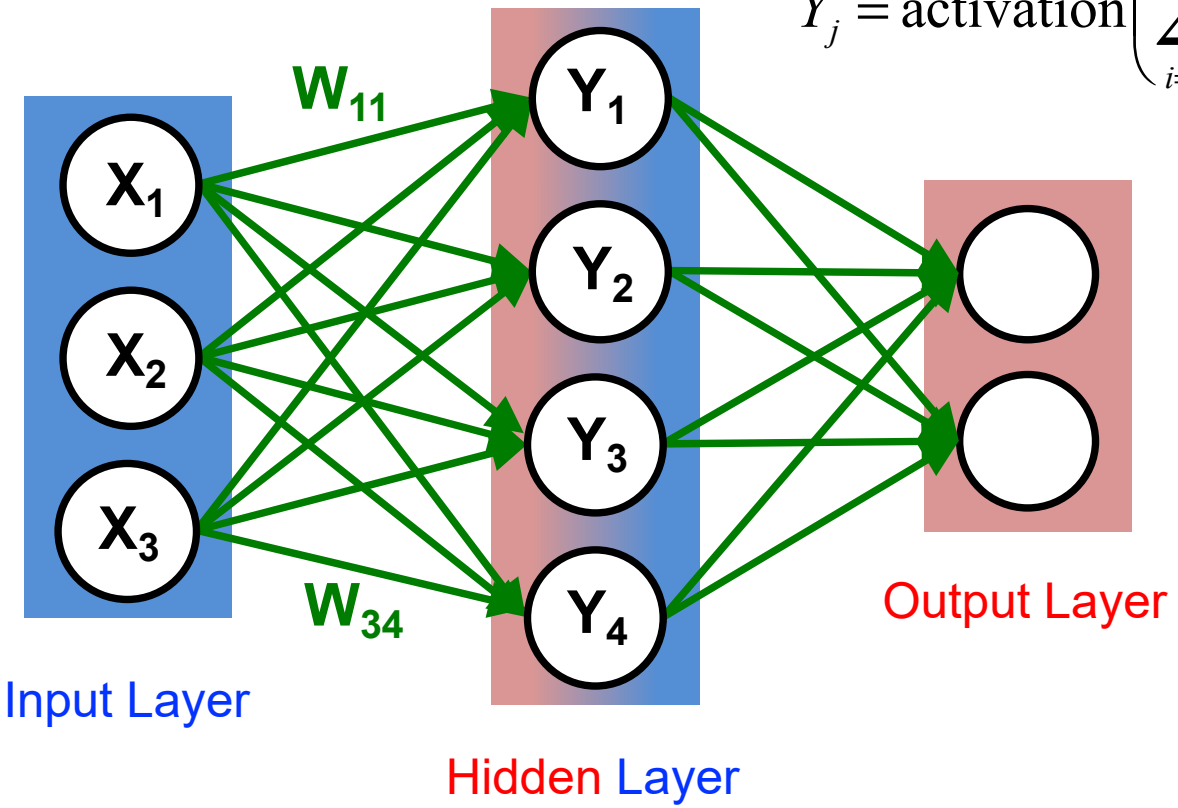
- Motivation for Hardware Acceleration
- Deep Neural Networks (DNN): Basics
- Accelerator Architecture Design
  - Compute: Efficient MAC Datapath
  - Storage: Memory Hierarchy
  - Communication: Interconnect Topologies
- Compiler Mapping Flow
  - Data reuse, Parallelism, Pipelining
- Neural Network Optimizations
  - Quantization
  - Sparsity

Deep Learning (aka DNN)



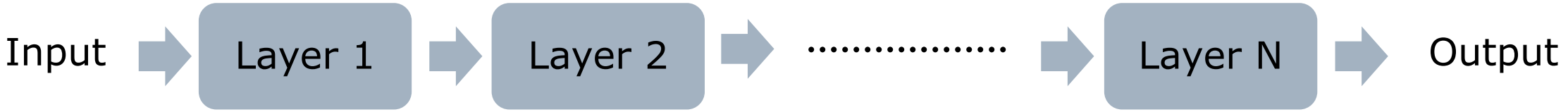
DNN: A Simple Example

$$Y_j = \text{activation}\left(\sum_{i=1}^{3} W_{ij} \times X_i\right)$$



Sze et al., *Synthesis Lectures on Computer Architecture, 2020*

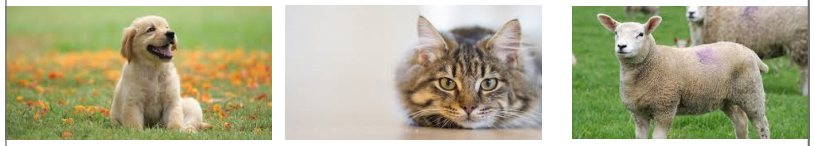
Practical DNNs



- 10s to 100s of layers
- Different types of layers
  - Convolution, Fully-connected, Depthwise-Separable, Attention etc.
- Process several Gigabytes of data
- Perform billions to trillions of operations
- Wide range of applications
  - Image processing, language processing, etc.

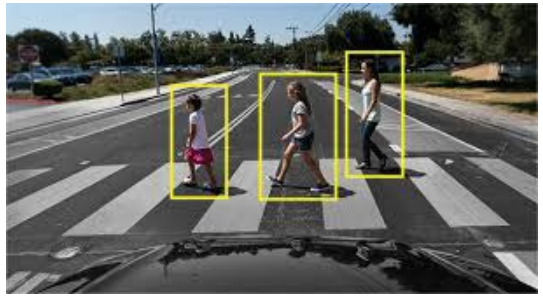
Examples of DNN Applications

Image Classification

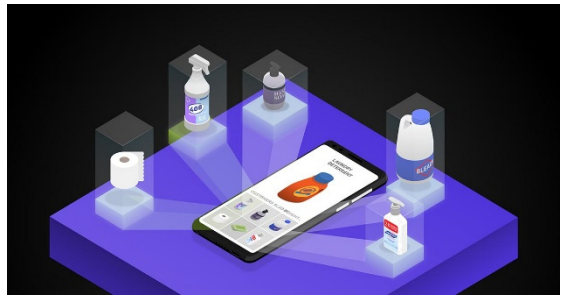


Dog Cat Sheep

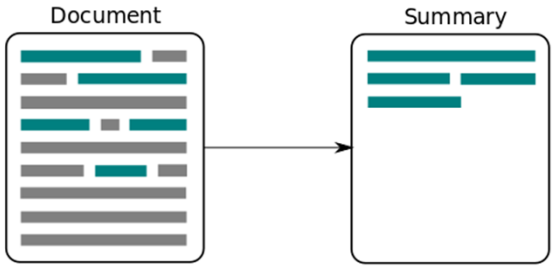
Object Detection



Recommendation Systems



Text Summarization



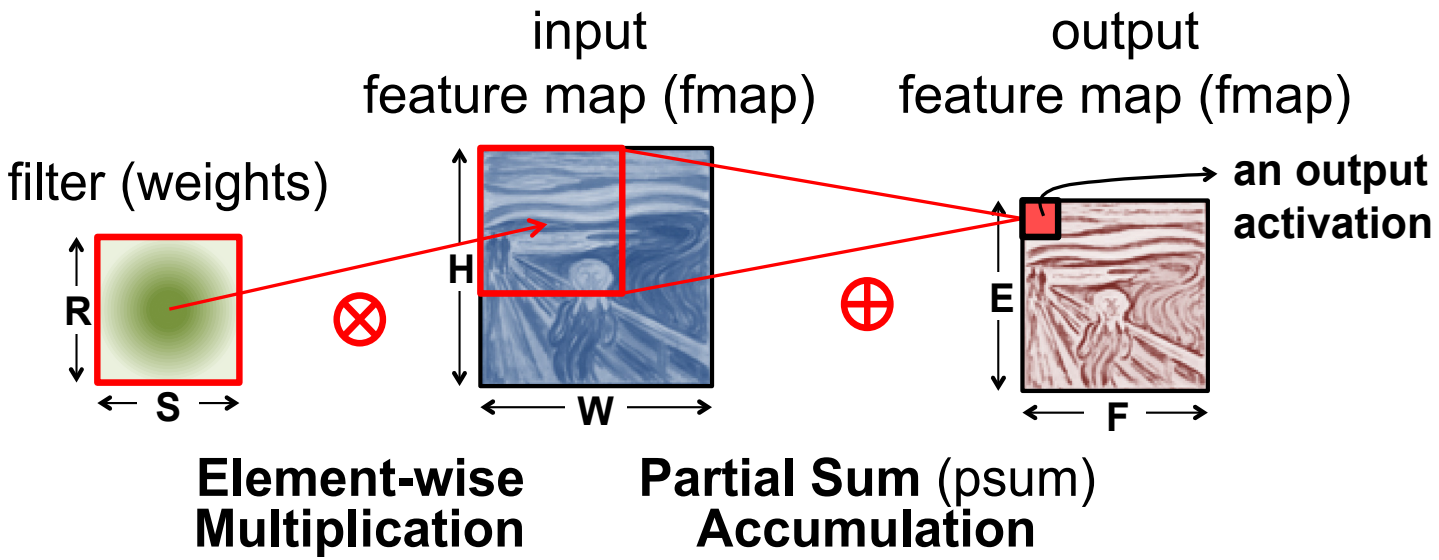
Automatic Speech Recognition

Image Captioning

Several Layer Types and Kernels

- Convolution layer
  - Many different variants also
    - Strided convolution, Dilated convolution, Groupwise convolution, Depthwise convolution, Pointwise convolution, Depthwise-Separable convolution
- Activation layer
  - ReLU, tanh, sigmoid, Leaky ReLU, Clipped ReLU, Swish
- Pooling layer
  - Max. pooling, Average pooling, Unpooling
- Fully-connected layer
- And many more .. Attention, Deconvolution, etc.

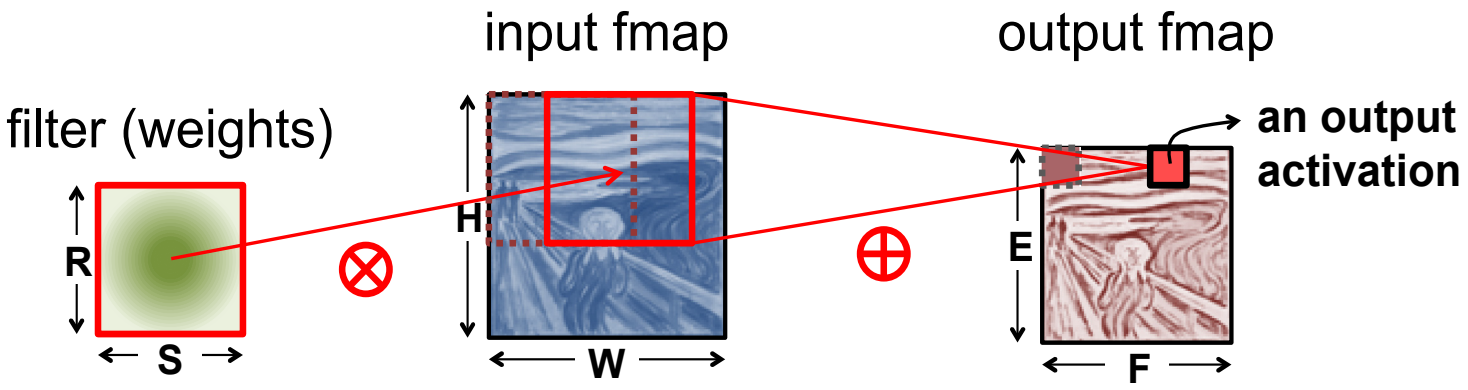
Convolution Layer




```
for e=[0:E)
  for f=[0:F)
    for r=[0:R)
      for s=[0:S)
        Out[e][f] += Weight[r][s] * Input[e+r][f+s]
```

Sze et al., *Synthesis Lectures on Computer Architecture, 2020*

Convolution Layer

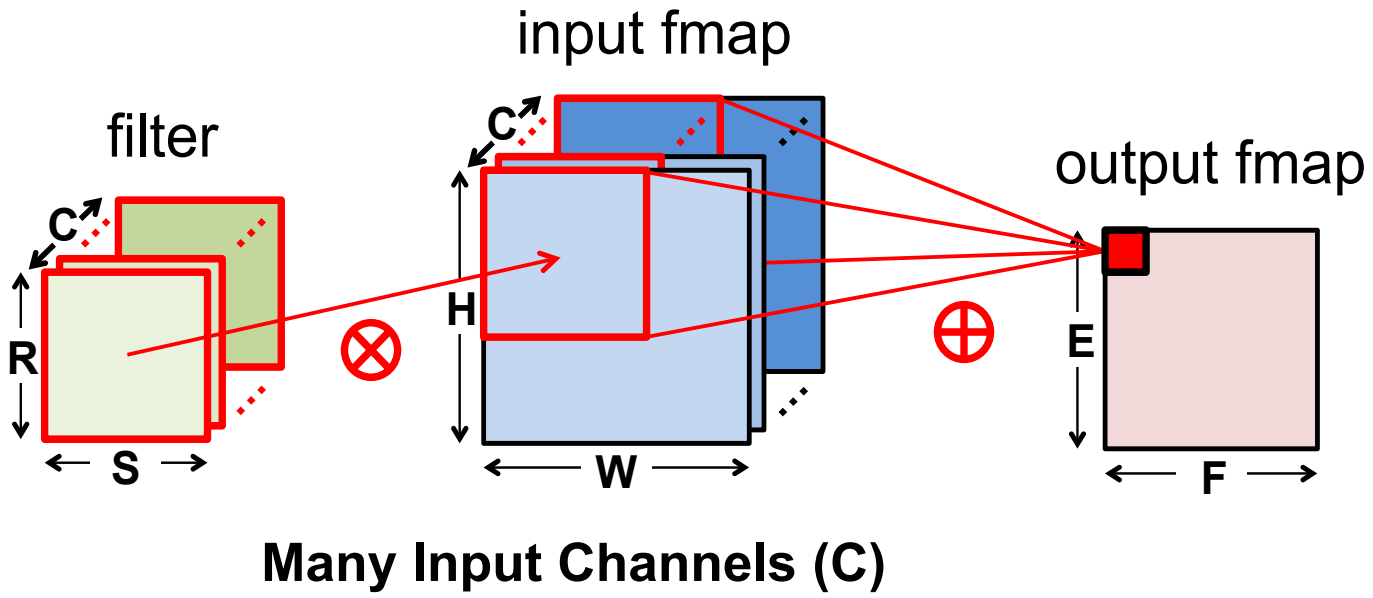


Sliding Window Processing

```
for e=[0:E)
  for f=[0:F)
    for r=[0:R)
      for s=[0:S)
        Out[e][f] += Weight[r][s] * Input[e+r][f+s]
```

Sze et al., *Synthesis Lectures on Computer Architecture, 2020*

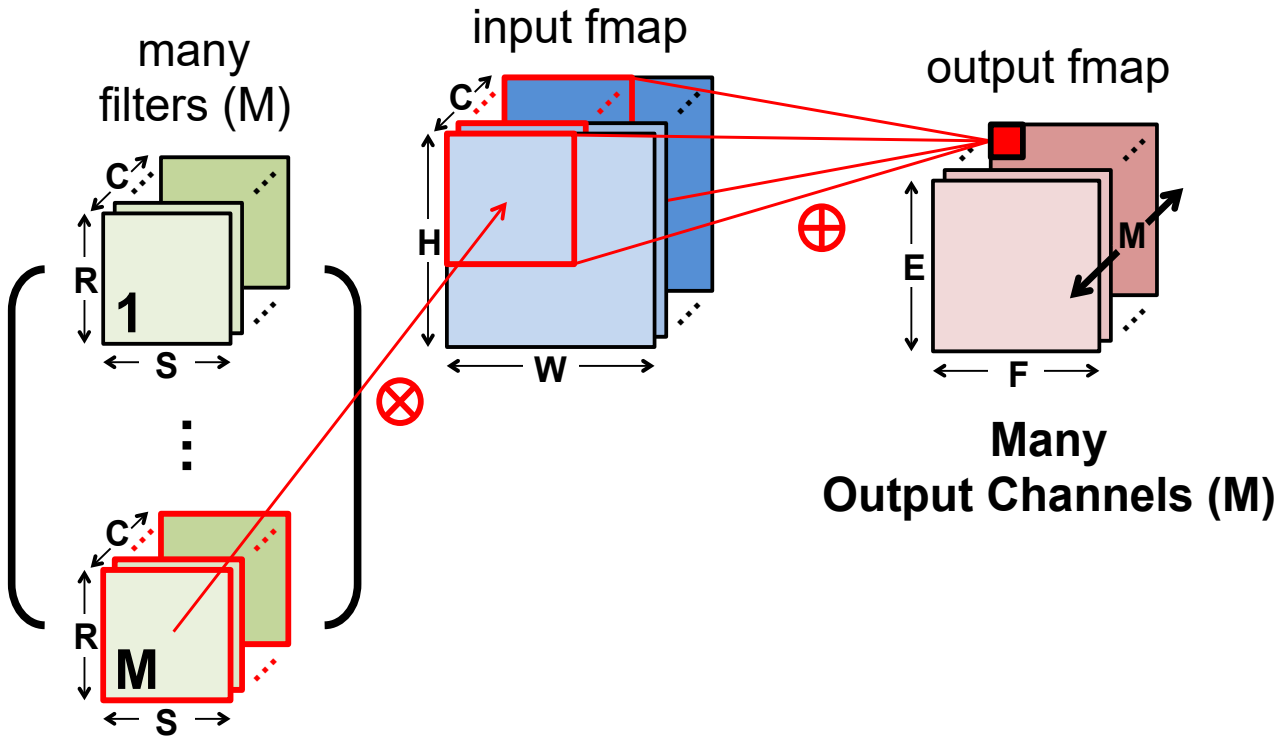
Convolution Layer




```
for e=[0:E)
 for f=[0:F)
  for c=[0:C)
   for r=[0:R)
    for s=[0:S)
     Out[e][f] +=
       Weight[r][s][c] *
       Input[e+r][f+s][c]
```

Sze et al., *Synthesis Lectures on Computer Architecture, 2020*

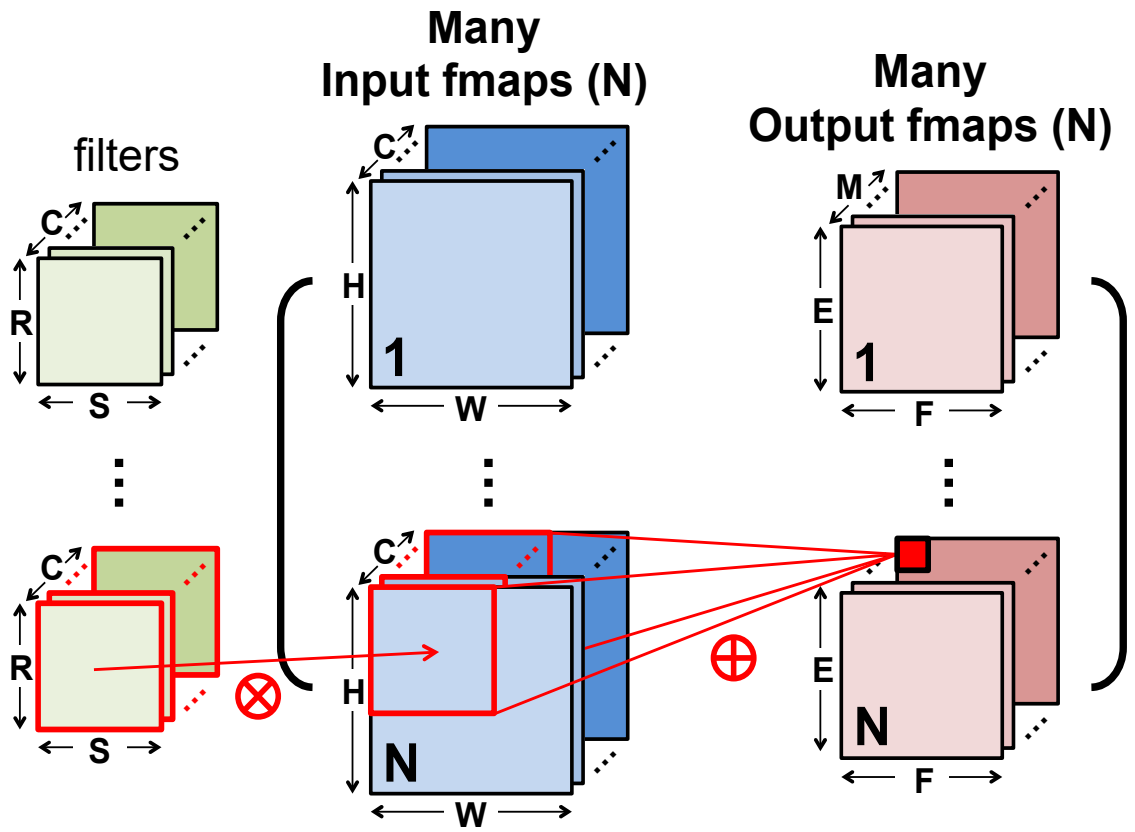
Convolution Layer




```
for m=[0:M)
 for e=[0:E)
  for f=[0:F)
   for c=[0:C)
    for r=[0:R)
     for s=[0:S)
      Out[e][f][m] +=
        Weight[r][s][c][m] *
        Input[e+r][f+s][c]
```

Sze et al., *Synthesis Lectures on Computer Architecture, 2020*

Convolution Layer




```
for n=[0:N)
 for m=[0:M)
  for e=[0:E)
   for f=[0:F)
    for c=[0:C)
     for r=[0:R)
      for s=[0:S)
       Out[e][f][m][n] +=
         Weight[r][s][c][m] *
         Input[e+r][f+s][c][n]
```

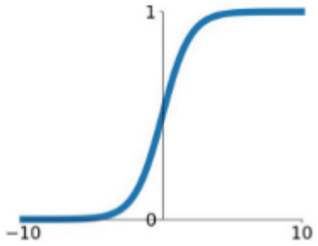
Sze et al., *Synthesis Lectures on Computer Architecture, 2020*

Activation Layer

- Introduce non-linearity in the network

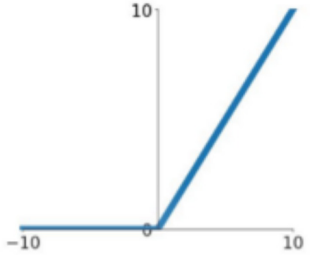
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



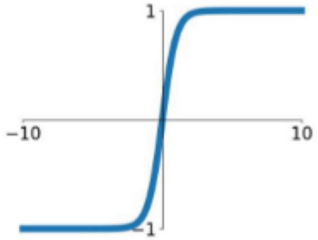
ReLU

$$\max(0, x)$$



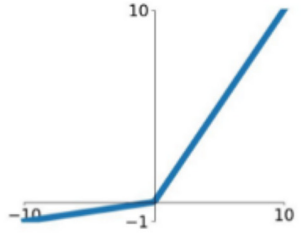
tanh

$$\tanh(x)$$



Leaky ReLU

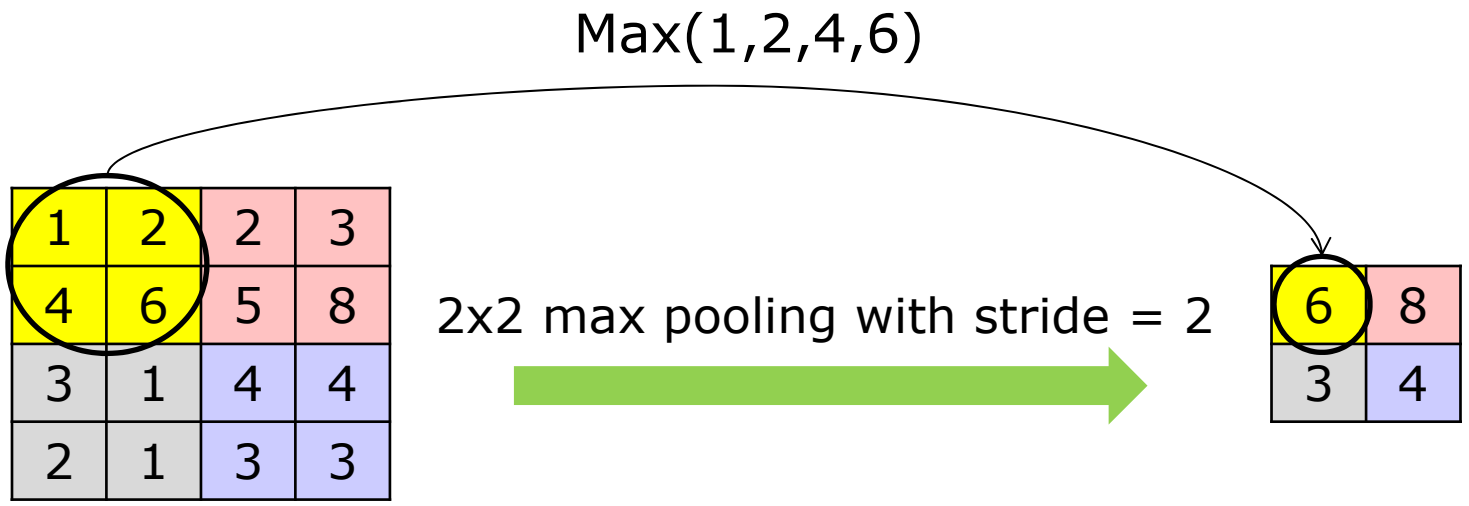
$$\max(0.1x, x)$$



Common Activation Layers in DNNs

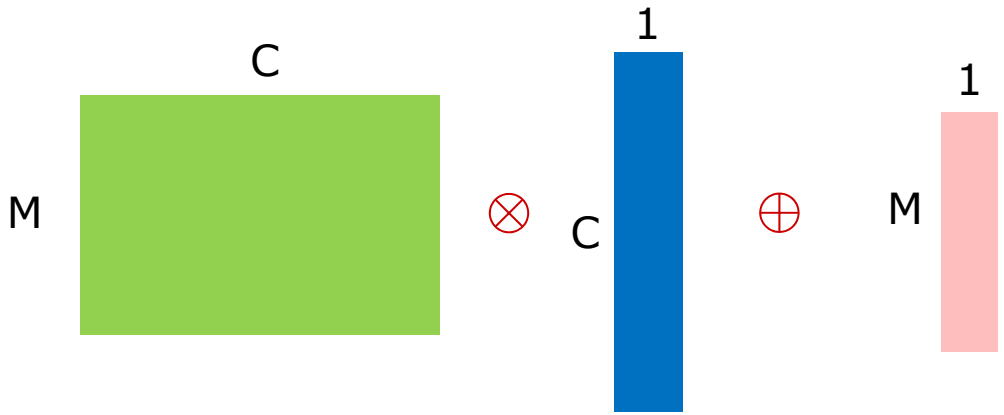
Pooling Layer

- Down-sampling operation to provide invariance against minor changes in the image such as shifting, rotation, etc.
- Can be different types
  - Max. pooling, Average pooling



Max. Pooling Example

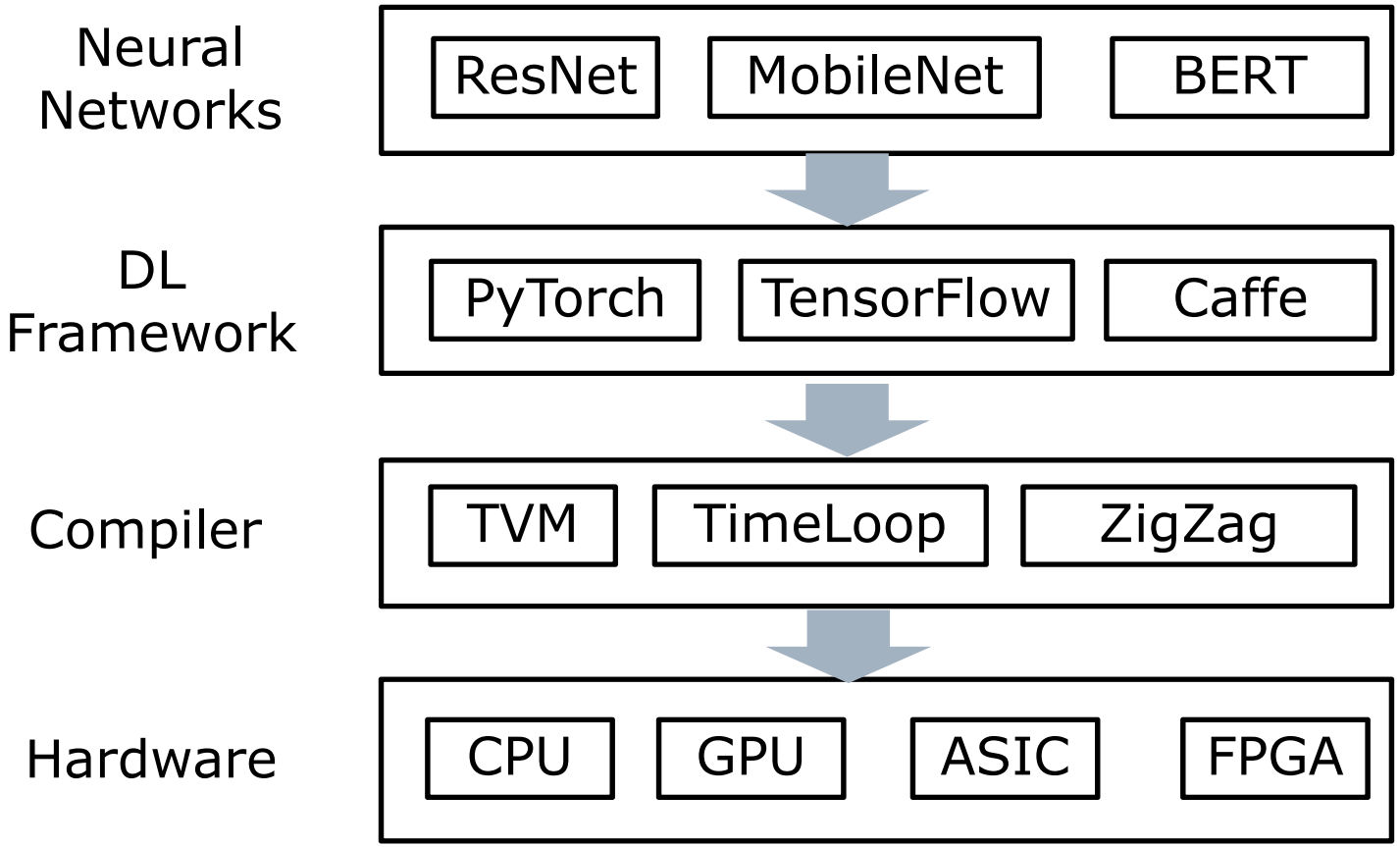
Fully-Connected Layer



Matrix-Vector Multiplication

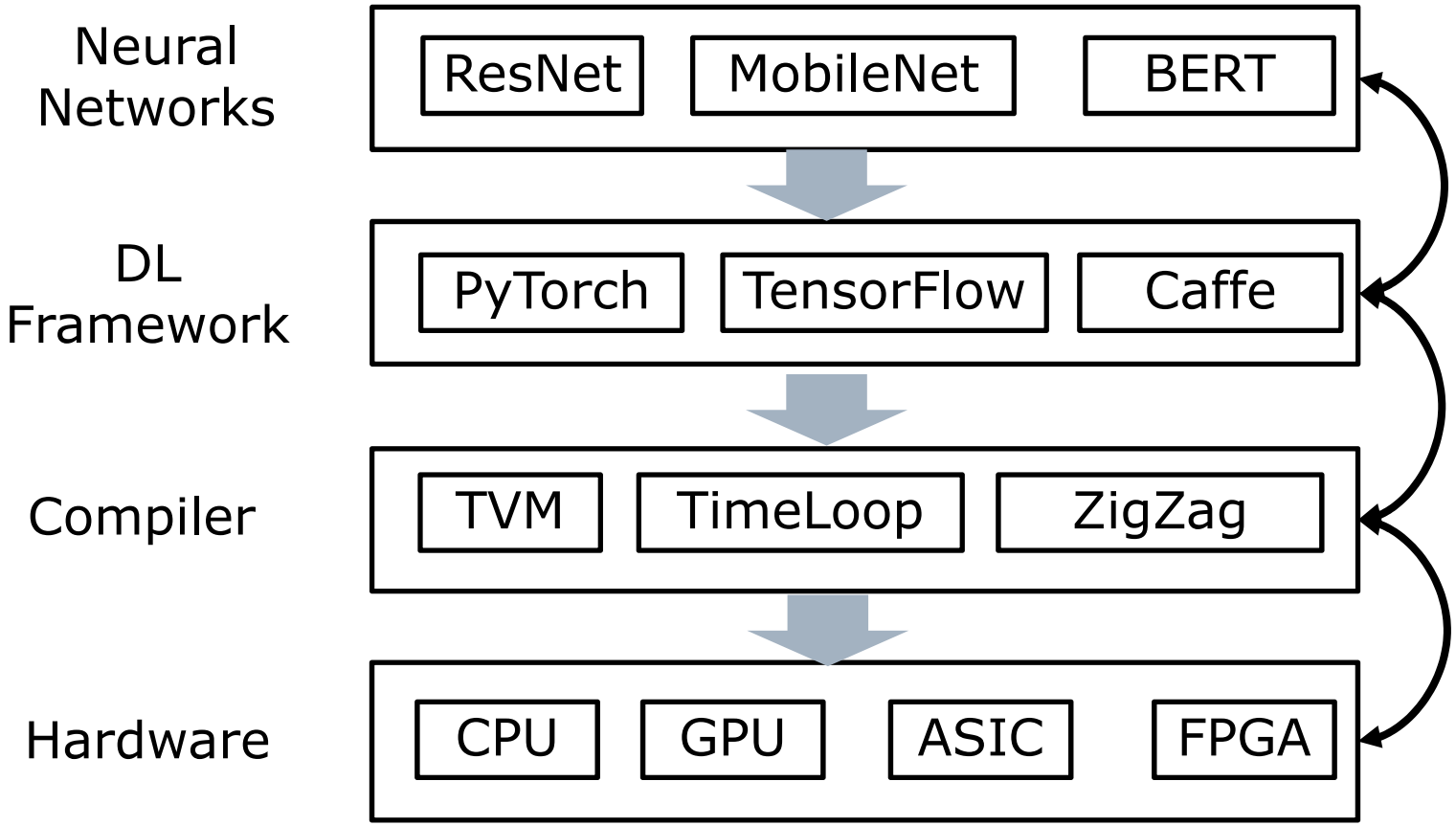
```
for m=[0:M)
 for c=[0:C)
   Out[m] +=
       Weight[m][c] *
       Input[c]
```

Deep Learning Stack



TimeLoop, *ISPASS 2019.* **TVM,** *OSDI 2018.* **ZigZag,** *arXiv 2020*

Deep Learning Stack



Co-design across different levels for efficient hardware

TimeLoop, *ISPASS 2019.* **TVM,** *OSDI 2018.* **ZigZag,** *arXiv 2020*

Evaluation Metrics

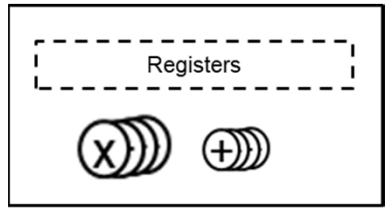
- **Accuracy**
  - % predicted correctly
- **Performance (Throughput, Latency)**
  - Inferences/sec, TOPS, delay
- **Energy efficiency**
  - Energy/inference, TOPS/W
- **Area efficiency**
  - Inference/sec/mm^2, TOPS/mm^2
- **Flexibility**
  - Support different types of neural networks and layers

Outline

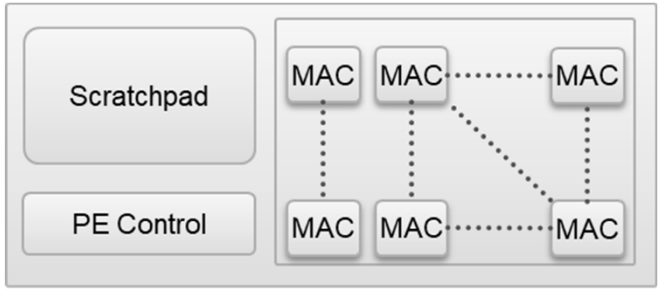
- Motivation for Hardware Acceleration
- Deep Neural Networks (DNN): Basics
- Accelerator Architecture Design
  - Compute: Efficient MAC Datapath
  - Storage: Memory Hierarchy
  - Communication: Interconnect Topologies
- Compiler Mapping Flow
  - Data reuse, Parallelism, Pipelining
- Neural Network Optimizations
  - Quantization
  - Sparsity

Hierarchical View of DL Accelerator

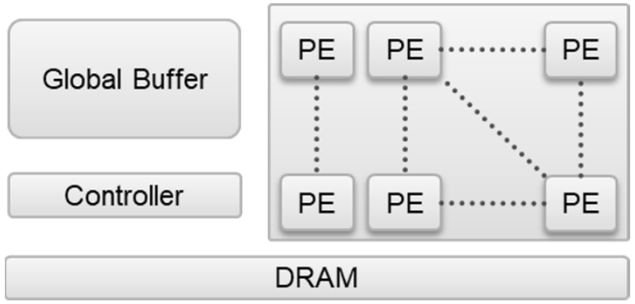
- Multiply & Accumulate (MAC) unit
  - Multiply-Accumulate Datapath
  - Registers
- Processing Element (PE)
  - Arrays of MAC units
  - Scratchpad
  - PE Control Finite State Machine (FSM)
- System
  - Array of PEs
  - Global buffer
  - Controller
  - DRAM



MAC unit



Processing Element



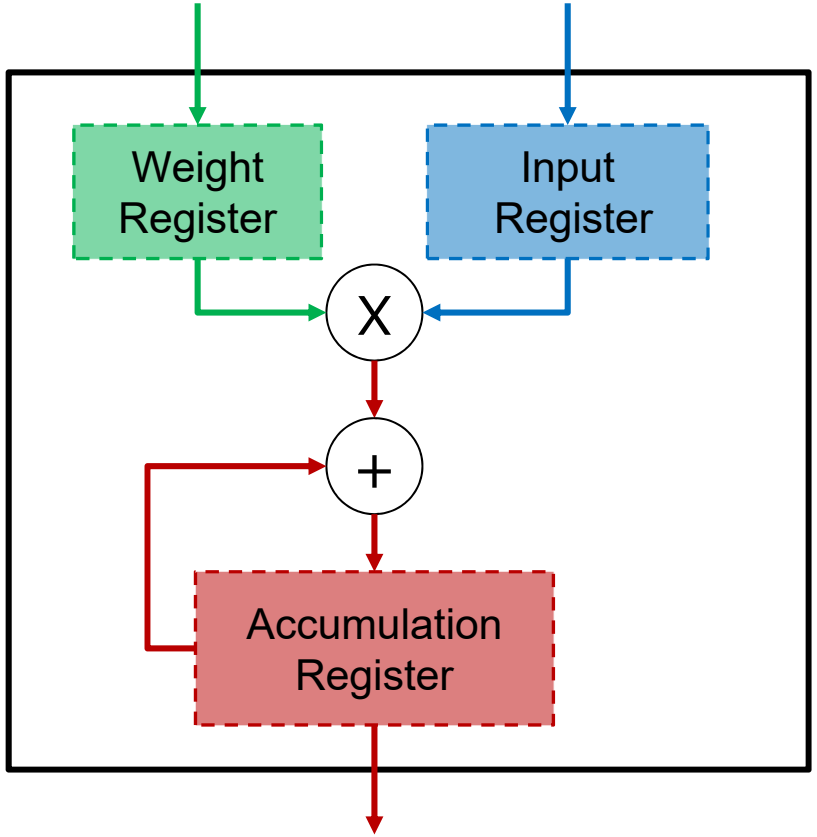
DL Accelerator System

Outline

- Motivation for Hardware Acceleration
- Deep Neural Networks (DNN): Basics
- Accelerator Architecture Design
  - Compute: Efficient MAC Datapath
  - Storage: Memory Hierarchy
  - Communication: Interconnect Topologies
- Compiler Mapping Flow
  - Data reuse, Parallelism, Pipelining
- Neural Network Optimizations
  - Quantization
  - Sparsity

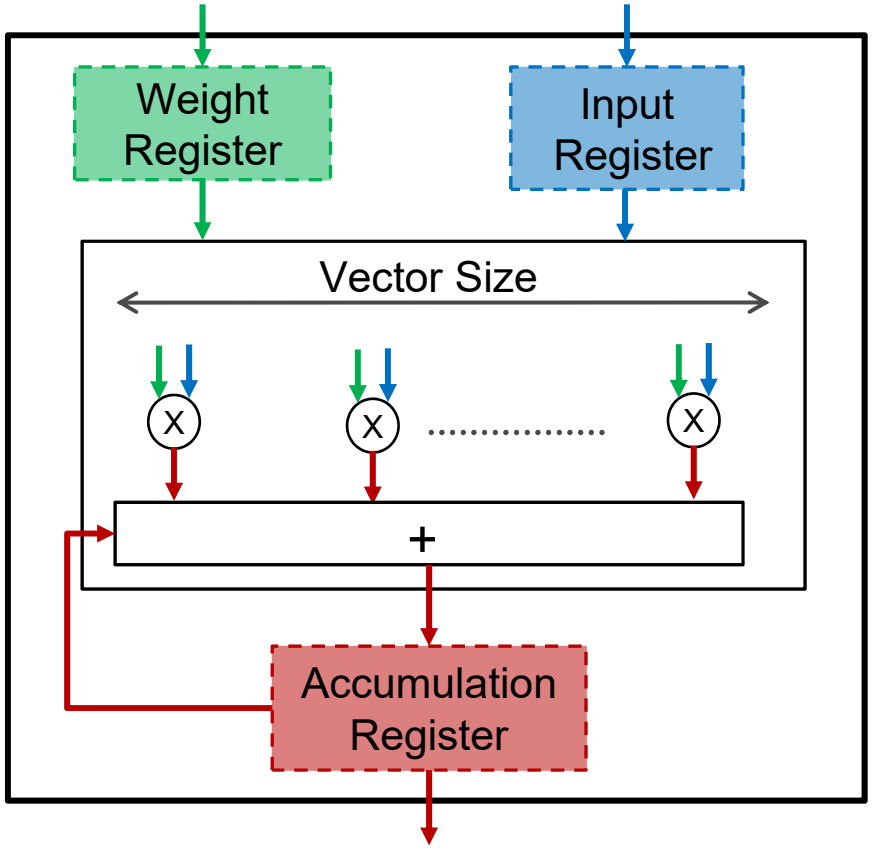
MAC Unit Implementations

- Scalar MAC unit
  - One Multiply and Accumulate operation per cycle
  - Registers are used to store values and achieve temporal reuse across cycles
  - Registers are optional
    - All operands cannot be reused across cycles
    - Typically, only one (sometimes two) operands are stored in registers
- Examples
  - Google **TPU**, **Eyeriss** *ISSCC 2016*



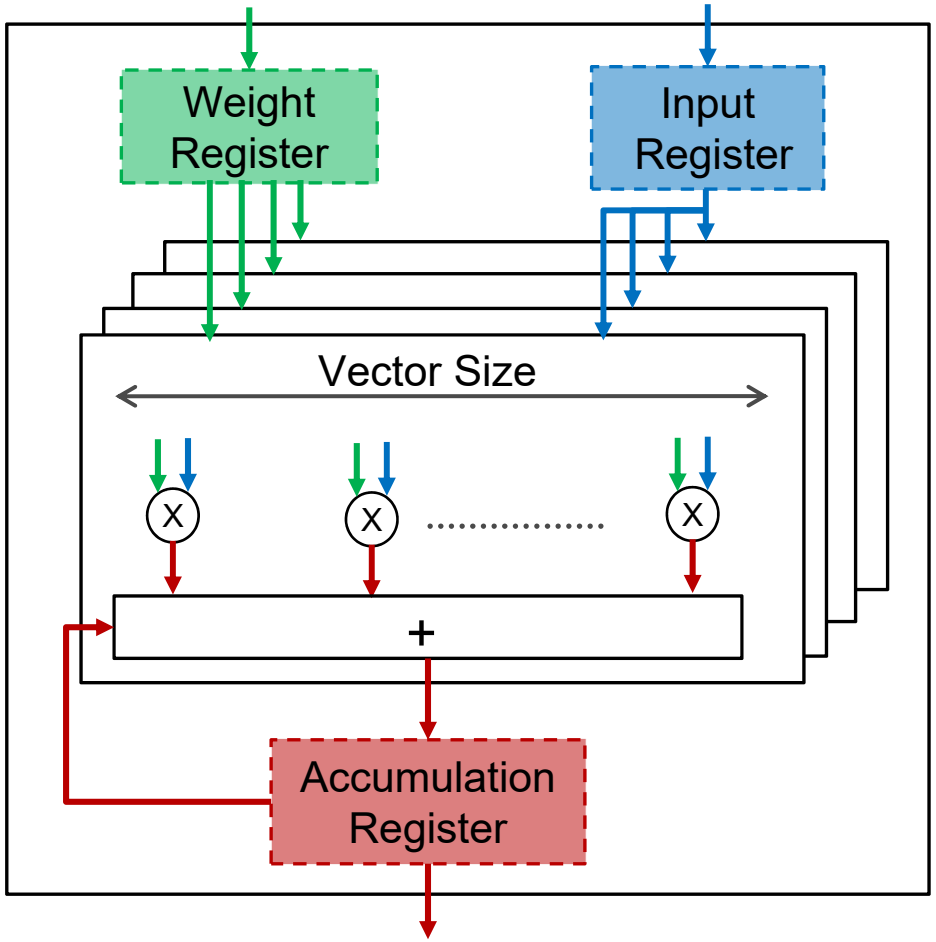
MAC Unit Implementations

- Vector MAC unit
  - Vector-wide multiply-accumulate operations every cycle
  - Performs vector dot-product
    - Takes a weight vector, an input activation vector, and a partial sum scalar as inputs
    - Computes a partial sum as output
  - Achieves spatial reuse of partial sum values by reducing them during dot-product computation
  - Registers can be used to store one or more operands for temporal reuse



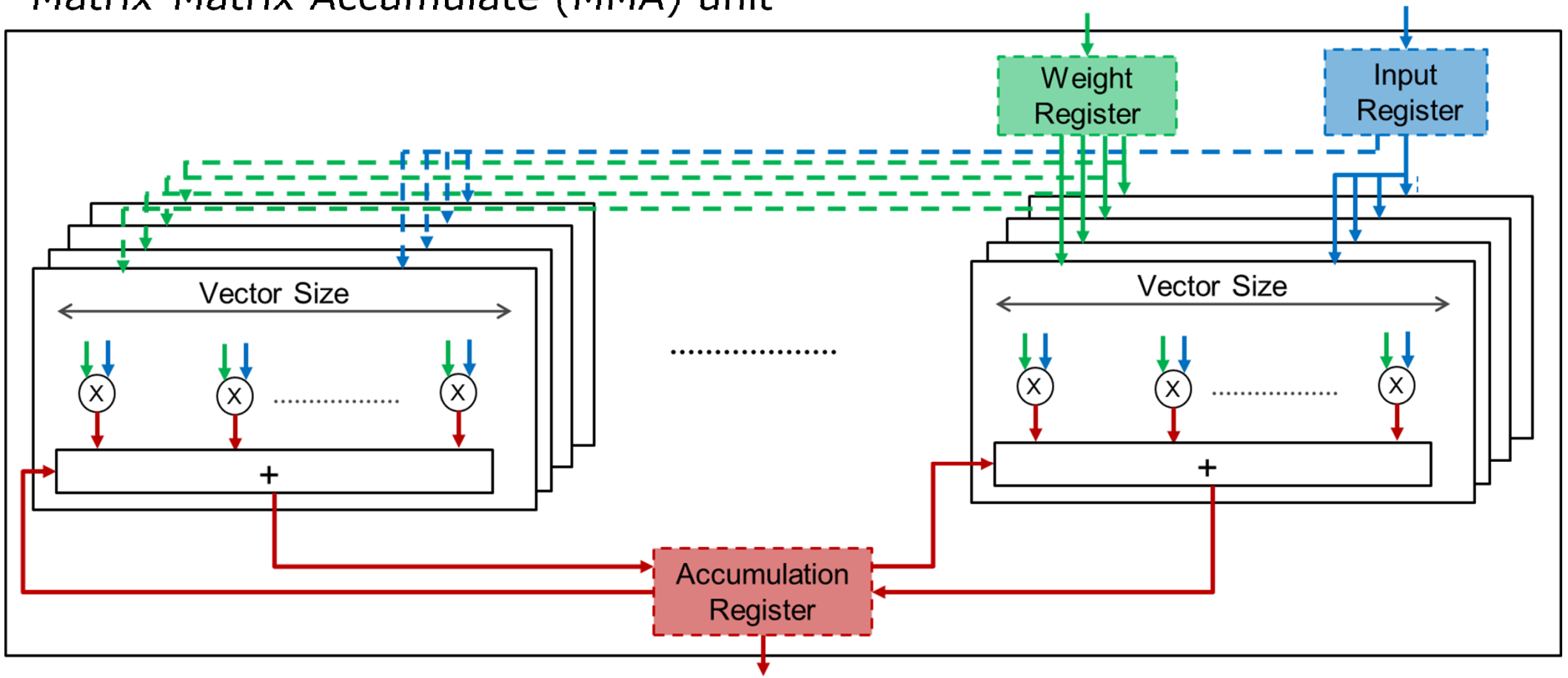
MAC Unit Implementations

- Matrix-Vector Accumulate (MVA) unit
  - Multiple vector-dot products every cycle
    - Example:
      - Takes a weight matrix, an input activation vector, and a partial sum vector as inputs
      - Computes a partial sum vector as output
  - Achieves spatial reuse of activation as well as partial sum values
  - Registers can be used to store one or more operands for temporal reuse
- Examples
  - **NVDLA**, **Venkatesan et al.** *HotChips 2019*, **Zimmer et al.** *JSSC 2020*



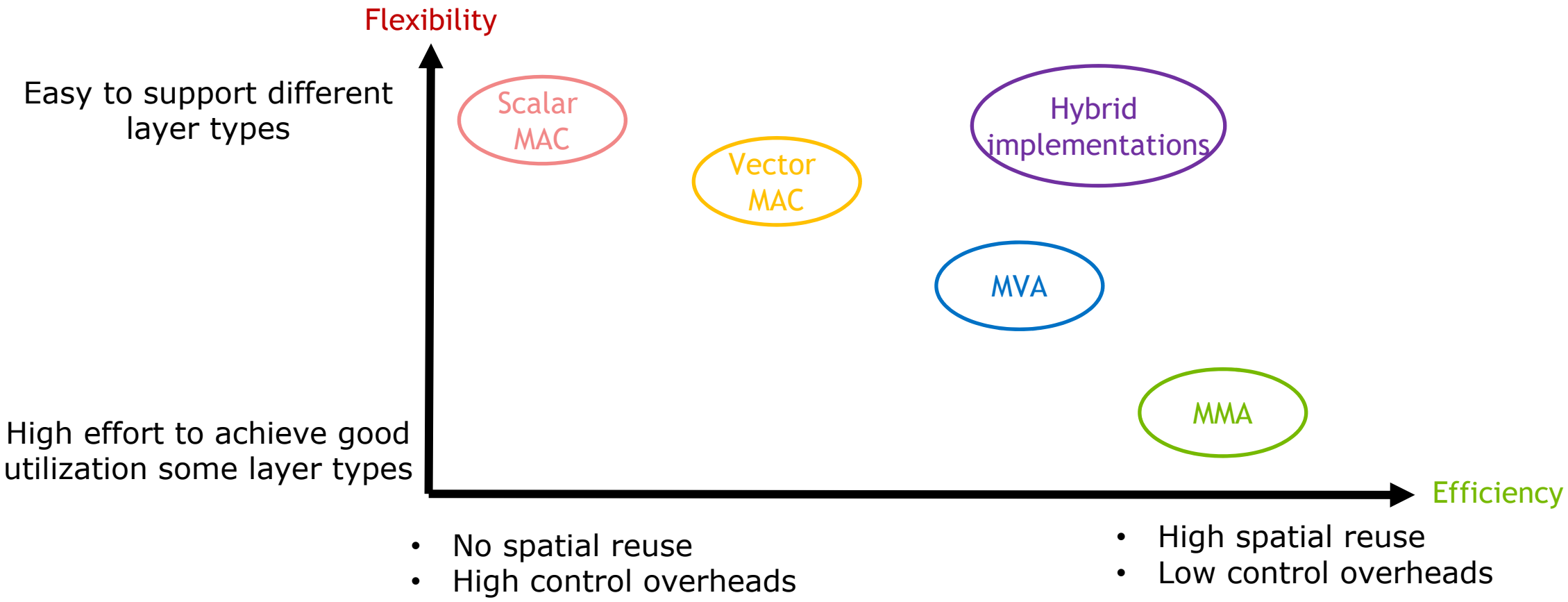
MAC Unit Implementations

- Matrix-Matrix Accumulate (MMA) unit



- Examples: NVIDIA **GPU Tensor cores**, Alibaba **Hanguang** *ISSCC 2020*

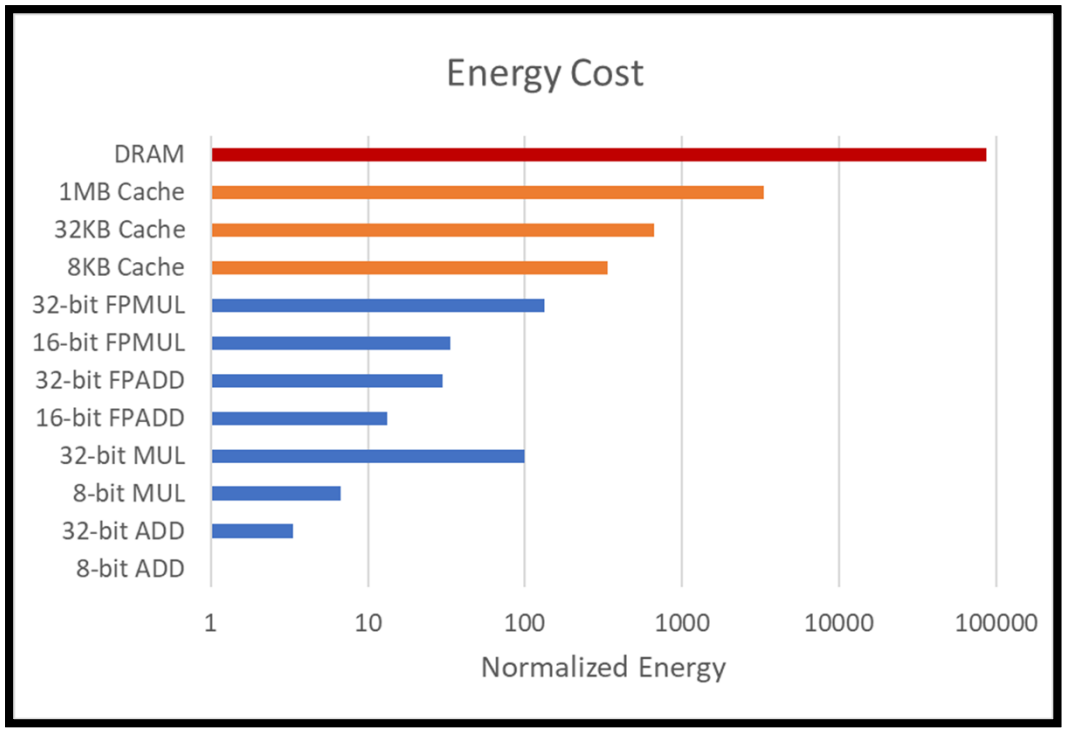
MAC Implementation Tradeoffs



Outline

- Motivation for Hardware Acceleration
- Deep Neural Networks (DNN): Basics
- Accelerator Architecture Design
  - Compute: Efficient MAC Datapath
  - Storage: Memory Hierarchy
  - Communication: Interconnect Topologies
- Compiler Mapping Flow
  - Data reuse, Parallelism, Pipelining
- Neural Network Optimizations
  - Quantization
  - Sparsity

Memory Hierarchy



Small Capacity,
Low Latency,
Low Energy

Large Capacity,
High Latency,
High Energy

Registers

Scratchpads

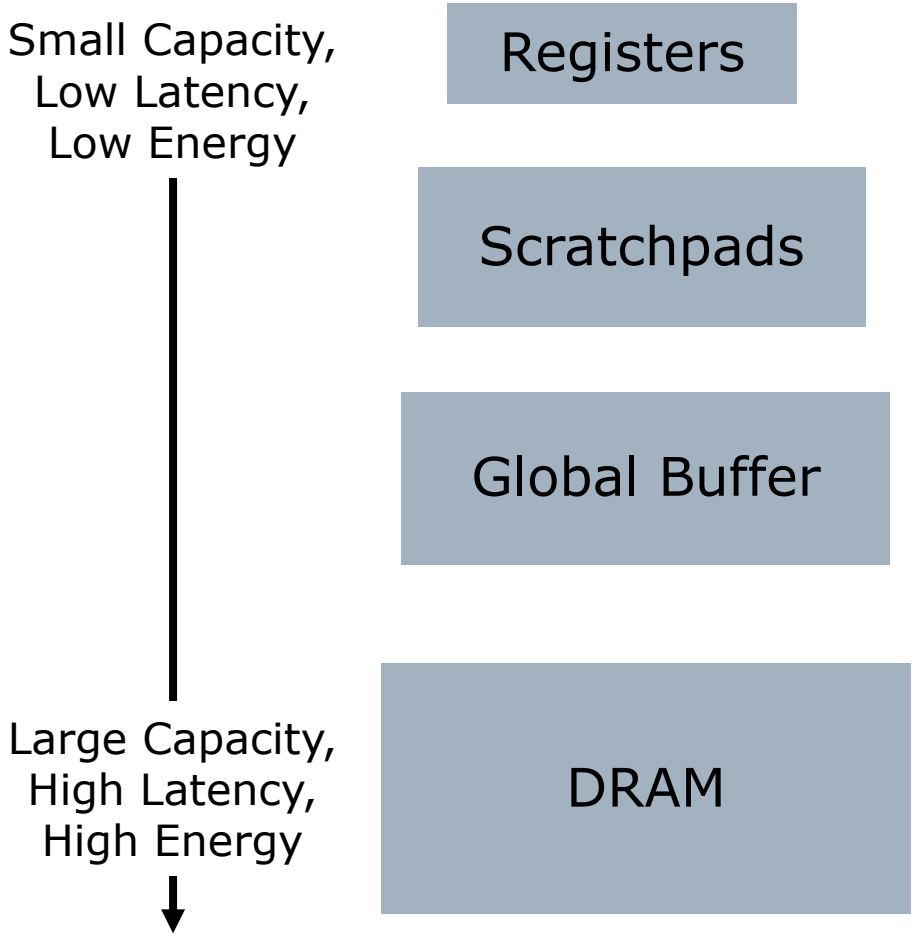
Global Buffer

DRAM

Horowitz, *ISSCC 2014*

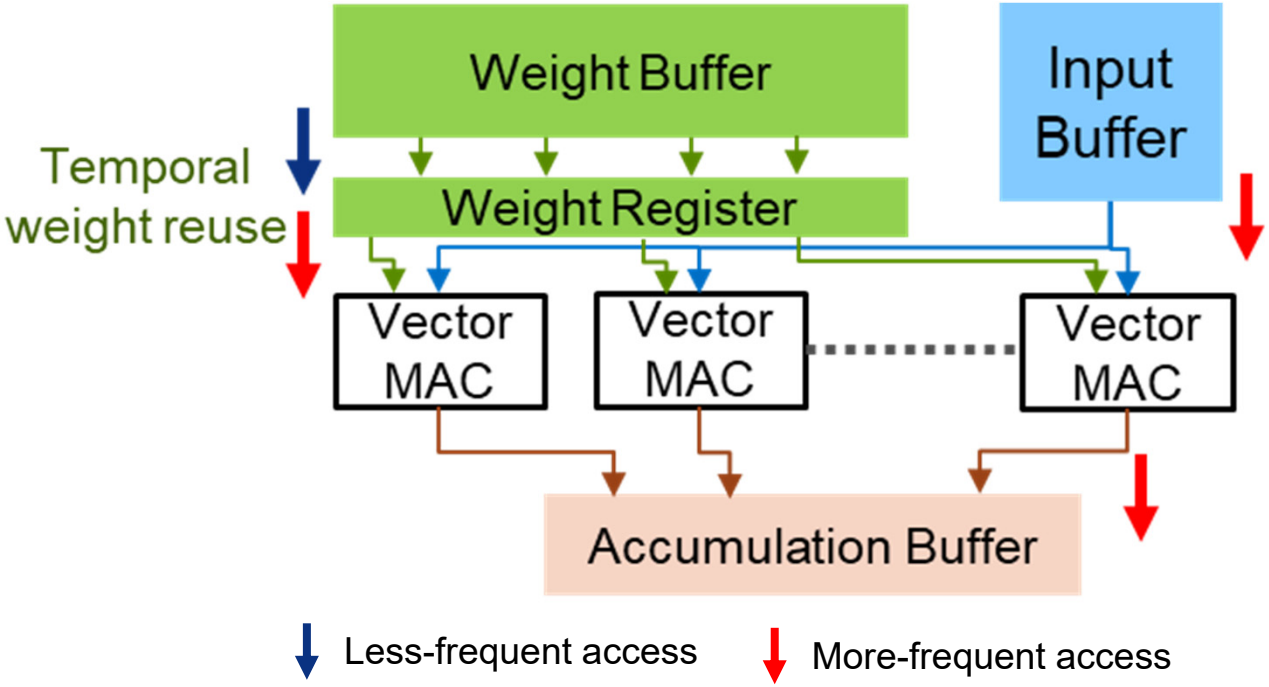
Memory Hierarchy

- Efficient reuse of data to achieve high performance and energy efficiency
- Optimal number of levels and sizing
- Buffer management to overlap communication and computation



Dataflows: Temporal Data Reuse

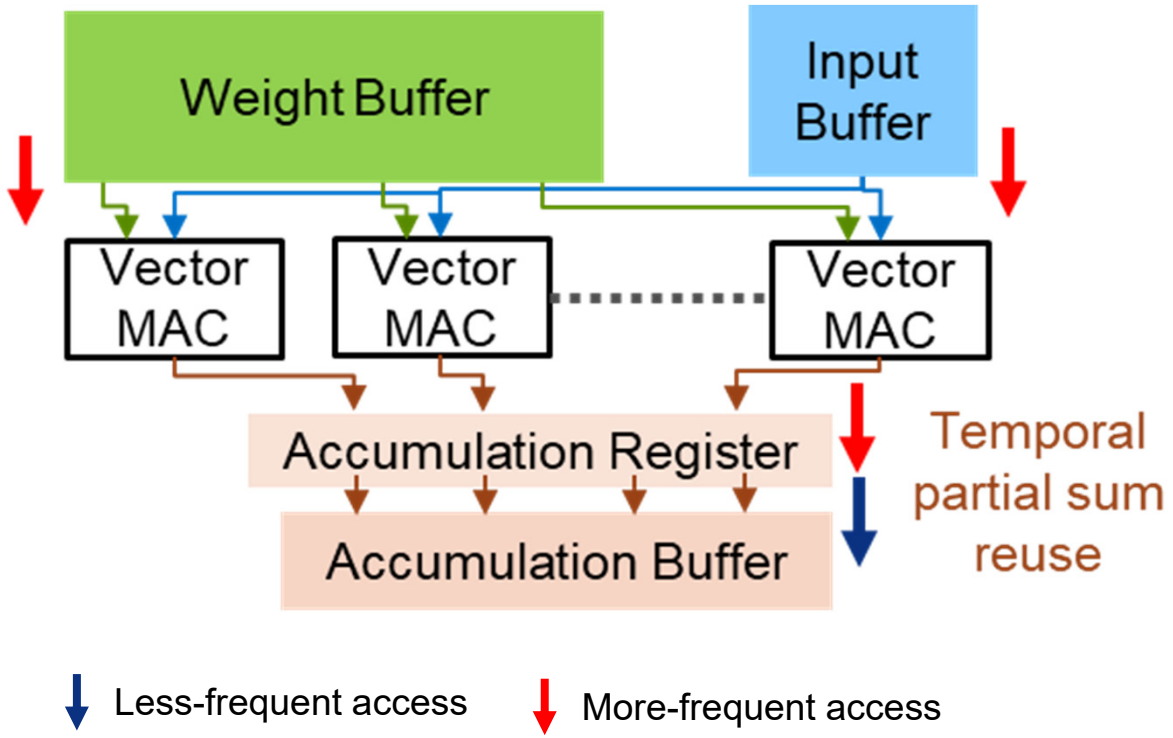
- Weight-Stationary (WS) Dataflow



- Examples: **Venkatesan et al.** *HotChips 2019,* **Zimmer et al.** *JSSC 2020,* Google **TPU, NVDLA**

Dataflows: Temporal Data Reuse

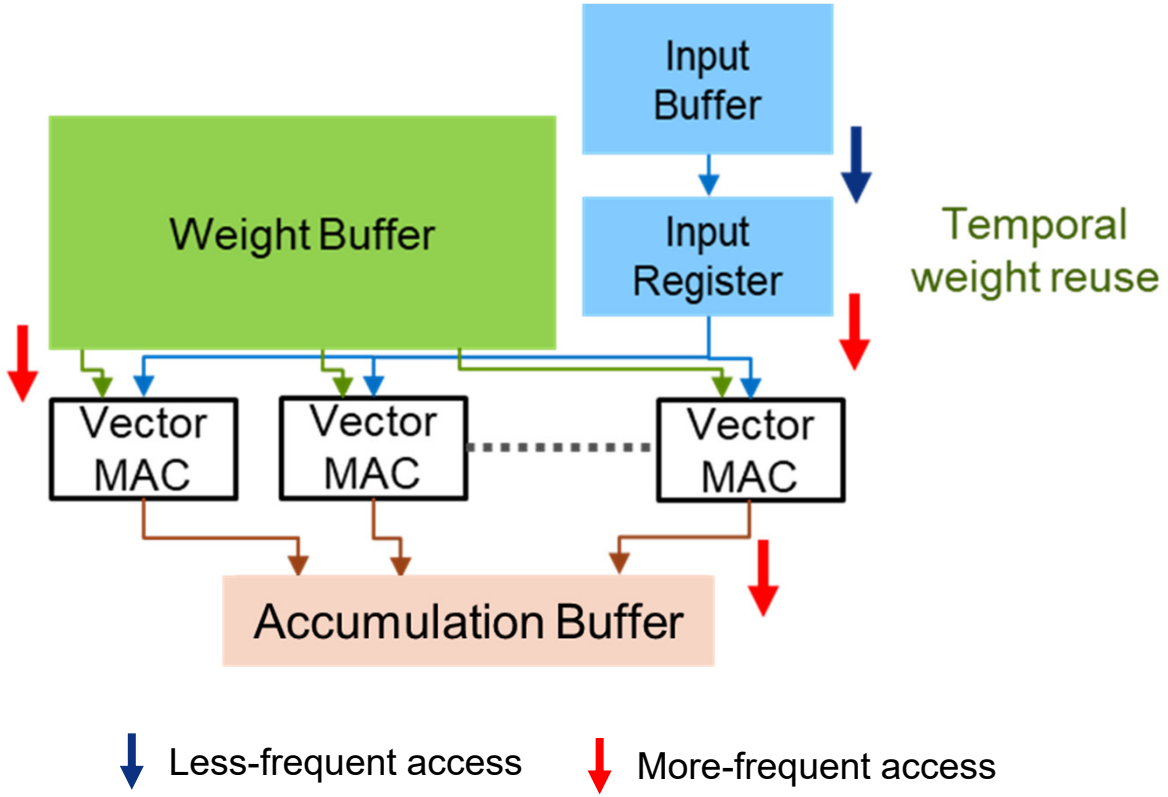
- Output-Stationary (OS) Dataflow



- Examples: **Moons et al.** *VLSI 2016,* **Thinker et al.** *VLSI 2017*

Dataflows: Temporal Data Reuse

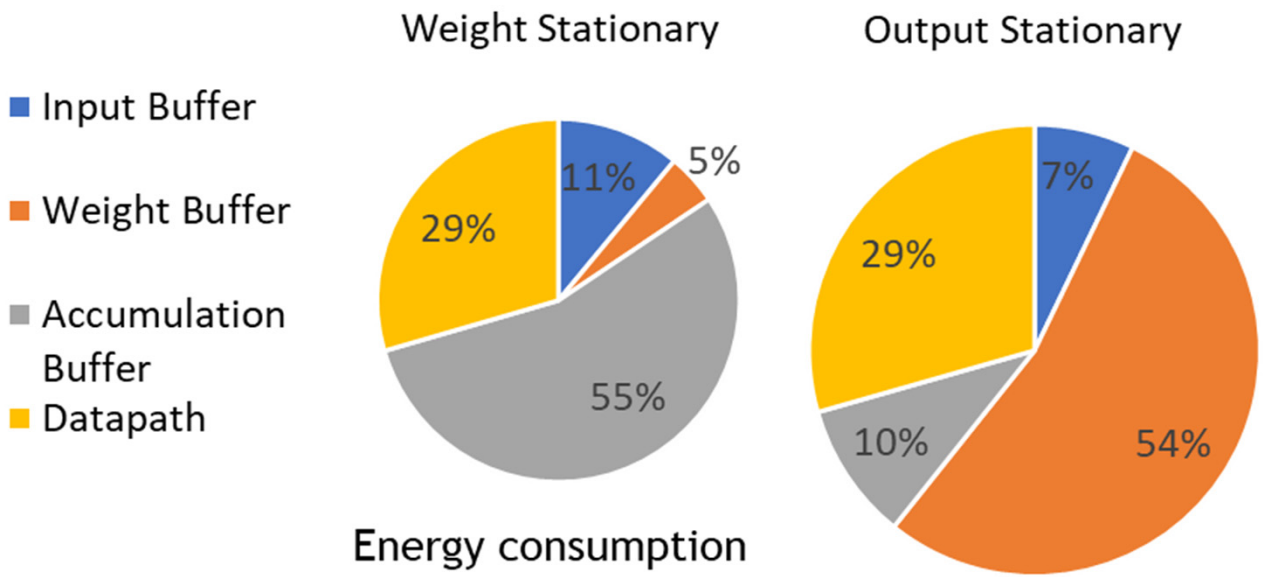
- Input-Stationary (IS) Dataflow



- Example: **SCNN**, *ISCA 2017*

Dataflows: Temporal Data Reuse

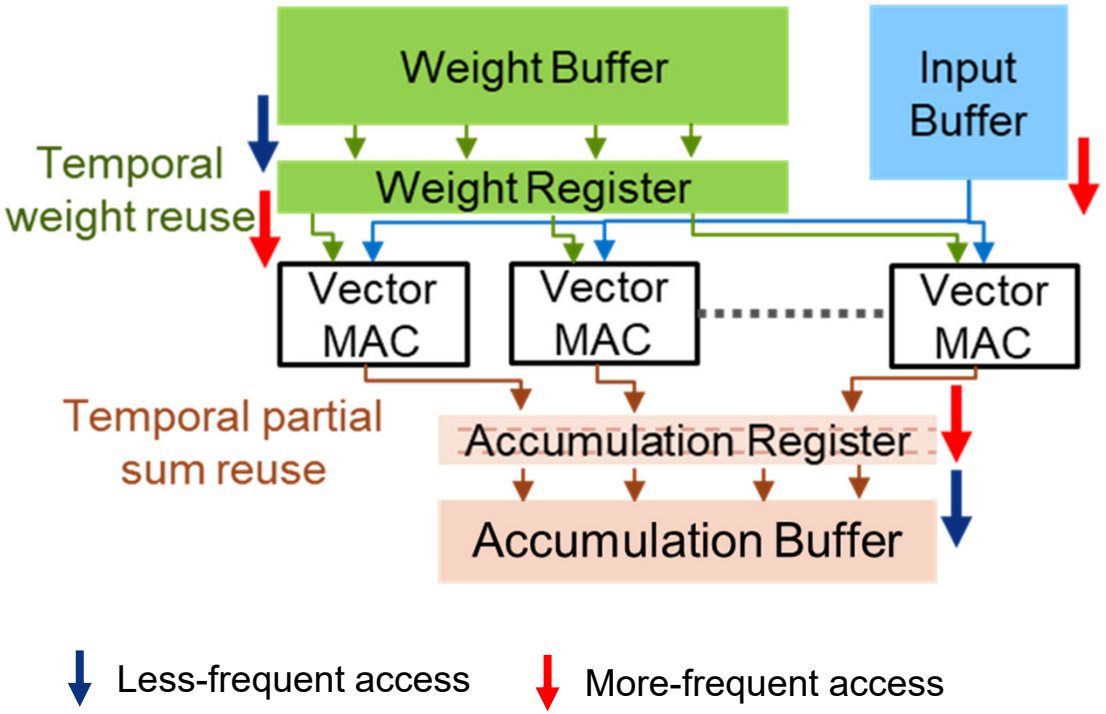
- Drawback of single data reuse
  - Operands with low or no reuse start to dominate energy consumption



Venkatesan et al., *ICCAD 2019*

Dataflows: Temporal Data Reuse

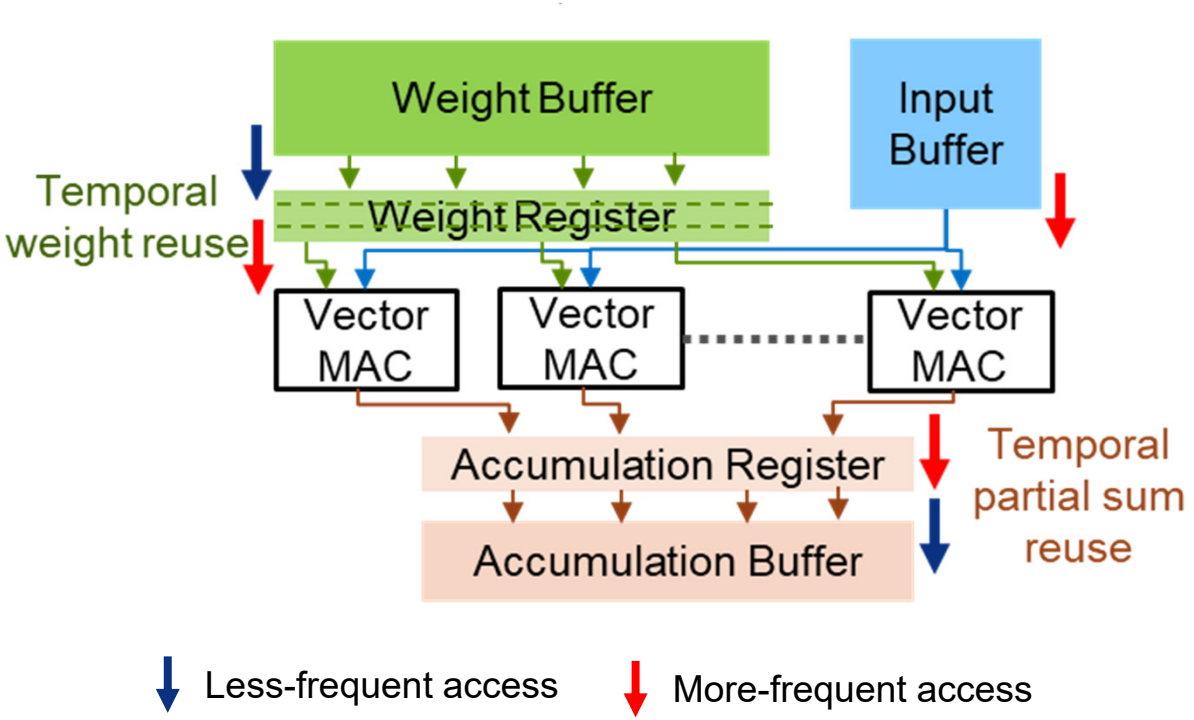
- Multi-level Dataflows
  - Output Stationary – Local Weight Stationary (OS-LWS) Dataflow



- Example: **MAGNet**, *ICCAD 2019*

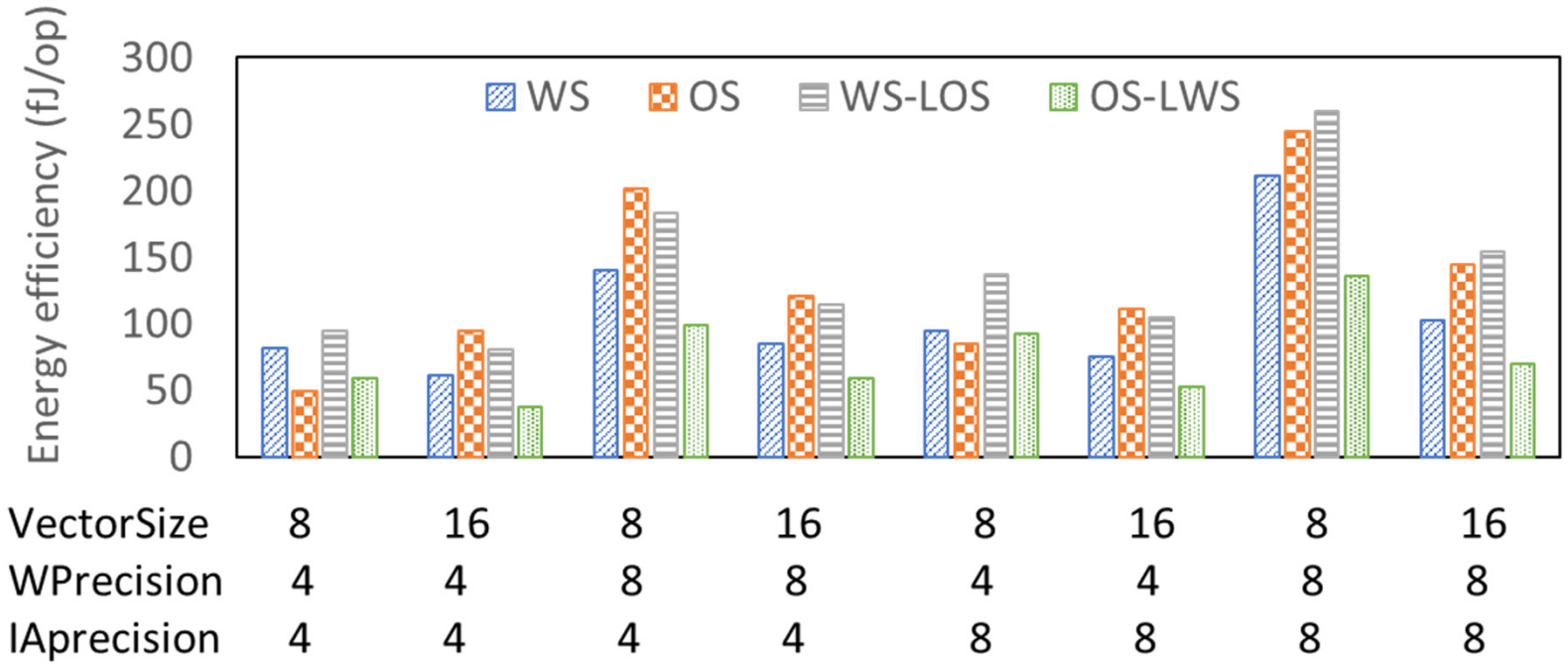
Dataflows: Temporal Data Reuse

- Multi-level Dataflows
  - Weight Stationary – Local Output Stationary (WS-LOS) Dataflow



- Example: **MAGNet**, *ICCAD 2019*

Comparison of Dataflows



Network: ResNet-50

Dataset: ImageNet

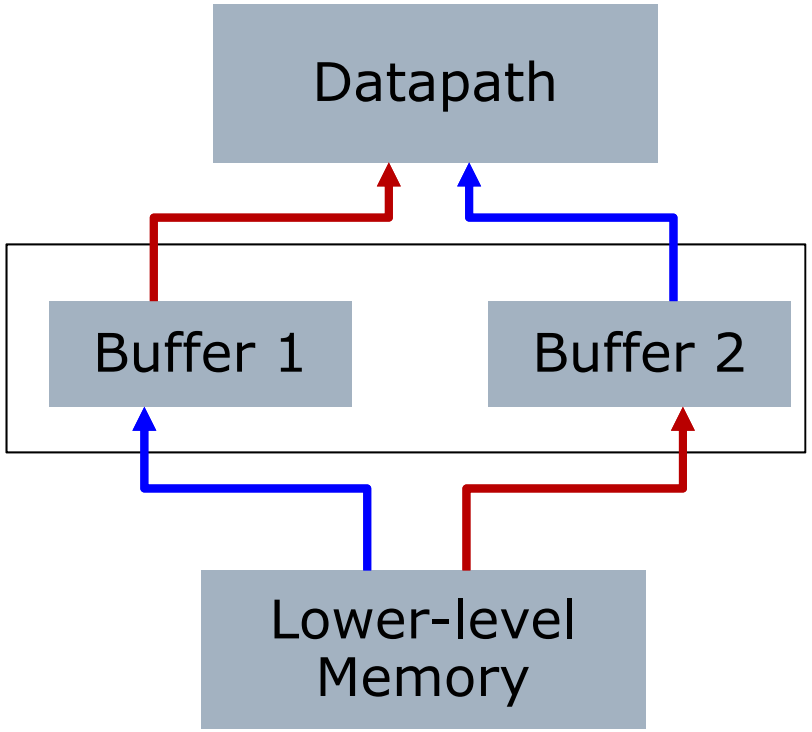
Technology: 16ff

- OS-LWS dataflow ➔ Temporal reuse in both weights and outputs
- Larger vector size ➔ Spatial input/output reuse, amortize control overheads

Venkatesan et al., *ICCAD 2019*

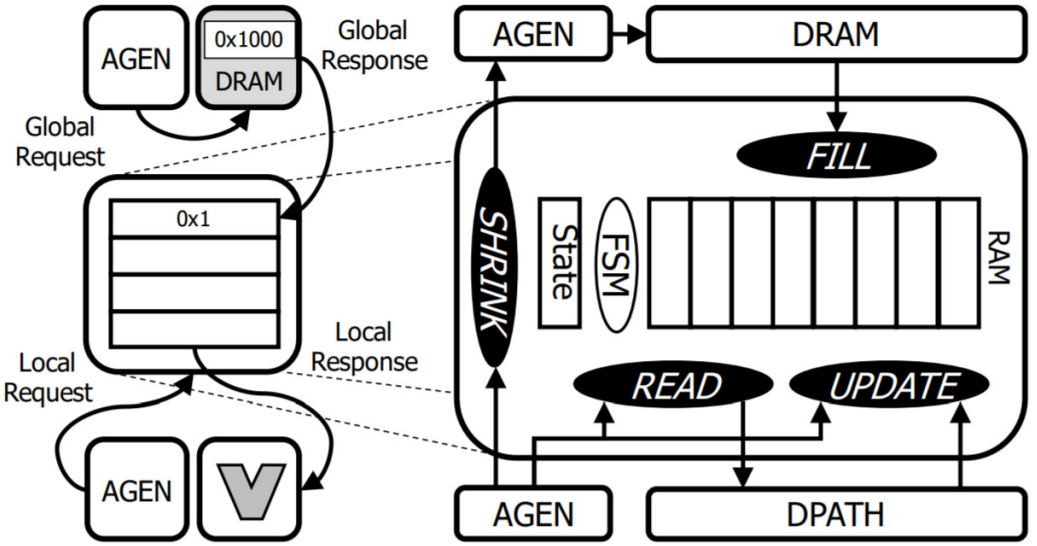
Compute-Communication Overlap

- Coarse-grained: DMA with double buffering



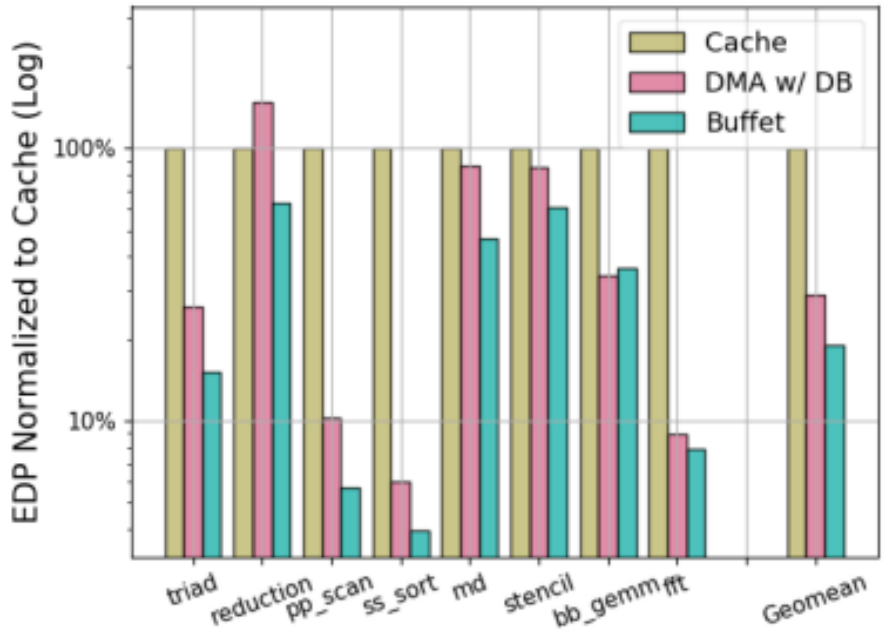
Buffet Buffer Manager

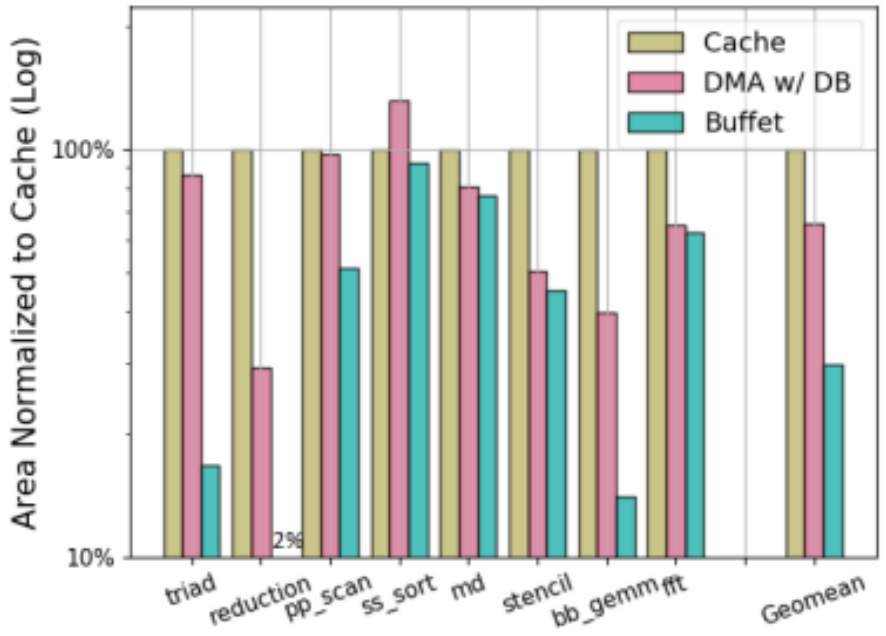
- Fine-grained compute-communication overlap
- Operations
  - Fill
    - Sequentially write data read from DRAM and set valid state
  - Read
    - Perform read if the address is valid, otherwise stall until address becomes valid
  - Update
    - Perform write operation to address and set valid state
  - Shrink
    - Invalidate addresses that are not in use and request data from lower-level memory



Pellauer et al., *ASPLOS 2019*

Buffet Buffer Manager





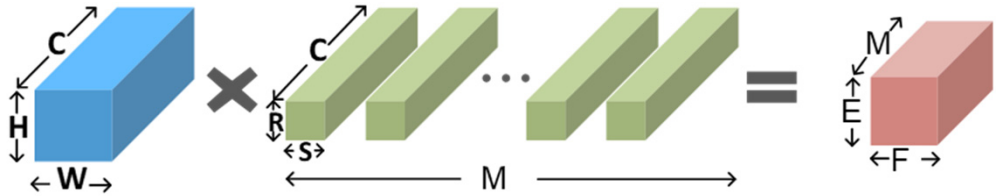
- Buffet achieves 2.3X reduction in energy-delay product (EDP) and 2.1X area efficiency gains over DMA with double buffering

Pellauer et al., *ASPLOS 2019*

Outline

- Motivation for Hardware Acceleration
- Deep Neural Networks (DNN): Basics
- Accelerator Architecture Design
  - Compute: Efficient MAC Datapath
  - Storage: Memory Hierarchy
  - Communication: Interconnect Topologies
- Compiler Mapping Flow
  - Data reuse, Parallelism, Pipelining
- Neural Network Optimizations
  - Quantization
  - Sparsity

Communication Patterns



PE PE PE PE

Interconnect

Memory

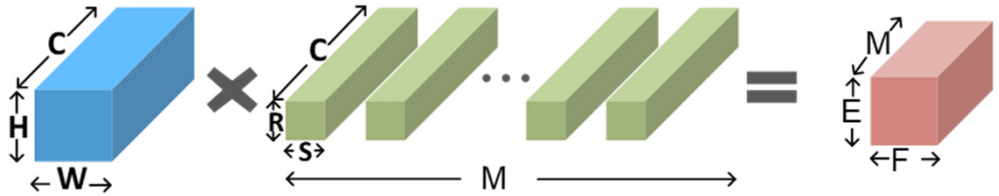
Weights

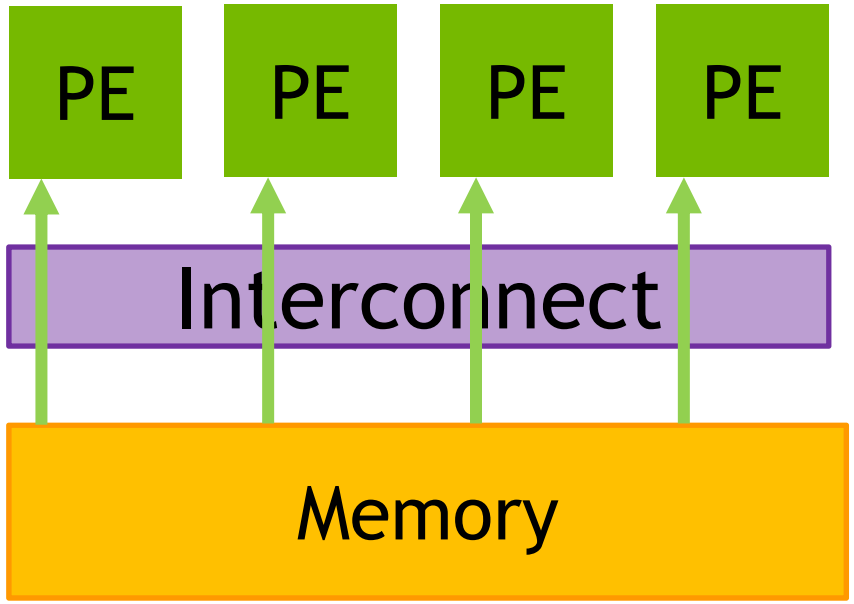
Input Activations

Output Activations

Partial Sums

Communication Patterns





Weights

Input Activations

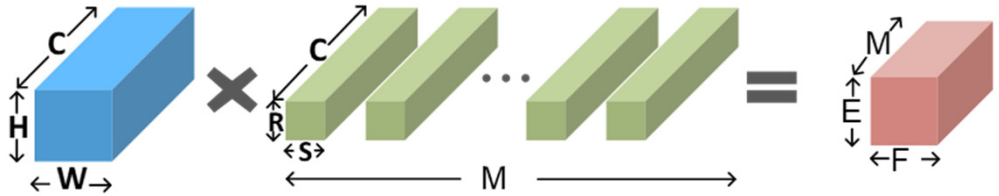
Output Activations

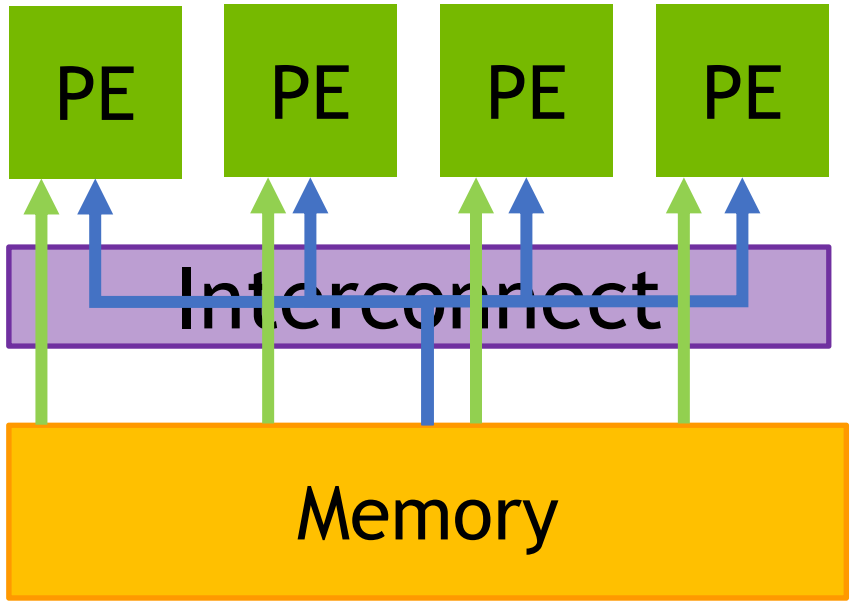
Partial Sums

Pattern-1

- Unicast Weights

Communication Patterns





Weights

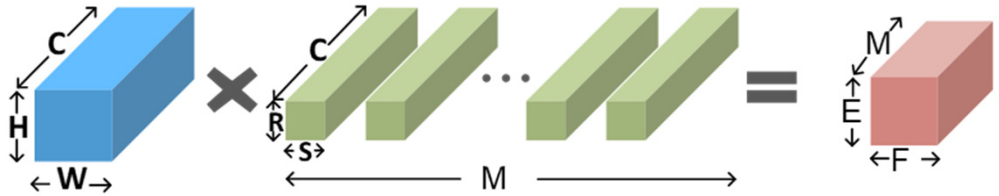
Input Activations

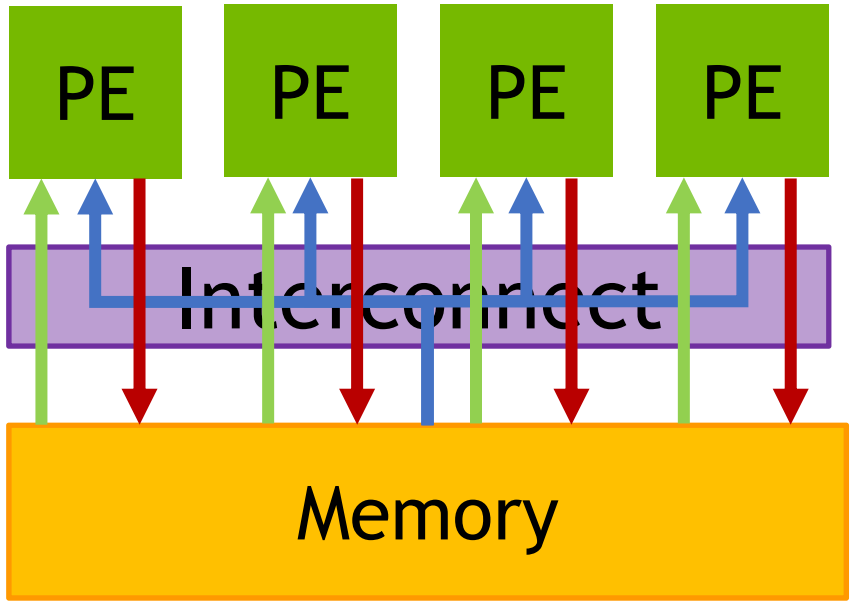
Output Activations

Partial Sums

Pattern-1
- Unicast Weights
- Multicast Input activations

Communication Patterns





Weights

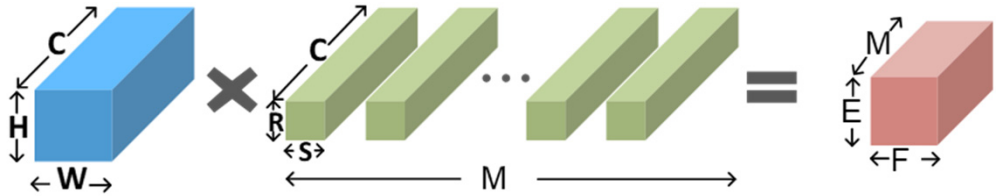
Input Activations

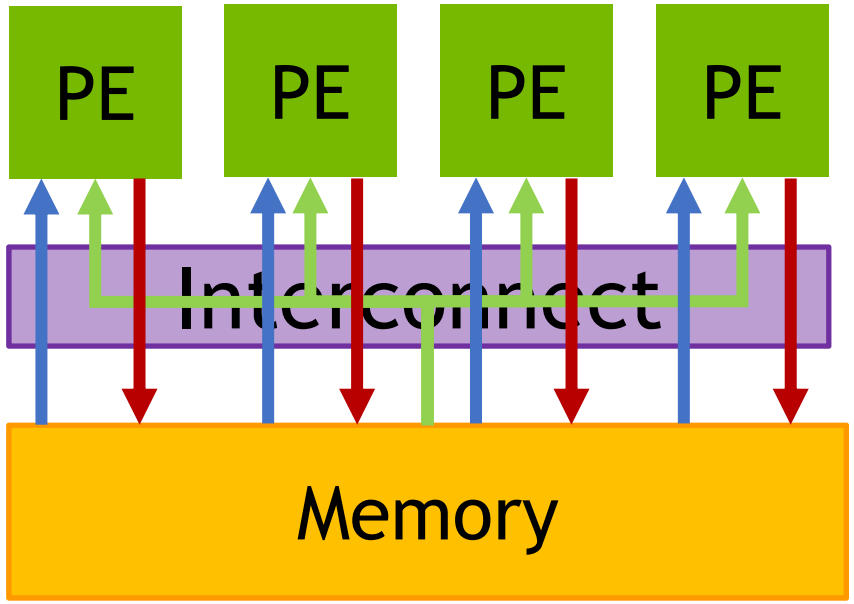
Output Activations

Partial Sums

Pattern-1
- Unicast Weights
- Multicast Input activations
- Unicast Output activations

Communication Patterns





Weights

Input Activations

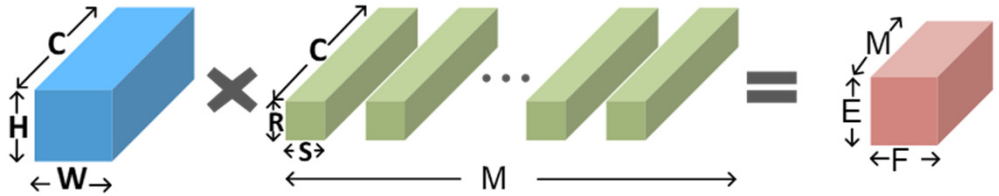
Output Activations

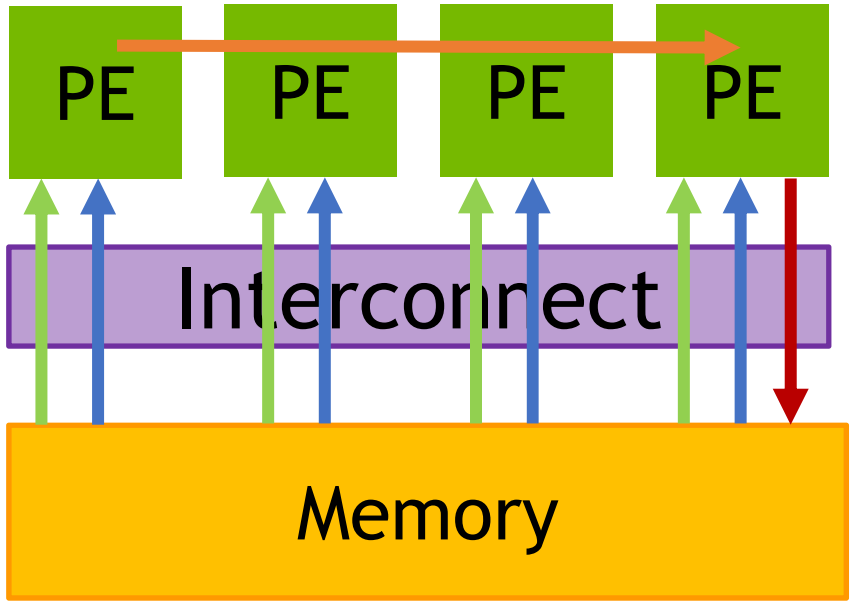
Partial Sums

Pattern-2

- Multicast Weights
- Unicast Input activations
- Unicast Output activations

Communication Patterns





Weights

Input Activations

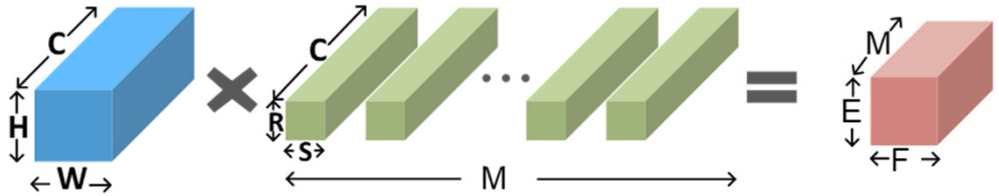
Output Activations

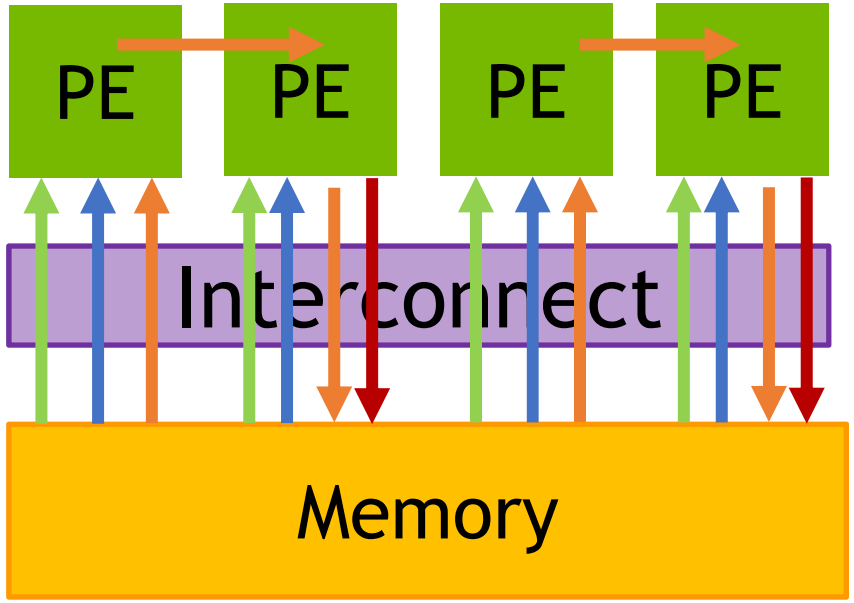
Partial Sums

Pattern-3

- Unicast Weights
- Unicast Input activations
- Unicast Partial sums
- Unicast Output activations

Communication Patterns





Weights

Input Activations

Output Activations

Partial Sums

Pattern-4

- Unicast Weights
- Unicast Input activations
- Unicast Partial sums
- Unicast Output activations

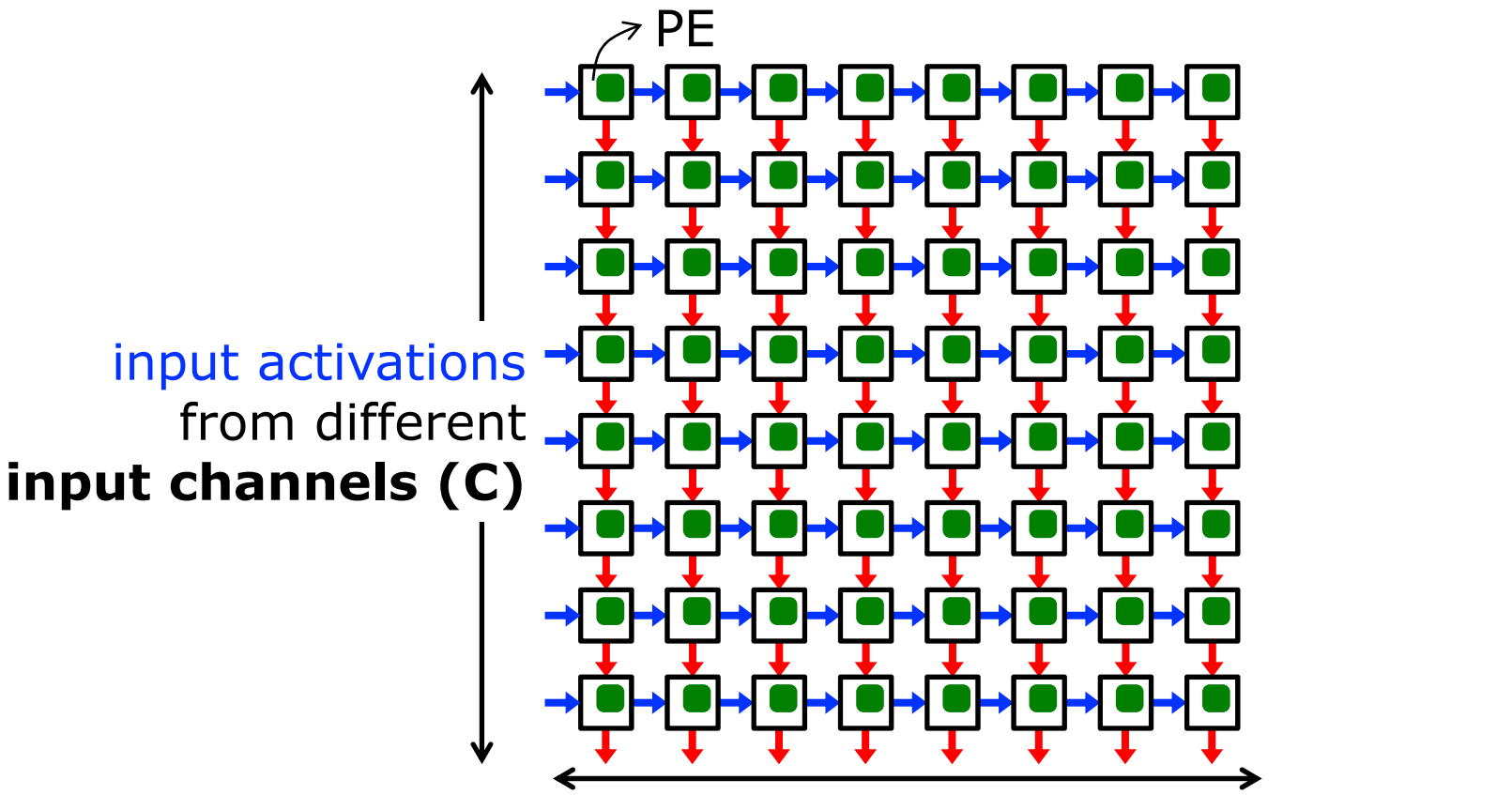
Interconnect Design

- In practice, we observe combinations of different patterns depending on the architecture

- Customized interconnects support specific patterns
  - e.g. Systolic Arrays

- Programmable interconnect support flexible patterns
  - e.g. Mesh network on-chip (NoC)

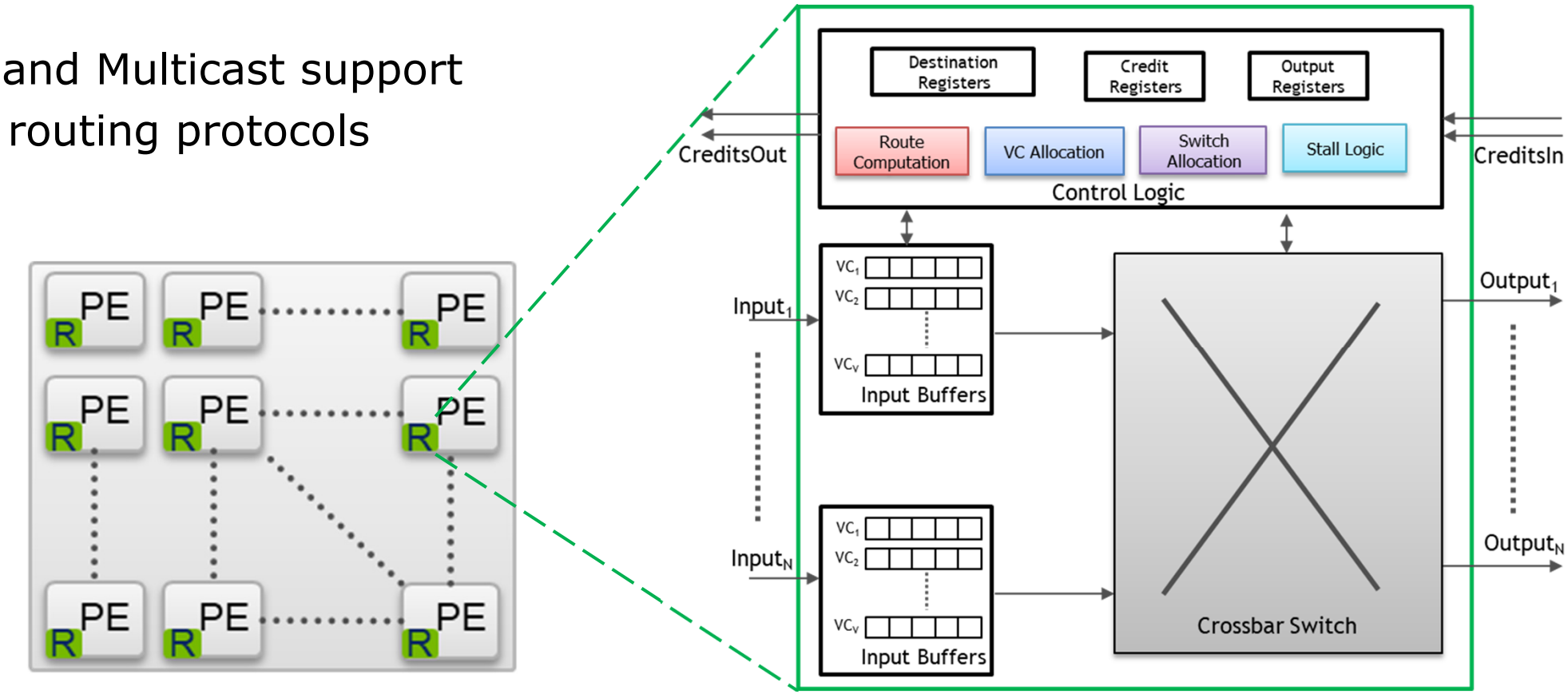
Systolic Array



Sze et al., *Synthesis Lectures on Computer Architecture, 2020*

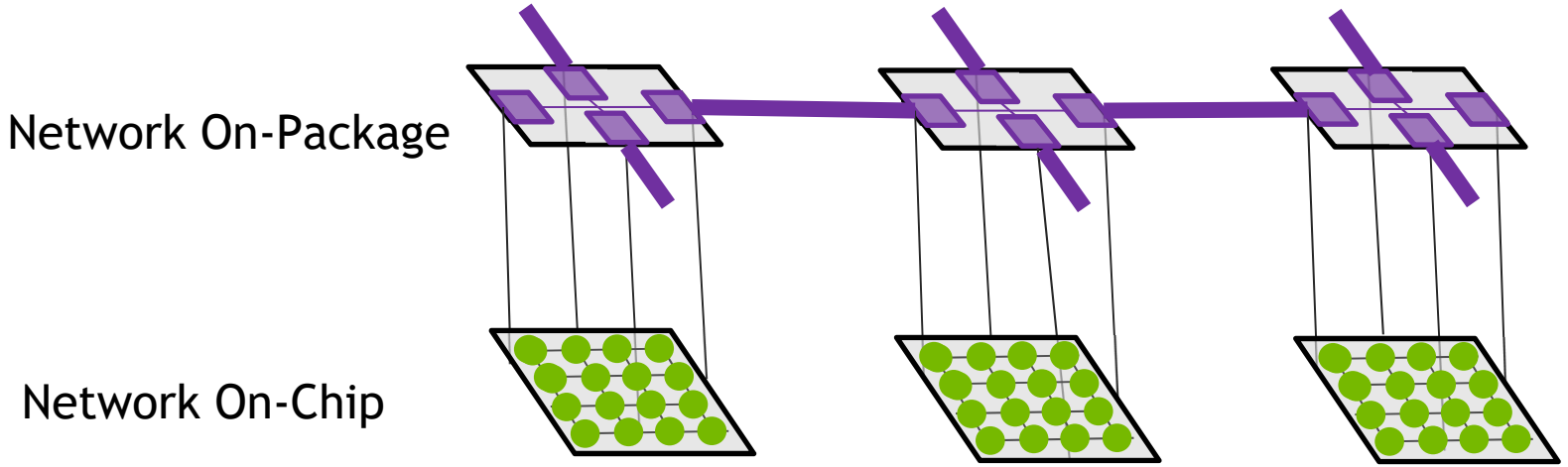
Mesh NoC

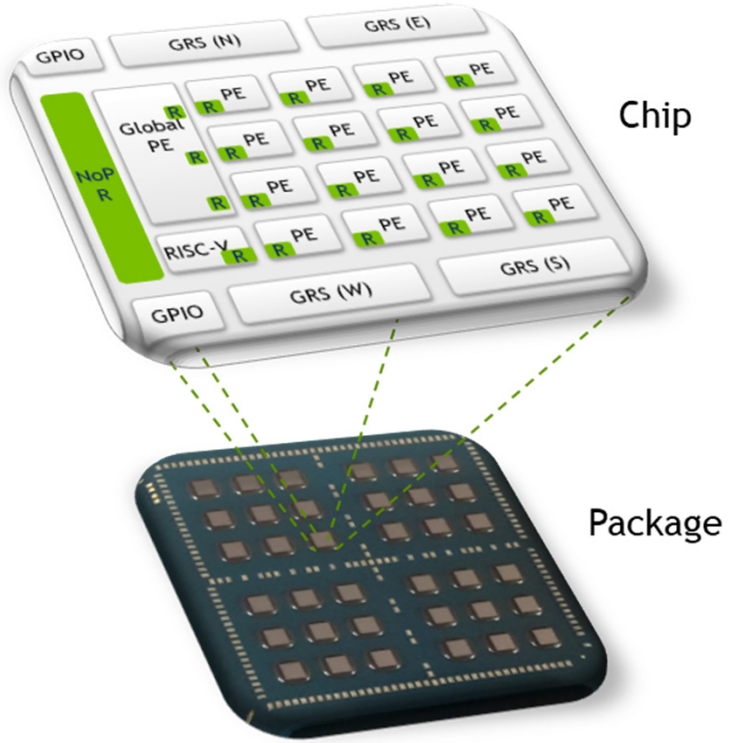
- Different packet sizes for different data types
- Unicast and Multicast support
- Flexible routing protocols



Hierarchical Network

- Efficient communication for large scale designs
- Reduces number of hops
- Reduces congestion
- Example:
  - Multi-Chip Module (MCM) based DL accelerator





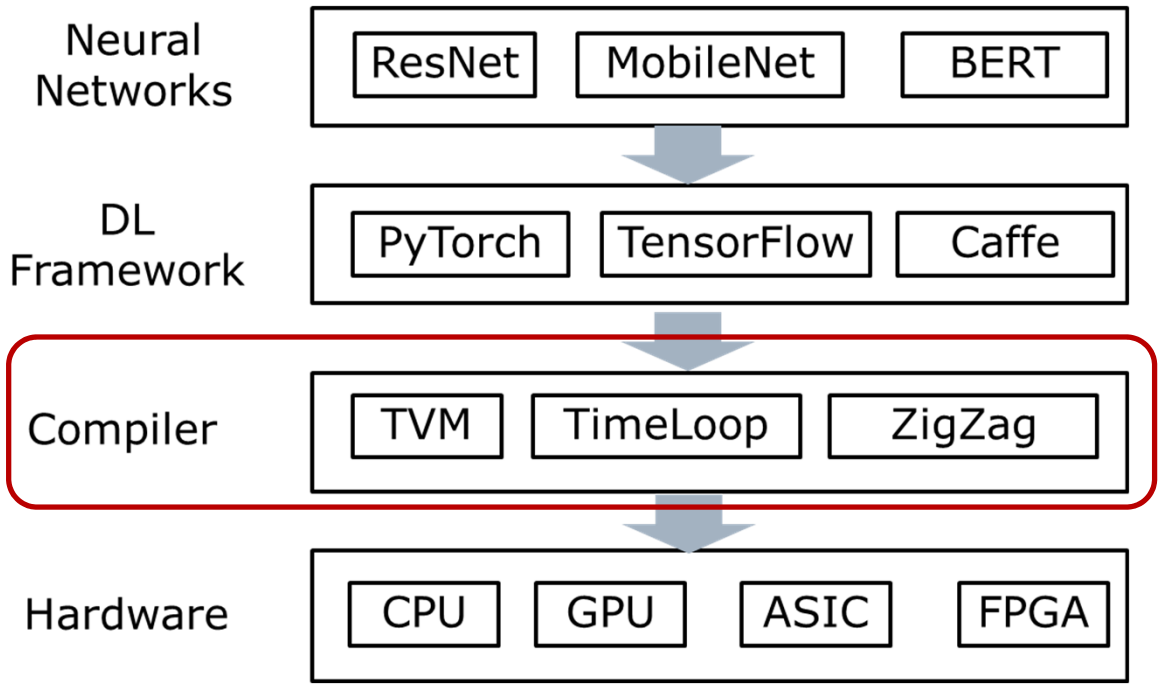
Venkatesan et al., *HotChips 2019*
Zimmer et al. *JSSC 2020*

Outline

- Motivation for Hardware Acceleration
- Deep Neural Networks (DNN): Basics
- Accelerator Architecture Design
  - Compute: Efficient MAC Datapath
  - Storage: Memory hierarchy
  - Communication: Interconnect topologies
- Compiler Mapping flow
  - Data reuse, Parallelism, Pipelining
- Neural Network Optimizations
  - Quantization
  - Sparsity

Compiler Mapping Flow

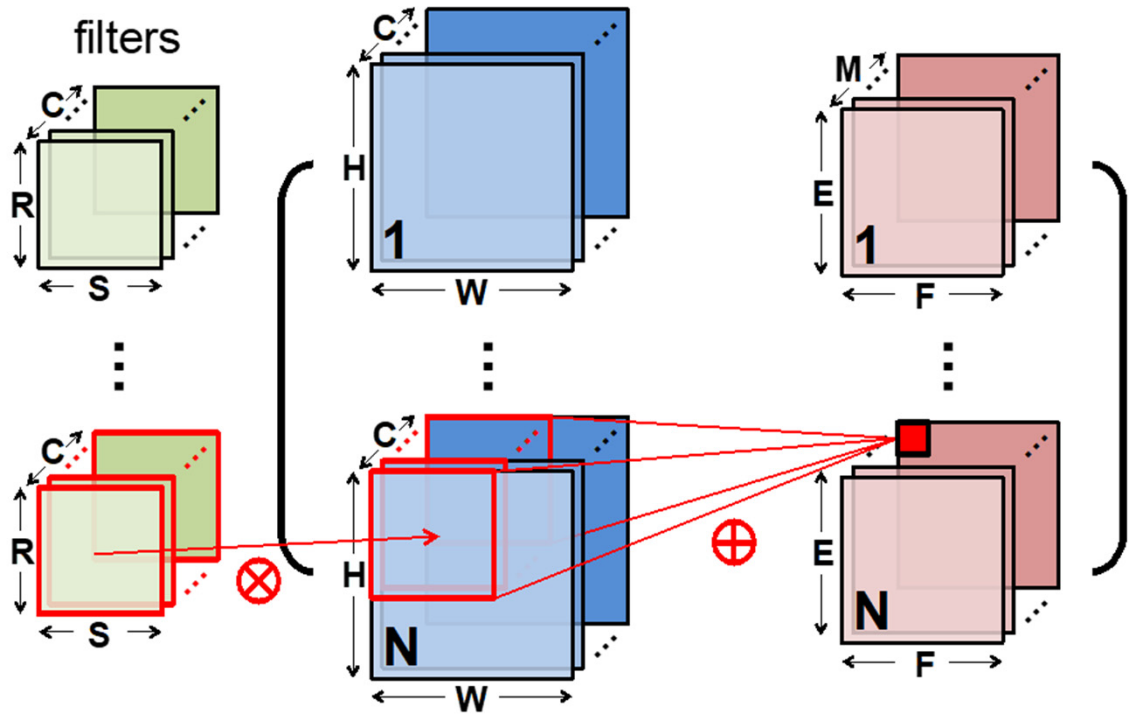
- Goal:
  - Efficiently execute the neural network in the target hardware
    - Performance
    - Energy efficiency
- Opportunities
  - Data reuse
  - Parallelism
  - Pipelining



Data Reuse Opportunities

Data type	Reuse
Input activations*	R * S * M
Weights	E * F * N
Output activations	R * S * C

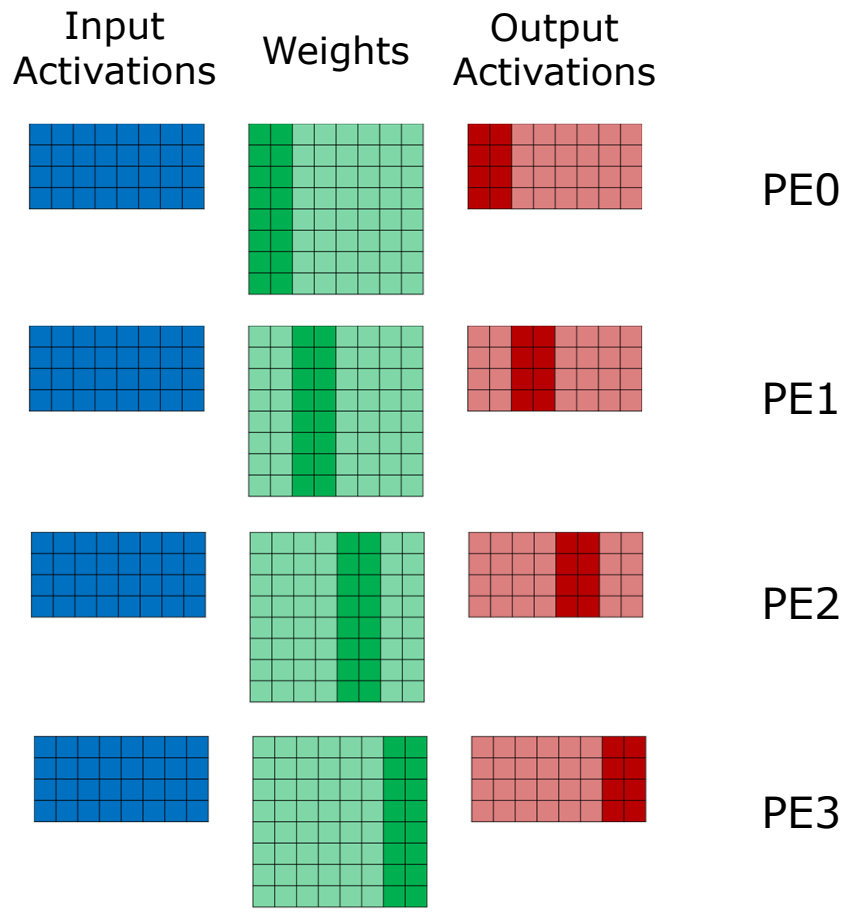
*except halo



Sze et al., *Synthesis Lectures on Computer Architecture, 2020*

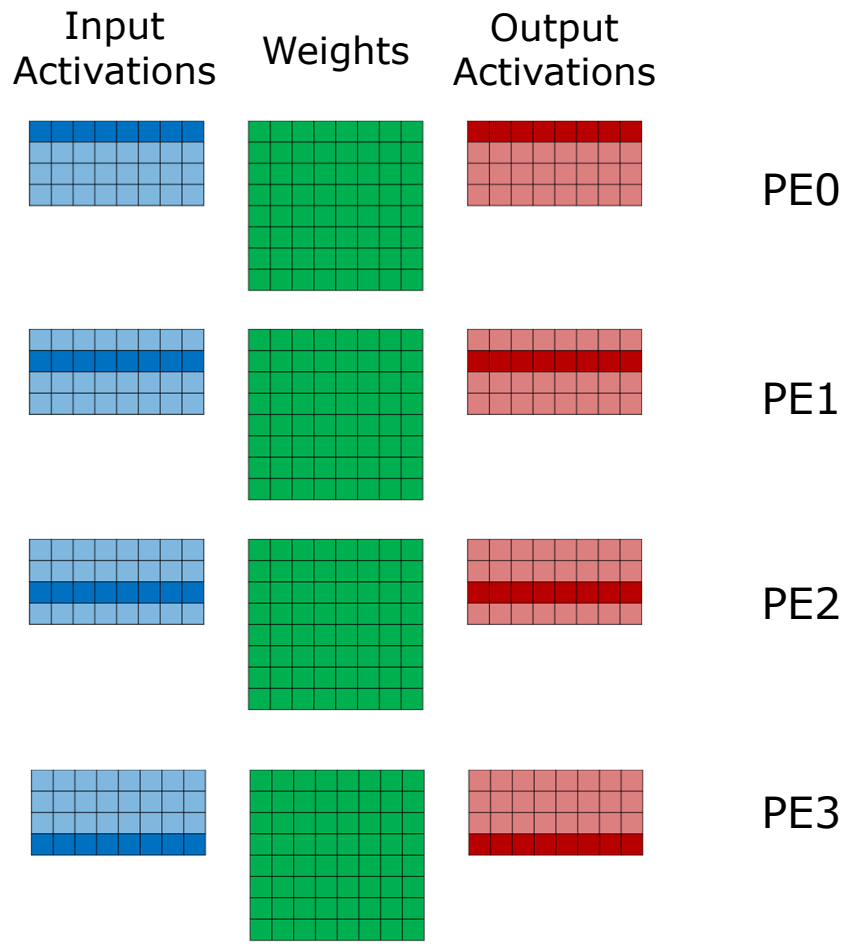
Model Parallelism

- Exploit parallelism across weights in a layer
- Example
  - An architecture implementing weight-stationary dataflow
  - Tile weights and distribute them to different PEs
  - Compute different output activations by streaming in the input activations



Data Parallelism

- Exploiting parallelism across activations in a layer
- Example
  - An architecture implementing input-stationary dataflow
  - Tile input activations and map to different PEs
  - Each PE computes different output activations by streaming in the weights



Mapping Tool: TimeLoop

- Loop-nest representation of convolution layer

```
for n=[0:N)
 for m=[0:M)
  for e=[0:E)
   for f=[0:F)
    for c=[0:C)
     for r=[0:R)
      for s=[0:S)
       Out[e][f][m][n]  +=
         Weight[r][s][c][m]  *
         Input[e+r][f+s][c][n]
```

- Loop-nest representation of an DNN accelerator

```
for n2=[0:N2)
 for e2=[0:E2)
  for f2=[0:F2)
   for m2=[0:M2)
    for c2=[0:C2)
```

System

```
for n1=[0:N1)
 for e1=[0:E1)
  for f1=[0:F1)
   for m1=[0:M1)
    for c1=[0:C1)
     for r1=[0:R)
      for s1=[0:S)
```

PE

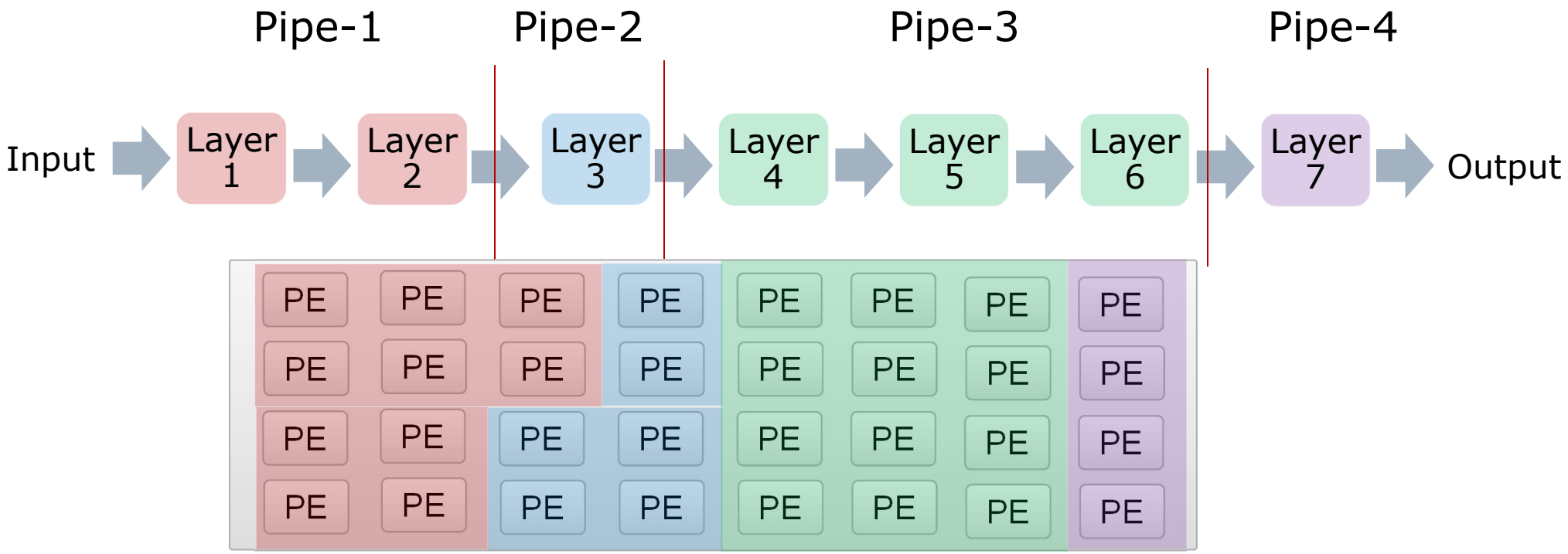
```
for c0=[0:C0)
 for m0=[0:M0)
  for e0=[0:E0)
   for f0=[0:F0)
    MACs
```

MAC unit

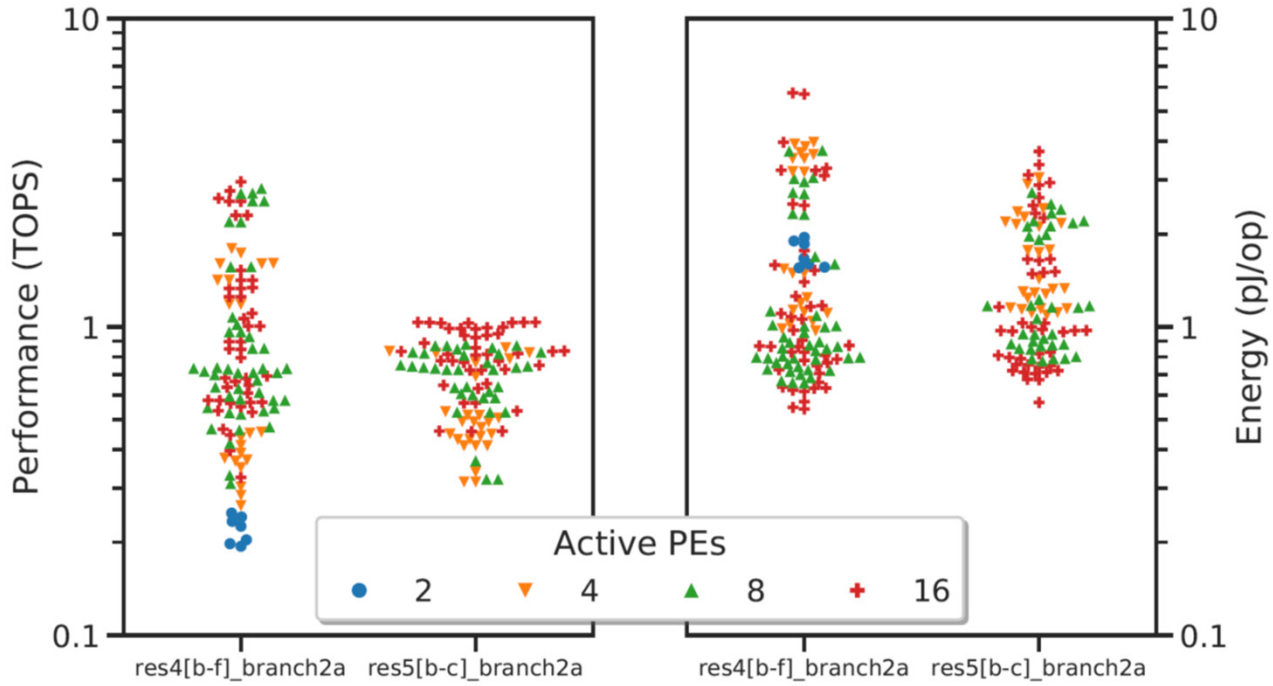
Parashar et al. *ISPASS, 2019.*

Pipelining

- Parallelism across layers of the network
- Execute one or more layers across different processing elements



Impact of Tiling on Hardware Efficiency



- Large number of possible tilings for a given layer and hardware configuration
- >10x difference in performance and energy
- Need to explore optimized tiling to achieve best energy and performance

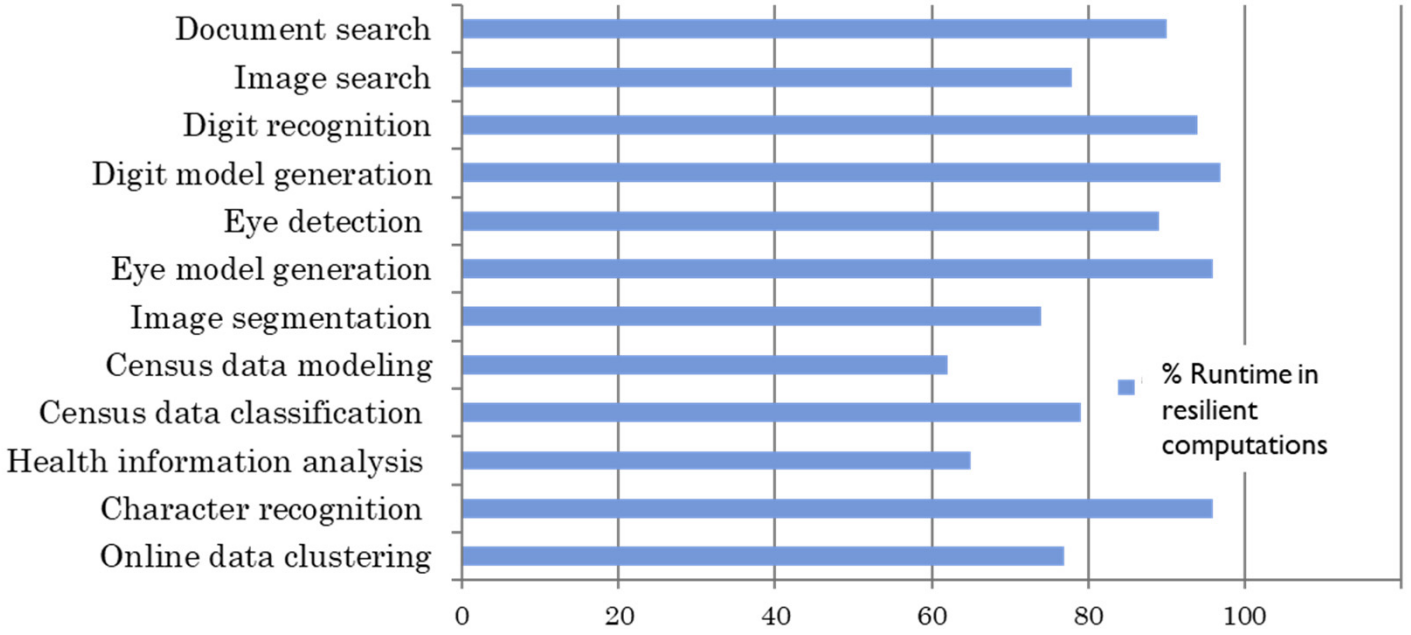
Venkatesan et al., *ICCAD 2019*

Outline

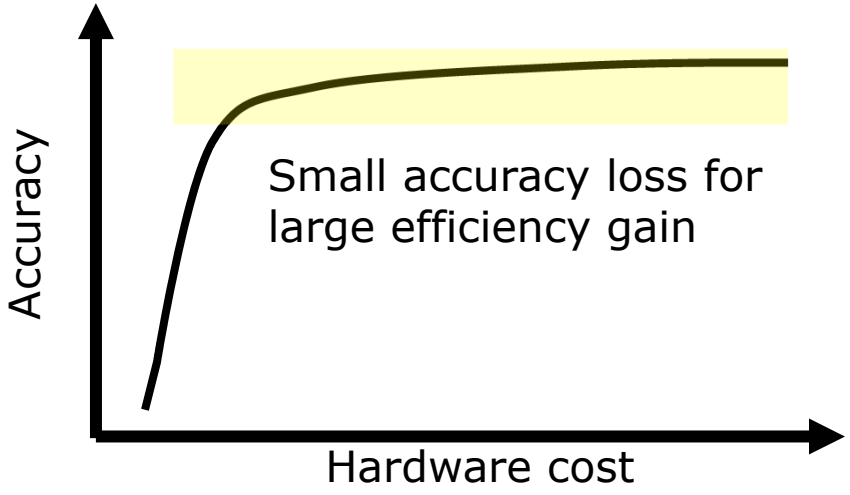
- Motivation for Hardware Acceleration
- Deep Neural Networks (DNN): Basics
- Accelerator Architecture Design
  - Compute: Efficient MAC Datapath
  - Storage: Memory hierarchy
  - Communication: Interconnect topologies
- Compiler Mapping flow
  - Data reuse, Parallelism, Pipelining
- Neural Network Optimizations
  - Quantization
  - Sparsity

Quantization

- Exploit application-level resiliency to perform compute at reduced precision
  - Improves energy efficiency, area efficiency and performance
    - Lower storage
    - Cheaper compute



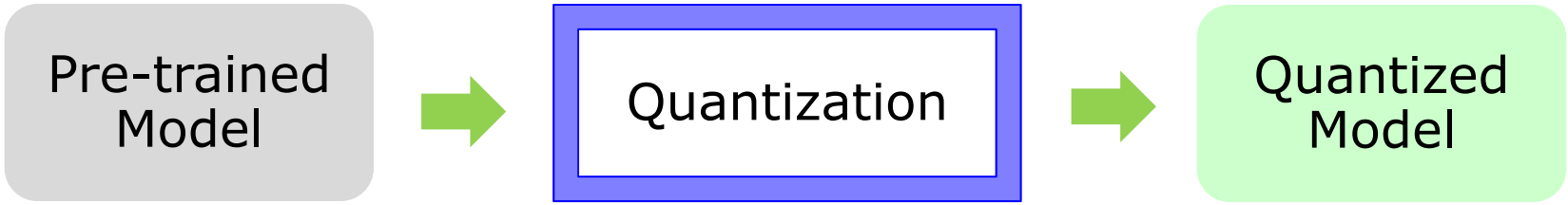
Chippa et al., *DAC 2013*



Accuracy vs. Efficiency tradeoff

Quantization

- Post-training Quantization

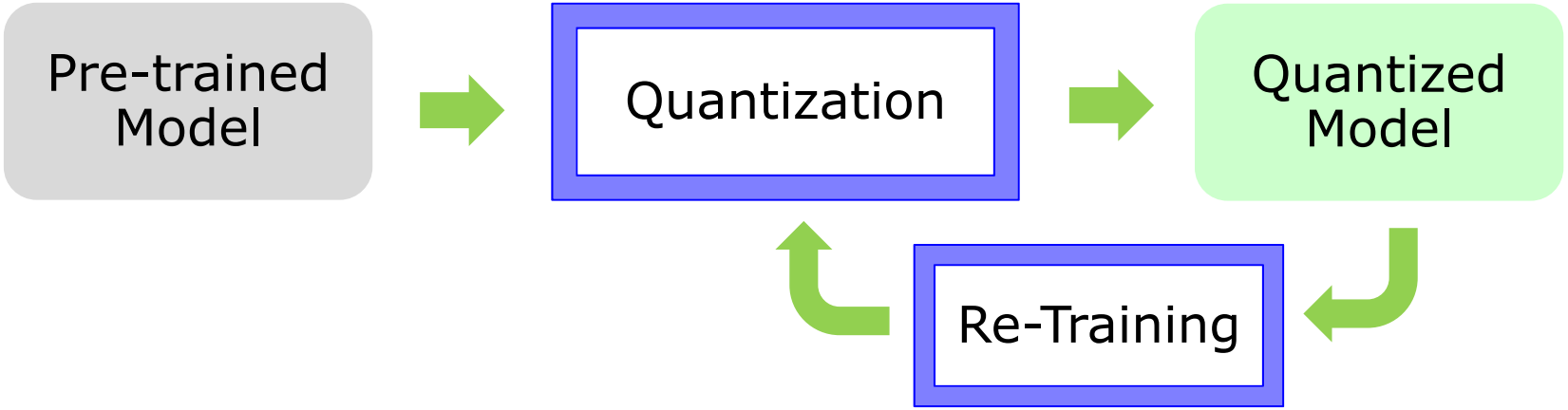


- Pretrained model is quantized to improve efficiency
- No need for training sets during model deployment

Wu et al. *arXiv, 2020*

Quantization

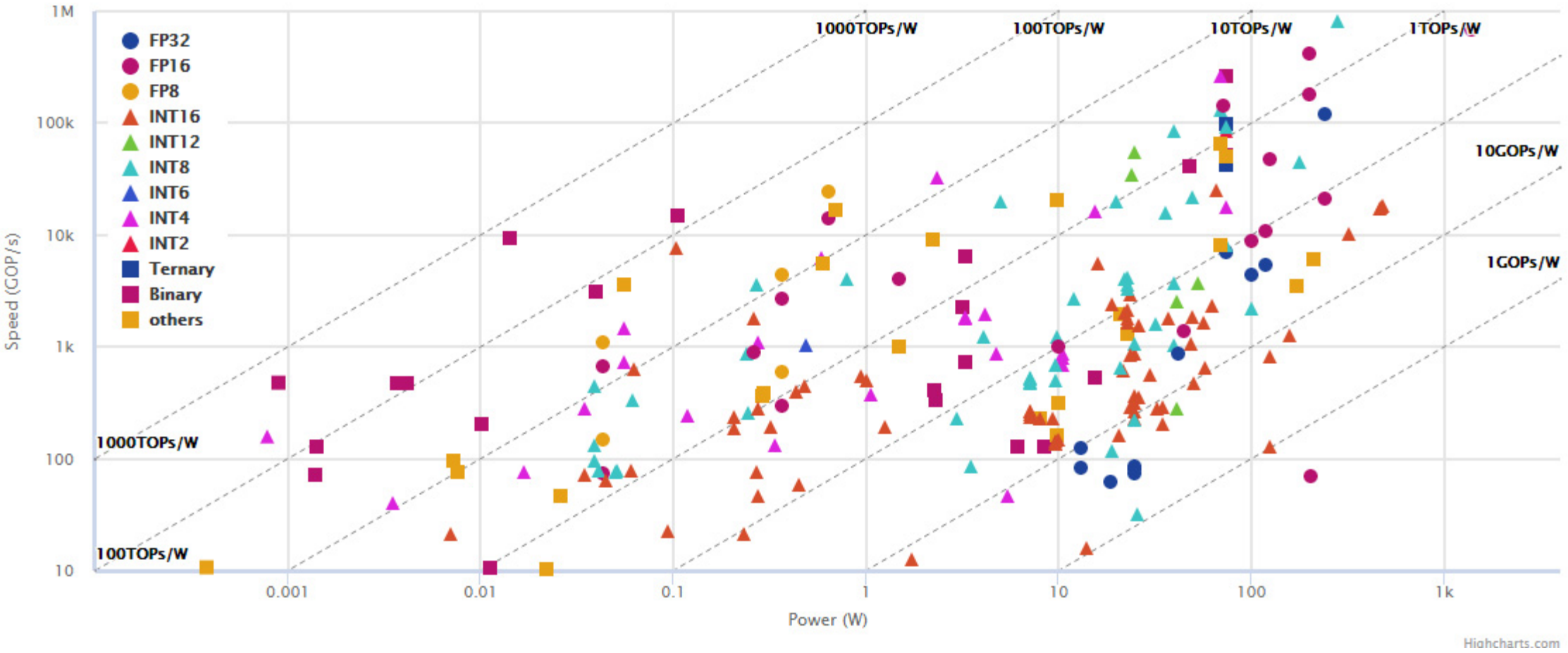
- Quantization-Aware Training



- Quantization to improve efficiency
- Retraining to recover accuracy

Wu et al. *arXiv, 2020*

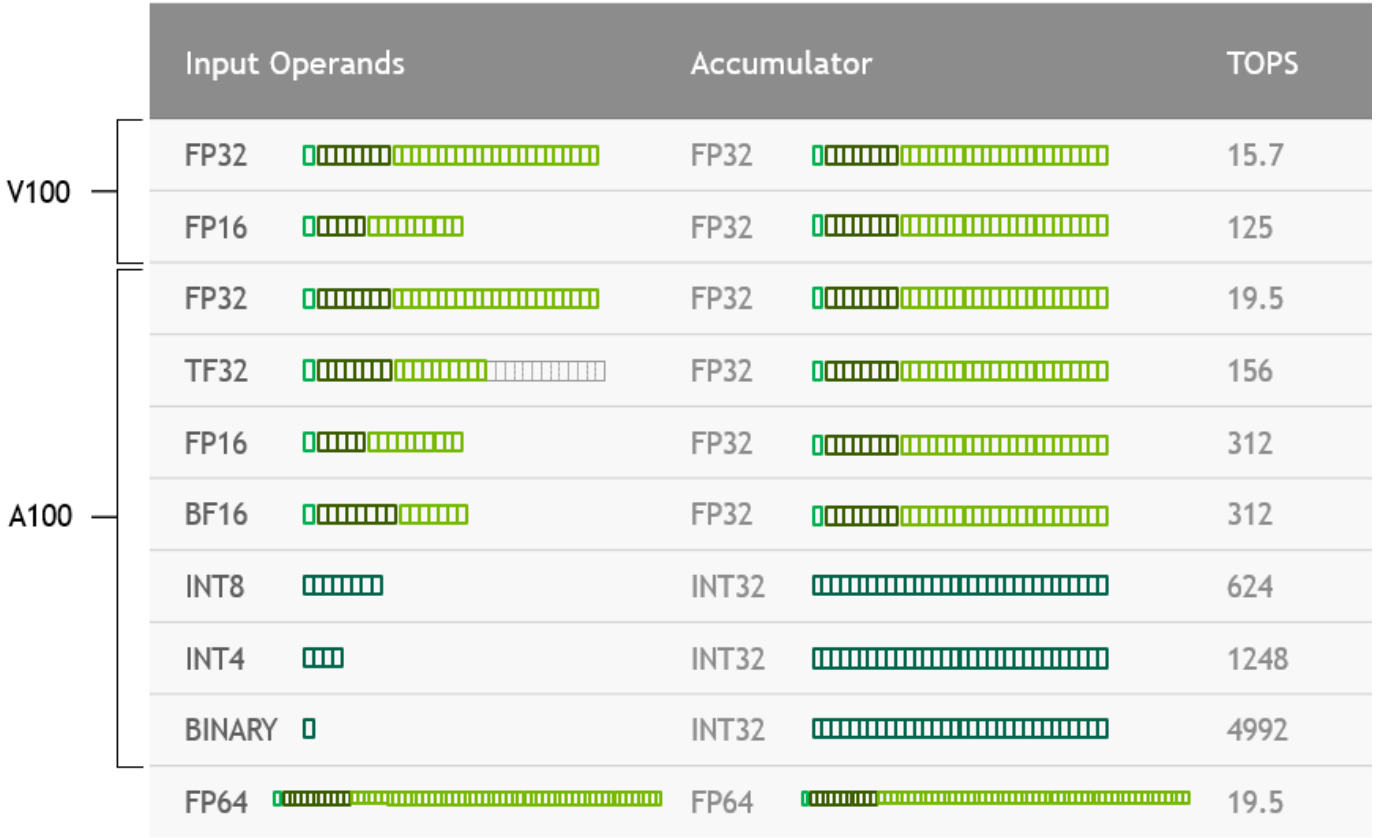
Benefits of Quantization



Source: https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/

Multiple Precision Support

- NVIDIA GPUs: Volta V100 and Ampere A100



	Input Operands	Accumulator	TOPS
V100	FP32	FP32	15.7
	FP16	FP32	125
A100	FP32	FP32	19.5
	TF32	FP32	156
	FP16	FP32	312
	BF16	FP32	312
	INT8	INT32	624
	INT4	INT32	1248
	BINARY	INT32	4992
	FP64	FP64	19.5

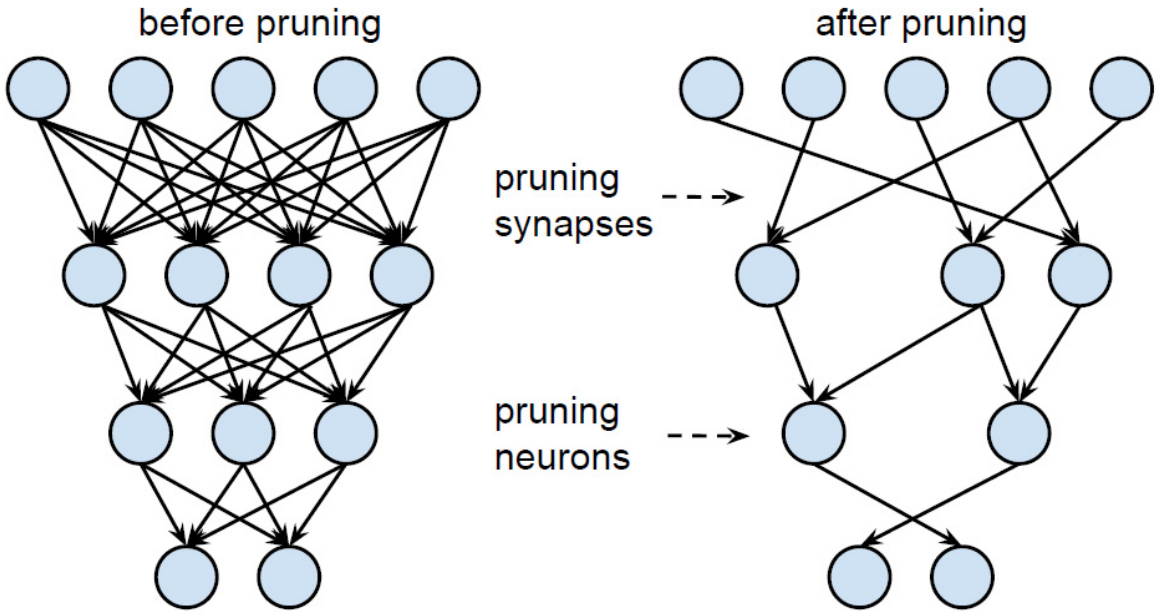
Ref: [NVIDIA A100 Tensor Core GPU Architecture whitepaper]

Outline

- Motivation for Hardware Acceleration
- Deep Neural Network: Basics
- Accelerator Architecture Design
  - Compute: Efficient MAC Datapath
  - Storage: Memory hierarchy
  - Communication: Interconnect topologies
- Compiler Mapping flow
  - Data reuse, Parallelism, Pipelining
- Neural Network Optimizations
  - Quantization
  - Sparsity

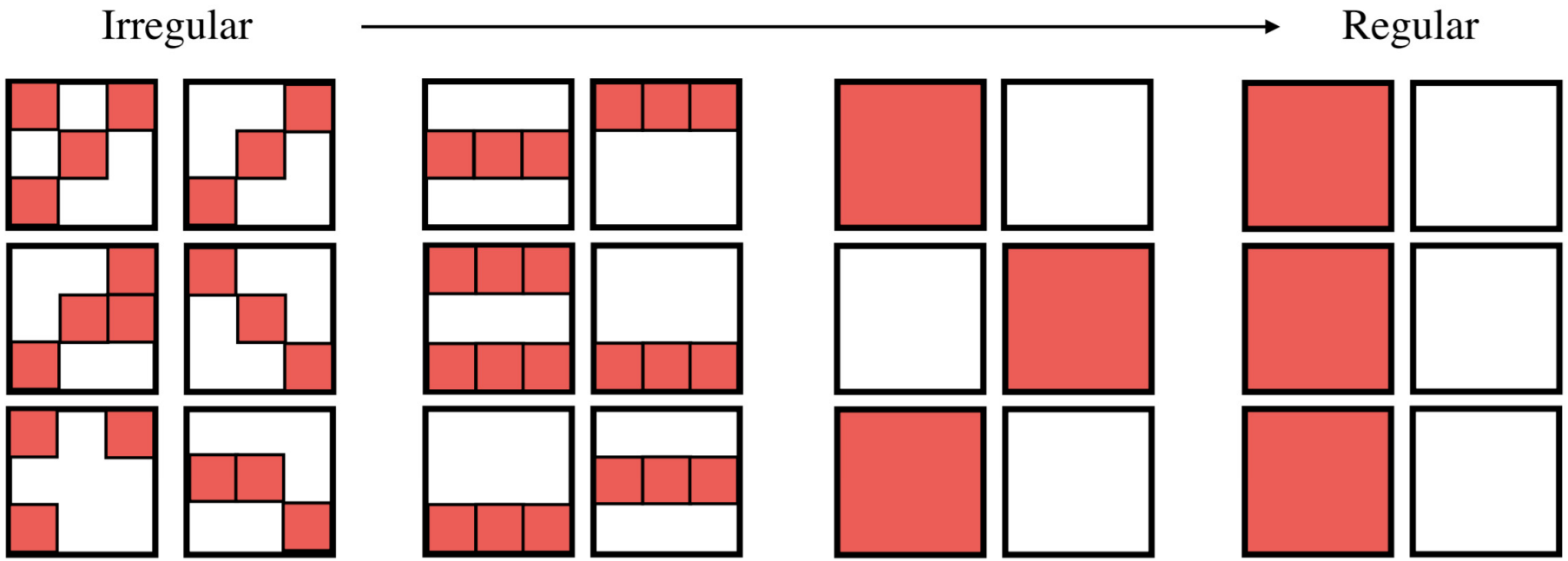
Sparsity

- Neural networks exhibit high degree of sparsity
  - Weights/connections are sparse
  - Activations at intermediate stages are sparse



Han et al., *NeurIPS* 2015

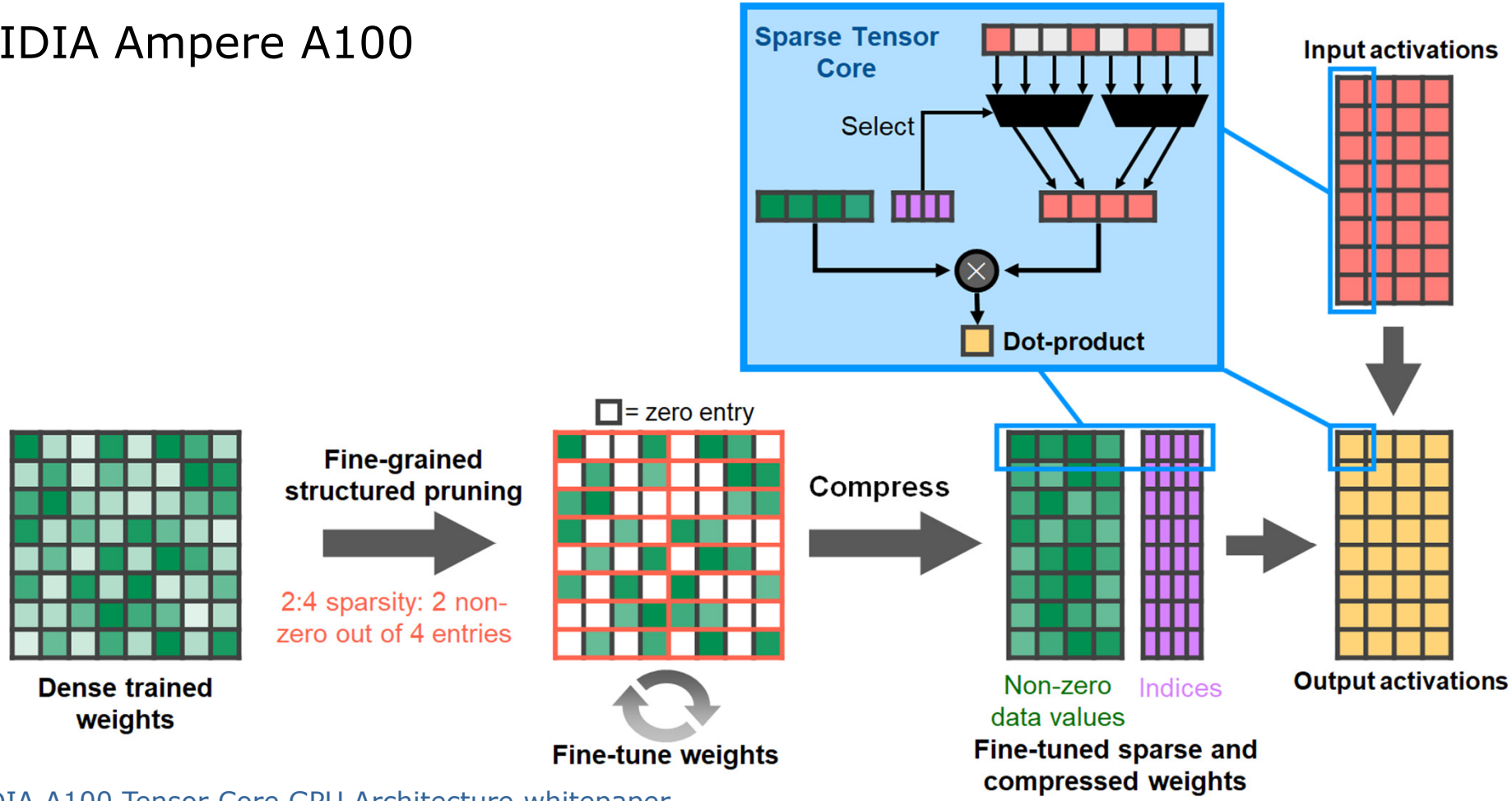
Types of Sparsity



Mao et al. NeurIPS 2017

Structured Sparsity

- NVIDIA Ampere A100



Ref: [NVIDIA A100 Tensor Core GPU Architecture whitepaper]

NVIDIA A100 Performance

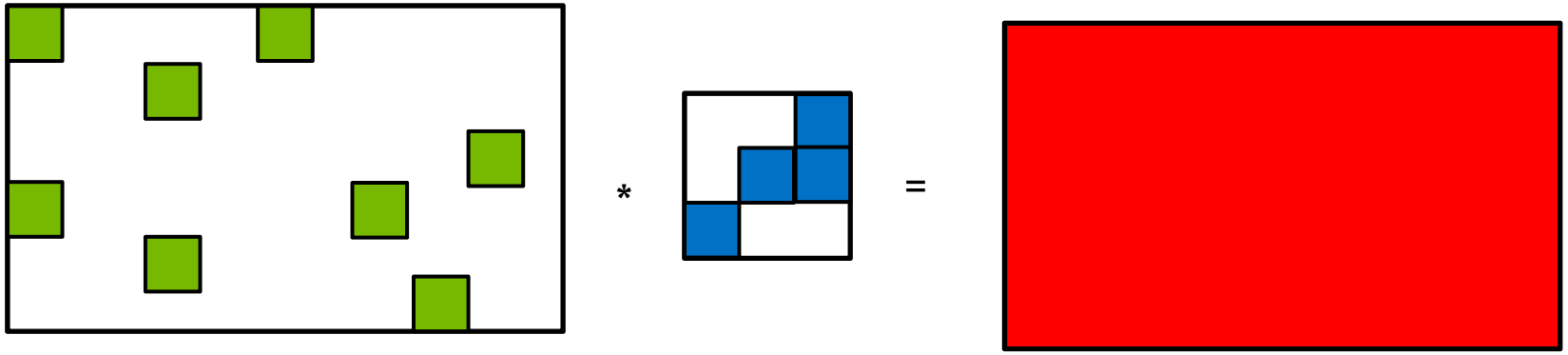
Peak FP64[1]	9.7 TFLOPS
Peak FP64 Tensor Core[1]	19.5 TFLOPS
Peak FP32[1]	19.5 TFLOPS
Peak FP16[1]	78 TFLOPS
Peak BF16[1]	39 TFLOPS
Peak TF32 Tensor Core[1]	156 TFLOPS | 312 TFLOPS[2]
Peak FP16 Tensor Core[1]	312 TFLOPS | 624 TFLOPS[2]
Peak BF16 Tensor Core[1]	312 TFLOPS | 624 TFLOPS[2]
Peak INT8 Tensor Core[1]	624 TOPS | 1,248 TOPS[2]
Peak INT4 Tensor Core[1]	1,248 TOPS | 2,496 TOPS[2]

1 - Peak rates are based on GPU Boost Clock.
2 - Effective TFLOPS / TOPS using the new Sparsity feature

Ref: NVIDIA A100 Tensor Core GPU Architecture whitepaper

Unstructured Sparsity

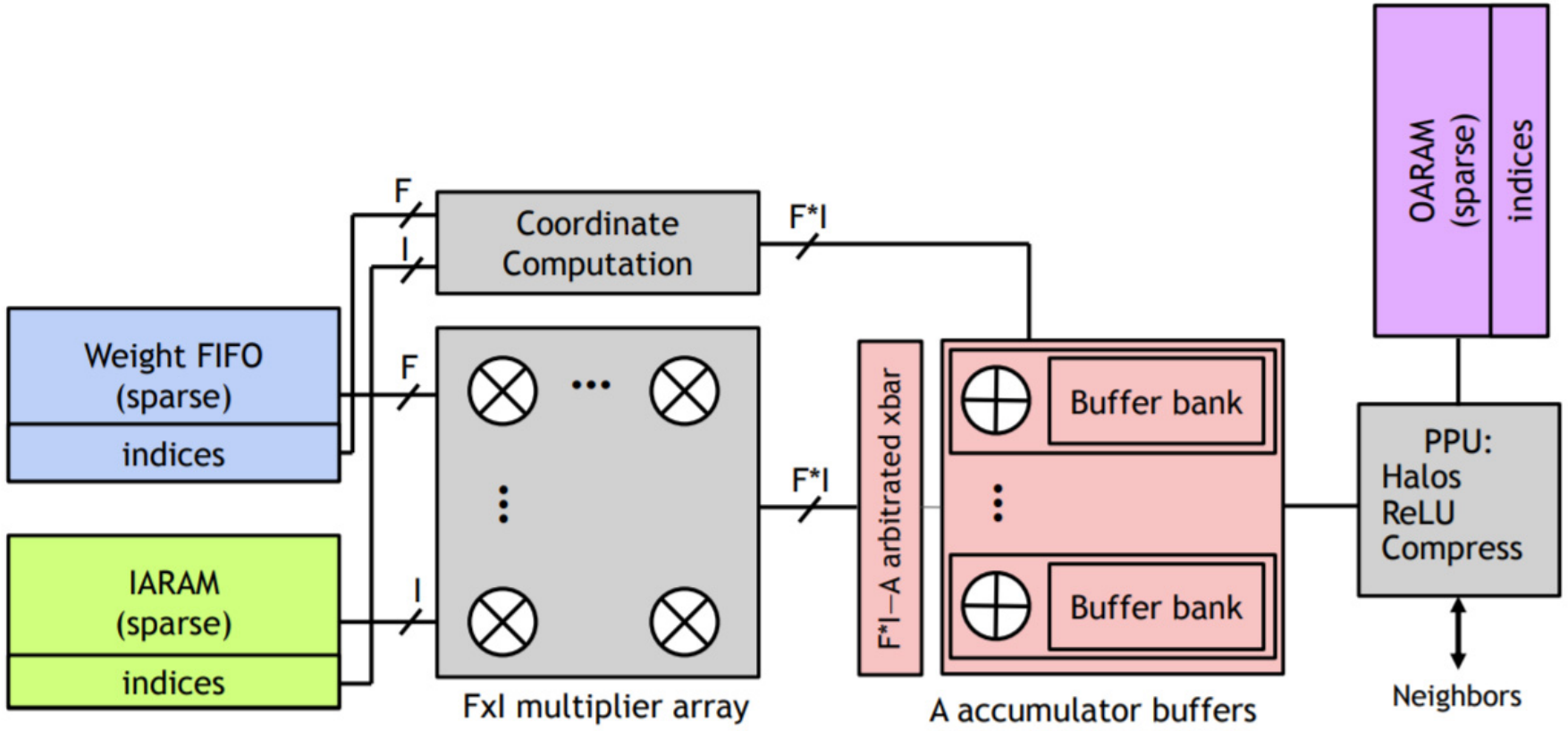
- SCNN
  - Only compute partial products where both operands are non-zero
  - Get rid of the idea of sliding convolution: doesn't make sense when most of the operands are 0
  - Vector ops are questionable: most elements of your vector are 0, don't know a priori which ones or how many



Parashar et al. *ISCA 2017*

Unstructured Sparsity

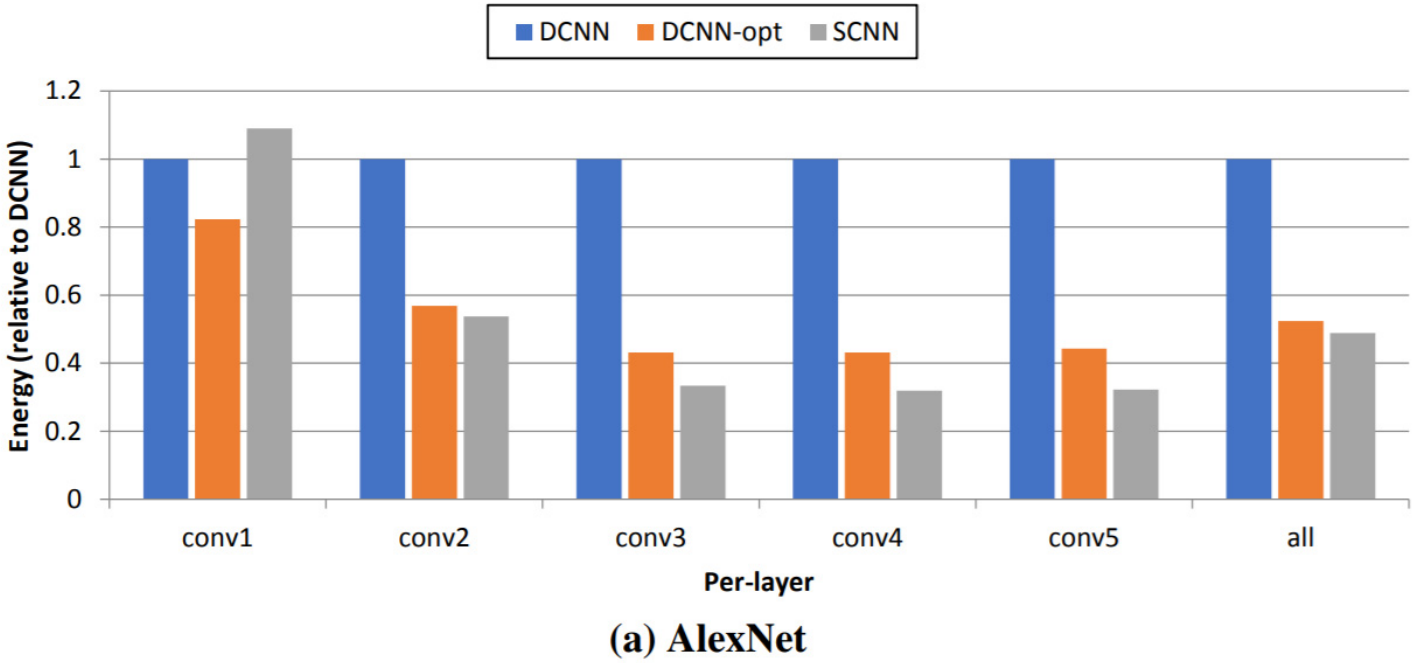
- SCNN



Parashar et al. *ISCA 2017*

Unstructured Sparsity

- SCNN

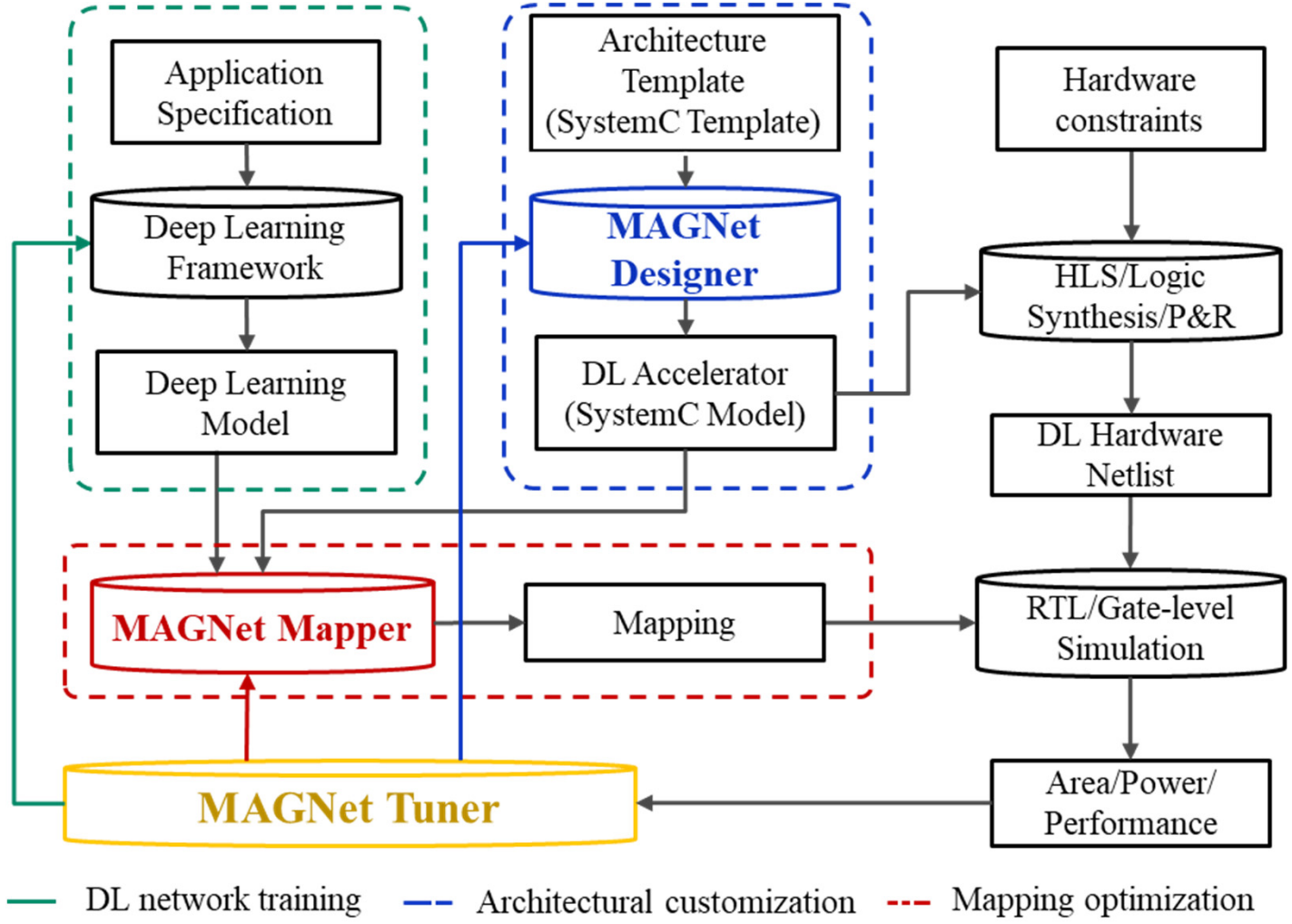


(a) AlexNet

- SCNN achieves 2.3X improvement in energy efficiency over Dense CNN (DCNN) accelerator

Parashar et al. *ISCA 2017*

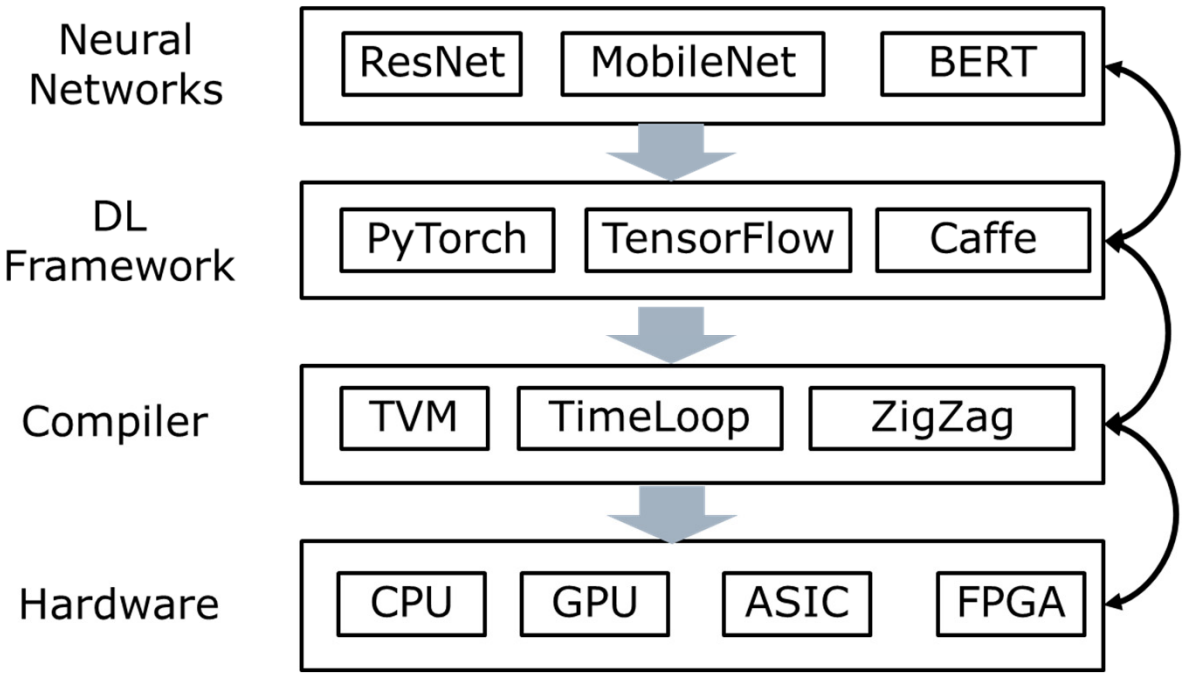
An End-to-End Optimization Flow



Venkatesan et al., *ICCAD 2019*

Summary

- Deep neural networks are increasing used across a wide range of applications
  - Large amounts of data
  - High computation demand
- Hardware acceleration is key for continued growth
- Co-design across algorithm-compiler-hardware can greatly improve efficiency



Papers to watch @ ISSCC 2021

- Session 9: ML Processors From Cloud to Edge
- 9.1 A 7nm 4-Core AI Chip with 25.6TFLOPS Hybrid FP8 Training, 102.4TOPS INT4 Inference and Workload-Aware Throttling
- 9.2 A 28nm 12.1TOPS/W Dual-Mode CNN Processor Using Effective-Weight-Based Convolution and Error-Compensation-Based Prediction
- 9.3 A 40nm 4.81TFLOPS/W 8b Floating-Point Training Processor for Non-Sparse Neural Networks Using Shared Exponent Bias and 24-Way Fused Multiply-Add Tree
- 9.4 PIU: A 248GOPS/W Stream-Based Processor for Irregular Probabilistic Inference Networks Using Precision-Scalable Posit Arithmetic in 28nm
- 9.5 A 6K-MAC Feature-Map-Sparsity-Aware Neural Processing Unit in 5nm Flagship Mobile
- 9.6 A 1/2.3inch 12.3Mpixel with On-Chip 4.97TOPS/W CNN Processor Back-Illuminated Stacked CMOS Image
- 9.7 A 184μW Real-Time Hand-Gesture Recognition System with Hybrid Tiny Classifiers for Smart Wearable Devices
- 9.8 A 25mm2 SoC for IoT Devices with 18ms Noise-Robust Speech-to-Text Latency via Bayesian Speech Denoising and Attention-Based Sequence-to-Sequence DNN Speech Recognition in 16nm
- 9.9 A Background-Noise and Process-Variation-Tolerant 109nW Acoustic Feature Extractor Based on Spike-Domain Divisive-Energy Normalization for an Always-On Keyword Spotting Device

Papers to watch @ ISSCC 2021

- Session 15: Compute-in-Memory Processors for Deep Neural Networks
  - 15.1 A Programmable Neural-Network Inference Accelerator Based on Scalable In-Memory Computing
  - 15.2 A 2.75-to-75.9TOPS/W Computing-in-Memory NN Processor Supporting Set-Associate Block-Wise Zero Skipping and Ping-Pong CIM with Simultaneous Computation and Weight Updating
  - 15.3 A 65nm 3T Dynamic Analog RAM-Based Computing-in-Memory Macro and CNN Accelerator with Retention Enhancement, Adaptive Analog Sparsity and 44TOPS/W System Energy Efficiency
  - 15.4 A 5.99-to-691.1TOPS/W Tensor-Train In-Memory-Computing Processor Using Bit-Level-Sparsity- Based Optimization and Variable-Precision Quantization
- Session 16: Compute-in-Memory
  - 16.1 A 22nm 4Mb 8b-Precision ReRAM Computing-in-Memory Macro with 11.91 to 195.7TOPS/W for Tiny AI Edge Devices
  - 16.2 eDRAM-CIM: Compute-In-Memory Design with Reconfigurable Embedded-Dynamic-Memory Array Realizing Adaptive Data Converters and Charge-Domain Computing
  - 16.3 A 28nm 384kb 6T-SRAM Computation-in-Memory Macro with 8b of Precision for AI Edge Chips
  - 16.4 An 89TOPS/W and 16.3TOPS/mm2 All-Digital SRAM-Based Full-Precision Compute-In Memory Macro in 22nm for Machine-Learning Edge Applications

References

- Overview and Benchmarking
  - V. Sze et al., "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE 2017
  - V. Sze et al., "Efficient Processing of Deep Neural Networks," Synthesis Lectures on Computer Architecture 2020
  - K. Guo et al., "Neural network accelerator comparison," [online]https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/.
  - V. J. Reddi et al., "MLPerf Inference Benchmark," arXiv 2019
  - V. Camus et al., "Survey of precision-scalable multiply-accumulate units for neural-network processing," AICAS 2019
  - H. Wu et al. "Integer quantization for deep learning inference: Principles and empirical evaluation" arXiv 2020

References

- Deep Learning Hardware
  - F. Sijstermans, "The NVIDIA deep learning accelerator," in Hot Chips 2018
  - NVIDIA A100 Tensor Core GPU Architecture whitepaper
  - B. Zimmer et al., "A 0.11pJ/Op, 0.32-128 TOPS, Scalable Multi-Chip-Module-based Deep Neural Network Accelerator with Ground-Reference Signaling in 16nm," VLSI 2019
  - N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in ISCA 2017
  - A. Parashar et al., "SCNN: An accelerator for compressed-sparse convolutional neural networks," in ISCA 2017
  - Y. Chen et al., "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," ISSCC 2016
  - S. Han et al., "EIE: Efficient inference engine on compressed deep neural network," in ISCA 2016

References

- Deep Learning Hardware
  - E. H. Lee et al., "LogNet: Energy-efficient neural networks using logarithmic computation," ICASSP 2017
  - H. Wu, "Low precision inference on GPUs" GTC 2019
  - J. R. Stevens et al. "Manna: An Accelerator for Memory-Augmented Neural Networks" MICRO 2019.
  - S. Han et al. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding" NeurIPS 2015
  - S. Venkataramani et al. "AxNN: energy-efficient neuromorphic systems using approximate computing" ISLPED 2014
  - Y. Lecun et al., Optimal Brain Damage," NeurIPS 1990

References

- Deep Neural Networks
  - A. Krizhevsky et al. “Imagenet classification with deep convolutional neural networks,” NeurIPS 2012
  - K. He et al. “Deep residual learning for image recognition,” CVPR 2016
  - M. Tan et al., “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” ICML 2019
  - A. Vaswani et al., “Attention is all you need,” NeurIPS 2017
  - M. Shoeybi et al., “Megatron-LM: Training MultiBillion Parameter Language Models Using Model Parallelism,” arXiv 2019
  - J. Choi et al. “PACT: Parameterized Clipping Activation for Quantized Neural Networks”, arxiv 2018
  - R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper”, arxiv 2018
  - A. Mishra et al. “Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy” ICLR 2018

References

- Modeling, Mapping and Exploration Tools
  - A. Parashar et al., “Timeloop: A Systematic Approach to DNN Accelerator Evaluation,” ISPASS 2019
  - R. Venkatesan et al., “MAGNet: A modular aaccelerator generator for neural networks.” ICCAD 2019
  - Y. N. Wu et al., “Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs,” ICCAD 2019
  - X. Yang et al., “Interstellar: using Halide’s scheduling language toanalyze DNN accelerators,” ASPLOS 2020
  - S. Jain et al., “RxNN: a frameworkfor evaluating deep neural networks on resistive crossbars,” TCAD 2020
  - T. Chen et al., “TVM: an automated end-to-end optimizing compiler for deep learning,” OSDI 2018
  - L. Mei et al., “ZigZag: A memory-centric rapid DNN accelerator design space exploration framework,” arXiv 2020