Edge-AI (Hardware)

Luis Piñuel

ArTeCS - UCM

Section 1

Intro

DNN computational complexity

- Large number of weights (high storage demand)
- Large number of operations (high computational complexity)
    - E.g. for each CNN layer: num. ops / weight = 2 x FMapSize

Common SW optimizations

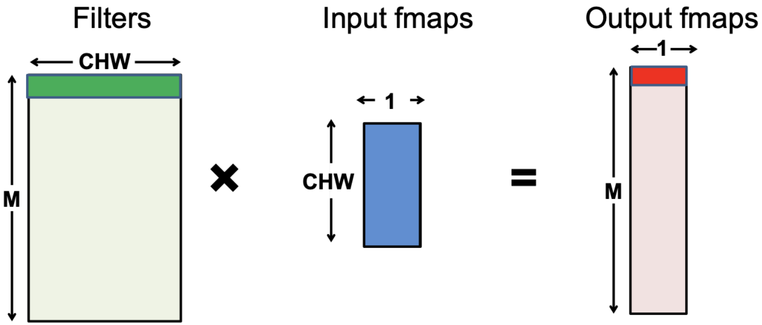
- Batch processing
  - Reuse weights for several input Fmaps (i.e. reduce data movements)
- Quantization:
  - Reduce storage/latency/energy per weight MAC
- Network pruning
  - Reduce number of weights without reducing accuracy (e.g. zero weights)
- Efficient kernel processing
  - Matrix-vector, matrix-matrix
  - Stencil
  - ReLU, Sigmoid, Htan

Efficient kernel processing

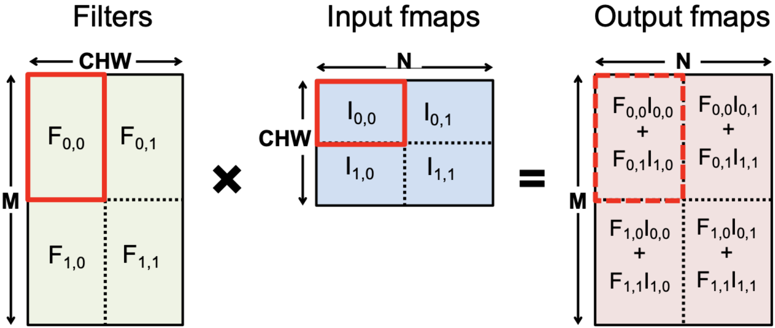
- Algorithmic transformations
  - Strassen multiplication
  - Winograd filter
- Efficient implementation
  - Cache blocking

Fully-Connected (FC) Layer

- Matrix–Vector Multiply:
  - Multiply all inputs in all channels by a weight and sum



Tiled Fully-Connected (FC) Layer

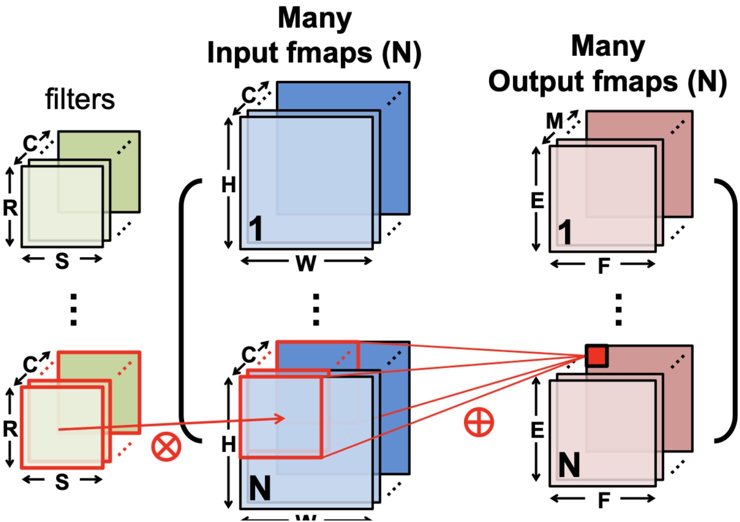


Matrix multiply tiled to fit in cache
and computation ordered to maximize reuse of data in cache

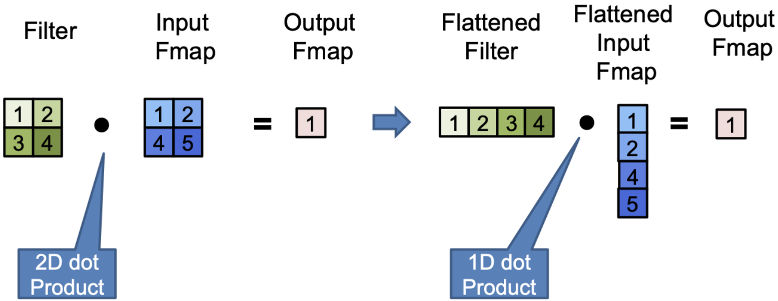
Fully-Connected (FC) Layer

- Implementation: **Matrix Multiplication (GEMM)**
  - **CPU:** OpenBLAS, Intel MKL, etc
  - **GPU:** cuBLAS, cuDNN, etc
- Library will note shape of the matrix multiply and select implementation optimized for that shape.
- Optimization usually involves proper tiling to storage hierarchy

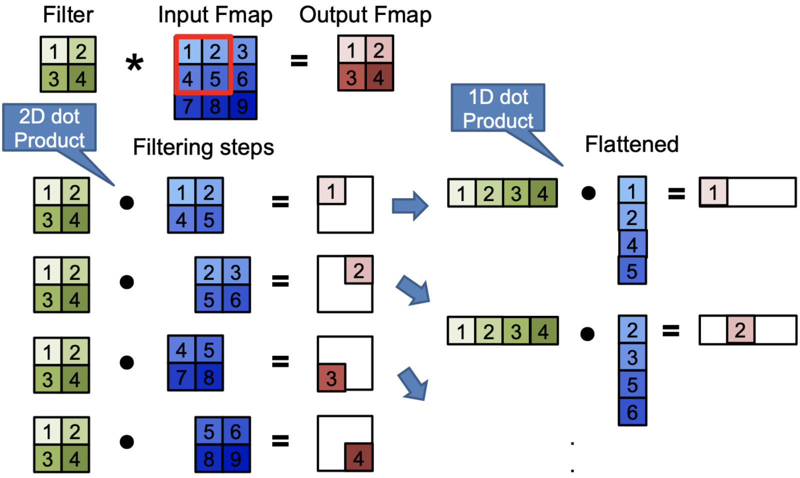
Convolution (CONV) Layer



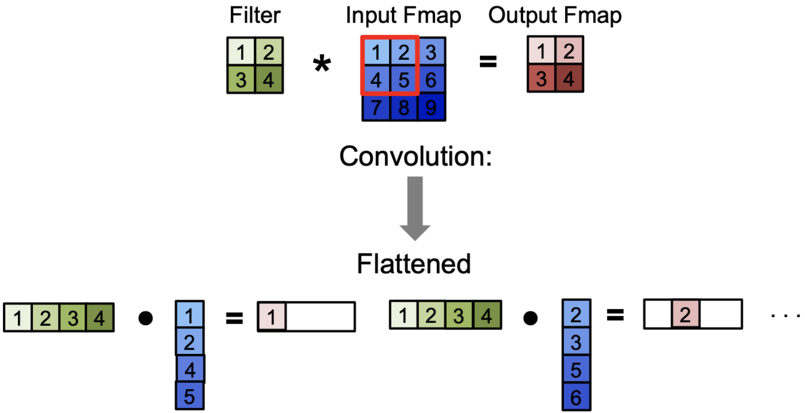
Flattened 2D Dot Product



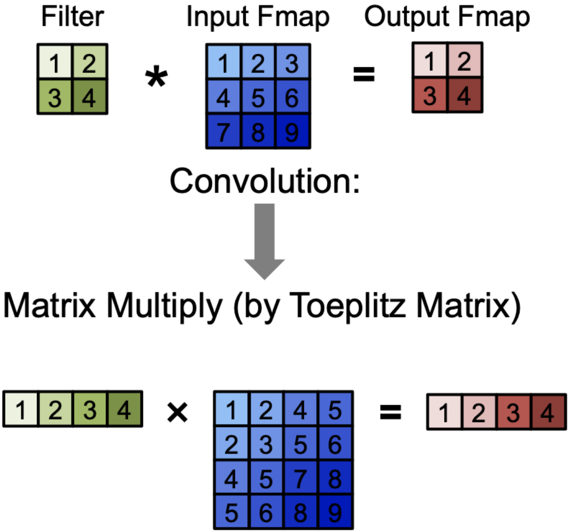
Convolution (CONV) Layer



Convolution (CONV) Layer



Convolution (CONV) Layer



Convert to matrix multiply using the **Toeplitz Matrix**

Strassen

$$\overbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}^{A} \overbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}^{B} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

8 multiplications + 4 additions

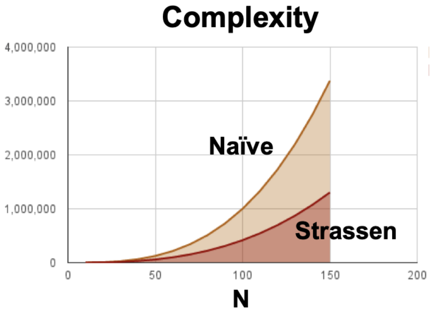
$$\begin{aligned} P1 &= a(f - h) \\ P2 &= (a + b)h \\ P3 &= (c + d)e \\ P4 &= d(g - e) \end{aligned} \qquad \begin{aligned} P5 &= (a + d)(e + h) \\ P6 &= (b - d)(g + h) \\ P7 &= (a - c)(e + f) \end{aligned}$$

$$AB = \begin{bmatrix} P5 + P4 - P2 + P6 & P1 + P2 \\ P3 + P4 & P1 + P5 - P3 - P7 \end{bmatrix}$$

7 multiplications + 18 additions

Strassen

- Reduce the complexity of matrix multiplication from $\Theta(N^3)$ to $\Theta(N^{2.807})$ by reducing multiplications



Comes at the price of reduced numerical stability and requires significantly more memory

Image Source: http://www.stoimen.com/blog/2012/11/26/computer-algorithms-strassens-matrix-multiplication/

Winograd 1D – F(2,3)

- Targeting convolutions instead of matrix multiply
- Notation: F(size of output, filter size)

$$F(2,3) = \overset{\text{input}}{\begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix}} \overset{\text{filter}}{\begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix}} = \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$$

6 multiplications + 4 additions

Winograd 1D – F(2,3)

- Targeting convolutions instead of matrix multiply
- Notation: F(size of output, filter size)

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

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

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

4 multiplications + 12 additions + 2 shifts

Winograd 2D - F(2x2, 3x3)

- 1D Winograd is nested to make 2D Winograd

Filter		Input Fmap		Output Fmap
$\begin{bmatrix} g_{00} & g_{01} & g_{02} \\ g_{10} & g_{11} & g_{12} \\ g_{20} & g_{21} & g_{22} \end{bmatrix}$	$*$	$\begin{bmatrix} d_{00} & d_{01} & d_{02} & d_{03} \\ d_{10} & d_{11} & d_{12} & d_{13} \\ d_{20} & d_{21} & d_{22} & d_{23} \\ d_{30} & d_{31} & d_{32} & d_{33} \end{bmatrix}$	$=$	$\begin{bmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \end{bmatrix}$

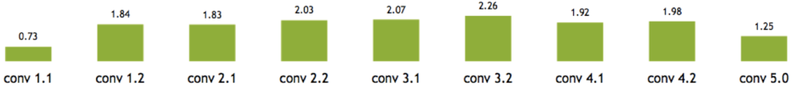
Original: 36 multiplications

Winograd: 16 multiplications → 2.25 times reduction

Winograd Performance Varies

Optimal convolution algorithm depends on convolution layer dimensions

Winograd speedup over GEMM-based convolution (VGG-E layers, N=1)



Meta-parameters (data layouts, texture memory) afford higher performance

Using texture memory for convolutions: 13% inference speedup

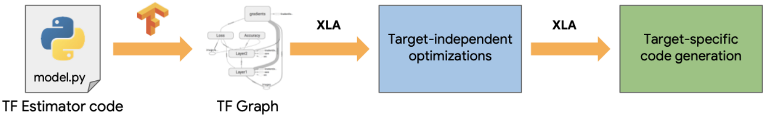
(GoogLeNet, batch size 1)

Winograd Summary

- Winograd is an optimized computation for convolutions
- It can significantly reduce multiplies
  - For example, for 3x3 filter by 2.25X
- But, each filter size (and output size) is a different computation.

V

Tensorflow XLA



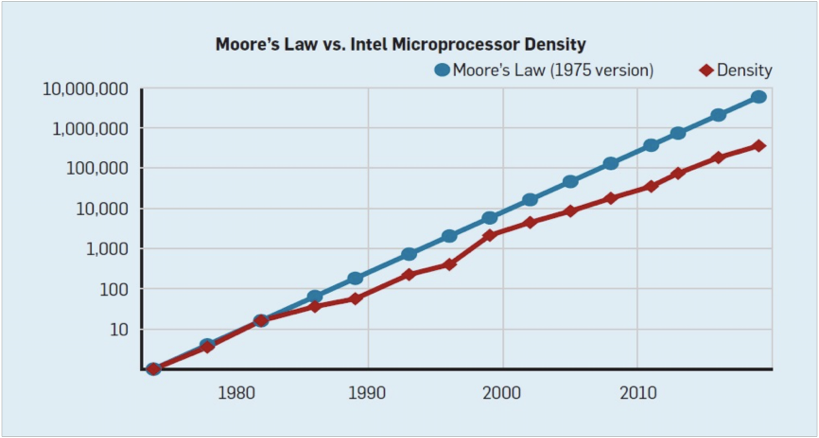
Outline of the rest of the lecture

- DNN Hardware Specialization
- DNN Accelerator Architectures
- Benchmarking
- Edge AI HW case studies:
  - Mobile
  - Embedded devices
  - Autonomous vehicles

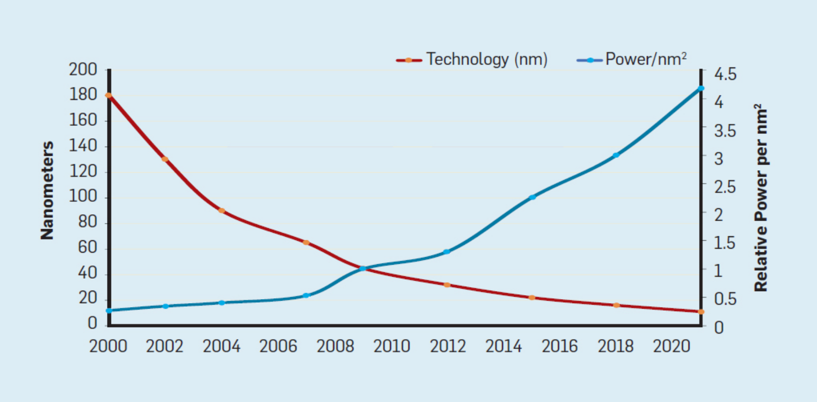
Section 2

HW Specialization

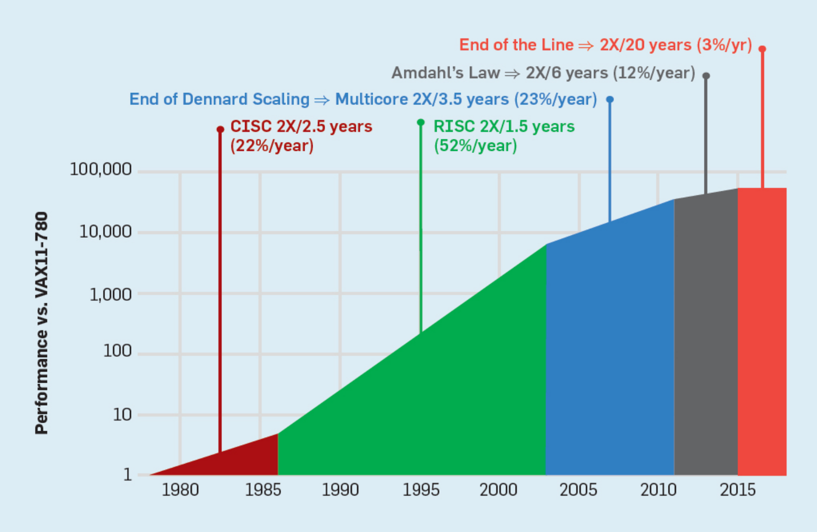
End of Moore Law



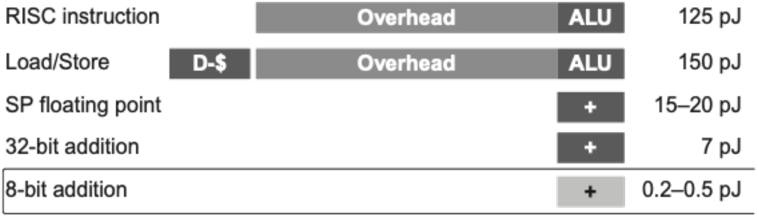
End of Dennard' scaling



Stagnation of performance



HW specialization is required



Domain-Specific Architectures

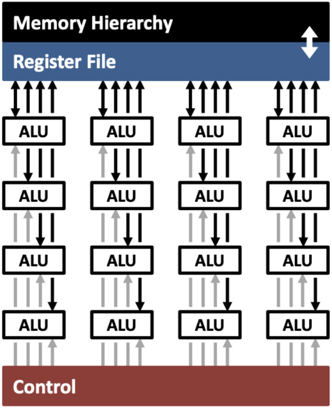
- **DSA Guidelines:**
  1. **Dedicated memories:** Use dedicated memories to minimize the distance over which data is moved.
  2. **Larger arithmetic unit:** Invest the resources saved from dropping advanced microarchitectural optimizations into more arithmetic units or bigger memories.
  3. **Easy parallelism:** Use the easiest form of parallelism that matches the domain.
  4. **Smaller data size:** Reduce data size and type to the simplest needed for the domain.
  5. **Domain-specific language:** Use a domain-specific programming language to port code to the DSA.

Section 3

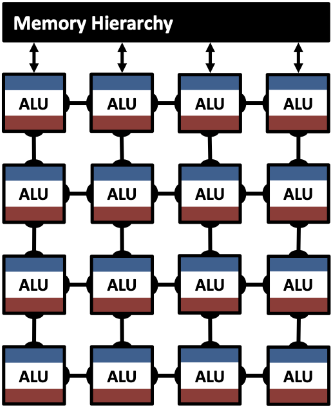
Accelerator Architectures

Highly-Parallel Compute Paradigms

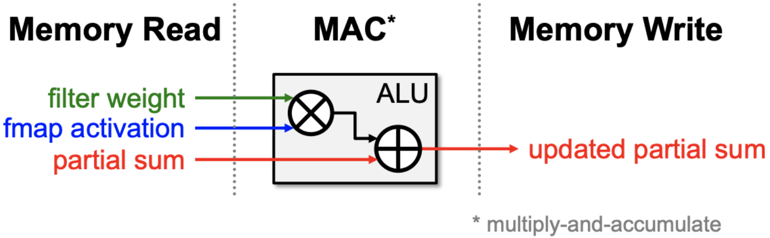
Temporal Architecture (SIMD/SIMT)



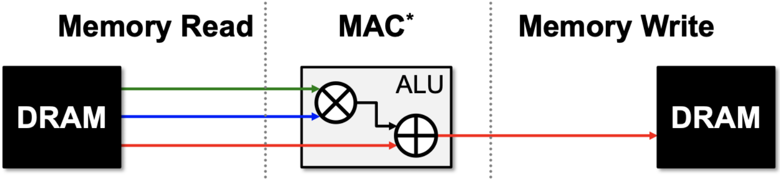
Spatial Architecture (Dataflow Processing)



Memory Access is the Bottleneck



Memory Access is the Bottleneck

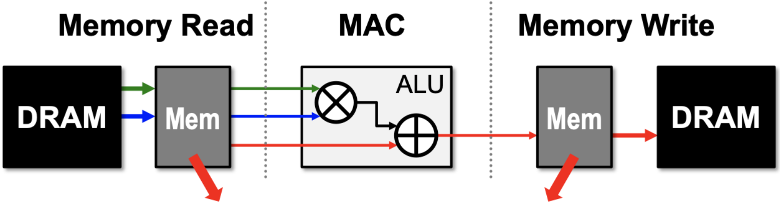


* multiply-and-accumulate

Worst Case: all memory R/W are **DRAM** accesses

- Example: AlexNet [NIPS 2012] has **724M** MACs
  → **2896M** DRAM accesses required

Leverage Local Memory for Data Reuse

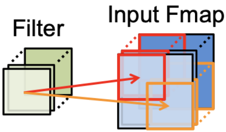


Extra levels of local memory hierarchy
Smaller, but **Faster** and **more Energy-Efficient**

Types of Data Reuse in DNN

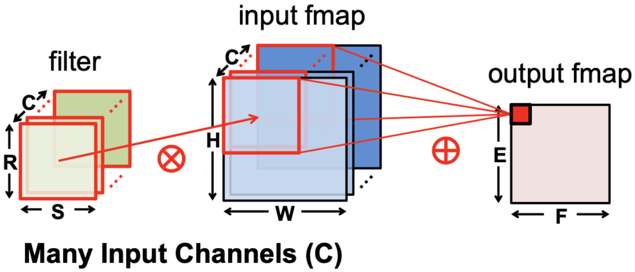
Convolutional Reuse

CONV layers only
(sliding window)



Reuse: Activations
Filter weights

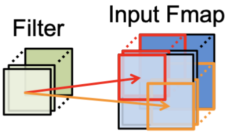
Convolution (CONV) Layer



Types of Data Reuse in DNN

Convolutional Reuse

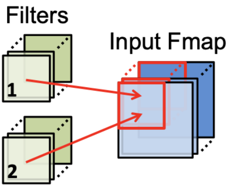
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

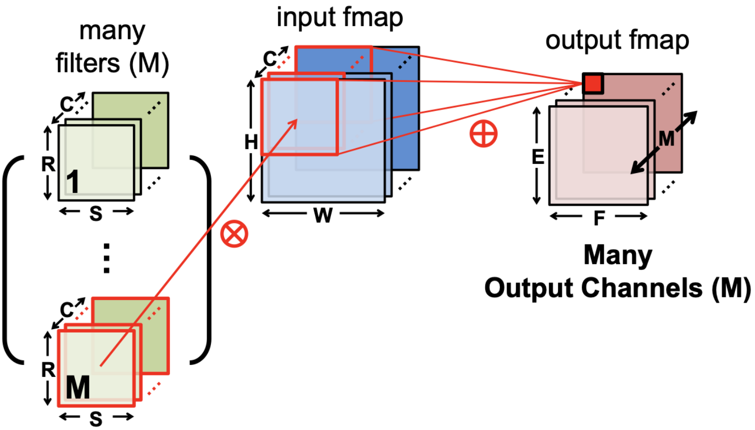
Fmap Reuse

CONV and FC layers



Reuse: Activations

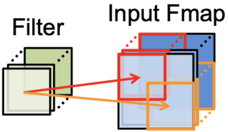
Convolution (CONV) Layer



Types of Data Reuse in DNN

Convolutional Reuse

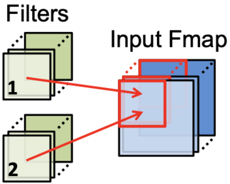
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

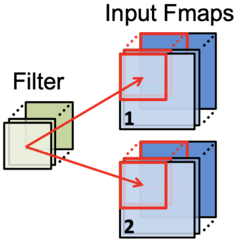
CONV and FC layers



Reuse: Activations

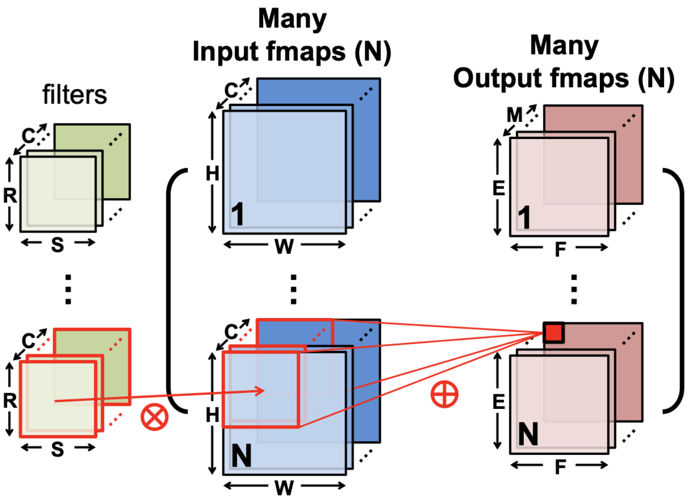
Filter Reuse

CONV and FC layers
(batch size > 1)



Reuse: Filter weights

Convolution (CONV) Layer



Types of Data Reuse in DNN

Convolutional Reuse

CONV layers only
(sliding window)

Reuse: Activations
Filter weights

Fmap Reuse

CONV and FC layers

Filters

Reuse: Activations

Filter Reuse

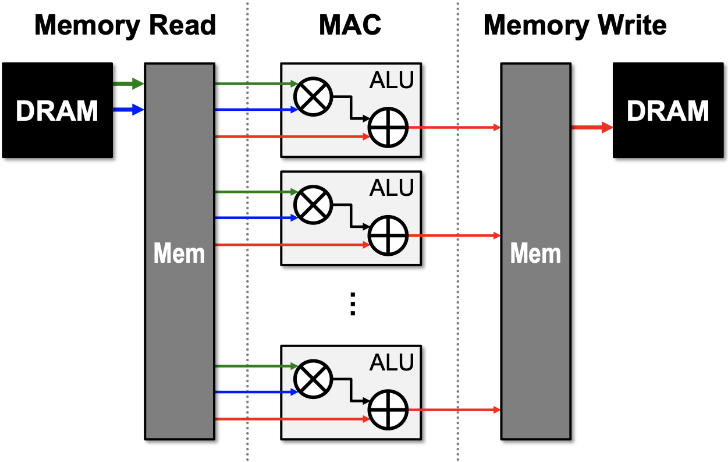
CONV and FC layers
(batch size > 1)

Input Fmaps

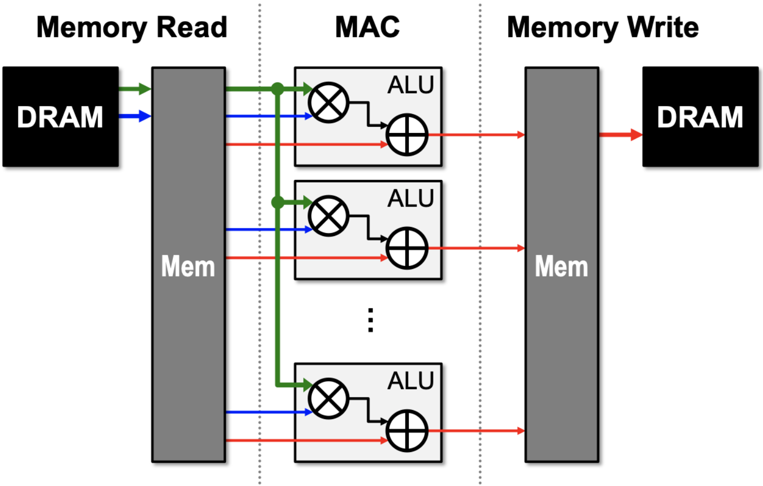
Reuse: Filter weights

If all data reuse is exploited, DRAM accesses in AlexNet can be reduced from **2896M** to **61M** (best case)

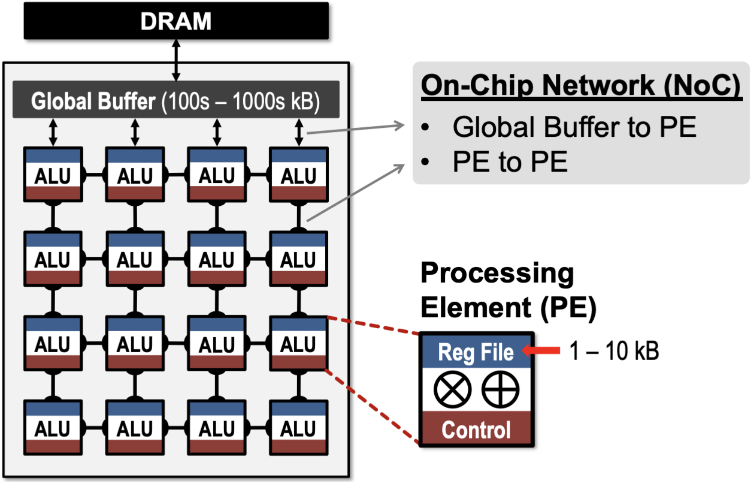
Leverage Parallelism for Higher Performance



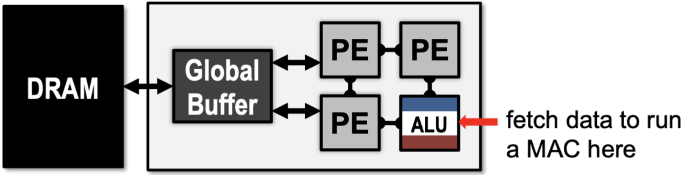
Leverage Parallelism for Spatial Data Reuse

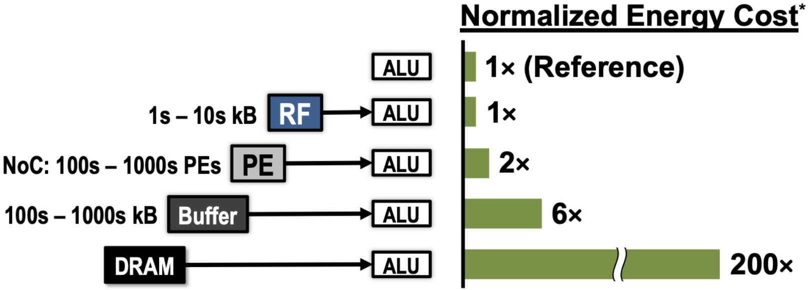


Spatial Architecture for DNN



Multi-Level Low-Cost Data Access

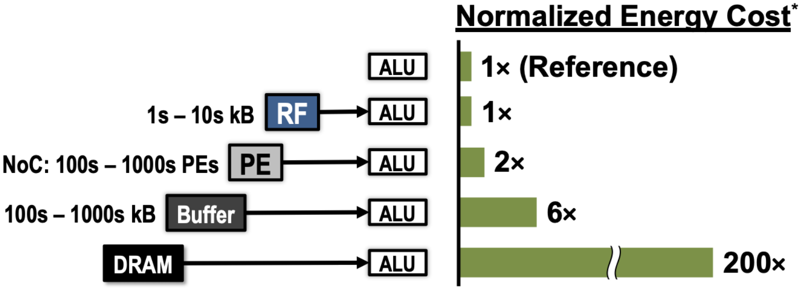




* measured from a commercial 65nm process

Multi-Level Low-Cost Data Access

A **Dataflow** is required to maximally exploit **data reuse** with the **low-cost memory hierarchy** and **parallelism**

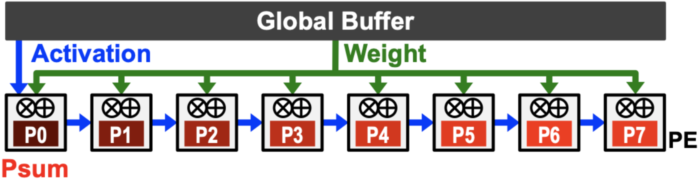


* measured from a commercial 65nm process

Dataflow Taxonomy

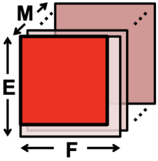
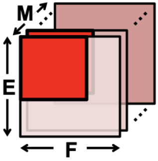
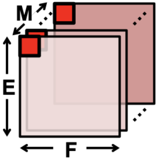
- **Output Stationary (OS)**
- **Weight Stationary (WS)**
- **Input Stationary (IS)**

Output Stationary (OS)

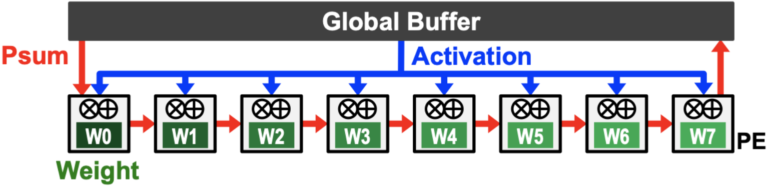


- **Minimize partial sum** R/W energy consumption
  - maximize local accumulation
- **Broadcast/Multicast filter weights** and **reuse activations spatially** across the PE array

Variants of Output Stationary

	OS_A	OS_B	OS_C
Parallel Output Region			
# Output Channels	Single	Multiple	Multiple
# Output Activations	Multiple	Multiple	Single
Notes	Targeting **CONV** layers		Targeting **FC** layers

Weight Stationary (WS)



- **Minimize** **weight** read energy consumption
  - maximize convolutional and filter reuse of weights
- **Broadcast** **activations** and **accumulate** **psums** **spatially** across the PE array.

WS Example: NVDLA (simplified)

Released Sept 29, 2017

Global Buffer

PE Array: *M x C MACs*

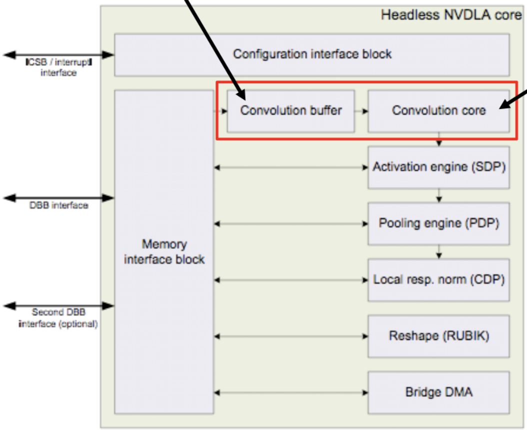
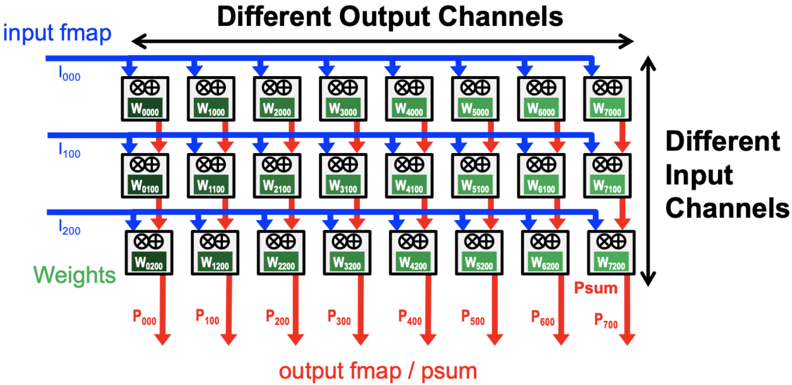


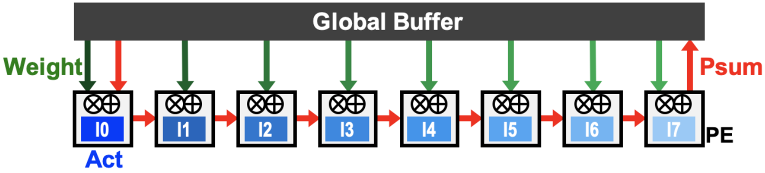
Image Source: Nvidia

http://nvdla.org

WS Example: NVDLA (simplified)



Input Stationary (IS)



- **Minimize activation** read energy consumption
  - maximize convolutional and fmap reuse of activations
- **Unicast weights** and **accumulate psums spatially** across the PE array.

Summary of DNN Dataflows

- Minimizing **data movement** is the key to achieving high **energy efficiency** for DNN accelerators
- Dataflow taxonomy:
  - **Output Stationary**: minimize movement of psums
  - **Weight Stationary**: minimize movement of weights
  - **Input Stationary**: minimize movement of inputs
- **Loop nest** provides a compact way to describe various properties of a dataflow, e.g., data tiling in multi-level storage and spatial processing.

Section 4

Benchmarking

Metrics for DNN Hardware

- **Accuracy**
  - Quality of result for a given task
- **Throughput**
  - Analytics on high volume data
  - Real-time performance (e.g., video at 30 fps)
- **Latency**
  - For interactive applications (e.g., autonomous navigation)
- **Energy and Power**
  - Edge and embedded devices have limited battery capacity
  - Data centers have stringent power ceilings due to cooling costs
- **Hardware Cost**
  - $$$

Metrics for DNN Hardware

- **Accuracy**
  - Difficulty of dataset and/or task should be considered
- **Throughput**
  - Number of cores (include utilization along with peak performance)
  - Runtime for running specific DNN models
- **Latency**
  - Include batch size used in evaluation
- **Energy and Power**
  - Power consumption for running specific DNN models
  - Include external memory access
- **Hardware Cost**
  - On-chip storage, number of cores, chip area + process technology

Comprehensive Coverage

- **All metrics** should be reported for fair evaluation of design tradeoffs
- Examples of what can happen if certain metric is omitted:
  - **Without the accuracy given for a specific dataset and task**, one could run a simple DNN and claim low power, high throughput, and low cost – however, the processor might not be usable for a meaningful task
  - **Without reporting the off-chip bandwidth**, one could build a processor with only multipliers and claim low cost, high throughput, high accuracy, and low chip power – however, when evaluating system power, the off-chip memory access would be substantial

Evaluation Process

The evaluation process for whether a DNN system is a viable solution for a given application might go as follows:

1. **Accuracy** determines if it can perform the given task
2. **Latency and throughput** determine if it can run fast enough and in real-time
3. **Energy and power consumption** will primarily dictate the form factor of the device where the processing can operate
4. **Cost**, which is primarily dictated by the chip area, determines how much one would pay for this solution

MLCommons

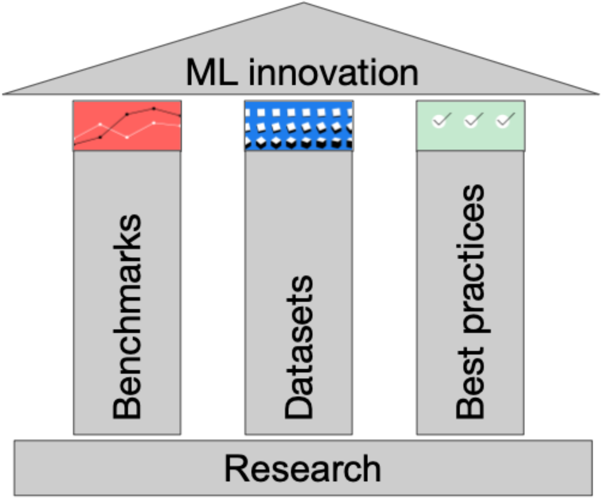
- What is MLCommons?
  - A global community (industry & academia) born from MLPerf benchmark effort

Founding Members

Alibaba Group 阿里巴巴集团	AMD	arm	Bai du 百度	CenTaur technology	cerebras
cisco	DELL EMC	d-MATRIX Intelligence Delivered	Enflame	FACEBOOK AI	FUJITSU
FURIOSA	GIGABYTE	Google	GML GrAI Matter Labs	GRAPHCORE	groq
地平线 Horizon Robotics	Hewlett Packard Enterprise	inspur	intel AI	KALRAY	LANDING AI
MEDIATEK	Microsoft	Myrtle.ai	Nettrix 宁畅	NVIDIA	oppo

ML Commons Mission

- Better ML for Everyone

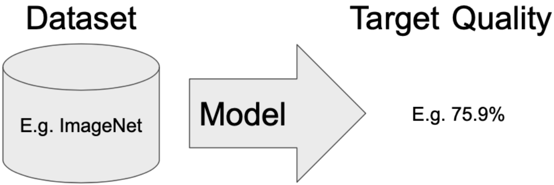


MLPerf

- What is MLPerf?
    - ML performance benchmarking effort with wide industry and academic support
    - Several benchmark suites for different targets:
        - Training
        - Training HPC
        - Inference: Datacenter
        - Inference: Mobile
        - Inference: Tiny

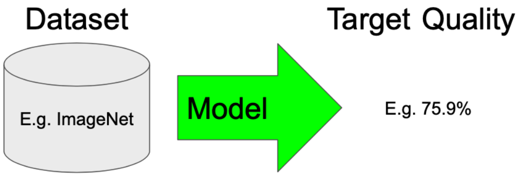
MLPerf Training

MLPerf Training benchmark definition



MLPerf Training - divisions

Two divisions with different model restrictions



Closed division: specific model e.g. ResNet v1.5 → direct comparisons

Open division: any model → innovation

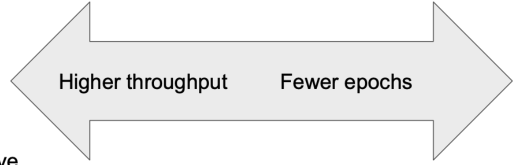
MLPerf Training - Metrics

Metric: time-to-train

Alternative is throughput
- Easy / cheap to measure

But can increase throughput
at cost of total time to train!

Time-to-train (end-to-end)
- Time to solution!
- Computationally expensive
- High variance
- **Least bad choice**



Lower precision
Higher batch size

Higher precision
Lower batch size

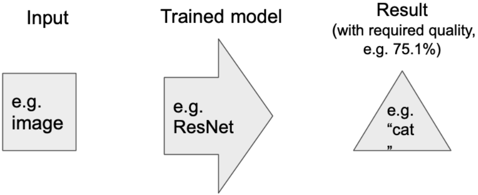
MLPerf Training - Workloads

MLPerf v1.0 Training Workloads

Use Case	Neural Network
Vision	ResNet-50 v1.5
	SSD ResNet-34
	Mask R-CNN
	3D UNET
Speech	RNN-T
Language	BERT Large
Commerce	DLRM
Research	Mini-Go

MLPerf Inference

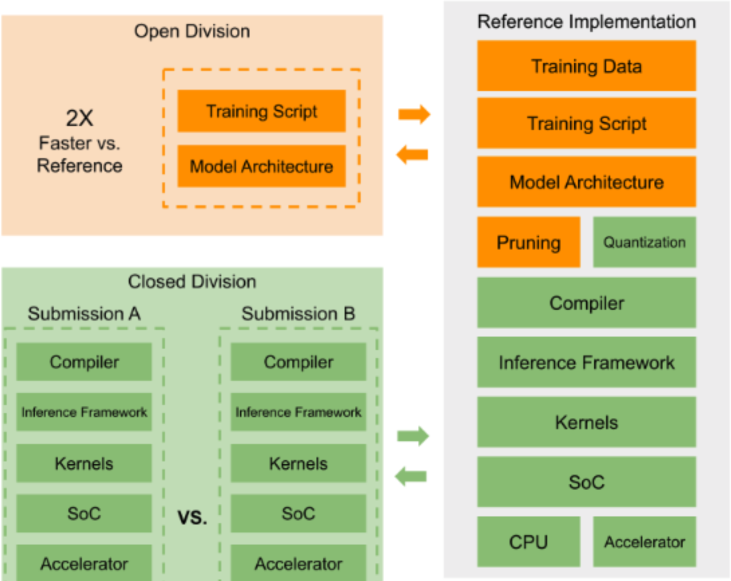
MLPerf inference definition



Submission division	Closed	Open
Inference	Strict rules Apples-to-apples ML system comparison	Permissive rules Better models than reference

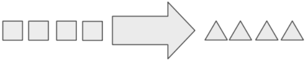
MLPerf benchmarking scope: ML systems (HW + SW)

MLPerf Inference - divisions

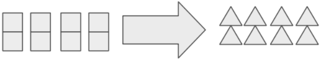


MLPerf Inference - Scenarios

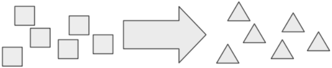
Four **scenarios** to handle different use cases



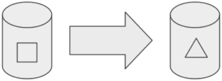
Single stream
(e.g. cell phone augmented vision)



Multiple stream
(e.g. multiple camera driving assistance)



Server
(e.g. translation app)



Offline
(e.g. photo sorting app)

MLPerf Inference - Workloads

MLPerf Inference v1.0 Workloads

Datacenter / Edge Inference

Use Case	Reference Network
Image Classifier	ResNet-50 v1.5
Object detector (large)	SSD ResNet-34
Object detector(small)	SSD MobileNet v1 (edge only)
3D medical imaging	3D UNET
Speech-to-text	RNN-T
NLP / Q&A	BERT Large
Recommendation	DLRM (datacenter only)

Data Center: Offline and Server scenario
Edge: Single Stream, Offline, (deprecating Multi-Stream)

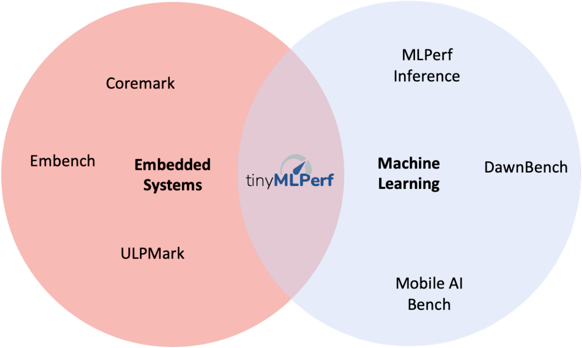
Mobile Inference

Use Case	Reference Network
Image Classifier	MobileNetEdge
Object Detector	MobileDet
Image Segmentation	DeepLab v3
NLP / Q&A	Mobile-BERT

Mobile: Single Stream, and Offline scenario

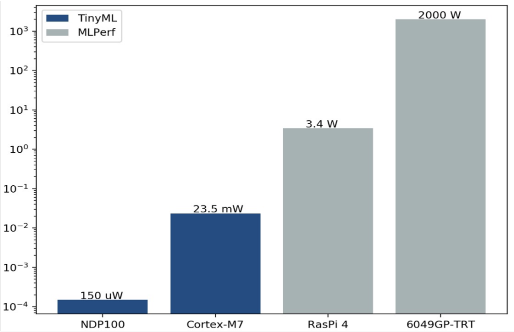
TinyMLPerf

Filling the Need



TinyMLPerf - Challenges

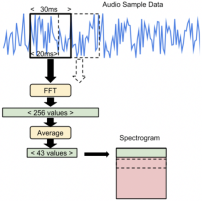
- Constrained Device
- No OS or Standard Libraries
- Heterogeneity
- Nascent Field



TinyMLPerf - Benchmarks

Four Benchmarks

Keyword Spotting



Warden, Pete. "Speech commands: A dataset for limited-vocabulary speech recognition." *arXiv preprint arXiv:1804.03209* (2018).

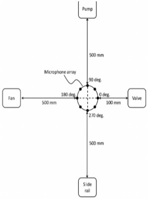
Visual Wake Words

(a) 'Person'

(b) 'Not-person'

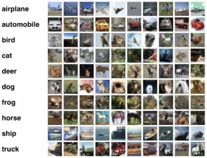
Chowdhery, Aakanksha, et al. "Visual wake words dataset." *arXiv preprint arXiv:1906.05721* (2019).

Anomaly Detection



Purohit, Harsh, et al. "MIMII dataset: Sound dataset for malfunctioning industrial machine investigation and inspection." *arXiv preprint arXiv:1909.09347* (2019).

Tiny Image Classification



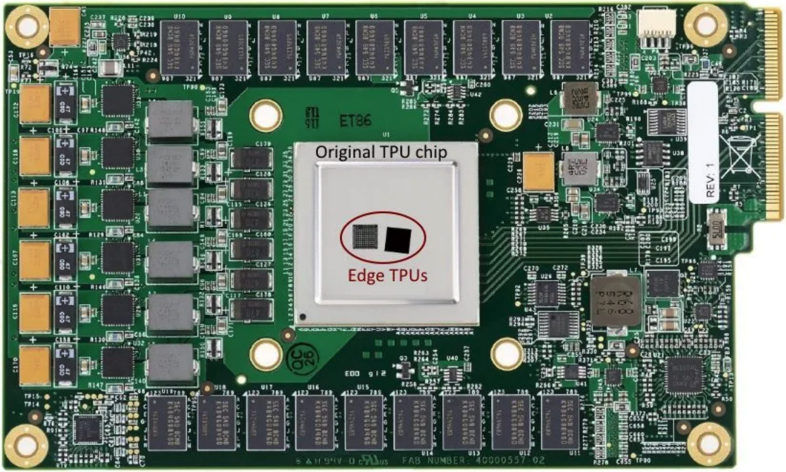
Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009): 7.

Section 5

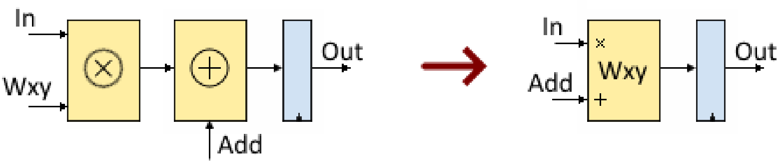
EAI case studies

Google Edge TPU

- Weight-stationary systolic architecture from Google
- Edge TPU smaller version than original Cloud TPU

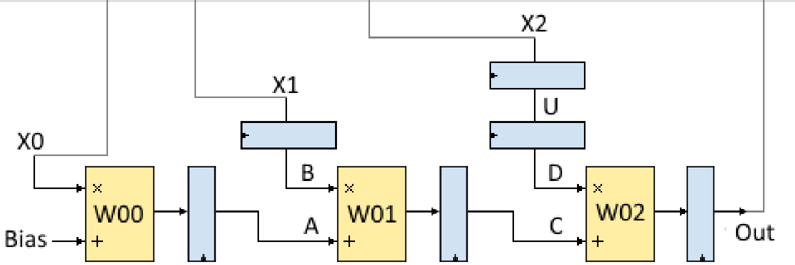


Google TPU - mull-add cell



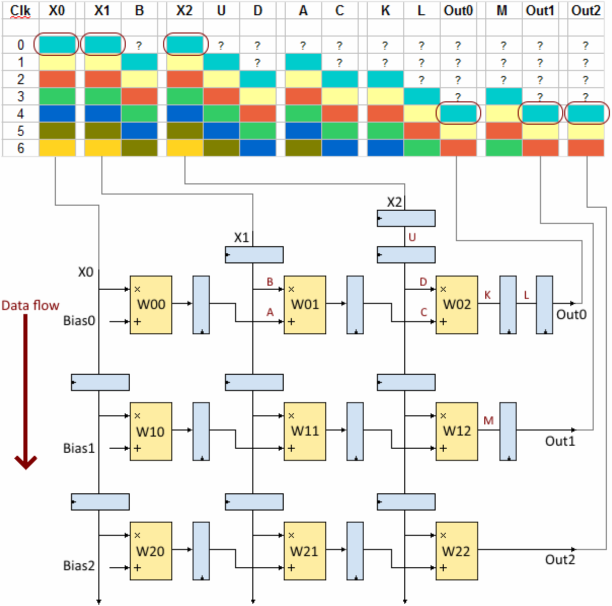
Google TPU - Three input neuron

Clk	X0	X1	B	X2	U	D	A	C	Out
0			?		?	?	?	?	?
1						?		?	?
2									?
3									
4									
5									
6									

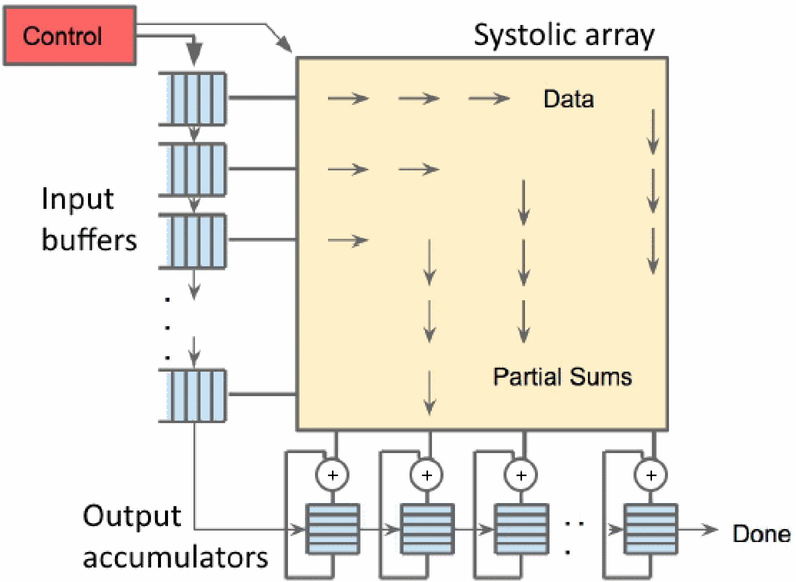


Out = X0W00 + X1W01 + X2W02 + Bias

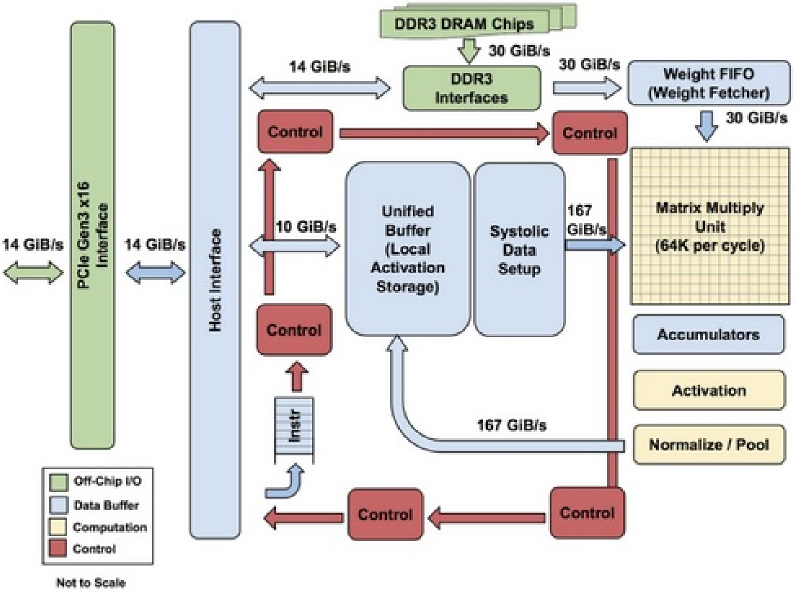
Google TPU - systolic array



Google TPU - systolic array (Google schematic)



Google Cloud TPU - architecture



Google Cloud TPU - architecture

- Main features
  - 4 TOPS (Tera Operations per Second)
  - 2 TOPS/W
  - INT8 ops
    - Requires quantization
    - Model compatibility issues
  - Some Benchmarks results

Other case studies (next lecture)

- Mobile
  - Apple Neural Engine - Group 6
  - Some apple competitors (Huawei, Samsung, . . .) - Group 5
- Embedded devices
  - ARM AI products with emphasis on Ethos NPUs - Group 4
  - Some Chinese accelerator - Group 3
- Autonomous vehicles
  - NVIDIA boards & accelerators (DLA, GPU, . . .) - Group 2
  - Tesla FSD - Group1

Work to do

- Live or recorded presentation:
  - Main features
  - Architecture
  - How the architecture follow the guidelines for DSA
  - Some metrics (performance, power, etc.)
  - Products were the accelerator is employed
- Short document (2-3 pages) summarizing this information and giving relevant references.