Lecture 09: CNN Dataflow & Hardware Accelerators

Recap

- Federated Learning
- Distributed DNN Training
- Distributed DNN Inference
- Speculative Decoding

Notes

- Project Proposal due Thursday
  - Signup your team
- Mid-semester Course Feedback
- Midterm Exam Review

Topics

- Hardware accelerator: Overview
- Convolutional operation conversion
- Hardware architecture of CNN accelerator
- Systolic array
- Popular accelerator design
  - Eyeriss
  - Diannao
  - Cnvlutin
  - EIE

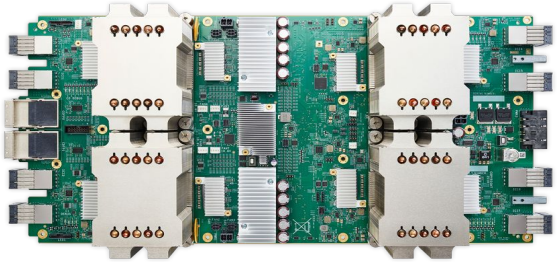
Hardware Support for DNN

- GPU is better than CPU in terms of throughput for both Neural Network training and inference.
  - GPU leverages the highly parallelized architecture of its computing units to handle computational intensive operations.
  - GPU has 10x-20x higher throughput than CPU.
- However, GPU:
  - General purpose.
  - Power consumption and latency is high.
  - Does not support sophisticated pruning and quantization algorithms.

NYU SAI LAB

Hardware Support for DNN

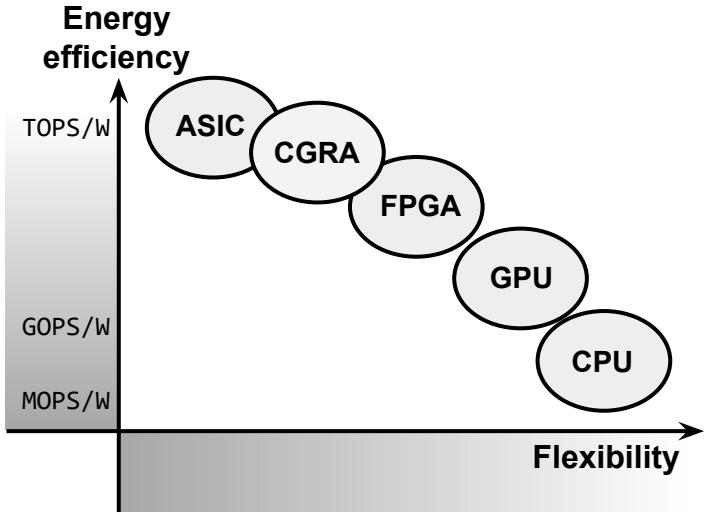
- ASIC-based implementations have been recently explored to accelerate the DNN inference.
  - Google's TPU, Apple's Neural Engine, Cerebras AI chip, …
- FPGA-based accelerators for DNN inference have been recently developed.
  - Has good programmability and flexibility
  - Short development cycles
  - Can be used as a benchmark before implementing on ASIC



Tensor Processing Unit (Google)

Alveo Accelerator Card (Xilinx)

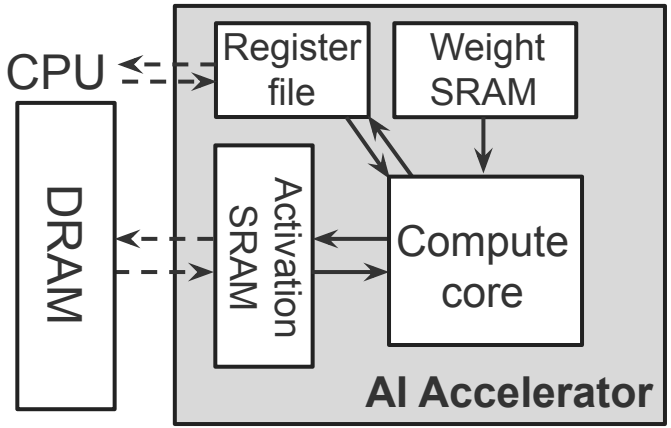
Flexibility & Performance

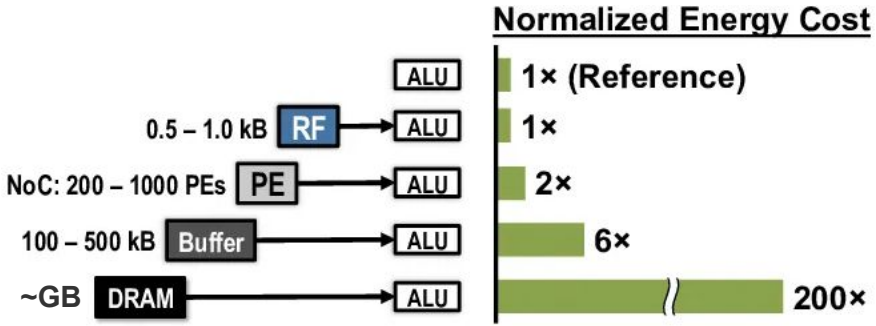


- ASIC offers the highest energy efficiency but is only suitable for specific applications.
- The CPU is a general-purpose processor but has the lowest energy efficiency.

AI Accelerator

- Making any chip is a costly, difficult and lengthy process typically done by teams of 10 to 1000's of people depending on the size and complexity of the chip.

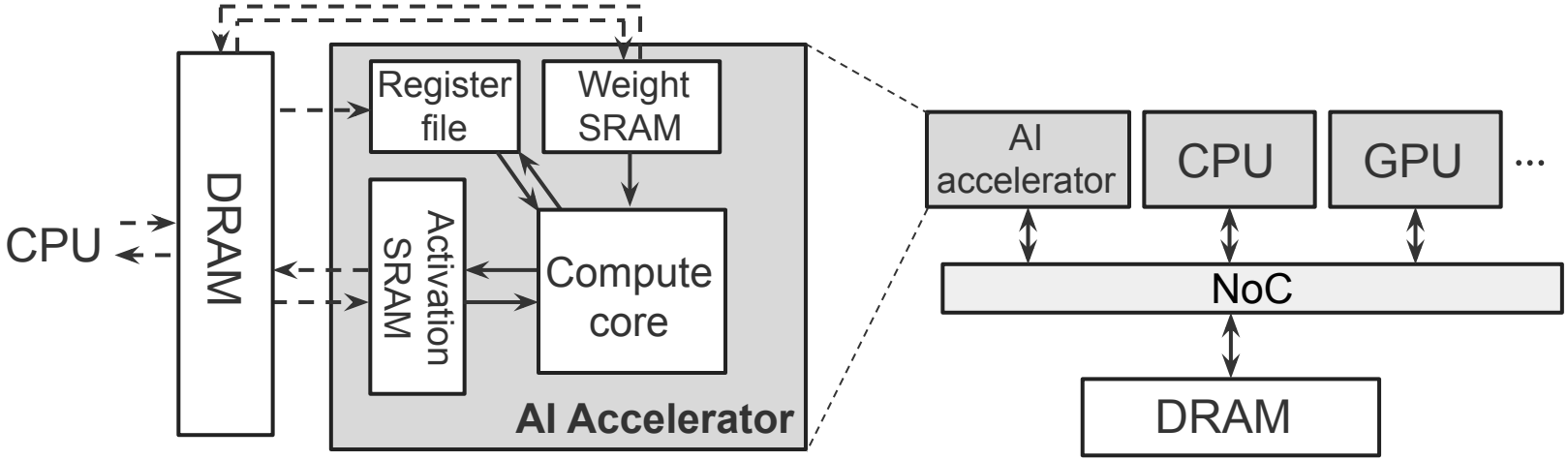




NYU SAI LAB

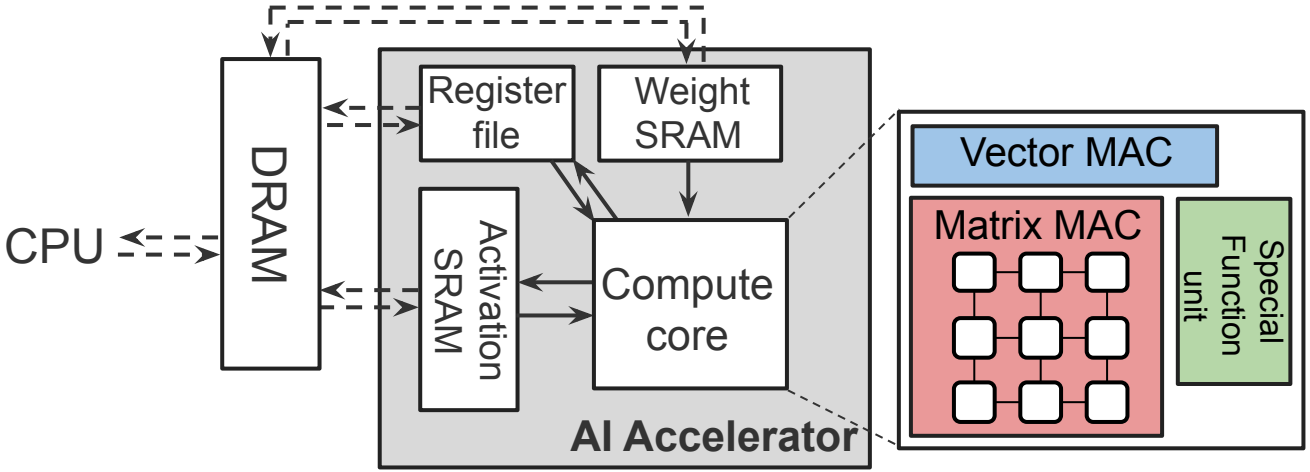
AI Accelerator

- The AI accelerator can execute part of the machine code that is related to the AI workload.



NYU SAI LAB

AI Accelerator



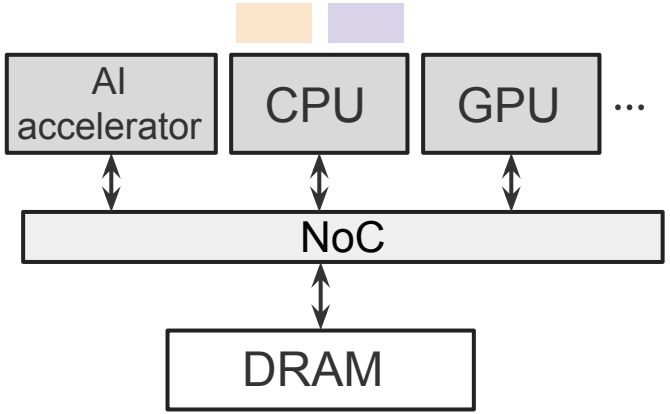
- The compute core consists of Multiply and accumulator (MAC) engine for 2D matrix multiplication.
- It also contains vector multiplier MAC as well as special function unit.

AI Accelerator

```
import torch
import torch.nn as nn
input = torch.randn(1, 1, 5, 5)
conv = nn.Conv2d(1, 1, kernel_size=3)
output = conv(input)
print(output)
```

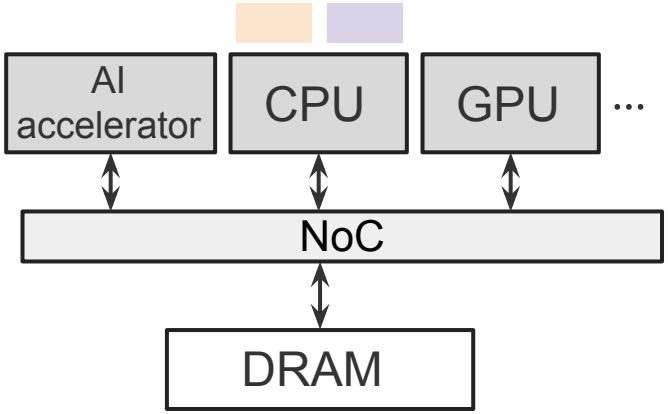
Compiler

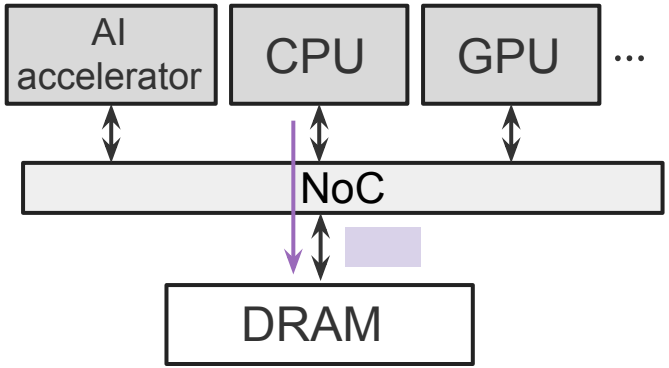
```
01001000 10001001 11011000
01001000 10000011 11000000 00001010
01001000 10000011 11101011 00000101
01001000 00111001 11011000
01110100 00000101
01001000 10001001 11000001
01001000 01101001 11001001 00000010
01001000 10001001 00001111
11101011 00000011
```



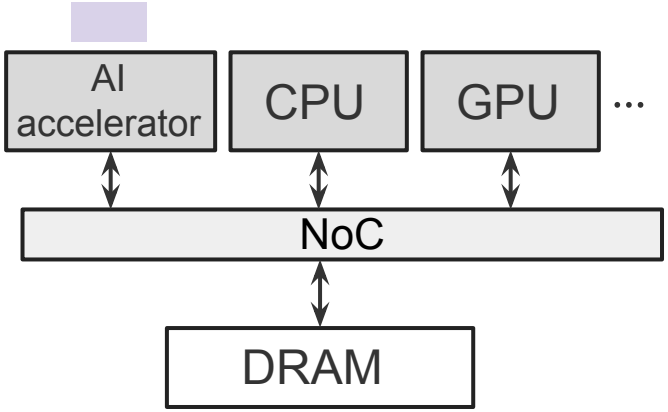
NYU SAI LAB

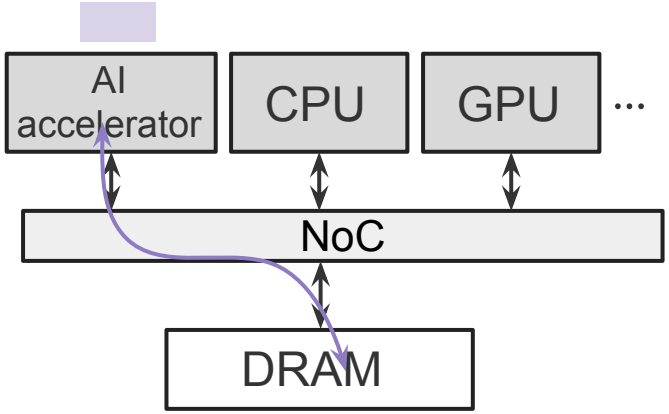
AI Accelerator



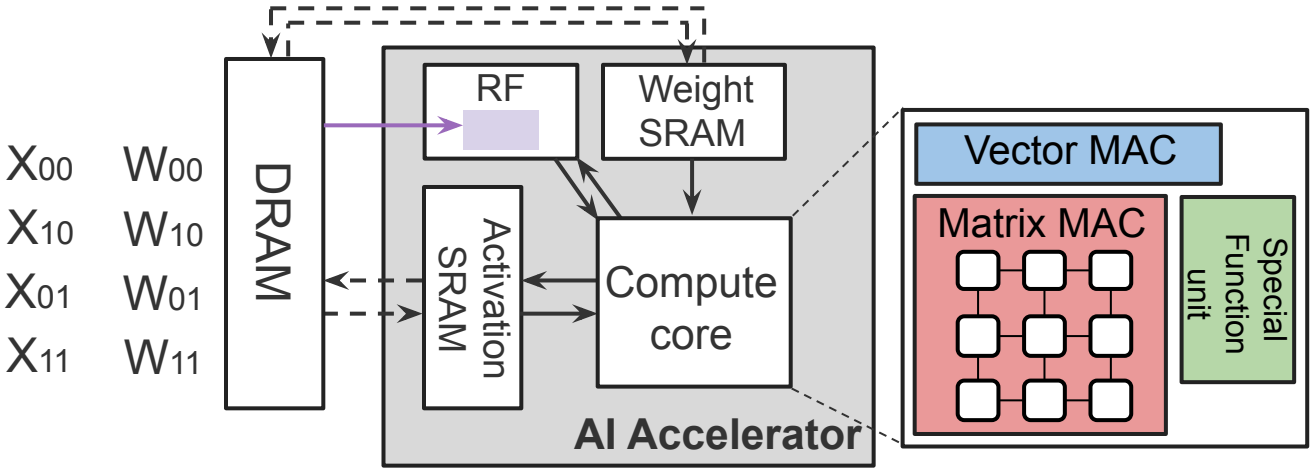


AI Accelerator



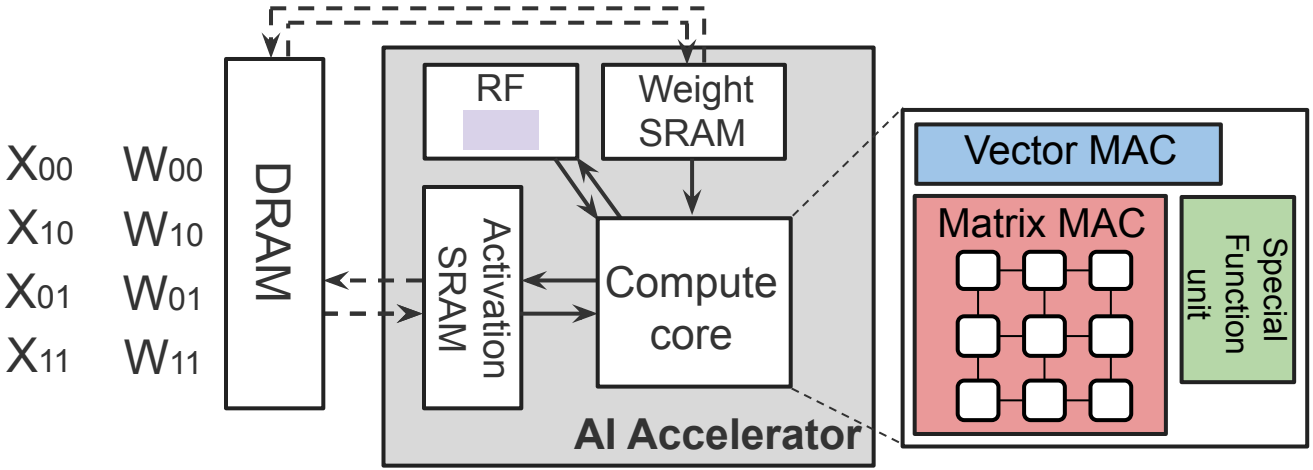


AI Accelerator



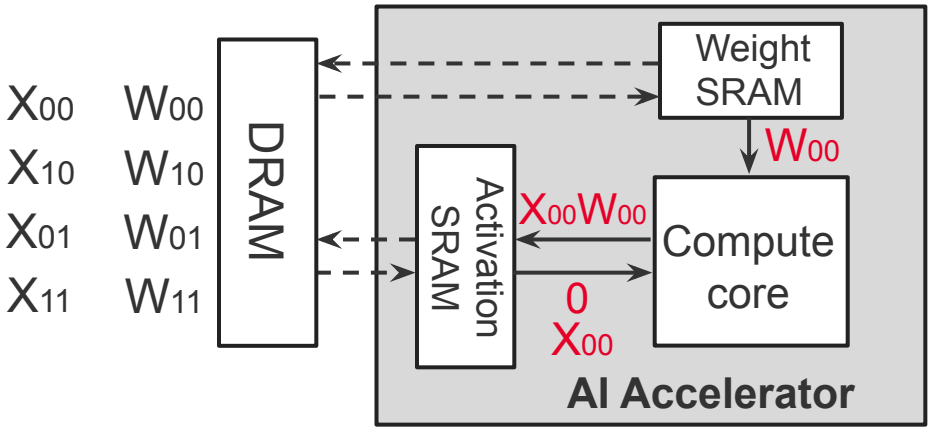
$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

AI Accelerator



$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

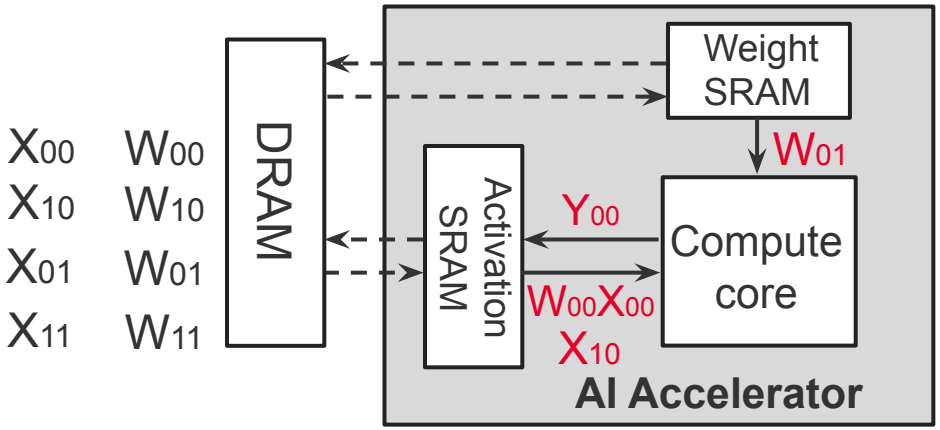
AI Accelerator



$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

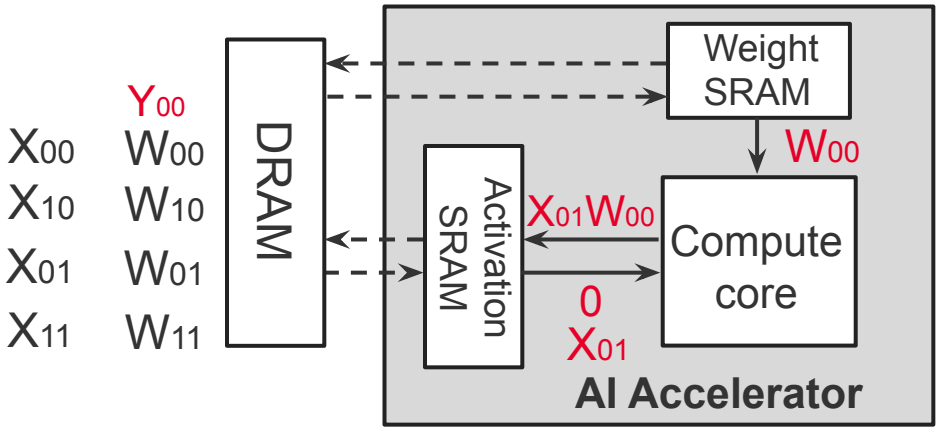
NYU SAI LAB

AI Accelerator



$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

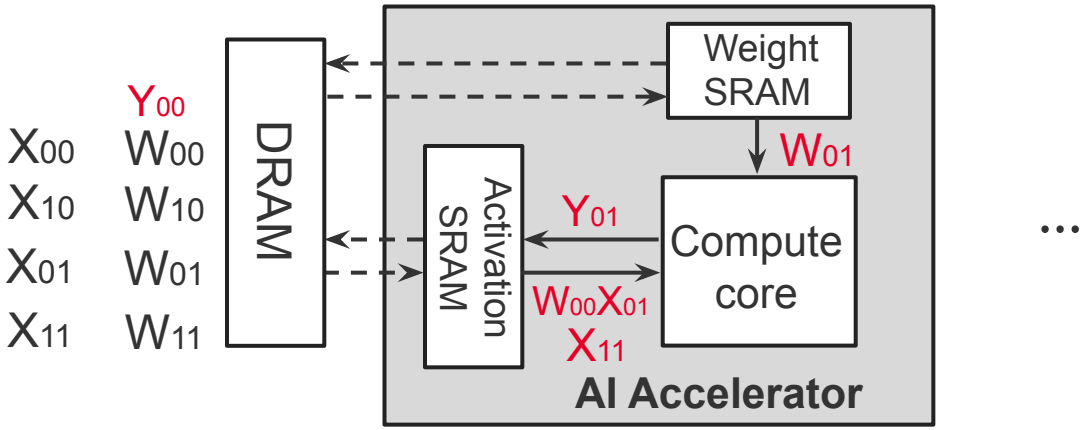
AI Accelerator



$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

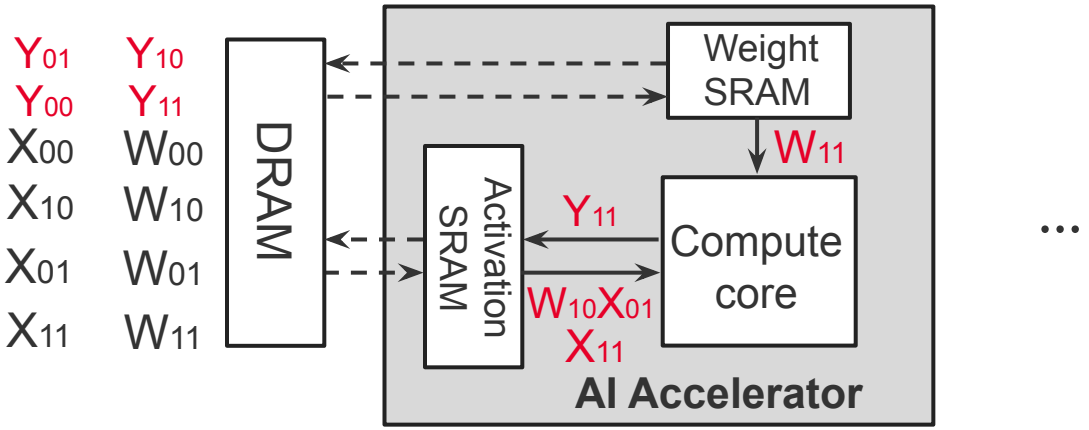
NYU SAI LAB

AI Accelerator



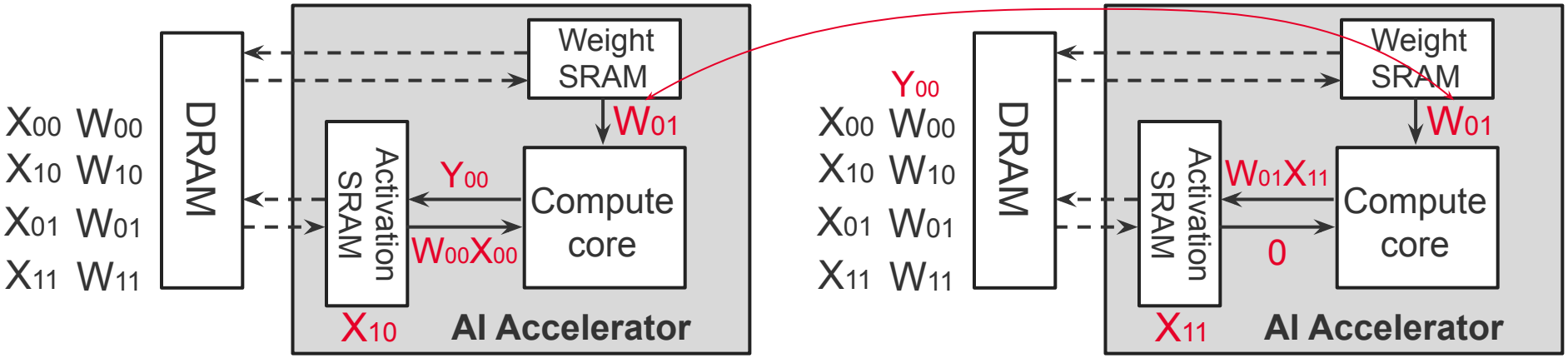
$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

AI Accelerator



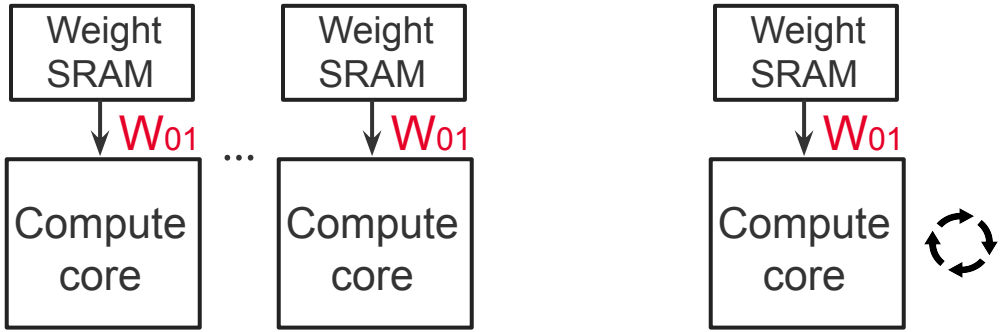
$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

Memory Access Reduction



- The computation and memory access pattern can be changed to minimize the computational cost without impacting the results.

Memory Access Reduction

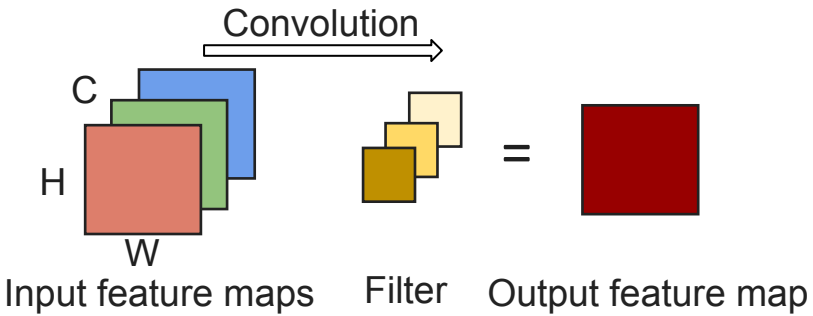


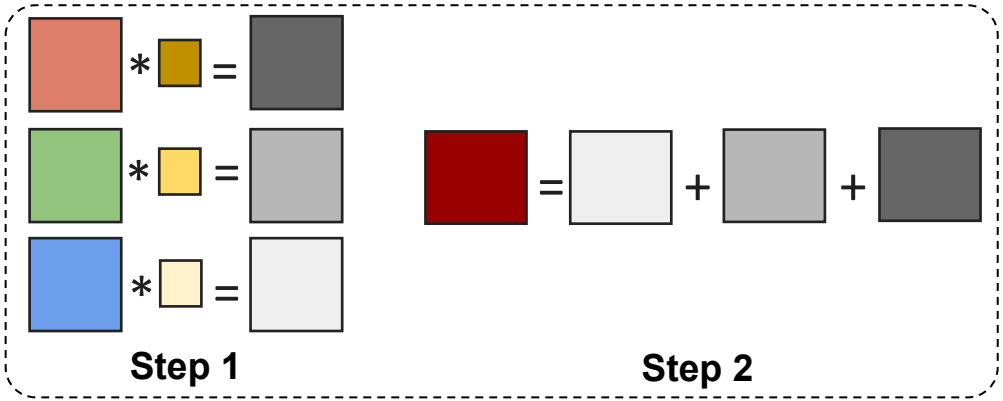
- It is preferable to minimize memory access by maximizing the reuse of loaded data.

Topics

- Hardware accelerator: Overview
- Convolutional operation conversion
- Hardware architecture of CNN accelerator
- Systolic array
- Popular accelerator design
  - Eyeriss
  - Diannao
  - Cnvlutin
  - EIE

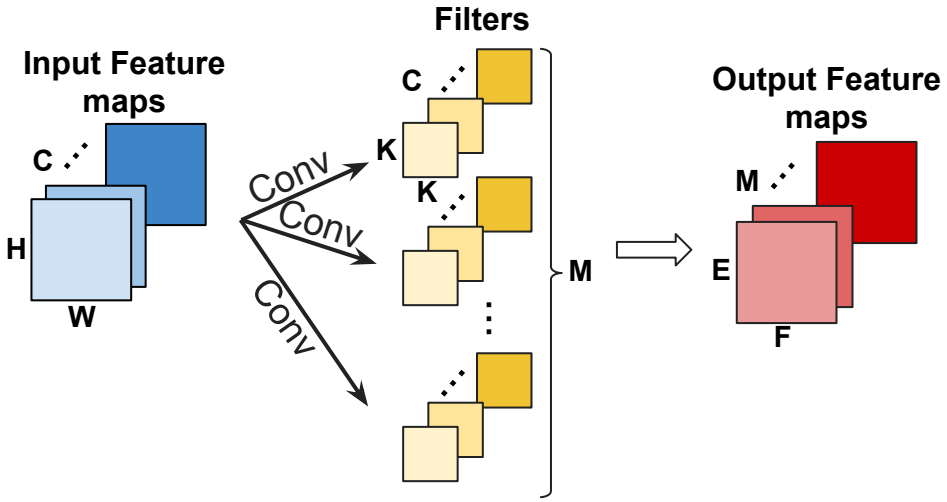
Convolutional Layers





- Core building block of a CNN, it is also the most computational intensive layer.

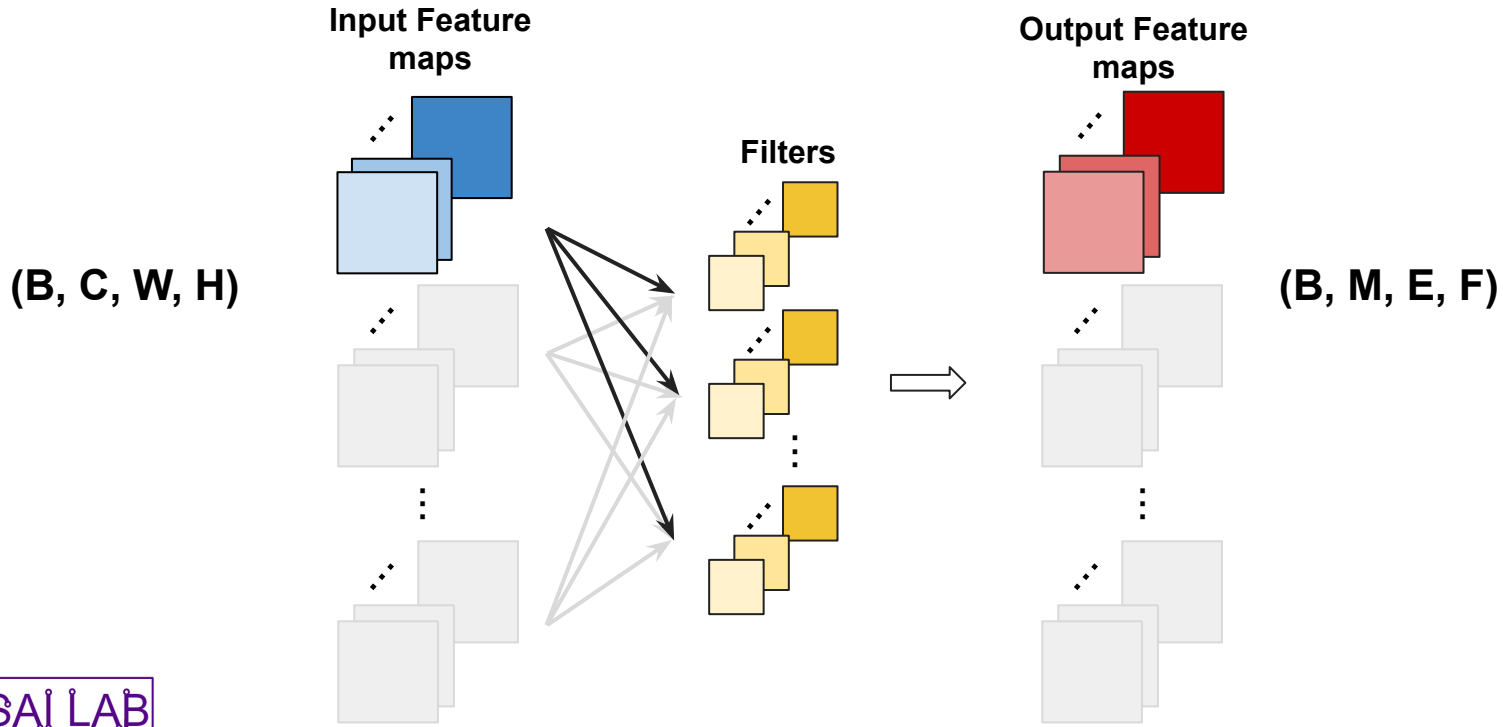
Convolution



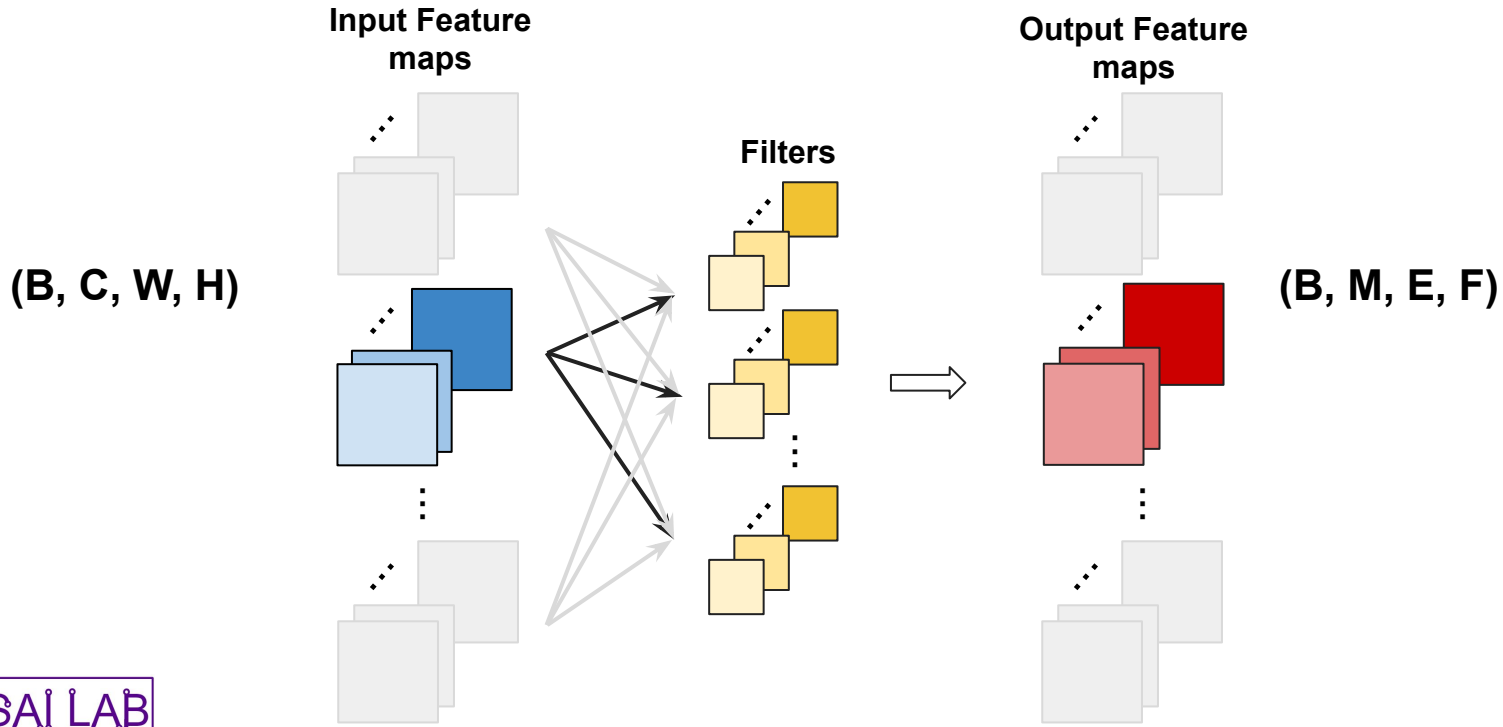
- Number of MACs: M×K×K×C×E×F
- Storage cost: 32×(M×C×K×K+C×H×W+M×E×F)

C: number of input channels
H,W: size of the input feature maps
M: number of weight filters
K: weight kernel size
E,F: size of the output feature maps

Convolution

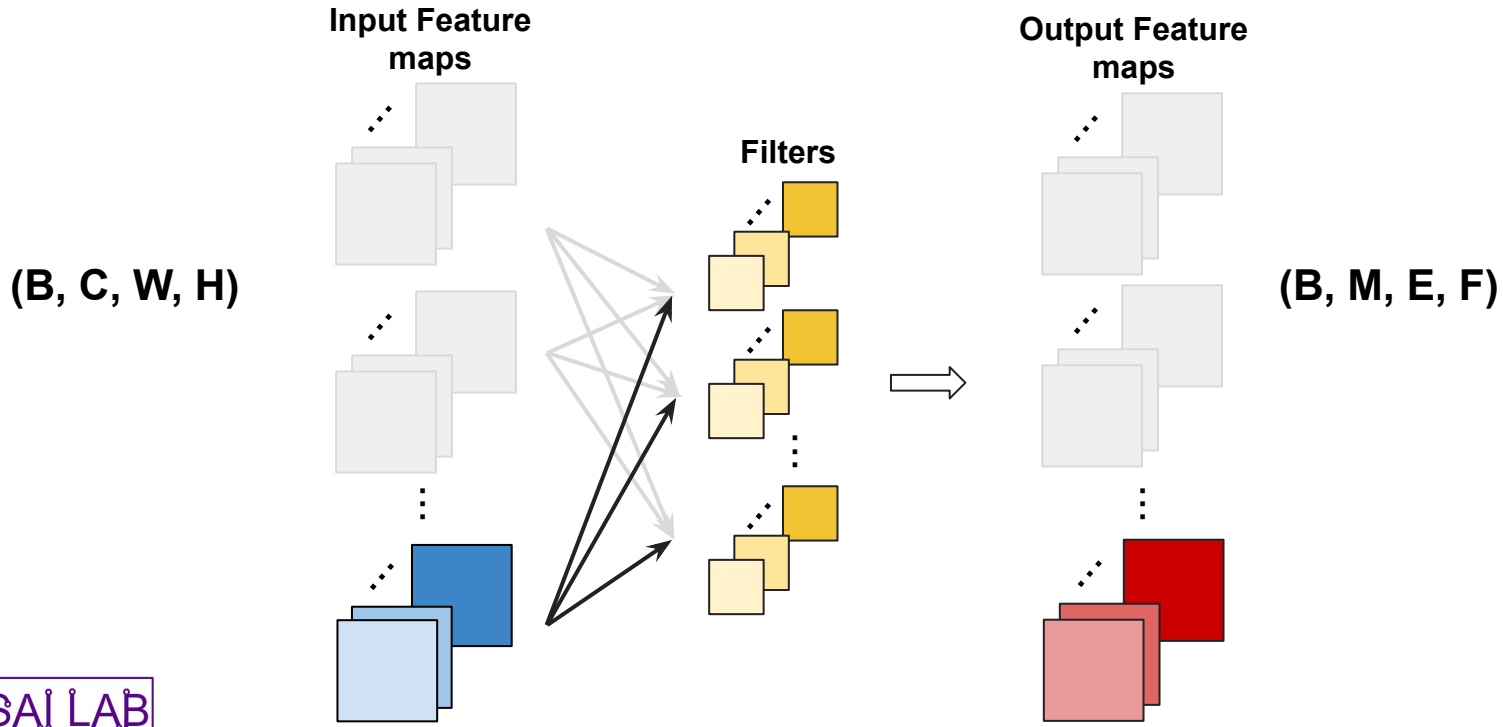


Convolution

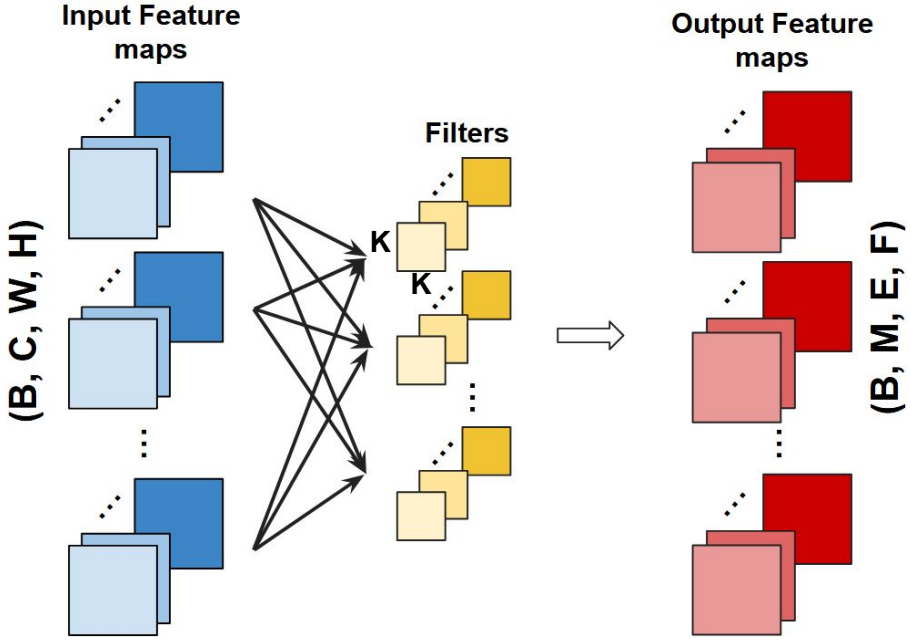


NYU SAI LAB

Convolution



Computational Cost: Standard Convolution



- Number of MACs: B×M×K×K×C×E×F
- Storage cost: 32×(M×C×K×K+B×C×H×W+B×M×E×F)

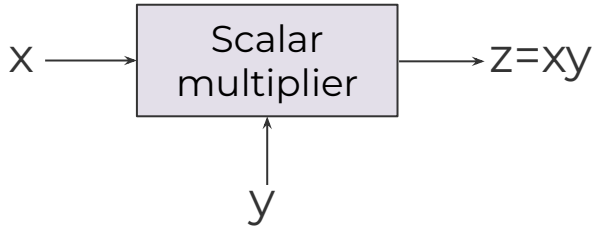
  B: batch size
  C: number of input channels
  H,W: size of the input feature maps
  M: number of weight filters
  K: weight kernel size
  E,F: size of the output feature maps

- We need to iterate over seven dimensions:
  - B, M, C, E, F, K(kernel width), K (kernel height)

NYU SAI LAB

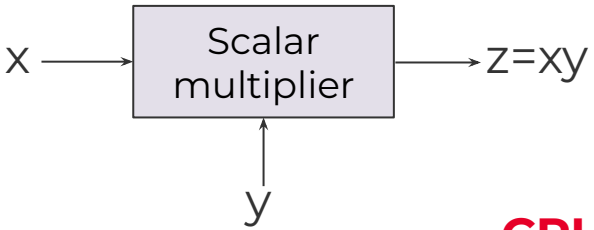
Computational Dataflow for CNN

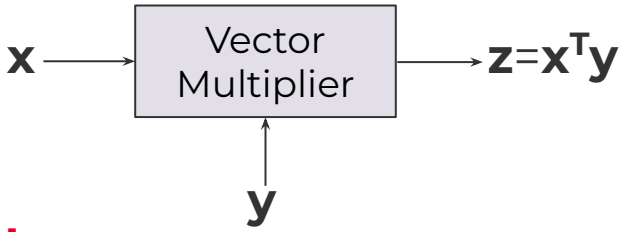
```
for b = 1 to B
   for m = 1 to M
      for c = 1 to C
         for w = 1 to E
            for h = 1 to F
               for k1 = 1 to K
                  for k2 = 1 to K
                     out[b][m][e][f] += in[b][c][e+k1-(K+1)/2][f+k2-(K+1)/2] * filter[m][c][k1][k2];
```



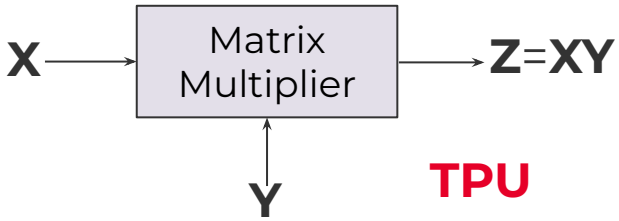
- This simple loop nest can be transformed in numerous ways to capture different reuse patterns of the activations and weights and to map the computation to a hardware accelerator implementation.
- A CNN's dataflow defines how the loops are ordered, partitioned, and parallelized
- We can use the scaler machine to compute the results of CNN using this for loop

Computational Dataflow for CNN

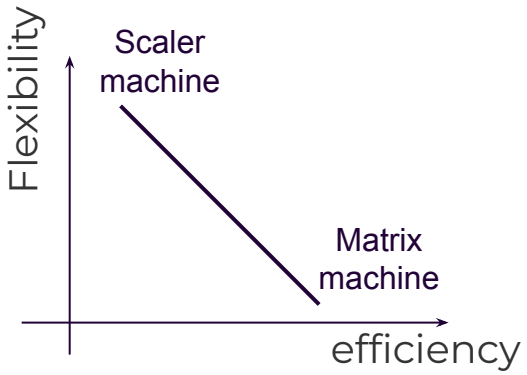




CPU & GPU

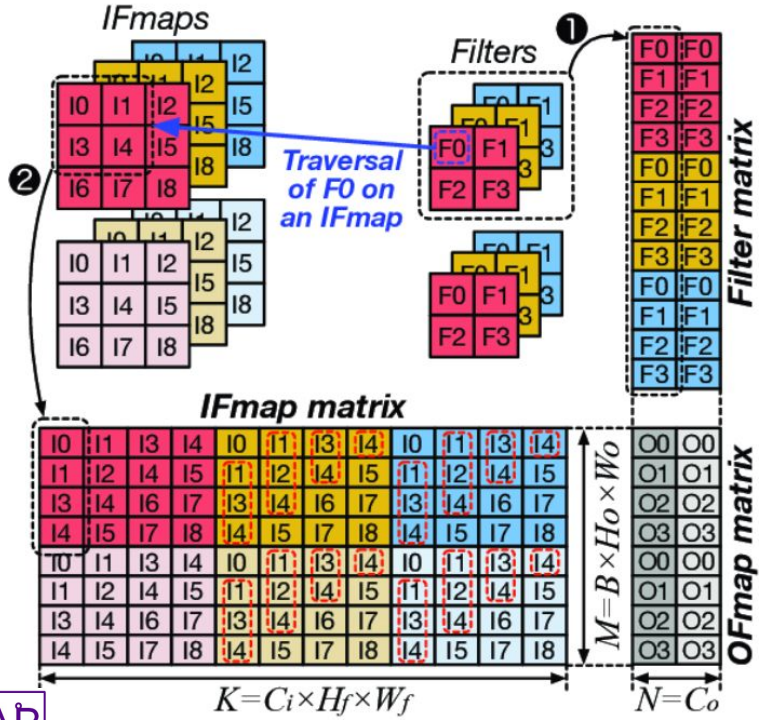


TPU



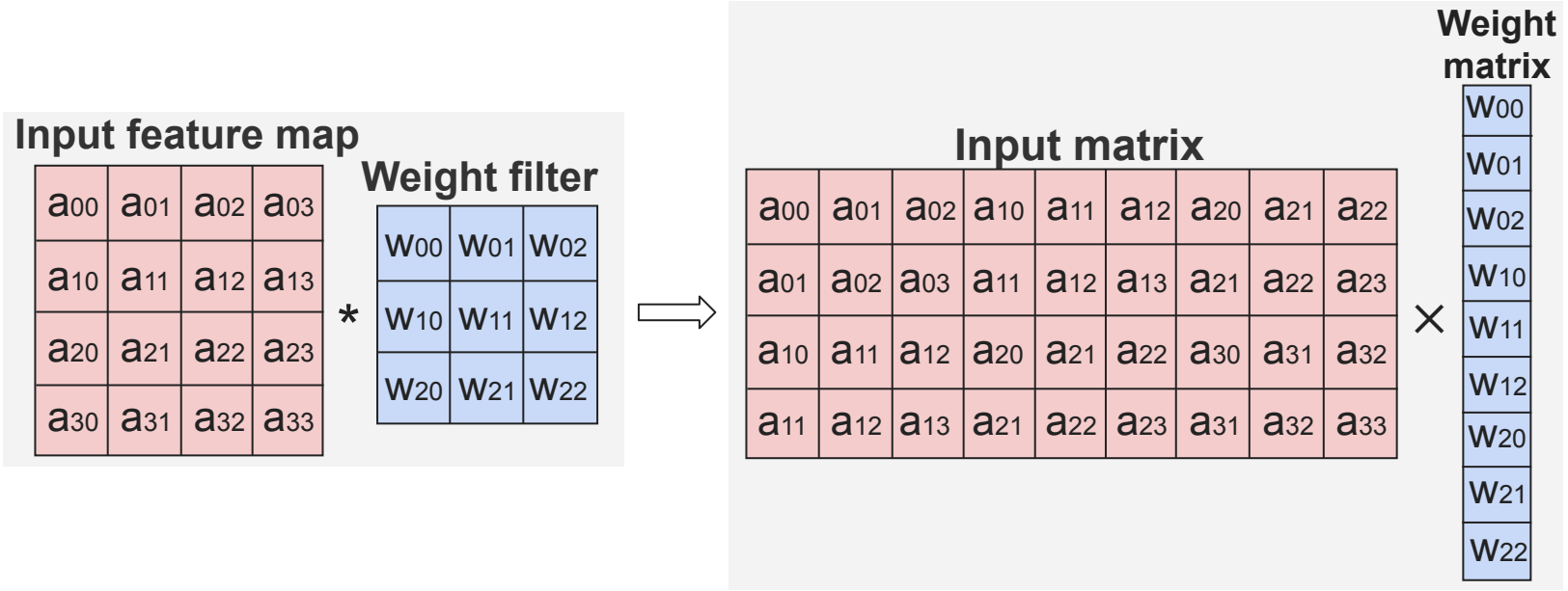
NYU SAI LAB

How to Convert to Matrix Multiplication?

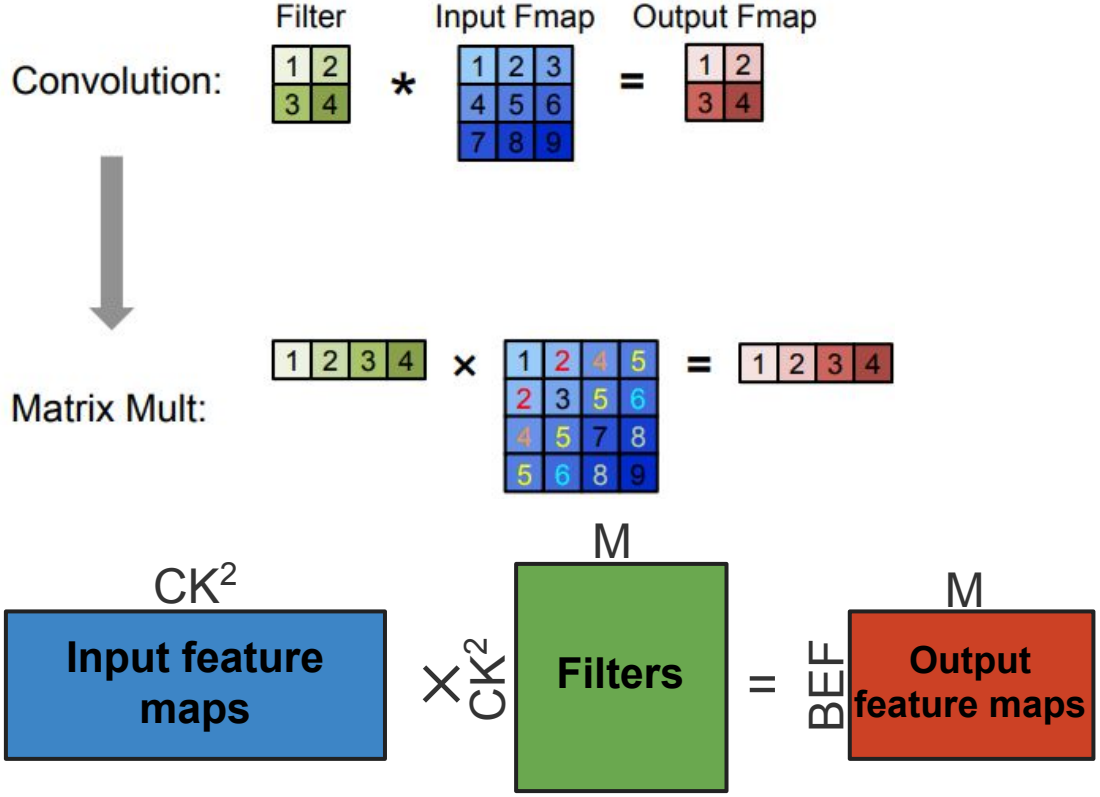


- A standard Convolutional operation can be converted to 2D matrix multiplication using **Im2Col** operations.

How to Convert to Matrix Multiplication?



How to Convert to Matrix Multiplication?



Tiling

- In order to handle matrix multiplication with large size, it is usually decomposed into tiles.

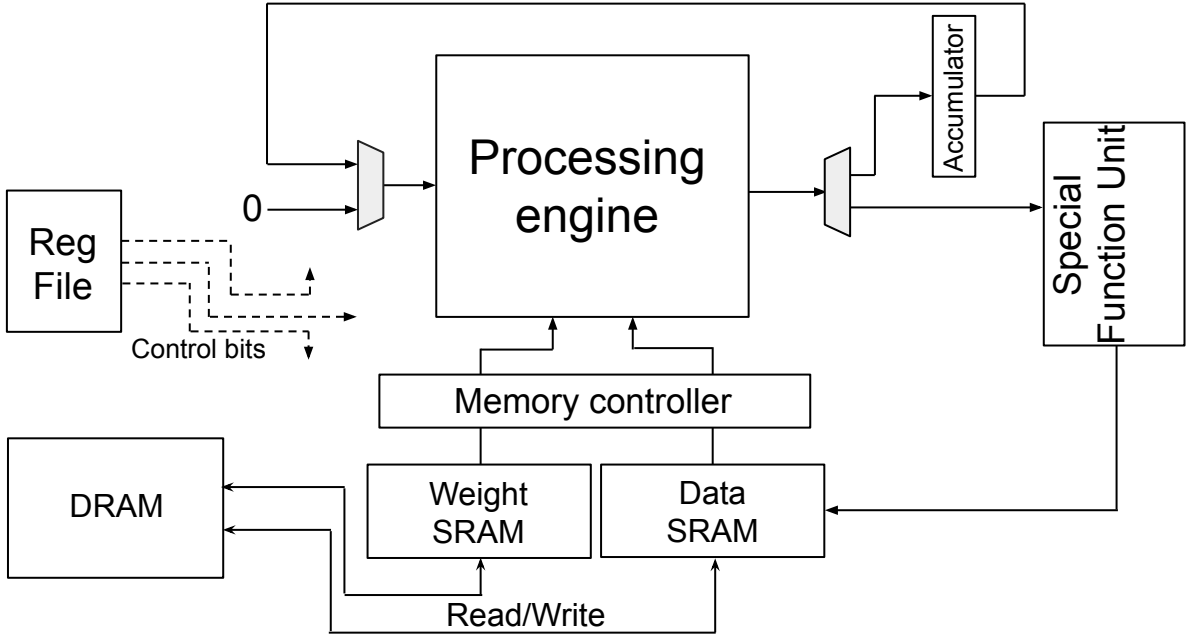
$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

- Each of W_{ij} and X_{ij} can be a sub-matrix.

Topics

- Hardware accelerator: Overview
- Convolutional operation conversion
- Hardware architecture of CNN accelerator
- Systolic array
- Popular accelerator design
  - Eyeriss
  - Diannao
  - Cnvlutin
  - EIE

Hardware Architectures for DNN Processing

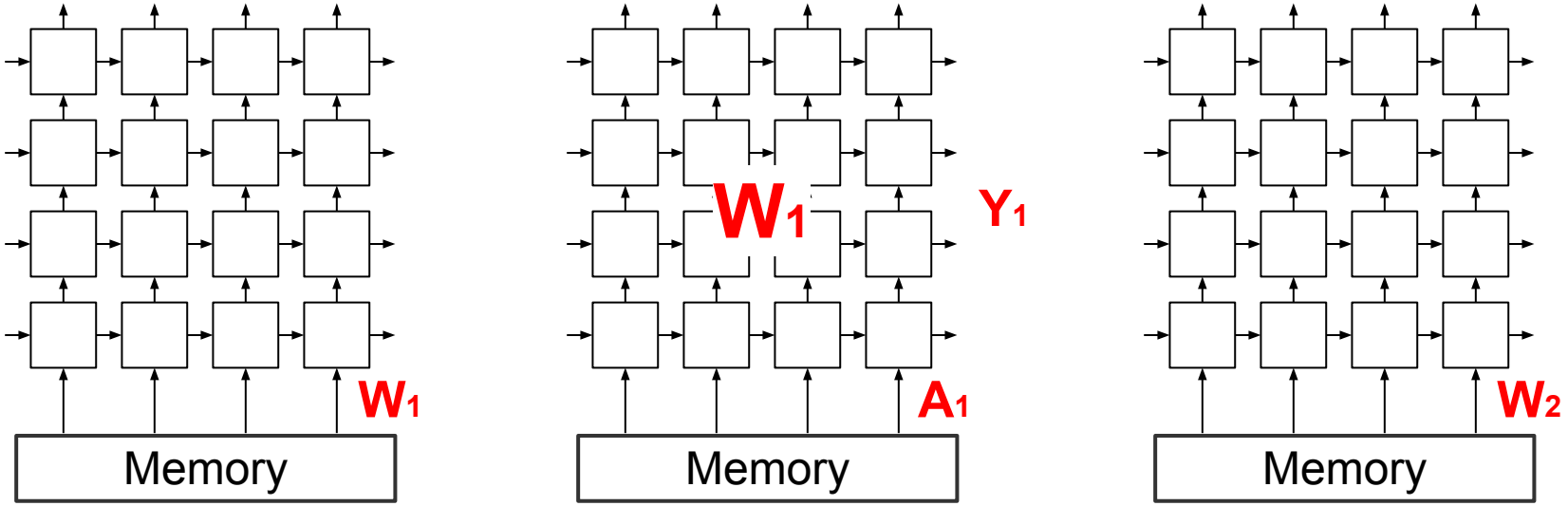


Major building blocks:

- Processing engine
- Accumulator
- Reg file
- Special function unit
- Memory subsystem
  - Weight SRAM
  - Data SRAM
  - DRAM

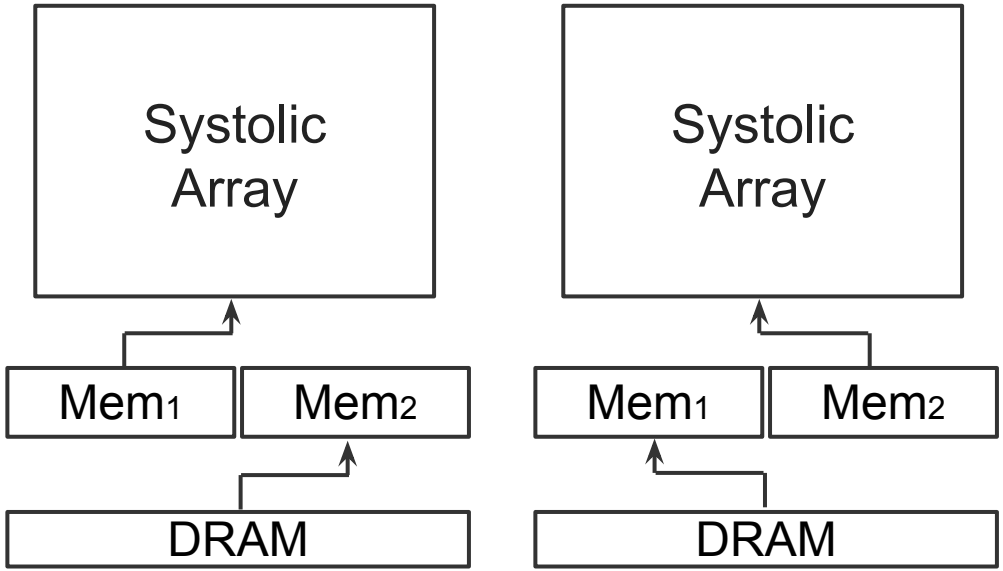
NYU SAI LAB

Computing Paradigms



- Spatial architecture can achieve great reuse of the extracted content, leading to a reduced memory access cost.

Double Buffering



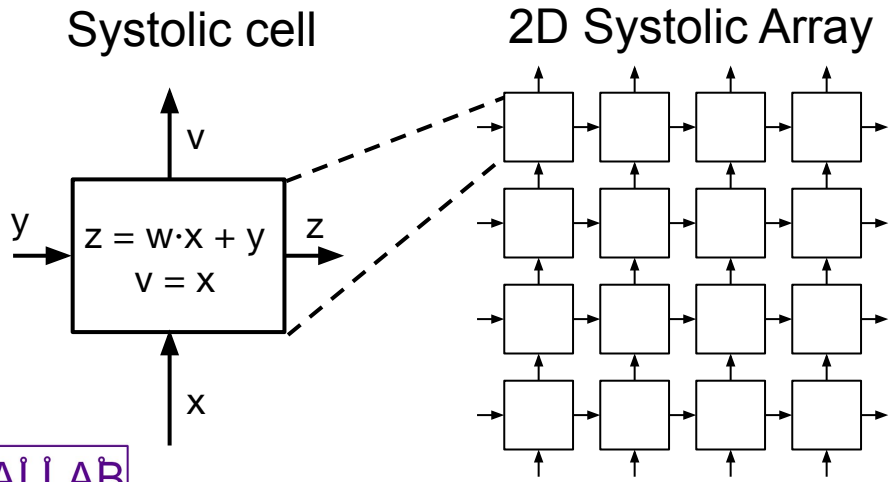
- Double buffering in hardware design is a technique used to improve the efficiency and performance of data processing, especially in systems that require smooth and continuous data transfer.
- The idea is to overlap the data production and consumption processes to avoid delays.

Topics

- Hardware accelerator: Overview
- Convolutional operation conversion
- Hardware architecture of CNN accelerator
- Systolic array
- Popular accelerator design
  - Eyeriss
  - Diannao
  - Cnvlutin
  - EIE

Systolic Array (Weight Stationary Version)

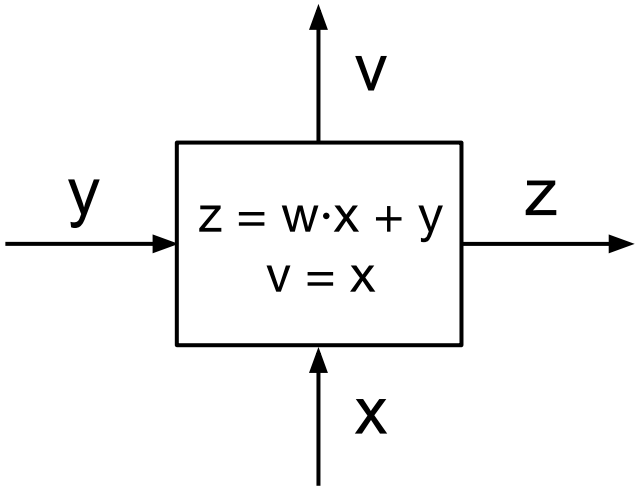
- Kung and Leiserson, "Systolic Arrays for VLSI," 1978 and Kung, "Why systolic architectures?' 1982
- 2D grid of multiplier-accumulators (MACs) for matrix multiplication
- Used by Google TPU for deep learning (2017), etc



TPU (Google)

NYU SAI LAB

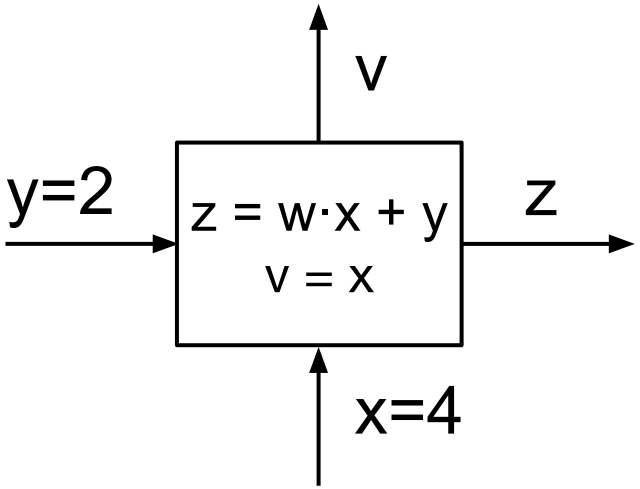
Systolic Cell



- Takes data (x and y) as input
- w stays in the systolic cell
- Performs a multiply-accumulate operation

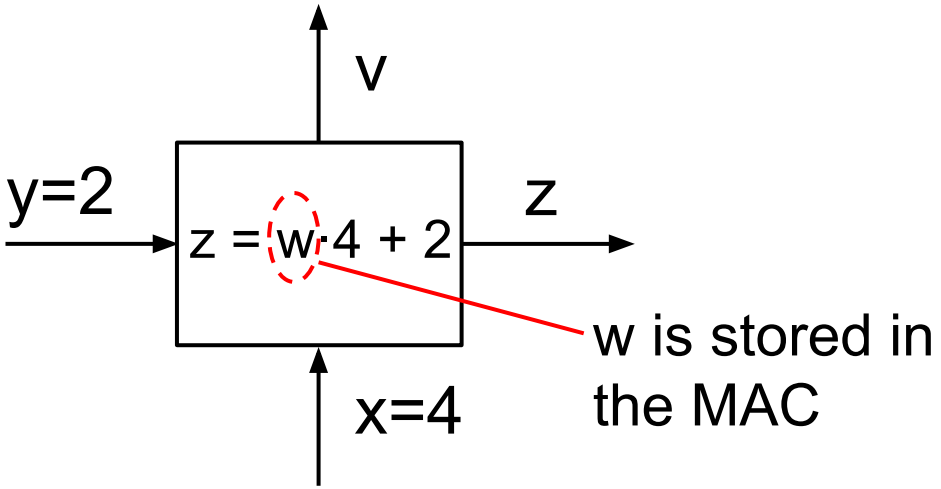
NYU SAI LAB

Systolic Cell

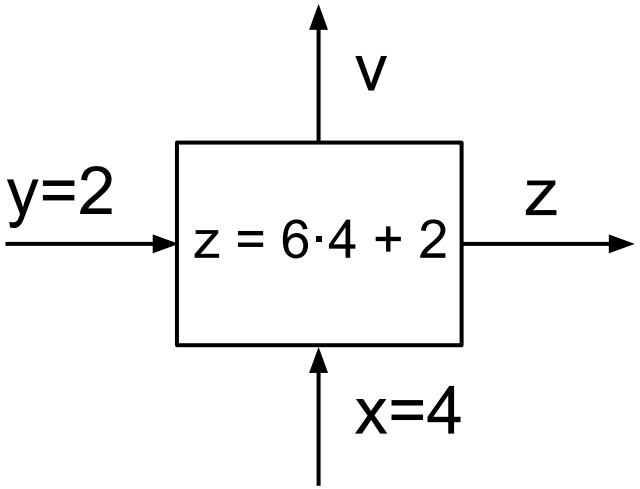


NYU SAI LAB

Systolic Cell

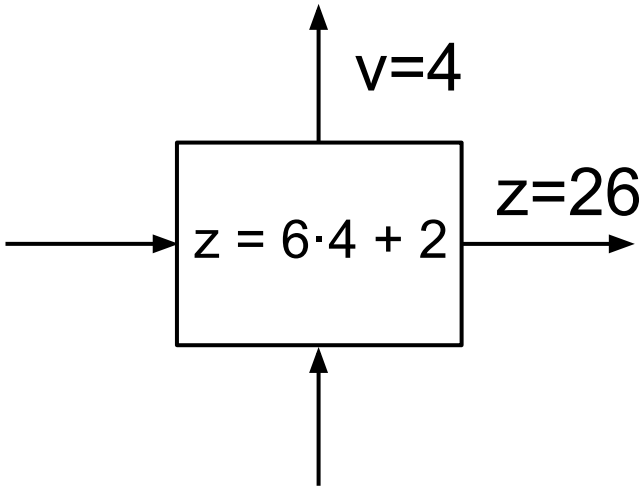


Systolic Cell



NYU SAI LAB

Systolic Cell

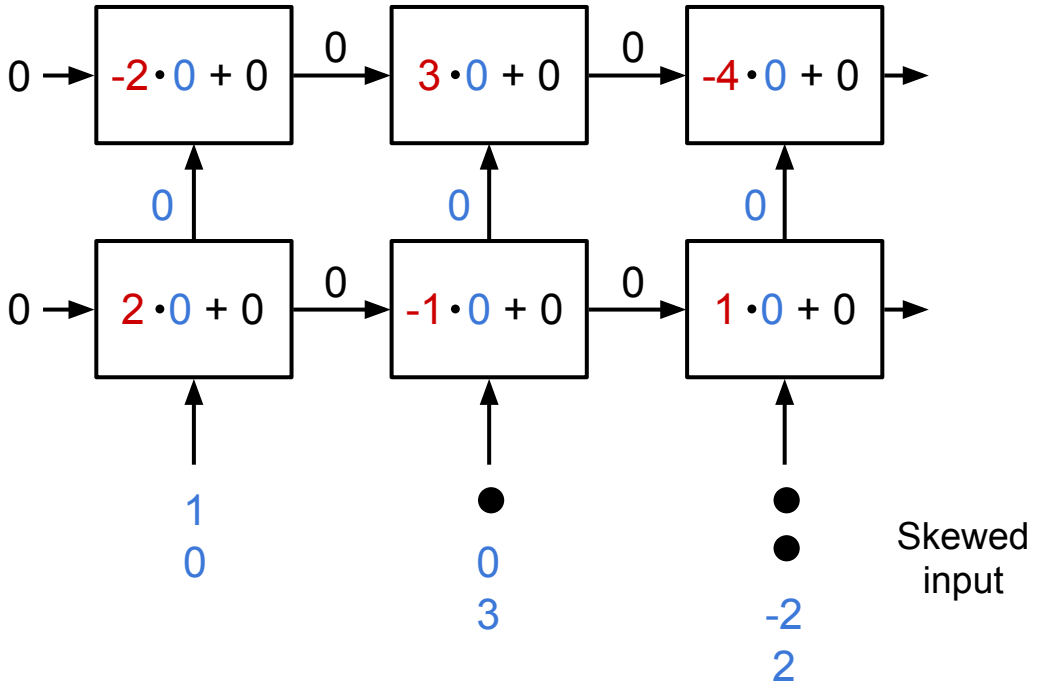


Visualizing Systolic Array Multiplication

Weight Matrix × Data Matrix = Result Matrix

$$\begin{bmatrix} 2 & -1 & 1 \\ -2 & 3 & -4 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 3 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 6 & 1 \end{bmatrix}$$

Weights in red are preloaded into the systolic array

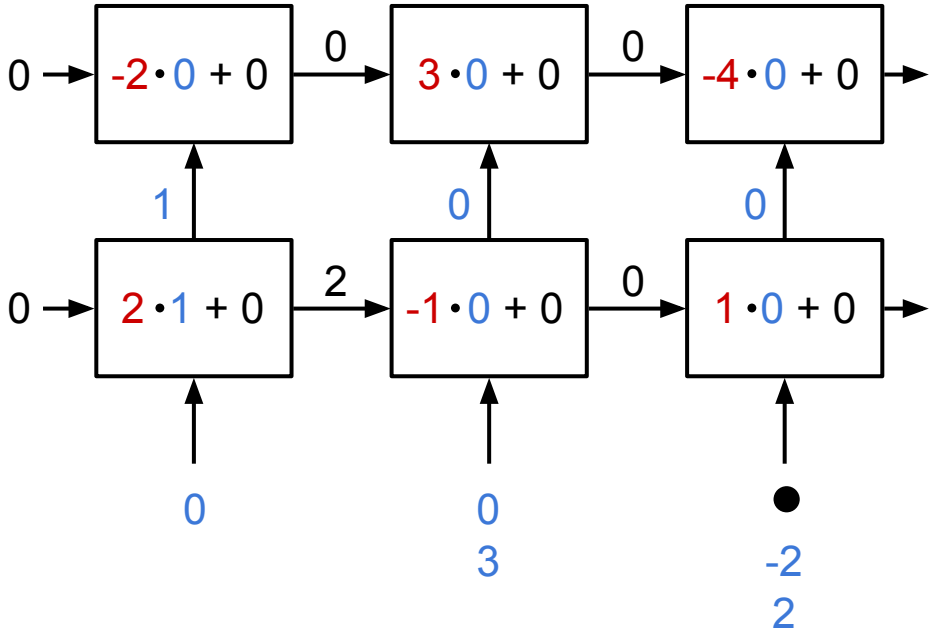


Visualizing Systolic Array Multiplication

Weight Matrix × Data Matrix = Result Matrix

$$\begin{bmatrix} 2 & -1 & 1 \\ -2 & 3 & -4 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 3 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 6 & 1 \end{bmatrix}$$

Weights in red are preloaded into the systolic array

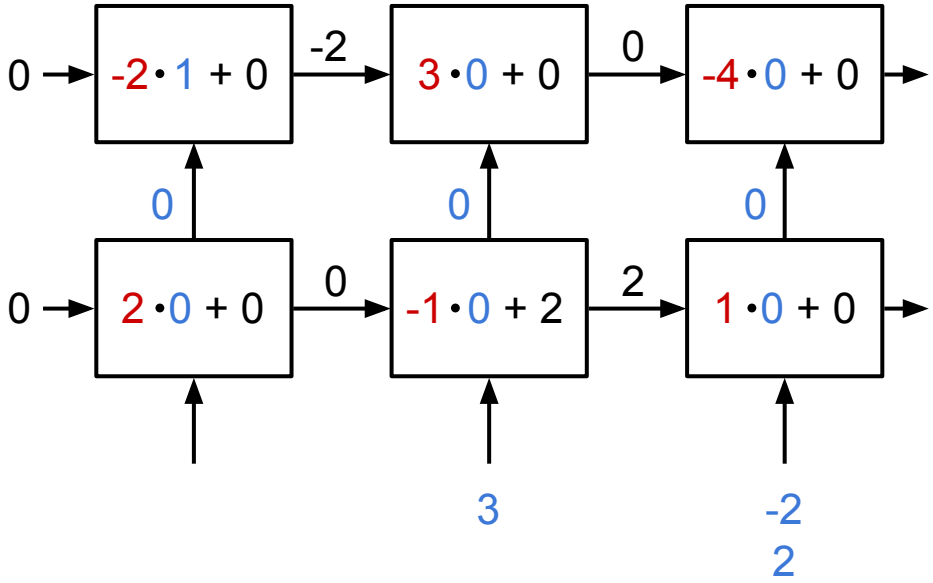


NYU SAI LAB

Visualizing Systolic Array Multiplication

Weight Matrix × Data Matrix = Result Matrix

$$\begin{bmatrix} 2 & -1 & 1 \\ -2 & 3 & -4 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 3 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 6 & 1 \end{bmatrix}$$



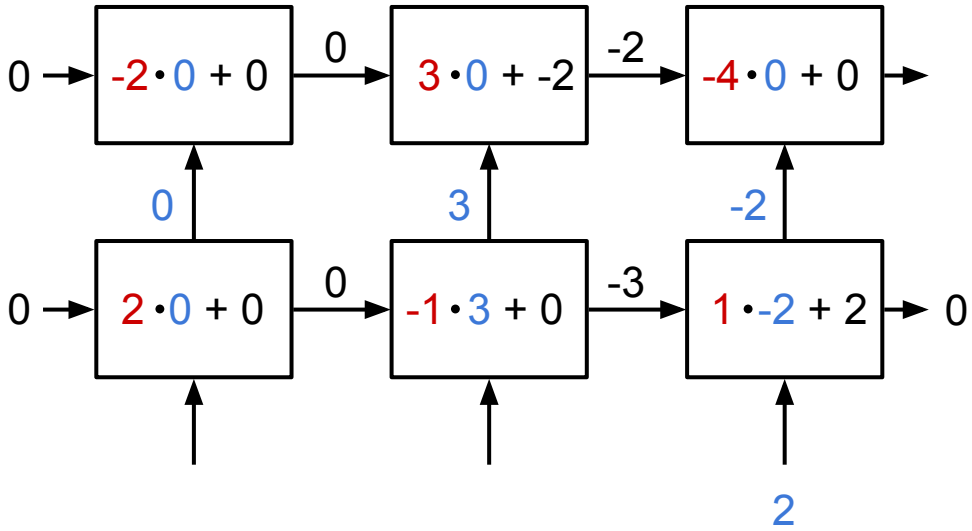
Weights in red are preloaded into the systolic array

NYU SAI LAB

Visualizing Systolic Array Multiplication

Weight Matrix × Data Matrix = Result Matrix

$$\begin{bmatrix} 2 & -1 & 1 \\ -2 & 3 & -4 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 3 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 6 & 1 \end{bmatrix}$$

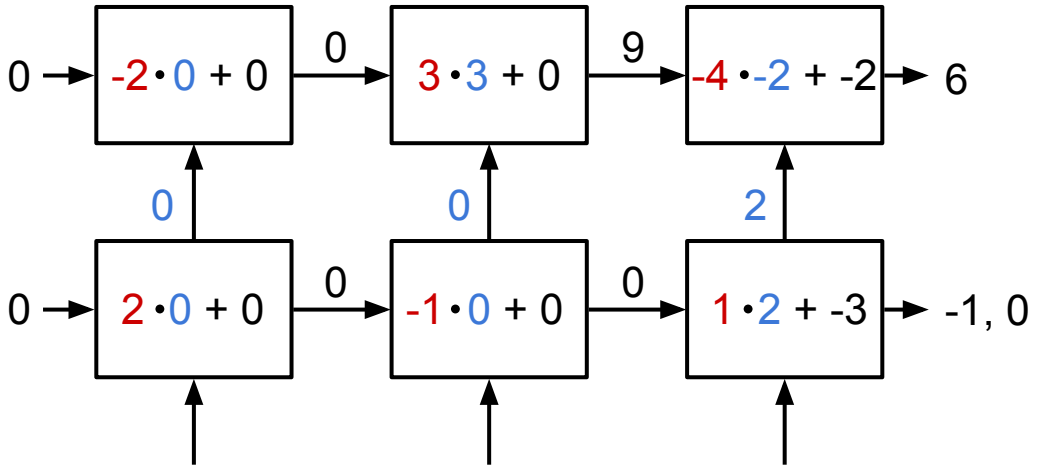


Weights in red are preloaded into the systolic array

NYU SAI LAB

Visualizing Systolic Array Multiplication

Weight Matrix			Data Matrix		Result Matrix	
2	-1	1	1	0	0	-1
-2	3	-4	0	3	6	1
			-2	2		



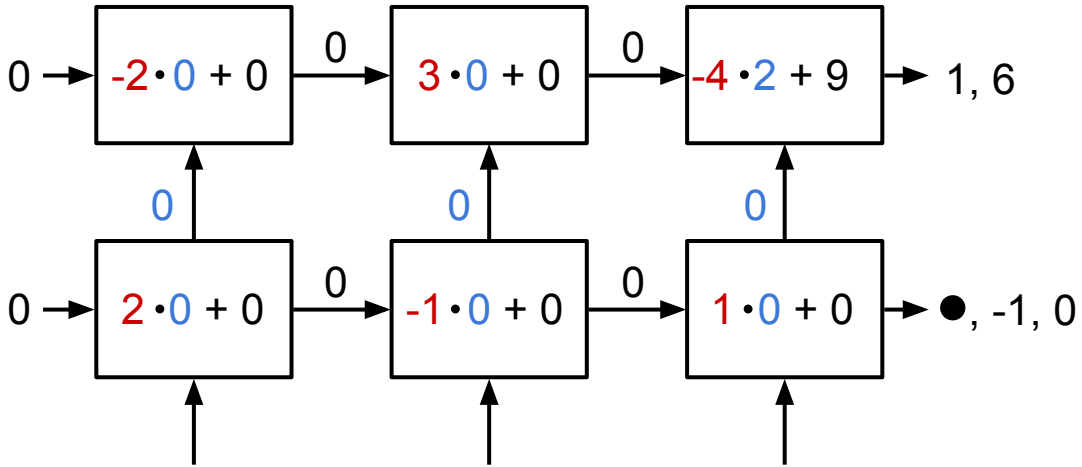
Weights in red are preloaded into the systolic array

NYU SAI LAB

Visualizing Systolic Array Multiplication

Weight Matrix × Data Matrix = Result Matrix

$$\begin{bmatrix} 2 & -1 & 1 \\ -2 & 3 & -4 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 3 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 6 & 1 \end{bmatrix}$$



Weights in red are preloaded into the systolic array

NYU SAI LAB

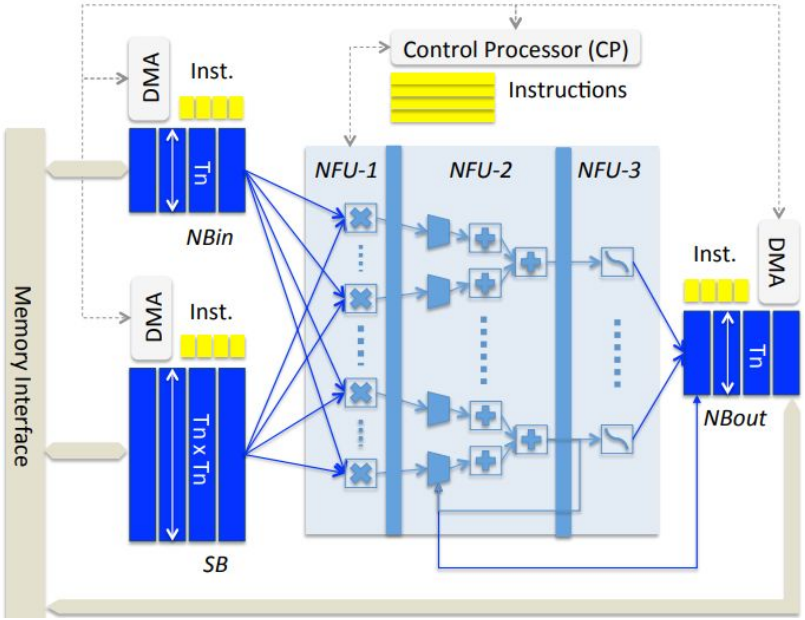
Topics

- Convolutional operation conversion
- Hardware architecture of CNN accelerator
- Systolic array
- Popular accelerator design
  - Eyeriss
  - Diannao
  - Cnvlutin
  - EIE

NYU SAI LAB

Chen, Yu-Hsin, et al. "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks." *IEEE journal of solid-state circuits* 52.1 (2016): 127-138.

Diannao

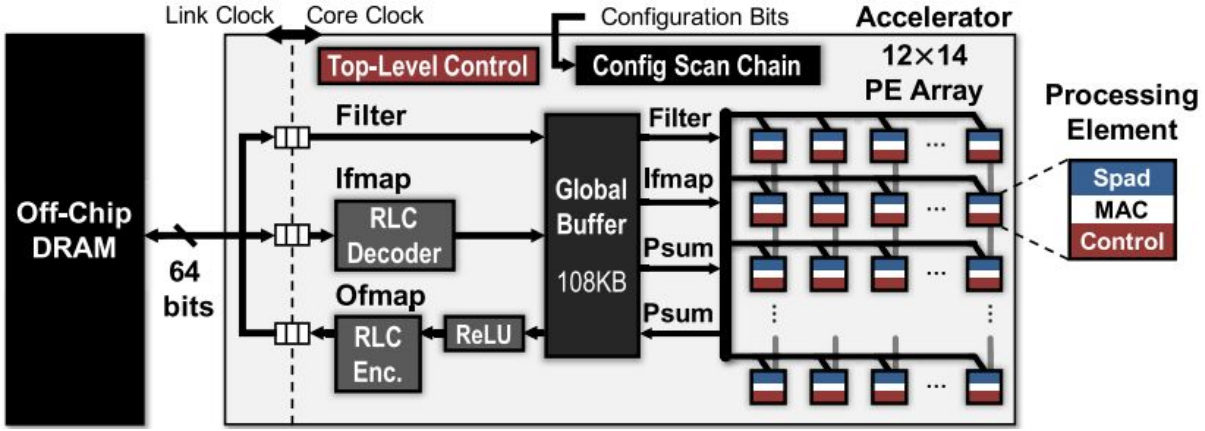


- The first popular end-to-end DNN (CNN) accelerator.
- Diannao is synthesized with 65nm using Synopsys tools, achieving a throughput of 482 GOP/s.
- NFU consists of three stages:
  - Multiplier units
  - Adder tree
  - Nonlinear unit

NYU SAI LAB

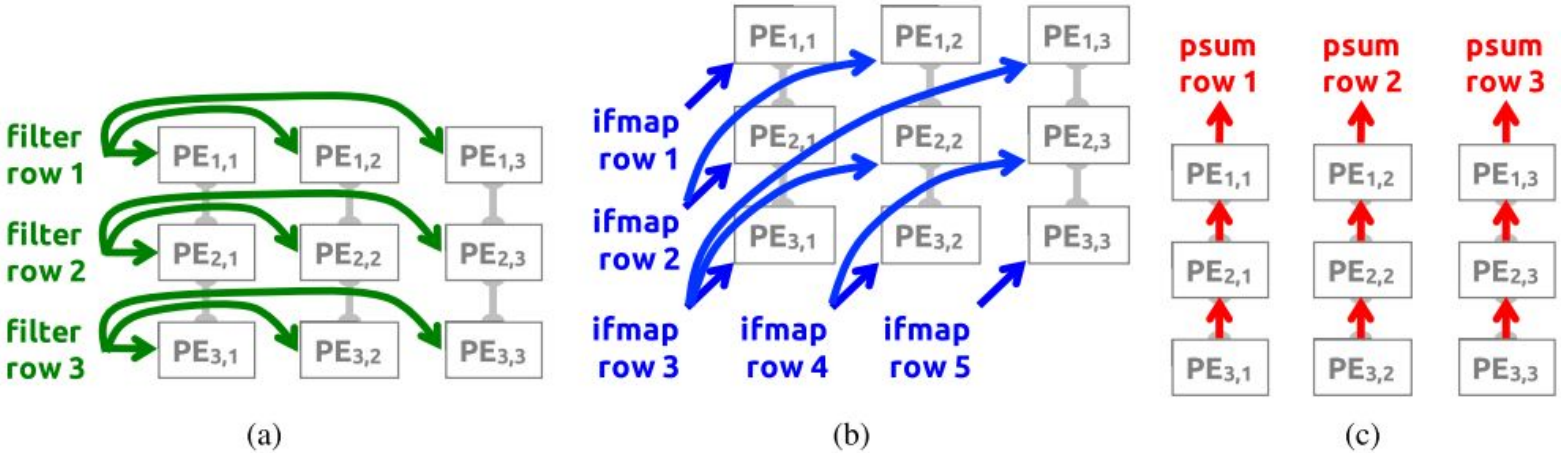
Chen, Tianshi, et al. "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning." *ACM SIGARCH Computer Architecture News* 42.1 (2014): 269-284.

Eyeriss



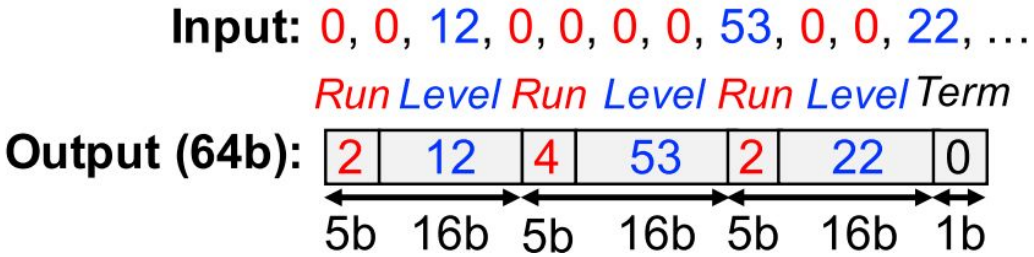
- Eyeriss optimizes for the energy efficiency of the entire system, including the accelerator chip and off-chip DRAM, for various CNN shapes by reconfiguring the architecture.
- The core clock domain consists of a spatial array of 168 PEs organized as a 12 × 14 rectangle, a 108-kB GLB, an RLC CODEC, and an ReLU module.

Data Reuse for Memory Access Reduction



- Reuse and accumulation of data within a PE set reduce accesses to the GLB and DRAM, saving data movement energy cost.

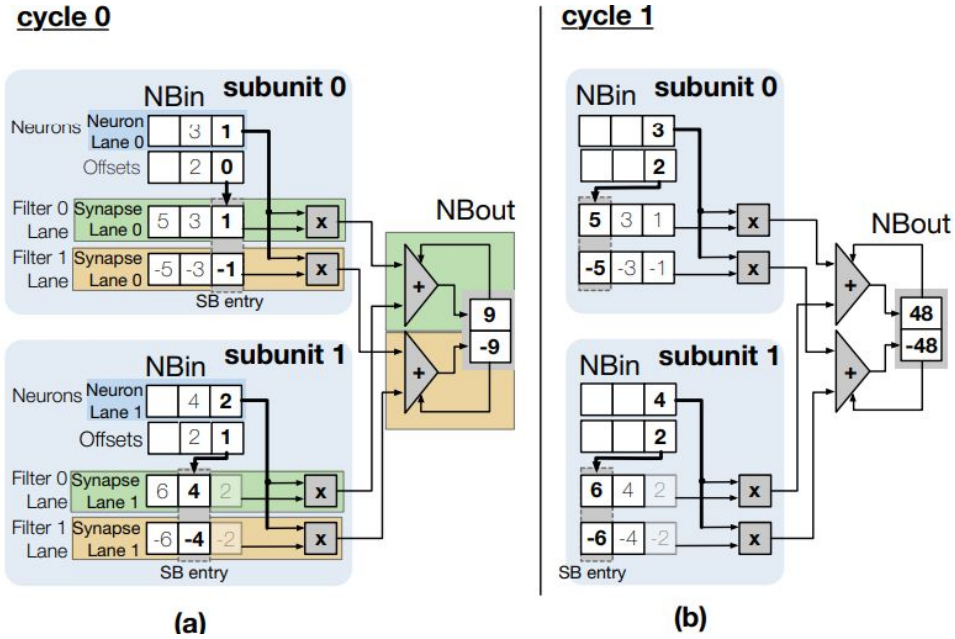
Rerun-length encoding



- RLC is used for compressing the input activation.

Cnvlutin

Input: [1, 0, 3] → [1, 3] (input) [0, 2] (offset)
Weight: [1, 3, 5]



- A large fraction of the computations performed by CNNs are intrinsically ineffectual as they involve a multiplication where one of the inputs is zero.
- Cnvlutin is a value-based approach to hardware acceleration that eliminates most of these ineffectual operations, improving performance and energy over a state-of-the-art accelerator with no accuracy loss.

NYU SAI LAB

Albericio, Jorge, et al. "Cnvlutin: Ineffectual-neuron-free deep neural network computing." *ACM SIGARCH Computer Architecture News* 44.3 (2016): 1-13.

Presentation

- Kangaroo: Lossless Self-Speculative Decoding via Double Early Exiting (Roshan)
- SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks (Lavanya, Murali)