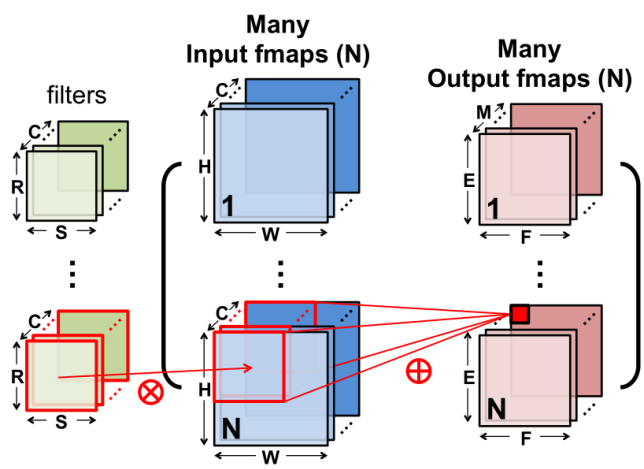
Review

- Last Class:
  - Systolic array and DNN basics
- Today's class:
  - DNN accelerator design based on systolic array
  - GPUs
- Announcement and reminder

36

36

Convolution



37

For DNN/CNN basics: https://eyeriss.mit.edu/

37

Convolution layer

Output fmaps — Biases — Input fmaps — Filter weights

$$\mathbf{O}[n][m][x][y] = \text{Activation}(\mathbf{B}[m] + \sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{k=0}^{C-1} \mathbf{I}[n][k][Ux+i][Uy+j] \times \mathbf{W}[m][k][i][j]),$$

$$0 \le n < N, 0 \le m < M, 0 \le y < E, 0 \le x < F,$$

$$E = (H - R + U)/U, F = (W - S + U)/U.$$

Shape Parameter	Description
N	fmap batch size
M	# of filters / # of output fmap channels
C	# of input fmap/filter channels
H/W	input fmap height/width
R/S	filter height/width
E/F	output fmap height/width
U	convolution stride

38

For DNN/CNN basics: https://eyeriss.mit.edu/

38

Naïve implementation in loops

Naïve 7-layer for-loop implementation:

for each output fmap value

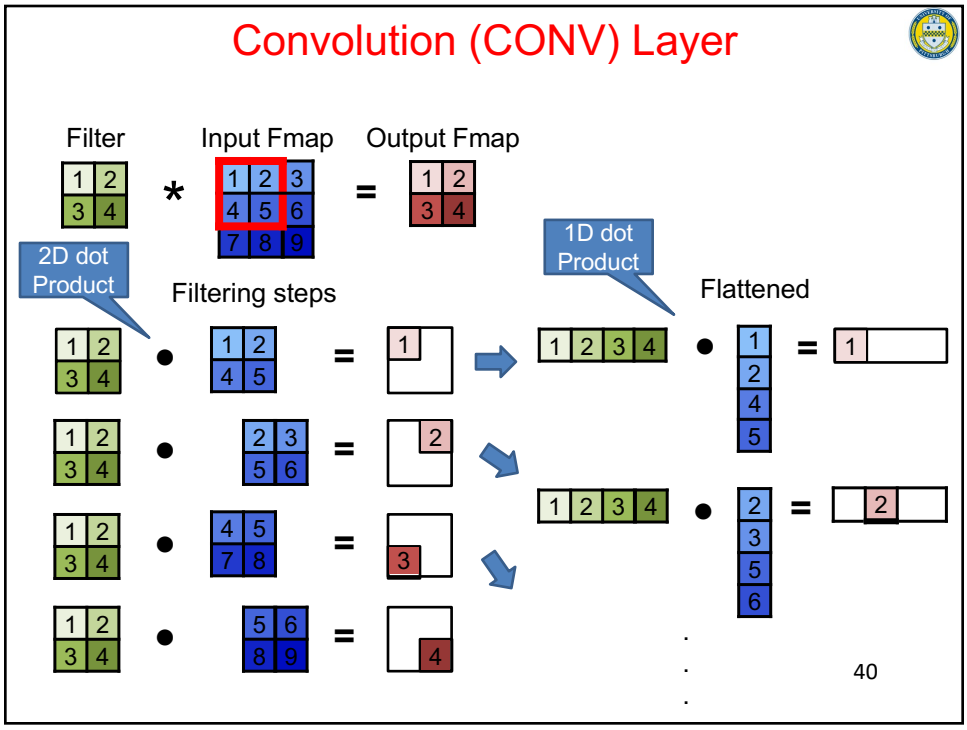
convolve a window and apply activation

```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {
                O[n][m][x][y] = B[m];
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        for (k=0; k<C; k++) {
                            O[n][m][x][y] += I[n][k][Ux+i][Uy+j] × W[m][k][i][j];
                        }
                    }
                }
                O[n][m][x][y] = Activation(O[n][m][x][y]);
            }
        }
    }
}
```

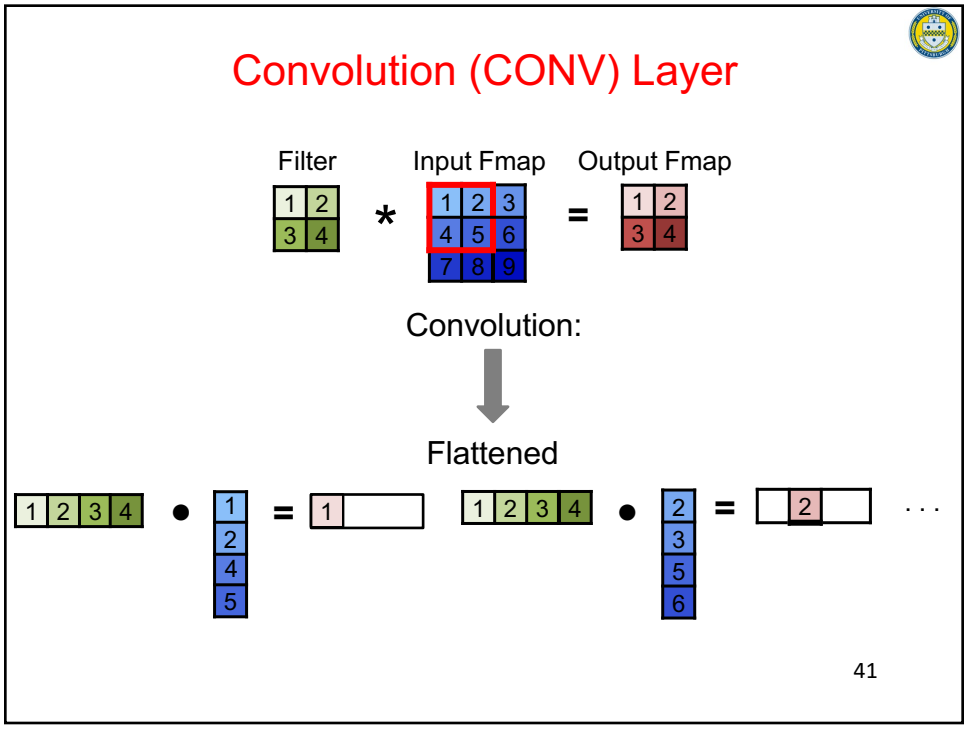
39

For DNN/CNN basics: https://eyeriss.mit.edu/

39

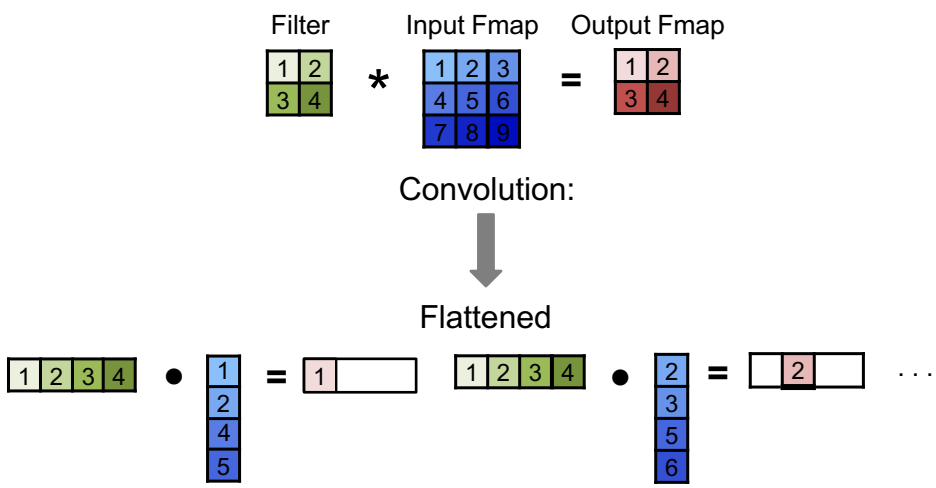

Convolution (CONV) Layer
Filter
Input Fmap
Output Fmap
2D dot Product
Filtering steps
1D dot Product
Flattened
40

40


Convolution (CONV) Layer
Filter
Input Fmap
Output Fmap
Convolution:
Flattened
41

41

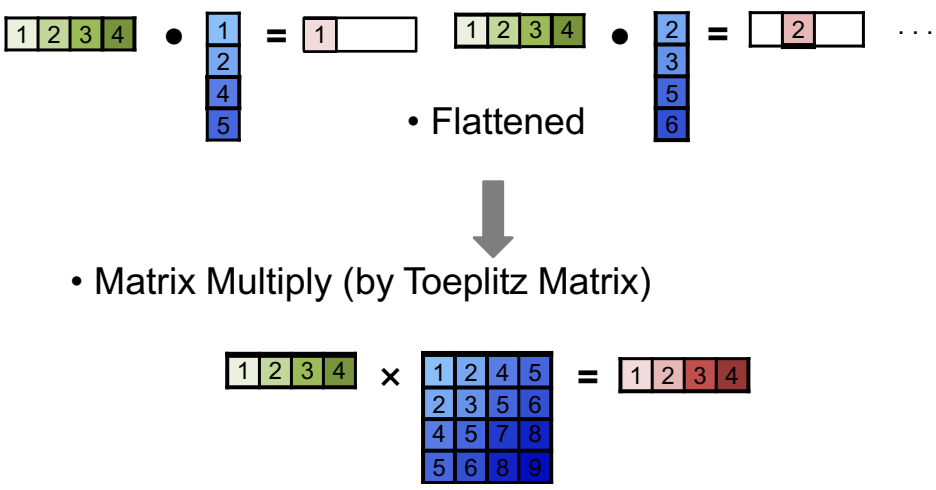
Convolution (CONV) Layer



42

42

Convolution (CONV) Layer



43

43

Convolution (CONV) Layer

Filter | Input Fmap | Output Fmap

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Convolution:

Matrix Multiply (by Toeplitz Matrix)

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

Convert to matrix multiply using the **Toeplitz Matrix** 44

44

Convolution (CONV) Layer

Filter | Input Fmap | Output Fmap

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Convolution:

Matrix Multiply (by Toeplitz Matrix)

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

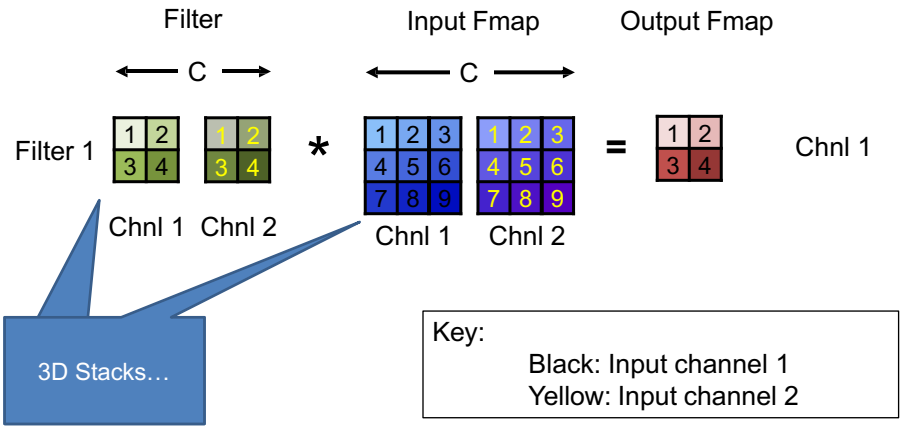
Data is repeated

45

45

Convolution (CONV) Layer

- Multiple Input Channels



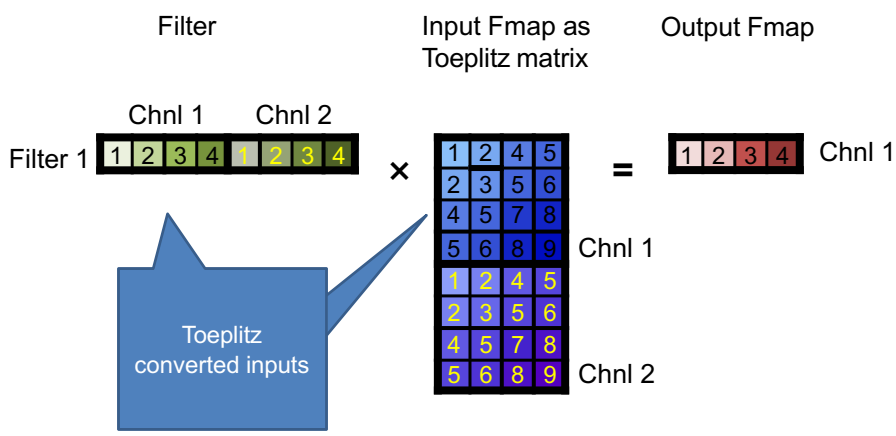
Key:
Black: Input channel 1
Yellow: Input channel 2

46

46

Convolution (CONV) Layer

- Multiple Input Channels

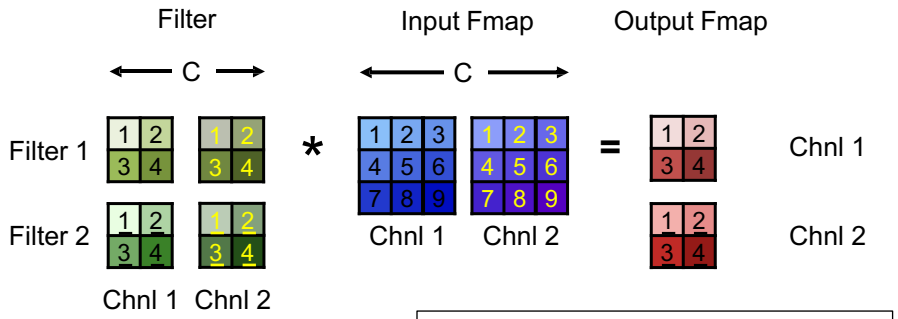


47

47

Convolution (CONV) Layer

- Multiple Input Channels and Output Channels



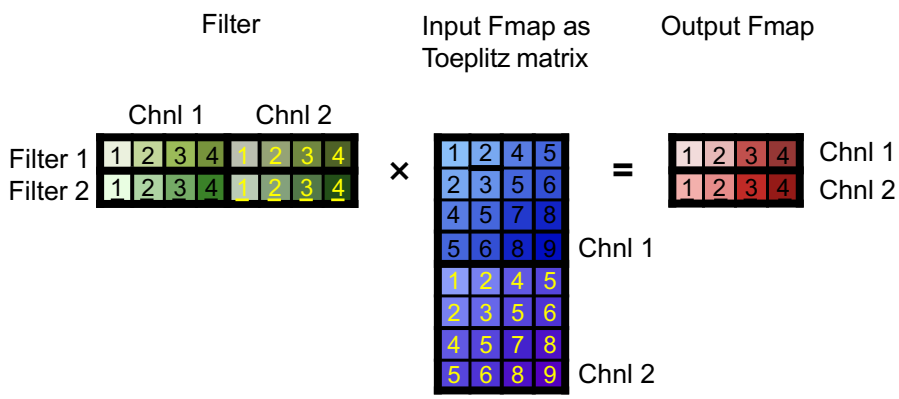
Key:
Black: Input channel 1
Yellow: Input channel 2
Underlined: Output channel 2

48

48

Convolution (CONV) Layer

- Multiple Input Channels and Output Channels

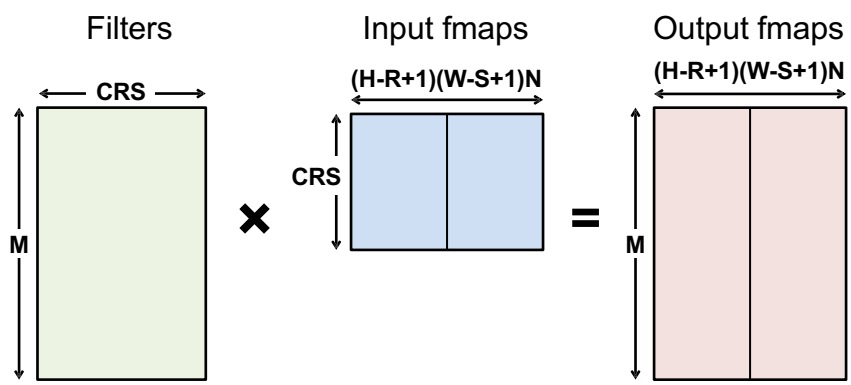


49

49

Convolution (CONV) Layer

- Dimensions of matrices for matrix multiply in convolution layers with batch size N



N=2 in example

50

50

How are they related with CNN/DNN?

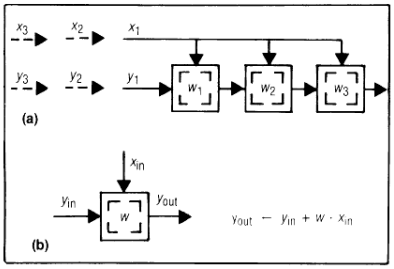


Figure 3. Design B1: systolic convolution array (a) and cell (b) where x_i's are broadcast, w_i's stay, and y_i's move systolically.

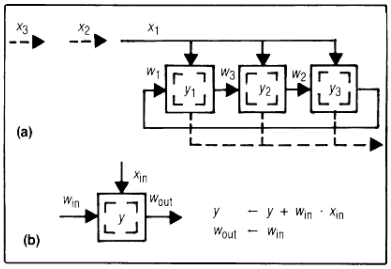


Figure 4. Design B2: systolic convolution array (a) and cell (b) where x_i's are broadcast, y_i's stay, and w_i's move systolically.

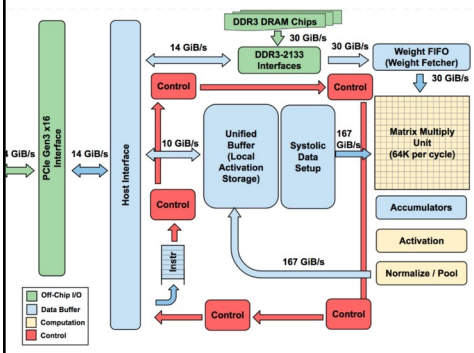
51

Ref: Why Systolic Architectures? – kung 1982

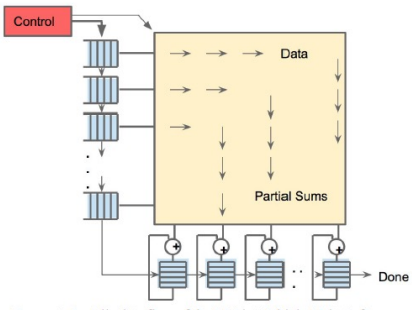
51

Fundamental design of PEs in TPU

Top-Level Architecture



Matrix Multiply Unit



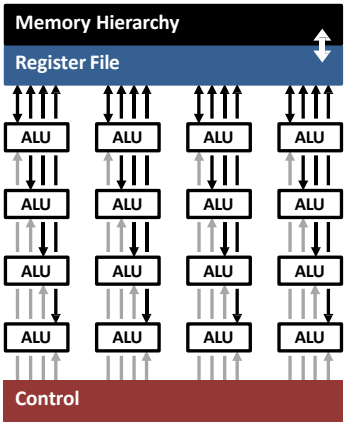
52

Ref: In-Datacenter Performance Analysis of a Tensor Processing Unit - google

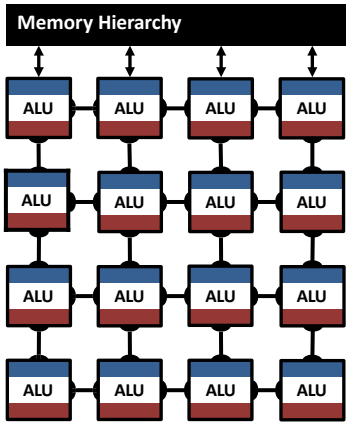
52

Highly-Parallel Compute Paradigms

Temporal Architecture (SIMD/SIMT)



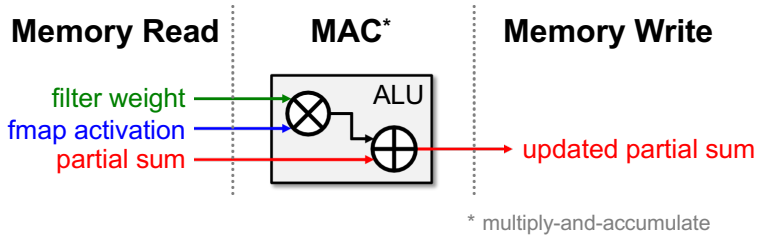
Spatial Architecture (Dataflow Processing)



53

53

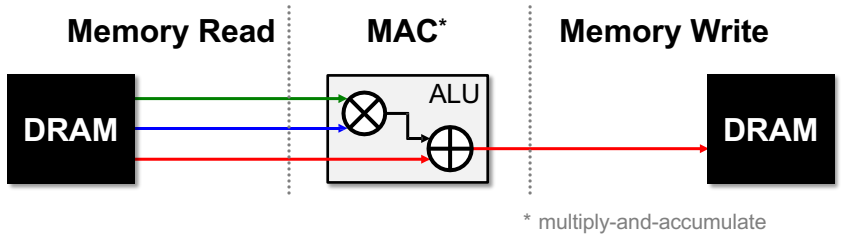
Memory Access is the Bottleneck



54

54

Memory Access is the Bottleneck



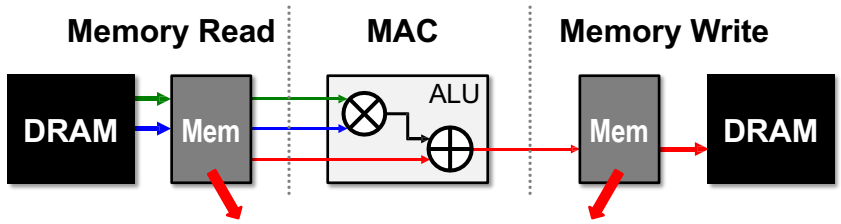
<u>Worst Case</u>: all memory R/W are **DRAM** accesses

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

55

55

Leverage Local Memory for Data Reuse



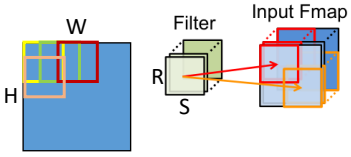
Extra levels of local memory hierarchy
Smaller, but **Faster** and **more Energy-Efficient**

56

56

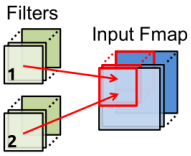
Type of reuse

Convolutional Reuse
CONV layers only
(sliding window)



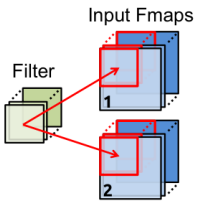
Reuse: Activations
Filter weights

Fmap Reuse
CONV and FC layers



Reuse: Activations

Filter Reuse
CONV and FC layers
(batch size > 1)



Reuse: Filter weights

Each filter is reused E*F times (E/F are 1 in FC)

Each pixel in Ifmap is reused R/min(U,R) *S/min(U,S)) times

Each pixel in Ifmap is reused M times

Each filer is reused N times

57

57

Types of Data Reuse in DNN

Convolutional Reuse

CONV layers only
(sliding window)

Fmap Reuse

CONV and FC layers

Filter Reuse

CONV and FC layers
(batch size > 1)

Input Fmaps

Filters

Input Fmap

Input Fmap

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

2

2

Reuse: Activations
Filter weights

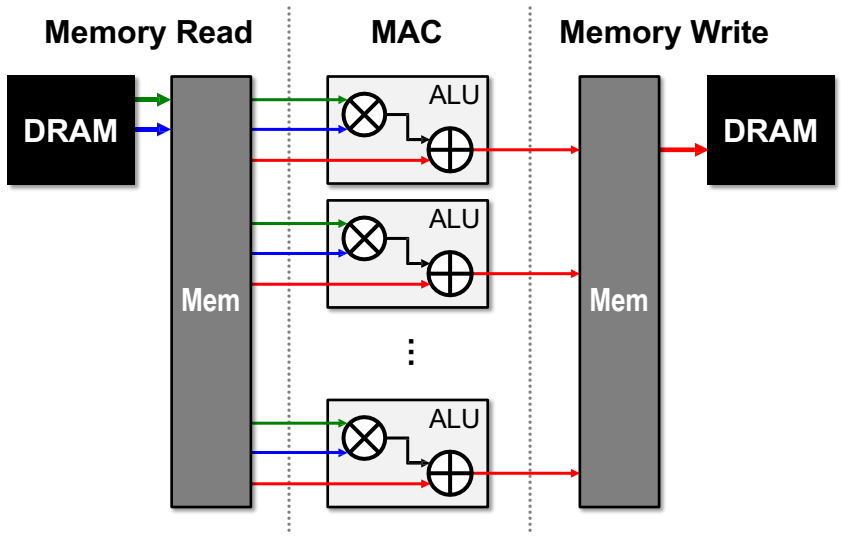
Reuse: Activations

Reuse: Filter weights

58

58

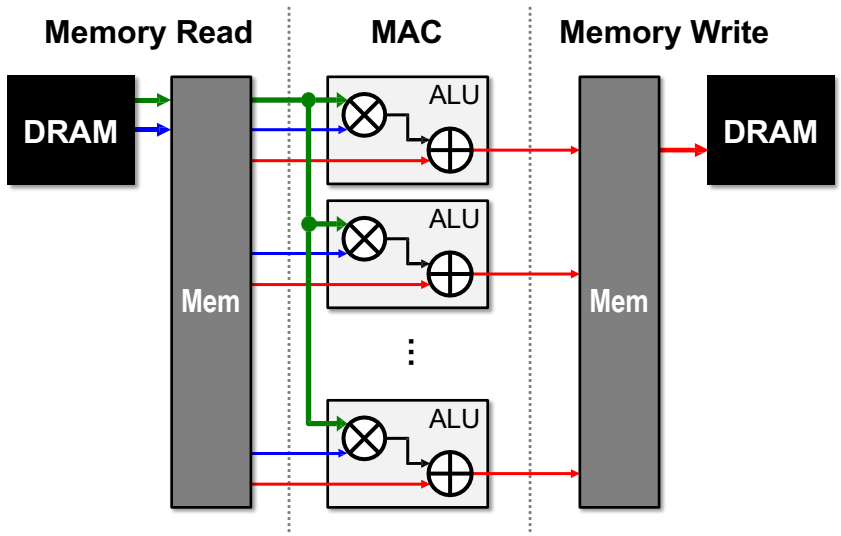
Leverage Parallelism for Higher Performance



59

59

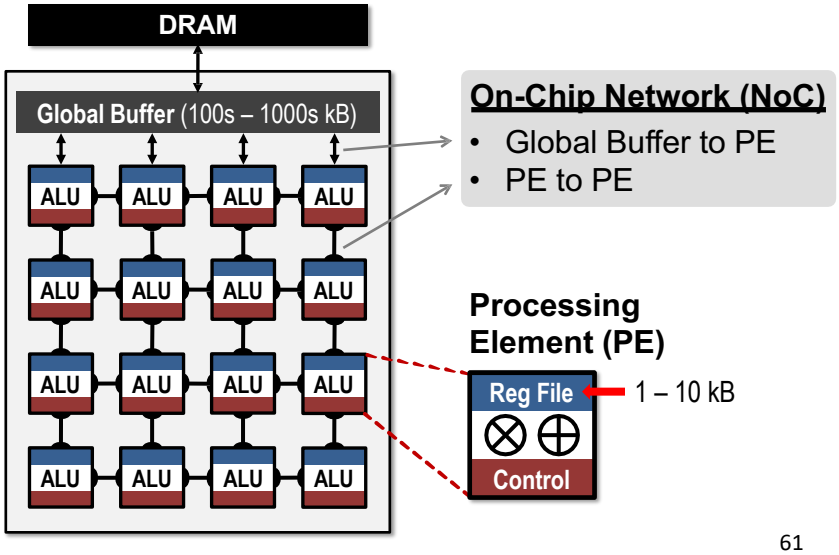
Leverage Parallelism for *Spatial* Data Reuse



60

60

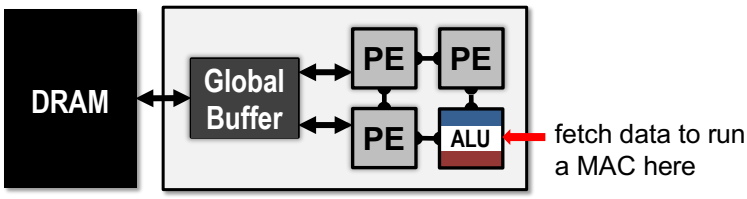
Spatial Architecture for DNN

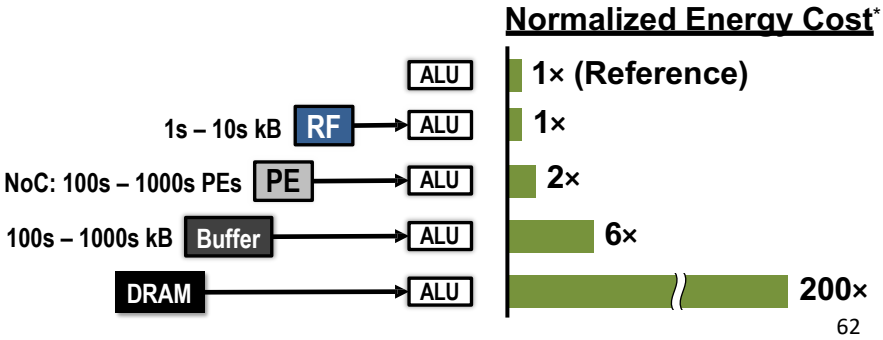


61

61

Multi-Level Low-Cost Data Access

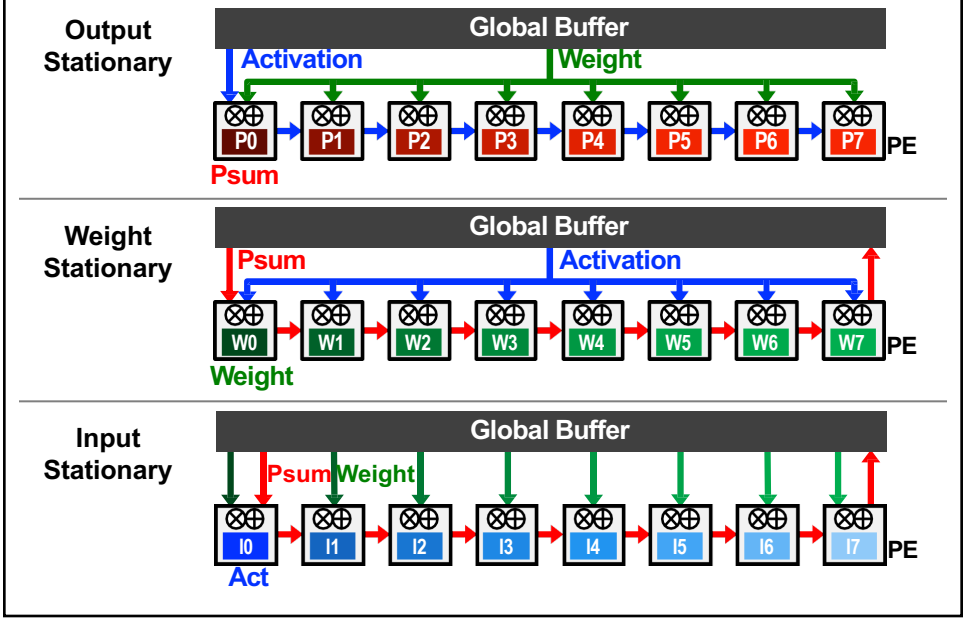




* measured from a commercial 65nm process

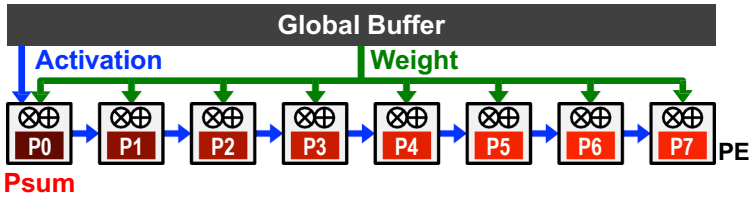
62

Dataflow Taxonomy



63

Output Stationary (OS)



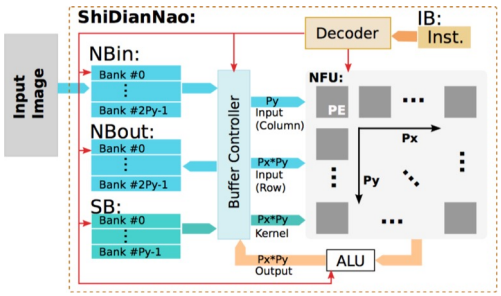
- **Minimize partial sum** R/W energy consumption
  - maximize local accumulation
- **Broadcast/Multicast filter weights** and **reuse activations spatially** across the PE array

64

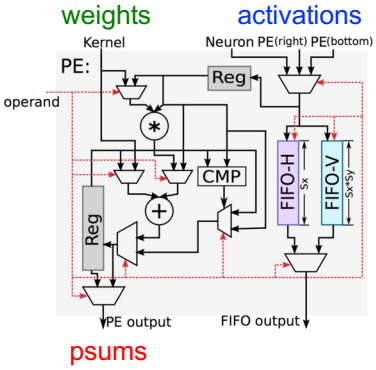
64

OS Example: ShiDianNao

Top-Level Architecture



PE Architecture

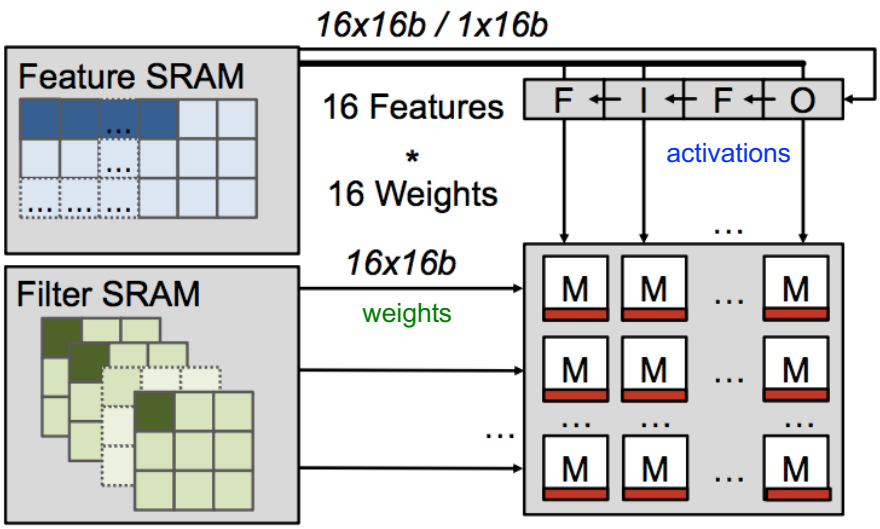


65

[Du et al., ISCA 2015]

65

OS Example: ENVISION

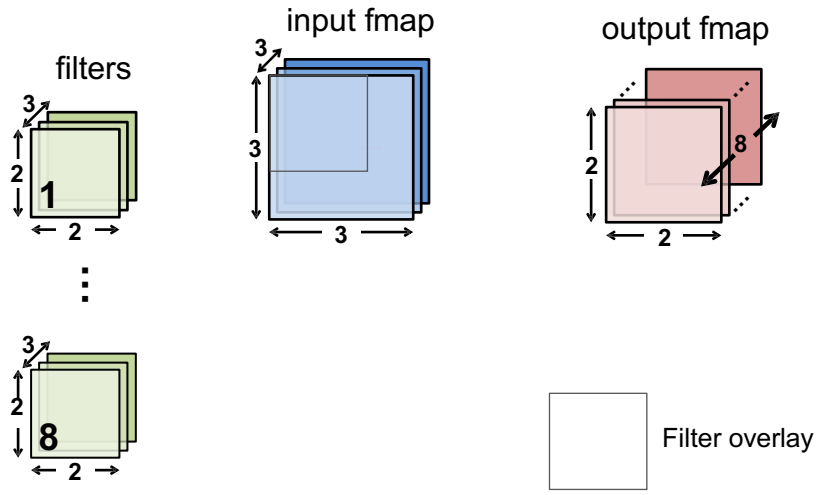


[Moons et al., VLSI 2016, ISSCC 2017]

66

66

OS Example

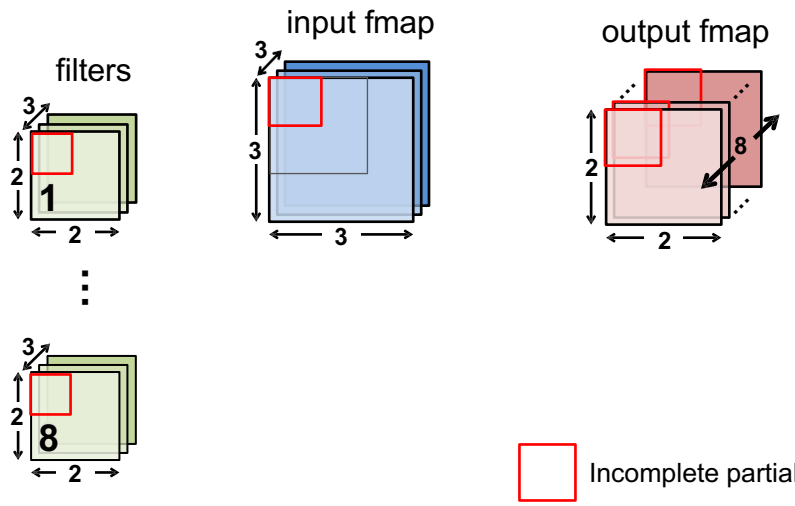


67

67

OS Example

Cycle through input fmap and weights (hold psum of output fmap)

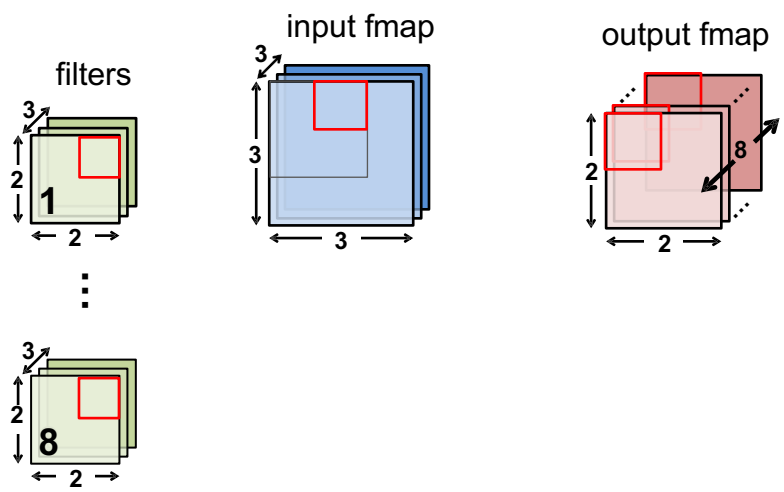


68

68

OS Example

Cycle through input fmap and weights (hold psum of output fmap)

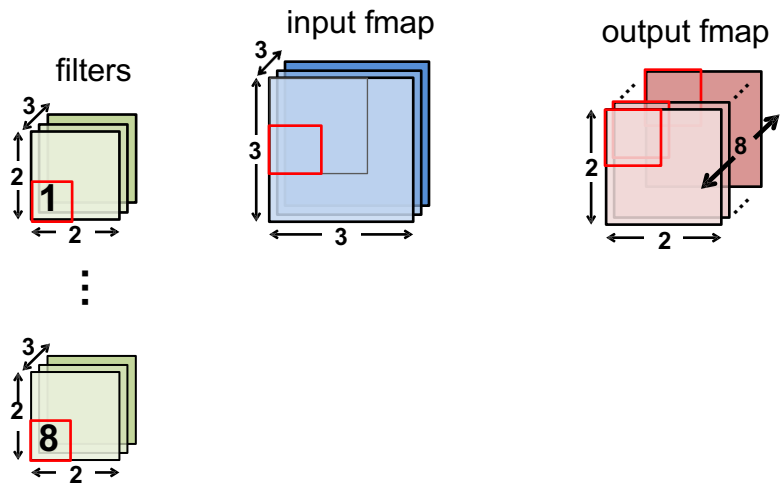


69

69

OS Example

Cycle through input fmap and weights (hold psum of output fmap)

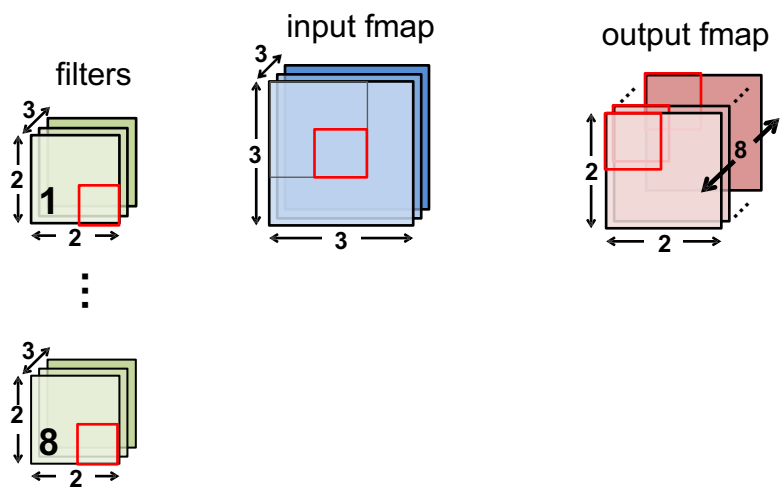


70

70

OS Example

Cycle through input fmap and weights (hold psum of output fmap)

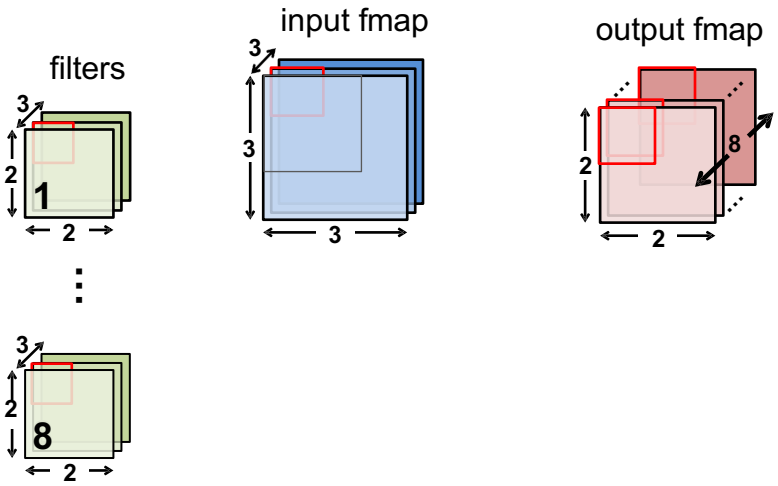


71

71

OS Example

Cycle through input fmap and weights (hold psum of output fmap)

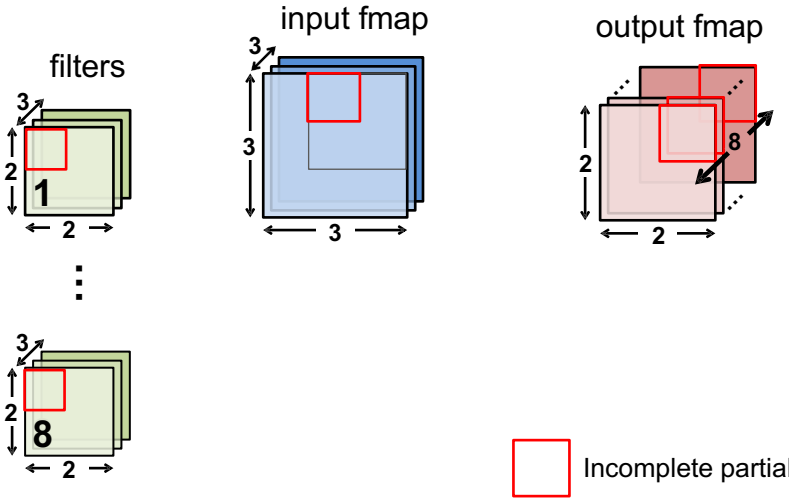


72

72

OS Example

Cycle through input fmap and weights (hold psum of output fmap)

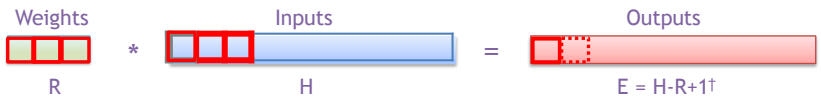


Incomplete partial sum

73

73

1-D Convolution – Output Stationary




```
int I[H];      // Input activations
int W[R];      // Filter weights
int O[E];      // Output activations

for (e = 0; e < E; e++)
    for (r = 0; r < R; r ++)
        O[e] += I[e+r] * W[r];
```

No constraints on loop permutations!

† Assuming: 'valid' style convolution

74

74

1-D Convolution

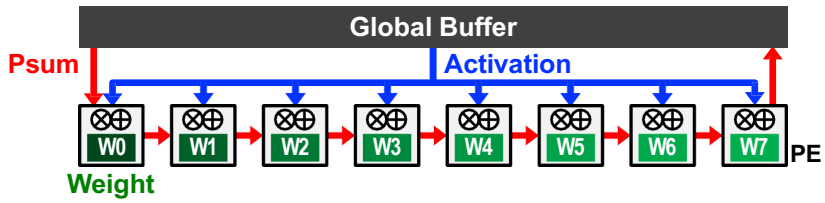

```
int I[H];      // Input activations
int W[R];      // Filter weights
int O[E];      // Output activations

for (r = 0; r < R; r++)
    for (e = 0; e < E; e++)
        O[e] += I[e+r] * W[r];
```

† Assuming: 'valid' style convolution

75

75

Weight Stationary (WS)



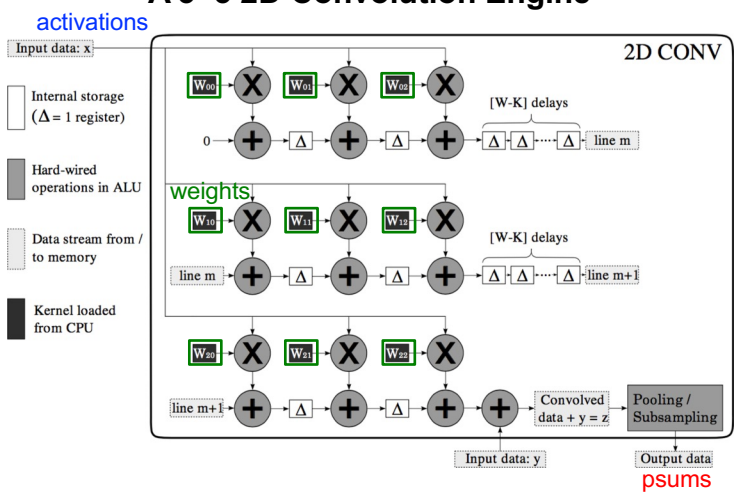
- **Minimize weight** read energy consumption
  - maximize convolutional and filter reuse of weights
- **Broadcast activations** and **accumulate psums spatially** across the PE array.

76

76

WS Example: nn-X (NeuFlow)

A 3×3 2D Convolution Engine



77

[Farabet et al., ICCV 2009]

77

WS Example: NVDLA (simplified)

Global Buffer

Released Sept 29, 2017

PE Array: *M x C MACs*

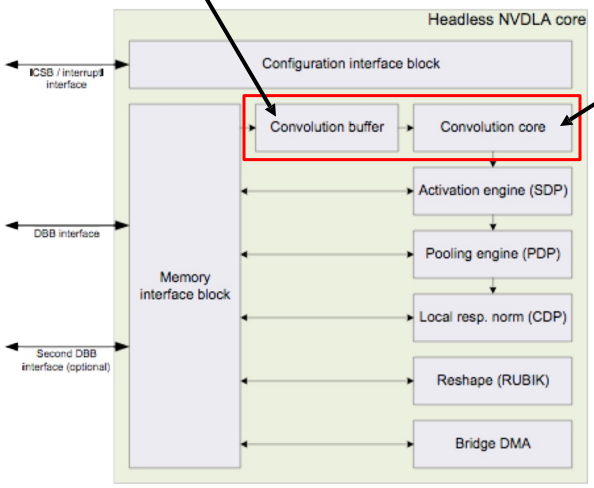


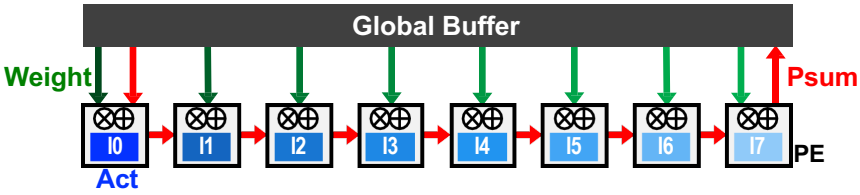
Image Source: Nvidia

http://nvdla.org

78

78

Input Stationary (IS)



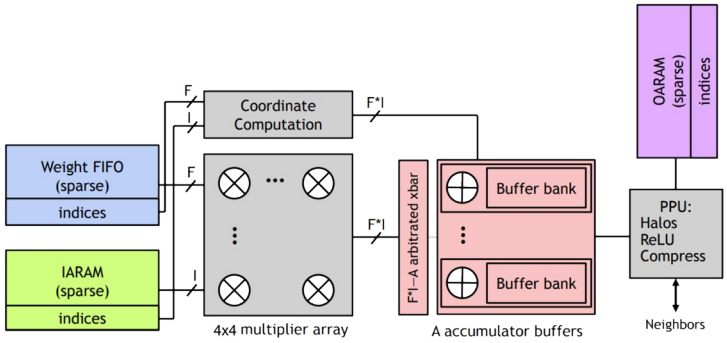
- **Minimize activation** read energy consumption
  - maximize convolutional and fmap reuse of activations
- **Unicast weights** and **accumulate psums spatially** across the PE array.

79

79

IS Example: SCNN

- Used for sparse CNNs
  - Sparse CNN is where many weights are zeros
  - Activations also have sparsity from ReLU



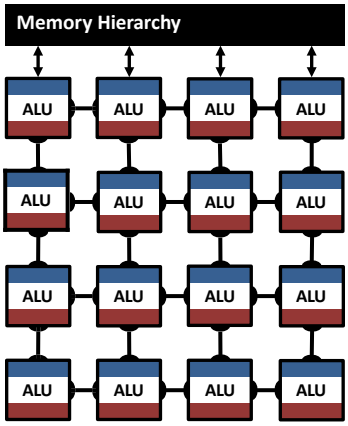
[Parashar et al., ISCA 2017]

80

80

Highly-Parallel Compute Paradigms

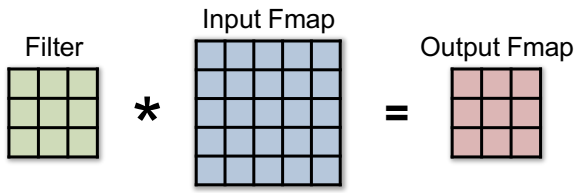
Spatial Architecture (Dataflow Processing)



81

81

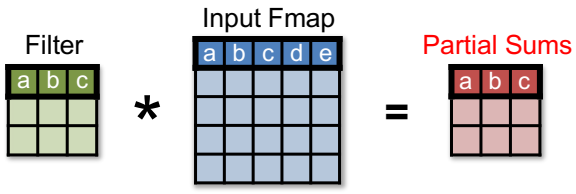
Row Stationary: Energy-efficient Dataflow

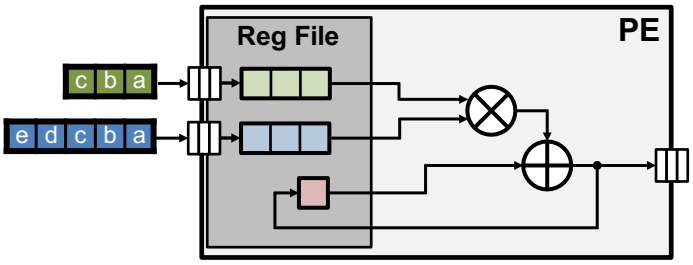


82

82

1D Row Convolution in PE

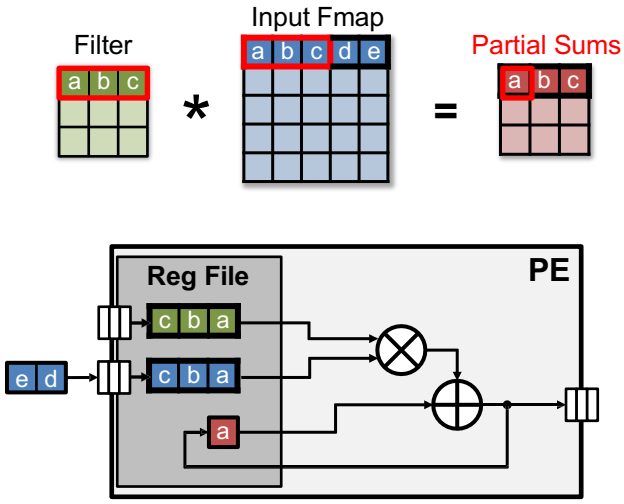




83

83

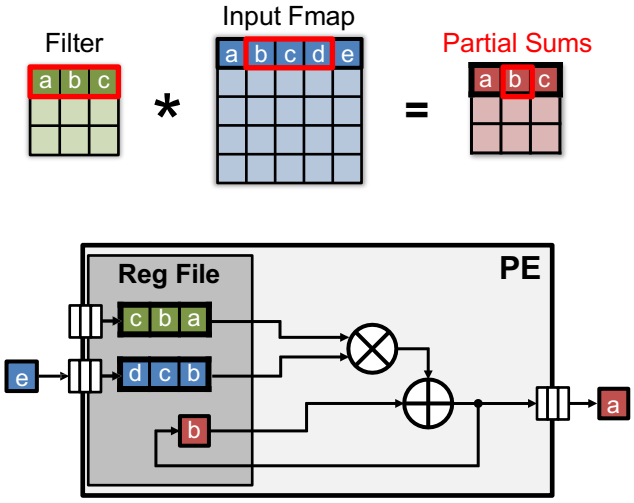
1D Row Convolution in PE



84

84

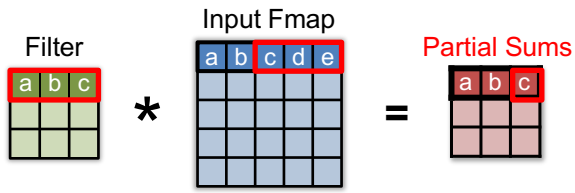
1D Row Convolution in PE

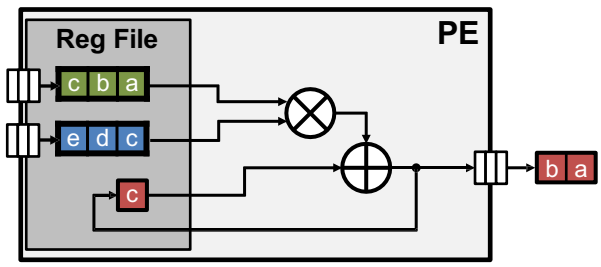


85

85

1D Row Convolution in PE



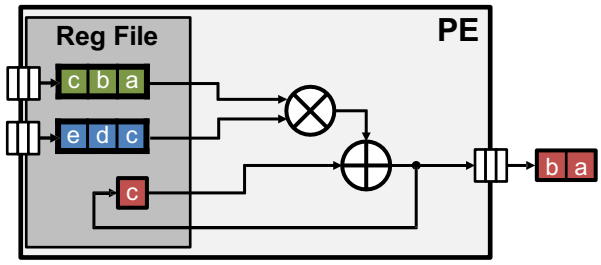


86

86

1D Row Convolution in PE

- Maximize row **convolutional reuse** in RF
  - Keep a **filter** row and **fmap** sliding window in RF
- Maximize row **psum accumulation** in RF



87

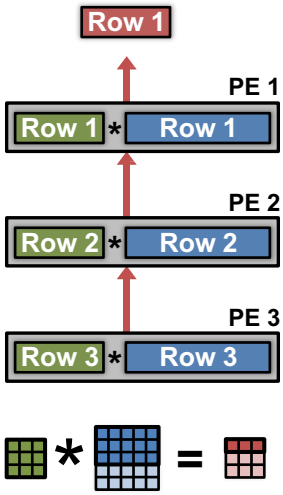
87

2D Convolution in PE Array

88

88

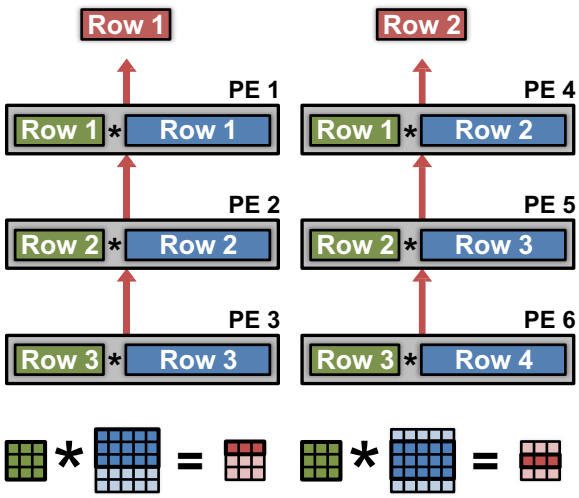
2D Convolution in PE Array



89

89

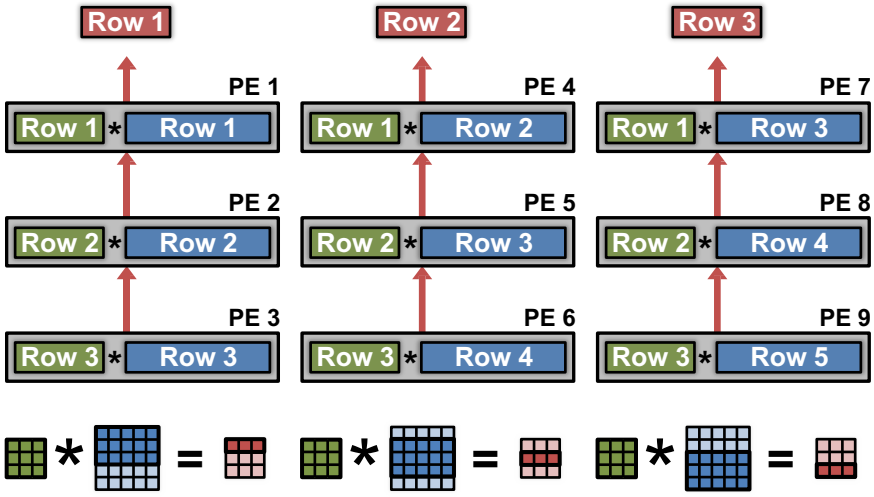
2D Convolution in PE Array



90

90

2D Convolution in PE Array



91

91

Dimensions Beyond 2D Convolution

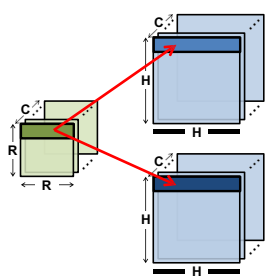
1 Multiple Fmaps 2 Multiple Filters 3 Multiple Channels

92

92

Filter Reuse in PE

1 Multiple Fmaps Multiple Filters Multiple Channels



	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1

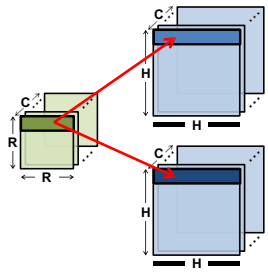
	Filter 1		Fmap 2		Psum 2
Channel 1	Row 1	*	Row 1	=	Row 1

93

93

Filter Reuse in PE

1 Multiple Fmaps Multiple Filters Multiple Channels



	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1
	Filter 1		**Fmap 2**		**Psum 2**
Channel 1	Row 1	*	Row 1	=	Row 1

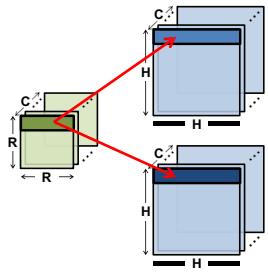
share the same filter row

94

94

Filter Reuse in PE

1 Multiple Fmaps Multiple Filters Multiple Channels



	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1
	Filter 1		**Fmap 2**		**Psum 2**
Channel 1	Row 1	*	Row 1	=	Row 1

share the same filter row

Processing in PE: **concatenate fmap rows**

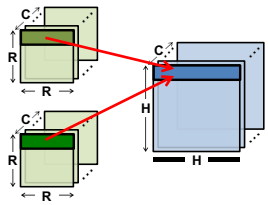
	Filter 1		Fmap 1 & 2		Psum 1 & 2
Channel 1	Row 1	*	Row 1 Row 1	=	Row 1 Row 1

95

95

Fmap Reuse in PE

Multiple Fmaps | **2 Multiple Filters** | Multiple Channels



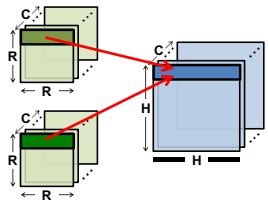
	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1
	Filter 2		**Fmap 1**		**Psum 2**
Channel 1	Row 1	*	Row 1	=	Row 1

96

96

Fmap Reuse in PE

Multiple Fmaps | **2 Multiple Filters** | Multiple Channels



	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1
	Filter 2		**Fmap 1**		**Psum 2**
Channel 1	Row 1	*	Row 1	=	Row 1

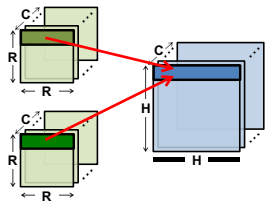
share the same fmap row

97

97

Fmap Reuse in PE

Multiple Fmaps ② **Multiple Filters** Multiple Channels



	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1
	Filter 2		**Fmap 1**		**Psum 2**
Channel 1	Row 1	*	Row 1	=	Row 1

share the same fmap row

Processing in PE: **interleave filter rows**

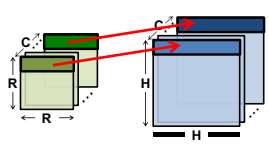
	Filter 1 & 2		Fmap 1		Psum 1 & 2
Channel 1		*	Row 1	=	

98

98

Channel Accumulation in PE

Multiple Fmaps Multiple Filters ③ **Multiple Channels**



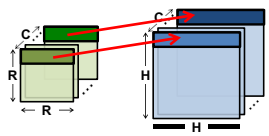
	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1
	Filter 1		**Fmap 1**		**Psum 1**
Channel 2	Row 1	*	Row 1	=	Row 1

99

99

Channel Accumulation in PE

Multiple Fmaps Multiple Filters ③ **Multiple Channels**



	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1
Channel 2	Row 1	*	Row 1	=	Row 1

accumulate psums

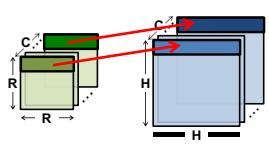
Row 1 + Row 1 = Row 1

100

100

Channel Accumulation in PE

Multiple Fmaps Multiple Filters ③ **Multiple Channels**



	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1
Channel 2	Row 1	*	Row 1	=	Row 1

accumulate psums

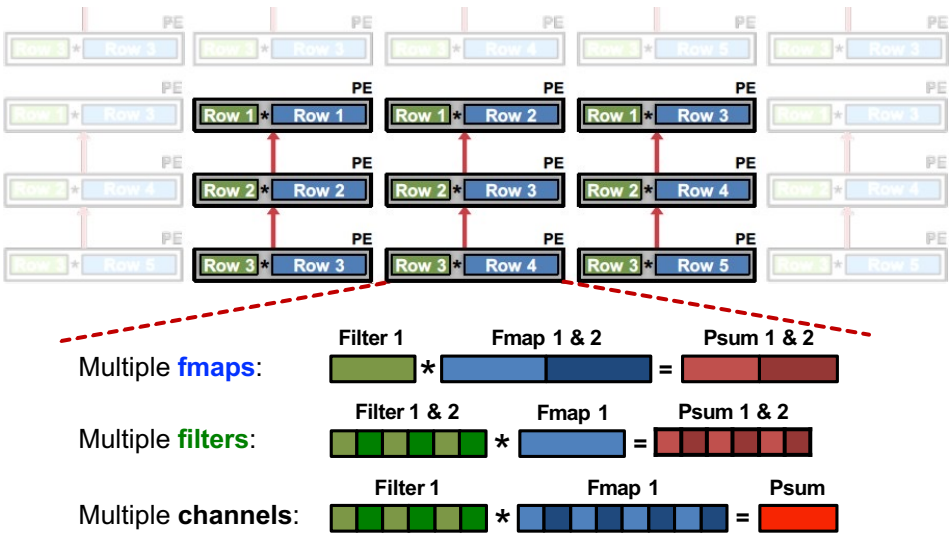
<u>Processing in PE</u>: **interleave channels**

	Filter 1		Fmap 1		Psum
Channel 1 & 2		*		=	Row 1

101

101

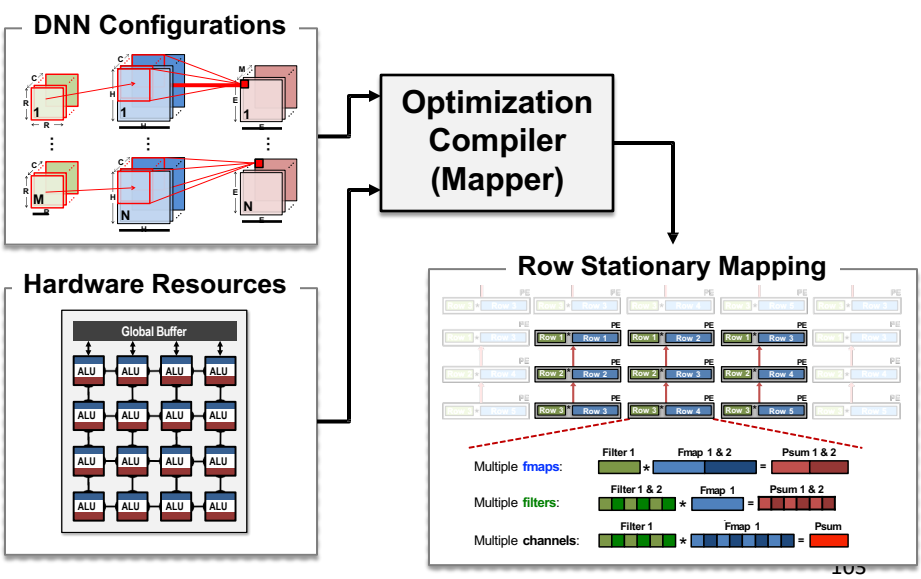
DNN Processing – The Full Picture



Map rows from **multiple fmaps**, **filters** and **channels** to same PE to exploit other forms of reuse and local accumulation

102

Optimal Mapping in Row Stationary



[Chen et al., ISCA 2016]

103

Other Optimizations?

- Data layout transformation
  - Transform how data is stored in the linear memory
  - Compiler job
- Processing in memory (PIM)
  - Leverage the structure of memory to perform the computation
- Approximate computing
  - Quantization
  - Pruning

104

104