

Kernel Computation

Instructor: Siting Liu

School of Information Science and Technology (SIST), ShanghaiTech University

Shanghai Institute of Semiconductor Innovation (SISI)

<https://faculty.sist.shanghaitech.edu.cn/liust/EE219/EE219.html>



Computer Architecture - EE219
faculty.sist.shanghaitech.edu.cn

立志成才 报国裕民



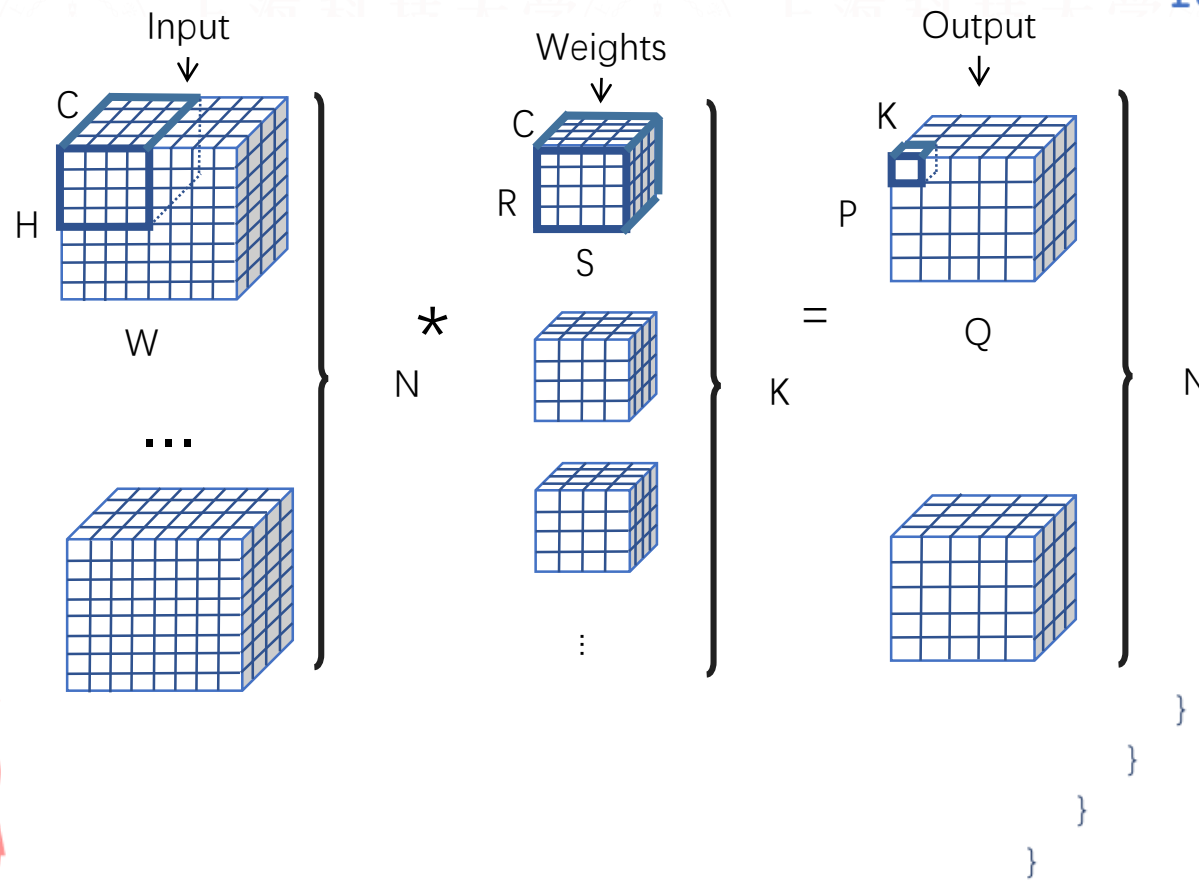
Review

- Schemes that reduce model size, latency and energy consumption
- What is next?
 - Implement the computations of neural networks
 - Kernel computations
 - Processing elements
 - Dataflow



Kernel Computation - Review

• Convolution



```

for (n=0; n<N; n++) {
    for (k=0; k<K; k++) {
        for (p=0; p<P; p++) {
            for (q=0; q<Q; q++) {
                OA[n][k][p][q] = 0;
                for (r=0; r<R; r++) {
                    for (s=0; s<S; s++) {
                        for (c=0; c<C; c++) {
                            h = p * stride - pad + r;
                            w = q * stride - pad + s;
                            OA[n][k][p][q] +=
                                IA[n][c][h][w]
                                * W[k][c][r][s];
                        }
                    }
                }
                OA[n][k][p][q] = Activation(OA[n][k][p][q]);
            }
        }
    }
}
    
```

for each output activation

convolution window

+ bias

Kernel Computation - Review

- Convolution
 - Nested loop or direct convolution (just covered)
 - Im2col $\rightarrow$ GEMM
 - FFT/iFFT
 - Winograd fast filtering algorithm

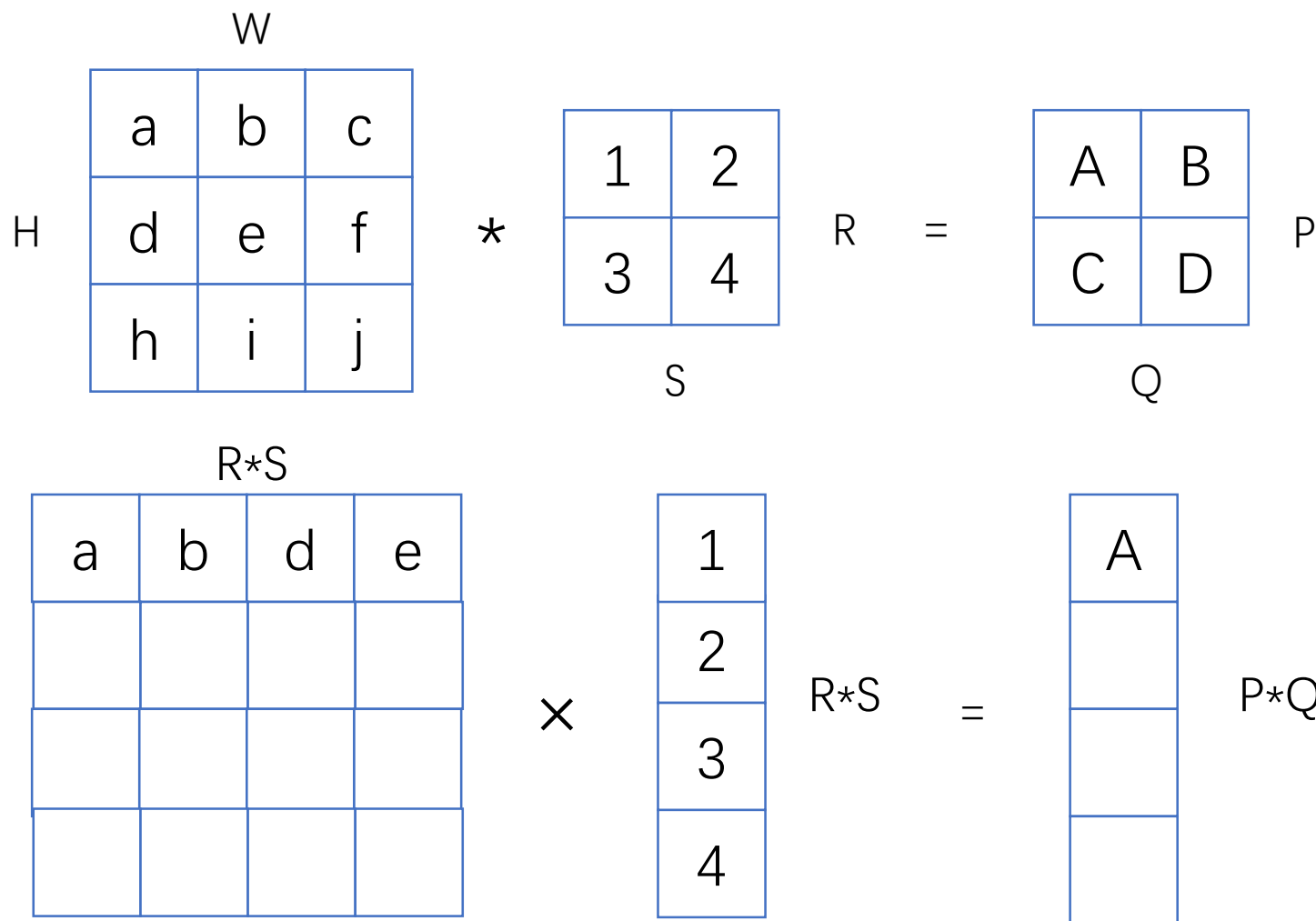


im2col

Conv.



GEMM

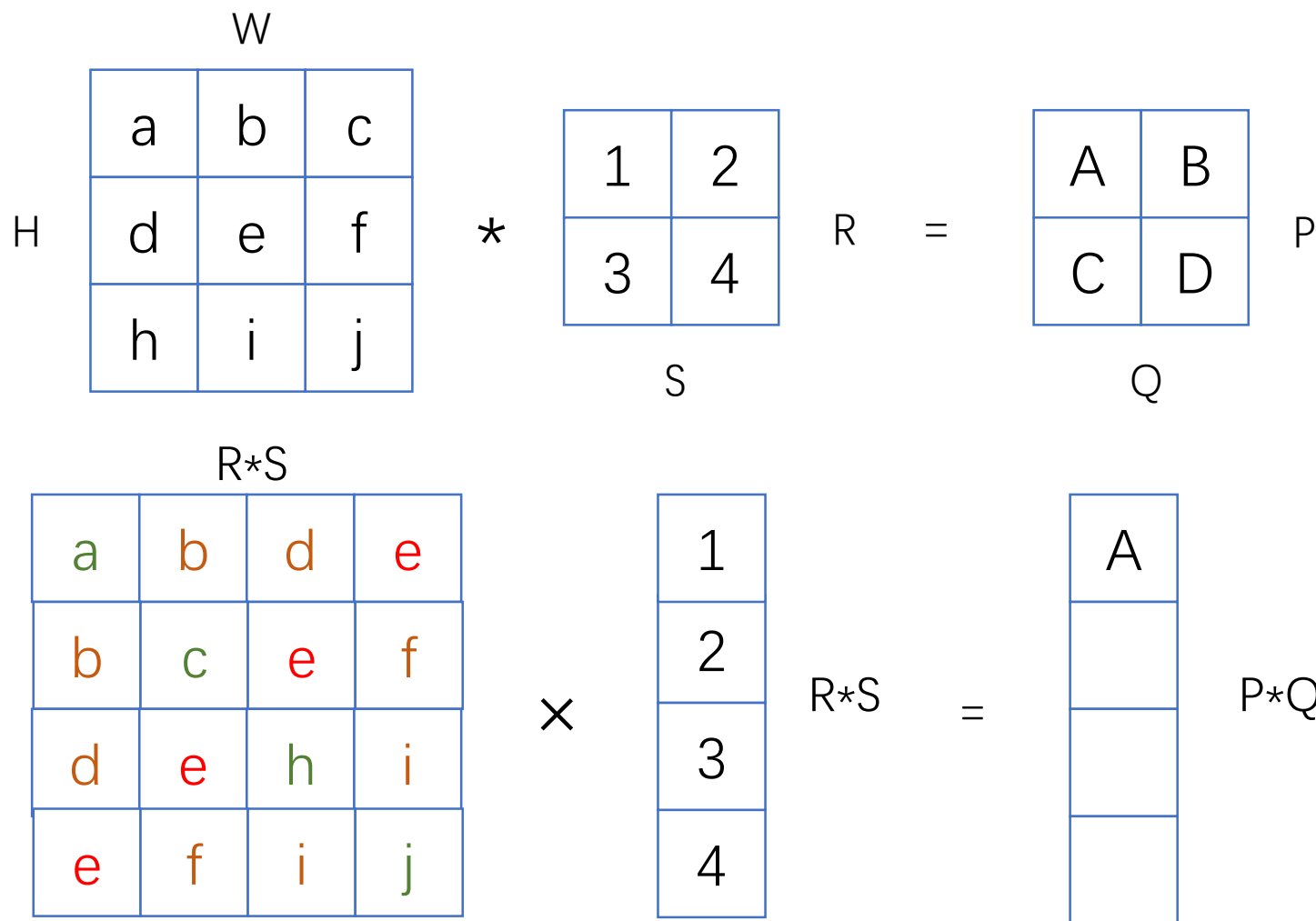


im2col (Cont'd)

Conv.



GEMM



Exercise

Conv.



GEMM

$$\begin{array}{c} H \\ \begin{array}{|c|c|c|c|} \hline a & b & c & d \\ \hline e & f & g & h \\ \hline i & j & k & l \\ \hline m & n & o & p \\ \hline \end{array} \end{array} \begin{array}{c} W \\ * \\ \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} \end{array} \begin{array}{c} S \\ R \\ = \\ \begin{array}{|c|c|} \hline A & B \\ \hline C & D \\ \hline \end{array} \end{array} \begin{array}{c} P \\ Q \end{array}$$

Stride = 2

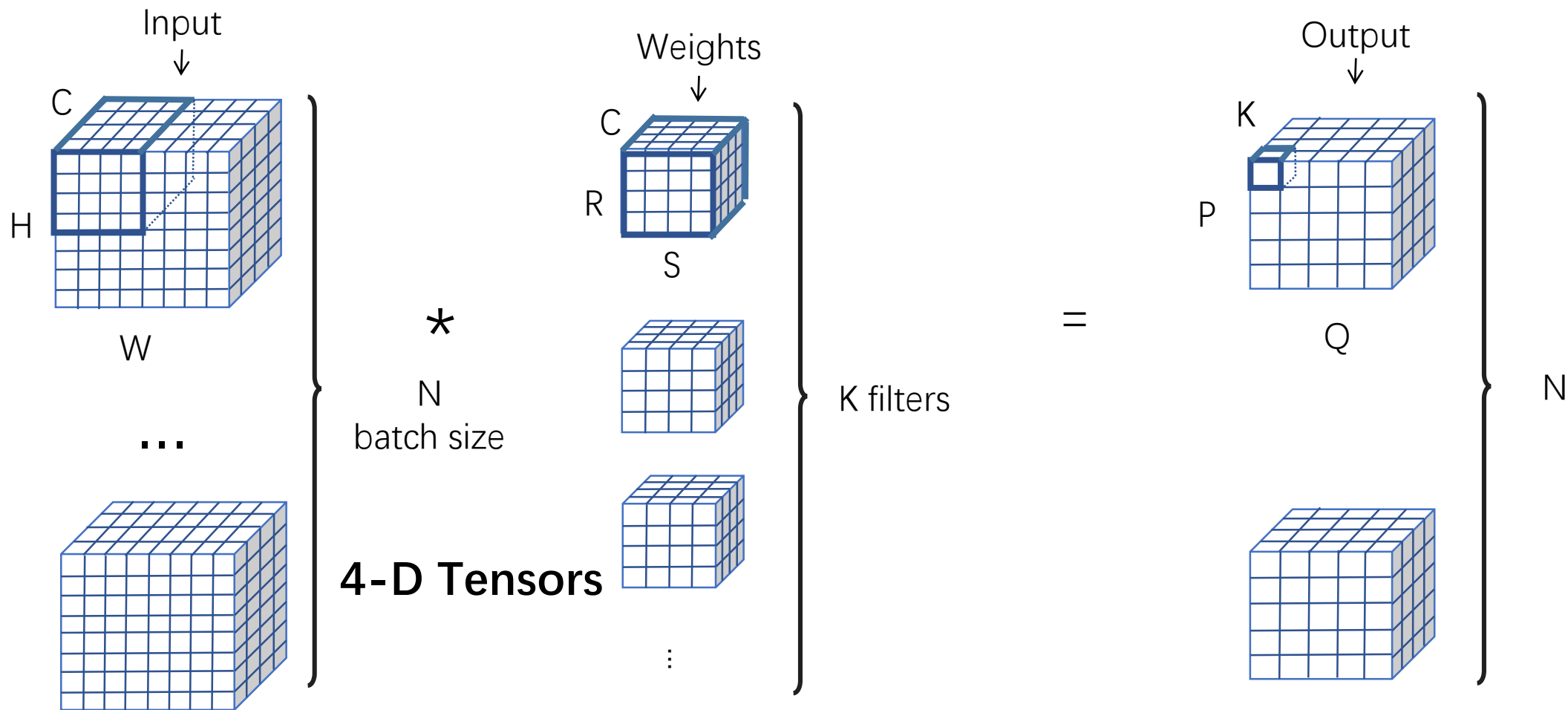
Dilation = 2

What about 3D inputs?

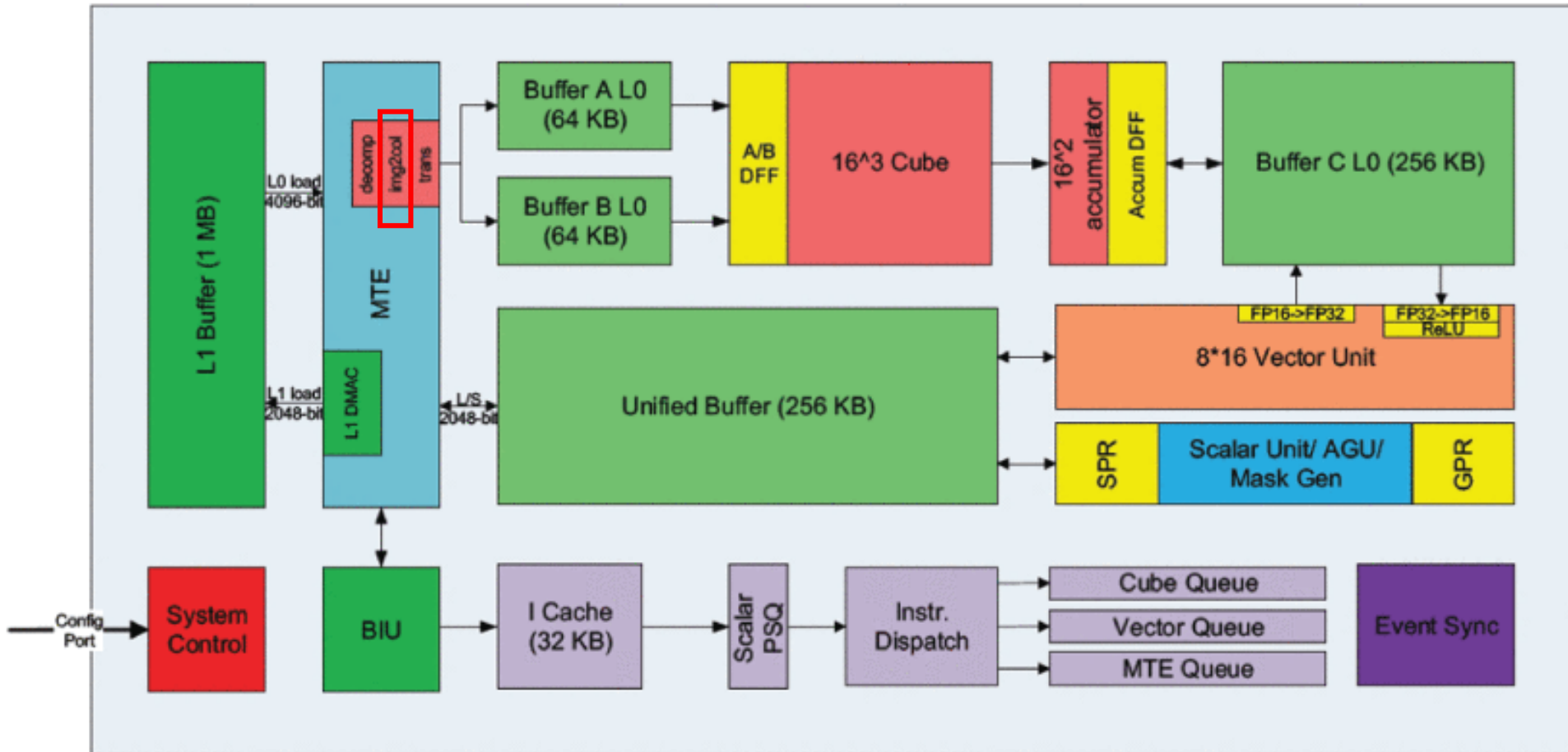
Require col2im as
post-processing



Memory Access



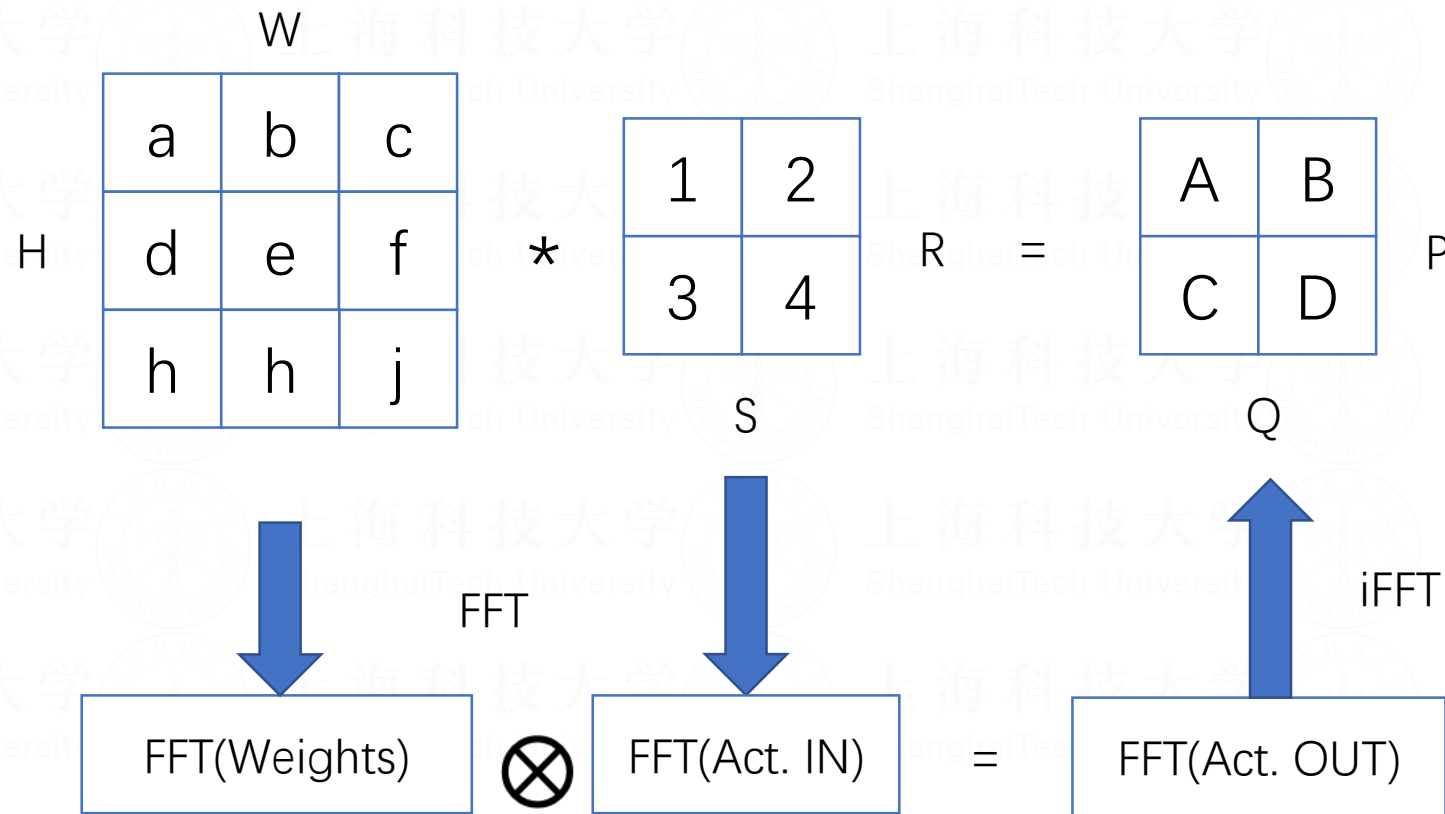
Explicit im2col: Ascend AI Processor Case Study



H. Liao et al., "Ascend: a Scalable and Unified Architecture for Ubiquitous Deep Neural Network Computing : Industry Track Paper," 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021, pp. 789-801.

FFT-based Conv. (Obsolete)

- Time-domain Conv. is equivalent to frequency-domain point-wise multiplication



FFT-based Conv. (Obsolete and Why?)

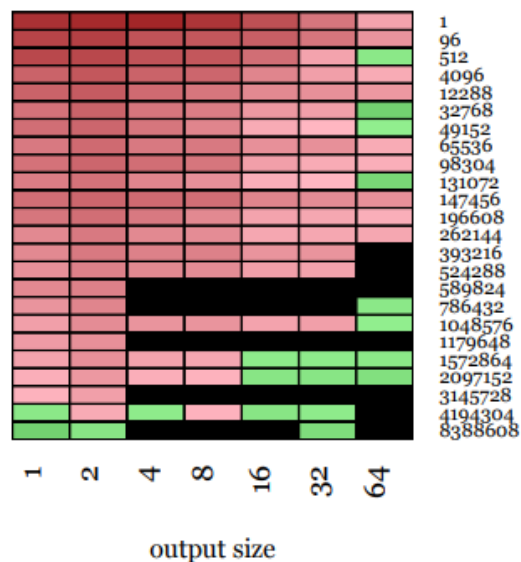


Figure 1: 3×3 kernel (K40m)

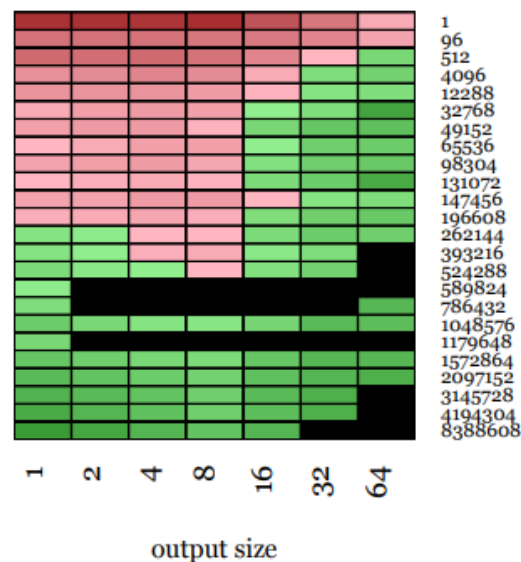


Figure 2: 5×5 kernel (K40m)

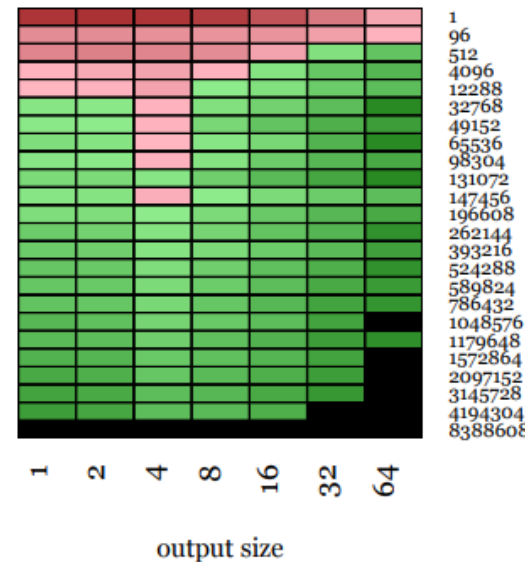


Figure 3: 7×7 kernel (K40m)

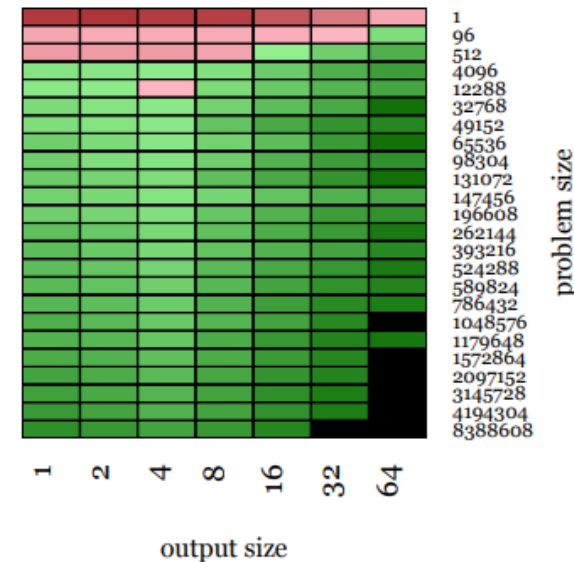
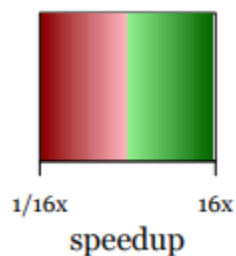


Figure 4: 9×9 kernel (K40m)



- Favor larger kernels/inputs
- FFT/iFFT overhead
- Not easy to do “stride” or “padding”

Winograd Transform

- Transform to reduce multiplications (supported in some software package/hardware, e.g., NVDLA, full version)
- Fast FIR filter $\mu(F(m, r))$ requires $m + r - 1$ multiplications.

- $F(2,3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$
2-output 3-tap

$$m_1 = (d_0 - d_2)g_0$$

$$m_4 = (d_1 - d_3)g_2$$

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

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

Winograd Transform (Cont'd)

- Transform to reduce multiplications
- **In matrix form:** $Y = A^T [(Gg) \otimes (B^T d)]$



Element-wise multiplication

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

$$\begin{aligned} m_1 &= (d_0 - d_2)g_0 & m_2 &= (d_1 + d_2) \frac{g_0 + g_1 + g_2}{2} \\ m_4 &= (d_1 - d_3)g_2 & m_3 &= (d_2 - d_1) \frac{g_0 - g_1 + g_2}{2} \end{aligned}$$

Transform matrix

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & -0.5 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

Winograd Transform for 2D Convolution

- 2D expansion: $\mu(F(m \times n, r \times s))$ requires $(m + r - 1)(n + s - 1)$ multiplications.

- Nested with itself: $Y = A^T[(GgG^T) \otimes (B^T dB)]A$

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

$$F(2 \times 2, 3 \times 3) = \begin{matrix} & \begin{matrix} X_0 & & X_1 & & X_2 \end{matrix} & \\ \begin{matrix} X_1 \\ X_2 \\ X_3 \end{matrix} & \begin{bmatrix} x_0 & x_1 & x_2 & | & x_4 & x_5 & x_6 & | & x_8 & x_9 & x_{10} \\ x_1 & x_2 & x_3 & | & x_5 & x_6 & x_7 & | & x_9 & x_{10} & x_{11} \\ x_4 & x_5 & x_6 & | & x_8 & x_9 & x_{10} & | & x_{12} & x_{13} & x_{14} \\ x_5 & x_6 & x_7 & | & x_9 & x_{10} & x_{11} & | & x_{13} & x_{14} & x_{15} \end{bmatrix} & \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \end{bmatrix} \end{matrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \begin{matrix} Y_0 \\ Y_1 \end{matrix}$$

x_0	x_1	x_2	x_3
x_4	x_5	x_5	x_7
x_8	x_9	x_{10}	x_{11}
x_{12}	x_{13}	x_{14}	x_{15}

*

w_0	w_1	w_2
w_3	w_4	w_5
w_6	w_7	w_8

Winograd Transform for 2D Convolution (Cont'd)

$$F(2 \times 2, 3 \times 3) = \begin{array}{c} \begin{array}{ccc} & X_0 & \\ & & X_1 \\ & & & X_2 \end{array} \\ \begin{bmatrix} x_0 & x_1 & x_2 & | & x_4 & x_5 & x_6 & | & x_8 & x_9 & x_{10} \\ x_1 & x_2 & x_3 & | & x_5 & x_6 & x_7 & | & x_9 & x_{10} & x_{11} \\ x_4 & x_5 & x_6 & | & x_8 & x_9 & x_{10} & | & x_{12} & x_{13} & x_{14} \\ x_5 & x_6 & x_7 & | & x_9 & x_{10} & x_{11} & | & x_{13} & x_{14} & x_{15} \end{bmatrix} \end{array} \begin{array}{c} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \end{bmatrix} \begin{array}{l} W_0 \\ \\ \\ W_1 \\ \\ \\ W_2 \end{array} \\ \\ \\ \end{array} = \begin{array}{c} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \\ \\ \\ \end{array} \begin{array}{c} Y_0 \\ \\ Y_1 \end{array}$$

$$F(2,3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix} \quad Y = A^T [(Gg) \otimes (B^T d)]$$

$$F(2 \times 2, 3 \times 3) = \begin{bmatrix} X_0 W_0 + X_1 W_1 + X_2 W_2 \\ X_1 W_0 + X_2 W_1 + X_3 W_1 \end{bmatrix}$$

$$= \begin{bmatrix} A^T [(GW_0) \otimes (B^T X_0)] + A^T [(GW_1) \otimes (B^T X_1)] + A^T [(GW_2) \otimes (B^T X_2)] \\ A^T [(GW_0) \otimes (B^T X_1)] + A^T [(GW_1) \otimes (B^T X_2)] + A^T [(GW_2) \otimes (B^T X_3)] \end{bmatrix}$$



Winograd Transform for 2D Convolution: Example

$$Y = \underbrace{A^T}_{\text{Output Transform}} \underbrace{[GWG^T]}_{\text{Filter Transform}} \underbrace{\otimes (B^T X B)}_{\text{Input Transform}} A$$

Transform matrix

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & -0.5 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 7 & 8 \\ 7 & 6 & 5 & 4 \\ 3 & 2 & 1 & 0 \end{bmatrix}$$

$$W = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

x_0	x_1	x_2	x_3
x_4	x_5	x_5	x_7
x_8	x_9	x_{10}	x_{11}
x_{12}	x_{13}	x_{14}	x_{15}

X

w_0	w_1	w_2
w_3	w_4	w_5
w_6	w_7	w_8

W

*



Winograd Implementations

- GWG^T can be pre-calculated and stored to further reduce the computations in NN inference
 - But it requires more storage & bandwidth (9-element $\rightarrow$ 16-element)
- Requires careful control of the dataflow
- Trade-offs with different m ($F(m \times m, 3 \times 3)$), with different G , B and A matrices (<https://github.com/andravin/wincnn>)



Summary

- Algorithm side
 - We analyze some popular NNs and find out the computing primitives
 - GEMM & conv. occupy most of the computations
 - Conv. can be turned into GEMM problems
 - Transforms such as Winograd can be used to reduce multiplications



Processing Elements and Datapath

Instructor: Siting Liu

School of Information Science and Technology (SIST), ShanghaiTech University

Shanghai Institute of Semiconductor Innovation (SISI)

<https://faculty.sist.shanghaitech.edu.cn/liust/EE219/EE219.html>



Computer Architecture - EE219
faculty.sist.shanghaitech.edu.cn

立志成才 报国裕民



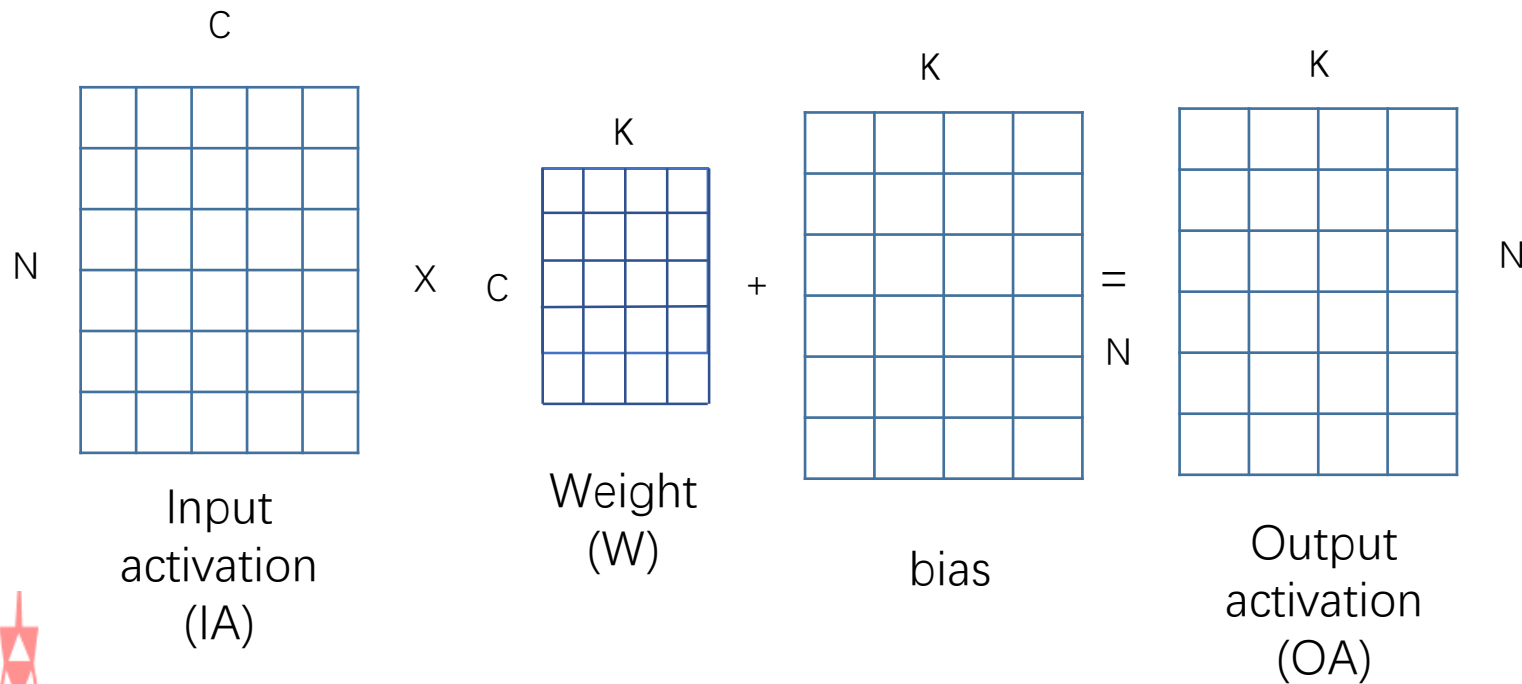
Review

- Algorithm analysis
- Model Compression
- Kernel computation
- Mainly matrix multiplication still
- Datapath design (processing elements & memory consideration)



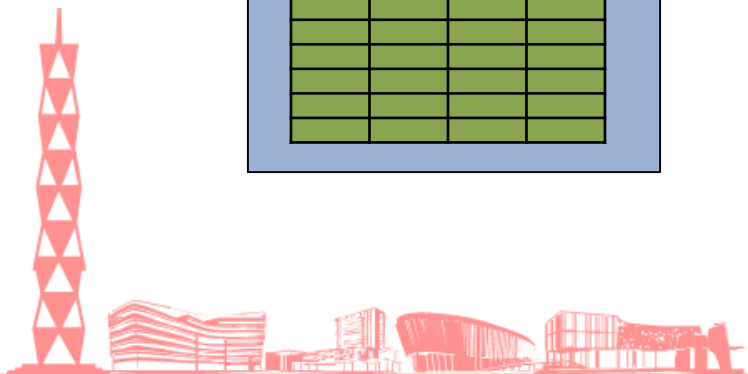
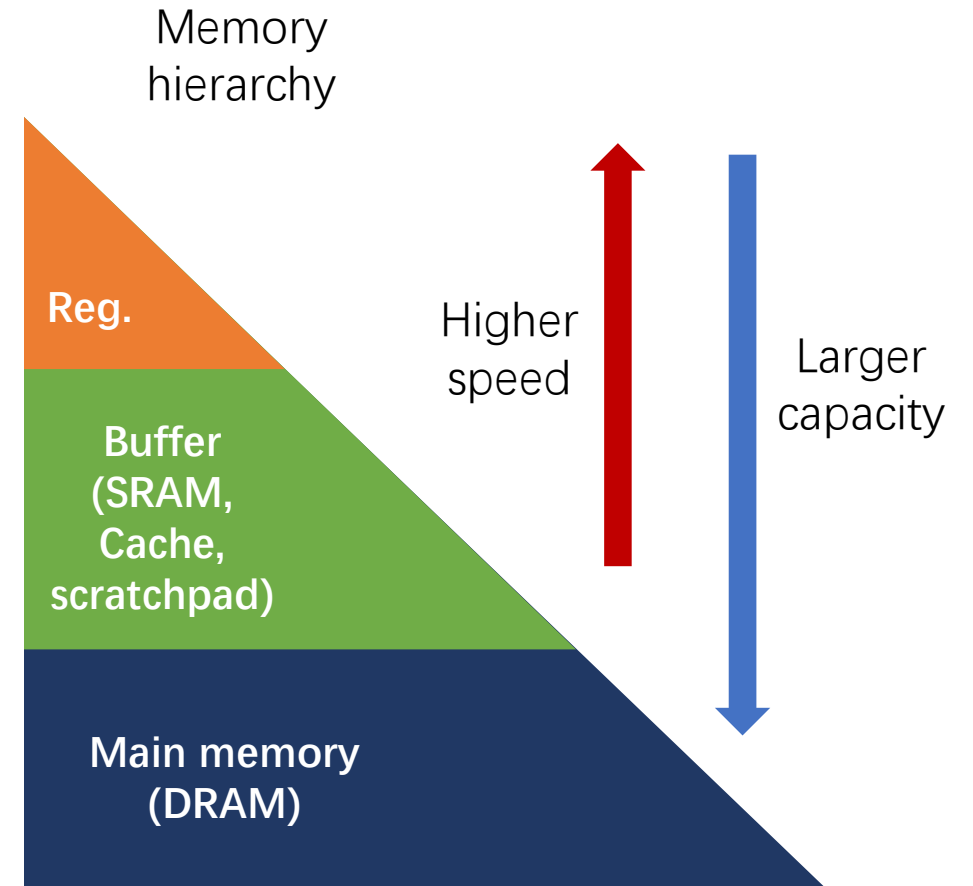
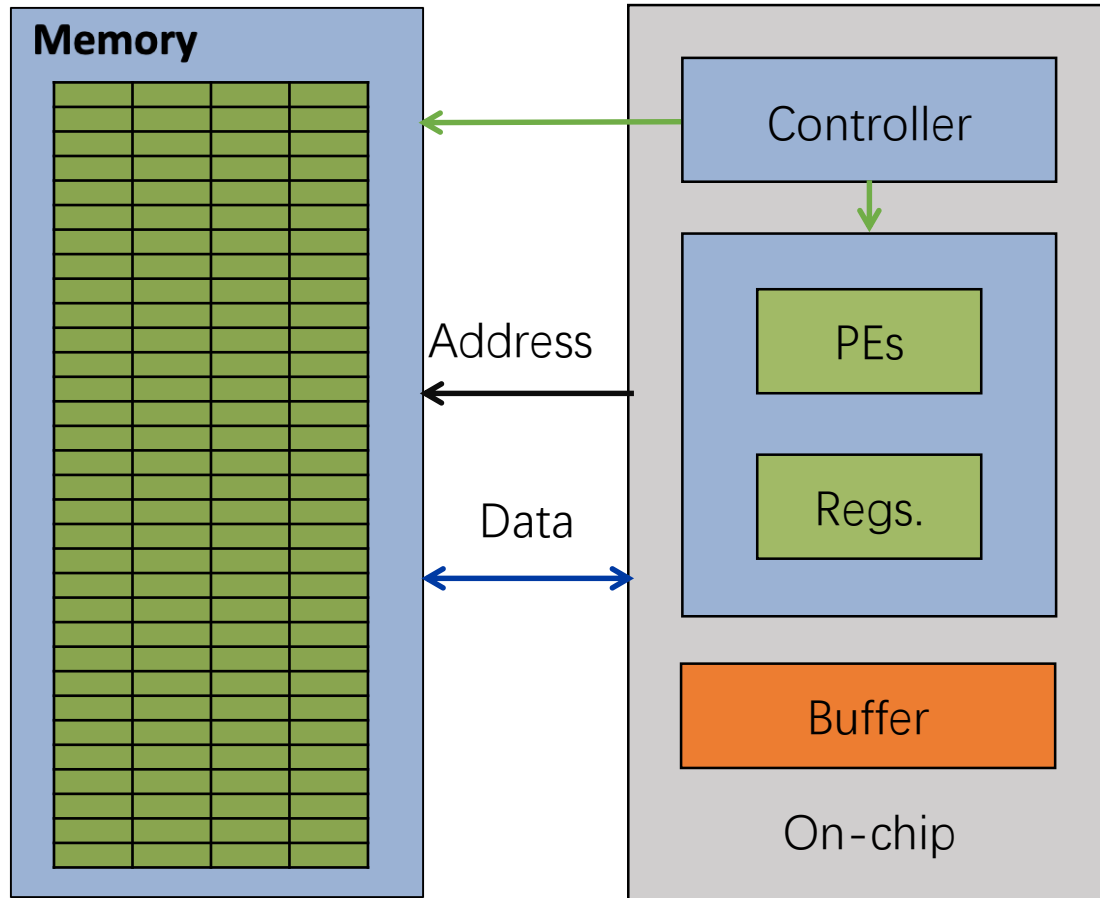
Targeting the Common Cases

- GEMM



```
for (n=0; n<N; n++) {  
    for (k=0; k<K; k++){  
        OA[n, k] = b[n, k];  
        for (c=0; c<C; c++){  
            OA[n, k] = OA[n, k] + IA[n, c]*W[c, k];  
        }  
        OA[n, k] = Act(OA[n, k]);  
    }  
}
```

General Digital System (Simplified)



Datapath – Basic PE

```
for (m=0; m<M; m++) {  
  for (n=0; n<N; n++){  
    OA[m, n] = b[m, n];  
    for (k=0; k<K; k++){  
      OA[m, n] = OA[m, n] + X[m, k]*W[k, n];  
    }  
    OA[m, n] = Act(OA[m, n]);  
  }  
}
```



$$C += A * B$$



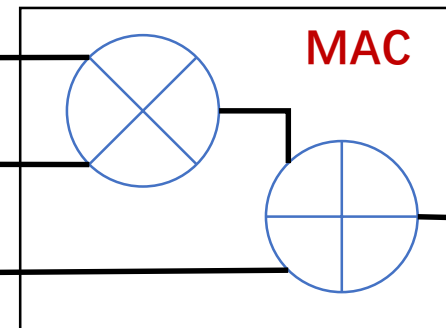
Memory read

Regs.

Filter weight

Fmap/activation

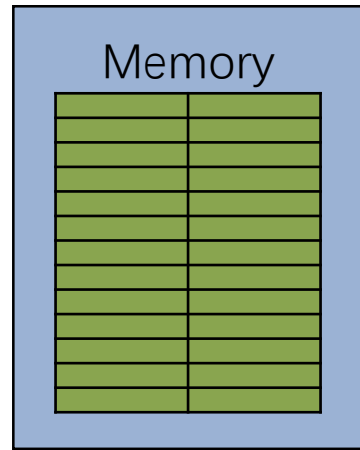
Partial sum



Memory write/read
Updated **partial sum**

Output-stationary Dataflow

- Maximize output reuse



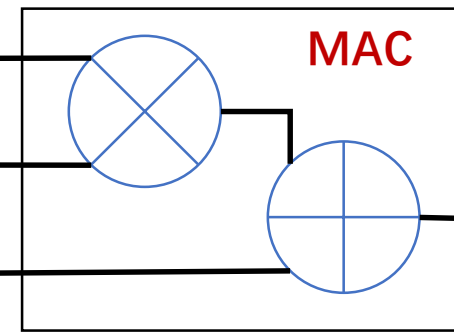
Memory read



Filter weight

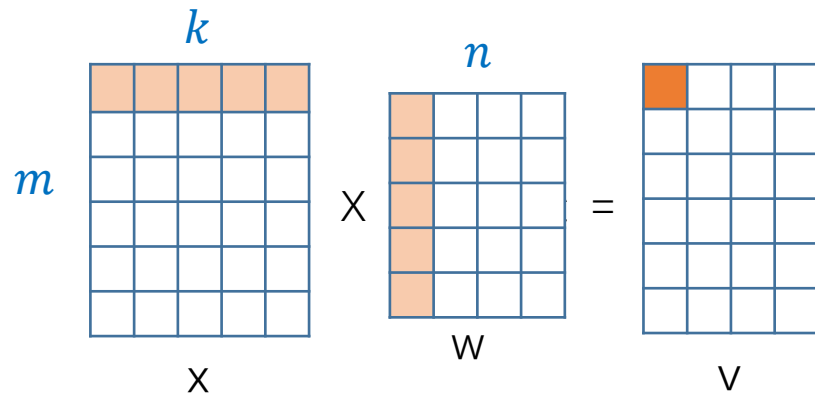
Fmap/activation

Partial sum



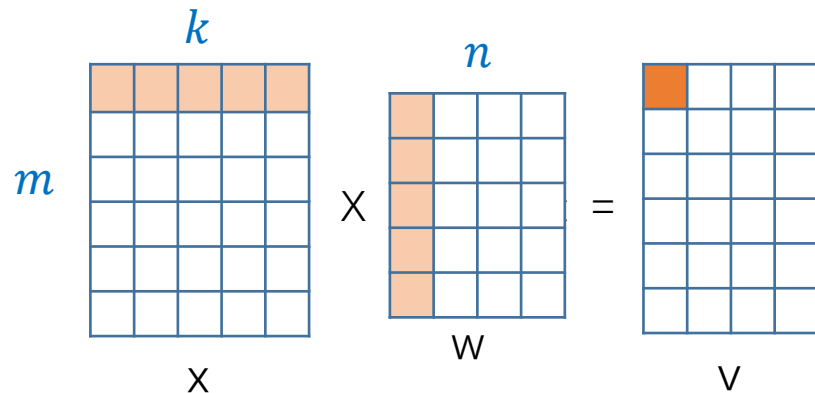
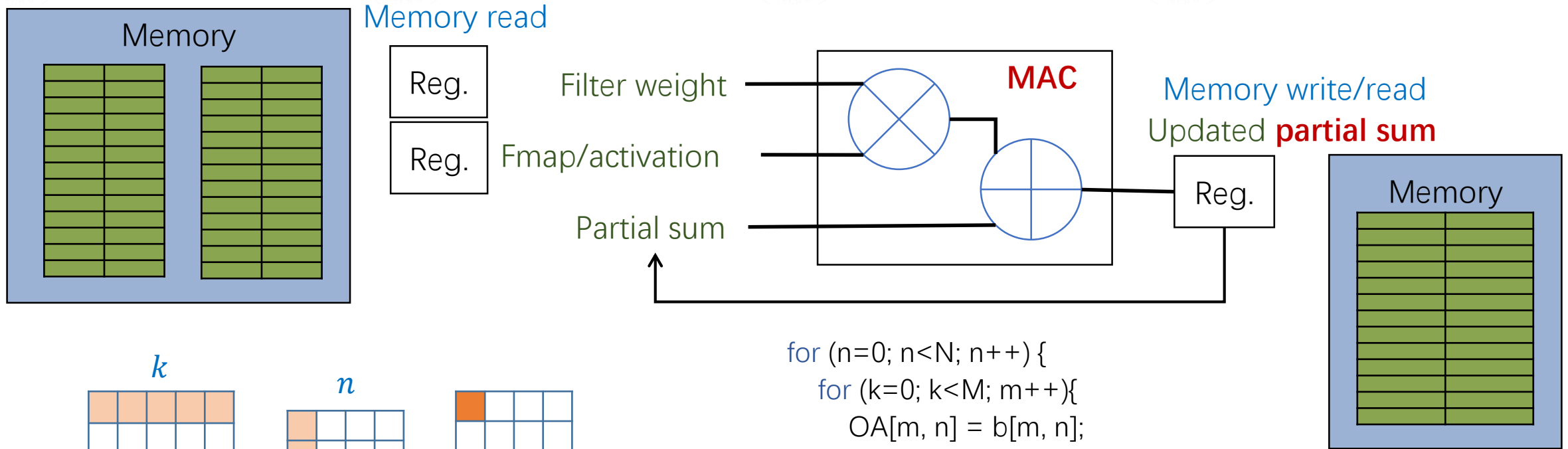
Memory write/read

Updated **partial sum**



Output-stationary Dataflow (Cont'd)

- Maximize output reuse

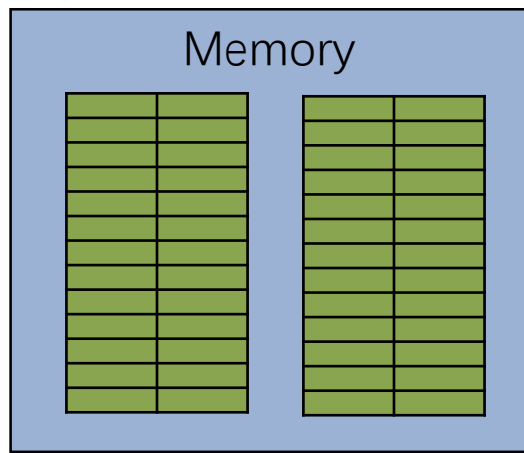


```

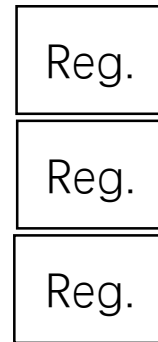
for (n=0; n<N; n++) {
  for (k=0; k<M; k++) {
    OA[m, n] = b[m, n];
    for (k=0; k<K; k++) {
      OA[m, n] = OA[m, n] + X[m, k]*W[k, n];
    }
    OA[m, n] = Act(OA[m, n]);
  }
}
    
```

Weight-stationary Dataflow

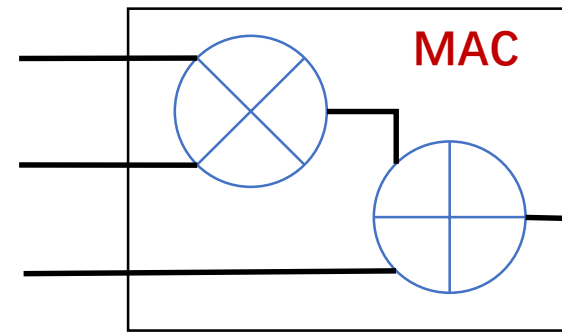
- Maximize weight reuse



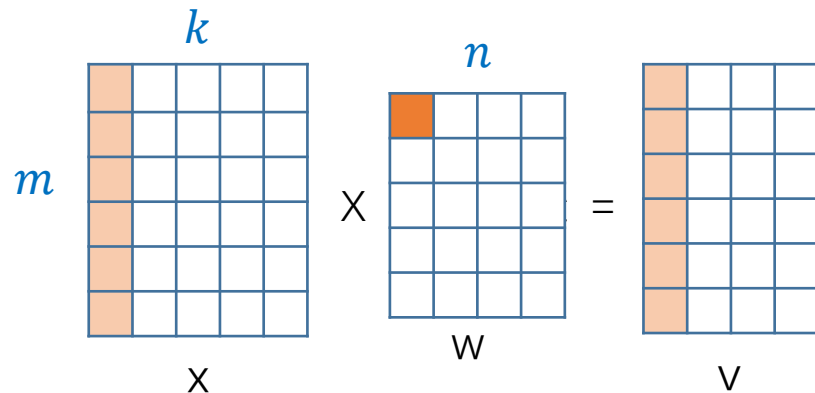
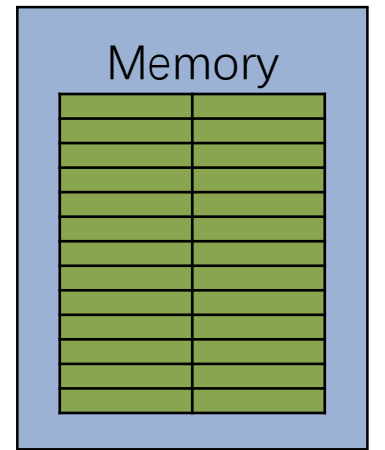
Memory read



Filter weight
Fmap/activation
Partial sum



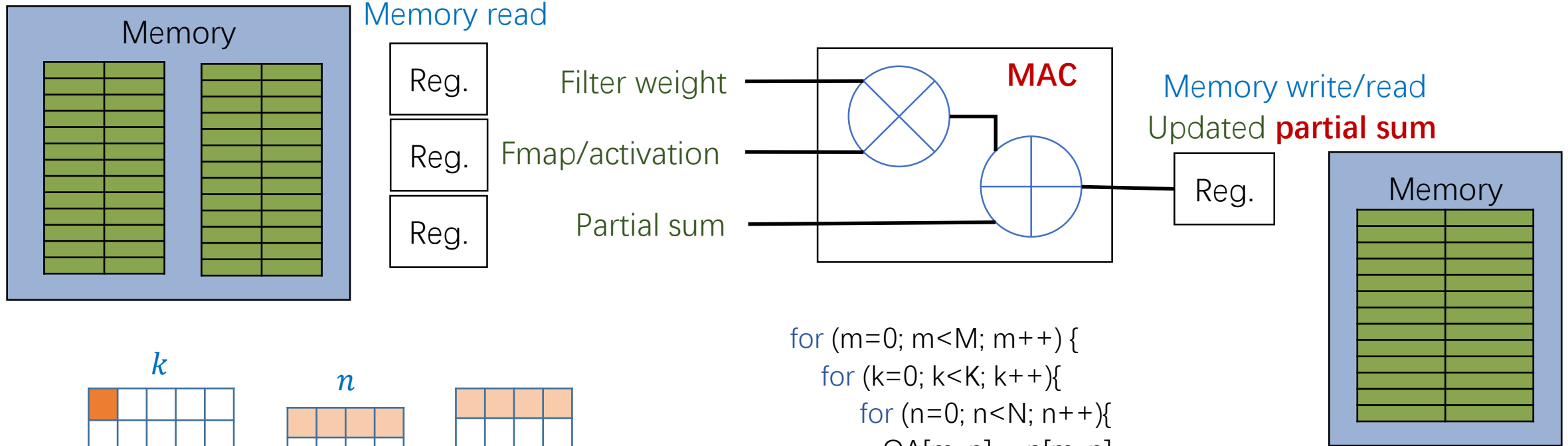
Memory write/read
Updated **partial sum**



```
for (n=0; n<N; n++) {
  for (k=0; k<K; k++){
    for (m=0; m<M; m++){
      OA[m, n] = p[m, n];
      OA[m, n] = OA[m, n]+ X[m, k]*W[k, n];
    }
    OA[m, n] = Act(OA[m, n]);
  }
}
```

Input-stationary Dataflow

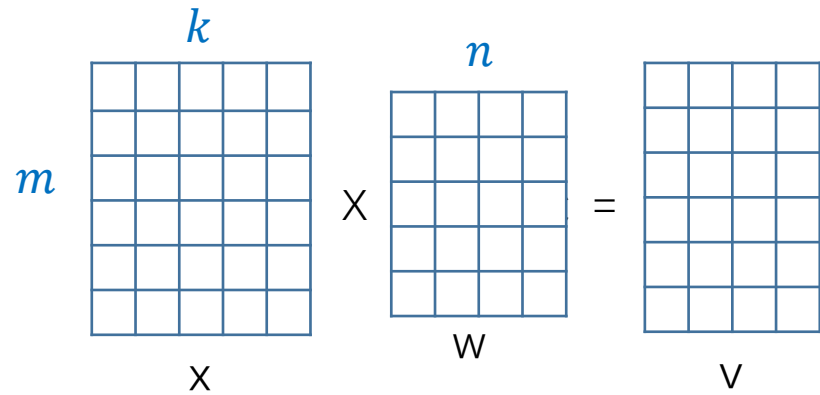
- Maximize input reuse



```

for (m=0; m<M; m++) {
  for (k=0; k<K; k++){
    for (n=0; n<N; n++){
      OA[m, n] = p[m, n];
      OA[m, n] = OA[m, n]+ X[m, k]*W[k, n];
    }
    OA[m, n] = Act(OA[m, n]);
  }
}
    
```

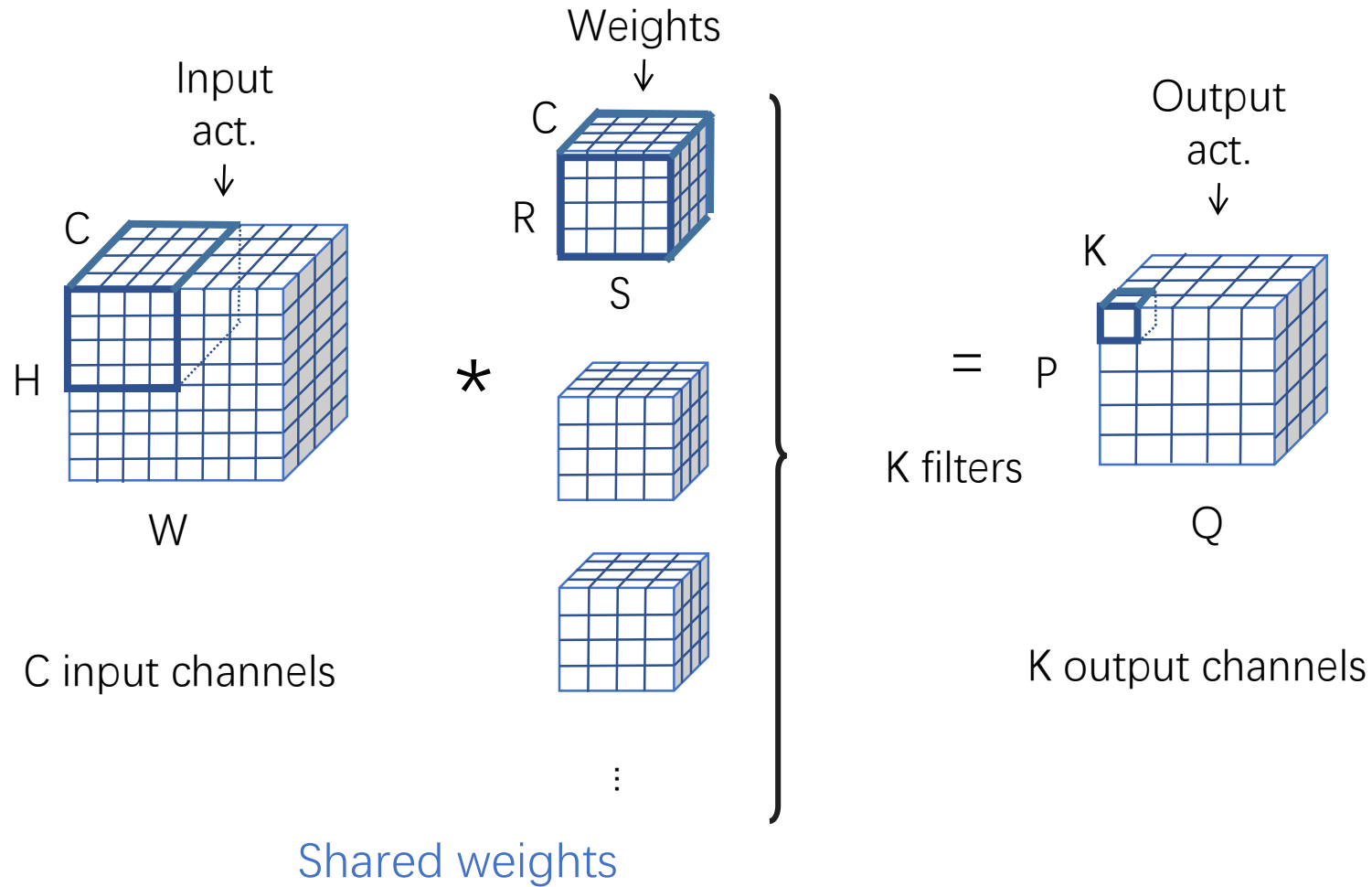
Data Reuse in GEMM



- Total MACs
- Input activations
- Weight
- Output Activations



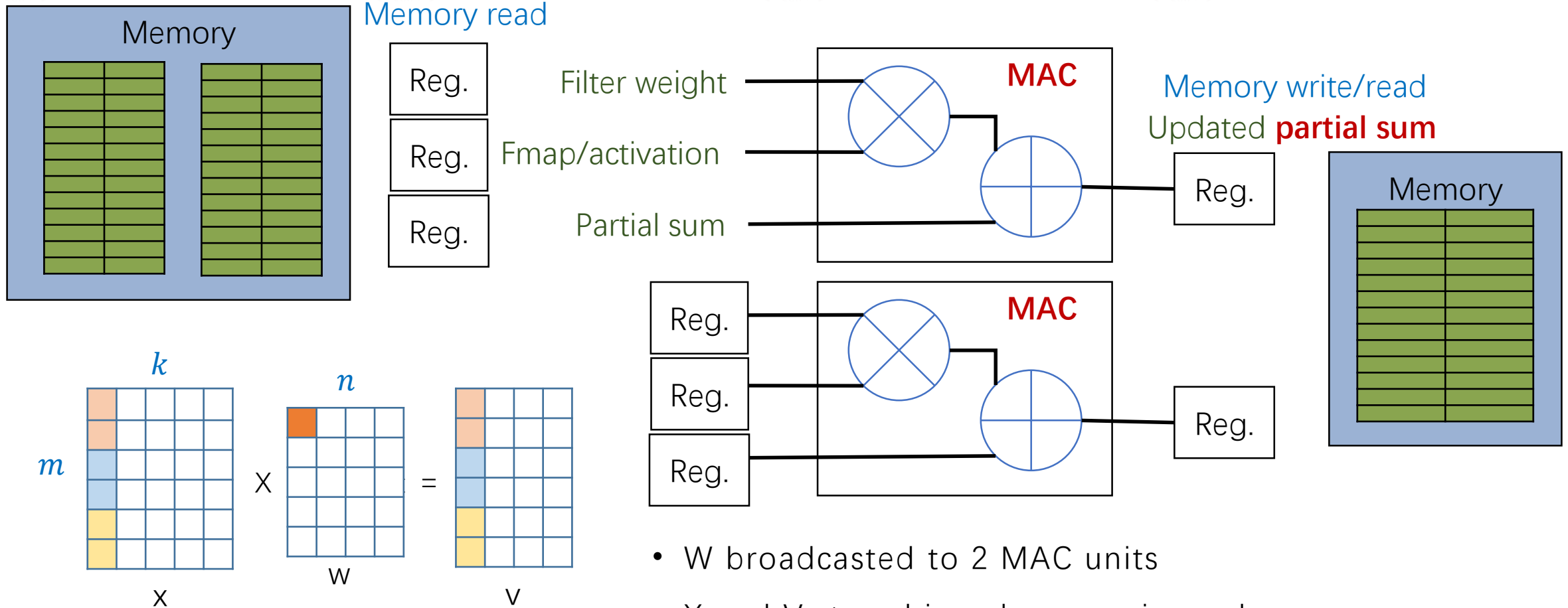
Data Reuse in Convolution



- Total MACs
- Input activations
- Weight
- Output Activations

Parallelization

- Weight-stationary for example



- W broadcasted to 2 MAC units
- X and V stored in column-major order

Summary

- Matrix multiplication can be implemented with basic PE or a MAC unit;
- Different dataflow (order of execution/for-loop) change the number of memory access, and it affects the energy cost and execution time;
- Increase the number of PEs (parallelism) improve the performance with ideal data layout and sufficient bandwidth.

