



CMSC 5743

Efficient Computing of Deep Neural Networks

Implementation 02: Direct Conv

Bei Yu

CSE Department, CUHK

byu@cse.cuhk.edu.hk

(Latest update: January 24, 2026)

2026 Spring

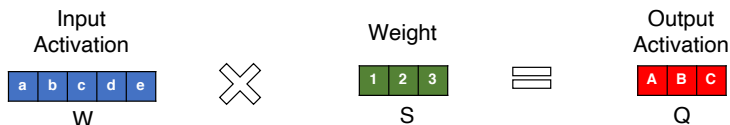


- ① Loop Reordering
- ② Direct Convolution
- ③ Dataflow Optimization



Loop Reordering

1D Convolution Example



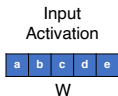
```
for(q=0; q<Q; q++){  
  for (s=0; s<S; s++){  
    OA[q] += IA[q+s] * W[s];  
  }  
}
```

**Output Stationary (OS)
Dataflow**

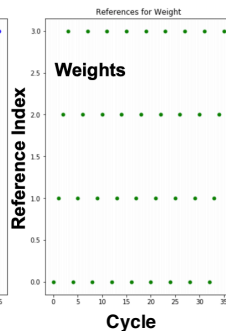
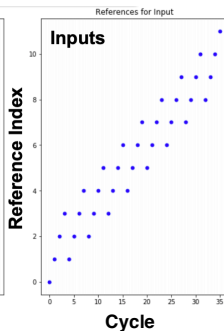
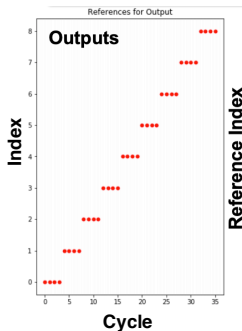
```
for (s=0; s<S; s++){  
  for(q=0; q<Q; q++){  
    OA[q] += IA[q+s] * W[s];  
  }  
}
```

**Weight Stationary (WS)
Dataflow**

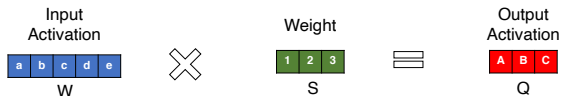
Buffer Access Pattern 1: Output Stationary



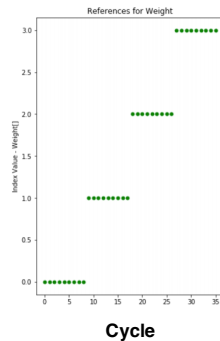
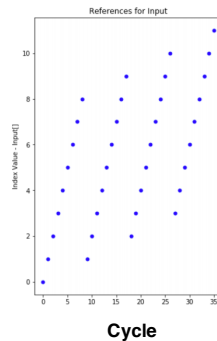
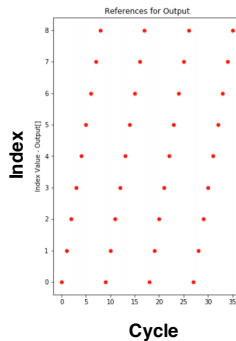
```
for (q=0; q<Q; q++){ // Q =9
    for (s=0; s<S; s++){ // S=4
        OA[q] += IA[q+s] * W[s];
    }
}
```



Buffer Access Pattern 2: Weight Stationary



```
for (s=0; s<S; s++){ // S=4
  for (q=0; q<Q; q++){ // Q =9
    OA[q] += IA[q+s] * W[s];
  }
}
```

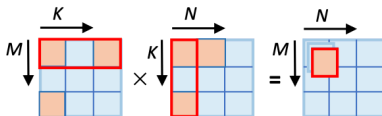


2D Convolution Example



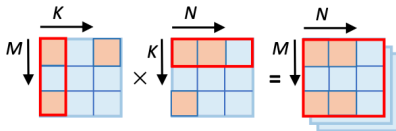
Inner-product
dataflow

```
for m in 0..M
  for n in 0..N
    for k in 0..K
      C[m,n] += A[m,k] * B[k,n]
```



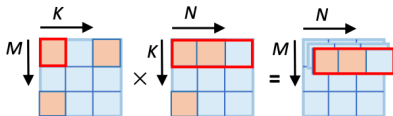
Outer-product
dataflow

```
for k in 0..K
  for m in 0..M
    for n in 0..N
      C[m,n] += A[m,k] * B[k,n]
```



Row-based
dataflow

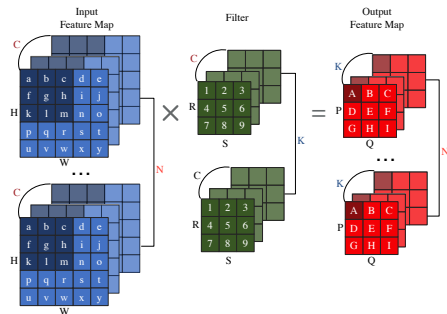
```
for m in 0..M
  for k in 0..K
    for n in 0..N
      C[m,n] += A[m,k] * B[k,n]
```



	InP	OutP	ROW
Input reuse (B)	Poor	Excellent	Poor
Output reuse (C)	Excellent	Poor	Good
Index intersection	Inefficient	Efficient	Efficient
Psum granularity	Scalar	Matrix	Vector



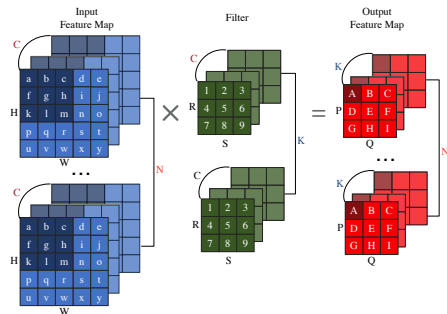
Direct Convolution



```

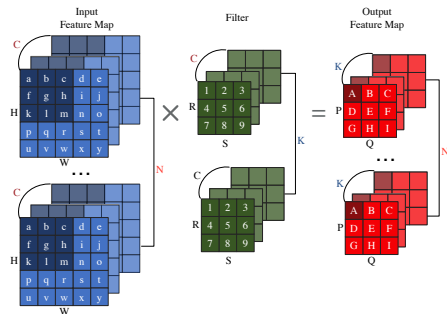
1  for (n=0; n<N; n++) {
2  for (k=0; k<K; k++) {
3  for (p=0; p<P; p++) {
4  for (q=0; q<Q; q++) {
5      OA[n][k][p][q] = 0;
6      for (r=0; r<R; r++) {
7      for (s=0; s<S; s++) {
8      for (c=0; c<C; c++) {
9          h = p * stride - pad + r;
10         w = q * stride - pad + s;
11         OA[n][k][p][q] += IA[n][c][h][w] * W[k][c][r][s];
12     } } } } } } }
    
```

Direct Convolution: Loop Ordering



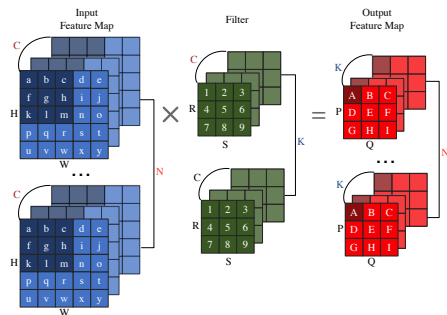
```
1  for (n=0; n<N; n++) {
2  for (r=0; r<R; r++) {
3  for (s=0; s<S; s++) {
4  for (c=0; c<C; c++) {
5  for (k=0; k<K; k++) {
6      float curr_w = W[r][s][c][k];
7      for (p=0; p<P; p++) {
8      for (q=0; q<Q; q++) {
9          h = p * stride - pad + r;
10         w = q * stride - pad + s;
11         OA[n][k][p][q] += IA[n][c][h][w] * curr_w;
12     } } } } } }
```

Direct Convolution: Loop Ordering + Unrolling



```
1  for (n=0; n<N; n++) {
2  for (r=0; r<R; r++) {
3  for (s=0; s<S; s++) {
4      spatial_for (c=0; c<C; c++) {
5      spatial_for (k=0; k<K; k++) {
6          float curr_w = W[r][s][c][k];
7          for (p=0; p<P; p++) {
8              for (q=0; q<Q; q++) {
9                  h = p * stride - pad + r;
10                 w = q * stride - pad + s;
11                 OA[n][k][p][q] += IA[n][c][h][w] * curr_w;
12             } } } } } }
```

Direct Convolution: Loop Ordering + Unrolling + Tiling



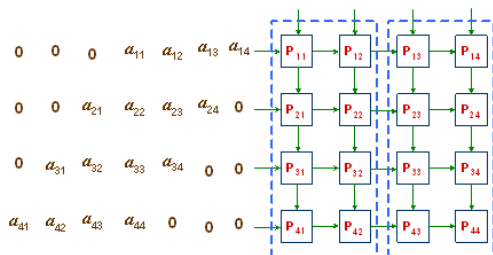
```
1  for (n=0; n<N; n++) {
2  for (r=0; r<R; r++) {
3  for (s=0; s<S; s++) {
4  for (c_t=0; c_t<C/16; c_t++) {
5  for (k_t=0; k_t<K/64; k_t++) {
6  spatial_for (c_s=0; c_s<16; c_s++) {
7  spatial_for (k_s=0; k_s<64; k_s++) {
8      int curr_c = c_t * 16 + c_s;
9      int curr_k = k_t * 64 + k_s;
10     float curr_w = W[r][s][curr_c][curr_k];
11     for (p=0; p<P; p++) for (q=0; q<Q; q++) {
12         h = p * stride - pad + r; w = q * stride - pad + s;
13         OA[n][curr_k][p][q] += IA[n][curr_c][h][w] * curr_w;
14     } } } } }
```



Dataflow Optimization

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & b_{14} \\ 0 & 0 & b_{13} & b_{24} \\ 0 & b_{12} & b_{23} & b_{34} \\ b_{11} & b_{22} & b_{33} & b_{44} \\ b_{21} & b_{32} & b_{43} & 0 \\ b_{31} & b_{42} & 0 & 0 \\ b_{41} & 0 & 0 & 0 \end{bmatrix}$$





[HPCA2020] Communication Lower Bound in Convolution Accelerators



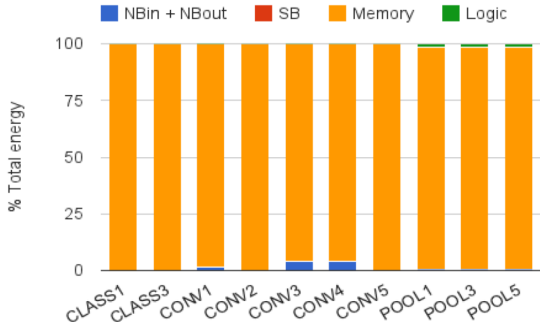
Case Study 2

Communication Lower Bound in CNN Accelerators

Memory Bottleneck in CNN Accelerators

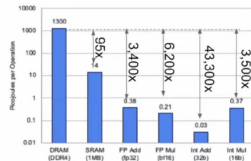
2

- **Memory access** consumes **most of total energy**
- **CNN accelerators** are mostly **memory dominant**



Lesson 6: It's the memory, stupid! (not the FLOPs)

- Energy limits modern chips, not number of transistors
- External memory access energy ~100X on chip memory access ~10,000X arithmetic operation
- Easy to scale up FLOPs/sec by adding many ALUs to balance energy of memory accesses
 - Also why DNN model developers should focus on reducing memory accesses versus reducing FLOPs



Jouppi N., Yoon, D.H., Jablin, T., Kurian, G., Laxton, J., Li, S., Ma, P., Ma, K., Patel, N., Prasad, S., Young, C., Zhou, Z., and Patterson, D. 2014. [Ten Lessons From Three Generations Shaped Google's TPUv2](#). In Proc. 48th International Symposium on Computer Architecture.

Google

20

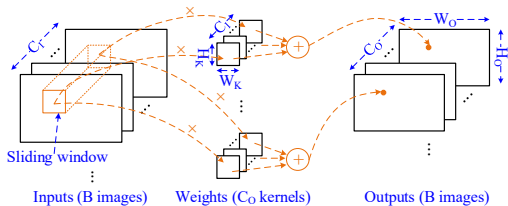
T. Chen et al., DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning, ASPLOS'14

Google slide, one of ten lessons learned from three generations TPUs

Convolutional Layer

3

- Complicated data reuse
 - Input reuse
 - Sliding window reuse
 - Weight reuse
 - Output reuse
- Finding minimum communication is difficult: **huge search space** caused by **7 levels of loops** and **complex data reuse schemes**



```

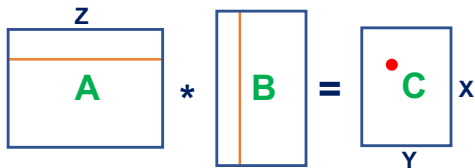
for (i = 0; i < B; i++) // Images in a batch
for (oz = 0; oz < Co; oz++) // Output channels
for (oy = 0; oy < Ho; oy++) // Output rows
for (ox = 0; ox < Wo; ox++) // Output columns
for (kz = 0; kz < Ci; kz++) // Input channels
for (ky = 0; ky < Hk; ky++) // Kernel rows
for (kx = 0; kx < Wk; kx++) // Kernel columns
    out[i][oz][oy][ox] +=
        in[i][kz][oy+ky][ox+kx] * w[oz][kz][ky][kx];
  
```



Communication in Matrix Multiplication

4

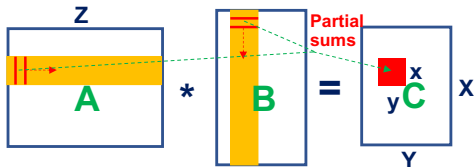
- Naive matrix multiplication



$$Q = 2XYZ + XY$$

$$\approx 2XYZ$$

- Communication-optimal matrix multiplication



$$Q = \frac{XY}{xy} (xZ + yZ) + XY$$

$$\approx XYZ \left(\frac{1}{x} + \frac{1}{y} \right) \geq \frac{2XYZ}{\sqrt{xy}}$$

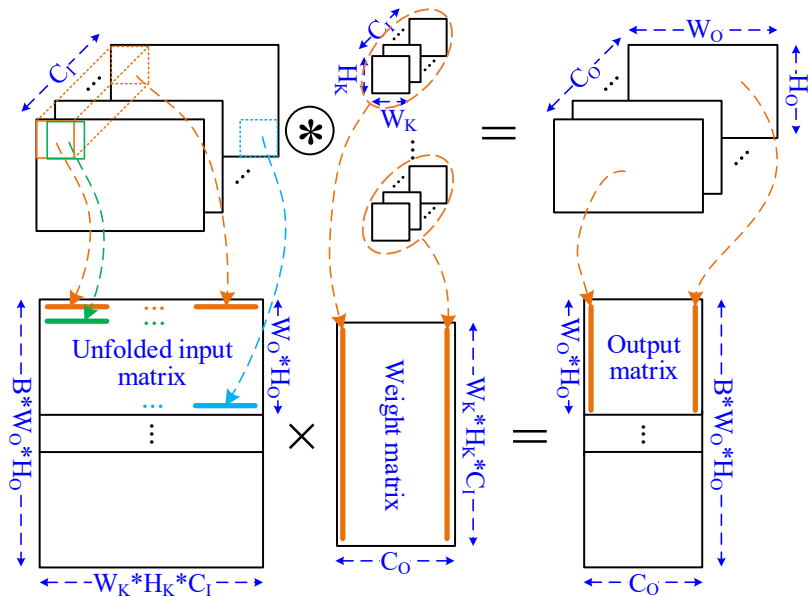
$$\geq \frac{2XYZ}{\sqrt{S}}$$

S : on-chip memory capacity



Relation between Convolution & Matrix Multiplication (im2col)

5



Observations

6

- Weights and outputs are just **reshaped** ---- without adding or removing elements
- Inputs are **unfolded** ---- all sliding windows (having overlapped elements) are explicitly expanded
- Convolution has only **one more level of data reuse (sliding window reuse)** than matrix multiplication

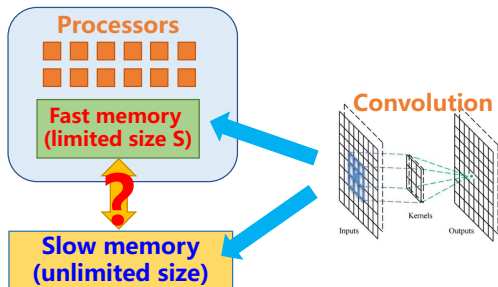
Communication-optimal convolution

= communication-optimal matrix multiplication + sliding window reuse?

Communication Lower Bound of Convolution

7

- Matrix multiplication only used to inspire derivation process, **there is not an actual conversion** in our implementation
- Theoretical derivation based on Red-Blue Pebble Game [1]



$$Q = \Omega \left(\frac{BW_O H_O C_O W_K H_K C_I}{\sqrt{RS}} \right)$$

$$R = \frac{W_K H_K}{D_W D_H} \quad \begin{array}{l} W_K \text{ \& } H_K: \text{ kernel size} \\ D_W \text{ \& } D_H: \text{ stride size} \end{array}$$

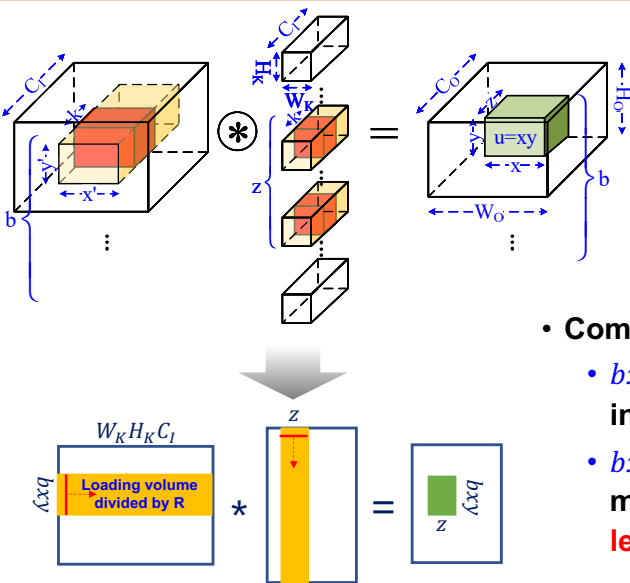
R: max reuse number of each input by sliding window reuse

Communication-optimal Dataflow

8

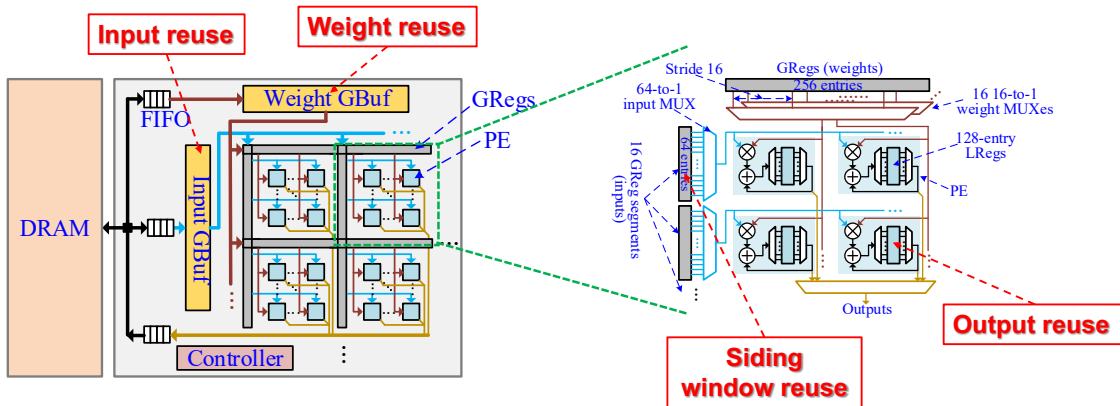
Tiling parameters
 $\langle b, x, y, z, k \rangle$

- **Communication-optimal tiling parameters**
 - $bxy \approx Rz$: balanced loading volumes of inputs & weights
 - $bxyz \approx S$ & $k = 1$: most of on-chip memory should be for Psums (using least inputs to produce most outputs)



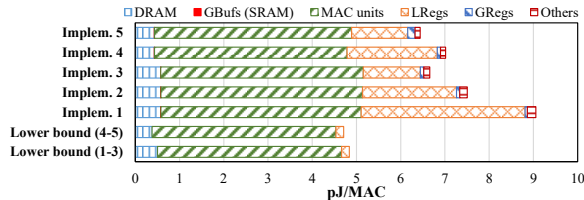
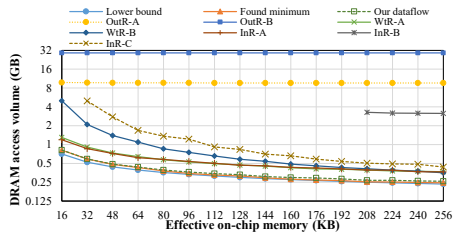
Communication-optimal Architecture

- Straightforward implementation of communication-optimal dataflow
- Elaborate multiplexer structure to adapt to different tiling parameters, no inter-PE data propagation



Simulation Results

10



DRAM access: 4.5% more than lower bound, >40% reduction than Eyeriss [1]

Energy consumption: 37-87% higher than lower bound