CMSC 5743

Efficient Computing of Deep Neural Networks

Implementation 02: Direct Conv

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

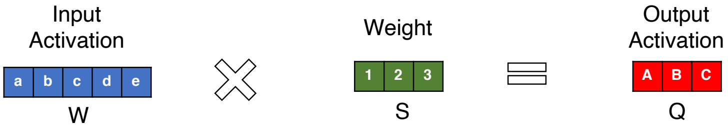
(Latest update: September 2, 2024)

2024 Fall

Overview

Loop Reordering

1D Convolution Example



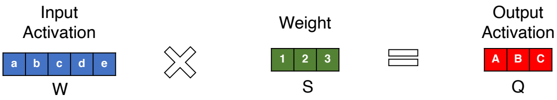

```
for(q=0; q<Q; q++){
  for (s=0; s<S; s++){
    OA[q] += IA[q+s] * W[s];
  }
}
```

Output Stationary (OS) Dataflow

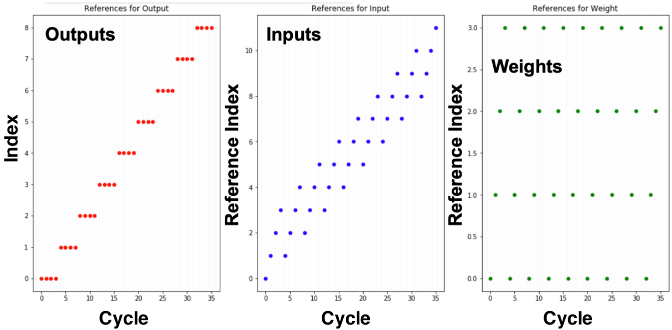
```
for (s=0; s<S; s++){
  for(q=0; q<Q; q++){
    OA[q] += IA[q+s] * W[s];
  }
}
```

Weight Stationary (WS) Dataflow

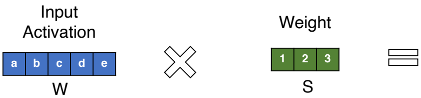
Buffer Access Pattern 1: Output Stationary



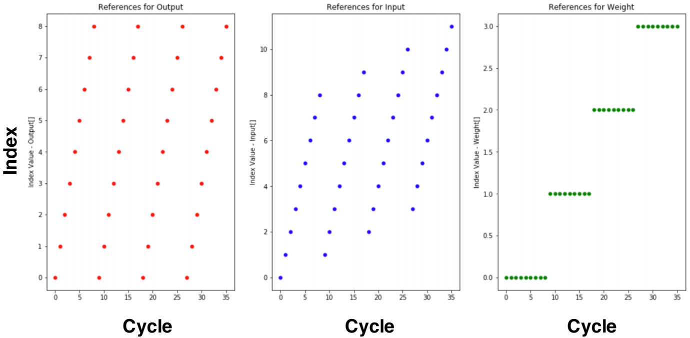

```
for(q=0; q<Q; q++){ // Q =9
  for (s=0; s<S; s++){ // S=4
    OA[q] += IA[q+s] * W[s];
  }
}
```



Buffer Access Pattern 2: Weight Stationary



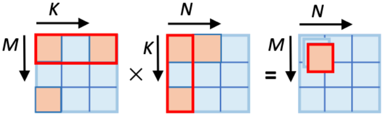

```
for (s=0; s<S; s++){// S=4
  for(q=0; q<Q; q++){// Q =9
    OA[q] += IA[q+s] * W[s];
  }
```



2D Convolution Example

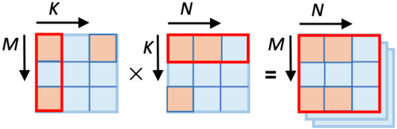
Inner-product dataflow

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



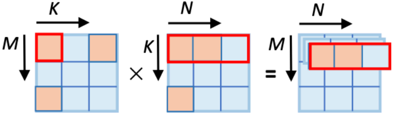
Outer-product dataflow

```
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

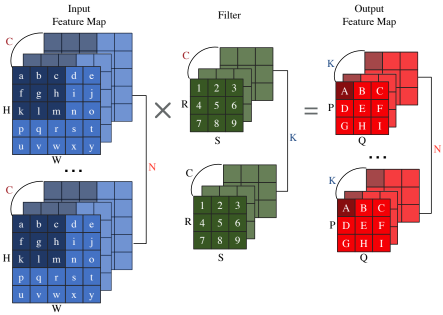
```
for m in 0..M
 for k in 0..K
  for n in 0..N
   C[m,n] += A[m,k] * B[k,n]
```



	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

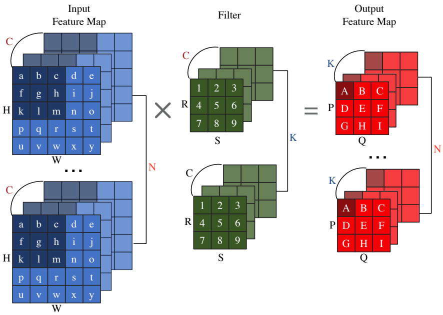
Direct Convolution

Direct Convolution



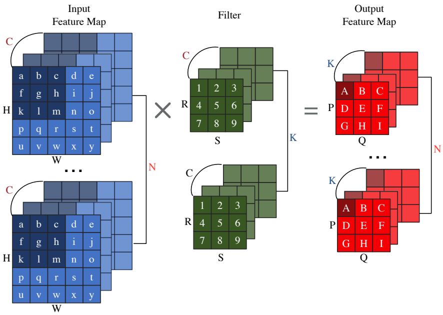

```
for (n=0; n<N; n++) {
for (k=0; k<K; k++) {
for (p=0; p<P; p++) {
for (q=0; q<Q; q++) {
    OA[n][k][p][q]= 0;
    for (r=0; r<R; r++) {
    for (s=0; s<S; s++) {
    for (c=0; c<C; c++) {
        h = p * stride - pad + r;
        w = q * stride - pad + s;
        OA[n][k][p][q] += IA[n][c][h][w]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

Direct Convolution: Loop Ordering



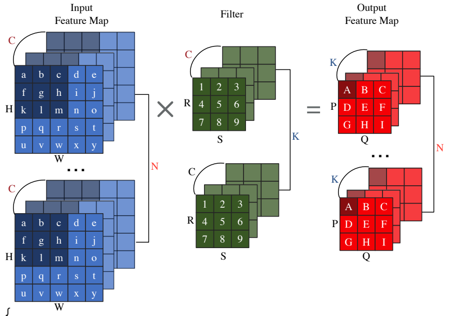

```
for (n=0; n<N; n++) {
for (r=0; r<R; r++) {
for (s=0; s<S; s++) {
for (c=0; c<C; c++) {
for (k=0; k<K; k++) {
    float curr_w = W[r][s][c][k];
    for (p=0; p<P; p++) {
    for (q=0; q<Q; q++) {
        h = p * stride - pad + r;
        w = q * stride - pad + s;
        OA[n][k][p][q] += IA[n][c][h][w]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

Direct Convolution: Loop Ordering + Unrolling




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

Direct Convolution: Loop Ordering + Unrolling + Tiling

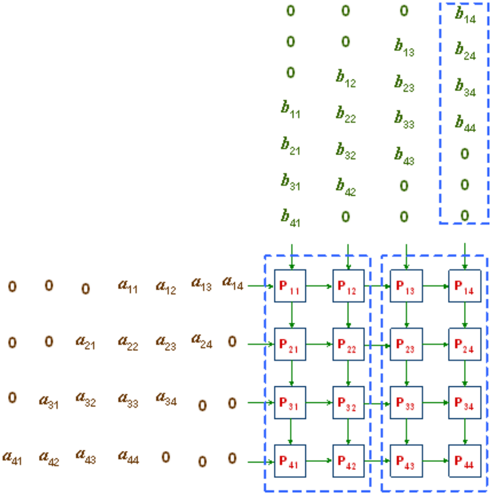



```
for (n=0; n<N; n++) {
for (r=0; r<R; r++) {
for (s=0; s<S; s++) {
for (c_t=0; c_t<C/16; c_t++) {
for (k_t=0; k_t<K/64; k_t++) {
spatial_for (c_s=0; c_s<16; c_s++) {
spatial_for (k_s=0; k_s<64; k_s++) {
    int curr_c = c_t * 16 + c_s;
    int curr_k = k_t * 64 + k_s;
    float curr_w = W[r][s][curr_c][curr_k];
    for (p=0; p<P; p++) for (q=0; q<Q; q++) {
        h = p * stride - pad + r; w = q * stride - pad + s;
        OA[n][curr_k][p][q] += IA[n][curr_c][h][w] * curr_w;
} } } } } }
```

Dataflow Optimization

Systolic Array

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



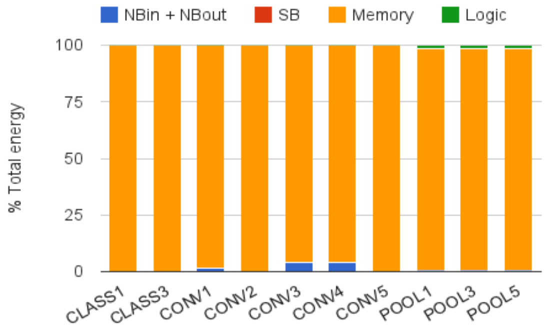
[HPCA2020] Communication Lower Bound in Convolution Accelerators

Case Study 2

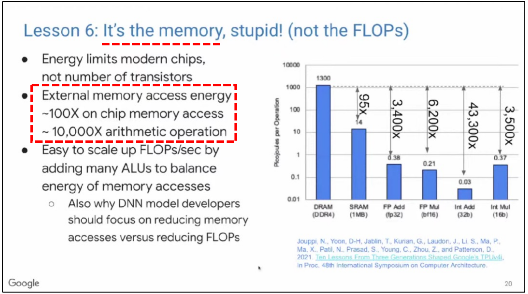
Communication Lower Bound in CNN Accelerators

Memory Bottleneck in CNN Accelerators

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



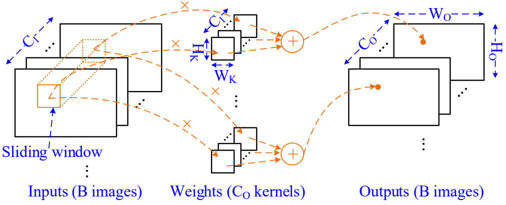
T. Chen et al., DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning, ASPLOS'14



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

Convolutional Layer

- **Complicated data reuse**
  - **Input reuse**
  - **Sliding window reuse**
  - **Weight reuse**
  - **Output reuse**

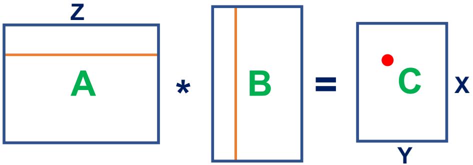


- **Finding minimum communication is difficult: huge search space caused by 7 levels of loops and complex data reuse schemes**

```
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];
```

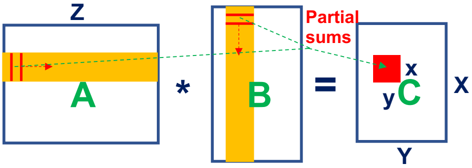
Communication in Matrix Multiplication

- **Naive matrix multiplication**



$$Q = 2XYZ + XY$$
$$\approx 2XYZ$$

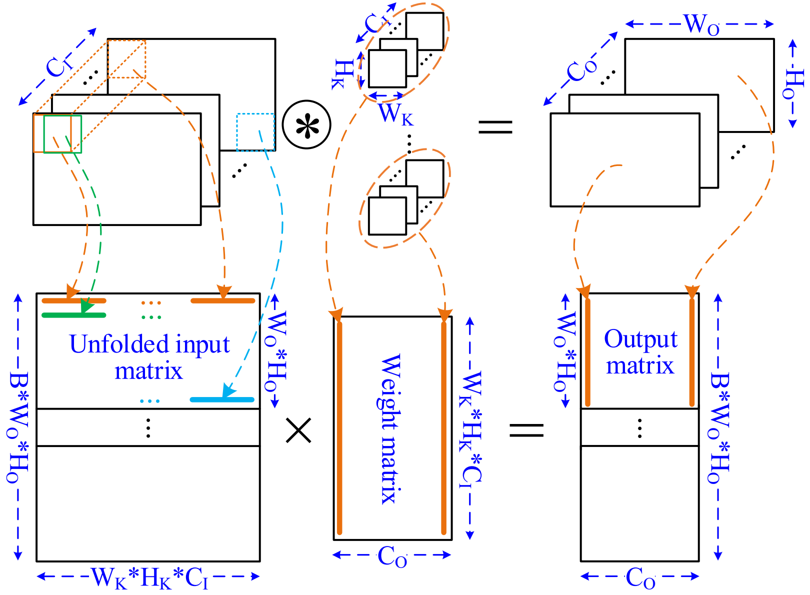
- **Communication-optimal matrix multiplication**



$$Q = \frac{XY}{xy}(xZ + yZ) + XY$$
$$\approx XYZ\left(\frac{1}{x} + \frac{1}{y}\right) \geq \frac{2XYZ}{\sqrt{xy}}$$
$$\geq \frac{2XYZ}{\sqrt{S}}$$

S: on-chip memory capacity

Relation between Convolution & Matrix Multiplication (im2col)



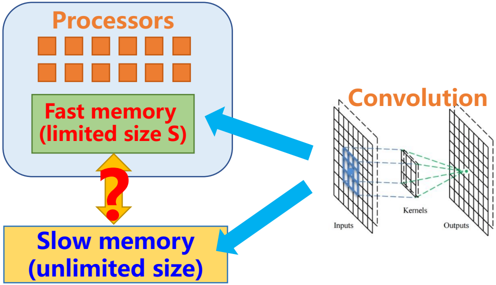
Observations

- 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

- **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]**



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

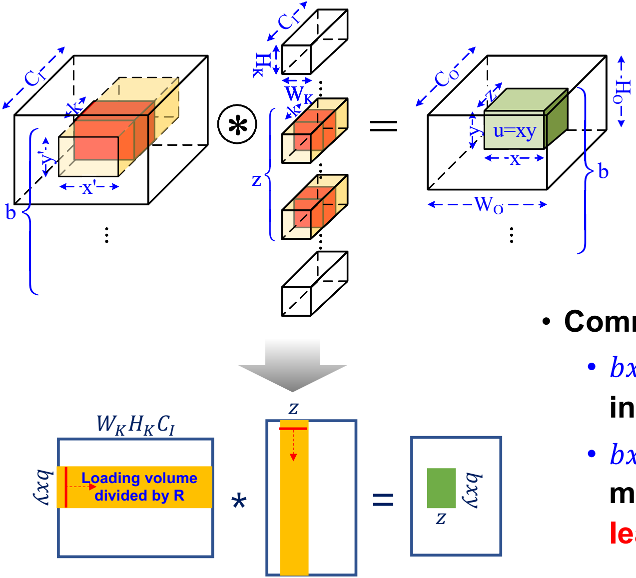
$$R = \frac{W_K H_K}{D_W D_H}$$

W_K & H_K: kernel size
D_W & D_H: stride size

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

[1] J.-W. Hong and H. T. Kung, "I/O Complexity: The Red-blue Pebble Game," in ACM Symposium on Theory of Computing (STOC), 1981, pp. 326-333

Communication-optimal Dataflow

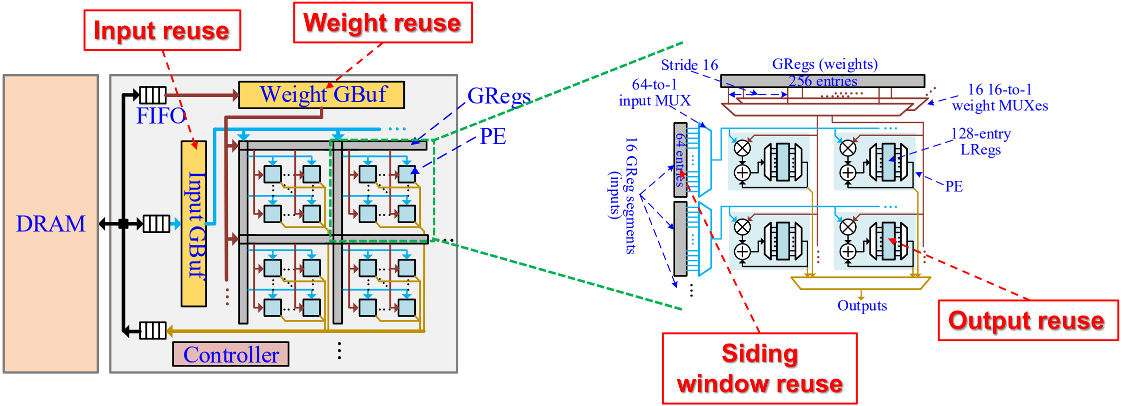


Tiling parameters
$< b, x, y, z, k >$

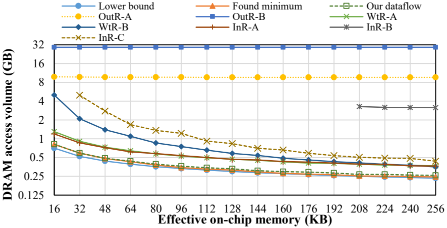
- **Communication-optimal tiling parameters**
  - $bxy \approx Rz$: **balanced loading volumes of inputs & weights**
  - $bxyz \approx S$ **&** $k = 1$: **most of on-chip memory should be for Psums** **(using least inputs to produce most outputs)**

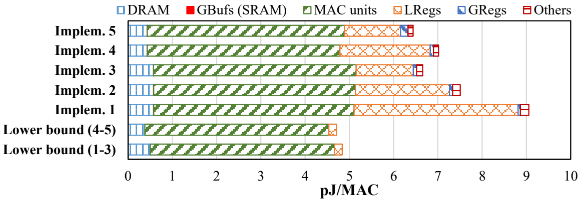
Communication-optimal Architecture

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



Simulation Results





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

Energy consumption: 37-87% higher than lower bound

[1] Y. H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in ISCA 2016