CENG 5030 Energy Efficient Computing

Lecture 04: Accurate Speedup I

Bei Yu

(Latest update: February 1, 2021)

Spring 2021

These slides contain/adapt materials developed by

- Minsik Cho and Daniel Brand (2017). "MEC: memory-efficient convolution for deep neural network". In: *Proc. ICML*
- Asit K. Mishra et al. (2017). "Fine-grained accelerators for sparse machine learning workloads". In: *Proc. ASPDAC*, pp. 635–640
- Jongsoo Park et al. (2017). "Faster CNNs with direct sparse convolutions and guided pruning". In: *Proc. ICLR*
- UC Berkeley EE290: "Hardware for Machine Learning" `https://inst.eecs.berkeley.edu/~ee290-2/sp20/`

Overview

Convolution 101

GEMM

Sparse Convolution

Direct Convolution

Overview

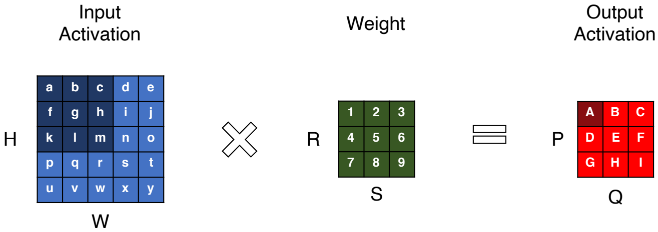
Convolution 101

GEMM

Sparse Convolution

Direct Convolution

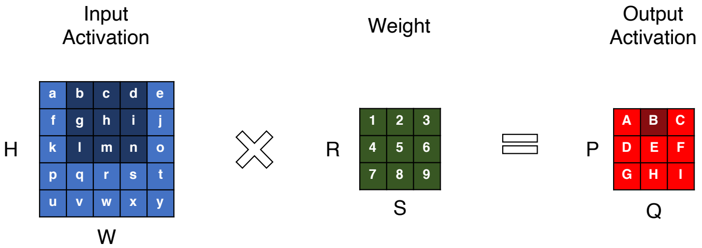
2D-Convolution



H: Height of Input Activation
W: Width of Input Activation
R: Height of Weight
S: Width of Weight
P: Height of Output Activation
Q: Width of Output Activation

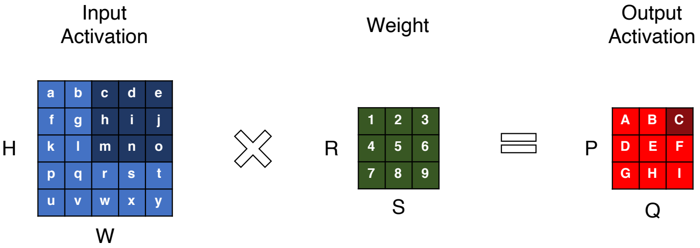
$$
\begin{aligned}
A = {} & a*1+b*2+c*3 \\
& +f*4+g*5+h*6 \\
& +k*7+l*8+m*9
\end{aligned}
$$

2D-Convolution



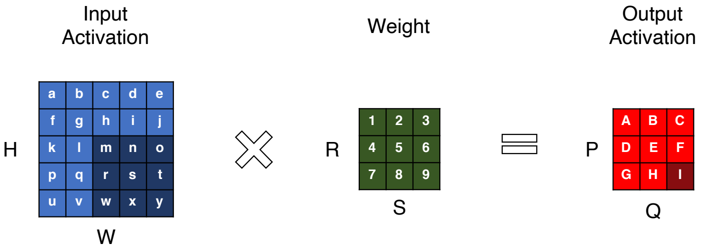
H: Height of Input Activation
W: Width of Input Activation
R: Height of Weight
S: Width of Weight
P: Height of Output Activation
Q: Width of Output Activation
stride: # of rows/columns traversed per step

2D-Convolution



H: Height of Input Activation
W: Width of Input Activation
R: Height of Weight
S: Width of Weight
P: Height of Output Activation
Q: Width of Output Activation
stride: # of rows/columns traversed per step

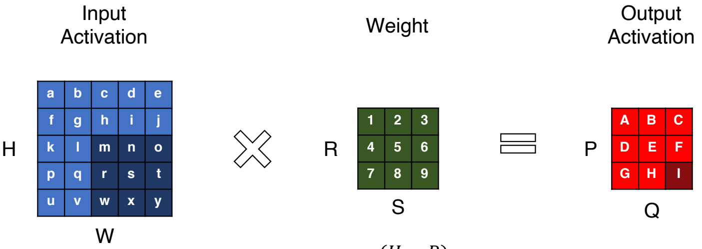
2D-Convolution



H: Height of Input Activation
W: Width of Input Activation
R: Height of Weight
S: Width of Weight
P: Height of Output Activation
Q: Width of Output Activation
stride: # of rows/columns traversed per step

$$
\begin{aligned}
I = {} & m * 1 + n * 2 + o * 3 \\
& + r * 4 + s * 5 + t * 6 \\
& + w * 7 + x * 8 + y * 9
\end{aligned}
$$

2D-Convolution

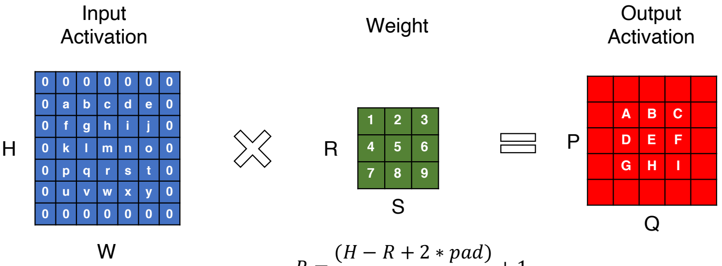


$$P = \frac{(H - R)}{stride} + 1$$

$$Q = \frac{(W - S)}{stride} + 1$$

H: Height of Input Activation
W: Width of Input Activation
R: Height of Weight
S: Width of Weight
P: Height of Output Activation
Q: Width of Output Activation
stride: # of rows/columns traversed per step

2D-Convolution

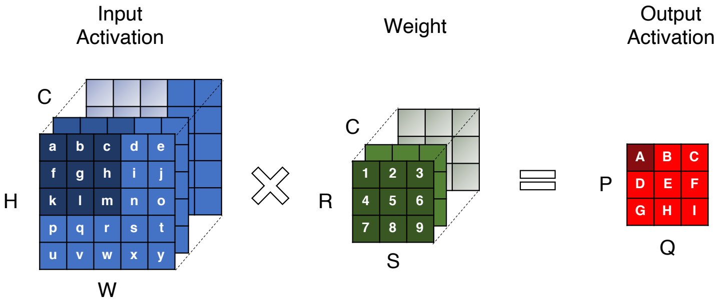


$$P = \frac{(H - R + 2 * pad)}{stride} + 1$$

$$Q = \frac{(W - S + 2 * pad)}{stride} + 1$$

H: Height of Input Activation
W: Width of Input Activation
R: Height of Weight
S: Width of Weight
P: Height of Output Activation
Q: Width of Output Activation
stride: # of rows/columns traversed per step
padding: # of zero rows/columns added

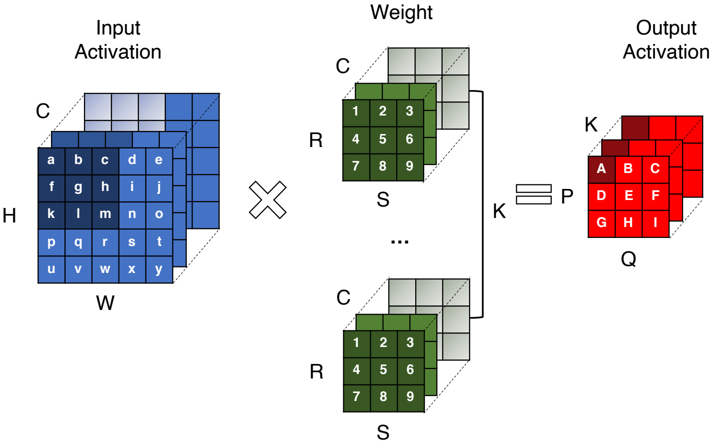
3D-Convolution



H: Height of Input Activation
W: Width of Input Activation
R: Height of Weight
S: Width of Weight
P: Height of Output Activation
Q: Width of Output Activation
stride: # of rows/columns traversed per step
padding: # of zero rows/columns added

C: # of Input Channels

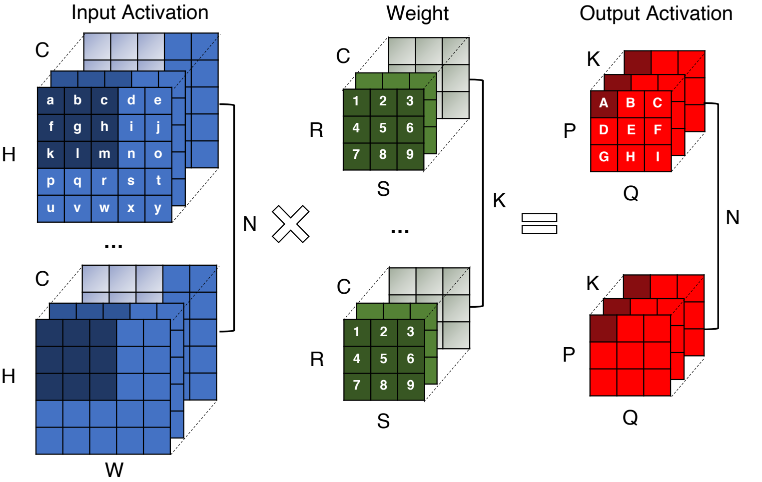
3D-Convolution



H: Height of Input Activation
W: Width of Input Activation
R: Height of Weight
S: Width of Weight
P: Height of Output Activation
Q: Width of Output Activation
stride: # of rows/columns traversed per step
padding: # of zero rows/columns added

C: # of Input Channels
K: # of Output Channels

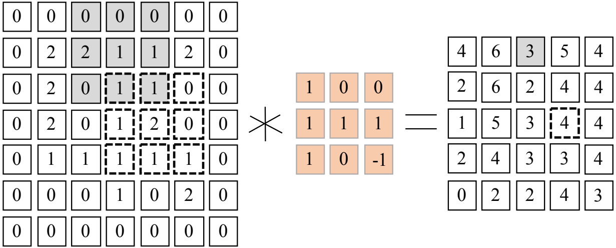
3D-Convolution



H: Height of Input Activation
W: Width of Input Activation
R: Height of Weight
S: Width of Weight
P: Height of Output Activation
Q: Width of Output Activation
stride: # of rows/columns traversed per step
padding: # of zero rows/columns added

C: # of Input Channels
K: # of Output Channels
N: Batch size

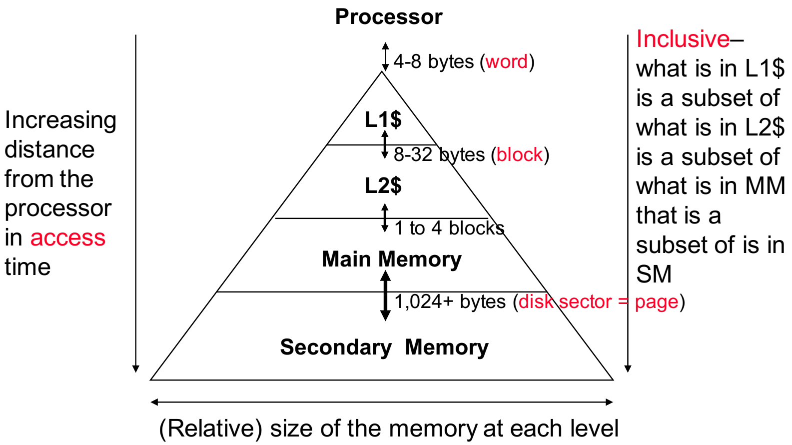
Convolution 101



Direct convolution: No extra memory overhead

- Low performance
- Poor memory access pattern due to geometry-specific constraint
- Relatively short dot product

Background: Memory System



- Spatial locality
- Temporal Locality

Overview

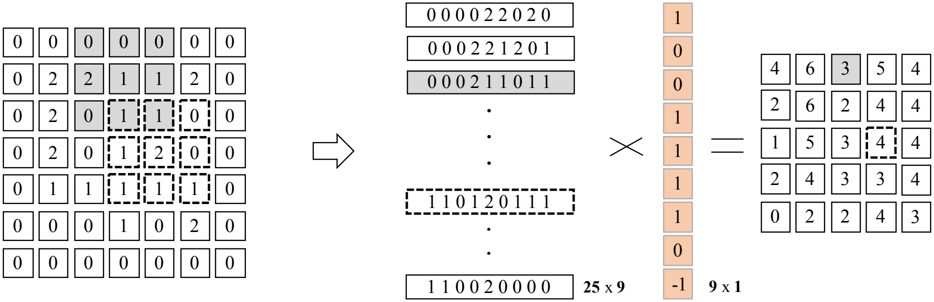
Convolution 101

GEMM

Sparse Convolution

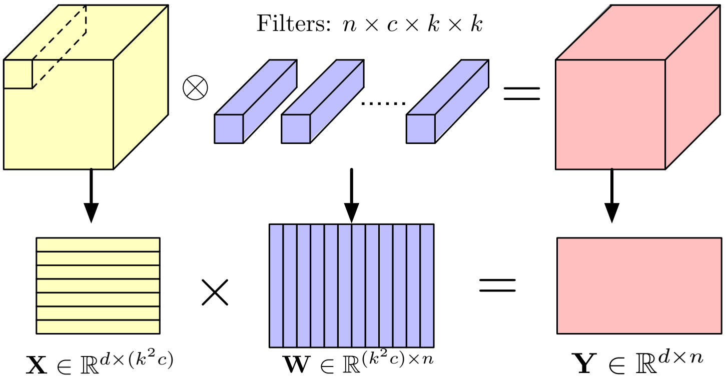
Direct Convolution

Im2col (Image2Column) Convolution



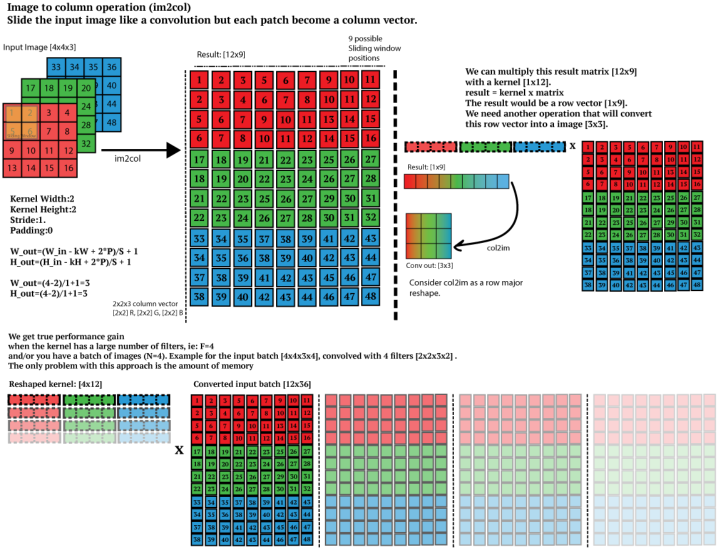
- Large extra memory overhead
- Good performance
- BLAS-friendly memory layout to enjoy SIMD/locality/parallelism
- Applicable for any convolution configuration on any platform

`Im2col` (Image2Column) Convolution



- Transform convolution to matrix multiplication
- Unified calculation for both convolution and fully-connected layers

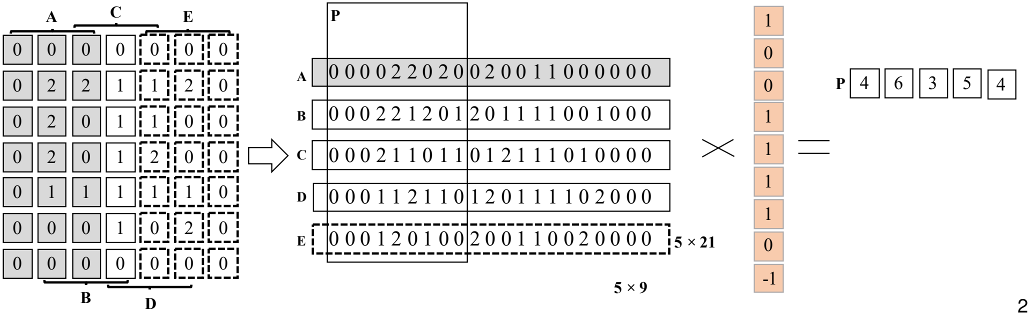
Im2col (Image2Column): Another View



---

[1] https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/deep_learning/convolution_layer/making_faster

SOTA 1: Memory-efficient Convolution

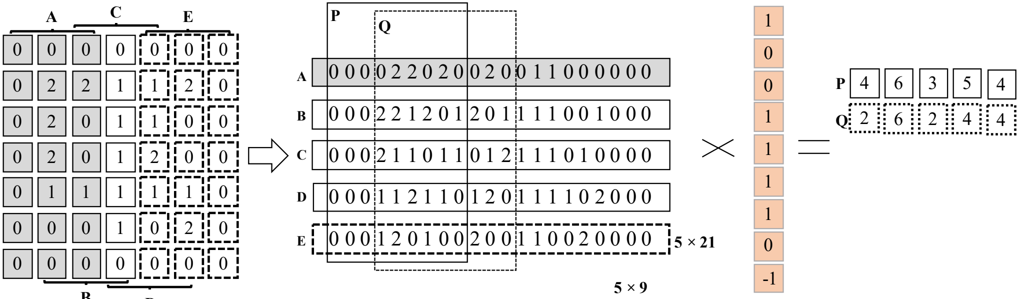


2

- Sub matrices in the lowered matrix will be “sgemm” ed in parallel
- Smaller memory foot print, cache locality, and explicit parallelism

[2] Minsik Cho and Daniel Brand (2017). “MEC: memory-efficient convolution for deep neural network”. In: *Proc. ICML*.

SOTA 1: Memory-efficient Convolution

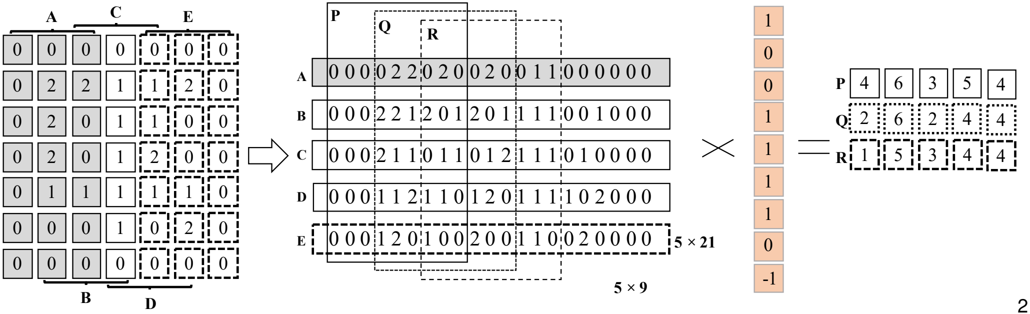


[2]

- Sub matrices in the lowered matrix will be “sgemm” ed in parallel
- Smaller memory foot print, cache locality, and explicit parallelism

[2] Minsik Cho and Daniel Brand (2017). “MEC: memory-efficient convolution for deep neural network”. In: *Proc. ICML*.

SOTA 1: Memory-efficient Convolution

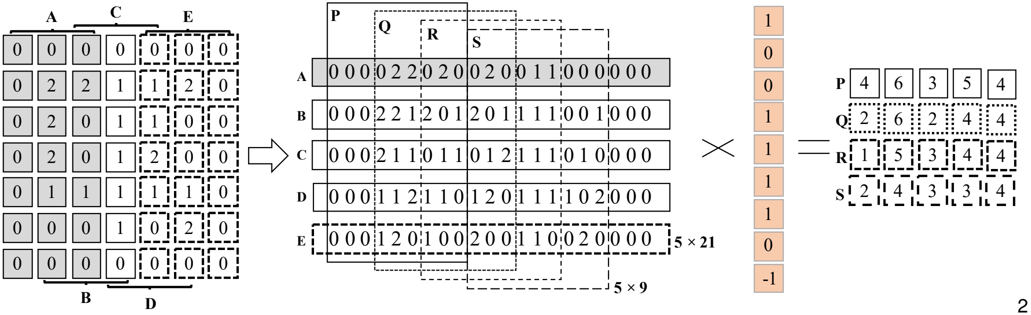


[2]

- Sub matrices in the lowered matrix will be "sgemm" ed in parallel
- Smaller memory foot print, cache locality, and explicit parallelism

[2] Minsik Cho and Daniel Brand (2017). "MEC: memory-efficient convolution for deep neural network". In: *Proc. ICML*.

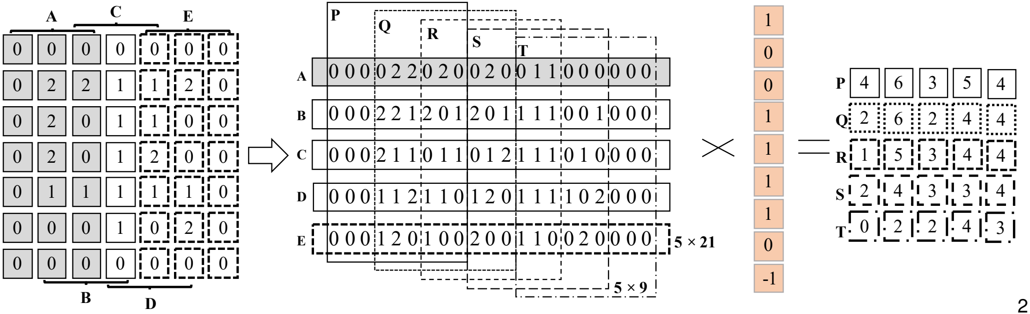
SOTA 1: Memory-efficient Convolution



- Sub matrices in the lowered matrix will be “sgemm” ed in parallel
- Smaller memory foot print, cache locality, and explicit parallelism

[2] Minsik Cho and Daniel Brand (2017). “MEC: memory-efficient convolution for deep neural network”. In: *Proc. ICML*.

SOTA 1: Memory-efficient Convolution



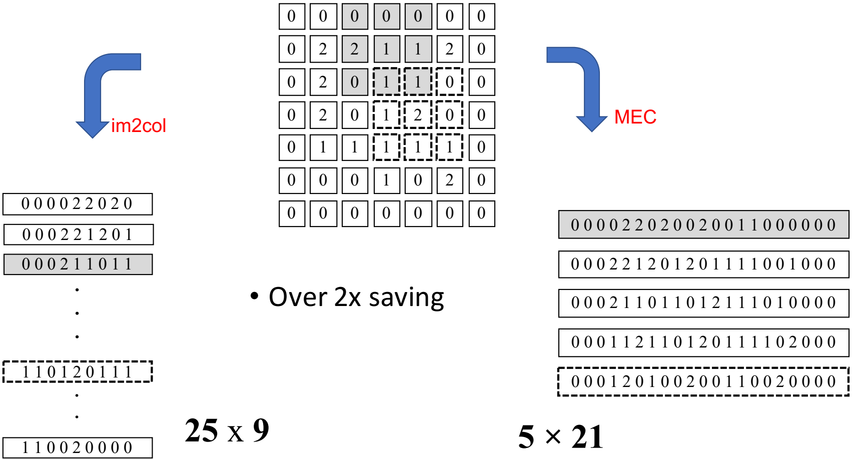
[2]

- Sub matrices in the lowered matrix will be “sgemm” ed in parallel
- Smaller memory foot print, cache locality, and explicit parallelism

[2] Minsik Cho and Daniel Brand (2017). “MEC: memory-efficient convolution for deep neural network”. In: *Proc. ICML*.

SOTA 1: Memory-efficient Convolution

Over $2\times$ memory saving[3]:



- Over 2x saving

[3] Minsik Cho and Daniel Brand (2017). "MEC: memory-efficient convolution for deep neural network". In: *Proc. ICML*.

Overview

Convolution 101

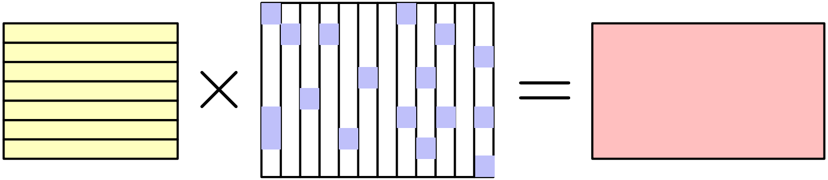
GEMM

Sparse Convolution

Direct Convolution

Sparse Convolution

- Our DNN may be redundant, and sometimes the filters may be sparse
- Sparsity can be helpful to overcome over-fitting



Sparse Convolution: Naive Implementation 1

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

Algorithm 1 Sparse Convlution Naive 1

```
for all w[i] do
    if w[i] = 0 then
        Continue;
    end if
    output feature map Y ← X × w[i];
end for
```

Sparse Convolution: Naive Implementation 1

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

Algorithm 2 Sparse Convlution Naive 1

for all w[i] **do**
    if w[i] = 0 **then**
        Continue;
    end if
    output feature map $Y \leftarrow X \times w$[i];
end for

BAD implementation for Pipeline!

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Sparse Matrix Representation

A

0	0	3	0
7	0	0	0
0	0	4	8
6	5	3	0
2	0	0	1
0	0	0	8

A matrix example

rowptr

- → row0 (3,2)
- → row1 (7,0)
- → row2 (4,2), (8,3)
- → row3 (6,0), (5,1), (3,2)
- → row4 (2,0), (1,3)
- → row5 (8,3)

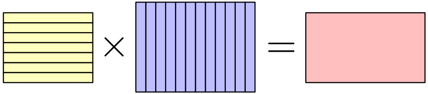
Compressed Sparse Row (CSR)

colptr

- → col0 (7,1), (6,3), (2,4)
- → col1 (5,3)
- → col2 (3,0), (4,2), (3,3)
- → col3 (8,2), (1,4), (8,5)

Compressed Sparse Column (CSC)

- CSR: Good for operation on feature maps
- CSC: Good for operation on filters
- We have better control on filters, thus usually CSC.



Sparse Convolution: Naive Implementation 2

matrix * sparse vector

$$\overset{X}{\begin{bmatrix} 0 & 0 & 3 & 0 \\ 7 & 0 & 0 & 0 \\ 0 & 0 & 4 & 8 \\ 6 & 5 & 3 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 8 \end{bmatrix}} * \overset{w}{\begin{bmatrix} 0 \\ 0 \\ \boxed{4} \\ 8 \end{bmatrix}} = \overset{Y}{\begin{bmatrix} 12 \\ 0 \\ 16 \\ 12 \\ 0 \\ 0 \end{bmatrix}}$$

$$\begin{bmatrix} 0 & 0 & 3 & 0 \\ 7 & 0 & 0 & 0 \\ 0 & 0 & 4 & 8 \\ 6 & 5 & 3 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 8 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ 4 \\ \boxed{8} \end{bmatrix} = \begin{bmatrix} 12 \\ 0 \\ 80 \\ 12 \\ 8 \\ 64 \end{bmatrix}$$

- BAD implementation for Spatial Locality!
- Poor memory access patterns

SOTA 2: Sparse Convolution

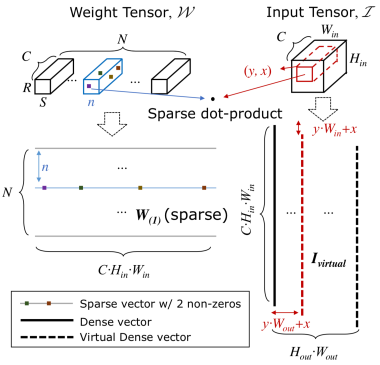


Figure 1: Conceptual view of the direct sparse convolution algorithm. Computation of output value at (y,x)th position of nth output channel is highlighted.

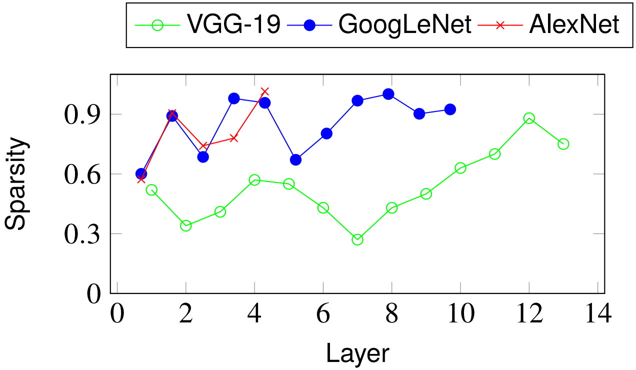
```
for each output channel n {
 for j in [W.rowptr[n], W.rowptr[n+1]) {
  off = W.colidx[j]; coeff = W.value[j]
  for (int y = 0; y < H_OUT; ++y) {
   for (int x = 0; x < W_OUT; ++x) {
    out[n][y][x] += coeff*in[off+f(0,y,x)]
   }
  }
 }
}
```

Figure 2: Sparse convolution pseudo code. Matrix **W** has *compressed sparse row* (CSR) format, where `rowptr[n]` points to the first non-zero weight of nth output channel. For the jth non-zero weight at (n,c,r,s), `W.colidx[j]` contains the offset to (c,r,s)th element of tensor `in`, which is pre-computed by layout function as $f(c,r,s)$. If `in` has CHW format, $f(c,r,s) = (cH_{in} + r)W_{in} + s$. The "virtual" dense matrix is formed on-the-fly by shifting `in` by $(0,y,x)$. [4]

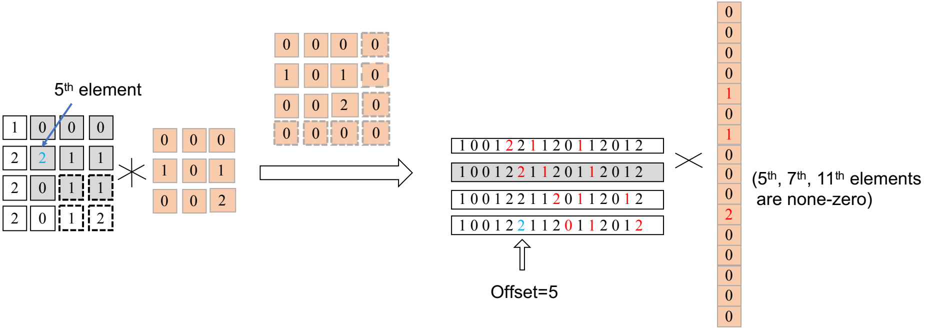
[4] Jongsoo Park et al. (2017). "Faster CNNs with direct sparse convolutions and guided pruning". In: *Proc. ICLR*.

Discussion: Sparse-Sparse Convolution

- Sparsity is a desired property for computation acceleration. (`cuSPARSE` library, direct sparse convolution, etc.)
- Sometimes not only the filters but also the input feature maps are sparse.



Discussion: Sparse-Sparse Convolution



- Efficient programming implementation required; (Improve pipeline efficiency)
- When sparsity$(input) = 0.9$, sparsity$(weight) = 0.8$, more than $10\times$ speedup;
- Some other issues:
  - How to be compatible with pooling layer?
  - Transform between dense & sparse formats

Overview

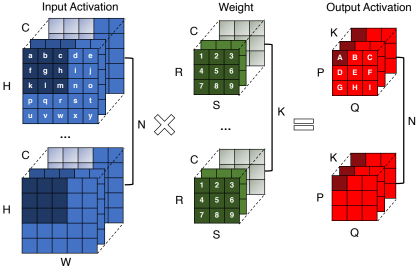
Convolution 101

GEMM

Sparse Convolution

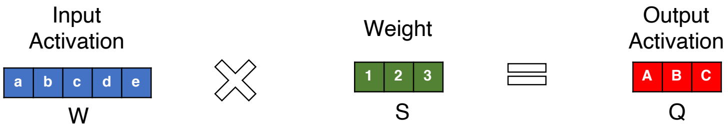
Direct Convolution

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];
            }
          }
        }
        OA[n][k][p][q]= Activation(OA[n][k][p][q]);
      }
    }
  }
}
```

1D Convolution Example



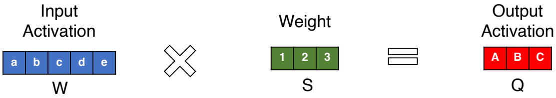

```
for(q=0; q<Q; q++){
  for (s=0; s<S; s++){
    OA[q] += IA[q+s] * W[s];
  }
}
```

Output Stationary (OS) Dataflow

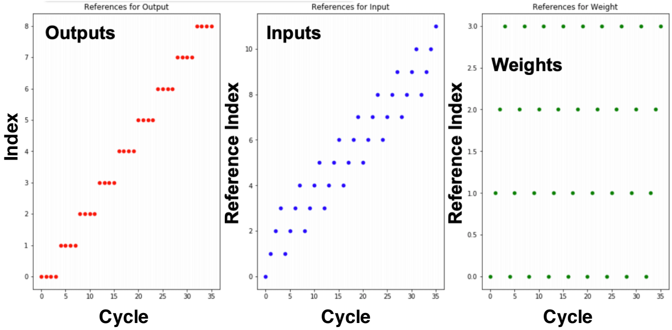
```
for (s=0; s<S; s++){
  for(q=0; q<Q; q++){
    OA[q] += IA[q+s] * W[s];
  }
}
```

Weight Stationary (WS) Dataflow

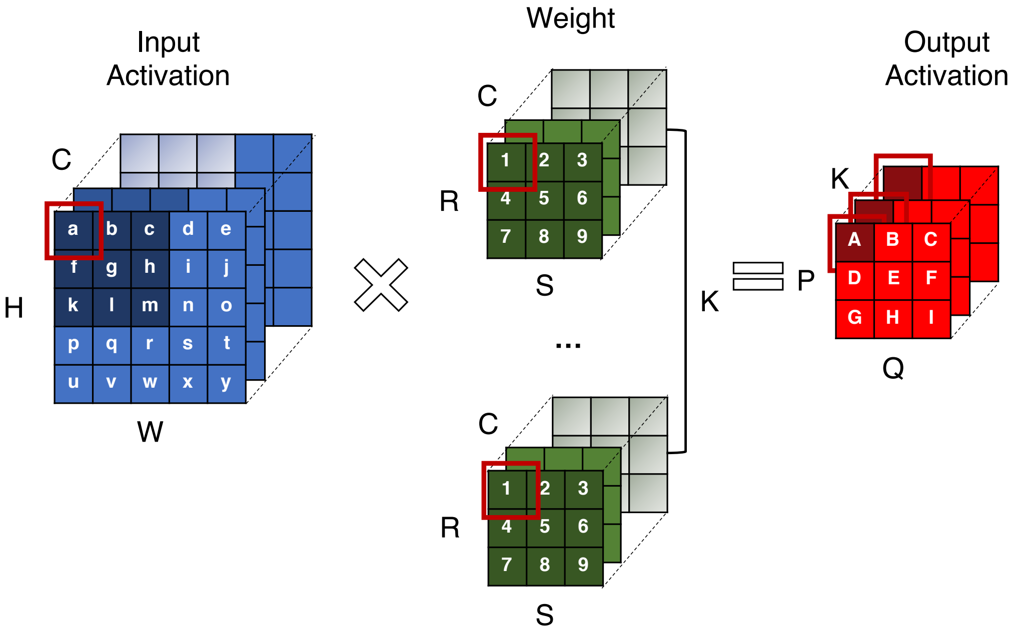
Buffer Access Pattern 1: Output Stationary



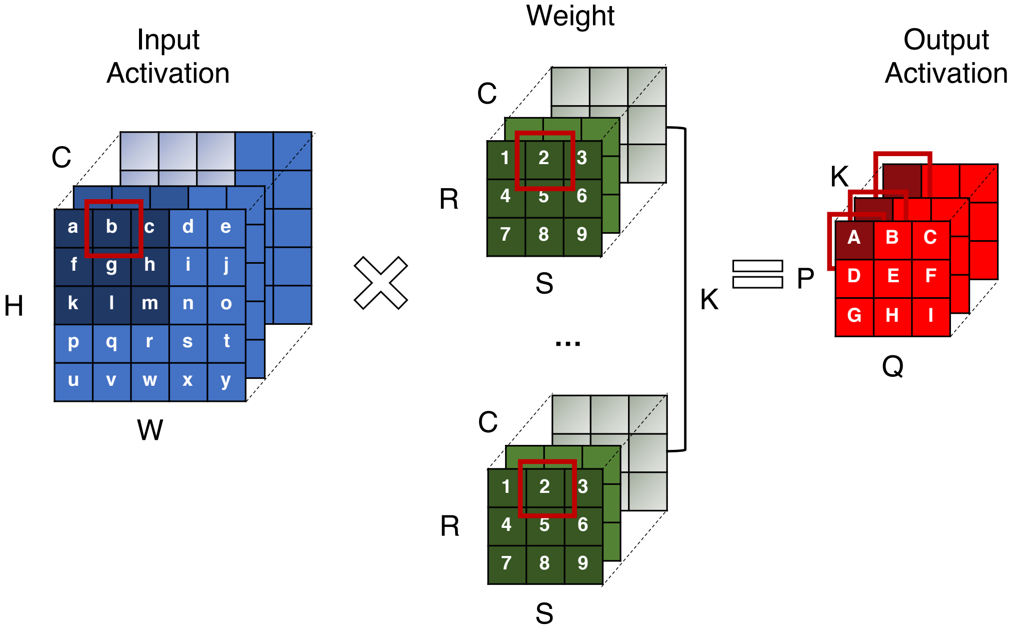

```
for(q=0; q<Q; q++){ // Q =9
  for (s=0; s<S; s++){ // S=4
    OA[q] += IA[q+s] * W[s];
  }
}
```



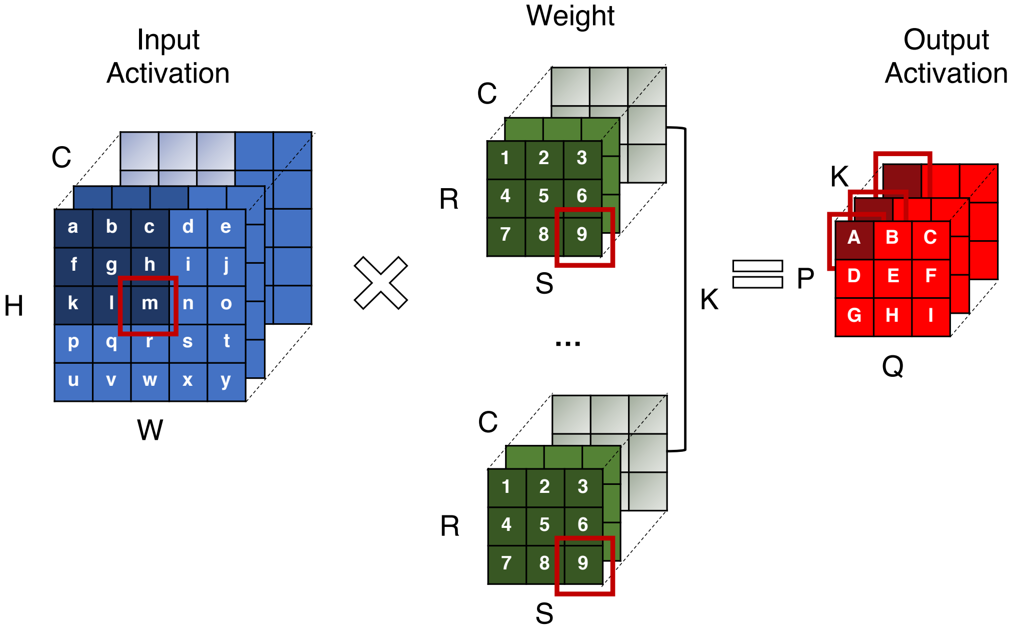
Output Stationary in 3D Convolution Scenario



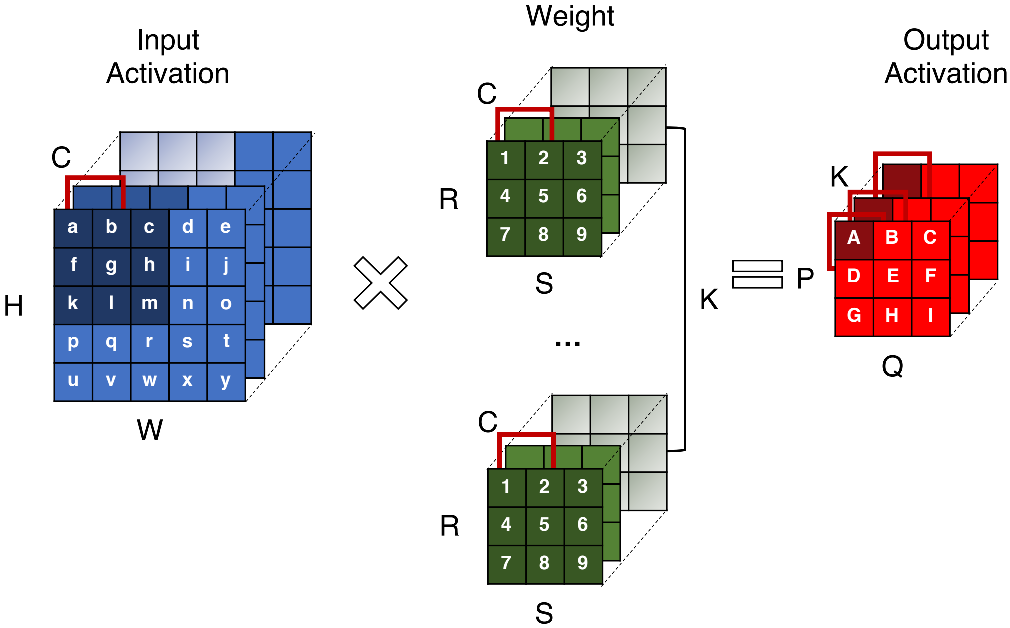
Output Stationary in 3D Convolution Scenario



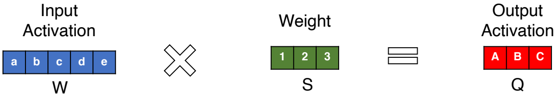
Output Stationary in 3D Convolution Scenario



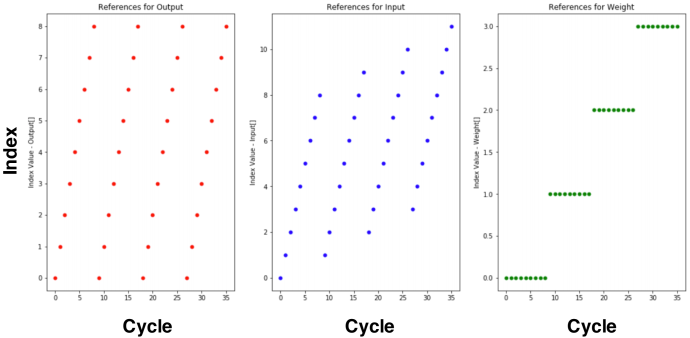
Output Stationary in 3D Convolution Scenario



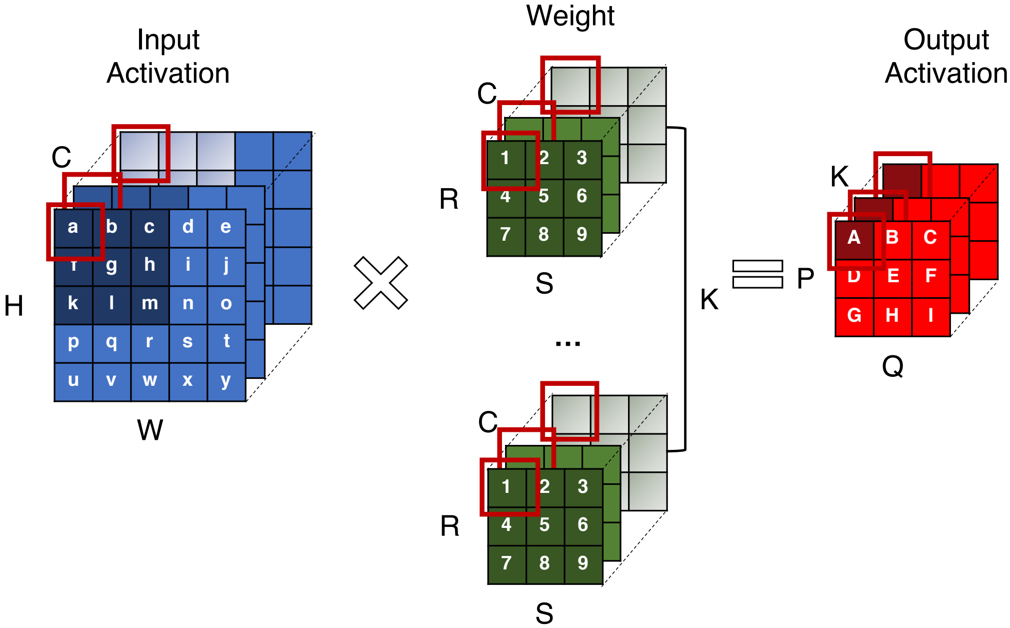
Buffer Access Pattern 2: Weight Stationary



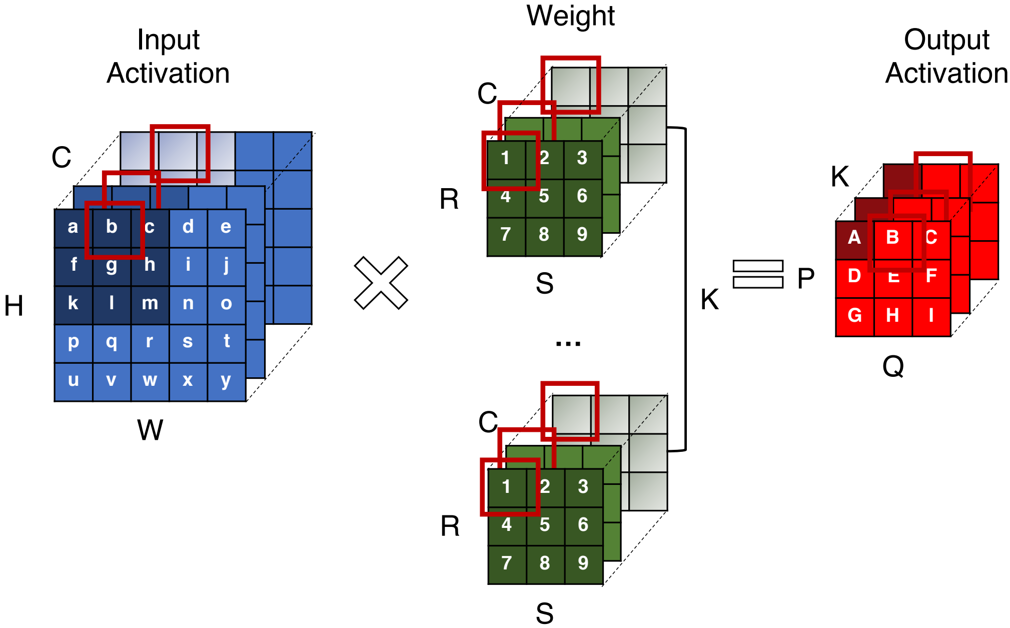

```
for (s=0; s<S; s++){// S=4
  for(q=0; q<Q; q++){// Q =9
    OA[q] += IA[q+s] * W[s];
  }
```



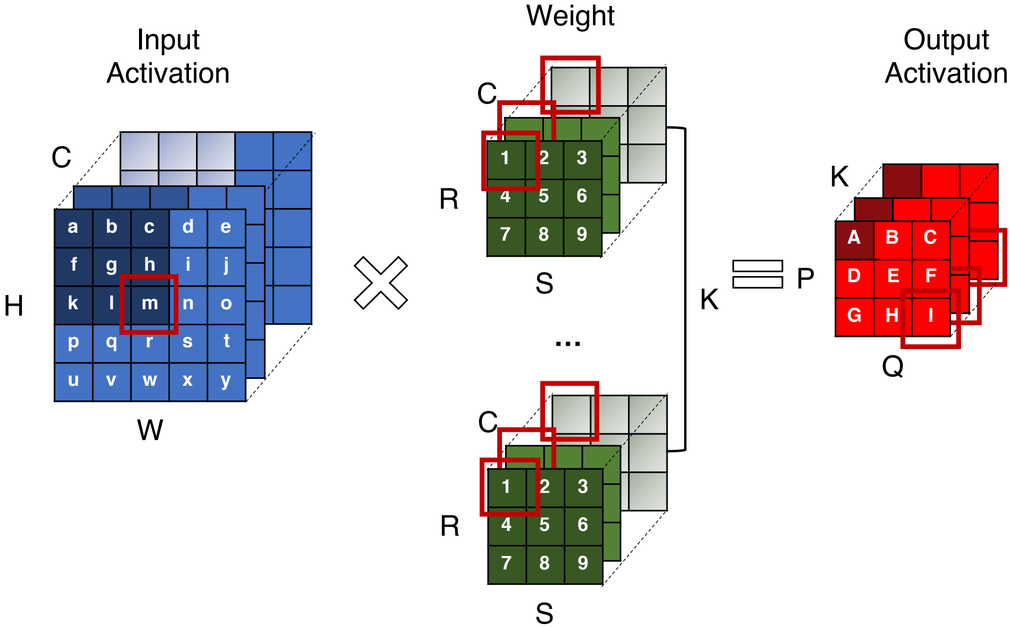
Weight Stationary in 3D Convolution Scenario



Weight Stationary in 3D Convolution Scenario



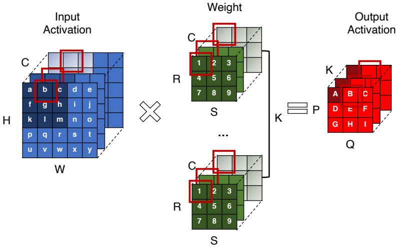
Weight Stationary in 3D Convolution Scenario



Dataflow

- Defines the execution order of the DNN operations in hardware
  - Computation Order
  - Data Movement Order
- Loop nest is a compact way to describe the execution order, i.e., dataflow, supported in hardware.
  - *for*: temporal for, describes the temporal execution order
  - *spatial_for*: describes parallel execution

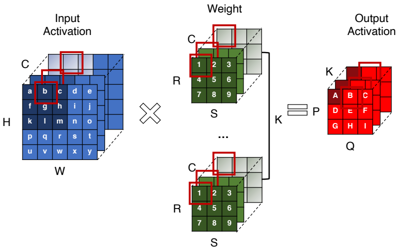
Weight Stationary Dataflow



- What we had before:

```
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];
            }
          }
        }
      }
    }
  }
}
```

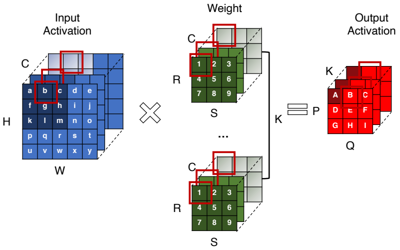
Weight Stationary Dataflow



- Change temporal 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;
            }
          }
        }
      }
    }
  }
}
```

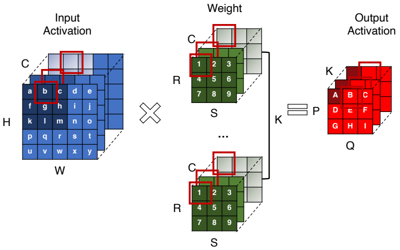
Weight Stationary Dataflow



- Apply spatial parallelism

```
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;
            }
          }
        }
      }
    }
  }
}
```

Weight Stationary Dataflow



- Apply temporal 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;
      }}}}
    }
  }
}
```