CS5120 VLSI System Design, Spring 2024

DNN Mapping Part 2: Tiling for Hardware Structure

黃稚存

Chih-Tsun Huang

cthuang@cs.nthu.edu.tw

國立清華大學
NATIONAL TSING HUA UNIVERSITY

資訊工程學系
Computer Science

Lecture 07

聲明

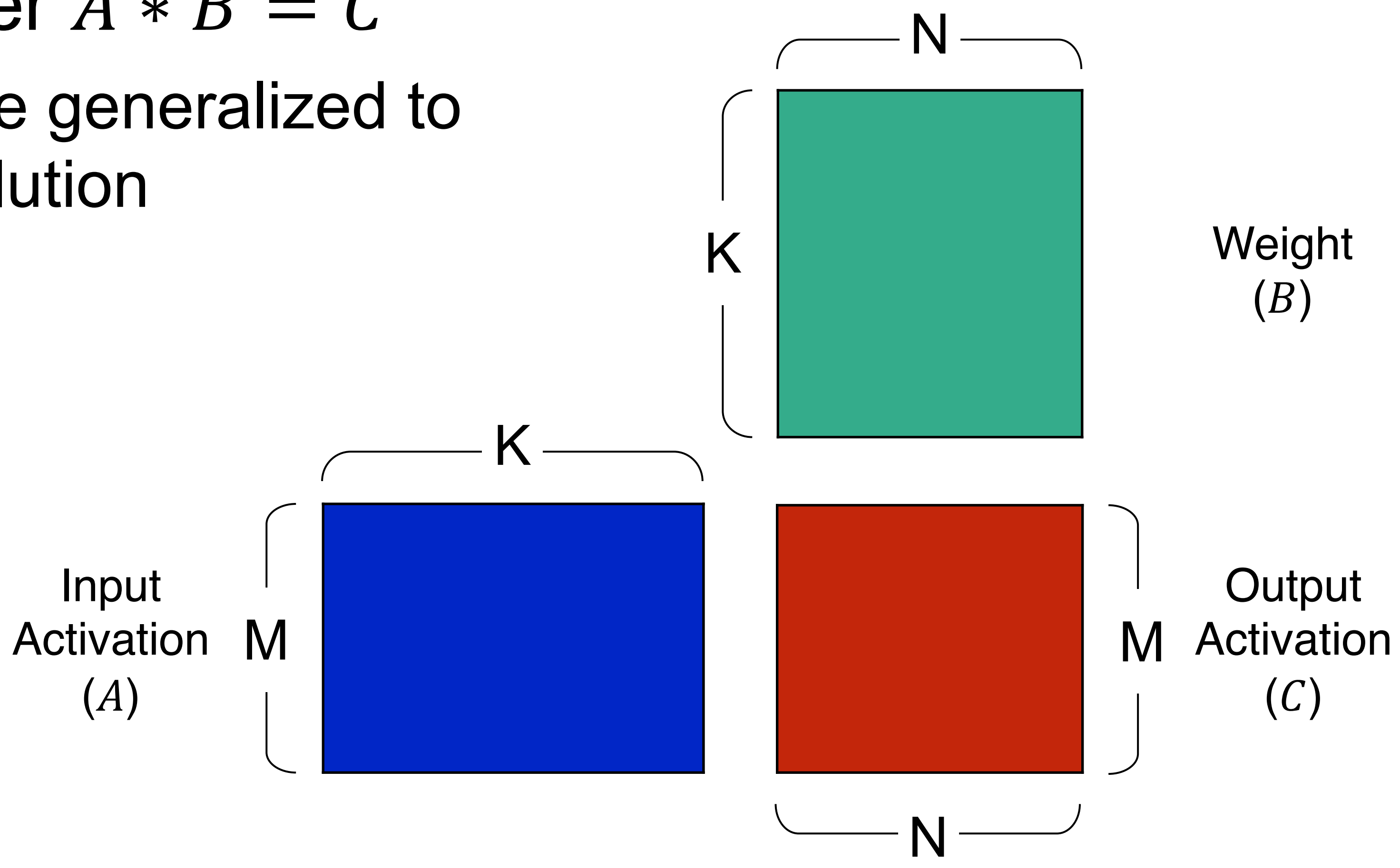
- 本課程之內容 (包括但不限於教材、影片、圖片、檔案資料等)，僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容(例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter, Google Drive, Dropbox 等等)。如有侵權行為，需自負法律責任。

Outline

- Hardware Resource Constraints
- Mapping Space
- Tuning of DNN Mapping

Recap: Matrix Multiplication for Fully-Connected Layer

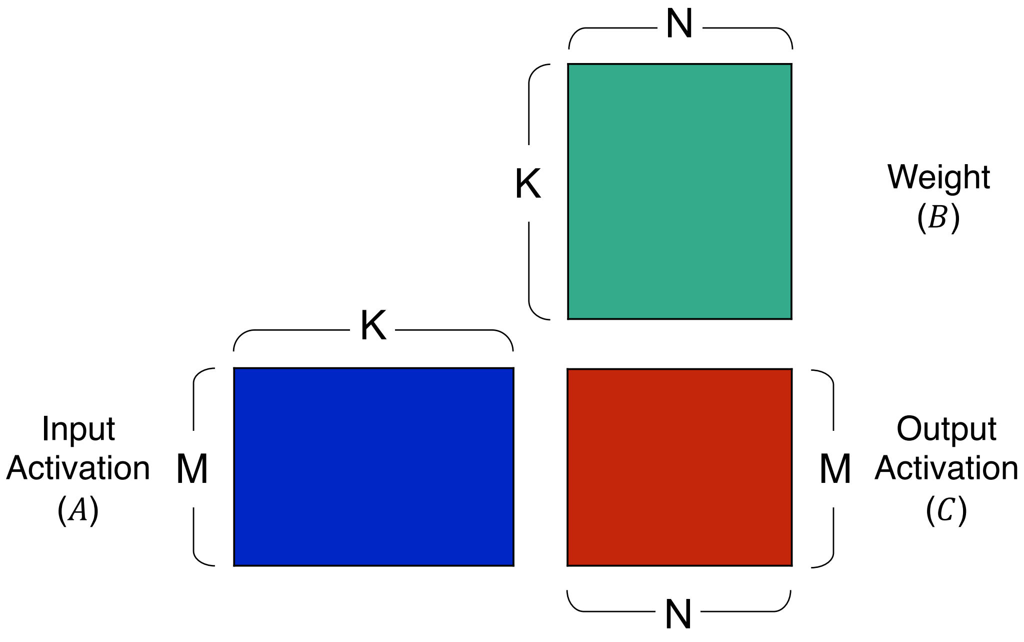
- Consider $A * B = C$
  - Can be generalized to convolution



[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Recap: Loop Nest for Matrix Multiplication

- Consider $A * B = C$



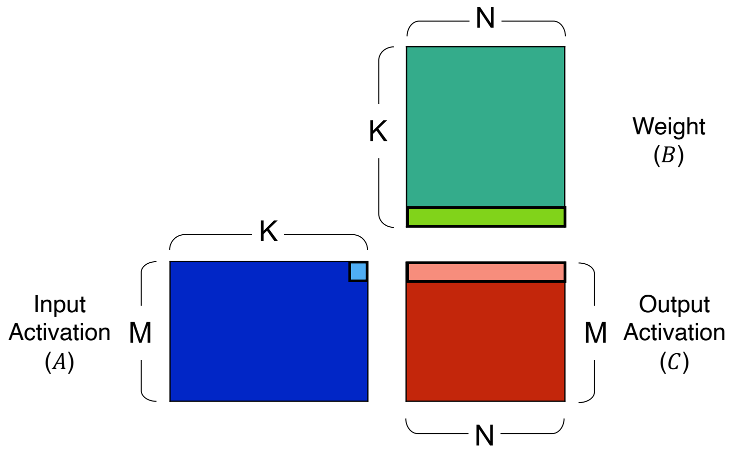

```
for (m=0; m<M; m++) {
  for (n=0; n<N; n++) {
    OA[n,m] = 0;
    for (k=0; k<K; k++) {
      OA[n,m] +=  IA[m, k] * W[k, n];
    }
    OA[n,m] = Activation(OA[n,m]);
  }
}
```

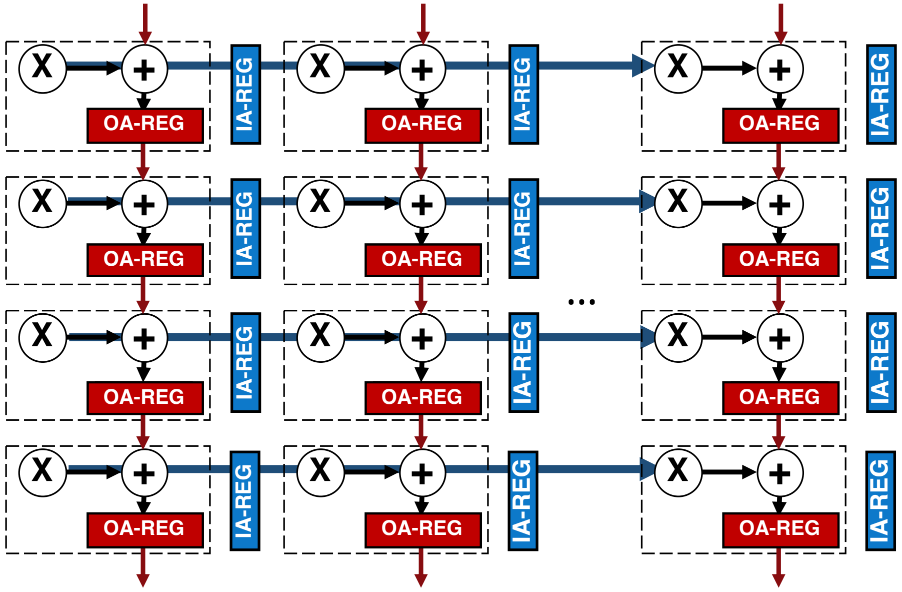
For each output activation

Reduction

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Recap: Mixed Datapath Optimization: TPU






```
for (m=0; m<M; m++) {
  parallel_for (n=0; n<N; n++) {
    OA[n,m] = 0;
    parallel_for (k=0; k<K; k++) {
      OA[n,m] +=  IA[m,k] * W[k,n];
    }
    OA[n,m] = Activation(OA[n,m]);
  }
}
```

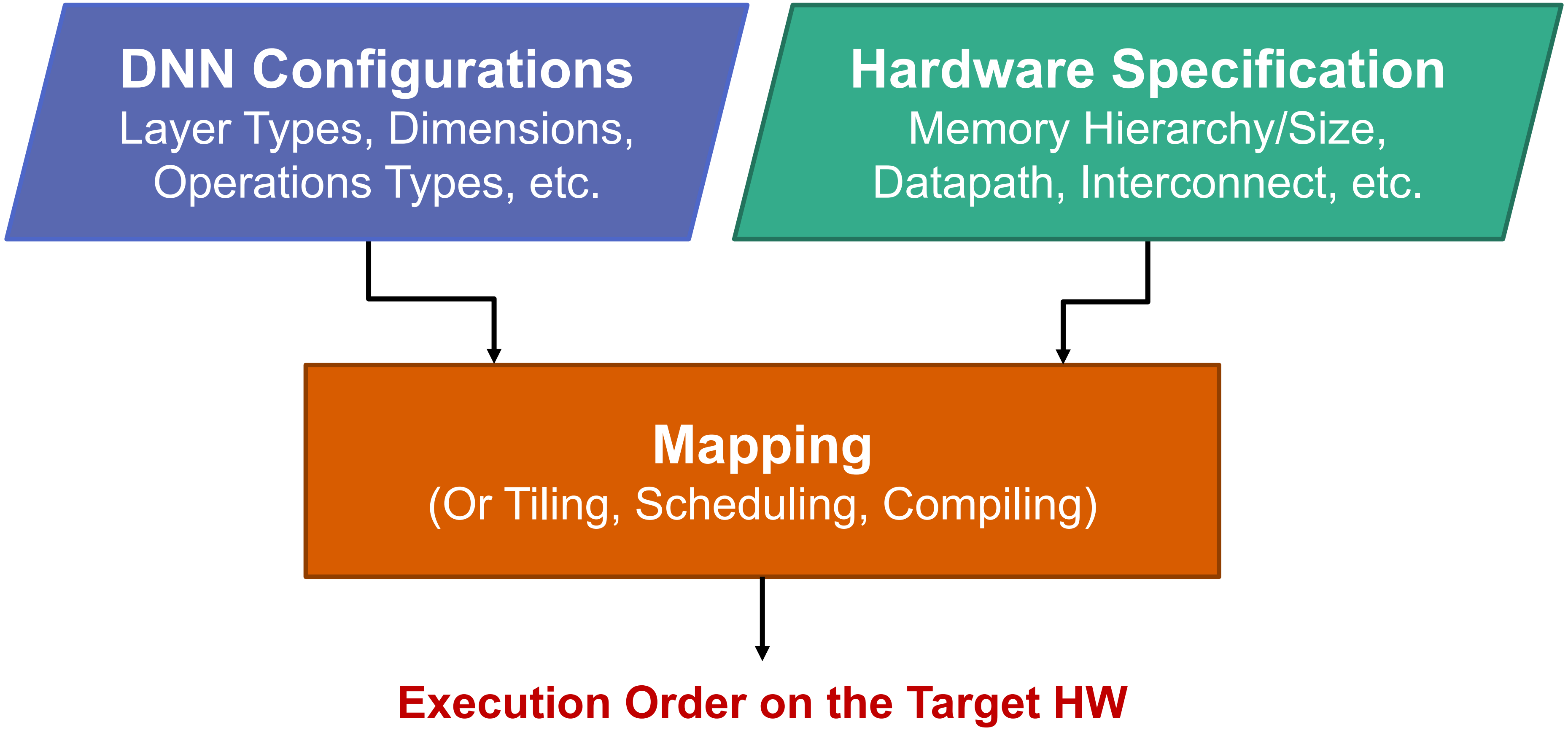
- Systolic accumulation
- Systolic multicast

→ Area vs. scalability

→ Latency vs. pipeline throughput

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

DNN Mapping Problem

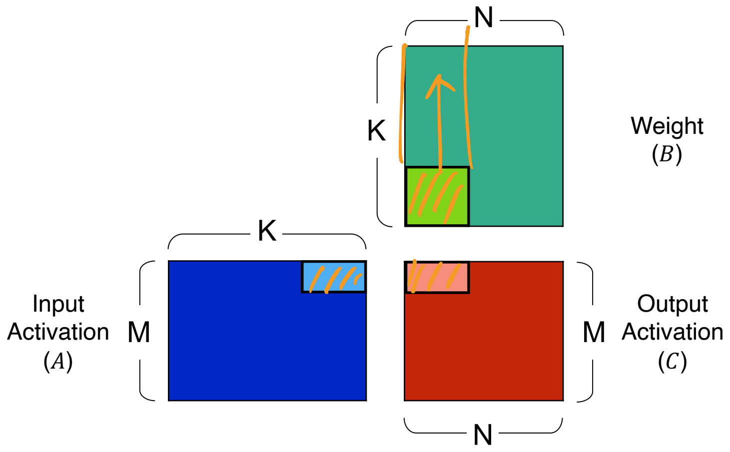


- Mapping objective: efficient for the performance (latency) and/or energy, etc.

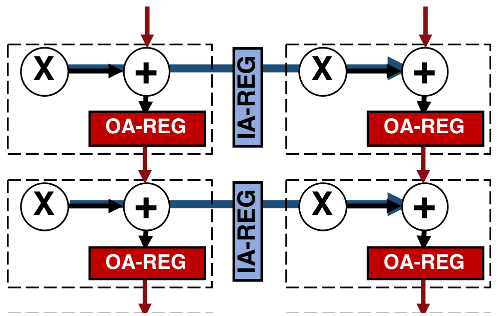
Hardware Resource Constraints

Constraint 1: Systolic Array Size < K and/or N

- DNN Configuration: Matrix Multiplication



- Hardware Specification: Systolic array size: $\boxed{2x2}$



- Notation: K?/N? ➔ loop bounds

$N = N_1 \times N_0$

$K = K_1 \times K_0$

```
for (m=0; m<M; m++) {
  for (n1=0; n1<N1; n1++) {
    OA[n1*N0:(n1+1)*N0,m] = 0;
    for (k1=0; k1<K1; k1++) {
      parallel_for (n0=0; n0<N0; n0++) {
        parallel_for (k0=0; k0<K0; k0++) {
          OA[n1*N0+n0,m] += IA[m,k1*K0+k0]
                    * W[k1*K0+k0,n1*N0+n0];
}}}}}
```

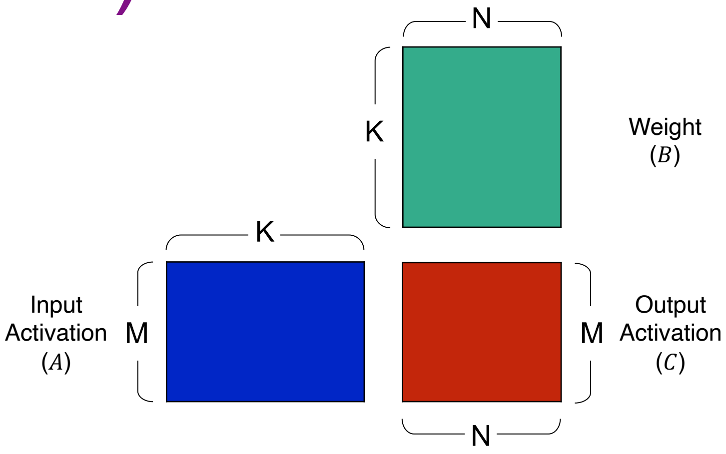
Temporal Tiling

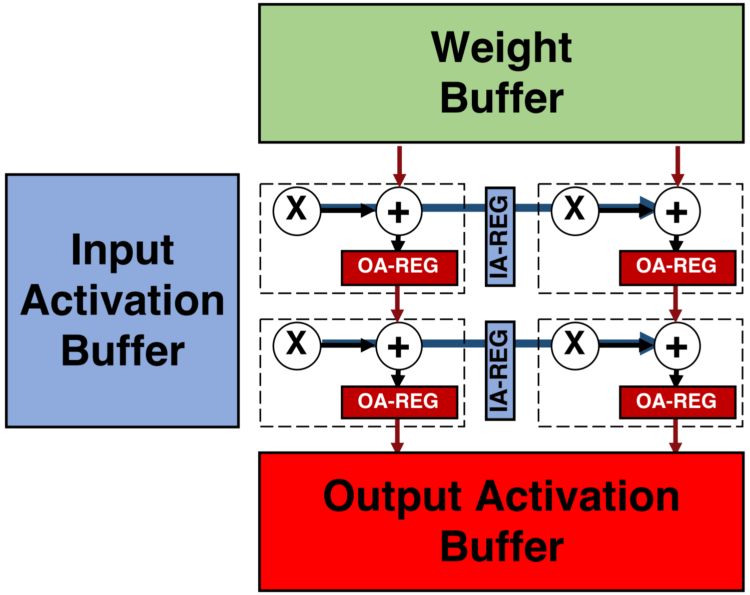
Spatial Tiling

在空間上展開

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Constraint 2: Weight Buffer Size < K * N (1/3)



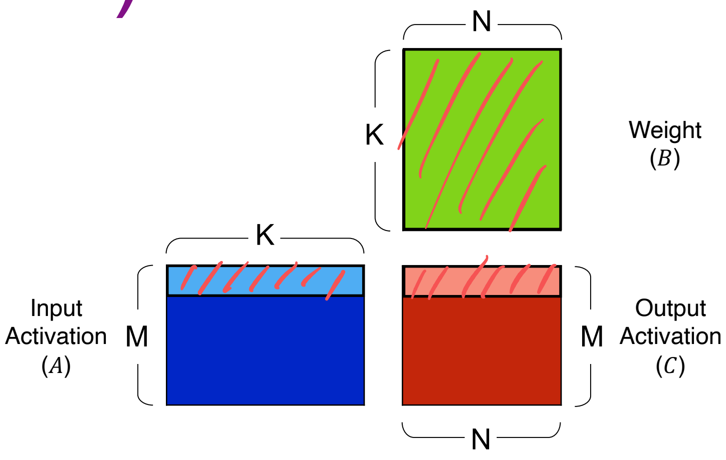


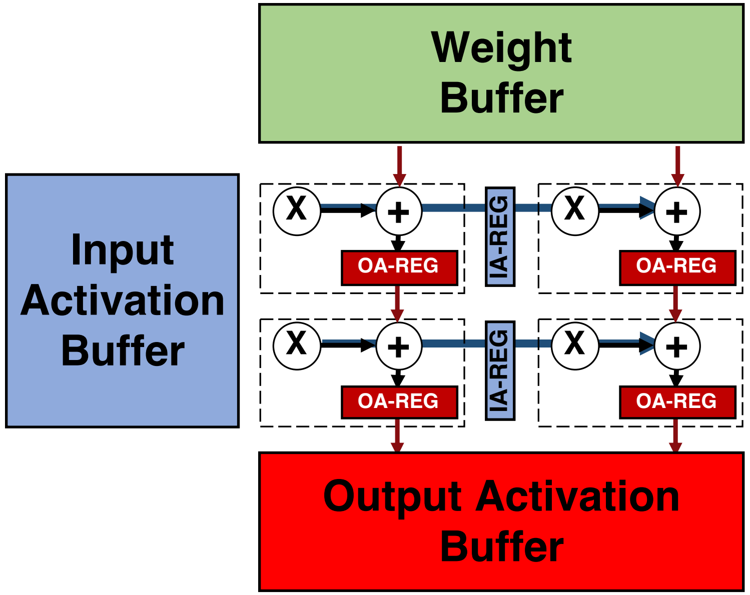
- Hardware Specification:
  - Systolic array size: 2x2
  - Explicit data movement (or data orchestration)

```
for (m=0; m<M; m++) {

  for (n1=0; n1<N1; n1++) {
    OA[n1*N0:(n1+1)*N0,m] = 0;
    for (k1=0; k1<K1; k1++) {
      parallel_for (n0=0; n0<N0; n0++) {
        parallel_for (k0=0; k0<K0; k0++) {
          OA[n1*N0+n0,m] += IA[m,k1*K0+k0]
                          * W[k1*K0+k0,n1*N0+n0];
  }}}}

}
```

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Constraint 2: Weight Buffer Size < K * N (2/3)





[Source: Prof. Sophia Shao, EE290-2, Berkeley]

- Hardware Specification:
  - Systolic array size: 2x2
  - Explicit data movement (or data orchestration)
  - Weight, input activation, output activation buffer sizes

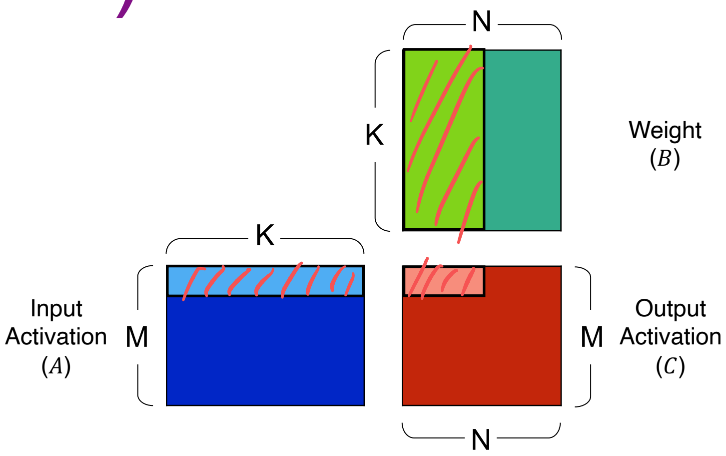
move data to buffer

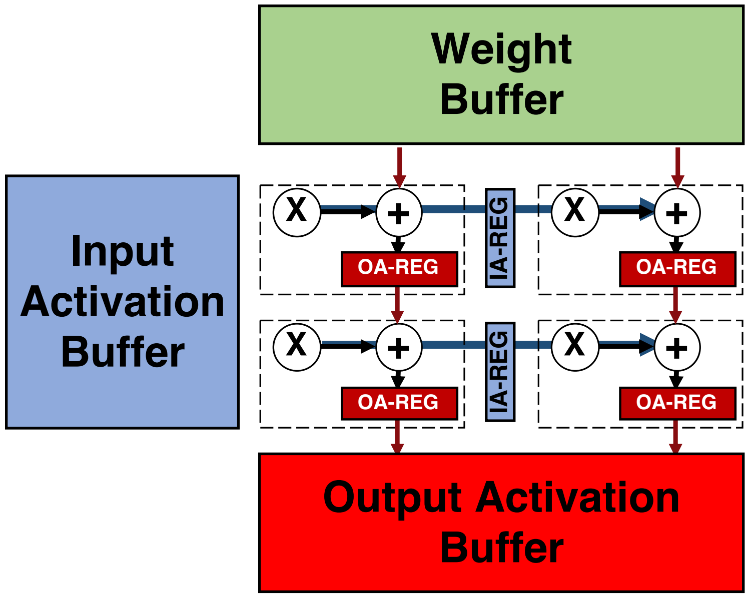
```
for (m=0; m<M; m++) {
  mvin(W[0:K,0:N]);         // W buffer >= K*N
  mvin(IA[m:m+1,0:K]);     // IA buffer >= 1*K
  for (n1=0; n1<N1; n1++) {
    OA[n1*N0:(n1+1)*N0,m] = 0;
    for (k1=0; k1<K1; k1++) {
      parallel_for (n0=0; n0<N0; n0++) {
        parallel_for (k0=0; k0<K0; k0++) {
          OA[n1*N0+n0,m] += IA[m,k1*K0+k0]
                           * W[k1*K0+k0,n1*N0+n0];
  }}}}
  mvout(OA[0:N,m:m+1]);   // OA buffer >= 1*N
}
```

整個weight matrix

一行input activation

Constraint 2: Weight Buffer Size < K * N (3/3)






```
for (m=0; m<M; m++) {
  // IA buffer: 1*K
  mvin(IA[m:m+1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer: N1*N0*K < K*N
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m:m+1]=0;
    for (n1=0; n1<N1; n1++) {
      for (k1=0; k1<K1; k1++) {
        parallel_for (n0=0; n0<N0; n0++) {
          parallel_for (k0=0; k0<K0; k0++) {
            OA[n2*N1*N0+n1*N0+n0,m]
              += IA[m,k1*K1+k0]
              * W[k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0,m:m+1]);
}}
```

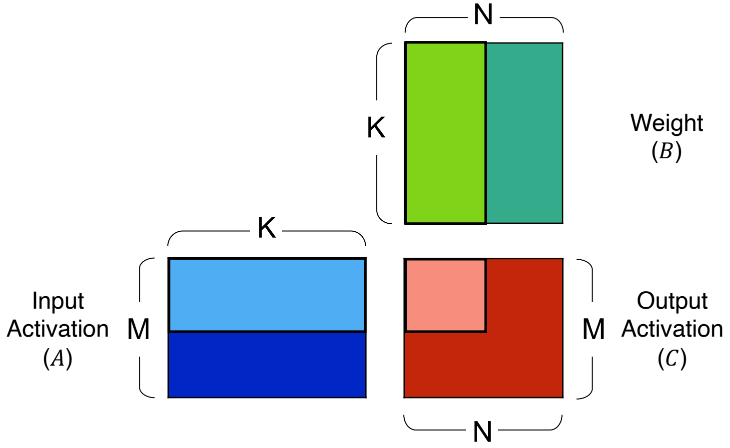
$N = N_2 \times N_1 \times N_0$

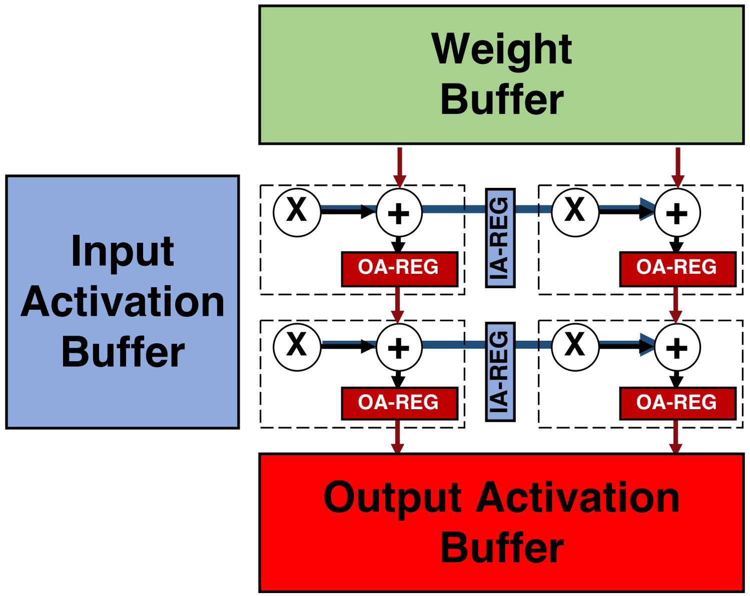
放不下整個 weight matrix

$N_1 \times N_0$

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Constraint 3: Input Buffer Size > 1 * K

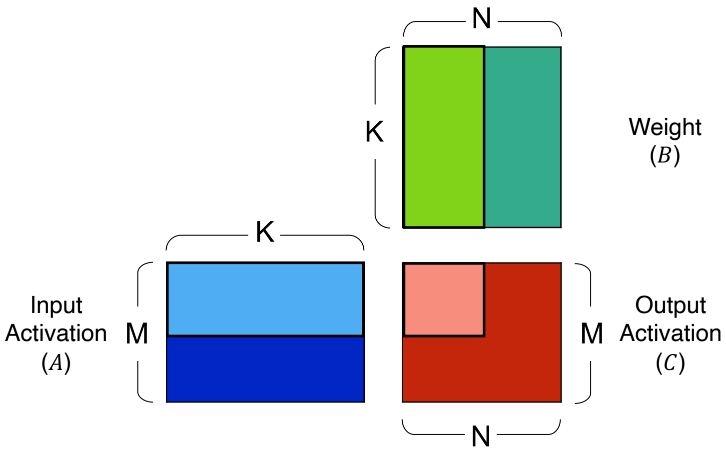




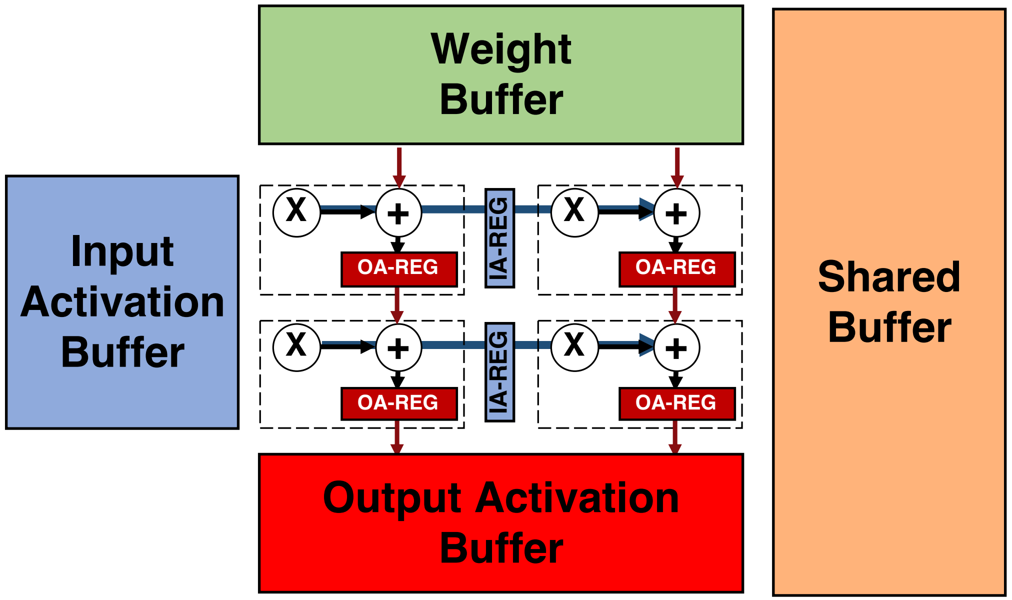

```
for (m2=0; m2<M2; m2++) {   → input activation buffer 放大M1倍
  // IA buffer: M1*K > 1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer: N1*N0*K < K*N
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m2*M1:(m2+1)*M1]=0;
    for (m1=0; m1<M1; m1++) { 每一輪做M1×K個
      for (n1=0; n1<N1; n1++) {
        for (k1=0; k1<K1; k1++) {
          parallel_for (n0=0; n0<N0; n0++) {
            parallel_for (k0=0; k0<K0; k0++) {
              OA[n2*N1*N0+n1*N0+n0,m2*M1+m1]
                += IA[m2*M1+m1,k1*K1+k0]
                * W [k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}}
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0, m2*M1:(m2+1)*M1]);
}}
```

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Constraint 4: Adding Another Shared Buffer



clusters
grouping




```
for (m3=0; m3<M3; m3++) {
 for (n3=0; n3<N3; n3++) {
  // Shared buffer blocking
   for (m2=0; m2<M2; m2++) {
     // IA buffer stores: M1*K
     mvin(IA[…:…]);
     for (n2=0; n2<N2; n2++) {
       // W buffer stores: N1*N0*K
       mvin(W[…:…,…:…]);
       OA[…:…,…:…]=0;
       for (m1=0; m1<M1; m1++) {
         for (n1=0; n1<N1; n1++) {
           for (k1=0; k1<K1; k1++) {
                …
     }}}}}
     mvout(OA[…:…,…:…]);
   }}
}}
```


[Source: Prof. Sophia Shao, EE290-2, Berkeley]

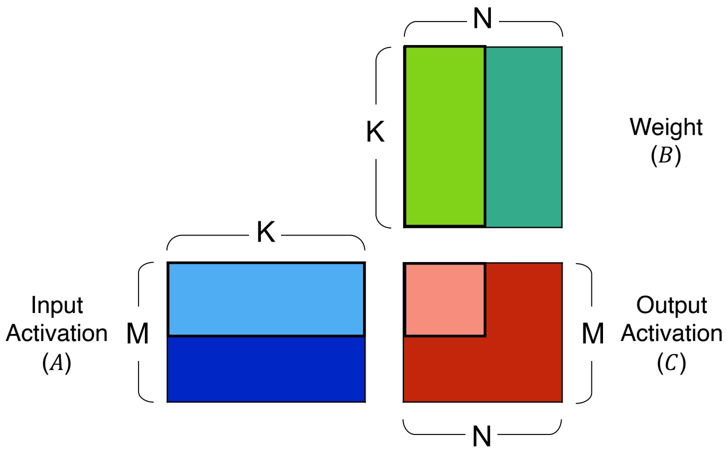
Mapping Space

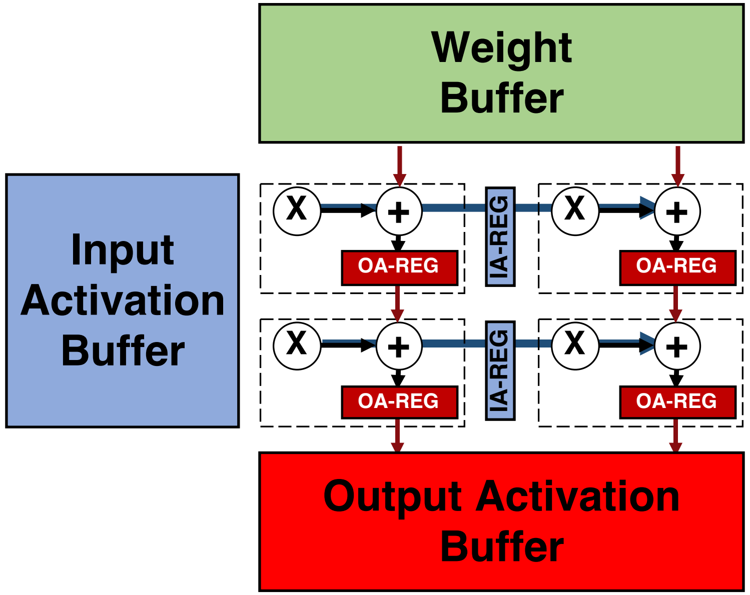
- Loop ordering
- Loop bound
- Spatial choice

Mapping Dimensions

- Loop ordering:
  - Which index goes to the inner/outer loop?
- Loop bounds:
  - What are the loop bounds (i.e., N?/K?/M?) for each loop?
- Spatial choice:
  - Which loop should be spatial/temporal?
    - Data/Model Parallelism

Mapping Problem Example

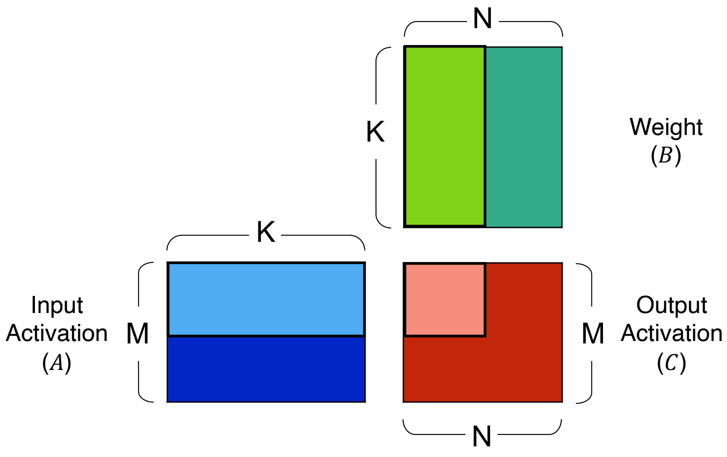


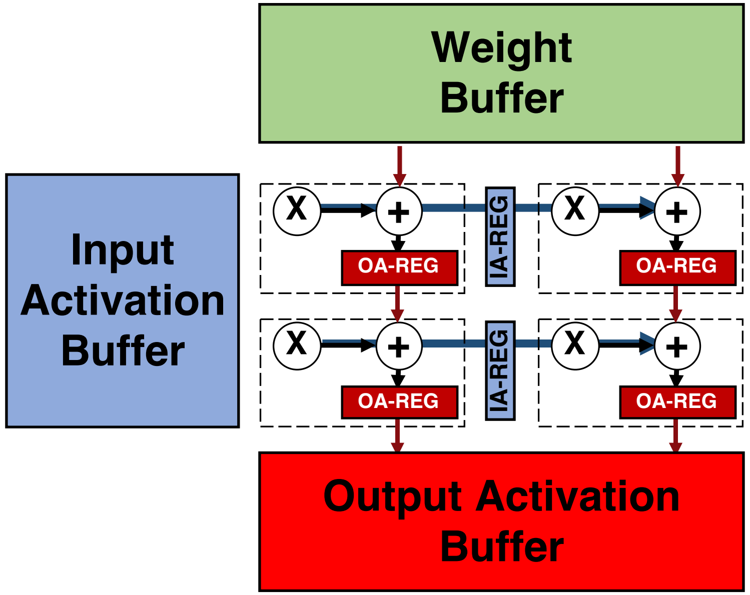



```
for (m2=0; m2<M2; m2++) {
  // IA buffer: M1*K > 1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer: N1*N0*K < K*N
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m2*M1:(m2+1)*M1]=0;
    for (m1=0; m1<M1; m1++) {
      for (n1=0; n1<N1; n1++) {
        for (k1=0; k1<K1; k1++) {
          parallel_for (n0=0; n0<N0; n0++) {
            parallel_for (k0=0; k0<K0; k0++) {
              OA[n2*N1*N0+n1*N0+n0,m2*M1+m1]
                += IA[m2*M1+m1,k1*K1+k0]
                * W [k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}}
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0,m2*M1:(m2+1)*M1]);
}}
```

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Loop Bound






```
for (m2=0; m2<M2; m2++) {
  // IA buffer: M1*K > 1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer: N1*N0*K < K*N
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m2*M1:(m2+1)*M1]=0;
    for (m1=0; m1<M1; m1++) {
      for (n1=0; n1<N1; n1++) {
        for (k1=0; k1<K1; k1++) {
          parallel_for (n0=0; n0<N0; n0++) {
            parallel_for (k0=0; k0<K0; k0++) {
              OA[n2*N1*N0+n1*N0+n0,m2*M1+m1]
                += IA[m2*M1+m1,k1*K1+k0]
                * W [k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}}                                compute_matmul();
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0,m2*M1:(m2+1)*M1]);
}}
```

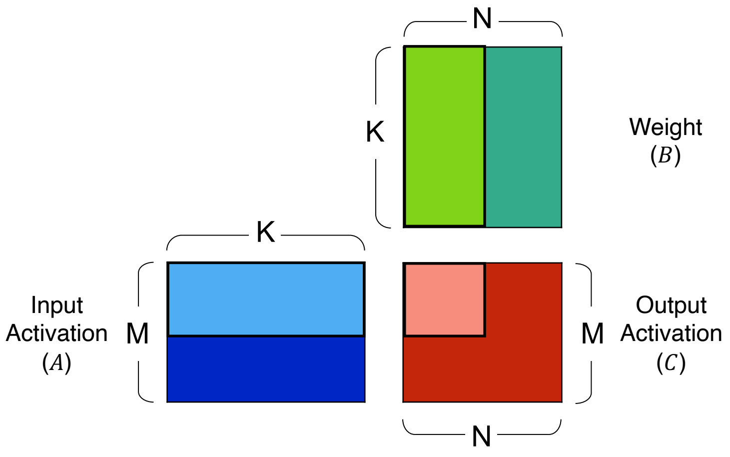
$N = N_2 \times N_1 \times N_0$

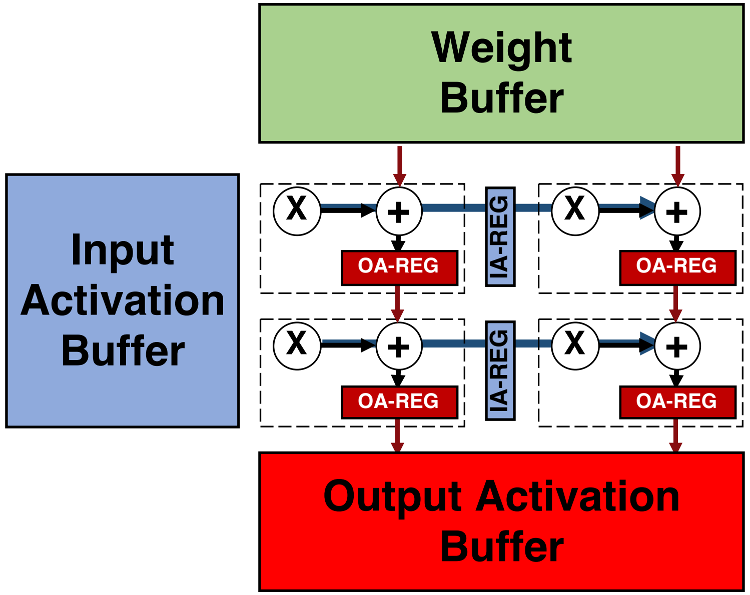
$M = M_1 \times M_2$

$K = K_1 \times K_0$

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Loop Ordering (1/2)






```
for (m2=0; m2<M2; m2++) {
  // IA buffer: M1*K > 1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer: N1*N0*K < K*N
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    compute_matmul(*W, *IA, *OA,
                   N2, N1, N0,
                   M2, M1,
                   K1, K0,
                   m2, n2);
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0,
          m2*M1:(m2+1)*M1]);
}}
```

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Loop Ordering (2/2)

$N = N_2 \times N_1 \times N_0$

$M = M_1 \times M_2$

$K = K_1 \times K_0$

- Option 1: Loops m2 → n2

```
for (m2=0; m2<M2; m2++) {
  // IA buffer: M1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer: N1*N0*K
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    compute_matmul(*W, *IA, *OA,
                   ...
                   m2, n2);
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0,
             m2*M1:(m2+1)*M1]);
}}
```

$M_2 \times M_1 \times K = M \times K$

$M_2 \times N_2 \times N_1 \times N_0 \times K$

$\| M_2 \times N \times K$

buffer內所有資料的移動量

IA Movement: M * K

W Movement: M2 * N * K

- Option 2: Loops n2 → m2

```
for (n2=0; n2<N2; n2++) {
  // W buffer: N1*N0*K
  mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
  for (m2=0; m2<M2; m2++) {
    // IA buffer: M1*K
    mvin(IA[m2*M1:(m2+1)*M1,0:K]);
    compute_matmul(*W, *IA, *OA,
                   ...
                   m2, n2);
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0,
             m2*M1:(m2+1)*M1]);
}}
```

$N_2 \times N_1 \times N_0 \times K = N \times K$

$N_2 \times M_2 \times M_1 \times K$

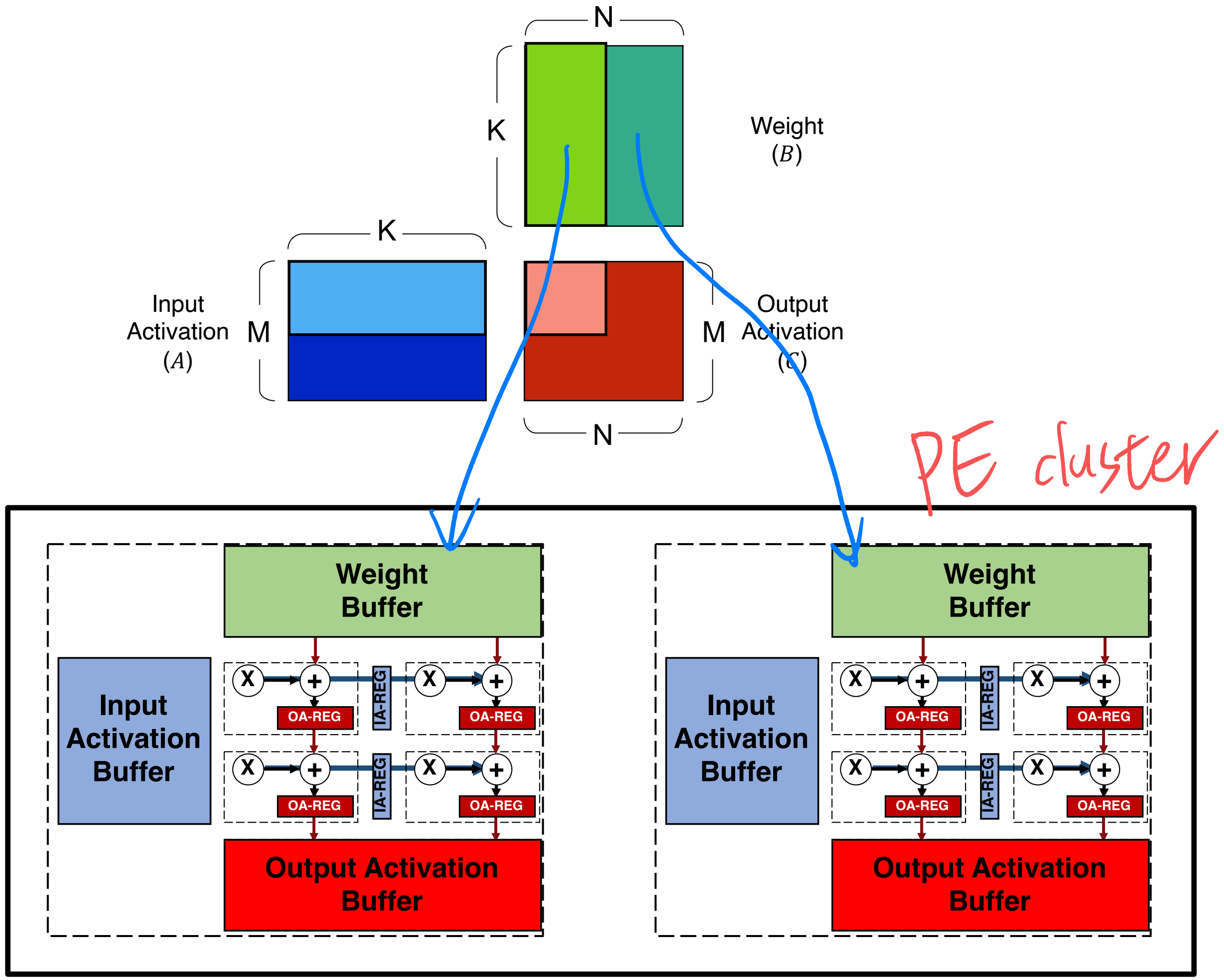
$\| N_2 \times M \times K$

IA Movement: N2 * M * K

W Movement: N * K

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

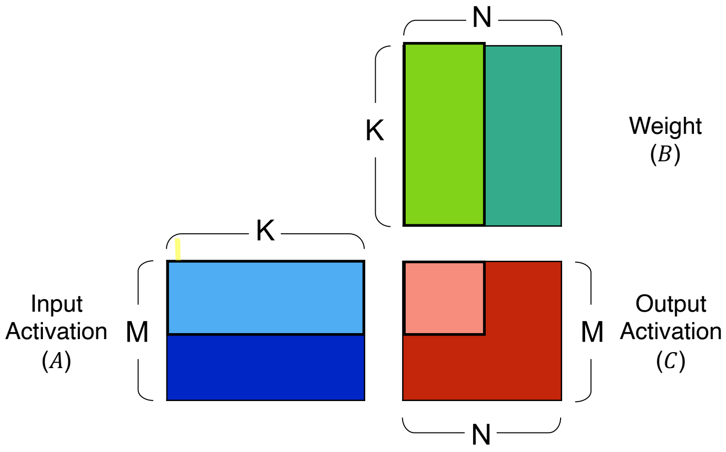
Spatial Choice: Model Parallelism

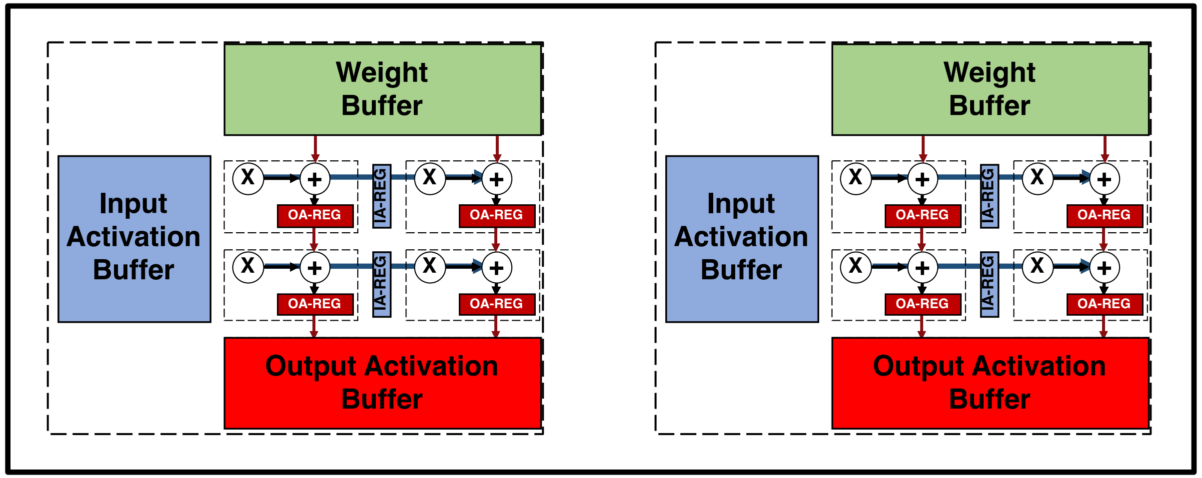



```
parallel_for (n2=0; n2<N2; n2++) {
  // Model Parallelism
  mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
  for (m2=0; m2<M2; m2++) {
    mvin(IA[m2*M1:(m2+1)*M1,0:K]);
    compute_matmul(*W, *IA, *OA,
                   ...
                   m2, n2);
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0,
          m2*M1:(m2+1)*M1]);
}}
```


[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Spatial Choice: Data Parallelism





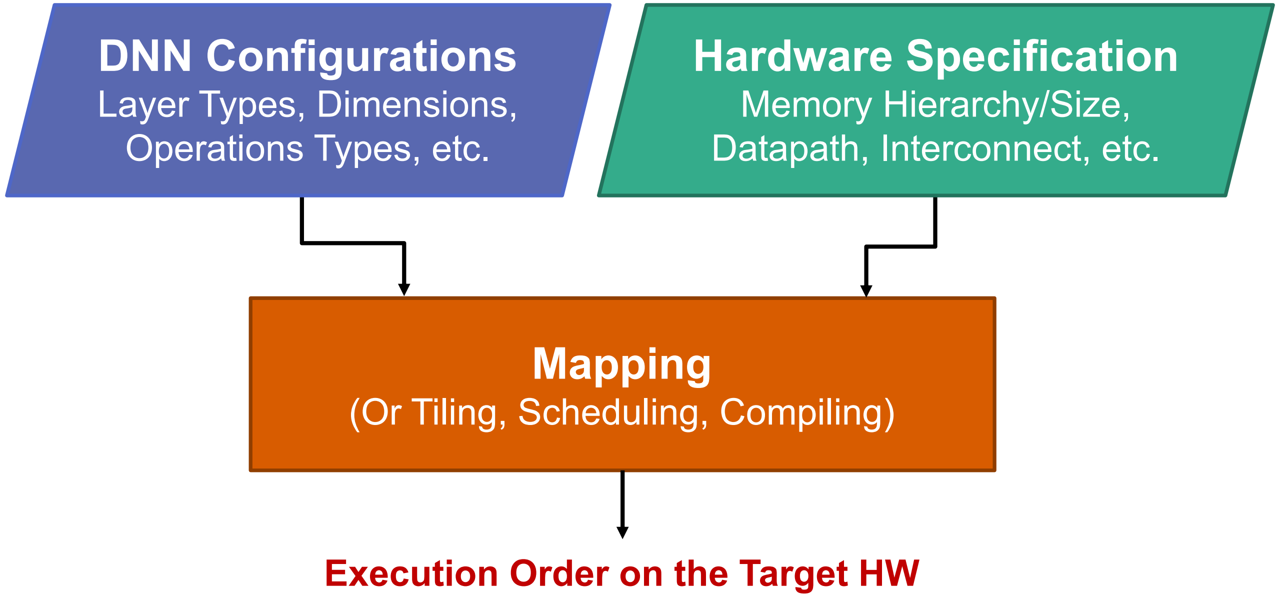

```
parallel_for (m2=0; m2<M2; m2++) {
  // Data Parallelism
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    compute_matmul(*W, *IA, *OA,
                   ...
                   m2, n2);
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0,
          m2*M1:(m2+1)*M1]);
}}
```


[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Tuning of DNN Mapping

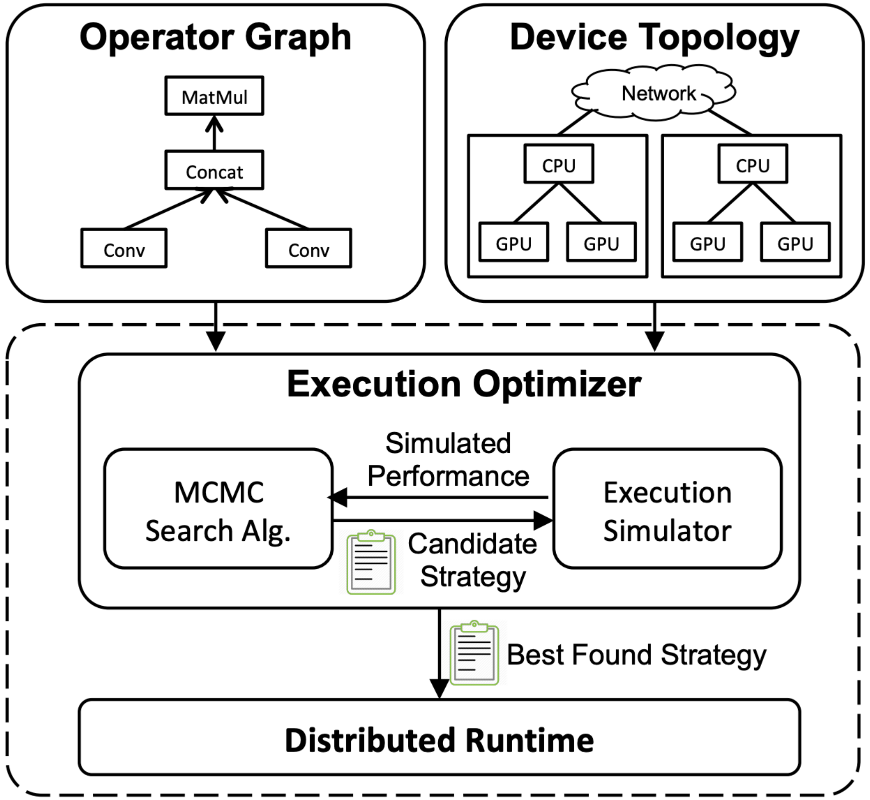
Mapping Dimensions

- DNN mapping problem ➔ an optimization problem
- Given:
  - DNN dimensions (N, H, W, C, R, S, K, stride, padding)
  - Hardware specifications (dataflow, memory hierarchy)
- Objective:
  - An optimal loop nest that minimizes latency and/or energy
    - Both temporal and spatial execution order
- Approaches:
  - Exhaustive search
  - Random search
  - Learning-based algorithms



Example: FlexFlow, SysML'2018

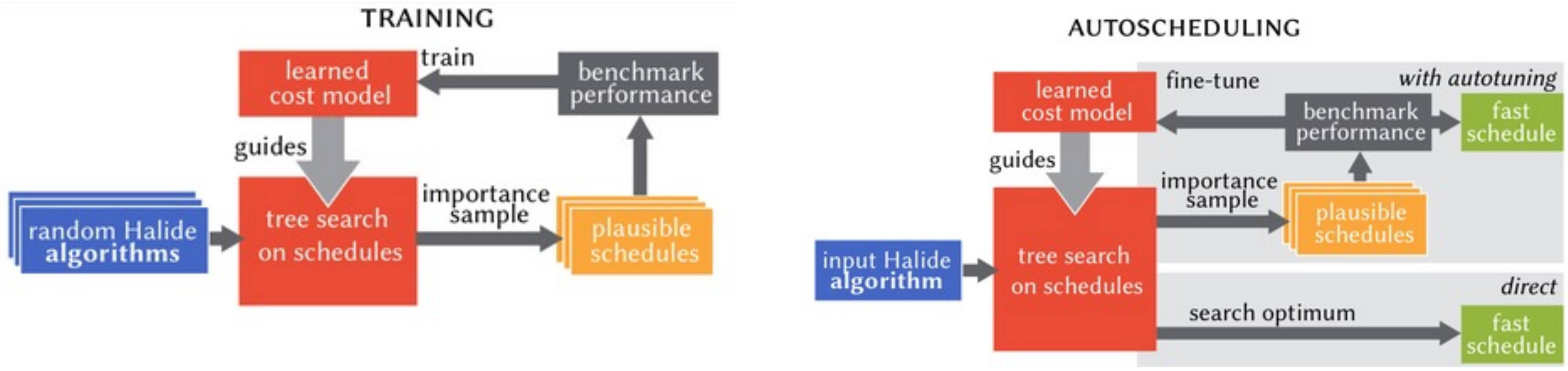
- "The optimizer uses a MCMC (Markov Chain Monte Carlo) search algorithm to explore the space of possible parallelization strategies and iteratively proposes candidate strategies that are evaluated by an execution simulator. "



Beyond Data and Model Parallelism for Deep Neural Networks, SysML 2018

Example: Halide, SIGGRAPH'2019

- "We generate schedules for Halide programs using tree search over the space of schedules guided by a learned cost model and optional autotuning. The cost model is trained by benchmarking thousands of randomly-generated Halide programs and schedules. The resulting code significantly outperforms prior work and human experts."

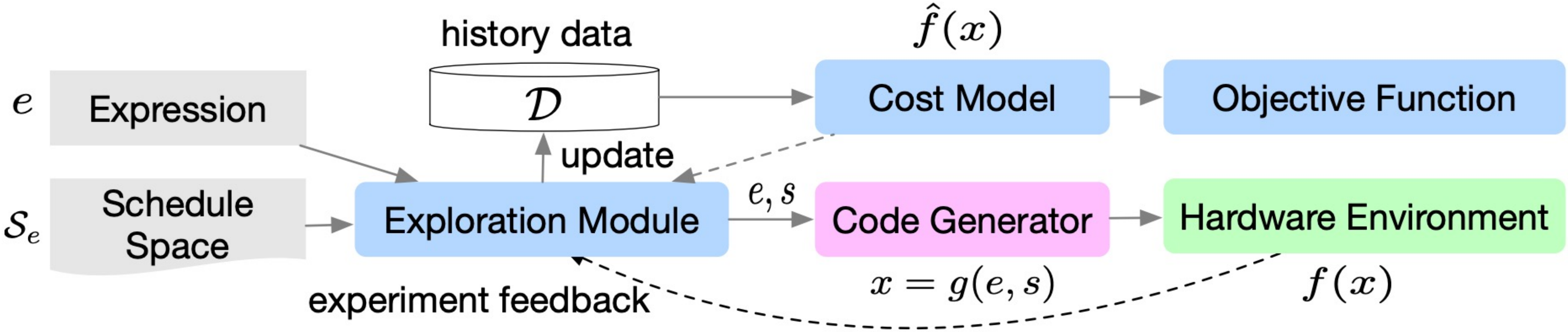


Learning to Optimize Halide with Tree Search and Random Programs, *SIGGRAPH 2019*

Example: TVM, NeurIPS'2018

auto tuning，找出一個最適合的mapping方式

- Learn domain-specific statistical cost models to guide the search of tensor operator implementations over billions of possible program variants
- Further accelerate the search using effective model transfer across workloads



Learning to Optimize Tensor Programs, *NeurIPS'2018*

Summary

- Temporal and Spatial Mapping based on hardware constraints
  - Memory hierarchy
  - Parallelism
- Tile the loops to improve reuse and parallelism
  - Loop ordering
  - Loop bounds
  - Spatial choices
- Navigate the large mapping space
  - Finding an optimized solution
  - Getting to the solution fast enough