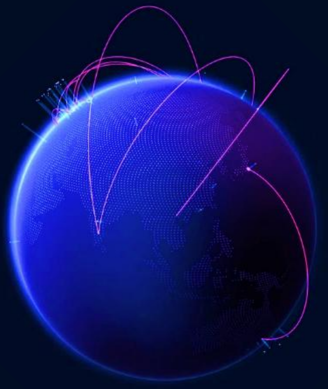
# A Convolution Neural Network Accelerator Design With Weight Mapping and Pipeline Optimization

DAC - 2023



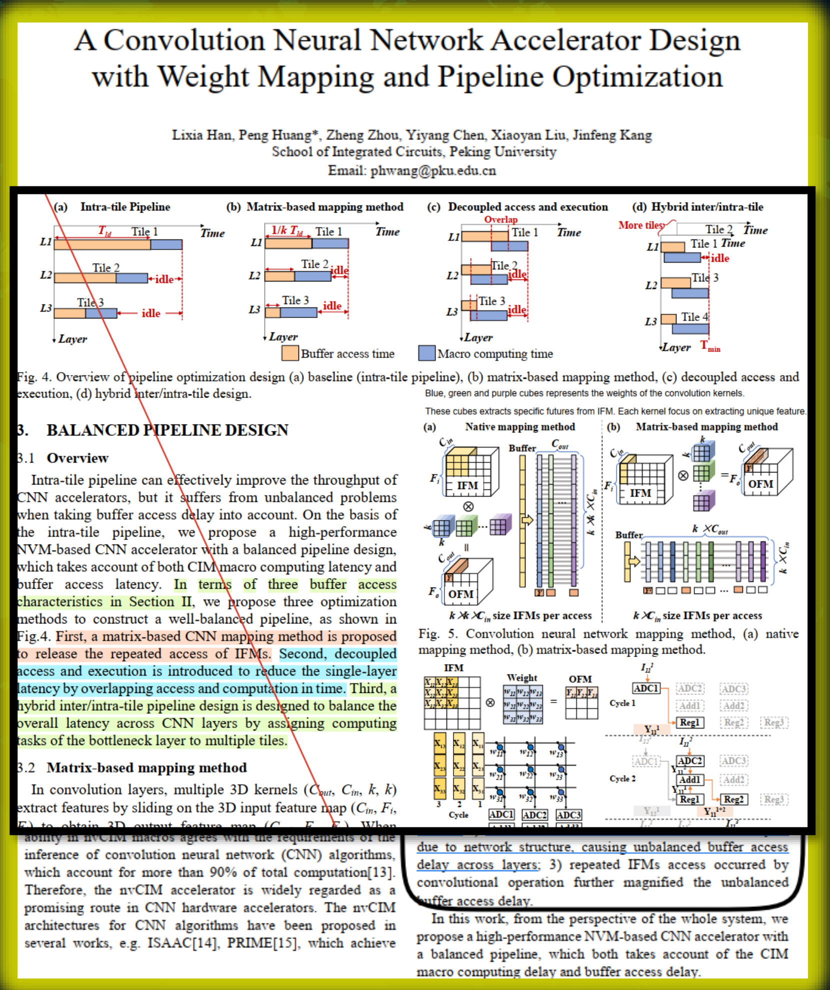
AI Accelerators

NVM-Based CIM

Matrix-Based Mapping

Optimized Pipeline

Evaluation



A Convolution Neural Network Accelerator Design with Weight Mapping and Pipeline Optimization

Lixia Han, Peng Huang*, Zheng Zhou, Yiyang Chen, Xiaoyan Liu, Jinfeng Kang
School of Integrated Circuits, Peking University
Email: phwang@pku.edu.cn

Fig. 4. Overview of pipeline optimization design (a) baseline (intra-tile pipeline), (b) matrix-based mapping method, (c) decoupled access and execution, (d) hybrid inter/intra-tile design.

## 3. BALANCED PIPELINE DESIGN

### 3.1 Overview

Intra-tile pipeline can effectively improve the throughput of CNN accelerators, but it suffers from unbalanced problems when taking buffer access delay into account. On the basis of the intra-tile pipeline, we propose a high-performance NVM-based CNN accelerator with a balanced pipeline design, which takes account of both CIM macro computing latency and buffer access latency. In terms of three buffer access characteristics in Section II, we propose three optimization methods to construct a well-balanced pipeline, as shown in Fig.4. First, a matrix-based CNN mapping method is proposed to release the repeated access of IFMs. Second, decoupled access and execution is introduced to reduce the single-layer latency by overlapping access and computation in time. Third, a hybrid inter/intra-tile pipeline design is designed to balance the overall latency across CNN layers by assigning computing tasks of the bottleneck layer to multiple tiles.

### 3.2 Matrix-based mapping method

In convolution layers, multiple 3D kernels ($C_{out}$, $C_{in}$, $k$, $k$) extract features by sliding on the 3D input feature map ($C_{in}$, $F_i$,

Blue, green and purple cubes represents the weights of the convolution kernels.

These cubes extracts specific futures from IFM. Each kernel focus on extracting unique feature.

(a) Native mapping method
(b) Matrix-based mapping method
$C_{in}$
$F_i$
IFM
Buffer
$C_{out}$
$F_o$
OFM
$k$
$k \times k \times C_{in}$
$k \times C_{out}$
$k \times C_{in}$
$k \times k \times C_{in}$ size IFMs per access
$k \times C_{in}$ size IFMs per access

Fig. 5. Convolution neural network mapping method, (a) native mapping method, (b) matrix-based mapping method.

ability in nvCIM macros agrees with the requirements of the inference of convolution neural network (CNN) algorithms, which account for more than 90% of total computation[13]. Therefore, the nvCIM accelerator is widely regarded as a promising route in CNN hardware accelerators. The nvCIM architectures for CNN algorithms have been proposed in several works, e.g. ISAAC[14], PRIME[15], which achieve

due to network structure, causing unbalanced buffer access delay across layers; 3) repeated IFMs access occurred by convolutional operation further magnified the unbalanced buffer access delay.

In this work, from the perspective of the whole system, we propose a high-performance NVM-based CNN accelerator with a balanced pipeline, which both takes account of the CIM macro computing delay and buffer access delay.

CMP 634 – Computer Architecture

Kürşat Çakal

# Problem

CNNs dominate modern AI applications, require **intensive computation** and **memory usage** (power, energy)

AI Accelerator is a **specialized computer system**; which targets to **optimize** computation and memory needs

- Cloud Computing (TPU, NPU, GPU)
- Internet of Things
- Sensor-Driven Tasks
- Robotics

Major problem is the **Latency** and **Efficiency** of current CNN Accelerators

- The buffer access amount is **huge (**14M IFMs/OFMs and 162M partial sums, VGG19 for CIM is 16Kbit**)**

- **Repeated** IFM (Input Feature Map) **access** by convolution operations **cause buffer delays**

- Computational and memory demands **varias** for **different layers**
    - Different **for Shallow** and **Deep Layers** in a CNN.

# Literature Review

Efficient VMM ability in **nvCIM** macros agrees with the requirements of the (CNN) algorithms.

Therefore, the **nvCIM accelerator** is widely regarded as a **promising** route in CNN hardware accelerators.

- **ISAAC** ([14]): (422 Cite Paper, 43 Cite Patent, 20000+ Read, 2016)
  - Introduced **raw pipelines** for layer-parallel computation.
  - Achieved >10× – 14× throughput improvement **but faced idle resources** in deeper layers.

- **PRIME** ([15]): (1825 Cite Paper, 23 Cite Patent, 20000+ Read, 2016)
  - Achieved >**100 TOPS throughput** for nvCIM.
  - **Struggles** with **layer imbalances** and static pipelines.

- **Fully Hardware Implemented CNN Algorihms** ([16, 17]): (1396 Cite 77000+ Access, 369 Cite, 97000+ Access)
  - Verified in principles and exhibited **high energy efficiency** by **accuracy loses**.

- **Intra-Tile Pipelines** ([18, 19]): (70 Cite + 132 Cite, 10000 Read+)
  - Divided layers into tile, focused only on intra-layer balance, **ignoring inter-layer bottlenecks**.

# Literature Review

Existing nvCIM pipeline designs face **critical challenges**:

- **Layer-Level Bottlenecks**: shallow layers often **stucks** and take longer to process large IFMs, over deep layers.

- **Hardware Utilization:** Raw/Intra-tile Pipelines **leave hardware resources idle** in deeper layers due to their **lower memory** and **computational demands**.

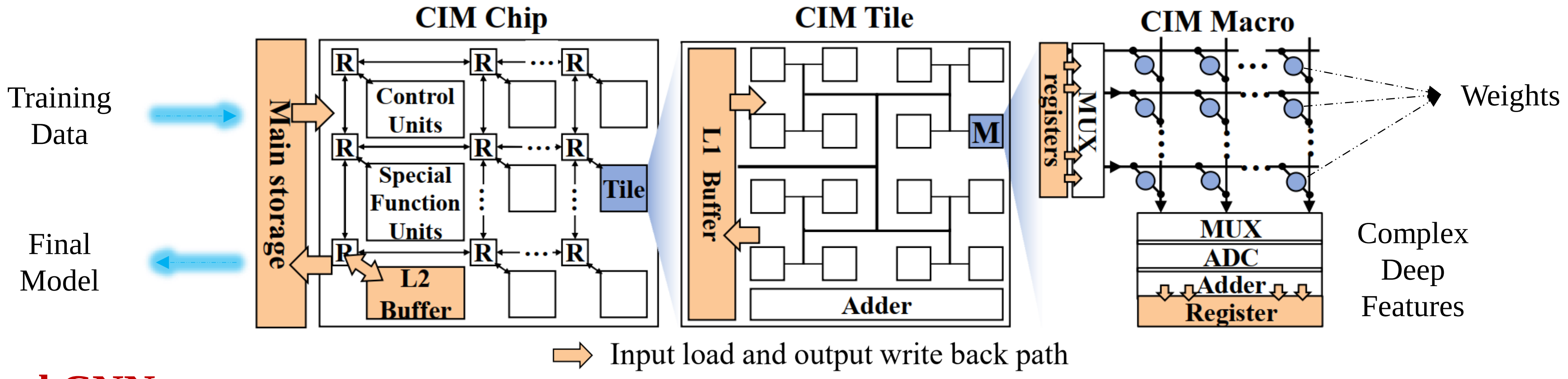
- **Inconsistent Layer Demands**: The previous pipelines optimizes a single layer, but does not handle imbalances across layers. Varying parameters across CNN cause imbalances which reduce **throughput** and **scalability**.

**CNN Operational Characteristics** are miss-discussed **which caused** by buffer access of IFM/OFM, partial sums, load/writeback.

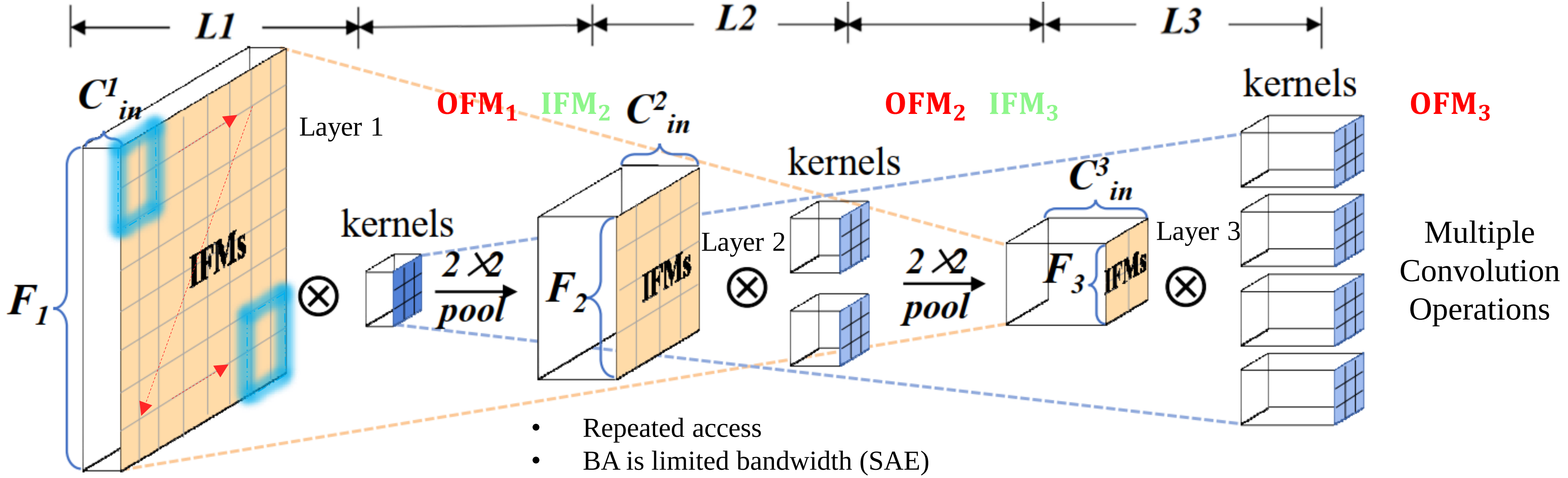
Authors address the **imbalanced hardware utilization** in previous methods and propose systematic approach.

# Overview

## NVM-Based CIM Diagram



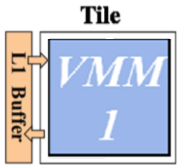
## Typical CNN



- Repeated access
- BA is limited bandwidth (SAE)
- BA delay varias → Layers
  - kernel and IFM size
  - channel dimension

# Proposed Study

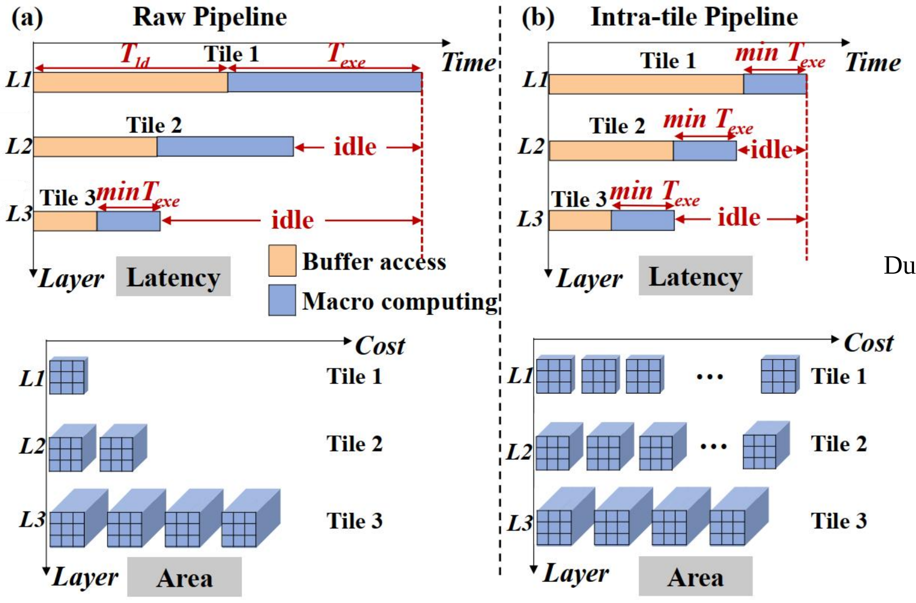
## Previous Pipeline Design

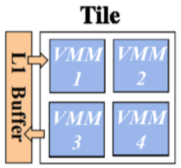


Low throughput.

High latency due to dependent sequential operations.

Suffers from significant idle time between layers.





Duplicated Weights more tiles unit are active on different slices of IFM.

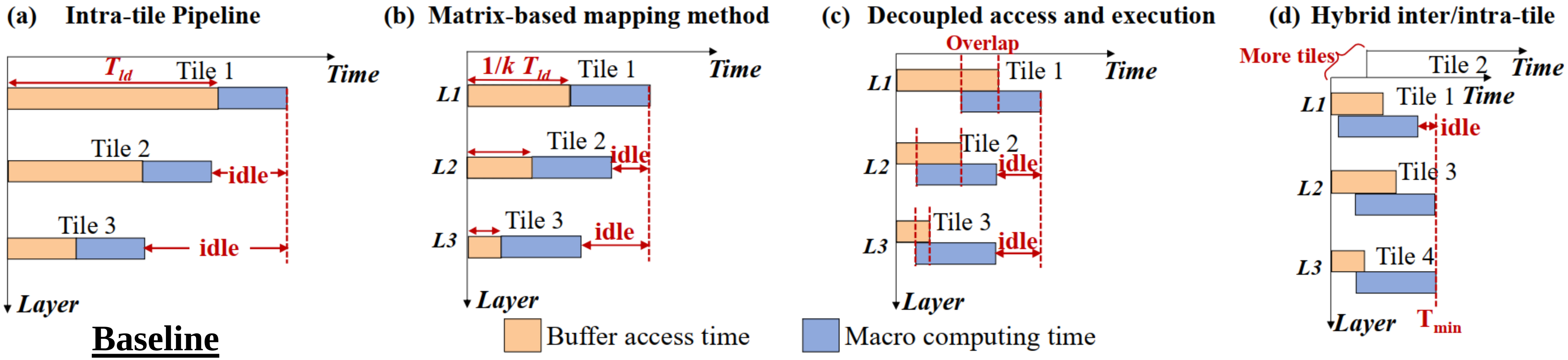
Layer Level Parallelism

Reduced computational delays.

Suffers from buffer access delays.

Suffers from cost.

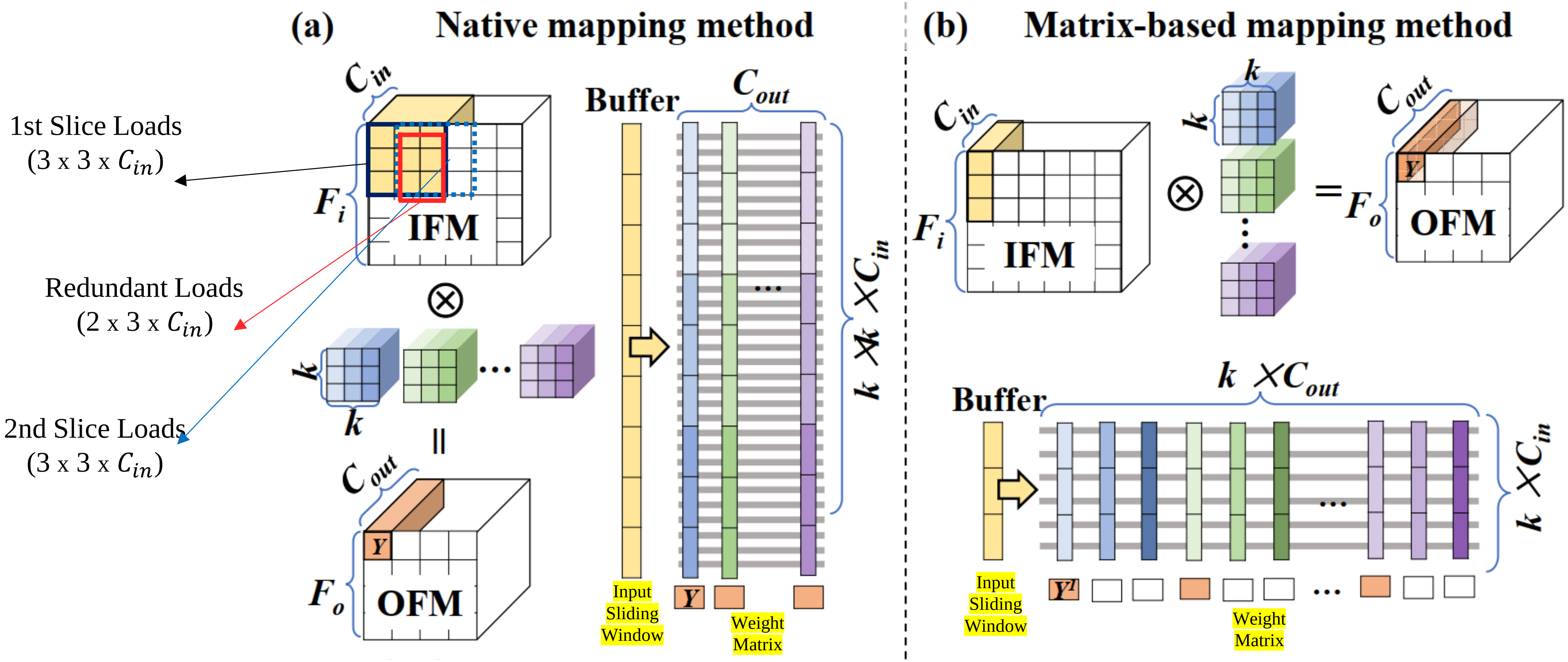
## Pipeline Optimization Design



**Baseline**

# Proposed Study

## Matrix-Based Mapping Method



1st Slice Loads
(3 x 3 x $C_{in}$)

Redundant Loads
(2 x 3 x $C_{in}$)

2nd Slice Loads
(3 x 3 x $C_{in}$)

k x k x $C_{in}$ size IFMs per access

k x $C_{in}$ size IFMs per access

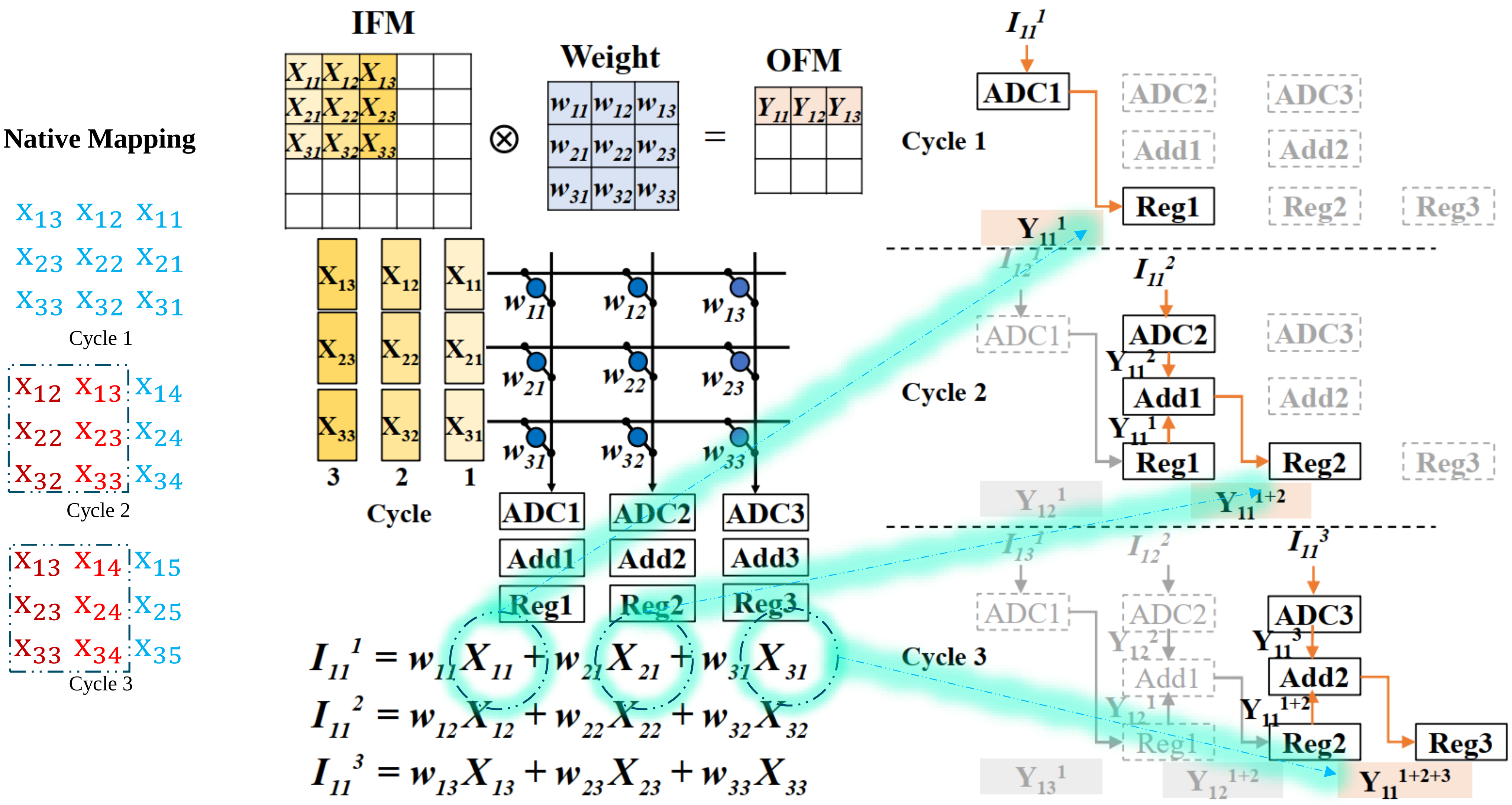
**Mapping in Raw & Intra-Tile Pipelines**

**Mapping in Proposed Pipeline**

focus on access strategy instead of adding up additional computation units

# Proposed Study

## Matrix-Based Mapping Method



$$I_{11}^{1} = w_{11}X_{11} + w_{21}X_{21} + w_{31}X_{31}$$

$$I_{11}^{2} = w_{12}X_{12} + w_{22}X_{22} + w_{32}X_{32}$$

$$I_{11}^{3} = w_{13}X_{13} + w_{23}X_{23} + w_{33}X_{33}$$

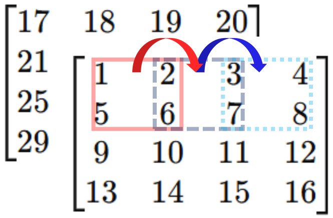
# Proposed Study

## Matrix-Based Mapping Method

| | |
|---|---|
| **IFM Size:** | 4x4x2 |
| **Kernel Size:** | 2x2 |
| **Number of Kernels:** | 3 |
| **Sliding Window:** | Stride 1 |

### Native Mapping Method

Each sliding 2×2×2 IFM slice is loaded once **per kernel**



$$\begin{bmatrix} 17 & 18 & 19 & 20 \\ 21 \\ 25 \\ 29 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$\text{IFM slice (C1)} = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}, \quad \text{IFM slice (C2)} = \begin{bmatrix} 17 & 18 \\ 21 & 22 \end{bmatrix}$$

Kernel 1: 2x2x2 loads

**Repeat** for Kernel 2-3

1 Slice: 24 loads

9 Slice: **216** loads

$$(\text{C1}) : \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \quad (\text{C2}) : \begin{bmatrix} 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 \\ 29 & 30 & 31 & 32 \end{bmatrix}$$

$$\text{Kernel 1 (K1)} : \begin{bmatrix} 1 & 0 \\ -1 & 2 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix}$$

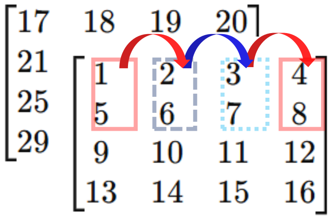
$$\text{Kernel 2 (K2)} : \begin{bmatrix} -1 & 1 \\ 0 & -2 \end{bmatrix}, \begin{bmatrix} 2 & 0 \\ -1 & 1 \end{bmatrix}$$

$$\text{Kernel 3 (K3)} : \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix}$$

$$\xrightarrow[\text{Access}]{1/k}$$

### Matrix-Based Method

Each sliding 2×2×2 IFM slice is loaded once per **sliding window**



$$\begin{bmatrix} 17 & 18 & 19 & 20 \\ 21 \\ 25 \\ 29 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$\text{Flattened slice} = [1, 2, 5, 6, 17, 18, 21, 22]$$

$$(\text{W}) = \begin{bmatrix} 1 & 0 & -1 & 2 & 0 & 1 & 1 & -1 \\ -1 & 1 & 0 & -2 & 2 & 0 & -1 & 1 \\ 0 & 1 & -1 & 1 & 1 & -1 & 0 & 2 \end{bmatrix}$$

Kernel 1: 2x2x2 loads

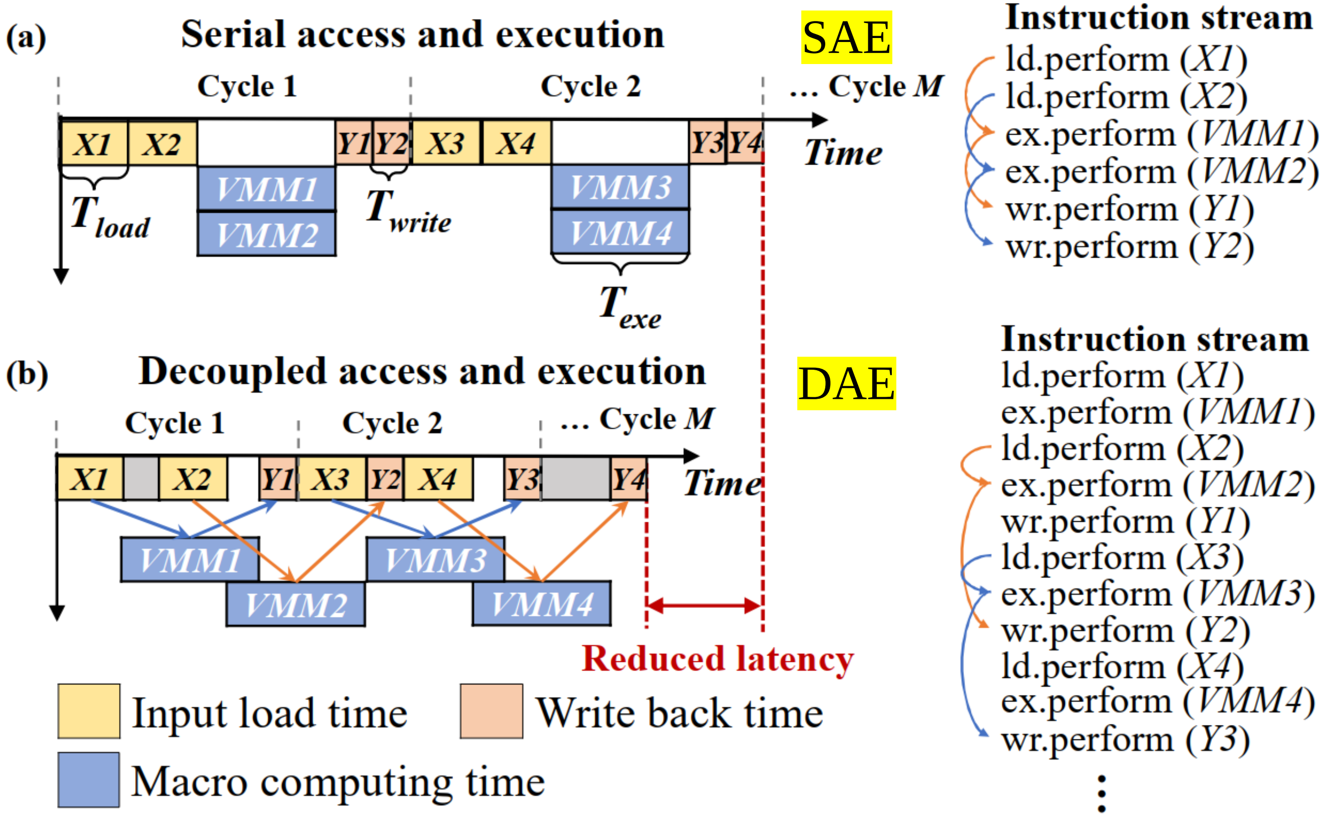
**Reuse** for Kernel 2-3

1 Slice: 8 loads

9 Slice: 72 loads

# Proposed Study

## Decoupled Access and Execution



SAE

**Instruction stream**
ld.perform (*X1*)
ld.perform (*X2*)
ex.perform (*VMM1*)
ex.perform (*VMM2*)
wr.perform (*Y1*)
wr.perform (*Y2*)

DAE

**Instruction stream**
ld.perform (*X1*)
ex.perform (*VMM1*)
ld.perform (*X2*)
ex.perform (*VMM2*)
wr.perform (*Y1*)
ld.perform (*X3*)
ex.perform (*VMM3*)
wr.perform (*Y2*)
ld.perform (*X4*)
ex.perform (*VMM4*)
wr.perform (*Y3*)
⋮

SAE

$$T_{total} = \left((T_{load} + T_{write}) \cdot N_p + T_{exe}\right) \cdot M$$

DAE - CD

$$T_{total} = (T_{exe} + T_{load} + T_{write}) \cdot M + (T_{load} + T_{write}) \cdot (N_p - 1)$$

DAE - BD

$$T_{total} = (T_{load} + T_{write}) \cdot (N_p \cdot M + N_p - 1)$$

Computation Dominates [$T_{exe} > T_{load} + T_{write}$]

$$R = (M - 1) \cdot (N_p - 1) \cdot (T_{load} + T_{write})$$

BufferAccess Dominates [$T_{load} + T_{write} > T_{exe}$]

$$R = T_{exe} \cdot M - (N_p - 1) \cdot (T_{load} + T_{write})$$

**focus on instruction flow to reduce single layer latency by hiding buffer acces under computation**

# Proposed Study

## Decoupled Access and Execution

| Parameter | Number | Description |
|---|---|---|
| $T_{load}$ | 2 | Input Load Time |
| $T_{write}$ | 1 | Output WB Time |
| $T_{exe}$ | 5 | Macro Execution Time |
| $N_p$ | 4 | Number of Parallel CIM Macro |
| $M$ | 8 | Number of Convolution Cycles |

SAE

$$T_{total} = \big((T_{load} + T_{write}) \cdot N_p + T_{exe}\big) \cdot M$$

$$= \big((2 + 1) \cdot 4 + 5\big) \cdot 8$$

$$= 136$$

DAE - CD

$$T_{total} = (T_{exe} + T_{load} + T_{write}) \cdot M + (T_{load} + T_{write}) \cdot (N_p - 1)$$

$$= (5 + 2 + 1) \cdot 8 + (2 + 1) \cdot (4 - 1)$$

$$= 73$$

DAE - BD

$$T_{total} = (T_{load} + T_{write}) \cdot (N_p \cdot M + N_p - 1)$$

$$= (2 + 1) \cdot (4 \cdot 8 + 4 - 1)$$

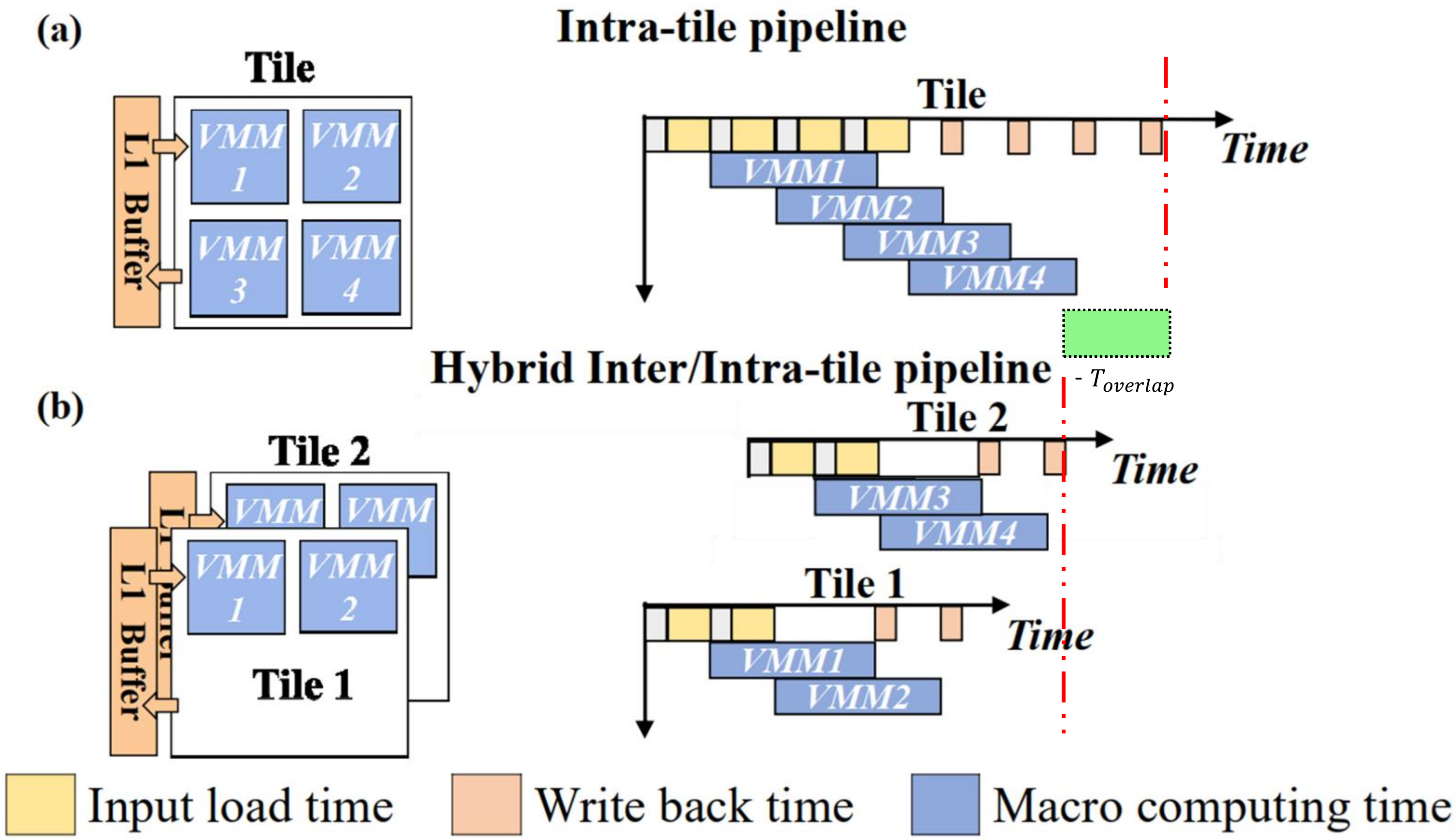
$$= 105$$

| Formula | Description | Result<br>(Time Units) |
|---|---|---|
| 1 | Serial Execution<br>(no overlap) | 136 |
| 2 | DA Execution Dominated<br>(partial overlap) | **73** |
| 3 | DA Access Dominated<br>(partial overlap) | **105** |

| Case | Formula | Reduction<br>(Time Units) |
|---|---|---|
| Serial Execution<br>(no overlap) | N/A | N/A |
| DA Execution Dominated<br>(partial overlap) | $(M - 1) \cdot \big(N_p - 1\big) \cdot (T_{load} + T_{write})$ | **63** |
| DA Access Dominated<br>(partial overlap) | $T_{exe} \cdot M - \big(N_p - 1\big) \cdot (T_{load} + T_{write})$ | **31** |

# Proposed Study

## Hybrid Inter/Intra Tile

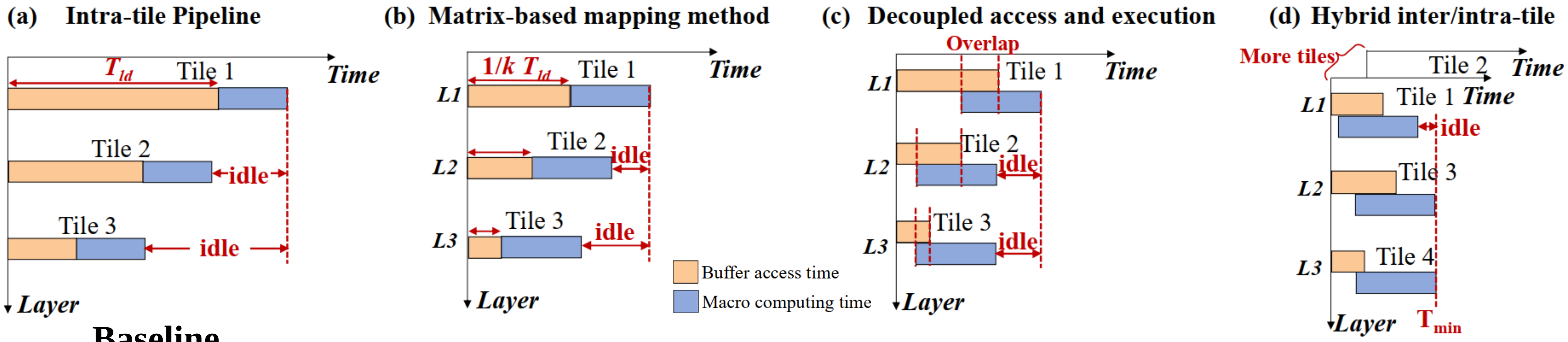


Reduced Latency

Area Cost

# Proposed Study

## Pipeline Optimization Design Summary



**Baseline**

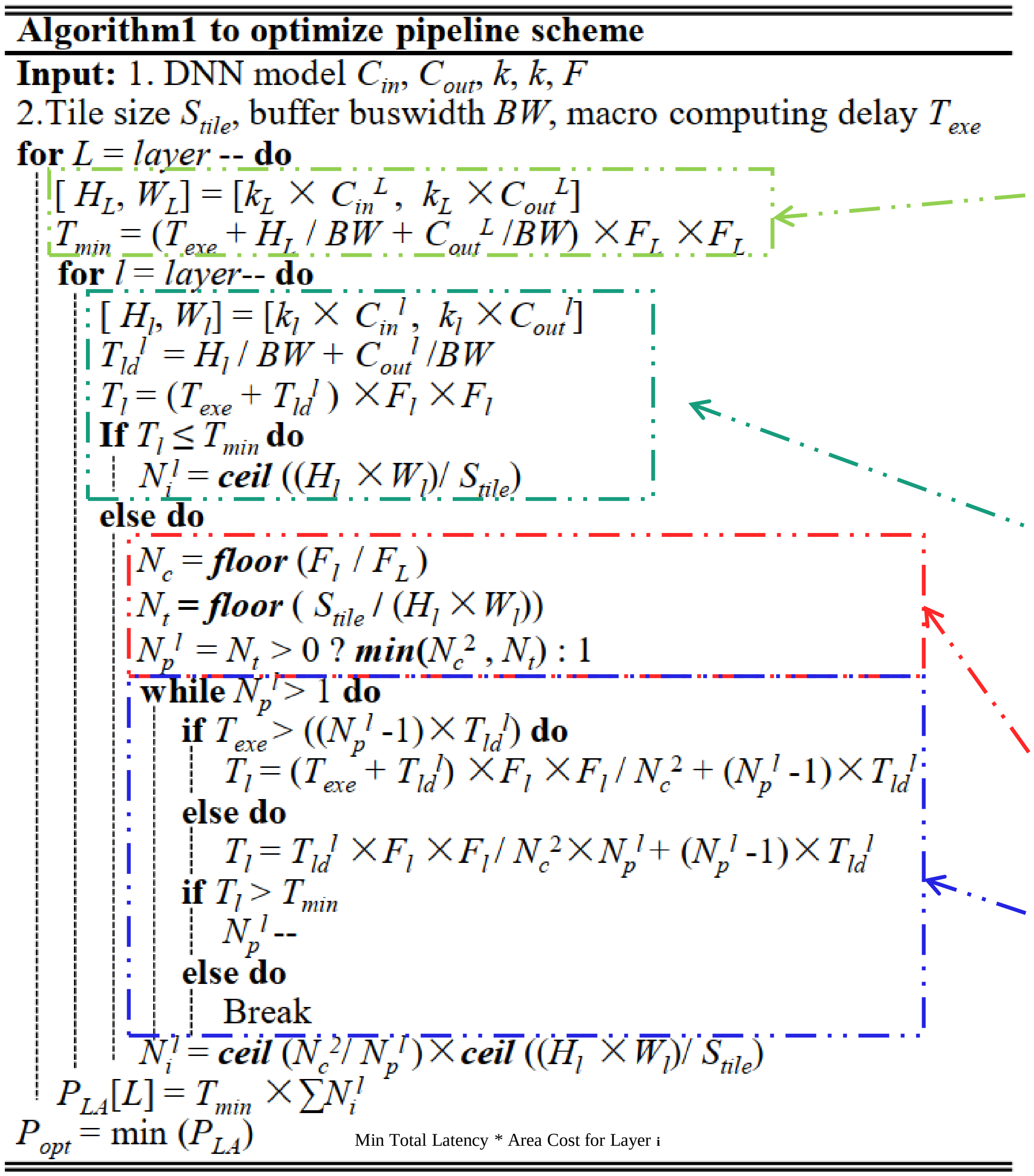
- ✓ First to speed up computation of bottleneck layers.
- × Suffers from unbalanced problems when taking buffer access delay into account.
- × Requires **weight duplication** across PEs, increasing memory

- ✓ Minimizes **buffer access delays** reusing IFM slices
- ✓ **Reduces** redundant data movement
- ✓ Leverages local **reuse** of weights and inputs

- ✓ Enables **overlapping** of load, compute, and write-back.
- ✓ Reduce total latency.
- ✓ **Hides memory access** under **computation** cycles.
- ✓ Scales better than serial methods for deeper CNNs and large IFMs.
- × HW/SW complexity increases due to **schedule and coordinate** parallel stages.

- ✓ Balanced workload across tiles (inter-tile) and within tiles (intra-tile)
- ✓ **Preventing Bottlenecks**
- ✓ **Reduce Idle States**
- ✓ High Throughput
- ✓ Suitable for Deep CNN with many layers and large feature maps
- × May require **complex scheduling** to synchronize tile operations
- × **Hardware cost increases** due to managing-tile communication.

# Proposed Study



**Algorithm1 to optimize pipeline scheme**

```
Input: 1. DNN model $C_{in}$, $C_{out}$, $k$, $k$, $F$
2.Tile size $S_{tile}$, buffer buswidth $BW$, macro computing delay $T_{exe}$
for $L = layer$ -- do
    $[H_L, W_L] = [k_L \times C_{in}^{L},\ k_L \times C_{out}^{L}]$
    $T_{min} = (T_{exe} + H_L / BW + C_{out}^{L} / BW) \times F_L \times F_L$
    for $l = layer$-- do
        $[H_l, W_l] = [k_l \times C_{in}^{l},\ k_l \times C_{out}^{l}]$
        $T_{ld}^{l} = H_l / BW + C_{out}^{l} / BW$
        $T_l = (T_{exe} + T_{ld}^{l}) \times F_l \times F_l$
        If $T_l \leq T_{min}$ do
            $N_i^l = \mathbf{ceil}\ ((H_l \times W_l) / S_{tile})$
        else do
            $N_c = \mathbf{floor}\ (F_l / F_L)$
            $N_t = \mathbf{floor}\ (S_{tile} / (H_l \times W_l))$
            $N_p^l = N_t > 0\ ?\ \mathbf{min}(N_c^2, N_t) : 1$
            while $N_p^l > 1$ do
                if $T_{exe} > ((N_p^l - 1) \times T_{ld}^{l})$ do
                    $T_l = (T_{exe} + T_{ld}^{l}) \times F_l \times F_l / N_c^2 + (N_p^l - 1) \times T_{ld}^{l}$
                else do
                    $T_l = T_{ld}^{l} \times F_l \times F_l / N_c^2 \times N_p^l + (N_p^l - 1) \times T_{ld}^{l}$
                if $T_l > T_{min}$
                    $N_p^l$--
                else do
                    Break
            $N_i^l = \mathbf{ceil}\ (N_c^2 / N_p^l) \times \mathbf{ceil}\ ((H_l \times W_l) / S_{tile})$
    $P_{LA}[L] = T_{min} \times \sum N_i^l$
$P_{opt} = \min\ (P_{LA})$          Min Total Latency * Area Cost for Layer i
```

1. **Operator Level**

- **Purpose**: Optimize **data mapping** and **reuse** for matrix to minimize buffer access.

- **Key Optimization:** Matrix-based weight mapping **reduces redundant IFM loading** and **maximizes local reuse** of data slices.

**2. Macro Level**

- **Purpose**: **Latency optimization** of a **CIM macro** for (L/W & Compute)

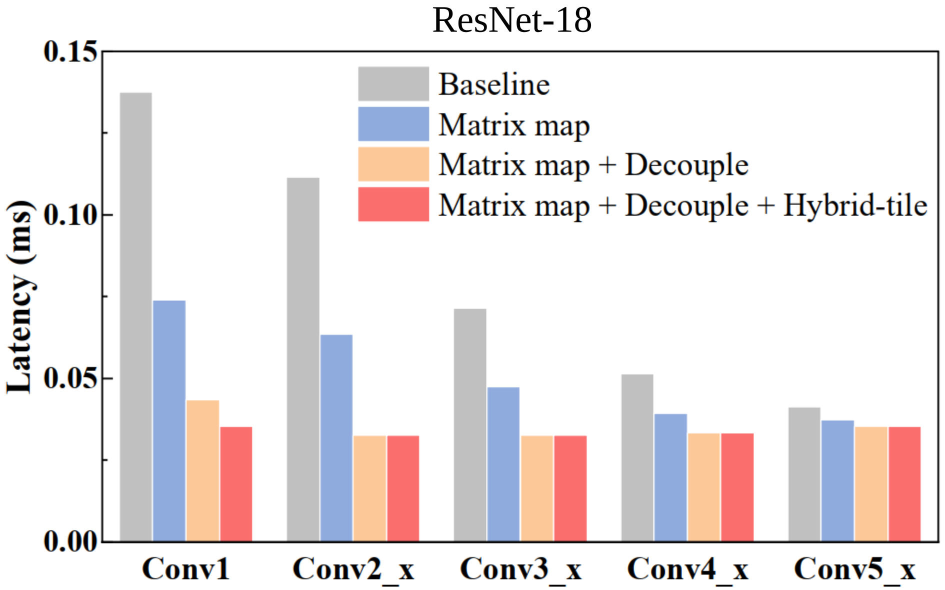
- **Key Optimization**: Decoupled access and execution, overlap memory access with computation to **hide access delays under execution**.

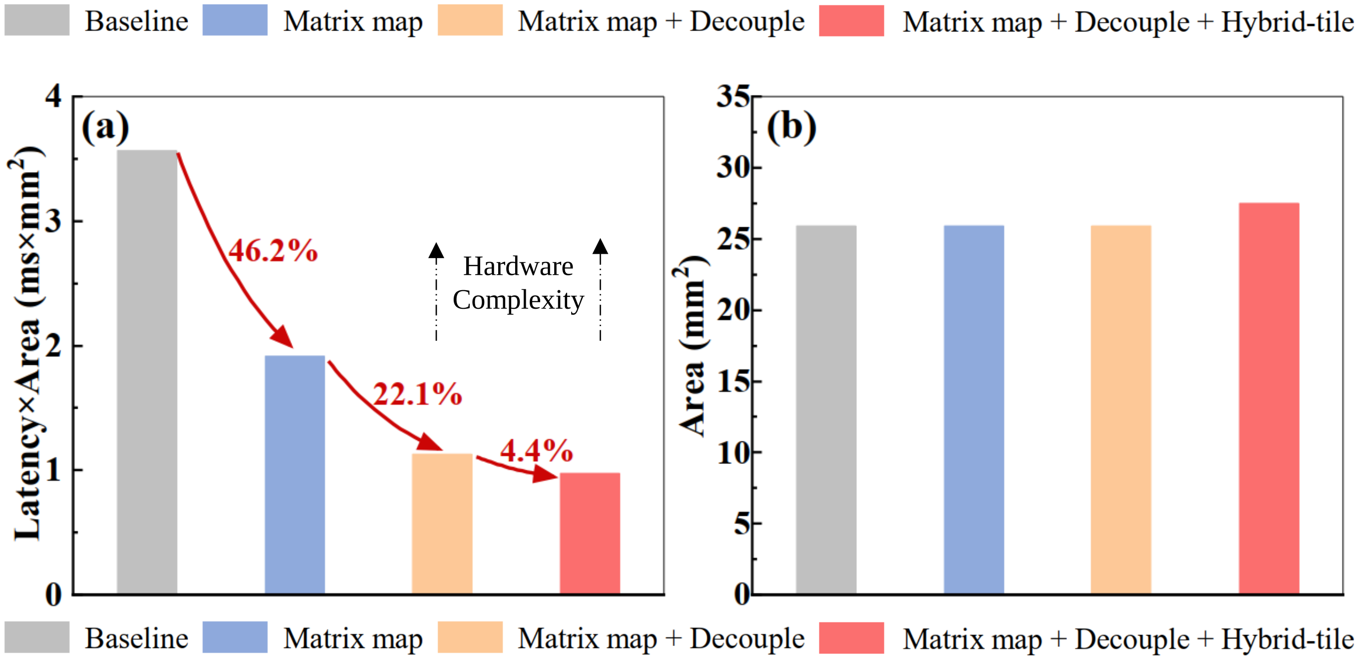
**3. System Level**

- **Purpose**: **Balancing workload, multiple tiles** to **avoid bottlenecks**

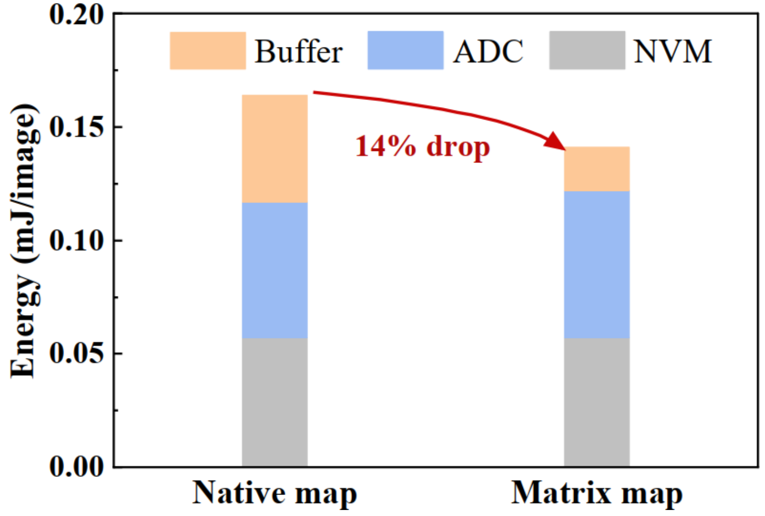
- **Key Optimization**: Hybrid inter/intra-tile pipeline
  - Coordinate execution within a tile (intra-tile parallelism)
  - Coordinate execution across tiles (inter-tile pipeline)

# Evaluation

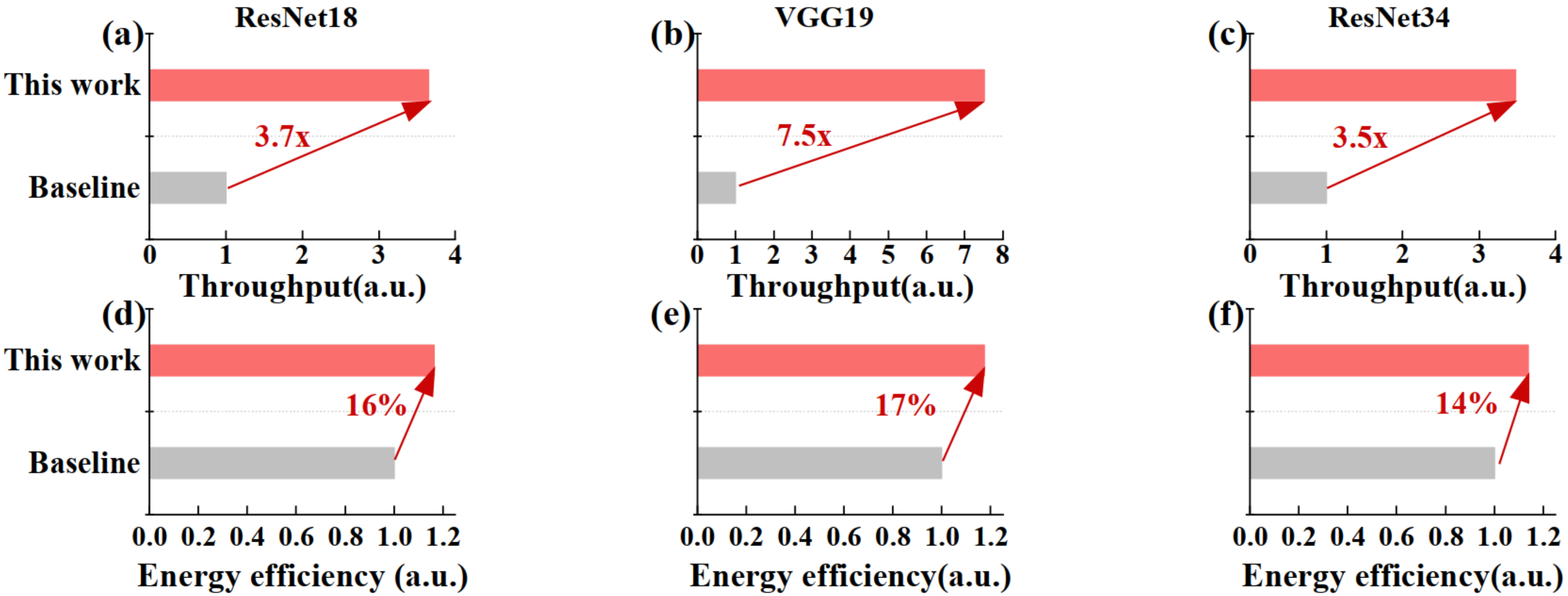

ResNet-18
Latency (ms)
0.15
0.10
0.05
0.00
Baseline
Matrix map
Matrix map + Decouple
Matrix map + Decouple + Hybrid-tile
Conv1
Conv2_x
Conv3_x
Conv4_x
Conv5_x


Baseline
Matrix map
Matrix map + Decouple
Matrix map + Decouple + Hybrid-tile
(a)
Latency×Area (ms×mm$^2$)
4
3
2
1
0
46.2%
22.1%
4.4%
Hardware
Complexity
(b)
Area (mm$^2$)
35
30
25
20
15
10
5
0
Baseline
Matrix map
Matrix map + Decouple
Matrix map + Decouple + Hybrid-tile

# Evaluation



Buffer
ADC
NVM
14% drop
Energy (mJ/image)
0.00
0.05
0.10
0.15
0.20
Native map
Matrix map



ResNet18
VGG19
ResNet34
(a)
(b)
(c)
This work
Baseline
3.7x
7.5x
3.5x
0
1
2
3
4
0
1
2
3
4
5
6
7
8
0
1
2
3
4
Throughput(a.u.)
Throughput(a.u.)
Throughput(a.u.)
(d)
(e)
(f)
This work
Baseline
16%
17%
14%
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Energy efficiency (a.u.)
Energy efficiency(a.u.)
Energy efficiency(a.u.)

# Key Achievements and Contributions

The study provides solutions to:

**Latency bottlenecks** due to sequential data access.
**Redundant** data loading and poor data reuse.
**Imbalanced workloads** across CIM macros.
**Scalability issues** with large CNN models.

Key achievements by study:

**Reduced Pipeline Latency:** Faster processing by minimizing idle time and bottlenecks.
**Improved Throughput:** Higher number of inputs processed per second.
**Energy Efficiency:** Reduction in memory access overheads lowers power consumption.
**Hardware Scalability:** The method is adaptable to various hardware accelerators with different configurations (e.g., buffer size, number of cores).

Experiments shows that proposed pipeline achieves **3.7x**, **7.5x** and **3.5x** througput (ResNet18,VGG19, and ResNet34) over basepipelines

# Questions