



Heterogeneous Memory Architecture Accommodating Processing-In-Memory on SoC For AIoT Applications

Kangyi Qiu, Yaojun Zhang, Bonan Yan, Ru Huang

Institute of Artificial Intelligence, Peking University, Beijing, China

School of Integrated Circuits, Peking University, Beijing, China

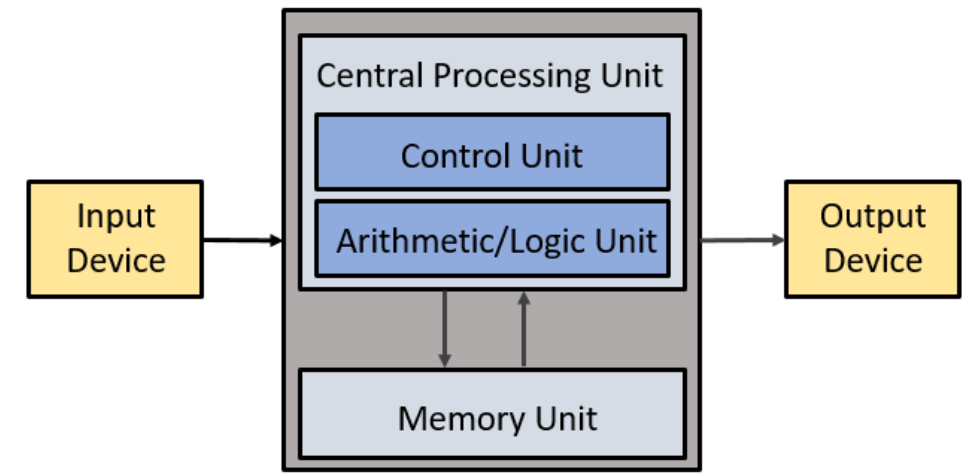
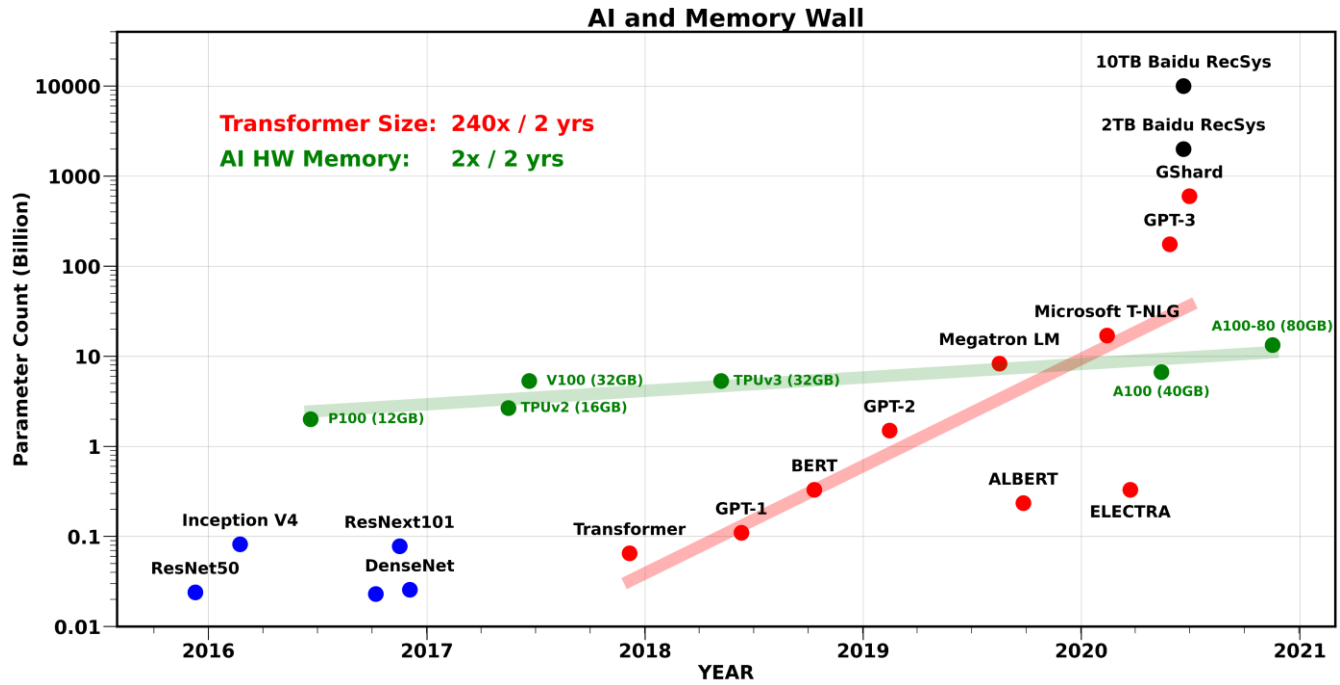
Pimchip Technology Co., Ltd., Beijing, China

corresponding authors email: {bonanyan, ruhuang}@pku.edu.cn

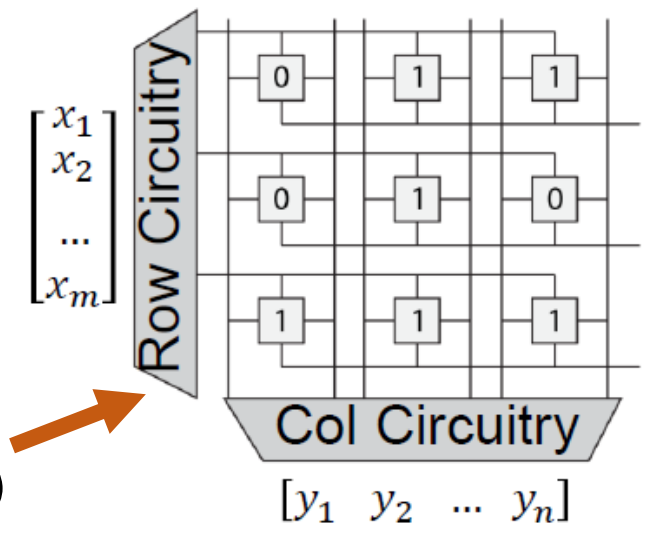
Outline

- Motivation
- Heterogeneous memory architecture
- HMA tensor mapping approach
- Experiment results
- Summary & prospective

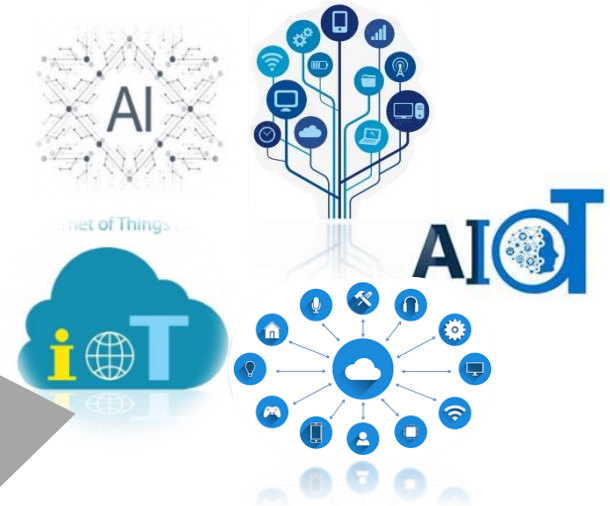
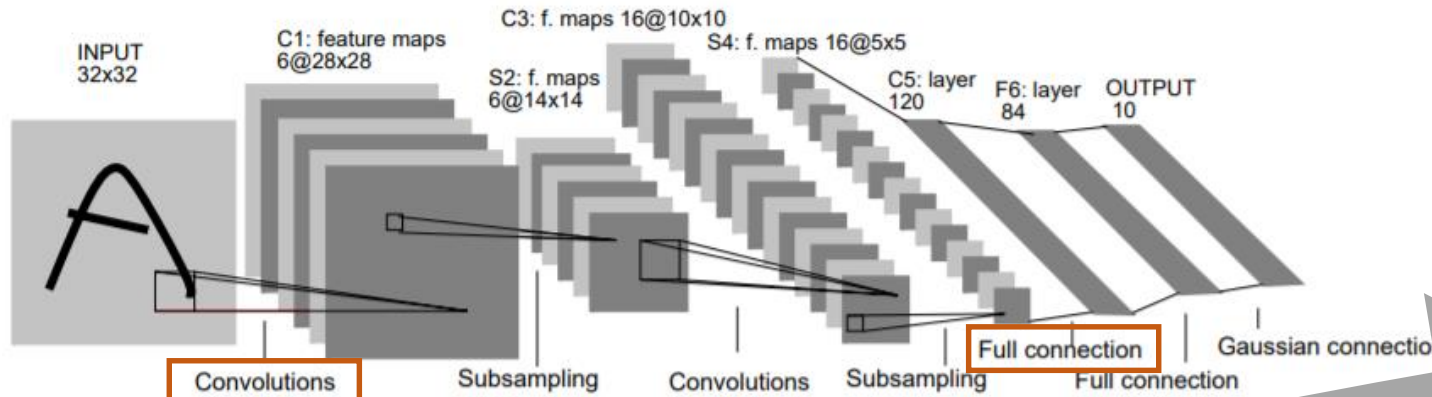
Motivation: Data Movement is Expensive



- ❑ storage and computing are separated
- ❑ computing power demand VS memory performance
- ❑ Data movement is expensive
 - ❑ Reduce data movement
 - ❑ Processing in-memory(PIM)



Motivation: Exponential Growth of Computing Power Demand



CNN: Convolutional Neural Networks

AIoT: artificial intelligence and Internet of things

GEMM: general matrix-multiplication

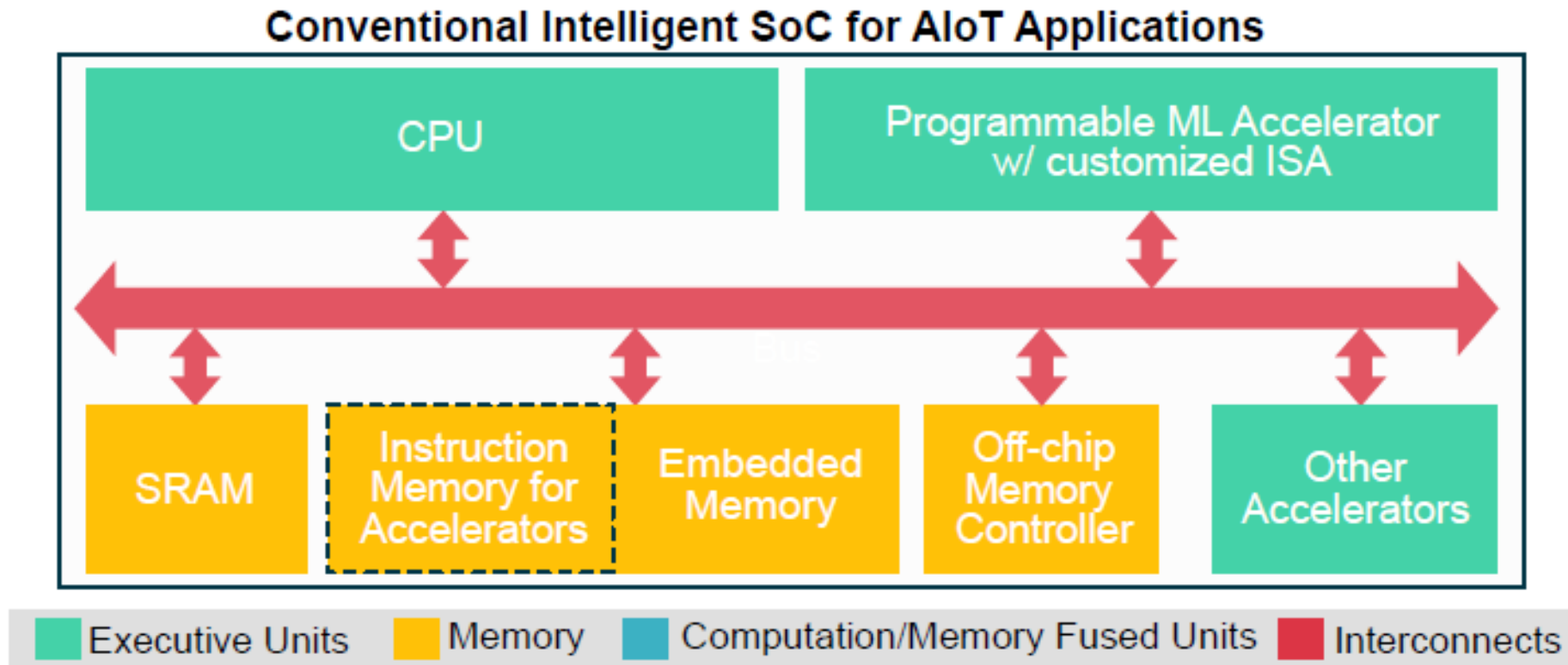
$$\begin{bmatrix} a_{00} & \cdots & a_{0K} \\ \vdots & \ddots & \vdots \\ a_{M0} & \cdots & a_{MK} \end{bmatrix} \times \begin{bmatrix} b_{00} & \cdots & b_{0N} \\ \vdots & \ddots & \vdots \\ b_{k0} & \cdots & b_{KN} \end{bmatrix} = \begin{bmatrix} c_{00} & \cdots & c_{0n} \\ \vdots & \ddots & \vdots \\ c_{m0} & \cdots & c_{mn} \end{bmatrix},$$

$$C_{ij} = \sum A_{ik} \cdot B_{kj}; \text{ Output} = \text{Input} * \text{weight}$$

Inference:

PIM + mapping approach
change inputs, fixed weights
reduce data movement

Motivation: Conventional Intelligent SoC may not Efficient



Have been researched

- Circuit level
- Programmable architecture

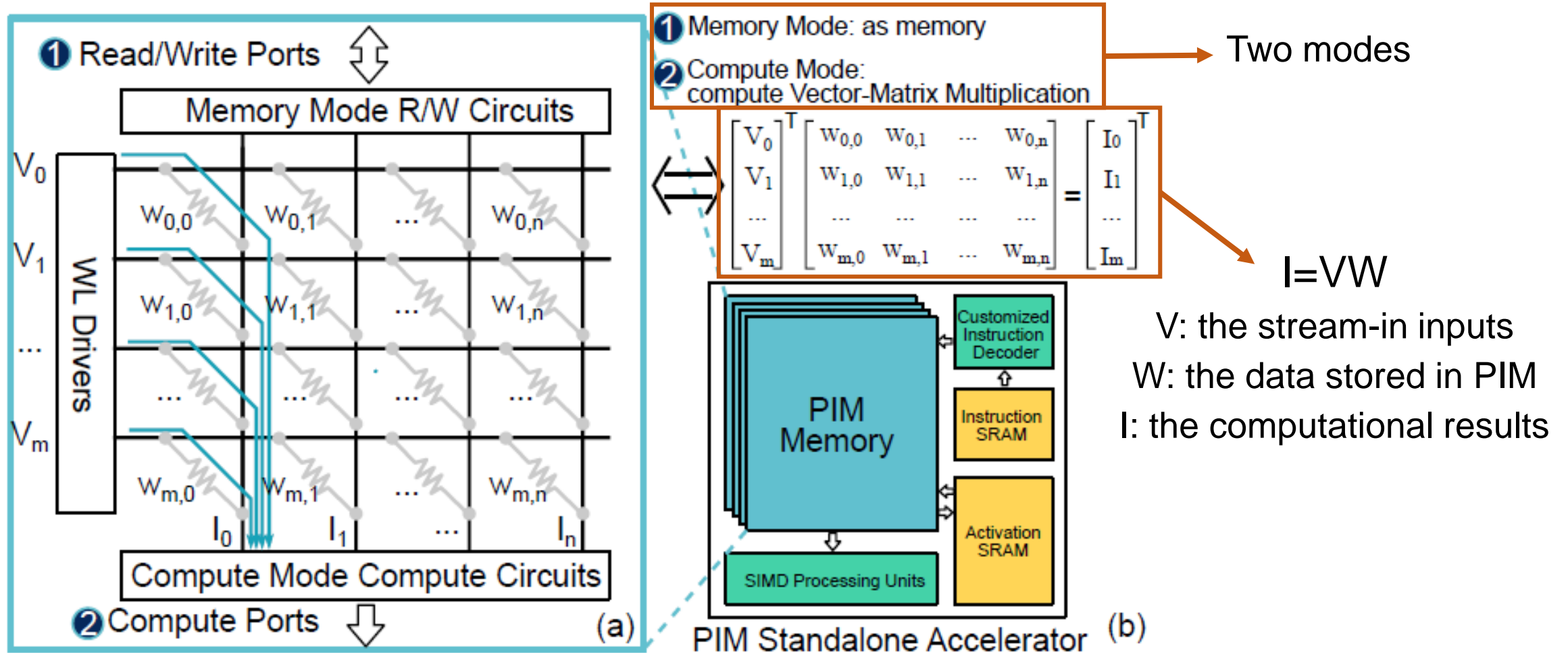
To be explored

- Interface PIM into SoC (system-on-chip)
- Reduce the difficulty of compilation

Major Contributions of this paper

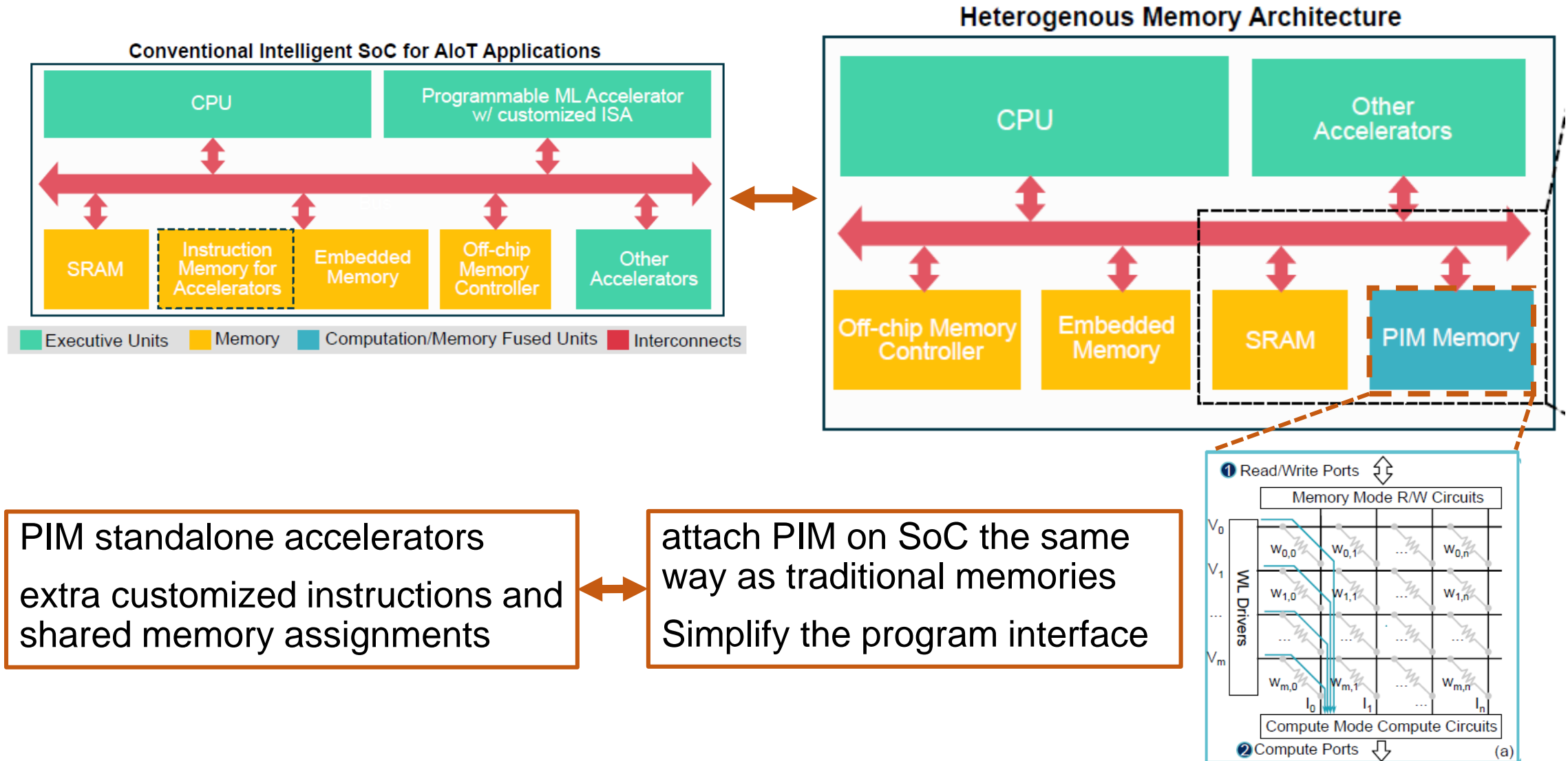
- Heterogeneous memory architecture (HMA)
 - The first architecture to clarify how to interface PIM to off-the-shelf SoC
 - Possess both PIM memories and traditional memories
 - simplifying the program interface,
- HMA tensor mapping approach
 - the software-to-hardware optimization
 - Partition tensors and deploy the GEMM tasks to the HMA
 - Provide a hardware-agnostic way to exploit PIM hardware
 - be used as a pre-design spec estimation

Heterogeneous memory architecture: PIM Standalone Accelerators



challenge: Compilation and deployment of software onto PIM hardware

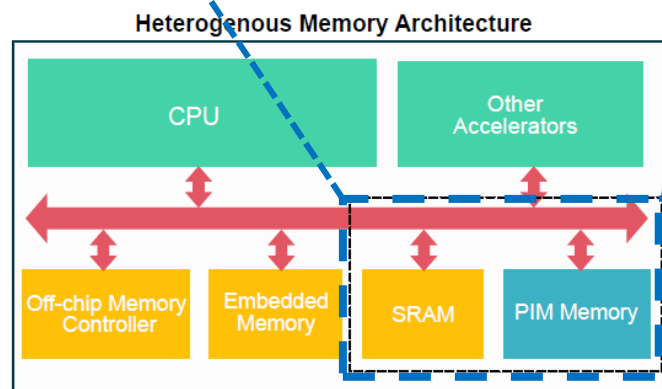
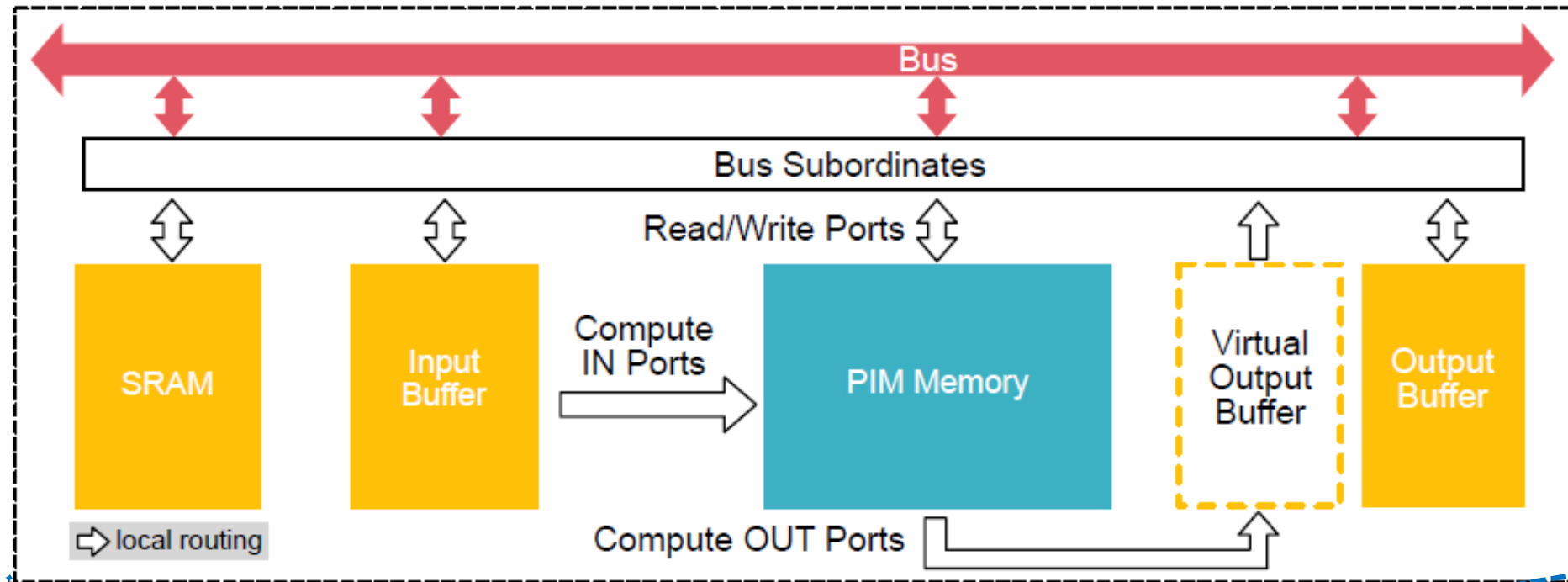
Heterogeneous memory architecture: Overall HMA Structure



PIM standalone accelerators
extra customized instructions and
shared memory assignments

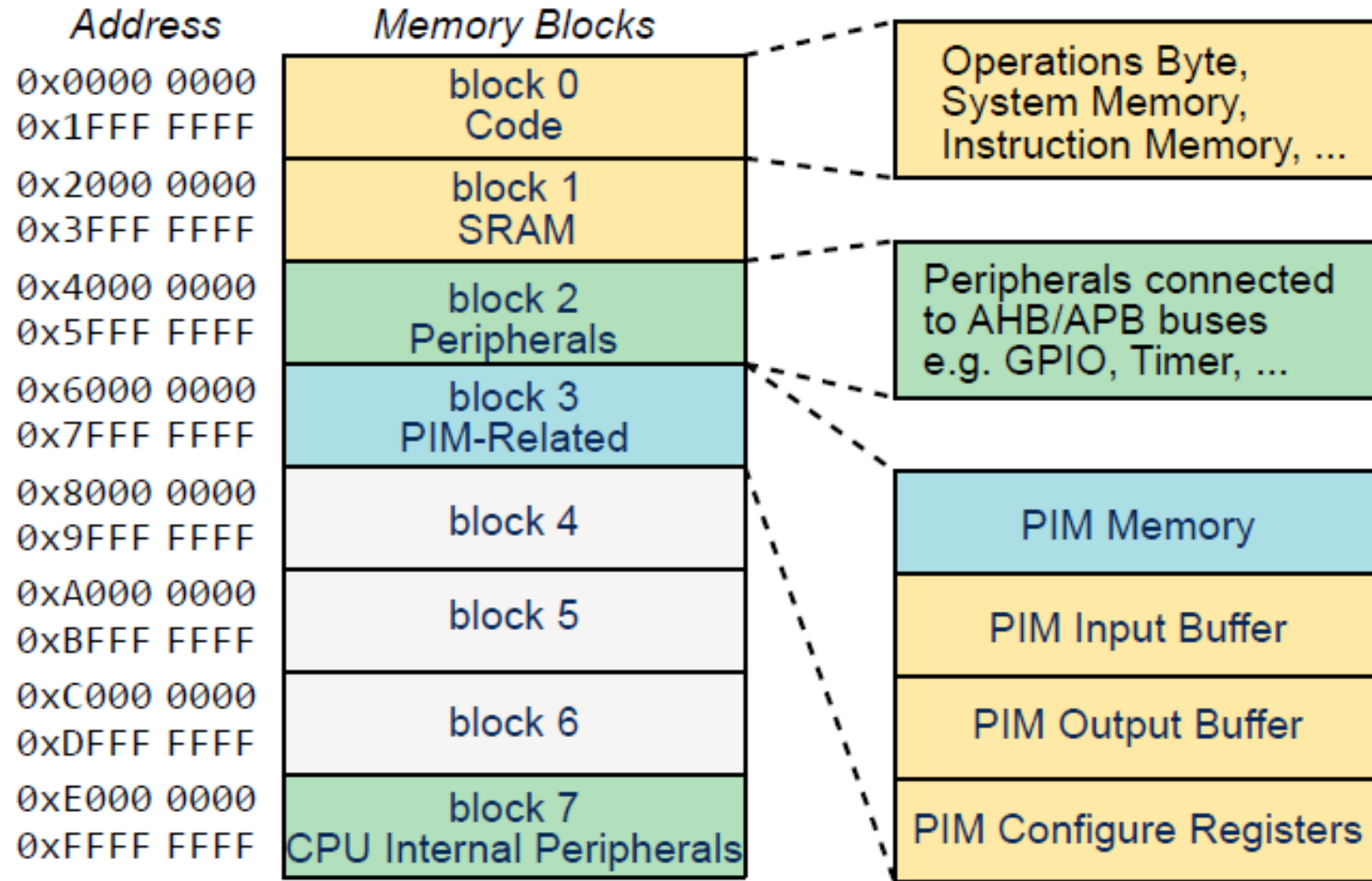
attach PIM on SoC the same
way as traditional memories
Simplify the program interface

HMA: Data Movement and Computing in Heterogeneous Memory



- input/output buffers to cache the input/output data
- PIM can access the data in the PIM input buffer
- CPU instructs PIM to be programmed

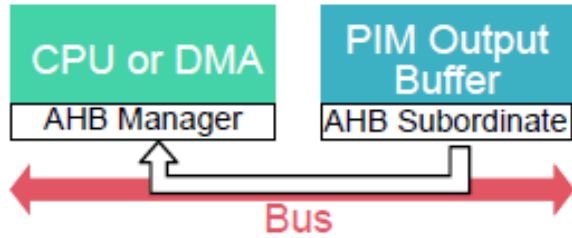
HMA: Address Assignment



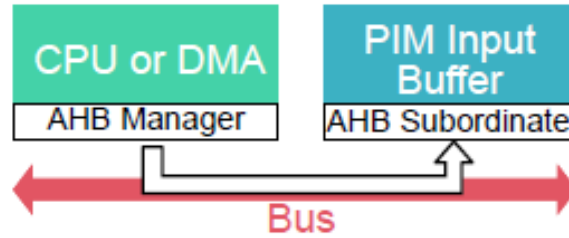
- The total width of the address is 32bit.
- Each 64MByte forms as a block.
- Different address widths and memory sizes will lead to different division methods

HMA: Data Transportation

(a) Obtain PIM Computation Results



(b) Feed PIM Inputs

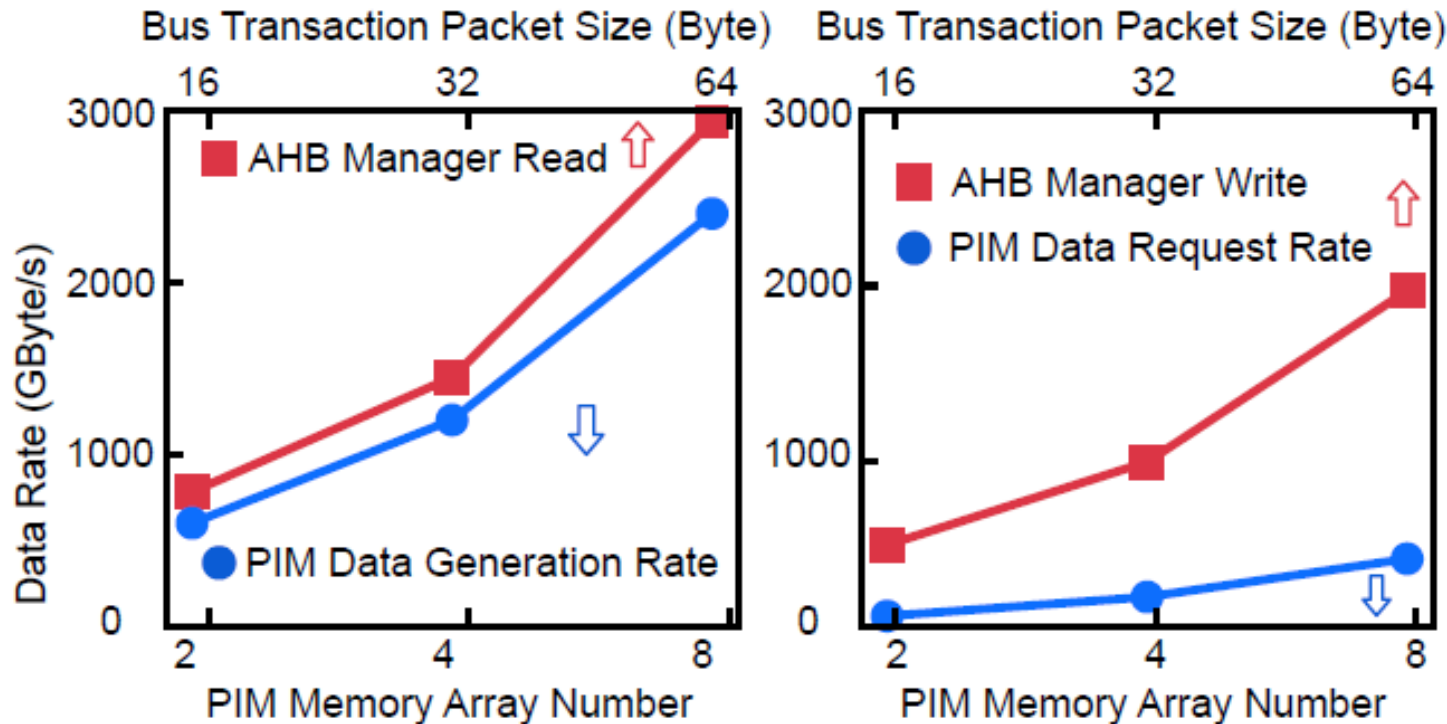


Capacity/Memory Array	256kb
Compute Latency@4b Precision	18.3ns
Max PIM Array Number/Chip	8

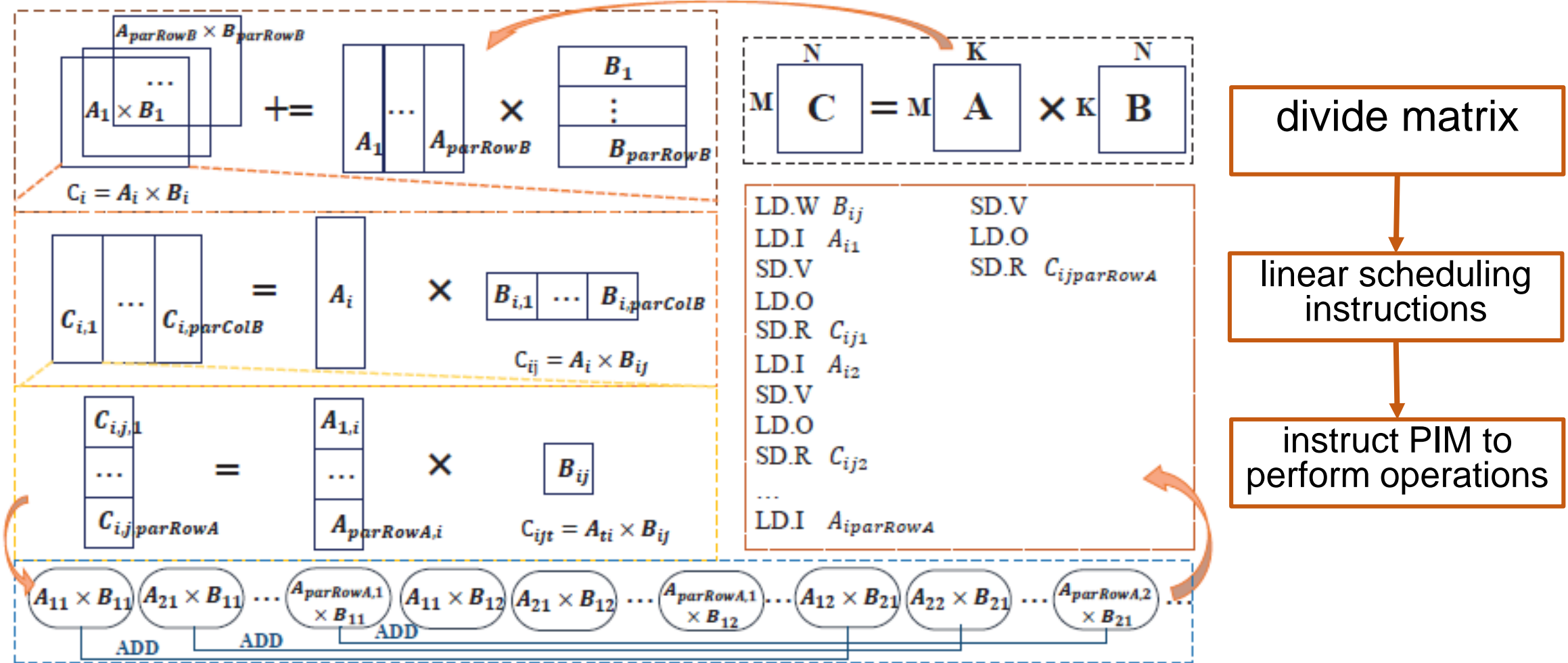
prerequisite:
Can afford adequate bandwidth?

experiment:
Case A, Case B
more PIM memory arrays require a higher data transfer rate

Conclusion:
the common used AHB can handle the PIM inputs and outputs without additional congestion or interconnection buffering

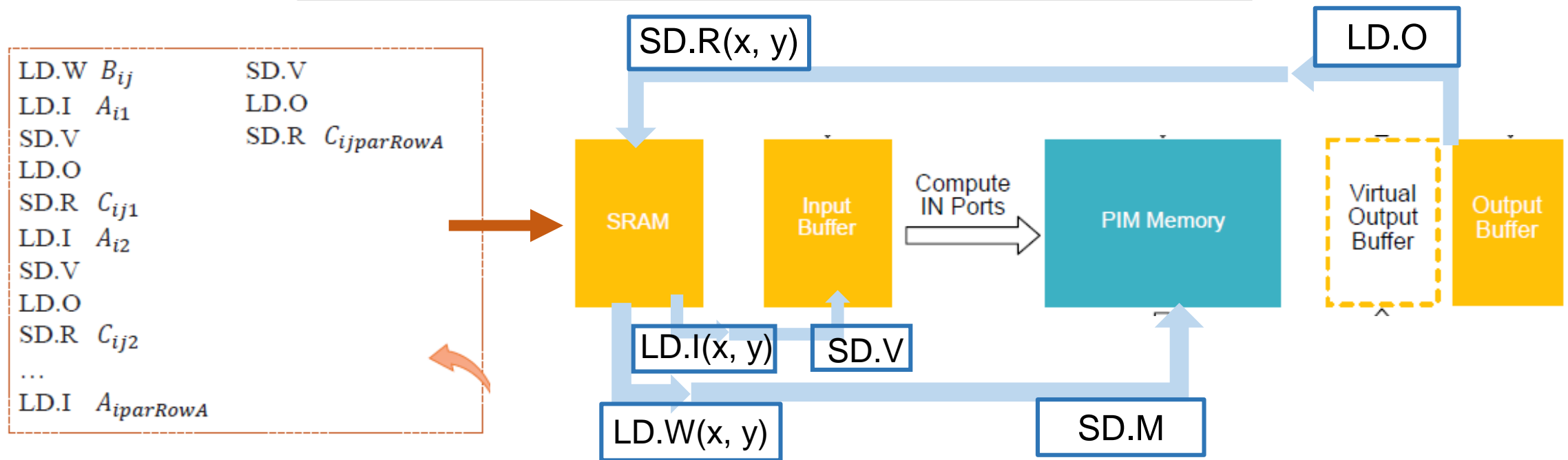


HMA tensor mapping approach: Visualization of the Mapping Approach



HMA tensor mapping approach: Instruction List

instruction	description
LD. I (x,y)	Load matrix A from traditional memory
LD. W (x,y)	Load matrix B from traditional memory
SD. R (x,y)	Store PIM results to traditional memory
SD. M	Store matrix B into PIM-memories
SD. V	Store matrix A into PIM input buffer
LD. O	Load the calculation results from PIM output buffer

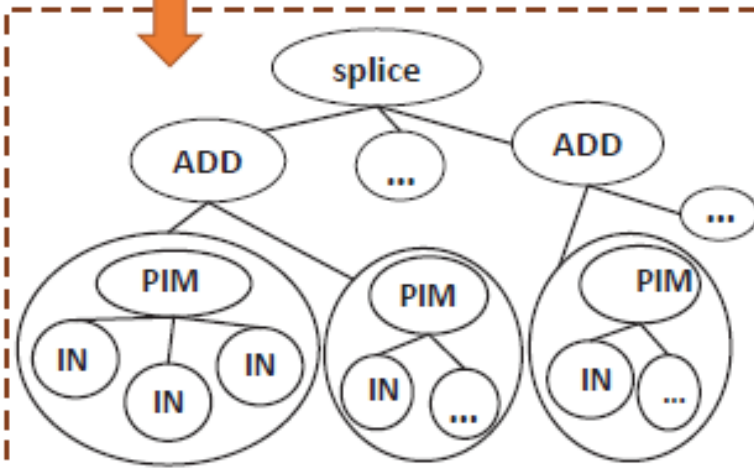


HMA tensor mapping approach: Pseudo Code Optimization

traditional GEMM

```

Input matrix A, B
Output C
for (int m = 0; m < M; m++) {
  for (int n = 0; n < N; n++) {
    C[m][n] = 0;
    for (int k = 0; k < K; k++) {
      C[m][n] += A[m][k] × B[k][n];
    }
  }
}
    
```



HMA tensor mapping approach

```

Input matrix A, B
Input  $Input_{row}, Input_{col}, PIM_{row}, PIM_{col}$ 
Output Schedule instruction set , result C
  get the size of A B:  $rowA, colA, rowB, colB$ 
  calculate  $parRowA, parColA, parRowB, parColB$ 
  for i = 0 to  $parRowB - 1$ 
    for j = 0 to  $parColB - 1$ 
      LD.W
      SD.M
      for p = 0 to  $parRowA - 1$ 
        LD.I
        SD.V
        calculate  $PIM(A_{ip}, B_{ij})$ 
        LD.O
        SD.R
      get the result C, calculate the response time
    
```

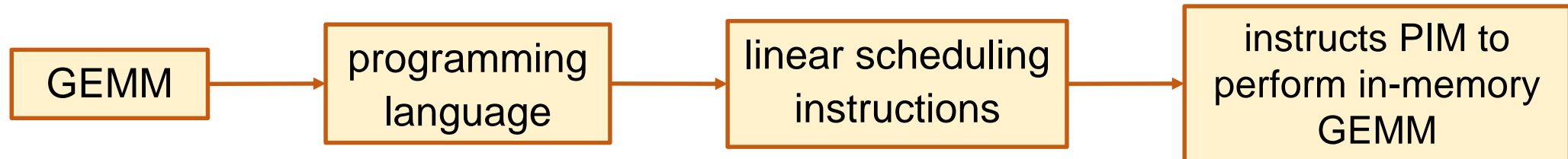
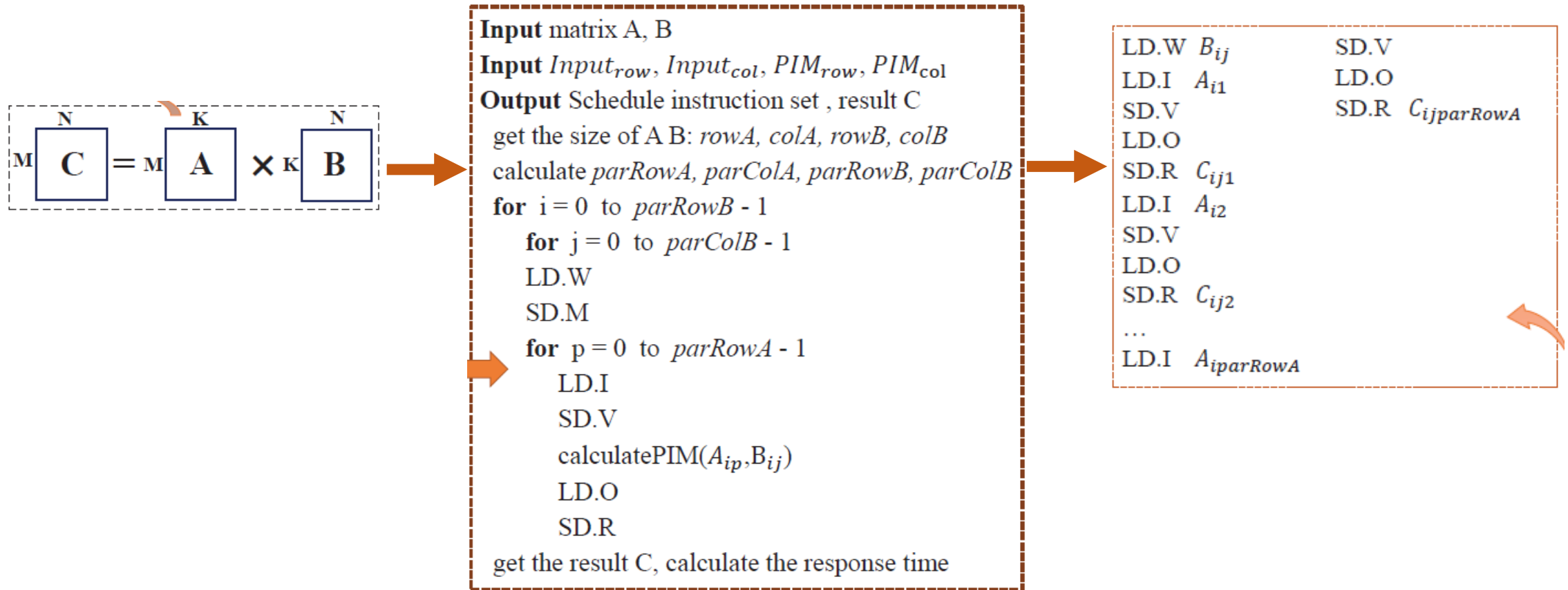
$C=A*B \rightarrow$ multiple $I=VW$

Appropriate division and scheduling \rightarrow Reduce data movement

Method:

fix weight matrix W,
change input matrix V

HMA tensor mapping approach: Overall Process



HMA tensor mapping approach: Memory Access Frequency

$$RT_{classic} = (2 + 1 + 1) \cdot m \cdot n \cdot k = 4 \cdot m \cdot n \cdot k$$

Classical calculation
formula of Memory
Access Frequency



$$RT_{PIM} = 6 \cdot parRowA \cdot parColA \cdot parColB$$

where $parRowA = m$

$$parColA = k / length(input_{col}),$$

$$parRowB = k / length(PIM_{row}),$$

$$parColB = n / length(PIM_{col})$$

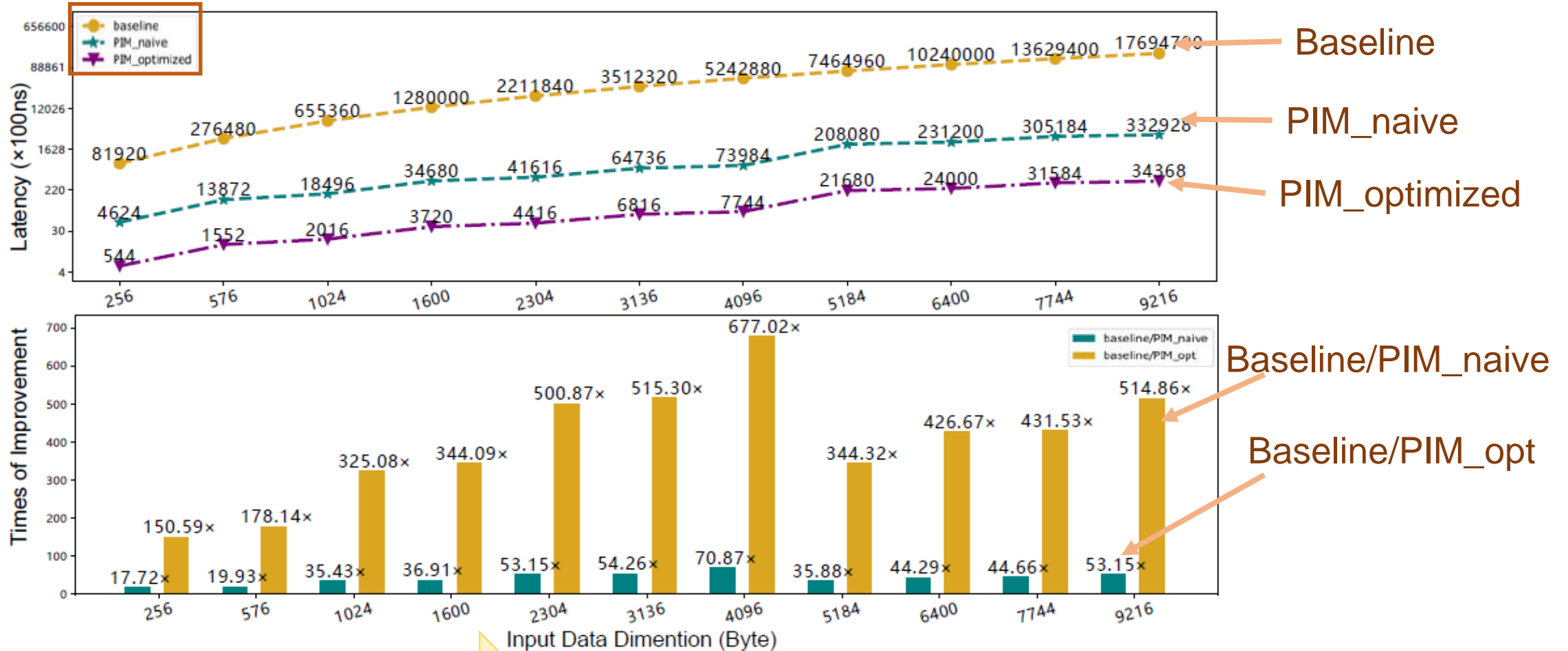
Memory Access
Frequency after using
HMA architecture



$$RT_{opt} = parRowB \cdot parColB \cdot (2 + 4 \cdot parRowA)$$

using multiple PIM cores and
the proposed mapping method

Experiment results: Compare Latency

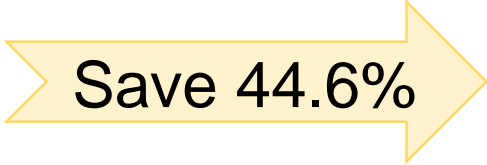





baseline → Improve 430 times → HMA + mapping approach

HMA without optimization approach → Improve 11 times → HMA + mapping approach

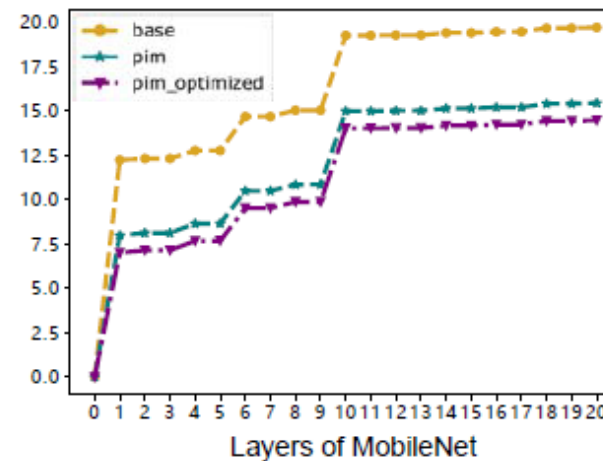
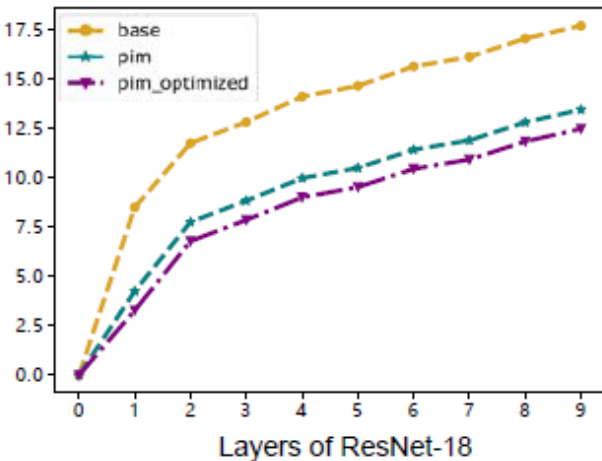
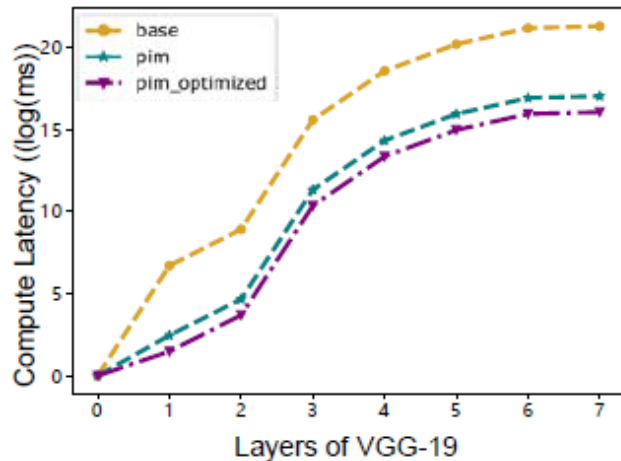
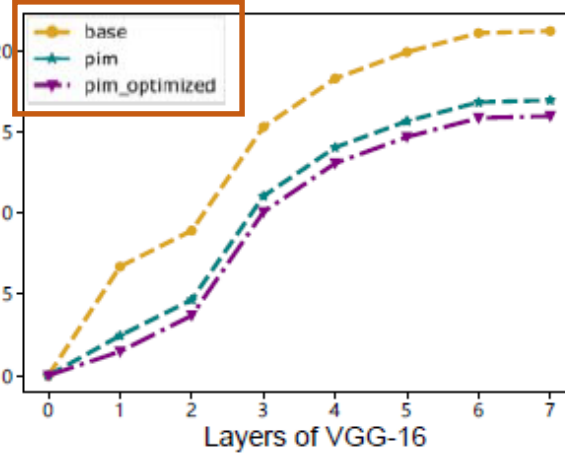
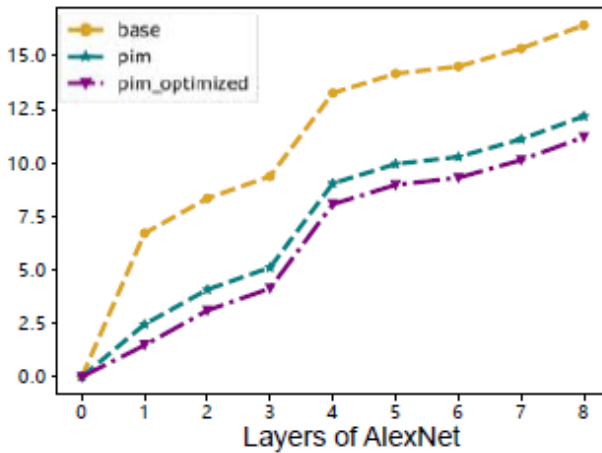
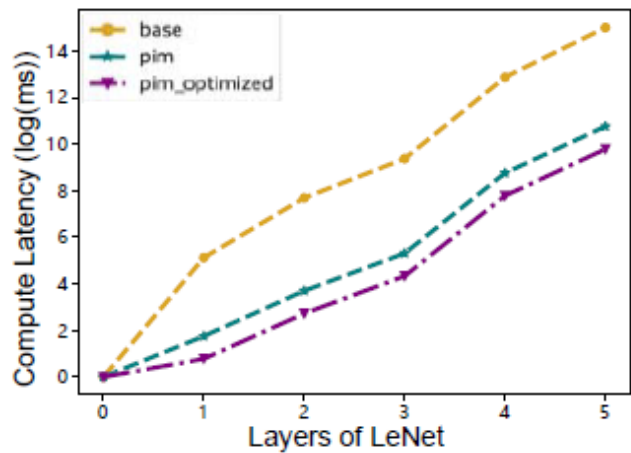
Experiment results: Area and Power Efficiency

	HMA (this work)	PUMA [10]
Area (Unit: mm ²)		
PIM Memory	2.25	
Controller and Bus	0.004	1.82
Overall	2.254	4.07
Power of On-Chip Interconnection (Unit: mW)		
Controller and Bus	1.07	445

The overall area: PUMA  Save 44.6%  HMA

peripheral circuit active power reduction: PUMA  416 times  HMA

Experiment results: DNN Acceleration Analysis



Base

PIM

PIM_optimized

select 6 DNNs to compare the acceleration by HMA and the HMA tensor mapping approach

Summary & Prospective

- ❑ Proposed a Heterogeneous Memory Architecture for improving the efficiency of PIM on conventional small-scale embedded SoC.
- ❑ Proposed a mapping algorithm to better exploit PIM's acceleration.
- ❑ Explored the power consumption and operation latency
- ❑ Great guidance for top-level software-hardware codesigns for PIM-related SoC design in its early design stages.



Thank you!

Kangyi Qiu