



# HiSpMV: Hybrid Row Distribution and Vector Buffering for Imbalanced SpMV Acceleration on FPGAs

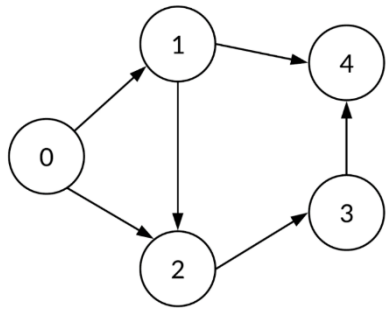
*Manoj B. Rajashekar, Xingyu Tian, Zhenman Fang*

**HiAccel Lab, Simon Fraser University, Canada**

<https://www.sfu.ca/~zhenman>

# Sparse-Matrix Vector Multiplication (SpMV)

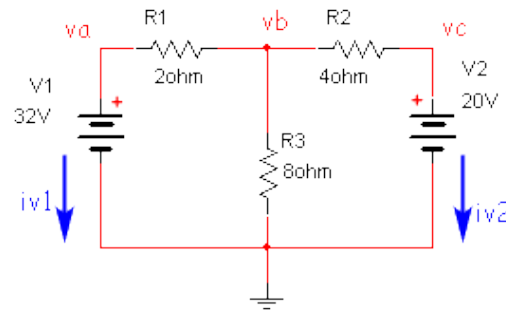
- Equation of SpMV operation:  $\vec{y} = \alpha \cdot A \times \vec{x} + \beta \cdot \vec{y}$
- Fundamental kernel in many scientific and engineering applications



Adjacency Matrix

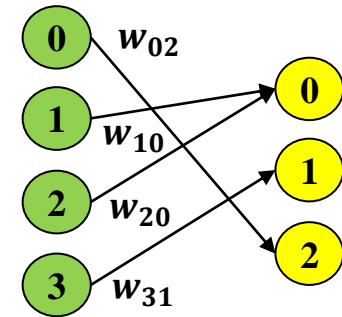
	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0

Graph Analytics



$$\begin{bmatrix} \frac{1}{R_1} & -\frac{1}{R_1} & 0 & 1 & 0 \\ -\frac{1}{R_1} & \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} & -\frac{1}{R_2} & 0 & 0 \\ 0 & -\frac{1}{R_2} & \frac{1}{R_2} & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ v_c \\ i_{v1} \\ i_{v2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ V1 \\ V2 \end{bmatrix}$$

Circuit Simulation



$$\begin{bmatrix} 0 & w_{10} & w_{20} & 0 \\ 0 & 0 & 0 & w_{31} \\ w_{02} & 0 & 0 & 0 \end{bmatrix}$$

Sparse Neural Network

# SpMV Example

- Equation of SpMV operation:  $\vec{y} = \alpha \cdot A \times \vec{x} + \beta \cdot \vec{y}$
- In SpMV, only non-zero elements are used in the computation

$$\vec{x} \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 \end{bmatrix} = \begin{bmatrix} (5 \cdot 1) \\ (8 \cdot 4) \end{bmatrix} = \begin{bmatrix} 5 \\ 32 \end{bmatrix} \vec{y}$$

# Challenges to Accelerate SpMV and Prior Solutions

## #1 Random memory access

$$\vec{x} \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 \end{bmatrix} = \vec{y} \begin{bmatrix} 5 \\ 0 \\ 32 \end{bmatrix}$$

# Challenges to Accelerate SpMV and Prior Solutions

## #1 Random memory access

- On-chip buffers for dense vectors

$$\vec{x} \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 32 \end{bmatrix} \vec{y}$$

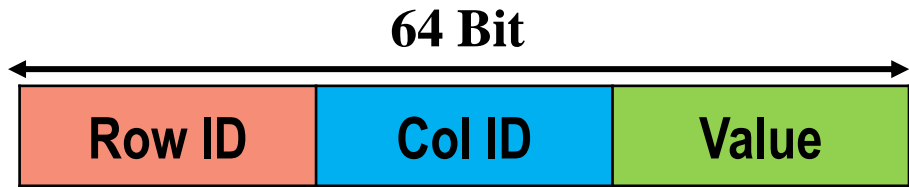
# Challenges to Accelerate SpMV and Prior Solutions

## #1 Random memory access

- On-chip buffers for dense vectors

## #2 High bandwidth requirement

- Multiple HBM channels to stream encoded sparse matrix



One element of sparse matrix in COO format

# Challenges to Accelerate SpMV and Prior Solutions

## #1 Random memory access

- On-chip buffers for dense vectors

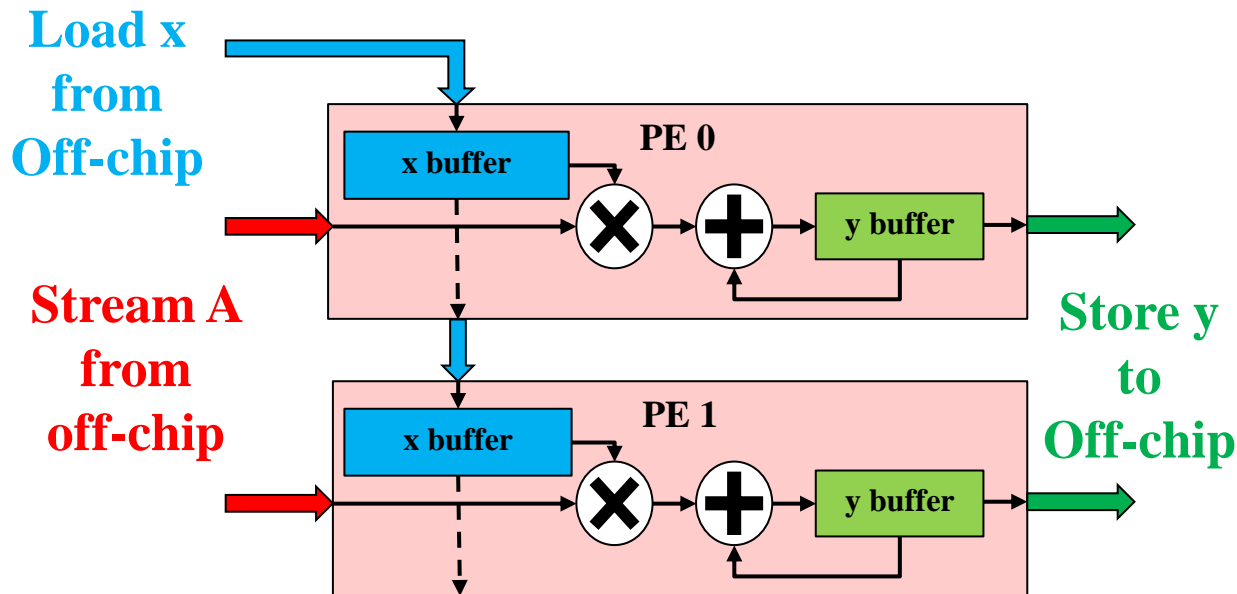
Cyclic row-wise distribution

## #2 High bandwidth requirement

- Multiple HBM channels to stream encoded sparse matrix

## #3 Workload distribution

	0	1	2	3
0		a		b
1	c			
2			d	
3	e			f



# Challenges to Accelerate SpMV and Prior Solutions

## #1 Random memory access

- On-chip buffers for dense vectors

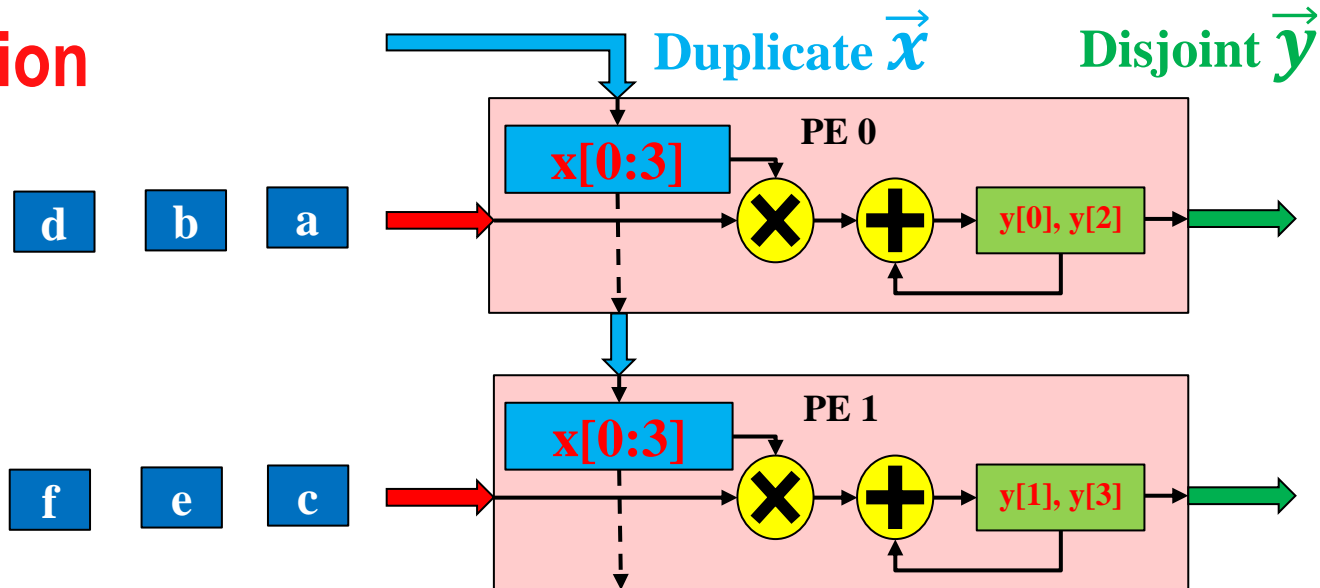
But this is still  
**NOT PERFECT!!**

## #2 High bandwidth requirement

- Multiple HBM channels to stream encoded sparse matrix

## #3 Workload distribution

















	0	1	2	3
0		a		b
1	c			
2			d	
3	e			f

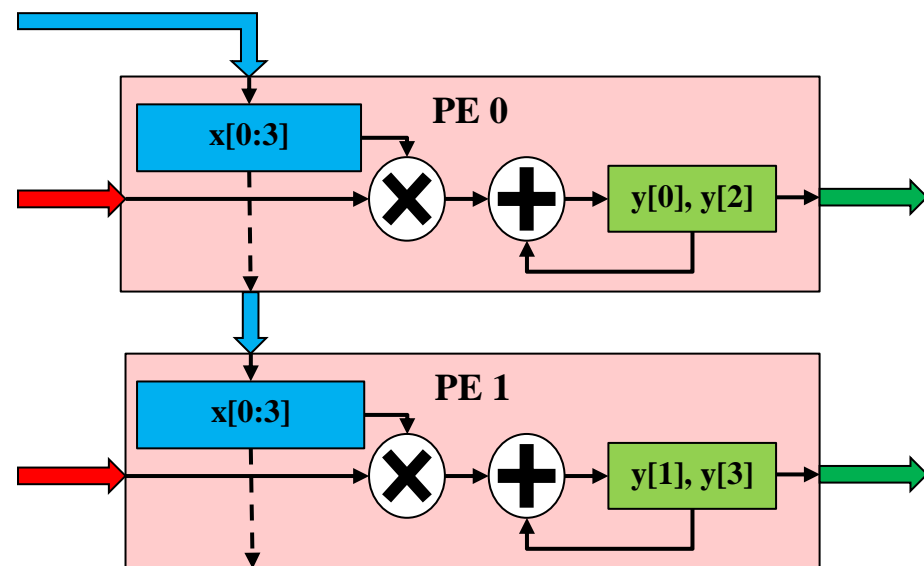




# Remaining Challenge #1: Imbalanced Row Distribution

## Imbalanced workload distribution

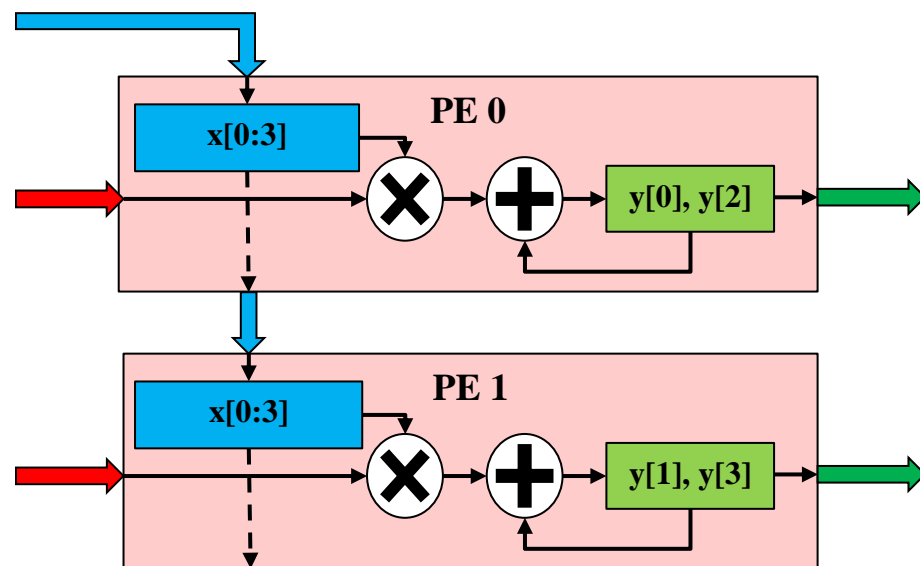
	0	1	2	3
0				
1				
2				
3				



# Our Solution for Imbalanced Row Distribution

## Imbalanced workload distribution

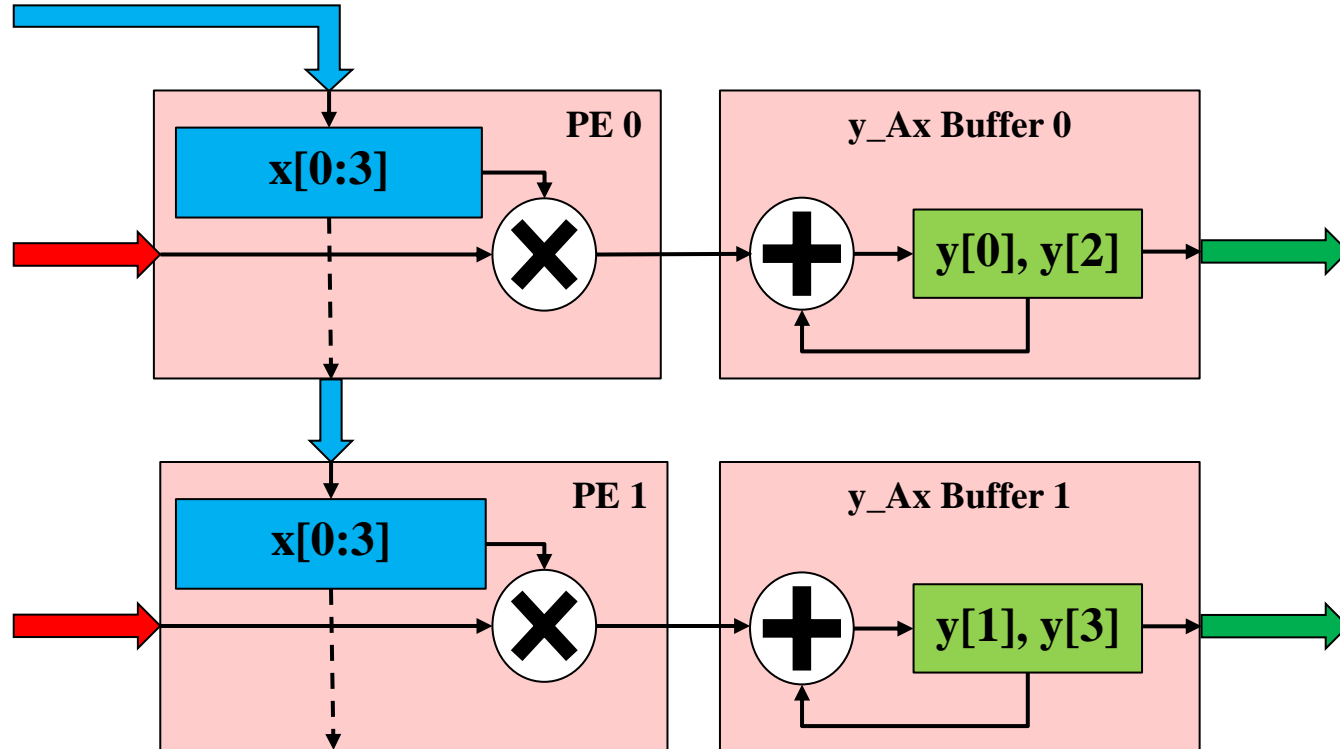
	0	1	2	3
0		a		
1	b	c	d	e
2				
3				f



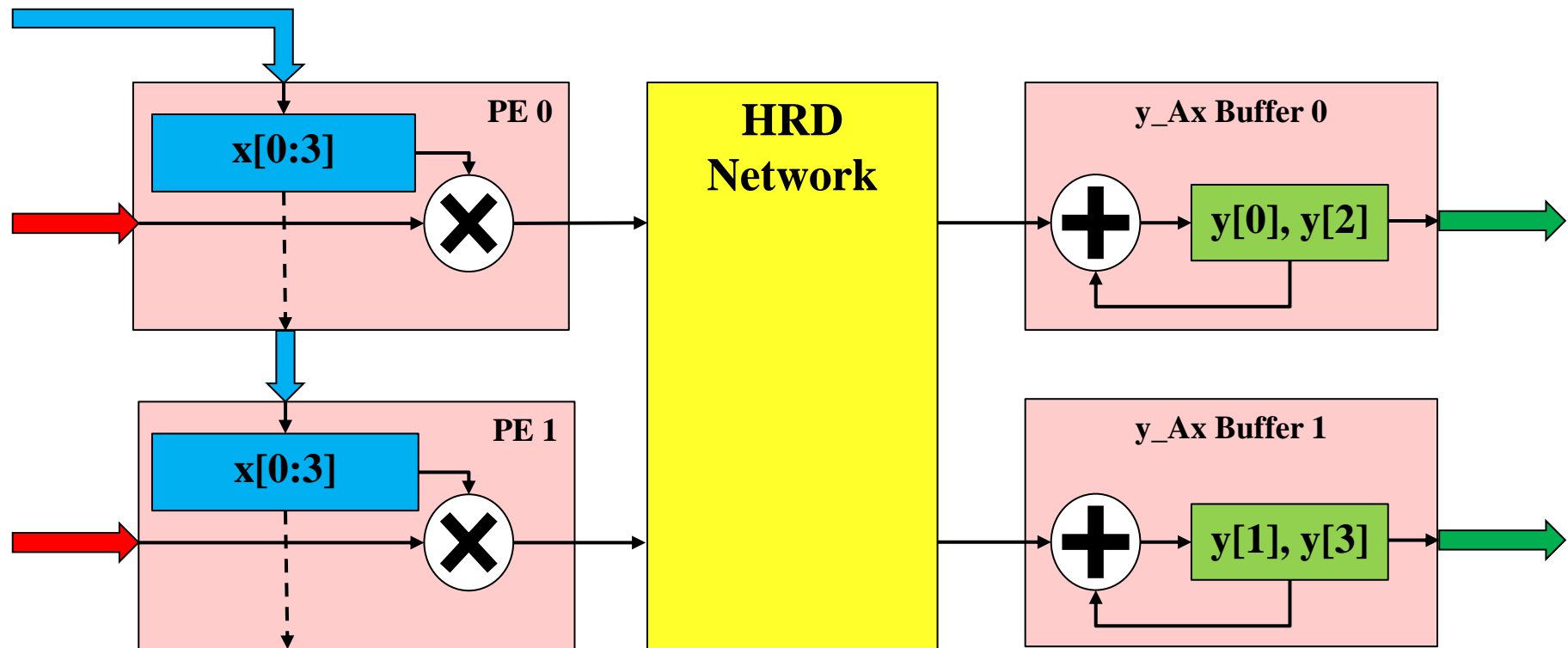
Make the hardware operate in **2** modes to achieve balance

- Inter-Row:** Individual PEs work on different rows as usual
- Intra-Row:** All the PEs work on the same row for dense rows

# Our Hybrid Row Distribution (HRD) Network

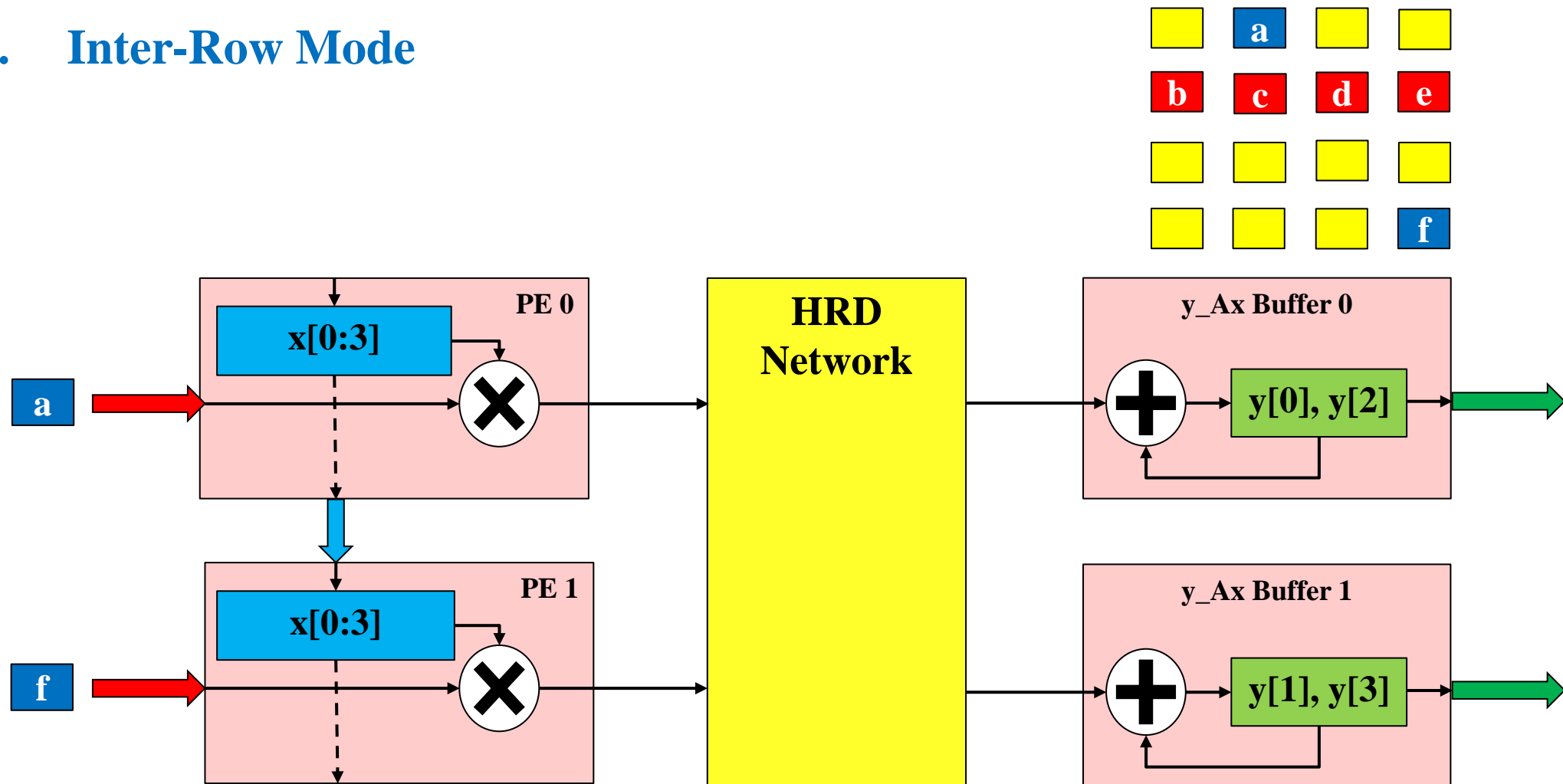


# Our Hybrid Row Distribution (HRD) Network



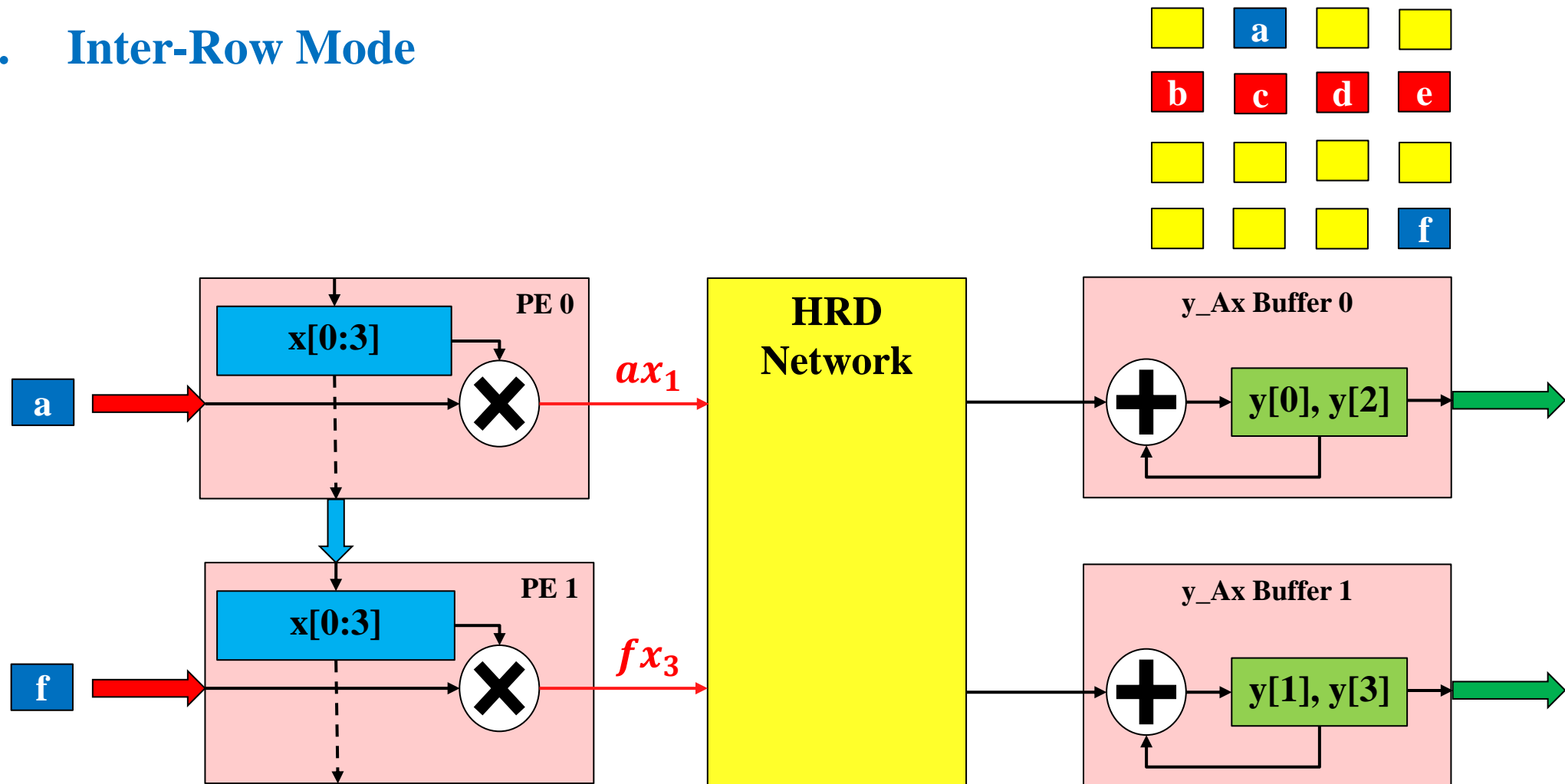
# Our Hybrid Row Distribution (HRD) Network

## i. Inter-Row Mode



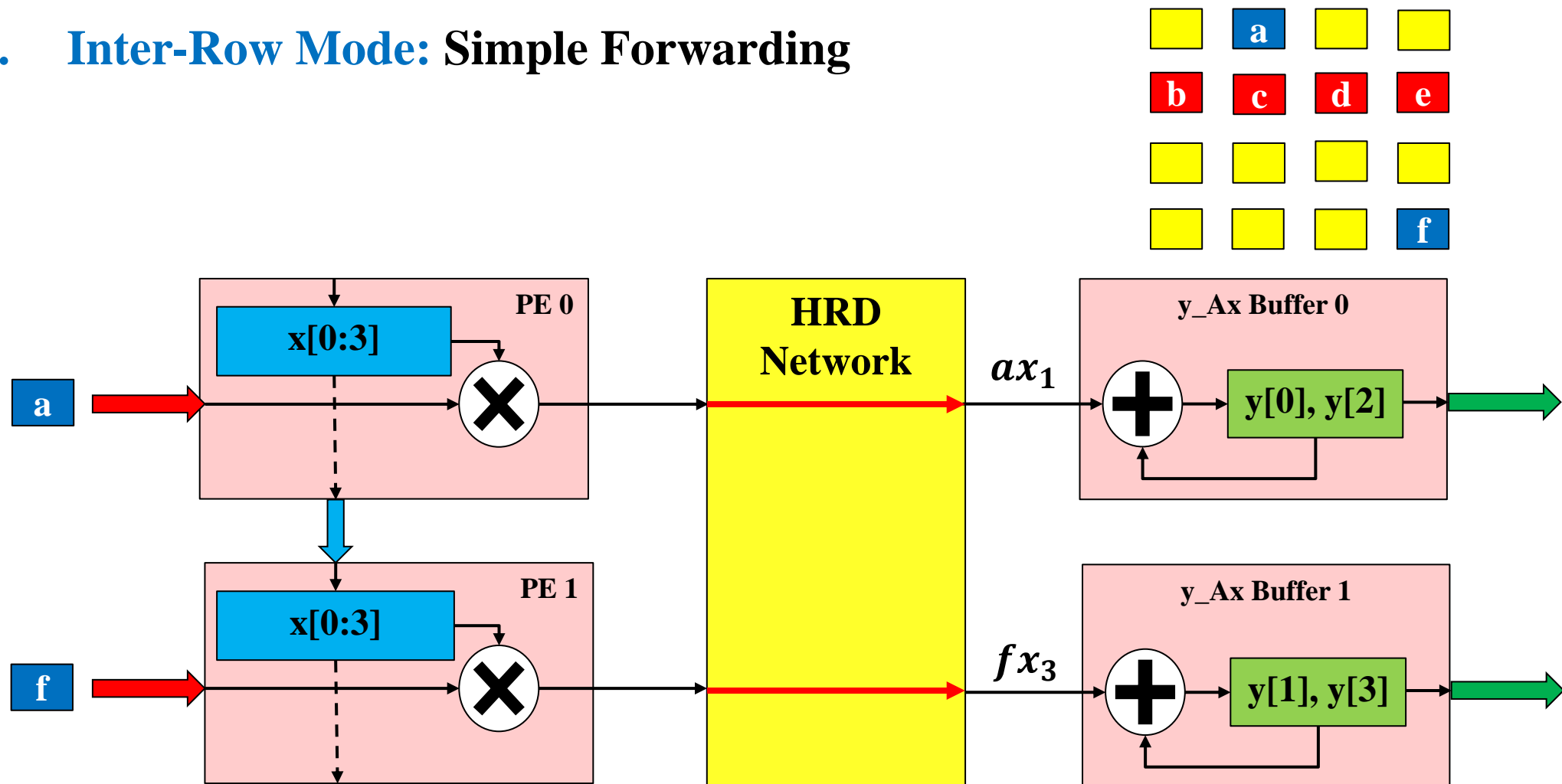
# Our Hybrid Row Distribution (HRD) Network

## i. Inter-Row Mode



# Our Hybrid Row Distribution (HRD) Network

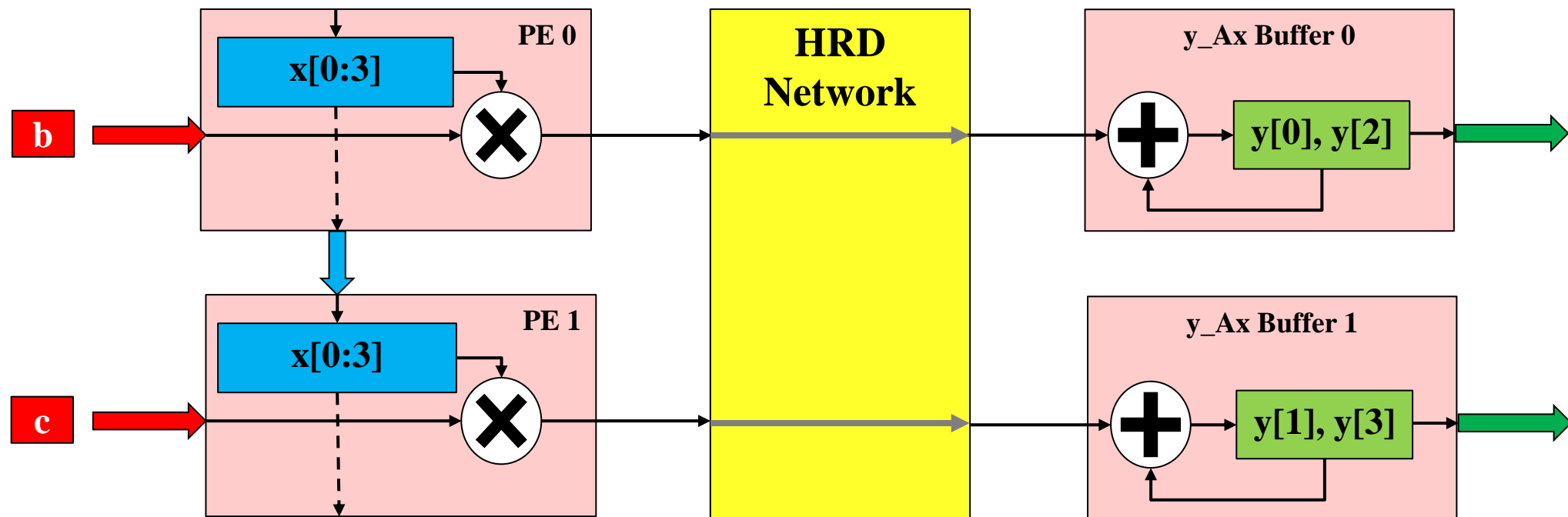
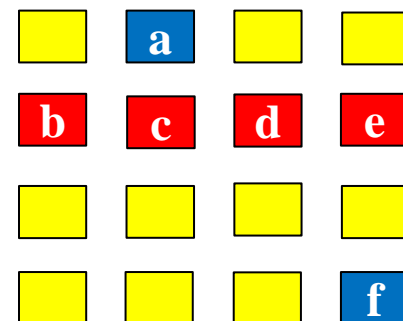
## i. Inter-Row Mode: Simple Forwarding



# Our Hybrid Row Distribution (HRD) Network

i. Inter-Row Mode: Simple Forwarding

ii. Intra-Row Mode:

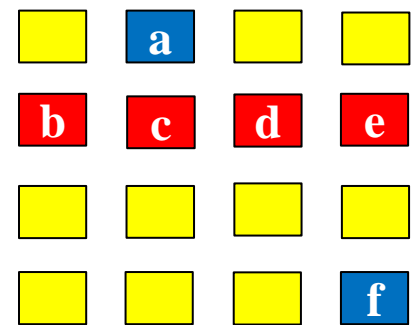




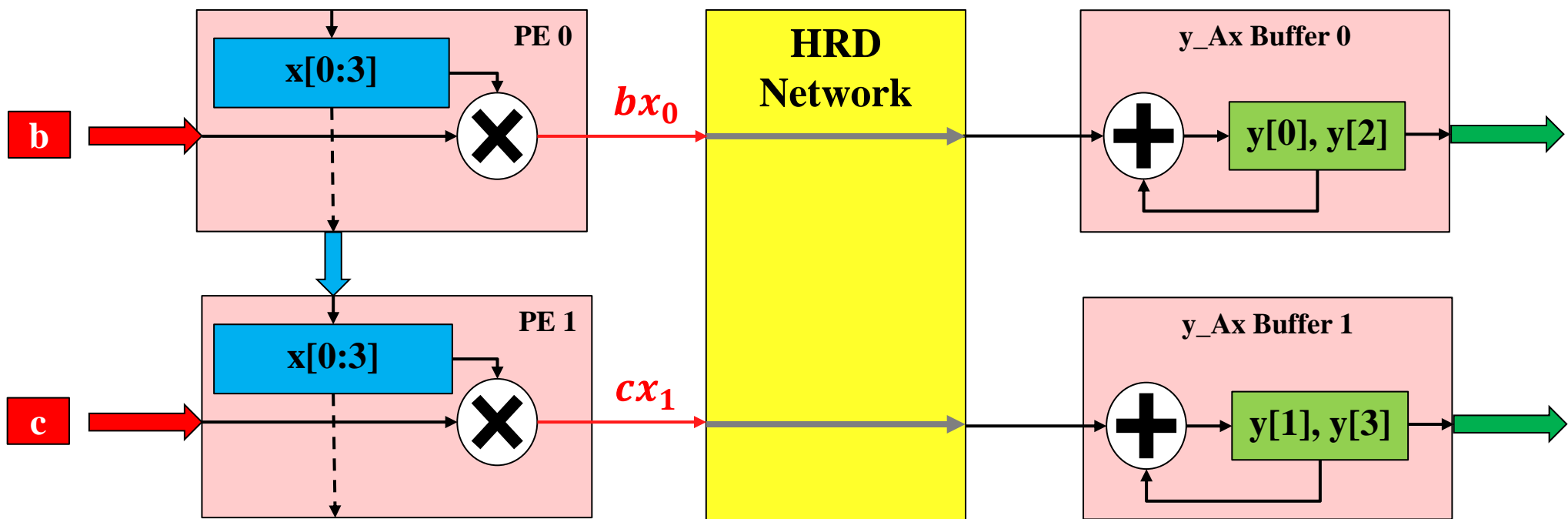
# Our Hybrid Row Distribution (HRD) Network

i. Inter-Row Mode: Simple Forwarding

ii. Intra-Row Mode:

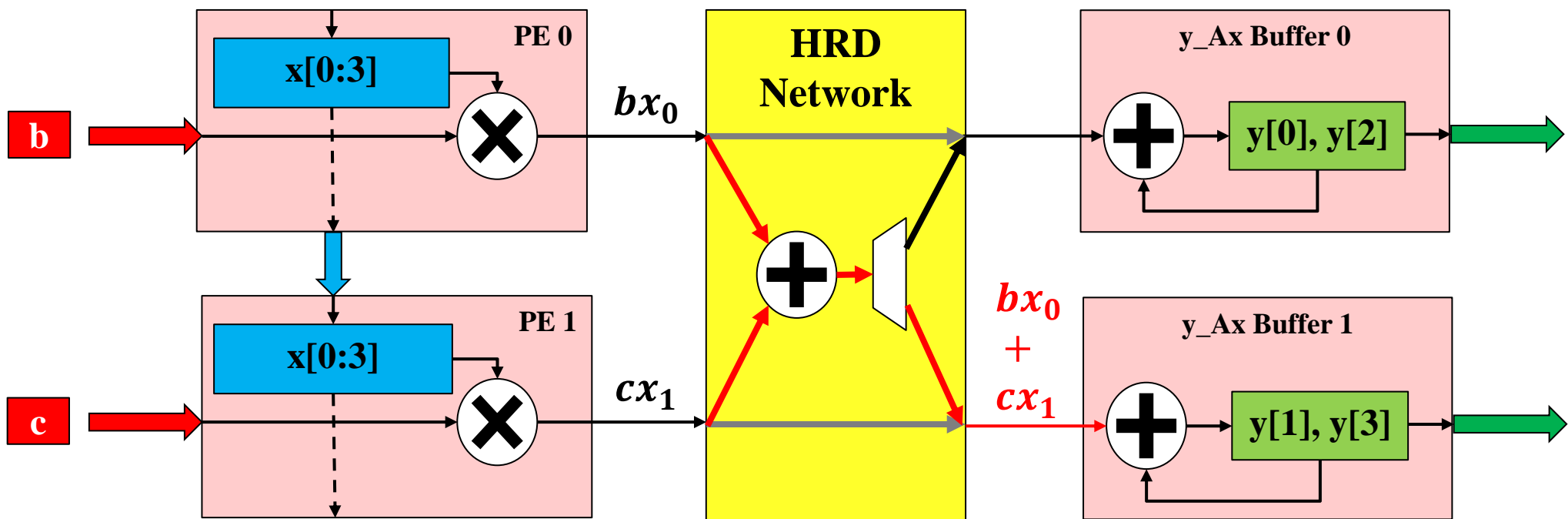
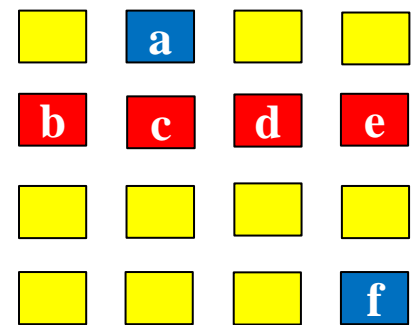


Both need to be added to  $y[1]$



# Our Hybrid Row Distribution (HRD) Network

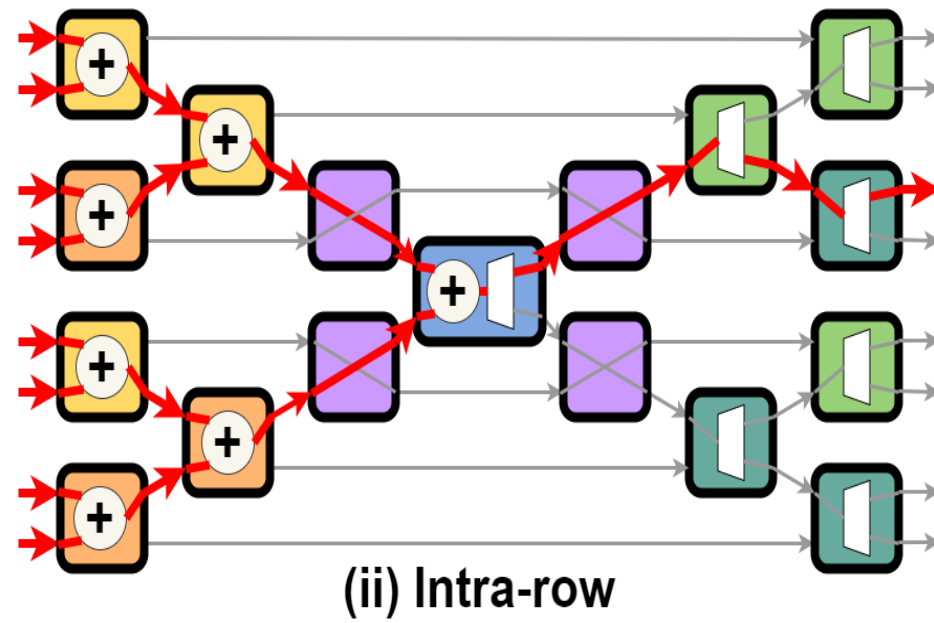
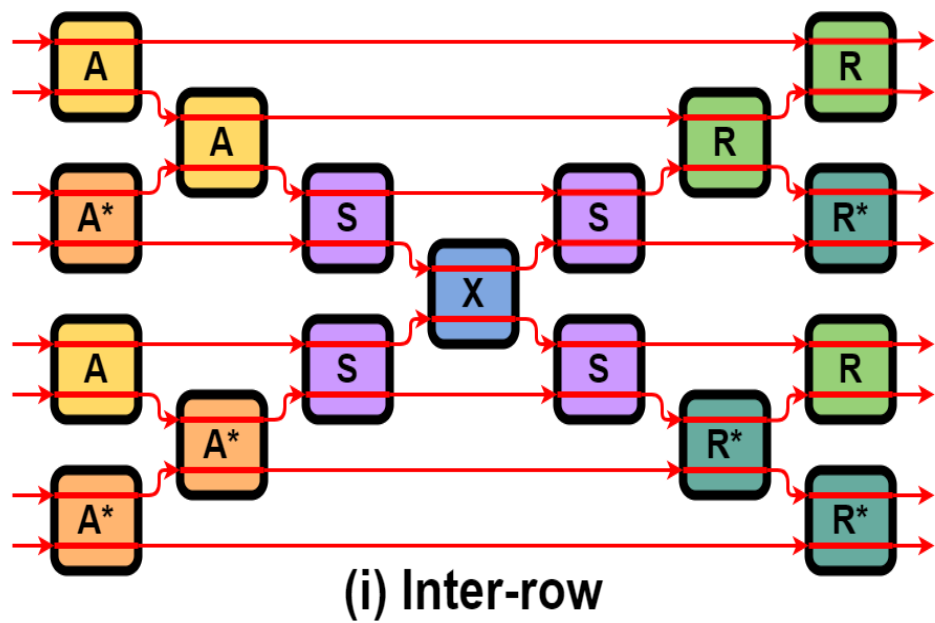
- i. Inter-Row Mode: Simple Forwarding
- ii. Intra-Row Mode: Reduce and Route



# Our Hybrid Row Distribution (HRD) Network

Scaling up for larger design, e.g., 8 PEs

- Modular and flexible

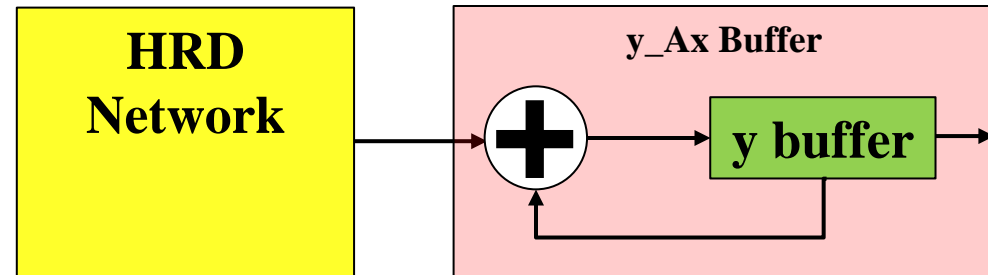
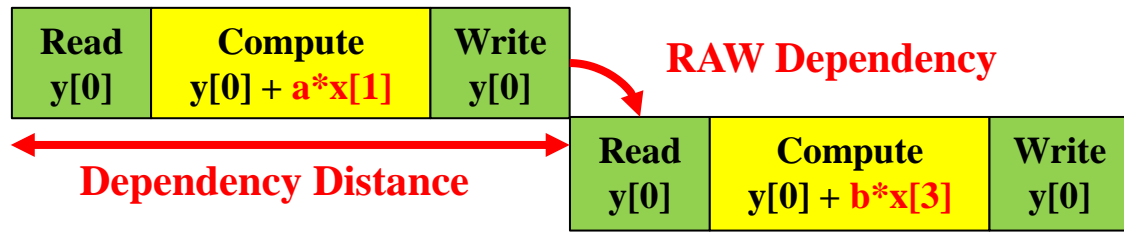


# Remaining Challenge #2: FP Accumulation Dependency

In FPGAs without hardened floating point accumulators, the accumulation takes multiple clock cycles

**A Matrix**

	0	1	2	3
0		a		b
1				
2	c	d		
3				

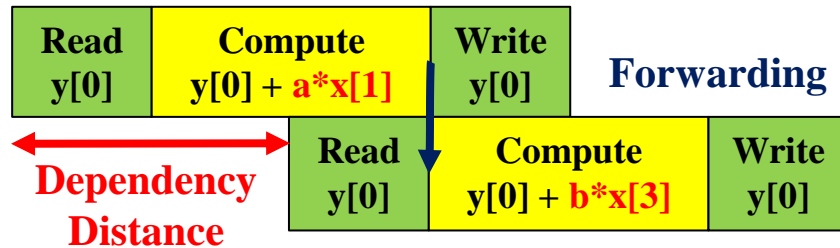
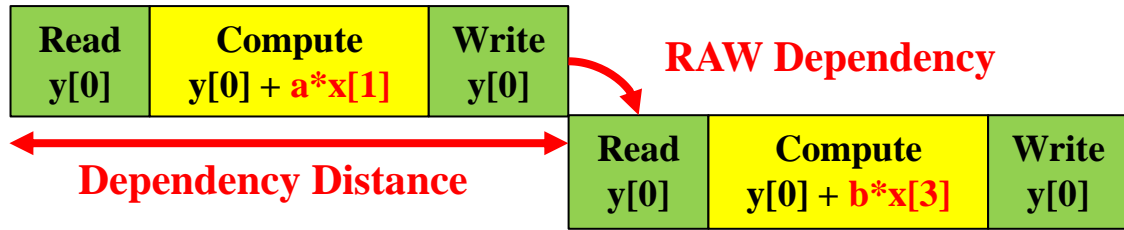


# Our Solution Part 1: Reduce Dependency Distance

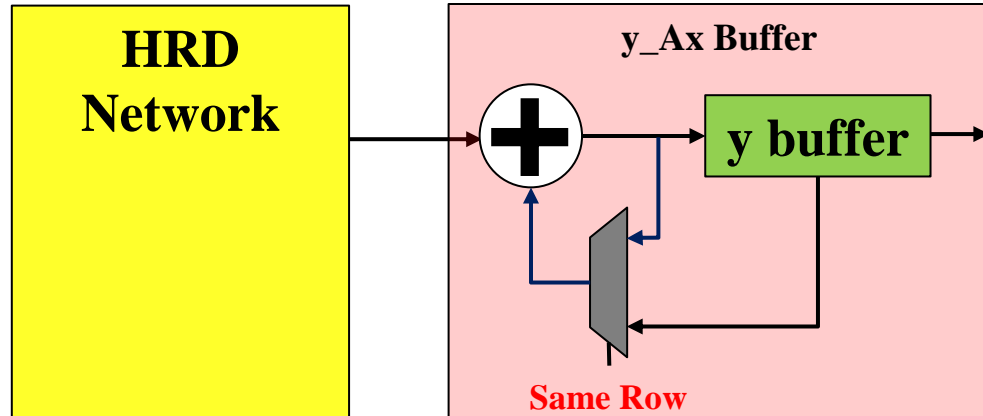
Remove buffer access latency

A Matrix

	0	1	2	3
0		a		b
1				
2	c	d		
3				



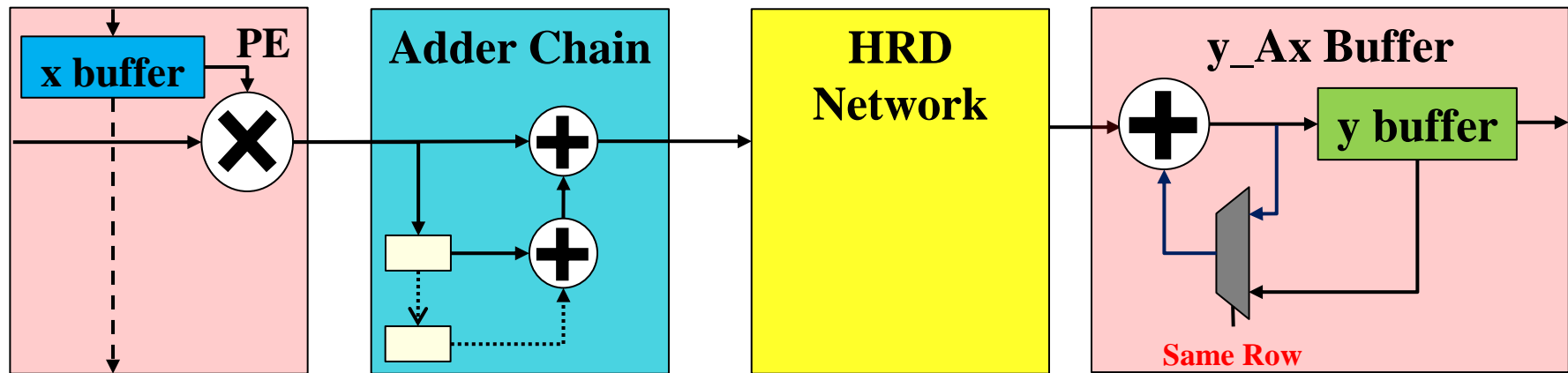
Local data forwarding reduces the Dependency Distance (DD); but still cannot achieve  $\Pi = 1$



# Our Solution Part 2: Pre-Accumulation Adder Chain

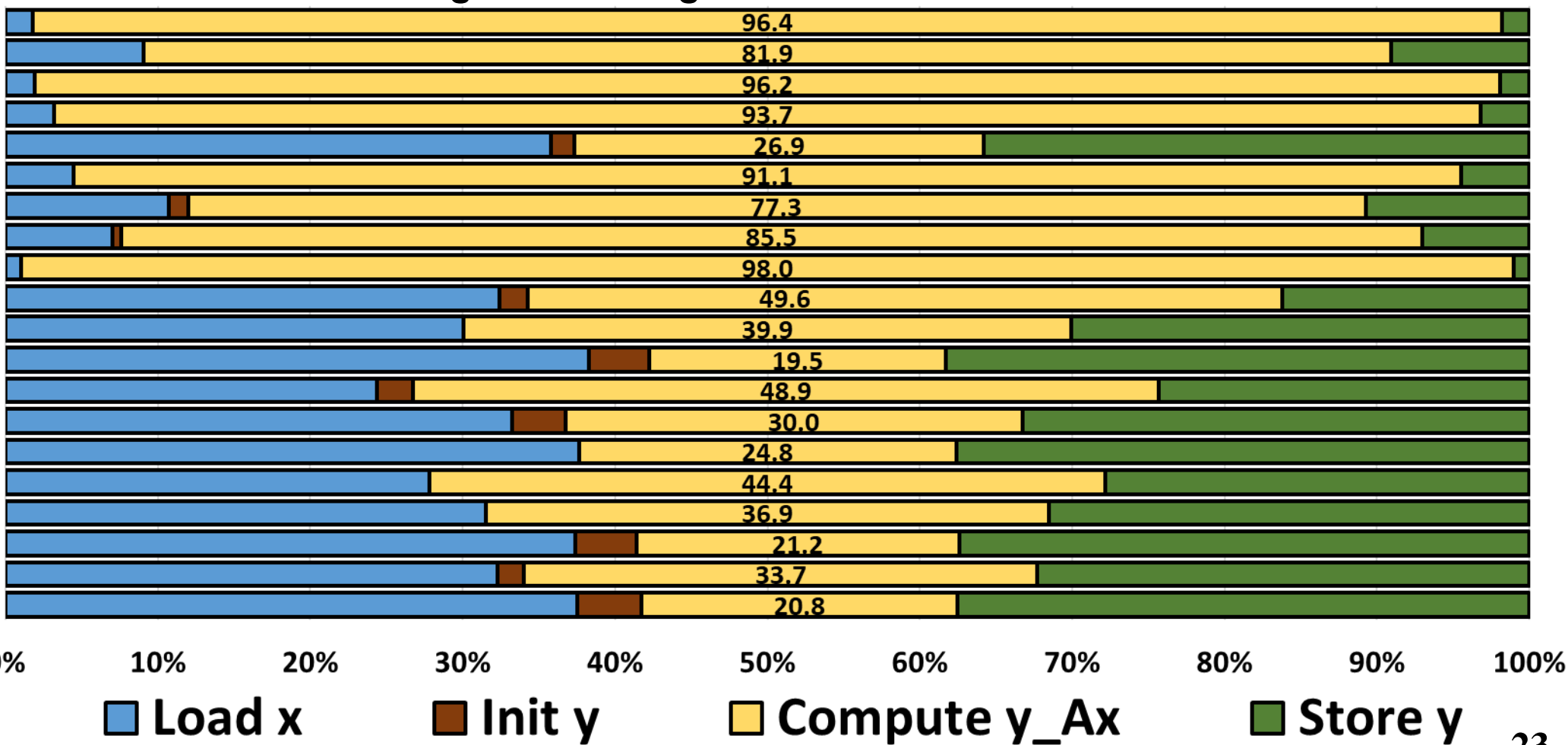
## Pre-accumulation adder chain

- Store and accumulate  $DD - 1$  results
- Achieve  $II = 1$

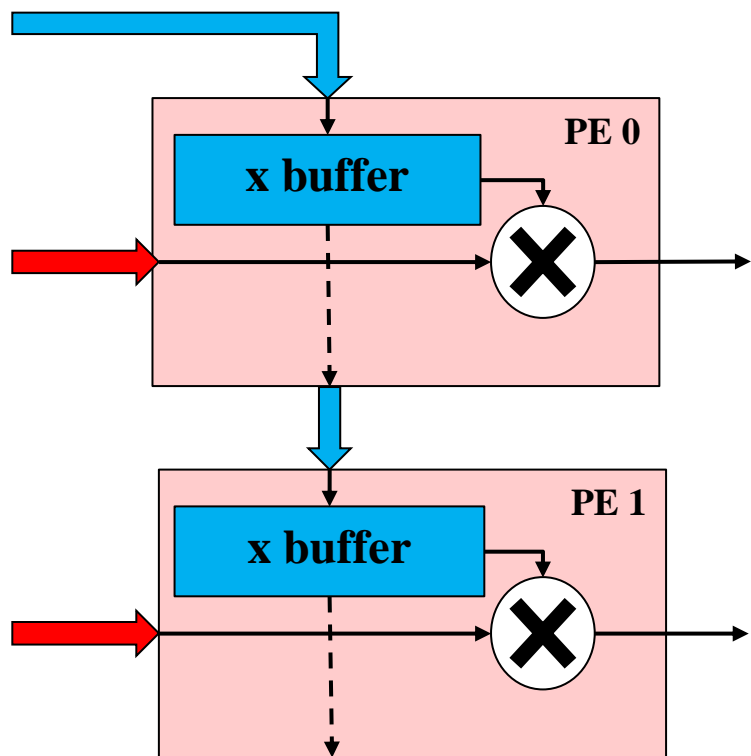


# New Performance Bottleneck

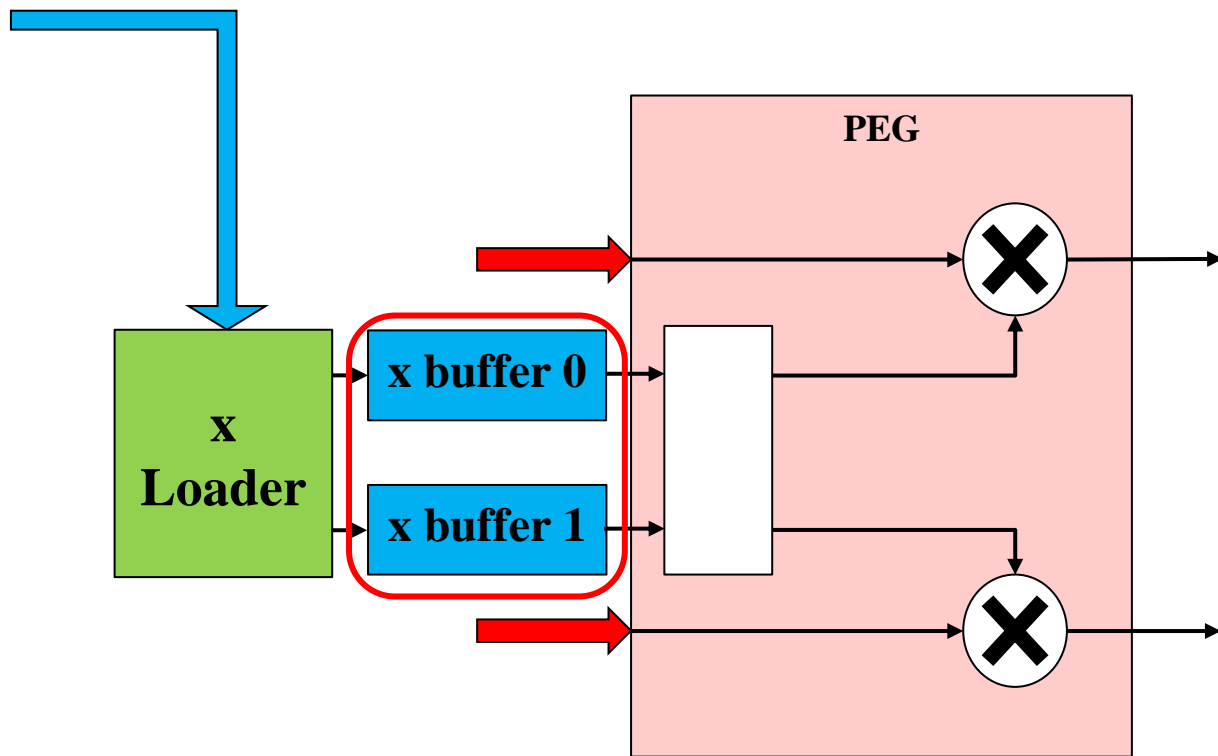
## Dense vectors loading and storing



# Our Solution: Hybrid Buffer



**Old Design**



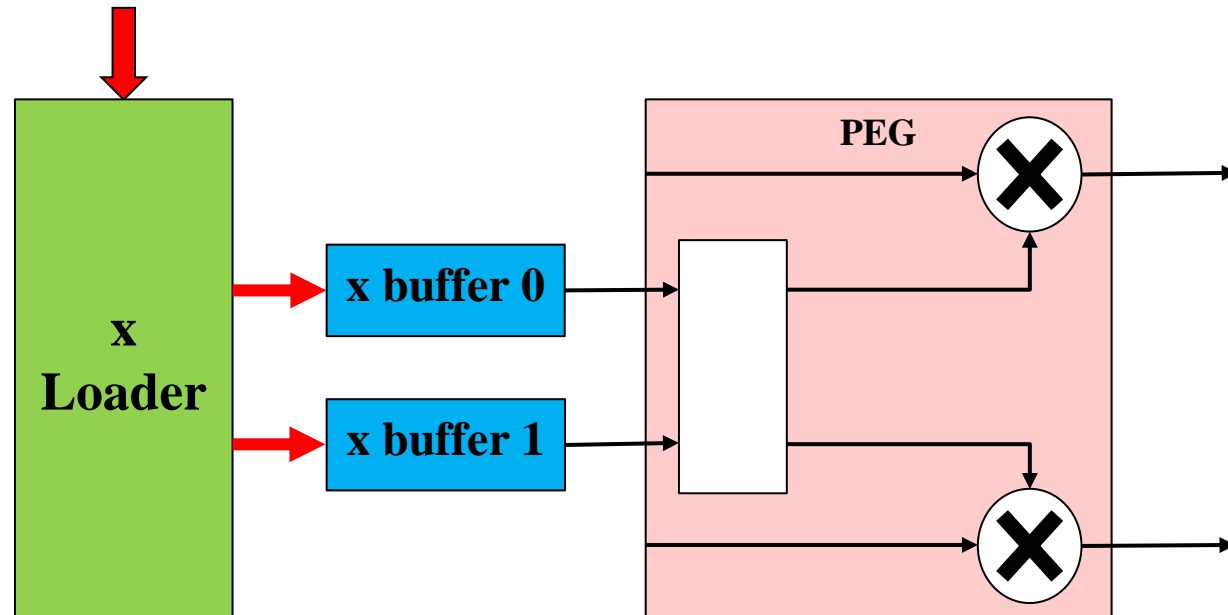
**New Design**



# Hybrid Buffer: Sequential Mode

## Sequential Mode

- Load to both buffers



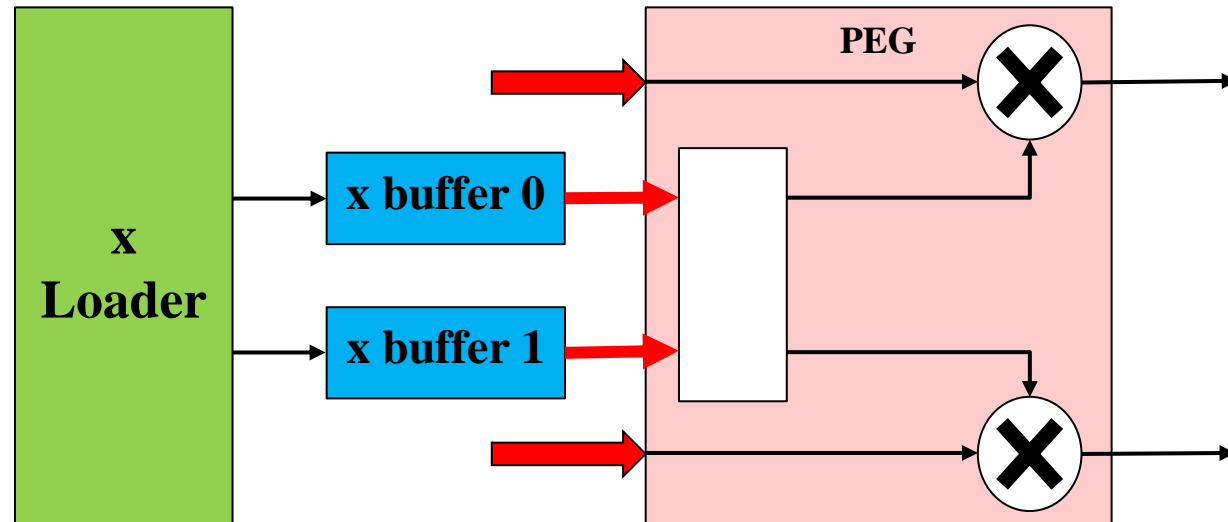
# Hybrid Buffer: Sequential Mode

## Sequential Mode

- Load to both buffers
- Consume from both buffers

Sequential mode time  
is Load time + Compute time

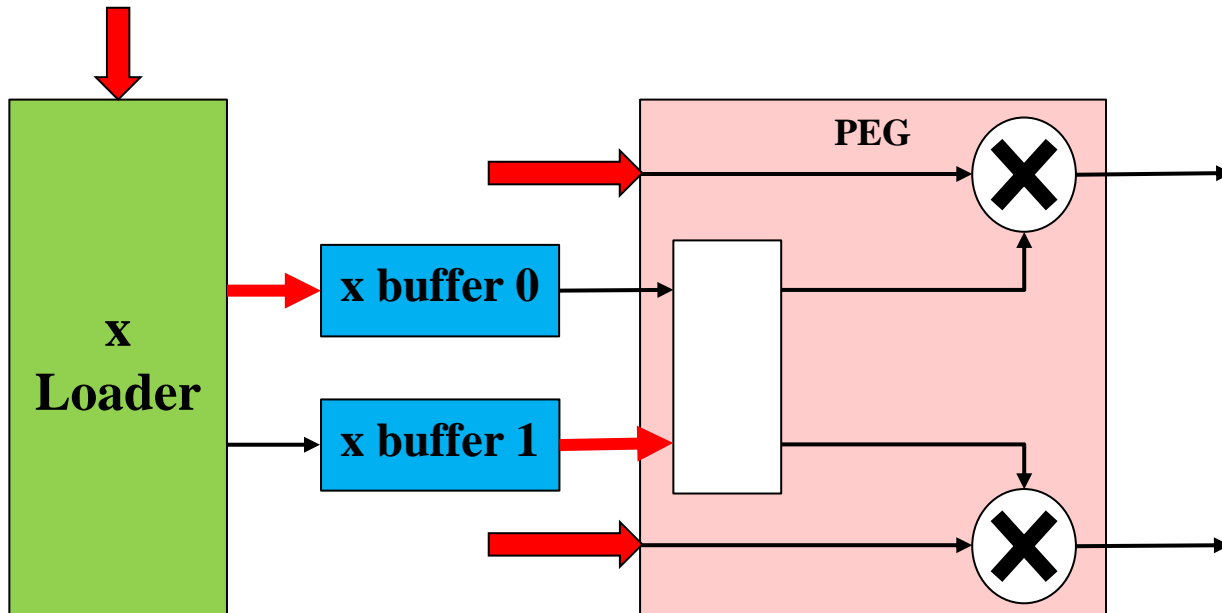
$$t_s = t_L + t_C$$



# Hybrid Buffer: Ping-Pong Mode

## Ping-Pong Mode

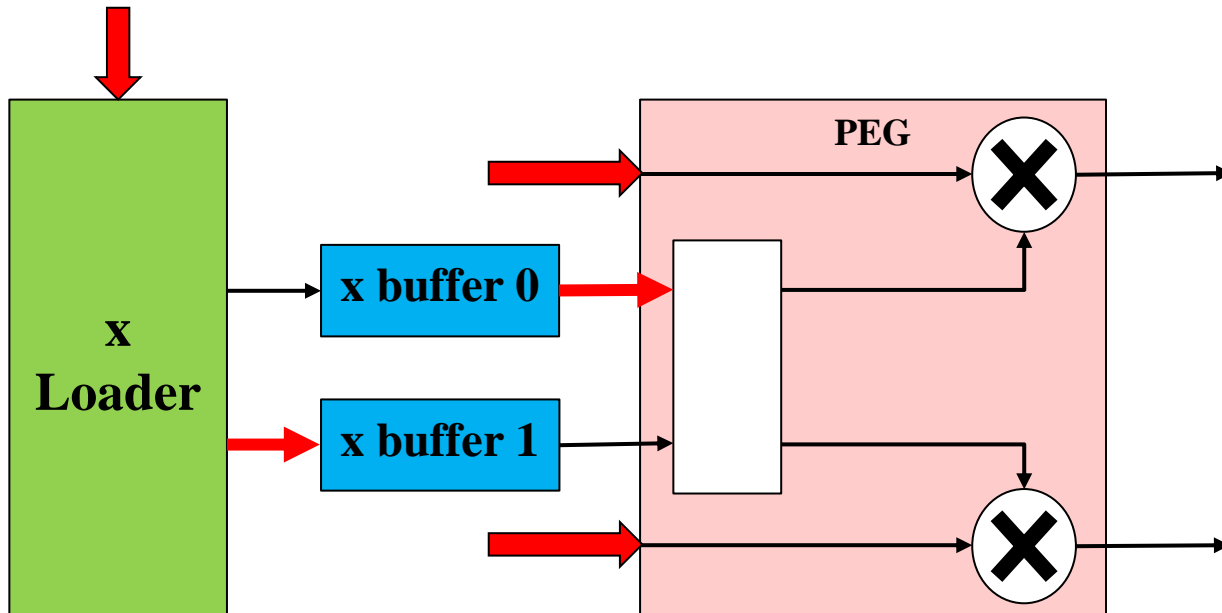
- Load to buffer 0 and consume from buffer 1



# Hybrid Buffer: Ping-Pong Mode

## Ping-Pong Mode

- Load to buffer 0 and consume from buffer 1
- Load to buffer 1 and Consume from buffer 0



Compute time will vary

$$t_c \leq t'_c \leq 2t_c$$

Ping-Pong mode time is the maximum between Load time and Compute time

$$t_p = \max(t_L, t'_c)$$

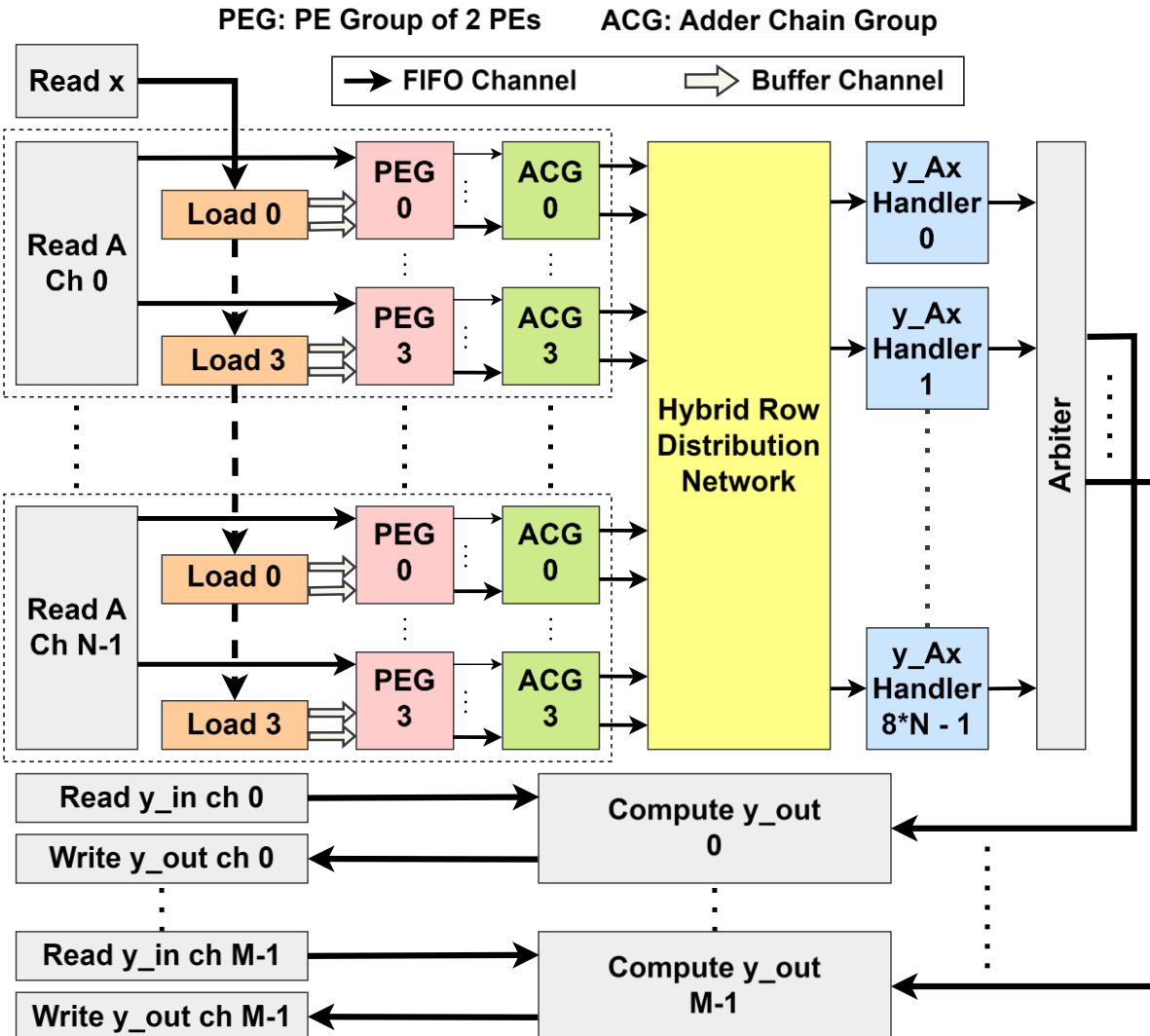
# Hybrid Buffer Implementation and Selection

Implemented using **PASTA buffer channels** [FCCM 2023]

Condition for Ping-Pong mode:

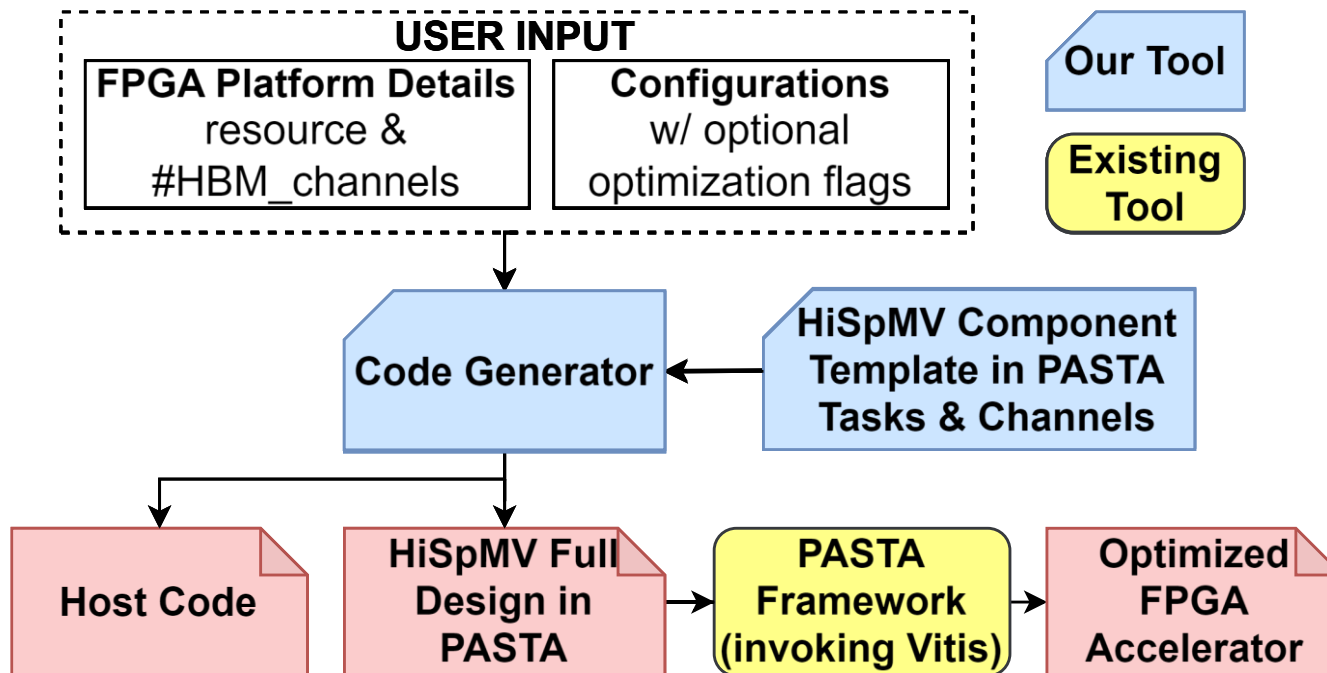
- $t_S > t_P \Rightarrow t_L + t_C > \max(t_L, t'_C)$
- When  $\max(t_L, t'_C) = t'_C$ ;  $t_L + t_C > t'_C$
- Worst case compute time  $t'_C = 2t_C$
- **When  $t_L > t_C$ , Ping-Pong mode will be dynamically picked**

# HiSpMV Overall Architecture: Put It Together



- **$N$  channels** to stream sparse matrix  $A$
- **$M$  channels** for dense vector  $y$
- **1 channel** for dense vector  $x$ , with chain broadcast
- **Both  $M$  and  $N$  are scalable**

# HiSpMV Code Generator



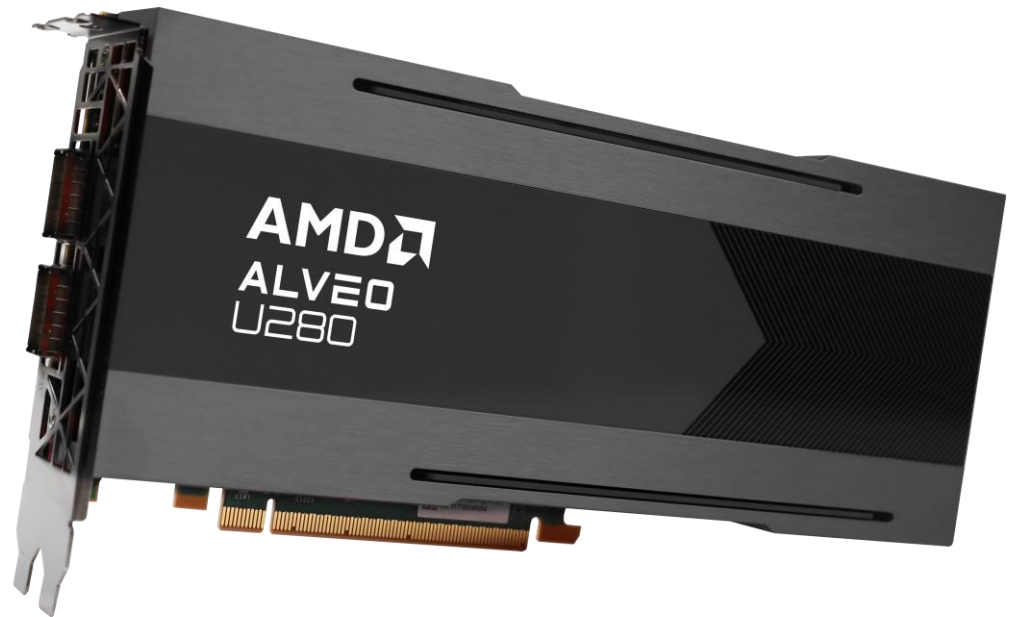
- ❑ Choose to build w/ or w/o **Adder Chains** and **Hybrid Buffer**
- ❑ Get estimated runtimes for **DSE**
- ❑ Generate HiSpMV design with custom values for **M and N**

- ❑ Open source: <https://github.com/SFU-HiAccel/HiSpMV>

# Experimental Setup 1

## FPGA Designs (Alveo U280)

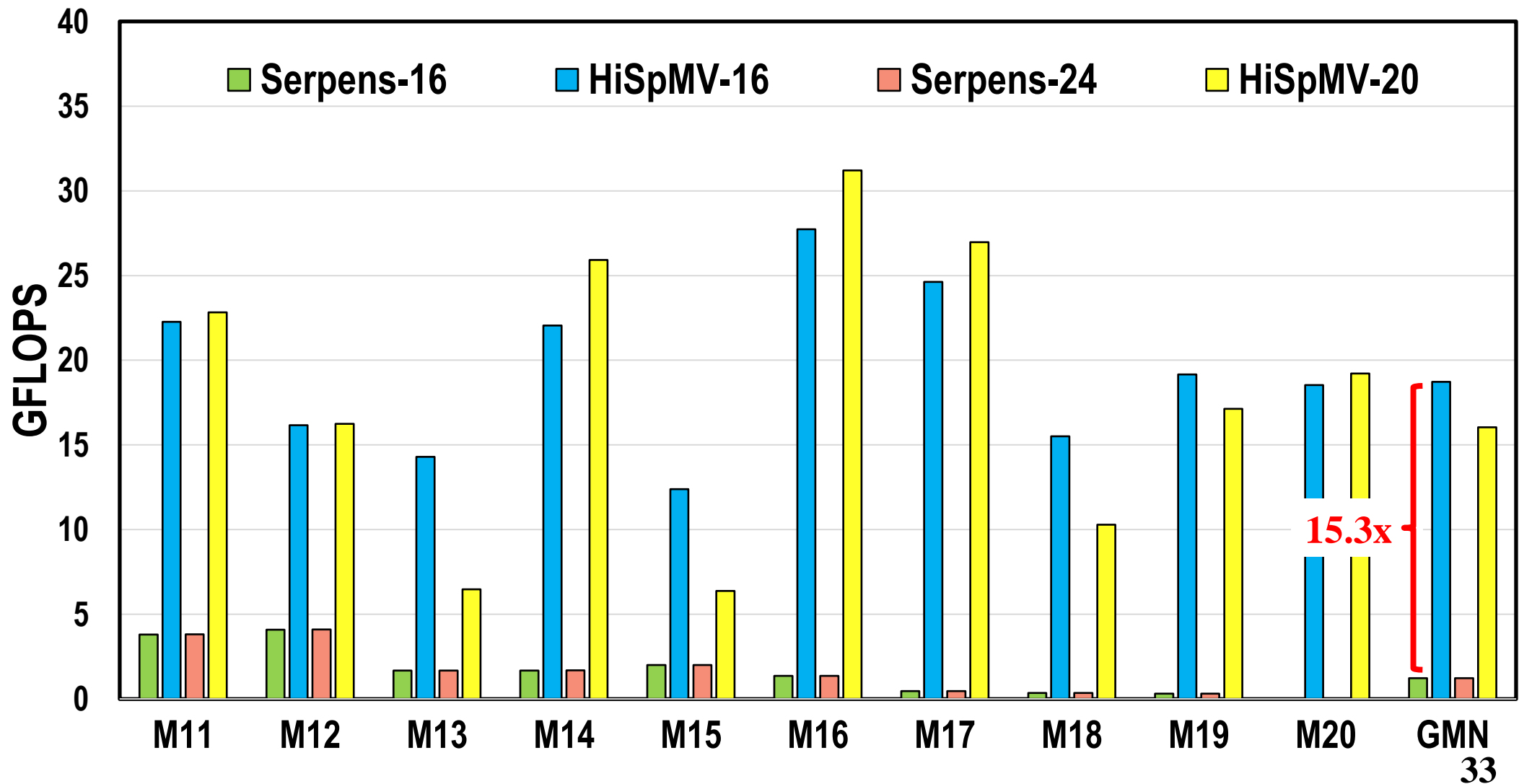
- **HiSpMV-16 (This Work)**
  - 128 PEs (All Optimizations)
- **HiSpMV-20 (This Work)**
  - 160 PEs (No Adder Chains)
- **Serpens-16 [DAC 2022]**
  - 128 PEs
- **Serpens-24 [DAC 2022]**
  - 192 PEs



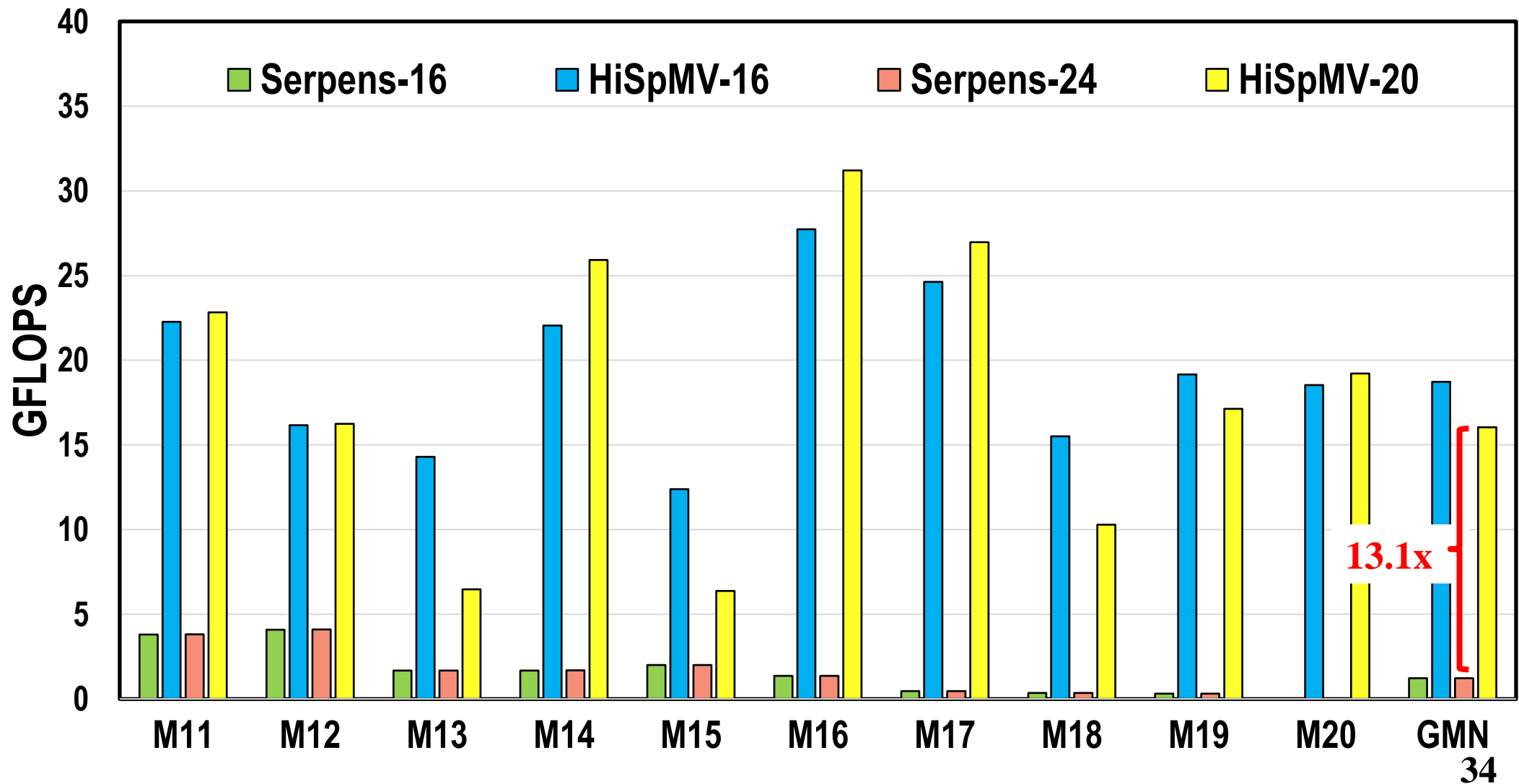
Process Technology	16 nm
HBM Bandwidth	460 GB/S



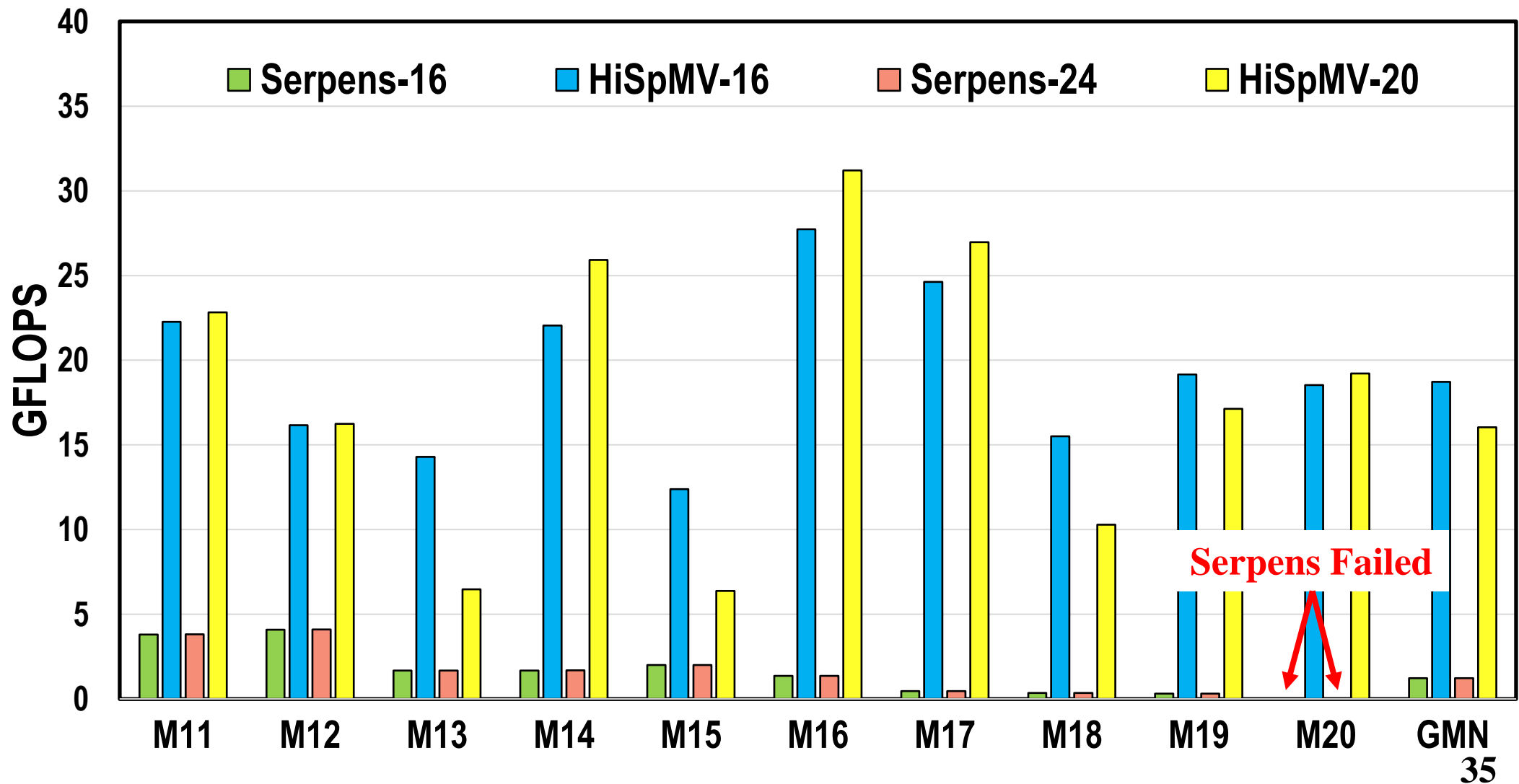
# Performance Comparison: Imbalanced Matrices



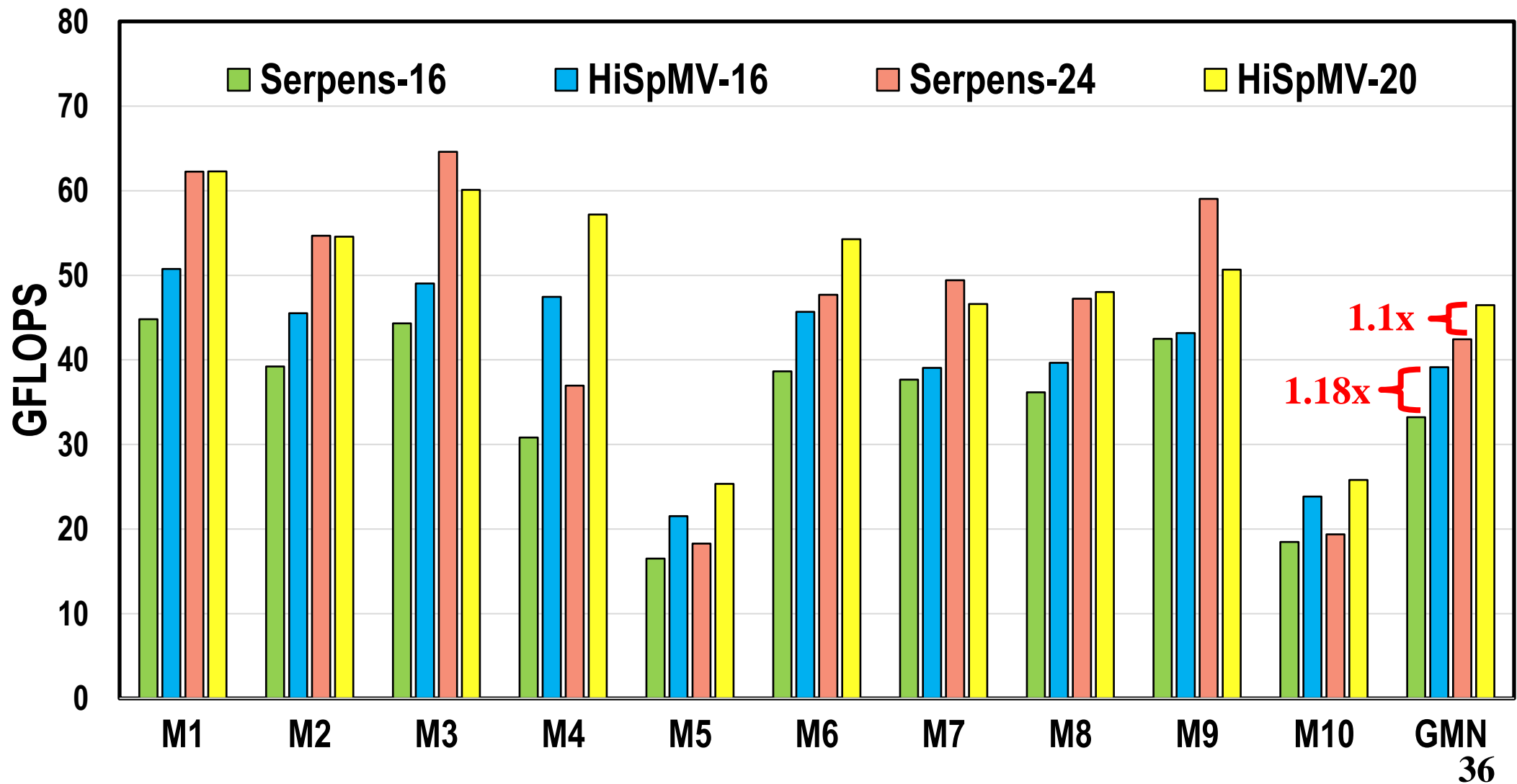
# Performance Comparison: Imbalanced Matrices



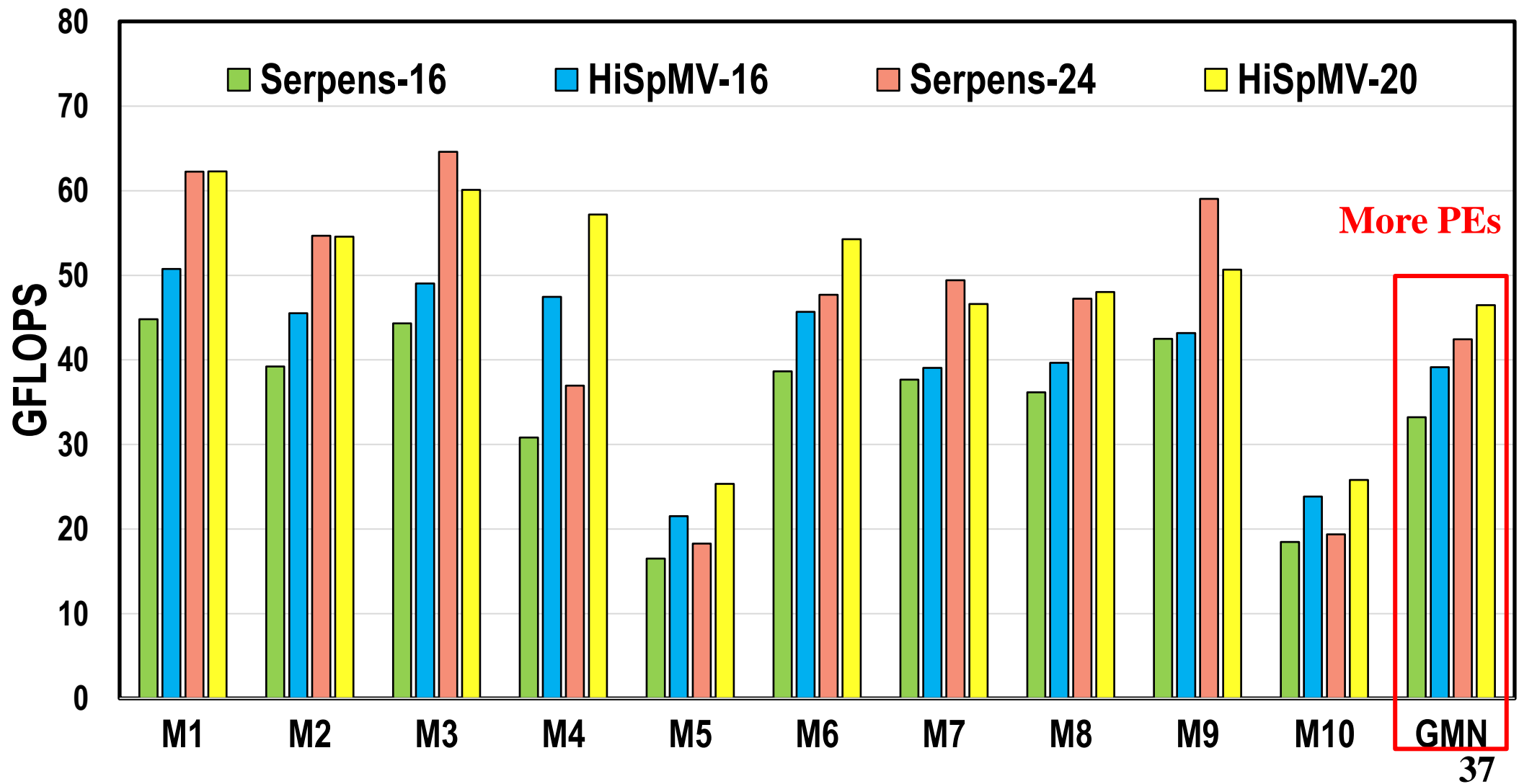
# Performance Comparison: Imbalanced Matrices





# Performance Comparison: Balanced Matrices





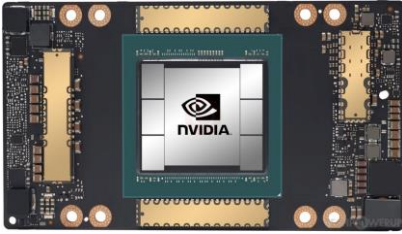
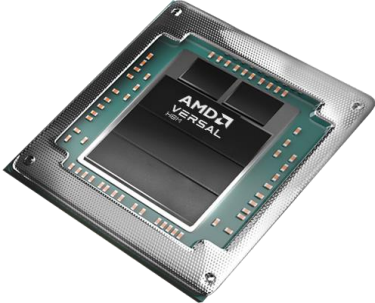
# Performance Comparison: Balanced Matrices



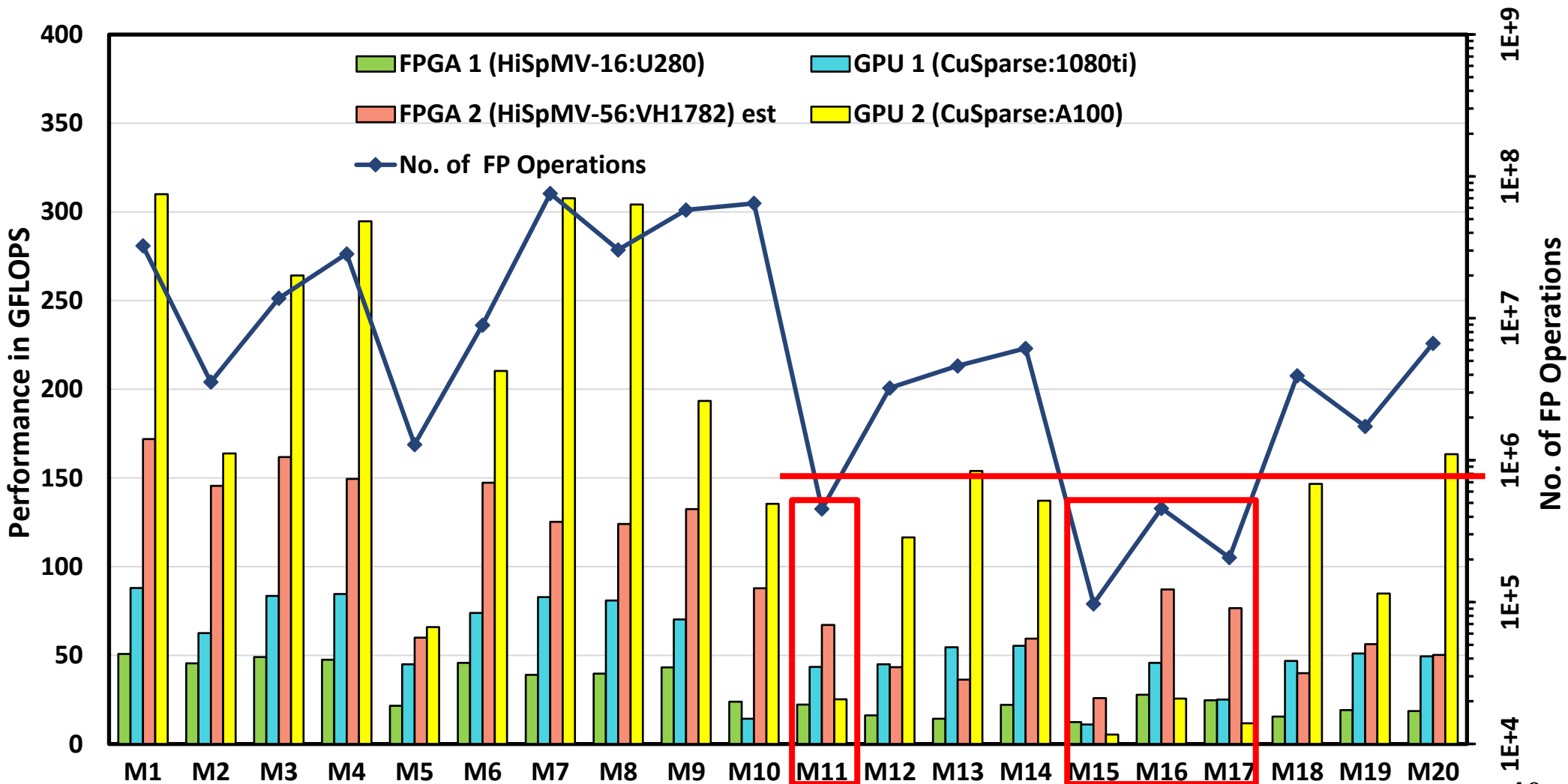
# Experimental Setup 2

Name	GPU 1	FPGA 1
Image		
Device	NVIDIA GTX 1080Ti	AMD Alveo U280
Kernel	NVIDIA CuSparse	HiSpMV-16
Memory Bandwidth	484 GB/s	460 GB/s
Process Technology	16 nm	16 nm

# Experimental Setup 2

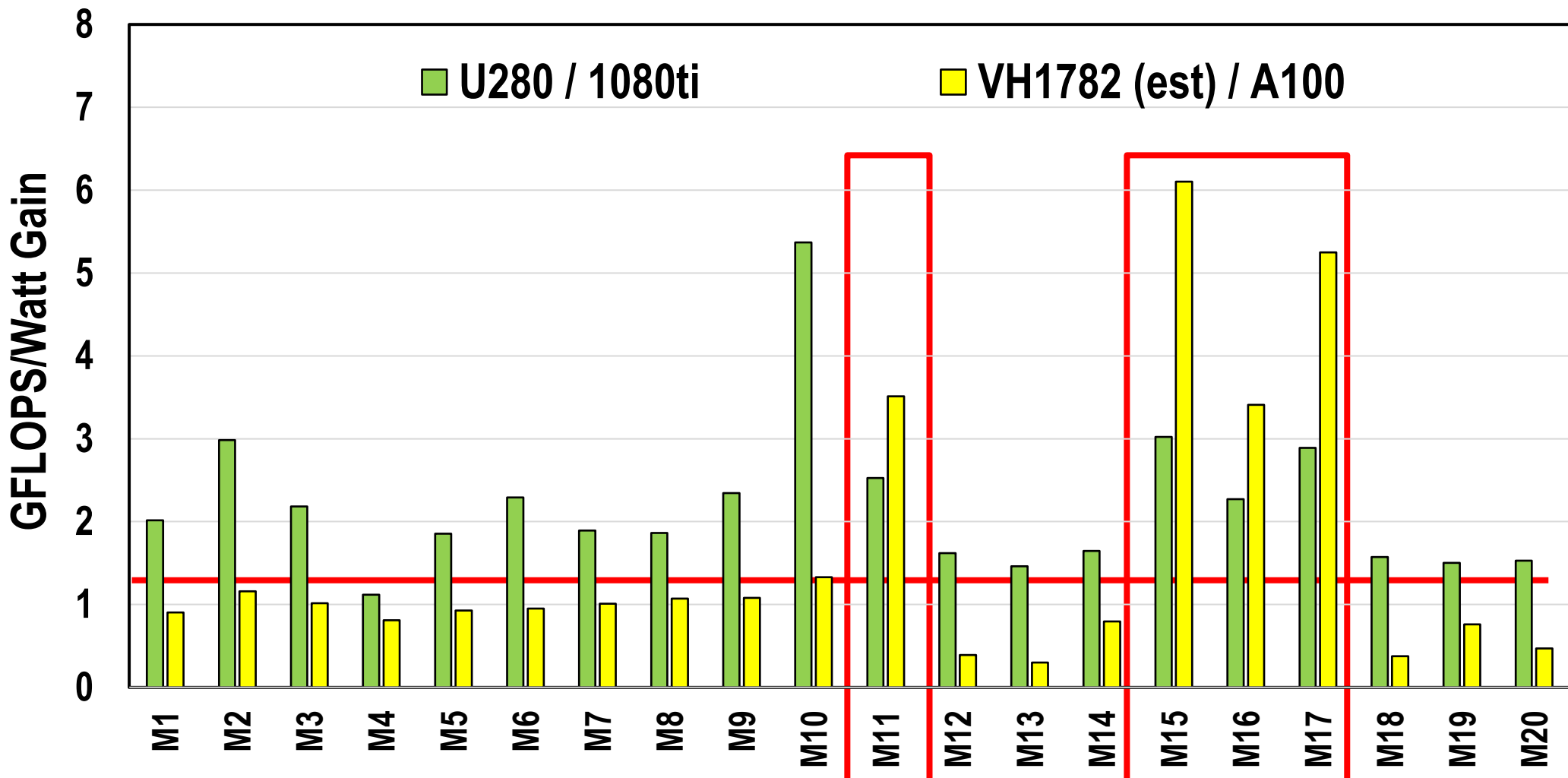
Name	GPU 1	FPGA 1	GPU 2	FPGA 2 (est)
Image				
Device	NVIDIA GTX 1080Ti	AMD Alveo U280	NVIDIA A100	AMD Versal VH1782
Kernel	NVIDIA CuSparse	HiSpMV-16	NVIDIA CuSparse	HiSpMV-56
Memory Bandwidth	484 GB/s	460 GB/s	1,555 GB/s	819 GB/s
Process Technology	16 nm	16 nm	7 nm	7 nm

# Performance Comparison





# Energy Efficiency Comparison



# Conclusion

## Key contributions in HiSpMV

- Hybrid row distribution for imbalanced workload
- Adder chains and register forwarding for FP accumulation dependency
- Hybrid buffering for dense vector access bottleneck
- Code generator for customizable Hardware

## Experimental results on Alveo U280 FPGA

- 15.3x geomean speedup over Serpens (SOTA) for imbalanced matrices
- 1.93x geomean better performance per watt over 1080ti GPU

Open source: <https://github.com/SFU-HiAccel/HiSpMV>

# Thank You!



Paderborn  
Center for  
Parallel  
Computing



HiAccel group website: <http://www.sfu.ca/~zhenman/group.html>