

pSyncPIM: Partially Synchronous Execution of Sparse Matrix Operations for All-Bank PIM Architectures

Daehyeon Baek^{1†}, Soojin Hwang², Jaehyuk Huh²

¹Technology Research, Samsung SDS

²School of Computing, KAIST

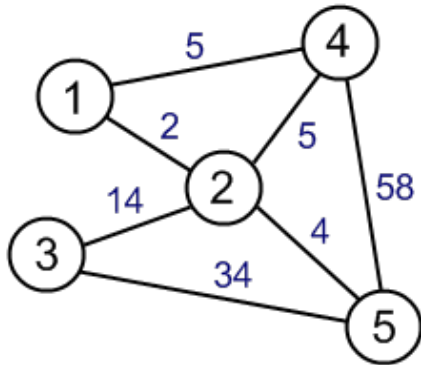
† This work was conducted at KAIST.



Major Problems of Sparse Matrix

Graph Applications

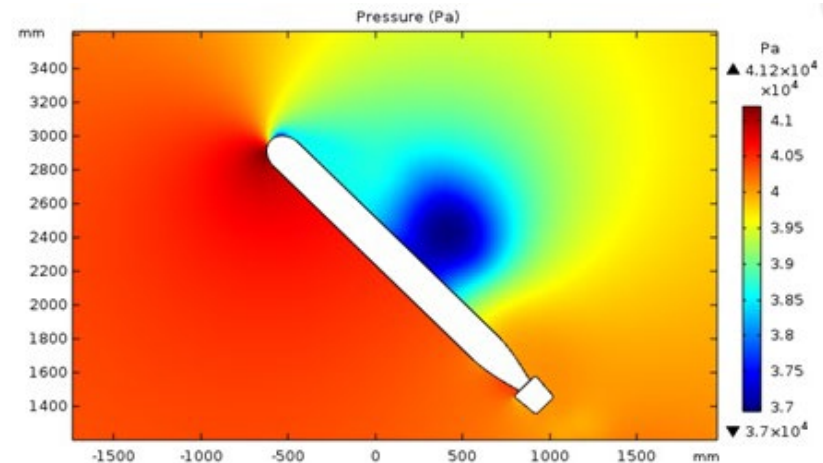
- Linear algebra as “compute language”
- Transformation by adjacency matrix
- GraphBLAS: framework for graphs



Graph Example

Numerical Linear Algebra

- Complex problems to linear systems
 - ex) differential equations
- Solve $A\vec{x} = \vec{b}$ for given A and \vec{b}

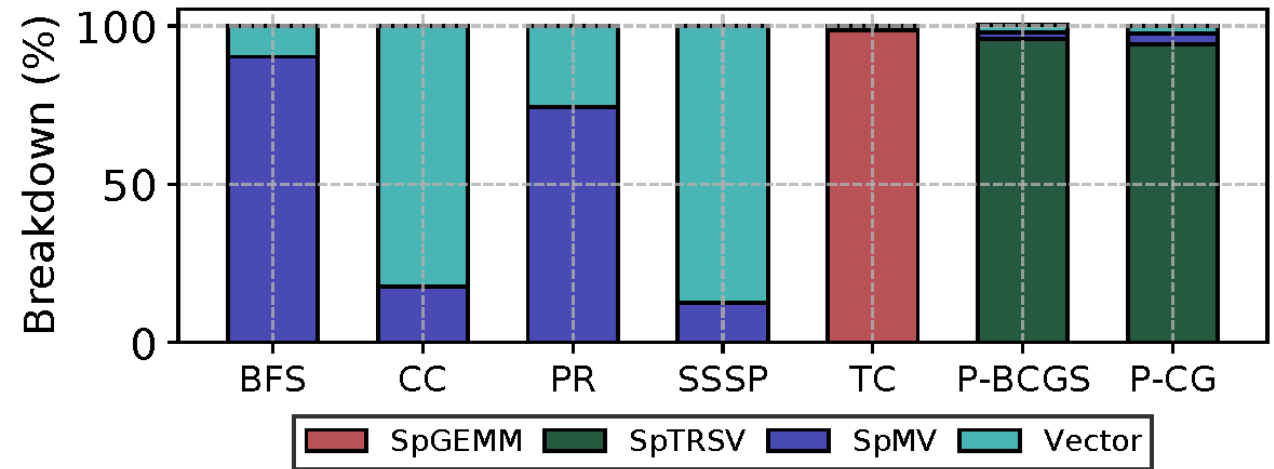


Fluid Dynamics Simulation

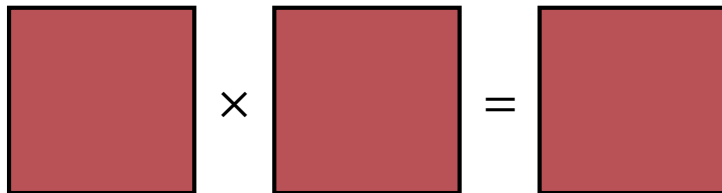
Targeted for < 1% density

Real-World Workload Analysis

- Graph benchmarks
 - Major bottleneck: **SpMV** & **vector ops**
 - Triangle Count: **SpGEMM**
- Linear system solve algorithms
 - Major bottleneck: **SpTRSV**

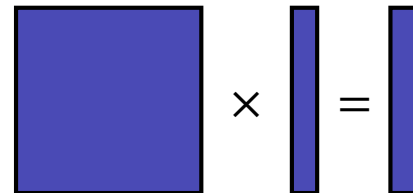


SpGEMM: $A \times B = C$

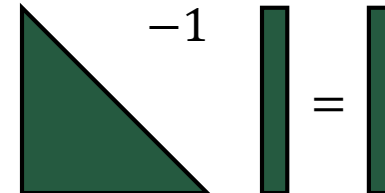


Compute >> Memory

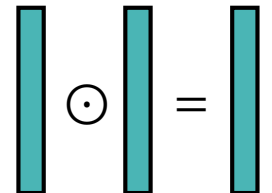
SpMV: $A \times b = c$



SpTRSV: $L^{-1}b = c$



Vector: $a \odot b = c$

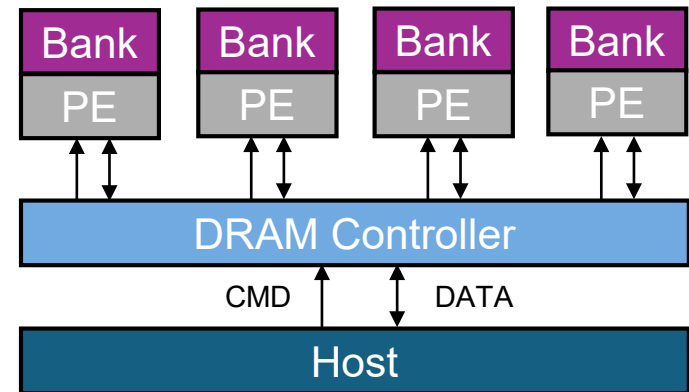
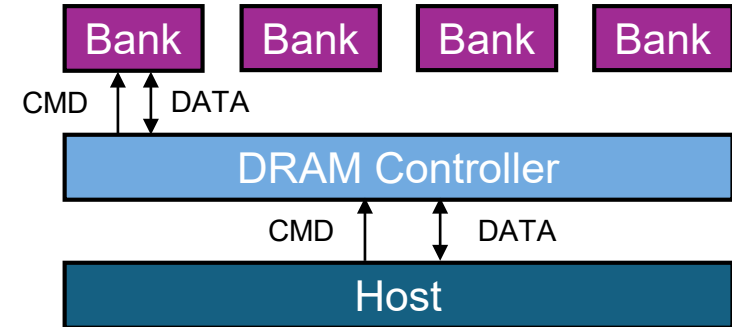


Compute \approx Memory

Commercial Processing-in-Memory

- DRAM
 - Multiple banks grouped in a channel
 - Controller commands a bank to Read/Write

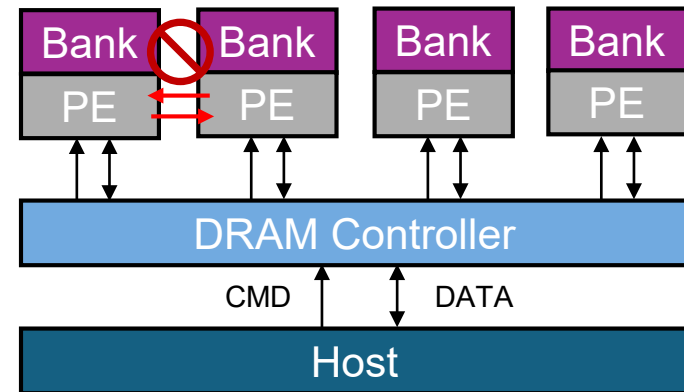
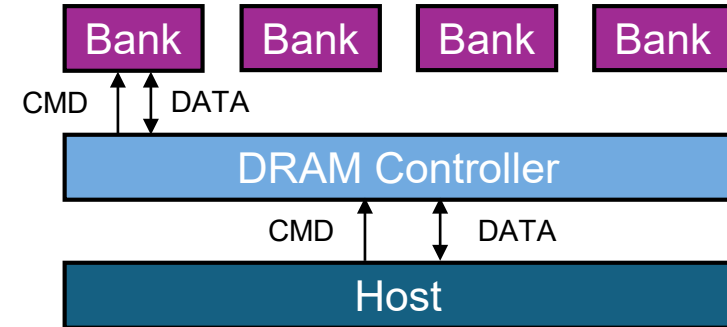
- All-bank Processing-in-Memory
 - Each PE reads/writes data **simultaneously**
 - Focused on **dense GEMV acceleration**
 - Each PE **cannot directly comm.** with each other



Commercial Processing-in-Memory


- DRAM
 - Multiple banks grouped in a channel
 - Controller commands a bank to Read/Write

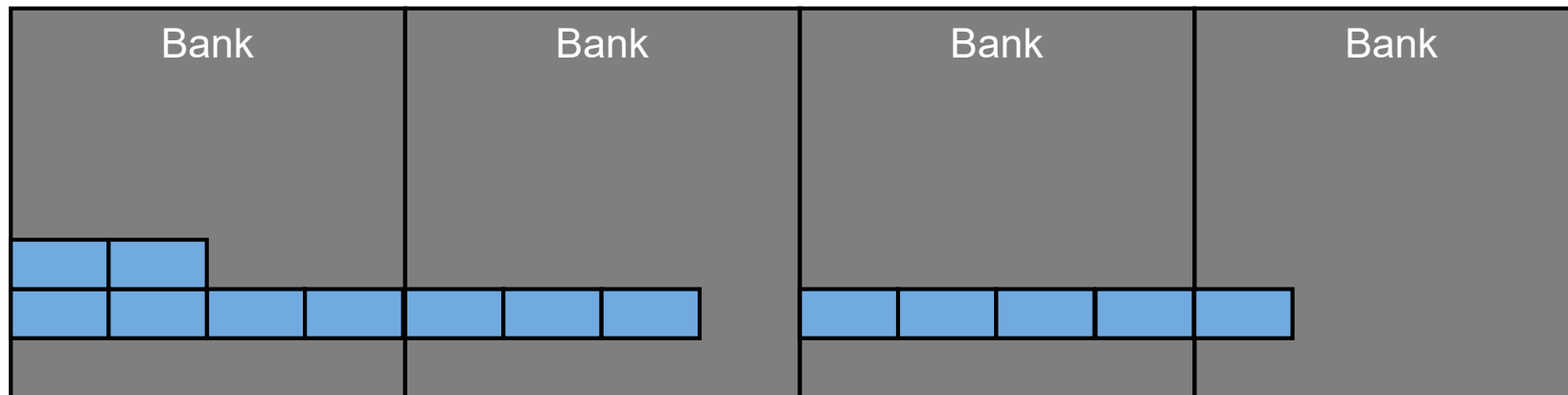
- All-bank Processing-in-Memory
 - Each PE reads/writes data **simultaneously**
 - Focused on **dense GEMV acceleration**
 - Each PE **cannot directly comm.** with each other



Challenges on PIM Implementation

- Industrial all-bank PIM
 - Dense matrix: even distribution
 - Sparse matrix: **uneven distribution**
 - Empty element problem
- Previous academic works
 - SpaceA¹, Gearbox²
 - The memory chip **generates mem. cmds**
 - Hybrid memory cube (HMC) based


Non-zero
element



1. X. Xie, Z. Liang, P. Gu, A. Basak, L. Deng, L. Liang, X. Hu, and Y. Xie, "SpaceA: Sparse Matrix Vector Multiplication on Processing-in-Memory Accelerator," in 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021, pp. 570–583.
2. M. Lenjani, A. Ahmed, M. Stan, and K. Skadron, "Gearbox: A Case for Supporting Accumulation Dispatching and Hybrid Partitioning in PIMBased Accelerators," in Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA'22), 2022, p. 218–230.


Challenges on PIM Implementation

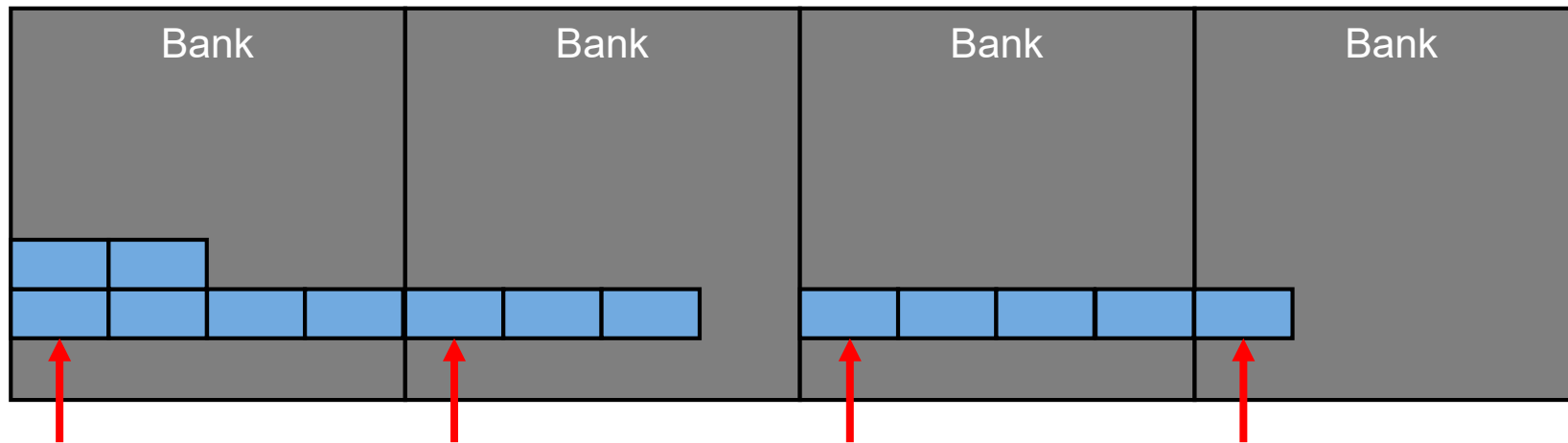
- Industrial all-bank PIM

- Dense matrix: even distribution
- Sparse matrix: **uneven distribution**
- Empty element problem

- Previous academic works

- SpaceA¹, Gearbox²
- The memory chip **generates mem. cmds**
- Hybrid memory cube (HMC) based


Non-zero
element



1. X. Xie, Z. Liang, P. Gu, A. Basak, L. Deng, L. Liang, X. Hu, and Y. Xie, "SpaceA: Sparse Matrix Vector Multiplication on Processing-in-Memory Accelerator," in 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021, pp. 570–583.
2. M. Lenjani, A. Ahmed, M. Stan, and K. Skadron, "Gearbox: A Case for Supporting Accumulation Dispatching and Hybrid Partitioning in PIMBased Accelerators," in Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA'22), 2022, p. 218–230.


Challenges on PIM Implementation

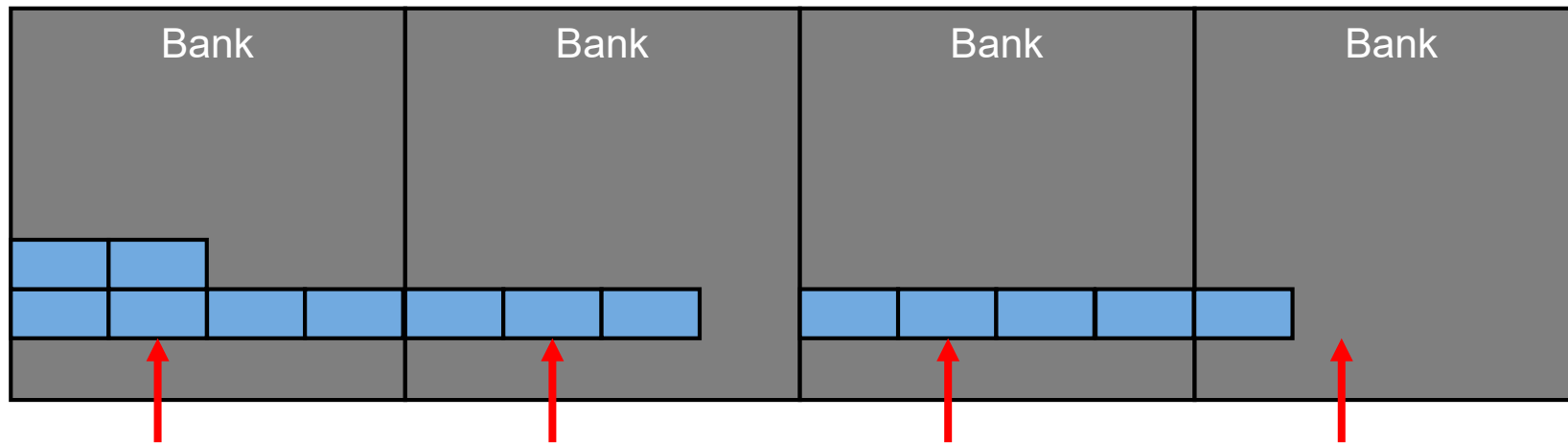
- Industrial all-bank PIM

- Dense matrix: even distribution
- Sparse matrix: **uneven distribution**
- Empty element problem

- Previous academic works

- SpaceA¹, Gearbox²
- The memory chip **generates mem. cmds**
- Hybrid memory cube (HMC) based


Non-zero
element



1. X. Xie, Z. Liang, P. Gu, A. Basak, L. Deng, L. Liang, X. Hu, and Y. Xie, "SpaceA: Sparse Matrix Vector Multiplication on Processing-in-Memory Accelerator," in 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021, pp. 570–583.
2. M. Lenjani, A. Ahmed, M. Stan, and K. Skadron, "Gearbox: A Case for Supporting Accumulation Dispatching and Hybrid Partitioning in PIMBased Accelerators," in Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA'22), 2022, p. 218–230.


Challenges on PIM Implementation

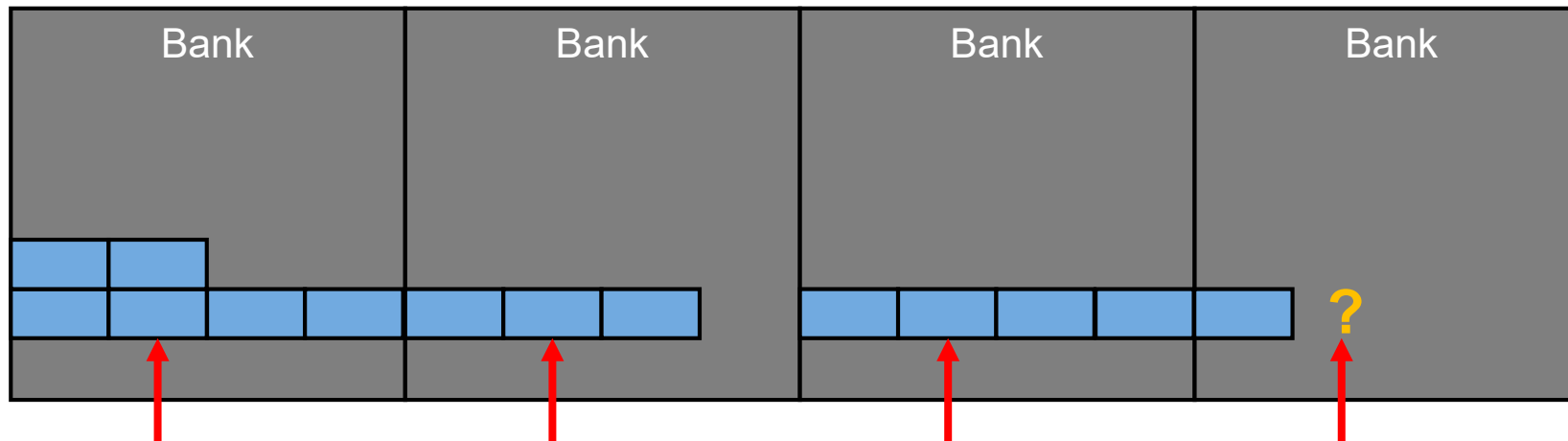
- Industrial all-bank PIM

- Dense matrix: even distribution
- Sparse matrix: **uneven distribution**
- Empty element problem

- Previous academic works

- SpaceA¹, Gearbox²
- The memory chip **generates mem. cmds**
- Hybrid memory cube (HMC) based


Non-zero
element



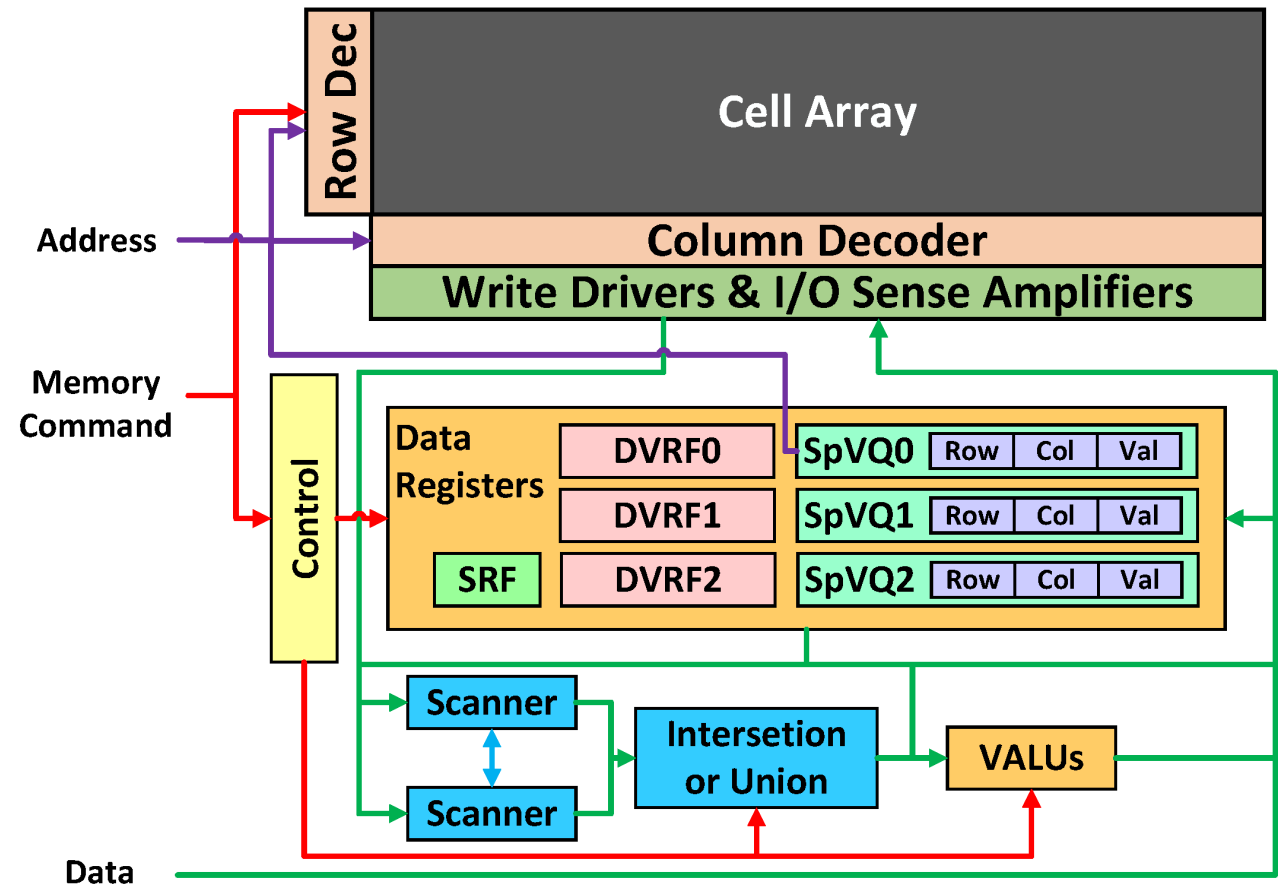
1. X. Xie, Z. Liang, P. Gu, A. Basak, L. Deng, L. Liang, X. Hu, and Y. Xie, "SpaceA: Sparse Matrix Vector Multiplication on Processing-in-Memory Accelerator," in 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021, pp. 570–583.
2. M. Lenjani, A. Ahmed, M. Stan, and K. Skadron, "Gearbox: A Case for Supporting Accumulation Dispatching and Hybrid Partitioning in PIMBased Accelerators," in Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA'22), 2022, p. 218–230.

Goals

- Proposes **partially synchronous execution** architecture
 - Maintain all-bank execution manner for host – PIM communication
 - **Conditional execution** for non-even workloads with SIMD instructions
- Efficient distribution algorithm for **SpMV memory mapping**
 - Reduces external memory I/O for remote bank access
- Suggest first **SpTRSV kernel acceleration**

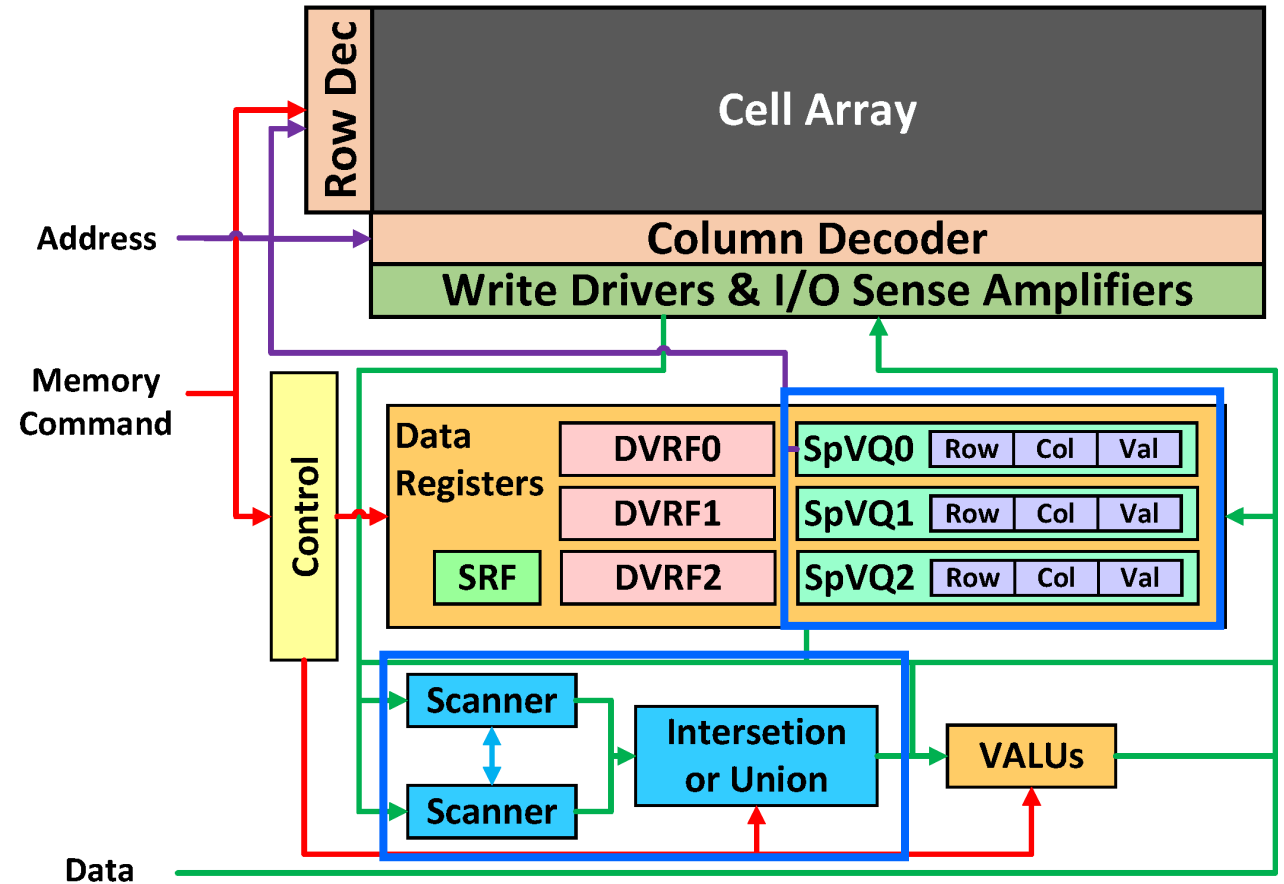
pSyncPIM Architecture

- Sparsity-aware PE architecture
 - Intersection/union **vector scanners**¹
 - Sparse vector **queues**
- **Flexible ISA** for various algebras
 - Supporting graph apps
 - ISA details are in the paper
- ALU performance
 - Full support for 32B/cycle BW
 - INT: 8 to 64 bits, FP: 16 to 64 bits



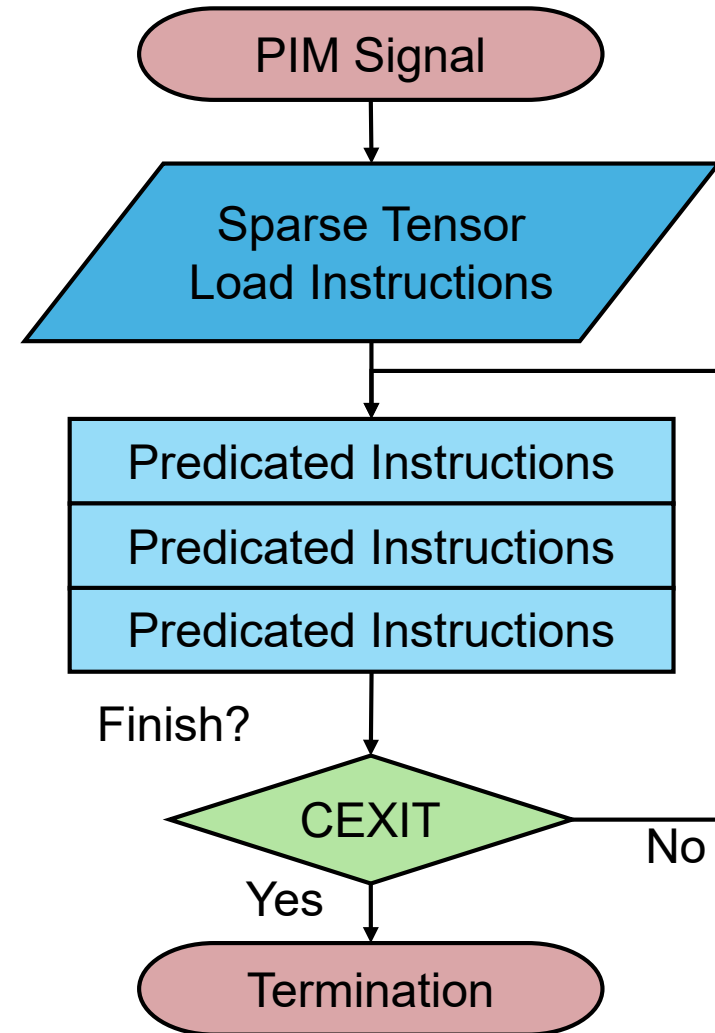
pSyncPIM Architecture

- Sparsity-aware PE architecture
 - Intersection/union **vector scanners**¹
 - Sparse vector **queues**
- **Flexible ISA** for various algebras
 - Supporting graph apps
 - ISA details are in the paper
- ALU performance
 - Full support for 32B/cycle BW
 - INT: 8 to 64 bits, FP: 16 to 64 bits



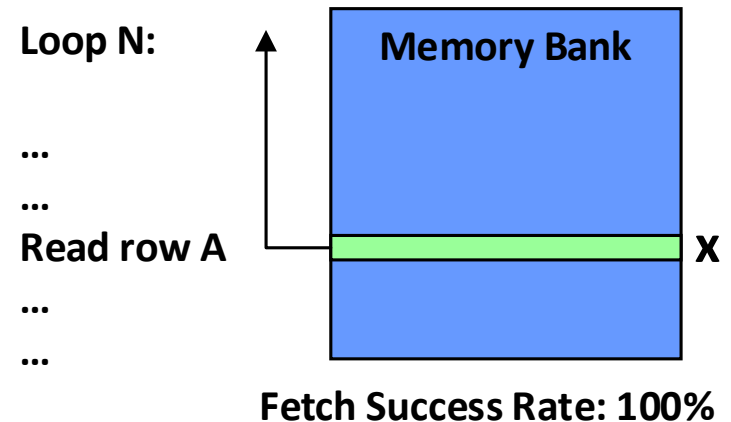
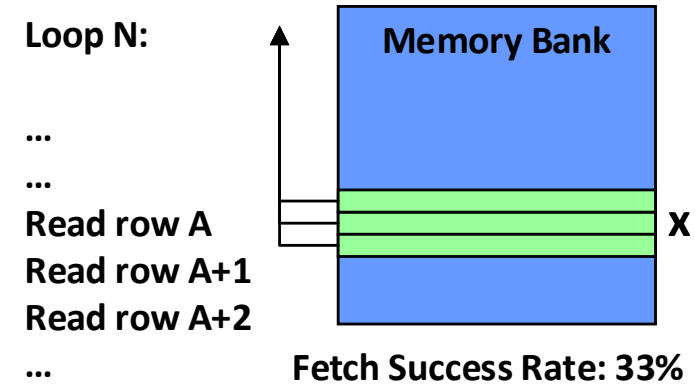
Conditional Execution & Exit

- **Predicated** execution of instructions
 - **Success/fails** depending on PE's status
 - **No branching** on success/failure
- Execute kernel as an **infinite loop**
 - CEXIT for **conditional exit** for each PE
- Synchronizing timings on exited PEs
 - **Normally activates and precharges** rows
 - **No data transfers** between bank & PE



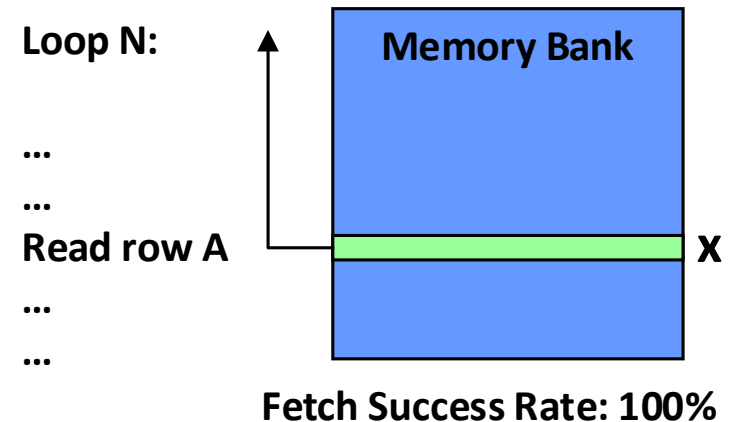
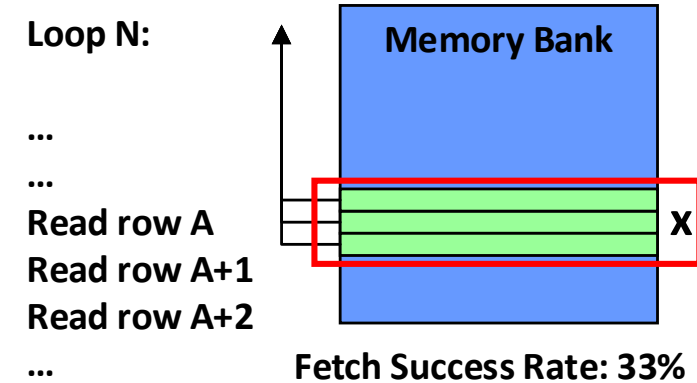
PIM Memory Mapping Limitations

- Data are mapped into each bank
- All banks should R/W same row
- I/O vector mapping problem:
 - PIM should R/W all mapped rows
 - Fetch/writeback success rate drops
- Bank configuration: 1KB row size
 - **Each bank stores at most 1KB vectors**
 - **Matrix size: 1KB limit** for row & col dimension



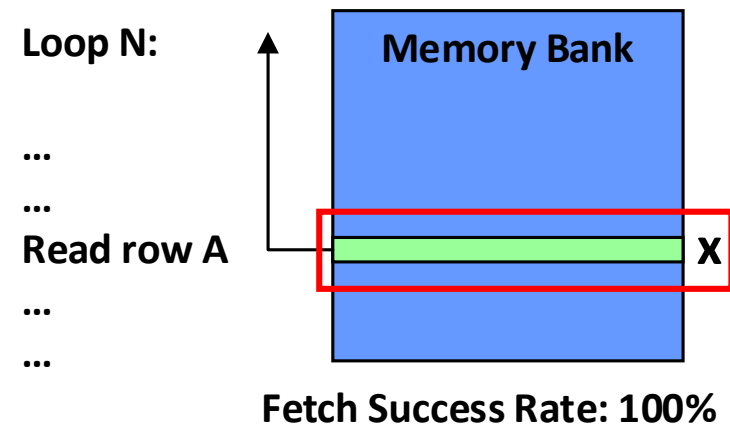
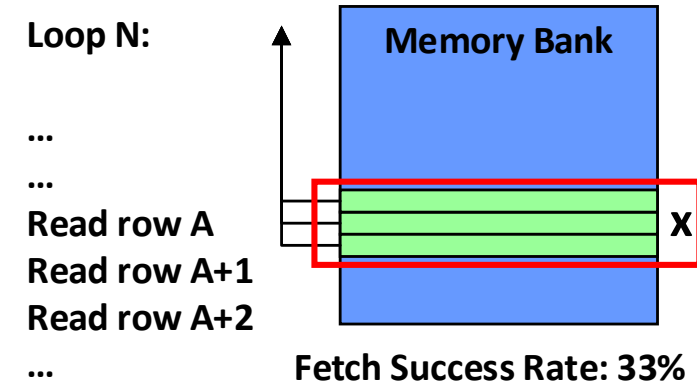
PIM Memory Mapping Limitations

- Data are mapped into each bank
- All banks should **R/W same row**
- I/O vector mapping problem:
 - PIM should **R/W all mapped rows**
 - Fetch/writeback **success rate drops**
- Bank configuration: 1KB row size
 - **Each bank stores at most 1KB vectors**
 - **Matrix size: 1KB limit** for row & col dimension



PIM Memory Mapping Limitations

- Data are mapped into each bank
- All banks should R/W same row
- I/O vector mapping problem:
 - PIM should R/W all mapped rows
 - Fetch/writeback success rate drops
- Bank configuration: 1KB row size
 - **Each bank stores at most 1KB vectors**
 - **Matrix size: 1KB limit for row & col dimension**

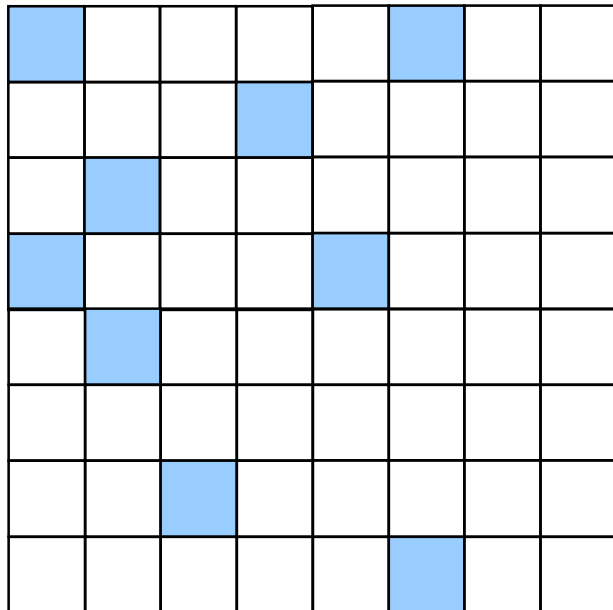


Naïve SpMV Distribution

Non-zero element



Input Matrix

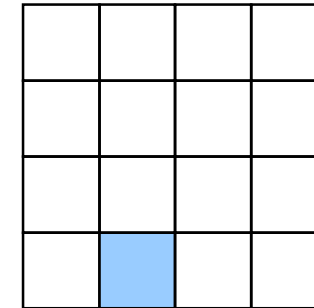
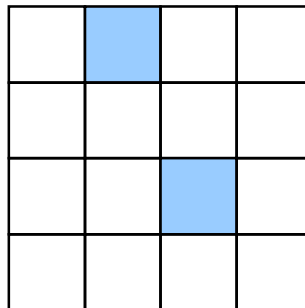
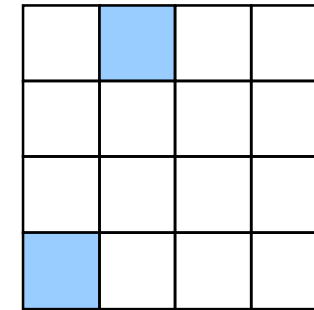
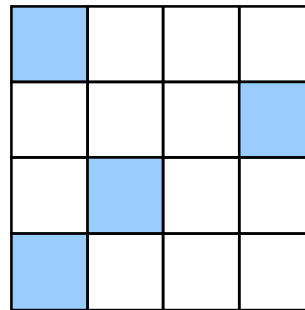


Naïve SpMV Distribution


Non-zero element

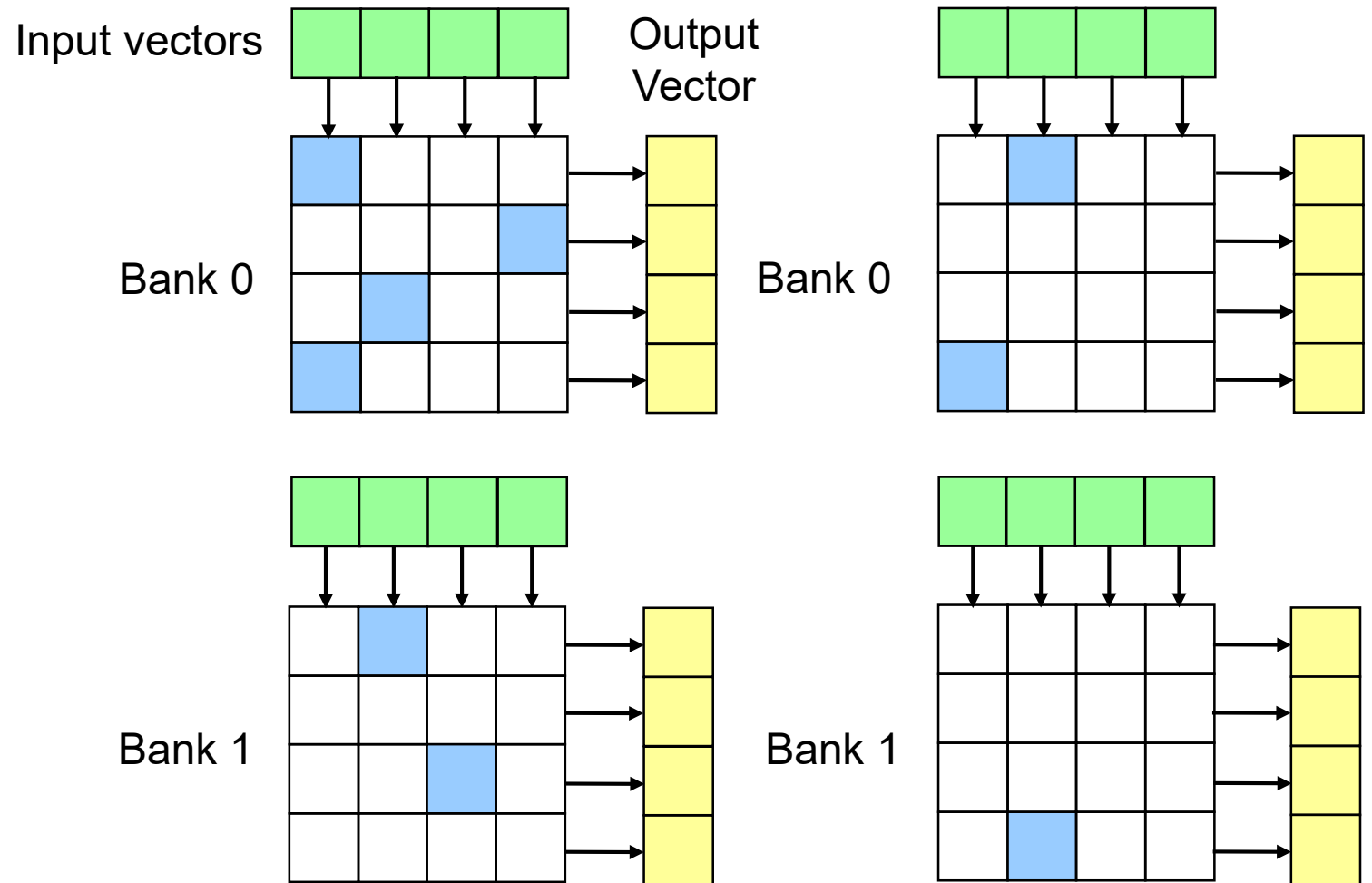


Input Matrix

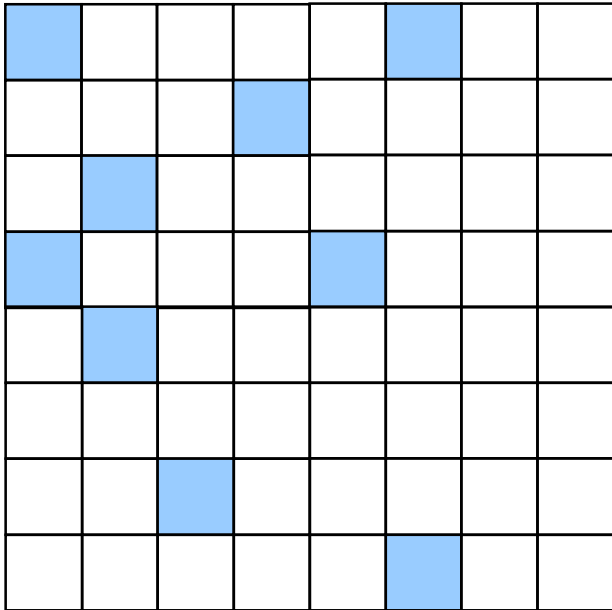


Naïve SpMV Distribution

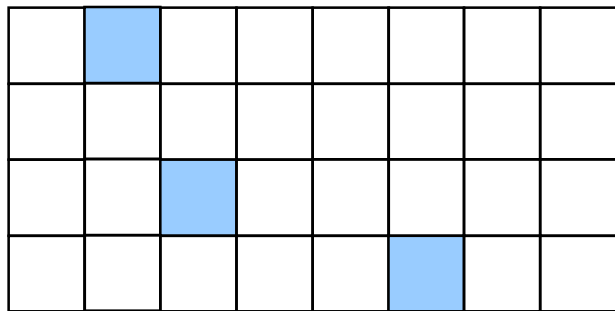
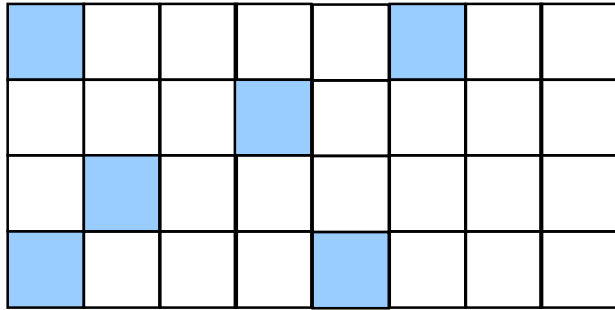
Non-zero element

Input Matrix



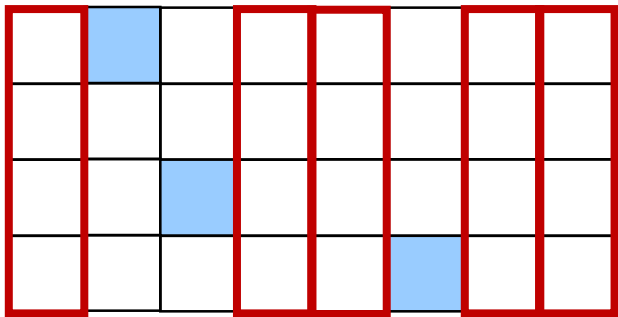
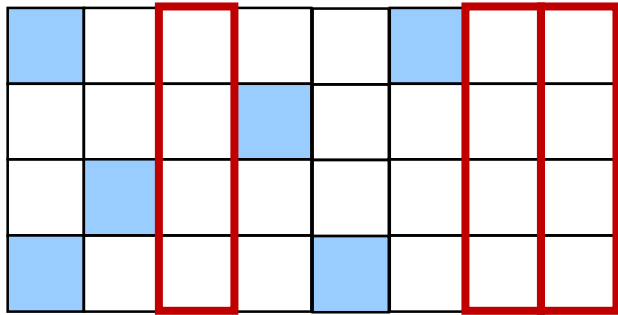
SpMV-aware Compression



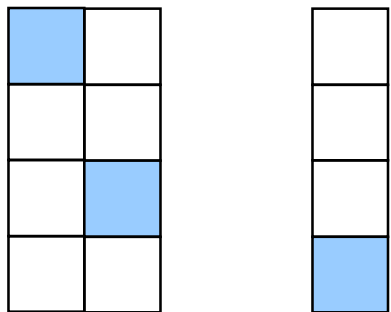
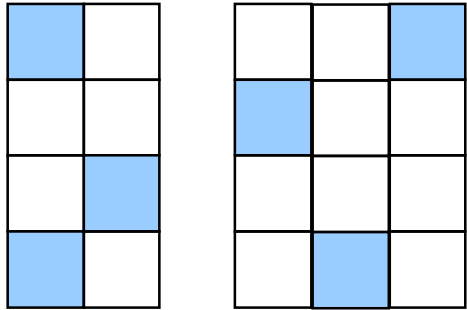
SpMV-aware Compression



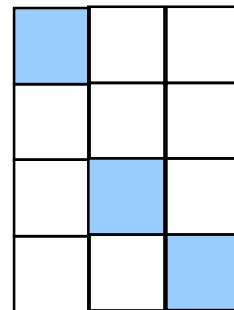
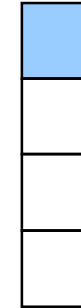
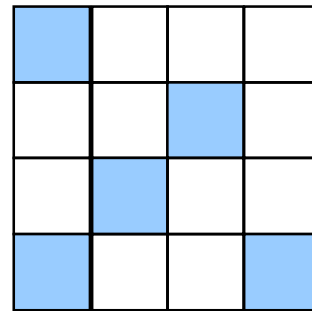
SpMV-aware Compression



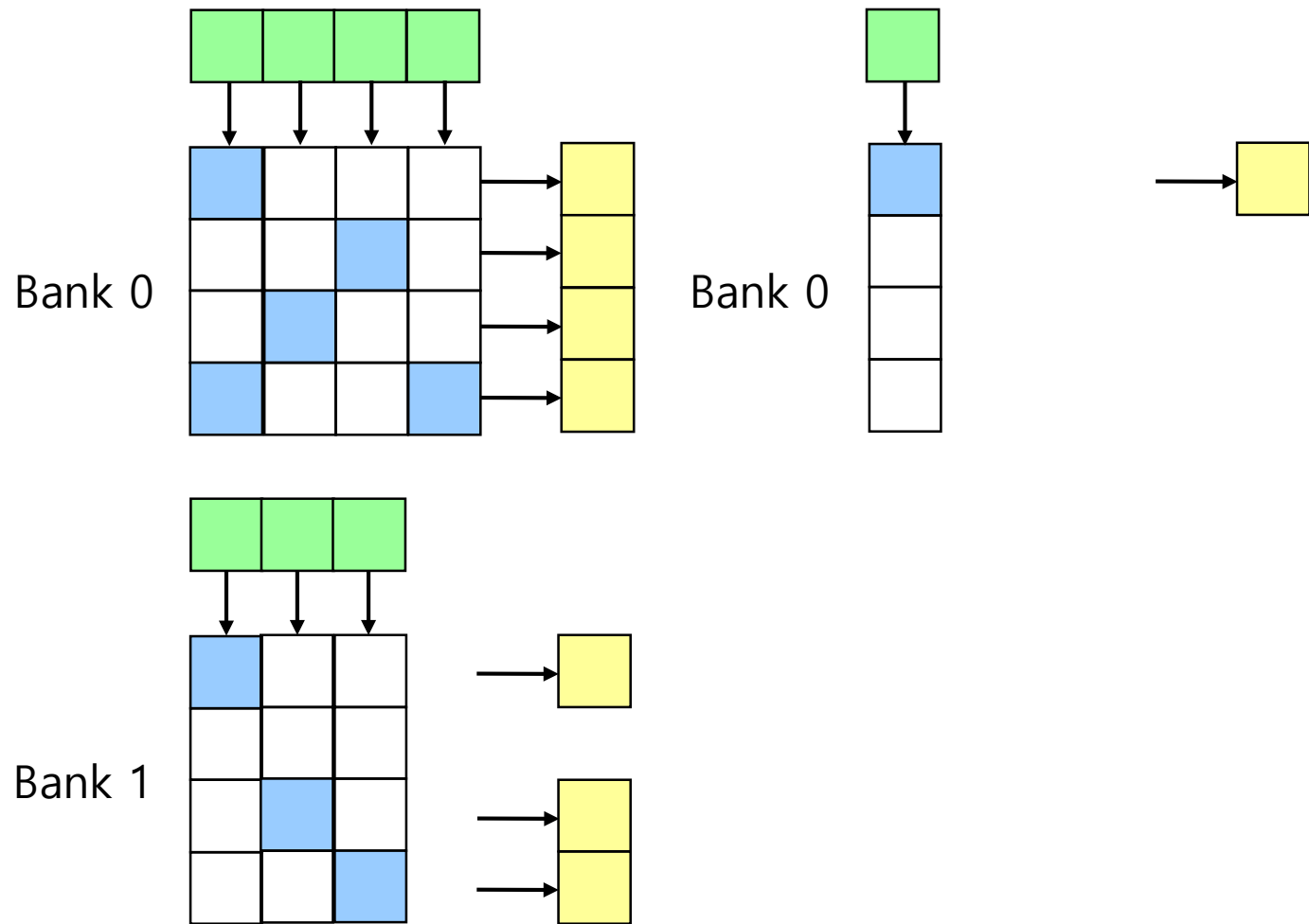
SpMV-aware Compression



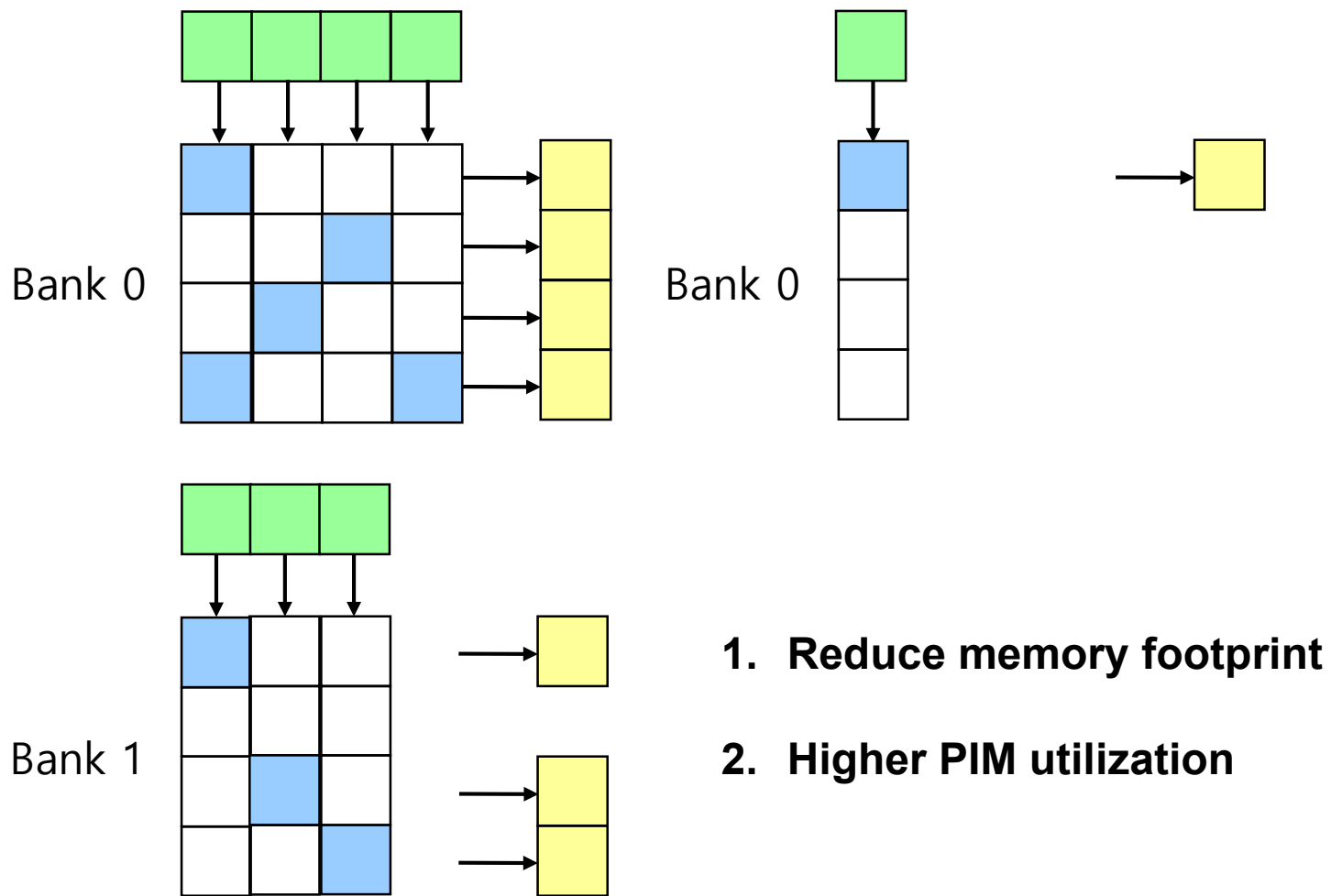
SpMV-aware Compression



SpMV-aware Compression

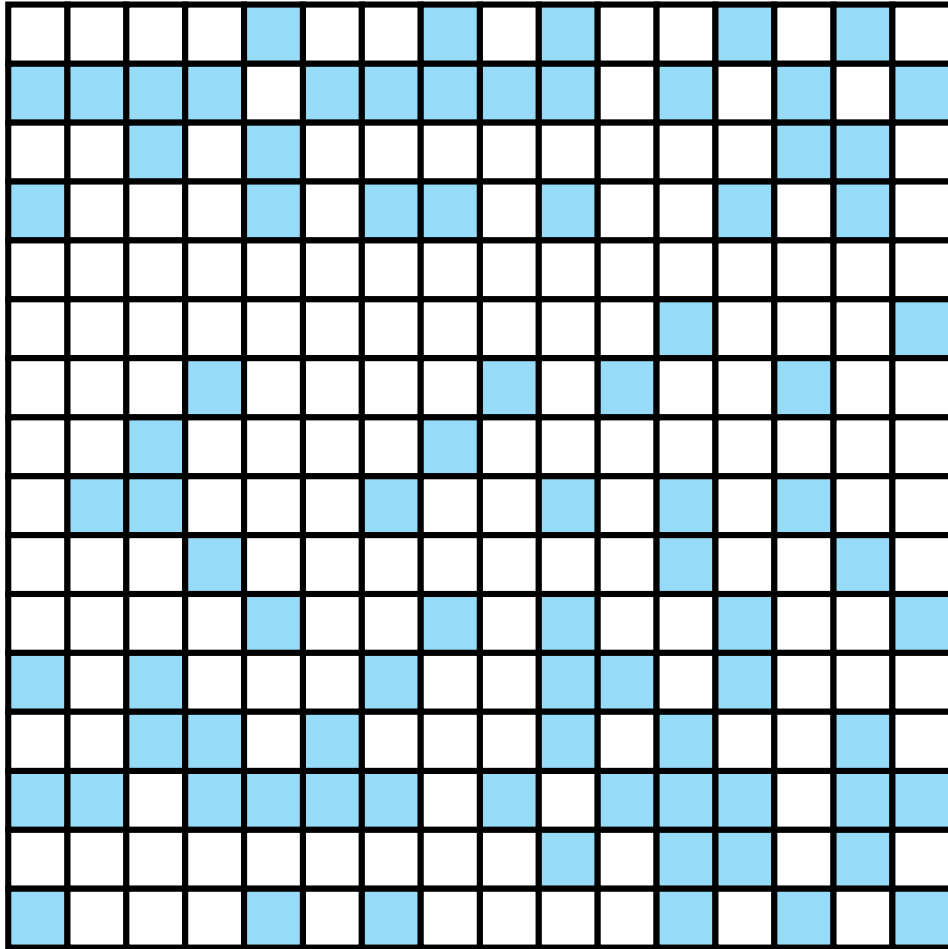


SpMV-aware Compression



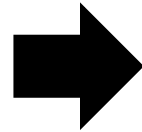
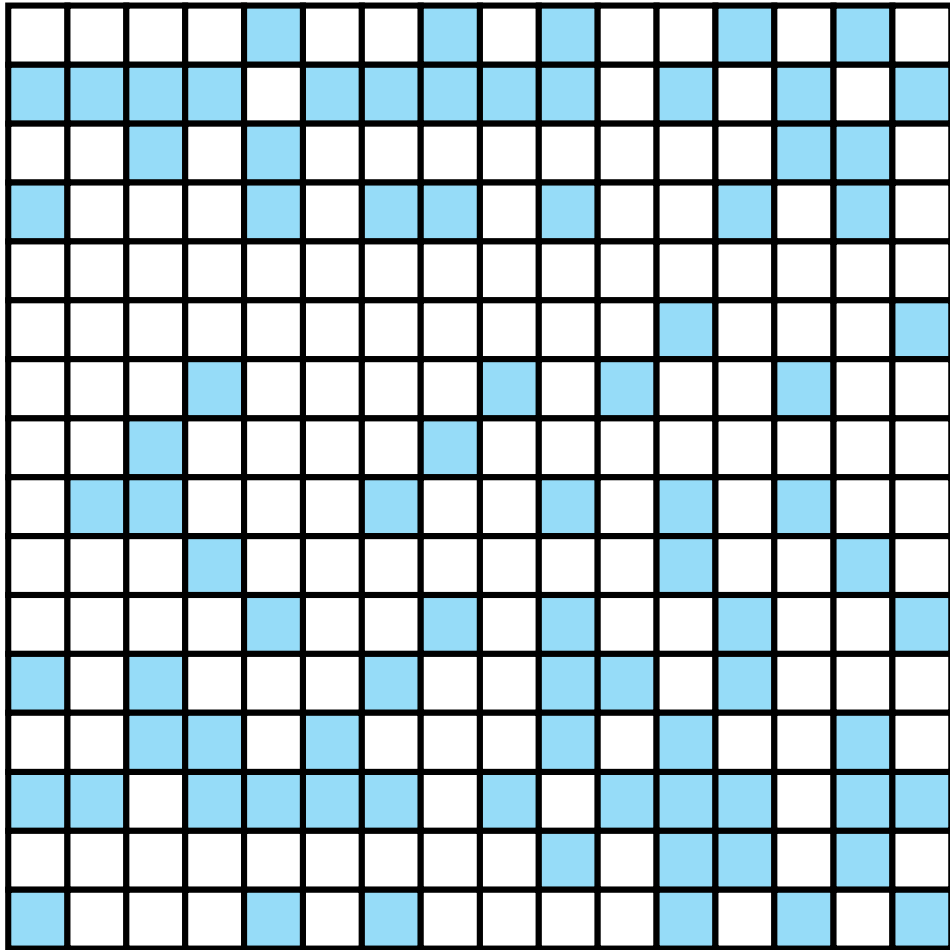
How To Load-Balance?

Original Sparse Matrix

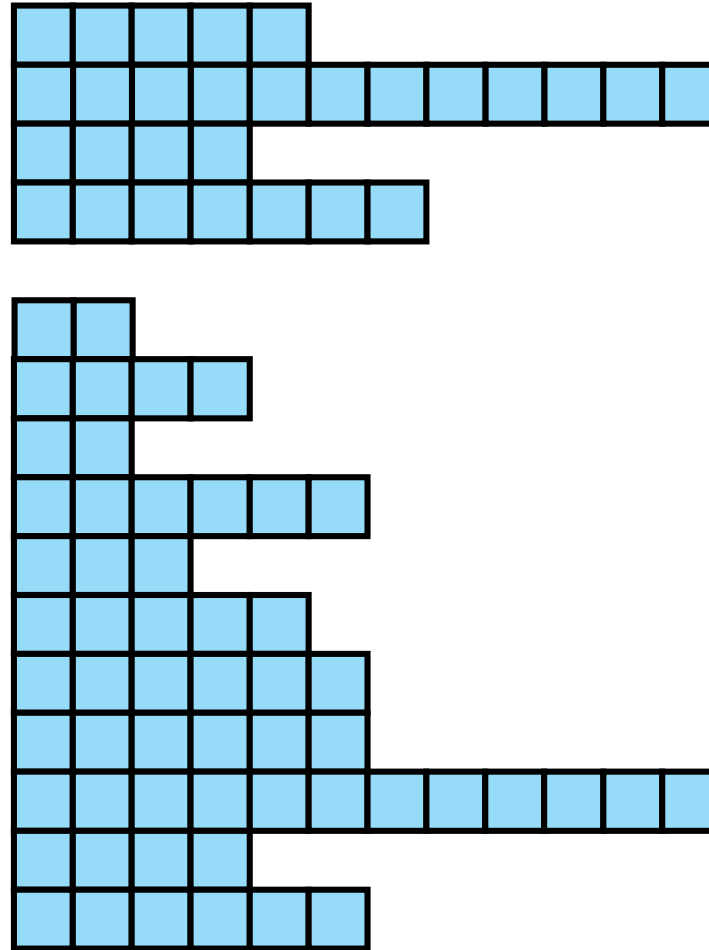


How To Load-Balance?

Original Sparse Matrix

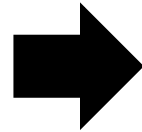
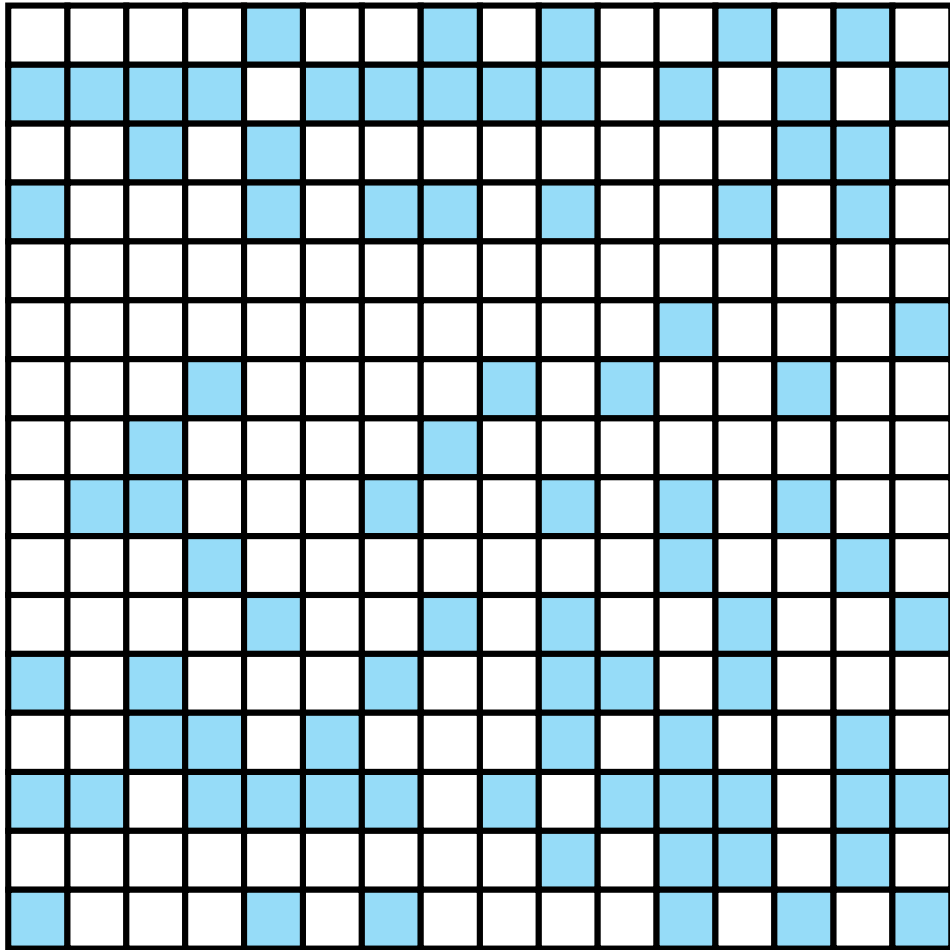


Compressed Sparse Matrix

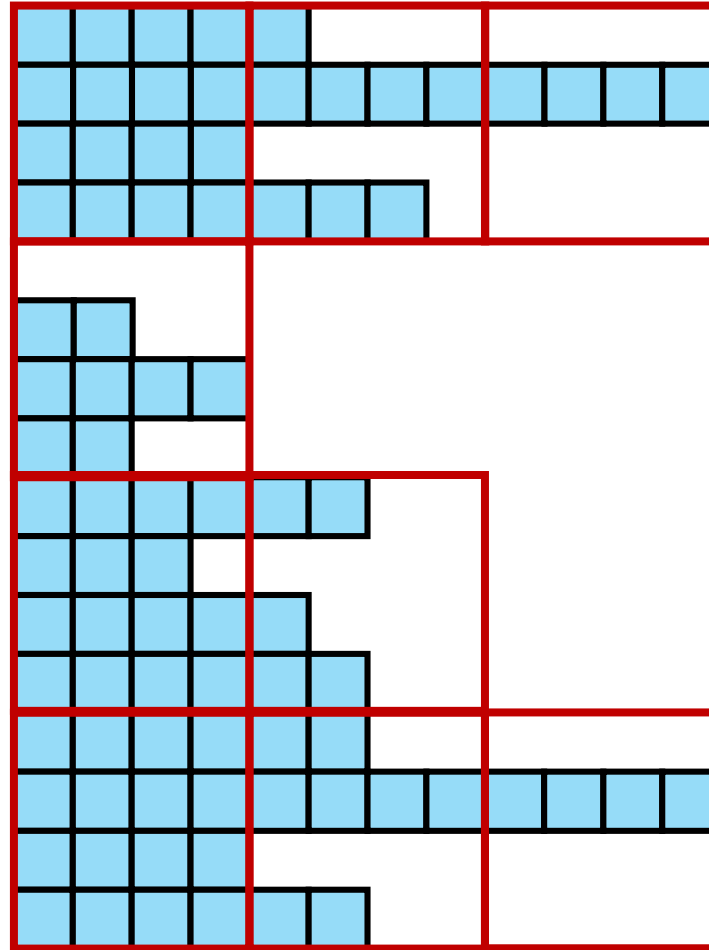


How To Load-Balance?

Original Sparse Matrix

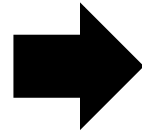
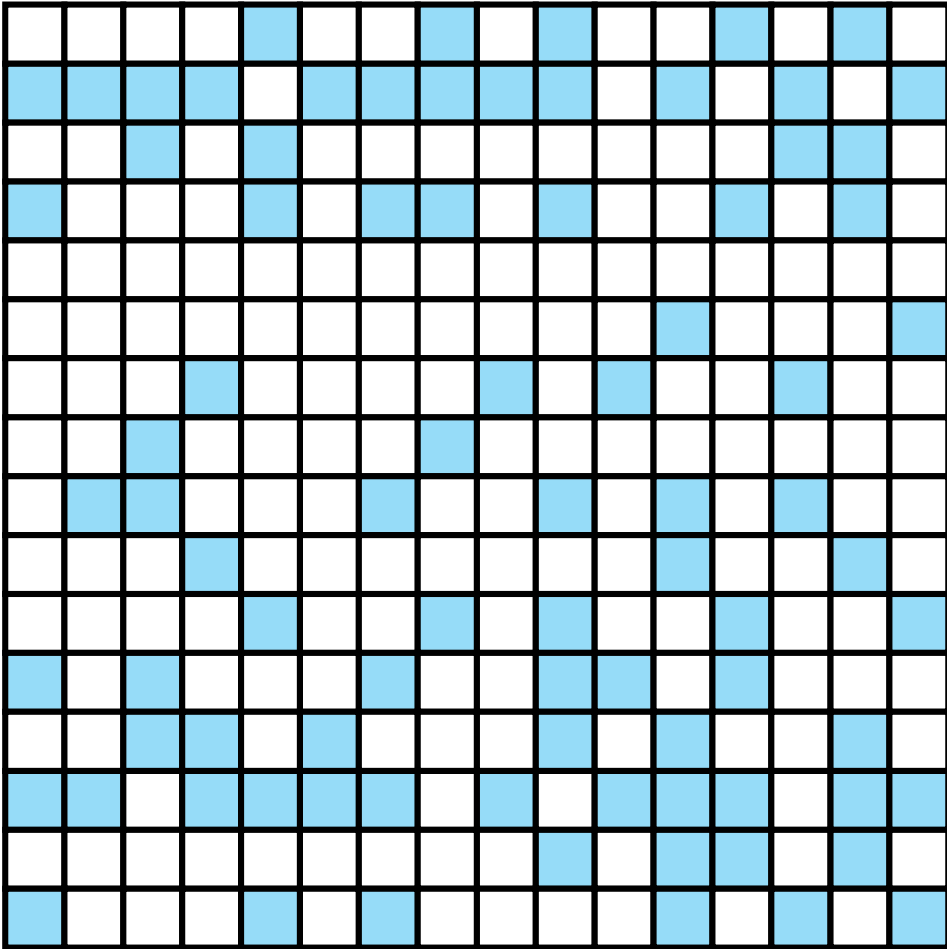


Compressed Sparse Matrix

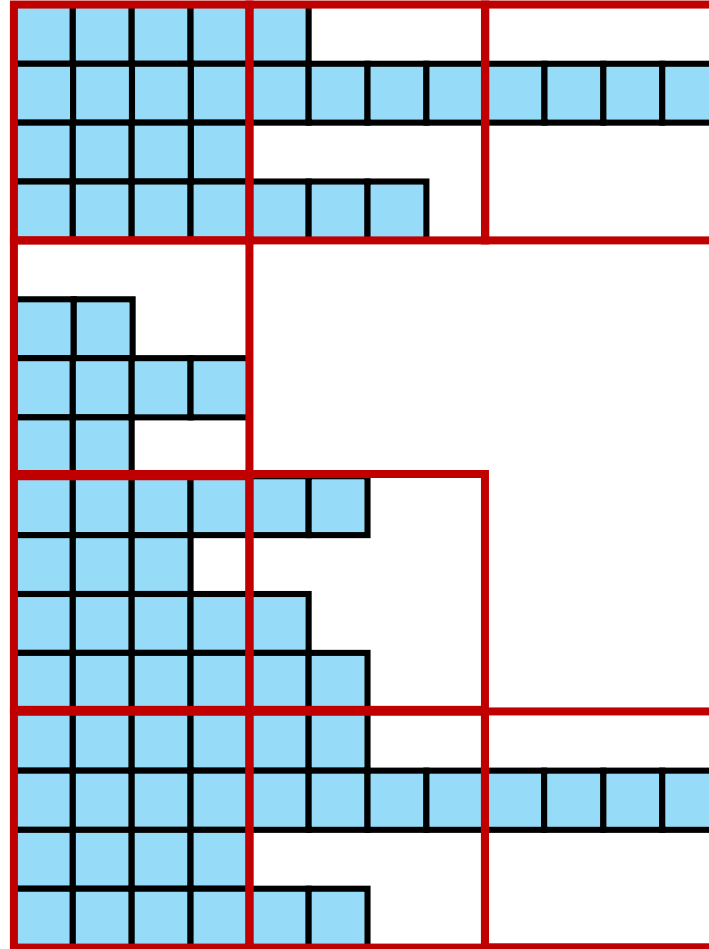


How To Load-Balance?

Original Sparse Matrix



Compressed Sparse Matrix

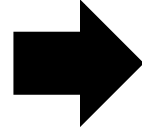
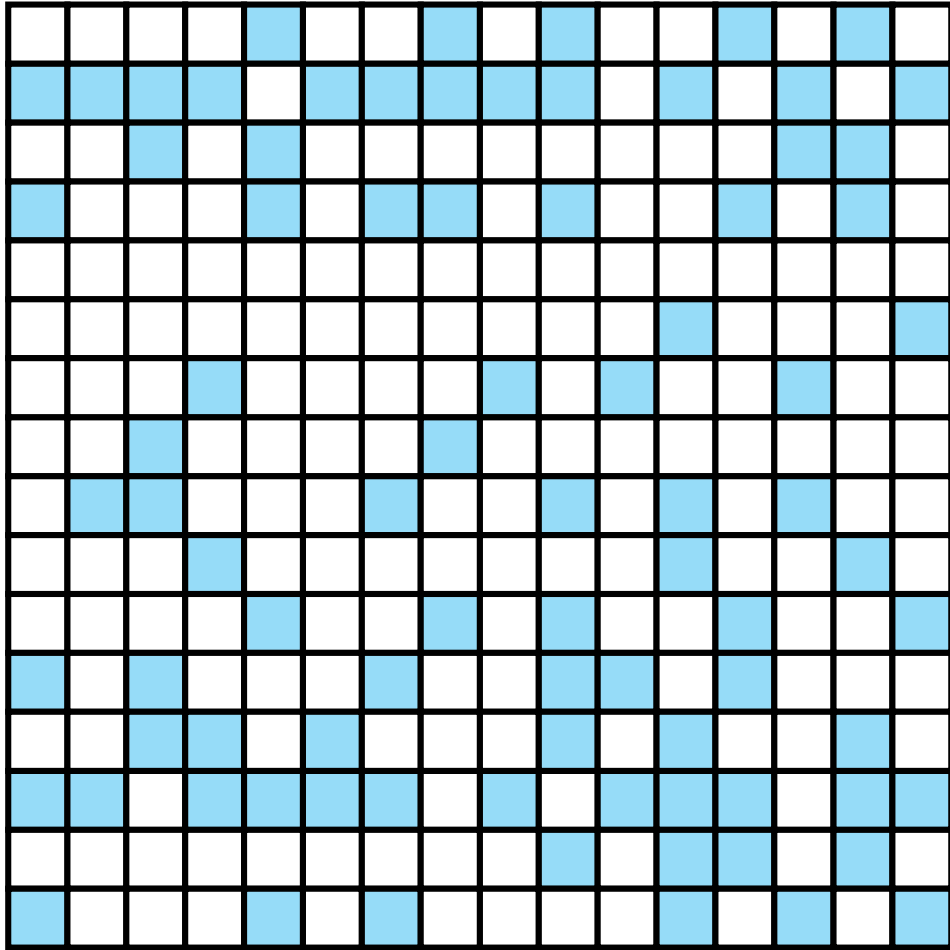


of nnzs

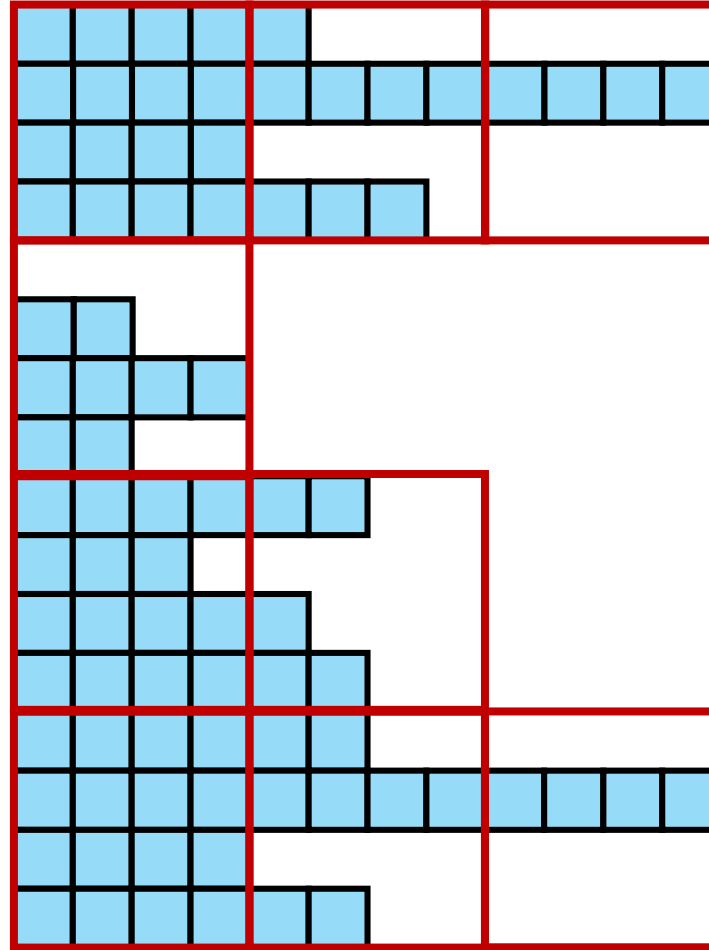
16	8	4
8		
15	5	
16	8	4

How To Load-Balance?

Original Sparse Matrix



Compressed Sparse Matrix



of nnzs

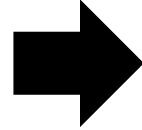
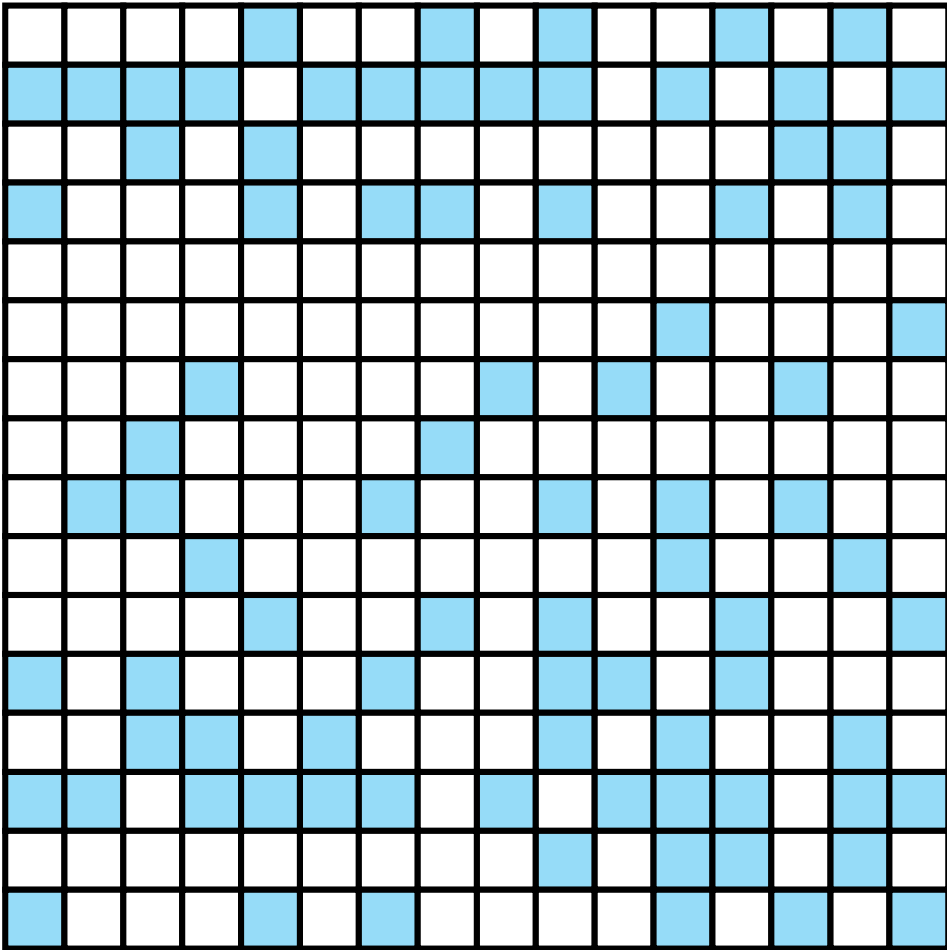
16 8 4
8
15 5
16 8 4

Bank Mapping

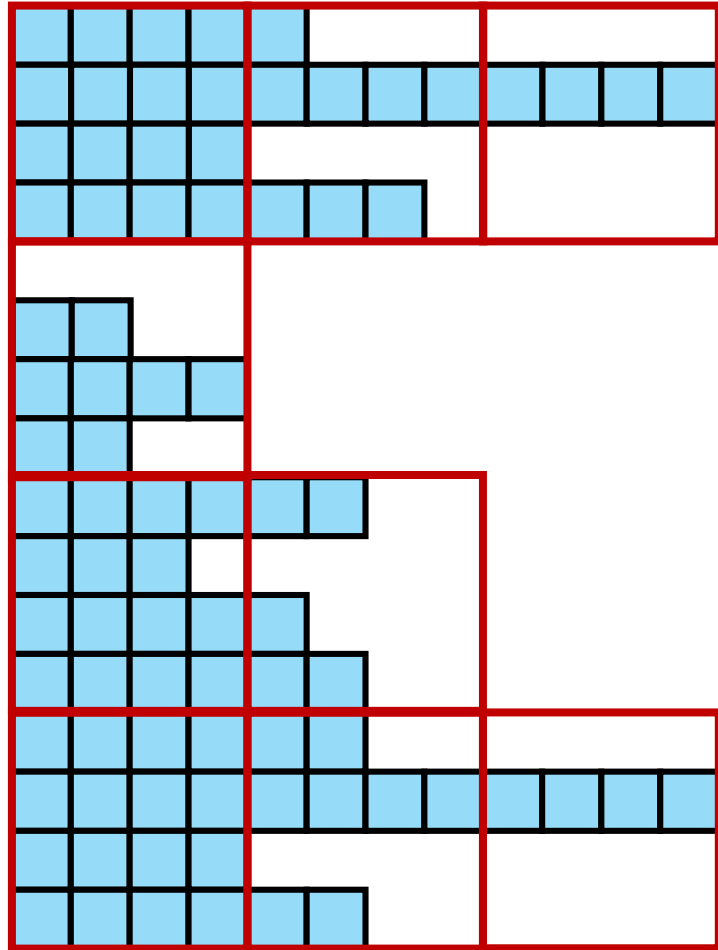
Bank	0	1	2	3
Batch 1	16	8	15	16

How To Load-Balance?

Original Sparse Matrix



Compressed Sparse Matrix



of nnzs

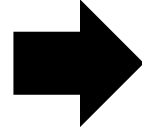
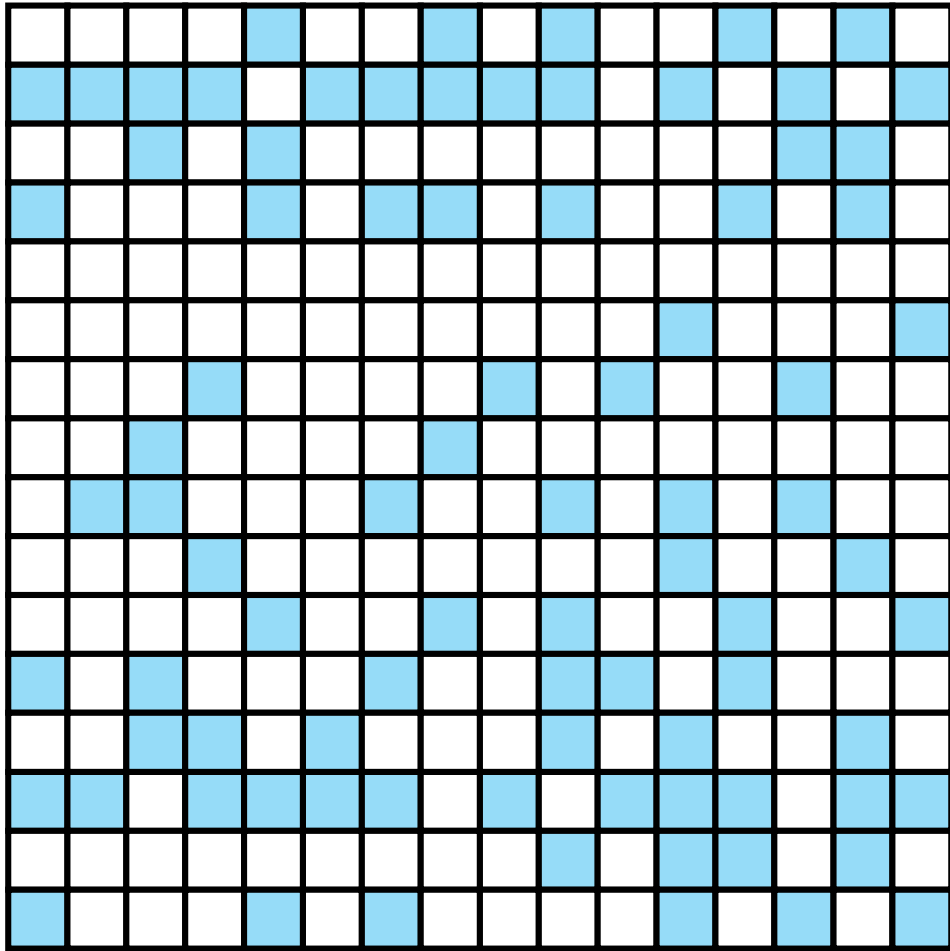
16 8 4
8
15 5
16 8 4

Bank Mapping

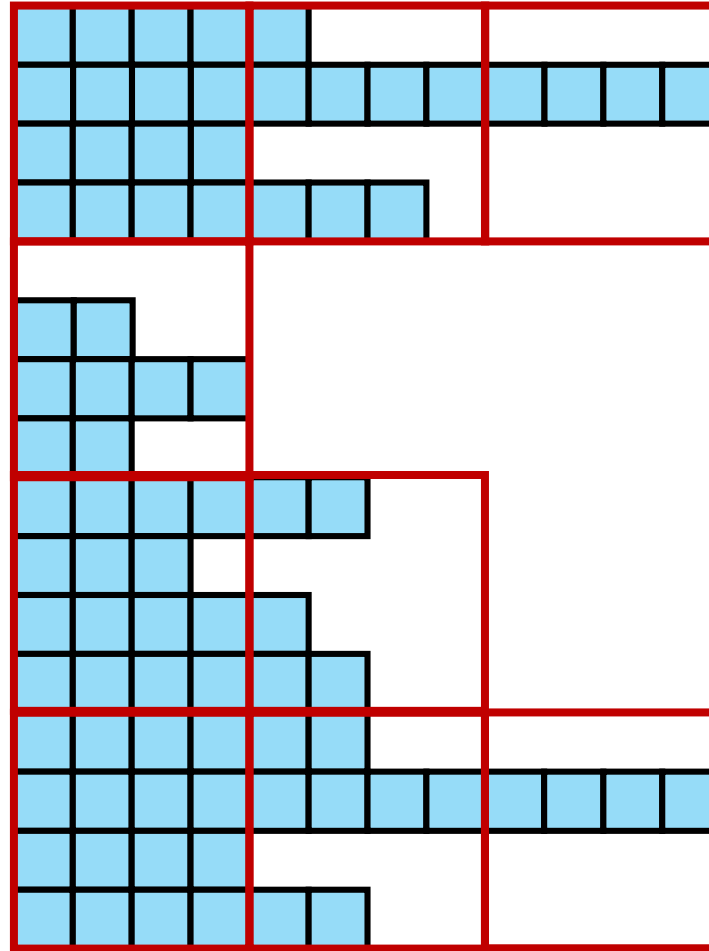
Bank	0	1	2	3
Batch 1	16	8	15	16
Batch 2	8	5	8	4

How To Load-Balance?

Original Sparse Matrix



Compressed Sparse Matrix



of nnzs

16	8	4
8		
15	5	
16	8	4

Bank Mapping

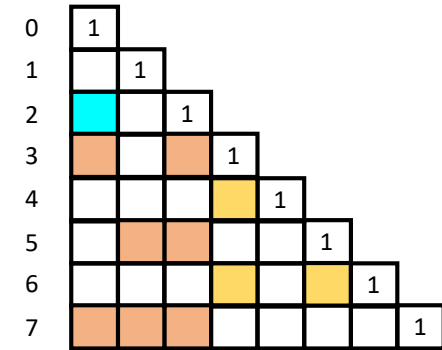
Bank	0	1	2	3
Batch 1	16	8	15	16
Batch 2	8	5	8	4
Batch 3	4			

Row Dependency Problem

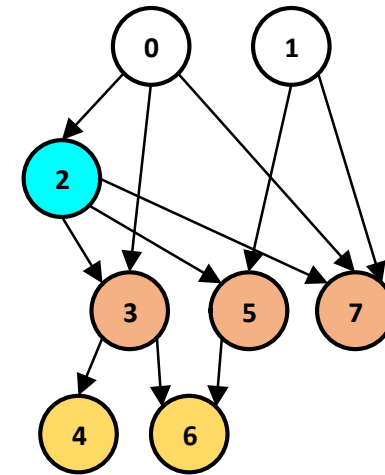
- SpTRSV: Gaussian elimination
 - Row/column-wise vector multiply & subtract
 - **Must compute previous rows for next row**
 - Data dependency between rows & columns

Algorithm 1 SpTRSV algorithm.

```
1:  $M$ :  $n \times n$  lower triangular matrix in COO format
2:  $b$ : input vector
3:  $x$ : output vector
4: for  $i = 0$  to  $n - 1$  do
5:    $s = 0$ 
6:   for all  $e = (i, c_e, v_e) \in M$  where  $c_e < i$  do
7:      $s += v_e \times x[c_e]$ 
8:   end for
9:    $l := (i, i, v_l) \in M$ 
10:   $x[i] = (b[i] - s) / v_l$ 
11: end for
```



Triangular Matrix



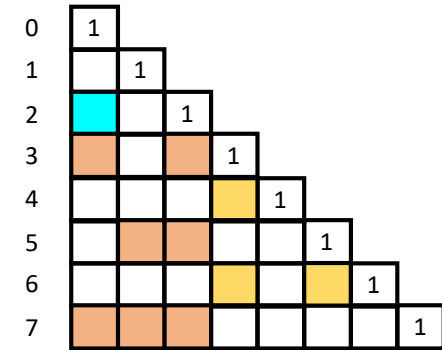
Row Dependency Graph

Row Dependency Problem

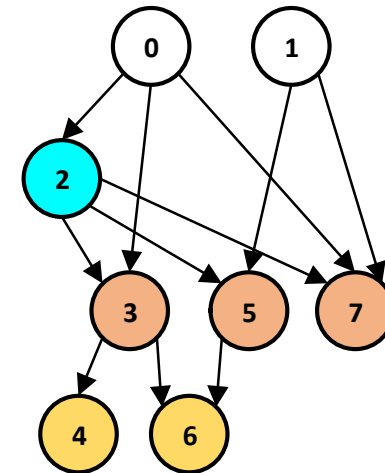
- SpTRSV: Gaussian elimination
 - Row/column-wise vector multiply & subtract
 - **Must compute previous rows for next row**
 - Data dependency between rows & columns

Algorithm 1 SpTRSV algorithm.

```
1:  $M$ :  $n \times n$  lower triangular matrix in COO format
2:  $b$ : input vector
3:  $x$ : output vector
4: for  $i = 0$  to  $n - 1$  do
5:    $s = 0$ 
6:   for all  $e = (i, c_e, v_e) \in M$  where  $c_e < i$  do
7:      $s += v_e \times x[c_e]$ 
8:   end for
9:    $l := (i, i, v_l) \in M$ 
10:   $x[i] = (b[i] - s) / v_l$ 
11: end for
```

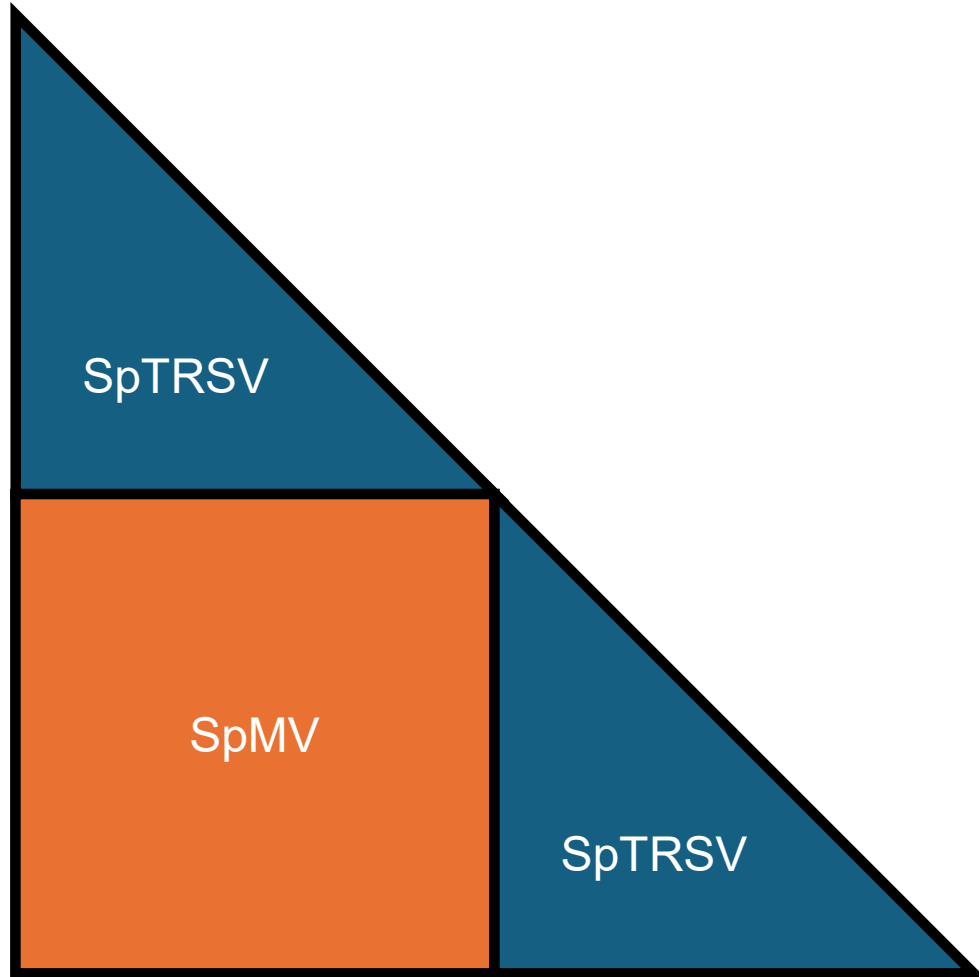


Triangular Matrix

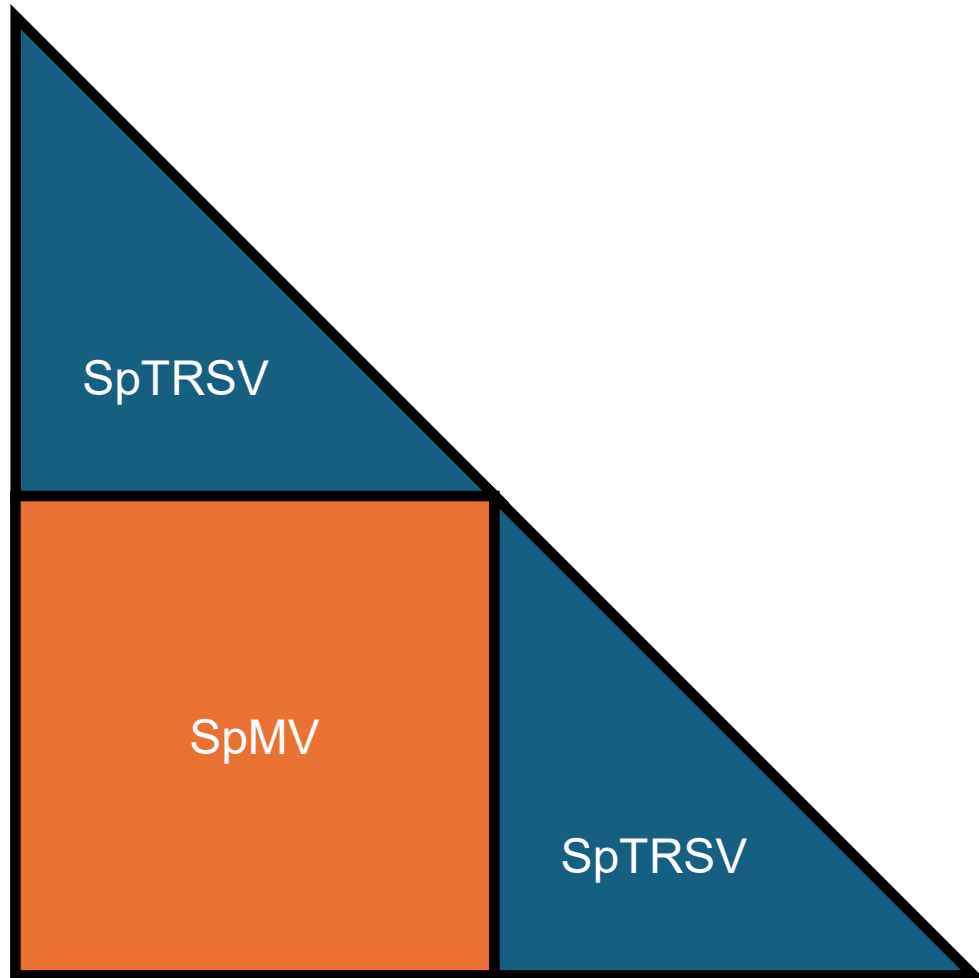


Row Dependency Graph

SpTRSV Block Algorithm (ICPP 2020¹)



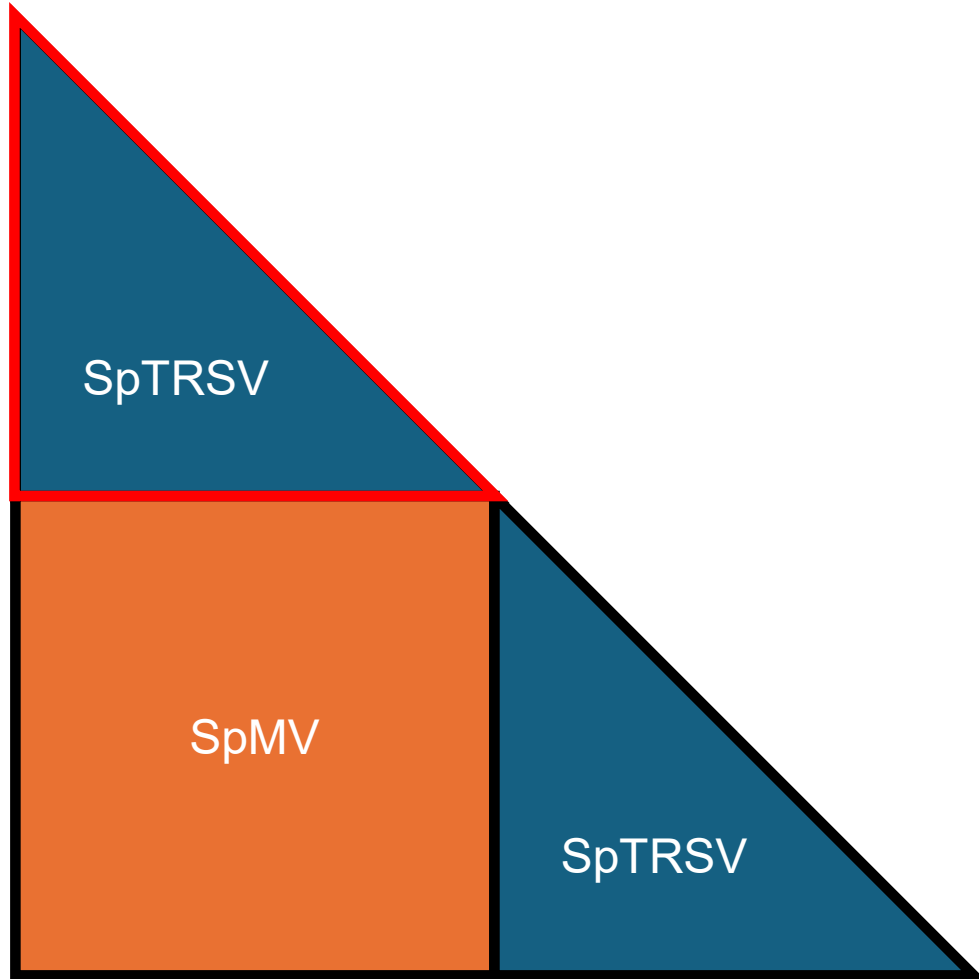
SpTRSV Block Algorithm (ICPP 2020¹)



Recursive SpTRSV Algorithm

1. $\bar{x}_1 = L_1^{-1} \bar{b}_1$ (Recursive SpTRSV)
2. $L_2 \bar{x}_2 = \bar{b}_2 - C \bar{x}_1 = \bar{b}_2'$ (SpMV)
3. $\bar{x}_2 = L_2^{-1} \bar{b}_2'$ (Recursive SpTRSV)

SpTRSV Block Algorithm (ICPP 2020¹)



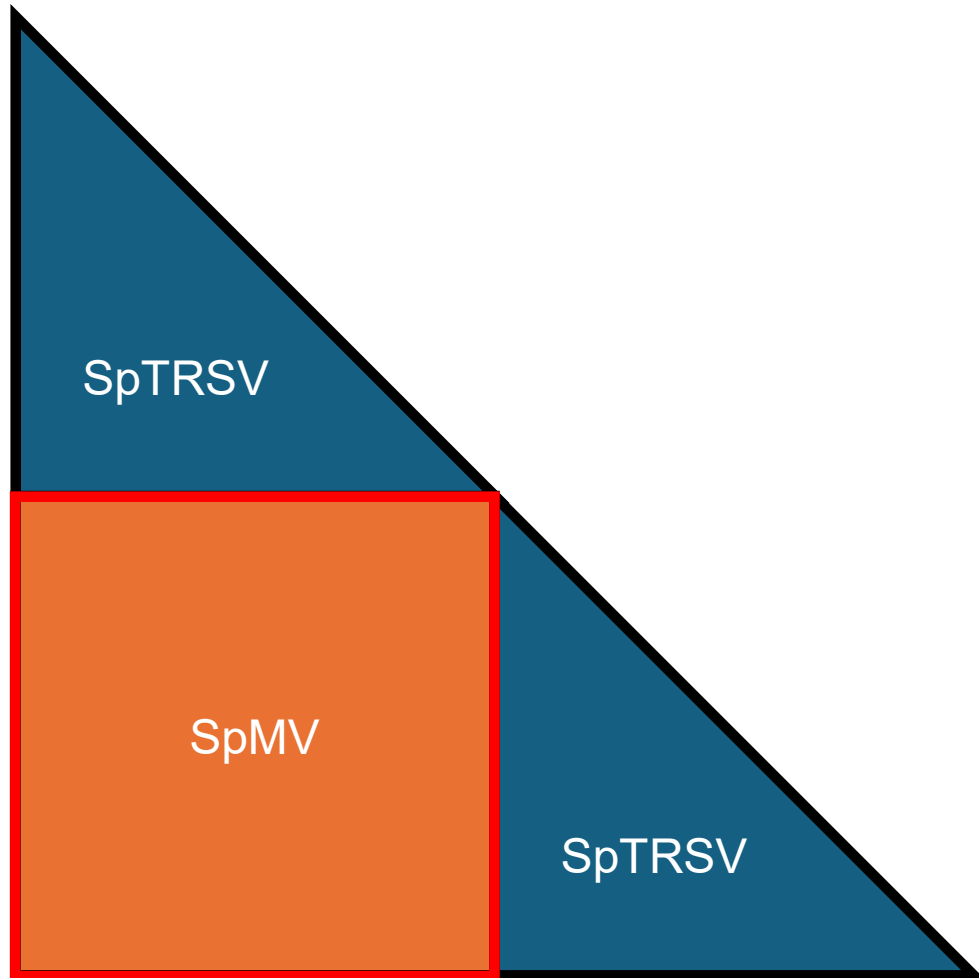
Recursive SpTRSV Algorithm

1. $\vec{x}_1 = L_1^{-1} \vec{b}_1$ (Recursive SpTRSV)

2. $L_2 \vec{x}_2 = \vec{b}_2 - C \vec{x}_1 = \vec{b}_2'$ (SpMV)

3. $\vec{x}_2 = L_2^{-1} \vec{b}_2'$ (Recursive SpTRSV)

SpTRSV Block Algorithm (ICPP 2020¹)



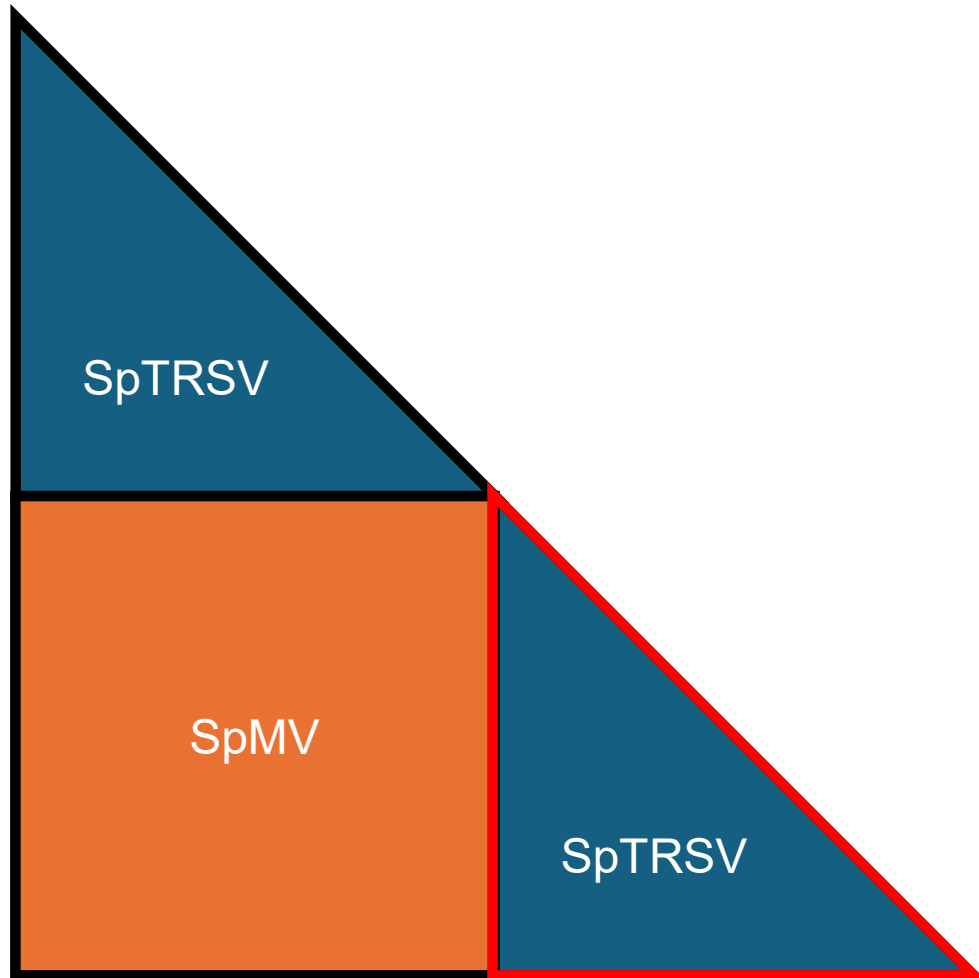
Recursive SpTRSV Algorithm

1. $\bar{x}_1 = L_1^{-1} \bar{b}_1$ (Recursive SpTRSV)

2. $L_2 \bar{x}_2 = \bar{b}_2 - C \bar{x}_1 = \bar{b}_2'$ (SpMV)

3. $\bar{x}_2 = L_2^{-1} \bar{b}_2'$ (Recursive SpTRSV)

SpTRSV Block Algorithm (ICPP 2020¹)



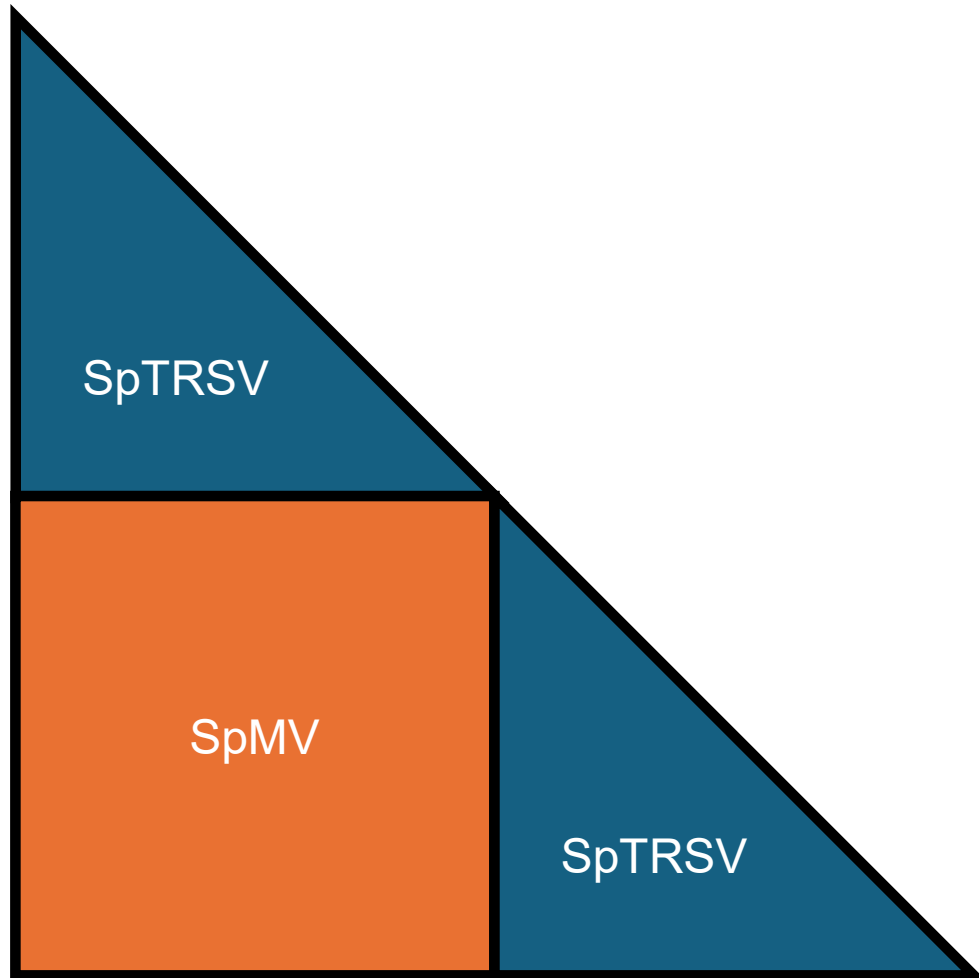
Recursive SpTRSV Algorithm

1. $\bar{x}_1 = L_1^{-1} \bar{b}_1$ (Recursive SpTRSV)

2. $L_2 \bar{x}_2 = \bar{b}_2 - C \bar{x}_1 = \bar{b}_2'$ (SpMV)

3. $\bar{x}_2 = L_2^{-1} \bar{b}_2'$ (Recursive SpTRSV)

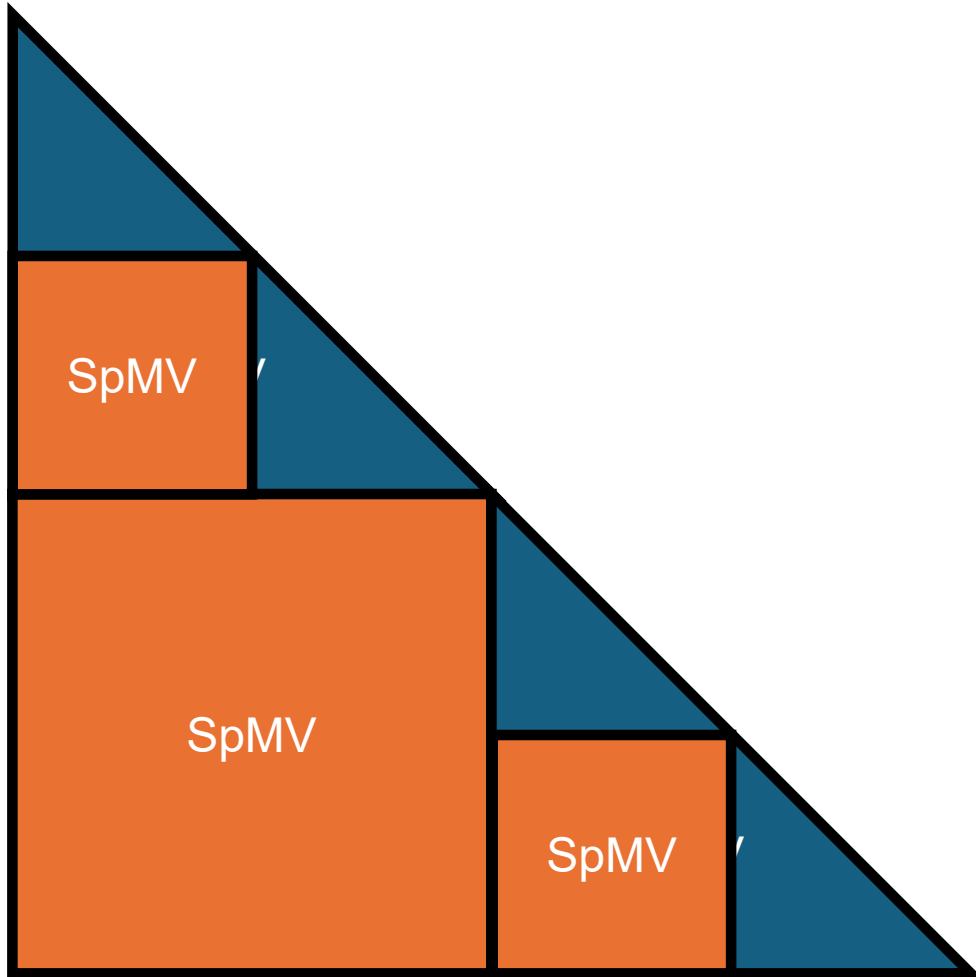
SpTRSV Block Algorithm (ICPP 2020¹)



Recursive SpTRSV Algorithm

1. $\bar{x}_1 = L_1^{-1} \bar{b}_1$ (Recursive SpTRSV)
2. $L_2 \bar{x}_2 = \bar{b}_2 - C \bar{x}_1 = \bar{b}_2'$ (SpMV)
3. $\bar{x}_2 = L_2^{-1} \bar{b}_2'$ (Recursive SpTRSV)

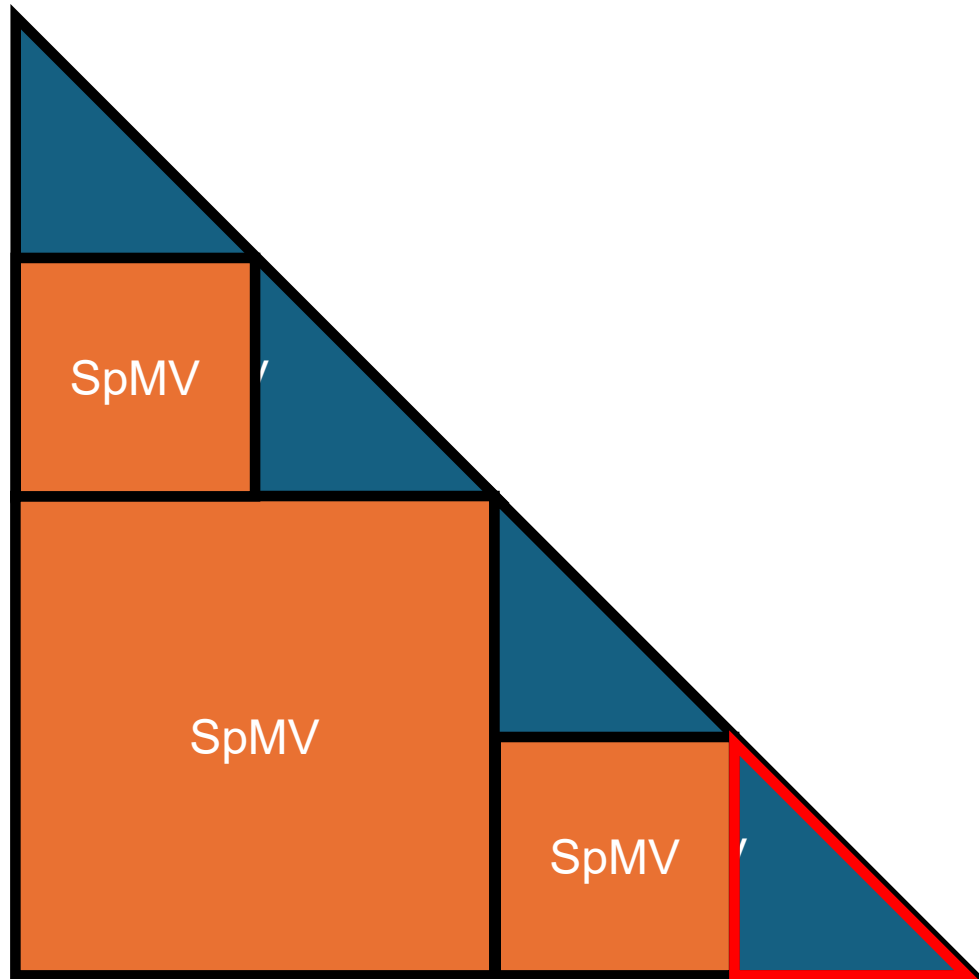
SpTRSV Block Algorithm (ICPP 2020¹)



Recursive SpTRSV Algorithm

1. $\bar{x}_1 = L_1^{-1} \bar{b}_1$ (Recursive SpTRSV)
2. $L_2 \bar{x}_2 = \bar{b}_2 - C \bar{x}_1 = \bar{b}_2'$ (SpMV)
3. $\bar{x}_2 = L_2^{-1} \bar{b}_2'$ (Recursive SpTRSV)

SpTRSV Block Algorithm (ICPP 2020¹)



Recursive SpTRSV Algorithm

1. $\vec{x}_1 = L_1^{-1} \vec{b}_1$ (Recursive SpTRSV)

2. $L_2 \vec{x}_2 = \vec{b}_2 - C \vec{x}_1 = \vec{b}_2'$ (SpMV)

3. $\vec{x}_2 = L_2^{-1} \vec{b}_2'$ (Recursive SpTRSV)

- Implementing a **unit SpTRSV** is sufficient!
- Unit matrix size: 256KB (~16k FP64) dimension
- Unit SpTRSV: **Column-wise** algorithm
- 50% memory bloating for unit triangular matrices

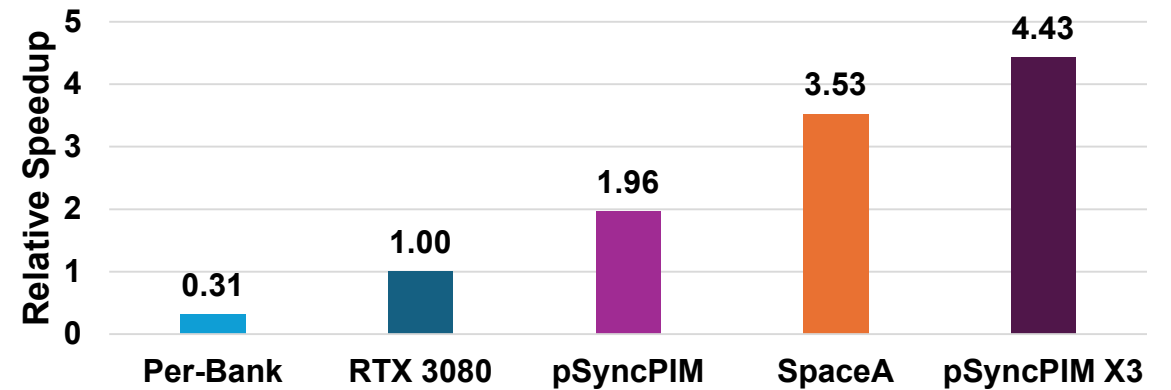
Experiment Setup

- DRAMSim3 based simulator (HBM2 256GB/s)
- Baseline
 - cuSPARSE library with NVIDIA Geforce RTX 3080
 - SpaceA for standalone PIM accelerator
- Benchmarks
 - 15 square sparse matrices from [SuiteSparse Matrix Collection](#)
- Comparison scenarios
 - GPU vs pSyncPIM vs standalone PIM accelerator
 - SpMV & SpTRSV
 - Real-world benchmarks

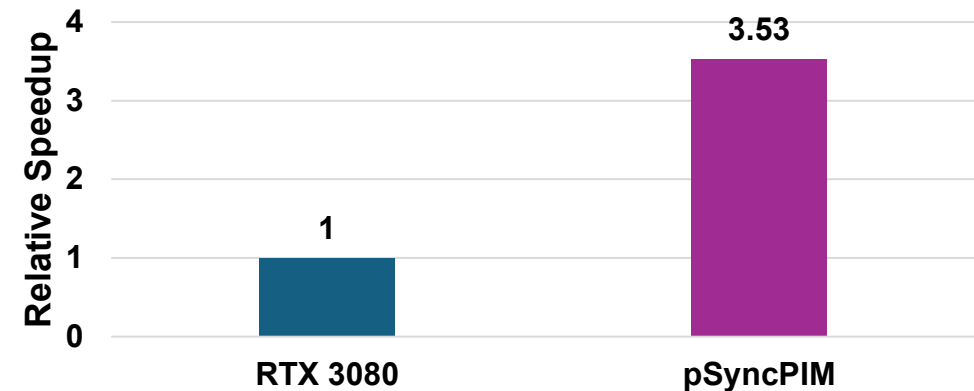
Real-World Benchmarks

- Kernel analysis
 - SpMV: **1.96×** boost over RTX 3080
 - SpMV **3× scenario**: **4.43×** boost
 - SpTRSV: **3.53×** boost over RTX 3080
- Real-world apps
 - **Graph**: **9×** to **156.7×** boost over GraphBLAST¹
 - Triangle Count: InnerSP + pSyncPIM config
 - **Linear systems**: **1.68×** to **2.88×** boost over CUDA
- Area: 68.99 mm^2
- Power limit: 5.0 W
- Other detailed results are in the paper!

SpMV Performance Result



SpTRSV Performance Result



pSyncPIM

- Commercial PIMs: **no support on sparse matrix** kernels
 - Dense matrix-vector multiplication focused
 - Not applicable to sparse matrix
- pSyncPIM
 - **Conditional** execution & exit for sparse matrix kernel execution
 - **Maintains the traditional DRAM interface** for communication with host
 - Proposed a **SpMV & SpTRSV** implementation
- Evaluation results
 - Can accelerate important memory-intensive sparse matrix kernels
 - End-to-end sparse matrix application PIM acceleration