# ASCELLA: Accelerating Sparse Computation by Enabling Stream Accesses to Memory

**Bahar Asgari**, Ramyad Hadidi, Hyesoon Kim

Georgia Tech

DATE 20

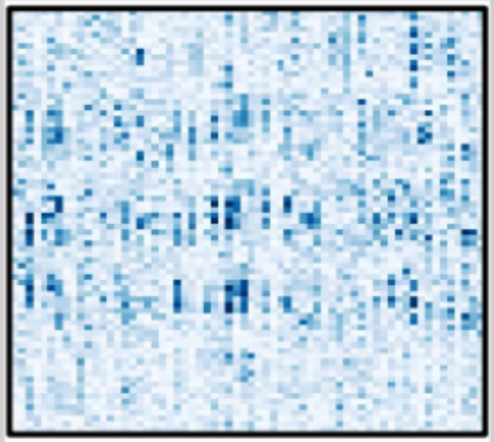
comparch

# Sparse matrices are everywhere!



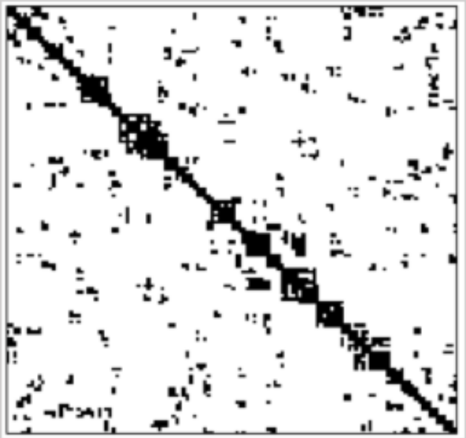
Sparse matrix vector multiplication (SpMV) is a main operation in:

**Neural Networks**



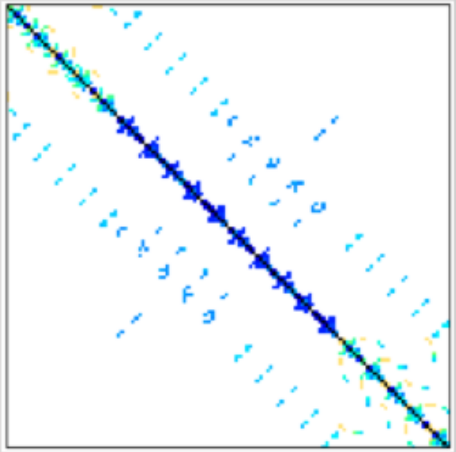
**The weight matrix is sparse**

**Graph Analytics**



**The adjacency matrix is sparse**

**Differential Equations**

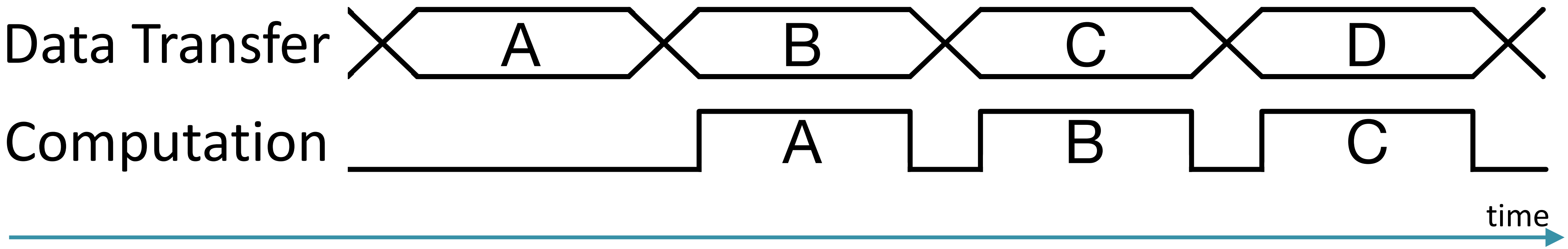


**The coefficient matrix is sparse**

# An ideal hardware accelerator for SpMV



- SpMV can be accelerated:
  - By stream data from memory
  - By using a parallel dot product engine
- Ideally, we want compute time and data-transfer time for blocks of data (e.g., A, B, C, and D) to be equal:

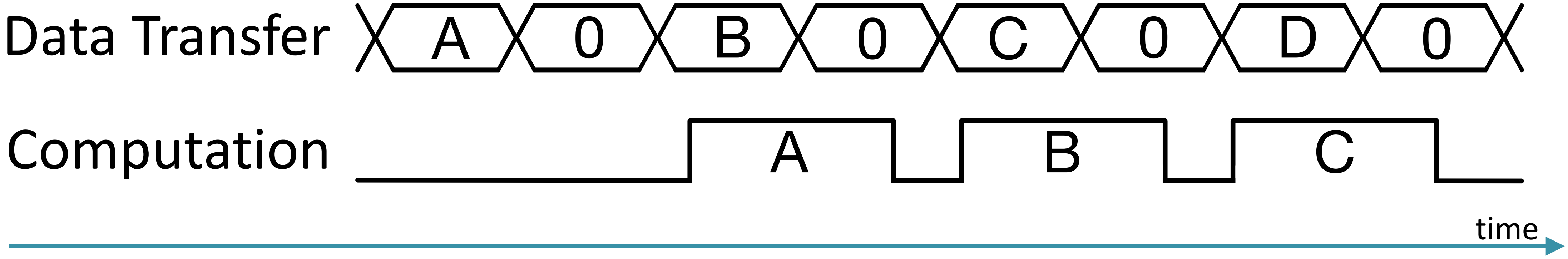



*a block is the unit of streaming data from memory

# Decompression can cause a bottleneck



- Sparse matrices are often stored and transferred in compressed forms
  - Example: compressed sparse row (CSR) and blocked CSR (BCSR)
- The compressed data must be first decompressed
- Decompression is slow and causes bottleneck



# Why decompression is slow?



CSR and BCSR use three vectors to represent a sparse matrix

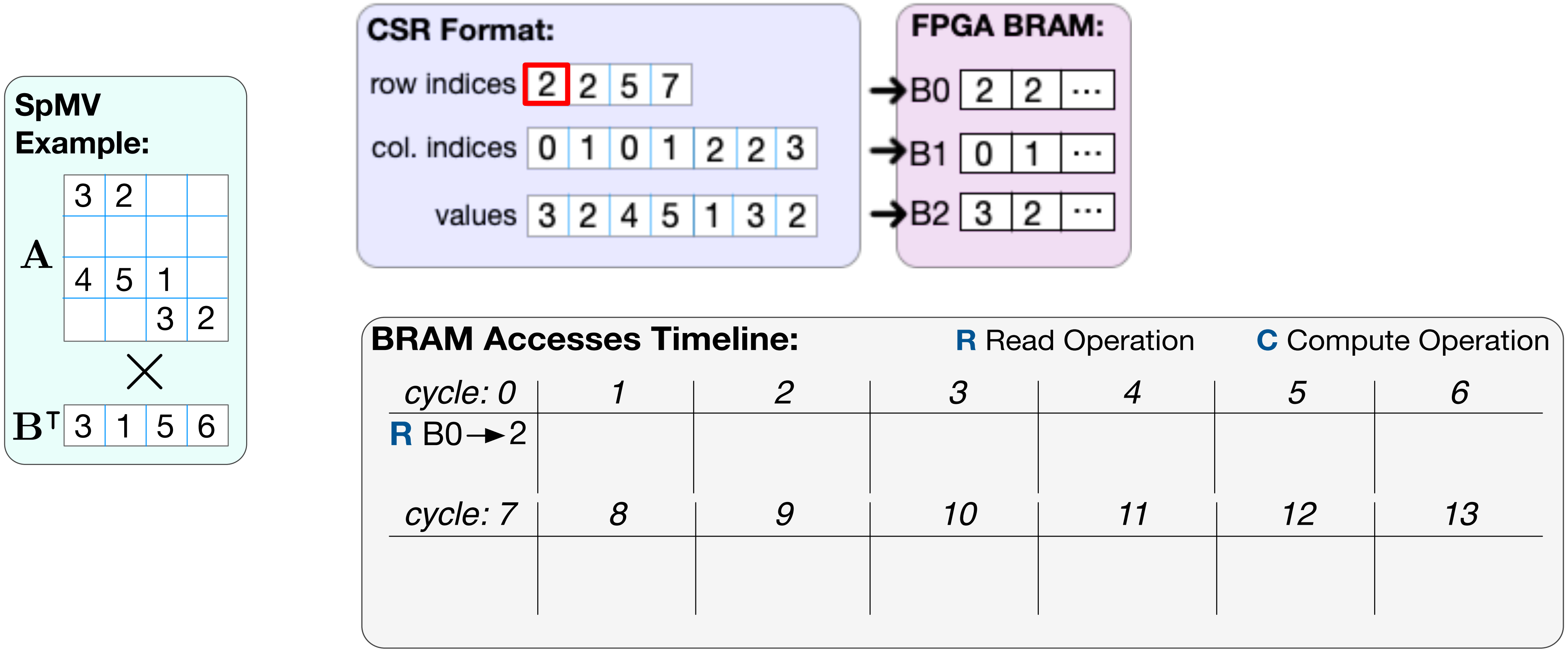
- Row indices (offsets)
- Column indices
- Values

To decompress a non-zero row, we need to

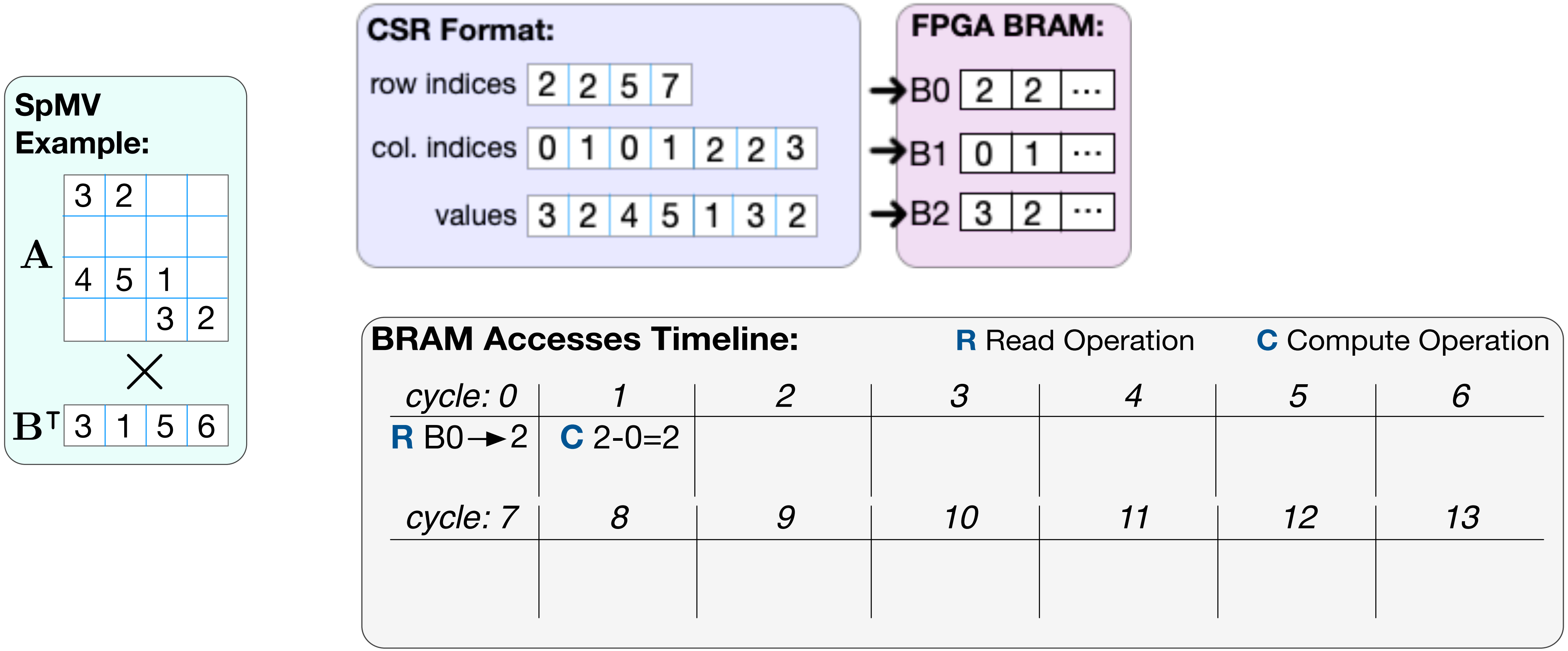
- First, read one element of row indices
- Then, read column indices and values as required

# Decompression from CSR format



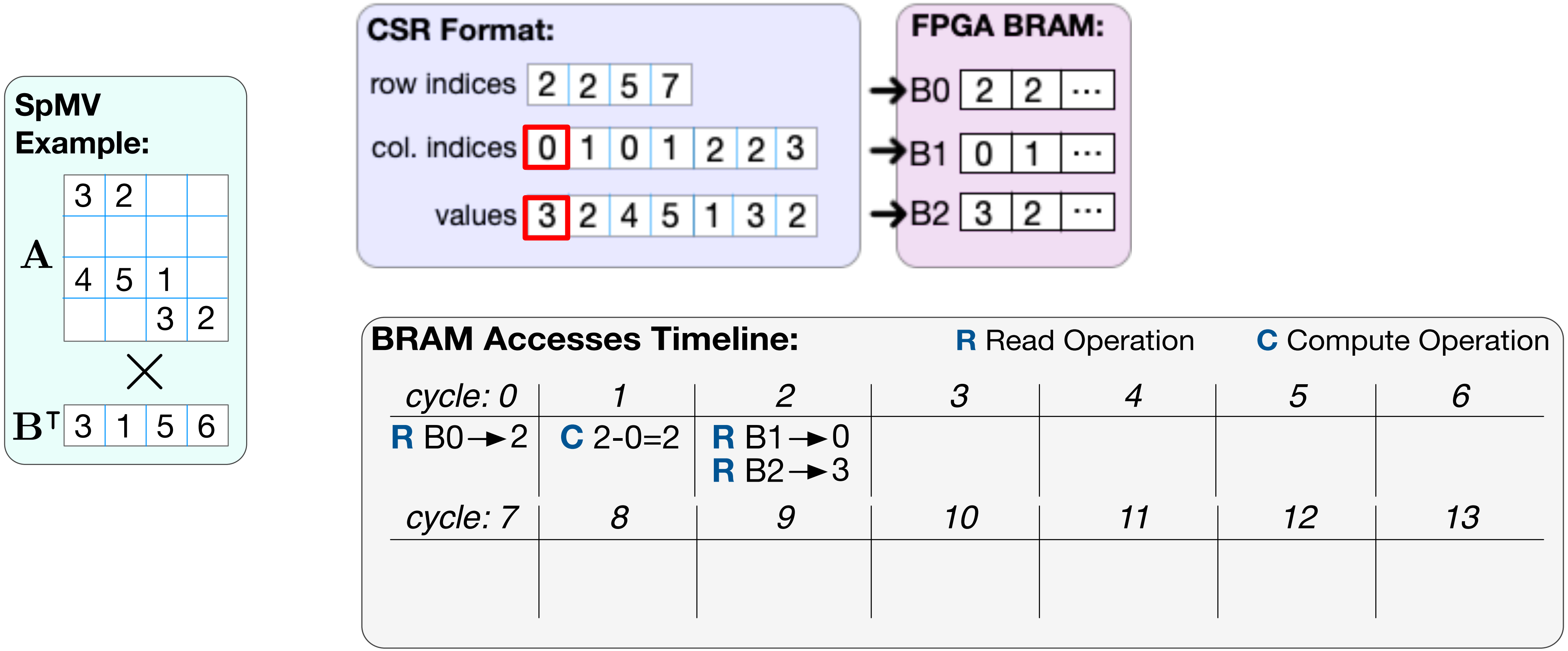


# Decompression from CSR format



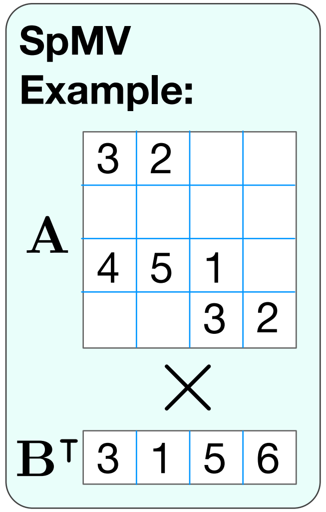
# Decompression from CSR format

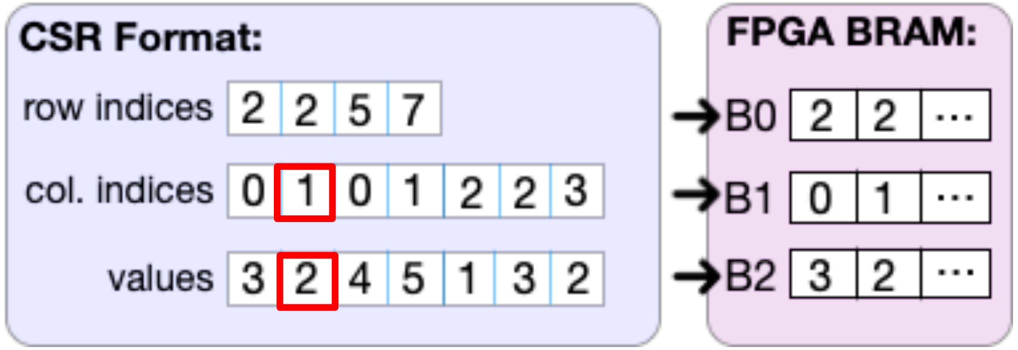




# Decompression from CSR format





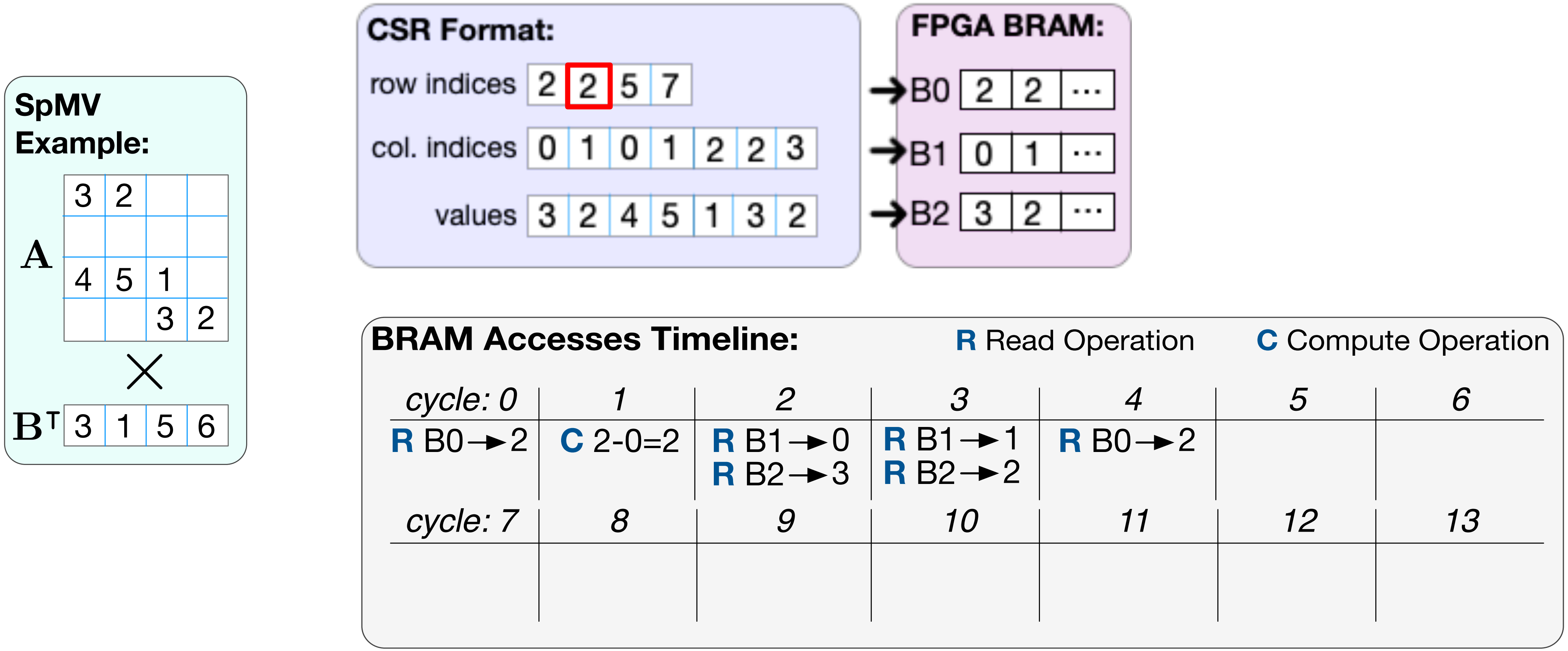




**BRAM Accesses Timeline:** **R** Read Operation **C** Compute Operation

| *cycle: 0* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|
| **R** B0→2 | **C** 2-0=2 | **R** B1→0<br>**R** B2→3 | **R** B1→1<br>**R** B2→2 | | | |
| *cycle: 7* | *8* | *9* | *10* | *11* | *12* | *13* |
| | | | | | | |

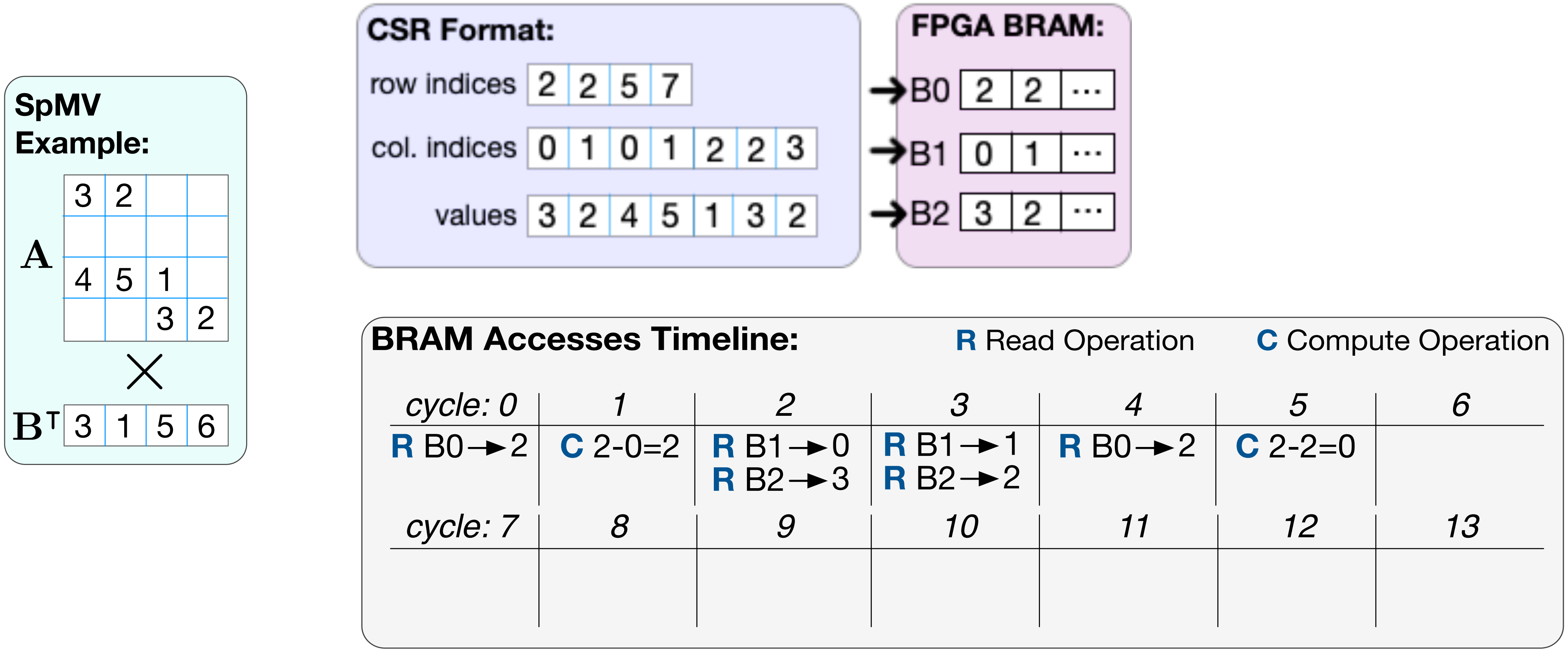
# Decompression from CSR format





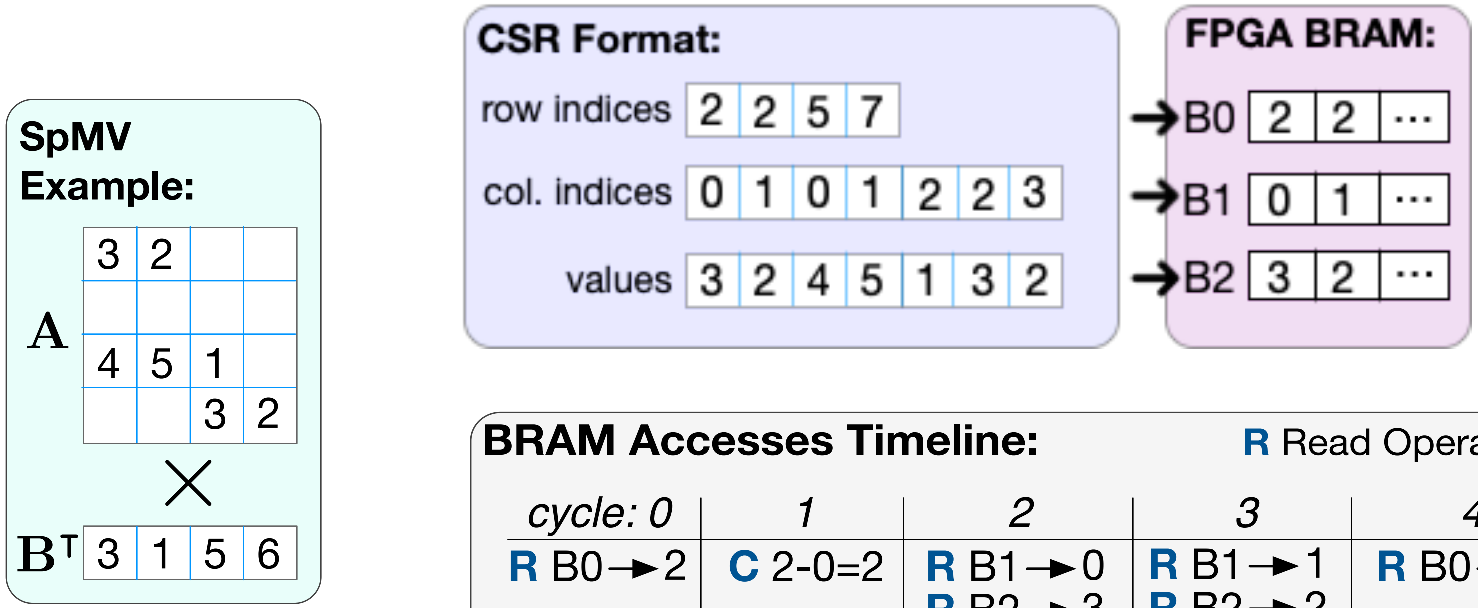
# Decompression from CSR format





# Decompression from CSR format





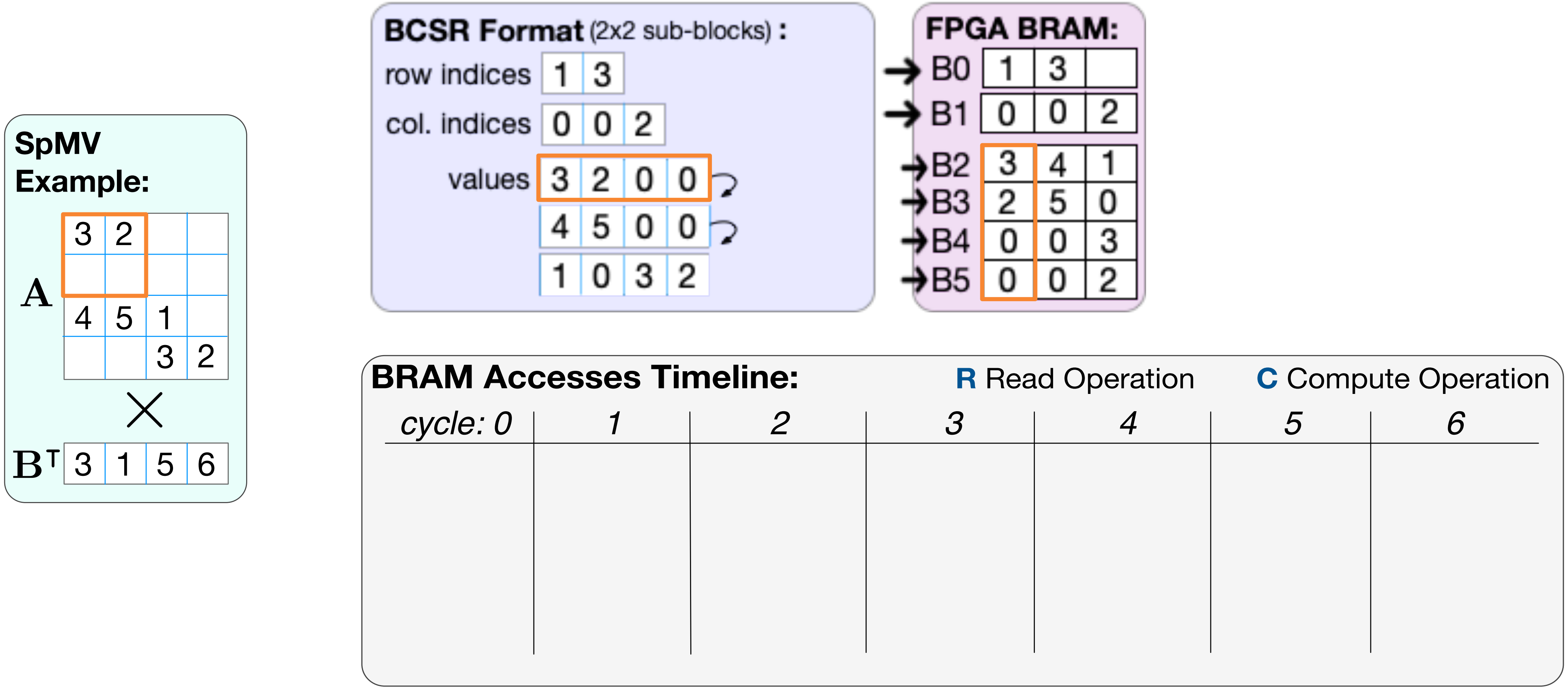

**BRAM Accesses Timeline:** **R** Read Operation **C** Compute Operation

| *cycle: 0* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|
| **R** B0→2 | **C** 2-0=2 | **R** B1→0<br>**R** B2→3 | **R** B1→1<br>**R** B2→2 | **R** B0→2 | **C** 2-2=0 | **R** B0→5 |

| *cycle: 7* | *8* | *9* | *10* | *11* | *12* | *13* |
|---|---|---|---|---|---|---|
| **C** 5-2=3 | **R** B1→0<br>**R** B2→4 | **R** B1→1<br>**R** B2→5 | **R** B1→2<br>**R** B2→1 | **C** 7-5=2 | **R** B1→2<br>**R** B2→3 | **R** B1→3<br>**R** B2→2 |

# Decompression from BCSR format





# Decompression from BCSR format



**SpMV Example:**

A

| | | | |
|---|---|---|---|
| 3 | 2 | | |
| | | | |
| 4 | 5 | 1 | |
| | | 3 | 2 |

×

$B^T$

| | | | |
|---|---|---|---|
| 3 | 1 | 5 | 6 |

**BCSR Format** (2x2 sub-blocks) **:**

row indices

| | |
|---|---|
| 1 | 3 |

col. indices

| | | |
|---|---|---|
| 0 | 0 | 2 |

values

| | | | |
|---|---|---|---|
| 3 | 2 | 0 | 0 |
| 4 | 5 | 0 | 0 |
| 1 | 0 | 3 | 2 |

**FPGA BRAM:**

| | | | |
|---|---|---|---|
| B0 | 1 | 3 | |
| B1 | 0 | 0 | 2 |
| B2 | 3 | 4 | 1 |
| B3 | 2 | 5 | 0 |
| B4 | 0 | 0 | 3 |
| B5 | 0 | 0 | 2 |

**BRAM Accesses Timeline:**

**R** Read Operation **C** Compute Operation

| *cycle: 0* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|
| **R** B0→1 | | | | | | |

# Decompression from BCSR format



**SpMV Example:**

$$A = \begin{bmatrix} 3 & 2 & & \\ & & & \\ 4 & 5 & 1 & \\ & & 3 & 2 \end{bmatrix} \times B^T = \begin{bmatrix} 3 & 1 & 5 & 6 \end{bmatrix}$$

**BCSR Format** (2x2 sub-blocks) **:**

| | | | | |
|---|---|---|---|---|
| row indices | 1 | 3 | | |
| col. indices | 0 | 0 | 2 | |
| values | 3 | 2 | 0 | 0 |
| | 4 | 5 | 0 | 0 |
| | 1 | 0 | 3 | 2 |

**FPGA BRAM:**

| | | | |
|---|---|---|---|
| B0 | 1 | 3 | |
| B1 | 0 | 0 | 2 |
| B2 | 3 | 4 | 1 |
| B3 | 2 | 5 | 0 |
| B4 | 0 | 0 | 3 |
| B5 | 0 | 0 | 2 |

**BRAM Accesses Timeline:** **R** Read Operation **C** Compute Operation

| *cycle: 0* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|
| **R** B0→1 | **C** 1-0=1 | | | | | |

# Decompression from BCSR format



**SpMV Example:**

A

| | | | |
|---|---|---|---|
| 3 | 2 | | |
| | | | |
| 4 | 5 | 1 | |
| | | 3 | 2 |

×

$B^T$

| | | | |
|---|---|---|---|
| 3 | 1 | 5 | 6 |

**BCSR Format** (2x2 sub-blocks) **:**

| | | | | |
|---|---|---|---|---|
| row indices | 1 | 3 | | |
| col. indices | 0 | 0 | 2 | |
| values | 3 | 2 | 0 | 0 |
| | 4 | 5 | 0 | 0 |
| | 1 | 0 | 3 | 2 |

**FPGA BRAM:**

| | | | |
|---|---|---|---|
| → B0 | 1 | 3 | |
| → B1 | 0 | 0 | 2 |
| → B2 | 3 | 4 | 1 |
| → B3 | 2 | 5 | 0 |
| → B4 | 0 | 0 | 3 |
| → B5 | 0 | 0 | 2 |

**BRAM Accesses Timeline:**

**R** Read Operation **C** Compute Operation

| *cycle: 0* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|
| **R** B0→1 | **C** 1-0=1 | **R** B1→0 | | | | |
| | | **R** B2→3 | | | | |
| | | **R** B3→2 | | | | |
| | | **R** B4→0 | | | | |
| | | **R** B5→0 | | | | |

# Decompression from BCSR format

**SpMV Example:**

A

| | | | |
|---|---|---|---|
| 3 | 2 | | |
| | | | |
| 4 | 5 | 1 | |
| | | 3 | 2 |

×

$B^T$

| | | | |
|---|---|---|---|
| 3 | 1 | 5 | 6 |

**BCSR Format** (2x2 sub-blocks) **:**

| | | | | |
|---|---|---|---|---|
| row indices | 1 | $\boxed{3}$ | | |
| col. indices | 0 | 0 | 2 | |
| values | 3 | 2 | 0 | 0 |
| | 4 | 5 | 0 | 0 |
| | 1 | 0 | 3 | 2 |

**FPGA BRAM:**

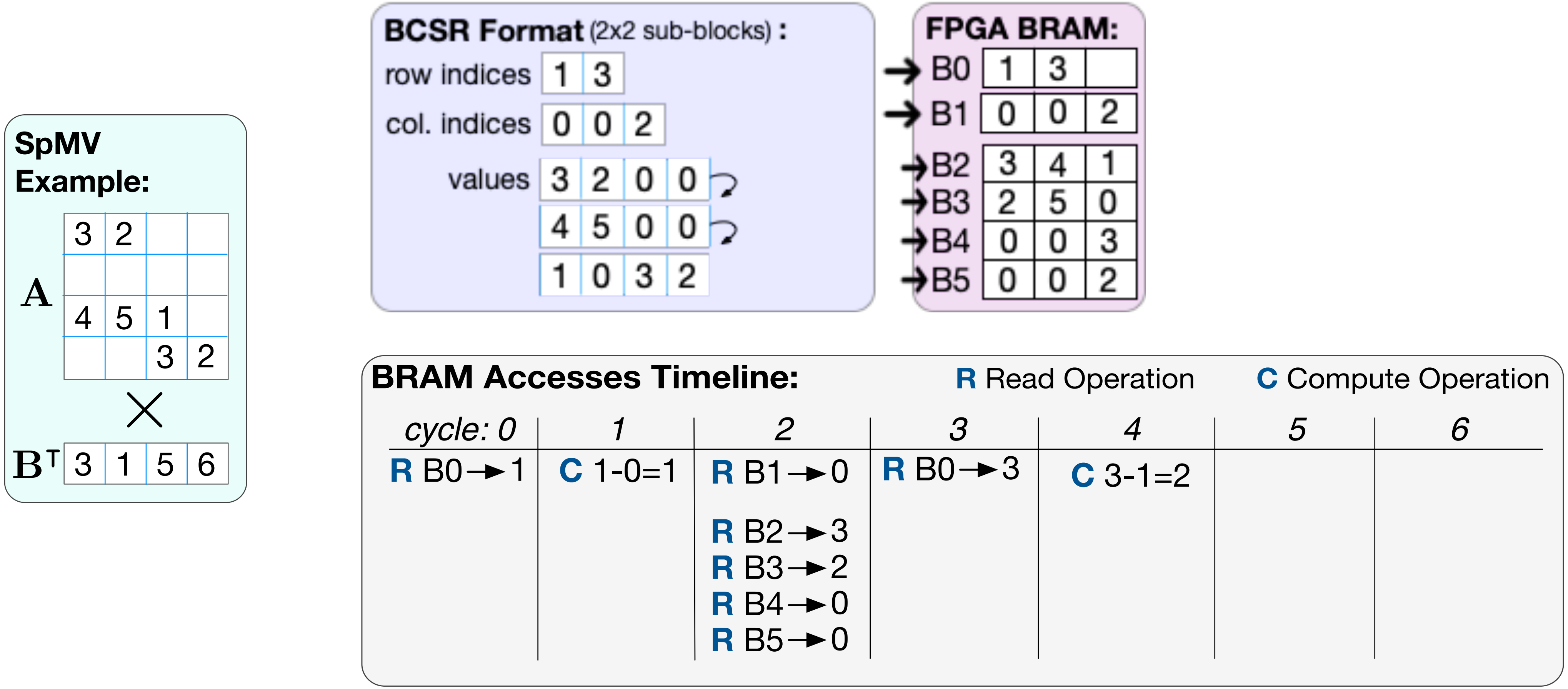
| | | | |
|---|---|---|---|
| → B0 | 1 | 3 | |
| → B1 | 0 | 0 | 2 |
| → B2 | 3 | 4 | 1 |
| → B3 | 2 | 5 | 0 |
| → B4 | 0 | 0 | 3 |
| → B5 | 0 | 0 | 2 |

**BRAM Accesses Timeline:** **R** Read Operation **C** Compute Operation

| *cycle: 0* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|
| **R** B0→1 | **C** 1-0=1 | **R** B1→0 | **R** B0→3 | | | |
| | | **R** B2→3 | | | | |
| | | **R** B3→2 | | | | |
| | | **R** B4→0 | | | | |
| | | **R** B5→0 | | | | |

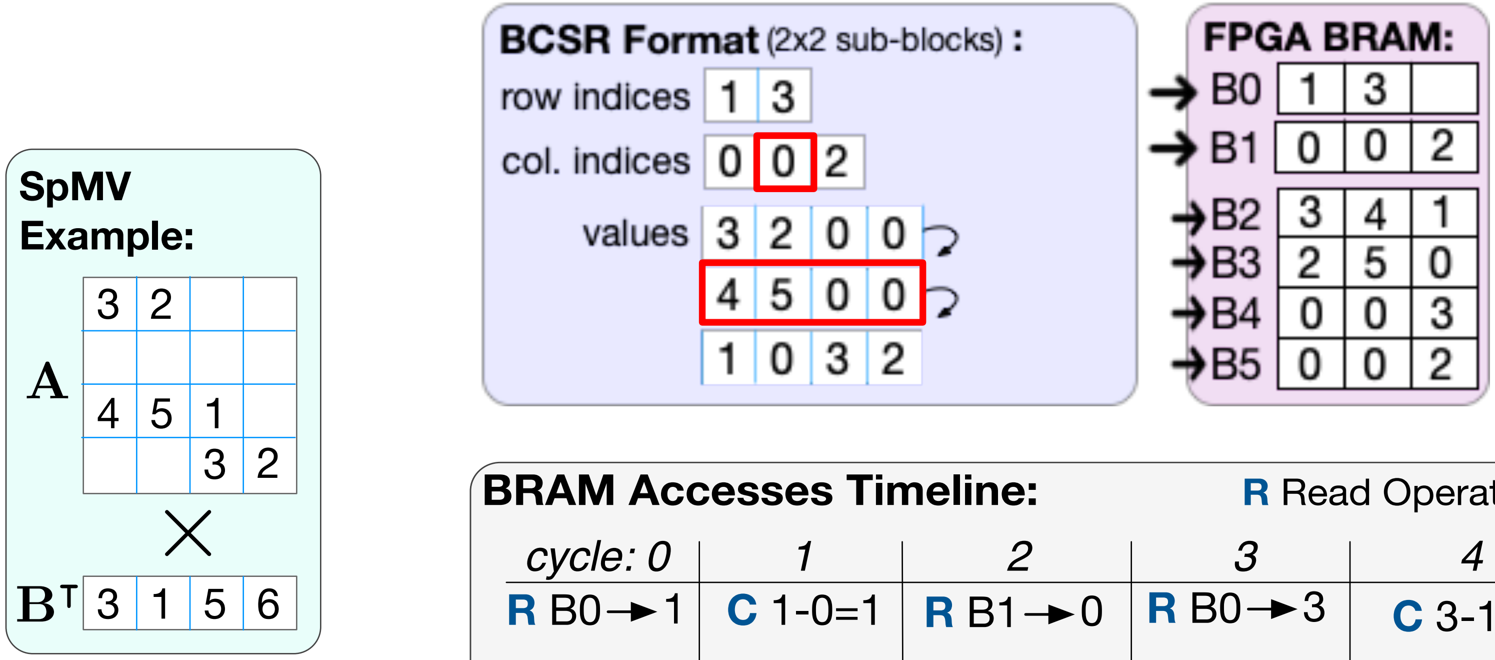
# Decompression from BCSR format





# Decompression from BCSR format

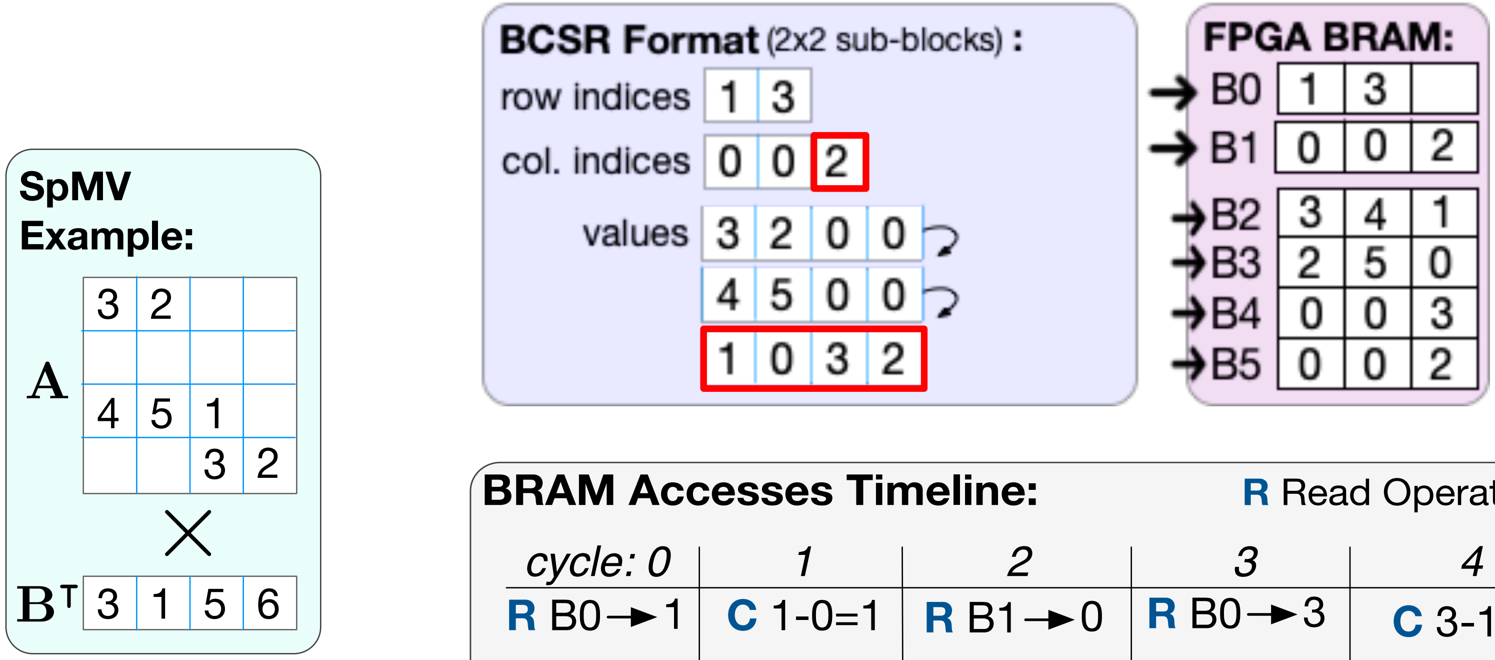





**BRAM Accesses Timeline:** **R** Read Operation **C** Compute Operation

| *cycle: 0* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|
| **R** B0→1 | **C** 1-0=1 | **R** B1→0 | **R** B0→3 | **C** 3-1=2 | **R** B1→0 | |
| | | **R** B2→3 | | | **R** B2→4 | |
| | | **R** B3→2 | | | **R** B3→5 | |
| | | **R** B4→0 | | | **R** B4→0 | |
| | | **R** B5→0 | | | **R** B5→0 | |

# Decompression from BCSR format




**BRAM Accesses Timeline:** **R** Read Operation **C** Compute Operation

| *cycle: 0* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|
| **R** B0→1 | **C** 1-0=1 | **R** B1→0 | **R** B0→3 | **C** 3-1=2 | **R** B1→0 | **R** B1→2 |
| | | **R** B2→3 | | | **R** B2→4 | **R** B2→1 |
| | | **R** B3→2 | | | **R** B3→5 | **R** B3→0 |
| | | **R** B4→0 | | | **R** B4→0 | **R** B4→3 |
| | | **R** B5→0 | | | **R** B5→0 | **R** B5→2 |

# Key Challenge of Decompressing CSR and BCSR



Creating each row of data has following overheads:

- One access to the meta data
- One computation

Reading the column indices and values is sequential because

- We do not know in advance which elements of column indices and values are going to be accessed.
- We cannot partition and allocate those two vectors across the blocks of BRAM to guarantee parallel reads.

# Key Insights and Solutions



To address the challenge we propose Ascella

Ascella achieves the ideal streaming for sparse problems by

- Avoiding extra accesses to meta data
- Providing deterministic parallel accesses to data

Ascella is an accelerator for SpMV that sustains a balance between computation and data-transfer time

# Contributions of Ascella



## Ascella uses a compressed format similar to list of lists (LIL[1]):

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 3 | 2 | | |
| 1 | | | | |
| 2 | 4 | 5 | 1 | |
| 3 | | | 3 | 2 |

→

values

| | | | |
|---|---|---|---|
| 3 | 2 | 1 | 2 |
| 4 | 5 | 3 | 0 |

row indices

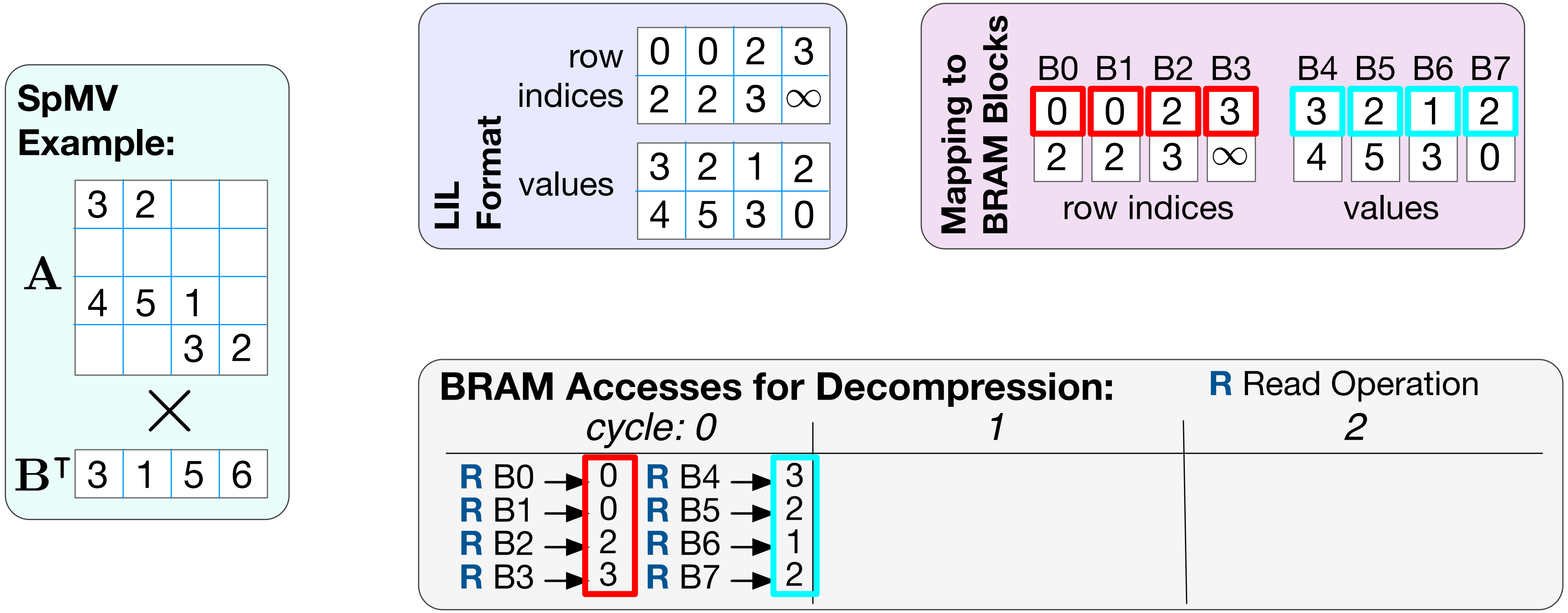
| | | | |
|---|---|---|---|
| 0 | 0 | 2 | 3 |
| 2 | 2 | 3 | $\infty$ |

## Ascella implements a lightweight microarchitecture that

- Connects the streamlines of memory to the parallel dot-product engine
- Enables ideal data streaming

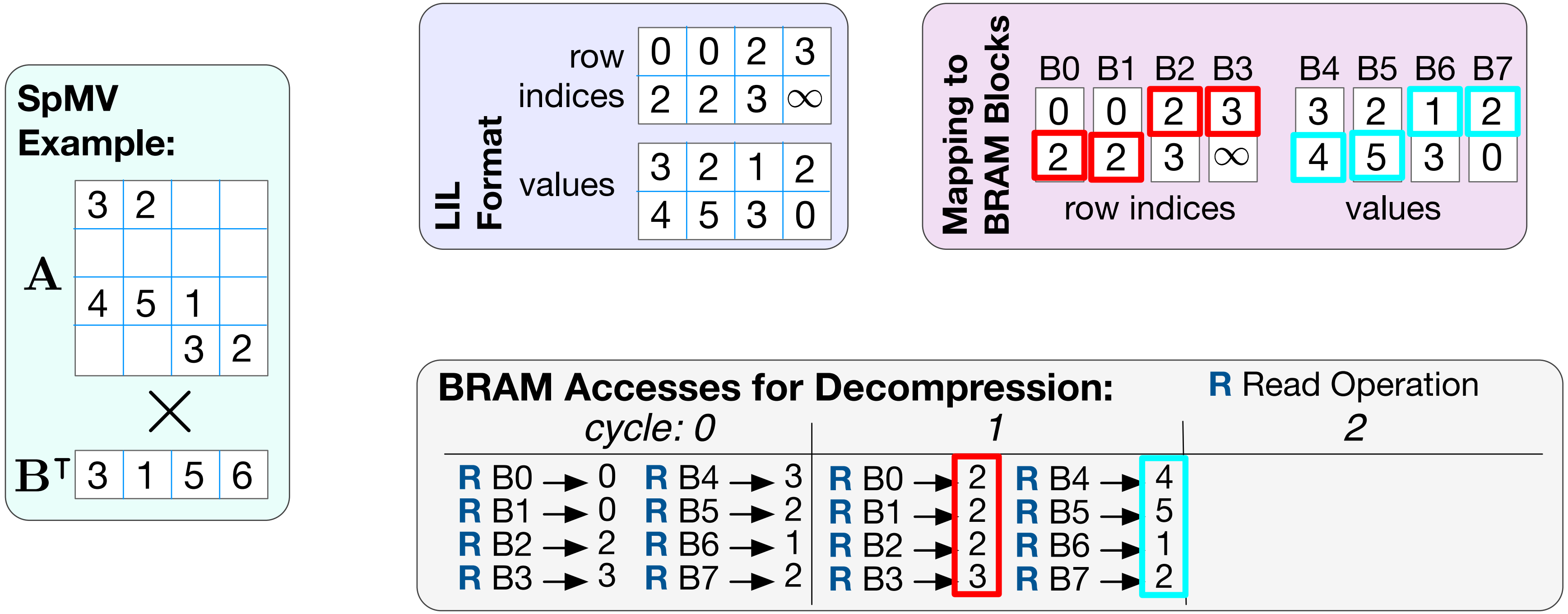
[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.lil_matrix.html

# Decompression mechanism of Ascella



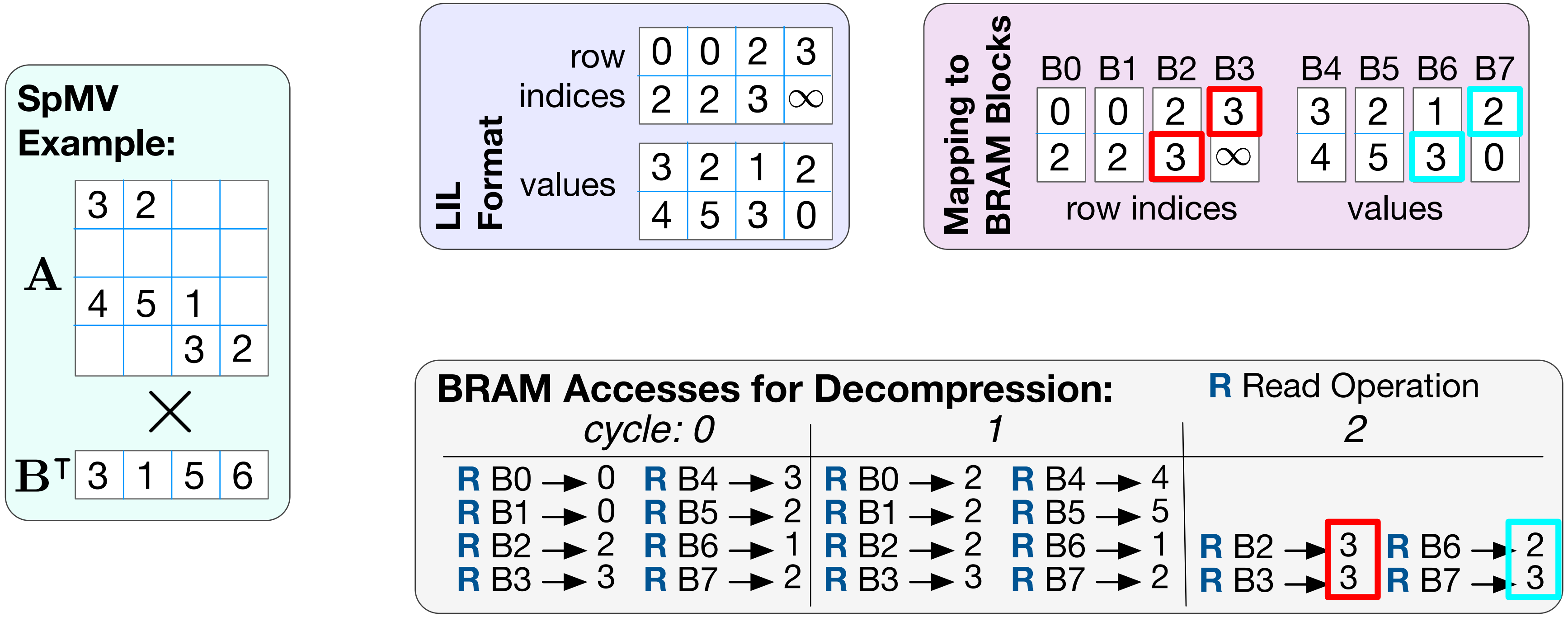
# Decompression mechanism of Ascella





# Decompression mechanism of Ascella



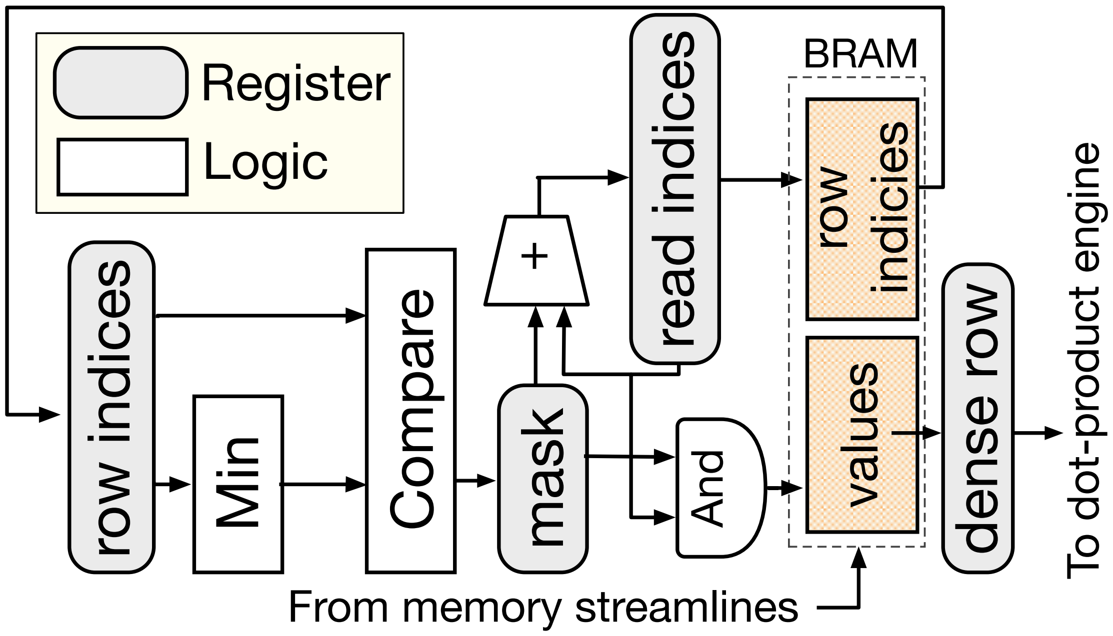


# Microarchitecture of Ascella



At each step of decompression:

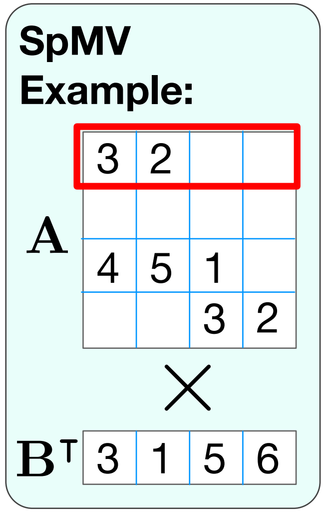
- ***read indices*** are used to to read the ***row indices***
- the minimum of ***row indices*** is used to create a ***mask***
- the mask
  - Selects values of ***dense row***
  - Updates the ***read indices***

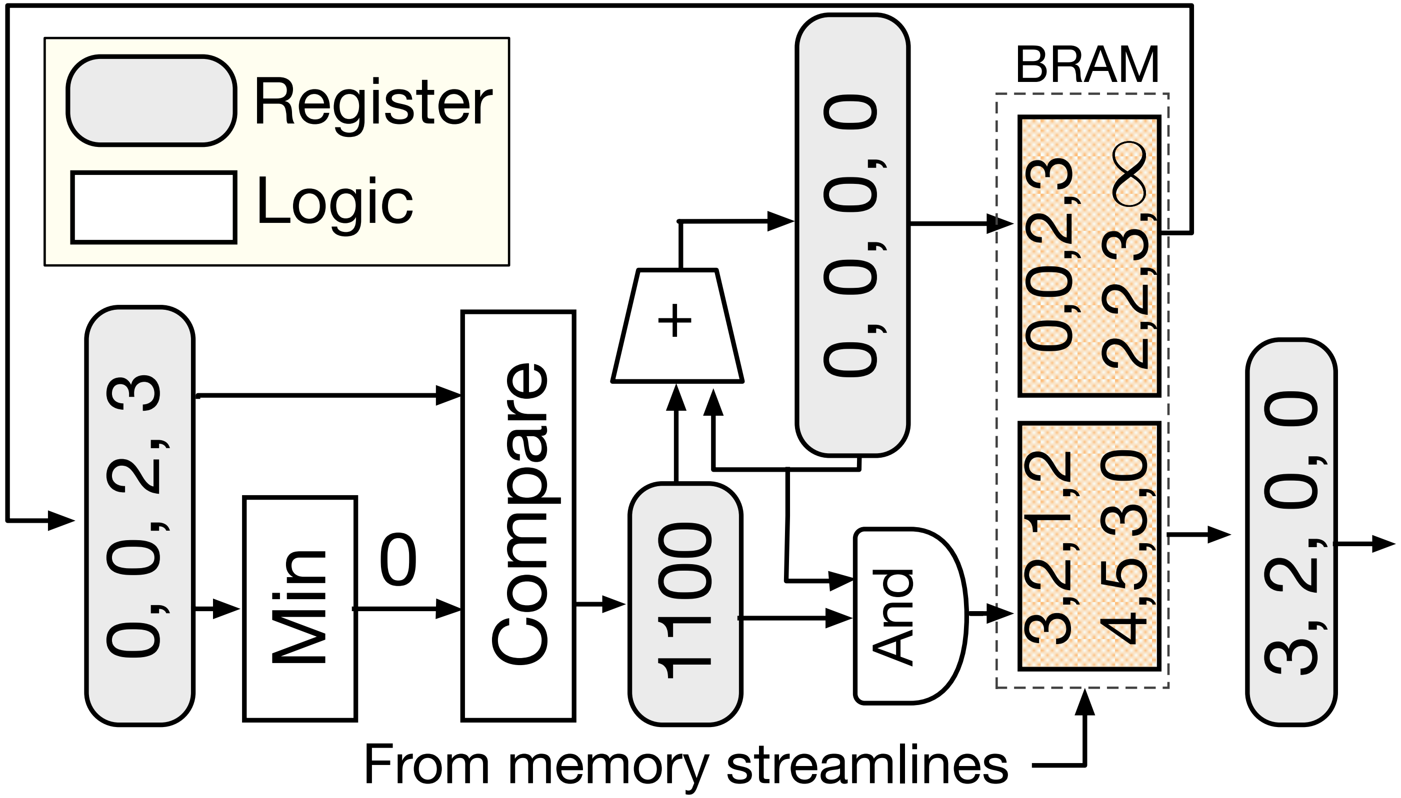


# Microarchitecture of Ascella



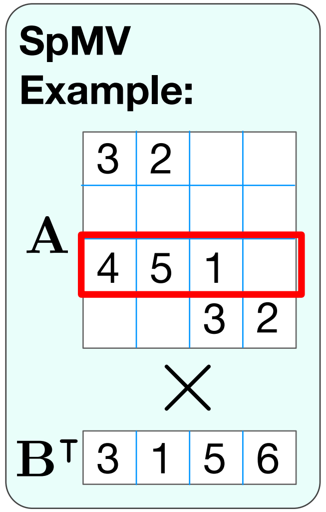
Creating the first row:

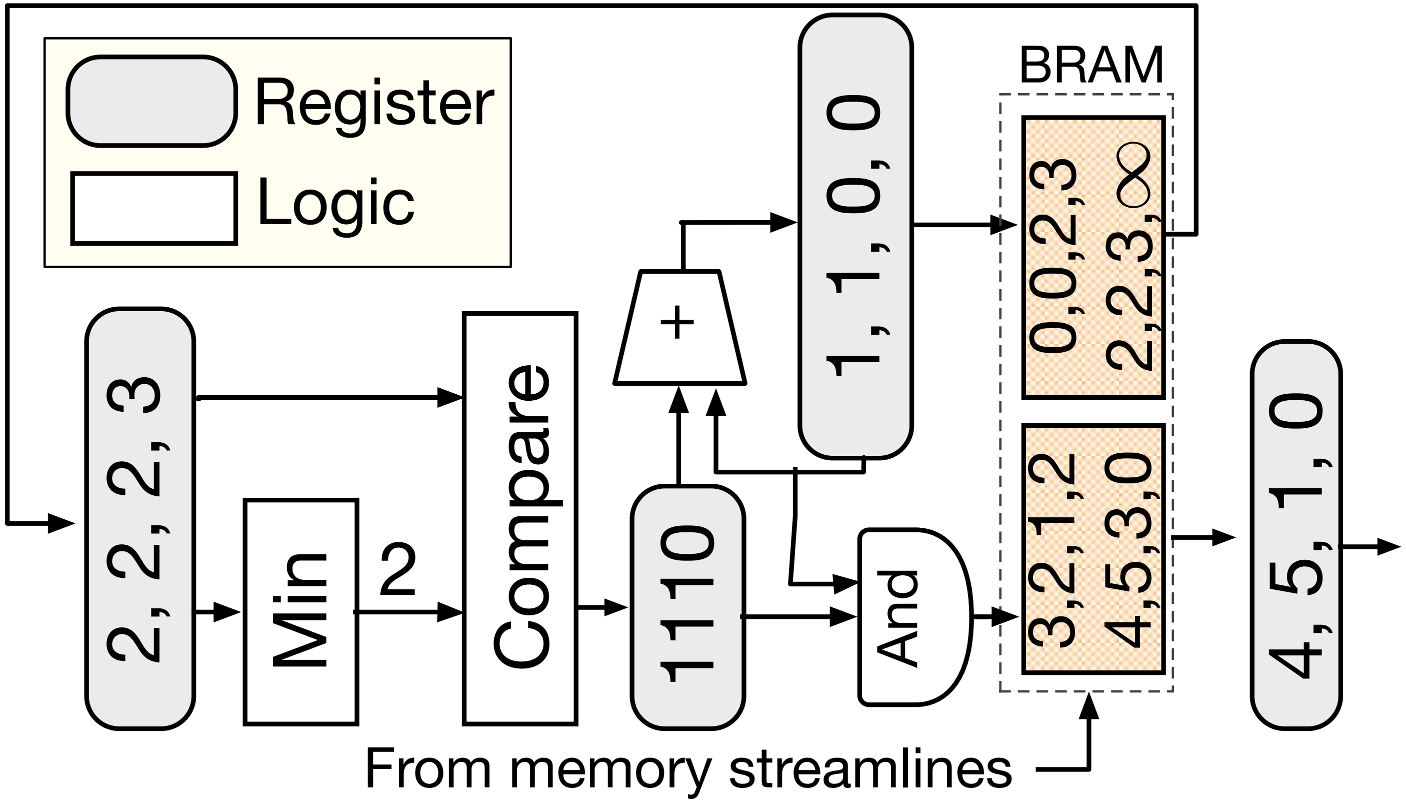


# Microarchitecture of Ascella

## Creating the third row:

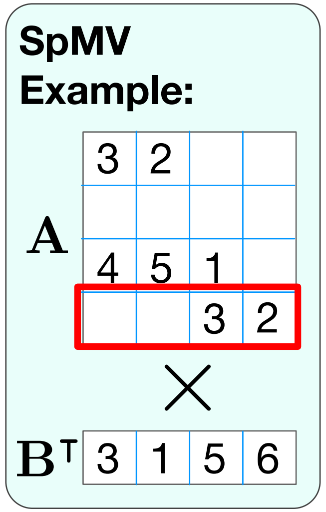


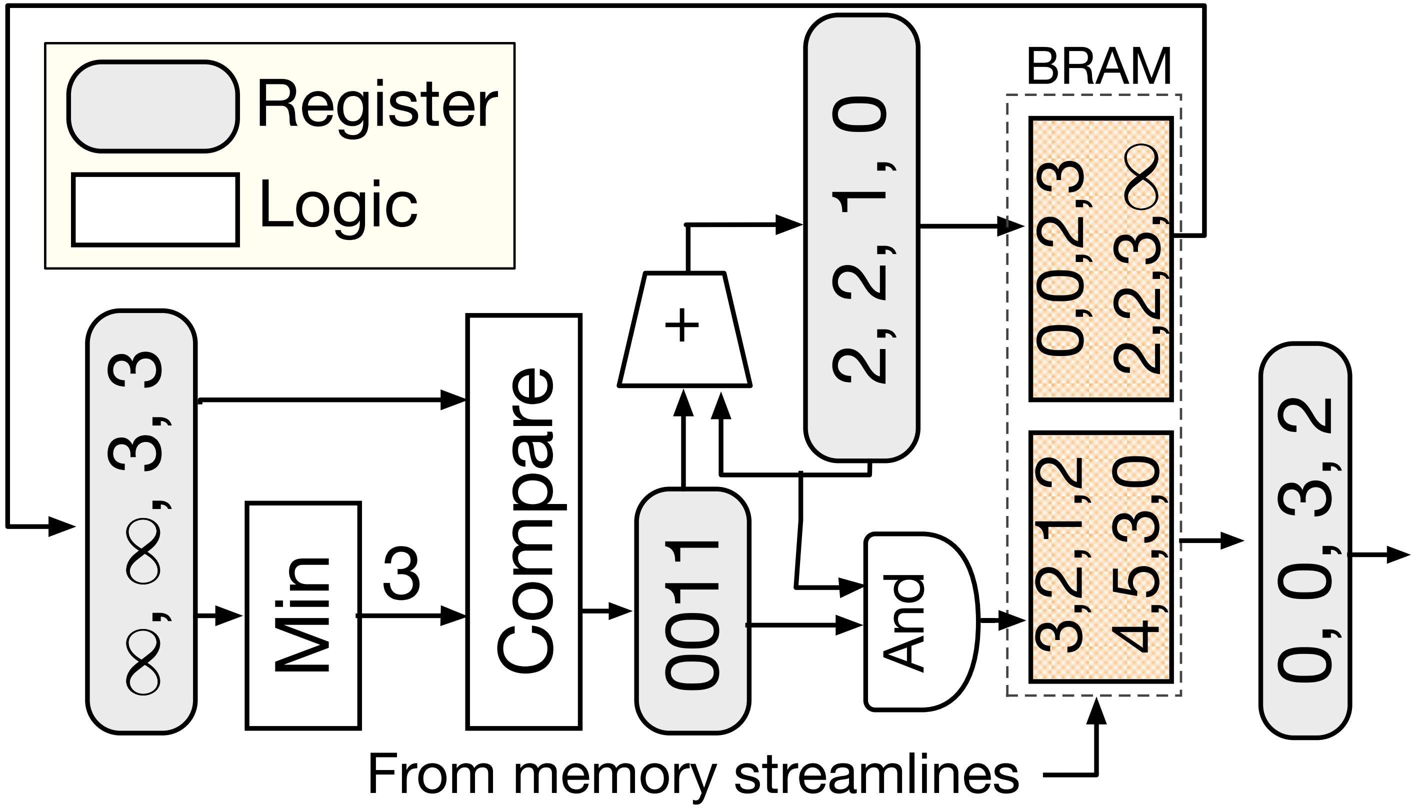



# Microarchitecture of Ascella



Creating the last row:



# Experimental Setup



We apply SpMV on a group of matrices from SuiteSparse collection

We implement Ascella and the baselines

- Using Xilinx Vivado HLS
- On ZYNQ XC7Z020FPGA.

Our comparison metrics are

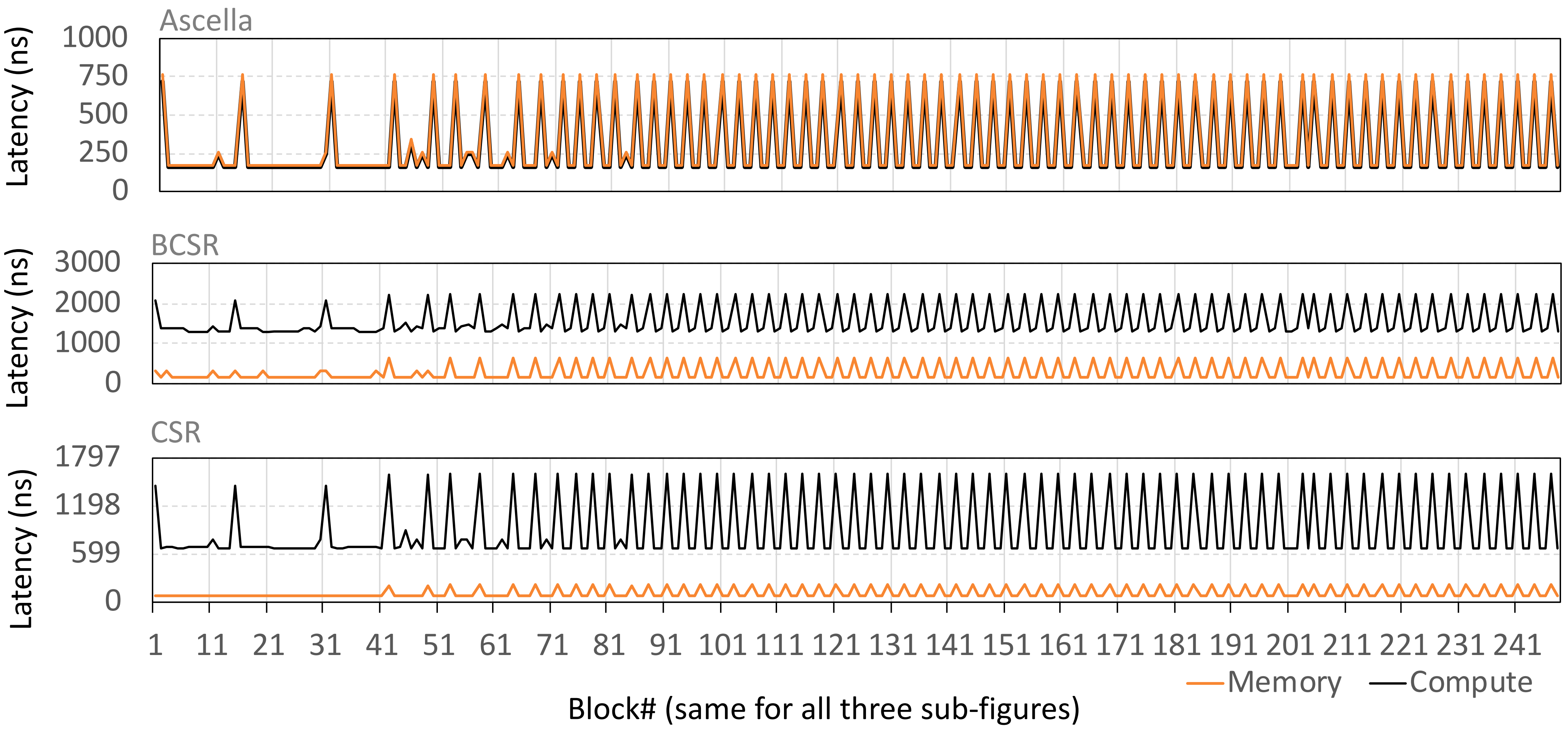
- Latency
- Recourse utilization

The configurations of Ascella and baselines include

- AXI stream interfaces
- DDR3 memory
- 100 MHz frequency
- 32-bit integers

# Performance Evaluation



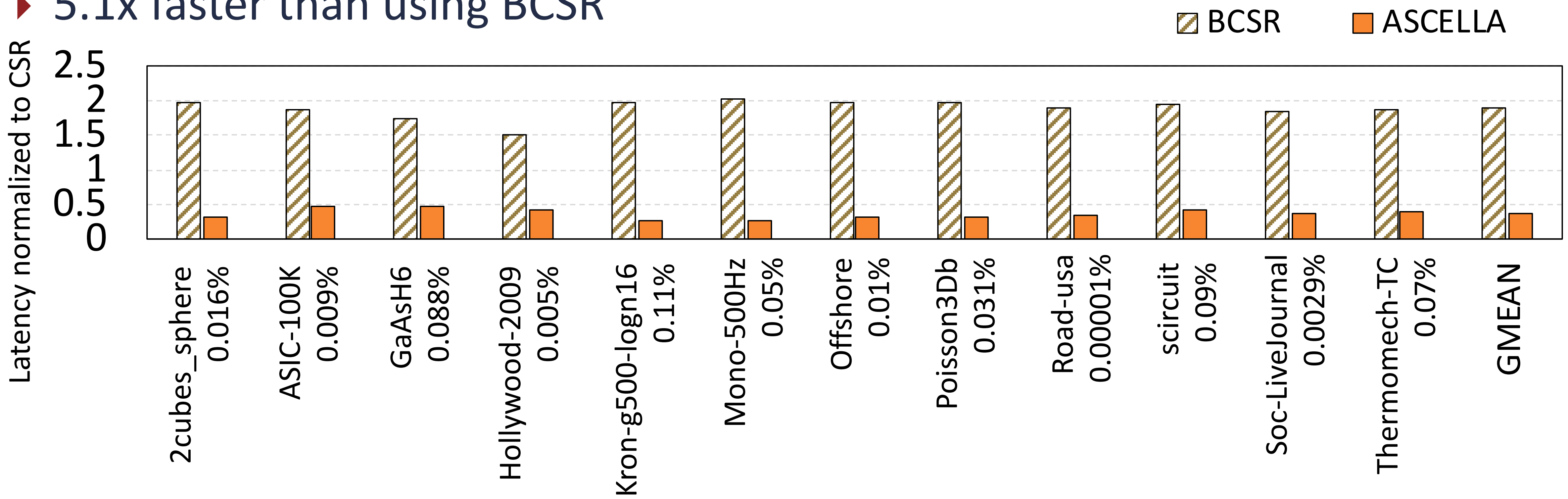



Dataset: thermomech-TC

# Performance Evaluation

On average, Ascella executes SpMV

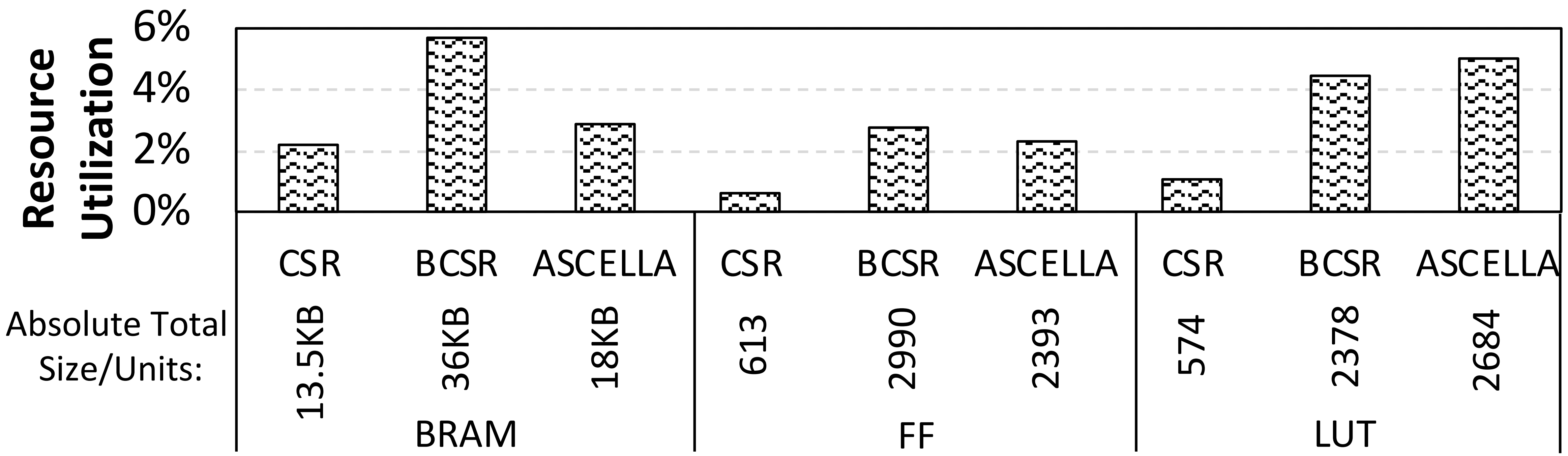
- 2.7x faster than using CSR
- 5.1x faster than using BCSR



# Resource Utilization



- BCSR and Ascella use more BRAM because of partitioning.
- CSR has the lowest flip-flop and look-up table (LUT) utilization because it does not implement any parallelism.



# Conclusions



Ascella is a streaming accelerator for sparse problems that

- Streams the non-zero values of sparse matrices and processes them as they come at the same pace
- Is a significant step towards accelerating larger sparse problems because its storage format
  - facilitates partitioning large matrices
  - is supported in Python libraries, which makes the implementation straightforward