SMASH

Co-Designing Software Compression and Hardware-Accelerated Indexing for Efficient Sparse Matrix Operations

Konstantinos Kanellopoulos, Nandita Vijaykumar, Christina Giannoula, Roknoddin Azizi, Skanda Koppula, Nika Mansouri Ghiasi, Taha Shahroodi, Juan Gomez Luna, Onur Mutlu

MICRO 2019

SAFARI ETH zürich Carnegie Mellon University

Reviewed by Tuan Pham Huu

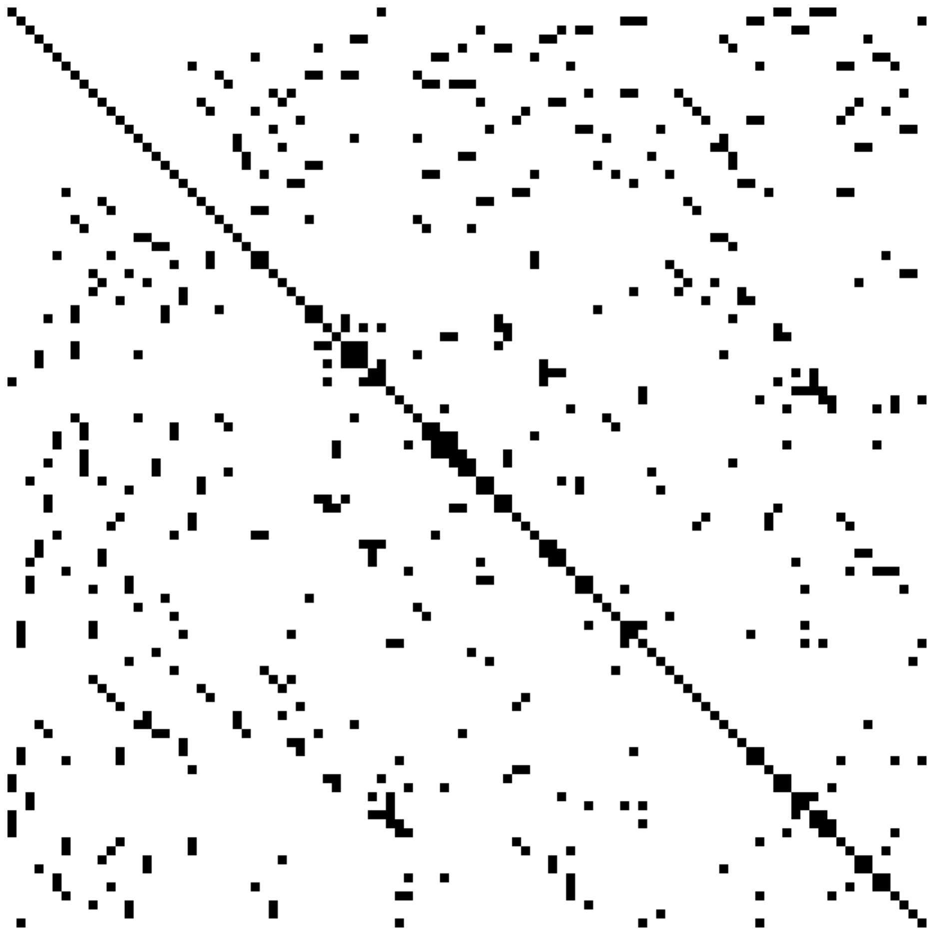
Executive Summary

- Many applications heavily rely on **sparse linear algebra**
- They require **effective compression formats** to mitigate **computational and storage overheads**
- Existing **formats** suffer from the **expensive cost** of finding the **position of non-zero elements**
- **SMASH**: **Hardware/Software cooperative mechanism** for **efficient non-zero elements discovery** and **sparse matrix operations**
  - **Software**: Efficient compression with a **hierarchy of bitmaps**
  - **Hardware**: Scans bitmaps to **find indices of non-zero elements**
- **38 - 44 %** faster than the state-of-the-art for **SpMV and SpMM**
- SMASH is **highly effective, low cost, and widely applicable**

Outline

- Definition and Applications of a **Sparse Matrix**
- Properties of an **Effective Compression Format**
- State-of-the-art: **CSR and BCSR**
- **Indexing Overhead** of CSR and BCSR
- **SMASH**: Effective Mechanism for Sparse Matrix Compression
- **Evaluation Methodology** and **Key Results**
- **Conclusion**
- **Critique** and **Discussion**

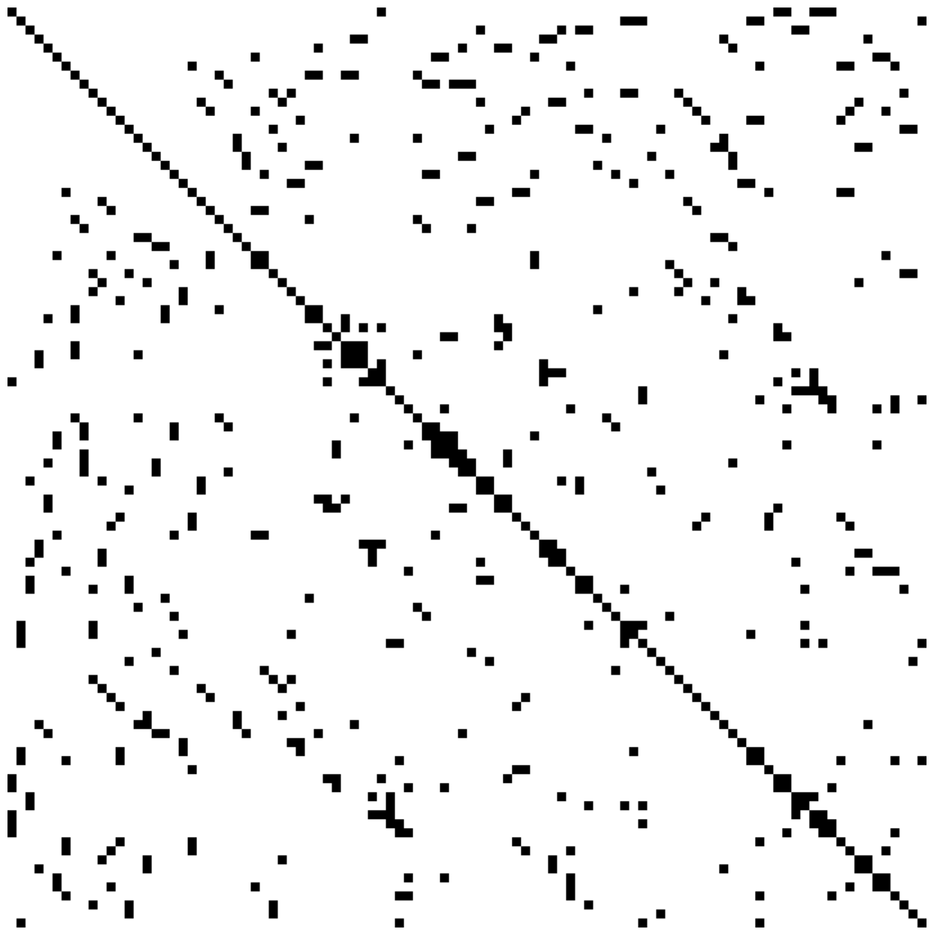
Definition: Sparse Matrix



- **Majority** of its values are 0
- **Widely used** in modern applications
- Vary in **size** and **sparsity**
- **Computations** with zeros are **unnecessary**
- **Storing** zeros is **wasteful** and/or **infeasible**

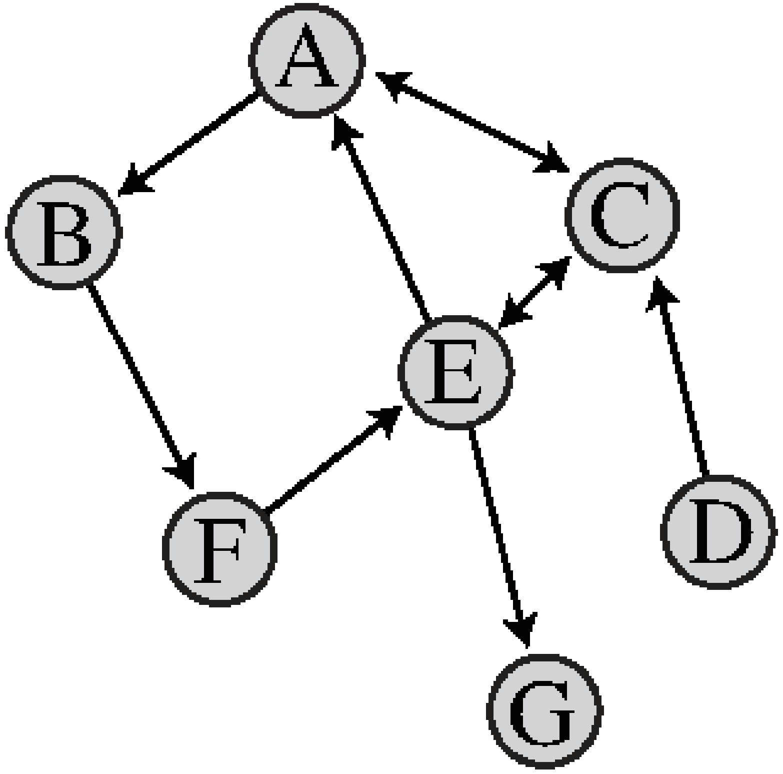
https://en.wikipedia.org/wiki/Sparse_matrix#/media/File:Finite_element_sparse_matrix.png

Definition: Sparse Matrix



Need
sparse matrix compression
to avoid
overheads

Application: Graph Analytics For Social Networks



Adjacency Matrix M

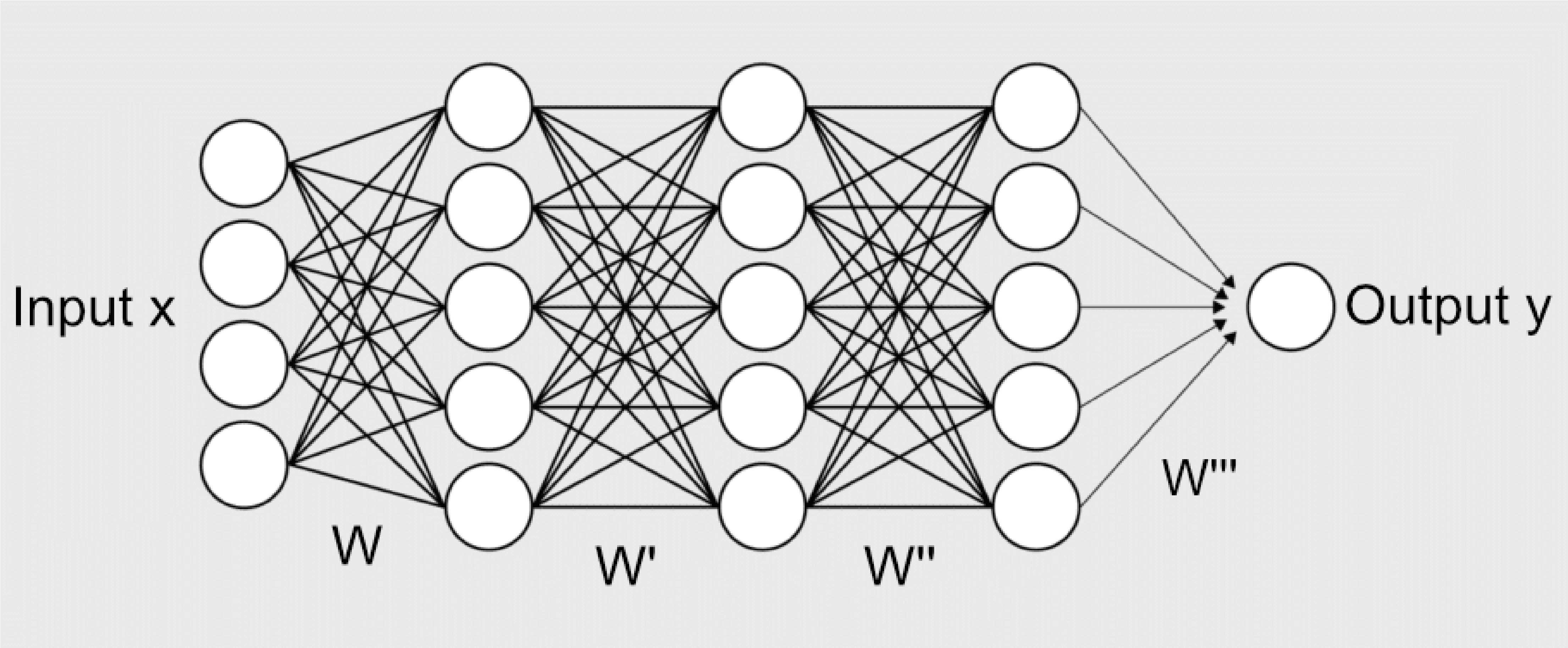
	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	0	0	0	0	0	1	0
C	1	0	0	0	1	0	0
D	0	0	1	0	0	0	0
E	1	0	1	0	0	0	1
F	0	0	0	0	1	0	0
G	0	0	0	0	0	0	0

https://www.semanticscholar.org/paper/Topics-in-social-network-analysis-and-network-O'Malley-Onnela/61ede53272f795e1115badd45f28164e34b98a37/figure/0

Application: Graph Analytics For Social Networks

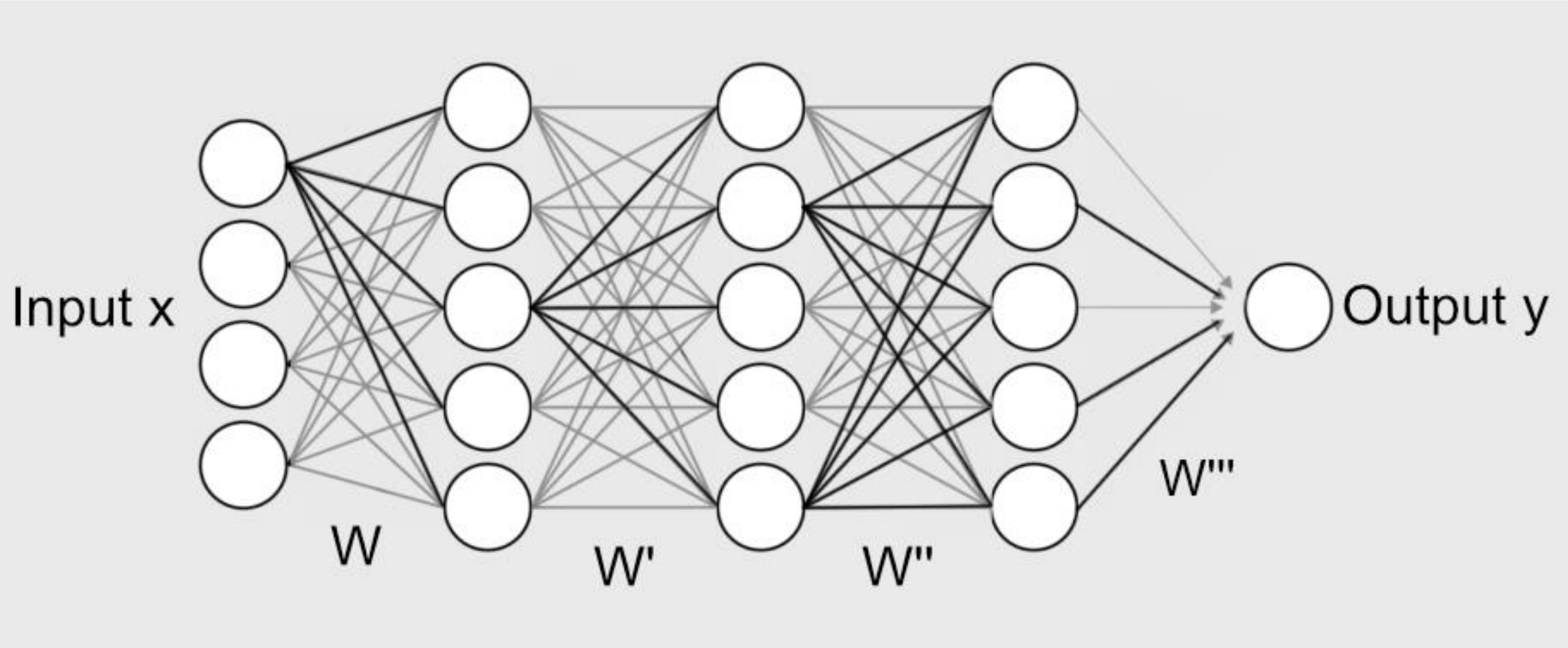
https://www.facebook.com/notes/facebook-engineering/visualizing-friendships/469716398919

Application: Sparse Weights in DNN with Dropout



https://dimensionless.in/what-is-neural-network/neural-network-schematic/

Application: Sparse Weights in DNN with Dropout

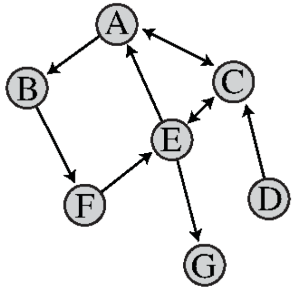


Application: Diagonal Matrix Computation

$$\underbrace{\begin{pmatrix} d_1 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 \\ 0 & 0 & 0 & d_4 & 0 \\ 0 & 0 & 0 & 0 & d_5 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}}_{x} = diag(A) \underset{cwise}{*} x = \underbrace{\begin{pmatrix} d_1 * x_1 \\ d_2 * x_2 \\ d_3 * x_3 \\ d_4 * x_4 \\ d_5 * x_5 \end{pmatrix}}_{y}$$

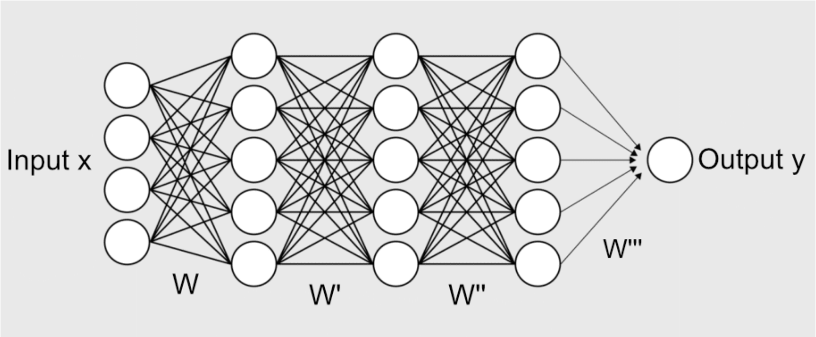
Exploiting the structure of A → From quadratic to linear complexity

Properties of an Effective Compression Format



Adjacency Matrix M

	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	0	0	0	0	0	1	0
C	1	0	0	0	1	0	0
D	0	0	1	0	0	0	0
E	1	0	1	0	0	0	1
F	0	0	0	0	1	0	0
G	0	0	0	0	0	0	0

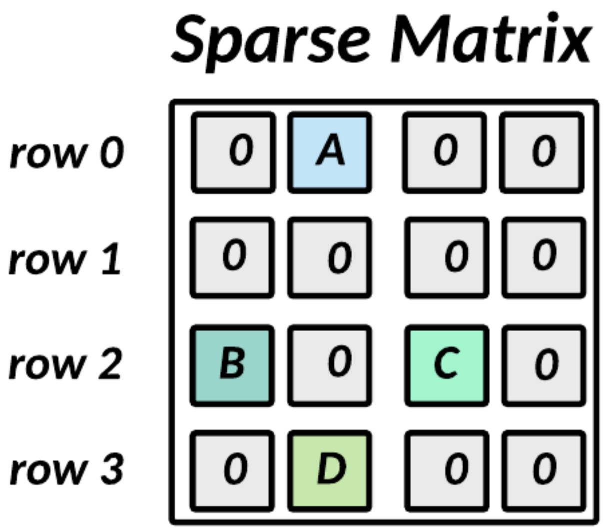


$$\underset{A}{\begin{pmatrix} d_1 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 \\ 0 & 0 & 0 & d_4 & 0 \\ 0 & 0 & 0 & 0 & d_5 \end{pmatrix}} \underset{x}{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}} = diag(A) \underset{cwise}{*} x = \underset{y}{\begin{pmatrix} d_1 * x_1 \\ d_2 * x_2 \\ d_3 * x_3 \\ d_4 * x_4 \\ d_5 * x_5 \end{pmatrix}}$$

1. **Storage Efficiency**
2. **Computational Efficiency**
3. **General Applicability**

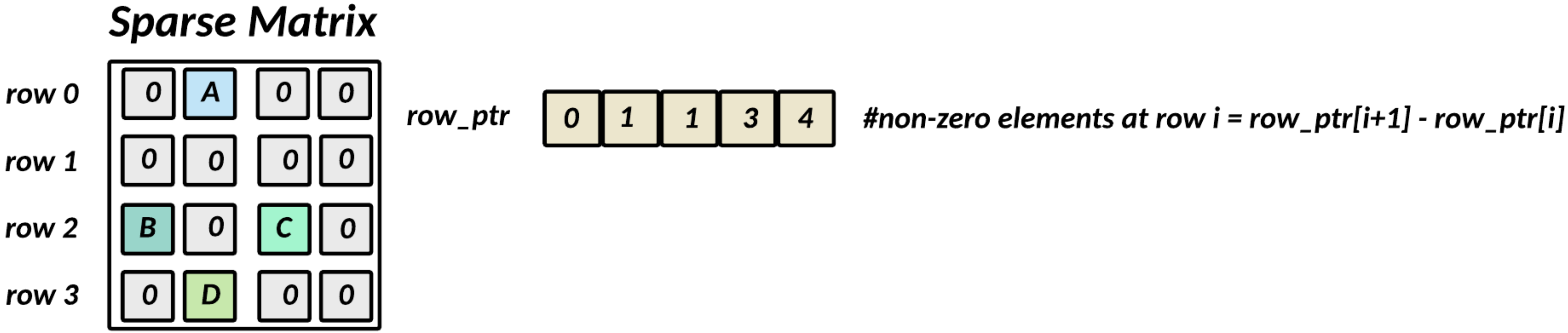
Compressed Sparse Row (CSR)

- **Most widely** used (Intel MKL, OpenBLAS, Eigen, etc.)
- Compressed Sparse Column (**CSC**) follows the same concept



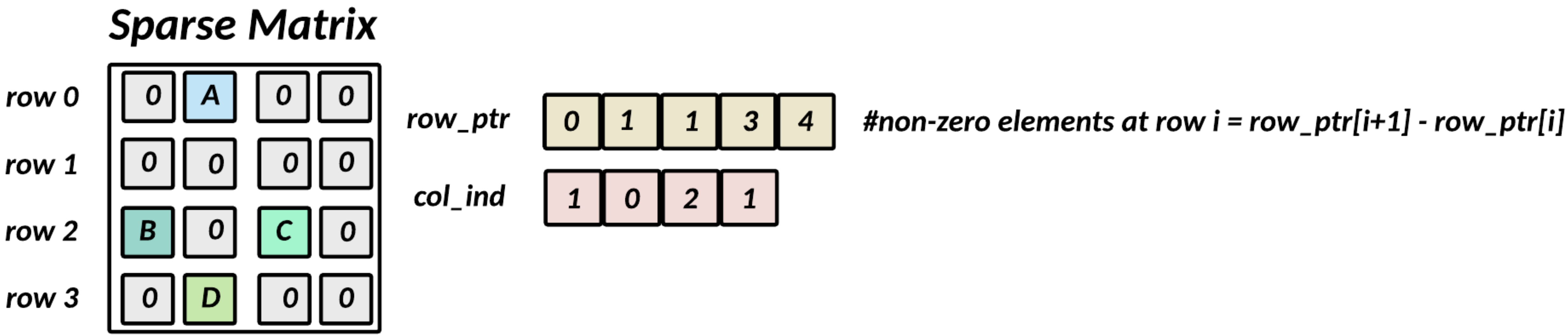
Compressed Sparse Row (CSR)

- **Most widely** used (Intel MKL, OpenBLAS, Eigen, etc.)
- Compressed Sparse Column (**CSC**) follows the same concept



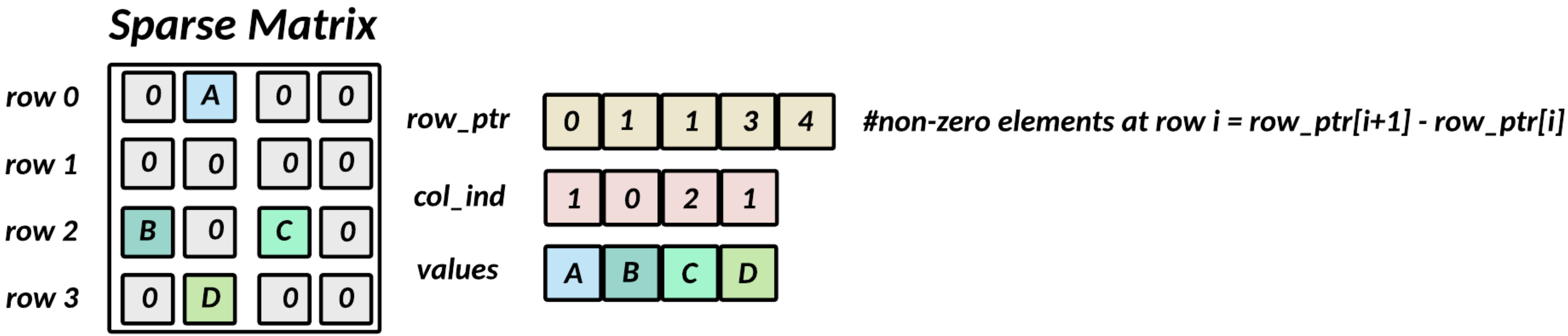
Compressed Sparse Row (CSR)

- **Most widely** used (Intel MKL, OpenBLAS, Eigen, etc.)
- Compressed Sparse Column (**CSC**) follows the same concept



Compressed Sparse Row (CSR)

- **Most widely** used (Intel MKL, OpenBLAS, Eigen, etc.)
- Compressed Sparse Column (**CSC**) follows the same concept



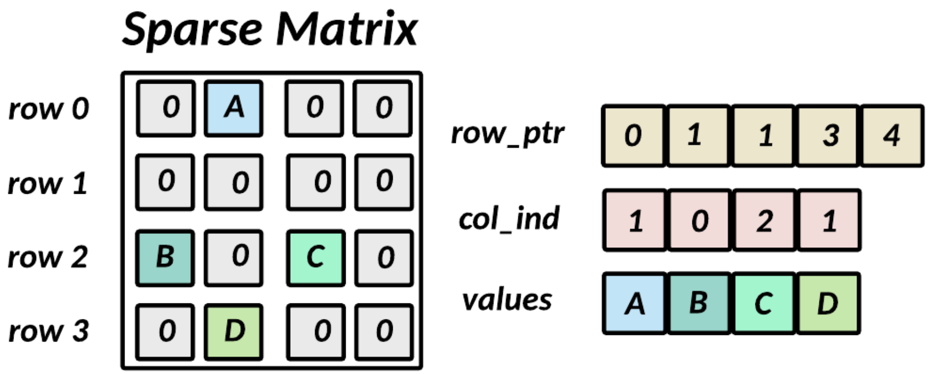
Compressed Sparse Row (CSR)

Advantages

- Enables **spatial locality**
- Only stores **non-zero** values (ignoring meta-data)

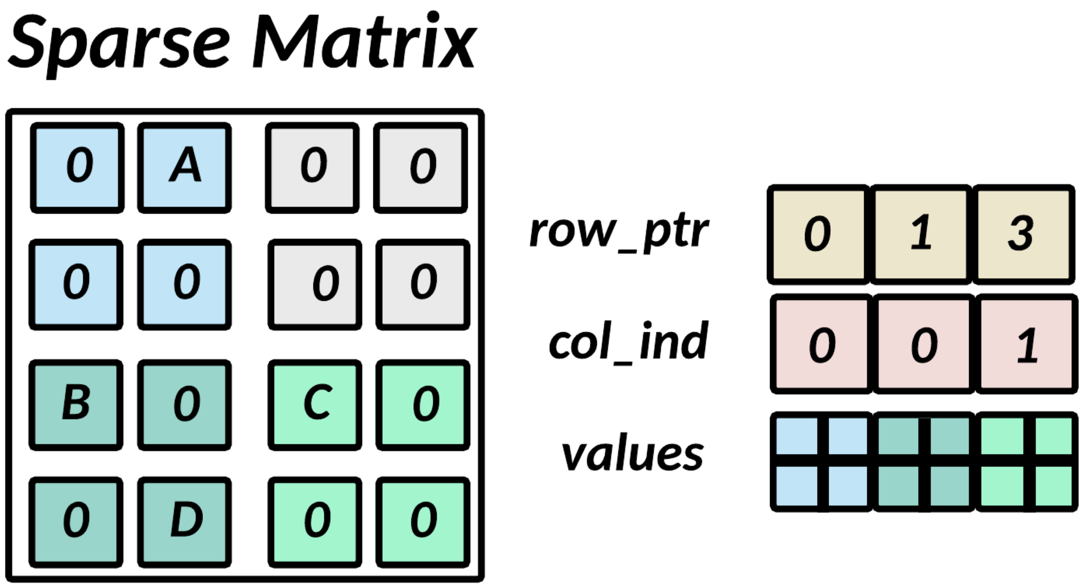
Disadvantages

- **Costly insertion** of non-zero values
- Difficult to obtain **temporal locality**



Block Compressed Sparse Row (BCSR)

- **Modification** of CSR



Block Compressed Sparse Row (BCSR)

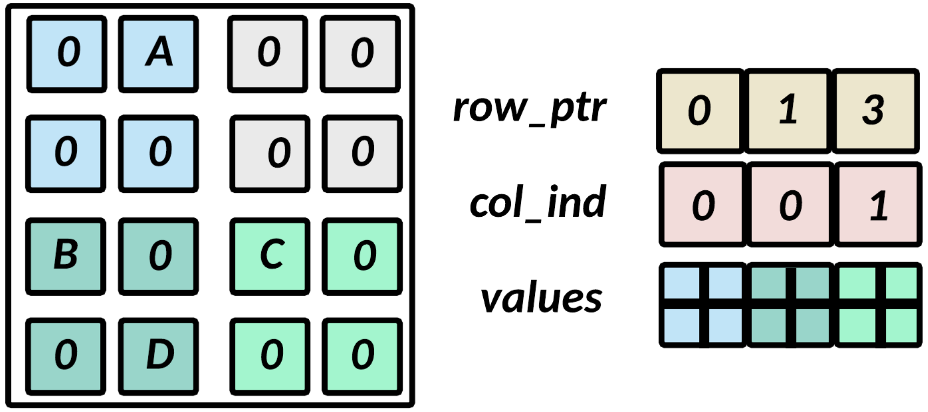
Advantages

- **Reduced storage** for indices
- Enables **blocking optimization** in matrix operations

Disadvantages

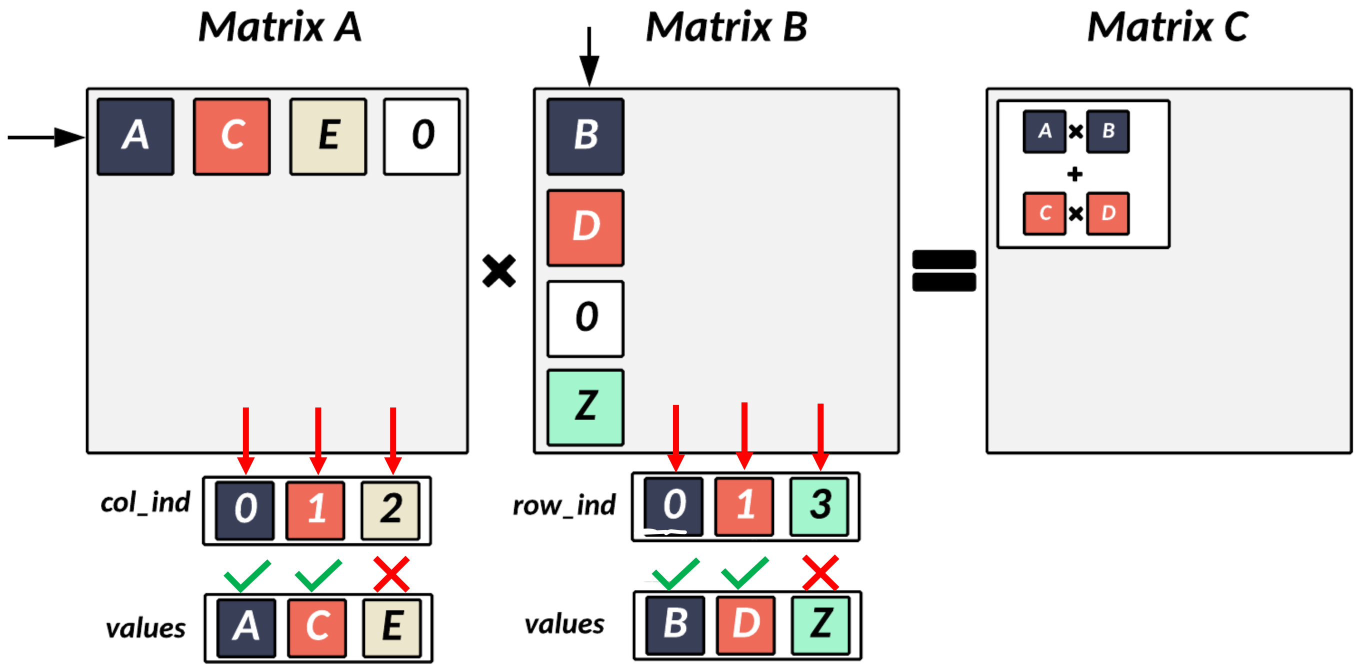
- Storage of **zeros**
- Computational **overhead**

Sparse Matrix



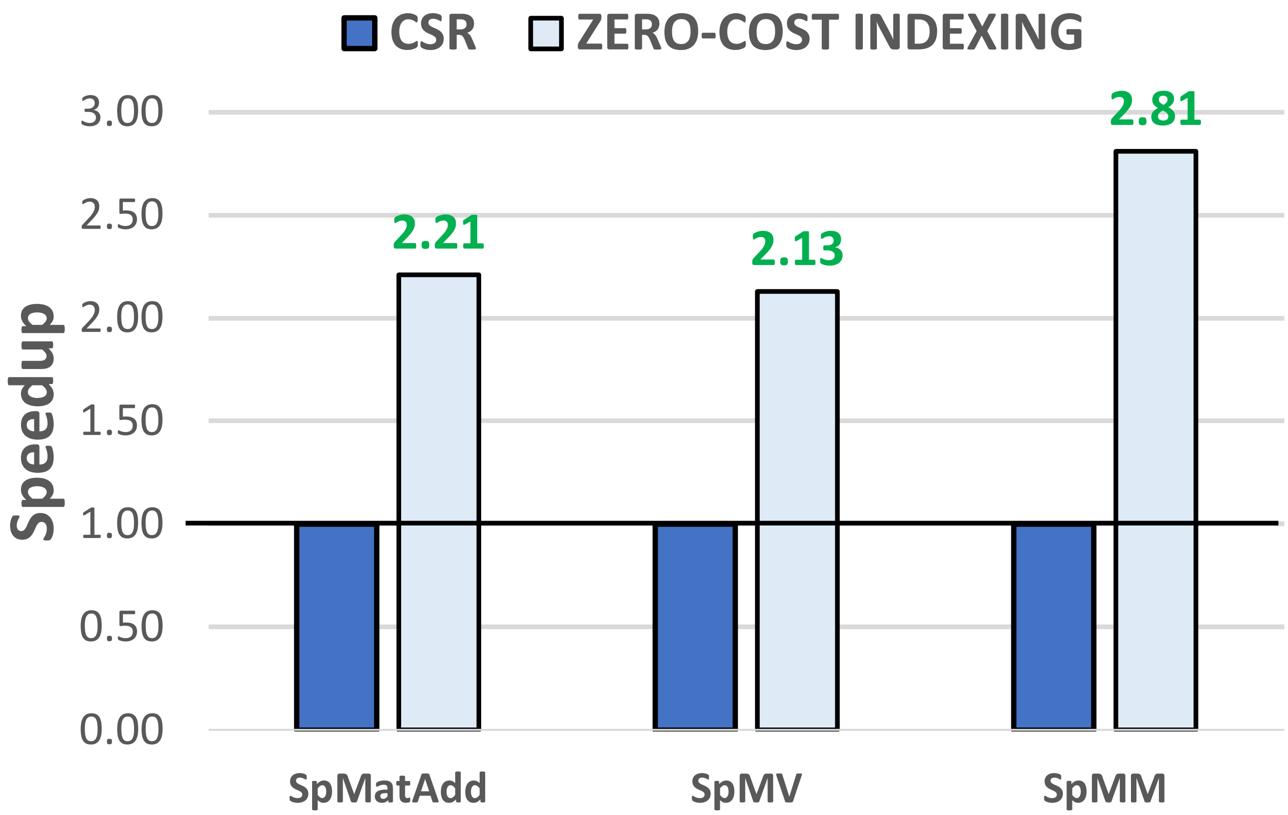
Indexing Overhead of CSR and BCSR

- **Multiple** memory instructions to get **position of non-zero value or block**
- **Comparison** of indices



Zero-cost Indexing for CSR and BCSR

Zero-cost indexing indicates room for improvement

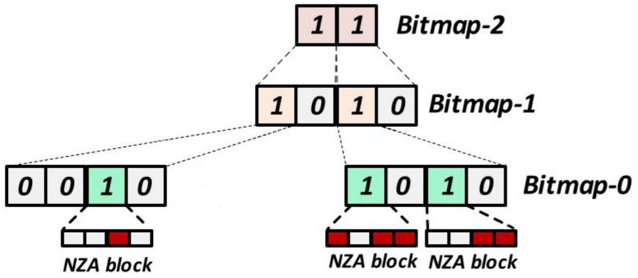


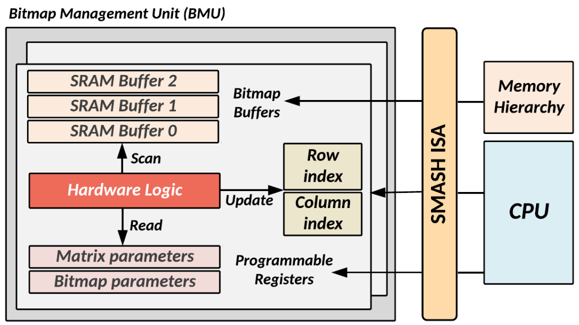
SMASH: A Novel Compression Approach

Hardware/Software Cooperative Mechanism

- Enables highly efficient sparse matrix compression and computation
- Works effectively for a diverse set of sparse matrices

$$\begin{pmatrix} 0&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 1&0&1&1&0&0&0&0\\ 0&0&1&1&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0 \end{pmatrix}$$



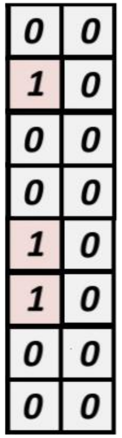


Software Compression

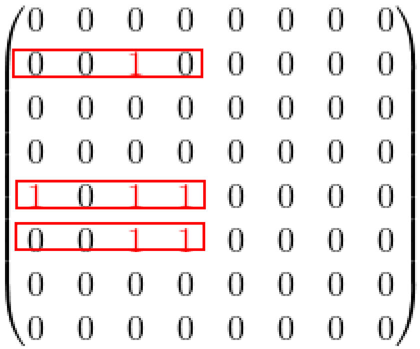
Hardware-accelerated Indexing

SMASH: Software Compression Scheme

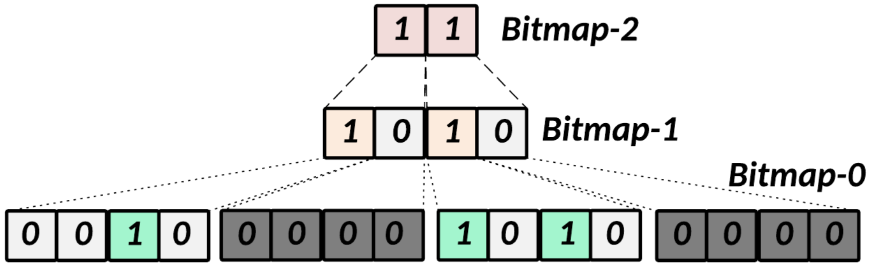
Single Bitmap



Non-zero blocks



Hierarchy of Bitmaps



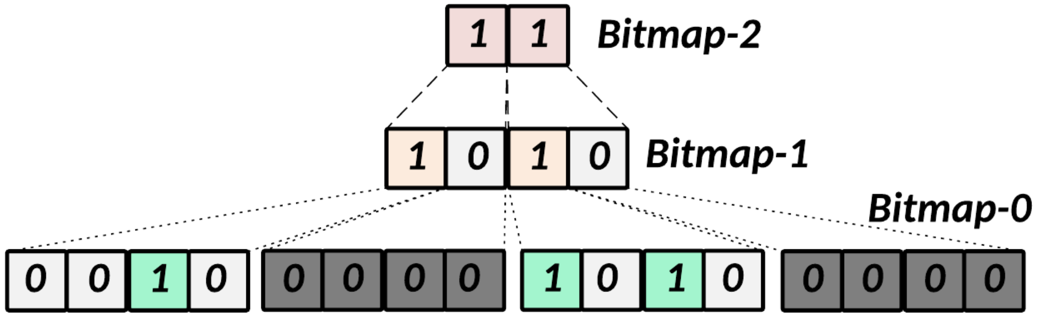
- Bit encodes the existence of a **non-zero** value in a region
- A lot of zero bits

- **Idea:** Apply encoding recursively to reduce the number of zero bits (**bottom-up**)

SMASH: Software Compression Scheme

Levels

- Indicate presence of non-zero values with **different granularity**



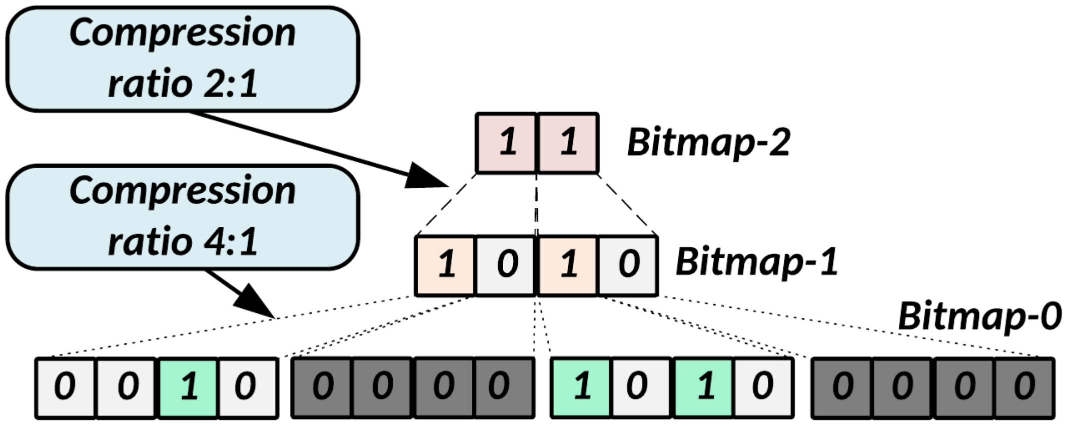
SMASH: Software Compression Scheme

Levels

- Indicate presence of non-zero values with **different granularity**

Compression Ratios

- **Configurable**, determine **change in granularity** between levels



SMASH: Software Compression Scheme

Levels

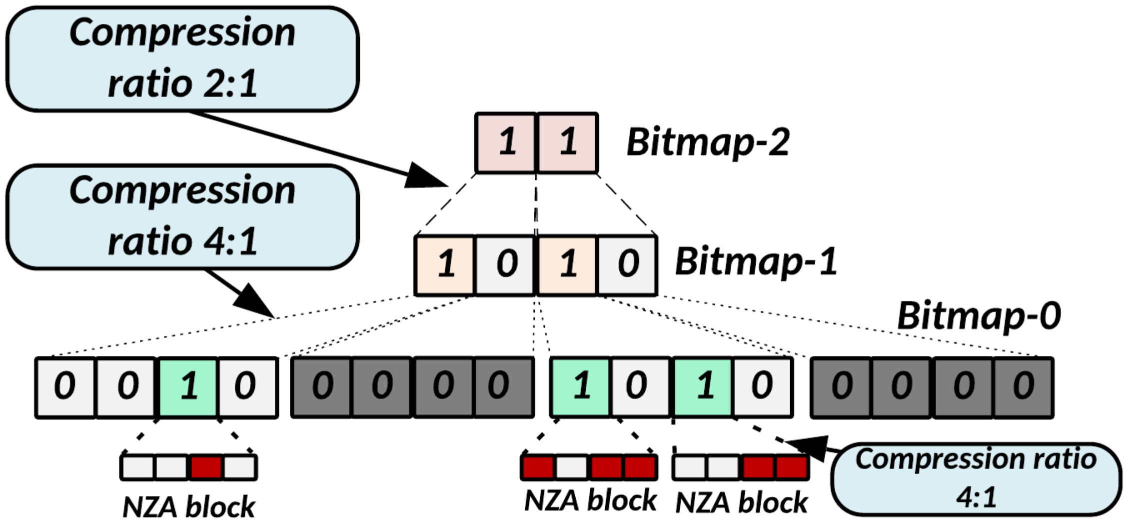
- Indicate presence of non-zero values with **different granularity**

Compression Ratios

- **Configurable**, determine **change in granularity** between levels

Non-zero Array (NZA)

- Stores **non-zero blocks** of matrix



SMASH: Software Compression Scheme

Levels

- Indicate presence of non-zero values with **different granularity**

Compression Ratios

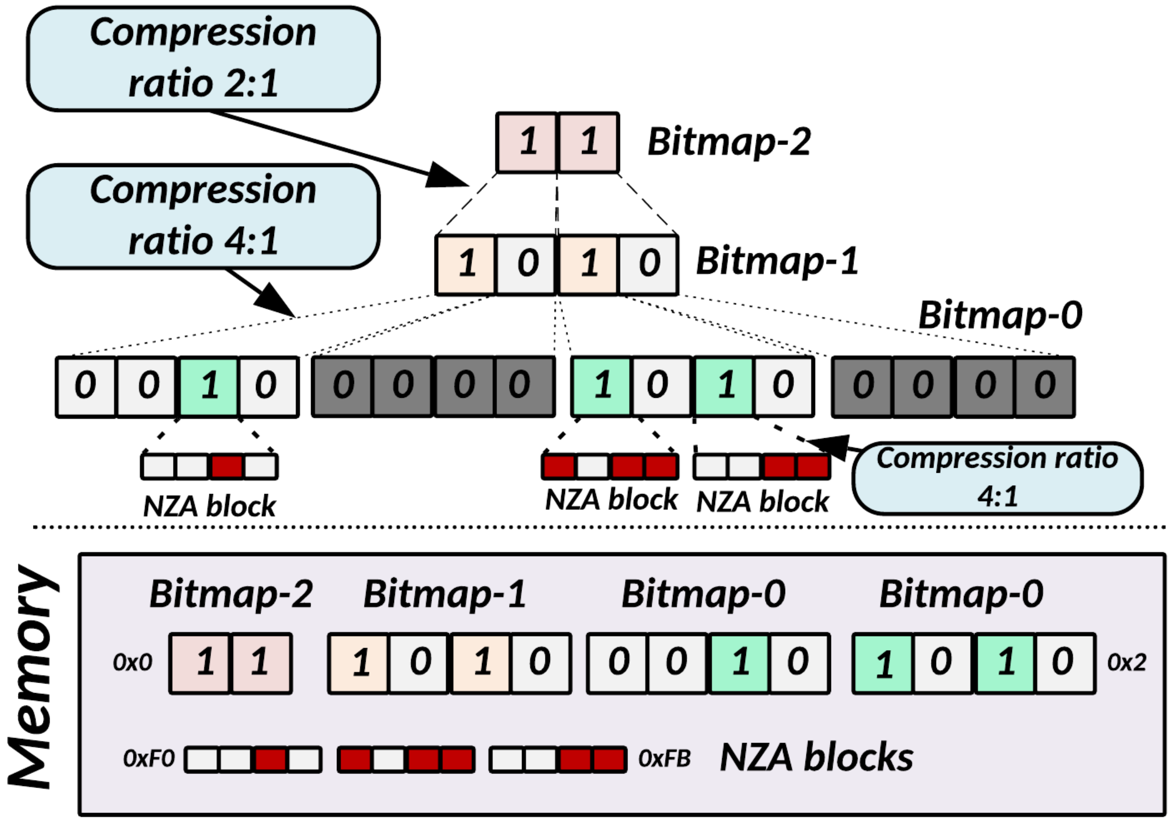
- **Configurable**, determine **change in granularity** between levels

Non-zero Array (NZA)

- Stores **non-zero blocks** of matrix

Representation in Memory

- Only store **necessary** information



SMASH: Software Compression Scheme

Efficient storage

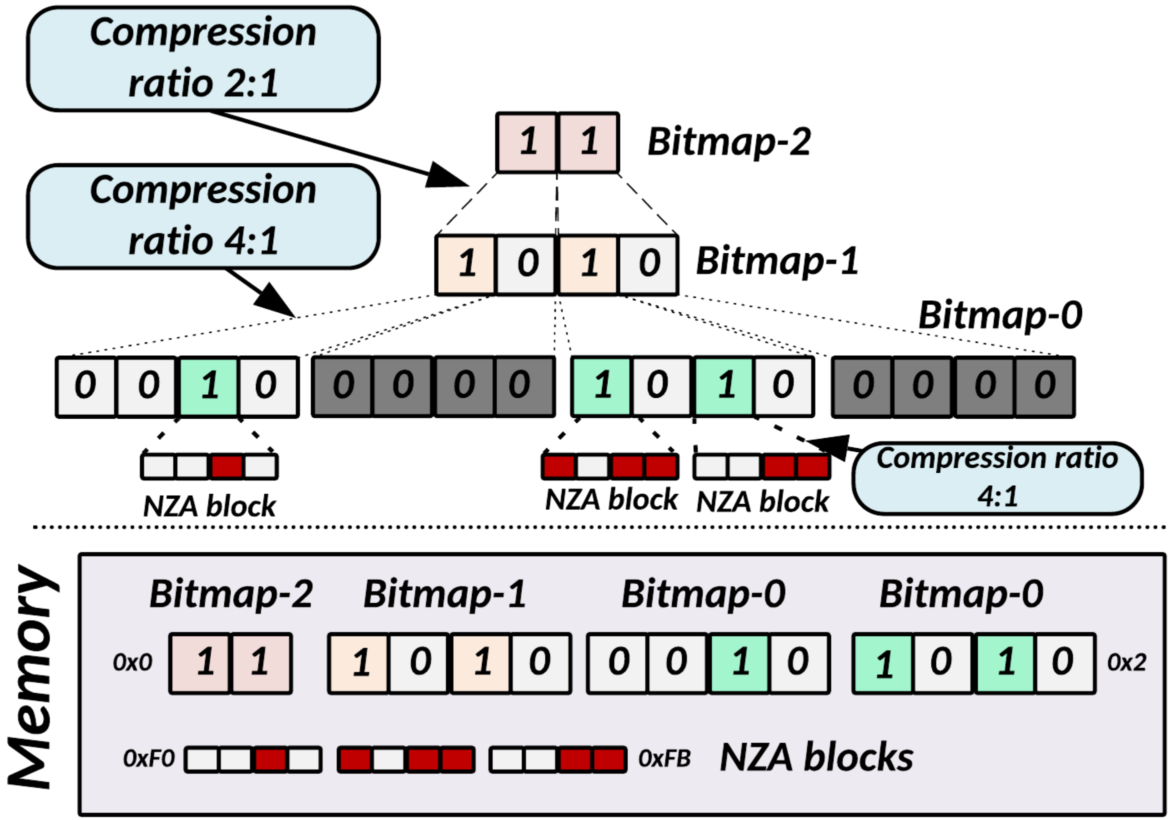
- Each non-zero block has only a few bits (= #levels) of meta-data

Fast indexing

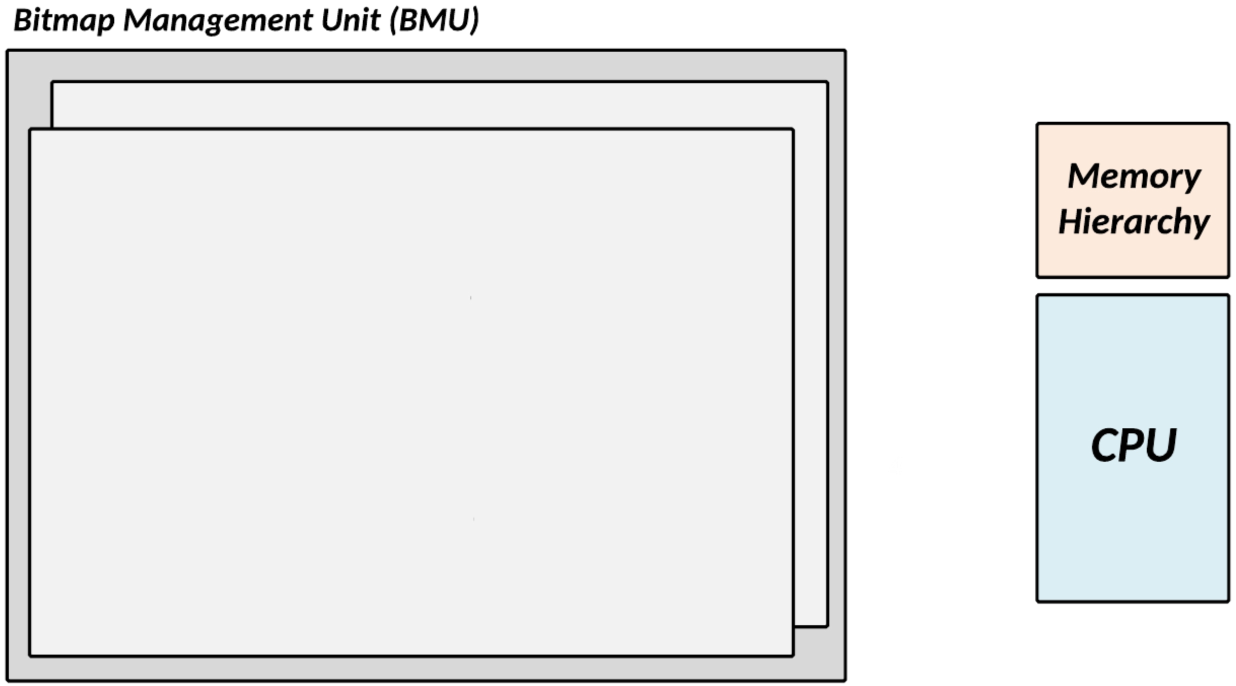
- Traverse bitmap hierarchy in a depth-first manner to compute the index
- Index is a simple sum of products

Enables hardware acceleration

- Little amount of data to transfer
- Facilitated by hierarchical structure

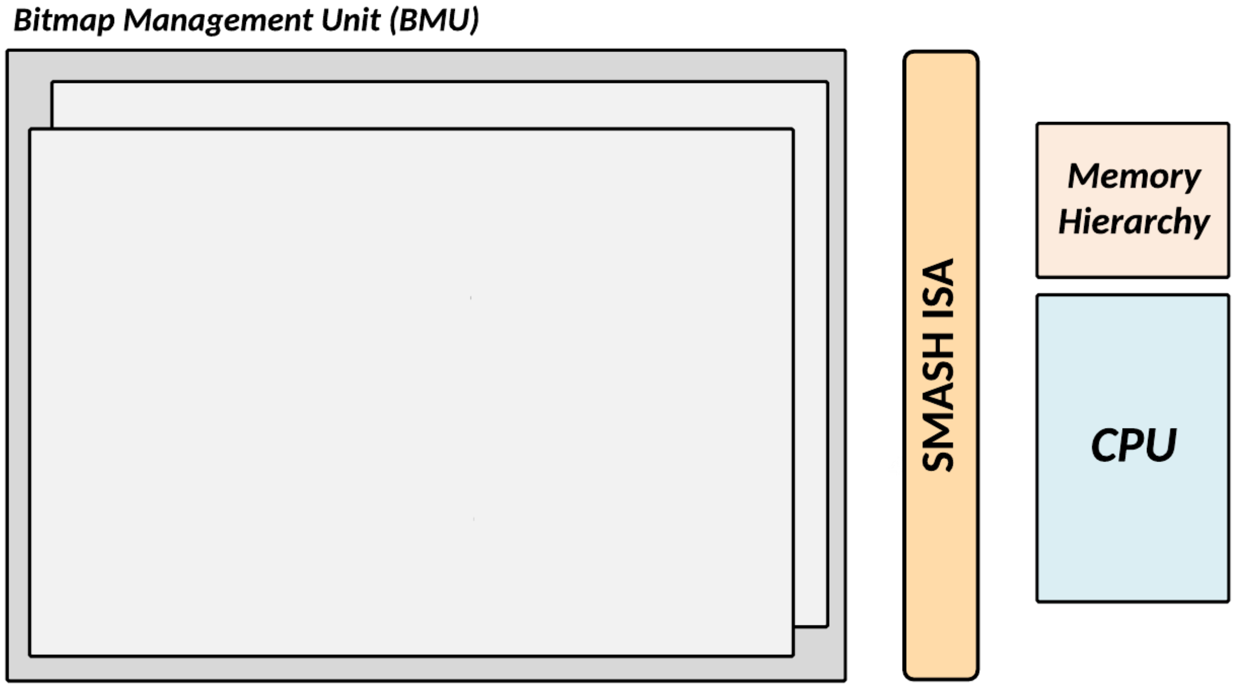


SMASH: Hardware Acceleration Scheme



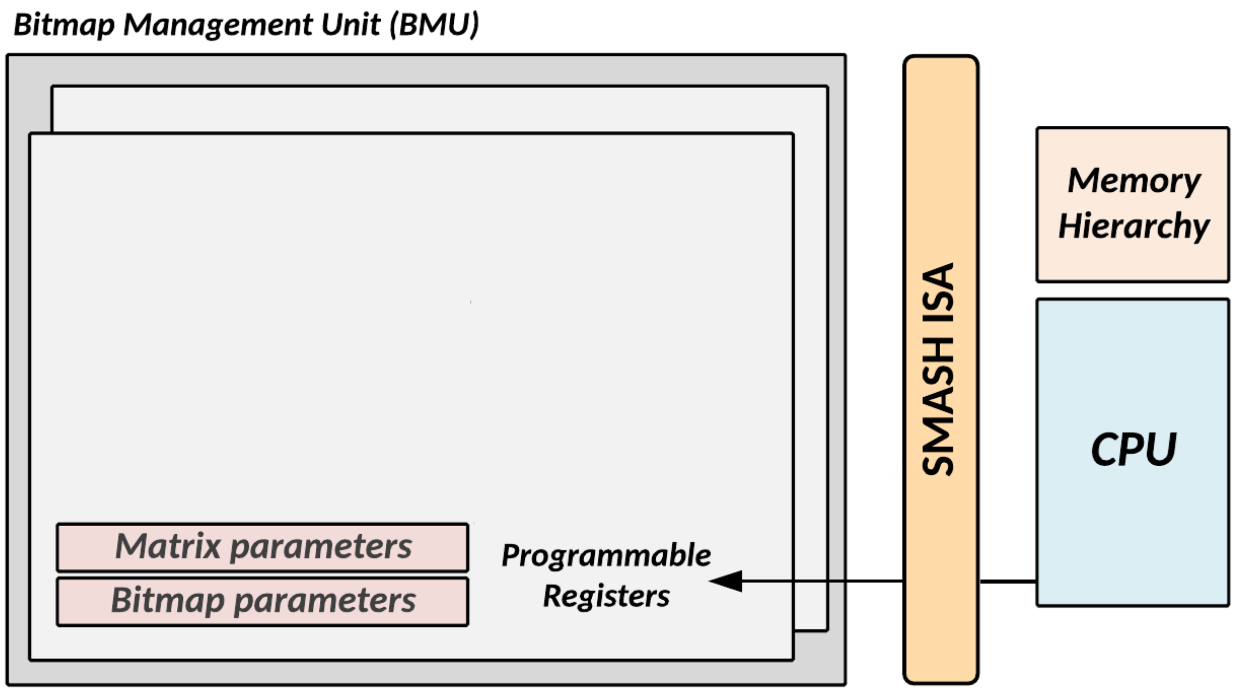
SMASH: Hardware Acceleration Scheme

- Communication over **SMASH ISA**



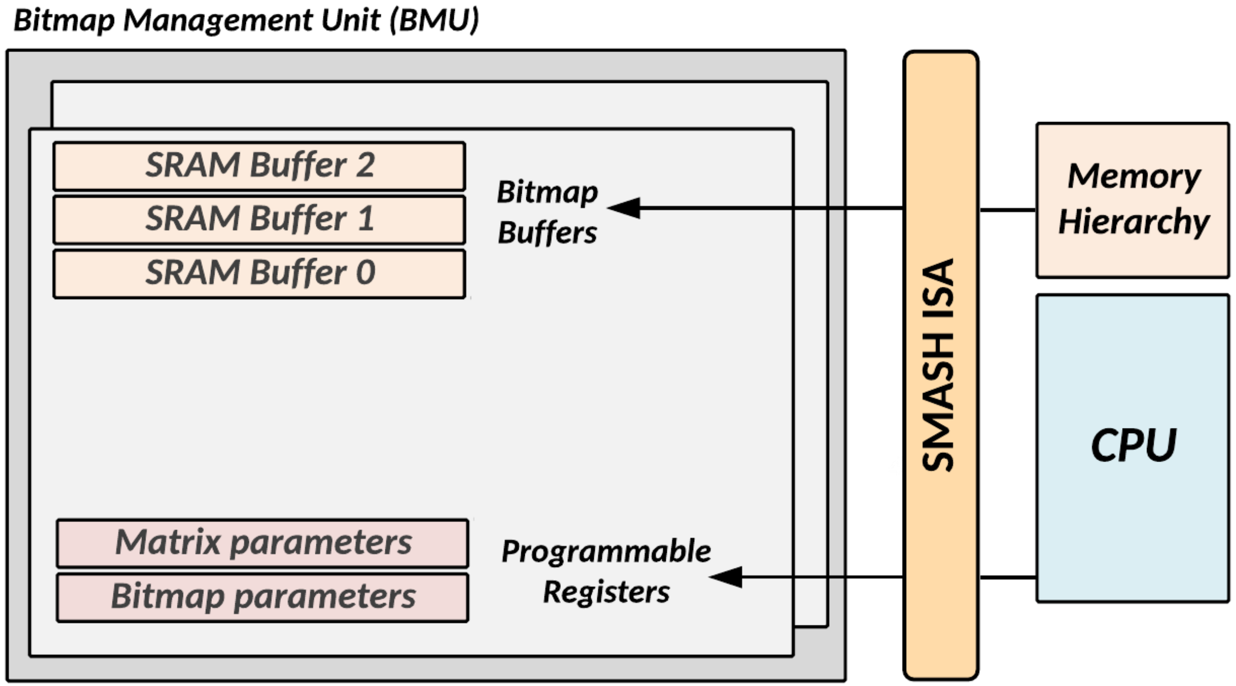
SMASH: Hardware Acceleration Scheme

- Communication over **SMASH ISA**
- Matrix size and compression ratios as **parameters**



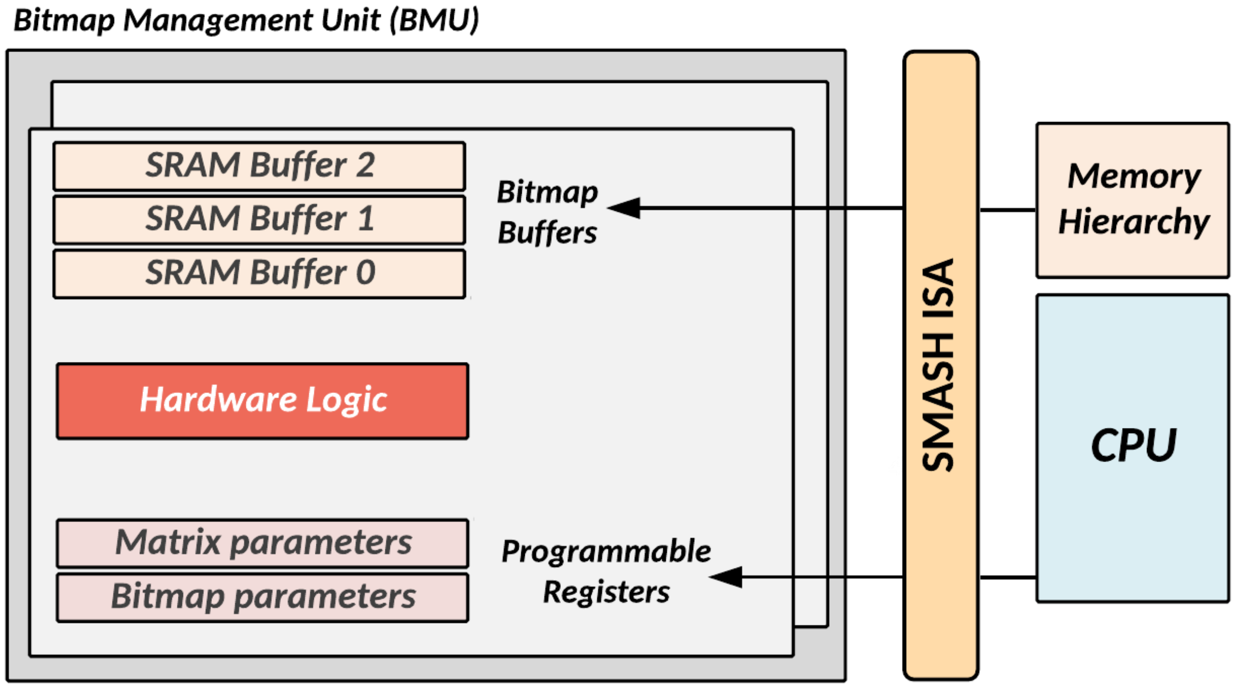
SMASH: Hardware Acceleration Scheme

- Communication over **SMASH ISA**
- Matrix size and compression ratios as **parameters**
- One **buffer** per bitmap level



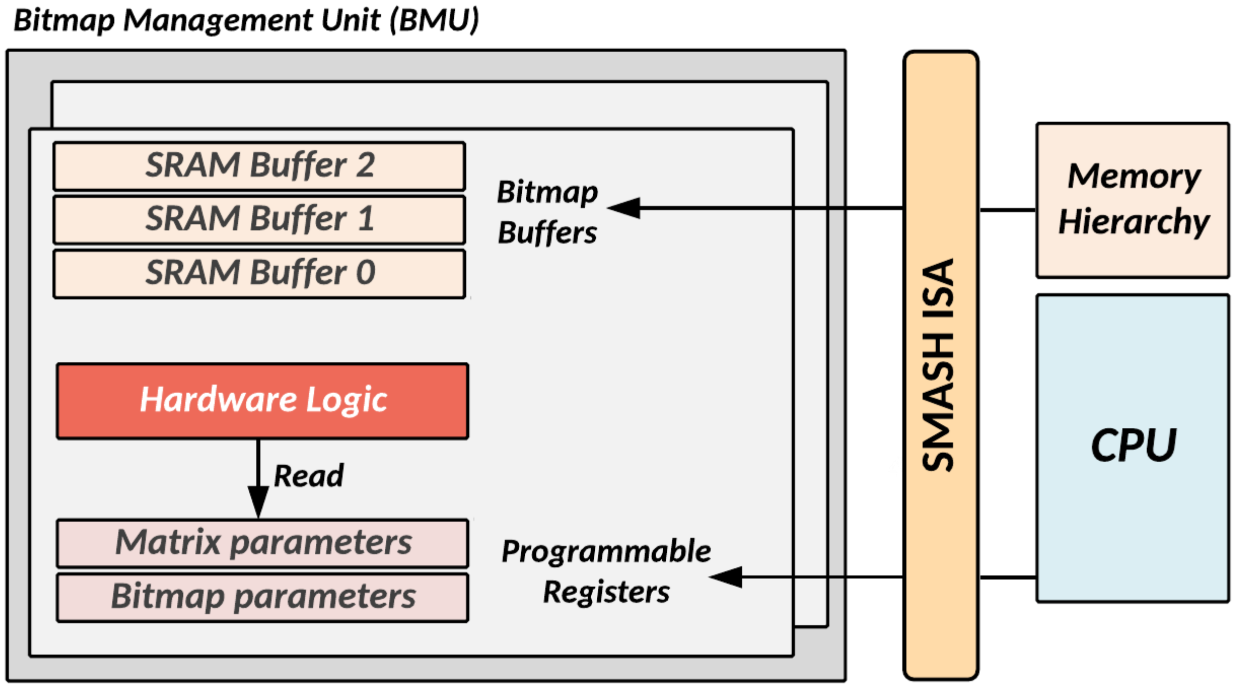
SMASH: Hardware Acceleration Scheme

- Communication over **SMASH ISA**
- Matrix size and compression ratios as **parameters**
- One **buffer** per bitmap level
- Hardware logic:



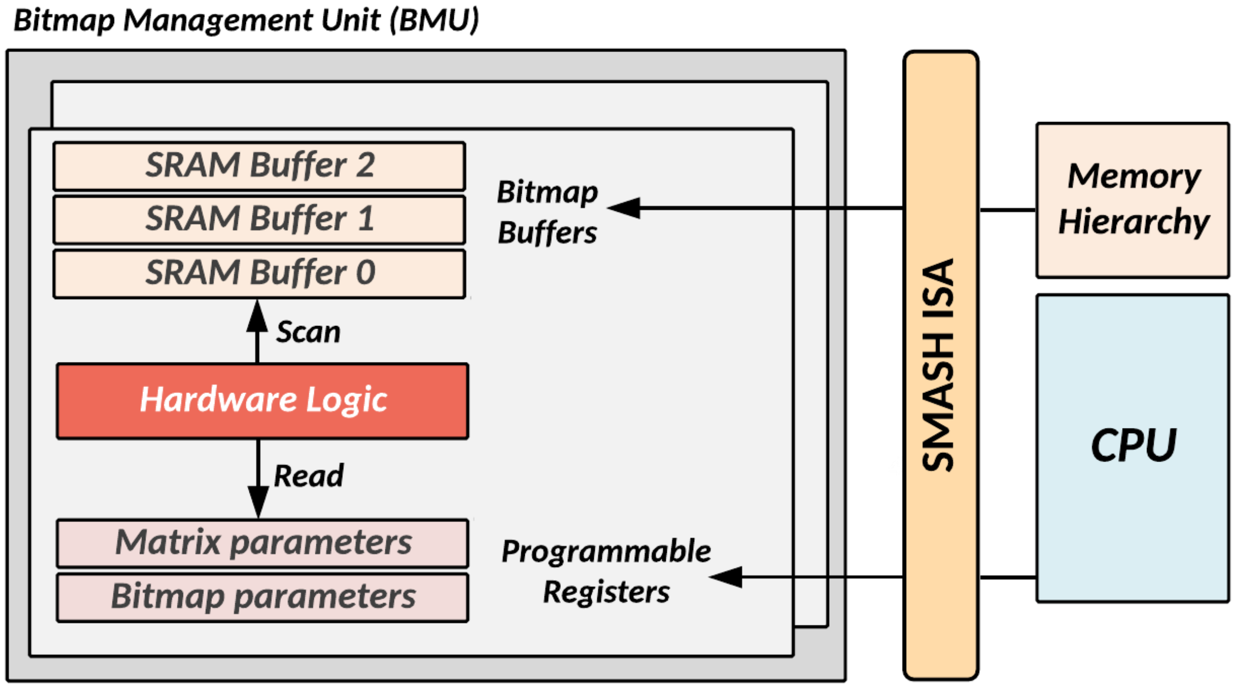
SMASH: Hardware Acceleration Scheme

- Communication over **SMASH ISA**
- Matrix size and compression ratios as **parameters**
- One **buffer** per bitmap level
- Hardware logic:
  - **Reads** parameters



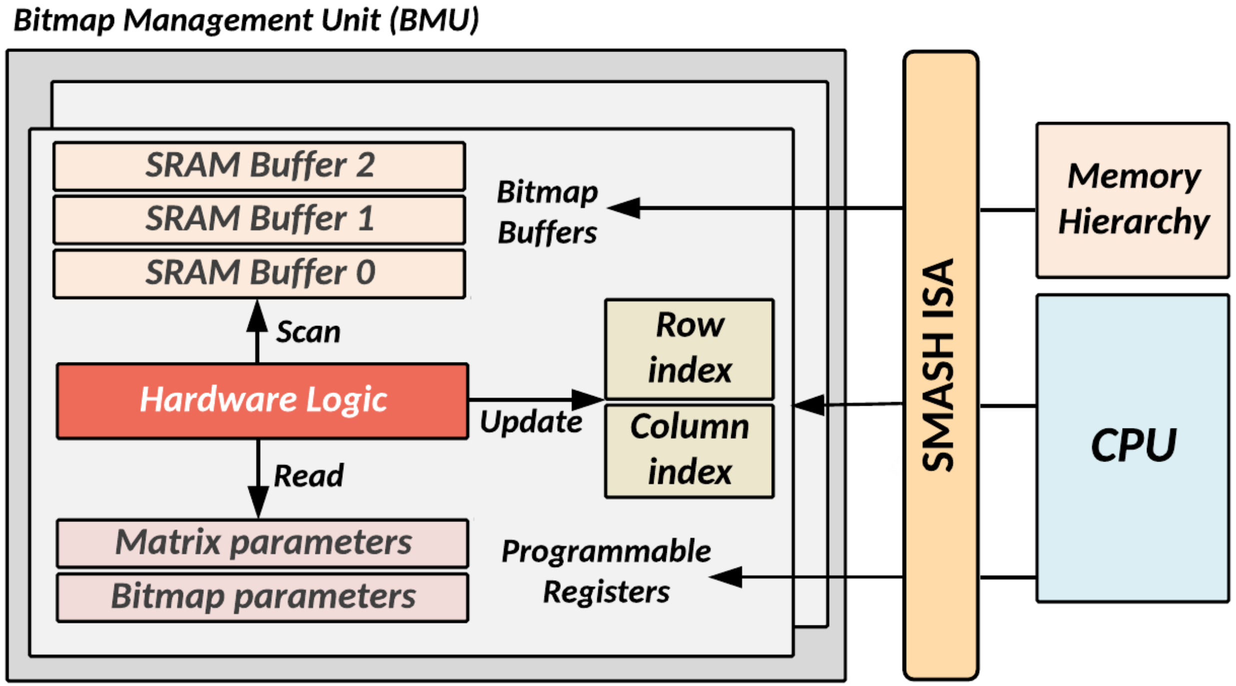
SMASH: Hardware Acceleration Scheme

- Communication over **SMASH ISA**
- Matrix size and compression ratios as **parameters**
- One **buffer** per bitmap level
- Hardware logic:
  - **Reads** parameters
  - **Scans** bitmaps for set bits



SMASH: Hardware Acceleration Scheme

- Communication over **SMASH ISA**
- Matrix size and compression ratios as **parameters**
- One **buffer** per bitmap level
- Hardware logic:
  - **Reads** parameters
  - **Scans** bitmaps for set bits
  - **Computes** index of next **non-zero block** and **stores** it into registers



Evaluation Methodology

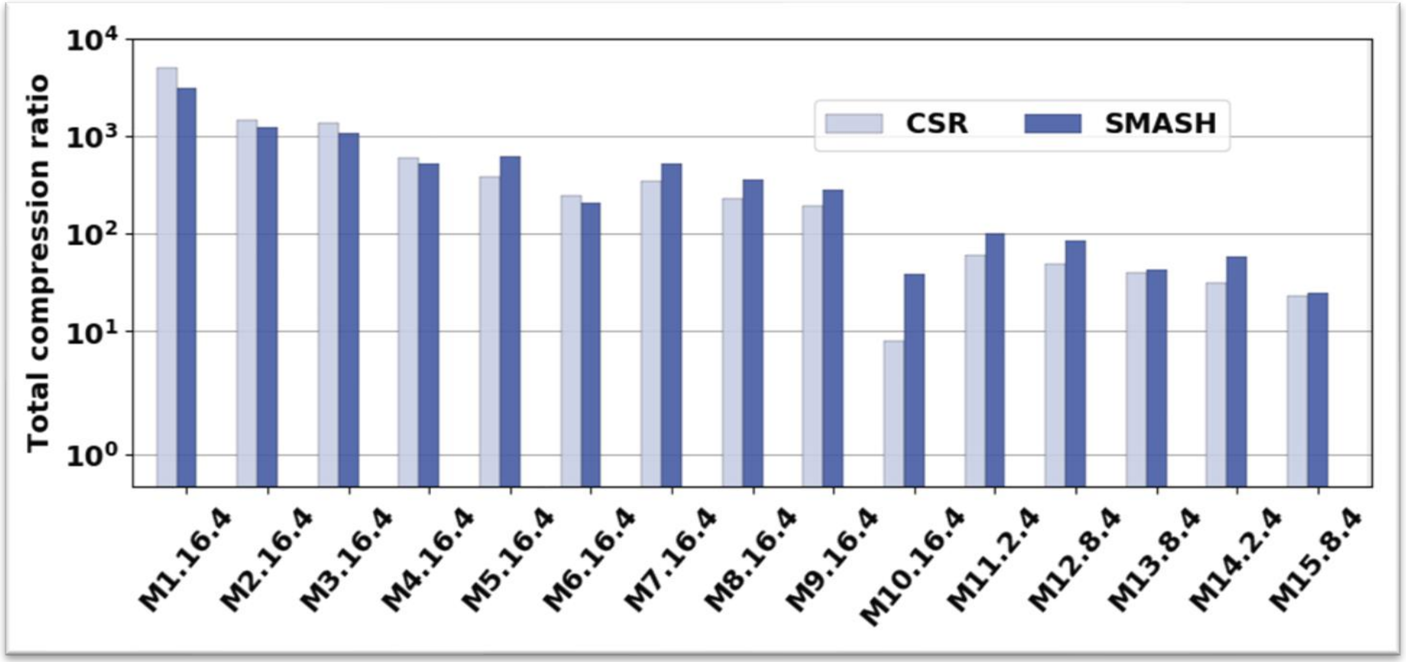
Simulator: ZSim Simulator

Workloads:

- **SpMV & SpMM**
  - 15 matrices
  - **Varied sparsity levels and non-zero distributions**
  - Sparsity level from 0.01% to 8.79%
- **PageRank & Betweenness Centrality**
  - 4 graphs
  - Number of edges from 1M to 3.3M

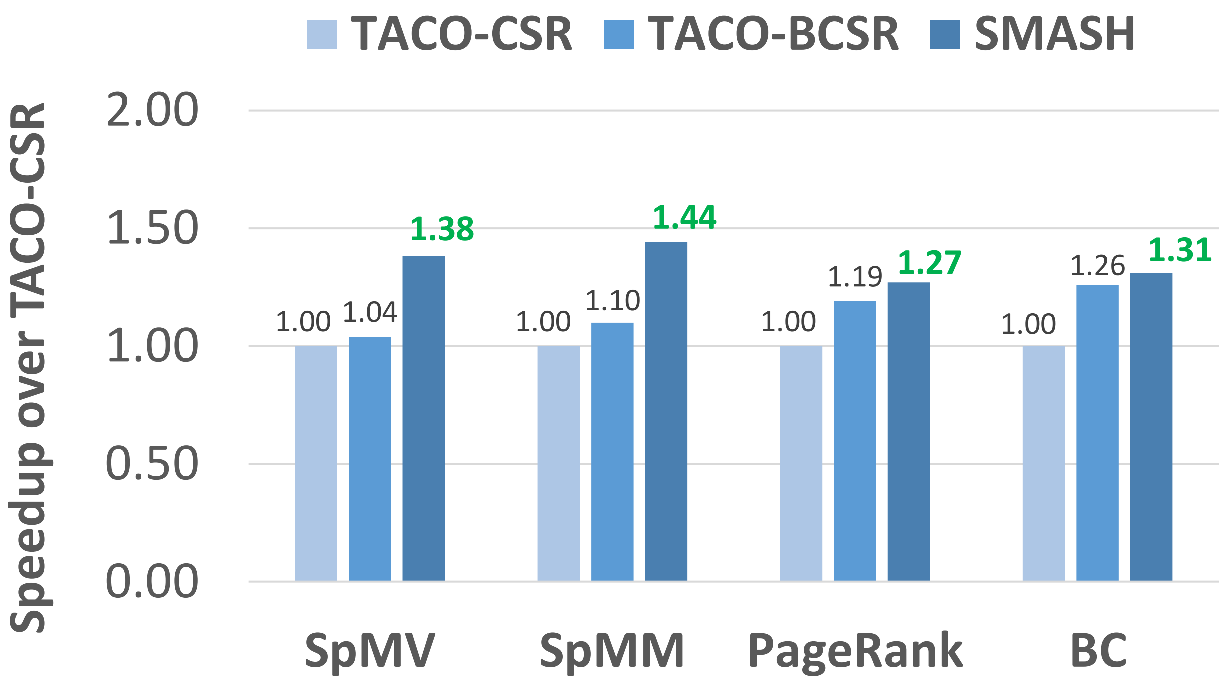
Evaluation Result: Storage Efficiency

1. **Storage** Efficiency
   - **CSR** is better for **very sparse** matrices
   - **SMASH** is better for **denser** matrices
   - Cost of storing **zeros** vs. Cost of storing **indices**



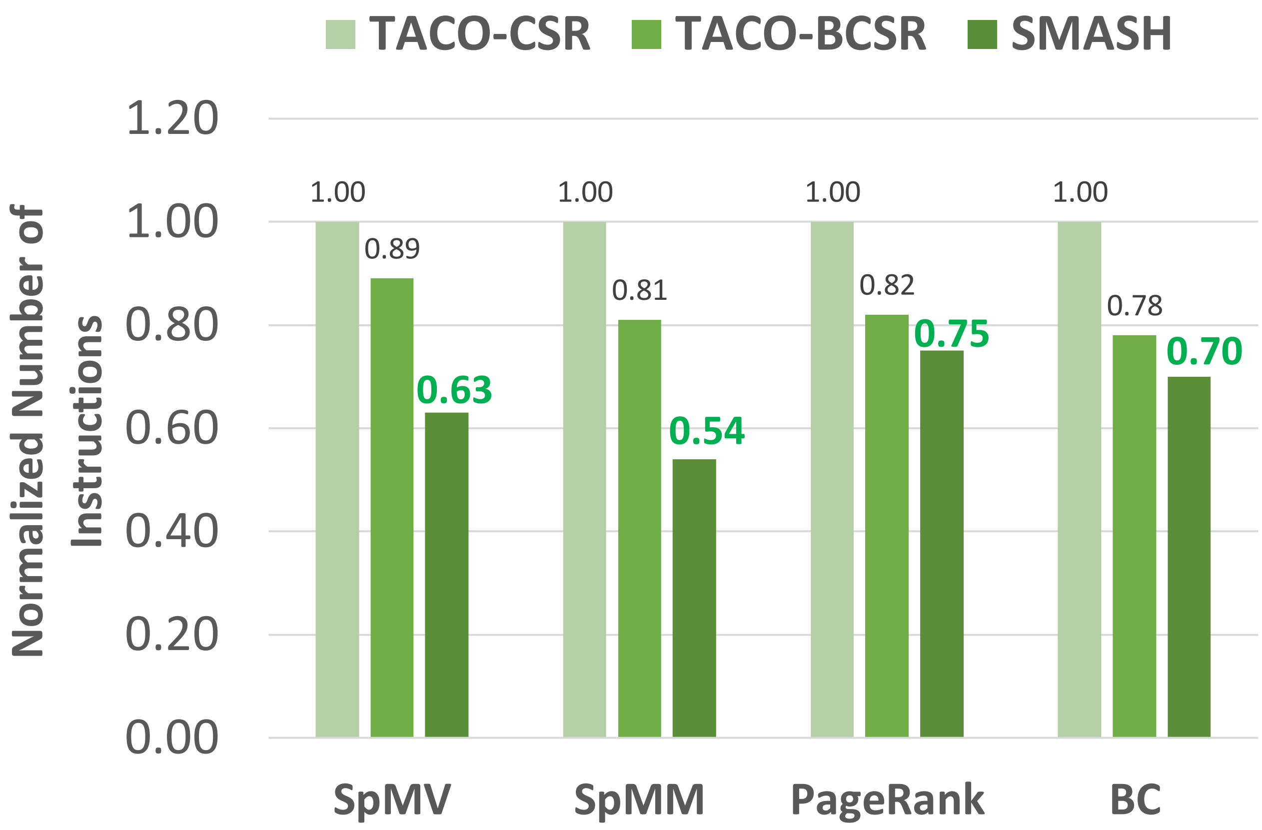
Evaluation Result: Computational Efficiency

1. **Storage** Efficiency
2. **Computational** Efficiency
   - Sparse Matrix Kernels
     - 40 % speedup compared to **CSR**
     - 30% speedup compared to **BCSR**
   - Lower speedup for graph computations



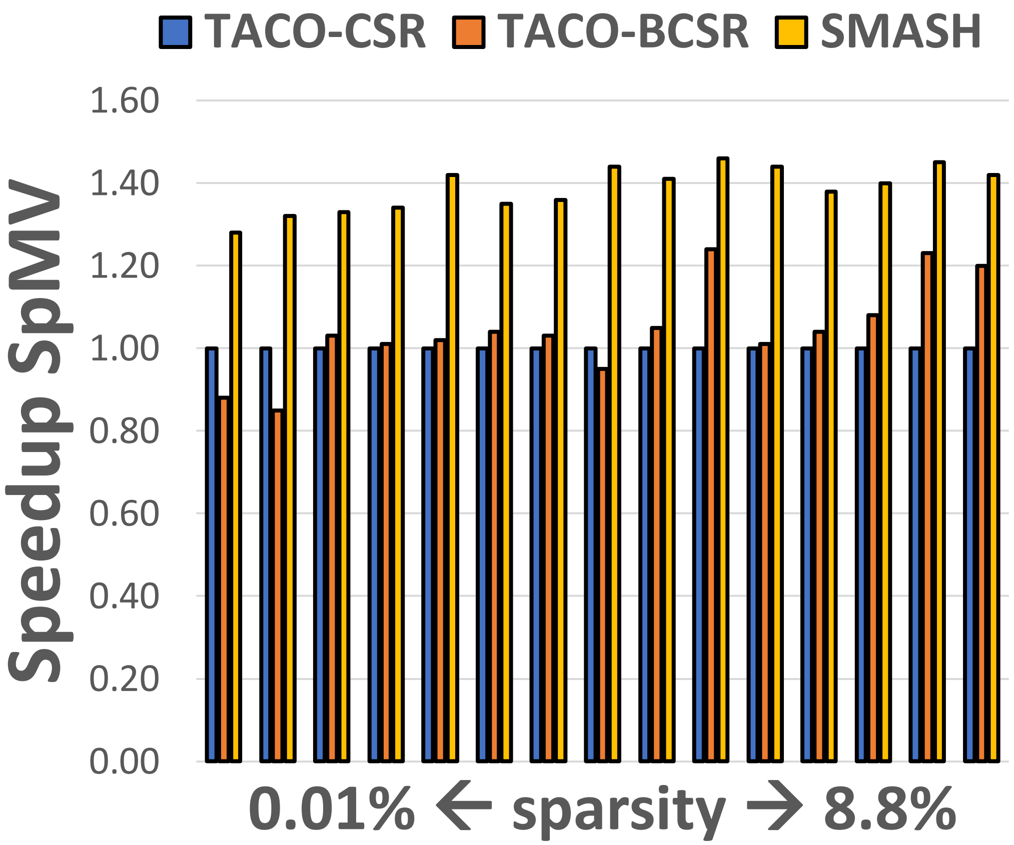
Evaluation Result: Computational Efficiency

1. **Storage** Efficiency
2. **Computational** Efficiency
   - SMASH requires fewer instructions compared to **CSR** and **BCSR**



Evaluation Result: General Applicability

1. **Storage** Efficiency
2. **Computational** Efficiency
3. **General Applicability**
   - Matrices with varied size, structure, and sparsity
   - Increasing the density of a matrix leads to higher indexing overhead

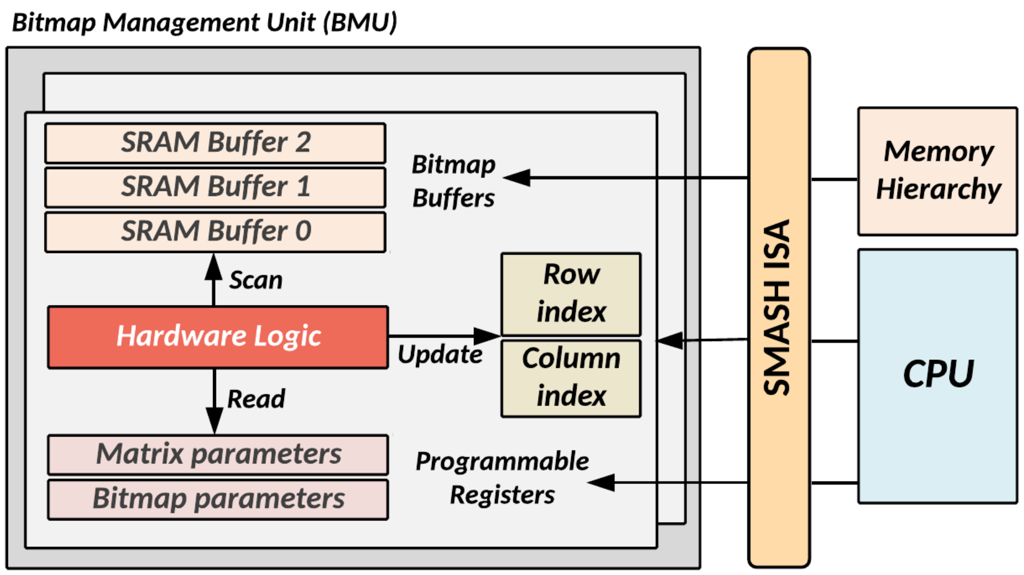


Evaluation Result: Die Area Overhead

BMU

- Support **4 matrices**
- **3 bitmap buffers** per matrix, each with 256 bytes of storage
- 140 bytes for **registers & counters**

0.076% area overhead over an Intel Xeon E5-2698 CPU core



Conclusion

- Many applications heavily rely on **sparse linear algebra**
- They require **effective compression formats** to mitigate **computational and storage overheads**
- Existing **formats** suffer from the **expensive cost** of finding the **position of non-zero elements**
- **SMASH**: **Hardware/Software cooperative mechanism** for **efficient non-zero elements discovery** to accelerate **sparse matrix operations**
  - **Software**: Efficient compression with a **hierarchy of bitmaps**
  - **Hardware**: Scans bitmaps to **find indices of non-zero elements**
- **38 - 44 %** faster than the state-of-the-art for **SpMV and SpMM**
- SMASH is **highly effective, low cost, and widely applicable**

Strengths

- Identifies **indexing as a key bottleneck** in sparse matrix compression schemes
- Among the first to propose a **hardware/software cooperative scheme** to accelerate sparse matrix indexing
- **General applicable**
  - CPU, GPU, other hardware accelerators etc.
  - Any sparse operation like Sparse LU, Sparse QR, etc.
- **Extensive evaluation** from many perspectives
  - Impact of compression ratio, sparsity levels, non-zero distribution, conversion overhead, software-only approach, die area overhead, etc.
- SMASH's implementation of the matrix kernels is **open source**
- **Well written**
  - Little **prior knowledge** required
  - Structure feels **very familiar…**

Weaknesses

- Finding of **optimal compression ratios** might be complicated
- Selection of **number of bitmap levels** is not explained
- Evaluation of **hardware-accelerated SMASH** only in a **simulator**
- **Dynamic updates** of sparse matrix is expensive
  - Requires reload of bitmaps into BMU to avoid coherency issues
  - Dynamic insertion of a non-zero value might require a large copy operation on the non-zero array

Thoughts, Ideas and Alternatives

- SMASH stores zero-containing **blocks** like BCSR, but only as **1D arrays**
  - Partitioning of matrix into **2D regions** would enable **blocking optimization**

0	0
1	0
0	0
0	0
1	0
1	0
0	0
0	0

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

0	1	0	0
0	0	0	0
1	1	0	0
0	0	0	0

- **Lossy compression formats** that approximate matrix values to reduce the memory footprint
  - CSR: Merge integer index (4 bytes) and float value (4 bytes) into a single value (4 bytes)
  - Value stores **index in the leftmost bits** and **value in the remaining bits**
  - **Loss of precision** for floating points values
  - **Loss of range** for integer indices

Thoughts, Ideas and Alternatives

SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks [Gondimalla et al., 2019]

- Acceleration of **sparse dot products** which are heavily used in CNN
- Representation of sparse vectors with **bit masks to indicate position of non-zero values**
- **Fast computation of the intersection** between non-zero value indices
- Authors note that **sparsity in machine learning models is lower than in other applications** (10% sparsity and more).
- They provide an analysis on when bitmasks are more efficient than conventional formats like CSR.

Thoughts, Ideas and Alternatives

ExTensor: An Accelerator for Sparse Tensor Algebra
[Hegde et al., 2019]

- Acceleration of **sparse linear algebra**
- Representation of sparse operands as **hierarchical trees**
- **Fast computation of the intersection** between non-zero element nodes or non-zero region subtrees to **avoid unnecessary scalar multiplications or even dot products**

Thoughts, Ideas and Alternatives

SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training [Qin et al., 2020]

- Acceleration of **sparse matrix-matrix multiplications** in DNN
- Showcase of the **importance** of matrix-matrix multiplications in **deep learning**
- Explanation on why **GPUs and TPUs cannot exploit sparsity** effectively
- As part of their architecture, they use a **bitmap format** to store operands
    - **Constant meta-data overhead** by using one bit per element
    - For denser matrices, bitmaps have **lower overhead** compared to **conventional formats**

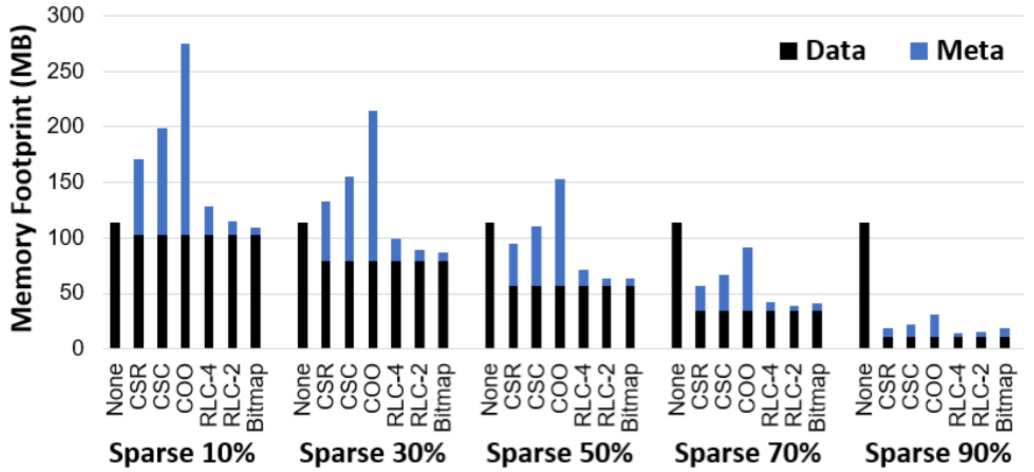


Figure 7: Matrix memory overhead with dimensions M=1632 and K=36548. Format comparisons include: None, CSR, CSC, COO, RLC-4, RLC-2, and Bitmap in the following order.

Key Takeaways

- It is essential to take the **sparsity of matrix operands** into account, if one wants to enable **applications** that operate on **very large and sparse matrices**
- While it takes more time to detect the **root of a problem**, resolving a problem at that level **can benefit a lot of** applications
    - Today's root: Sparse matrix indexing
- One can **benefit from orthogonal concepts of other approaches**
    - BCSR's blocking optimization has the potential to improve SMASH even further

Questions?

Discussion

- How does the **distribution of non-zero elements** affect SMASH?
- How can we reduce the **overhead of zero element computations**?
- How can we use SMASH to accelerate **parallelized matrix computations**?
- Where should we employ **approximation** to reduce the indexing overhead?
  - Directly in the compression format?
  - Approximate memory?
  - Other methods?
- Which **concepts from previous presentations** can be used to reduce the indexing overhead?
- Can you think of any other applications that can benefit from the **use of hierarchical bitmaps**?

Reducing Overhead of Zero Element Computations

Page Overlays:
An Enhanced Virtual Memory Framework to Enable Fine-grained Memory Management
[Seshadri et al., 2015]

- Can significantly boost **system performance and efficiency** while largely retaining the structure of the **existing virtual memory framework**
- Enables the mapping of virtual pages to **overlays** which only **contain a few cache lines**
- With overlays the fetching of and the computation with **zero-only cache lines of SMASH's non-zero blocks** can be avoided
- Can be used to enable **more efficient sparse matrix updates**

Reducing Overhead with Approximate Memory

EDEN:
Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM
[Koppula et al., 2019]

- Nina Richter will present this paper next week, so I will not spoil her presentation today ☺