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

Presented at: MICRO 2019
Executive Summary

- Many important workloads heavily make use of **sparse matrices**
  - Matrices are **extremely sparse**
  - **Compression** is essential to avoid storage/computational overheads
- Shortcomings of existing compression formats
  - Expensive discovery of the positions of non-zero elements or
  - Narrow applicability
- **SMASH**: hardware/software cooperative mechanism for efficient sparse matrix storage and computation
  - **Software**: Efficient compression scheme using a hierarchy of bitmaps
  - **Hardware**: Hardware unit interprets the bitmap hierarchy and accelerates indexing
- Performance improvement: **38% and 44%** for SpMV and SpMM over the widely used CSR format
- SMASH is **highly efficient, low-cost and widely applicable**
1. Sparse Matrix Basics
2. Compression Formats & Shortcomings
3. SMASH
4. Evaluation
5. Conclusion
Sparse Matrix Operations are Widespread Today

Recommender Systems

Graph Analytics

Neural Networks

• Collaborative Filtering

• PageRank
• Breadth First Search
• Betweenness Centrality

• Sparse DNNs
  Graph Neural Networks
Real World Matrices Have High Sparsity

0.0003% non-zero elements

2.31% non-zero elements

Sparse matrix compression is essential to enable efficient storage and computation
## Presentation Outline

1. Sparse Matrix Basics
2. Compression Formats & Shortcomings
3. SMASH
4. Evaluation
5. Conclusion
Limitations of Existing Compression Formats

1. General formats optimize for storage

   →

   Expensive discovery of the positions of non-zero elements
Widely Used Format: Compressed Sparse Row

• Used in multiple libraries & frameworks: Intel MKL, TACO, Ligra, Polymer

• Stores only the non-zero elements of the sparse matrix

• Provides high compression ratio
Basics of CSR: Indexing Overhead

Multiple instructions are needed to discover the position of a non-zero element

Requires multiple data-dependent memory accesses
Indexing Overhead in Sparse Kernels

**Sparse Matrix Vector Multiplication (SpMV)**

Indexing for every non-zero element of the sparse matrix to multiply with the corresponding element of the vector.

**Sparse Matrix Matrix Multiplication (SpMM)**

Index matching for every inner product between the 2 sparse matrices.

Indexing is expensive for major sparse matrix kernels
Indexing in SpMV

**CSR-based SpMV**

SpMV requires indexing for every non-zero element
Indexing in SpMM

SpMM requires indexing for every inner product
Performing Indexing with Zero Cost

Reducing the cost of indexing can accelerate sparse matrix operations.
Limitations of Existing Compression Formats

1. General formats optimize for storage → Expensive discovery of the positions of non-zero elements

2. Specialized formats assume specific matrix structures and patterns (e.g., diagonals) → Narrow applicability
Our Goal

Design a sparse matrix compression mechanism that:

• **Minimizes** the indexing **overheads**

• Can be used across a **wide range** of sparse matrices and sparse matrix operations

• Enables **high compression ratio**
Presentation Outline

1. Sparse Matrix Basics
2. Compression Formats & Shortcomings
3. SMASH
   - Software Compression Scheme
   - Hardware Acceleration Unit
   - Cross-Layer Interface
4. Evaluation
5. Conclusion
SMASH: Key Idea

Hardware/Software cooperative mechanism:
• Enables **highly-efficient** sparse matrix compression and computation
• **General** across a diverse set of sparse matrices and sparse matrix operations

Software
Efficient compression using a Hierarchy of Bitmaps

Hardware
Unit that scans bitmaps to accelerate indexing

SMASH ISA
Presentation Outline

1. Sparse Matrix Basics
2. Compression Formats & Shortcomings
3. SMASH
   - Software Compression Scheme
   - Hardware Acceleration Unit
   - Cross-Layer Interface
4. Evaluation
5. Conclusion
SMASH: Software Compression Scheme

Software

Efficient compression using a Hierarchy of Bitmaps

• Encodes the positions of non-zero elements using bits to maintain low storage overhead
SMASH: Software Compression Scheme

BITMAP:
Encodes if a block of the matrix contains any non-zero element

MATRIX

<table>
<thead>
<tr>
<th>NZ</th>
<th>ZEROS</th>
<th>ZEROS</th>
<th>ZEROS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEROS</td>
<td>NZ</td>
<td>ZEROS</td>
<td>ZEROS</td>
</tr>
<tr>
<td>ZEROS</td>
<td>ZEROS</td>
<td>ZEROS</td>
<td>ZEROS</td>
</tr>
<tr>
<td>ZEROS</td>
<td>ZEROS</td>
<td>NZ</td>
<td>NZ</td>
</tr>
</tbody>
</table>

BITMAP

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Might contain high number of zero bits

Idea: Apply the same encoding recursively to compress more effectively
Hierarchy of Bitmaps

- **Non-Zero Values Array**

  - **Matrix**
    - Zero Region
    - Non-Zero Region
  
  - **Bitmaps**
    - Bitmap 0
    - Bitmap 1
    - Bitmap 2

- **Compression Ratios**
  - Ratio 4:1
  - Ratio 2:1

- **Storage**
  - Efficient

- **Fast Indexing**

- **Hardware Friendly**
### Presentation Outline

1. Sparse Matrix Basics
2. Compression Formats & Shortcomings
3. SMASH
   - Software Compression Scheme
   - Hardware Acceleration Unit
   - Cross-Layer Interface
4. Evaluation
5. Conclusion
SMASH: Hardware Acceleration Unit

Hardware

Unit that scans bitmaps to accelerate indexing

• **Interprets** the **Bitmap Hierarchy** used in software
• **Buffers** the **Bitmap Hierarchy** and **scans** it using bitwise operations
Bitmap Management Unit (BMU)
# Presentation Outline

1. Sparse Matrix Basics
2. Compression Formats & Shortcomings
3. SMASH
   - Software Compression Scheme
   - Hardware Acceleration Unit
   - Cross-Layer Interface
4. Evaluation
5. Conclusion
SMASH: Cross-Layer Interface

Need for a cross-layer interface that enables software to control the BMU

- Communicate the parameters needed to calculate the index
- Query the BMU to retrieve the index of the next non-zero element

Enables SMASH to flexibly accelerate a diverse range of operations on any sparse matrix

SMASH ISA
MATINFO
BMAPINFO
RDBMAP
PBMAP
RDIND

SAFARI
## Use Case: SpMV

**BMU**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Matrix Parameters**

**Bitmap Parameters**

**ROW INDEX**

**COLUMN INDEX**

**STREAM THROUGH THE NON-ZERO BLOCKS**

**NZ**

**VECTOR**

**NZA**

Index the vector using the BMU output

Query the BMU to discover the position of the NZ block

Use Case: SpMV

1. Query the BMU to discover the position of the NZ block.
2. Stream through the non-zero blocks.
3. Index the vector using the BMU output.
1. Sparse Matrix Basics
2. Compression Formats & Shortcomings
3. SMASH
   - Software Compression Scheme
   - Hardware Acceleration Unit
   - Cross-Layer Interface
4. Evaluation
5. Conclusion
Methodology

**Simulator:** *ZSim* Simulator [1]

**Workloads:**

- **Sparse Matrix Kernels**
  
  SpMV & SpMM from TACO [2]

- **Graph Applications**
  
  PageRank & Betweenness Centrality from Ligra [3]

**Input datasets:**

15 diverse sparse matrices & 4 graphs from the Sparse Suite Collection [4]

---


## Evaluated Sparse Matrices

<table>
<thead>
<tr>
<th>Name</th>
<th># Rows</th>
<th>Non-Zero Elements</th>
<th>Sparsity (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1: descriptor_xingo6u</td>
<td>20,738</td>
<td>73,916</td>
<td>0.01</td>
</tr>
<tr>
<td>M2: g7jac060sc</td>
<td>17,730</td>
<td>183,325</td>
<td>0.06</td>
</tr>
<tr>
<td>M3: Trefethen_20000</td>
<td>20,000</td>
<td>554,466</td>
<td>0.14</td>
</tr>
<tr>
<td>M4: IG5-16</td>
<td>18,846</td>
<td>588,326</td>
<td>0.17</td>
</tr>
<tr>
<td>M5: TSOPF_RS_b162_c3</td>
<td>15,374</td>
<td>610,299</td>
<td>0.26</td>
</tr>
<tr>
<td>M6: ns3Da</td>
<td>20,414</td>
<td>1,679,599</td>
<td>0.40</td>
</tr>
<tr>
<td>M7: tsyl201</td>
<td>20,685</td>
<td>2,454,957</td>
<td>0.57</td>
</tr>
<tr>
<td>M8: pkustk07</td>
<td>16,860</td>
<td>2,418,804</td>
<td>0.85</td>
</tr>
<tr>
<td>M9: ramage02</td>
<td>16,830</td>
<td>2,866,352</td>
<td>1.01</td>
</tr>
<tr>
<td>M10: pattern1</td>
<td>19,242</td>
<td>9,323,432</td>
<td>2.52</td>
</tr>
<tr>
<td>M11: gupta3</td>
<td>16,783</td>
<td>9,323,427</td>
<td>3.31</td>
</tr>
<tr>
<td>M12: nd3k</td>
<td>9,000</td>
<td>3,279,690</td>
<td>4.05</td>
</tr>
<tr>
<td>M13: human_gene1</td>
<td>22,283</td>
<td>24,669,643</td>
<td>4.97</td>
</tr>
<tr>
<td>M14: exdata_1</td>
<td>6,001</td>
<td>2,269,500</td>
<td>6.30</td>
</tr>
<tr>
<td>M15: human_gene2</td>
<td>14,340</td>
<td>18,068,388</td>
<td>8.79</td>
</tr>
</tbody>
</table>
Performance Improvement Using SMASH

SMASH provides significant performance improvements over state-of-the-art formats

SAFARI
SMASH significantly reduces the number of executed instructions.
SMASH provides speedups regardless of the sparsity of the matrix
Storage Efficiency

SMASH efficiently compresses sparse matrices
Hardware Area Overhead

SMASH configuration

• Support for 4 matrices in the BMU
• 256 bytes / SRAM buffer
• 140 bytes for registers & counters

0.076% area overhead over an Intel Xeon CPU

SMASH incurs negligible area overhead
Other Results in the Paper

- Compression ratio sensitivity analysis
- Distribution of non-zero elements
- Detailed results for SpMM
- Conversion from CSR to SMASH overhead
- Software-only approaches executed in a real machine
Summary & Conclusion

- Many important workloads heavily make use of **sparse matrices**
  - Matrices are **extremely sparse**
  - **Compression** is essential to avoid storage/computational overheads
- Shortcomings of existing compression formats
  - **Expensive discovery of the positions of non-zero elements** or
  - **Narrow applicability**
- **SMASH**: hardware/software cooperative mechanism for efficient sparse matrix storage and computation
  - **Software**: Efficient compression scheme using a hierarchy of bitmaps
  - **Hardware**: Hardware unit interprets the bitmap hierarchy and accelerates indexing
- Performance improvement: **38% and 44%** for **SpMV** and **SpMM** over the widely used CSR format
- SMASH is highly efficient, low-cost and widely applicable
Discussion

• How does SMASH work when the algorithm traverses the matrix in an irregular way?
• Remove decompression from the critical path?
• Use In-memory bitwise operations to find which operations result in 0s (before execution)?
• Determine the bitmap parameters on the fly?
• Can we use a similar mechanism in GPUs?
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
Compression Ratio

SMALL BITMAPS VS ZERO COMPUTATION

Bitmap-0

MATRIX

Six 0’s

NZA

A
B

A
B

Two 0’s

NZA

A
B

SAFARI
CSR-based SpMV

**CSR-based SpMV**

- **row_ptr**: 0 1 3 4 5
- **col_ind**: 0 2 3 1 3
- **VECTOR**: 0 1 2 3 4 5

**INDIRECT ACCESS TO LOAD THE VECTOR**
Effect of Compression Ratio
Distribution of non-zero elements

Matrix 13

Speedup

HIGHLY SCATTERED  ➔  HIGHLY GATHERED
SMASH enables highly efficient storage of sparse matrices.
Compression Ratio Selection

SMALL BITMAPS VS ZERO COMPUTATION

Bitmap-0

Six 0’s

NZA

A

B

MATRIX

Bitmap-0

Two 0’s

NZA

A

B

MATRIX

SAFARI