KU LEUVEN

DEPARTMENT OF COMPUTER SCIENCE

Algorithms for Parallel Shared-Memory Sparse Matrix-Vector Multiplication on Unstructured Matrices

Kobe Bergmans[1], Karl Meerbergen[1] and Raf Vandebril[1]

[1]KU Leuven, Dept. Computer Science, NUMA section

kobe.bergmans@kuleuven.be

Problem Statement

Sparse matrix-vector (SpMV) multiplication is an important kernel in scientific computing and graph analysis. Therefore, an efficient parallel algorithm is pivotal.

Unstructured sparse data appear more frequently due to large-scale data collection. E.g: social media, shopping data, road networks, ...

Two performance bottlenecks:

1. **Parallel load balancing** because of the unpredictable nonzero structure.
2. **Low arithmetic intensity** due to sparse matrix storage formats only storing the nonzero elements.

Goal: Combine aspects of the current state-of-the-art to create improved algorithms.

Standard Storage Formats

Coordinate format (COO) stores three arrays: two for the row and column indices, and one to store the nonzero entries.

Compressed Row Storage (CRS) uses less storage. The nonzero elements are sorted row-wise and the row indices are compressed by storing row pointers to the start of each row.

$$\begin{bmatrix} 0 & 2 & 5 & 0 \\ 3 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 3 & 0 & 0 \\ 2 & 0 & 9 & 0 \end{bmatrix} \Rightarrow \begin{cases} \texttt{row\_ptr} = [0, 2, 4, 5, 6, 8] \\ \texttt{col\_ind} = [1, 2, 0, 1, 3, 1, 0, 2] \\ \texttt{data} \quad\;\; = [2, 5, 3, 2, 1, 3, 2, 9] \end{cases}$$

State-of-the-art Algorithms

Merge-based

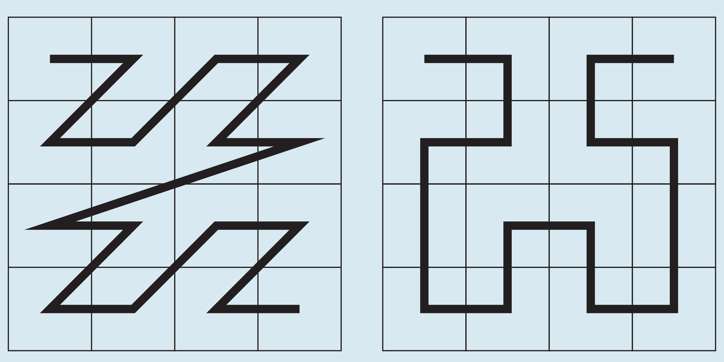
- CRS format.
- Perfect static load balancing in multiply-add and memory operations.

Compressed Sparse Blocks (CSB)

- Subdivides matrix into sparse subblocks.
- Elements inside blocks are stored using compressed COO format. Row and column indices are compressed into a 32 bit integer. Elements are sorted using the Z-Morton order.
- Blocks are stored using row-wise block pointers.
- Parallelization using *OpenMP* tasking. Tasks are multiplications of a block row with the possibility of further subdivision.

Row-distributed Block CO-H (BCOH)

- Statically subdivide matrix rows by equally distributing nonzero entries across thread.
- Subdivision into sparse subblocks in each partition.
- Nonzero elements inside blocks are stored using compressed CRS variant. The row pointers and columns indices are stored as 16 bit integers.
- Blocks are stored in Hilbert order, using a CRS variant.



Hybrid Algorithms

Six new algorithms are developed by combining optimizations of the state-of-the-art.

CSB Hilbert (CSBH)

- Elements inside blocks are ordered using the Hilbert curve.

BCOH Compression (BCOHC)

- Storage inside blocks using compressed COO.

BCOH Compression Hilbert (BCOHCH)

- All nonzero elements in a partition are sorted using the Hilbert curve. So, blocks and the nonzero elements inside them are ordered.

BCOH Compression Hilbert Pointer (BCOHCHP)

- Blocks are stored using row-wise block pointers.

Merge Blocking (MergeB)

- Subdivides matrix into sparse subblocks.
- Storage inside blocks using compressed COO. Blocks are stored using CRS.
- Static parallelization using the Merge algorithms where a multiply-add operations is replaced by a block multiplication, and memory operations consist of storing chunks of the output vector.

Merge Blocking Hilbert (MergeBH)

- Elements inside blocks are ordered using the Hilbert curve.

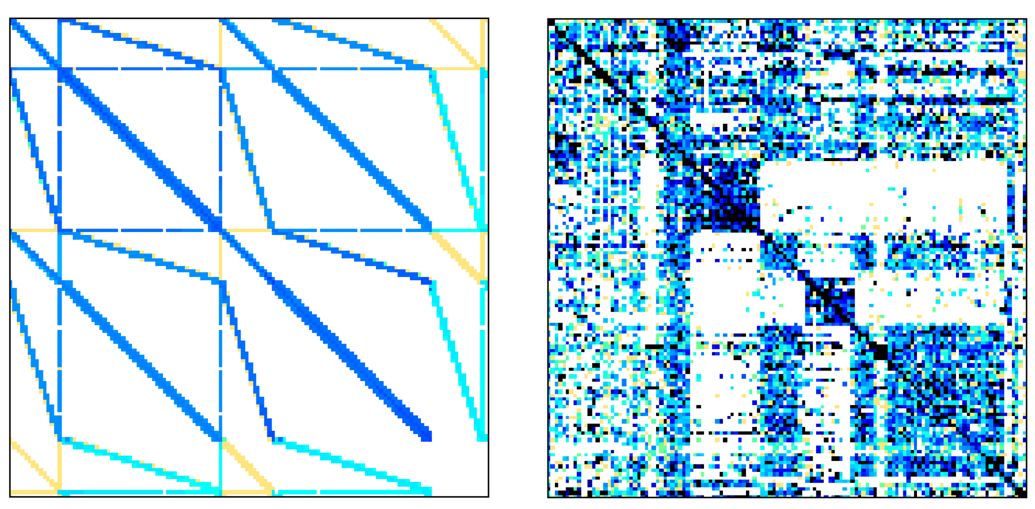
Evaluation

- High-performance, open-source, C++ implementation. Parallelized using *OpenMP*.
- Three test machines. Two NUMA systems and one UMA system.

Name	CPU	RAM
Sapphire Rapids	2×48 cores, 2.1 GHz	4800 MHz DDR5
Ice Lake NUMA	2×36 cores, 2.4 GHz	3200 MHz DDR4
Ice Lake UMA	36 cores, 2.4 GHz	3200 MHz DDR4

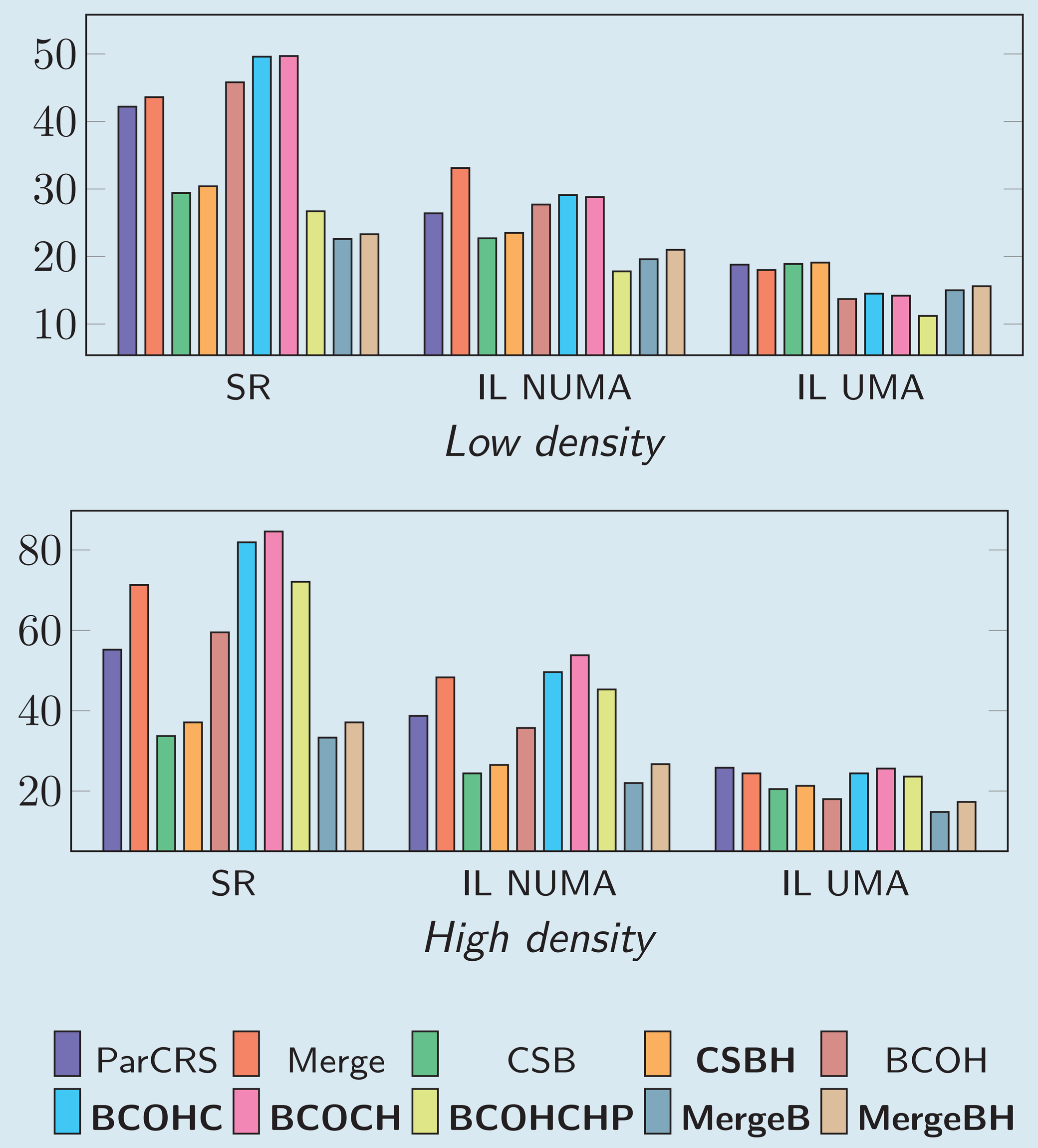
- 16 square unstructured test matrices from the SuiteSparse Matrix Collection. Split into two test groups: low ($\leq 10^{-6}$) and high density ($> 10^{-6}$).
- Parallel SpMV multiplication speedup is measured compared to the sequential CRS algorithm. Performance is averaged over all test matrices.
- Conversion time between COO format and required format is presented in terms of the time required by an amount of SpMV multiplications.

Matrix Structure

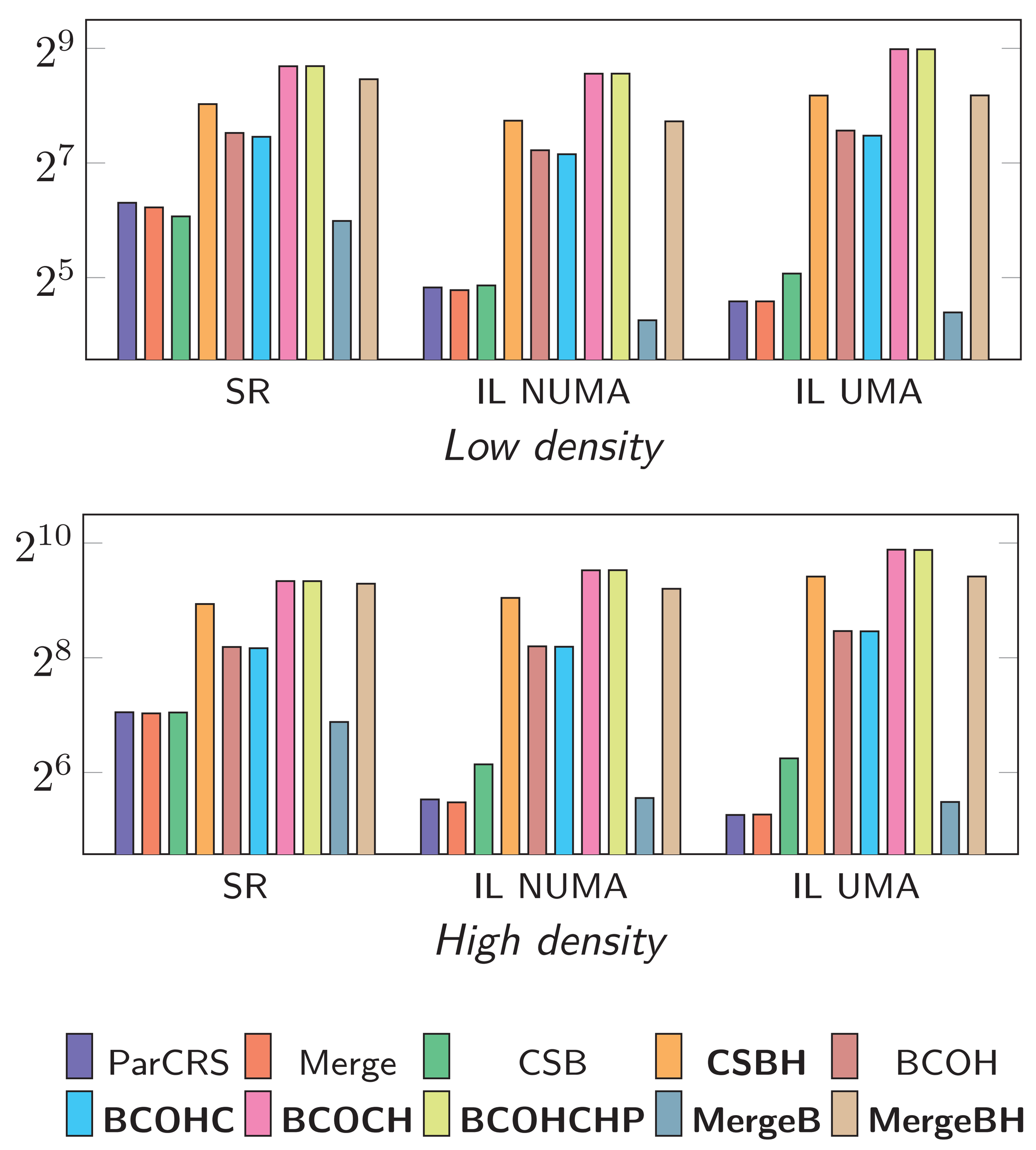


Structured matrix *Unstructured matrix*

SpMV Multiplication Results



Conversion Results



Conclusions

- The new **BCOH Compression (Hilbert)** algorithm achieves high performance for all types of matrices on NUMA machines. For high density matrices it **outperforms the current state-of-the-art by up to 19%.**
- Higher nonzero density increases parallel speedup.
- Complex storage formats induce a high conversion cost.
- Hilbert ordering has a positive effect on SpMV multiplication speedup, and improves over Z-Morton ordering. But, the cost of conversion increases significantly.

References

[1] Kobe Bergmans, Karl Meerbergen, & Raf Vandebril. (2025). Algorithms for Parallel Shared-Memory Sparse Matrix-Vector Multiplication on Unstructured Matrices. arXiv:2502.19284.

[2] Timothy A. Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. ACM Trans. Math. Softw. 38, 1, Article 1 (November 2011)

KU Leuven
Department of Computer Science
Celestijnenlaan 200A box 2402
3001 Leuven, Belgium