Experimental runtime analysis of algorithms for sparse binary matrix-vector products

Peter Ivony

Supervisor: Univ.-Prof. Dipl.-Ing. Dr. Wilfried Gansterer, M.Sc.

Faculty of Computer Science, University of Vienna

June 25, 2024

Overview

- Goals
- Related Work
- Preliminaries
- Ideas
- Implementation
- Results
- Future Work

Goals

- Review currently available high-performance algorithms for SBM-V multiplication
- Discuss potential improvements/propose new algorithms
- Implement and benchmark the proposed algorithm
- Evaluate benchmark results

Related Work

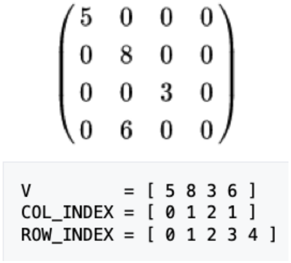
- Experimentation on different architectures
  - Sparse and dense matrix multiplication hardware for heterogeneous multi-precision neural networks [4]
  - Sparse matrix multiplication on an associative processor [7]
  - Sparse Binary Matrix-Vector Multiplication on Neuromorphic Computers [5]
- Libraries
  - A Sparse Matrix Library in C++ for High Performance Architectures [1]
  - SparseX [2]
- General algorithms
  - Fast Sparse Matrix Multiplication [8]
  - Automatic Performance Tuning of Sparse Matrix Kernels [6]
  - Mailman algorithm [3]

Ideas

- Use appropriate data structure
  - CSR / Modified CSR
- Use different algorithm
  - Mailman
  - Precompute results to a map
  - Partial sum method

CSR / Modified CSR

- Efficient sparse matrix storage format
- For binary matrices V array can be omitted
- More efficient in storage if $NNZ < \frac{m(n-1)-1}{2}$
- Runtime depends on number of non-zero entries: $\mathcal{O}(NNZ)$



$$\begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{pmatrix}$$

```
V         = [ 5 8 3 6 ]
COL_INDEX = [ 0 1 2 1 ]
ROW_INDEX = [ 0 1 2 3 4 ]
```

Figure: Example for matrix storage in CSR format. Source: Wikipedia

Mailman

- Works for matrices over fininte alphabets: $M \in \Sigma^{m \times n}$
- In binary case: $\Sigma = \{0, 1\}$
- Decompose $M = UP$, where
  - $U_n \in \Sigma^{n \times |\Sigma|^n}$ - contains all possible columns over Σ of length n
  - $P(i, j) = \delta(U^{(i)}, A^{(j)})$ - contains n ones, rest are zero
- Construct P - preprocess in $\mathcal{O}(nm)$
- U can be applied in $\mathcal{O}(4n)$ maximum - recursive construction and application
- Final runtime: $\mathcal{O}(mn/\log m)$

Precompute to map

- Use a map with rows in binary form as keys and corresponding sum result as value
- Precompute all key-value pairs, where row contains at most k ones
- Get result vector entries by querying the map
  - Hit $\rightarrow$ use the saved value
  - Miss $\rightarrow$ compute the value using the naive algorithm
- Multiplying the same vector with a different matrix does not require precomputation again
- Optimal value of k depends on sparsity
  - k too low $\rightarrow$ many misses
  - k too large $\rightarrow$ precomputation too expensive (can be offset by large amount of SPMV products with the same vector)

Partial sum method

- Each row corresponds to a partial sum of input vector elements
- $X = \{M_{(k)} | k \in \{0, 1, \ldots, n\}\}$
- Find set of rows $K \subseteq X$, where $\bigoplus_{x \in K} x = y$, s.t. $y \in X \setminus K$
- Try to find as many rows like y as possible to reduce # of additions

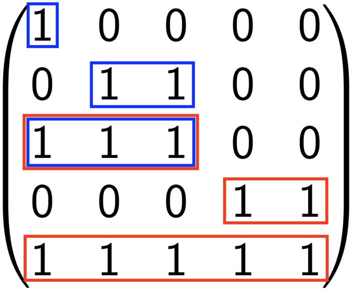


Figure: Number of additions reduced from 8 to 4

Implementation

- Software framework
- Software design

Software framework

- C++23 for main computations
- Python with NumPy, Pandas and Matplotlib for plot creation
- Makefile generates executable files
- GoogleTest for unit and performance tests
- Included OpenBLAS library to compare performance
- Codebase can be found on GitHub

Software design

- `IMatrix<T>` interface is implemented by multiple classes - uses templating to set data type stored
- Matrix types
  - `Matrix<N,M,T>` - general
  - `SparseMatrix<T>` - CSR storage format
  - `SparseBoolMatrix` - without *VALUES*
  - `RawBoolMatrix<N,M>` - for BLAS operations
  - `BitsetMatrix<N,M>` - uses an $N \times M$ bitset data structre
- RandomMatrixGenerator
- **MatrixProduct**
- Utils

Implemented algorithms

- Naive
- Optimized - using SparseBoolMatrix
- BLAS - using `cblas_dgemv` call
- Precompute to map

Tests

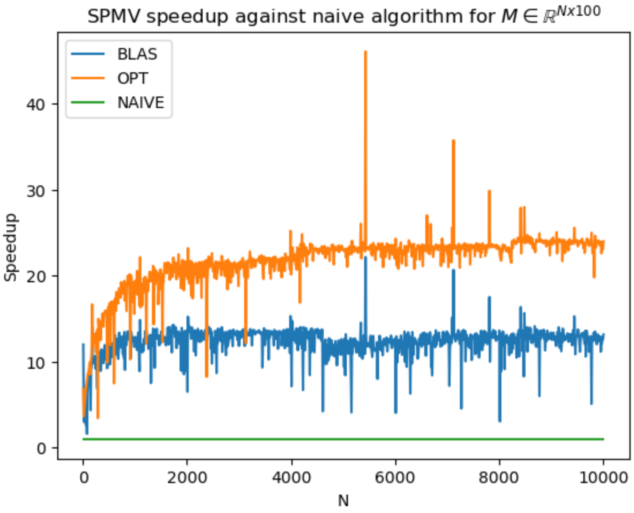
- Invoked using GoogleTest
- Unit tests for correctness
    - Matrix classes
    - Map key generation
    - Matrix product tests
- Performance tests - generate results to CSV for plotting
    - changing sparsity
    - excluding values below a certain threshold
    - changing matrix dimensions

Issues

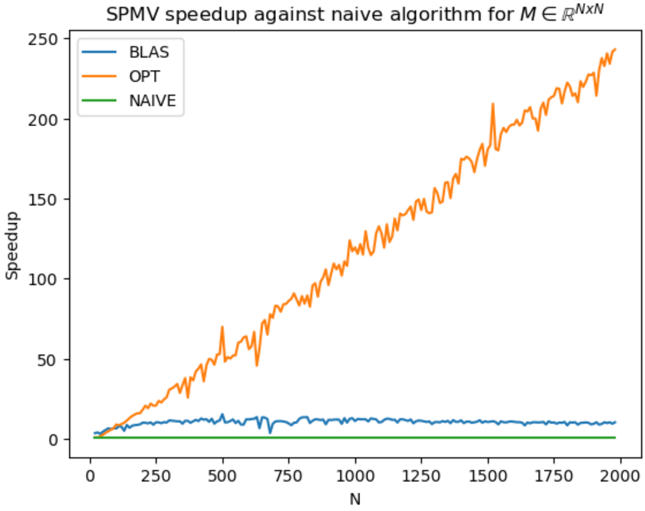
- Precompute to map $\rightarrow$ map access was too expensive in practice
- Partial sum method
  - Difficult to find optimal way to gather row sets, which fit the condition
  - Since addition is cheap (compared to multiplication), generally it is faster
  - More efficient for dense matrices, since number of additions is much higher
- Couldn't instantiate dimension tests at runtime, because of templating $\rightarrow$ `generate_tests.ipynb` helps to write test cases individually to file

Results I

- Sparsity set to $1/(N+M)$

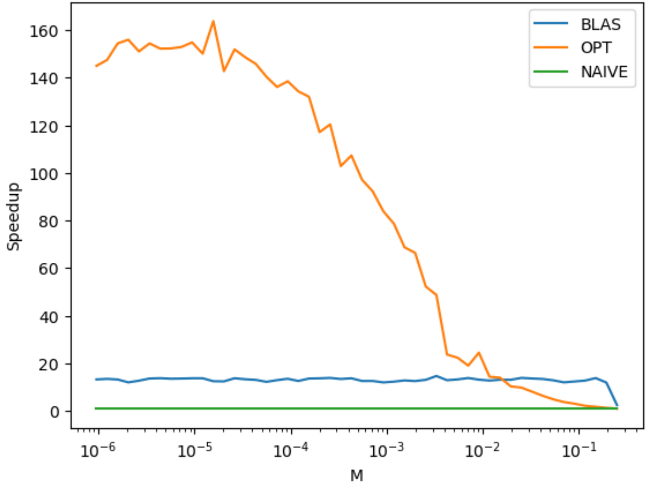


Results II


SPMV speedup against naive algorithm for $M \in \mathbb{R}^{NxN}$
BLAS
OPT
NAIVE
Speedup
N
0
50
100
150
200
250
0
250
500
750
1000
1250
1500
1750
2000

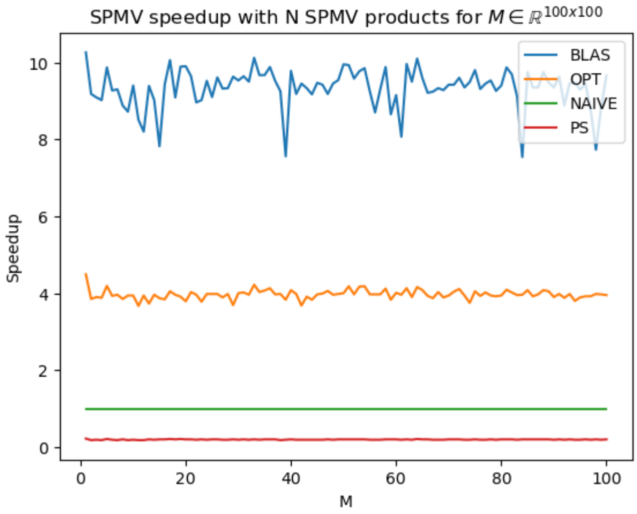
Results III

SPMV speedup against naive algorithm for $M \in \mathbb{R}^{10^3 x 10^3}$ with changing sparsity



Results IV

- Sparsity $= 0.5$



Conclusion & Future Work

- Main factor for runtime reduction is sparsity
- General requirement - efficient bit manipulation
- Implement map precompute with more efficient map data structure (possibly in other language)

Literature

[1] Jack Dongarra et al. “A Sparse Matrix Library in C++ for High Performance Architectures”. In: *Proceedings of the Second Object Oriented Numerics Conference* (May 1997).

[2] Athena Elafrou et al. “SparseX: A Library for High-Performance Sparse Matrix-Vector Multiplication on Multicore Platforms”. In: *ACM Trans. Math. Softw.* 44.3 (Jan. 2018). ISSN: 0098-3500. DOI: 10.1145/3134442. URL: https://doi.org/10.1145/3134442.

[3] Edo Liberty and Steven W. Zucker. “The Mailman algorithm: A note on matrix–vector multiplication”. In: *Information Processing Letters* 109.3 (2009), pp. 179–182. ISSN: 0020-0190. DOI: https://doi.org/10.1016/j.ipl.2008.09.028. URL: https://www.sciencedirect.com/science/article/pii/S0020019008002949.

[4] Jose Nunez-Yanez and Mohammad Hosseinabady. “Sparse and dense matrix multiplication hardware for heterogeneous multi-precision neural networks”. In: *Array* 12 (2021), p. 100101. ISSN: 2590-0056. DOI: https://doi.org/10.1016/j.array.2021.100101. URL: https://www.sciencedirect.com/science/article/pii/S259000562100045X.

[5] Catherine D. Schuman et al. “Sparse Binary Matrix-Vector Multiplication on Neuromorphic Computers”. In: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2021, pp. 308–311. DOI: 10.1109/IPDPSW52791.2021.00054.

[6] Richard Wilson Vuduc and James W. Demmel. “Automatic performance tuning of sparse matrix kernels”. AAI3121741. PhD thesis. 2003.

[7] L. Yavits, A. Morad, and R. Ginosar. “Sparse Matrix Multiplication On An Associative Processor”. In: *IEEE Transactions on Parallel and Distributed Systems* 26.11 (2015), pp. 3175–3183. DOI: 10.1109/TPDS.2014.2370055.

[8] Raphael Yuster and Uri Zwick. “Fast sparse matrix multiplication”. In: *ACM Trans. Algorithms* 1.1 (July 2005), pp. 2–13. ISSN: 1549-6325. DOI: 10.1145/1077464.1077466. URL: https://doi.org/10.1145/1077464.1077466.