

Accelerating Graph Neural Networks

João Alves^{1,2}, Siegfried Benkner¹

¹Research Group Scientific Computing, University of Vienna

Alexandre Francisco², Luis Russo²,

²INESC-ID, Lisboa

*Work received funding from EuroHPC JU, Grant Agreement No 101118139 Inno4scale

Outline

- CBM - New Compression Format for Binary Matrices
- Matrix Products w/ CBM
- Application to Graph Neural Networks
- Preliminary Results
- Conclusion

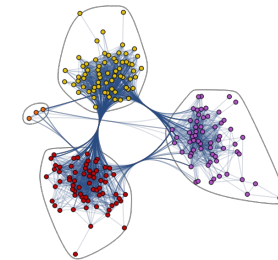
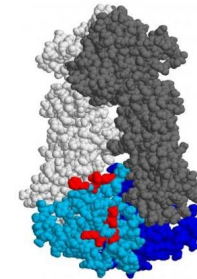
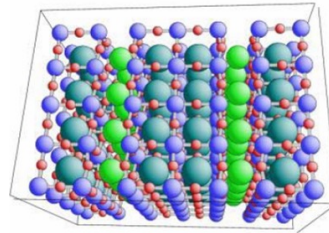
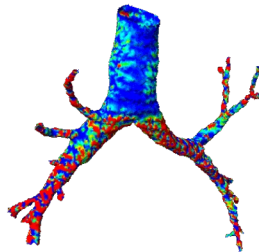
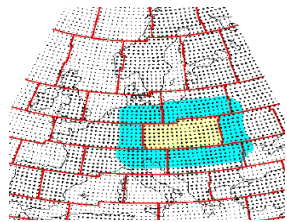
Research Group Scientific Computing

Research in

- languages, compilers, runtimes
- algorithms
- tools

Programmability
Performance
Portability

to help users solving complex problems on parallel and distributed systems.



Edge/IoT

Supercomputers/Cloud

Research Group Scientific Computing

Selected Projects

- EU Project PEPPER Performance Portability and Programmability for Heterogeneous Manycore Architectures (StarPU runtime)
- EU Project Autotune (Periscope Tuning Framework)
- FWF Project Dynamic Runtime Systems For Future Parallel Architectures (OCR)
- CHIST-ERA Sustainable Watershed Management Through IoT-Driven AI
- **EU Innovation Study – CBM4Scale**

EU Project Inno4Scale

<https://www.inno4scale.eu>

Innovative Algorithms For Applications On European Exascale Supercomputers

Partners: BSC, scapos, HLRS, PRACE

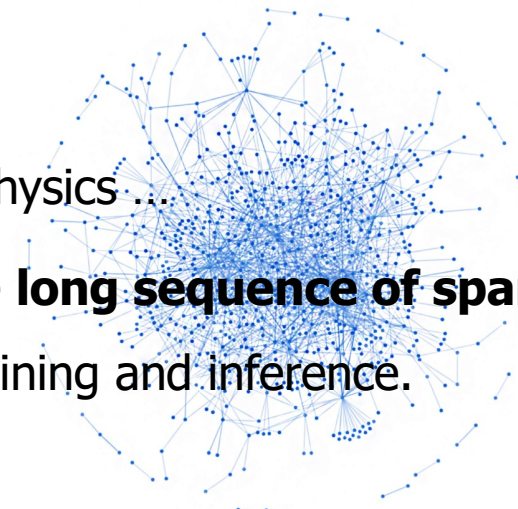
- Inno4scale supports the EuroHPC JU with **22 focused innovation studies** that realize proof-of-concept demonstrators for **innovative algorithms and their application.**
- An innovation study typically runs for 12 months with an effort of 24 PMs.

→ **CBM4Scale Innovation Study** (INESC-ID Lisbon, University of Vienna)

<https://www.inno4scale.eu/cbm4scale/>

CBM4scale Innovation Study

- Design and implementation of algorithms for novel **Compressed Binary Matrix (CBM) storage format** and **optimization** of matrix **operations**.
- Application to **Graph Neural Networks (GNN)**
 - Learning on graph datasets: Social Networks, Biology, Physics ...
 - Sparse matrix representations (**COO, CSR**) to accelerate **long sequence of sparse matrix multiplications** (SpMMs), which dominates training and inference.
- **Our Improvements**



For an unweighted graph, the resulting binary adjacency matrix can be often represented more efficiently than COO or CSR with a differential compression scheme. → **CBM Format**

Compressed Binary Matrix (CBM) Format

Key idea: Differential Compression

- Represent differences (deltas) of a row wrt. another row: $R_b = (R_a \cup \Delta^+) \setminus \Delta^-$
→ Number of deltas can be much lower than number of non-zeros (nnz) represented by CSR and COO formats.

Original Matrix

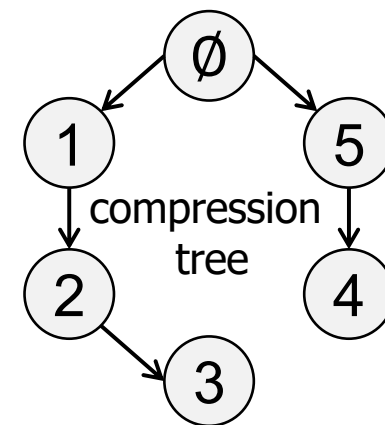
$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

$\text{nnz}(A) = 12$

CBM Format

$$A' = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

$\text{nDeltas}(A') = 7$



CBM Format - Benefits

- Improved **compression rate** up to **11x**.
 - Reduced memory footprint and traffic
- Reduced number of operations required to compute matrix products.
 - **Speedup of SpMM** up to **5x**
- CBM format is **never worse than CSR or COO**.
 - Number of deltas is upper-bounded by $\text{nnz}(A)$
- CBM format is also applicable to **column-scaled matrices**.
 - i.e., the product of a binary and a diagonal matrix
e.g.: A.D, D.A.D, A.D

$$\begin{pmatrix} 5 & 3 & 0 & 0 & 0 \\ 5 & 3 & 8 & 0 & 0 \\ 5 & 0 & 8 & 0 & 0 \\ 0 & 0 & 8 & 4 & 6 \\ 0 & 0 & 8 & 4 & 0 \end{pmatrix}$$

column-scaled matrix

CBM Format - Construction Algorithm (1)

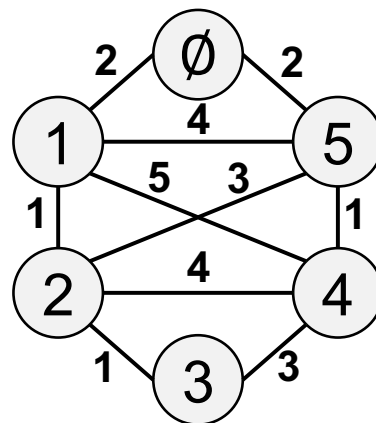
Challenge: Find a compression that **minimizes number of deltas**.

- **Step 1:** Compute **all-pairs Hamming-distance** to measure **dissimilarity**.
- **Step 2:** Find a **Minimum Spanning Tree (MST)** rooted in virtual node \emptyset .
- **Step 3:** Compute **list of deltas Δ^+ and Δ^-** for each edge of the MST.

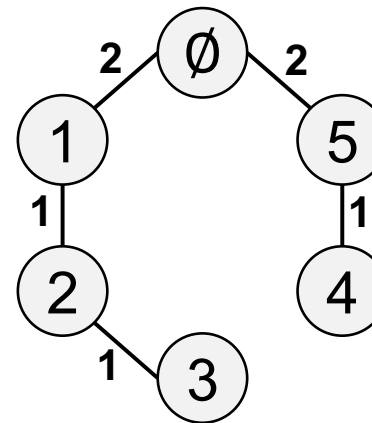
Original Matrix

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

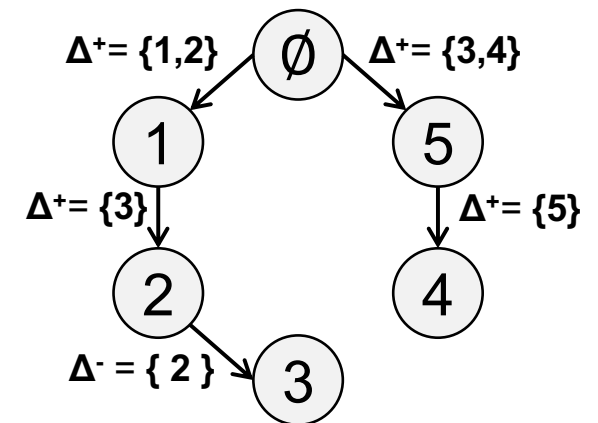
Step 1



Step 2



Step 3



CBM Format - Construction Algorithm (2)

CBM Format = Matrix of Deltas + Compression Tree

Original Matrix

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

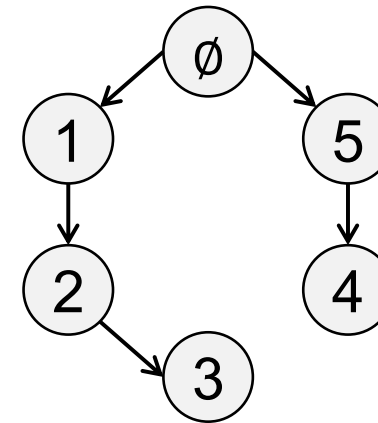
Steps 1-3



CBM Format

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

matrix of deltas



compression tree

Time complexity CBM for $A^{m \times n}$: $O(\underbrace{(m+1) \text{ nnz}(A)}_{\text{Step 1 + 3}} + \underbrace{m^2 \log(m)}_{\text{Step 2}})$

CBMSpMM Algorithm

Multiply matrix **A** in CBM format with dense matrix **B**:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$$

A **B**

Step 1:

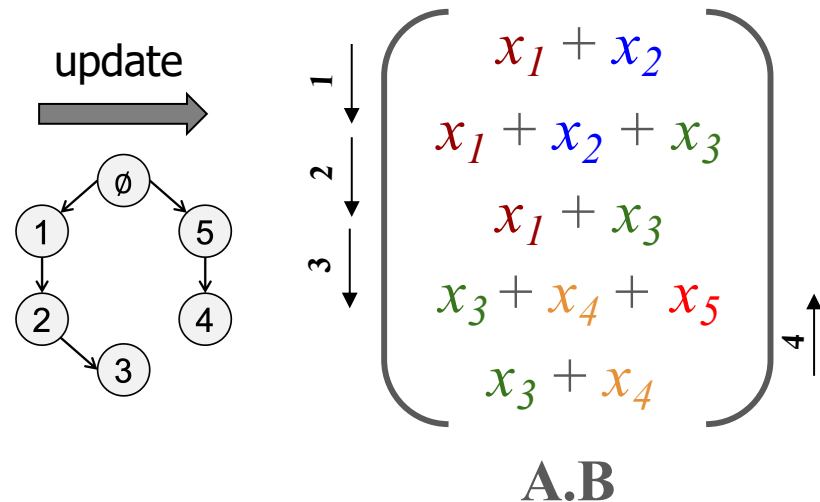
Multiply matrix of deltas **A'** with **B**
(e.g., use Intel MKL SpMM kernels with CSR format for **A'**)

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ x_3 \\ -x_2 \\ x_5 \\ x_3 + x_4 \end{pmatrix}$$

A' **B**

Step 2:

Update results traversing compression tree
(leveraging AXPY kernels and reusing already computed values).



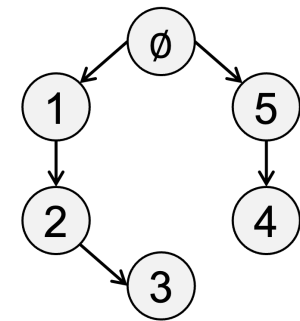
CBMSpMM Algorithm - Parallelism

Step 1: Multiply matrix of deltas \mathbf{A}' with \mathbf{B} using CSR format

- Use highly parallel kernels (e.g. Intel MKL, NVIDIA cuSPARSE) that exploit **vectorization** and **multithreading** and perform load balancing.

Step 2: Update resulting matrix by traversing compression tree

- Parallelization over branches in tree while observing dependences within.
- Implemented using **OpenMP**.



Tuning parameter alpha: if the number of similarities between two rows is less than **alpha** the row is not compressed (but rooted directly under node \emptyset).

→ Increases the degree of parallelism during update

Graph Neural Networks

- Training and inference in graph neural networks is dominated by long **sequence of sparse/dense matrix products**.
- For example, the inference of a **2-layer GCN** requires **2 sparse-dense matrix multiplications**, two dense-dense matrix multiplications and an element-wise activation function: $\hat{\mathbf{A}} \sigma(\hat{\mathbf{A}} \mathbf{X} \mathbf{W}^0) \mathbf{W}^1$
 - $\hat{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}} (\mathbf{A} + \mathbf{I}) \mathbf{D}^{-\frac{1}{2}}$ is the normalized Laplacian **adjacency matrix** of the graph that can be represented with the **CBM format**
 - \mathbf{X} is the dense node feature matrix (e.g., 500 rows)
 - $\mathbf{W}^0, \mathbf{W}^1$ are dense learnable matrices for the two layers (e.g., 500 columns)
 - σ denotes a ReLU activation function

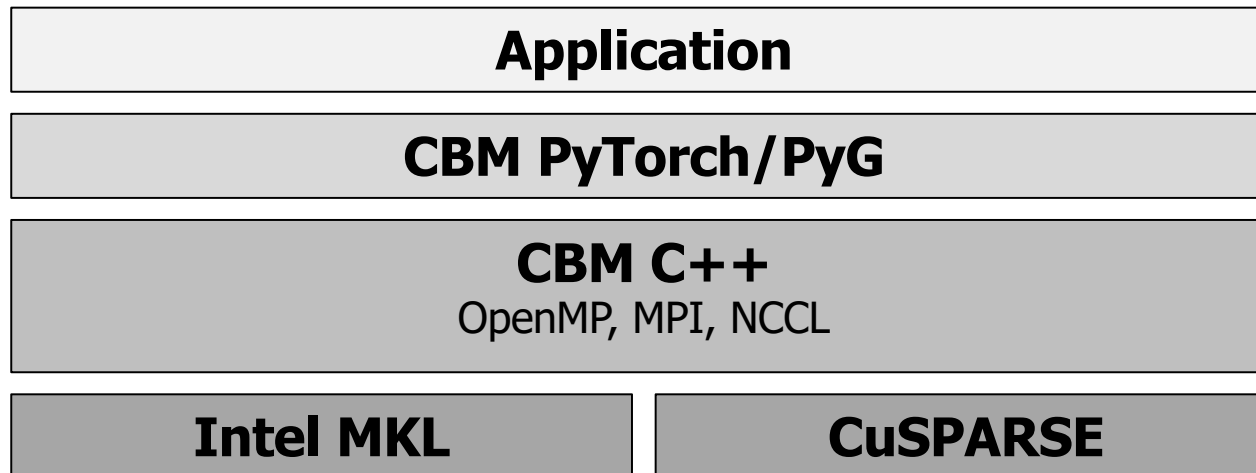
CBM Software Architecture

Integration in to PyTorch

- Popular open-source deep learning framework.
- CBM implementation available at: <https://github.com/cbm4scale>

Software Architecture

- Programmability: high-level PyTorch interfaces with CBM
- Performance & Portability: rely on Intel MKL and NVIDIA CuSPARSE



Preliminary Results – CPU

Compare **CBM** with **Intel MKL CSR** on a Xeon Gold 6130 (16 cores).

- Size Reduction = Matrix size CSR/Matrix size CBM
- **Speedup SpMM** = Time CSRSpMM/Time CBMSpMM
multiply adjacency matrix with dense feature matrix of size 500
- **Speedup Inference** = Inference Time CSR/Inference Time CBM
inference stage of a 2-layer GCN with 500 features

Graph	Nodes	Edges	Average In-degree	Size Reduction	Speedup SpMM	Speedup Inference
coPapersCiteseer	434,102	32,073,440	74.8	9.9	5.0	2.7
coPapersDBLP	540,486	30,491,458	57.4	6.0	2.7	1.8
COLLAB	372,474	24,572,158	65,9	11.0	5.3	2.0

Software: Python 3.11, **PyTorch**, C++, Intel MKL 24.0, OpenMP 4.5, CentOS Linux 7

(Very) Preliminary Results - GPU

Compare CBM with NVIDIA cuSPARSE CSR on NVIDIA V100

- Size Reduction = Matrix size CSR/Matrix size CBM
- **Speedup SpMM** = Time CSRSpMM/Time CBMSpMM
multiply adjacency matrix with dense feature matrix of size 500

Graph	Nodes	Edges	Average In-degree	Size Reduction	Speedup SpMM
coPapersCiteseer	434,102	32,073,440	74.8	9.9	2.3
coPapersDBLP	540,486	30,491,458	57.4	6.0	1.7
COLLAB	372,474	24,572,158	65,9	11.0	2.6

Software: Python 3.11, **PyTorch**, C++, CUDA 12.3, CentOS Linux 7

Ongoing work: optimization of update step

Conclusion

Compressed Binary Matrix (CBM) Format

- Reduce memory footprint of graphs and binary matrices compared to CSR, COO
- Accelerate sequence of SpMMs underlying GNN training and inference
- Portability by relying on *standard* numerical libraries (MKL, CuSPARSE)

Ongoing Work

- Optimizations for GPUs (CUDA)
- Multi-Node Parallelization and Multi-GPU (MPI, NCCL)
- Large-Scale experiments with GNN applications on EuroHPC JU machines

Technical Report

arXiv preprint arXiv:2409.02208, Sept. 2024

ACCELERATING GRAPH NEURAL NETWORKS WITH A NOVEL MATRIX COMPRESSION FORMAT

**João N. F. Alves^{1,2,3,4}, Samir Moustafa^{1,2}, Siegfried Benkner¹,
Alexandre P. Francisco^{3,4}, Wilfried N. Gansterer¹ & Luís M. S. Russo^{3,4}**

¹Faculty of Computer Science, University of Vienna, Vienna, Austria

²UniVie Doctoral School Computer Science, University of Vienna, Vienna, Austria

³Instituto Superior Técnico de Lisboa, Universidade de Lisboa, Lisbon, Portugal

⁴INESC-ID Lisboa, Lisbon, Portugal