Accelerating GCN Inference on Small Graphs

Hanwen Dai, **Changbo Chen**, Yuxuan Song

Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences

Euro-Par Workshop: HiPES 2024
August 26, 2024

Outline

Outline

Graph Neural Network (GNN)

Many applications [Zhang, 2019, Khemani et al., 2024]

- A huge graph: social networks, knowledge representations.
- A large number of (small) graphs: molecular graphs in bioinformatics.

Many variants

- Graph convolutional network (GCN)and [Kipf and Welling, 2017]
- Graph sample and aggregate network (GraphSAGE) [Hamilton et al., 2017]
- Graph attention network (GAT) [Velivcković et al., 2018]

Graph Convolutional Network (GCN)

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)}\right), \quad \ell = 0, 1, \cdots, L-1, \tag{1}$$

Related work

Related work on accelerating GNN/GCN inference

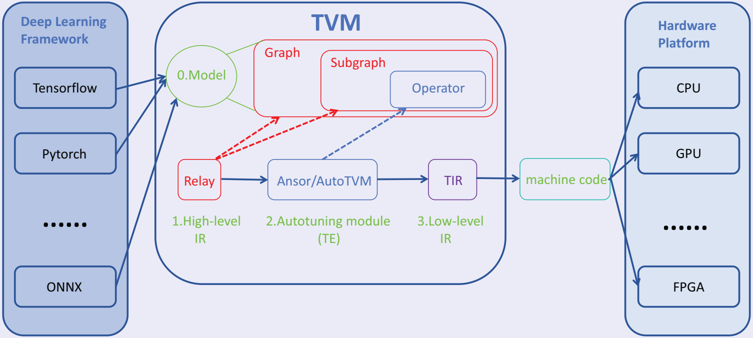
- Survey papers: [Liu et al., 2022, Abadal et al., 2022].
- Algorithm:
  - Reducing the feature dimension [Yik et al., 2022].
  - Sampling multi-hop neighbours [Zhang et al., 2023a].
  - Model reduction [Tan et al., 2023].
- Implementation:
  - Software implementation: DGL, PyG, DGI [Yin et al., 2023].
  - Hardware implementation: [Zhang et al., 2023b].

Related work on accelerating batched matrix multiplication

- Dimensions are the same in a batch: batch_matmul in TVM.
- Variable dimensions in a batch: MAGMA, Intel MKL.
- Our previous work on accelerating batched matrix multiplication for variable small sizes based on TVM [Dai and Chen, 2024].

Deep Learning Compiler: TVM [Chen et al., 2018]

Work flow of TVM



Supports in TVM for accelerating GCN

- TVM has introduced sparse tensors to support the inference of GCN.
- Internally, sparse tensors will be converted into dense ones.
- Currently TVM only supports inference on a single graph.

Outline

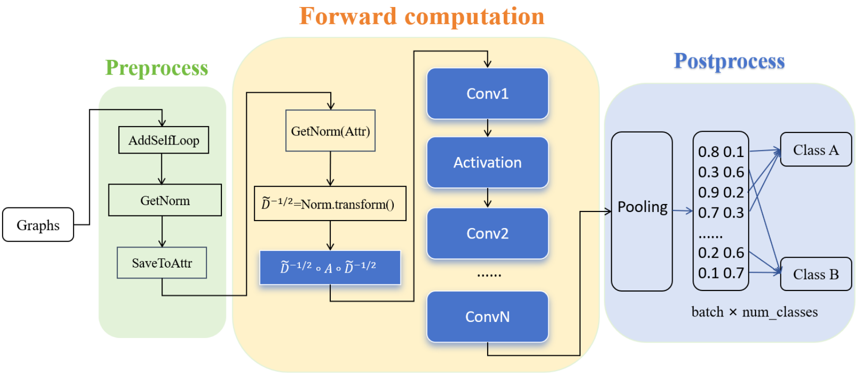
Main contribution and the optimizations deployed

- We replace (single large) sparse-dense matrix multiplication with (batched small) ones in GCN.
- We rearrange the order of basic operators in the forward computation of GCN to avoid redundant computation.
- We implement these optimizations in TVM to provide efficient GCN inference for batched graphs.

A series of optimization techniques deployed

- Replacing sparse operators with dense ones in both DGL and TVM.
- Reordering basic operators to avoid redundant computations in both DGL and TVM.
- Applying associative law to reduce number of arithmetic operations in TVM.
- Providing batched dense matrix multiplications targeting for small matrices in both DGL and TVM.
- Utilizing TVM complier optimization (mainly constant folding).

The different stages of GCN inference



- The preprocessing part includes loading input graph data and preparing adjacency matrix.
- The computing part includes normalization of adjacency matrix, convolution and activation.
- After getting output of the last layer, we apply postprocessing to obtain embedding or classification result of node, edge or graph.

Replacing sparse-dense matrix multiplication by dense ones

The intuition

- Cons: This optimization increases the number of arithmetic operations.
- Pros: Fully leverages GEMM optimizations, such as cache reuse and vectorization.
- For small matrices, overhead can be compensated by benefits.

Implementation details

- For DGL, we propose DGL*-Dense by uniformly adopting numpy.dot to replace torch.sparse.mm.
- For TVM, we propose TVM*-Dense for employing relay.nn.dense instead of relay.nn.sparse_dense.

Hadamard product

Defined for matrices of the same size

- $H = A \odot B$, defined as $H_{ij} = A_{ij} * B_{i,j}$.

Accelerating product of a diagonal matrix with a dense one

- $D_{m\times m}$ is a diagonal matrix with $D_{ii} = d_i$.
- We want to compute $D \cdot A$ efficiently.
- Let $e = [1, ..., 1]^t$ and $d = [d_1, \ldots, d_m]^t$.
- $D \cdot A$ can be computed as: $(D \cdot e \cdot e^t) \odot A$.
- $D \cdot e \cdot e^t = [d, \ldots, d]$.
- By $D \circ A$, we mean compute $D \cdot A$ in a Hadamard product way.
- Similarly, we have
  - $A \cdot D$ can be computed as: $A \odot (e \cdot e^t \cdot D)$.
  - By $A \circ D$, we mean compute $A \cdot D$ in a Hadamard product way.

Reordering basic operators (I): Basic idea

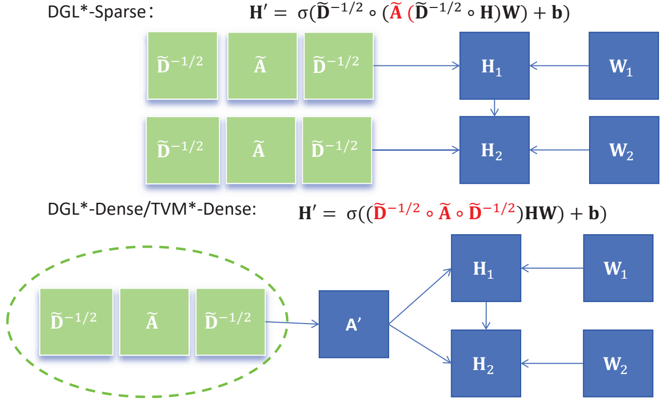


Figure: Basic operators reordering in a two-Layer GCN

- Pros: Efficiently compute product of diagonal degree matrix $\widetilde{\mathbf{D}}^{-\frac{1}{2}}$ by another dense matrix by utilizing Hadamard product operation.
- Cons: Redundant computation of $\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}$ when $\#conv > 1$.

Reordering basic operators (II): Some extra details

Complexity analysis

- Size of the adjacency matrix **A**: $m * m$.
- Size of the feature matrix **H**: $m * p_i$.
- Size of the weight matrix **W**: $p_i * q_i$.
- The saved number of floating-point operations is $m \sum_{i=1}^{n} (p_i + q_i) - 2m^2$.

Implementation of *Conv* layer from TVM

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}} \circ \left(\mathbf{W}^{(\ell)^T}\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}} \circ \mathbf{H}^{(\ell)}\right)^T \widetilde{\mathbf{A}}^T\right)^T + \mathbf{b}^{(\ell)}\right) \tag{2}$$

This is because implementation of matrix multiplication in Relay layer of TVM only accommodates $C = \mathbf{A}\mathbf{B}^T$ through operator relay.nn.dense.

Exploiting the associative law of matrix multiplication

Recall the core computation of GCN

$$\mathbf{A}_{m\times m} * \mathbf{H}_{m\times p} * \mathbf{W}_{p\times q} \tag{3}$$

Simple complexity analysis

- Order $(\mathbf{A} * \mathbf{H}) * \mathbf{W}$ incurs $2(m^2p + mpq)$ FLOPS.
- Order $\mathbf{A} * (\mathbf{H} * \mathbf{W})$ incurs $2(m^2q + mpq)$ FLOPS.

Implementation details

- Depending on values of p and q, one can choose computing order incurring the smallest number of FLOPS.
- DGL utilizes this feature.
- Our implementation DGL* and TVM* also utilizes this feature.

The other two optimizations utilizing TVM

Batched matrix multiplication for TVM

- Treat sparse matrices as dense ones.
- Utilize batch_matmul in TVM for batched dense matrix multiplication of the same size.
- TVM*-B groups matrices of the same size into one group.
- TVM*-M firstly sorts matrices by their dimensions and performs zero-padding on adjacency matrices to match the maximum dimension of adjacency matrices in the batch (default size: 32).

TVM compiler level optimization with constant folding

- Constant folding: identifies a constant expression and replace it with a constant value at compile time.
- The adjacency matrices, degree vectors, and weight matrices in GCN are stored in the relay layer of TVM as constant expressions.

Outline

Information of selected datasets and experimental environment

Table: Information of selected datasets.

Name	#Graphs	$\#\text{Nodes}_{\max}$	#Classes	Application	Accuracy
AIDS	2000	95	2	small molecules	98.35%
BZR	405	57	2	small molecules	80.99%
COX2	467	56	2	small molecules	78.16%
DHFR	756	71	2	small molecules	71.29%
Cuneiform	267	36	30	computer vision	70.41%
Letter-low	2250	8	15	computer vision	84.13%
Synthie	400	99	4	Synthetic	93.00%

- Seven datasets from TUDataset Morris et al. [2020] are selected for performance evaluation.
- GCNs are pretrained to obtain reasonable accuracies (training/testing=4/1).
- Intel i7-9700F @ 3.0 GHz, 16 GB DDR4-2666.
- LLVM 13.0.0, g++ 9.4.0, TVM 0.12.0 and DGL 2.1.

Different implementations to compare

- DGL: Current implementation of GCN inference in DGL.
- DGL*-reimplementSparse: A re-implementation of DGL, featuring a rewritten convolution implementation in the PyTorch platform.
- DGL*-Sparse: essentially DGL*-reimplementSparse but only timing the most compute-intensive four parts for fair comparision.
- DGL*-DirectDense: A direct translation of DGL-sparse with sparse matrix multiplications replaced by dense ones.
- DGL*-Dense: Re-arranging the order of dense operations.
- TVM: Current implementation of TVM on GCN inference for single graph.
- TVM*-Dense: Replacing the sparse tensor operations by dense ones and re-arranging the order of dense operations.
- TVM*-B: Support batch processing on TVM through combining matrices in same dimension, which is not affected by batch size.
- TVM*-M: Support batch processing on TVM through padding to the maximum dimension in a batch.

The common computations of all implementations

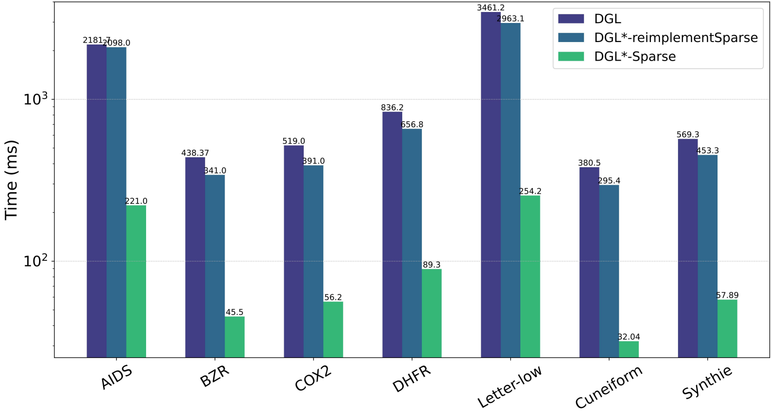
The four parts

- Hadamard product (H-product): $A^* = \widetilde{\mathbf{D}}^{-\frac{1}{2}} \circ \widetilde{\mathbf{A}} \circ \widetilde{\mathbf{D}}^{-\frac{1}{2}}$.
- Conv: $A^*\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)}$.

Method	H-product (ms)	Conv1 (ms)	ReLu (ms)	Conv2 (ms)
DGL*-Sparse	42.9	93.4	**11.3**	73.5
DGL*-DirectDense	42.1	70.0	12.2	48.0
DGL*-Dense	**18.0**	**60.7**	12.3	**40.1**

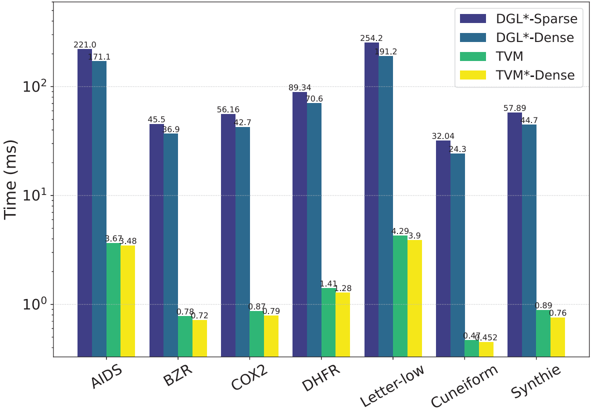
- The table reports the timings of three implementations in DGL on AIDS dataset.
- Replacing the sparse operators by dense ones brings speedup.
- Re-ordering the computation also brings speedup.

End-to-end evaluation



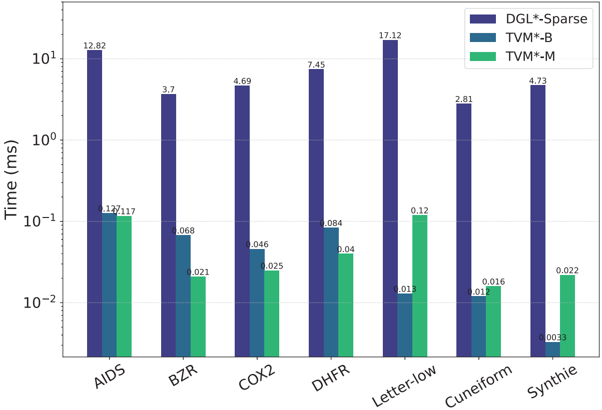
- DGL*-reimplementSparse has similar performance with original DGL.
- The difference with DGL*-Sparse shows the overhead of preprocessing and postprocessing part are high.

Performance of relevant implementations on handling graphs one by one



- DGL*-Dense achieves $1.3\times$ on average over DGL*-Sparse.
- TVM*-Dense achieves an average speedup of $1.1\times$ over TVM.
- TVM achieves on average $20\times$ speedup over DGL*-Dense.

Performance of relevant implementations on handling graphs in batch



- Batch size: 32.
- TVM*-B achieves an average speedup of $475.6\times$ over batched DGL*-Sparse.
- TVM*-M achieves an average speedup of $170.4\times$ over batched DGL*-Sparse.

Why do the performance of TVM*-B and TVM*-M vary with data?

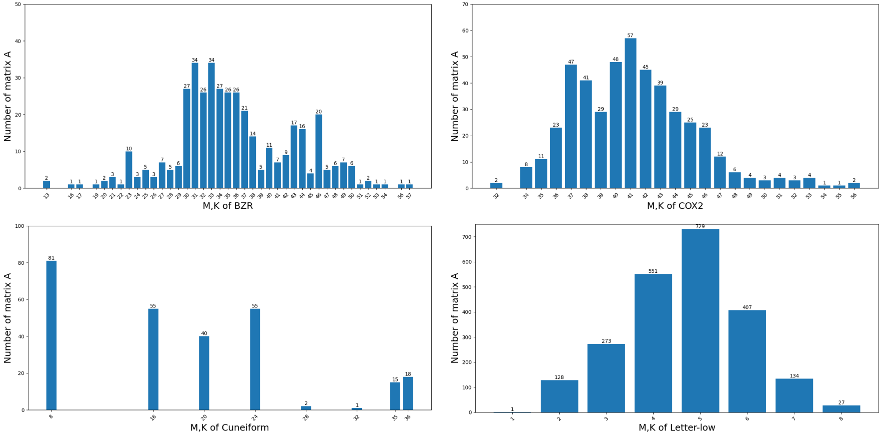
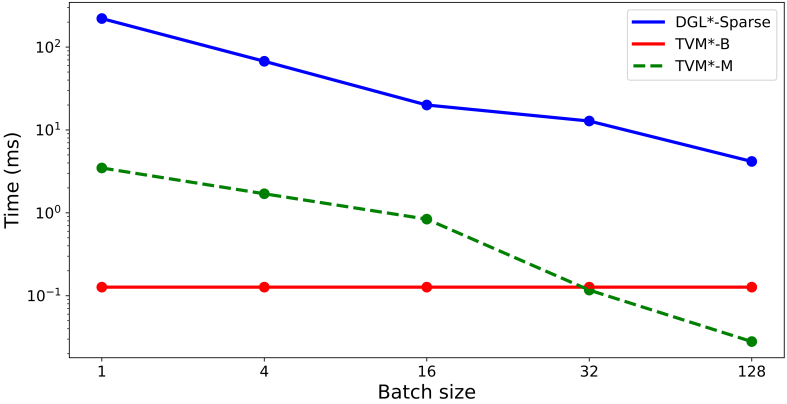


Figure: Comparison of matrix size distribution on different datasets.

- The dimensions have a wide range for the datasets BZR and COX2.
- The dimensions are highly centralized for Cuneiform and Letter-low.

Performance of batch processing on AIDS as the batch size increases



- TVM*-B ignores the given batch size and merges matrices of the same dimensions into a batch.

Outline

Conclusion and future work

- Targeting on small size graphs, we propose implementing GCN inference fully relying on dense operators.
- Several optimization strategies were proposed, such as replacing single sparse matrix multiplication by efficient batched dense matrix multiplication with TVM support and rearranging the order of basic operators.
- Experiments show that our method outperforms DGL and TVM on small graph datasets from real applications.

Future work

- Reducing the overhead of components other than the "most compute-intensive operations".
- Migrate the acceleration techniques to GNNs other than GCNs.

Jiawei Zhang. Graph neural networks for small graph and giant network representation learning: An overview. *arXiv preprint arXiv:1908.00187*, 2019.

Bharti Khemani, Shruti Patil, Ketan Kotecha, and Sudeep Tanwar. A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. *Journal of Big Data*, 11(1):18, 2024.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Petar Velivcković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2018.

Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, Dongrui Fan, Shirui Pan, and Yuan Xie. Survey on graph neural network acceleration:

An algorithmic perspective. In *Proceedings of the 31th International Joint Conference on Artificial Intelligence (IJCAI 2022)*, pages 5521–5529, 2022.

Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Comput. Surv.*, 54(9):191:1–191:38, 2022.

Jason Yik, Sanmukh R. Kuppannagari, Hanqing Zeng, and Viktor K. Prasanna. Input feature pruning for accelerating GNN inference on heterogeneous platforms. In *2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 282–291, 2022.

Dalong Zhang, Xianzheng Song, Zhiyang Hu, Yang Li, Miao Tao, Binbin Hu, Lin Wang, Zhiqiang Zhang, and Jun Zhou. InferTurbo: A scalable system for boosting full-graph inference of graph neural network over huge graphs. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 3235–3247, 2023a.

Qiaoyu Tan, Daochen Zha, Ninghao Liu, Soo-Hyun Choi, Li Li, Rui Chen, and Xia Hu. Double wins: Boosting accuracy and efficiency of graph neural networks by reliable knowledge distillation. In *2023 IEEE International Conference on Data Mining (ICDM)*, pages 1343–1348, 2023.

Peiqi Yin, Xiao Yan, Jinjing Zhou, Qiang Fu, Zhenkun Cai, James Cheng, Bo Tang, and Minjie Wang. DGI: An easy and efficient framework for GNN model evaluation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5439–5450, 2023.

Bingyi Zhang, Hanqing Zeng, and Viktor K. Prasanna. GraphAGILE: An FPGA-based overlay accelerator for low-latency GNN inference. *IEEE Transactions on Parallel and Distributed Systems*, 34(9):2580–2597, 2023b.

Hanwen Dai and Changbo Chen. Accelerating batched matrix multiplication for variable small sizes based on TVM (in chinese). Accepted, 2024.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX*

Symposium on Operating Systems Design and Implementation (OSDI 18), pages 578–594, 2018.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.