iSpLib: A Library for Accelerating Graph Neural Networks using Auto-tuned Sparse Operations

Md Saidul Hoque Anik
Department of Intelligent Systems Engineering
Indiana University Bloomington
Bloomington, Indiana, USA
mdshoque@iu.edu

Pranav Badhe
Department of Intelligent Systems Engineering
Indiana University Bloomington
Bloomington, Indiana, USA
pbadhe@iu.edu

Rohit Gampa
Department of Intelligent Systems Engineering
Indiana University Bloomington
Bloomington, Indiana, USA
rgampa@iu.edu

Ariful Azad
Department of Intelligent Systems Engineering
Indiana University Bloomington
Bloomington, Indiana, USA
azad@iu.edu

ABSTRACT

Core computations in Graph Neural Network (GNN) training and inference are often mapped to sparse matrix operations such as sparse-dense matrix multiplication (SpMM). These sparse operations are harder to optimize by manual tuning because their performance depends significantly on the sparsity of input graphs, GNN models, and computing platforms. To address this challenge, we present iSpLib, a PyTorch-based C++ library equipped with auto-tuned sparse operations. iSpLib expedites GNN training with a cache-enabled backpropagation that stores intermediate matrices in local caches. The library offers a user-friendly Python plug-in that allows users to take advantage of our optimized PyTorch operations out-of-the-box for any existing linear algebra-based PyTorch implementation of popular GNNs (Graph Convolution Network, GraphSAGE, Graph Inference Network, etc.) with only two lines of additional code. We demonstrate that iSpLib obtains up to 27x overall training speedup compared to the equivalent PyTorch 2.1.0 and PyTorch Geometric 2.4.0 implementations on the CPU. Our library is publicly available at https://github.com/HipGraph/iSpLib[1].

CCS CONCEPTS

• **Software and its engineering** → **Software libraries and repositories**; • **Mathematics of computing** → **Graph algorithms**; • **Computing methodologies** → *Learning latent representations.*

KEYWORDS

Graph Neural Network, Autotuning, Parallel Computing, Sparse-dense Matrix Multiplication, Autodiff, Backpropagation

[1]https://doi.org/10.5281/zenodo.10806511

WWW '24 Companion, May 13–17, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0172-6/24/05
https://doi.org/10.1145/3589335.3651528

ACM Reference Format:
Md Saidul Hoque Anik, Pranav Badhe, Rohit Gampa, and Ariful Azad. 2024. iSpLib: A Library for Accelerating Graph Neural Networks using Auto-tuned Sparse Operations . In *Companion Proceedings of the ACM Web Conference 2024 (WWW '24 Companion), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3589335.3651528

1 INTRODUCTION

Over the last decade, graph neural networks (GNNs) have demonstrated remarkable effectiveness in learning from graph-structured data. This success has led to the development of several prominent libraries for GNNs, such as PyTorch Geometric (PyG)[2], DGL[10], and CogDL [1], among others. Across these libraries, the fundamental computations for GNN forward and backward propagation primarily rely on two sparse operations: (a) sampled dense-dense matrix multiplication (SDDMM) and (b) sparse-dense matrix multiplication (SpMM). Consequently, the overall performance of these libraries is often influenced significantly by the efficiency of SDDMM and SpMM. To enhance the speed of these sparse operations, we developed a general-purpose C++ library known as iSpLib. This library can be seamlessly integrated into the backend of PyTorch-based GNN training, facilitating accelerated GNN training and inference on various multi-core processors.

iSpLib takes advantage of the insight that high-level operations within GNNs, such as graph convolutions and message passing, can be mapped to sparse linear algebra. Consequently, the library optimizes backend computations while preserving the familiar Python interface for users. iSpLib is designed with four key objectives:

(a) General-purpose: iSpLib supports various GNN models, including GCN[5], GraphSAGE [3], and GIN [11], utilizing operations such as SpMM, SDDMM, and their combination known as FusedMM [8]. To accommodate a broad spectrum of GNNs, we break down the overall computation into smaller matrix and vector kernels. These micro kernels allow users to define their own operations.

(b) Fast and scalable GNN training and inference: iSpLib significantly accelerates GNN training compared to existing sparse libraries like PyTorch-sparse [9]. This speed is achieved through a

harmonious combination of efficient parallelization, thread scheduling, loop unrolling, register blocking, and data caching during backpropagation.

(c) Performance portability: iSpLib exhibits consistent performance across diverse multi-core processors without requiring users to manually optimize their code. Automated tuning based on different hardware features, such as SIMD intrinsics, vector lengths, and register sizes, enables iSpLib to generate optimized code for the target platform.

(d) Easy-to-use library: iSpLib seamlessly integrates into PyTorch-based libraries, such as PyG [2], allowing users to divert computations to iSpLib with just one or two lines of code. This approach enables users to leverage efficient and auto-tuned kernels without explicit modifications to their existing code.

At present, iSpLib exclusively supports CPU-based tuning for Intel, AMD, and ARM processors. Our testing involved iSpLib's compatibility with PyG and various standalone GNN models, including the PyTorch-based GCN developed by the authors. We observed significant performance improvements when iSpLib was integrated with PyG, resulting in a GNN training speedup of up to 27× for GCN, 12× for GraphSAGE-sum, 8× for GraphSAGE-mean, and 18× for Graph Isomorphism Network (GIN). Importantly, these speed enhancements were achieved while maintaining the same level of accuracy attained without iSpLib and concealing all low-level optimizations from the users.

2 RELATED WORKS

Researchers used various approaches to expedite GNN training time. SparseTIR [12] reports up to 7.45x speedup for sparse convolution using compilation abstraction. Lenadora et al. [6] proposes a data-driven adaptive strategy and reports up to 1.99x speedup on graph convolutional networks. You et al. [7] focus on choosing a suitable sparse matrix storage format to improve the GNN training performance and report up to 3x performance improvement in GNN running time. Wang et al. [4] proposes FeatGraph that accelerates GNN training by co-optimizing graph traversal and feature dimension computation and reports up to 32x speedup on CPU.

3 LIBRARY DESIGN

3.1 Overview

The codebase of iSpLib consists of Python, C++, and C code (see Figure 1). The highly efficient kernels are generated in pure C. A C++ PyTorch wrapper connects the forward and backward propagation with the generated sparse kernels and provides the abstraction between PyTorch Tensors and the C++ array. A Python interface of iSpLib provides a ready-to-use matmul function for performing sparse-dense matrix multiplication. iSpLib also includes a PyTorch Geometric plug-in that allows users to use our matrix multiplication in popular GNN implementations when the dataset format is compatible.

3.2 Auto-tuning Mechanism

iSpLib has an auto-tuning mechanism that suggests the optimal embedding size for a given user environment. iSpLib probes the hardware to determine SIMD vector length and generates kernels for various multiples of these vector lengths (VLEN). When the embedding dimension is not a multiple of VLEN, we use a trusted kernel that is still efficient with balanced multithreading, but it does not perform loop unrolling. The auto-tuning feature allows users to tune the library against a given dataset by generating a comparison chart for speedup on the generated kernels over the trusted kernels for a sequence of embedding sizes (K). The tuning graph is typically a bell-shaped curve whose peak corresponds to the ideal embedding size (K) since it is where the generated kernel achieved the most speedup compared to the trusted kernel.

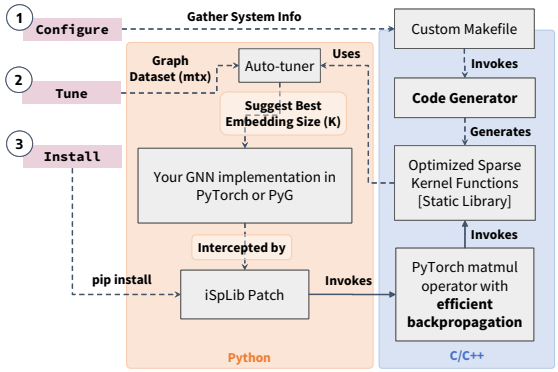


Figure 1: Overview of the iSpLib Library

3.3 Efficient Backpropagation

Another major source of iSpLib's speedup is efficient backpropagation. iSpLib's intelligent matrix-multiplication kernel is designed to evaluate common expressions required between training epochs and cache them locally. The caching mechanism greatly reduces the time spent in backpropagation especially when the input graph is large or the number of epochs is high.

3.4 Semiring Support

The sparse-dense matrix multiplication of iSpLib also supports various semirings. This is particularly useful for training GraphSAGE that involves aggregation methods other than sum. iSpLib's matmul operator provides functionality for min, max, and mean reduction operations on top of the standard sum operation. Currently, only the sum reduction operation supports generated kernels in our library.

3.5 Matmul Interface and Dependency

iSpLib provides a PyTorch-based interface for sparse-dense matrix multiplication. The matmul function receives a SparseTensor[2], a dense matrix in typical 2-d PyTorch tensor format, and optionally a reduction operation string having either 'sum', 'min', 'max', or 'mean' as parameters. Pytorch_sparse library provides a transform function to convert existing PyTorch-based datasets into SparseTensor format. After dataset conversion, our matmul function can be used to develop any GNN that requires sparse-dense matrix multiplication.

[2]A sparse matrix data structure provided by pytorch_sparse Python library

Table 1: Datasets

Graph Dataset	Feature Length	Prediction class	Node Count	Edge Count
Reddit	602	41	232,965	11,606,919
Reddit2	602	41	232,965	23,213,838
OGBN-mag	128	349	736,389	135,680,469
Amazon Products	200	107	1,569,960	264,339,468
OGBN-Product	100	47	2,449,029	61,859,140
OGBN-Protein	8	112	154,154	159,462

3.6 PyTorch Geometric Integration

Finally, iSpLib provides PyG 'patch' and 'unpatch' functions, allowing users to seamlessly integrate our auto-tuned matmul function into existing PyG implementation of GNNs involving sparse-dense matrix multiplication. This can be done by importing the iSplib library at the top of the PyG implementation code and invoking the patch function. The users can also disengage the iSpLib interception at any point of code by invoking the 'unpatch' method. iSpLib also provides a decorator for patching a single function in the PyG implementation.

4 EXPERIMENTAL SETTING

We measure the training time of various implementations of several two-layer GNNs on node classification tasks and compare them against iSpLib. For comparison, we test model variants from PyTorch 2.1.0 (sparse), PyTorch < 2.0 (sparse), PyTorch 2 non-sparse (message passing) model, and PyTorch 2 torch.compile method against iSpLib. We select Graph Convolution Network (GCN), GraphSAGE (sum and mean), and Graph Inference Network (GIN) as they are widely used for benchmarking.

We select six large graph datasets and perform a one-dim node prediction task for 30-100 epochs while measuring the average training time. We conduct all experiments in two high-configuration CPUs: (a) an Intel Skylake CPU with 48 cores and 256GB memory, and (b) an AMD EPYC 7763 64-core Processor with 527GB memory. Table 1 presents the datasets used in our experiment.

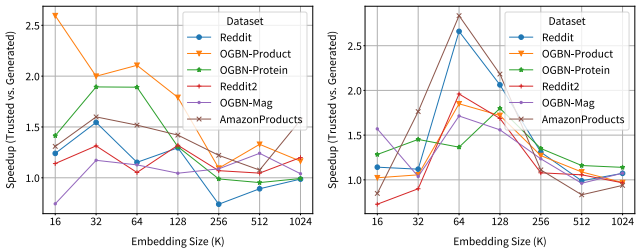
5 RESULTS

Auto-tuning results. We generate the tuning graph for both Intel and AMD CPUs and show the result in Figure 2 for embedding sizes 16, 32, 64, 128, 256, 512, and 1024 for all six datasets. These figures identify the most efficient embedding sizes for the corresponding CPUs (32 for Intel and 64 for AMD). We use these values to run the GNNs for both CPUs.

GNN training performance. We train three GNN models using iSpLib and four other settings and show the average per-epoch training time in Figure 3. Due to the limitation of space, we omit the results for GraphSAGE-MEAN. Since iSpLib is a drop-in replacement of PyTorch SpMM operation, it does not alter the results found in PyTorch. Thus the training and testing accuracy remains the same for all GNNs. Figure 3 shows that iSpLib can accelerate GNN training significantly when compared with other frameworks. However, the speedup varies across GNN models and datasets.

Performance across GNN models. Figure 3 shows that iSpLib achieves better speedup for GCN compared to GraphSAGE and GIN. This is because GCN typically performs a linear projection on the feature matrix before running the convolution. This step projects high-dimensional features into low-dimension space for which tuned kernels perform better (see Figure 2). GraphSAGE and GIN do not have the initial projection of the feature matrix and perform SpMM with original features. This makes iSpLib relatively less effective for GraphSAGE and GIN. However, for datasets that originally had a lower feature count in the feature matrix such as OGBN-Protein (feature size: 8), we observe GCN-like speedup in GraphSAGE and GIN for both Intel and AMD CPUs.

Comparison with other GNN frameworks. Additionally, we compare CogDL's [1] equivalent GCN implementation w.r.t. iSpLib and observe up to 43x speedup for various datasets. We also compare iSpLib with PyTorch 2.1 on the vanilla implementation of GCN and observe up to 93x speedup for the Reddit dataset on Intel CPU.



(a) Intel **(b) AMD**

Figure 2: Tuning Graph for various CPUs

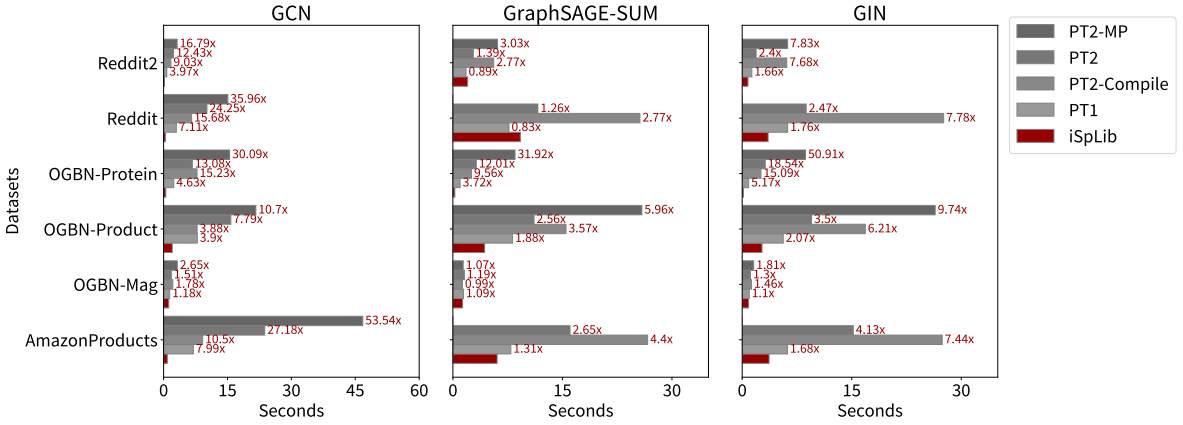
6 DISCUSSION

We observe that the embedding sizes suggested by the autotuner are generally low for our experimental environments, i.e., we have a better chance of seeing an improved performance from generated kernels for smaller embedding sizes. This is because iSpLib's generated kernels perform register blocking to reduce cache misses. For larger embedding sizes, it has to allocate more values to the register, causing register spilling and increased cache misses.

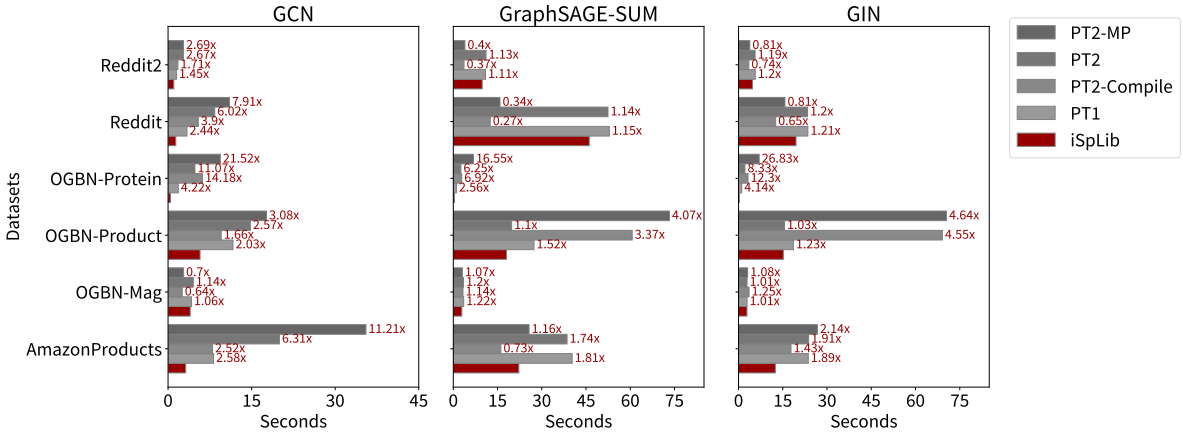
We see less overall speedup for OGB-Mag since it is a smaller graph compared to others. Caching a smaller graph has less impact on the speedup in backpropagation, thus caching the expressions does not improve the training time significantly. Nevertheless, we observe better performance in iSpLib compared to the other frameworks for most scenarios due to the usage of efficient kernels.

7 CONCLUSION

We develop a user-friendly sparse matrix library called iSpLib that works as a drop-in replacement for existing PyTorch and PyG equivalent operations. iSpLib provides auto-tuned and customized kernels for target user environments. The library also supports backpropagation that caches repetitive data during the training phase. We observe up to 27× speedup w.r.t. equivalent PyTorch 2 implementation for training larger graphs on popular GNNs.



(a) Intel



(b) AMD

Figure 3: Average per-epoch training time and speedup for iSpLib w.r.t. other frameworks [PT2: PyTorch 2.1, PT1: PyTorch < 2, PT2-Compile: PyTorch 2 torch.compile, PT2-MP: PyTorch 2 Message Passing paradigm]

ACKNOWLEDGMENTS

This research is supported by the NSF OAC-2112606 and OAC-2339607 grants and DOE DE-SC0022098 and DE-SC0023349 awards. We extend our appreciation to the PyTorch_Sparse community for open-sourcing their project. Our project has significantly benefited from their coding style and implementation.

REFERENCES

[1] Yukuo Cen, Zhenyu Hou, Yan Wang, Qibin Chen, Yizhen Luo, Xingcheng Yao, Aohan Zeng, Shiguang Guo, Peng Zhang, Guohao Dai, et al. 2021. CogDL: An extensive toolkit for deep learning on graphs. *arXiv preprint arXiv:2103.00959* 7, 8 (2021).

[2] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[3] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[4] Yuwei Hu, Zihao Ye, Minjie Wang, Jiali Yu, Da Zheng, Mu Li, Zheng Zhang, Zhiru Zhang, and Yida Wang. 2020. Featgraph: A flexible and efficient backend for graph neural network systems. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–13.

[5] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[6] Damitha Lenadora, Vimarsh Sathia, Gerasimos Gerogiannis, Serif Yesil, Josep Torrellas, and Charith Mendis. 2023. Input-sensitive dense-sparse primitive compositions for GNN acceleration. *arXiv preprint arXiv:2306.15155* (2023).

[7] Shenghao Qiu, Liang You, and Zheng Wang. 2021. Optimizing sparse matrix multiplications for graph neural networks. In *International Workshop on Languages and Compilers for Parallel Computing*. Springer, 101–117.

[8] Md Khaledur Rahman, Majedul Haque Sujon, and Ariful Azad. 2021. FusedMM: A unified sddmm-spmm kernel for graph embedding and graph neural networks. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 256–266.

[9] Rusty1s. [n. d.]. GitHub - rusty1s/pytorch_sparse: PyTorch Extension Library of Optimized Autograd Sparse Matrix Operations. https://github.com/rusty1s/pytorch_sparse

[10] Minjie Yu Wang. 2019. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*.

[11] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[12] Zihao Ye, Ruihang Lai, Junru Shao, Tianqi Chen, and Luis Ceze. 2023. SparseTIR: Composable abstractions for sparse compilation in deep learning. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 660–678.