HW-SW Co-Optimization of Graph Neural Network (GNN)

Prince Kumar
Department of ECE
National Institute Of Technology, Tiruchirappalli
Email: pk617571@gmail.com

***Abstract*—This work investigates HW-SW co-optimization to reduce the inference latencya and training time of Graph Neural Networks (GNNs). The large adjacency matrices inherent in GNNs lead to significant computational demands during inference and training. We propose a software optimization technique that re-arranges these matrices to group highly connected nodes together. These re-arranged chunks are then processed on GPU accelerator as part of our hardware optimization strategy. This combined HW-SW optimization approach aims to reduce computational overhead, introducing a trade-off between accuracy and latency. By training and doing inference on widely used GNN models (GCN, GraphSAGE, GIN, and SGC) on famous datasets Cora, CiteSeer, PubMed, we assess the impact of our method on traing time, test accuracy and inference speed. This study seeks to identify an optimal configurations that balances test accuracy and inference latency.**

***Index Terms*—Graph Neural Network, Optimization, Inference Latency, HW-SW Co-optimization**

I. Introduction

Most real-world data can be structured into entities (nodes) and relationships (edges), making graphs a crucial data structure for representing complex data. Graph Neural Networks (GNNs) have emerged as powerful tools capable of processing graph-structured data, with applications in various fields including social network analysis, recommendation systems, fraud detection, traffic prediction, and scientific discovery.

Despite their versatility, training and inferring GNNs is computationally intensive due to the large size of adjacency matrices. This work explores a HW-SW co-optimization method that introduces a trade-off between accuracy and latency.

We propose the following optimizations:

- **Software Optimization:** Partitioning the graphs into smaller highly connected sub-graphs and Re-arranging the adjacency matrices leveraging the inherent locality of information in graphs and thereby reducing computations.
- **Hardware Optimization:** Performing training and inference on GPUs or accelerators instead of traditional CPUs to further reduce computational demands.

By combining these optimizations, we aim to study the trade-off between accuracy and inference latency. Our approach will be evaluated on widely used GNN models (GCN [1], GraphSAGE [2], GIN [3], and SGC [4]) using plantoid datasets namely Cora [5], CiteSeer [6] and PubMed [5], implemented on a real computer system. We hope this study will provide valuable insights into co-optimization strategies for practical GNN applications.

II. Related Work

Recent efforts have focused on optimizing GNN computation during inference. Studies such as [7] and [8] propose novel hardware processors to enhance the efficiency of GNN inference. On the software side, [9] introduces an efficient inference system that automates the translation of GNN training code for layer-wise execution. Additionally, [10] and [11] suggest accelerating GNN inference through dimension pruning in each layer, which minimizes computational load with negligible accuracy loss.

An innovative approach by [12] presents an online propagation framework and node-adaptive methods to optimize propagation depth based on topological information, reducing redundant feature propagation. Sampling techniques to maintain accuracy while enhancing inference performance are discussed in [13] and [14]. Furthermore, [15] explores HW-SW co-optimization with a specialized hardware accelerator and dynamic kernel-to-primitive mapping to exploit matrix sparsity and reduce inference latency.

This work extends these ideas by proposing a novel HW-SW co-optimization method through adjacency matrix rearrangement, running the models on GPUs or accelerators to achieve a balance between accuracy and inference latency.

III. Methodology

The methodology involves partitioning the graph using the Multi-level Recursive Bisection algorithm [16] by using METIS tool [17], followed by training and evaluation of various GNN models. The steps are as follows:

1) Train and test GNN models (GCN, GIN, GraphSAGE, and SGC) on the original datasets with original adjacency matrix to establish baseline accuracy and latency speed.
2) Partition the graph into parts using Multi-level Recursive Bisection algorithm, with number of partitions ranging from 1 to 1000.
3) Re-arrange the adjacency matrix according to the partitions and retrain the models.
4) Evaluate the trade-off between accuracy and inference latency.
5) Repeat above steps on GPU along with CPU

A. Datasets

The experiments were conducted on three commonly used citation network datasets: Cora, CiteSeer, and PubMed from pytorch plantoid datasets [18]

1) Cora Dataset:

- **Nodes (Papers):** 2,708
- **Edges (Citations):** 10,556
- **Classes:** 7
- **Features:** Bag-of-words (1,433-dimensional)

2) CiteSeer Dataset:

- **Nodes :** 3,327
- **Edges :** 9,104
- **Classes:** 6
- **Features:** Bag-of-words (3,703-dimensional)

3) PubMed Dataset:

- **Nodes :** 19,717
- **Edges :** 88,648
- **Classes:** 3
- **Features:** Bag-of-words (500-dimensional)

B. GNN Models

The following GNN architectures were used in the experiments:

- Graph Convolutional Network (GCN)
- Graph Isomorphism Network (GIN)
- GraphSAGE
- Simple Graph Convolution (SGC)

C. Partitioning Graph and Re-arranging Adjacency Matrix

We partition the graph of the datasets into sub-graphs by removing minimum number of edges. We used METIS tool to partition the graph of the dataset. Partition leads to reduction in number of edges. We re-arrange nodes in such a way that highly connected nodes are nearby. For example, After partitioning Cora dataset into 3 partitions and re-arranging nodes, we obtain adjacency matrix looks like in Figure.1

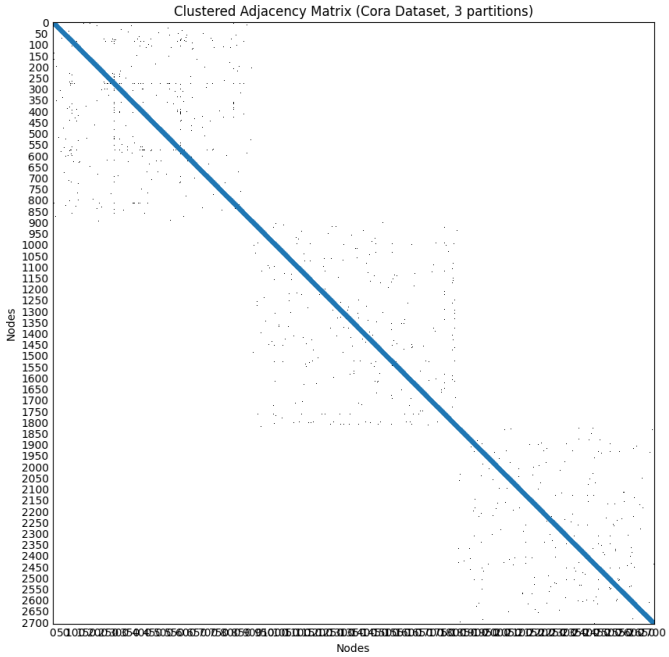

Fig. 1. Cora dataset paritioned into 3 partitions

IV. Experiments and Results

A. System Configuration

Experiments and computations were conducted using Google Colaboratory (Colab), a cloud-based Python development environment that supports free access to GPUs.

1) Hardware Configuration: The system configuration utilized in Google Colab is as follows:

- **CPU:** Intel Xeon CPU.
- **GPU:** NVIDIA Tesla T4 GPU provided by Google Colab.
- **Memory:** 12-25 GB of RAM, depending on the instance type.

2) Software Configuration:

- **Operating System:** Ubuntu 18.04 LTS (Linux-based).
- **Python Version:** Python 3.7.11.
- **Python Libraries:** PyTorch 1.9.0, torch_geometric 2.0.1, NetworkX 2.6.2, matplotlib 3.4.3.
- **Other Libraries:** NumPy, SciPy, sklearn.

3) Colab Notebook Environment: All experiments were performed in Jupyter notebooks hosted on Google Colab, accessed through a web browser interface. The notebook environment provided by Colab allowed for interactive code execution and seamless integration with Google Drive for data storage and retrieval.

4) References: For more information on Google Colab and its capabilities, refer to the official documentation:

- https://colab.research.google.com/notebooks/intro.ipynb

B. Model Parameters

For the Graph Models, we used Adam optimizer with learning rate as 0.01 and Weight Decay 5e-4. For loss calculation, Cross Entropy is used. We set Number of epochs for training as 200 based on observations. GCN, GIN and GraphSAGE models have three layers i.e., One input layer, one output layer and one hidden layer. Size of the hidden layer is 16 SGC model has no hidden layers. All the models has input layer of size equal to size of features and output layer of size equal to number of classes

C. Baseline Performance

The baseline performance was established by training the GNN models on the original adjacency matrices of the datasets over CPU. We trained the models over 200 epochs. We split the dataset into train set and test set at 8:2 ratio. We obtained the inference latency and accuracy over testing set. Training time, test accuracy and inference latency were recorded as shown in table.I, II and III.

TABLE I
CORA DATASET - BASELINE PERFORMANCE

Model	Training Time (ms)	Inference Latency (ms)	Test Accuracy
GCN	35.72	12.67	80.3%
GIN	115.80	78.79	71.8%
GraphSAGE	102.78	71.79	79.03%
SGC	23.49	9.05	80%

TABLE II
CITESEER DATASET - BASELINE PERFORMANCE

Model	Training Time (ms)	Inference Latency (ms)	Test Accuracy
GCN	39.44	17.61	68.2%
GIN	274.44	233.34	56.9%
GraphSAGE	276.99	214.24	65.8%
SGC	40.42	17.17	66.53%

D. Partitioning Impact

1) Drop in Number of edges: Partitioning the graph into sub-graphs involves the removal of edges as minimal as possible, which may lead to a decrease in the total edge count. The table IV and 2 shows the percentage of edges dropped because of partitioning.

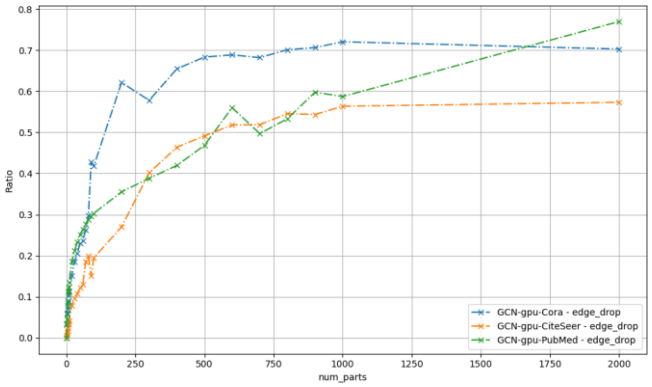

Fig. 2. Number of partitions vs Edge drop ratio

2) Test Accuracy: Partitioning the graph by dropping the edges has impact on test accuracy. In most of the cases, the impact is negative. This is because the edges plays important role along with features of each node. In very less cases, the impact is positive because the features of each node has more influence on accuracy compared to edge connections.

TABLE III
PUBMED DATASET - BASELINE PERFORMANCE

Model	Training Time (ms)	Inference Latency (ms)	Test Accuracy
GCN	68.52	30.81	78.03%
GIN	349.44	311.60	74.8%
GraphSAGE	328.91	253.80	76.33%
SGC	31.30	12.66	77.9%

TABLE IV
NUMBER OF EDGES REMOVED AGAINST THE NUMBER OF PARTITIONS

sub-graphs	Cora	CiteSeer	PubMed
2	378 (3.58%)	52 (0.57%)	2770 (3.12%)
3	616 (5.84%)	54 (0.59%)	3954 (4.46%)
4	650 (6.16%)	122 (1.34%)	5148 (5.81%)
5	744 (7.05%)	152 (1.67%)	6776 (7.64%)
6	904 (8.56%)	224 (2.46%)	8020 (9.05%)
7	952 (9.02%)	308 (3.38%)	9688 (10.93%)
8	1096 (10.38%)	302 (3.32%)	10158 (11.46%)
9	1186 (11.24%)	356 (3.91%)	10824 (12.21%)
10	1208 (11.44%)	372 (4.09%)	11938 (13.47%)
20	1598 (15.14%)	722 (7.93%)	16500 (18.61%)
30	1956 (18.53%)	874 (9.60%)	18800 (21.21%)
40	2174 (20.59%)	980 (10.76%)	20724 (23.38%)
50	2424 (22.96%)	1110 (12.19%)	22316 (25.17%)
60	2502 (23.70%)	1182 (12.98%)	23420 (26.42%)
70	2768 (26.22%)	1682 (18.48%)	24578 (27.73%)
80	3134 (29.69%)	1820 (19.99%)	25506 (28.77%)
90	4512 (42.74%)	1378 (15.14%)	26228 (29.59%)
100	4418 (41.85%)	1774 (19.49%)	26886 (30.33%)
200	6558 (62.13%)	2458 (27.00%)	31484 (35.52%)
300	6100 (57.79%)	3658 (40.18%)	34426 (38.83%)
400	6908 (65.44%)	4222 (46.38%)	37180 (41.94%)
500	7212 (68.32%)	4474 (49.14%)	41522 (46.84%)
600	7268 (68.85%)	4716 (51.80%)	49656 (56.01%)
700	7198 (68.19%)	4722 (51.87%)	44066 (49.71%)
800	7400 (70.10%)	4968 (54.57%)	47260 (53.31%)
900	7456 (70.63%)	4946 (54.33%)	52986 (59.77%)
1000	7604 (72.03%)	5130 (56.35%)	52054 (58.72%)
2000	7418 (70.27%)	5218 (57.32%)	68220 (76.96%)
3000	-	5254 (57.71%)	72110 (81.34%)
4000	-	-	73584 (83.01%)
5000	-	-	75166 (84.79%)
6000	-	-	75722 (85.42%)
7000	-	-	76352 (86.13%)
8000	-	-	76688 (86.51%)
9000	-	-	76966 (86.82%)
10000	-	-	76876 (86.72%)

In Cora dataset, With the number of partitions less that 10 (11.44% edge drop) , The drop in accuracy is less than 5% which is manageable. But after the 10 number of partitions, the accuracy drop is high and it keeps increasing as shown in figure. 3.

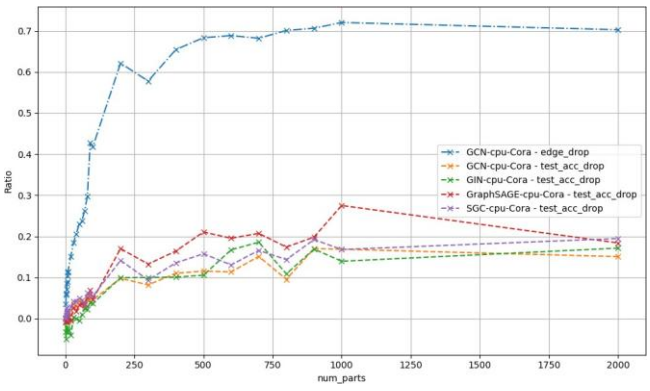

Fig. 3. Cora - Number of partitions vs Drop in test accuracy ratio

In CiteSeer dataset, With the number of partitions less that 400 (46.38% edge drop), The drop in accuracy is less than 5% which is manageable. This is because the edges of the graph

has less impact on test accuracy compared to features of each node. And after 400 number of partitions, the accuracy drop is increasing with low rate of change as shown in Figure. 4.

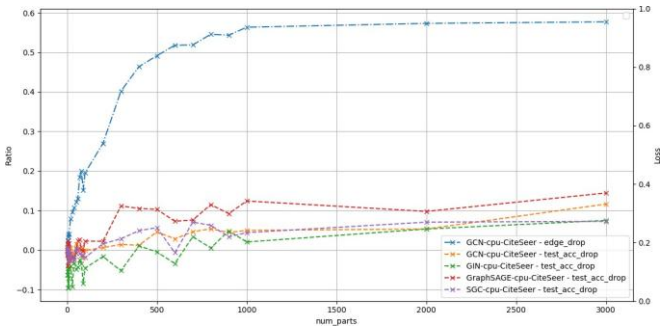

Fig. 4. CiteSeer - Number of partitions vs Drop in test accuracy ratio

In PubMed dataset, With the number of partitions less that 100 (30.33% edge drop), The drop in accuracy us less than 5% which is manageable. This is because the edges of the graph has less impact on test accuracy compared to features of each node. And after 100 number of partitions, the accuracy drop is increasing with low rate of change as shown in Figure. 5 .

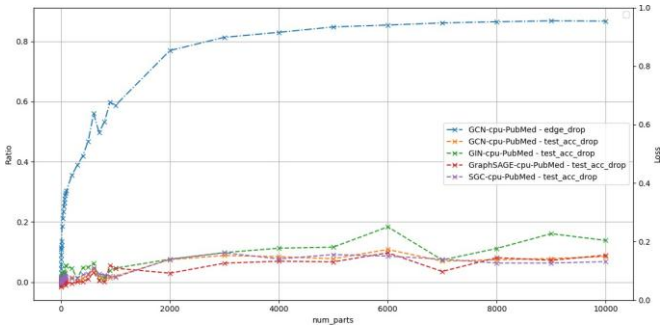

Fig. 5. PubMed - Number of partitions vs Drop in test accuracy ratio

3) Training time: Partitioning the graph by dropping the edges has positive impact on training time i.e., training time is reduced due to drop in edges. Aggregation and Combination are the two main phases of GNN for training and testing. Reduction in training time is because, with less number of edges, the time taken for aggregation phase is lesser. But it does not effect combination phase.

In Cora dataset, the drop is training time is more observed after 30% of edges dropped as shows in fig.6.

In CiteSeer dataset, the drop in training time is more observed after 25% of edges dropped as shown in fig. 7.

In PubMed dataset, the drop in training time is more observed after 20% of edges dropped as shown in fig. 8.

4) Inference Latency: Partitioning the graph by dropping the edges has positive impact on inference latency i.e., time taken for forward pass is reduced due to drop in edges. This is because , with less number of edges, the time taken for aggregation phase is less. But it does not effect combination phase.

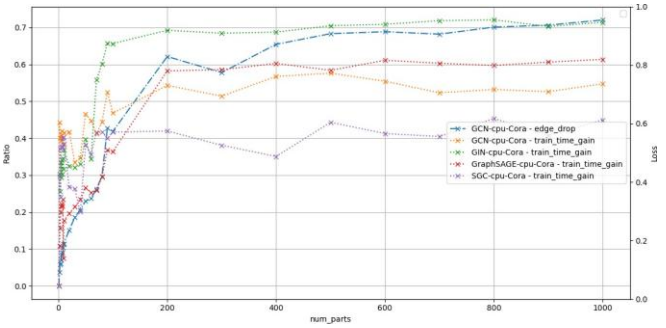

Fig. 6. Cora - Number of partitions vs Drop in Training Time

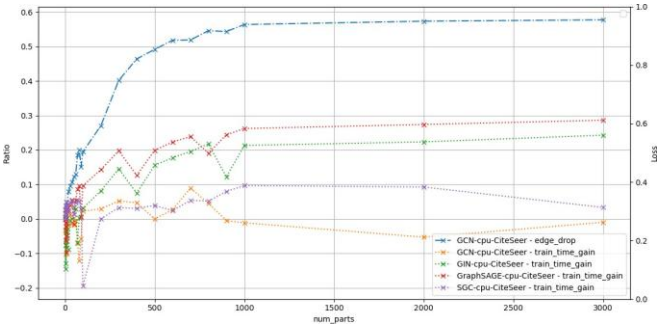

Fig. 7. CiteSeer - Number of partitions vs Drop in Training Time

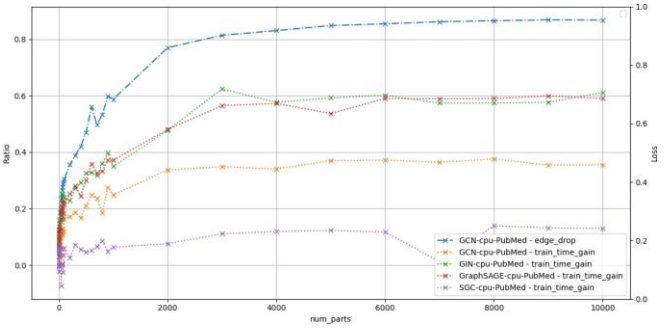

Fig. 8. PubMed - Number of partitions vs Drop in Training Time

For Cora Dataset, the drop in inference latency is observed with number of partitions greater than 2 i.e., above 5% for edge drop as shown in figure. 9. Drop in inference latency is saturates at 60% after 200 partitions (62% edge drop).

For CiteSeer Dataset, the drop in inference latency is observed with number of partitions greater than 200 i.e., above 27% for edge drop as shown in figure. 9. Drop in inference latency is saturates at 20% after 500 partitions (50% edge drop).

For PubMed Dataset, the drop in inference latency is observed with number of partitions greater than 10 i.e., above 13.47% for edge drop as shown in figure. 11. Drop in inference latency is saturates at 60% after 2000 partitions (77% edge drop).

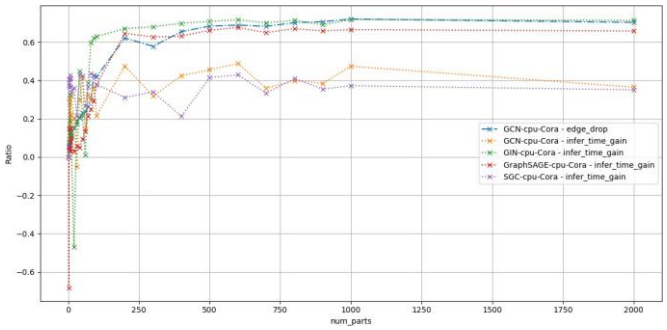

Fig. 9. Cora - Number of partitions vs Drop in Inference latency ratio

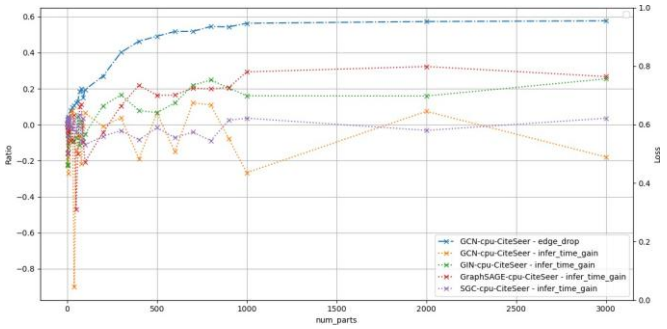

Fig. 10. CiteSeer - Number of partitions vs Drop in Inferency latency ratio

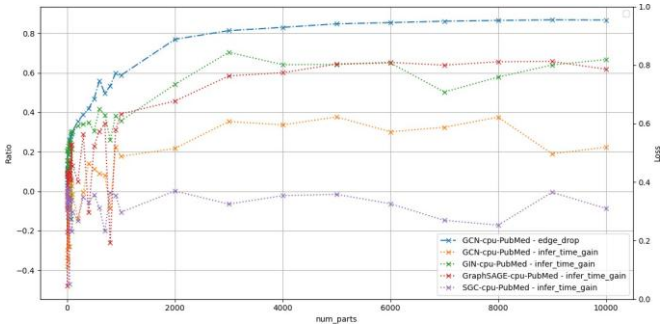

Fig. 11. PubMed - Number of partitions vs Drop in Inferency latency ratio

5) Conclusion: The impact of graph partitioning has clear impact on test accuracy, training time and inference latency. And the optimal number of partitions is subjective to the datasets and graph models used. For the dataset and models used in this work, the optimal number of partitions considering the good drop on inference latency at the cost of lesser test accuracy loss is as shown in the table. V

TABLE V
OPTIMAL NUMBER OF PARTITIONS

Dataset	sub-graphs (edge drop)	Testing Accuracy Loss	Training time gain	Inference Latency gain
Cora	7 (9.02%)	2%	20-40%	10-40%
CiteSeer	300 (40.18%)	4%	5-40%	2-18%
PubMed	70 (27.73%)	5%	2-28%	2-30%

TABLE VI
CORA DATASET - TRAINING TIME GAIN OVER GPU

Model	Training Time On CPU (ms)	Training Time On GPU (ms)	Gain
GCN	21.47	3.56	83.40%
GIN	80.72	3.34	95.85%
GraphSAGE	80.62	3.46	95.70%
SGC	14.62	1.43	90.18%

TABLE VII
CORA DATASET - INFERENCE LATENCY GAIN OVER GPU

Model	CPU Inference Latency (ms)	GPU Infernce Latency (ms)	Gain
GCN	9.88	1.71	82.61%
GIN	52.87	0.83	98.42%
GraphSAGE	61.38	1.09	98.21%
SGC	5.19	0.11	97.78%

E. GPU acceleration

We trained and tested the same models over GPU with optimal partitions as mentioned in table.V. And We obtained the training time gain and inference latency gain.

1) Cora Dataset: For Cora dataset, training time gain is ranging from 83.4% to 95.85% (as shown in table. VI) while inference latency gain ranging from 82.61% to 98.42% as shown in table.VII.

2) CiteSeer Dataset: For CiteSeer dataset, training time gain is ranging from 91.88% to 98.05% (as shown in table. VIII) while inference latency gain ranging from 91.49% to 99.7% as shown in table.IX.

3) PubMed Dataset: For PubMed dataset, training time gain is ranging from 95.03% to 97.89% (as shown in table. X) while inference latency gain ranging from 95.7% to 99.74% as shown in table.XI.

V. CONCLUSION AND FUTURE WORK

This work demonstrated the potential of HW-SW co-optimization for GNNs through graph partitioning. The re-arrangement of adjacency matrices significantly reduced training time and inference latency with a manageable loss in

TABLE VIII
CITESEER DATASET - TRAINING TIME GAIN OVER GPU

Model	Training Time On CPU (ms)	Training Time On GPU (ms)	Gain
GCN	40.66	3.30	91.88%
GIN	292.05	5.97	97.96%
GraphSAGE	304.20	5.92	98.05%
SGC	39.03	1.67	95.72%

TABLE IX
CITESEER DATASET - INFERENCE LATENCY GAIN OVER GPU

Model	CPU Inference Latency (ms)	GPU Infernce Latency (ms)	Gain
GCN	19.46	1.66	91.49%
GIN	245.93	0.74	99.70%
GraphSAGE	223.38	0.89	99.60%
SGC	17.50	0.15	99.15%

TABLE X
PUBMED DATASET - TRAINING TIME GAIN OVER GPU

Model	Training Time On CPU (ms)	Training Time On GPU (ms)	Gain
GCN	65.02	3.23	95.03%
GIN	315.32	6.64	97.89%
GraphSAGE	306.70	6.46	97.89%
SGC	31.48	1.55	95.07%

TABLE XI
PUBMED DATASET - INFERENCE LATENCY GAIN OVER GPU

Model	CPU Inference Latency (ms)	GPU Infernce Latency (ms)	Gain
GCN	41.06	1.77	95.70%
GIN	282.99	0.74	99.74%
GraphSAGE	306.35	0.92	99.70%
SGC	13.51	0.11	99.21%

accuracy. Additionally, Training and Testing on GPU has gained more advantage. Future work will focus on exploring more advanced partitioning techniques and their impact on different GNN architectures. Additionally, real-world deployment scenarios and hardware accelerators will be considered to further enhance the performance of GNNs.

VI. REFERENCES

REFERENCES

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017. [Online]. Available: https://arxiv.org/abs/1609.02907

[2] W. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. [Online]. Available: https://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs

[3] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations (ICLR)*, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km

[4] J. Li, M.-T. Luong, D. Jurafsky, and E. Hovy, "Simplified graph convolutional network," *arXiv preprint arXiv:1902.07153*, 2020. [Online]. Available: https://arxiv.org/abs/1902.07153

[5] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, p. 93, Sep. 2008. [Online]. Available: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2157

[6] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: https://networkrepository.com

[7] K. Kiningham, P. Levis, and C. Ré, "Grip: A graph neural network accelerator architecture," *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 914–925, 2023.

[8] K. Chen, H. Su, C. Liu, and Y. Li, "Optimizing gnn inference processing on very long vector processor," in *Algorithms and Architectures for Parallel Processing*. Springer, 2024, pp. 214–229.

[9] P. Yin, X. Yan, J. Zhou, Q. Fu, Z. Cai, J. Cheng, B. Tang, and M. Wang, "Dgi: Easy and efficient inference for gnns," *arXiv preprint arXiv:2211.15082*, 2022.

[10] H. Zhou, A. Srivastava, H. Zeng, R. Kannan, and V. Prasanna, "Accelerating large scale real-time gnn inference using channel pruning," in *Proc. VLDB Endow.*, vol. 14, no. 9, 2021, pp. 1597–1605.

[11] W. Zhang, J. Sun, and G. Sun, "Accelerating gnn inference by soft channel pruning," in *2022 IEEE 13th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, 2022, pp. 1–6.

[12] X. Gao, W. Zhang, J. Yu, Y. Shao, Q. V. Nguyen, B. Cui, and H. Yin, "Accelerating scalable graph neural network inference with node-adaptive propagation," 2024.

[13] T. Kaler, N. Stathas, A. Ouyang, A. Iliopoulos, T. Schardl, C. Leiserson, and J. Chen, "Accelerating training and inference of graph neural networks with fast sampling and pipelining," 2022.

[14] Z. Wang, Y. Wang, C. Yuan, R. Gu, and Y. Huang, "Empirical analysis of performance bottlenecks in graph neural network training and inference with gpus," *Neurocomputing*, vol. 446, pp. 195–206, 2021.

[15] B. Zhang and V. Prasanna, "Dynasparse: Accelerating gnn inference through dynamic sparsity exploitation," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2023, pp. 233–244.

[16] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," in *SIAM Journal on Scientific Computing*, vol. 20, no. 1. Society for Industrial and Applied Mathematics, 1998, pp. 359–392. [Online]. Available: https://doi.org/10.1137/S1064827595287997

[17] Karypis, George and Kumar, Vipin, "METIS: Serial graph partitioning and fill-reducing matrix ordering," http://glaros.dtc.umn.edu/gkhome/metis/metis/overview, Accessed: 2024-06-29.

[18] PyTorch Geometric Contributors, "PyTorch Geometric Documentation," https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.datasets.Planetoid.html, Accessed: 2024-06-29.