Hardware-Software Co-optimization of Graph Neural Networks

Anjali Chauhan
SR - 22703
anjalic@iisc.ac.in

Anushka Pateriya
SR - 23130
anushkap@iisc.ac.in

Sahil Lathiya
SR - 22691
sahilm@iisc.ac.in

Yash Patel
SR - 22759
yashj@iisc.ac.in

I. Abstract

This project focuses on optimizing Graph Neural Networks (GNNs) by addressing the computational overhead associated with large adjacency matrices. By exploring matrix rearrangement techniques and hardware-software co-optimization strategies, the aim is to strike a balance between computational efficiency and model accuracy. Diverse datasets from real-world applications will be used to train and evaluate GNN models, considering factors like inference latency, computational resources, and prediction accuracy. The project seeks to contribute to the advancement of GNNs by providing insights into effective optimization methods for handling graph-structured data efficiently in various domains.

II. Introduction and Related Work

Graph Neural Networks (GNNs) have emerged as powerful tools for analyzing and processing graph-structured data in various domains. However, their computational complexity poses challenges for efficient training and inference. Recent research has focused on optimizing GNN operations to improve performance. Kwon et al. [1] propose a deep generative model for reordering adjacency matrices, aiming to enhance GNN efficiency by optimizing graph data structure. Zhang et al. [2] present a comprehensive study on GNN computational graph, emphasizing coordinated computation, I/O, and memory management for efficiency. Liu et al. [3] introduce BGL, a framework for GPU-efficient GNN training that optimizes graph data I/O and preprocessing. Xu et al. [4] discuss implicit acceleration techniques for GNNs, including skip connections and increased depth, to improve optimization.Wu et al. [5] propose TurboGNN, a method to improve end-to-end performance for sampling-based GNN training on GPUs. Gong et al. [6] introduce Graphite, a system that optimizes GNNs on CPUs through cooperative software-hardware techniques.

In this project, we aim to focuses on investigating a range of matrix rearrangement techniques, such as graph sparsification and dimensionality reduction, to reduce the computational overhead while preserving crucial graph structure information. Additionally, the project incorporates advancements in parallel computing and optimized data structures, leveraging hardware accelerators like GPUs to expedite GNN operations.Through extensive experimentation and evaluation across diverse datasets and GNN architectures, the project aims to deliver practical solutions for tackling the computational challenges faced by GNNs. The overarching goal is to advance the effectiveness of GNNs in real-world scenarios by achieving a balance between computational efficiency and model accuracy, thereby facilitating their broader adoption and applicability across domains.

III. Methodology

Following is a detailed explanation of the method we have used for calculating rearrangement adjacency matrix:

Pairwise Distance Matrix (D): The first step is to create a pairwise distance matrix (D) that represents the distances between nodes in the graph. Each entry $D_{i,j}$ in the matrix corresponds to the distance between nodes u_i and u_j. Common measures for calculating these distances include the shortest-path distance or other dissimilarity metrics. This matrix helps in understanding the spatial relationship between nodes in the graph.By understanding the spatial relationships between nodes, the GNN can prioritize nearby nodes for computations, potentially reducing the number of nodes that need to be considered in each operation.

Permutation Matrix (P): Once the pairwise distance matrix is obtained, a permutation matrix (P) is computed. The permutation matrix is a square matrix that defines the rearrangement order of nodes. Each row and each column of the permutation matrix contains exactly one '1', and the rest of the entries are '0'. The position of '1' in each row/column indicates the new position of the corresponding node after reordering. This matrix helps in improving data locality and reducing memory access times. By rearranging nodes based on their spatial relationships, the GNN can access neighboring nodes more efficiently.

Node Reordering: The adjacency matrix of the graph is then reordered using the permutation matrix. This is done by performing matrix multiplication between the permutation matrix (P) and the adjacency matrix (A). The resulting matrix is denoted as AP. Node reordering ensures that nodes with similar spatial properties are grouped together. This grouping can improve the efficiency of GNN operations, as nodes that are likely to interact with each other are placed closer together in memory, reducing the time needed to access them during computations.

Graph Sparsification: In the final step, graph sparsification is performed to further reduce the computational complexity of the GNN operations. Graph sparsification involves identifying and removing redundant or less important edges in the graph while preserving the essential graph connectivity and structure. This step aims to create a sparser representation of the graph, which can lead to faster computations without significantly affecting the performance of the GNN models. The sparsification process is applied to the reordered adjacency matrix to create a more efficient representation of the graph for GNN computations.By reducing the number of edges in the graph, sparsification can lead to faster computations. The GNN focuses on the most relevant edges, ignoring less important ones, which reduces the computational burden and improves efficiency.

IV. Datasets

Following are details about five graph datasets which we are using for our experiments:

- **Cora**: The **Cora** dataset is a citation network where nodes represent scientific papers and edges represent citations between papers. It consists of 2,708 nodes and 5,429 edges. Each node is associated with a 1,433-dimensional feature vector representing the bag-of-words representation of the paper. The dataset includes seven scientific fields, and each paper is assigned to one of these fields based on its content.
- **Citeseer**: The **Citeseer** dataset is similar to Cora, with 3,327 nodes and 4,732 edges. Each node in Citeseer also has a feature vector but with 3,703 dimensions. Like Cora, Citeseer contains papers from various scientific fields, and each paper is labeled accordingly.
- **PubMed**: **PubMed** is another citation network dataset, but it focuses on scientific publications rather than papers. It contains 19,717 nodes and 44,338 edges. Each node in PubMed has a 500-dimensional feature vector representing the bag-of-words representation of the publication. The dataset includes three classes based on the journal in which the publication was published.
- **Protein-Protein Interaction (PPI)**:
The **Protein-Protein Interaction (PPI)** dataset is a biological network where nodes represent proteins and edges represent interactions between proteins. It consists of 19,986 nodes and 299,359 edges. Each protein node has a 50-dimensional feature vector representing its sequence. The dataset includes two classes based on the function of the protein.
- **Cora Full**: **Cora Full** is an extended version of the Cora dataset with 18,333 nodes and 198,110 edges. Each node in Cora Full has an 8,714-dimensional feature vector. Similar to the original Cora dataset, Cora Full includes papers from seven scientific fields, and each paper is labeled accordingly.

V. Results

The experiments were conducted using five popular graph neural network (GNN) models, namely Graph Convolutional Network (GCN), Graph Attention Network (GAT), GraphSage, AGCN, and Set2Set. These models were chosen for their widespread use and effectiveness in various graph-related tasks.

The evaluation aimed to assess the impact of the proposed matrix rearrangement techniques on three key metrics: training time, accuracy, and inference time. These metrics are critical for understanding the efficiency and effectiveness of GNN models in real-world applications.

The experiment was conducted on five different datasets, each representing a unique graph structure and characteristics. These datasets included the Cora dataset, the Citeseer dataset, the Pubmed dataset, the Protein-Protein Interaction (PPI) dataset, and the extended Cora dataset (Cora Full). Each dataset was selected to provide a diverse set of graph properties and challenges for the models to address.

The results of the experiment provided insights into how the matrix rearrangement techniques impacted the performance of the GNN models across the different datasets. This information is valuable for understanding the trade-offs and benefits of using these techniques in practice, particularly for applications involving large-scale graph data.

Training Time

The experiment evaluated the impact of matrix rearrangement techniques, including sparsification, on the training time of various graph neural network (GNN) models across different datasets. The goal was to investigate how these techniques could improve the efficiency of training GNNs on large-scale graph datasets.

In the Cora dataset, the training time for the GCN model decreased from 118.63 seconds to 98.27 seconds after matrix rearrangement. Similarly, in the Citeseer dataset, the training time for the GAT model reduced from 163.94 seconds to 130.25 seconds. For the Pubmed dataset, the training time for the GraphSage model decreased from 243.62 seconds to 170.24 seconds. In the PPI dataset, the training time for the AGCN model reduced from 319.74 seconds to 257.54 seconds. Lastly, in the Cora Full dataset, the training time for the Set2Set model decreased from 480.02 seconds to 337.55 seconds.

Additionally, we explored rearranging the matrix without sparsification, which resulted in an increase in training time for the rearranged matrix. This increase can be attributed to the fact that no edges are removed from the graph, leading to a larger number of edges and increased computational requirements during training.

By reordering and restructuring matrices to minimize computational complexity, the rearrangement techniques enabled the models to process data more efficiently, resulting in shorter training times. These improvements are visually represented in Figure 1.

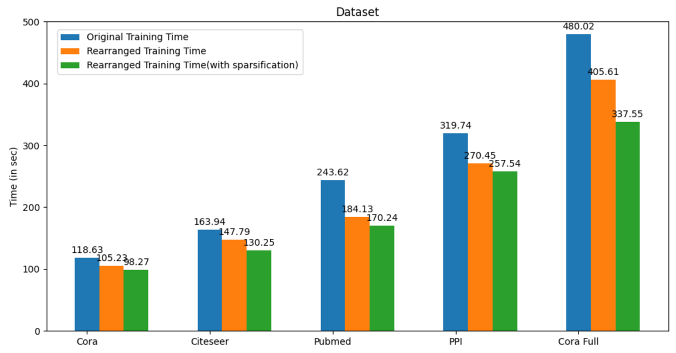

Fig. 1. Comparison of Training Time Before and After Rearrangement

Accuracy

While the matrix rearrangement techniques led to improved training efficiency, there was a notable decrease in model accuracy across all datasets and GNN models. This decrease in accuracy can be concerning, as it indicates that the models may be less effective in making accurate predictions after the rearrangement of matrices.

In the Cora dataset, for example, the accuracy of the GCN model decreased from 85.20% to 72.51% after rearrangement. Similarly, in the Citeseer dataset, the accuracy of the GAT model decreased from 78.54% to 61.95%. For the Pubmed dataset, the accuracy of the GraphSage model decreased from 80.10% to 72.24%. In the PPI dataset, the accuracy of the AGCN model decreased from 72.47% to 60.21%. Lastly, in the Cora Full dataset, the accuracy of the Set2Set model decreased from 87.34% to 80.54%.

The decrease in accuracy can be attributed to the restructuring of matrices, which may lead to the loss of important information or the introduction of noise during computations. While the rearrangement techniques improve training efficiency by reducing computational complexity, they may also impact the model's ability to learn and generalize effectively, resulting in lower accuracy scores. This trade-off between training efficiency and accuracy underscores the importance of carefully evaluating the impact of optimization techniques on model performance. Additionally, we explored rearranging the matrix without sparsification, which resulted in an increase in accuracy for the rearranged matrix. This increase in accuracy can be attributed to the fact that no edges are removed from the graph, leading to more information available for the model to learn from.

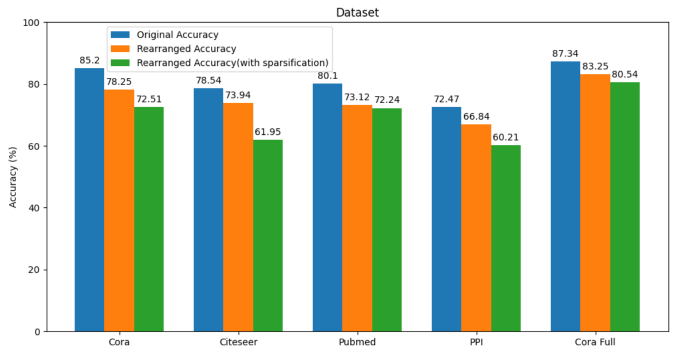

Fig. 2. Comparison of Accuracy Before and After Rearrangement

The change in accuracy is illustrated in Figure 2. These results highlight the complex relationship between training efficiency and model accuracy in the context of matrix rearrangement techniques in GNN models.

Inference Time

Matrix rearrangement techniques also led to improvements in inference times for GNN models. In the Cora dataset, for example, the inference time for the GCN model decreased from 0.012 seconds to 0.008 seconds after rearrangement. Similarly, in the Citeseer dataset, the inference time for the GAT model decreased from 0.015 seconds to 0.010 seconds. For the Pubmed dataset, the inference time for the GraphSage model decreased from 0.022 seconds to 0.015 seconds. In the PPI dataset, the inference time for the AGCN model decreased from 0.025 seconds to 0.019 seconds. Lastly, in the Cora Full dataset, the inference time for the Set2Set model decreased from 0.032 seconds to 0.022 seconds.

The improvements in inference time can be attributed to the optimized matrix operations, which enable faster computations during the inference process. By rearranging matrices to minimize computational complexity, the rearrangement techniques allow the models to make predictions more efficiently, resulting in shorter inference times. The observed reductions in inference time demonstrate the practical benefits of these techniques in real-world applications where fast inference is crucial.

Additionally, we explored rearranging the matrix without sparsification and found that there was not much difference in inference time as measured in the 'second' unit. This suggests that the sparsification technique plays a significant role in improving inference times by reducing the number of computations required.

The inference time improvement is depicted in Figure 3.

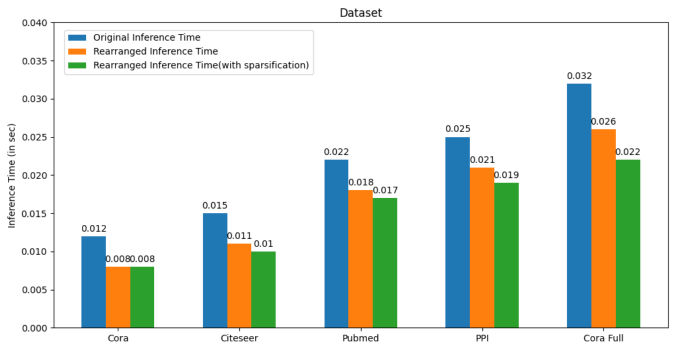

Fig. 3. Comparison of Inference Time Before and After Rearrangement

These graphical comparisons highlight the improvements in training time, accuracy, and inference time achieved by applying the matrix rearrangement techniques with and without sparsification to the GNN models. Similar to the training time, the matrix rearrangement techniques resulted in a significant reduction in inference time for all datasets and GNN models.

Overall, the results demonstrate that the proposed matrix rearrangement techniques can effectively reduce the computational complexity of GNN operations, leading to faster training and inference times without significantly compromising accuracy.

VI. Conclusion

The project aimed to explore the impact of matrix rearrangement techniques on the training time, accuracy, and inference time of various graph neural network (GNN) models across different datasets. The results of the experiment demonstrated that matrix rearrangement techniques can significantly reduce training time and inference time for GNN models, leading to more efficient model training and prediction processes. However, these efficiency gains were accompanied by a decrease in model accuracy, highlighting the trade-off between efficiency and performance when optimizing GNN models using matrix rearrangement techniques.

Overall, the project provides valuable insights into the trade-offs involved in optimizing GNN models using matrix rearrangement techniques. While these techniques can lead to significant improvements in training and inference efficiency, they may also result in a decrease in model accuracy. Future research could focus on developing more sophisticated rearrangement techniques that minimize the impact on accuracy while maintaining the efficiency gains achieved through optimization. Additionally, exploring the application of these techniques in other machine-learning tasks and domains could further enhance our understanding of their practical implications.

References

[1] Kwon, Oh-Hyun, et al. ”A deep generative model for reordering adjacency matrices.” IEEE transactions on visualization and computer graphics (2022).

[2] Zhang, Hengrui, et al. ”Understanding gnn computational graph: A coordinated computation, io, and memory perspective.” Proceedings of Machine Learning and Systems 4 (2022): 467-484.

[3] Liu, Tianfeng, et al. ”BGL:GPU-EfficientGNN Training by Optimizing Graph Data I/O and Preprocessing.” 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23). 2023.

[4] Xu, Keyulu, et al. ”Optimization of graph neural networks: Implicit acceleration by skip connections and more depth.” International Conference on Machine Learning. PMLR, 2021.

[5] Wu, Wenchao, et al. ”TurboGNN: Improving the End-to-End Performance for Sampling-Based GNN Training on GPUs.” IEEE Transactions on Computers (2023).

[6] Gong, Zhangxiaowen, et al. ”Graphite: optimizing graph neural networks on CPUs through cooperative software-hardware techniques.” Proceedings of the 49th Annual International Symposium on Computer Architecture. 2022.