PROJECT Phase I

Ampolu Sahithi (12040130) Athulitha Yashaswini (12040610)
Chandramalika Ravipati (12041210)

April 7, 2024

TLPGNN: A Lightweight Two-Level Parallelism Paradigm for Graph Neural Network Computation on GPU

Problem Statement

We aim to boost the efficiency of the convolution operation in Graph Neural Networks (GNNs), as it accounts for up to 60% of total computation time as stated in the paper. Enhancing this process will significantly improve overall GNN performance. We'll focus on optimizing how GNNs analyze and process data, particularly in the convolution step. By making this operation more efficient, we can speed up the GNN's ability to understand and learn from complex data structures like graphs. This optimization effort involves refining the algorithms and techniques used in convolution to reduce computational overhead and streamline processing. Ultimately, the goal is to make GNNs faster and more effective at tasks like node classification, link prediction, and graph embedding.

Approach and Implementation

The paper we took for reference, TLPGNN, has performed various tasks, starting from comparing the disadvantages of excessive atomic operations, using more kernel functions and non-coalesced memory access, and introducing a two-level parallelism code for better computation of the convolution operation used in GNN models.

We cloned the GitHub code and made the necessary changes to run the code in the Google Colab cells. Initially, we faced issues with the dependencies.

We also used DGL library GCN and GAT models to compare the speed-up provided by the TLPGNN code. Created a single convolution layer model for both GAT and GCN models, the time it took to run is presented in the table below. The TLPGNN uses both vertex parallelism and feature parallelism. By vertex parallelism, we mean each vertex is given to a warp, and it performs better than each vertex given to a thread because it reduces the divergence issue. The second level of parallelism is feature parallelism, where all the threads in the warp first process the feature vector of one vertex and then go to the other. This enables memory coalescing because all the threads in a warp are accessing adjacent memory locations.

Preliminary Result - 1:

Graph Model	**Dataset Name**	**Dataset size**	**Runtime with TLPGNN**	**Runtime with DGL library**	**Speed up**
GCN	Citeseer	3327×3703	1.781	12.137	6.815

Table 1: Preliminary Result - 1

The dataset size in the table is number of nodes x size of the feature vector.
From the table, we can conclude that TLPGNN has quite better convolution performance than the DGL library as claimed in the paper. But, when we observe, there is library overhead in DGL as it calls many different functions to perform various operations. We tried to simplify the DGL library code as much as possible, but still, there is library overhead, giving a speedup of 6.815. There has been no mention of the library overhead in the paper. According to the values given in the paper, the speedup for TLPGNN over DGL for GCN is $\left(\frac{0.4}{0.026}\right) = 15.385$.

Preliminary Result - 2:

Graph Model	**Dataset Name**	**Dataset size**	**Runtime with TLPGNN**	**Runtime with DGL library**	**Speed up**
GAT	Citeseer	3327×3703	3.289	96.841	29.444

Table 2: Preliminary Result - 2

We also started to apply more optimizations like using shared memory for the TLPGNN code. But as of now, it is incomplete.

References

- TLPGNN: `https://github.com/charlifu/TLPGNN`
- DGL: `https://github.com/dmlc/dgl`
- Citeseer: `https://github.com/charlifu/TLPGNN/tree/main/data/citeseer`