World Economics & Finance Bulletin (WEFB)
Available Online at: https://www.scholarexpress.net
Vol. 31, February, 2024
ISSN: 2749-3628,

ACCELERATING GRAPH NEURAL NETWORK TRAINING WITH COMMUNITY-BASED SAMPLING TECHNIQUES

Hamidullah Mahzon
master student of the faculty of Software Engineering
China West Normal University
Supervisor: **He Jialin**

Article history:

Received:	4th December 2023
Accepted:	7th January 2024
Published:	14th February 2024

Abstract:

This paper presents a GNNear accelerator to tackle these challenges. GNNear adopts a DIMM-based memory system to provide sufficient memory capacity. To match the heterogeneous nature of GNN training, we offload the memory-intensive Reduce operations to in-DIMM Near-Memory-Engines (NMEs), making full use of the high aggregated local bandwidth. Recently, Graph Neural Networks (GNNs) have become state-of-the-art algorithms for analyzing non-euclidean graph data. However, to realize efficient GNN training is challenging, especially on large graphs. The reasons are many-folded: 1) GNN training incurs a substantial memory footprint. Full-batch training on large graphs even requires hundreds to thousands of gigabytes of memory. 2) GNN training involves both memory-intensive and computation-intensive operations, challenging current CPU/GPU platforms. 3) The irregularity of graphs can result in severe resource under-utilization and load-imbalance problems.

Keywords: Graph neural networks, distributed training, Billion-scale graphs, accelerating graph neural, network training**.**

1 INTRODUCTION

When a graph is too big to fit in the memory of a single training machine, the graph is often partitioned across multiple machines, and inter-machine communication is used to request and provide the relevant graph data needed by each machine to train the GNN model. We have observed that, oftentimes, the features associated with the graph nodes take up the bulk of the graph representation size. Often, the node features take up more than 90% of the memory needed to represent the graph.

Guided by this observation, we designed a new partitioning strategy, hybrid partitioning, that replicates the relatively small graph topology information (the graph's adjacency matrix) across all training machines, and only partitions the graph's node features, as illustrated. Graph Neural Networks (GNNs) have achieved great success across different graph-related tasks (Hamilton et al., 2017; Hu et al., 2020; Ying et al., 2018; Jiang et al., 2022; Zhou et al., 2022; 2023). However, despite its effectiveness, the training of GNNs is very time-consuming. Specifically, GNNs are characterized by an interleaved execution that switches between the aggregation and update phases. Namely, in the aggregation phase, every node aggregates messages from its neighborhoods at each layer, which is implemented based on sparse matrix-based operations (Fey & Lenssen, 2019; Wang et al., 2019). In the update phase, each node will update its embedding based on the aggregated messages, where the update function is implemented with dense matrix-based operations (Fey & Lenssen, 2019; Wang et al., 2019). In Figure 1, SpMM and MatMul are the sparse and dense operations in the aggregation and update phases, respectively. Through profiling, we found that the aggregation phase may take more than 90% running time for GNN training. This is because the sparse matrix operations in the aggregation phase have many random memory accesses and limited data reuse, which is hard to be accelerated by community hardwares (e.g., CPUs and GPUs) (Duan et al., 2022b; Han et al., 2016; Duan et al., 2022a). Thus, training GNNs with large graphs is often time-inefficient. Because of the ability to learn both the structure and attributes of the graphs at the same time, Graph neural networks (GNN) is widely used in many fields such as node classification and link prediction in recommendation systems social networks biomedical science knowledge graph. Since training GNN is a very time- and resource-consuming task general-purpose graphics processing units (GPUs) are often used to accelerate the training process. Several general

World Economics & Finance Bulletin (WEFB)
Available Online at: https://www.scholarexpress.net
Vol. 31, February, 2024
ISSN: 2749-3628,

GNN learning frameworks have been developed, such as.

The core computations in GNN training come from the continuous information gathering from neighboring vertices and updating the vertices' feature vectors through a neural network. Multiple such layers can be stacked to aggregate multiple hop messages. Usually, a GNN has 2-3 layers. Challenges still remain to train GNN efficiently. First, many real-world social graphs are of huge size with rich attribute information. For example, ogbn-papers100M [10] has 111 M vertexes, 1.6B edges with 53 GB of vertex feature, while the memory capacity of commercially available GPUs is usually tens of GB, (e.g., 16 GB for NVIDIA P100 GPU). Second, the multi-layer stacking structure like a deep learning network increases the memory footprint of the full graph-based training. To solve the scalability problem, the method of sampling-based training is proposed. In the sampling-based training, subgraphs are extracted by starting from the training vertexes and continuously sampling the neighboring vertices within L-hops. A fixed number of neighbors are selected (sampled) in each layer based on specific sampling strategies such as random sampling, weighted sampling, and random walk. The sampling can reduce both computations and memory requirements in one iteration. Finally, all the training vertices are processed in mini-batches, and the model parameters are updated iteratively until the model converges.

In the sampling-based training, the existing systems such as DGL PYG and PinSage adopt the hybrid CPU+GPU mode. In this mode, the whole training process is divided into three stages:

i) subgraphs sampling,
ii) feature extraction and transmission, and
iii) the actual GNN training. First, the structure and feature data of the graph are stored in the CPU memory.

The CPU is responsible for sampling the graph and generating subgraphs for training. Next, the sampled subgraphs and the collected features are transferred to the GPU, where the GNN training is performed. As the CPU memory capacity is usually much larger than that of GPUs, this mode can support the training of huge graphs in the single- or multi-GPU setting. However, the feature transmission and the CPU-based sampling may become the performance bottleneck due to the low PCIe bandwidth but fast GPU training, which leads to low GPU utilization since the GPU may have to wait for sampled subgraphs.

3 MATERIALS AND METHODS

For a graph represents a vertex (node) in the graph with the feature set denoted by fv, and the edge between two vertices represents the relationship between them. A GNN model learns the high-dimensional feature representation of each node by gathering the information from its neighboring nodes in the previous layer and updating its feature vector following the topology of the deep network (such as Multilayer Perceptrons) iteratively. The core computations in a GNN layer can be divided into two stages: message aggregation and feature transformation. The three optimizations respectively improve the CPU throughput of neighborhood sampling and expansion); reduce slicing overhead (yellow boxes); and enable overlapped GPU transfers and computations (red and blue boxes). With a reasonably high CPU-to-GPU ratio, as is often the case in modern computing clusters, these optimizations almost eliminate GPU idle time, enabling fast training at a speed commensurate with that of the core training operations. Additionally, this work studies inference. Although tradeoffs among accuracy, speed, and memory requirements have been studied extensively for training, they are relatively under-studied for inference. We conduct an empirical analysis that indicates that neighborhood sampling in inference sacrifices prediction accuracy only marginally. This suggests that mini-batch inference with neighborhood sampling is a viable alternative to layer-wise inference with full neighborhoods, yielding accuracy comparable to the latter but with a much lower memory footprint. As an added advantage, model architecture code can be reused between training and inference, simplifying development.

Sampling-Based Training

In the layer-stacked GNN training model, processing every node in a graph is not viable for large graphs due to the large computation and memory footprint. Also, not every node in a graph is labeled for computing the loss and the gradient and updating the model parameters. Inspired by mini-batch-based training in deep learning, sampling-based training is introduced. At each iteration, the nodes in a mini-batch are randomly shuffled and selected as seed nodes in training. Starting from the seed nodes, a fixed number of neighboring nodes are sampled from all neighbors. This process iterates for K times in a K-layer GNN. In each iteration, a subgraph is generated (e.g., bipartite graph blocks in DGL). A subgraph consists of the destination vertices, which are the source vertices from the last iteration, and their neighbors that are sampled (the sampled neighbors become the destination vertices

World Economics & Finance Bulletin (WEFB)
Available Online at: https://www.scholarexpress.net
Vol. 31, February, 2024
ISSN: 2749-3628,

of the subgraph in the next iteration). This way, each vertex has the same number of edges in the subgraphs, which reduces the computation complexity in GNN training and improves the regularity of the message aggregation for the subgraphs. It has been shown that the sampling-based methods can achieve accuracy competitive with the training of the full graph.

In the sampling-based training, the hybrid CPU+GPU computation mode is widely adopted in previous GNN systems. In this mode, an iteration in the GNN training can be further divided into three stages: i) CPU sampling, ii) subgraph and feature transfer; iii) GNN training. The time spent by the CPU in sampling the subgraphs and in transferring the high-dimensional vertex features dominates the entire GNN training process, which forms a severe performance bottleneck. Usually, a GNN uses a shallow network structure with less than four layers. Given the limited bandwidth of PCIe (usually less than 16 GB/s) and the large amount of feature data that needs to be transferred (e.g., 53 GB for the ogbn-papers100 M graph), the huge communication cost can hardly be hidden by GNN computations in such shallow network structures.

To reduce the communication cost, the GPU-based caching technique is proposed in a system called PaGraph. PaGraph caches as many features of the high-degree vertices as possible in the GPU memory. The caching technique has been shown to be effective, especially on large graphs where the node degree follows the power-law distribution.

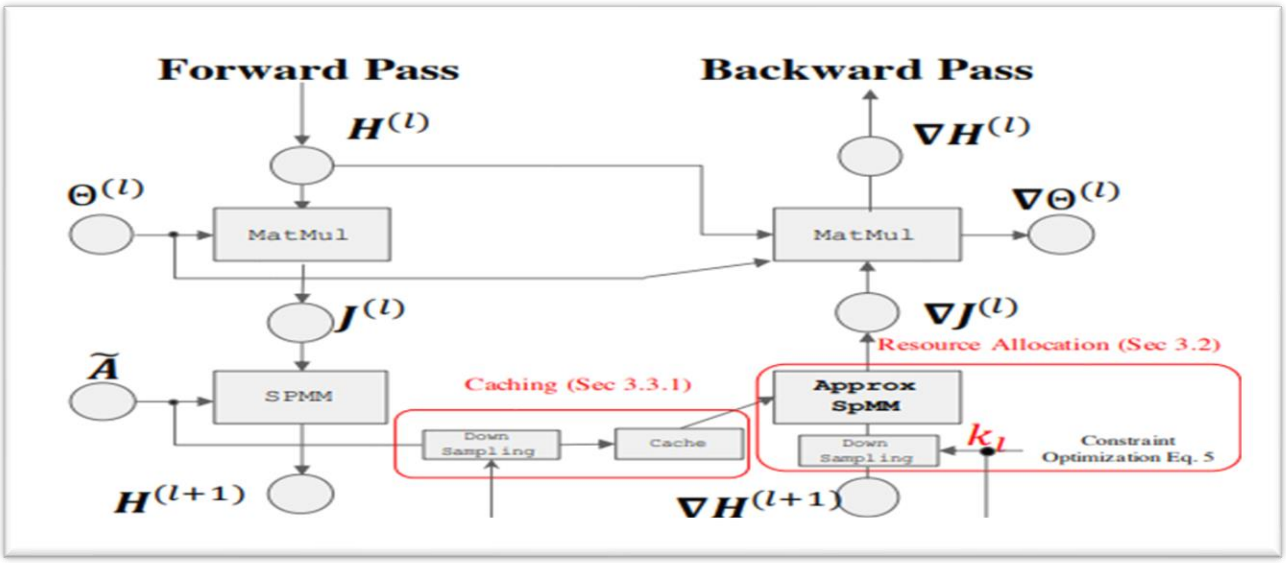

Figure 2: Overview of RSC .

For convenience, ReLU is ignored. RSC only replace the SpMM in the backward pass with its approximated version using top-k sampling.

RESULTS AND DISCUSSION

Graph sampling has to be done during each training iteration. It is thus imperative that graph sampling is done as fast as possible. The typical sampling pipeline as implemented in popular GNN libraries, such as DGL (a popular GNN training library), involves multiple steps that each generate intermediate tensors that have to be written to, and then read from, memory.

World Economics & Finance Bulletin (WEFB)
Available Online at: https://www.scholarexpress.net
Vol. 31, February, 2024
ISSN: 2749-3628,

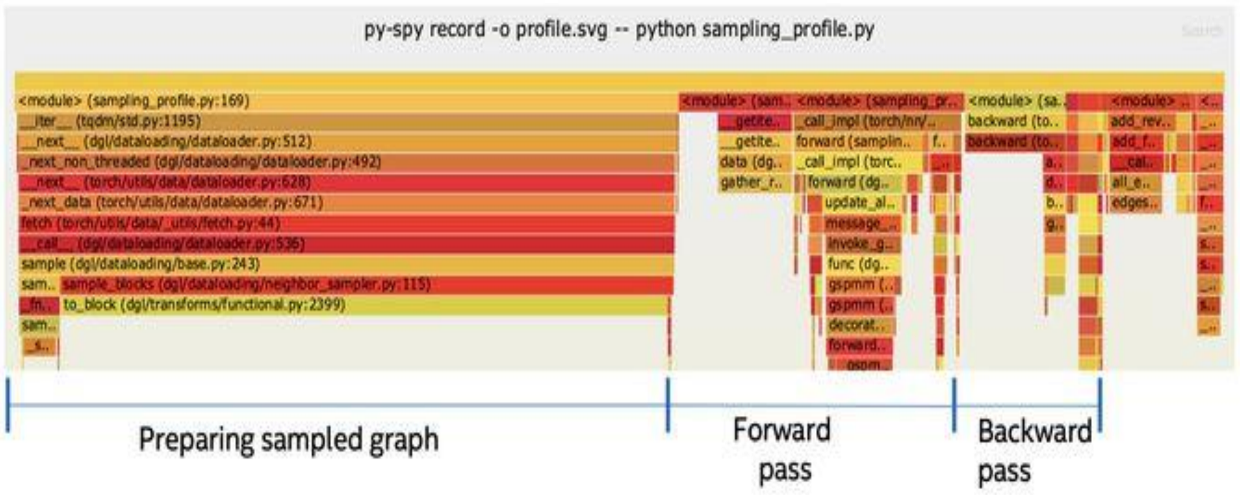

Figure 2. Flame graph showing the fraction of time spent on graph sampling, forward pass, and backward pass. We used DGL and trained on a 2-socket 3rd Generation Intel® Xeon® processor.

On large graphs, GNN training proceeds in the unit of minibatches. Due to edge connections, the graph nodes are not I.I.D distributed, and thus cannot be sampled uniformly at random as minibatch data points. State-of-the-art methods construct minibatches by sampling on each GNN layer (i.e., layer sampling). The vanilla GCN and its successor GraphSAGE sample by tracking down the inter-layer connections. Their approaches preserve the training accuracy of the original model, but the parallel training is not work-efficient due to a phenomenon often referred to as "neighbor explosion".

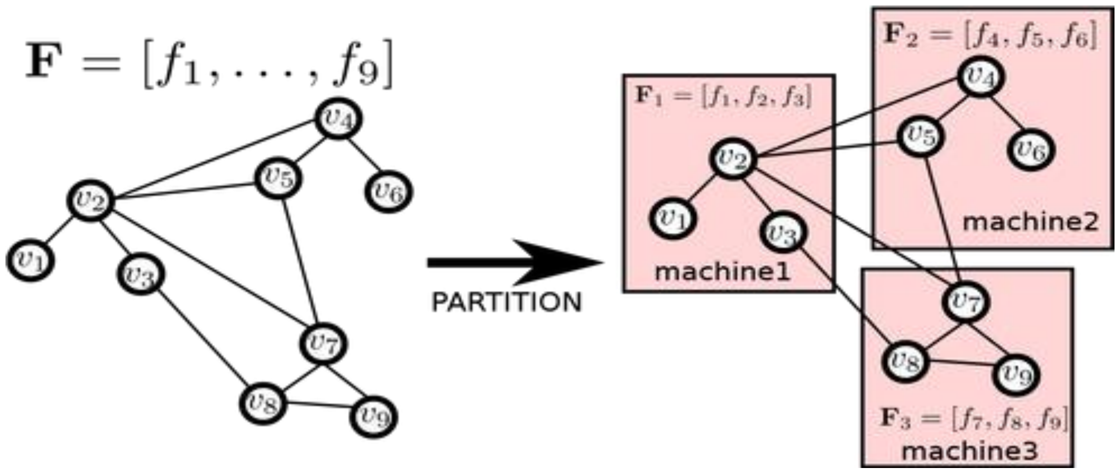
$\mathbf{F} = [f_1, \ldots, f_9]$
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9
PARTITION
$\mathbf{F}_1 = [f_1, f_2, f_3]$
machine1
$\mathbf{F}_2 = [f_4, f_5, f_6]$
machine2
$\mathbf{F}_3 = [f_7, f_8, f_9]$
machine3

Figure 3. Partitioning of a toy graph (with nine nodes) to enable distributed training. Traditional distributed GNN training libraries partition both the graph topology and the node features

Graph embedding is a powerful dimensionality reduction technique to facilitate downstream graph analytics. The embedding process converts graph nodes with unstructured neighbor connections into points in a low-dimensional vector space. Embedding is essential for a wide range of tasks such as content recommendation, traffic forecasting, image recognition and protein function prediction. Among the various embedding techniques, Graph Neural Networks (GNNs) (including Graph Convolutional Network (GCN) and its variants,) have attained much attention. GNNs

World Economics & Finance Bulletin (WEFB)
Available Online at: https://www.scholarexpress.net
Vol. 31, February, 2024
ISSN: 2749-3628,

produce accurate and robust embedding without the need of manual feature selection.

Performance analysis of batch preparation Batch preparation comprises two steps: (a) neighborhood sampling to obtain the mini-batch induced subgraph, and (b) slicing the feature and label tensors to extract the parts that correspond to the sampled subgraph. Both steps are parallelized: sampling uses a PyTorch DataLoader and multiprocessing, and slicing uses multiple OpenMP threads in a single process. The relative performance of sampling and slicing is not easily obtained from per-line measurements, as sampling is performed asynchronously with the main execution thread. As such, we investigate the performance of sampling and slicing using separate targeted benchmarks. Batch preparation time is dominated by the neighborhood sampling time, requiring 7.2 seconds with 20 worker processes. Slicing, by comparison, takes just 1.2 seconds when parallelized with 20 OpenMP threads using PyTorch's parallel slicing code. Even a conservative analysis of the performance breakdown in Tables 1 and 2 implies that neighborhood sampling is a substantial bottleneck in GNNs. For PyG to perform sampling at a pace that can keep a single GPU busy and hide sampling latency on ogbn-products, sampling throughput must be improved by at least 3×. When using multiple GPUs per machine, the required speedup is higher.

CONCLUSION

Optimization in the above two steps can be generalized to support multiple kinds of GNN models and sampling algorithms. We achieve work-efficiency by avoiding "neighbor explosion", as each layer of our minibatched GNN contains the same number of neurons corresponding to the subgraph nodes. Finally, we achieve learning accuracy since our sampled subgraphs preserve connectivity characteristics of the original training graph. The main contributions of this paper are:

•We propose a parallel GNN training algorithm based on graph sampling:

–Accuracy is achieved since the sampler returns small, representative subgraphs of the original graph.

–Efficiency is optimized since we always build complete GNNs on the minibatch subgraphs to avoid "neighbor explosion" in deeper layers.

–Scalability is achieved with respect to number of processing cores, graph size and GNN depth by parallelizing various key steps.

–We propose a novel data structure that supports fast, incremental and parallel updates to a probability distribution. Our parallel sampler based on this data structure theoretically and empirically achieves near-linear scalability with respect to number of processing units.

–We parallelize all the key operations to scale the overall minibatch training to a large number of processing cores. Specifically, for subgraph feature propagation, we perform intelligent partitioning along the feature dimension to achieve close-to-optimal DRAM and cache performance.

We propose a runtime scheduling algorithm for training:

–By rearranging the order of various operations, we significantly reduce the training time under a wide range of model configurations.

–By partition scheduling and node clipping of subgraphs, we improve the feature propagation performance by better cacheline alignment.

We show that our parallelization and scheduling techniques are applicable to a number of GNN architectures (including graph convolution and graph attention) and graph sampling algorithms (including random edge sampling and variants of random walk sampling). We perform thorough evaluation on a 40-core Xeon server. Compared with serial implementation, we achieve 15× overall training time speedup. Compared with state-of-the-art minibatch methods, our training achieves up to 7.8× speedup without accuracy loss.

REFERENCES

1. 2019. Kgat: Knowledge graph attention network for recommendation. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 950--958.
2. Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In Proceedings of the 42nd Annual International Symposium on Computer Architecture. 105--117.
3. Mohammed Alandoli, Mohammed Shehab, Mahmoud Al-Ayyoub, Yaser Jararweh, and Mohammad Al-Smadi. 2016. Using GPUs to speed-up FCM-based community detection in Social Networks. In 2016 7th International Conference on Computer Science and Information Technology (CSIT). 1--6.
4. Bahar Asgari, Ramyad Hadidi, Jiashen Cao, Da Eun Shim, Sung Kyu Lim, and Hyesoon Kim. 2021. FAFNIR: Accelerating Sparse Gathering by Using Efficient Near-Memory Intelligent Reduction. In IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021, Seoul, South Korea, February 27 - March 3, 2021. IEEE, 908--920.

World Economics & Finance Bulletin (WEFB)
Available Online at: https://www.scholarexpress.net
Vol. 31, February, 2024
ISSN: 2749-3628,

5. Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. 2016. Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems. In 2016 49th annual IEEE/ACM international symposium on Microarchitecture (MICRO). IEEE, 1--13.
6. Aleksandar Bojchevski and Stephan Günnemann. 2017. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. arXiv preprint arXiv:1707.03815 (2017).
7. Zhenkun Cai, Xiao Yan, Yidi Wu, Kaihao Ma, James Cheng, and Fan Yu. 2021. DGCL: an efficient communication library for distributed GNN training. In EuroSys '21: Sixteenth European Conference on Computer Systems, Online Event, United Kingdom, April 26--28, 2021, Antonio Barbalace, Pramod Bhatotia, Lorenzo Alvisi, and Cristian Cadar (Eds.). ACM, 130--144.
8. Jiaxian Chen, Guanquan Lin, Jiexin Chen, and Yi Wang. 2021. Towards efficient allocation of graph convolutional networks on hybrid computation-in-memory architecture. Science China Information Sciences 64, 6 (2021), 1--14.
9. Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. arXiv preprint arXiv:1801.10247 (2018).
10. Lawson, C. L., Hanson, R. J., Kincaid, D., , and Krogh, F. T. Basic linear algebra subprograms for FORTRAN usage. ACM Trans. Math. Soft., 5:308–323, 1979.
11. Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. In ICLR, 2016.
12. Li, Y., Yu, R., Shahabi, C., and Liu, Y. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In ICLR, 2018.