Betty: Enabling Large-Scale GNN Training with Batch-Level Graph Partitioning and Tiered Memory

Shuangyan Yang
University of California, Merced
syang127@ucmerced.edu

Minjia Zhang
Microsoft Research
minjiaz@microsoft.com

Wenqian Dong
University of California, Merced and
Florida International University

Dong Li
University of California, Merced
dli35@ucmerced.edu

1 Introduction

Graph neural network (GNN) has emerged as an effective paradigm to learn rich relation and interaction information in irregular graph-based structures. Recent efforts show that GNN training efficiency or accuracy can be improved by using larger batch sizes [4, 5], training with more sophisticated aggregators [9, 11], increasing aggregation depth [7], using a larger sampling rate [15], or using deeper and wider neural encoders [6]. However, despite leading to promising results, the improvements often come at a cost of significantly increased memory consumption. To work around the memory capacity bottleneck, prior work explored both algorithmic (sampling [8, 13, 16]) and system optimizations (DGL [12], PyTorch Geometric [1], and NeuGraph [10]). However, algorithmic method requires careful consideration of the sampling strategy and may cause loss of important neighbor information that hurts the final model accuracy. For system methods, as the batch size or aggregation depth increases, especially when using more memory-intensive aggregators, GNN training can still run out of memory. Frameworks such as DGL also support distributed training for GNN (DistDGL [14]), where they partition the graph across multiple GPUs and/or multiple nodes, and leverage the aggregated memory from multiple GPUs to scale out the GNN training. While being an effective approach, these methods often increase the hardware cost of training GNN models significantly.

To analyze the scalability bottleneck, we analyze the memory usage of a popular GNN network GraphSAGE [2] with a large dataset obgn-products [3] on an NVIDIA RTX6000 GPU with 24GB memory. Figure 1 shows that GNN scalability has been severely limited by the GPU memory capacity. While simple aggregators such as Mean and Pool incur <10GB memory consumption, more advanced aggregators such as LSTM are much more memory hungry and easily lead to over 24GB of memory consumption and OOM error, making training with these more advanced aggregators infeasible on larger datasets. Similarly, as we increase the aggregation depth, the memory consumption increases almost exponentially and runs into OOM with deeper GNNs (e.g., running OOM at the 4th-layer as in Figure 1.b). Furthermore, the limited memory also prevents GNN from using wider hidden

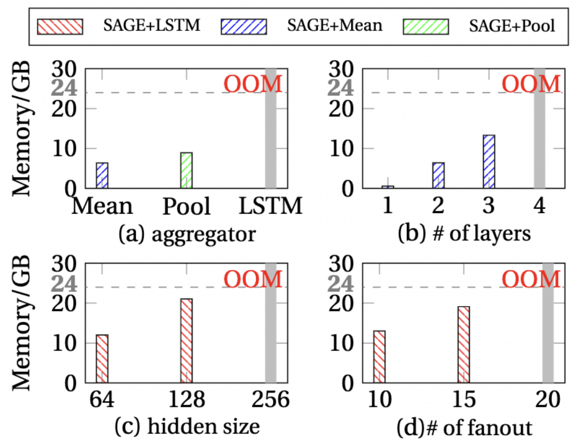

Figure 1. The memory consumption of running GraphSAGE on ogbn-products. (a) Comparison results between neighbor aggregators. The number of SAGE layers is 2, the hidden size is 256, and the fanout degree for the two layers is 10 and 25 respectively. (b) Comparison results varying the number of SAGE layers. The aggregator is Mean and the hidden size 256. For the four layers of SAGE, the fanout degree is 10, 25, 30 and 40 respectively. (c) Comparison varying the hidden size. Similar to the configuration in (b), but varying the hidden dimension sizes from 64 to 256. (d) Comparison results varying fanout degree. The SAGE layer is 1, the hidden size is 256 and the aggregator is LSTM.

size (Figure 1.c) and larger fanout degree for aggregation (Figure 1.d). As such, the memory capacity has become a severe bottleneck for data scientists and practitioners to use more advanced GNN training methods. As we show in Section 3, Betty[1] avoids OOM and enables those memory-consuming cases.

Based on the analysis of GNN scalability bottlenecks, we identify feature vectors and their corresponding hidden maps involved in aggregation as a major source of memory consumption for GNN training. To reduce the memory usage, one effective method is via batch-level partitioning, where

[1]The full paper appears in ASPLOS'23 and can be found in https://doi.org/10.1145/3575693.3575725

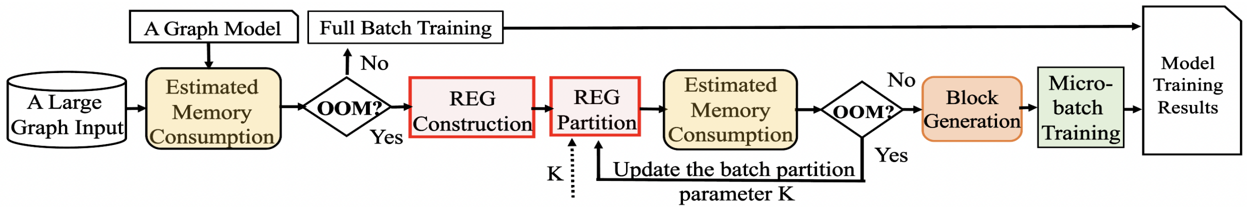

Figure 2. Workflow of Betty.

a mini-batch is partitioned into K micro-batches. In micro-batches training, the loss and gradients are calculated after each micro-batch. Instead of updating the model parameters after each backpropagation, the training process waits and accumulates the gradients over consecutive micro-batches, and ultimately updates the parameters based on the accumulated gradients from the entire full batch. With this simple optimization, the GNN can be trained with a much lower memory consumption while achieving same convergence without any changes to hyper-parameters or optimizer.

Batch-level partitioning in GNN is complex due to intricate dependencies. Unlike traditional deep learning with a 1:1 mapping between inputs and labels, GNNs have more complex relationships (N:M). This complexity results in two challenges: (1) *High redundancy* occurs as shared neighbors are duplicated across micro-batches, creating inefficiency. (2) *Load imbalance* arises when a few micro-batches, due to partitioning strategies we selected, consume significantly more memory, increasing the device's maximum memory footprint.

To address these issues, we propose a solution for GNN batch-level partitioning. We use a multi-level bipartite graph partitioning approach and introduce two key techniques. (1) Betty creates a redundancy-embedded graph and simplify the redundancy reduction problem. (2) Betty implements a memory-aware partitioning method to reduce memory usage effectively.

2 Overview

Betty reduces the memory consumption of GNN training via the batch-level partitioning and using both CPU and GPU memory to train advanced GNNs on single GPU. We formulate the batch-level partitioning in GNN as a multi-level bipartite graph partitioning problem and study how such partitioning can be done while allowing the GNN training to leverage accumulative gradients of the partitioned micro-batches to achieve the same training results as the original batch without the need of adjusting any hyperparameters from model scientists.

While the batch-level partitioning reduces the memory footprint, it introduces challenges of high redundancy across partitioned micro-batches and load imbalance. We introduce two novel techniques to mitigate the redundancy. First, construct a redundancy-embedded graph (REG) and transform the redundancy reduction problem to an equivalent min-flow cost cut problem, then solve it via a high performance min-flow cut algorithm; Second, we use memory-aware graph partitioning via accurate estimation of the memory usage of GNN batches. This method allows Betty to quickly figure out how many partitions a batch should be split to meet a memory capacity constraint. Figure 2 overviews the workflow of using Betty.

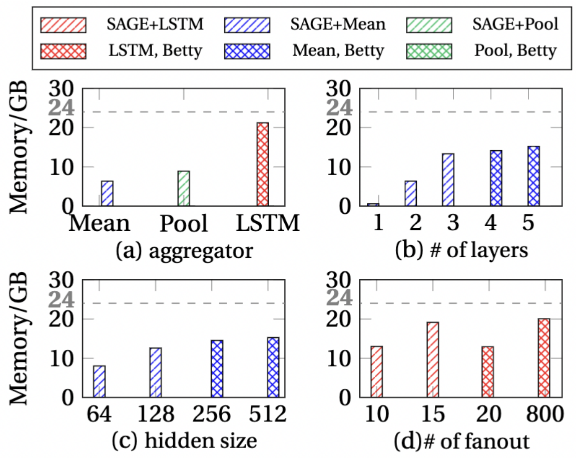

Figure 3. Betty addresses the memory capacity problem presented in Figure 1.

3 Evaluation

We conduct extensive experiments on different sizes of graphs, including a billion-scale graph to demonstrate how Betty enables large-scale GNN training on single GPU without suffering from out of memory (OOM) and losing accuracy. Evaluating with GraphSAGE on the dataset ogbn-products, Betty is able to run a sophisticated aggregator LSTM (Figure 1.a $\longrightarrow$ Figure 3.a) using nine micro-batches; (Figure 1.b $\longrightarrow$ Figure 3.b) runs the GNN model with more layers (4 and 5 layers) using 3 and 60 micro-batches respectively; (Figure 1.c $\longrightarrow$ Figure 3.c) runs the GNN model with larger hidden sizes (256 and 512) using 3 and 32 micro-batches respectively; (Figure 1.d $\longrightarrow$ Figure 3.d) increases fanout to 20 and 800 using 2 and 13 micro-batches respectively. Compared with a set of other graph partition algorithms (Metis, range, and random), Betty improves computation efficiency by 20.6%, 21.1%, and 22.9% respectively.

References

[1] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

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

[3] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[4] Yaochen Hu, Amit Levi, Ishaan Kumar, Yingxue Zhang, and Mark Coates. On batch-size selection for stochastic training for graph neural networks. 2020.

[5] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. Improving the accuracy, scalability, and performance of graph neural networks with roc. *Proceedings of Machine Learning and Systems*, 2:187–198, 2020.

[6] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

[7] Guohao Li, Chenxin Xiong, Ali K. Thabet, and Bernard Ghanem. DeeperGCN: All You Need to Train Deeper GCNs. *CoRR*, abs/2006.07739, 2020.

[8] Zhiqi Lin, Cheng Li, Youshan Miao, Yunxin Liu, and Yinlong Xu. PaGraph: Scaling GNN Training on Large Graphs via Computation-Aware Caching. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020.

[9] Zhilong Lu, Weifeng Lv, Zhipu Xie, Bowen Du, and Runhe Huang. Leveraging graph neural network with lstm for traffic speed prediction. In *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 74–81. IEEE, 2019.

[10] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. {NeuGraph}: Parallel deep neural network computation on large graphs. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 443–458, 2019.

[11] Chenyang Si, Wentao Chen, Wei Wang, Liang Wang, and Tieniu Tan. An attention enhanced graph convolutional lstm network for skeleton-based action recognition. In *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1227–1236, 2019.

[12] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep Graph Library: A Graph-centric, Highly-performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315*, 2019.

[13] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs. *CoRR*, abs/2010.05337, 2020.

[14] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. Distdgl: distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 36–44. IEEE, 2020.

[15] Da Zheng, Xiang Song, Chengru Yang, Dominique LaSalle, Qidong Su, Minjie Wang, Chao Ma, and George Karypis. Distributed hybrid cpu and gpu training for graph neural networks on billion-scale graphs. *arXiv preprint arXiv:2112.15345*, 2021.

[16] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. AliGraph: A Comprehensive Graph Neural Network Platform. *Proceedings of the VLDB Endowment*, 12(12):2094–2105, 2019.