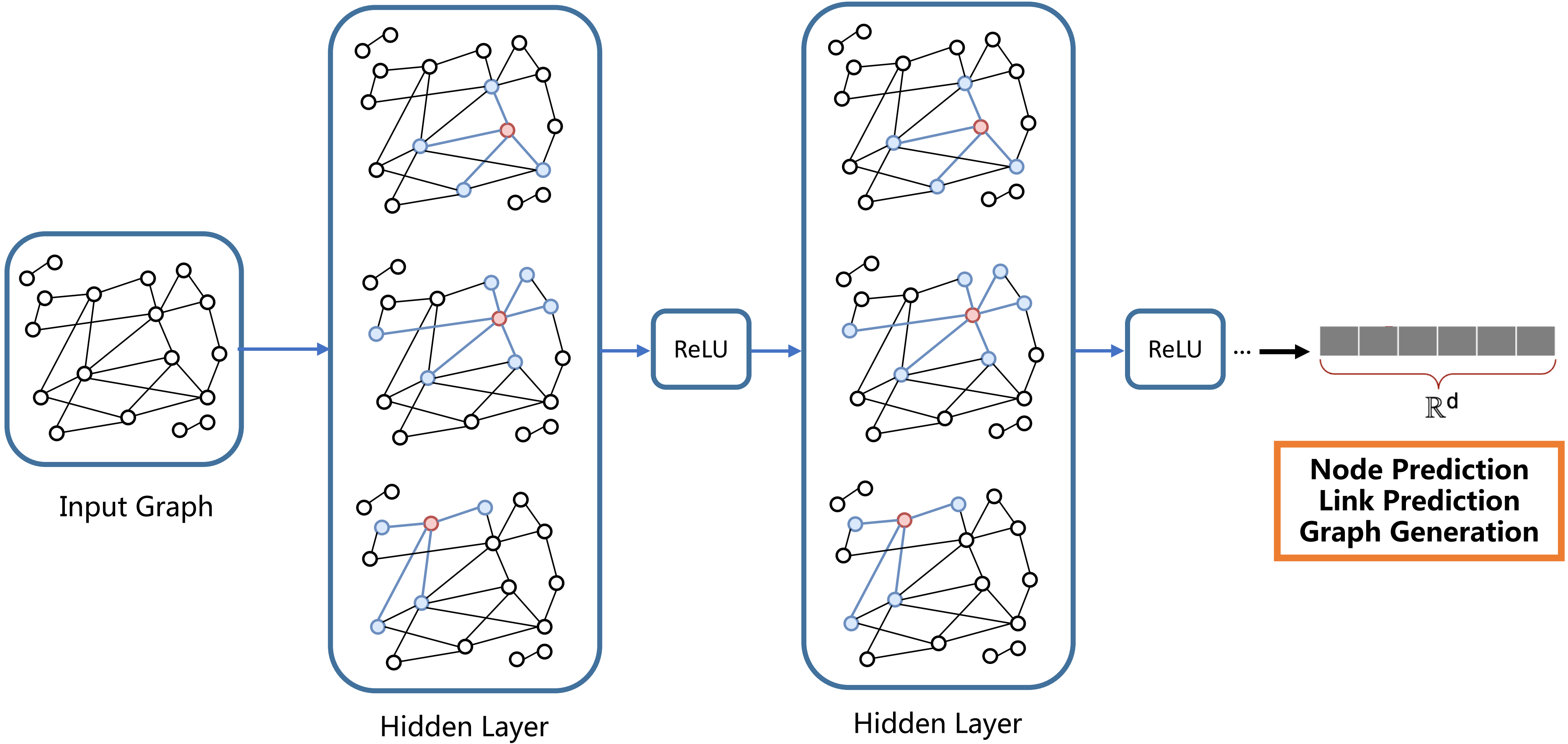
BGL: GPU-Efficient GNN Training by Optimizing Graph Data I/O and Preprocessing

Tianfeng Liu*, Yangrui Chen*, Dan Li, Chuan Wu, Yibo Zhu, Jun He
Yanghua Peng, Hongzheng Chen, Hongzhi Chen, Chuanxiong Guo

tianfeng.leo@gmail.com

April 17, 2023

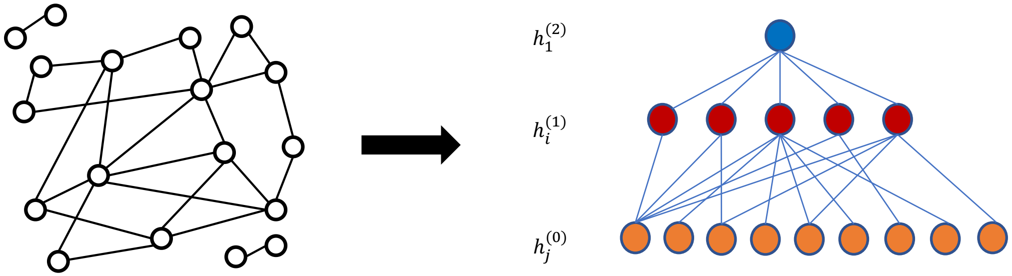
GNN: Deep Learning on Graphs



GNN Training on Large-scale Graphs

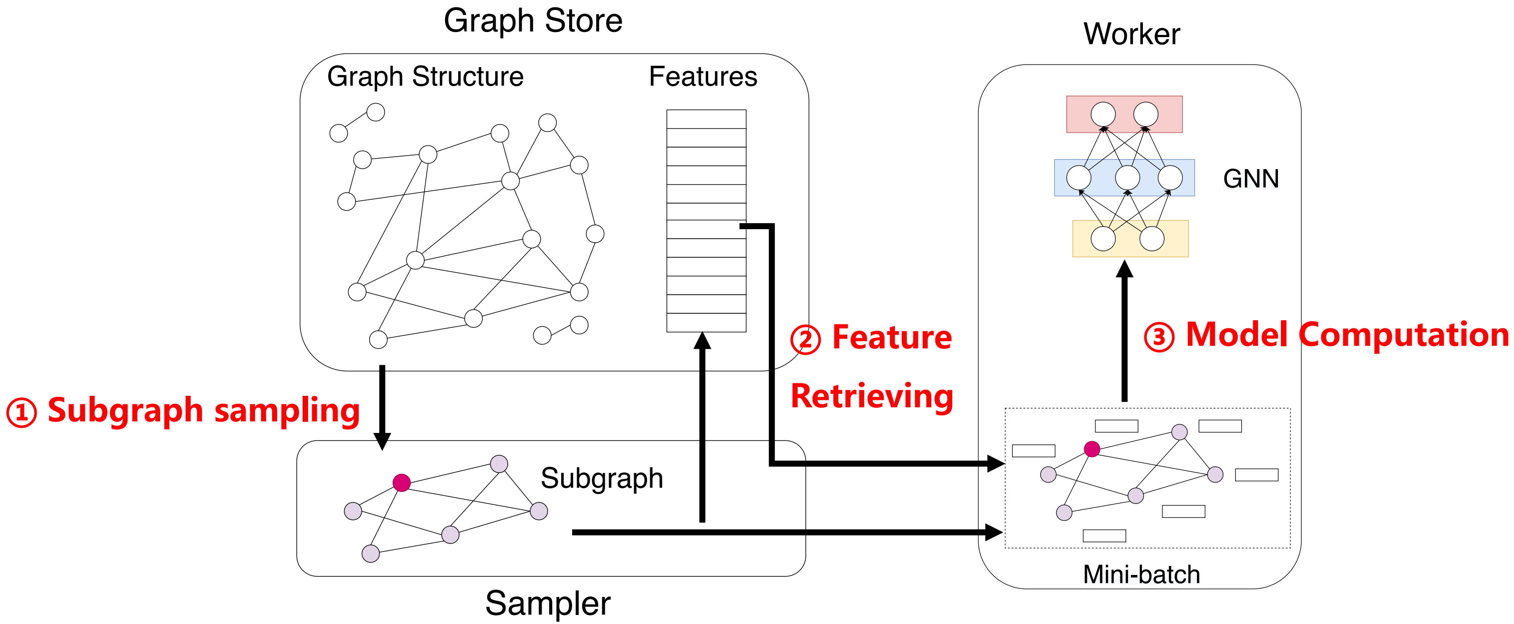
Sampling-based GNN Training

- Full-batch training needs large memory to load the entire graphs, which cannot scale to very large graphs, such as billion-node graphs
- Existing training systems adopt the sampling-based training method, which samples a subgraph from original graphs and constructs a mini-batch as the input of GNN model



Architecture of Sampling-based Training

Components and stages of sampling-based training

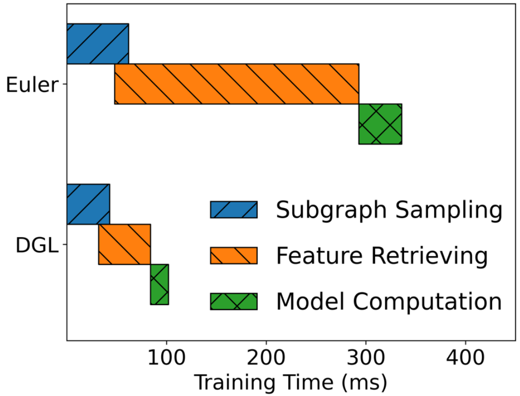


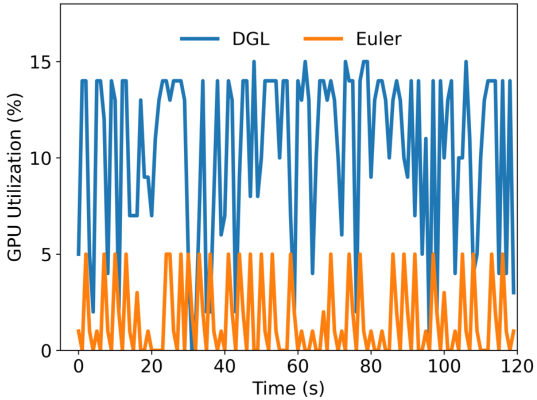
We refer to the first two stages as **Data I/O and Preprocessing**

Data I/O and Preprocessing Bottleneck

Existing systems suffer from preprocessing bottleneck

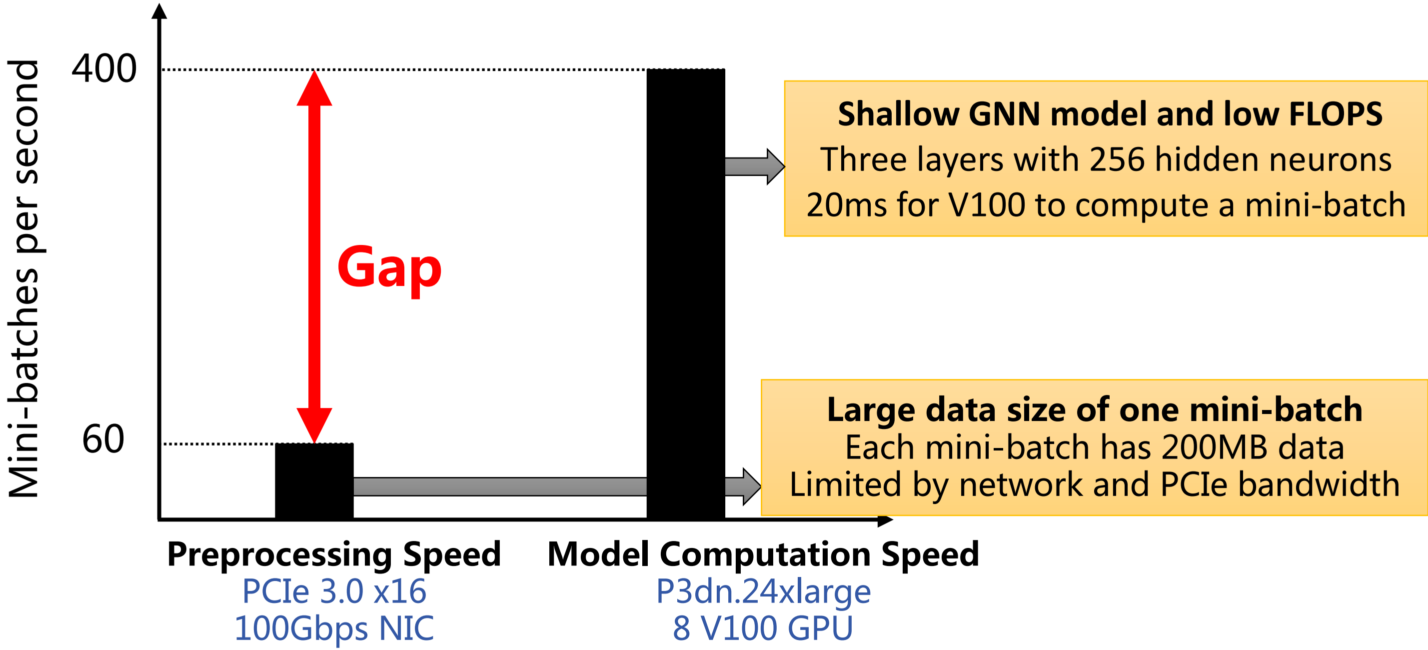
- 87% and 82% of the training time were spent in data I/O and preprocessing by Euler and DGL, respectively
- The maximum GPU utilization of DGL and Euler is 15% and 5%, respectively



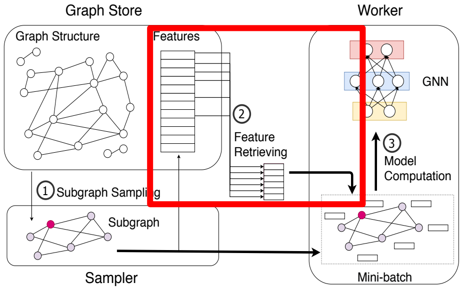


Data I/O and Preprocessing Bottleneck

A huge gap between preprocessing and model computation



Challenge #1 in Removing Bottleneck



Ineffective caching for node feature retrieving

- Node feature retrieving renders the biggest bottleneck
  - 97% of data in mini-batches are node features
- PaGraph[SoCC 20] adopts a static cache policy to reduce the traffic volume
  - Cache node features of high degree nodes

Tradeoff between static cache policy and dynamic cache policy

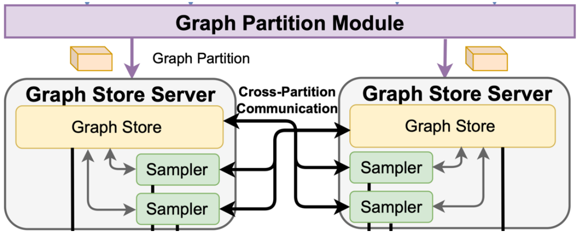
- Static cache policy has small cache overheads, but low cache hit ratios
- Dynamic cache policy has high hit ratios, but large cache overheads

Can we achieve a good trade-off between hit ratios and overheads?

Challenge #2 in Removing Bottleneck

Existing partition algorithm is not scalable and friendly for GNN

- Subgraph sampling renders another major bottleneck



Goal of ideal graph partition algorithm

- Preserve multi-hop connectivity
- Balance training nodes
- Scale to billion-node graphs

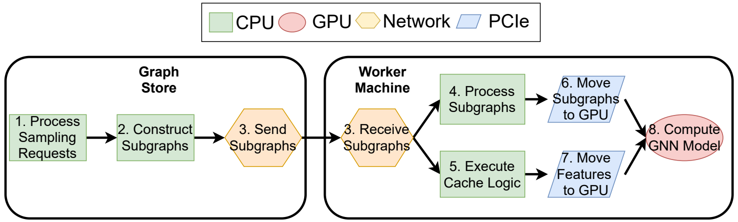
Partition Algorithms	Scalability to Giant Graphs	Balanced Training Nodes	Multi-hop Connectivity
Random [2, 30]	✓	✓	✗
METIS [32] & ParMETIS [33]	✗	✓	✓
GMiner [10]	✓	✗	✗
PaGraph [38]	✗	✓	✓

We need an algorithm which is scalable and friendly to subgraph sampling

Challenge #3 in Removing Bottleneck

Training pipeline of GNN is much more complex than DNN

- Different stages consume different CPU/PCIe/Network resources



Different data preprocessing stages contend for resources

- If all stages freely compete for resources, contention leads to poor performance
- Existing training systems largely ignore this problem

We need to alleviate resource contention and balance time of stages

Overview of BGL

Feature cache engine with algorithm-system co-design for Challenge #1

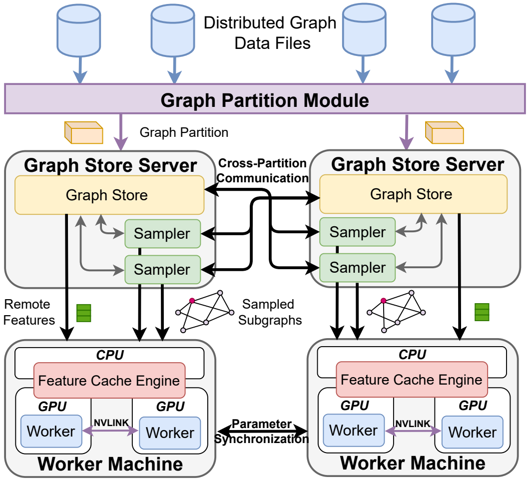
- Proximity-aware ordering to improve temporal locality
- Multi-GPU cache supporting dynamic cache

New graph partition algorithm for Challenges #2

- Multi-level coarsening to reduce the size of graph
- New partitioning heuristic considering both multi-hop connectivity and training workload balancing

Resource isolation for Challenges #3

- Formulate as an optimization problem
- Assign isolated resources to minimize the maximal time of each stages

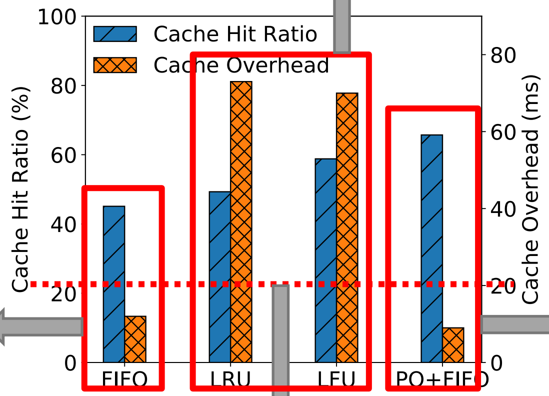


Feature Cache Engine

Which dynamic cache policy should use?

- We implement three popular polices, FIFO, LRU, LFU, whose operations are $O(1)$

LRU and LFU have intolerable cache overhead, much higher than computation time



FIFO meets the throughput requirement, but cache hit ratio is low

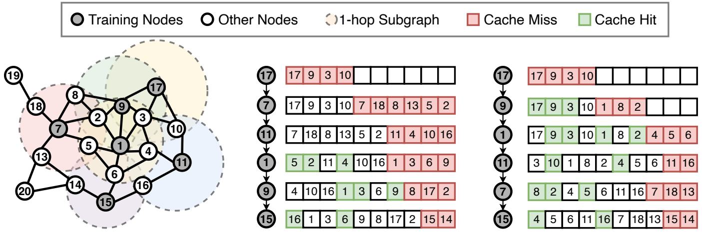
We propose **proximity-aware ordering (PO)** to improve FIFO hit ratios

GPU computation time per batch

Feature Cache Engine

Proximity-aware ordering

- Change the order of selecting training nodes
  - Select training nodes in traversal-based ordering, such as BFS order
- Insight
  - Each node appears more than once among different mini-batches
  - Reuse data by caching features in nearby batches (a.k.a., **temporal locality**)
  - BFS improves the chance of appearance of the same nodes in nearby batches



(a) A Sample Graph with 6 Training Nodes

(b) Cache using Random Ordering {17 7 11 1 9 15}

(c) Cache using Proximity-aware Ordering {17 9 1 11 7 15}

PO improves FIFO cache hits from 8 to 14

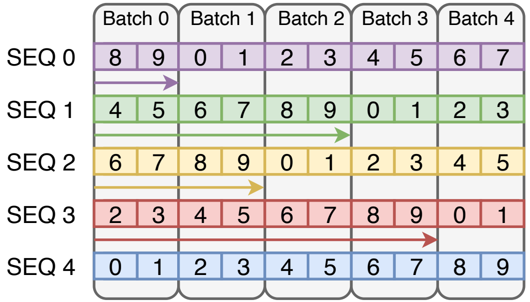
Feature Cache Engine

Trade-off between temporal locality and model convergence

- Traversal-based ordering improves temporal locality but harms convergence
- Random ordering guarantees convergence but has poor temporal locality

PO balances the above trade-off based on SGD property

- Insight : SGD is robust enough, hence, slightly relaxing IID assumption does not influence convergence rate
- Introduce two types of randomness
  - Multiple sequences with random BFS roots
  - Circularly shifting each BFS sequences



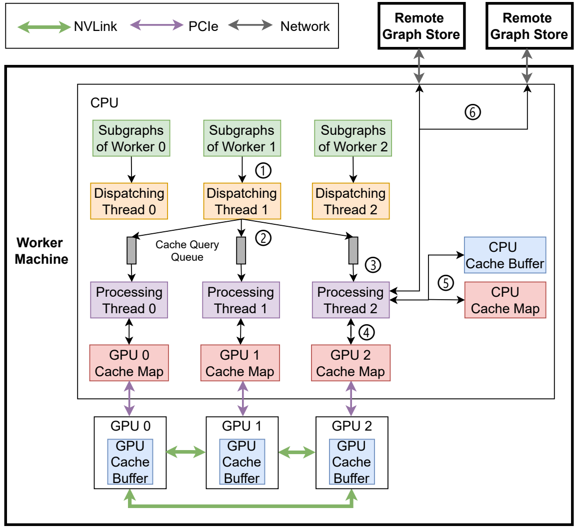
Feature Cache Engine

Maximizing cache size to increase cache hit ratios

- Insight: GNN model is small and shallow, hence, large memory is unused
- Two-level cache jointly using large and free CPU and multiple GPU memory

Multi-GPU Cache

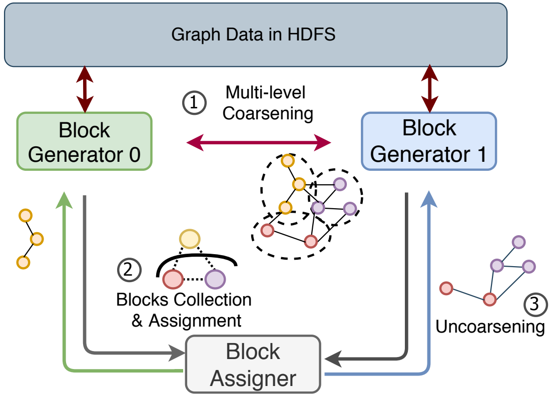
- Use NVLink for high-bandwidth and low latency inter-GPU communication and alleviate traffic in PCIe links
- Cache workflow which guarantees consistency of mutable cache buffers on dynamic cache policy



Graph Partition Module

Partition Workflow

- Multi-level Coarsening
  - Use multi-source BFS to preserve connectivity
  - Merge small blocks to reduce block numbers
- Block Collection and Assignment
  - Apply a greedy assignment heuristics to each block
- Uncoarsening
  - Map blocks to nodes of original graphs



This algorithm has low time complexity and is friendly to billion-node graphs

Graph Partition Module

Assignment Heuristic

- We propose a new heuristic for assigning blocks by considering GNN requirements

$$\max_{i\in[k]} \left\{ \boxed{\left(\sum_j \left|P(i)\cap\Gamma^j(B)\right|\right)} \cdot \boxed{\left(1-\frac{|T(i)|}{C_T}\right)} \cdot \boxed{\left(1-\frac{|P(i)|}{C}\right)} \right\}$$

Multi-hop Block Neighbor
Assign the current block to a partition with the maximum number of neighbors

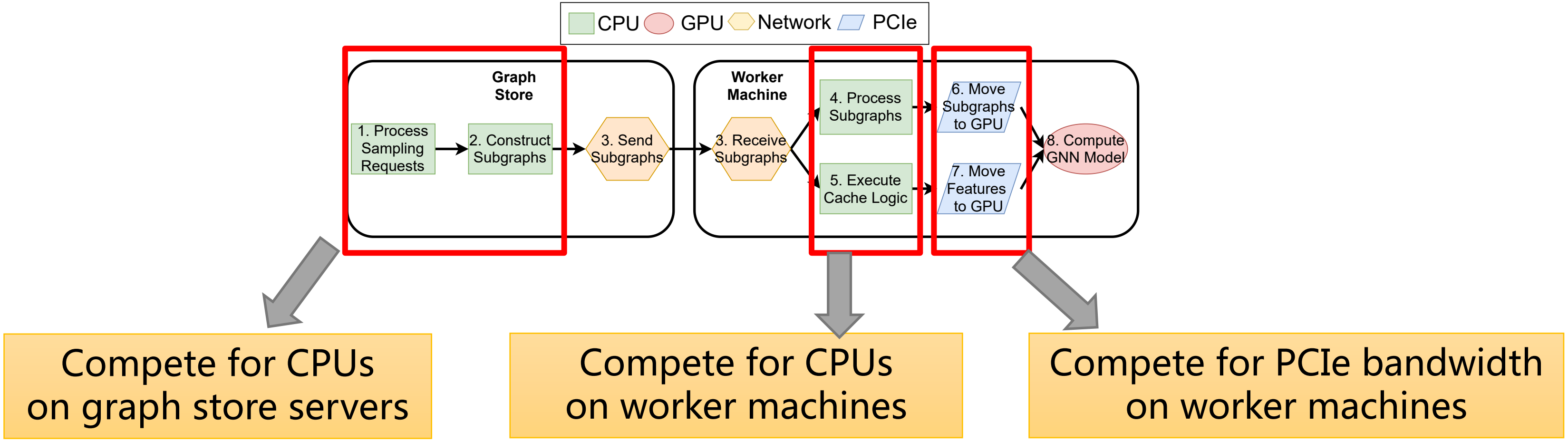
Training Node Penalty
Enforce each partition has the same number of training nodes.

Node Penalty
Balance the number of nodes among different partitions

GNN Training Pipeline

Asynchronous Pipeline Stages

- We divide GNN training into 8 asynchronous pipeline stages



If all processes freely compete for resources, resource contention leads to poor performance!

GNN Training Pipeline

Profiling-based Resource Allocation

- Profile the execution time of each stage and assign isolated resources to them

Optimization Goal: minimize the maximal completion time of all stages

Assumption: linear acceleration for all stages except caching.
For caching stage, use a fitting function $f(c_4) = a/c_4 + d$

$$\boxed{\min\ \max}\left\{\frac{T_1}{c_1}, \frac{T_2}{c_2}, T_{net}, \frac{T_3}{c_3}, \frac{D_I}{b_I}, \boxed{f(c_4)}, \frac{D_{II}}{b_{II}}, T_{gpu}\right\}$$

$$\boxed{\text{s.t.} \quad c_1 + c_2 \leqslant C_{gs}, \quad c_3 + c_4 \leqslant C_{wm}, \quad b_I + b_{II} \leqslant B_{pcie}}$$

Constraints: the resource capacity of CPU cores and PCIe bandwidth, C_{gs}, C_{wm}, B_{pcie}

Evaluation of BGL

Experimental Environment

- 4 GPU severs: 8 V100 GPU (with NVLink v2), 96 CPU cores, 356GB DRAM
- 32 CPU servers: 96 CPU cores, 480GB DRAM, connected with 100Gbps NIC

Systems

- Compared BGL against Euler, DGL, PyG, PaGraph

Graphs

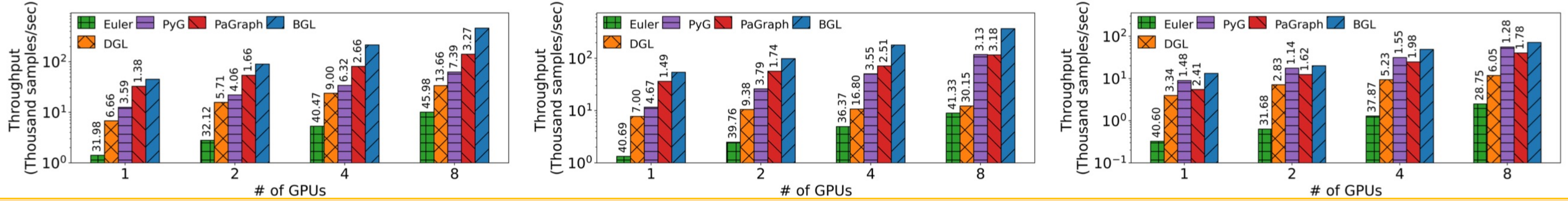
- Three graphs from million to billion nodes

	Ogbn-products	Ogbn-papers	User-Item
Nodes	2.44M	111M	1.2B
Edges	123M	1.61B	13.7B
Feature Dimension	100	128	96
Classes	47	172	2
Training Set	196K	1.20M	200M
Validation Set	393K	125K	10M
Test Set	2.21M	214K	10M

GNN Model

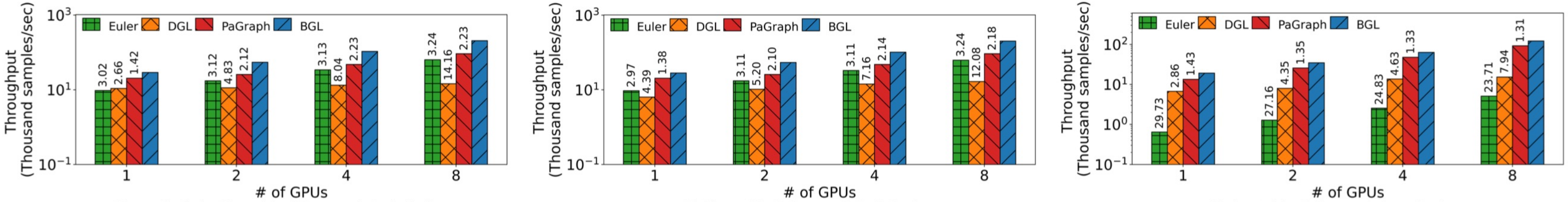
- GCN, GraphSAGE, GAT, three layer (128 hidden)
- Batch size 1000, fanout {5,10,15}

Overall Performance



BGL outperforms all other systems, and the geometric mean of speedups over PaGraph, PyG, DGL and Euler is 1.91x, 3.02x, 7.04x and 20.68x, respectively.

Figure 11: Training throughput of 3 GNN models on Ogbn-papers in *log* scale. Numbers above bars are speedups of BGL over other systems.



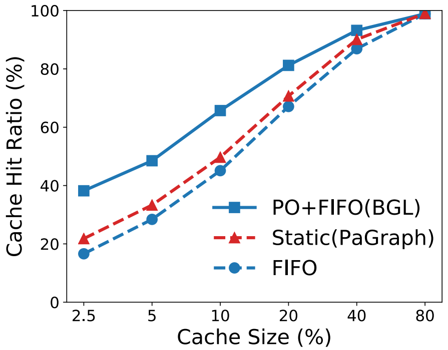
(a) GraphSAGE (1.42x - 14.16x) (b) GCN (1.38x - 12.08x) (c) GAT (1.31x - 29.73x)

Figure 12: Training throughput of 3 GNN models on User-Item in *log* scale. Numbers above bars are speedups of BGL over other systems.

Improvements of Feature Cache Engine

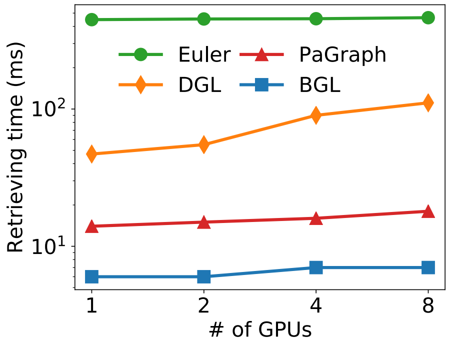
BGL achieves highest cache hit ratios

- PO+FIFO improves 20% cache hit ratios on Ogbn-papers compared with PaGraph static cache policy



BGL reduces the feature retrieving time

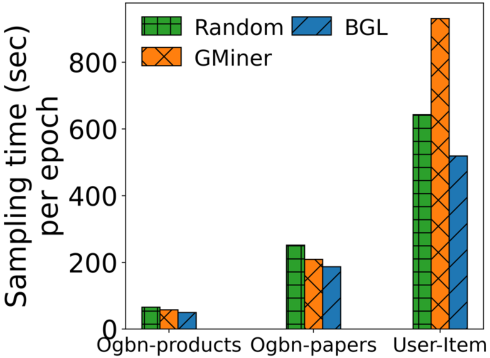
- The reduction is 98%, 88% and 57% for Euler, DGL and PaGraph respectively

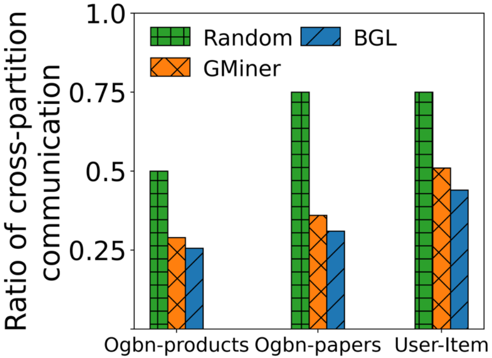


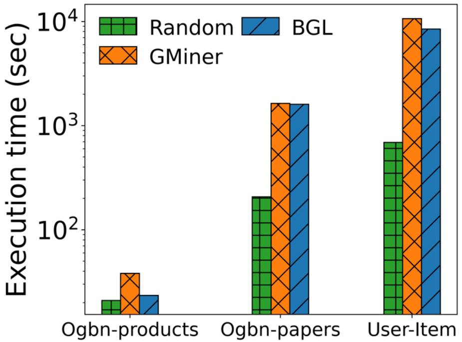
Improvements of Graph Partition Algorithm

BGL reduces sampling time and partitioning time

- BGL reduces 10%-20% sampling time during GNN training
- BGL reduces the cross-partition communication of sampling from 25% to 44%
- The execution time of BGL is faster than well-optimized GMiner, with 20% reduction



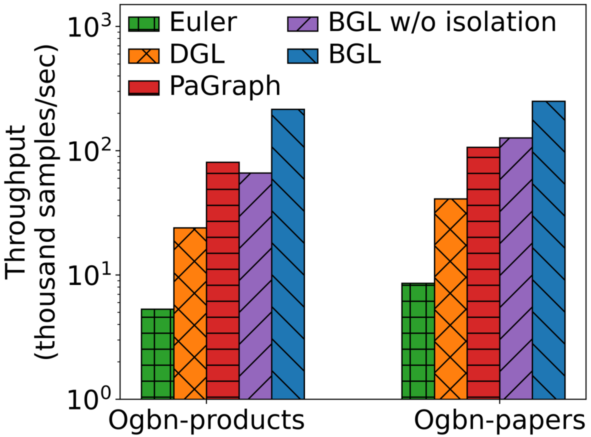




Improvements of Resource Isolation

BGL achieves best performance after resource isolation

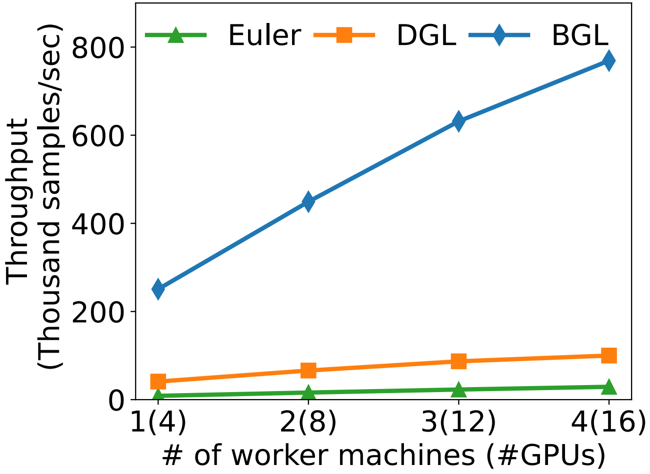
- The speedup is 2.7x, compared to the naïve resource allocation strategy
- BGL without resource isolation is even worse than PaGraph in Ogbn-products



Scalability to Multiple Worker Machines

BGL has good scalability when scaling to multiple machines

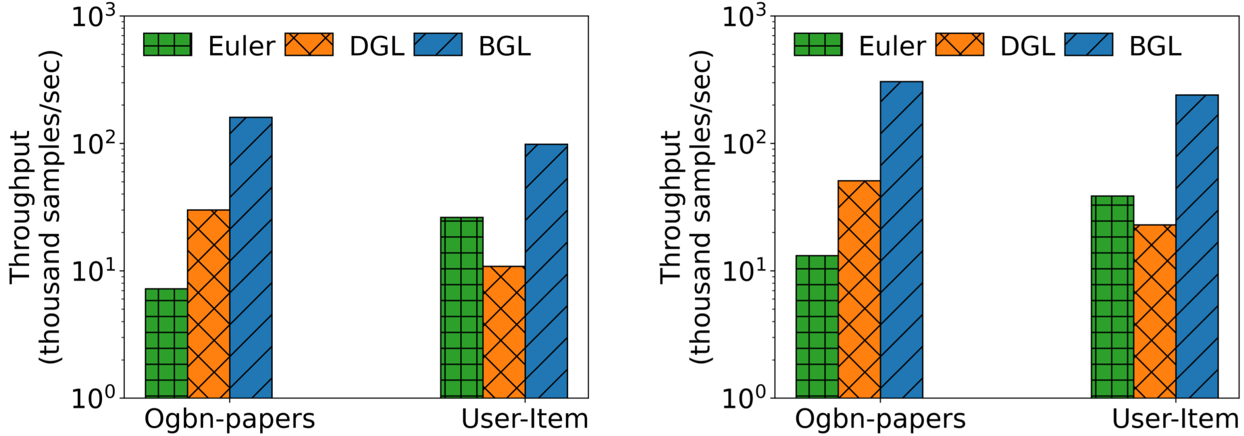
- BGL achieves 76% of linear scalability
- Feature cache engine cannot share GPU memory across machines due to NVLink v2. This fact limits the BGL scalability



Impact of Hyper Parameters

BGL is robust to different hyper parameters

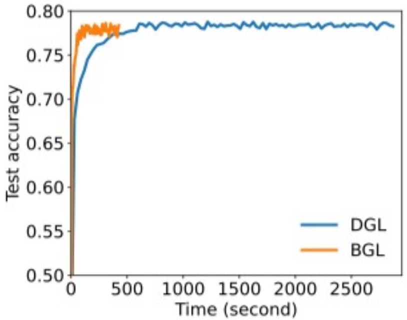
- The speedup is 10.44x and 7.50x for Euler and DGL, respectively



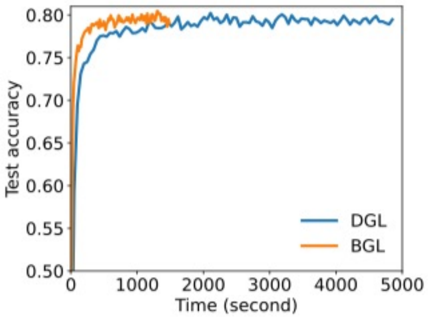
(a) BS 1000, 3 hops, FO {10,10,10}. (b) BS 500, 2 hops, FO {10,25}.

Model Accuracy

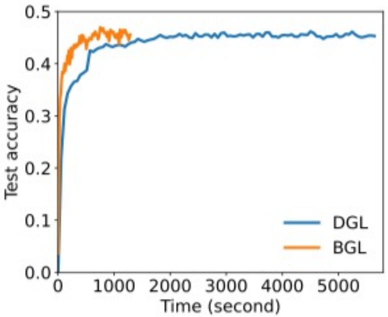
BGL achieves the same accuracy as the original DGL but faster



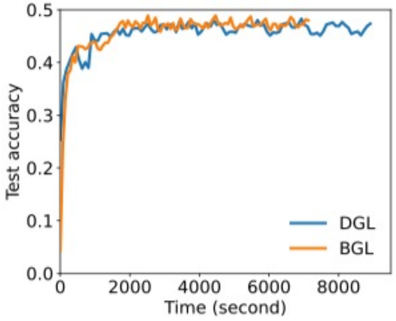
(a) GraphSAGE on Ogbn-products



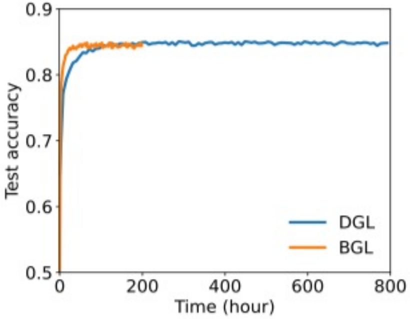
(b) GAT on Ogbn-products



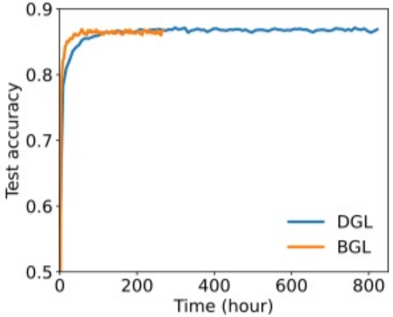
(c) GraphSAGE on Ogbn-papers



(d) GAT on Ogbn-papers



(e) GraphSAGE on User-Item



(f) GAT on User-Item

Conclusion

- **We find the performance of existing GNN training systems are limited by the data I/O and preprocessing bottleneck**
- **We propose BGL to alleviate preprocessing bottleneck**
  - Feature cache engine to reduce the traffic of feature retrieving
  - Novel graph partition algorithm to reduce the traffic of subgraph sampling
  - Profiling-based resource allocation to reduce resource contention
- **BGL outperforms four state-of-the-art systems**
  - The improvements ranges from 1.91x to 20.68x
- **We will open source BGL on github**
  - https://github.com/leodestiny/BGL_NSDI2023

Thanks everyone for listening!

Q&A