RICE UNIVERSITY

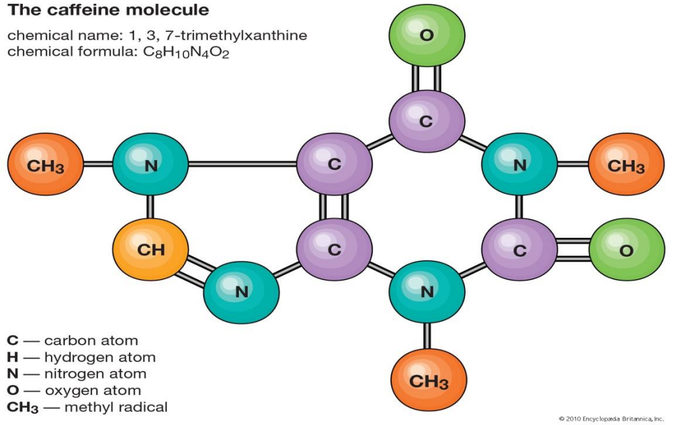
Week-8: Graph Neural Networks

Yuke Wang

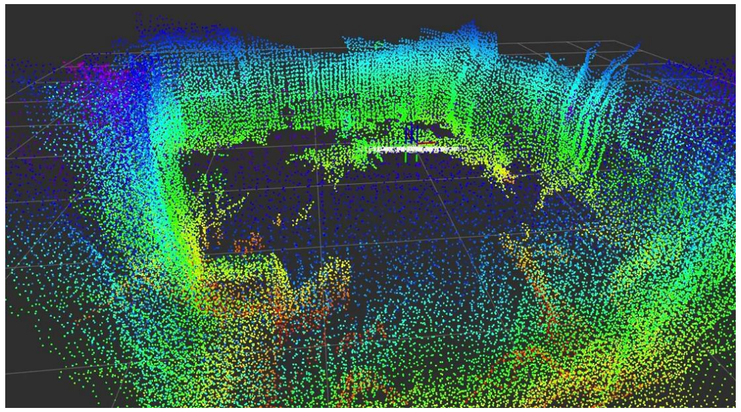
Graphs are everywhere...

Social Networks

Financial Services



Molecular chemistry



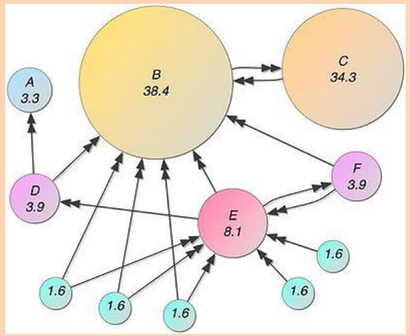
Point Cloud

Power Grid

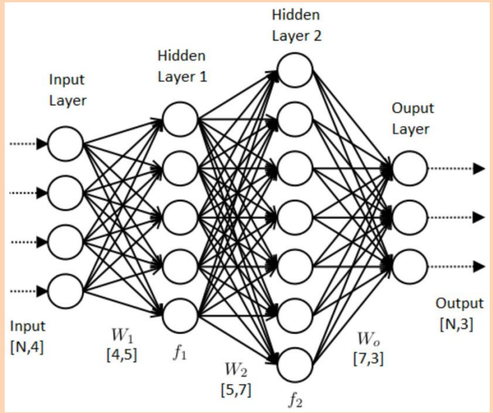
Molecular Biology

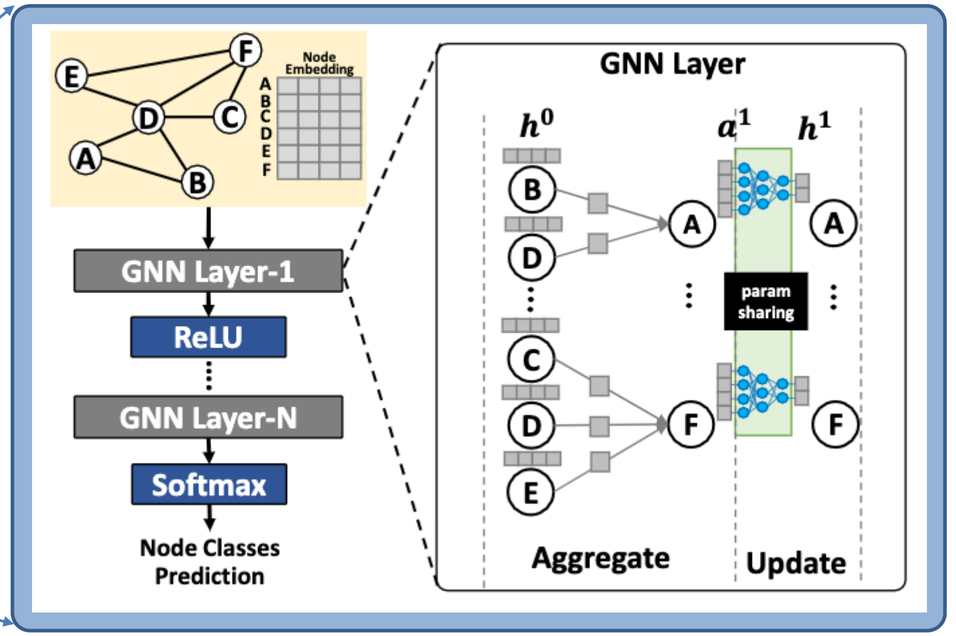
GNN is the key for graph learning

GNN =

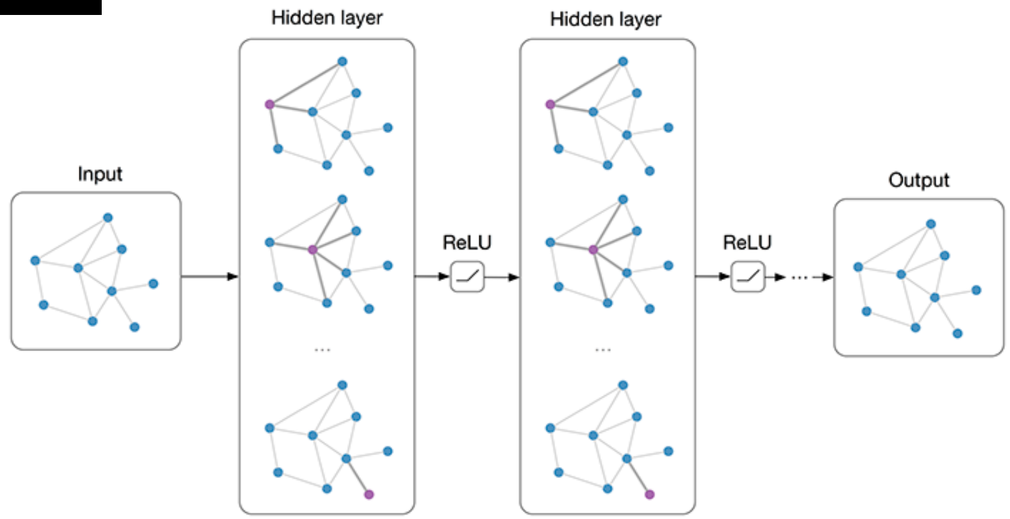


+

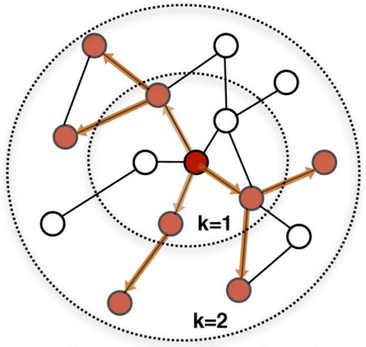




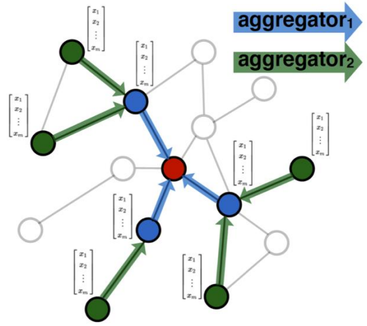
GCN



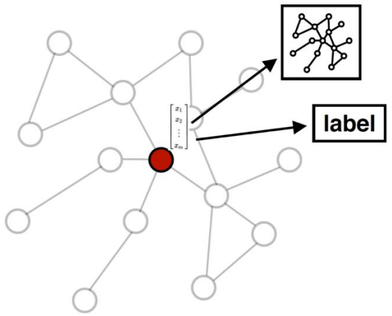
GraphSAGE



1. Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

GNN: *Challenges*

- **Diverse GNN Input:**
  - Real-world graphs are huge in size, such as billions/trillions of nodes/edges.
  - Real-world graphs have highly irregular structure (random edge connections).
- **Diverse GNN Computation:**
  - GNN sparse computation (scatter-and-gather) is intrinstically inefficient for memory access and ill-suited for parallelization on modern platforms.
  - GNN computation is an iterative process, with non-trivial computation cost and data movements.

Socal Network graphs, such as Twitter graph from SNAP[1], consists of 476 million tweets.

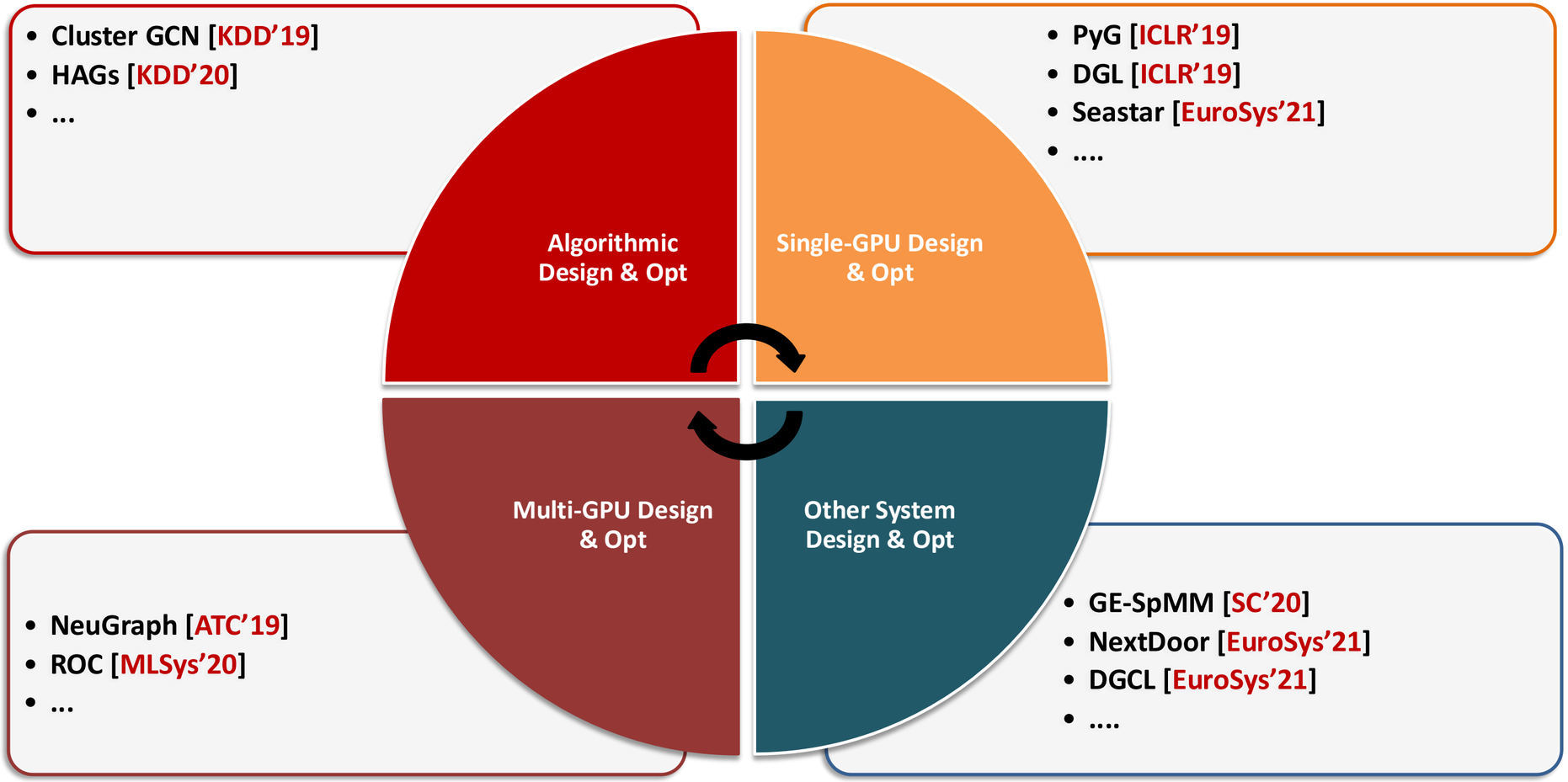
Real-world graphs usually demonstrate highly-scatter distribution of node degrees.

For an graphs with *E* edges and each nodes with *D* dimension, the time complexity of a single SAG operation is high as O(ED).

General GNN training processs consists of several hunderd iterations to converge.

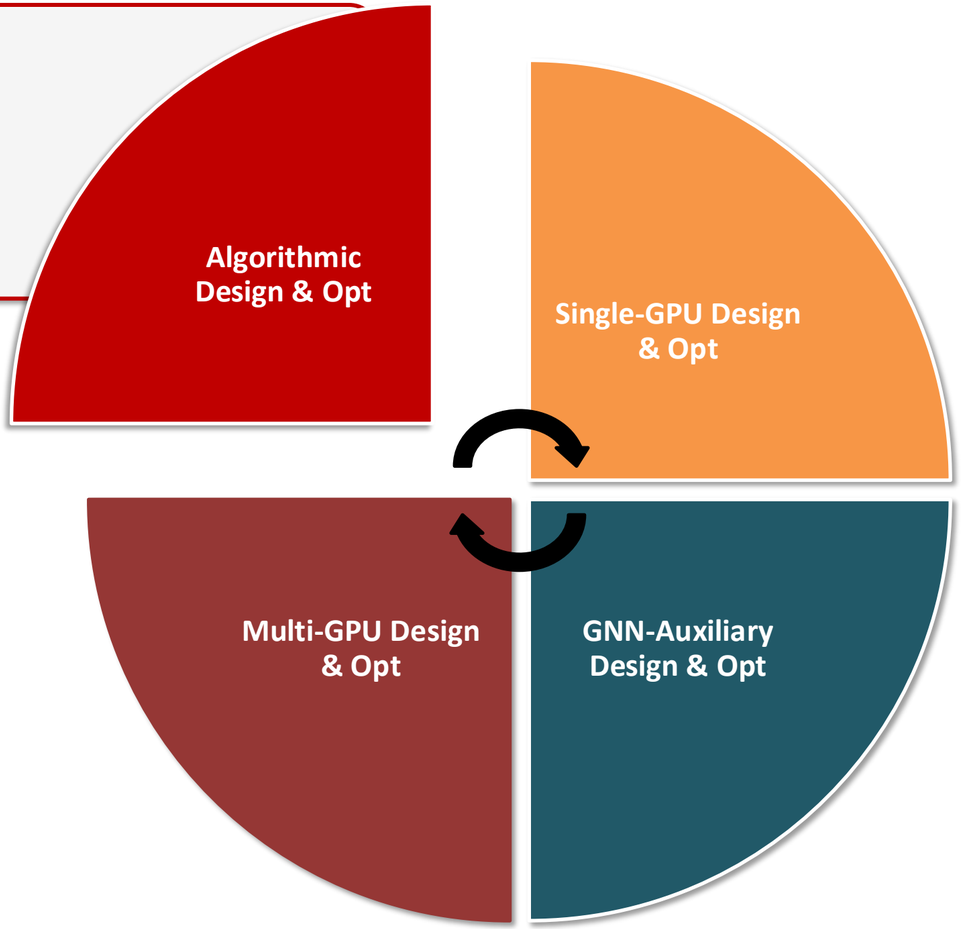
[1] SNAP Datasets: Stanford Large Network Dataset Collection.

Overview


• Cluster GCN [KDD'19]
• HAGs [KDD'20]
• …
Algorithmic Design & Opt
Single-GPU Design & Opt
• PyG [ICLR'19]
• DGL [ICLR'19]
• Seastar [EuroSys'21]
• ….
Multi-GPU Design & Opt
Other System Design & Opt
• NeuGraph [ATC'19]
• ROC [MLSys'20]
• …
• GE-SpMM [SC'20]
• NextDoor [EuroSys'21]
• DGCL [EuroSys'21]
• ….

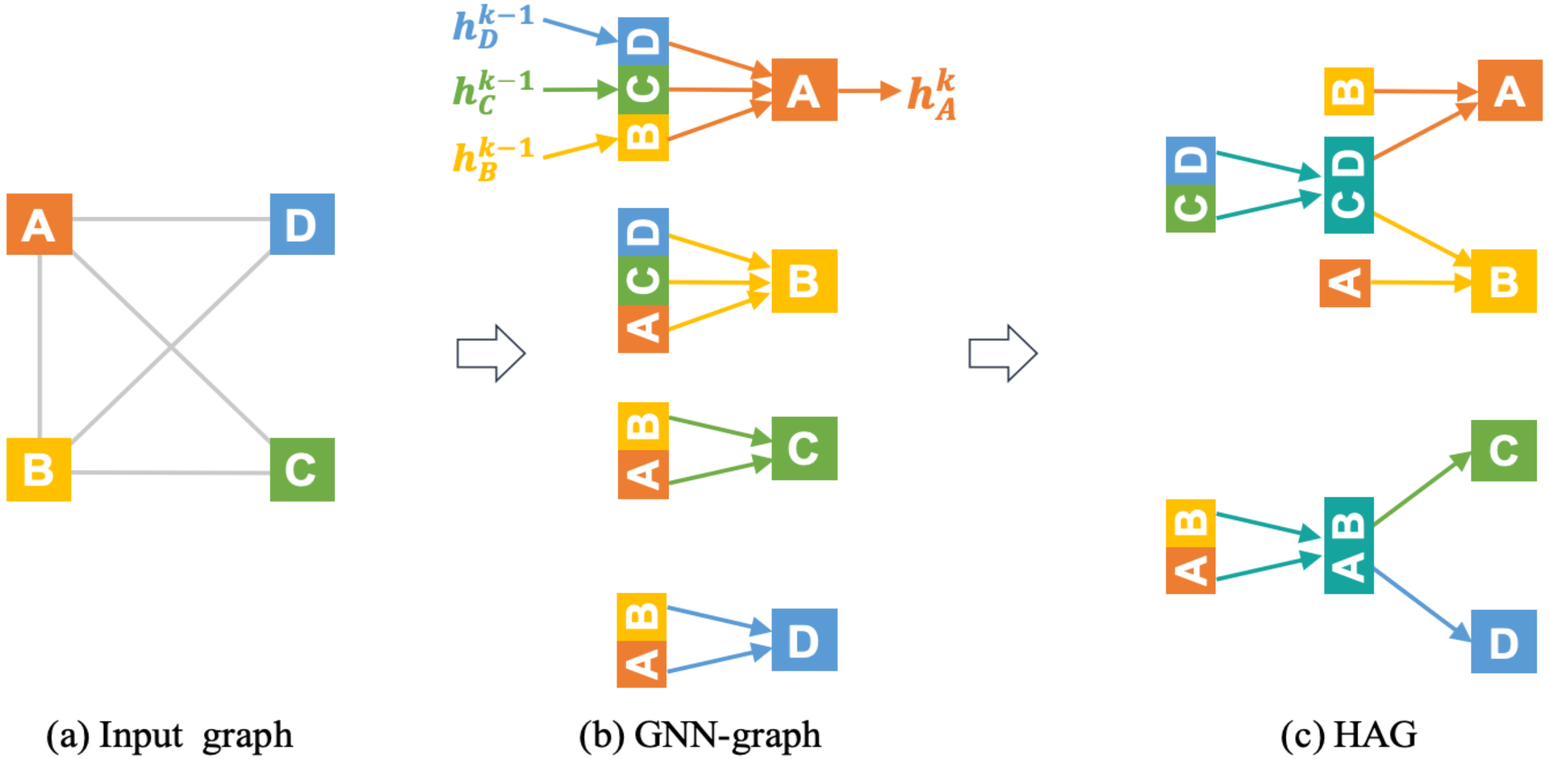
Algorithmic Design & Optimization

- Cluster GCN [KDD'19]
- HAGs [KDD'20]
- ...



HAG

Jia, et al. ***Redundancy-free computation graphs for graph neural networks.*** KDD'20.



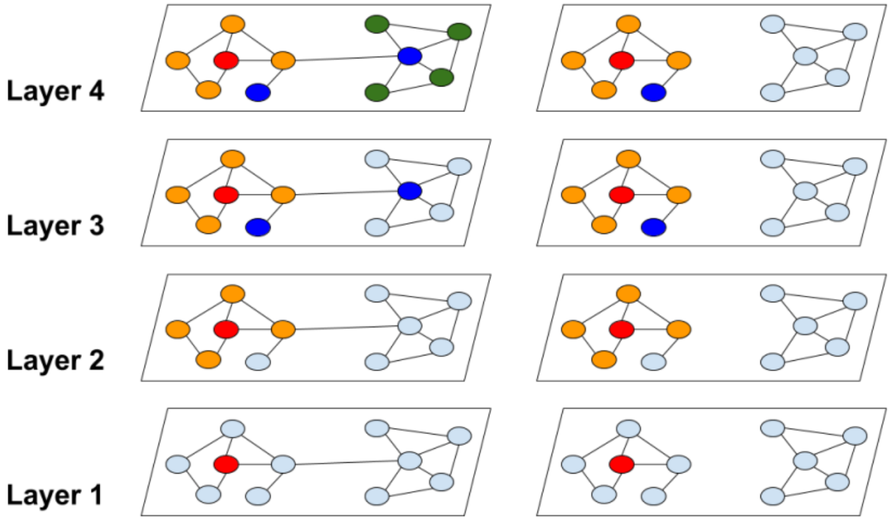
- **Problem:** common neighbors are shared between different nodes, this leads to repeated and inefficient computations.
- **Highlight:** A new GNN graph representation that explicitly avoids redundancy by managing intermediate aggregation results hierarchically.

Increasing the end-to-end training throughput by up to 2.8x and reducing the aggregations and data transfers in GNN training by up to 6.3x and 5.6x, while maintaining the original model accuracy.

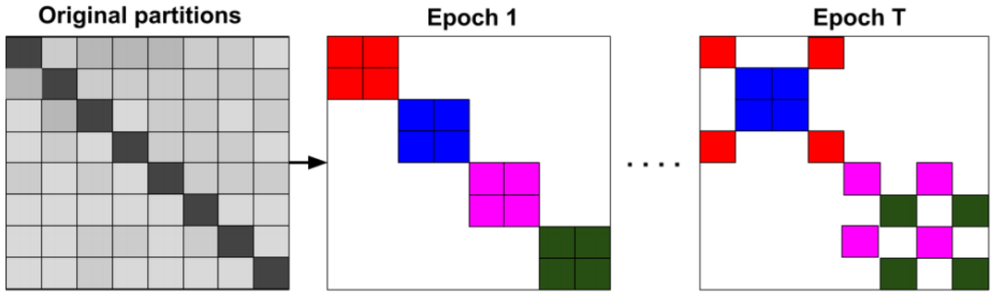
Cluster GCN

Chiang, et al. ***Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks.*** SIGKDD'2019.

- **Cluster-based Neighbor Expansion.**



- **Stochastic Multiple Partitions Scheme.**

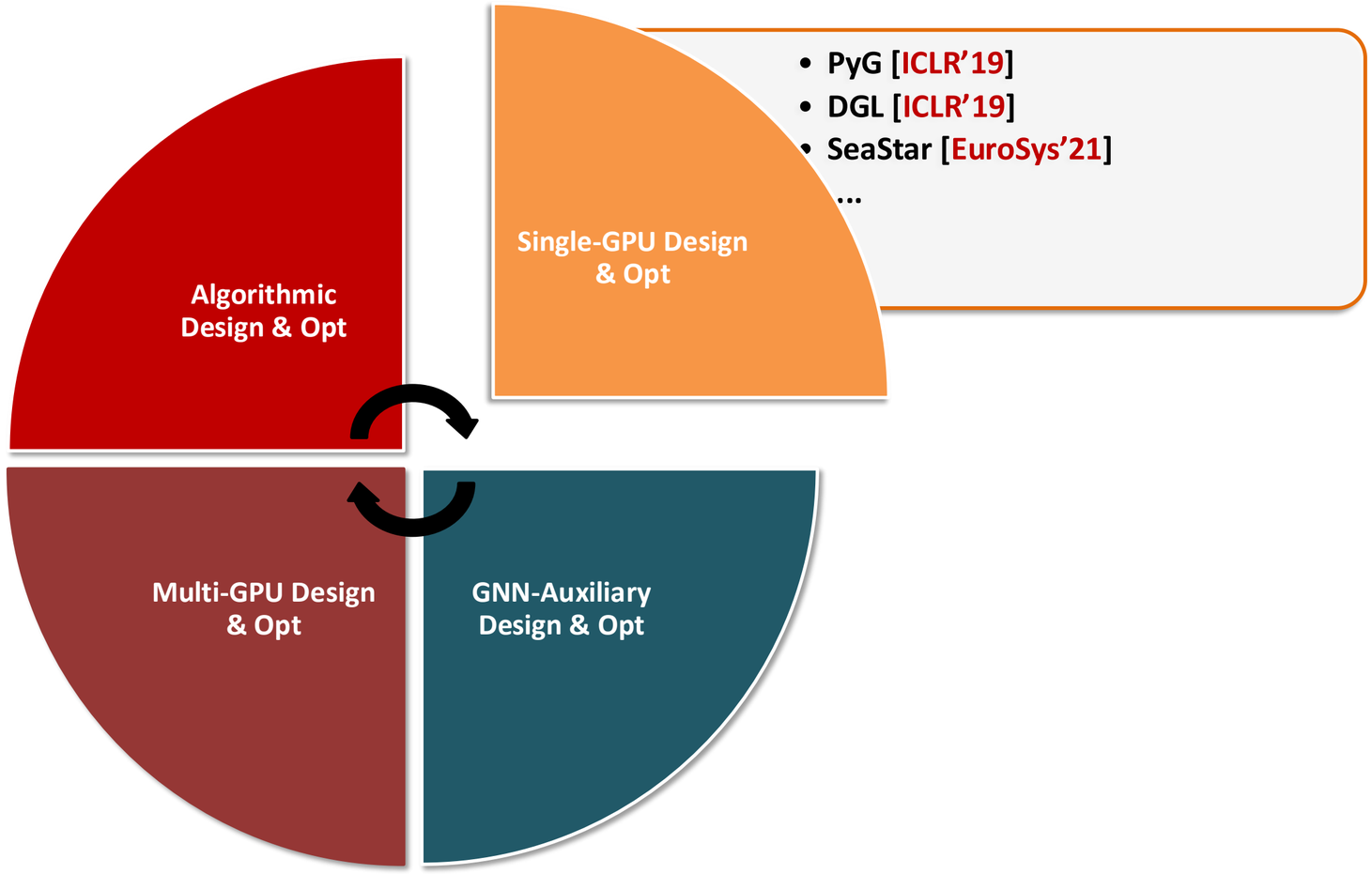


- **<u>Problem:</u>** Existing GNN training algorithms suffer from a high computational cost for keeping the entire graph and the embedding of each node in memory.

- **<u>Highlight:</u>** A novel GCN algorithm that is suitable for SGD-based training by exploiting the graph clustering structure.

Training a 3-layer GCN on Amazon2M with 2 million nodes and 61 million edges (5x of Reddit), Cluster-GCN is faster than the previous state-of-the-art VR-GCN (1523 seconds vs 1961 seconds) and using much less memory (2.2GB vs 11.2GB).

Single-GPU Design & Optimization

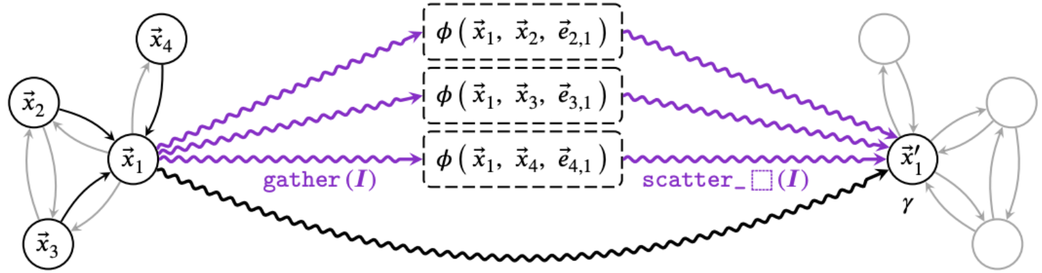


Pytorch-Geometric

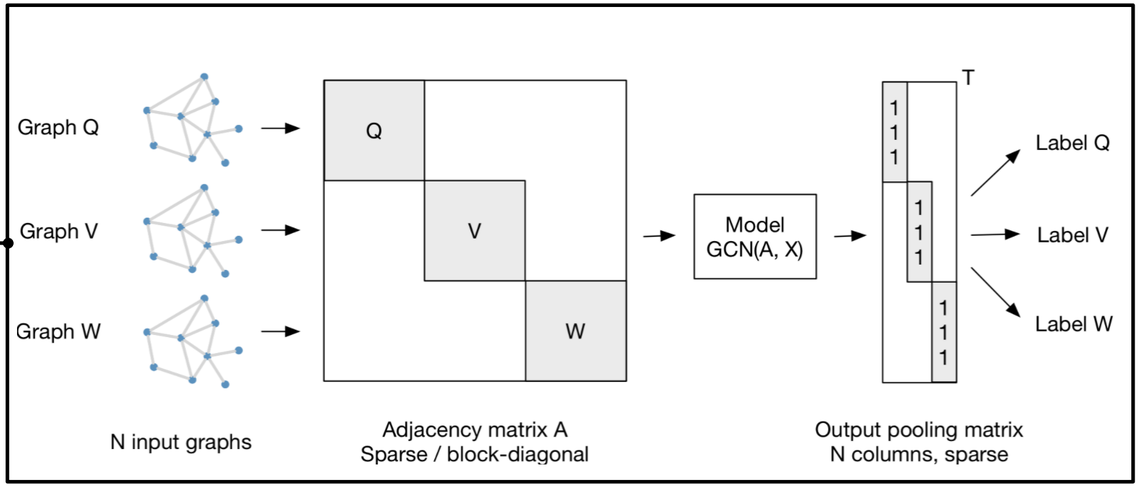
Fey et al. ***Fast graph representation learning with PyTorch Geometric***. ICLR'19.

1. Programming Model.

$$\vec{x}'_i = \gamma\left(\vec{x}_i, \square_{j \in \mathcal{N}(i)} \phi\left(\vec{x}_i, \vec{x}_j, \vec{e}_{j,i}\right)\right)$$



2. Mini-Batch Handling.



Deep Graph Library

Wang, et al. ***Deep graph library: A graph-centric, highly-performant package for graph neural networks.*** ICLR'19.

1. Programming Model.

$$\text{Edge-wise: } \mathbf{m}_k^{(t)} = \phi^e(\mathbf{e}_k^{(t-1)}, \mathbf{v}_{r_k}^{(t-1)}, \mathbf{v}_{s_k}^{(t-1)})$$

$$\text{Node-wise: } \mathbf{v}_i^{(t)} = \phi^v(\mathbf{v}_i^{(t-1)}, \bigoplus_{\substack{k \\ \text{s.t. } r_k=i}} \mathbf{m}_k^{(t)}),$$

2. Graph Storage Management.

Use **DGLGraph** object. Minimize users efforts in choosing graph storage format. (e.g., COO, CSR)

3. Tensorization and Kernel Fusion.

Operate on a batch of edge of features.
Fuse neighbor propagation and SUM reduction.

PyG *Vs.* DGL

- **Coverage of GNN models.**

		GNet	NGra	Euler	PyG	DGL
Message Passing	arbitrary ϕ^e	✓	✓	✓	✓	✓
	arbitrary ϕ^v	✓	✓	✓	✓	✓
	arbitrary $\bigoplus$	✓	✗	✓	✗	✓
Propa-gation Order	full	✓	✓	✗	✓	✓
	partial	✗	✗	✗	✓	✓
	random walk	✗	✗	✗	✗	✓
	sampling	✗	✗	✓	✗	✓
Graph Type	many & small	✓	✗	✗	✓	✓
	single & giant	✗	✓	✓	✗	✓
	dynamic	✗	✗	✗	✗	✓
System	multi-platform	✗	✗	✗	✗	✓

- **Performance of Execution.**

Dataset		Model	Accuracy	Time		Memory	
$\|V\|$	$\|E\|$			PyG	DGL	PyG	DGL
Cora		GCN	81.31 ± 0.88	**0.478**	0.666	1.1	1.1
3K	11K	GAT	83.98 ± 0.52	1.608	**1.399**	1.2	**1.1**
CiteSeer		GCN	70.98 ± 0.68	**0.490**	0.674	1.1	1.1
3K	9K	GAT	69.96 ± 0.53	1.606	**1.399**	1.3	**1.2**
PubMed		GCN	79.00 ± 0.41	**0.491**	0.690	1.1	1.1
20K	889K	GAT	77.65 ± 0.32	1.946	**1.393**	1.6	**1.2**
Reddit		GCN	93.46 ± 0.06	*OOM*	**28.6**	*OOM*	**11.7**
232K	114M						
Reddit-S		GCN	N/A	29.12	**9.44**	15.7	**3.6**
232K	23M						

Seastar

Wu et al. ***Seastar: vertex-centric programming for graph neural networks***. Eurosys'21.

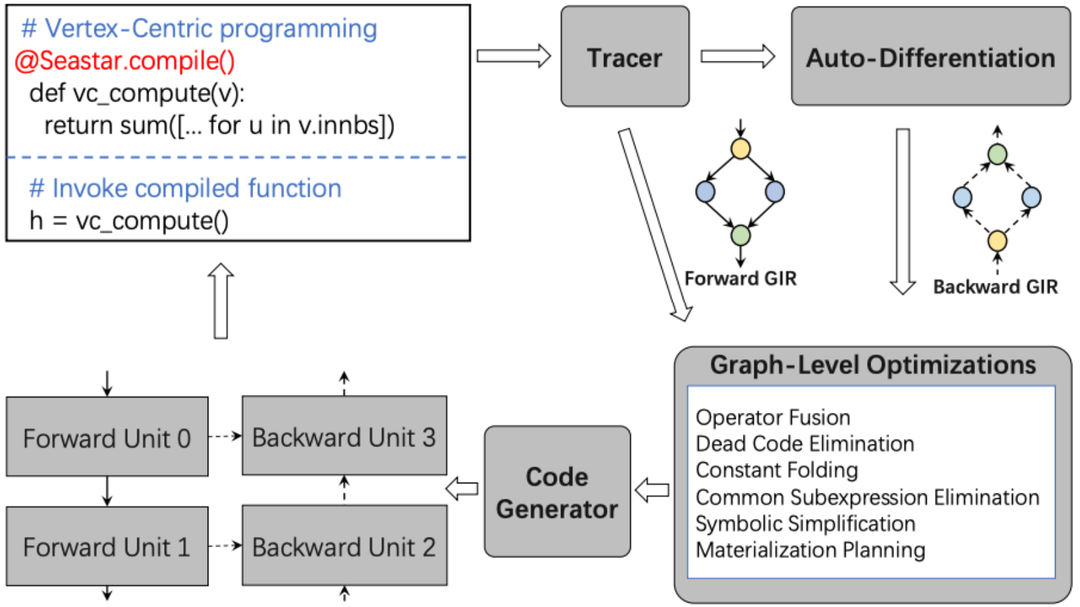
- **Problem:** Existing GNN frameworks use GNN-specific operators plugged into existing deep learning systems, incuring high memory consumption, poor data locality, and large semantic gap between algorithm design and implementation.
- **Highlight:** 1) A vertex-centric programming model for GNN training on GPU. 2) A novel optimizations to produce highly efficient fused GPU kernels for forward and backward passes in GNN training

Seastar achieves up to 2 and 8x less memory consumption, and 14 and 3x faster execution, compared with DGL and PyG.

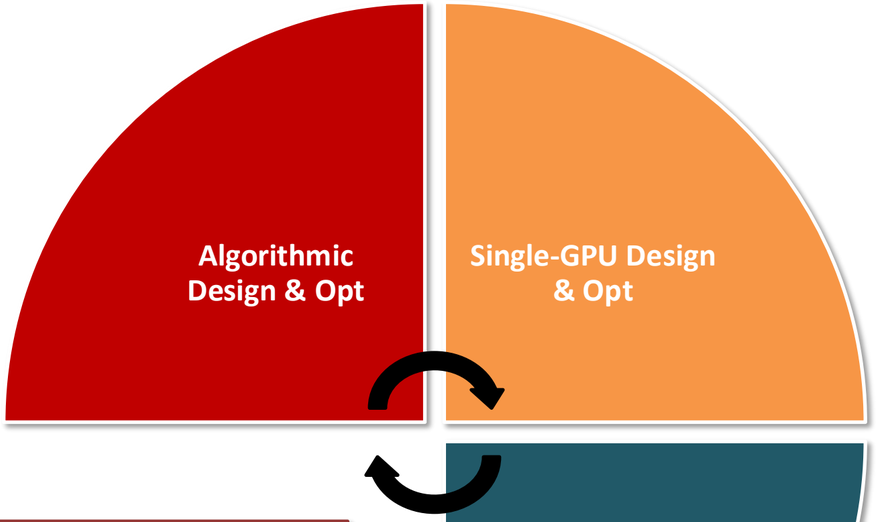
➢ Vertex-Centric Programming.

```
1  #-----------------Seastar GCN------------------#
2  def gcn_forward(self, h, norm):
3    @Seastar.compile(v_feature={'norm':norm,'h':h})
4    def vc_compute(v):
5      return sum([torch.mm(u.h, self.W) * u.norm
6                  for u in v.innbs])
7  return torch.sigmoid(vc_compute() + self.bias)
```

➢ Overall Optimization and Code Generation Flow.



Multi-GPU Design & Optimization



- NeuGraph [ATC'19]
- RoC [MLSys'20]
- …

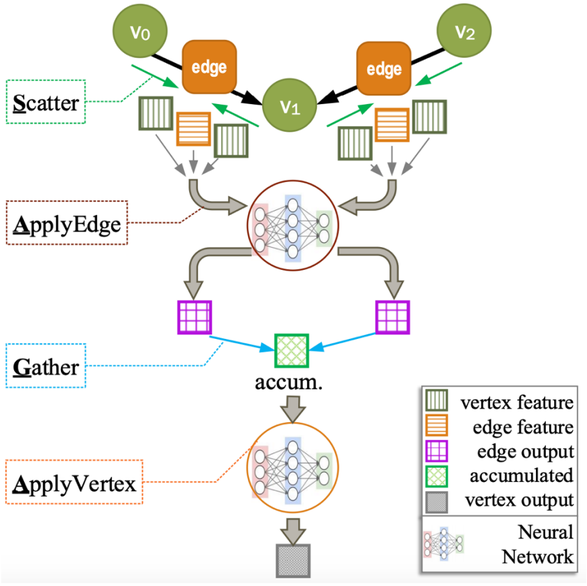
NeuGraph

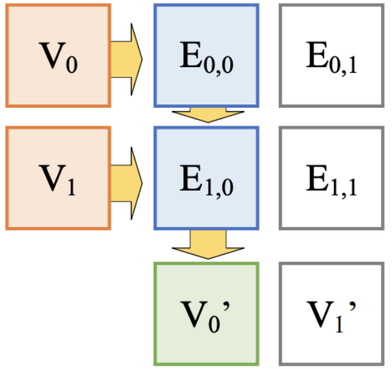
Lingxiao et al. ***NeuGraph: Parallel Deep Neural Network Computation on Large Graphs***. USENIX ATC'19.

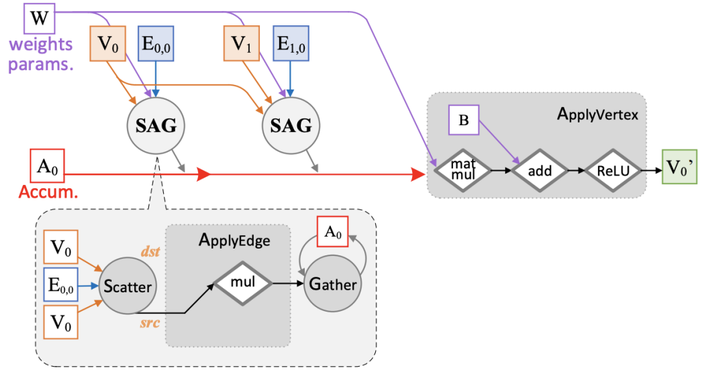
- **Problem:** Poor scalability of the existing GNN frameworks on large, densely-connected real-world graphs.
- **Highlight:**
  - A new framework that bridges the graph and dataflow models to support efficient and scalable parallel neural network computation on graphs.
  - Graph computation optimizations into the management of data partitioning, scheduling, and parallelism in dataflow-based deep learning frameworks

NeuGraph achieves up to 2.5x and 5.0x speedup over the Tensorflow-based GNN design.

➢ **SAGA Programming Model & GNN Dataflow Translation.**

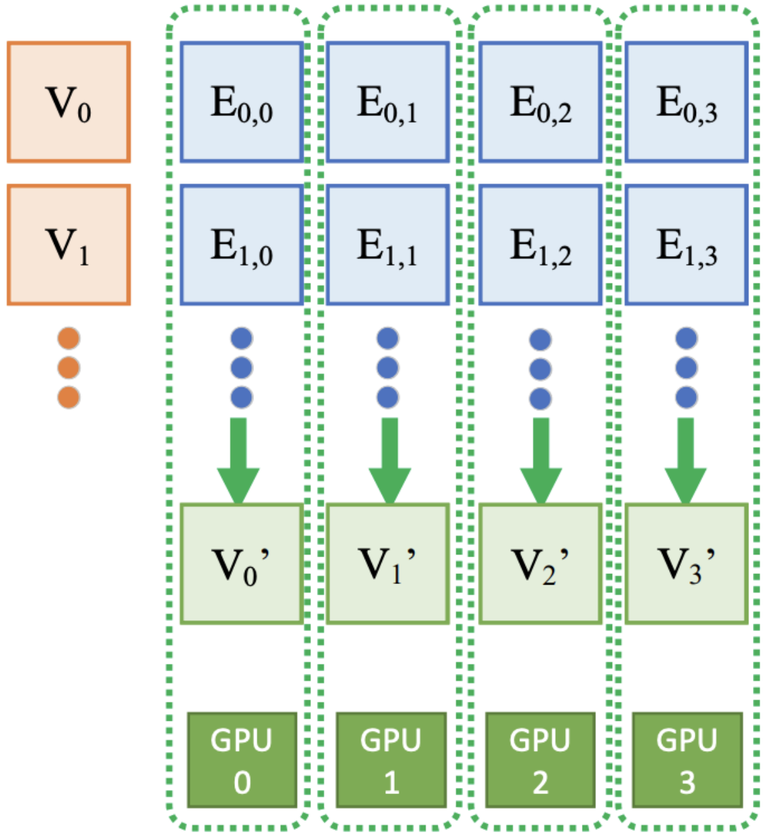




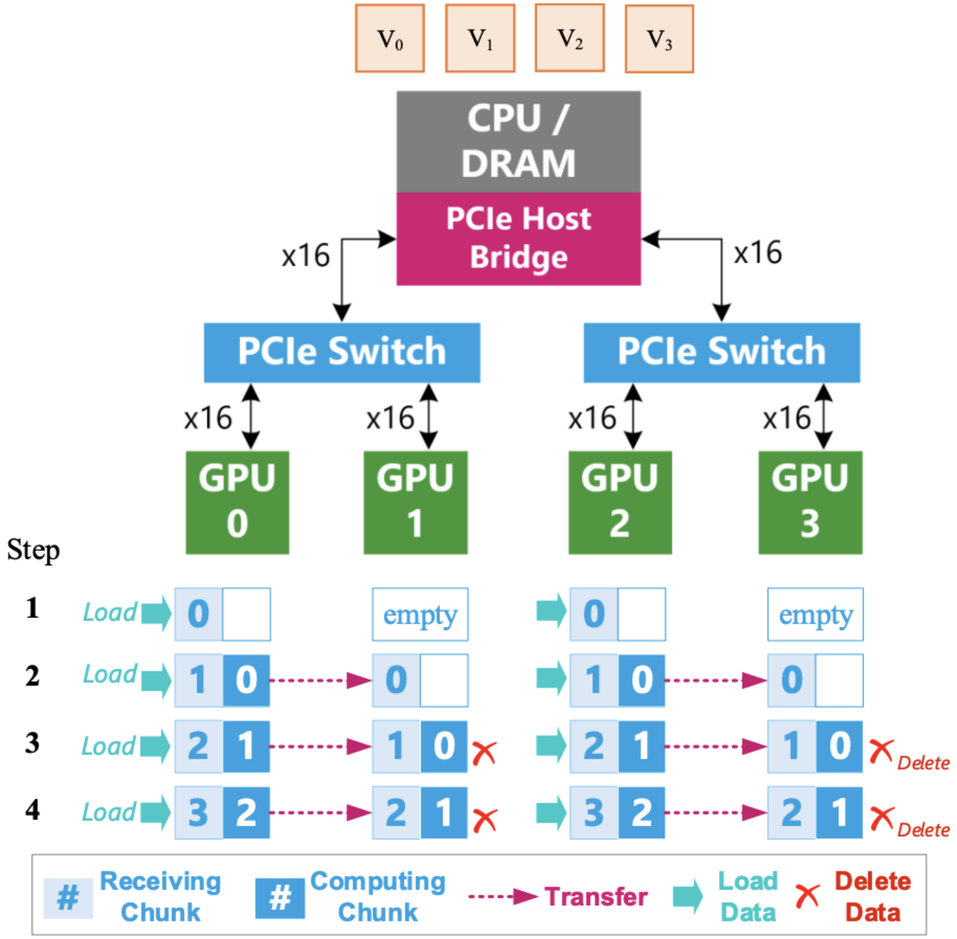


NeuGraph: *Multi-GPU Support*

- **2-D Graph Partitioning.**



- **Chain-based Graph Data Forwarding.**



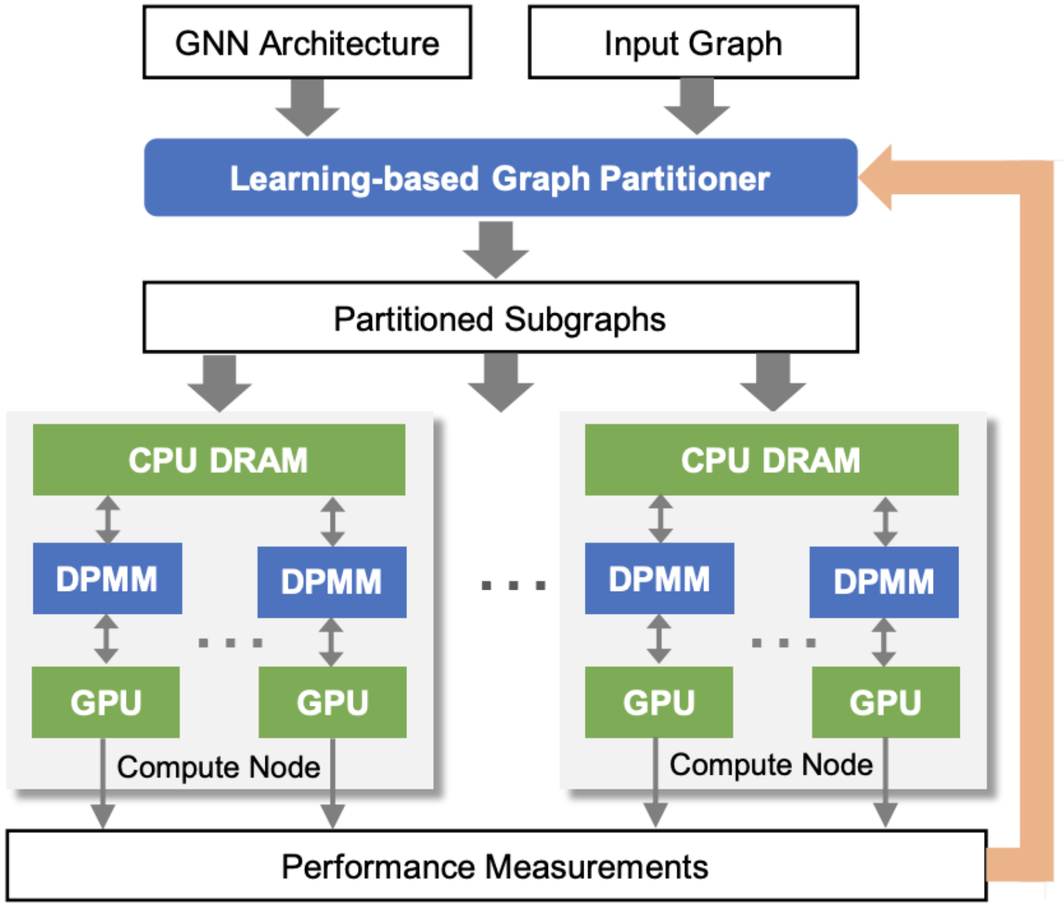
ROC

Z.jia. ***Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with Roc.*** MLsys'20.

- **Problem:** The current lack of system support has limited the potential application of GNN algorithms on large-scale graphs, and has also prevented the exploration of larger and more sophisticated GNN architectures.

- **Hightlight:**
  - A distributed multi-node multi-GPU framework for fast GNN computation.
  - Learning-based graph partitioning.
  - Communication-Minimization Memory Management.
  - Full graph training on larger and deeper GNNs (e.g., 1.5% over sampling method on Reddit).

ROC is up to 4× faster than existing GNN frameworks on a single machine, and can scale to multiple GPUs on multiple machines.



Roc: *Partitioning and Caching*

➢ Cost Model Learning-based Partitioning

We formalize the cost for running a GNN layer l on an input graph $\mathcal{G}$ as follows.

$$t(l,v) = \sum_i w_i(l)x_i(v) \tag{4}$$

$$t(l,\mathcal{G}) = \sum_{v\in\mathcal{G}} t(l,v) = \sum_{v\in\mathcal{G}}\sum_i w_i x_i(v) \tag{5}$$

$$= \sum_i w_i \sum_{v\in\mathcal{G}} x_i(v) = \sum_i w_i x_i(\mathcal{G}) \tag{6}$$

where v denotes a vertex in the input graph $\mathcal{G}$, $w_i(l)$ is a trainable parameter for layer l, $x_i(v)$ is the i-th feature of v, and $x_i(\mathcal{G})$ sums up the i-th feature of all vertices in $\mathcal{G}$.

Our model minimizes the mean square error over all available data points.

$$Loss(l) = \frac{1}{N}\sum_{i=1}^{N}\big(t(l,\mathcal{G}_i) - y(l,\mathcal{G}_i)\big)^2 \tag{7}$$

➢ Cost-Minimization Tensor Caching

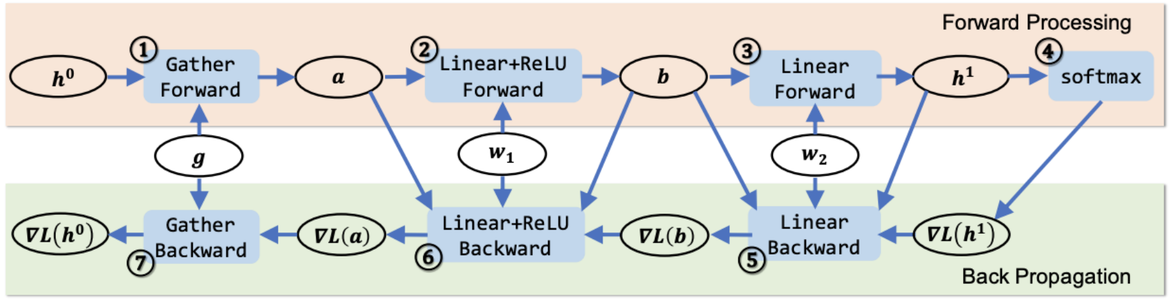


Table 3. All the valid states and their activation tensors for the GNN architecture in Figure 3.

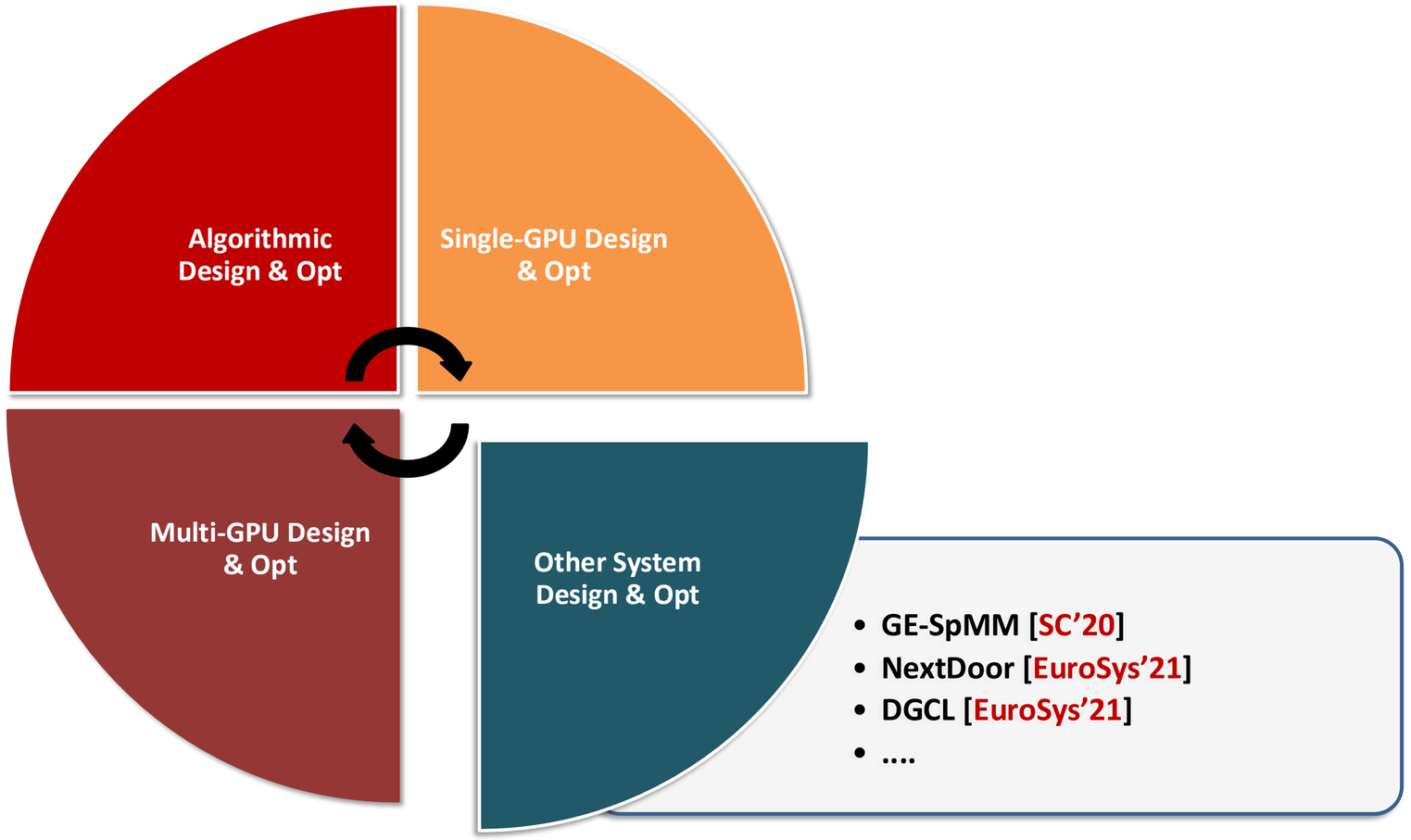
Valid State $\mathcal{S}$	Activation Tensors $\mathcal{A}(\mathcal{S})$
{①}	$\{g, a\}$
{①, ②}	$\{g, a, b, w_1\}$
{①, ②, ③}	$\{g, a, b, h^1, w_1, w_2\}$
{①, ②, ③, ④}	$\{g, a, b, w_1, w_2, \nabla L(h^1)\}$
{①, ②, ③, ④, ⑤}	$\{g, a, b, w_1, \nabla L(b)\}$
{①, ②, ③, ④, ⑤, ⑥}	$\{g, a, \nabla L(a)\}$
{①, ②, ③, ④, ⑤, ⑥, ⑦}	{}

Algorithm 1 A recursive dynamic programming algorithm for computing minimum data transfers. $\textsc{In}(o_i)$ and $\textsc{Out}(o_i)$ return the input and output tensors of the operation o_i, respectively, and size($\mathcal{T}$) returns the memory space required to save all tensors in $\mathcal{T}$.

Input: An input graph g, a GNN architecture $\mathcal{G}$, and the GPU device memory capacity *cap*.
Output: Minimum data transfers required to compute $\mathcal{G}$ on g within capacity *cap*.
▷ $\mathcal{D}$ is a database storing all computed $\textsc{Cost}$ functions.

function $\textsc{Cost}(\mathcal{S}, \mathcal{T})$
  if $(\mathcal{S}, \mathcal{T}) \in \mathcal{D}$ **then**
    return $\mathcal{D}(\mathcal{S}, \mathcal{T})$
  if $\mathcal{S}$ is $\varnothing$ **then**
    return size($\mathcal{T}$)
  $cost \leftarrow \infty$
  for $o_i \in \mathcal{S}$ **do**
    if $(\mathcal{S} \setminus o_i)$ is a valid state **then**
      $\mathcal{S}' \leftarrow \mathcal{S} \setminus o_i$
      $\mathcal{T}' \leftarrow (\mathcal{T} \setminus \textsc{Out}(o_i)) \cap \mathcal{A}(\mathcal{S}')$
      $xfer \leftarrow \text{size}(\textsc{In}(o_i) \setminus \mathcal{T}')$
      if $\text{size}(\mathcal{T} \cup \textsc{In}(o_i) \cup \textsc{Out}(o_i)) \le cap$ **then**
        $cost = \min\{cost, \textsc{Cost}(\mathcal{S}', \mathcal{T}') + xfer\}$
  $\mathcal{D}(\mathcal{S}, \mathcal{T}) \leftarrow cost$
  return $\mathcal{D}(\mathcal{S}, \mathcal{T})$

Other System Design & Optimization



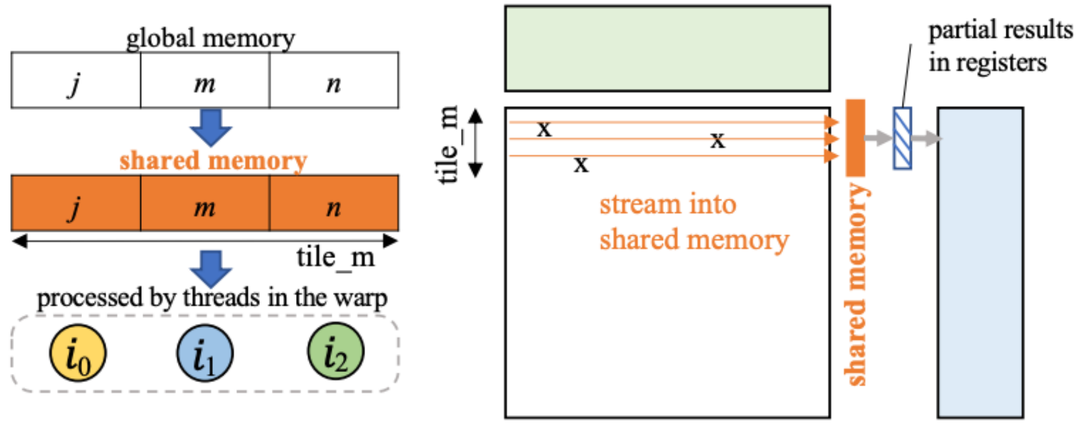
GE-SpMM

Guyue Huang et al. ***GE-SpMM: general-purpose sparse matrix-matrix multiplication on GPUs for graph neural networks.*** (SC'20)

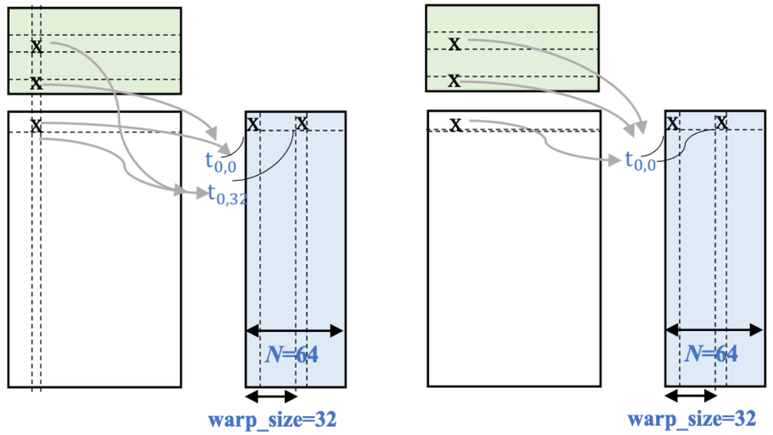
- **Problem:**
  - General GNN algorithms require SpMM-like operations (e.g., pooling) are not supported in current high-performance GPU libraries.
  - High-overhead data format conversion.
  - Lack of SpMM tailored optimizations.

- **Hightlight:**
  - No preprocessing overheads and support general GNN algorithms.
  - Coalesced Row Caching method to process columns in parallel and ensure efficient coalesced access to the GPU global memory.
  - Coarsegrained Warp Merging method to reduce redundant data loading among GPU warps.

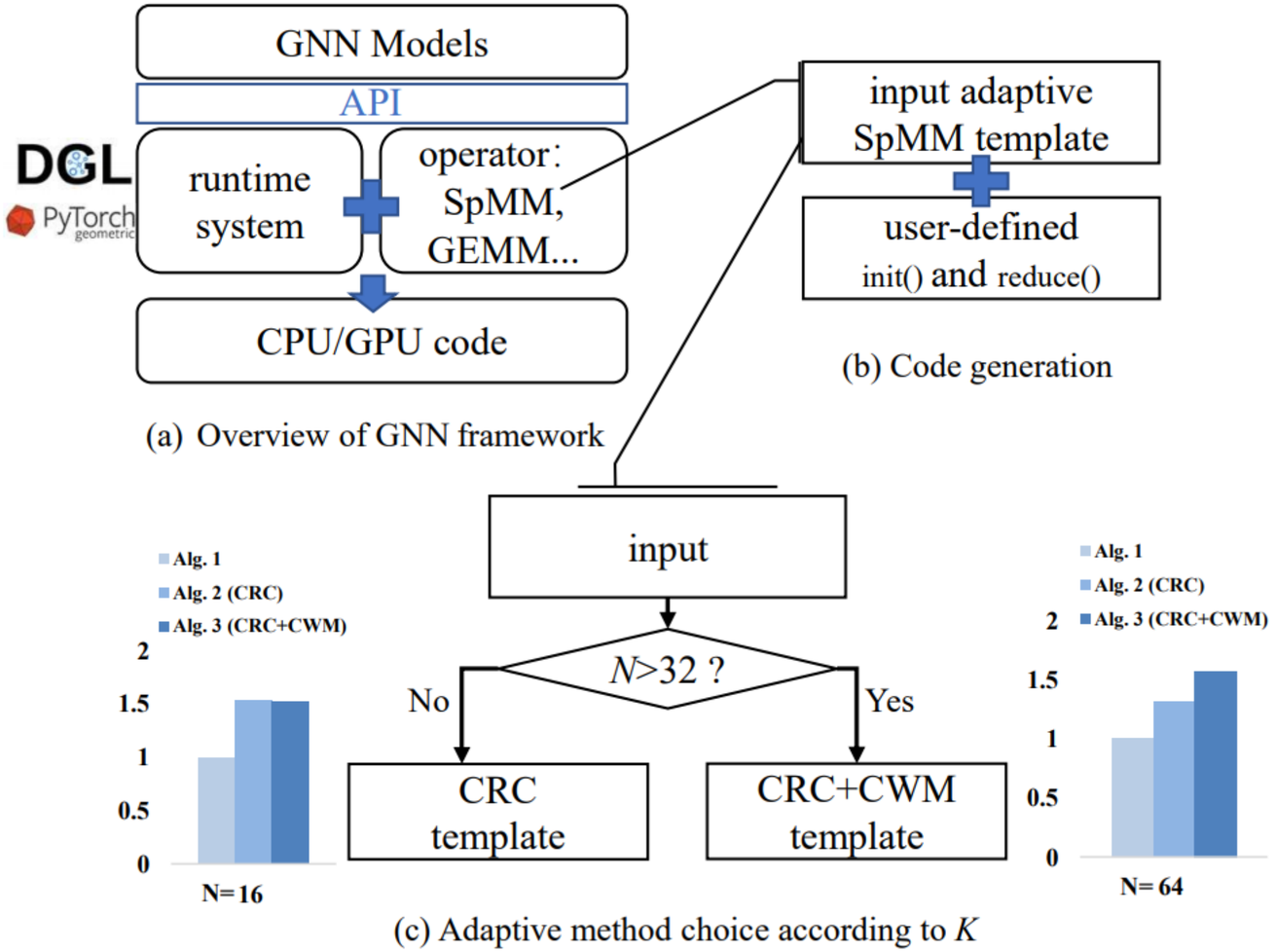
➢ **Coalesced Row Caching**



➢ **Coarse-grained Warp Merging**



GE-SpMM: *Overall Flow and Integration*



up to 1.41× speedup over NVIDIA cuSPARSE
up to 1.81× over GraphBLAST.
3.67× speedup on GCN and GraphSAGE.

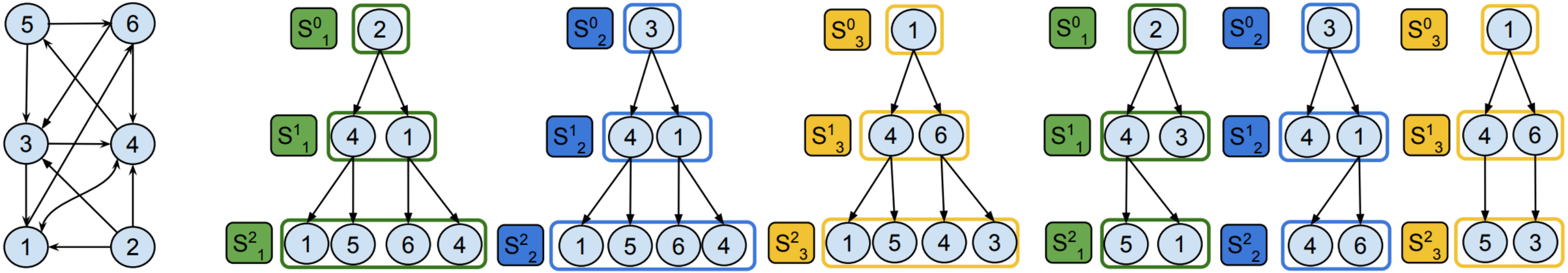
NextDoor

Abhinav Jangda et al. ***Accelerating graph sampling for graph machine learning using GPUs***. EuroSys '21

- **Problem:** Sampling is an "embarrassingly parallel" problem and may appear to lend itself to GPU acceleration but the irregularity of graphs makes it hard to use GPU resources effectively.

- **Hightlight:** 1) A system designed to effectively perform graph sampling on GPUs; 2) a new approach to graph sampling that we call transit-parallelism, which allows load balancing and caching of edges; 3) a high-level abstraction for writing a variety of graph sampling algorithms

---



(a) Graph **(b)** 2-hop Neighborhood sampling **(c)** Layer Sampling

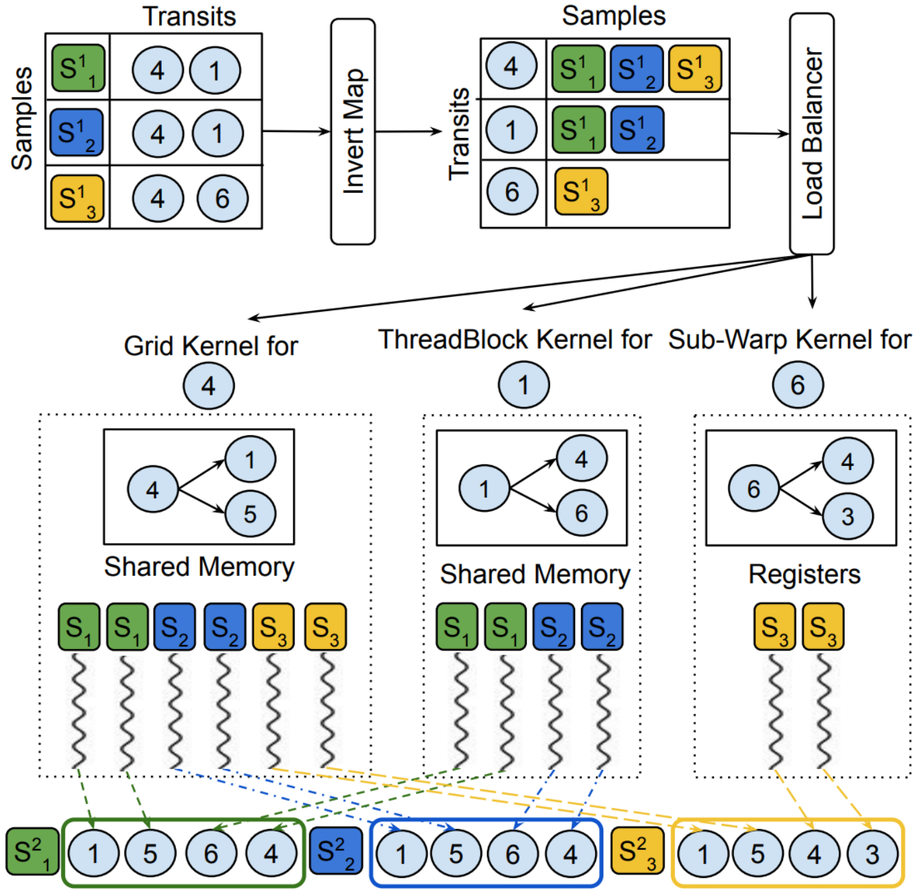
NextDoor: *Design and Optimizations*

- **API for GraphSAGE 2-hop sampling.**

```
1 Vertex next(s, transits, srcEdges, step) {
2   int idx = randInt(0, srcEdges.size());
3   return srcEdges[idx];}
4 int steps() {return 2;}
5 int sampleSize(int step) {
6   return (step == 0) ? 25 : 10;}
7 bool unique(int step) {return false;}
8 SamplingType samplingType()
9 {return SamplingType::Individual;}
10 Vertex stepTransits(step, s, transitIdx)
11 {return s.prevVertex(1, transitIdx);}
```

NextDoor provides an order of magnitude speedup over KnightKing (Yang et al, SOSP'19) for all random walk applications, with speedups ranging from 26.1x to 50x.

- **Transit Parallel execution of second step of sampling.**

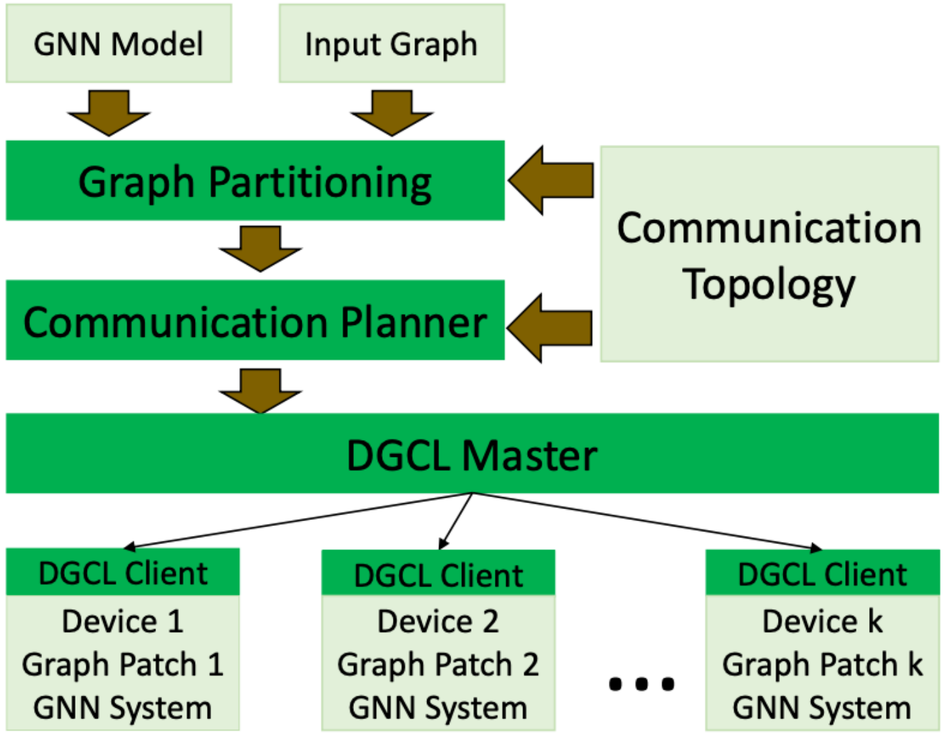


DGCL

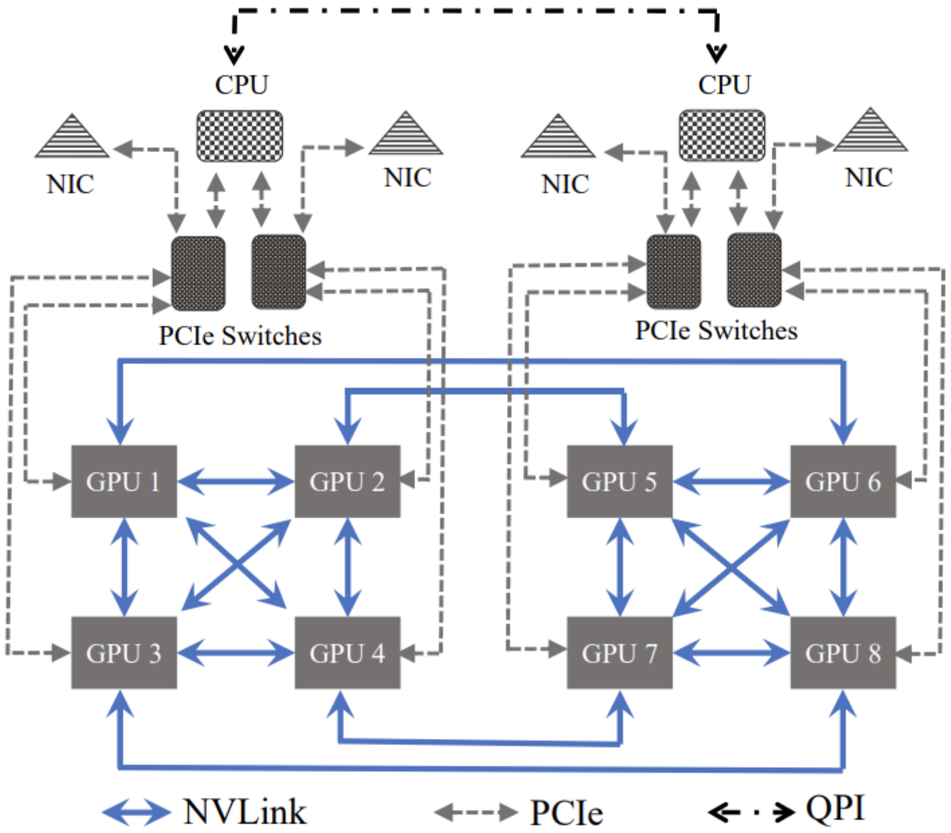
Cai, Zhenkun, et al. ***DGCL: an efficient communication library for distributed GNN training.*** Eurosys 2021.

- **Problem:**
  - Distributed GNN training, a peer-to-peer communication strategy suffers from high communication overheads.
  - Different GPUs require different remote vertex embeddings, which leads to an irregular communication pattern and renders existing communication planning solutions unsuitable.
- **Hightlight:**
  - A distributed graph communication library (DGCL) for efficient GNN training on multiple GPUs.
  - A communication planning algorithm tailored for GNN training
  - DGCL can be easily adopted to extend existing single-GPU GNN systems to distributed training.

➢ **Overview of DGCL**



DGCL: *Algorithmic design*

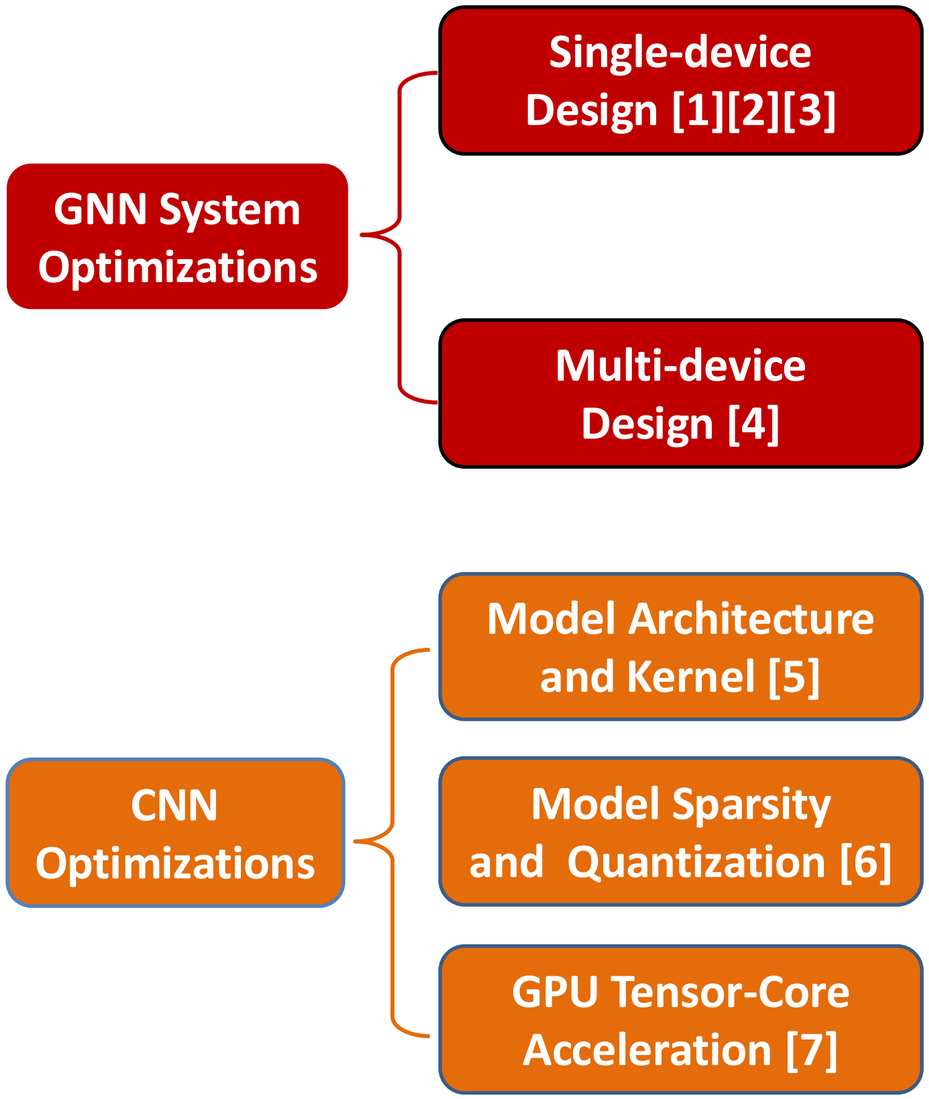


Algorithm 1 Shortest Path Spanning Tree (SPST)

Input: The communication topology graph $D(V', E')$, the source GPU s_u and the destination GPUs D_u for each vertex $u \in V$.
Output: The communication strategies for all vertices in V.
for $u \in V$ **do**
  $N_{src} = \{s_u\}$
  for i from 1 to $|D_u|$ **do**
    $C = incrmentalLinkCost(D, S)$ ▷ *Use Algorithm 2*
    Run $dijkstra(N_{src}, D, C, D_u \backslash N_{src})$
    Let p be the shortest path among the set of shortest paths from each $d \in N_{src}$ to each $d' \in D_u \backslash N_{src}$ computed in Dijkstra algorithm
    $N_{src} = N_{src} \cup \{$ vertices on $p\}$
    $S = S \cup \{$ edges on $p\}$
  end for
end for
Return: S

Reduce the communication time of the peer-to-peer communication by 77.5% on average and the training time for an epoch by up to 47%.

My Research



[1] Y. Wang et al. ***GNNAdvisor: An Efficient Runtime System for GNN Acceleration on GPUs.*** **OSDI'21**
[2] Y. Wang et al. ***TC-GNN: Accelerating Sparse Graph Neural Network Computation Via Dense Tensor Core on GPUs*** (In Submssion)
[3] Y. Wang et al. ***QGTC: Accelerating Quantized GNN via GPU Tensor Core.*** (In Submssion)
[4] Y. Wang *et al.* ***Accelerating Graph Neural Networks on Multi-GPU Platforms*** (In Progress)

[5] Y. Wang et al. ***DSXplore: Optimizing Convolutional Neural Networks via Sliding-Channel Convolutions***. **IPDPS'21**.
[6] L. Liu et al. ***Boosting Deep Neural Network Efficiency with Dual-Module Inference*** **ICML'20**.
[7] B. Feng et al. ***APNN-TC: Accelerating Arbitrary Precision Neural Networks on Ampere GPU Tensor Core*** (In Submission)

RICE UNIVERSITY

yuke.wang@rice.edu