D-MGN: A Distributed Mesh Graph Neural Network for Scalable Engineering Simulation

Fu Lin*, Jiasheng Shi*, Weixiong Rao*, Jiachuan Wang†, Lei Chen†
Chunyan Zhu‡, Haihua Wang‡, Shuo Xie§
* *Tongji University, China*
† *Hong Kong University of Science and Technology, China*
‡ *Yanfeng International Automotive Technology Co., Ltd., China*
§ *Yanfeng International Seating System Co., Ltd., China*
*{2231531, wxrao}@tongji.edu.cn; † jwangey@connect.ust.hk, † leichen@cse.ust.hk

***Abstract*—With the success of deep learning, researchers have developed Graph Neural Networks (GNNs) in the domain of engineering simulation. To represent simulation objects, GNN models are built on mesh graphs consisting of mesh elements. Unfortunately, due to the difference between engineering simulation and well-known GNN applications such as social networks, existing distributed GNN models do not perform well on large-scale mesh graphs, suffering from intensive message passing and communication overhead across distributed workers. To tackle this issue, in this paper, we propose a novel distributed mesh graph network D-MGN. By a vertex-cut partitioning policy, D-MGN divides large mesh graphs into partitions. After the partitions are assigned to distributed workers, D-MGN allows local communication-free message passing within each worker, and performs distributed message aggregation with rather low communication cost. Evaluation on an example dataset demonstrates that D-MGN outperforms a recent work SAC by 38.66% lower errors, 54.67% faster convergence time, and 119.22% higher speedup ratio on 8 GPUs.**

***Index Terms*—Distributed Graph Neural Network (GNN), Graph Partitioning, Numerical Simulation.**

I. Introduction

With the recent success of deep learning, researchers have developed various learning models, particularly Graph Neural Networks (GNNs), in the engineering domain of numerical simulation. Example applications include structural mechanics [1], [2] and aerodynamics [3], [4] . Unlike traditional numerical simulations with running time ranging from minutes to even days, GNN models achieve much faster prediction time (e.g., milliseconds) and acceptable accuracy. To represent engineering simulation, GNN models are built on mesh graphs, consisting of discretized mesh elements (e.g., triangles or tetrahedra). The number of mesh elements is typically large, e.g., tens of thousands or millions, for accurate simulation.

Unfortunately, existing distributed GNN models [5]–[7] may not perform well on large mesh graphs. This is mainly due to the difference between engineering simulation and the widely studied GNN applications, e.g., social networks. Specifically, the node degrees of social networks frequently follow a power-law distribution. Distributed GNN models [5]–[7] frequently adopt the so-called *edge-cut* graph partitioning policy on social networks for parallel training. That is, each graph vertex is within only one worker, and some graph edges are across distributed workers. Yet, the degree distribution of mesh graphs in this paper is rather even. As shown in the example stress field simulation of Fig. 1, the car steering wheel is discretized into mesh elements with an even distribution of node degrees. Our real steering wheel dataset shows that node degrees range from 3 to 39 with an average of 5.55. Given the even mesh graphs, the edge-cut partitioning policy suffers from many edge-cuts across distributed workers, at the cost of intensive message passing and communication overhead.

Corresponding author: Weixiong Rao (wxrao@tongji.edu.cn).

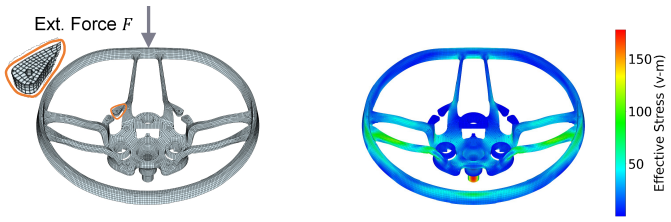

Fig. 1: Numerical Simulation of a car steering wheel. From left to right: (a) Simulation input: mesh structure and external force F. (b) Simulation result: stress field computed by a numerical solver.

To address the issues above, in this paper, we develop a novel distributed GNN model for large-scale engineering simulation, namely *D*istributed Vertex-cut Mesh Graph Network (D-MGN), by two following techniques. (*1*) We first propose a vertex-cut graph partitioning scheme to divide a mesh graph into multiple partitions by edge directions. That is, those edges with similar edge directions are grouped together into the same partitions. Thus, the member edges within graph partitions involve similar edge directions, and each edge is within only one partition. Instead, for a certain graph vertex, we replicate its replicas across graph partitions. (*2*) After the partitions are assigned to distributed workers, D-MGN allows local communication-free message passing within each worker, and requires rather low communication cost to perform distributed message aggregation across distributed workers. In summary, we make the following contribution.

- To the best of our knowledge, D-MGN is the first to perform scalable engineering simulation via distributed GNN on mesh graph partitions.

- D-MGN adopts (*i*) local message passing (MP) within graph partitions with no communication cost and further tunes a small number of local MP iteration steps for high efficiency, and (*ii*) distributed MP only one time to aggregate graph embeddings across partitions with trivial communication cost.
- Evaluation on two datasets demonstrates the superiority of D-MGN. For example, on the Beam dataset, compared to the very recent work SAC [5], D-MGN leads to 38.66% smaller RMSE, 54.67% faster convergence time, and 119.22% higher speedup ratio on 8 GPUs. Moreover, during an almost 15-month deployment in an automotive company, engineers have evaluated D-MGN on 529 new steering wheel products with 16.5% shorter product design cycles.

II. Related Work

GNN-based Simulation Model. GNN has emerged as a promising solution to model the topologies and interactions of physical systems. MGN [8] exploits GNN to simulate various physical systems by leveraging mesh graphs to represent the geometric structure of such systems. Yet, flat GNNs in these works do not perform well in learning complex geometric structure due to the limited range of message passing. To address this issue, MS-MGN [4] and other works [3], [9] introduce hierarchical GNNs to extend the range of message passing for better receptive field on coarser mesh elements. However, these approaches perform only on hundreds of mesh elements, leading to low scalability.

Distributed GNN Training. Since GNNs perform message passing between every node and its neighbors, efficient GNN training over large graphs becomes a challenge due to high communication cost across distributed workers [7]. DistDGL [6] exploits METIS [10] for edge-cut graph partitioning to minimize total communication cost, while SAC [5] employs Graph-VB [11] to balance inter-partition communication. G3 [12] eliminates GNN level-wise synchronization by sending the updated embeddings to other workers in a peer-to-peer fashion, instead of waiting for all workers to complete the computation within a level. MGG [13] divides computing tasks into local and remote tasks (where the latter ones require communication), and performs local tasks during the remote task communication phase to reduce waiting time. However, they still suffer from peer-to-peer communication overhead. Instead, Cluster-GCN [14] exploits an edge-cut-based SM-Cluster partitioning approach to cluster similar nodes together. During the training phase, Cluster-GCN focuses on local learning within each partition, and yet completely ignores the message passing of edge-cuts across partitions. DistGNN [15] adopts vertex-cut partitioning and allows using stale embeddings, instead of message passing, during training to avoid waiting for communication to complete. Thus, the two works [14], [15] mitigate distributed communication cost during training, leading to high scalability but at the cost of lower accuracy (caused by the missed cross-partition embeddings).

III. Problem Definition

Mesh Graph. To perform numerical simulation, professional engineers can exploit a mesh generator (e.g., HyperMesh) to discretize the continuous geometric domain $\mathbf{D}$ of the simulation object into surface or volume mesh elements, e.g., triangles or tetrahedra, depending upon simulation requirements. This discretization creates a mesh graph $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{C})$. Here, $\mathbf{V} = \{\mathbf{v}_i\}$ indicates mesh nodes with space coordinates $\mathbf{x}_i$, and $\mathbf{E} = \{\{\mathbf{v}_i, \mathbf{v}_j\}\}$ with $i \neq j$ denotes the edges, and $\mathbf{C} = \{\mathbf{c}_i\}$ is the set of mesh elements surrounded by nodes and edges. Each element $\mathbf{c}_i$ is a subdomain of $\mathbf{D}$ (e.g., triangles in a triangular mesh structure) and $\cup \mathbf{c}_i = \mathbf{D}$.

Initial-boundary conditions. Initial conditions indicate the initial node state $u_0(\mathbf{v}_i, 0)$ at time $t = 0$. Boundary conditions $u(\mathbf{v}_i, t)$ define the behavior on the boundary of the geometric domain $\mathbf{D}$ at a certain time t. In Fig. 1, for the external force F applied to those nodes $\mathbf{V}' \subseteq \mathbf{V}$, we have $u(\mathbf{v}_i, t) = f(t)$ with $\mathbf{v}_i \in \mathbf{V}'$ and $f(t) = F/|\mathbf{V}'|$, assuming that the force F is applied to every node $\mathbf{v}_i \in \mathbf{V}'$.

Given the mesh graph and initial-boundary conditions above, a traditional numerical solver can be used to derive simulation results $u(\mathbf{v}_i, t)$, i.e., the node state $\mathbf{v}_i$ at convergent time step t. Fig. 1 gives an example simulation result $u(\cdot)$, i.e., the stress field on each vertex. The numerical simulation typically requires high running time, ranging from *minutes to even days*, depending upon the number of mesh elements and initial-boundary conditions. By using numerical simulation results as ground truth, we expect that the learned GNN model provides fast prediction time, e.g., *seconds*.

Definition 1 (Mesh Graph Simulation Regression Model): Given a large mesh graph $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{C})$ with initial-boundary conditions u_0 and F, we learn a mesh graph neural network regression model $\mathcal{R}(\cdot)$ with $U = \mathcal{R}(\mathbf{G}, u_0, F)$, where $U = \{u(\mathbf{v}_i, t)\}$ is the ground truth.

To learn the model above, we follow the widely used message passing scheme [4], [8] to update the edge and node state embeddings of the mesh graph in the l-th iteration.

$$\vec{\mathbf{e}}_{ij}^{\,l+1} \leftarrow f_E(\vec{\mathbf{e}}_{ij}^{\,l}, \mathbf{v}_i^l, \mathbf{v}_j^l), \quad \mathbf{v}_i^{l+1} \leftarrow f_V(\mathbf{v}_i^l, \textstyle\sum_j \vec{\mathbf{e}}_{ij}^{\,l+1}) \tag{1}$$

In this equation above, f_E and f_V are Multi-Layer Perceptrons (MLP). For brevity, we denote the message passing update (in short MP) operation in Eq. (1) as follows.

$$(\vec{\mathbf{e}}_{ij}^{\,l+1}, \mathbf{v}_j^{l+1}) \leftarrow \mathrm{MP}(\vec{\mathbf{e}}_{ij}^{\,l}, \mathbf{v}_i^l, \mathbf{v}_j^l) \tag{2}$$

Definition 2 (Vertex-cut Graph Partitioning): A large graph $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$ is divided into K partitions $\{\mathbf{G}_k = (\mathbf{V}_k, \mathbf{E}_k)\}$ with $\mathbf{E} = \cup \mathbf{E}_k$ and $\mathbf{E}_k \cap \mathbf{E}_{k'} = \emptyset$ for any $1 \leq k \neq k' \leq K$, while vertices may be replicated across partitions.

Definition 3 (Distributed GNN Training): Given a large mesh graph $\mathbf{G}$ with K partitions, the distributed GNN model assigns such partitions onto a number P of distributed workers, i.e., GPUs. Each worker processes local partitions (i.e., a subset of graph nodes and edges) in parallel for speedup. The goal is to balance the workloads and minimize inter-worker communication overhead meanwhile with acceptable accuracy of the regression model $\mathcal{R}(\cdot)$.

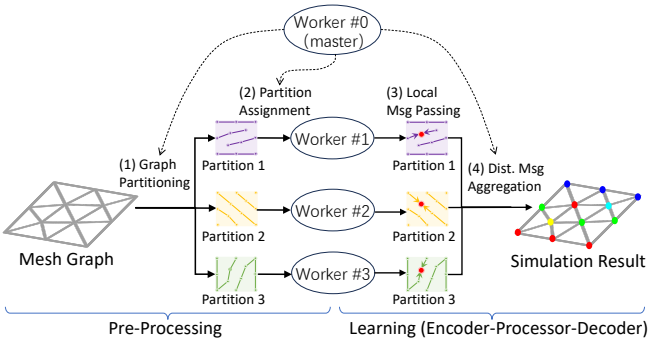

Fig. 2: Workflow of D-MGN

IV. Solution Detail

A. Overview

Fig. 2 shows an example workflow of D-MGN in multiple workers, involving two following stages. In the *pre-processing* stage, when given a large mesh graph and distributed workers, the master node (e.g., # 0) of D-MGN first adopts the proposed vertex-cut policy to generate 3 graph partitions, and then assigns the partitions to the workers (# 1, ... , # 3).

In the *learning* stage, D-MGN logically follows the Encoder-Processor-Decoder framework [4], [8]. An MLP encoder initializes node and edge embeddings, the processor exploits message passing (MP) to update node and edge embeddings, and the decoder finally generates simulation results from the updated embeddings. Physically, in Fig. 2, the MP involves *local MP* within each worker and *distributed aggregation* across workers. Each worker exploits the encoder and local MP within each graph partition to learn local node and edge embeddings in parallel. Given the vertex-cut partitioning in Def. 2, each graph node is with distributed replicas of node embeddings. Thus, via distributed MP, D-MGN collects the three replicas to aggregate the collected embeddings, which are decoded to generate final simulation results. D-MGN requires rather low communication overhead caused only by distributed aggregation across workers, but with no communication cost during the local MP within workers.

D-MGN can comfortably extend the workflow above to support hierarchical mesh graphs using a level-by-level assignment. Starting from the bottom-level graph consisting of a few coarse elements at large size, D-MGN chooses an available worker to accommodate such coarse mesh graphs, until the capacity limit of the chosen worker is met. If the currently chosen worker cannot accommodate an entire mesh graph at a certain level, D-MGN performs the vertex-cut graph partitioning scheme, such that the worker has sufficient capacity to accommodate only a subset of graph partitions. By repeating this level-by-level assignment, D-MGN distributes mesh graphs and partitions to workers. After this assignment, D-MGN again performs local MP and distributed aggregation above to generate final simulation results.

B. Vertex-cut Graph Partitioning

In Fig. 3, D-MGN performs the vertex-cut policy to divide a certain mesh graph into K partitions and next assigns the partitions to distributed workers. For simplicity, the master node of D-MGN exploits the classic k-means algorithm to cluster graph edges by edge directions. That is, by the cosine similarity between edge directions, D-MGN groups graph edges with similar directions into the same clusters (i.e., partitions). During the grouping step, D-MGN computes an *edge direction vector* by $\vec{\mathbf{e}}_{ij} = \mathbf{x}_j - \mathbf{x}_i$, and the *cluster direction vector* by the average of its member edge direction vectors. D-MGN stops the clustering step when cluster membership does not change, and finally generates K partitions. Practically, we set K = 3 or 4 to balance partition size and edge direction diversity. A too small K may not sufficiently differentiate edge directions while a too large K leads to small partitions, compromising global representation. In Fig. 3, we divide the mesh graph into 3 partitions. Each edge appears within only one partition and yet every node is replicated across three partitions (see the vertex-cut graph partition in Def. 2).

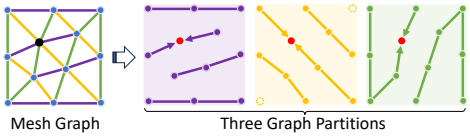

Fig. 3: Vert-Cut Graph Partitioning by Edge Directions

The graph division scheme above offers two following advantages. (*1*) The member edges within each partition diversely appear within the original graph (i.e., the entire geometric domain). This *diverse distribution* can better learn global representation. Instead, for the edge-cut-based partitioning widely used in previous works [5], [6], [12], [14], the member nodes and edges within each partition are densely clustered within a *small area* of the geometric domain, not good for global representation. (*2*) Within each partition, the edges are further connected by graph vertices into disjoint connected components (CCs). The three partitions from left to right consist of 4, 5, and 2 CCs, respectively. We will show that D-MGN exploits these partitions and CCs to more efficiently perform local message passing *in parallel*.

C. Encoder and Local Message Passing

In this section, we give the details of the encoder and local MP to learn local node and edge embeddings. The two steps occur locally within each worker with no communication.

Encoder. The encoder learns initial edge and node embeddings via Multi-Layer Perceptrons (MLP) on initial node states (regarding initial-boundary conditions) in an input large mesh graph $\mathbf{G}$. Note that D-MGN can support hierarchical mesh graphs. Thus, for each coarse graph, D-MGN can generate the associated initial node states by barycentric interpolation [16] on the input mesh graph $\mathbf{G}$. When given the interpolated initial node and edge states on coarse graphs, we exploit MLP to generate the associated initial node and edge embeddings.

Local MP. The processor updates local node and edge embeddings within graph partitions via the proposed *edge-directed* MP. For a given graph partition $\mathbf{G}^k$ with $1 \leq k \leq K$, its member edges share similar edge directions. D-MGN

performs the edge-directed MP on this partition to update the local node and edge embeddings.

$$(\vec{\mathbf{e}}_{ij}^{k,l}, \mathbf{v}_i^{k,l}) \leftarrow \mathrm{MP}(\vec{\mathbf{e}}_{ij}^{k,l-1}; \mathbf{v}_i^{k,l-1}) \tag{3}$$

In the equation above, a key parameter is the number of L^k MP iteration steps with $1 \leq l \leq L^k$. We tune the parameter L^k as follows. Recall that in Fig. 3, each graph partition involves multiple connected components (CCs). By computing the diameter of each CC within a partition, we choose the largest one as the number L^k.

Choosing the above largest CC diameter makes sense, offering two unique advantages. (*i*) For the local edge-directed MP, D-MGN in parallel performs local MP on the granularity of much smaller CCs. Compared to the original MP on the entire mesh graph or a partition, the local MP on the CCs leads to much *smaller MP overhead*. (*ii*) Meanwhile, due to the diverse distribution of graph edges across the entire geometric domain, the edge-directed MP ensures *sufficient coverage* of the entire mesh graph for better global view.

D. Distributed Aggregation and Decoder

After the local node and edge embeddings are learned above, the master node in D-MGN initiates the distributed aggregation of local embeddings across the workers. Each worker then aggregates node embeddings from others and finally decodes the aggregated embeddings to generate simulation results. Unlike the local MP above requiring the iterations of L^k MP steps, D-MGN performs the distributed aggregation only one time, leading to trivial communication cost.

Distributed Aggregation. After updating node embeddings by the local edge-directed MP above, each worker aggregates the needed replicas of node embeddings from other workers.

$$\mathbf{v}_i \leftarrow g_V([\mathbf{v}_i^{1,L^k}, \ldots, \mathbf{v}_i^{K,L^k}]) \tag{4}$$

In the equation above, D-MGN concatenates the K replicas of node embeddings $\mathbf{v}_i^k$, which are then aggregated by MLP g_V to generate the final node embeddings.

Decoder. After the MP is performed on the fine mesh graph G, the decoder exploits MLP to convert the aggregated node embeddings back to the needed simulation results (e.g., stress field) of every node on the mesh graph $\mathbf{G}$.

Algorithm 1: Distributed Workflow of D-MGN

Input: Graph Partitions $\mathbf{G}^1, \ldots, \mathbf{G}^K$
Output: Simulation Result $\hat{u}(\mathbf{v}_i)$ with $\mathbf{v}_i \in \mathbf{G}$

1 Assign graph partitions to available workers;
2 **for** $1 \leq k \leq K$ **do** // the k-th partition
3 Encode an initial node embedding $\mathbf{v}_i^{k,0}$;
4 Encode an initial edge embedding $\vec{\mathbf{e}}_{ij}^{k,0}$;
5 **for** $1 \leq l \leq L^k$ **do** // the l-th step
6 Update $\mathbf{v}_i^{k,l}$ and $\vec{\mathbf{e}}_{ij}^{k,l}$ by local MP; // Eq. (3)
7 **end**
8 **end**
9 Update $\mathbf{v}_i$ by dist. aggregation; // Eq. (4)
10 $\hat{u}(\mathbf{v}_i) \leftarrow$ decode($\mathbf{v}_i$) for every node $\mathbf{v}_i \in \mathbf{G}$;

E. Algorithm Detail

Finally, Alg. 1 gives the main steps of distributed GNN learning stage, by taking the K graph partitions as input and generating the simulation results of every node in the input mesh graph $\mathbf{G}$ as output. This algorithm mainly involves two parts: the lines 1-8 within each partition (a.k.a worker) to perform the initial encoding and local MP, and the lines 9-10 by the master node to perform distributed aggregation and final decoding to generate simulation results.

Here, the encoder, local MP, distributed aggregation, and decoder above are all implemented by MLPs. Each MLP consists of two fully connected layers of hidden embedding size of 128 with ReLU activations and layer normalization. By the input mesh graph $\mathbf{G}$ and ground truth of simulation results, we use the loss function $\mathcal{L} = \frac{1}{n}\left(\hat{u}(\mathbf{v}_i) - u(\mathbf{v}_i)\right)^2$ to learn the network parameters of MLPs by the Adam optimizer.

V. Evaluation

A. Experimental Setup

1) Datasets: We use two datasets: one synthetic (Beam) and one real (SteeringWheel). For each dataset, we use 80% of samples for training, 10% for validation, and 10% for testing.

TABLE I: Statistics of Two Datasets

Dataset	Beam	SteeringWheel
Total # of Graph Samples	555	239
Avg. Nodes per Graph Sample	213691	72061
Avg. Edges per Graph Sample	638339	200526
Max./Min./Avg. Degree per Node	8/3/5.97	39/3/5.55
Numerical Solver	ABAQUS	LS-DYNA
Numerical Sim. Time	8.5min	20min

- **Beam**. We generate the 2D Beam dataset by a numerical solver ABAQUS [17] to simulate deformation responses of rectangular beams subjected to an external force. With the size 15×100 mm^2, each Beam sample contains a circular hole with a 5 mm diameter. We vary the hole's center from the initial center $\langle 5, 5\rangle$ mm by a 2.5mm step size over 3 horizontal and 37 vertical steps, respectively, and generate 111 samples. By fixing the bottom plane of the Beam structure, we apply a 300N (Newton) force with angles varying from -60° to 60° by a 30° step size, generating 5 loading settings. After dividing each Beam into triangular mesh elements, we exploit ABAQUS to have the numerical stress field as ground truth.
- **SteeringWheel**. The real dataset includes 239 samples of 3D steering wheels, provided by an automotive supplier. Following an industry trial standard, engineers apply a 700N force along the negative z-axis at the wheel rim, with the steering column fixed at the bottom plane. Each sample is divided into hybrid mesh elements consisting of hexahedrons, pentahedrons and tetrahedrons. Engineers employ the industry-level numerical solver LS-DYNA [18] to generate the stress field as ground truth.

2) Baselines: To the best of our knowledge, previous works focus on distributed GNN models on flat graphs. Thus, we first compare D-MGN with four distributed GNN models on

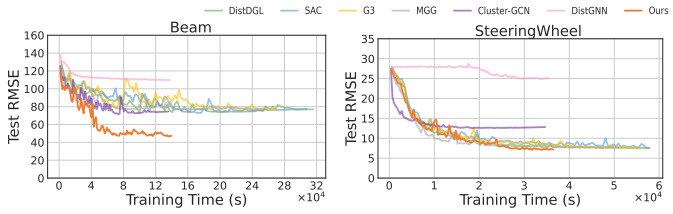

Fig. 4: Time-to-accuracy Study.

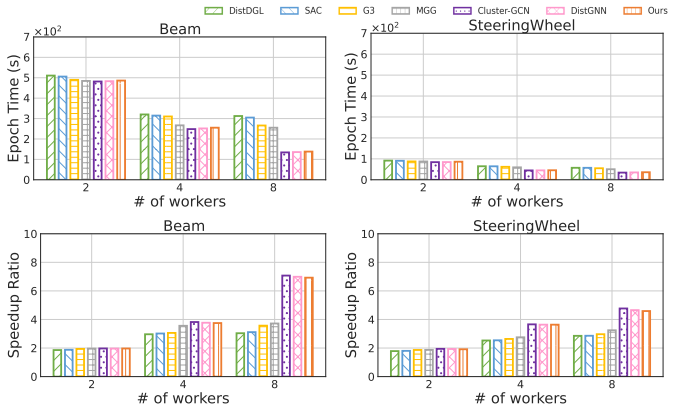

Fig. 5: Scalability Study (Top: Epoch Time; Bottom: Speedup).

flat mesh graphs for fairness, and finally evaluate D-MGN on hierarchical mesh graphs.

DistDGL [6] uses METIS for edge-cut graph partitioning and node replication to reduce communication; SAC [5] optimizes communication cost by only sharing necessary data and balancing partitioning via Graph-VB; G3 [12] speeds up GNN training by sharing embeddings peer-to-peer and overlapping computation and communication; MGG [13] reduces waiting time by doing local tasks during communication; Cluster-GCN [14] exploits the so-called SM-Cluster graph partitioning approach to achieve linear scalability by removing feature communication during training; DistGNN [15] adopts vertex-cut partitioning via Libra and achieves high scalability by using stale embeddings during training.

3) Evaluation Metric: We measure the *prediction error* by $\text{RMSE} = \left(\frac{1}{N}\sum_{i=1}^{N}\frac{1}{n_i}\sum_{j=1}^{n_i}(\hat{u}_{ij}-u_{ij})^2\right)^{\frac{1}{2}}$, where n_i is the number of nodes in i-th graph sample, and u_{ij} and $\hat{u}_{ij}$ are the ground truth and predicted value of the j-th node in the i-th graph sample, respectively. In addition, we compute the *epoch time* by the total training time to process the entire training dataset, and *speedup ratio* by the ratio of the epoch time on multiple GPUs against the one on a single GPU.

4) Environment: We implement D-MGN and baselines by PyTorch v2.1.0 and CUDA v12.1, and evaluate experiments on 4 servers, each with two 16-core Xeon(R) Gold 6430 CPUs and 2 NVIDIA GeForce RTX 4090 GPUs, thus total 8 GPUs.

B. Performance Study

1) Time-to-accuracy Study: Fig. 4 first gives the time-to-accuracy study on 8 GPUs in 4 servers. On both datasets, D-MGN consistently performs best with the smallest RMSE and fastest convergence time among all seven distributed GNN models. For example, on the Beam dataset, D-MGN leads to 38.66% smaller RMSE and 54.67% faster convergence time than the most recent work SAC [5]. Compared to Cluster-GCN and DistGNN, D-MGN involves almost equally convergence time but with much lower RMSE, mainly because Cluster-GCN and DistGNN miss the MP across partitions.

2) Scalability Study: Fig. 5 (top) gives the epoch (training) time of seven models on 2 - 8 GPUs. (*i*) When the number of GPUs grows from 2 to 8, the epoch time of DistDGL, SAC, G3 and MGG, all first decrease and then converges to a stable value. It is mainly because the increased communication overhead across GPUs may compromise the benefit of parallel MP. (*ii*) Instead, consistent with Fig. 4, Cluster-GCN, DistGNN and D-MGN lead to smaller epoch time. It is mainly because they both in parallel perform local MP within partitions without communication cost.

Fig. 5 (bottom) plots the speedup ratios of seven models. For example on 8 GPUs, DistDGL, SAC, G3 and MGG, are all with the speedup ratios $\approx 3\times$ for both datasets. In contrast, Cluster-GCN, DistGNN and D-MGN achieve rather high speedup ratios $\approx 7\times$. Here, although suffering from a slightly lower speedup ratio, D-MGN outperforms Cluster-GCN and DistGNN by much smaller RMSE. As shown in Fig. 4, compared to Cluster-GCN, D-MGN leads to 36.30% and 44.23% smaller RMSE on two datasets. In summary, D-MGN outperforms Cluster-GCN and DistGNN with much lower RMSE, and comparable epoch time and speedup ratios.

We give the behind rationale of the evaluation result above as follows. DistDGL, SAC, G3 and MGG all require distributed communication across workers at every MP iteration step. When on 8 workers, their communication cost becomes much significant. In contrast, both Cluster-GCN, DistGNN and D-MGN do not require communication during local MP within each partition. D-MGN further requires distributed aggregation and leads to slightly smaller speedup ratios. However, the higher speedup ratios of Cluster-GCN and DistGNN are caused by the missed message passing across partitions, at the cost of rather higher RMSE.

In terms of *prediction time*, on the two datasets, D-MGN only takes on average 149.51 ms and 50.92 ms per sample on 8 GPUs, significantly faster than the FEM solving time of 8.5 min and 20 min, respectively (see Table I). Note that due to the more graph nodes and edges, the Beam dataset leads to higher prediction time than the SteeringWheel dataset.

3) Load Balancing: As shown in Fig. 6, D-MGN illustrates roughly equal numbers of graph nodes and edges within each of the 8 partitions (workers), leading to the nearly same epoch time across the 8 GPU workers.

TABLE II: Evaluation of 3 Hierarchical GNNs on 8 GPUs

	Beam			SteeringWheel		
Level	RMSE	Epoc	Speedup	RMSE	Epoc	Speedup
1	46.78	138.20	6.93	7.15	36.06	4.59
2	11.11	133.30	6.88	6.69	31.91	4.45
3	6.43	126.51	6.85	6.35	26.86	4.35

When the number K of partitions varies from 1 to 8, D-MGN essentially involves different neural network structures,

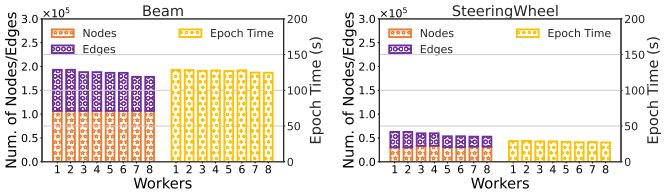

Fig. 6: Workload Statistics on 8 GPUs.

learns significantly various MLP network parameters, and thus leads to different RMSE. Our further evaluation reveals that $K = 4$ partitions lead to the least RMSE. Either a too small or too large number K results in higher RMSE.

4) Study of Hierarchical Mesh Graphs: Finally, in Table II, we first generate three-level mesh graphs (say G_1, G_2 and G_3 with the smallest, middle, and greatest element size) by the popular Delaunay triangulation [19], and next evaluate the performance on three hierarchical GNN models using the only one-level mesh graph $\mathbf{G}^1$, two-level graphs $\mathbf{G}^1, \mathbf{G}^2$ and three-level graphs ($\mathbf{G}^1, \mathbf{G}^2$ and $\mathbf{G}^3$). Here, 3-level mesh graphs are sufficient, because coarse mesh graphs with too large element size suffer from the element fragmentation issue [20].

In this table, when the number of total levels grows from 1 to 3, the RMSE and epoch time become smaller and yet the speedup ratio remains roughly consistent with slight decreases. That is, multilevel mesh graphs effectively represent global and local views together with smaller RMSE and faster training convergence time. Meanwhile, multilevel mesh graphs lead to more communication cost across workers, and the speedup ratio becomes smaller. However, since the computational overhead of MP mainly focus on fine mesh graphs, multilevel mesh graphs (i.e., more coarse graphs) may not lead to significant decrease of the speedup ratio.

Deployment: We have deployed D-MGN in the production environment of an automotive company for almost 15 months. After D-MGN is trained by 239 real steering wheel history data, engineers have evaluated the stress field of 529 new steering wheel products. Compared to traditional FEM solvers, D-MGN provides fast simulation prediction results, and engineers then quickly re-optimize product design with the 16.5% shorter product design cycles from prototype design, simulation to trial of car steering wheels.

VI. Conclusion and On-going Work

In this paper, we propose a novel distributed mesh graph network D-MGN for efficient GNN training in distributed workers. By the proposed vertex-cut partitioning policy, D-MGN assigns each edge uniquely into only one partition (a.k.a worker), meanwhile with distributed replicas of a certain node across partitions. Thus, D-MGN allows local message passing within each worker with no communication, and requires rather low communication cost to perform distributed message aggregation across distributed workers. Our preliminary evaluation on two datasets demonstrates the superiority of D-MGN. As on-going works, we plan to develop scalable GNN models used for large-scale numerical simulation with tens of millions of mesh elements on hundreds of distributed workers.

Acknowledge: This work is partially supported by National Key R&D Program of China (2022YFE0208000, 2023YFF0725100), Shanghai Key Lab of Vehicle Aerodynamics and Vehicle Thermal Management Systems, National Science Foundation of China (NSFC) under Grant No. U22B2060, Guangdong-Hong Kong Technology Innovation Joint Funding Scheme Project No. 2024A0505040012, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19,AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, Guangdong Province Science and Technology Plan Project 2023A0505030011, Guangzhou municipality big data intelligence key lab, 2023A03J0012, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Zhujiang scholar program 2021JC02X170, Microsoft Research Asia Collaborative Research Grant, HKUST-Webank joint research lab and 2023 HKUST Shenzhen-Hong Kong Collaborative Innovation Institute Green Sustainability Special Fund, from Shui On Xintiandi and the InnoSpace GBA.

References

[1] F. Lin, J. Shi, Z. Gao, Z. Chu, Q. Ma, H. Yu, and W. Rao, "Physical system simulation based on deep representation learning for 3d geometric features," *Journal of Computer Applications (Chinese Journal)*, pp. 1–11, 2023.

[2] J. Shi, F. Lin, and W. Rao, "Learning to simulate complex physical systems: A case study," in *CIKM 2023*, pp. 4284–4288, 2023.

[3] M. Lino, S. Fotiadis, A. A. Bharath, and C. D. Cantwell, "Towards fast simulation of environmental fluid mechanics with multi-scale graph neural networks," in *ICLR Workshop on AI4ESS*, 2022.

[4] M. Fortunato, T. Pfaff, P. Wirnsberger, A. Pritzel, and P. W. Battaglia, "MultiScale MeshGraphNets," *CoRR*, vol. abs/2210.00612, 2022.

[5] U. Mukhopadhyay, A. Tripathy, O. Selvitopi, K. A. Yelick, and A. Buluç, "Sparsity-aware communication for distributed graph neural network training," in *ICPP 2024*, pp. 117–126, 2024.

[6] D. Zheng, X. Song, C. Yang, D. LaSalle, and G. Karypis, "Distributed hybrid CPU and GPU training for graph neural networks on billion-scale heterogeneous graphs," in *KDD '22*, pp. 4582–4591, 2022.

[7] M. Besta and T. Hoefler, "Parallel and distributed graph neural networks: An in-depth concurrency analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 5, pp. 2584–2606, 2024.

[8] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, "Learning mesh-based simulation with graph networks," in *ICLR 2021*, 2021.

[9] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, A. M. Stuart, K. Bhattacharya, and A. Anandkumar, "Multipole graph neural operator for parametric partial differential equations," in *NeurIPS 2020*, 2020.

[10] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

[11] S. Acer, R. O. Selvitopi, and C. Aykanat, "Improving performance of sparse matrix dense matrix multiplication on large-scale parallel systems," *Parallel Comput.*, vol. 59, pp. 71–96, 2016.

[12] X. Wan, K. Xu, X. Liao, Y. Jin, K. Chen, and X. Jin, "Scalable and efficient full-graph GNN training for large graphs," *Proc. ACM Manag. Data*, vol. 1, no. 2, pp. 143:1–143:23, 2023.

[13] Y. Wang, B. Feng, Z. Wang, T. Geng, K. J. Barker, A. Li, and Y. Ding, "MGG: accelerating graph neural networks with fine-grained intra-kernel communication-computation pipelining on multi-GPU platforms," in *OSDI 2023*, pp. 779–795, 2023.

[14] W. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C. Hsieh, "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks," in *KDD 2019*, pp. 257–266, 2019.

[15] V. Md, S. Misra, G. Ma, R. Mohanty, E. Georganas, A. Heinecke, D. Kalamkar, N. K. Ahmed, and S. Avancha, "DistGNN: scalable distributed training for large-scale graph neural networks," in *SC '21*, 2021.

[16] J. Berrut and L. N. Trefethen, "Barycentric Lagrange Interpolation," *SIAM Rev.*, vol. 46, no. 3, pp. 501–517, 2004.

[17] Dassault Systèmes, "Abaqus finite element analysis." https://www.3ds.com/products/simulia/abaqus, 2024.

[18] ANSYS, Inc., "LS-DYNA." https://lsdyna.ansys.com/, 2024.

[19] Y. S. Elshakhs, K. M. Deliparaschos, T. Charalambous, G. Oliva, and A. C. Zolotas, "A comprehensive survey on Delaunay triangulation: Applications, algorithms, and implementations over CPUs, GPUs, and FPGAs," *IEEE Access*, vol. 12, pp. 12562–12585, 2024.

[20] B. Wördenweber, "Finite element mesh generation," *Comput. Aided Des.*, vol. 16, p. 285–291, Sept. 1984.