SaGNN: a Sample-based GNN Training and Inference Hardware Accelerator

Haoyang Wang, Shengbing Zhang, Kaijie Feng, Miao Wang, Zhao Yang
Northwestern Polytechnical University, Xi’an, China
haoyangwang@mail.nwpu.edu.cn; yz70528@mail.nwpu.edu.cn

Abstract—Graph neural networks (GNNs) operations contain a large number of irregular data operations and sparse matrix multiplications, resulting in the under-utilization of computing resources. The problem becomes even more complex and challenging when it comes to large graph training. Scaling GNN training is an effective solution. However, the current GNN operation accelerators do not support the mini-batch structure. We analyze the GNN operational characteristics from multiple aspects and take both the acceleration requirements in the GNN training and inference process into account, and then propose the SaGNN system structure. SaGNN offers multiple working modes to provide acceleration solutions for different GNN frameworks while ensuring system configurability and scalability. Compared to related works, SaGNN brings 5.0x improvement in system performance.

Index Terms—Graph Neural Networks, Sample-based GNN training, Hardware accelerator.

I. INTRODUCTION

Graph Neural Networks (GNNs) is a powerful extension of deep neural networks (DNNs) that have been widely used in recent years [13]. GNN technology is applied at various levels of abstraction of the graph, including Link Prediction, Vertex Classification, and Graph Clustering [1]–[3], [9], [17], [18]. The properties of GNNs, in comparison to typical DNNs, bring new challenges for computing systems.

The input of a GNN is not a set of independent data objects but instead graphs with interconnected and interdependent vertices. The graph’s connectivity relationship is typically represented by a sparse adjacent matrix [16]. GNN’s layers are described as message-passing processes. Each vertex collects the features of its neighbors [4], [5], and then matrix multiplication (MM) is used to update vertex data. GCN [6] is a classic computing model that include two-layer operations, each of which contains aggregation and combination. Training GNNs presents distinct challenges according to the irregularity of graph data, since irregular operations and sparse matrix multiplications cause the under utilization of computing resources.

The problem becomes even more serious when it comes to large graph training. A large amount of node information and link relationships further complicate computing processes, generating higher demands to the scalability of the computing system. A proposed solution is sample-based GNN training [3], [19], [20]. Instead of putting the entire graph into the computing system, the sample-based training method divides the workload into several node-wise mini-batches. Mini-batch training is efficient and more scalable than whole-graph training. However, the mini-batch model is not supported by the present GNN processing accelerators [10]–[12].

The contribution of our work is as follows: we first evaluate GNN operating characteristics over several dimensions while keeping acceleration requirements in GNN training and inference in consideration, then we propose SaGNN, a sample-based system architecture to address GNN’s challenges and it brings significant performance improvement. SaGNN supports a variety of working modes, that can provide acceleration solutions for diverse GNN architectures while ensuring system configurability and scalability.

II. ALGORITHM AND CHALLENGES

We first introduce GNN algorithm in this section. Following the analysis of GNN characteristics, we then elaborate on the proposed solution scheme for scale-based GNN acceleration.

A. Graph Neural Networks

Fig. 1 shows the typical GNN algorithm [4]–[6]. A GNN model contains multiple graph convolutional layers. The vectors on each node perform two types of calculations: aggregation and combination. During a layer of operation loop, the node collects information from its neighbor nodes through aggregation operation and updates the vector on the node through combination operation. The aggregation and combination operations of the k-th layer for node v can be described as:

\[ a_v^k = \text{Aggregate}(h_{u,v}^{k-1} | u \in \text{Neighbor}(v)), \]
\[ h_v^k = \text{Combine}(a_v^k, W_e^k). \]  (1)

in which \( h_v^k \) is the hidden feature vector of v at the k-th layer, \( a_v^k \) represents the aggregation feature vector, \( W_e^k \) denotes the shared weight parameters for combination of layer k, and Neighbor(v) includes all the neighbors of the central node.

Edge update is another common GNN function; formula 2 illustrates the operation of edge update.

\[ e_{u,v} = \sigma(W_e^k : h_{u,v}^{k-1}). \]  (2)

where \( e_{u,v} \) is the edge update result vector for node pair u and v, \( h_{u,v}^{k-1} \) represents the hidden feature vector of u at the (k-1)-th layer, and \( W_e^k \) is the shared weight parameters for edge update of layer k.
Many GNN models have edge update processing requirements, while some GNN models do not have edge update operations. Directly providing hardware computing resources for the edge update operation is a natural solution, but when the system accelerates the model that does not require edge update, this module will remain idle, resulting in a waste of computing resources.

Table I shows common GNN datasets, indicating that even though the number of GNN data edges is larger than the number of nodes, the order of magnitude of the edges cannot reach the square of the order of the node. As a result, there is a large amount of data in the adjacent matrix that is zero.

### B. Challenges

**a) Irregular Computations:** Full-graph training, which takes all of each node’s neighbors into consideration, is a natural GNN training idea, but this method has numerous obstacles. First, the large graph GNN training procedure contains a large number of operations, and full-graph training presents significant challenges to storage bandwidth and computing capacity. Second, the data relevance of full-graph training harms the system’s performance. Moreover, the scalability of full-image training is inadequate; as GNN develops, the size of the graph increases, and full-image training cannot fulfill the requirements of large-scale distributed GNN. Sampling can effectively alleviate the problem of irregular computation and sparse characteristics created by GNN, but it requires the hardware architecture to support the mini-batch computing method.

**b) Configurability and Flexibility:** Different GNN models have various operation patterns, which sets a higher requirement for hardware system configurability and flexibility. Existing work still has the drawback of being unable to support certain operations in GNN [10]–[12], for example, edge update. Furthermore, allocating particular computational components to every operation mode can easily result in non-reusable resource overhead. It is essential to consider not only supporting various GNN models, but also system performance issues while evaluating hardware acceleration. GNNs have complicated and diverse execution process requirements, which demand hardware structures that provide configurable execution processes for multiple operation modes.

### C. Proposed Solution

We propose a node-wise, sampling-based GNN inference and training acceleration system that has the following two advantages:

In the process of GNN inference, node-wise aggregation methods is used to make use of graph characteristic rather than directly adopting the idea of full-graph operation. Compared with the method of computing with sparse full-graph connection matrices, the node-wise approach makes the aggregation operation dense and regular, thereby reducing resource overhead.

The node-wise, sampling-based GNN training method can improve GNN training performance from two aspects. First, the amount of computation in the training process is reduced, and the overall speed of system training is improved. Second, this approach uses the mini-batch idea, where each operation takes only a subset of the neighbor nodes connected to the core node. The sampling and node-wise training methods divide a training task of a large graph into many small tasks, which can be processed in parallel. Therefore, the structure is much flexible and highly scalable.

### III. PROPOSED ARCHITECTURE FOR GNN ACCELERATION

In this section, we present the hardware architecture design for GNN acceleration: SaGNN. First, we introduce the SaGNN’s overall architecture and describe the fundamental connections between the components. Following that, we describe the design details of the main functional modules
and how they support the implementation of the corresponding GNN function. At the end of this section, we explain how the system can fulfill different GNN tasks by controlling dataflow to support various types of operation modes.

A. Overview of System Design

Fig. 2 shows the framework for SaGNN. We employ the global interface to connect high-bandwidth memory (HBM)’s resources so that the system can enhance bandwidth consumption by prefetching and organizing data. Graph connection information is stored in Structure Buffer. The Sample Engine deconstructs the full-graph sparse connection into the mini-batches, and sends the afterward task to the processing units, as known as Aggregation Processing Engine (Aggregation PE) and Matrix Multiplication Processing Engines (MMPEs), by accessing the data of the Vertex Buffer, which stores the node information. Weight Buffer is used to providing weight data for MM-based operations of GNN including edge update and combination. The edge update and combination tasks are performed in MMPE. The data that completes a layer of GNN operations is sent back to the Output Buffer.

We made two identical MMPEs to fulfill the computational requirements of various GNN models. We use two MMPEs units respectively to execute edge update and node combination for the GNN models with the edge update operation; for GNN models that do not have edge update operations, we provide configurable data paths so that both MMPEs can be used for combination operations. The Global Controller generates control signals for configuring data path selection. Through the application of this system architecture design, SaGNN can not only satisfy the processing demands of various GNN models, but also ensure that the execution units do not encounter a large number of idle phenomena when executing certain GNN models.

B. Hardware Module Design

Fig. 3 shows the Sample Engine. The input data is the full-graph adjacent matrix, from which we gather information of the connection relationships between the central node and other nodes. The neighbor node IDs and addresses are generated based on the central node ID and connection relationship. In this way, the full graph’s sparse connection information is converted into the mini-batch dense data structure and the graph structure’s irregular operations are converted into regular operations. Also, it removes data from neighbor nodes that do not participate in aggregation. As an example, consider random sampling [7], Neighbor Sampling module removes some edges between central node and its neighbors proportionally through group sampling. When the GNN model execution process does not include a sampling operation, the sample step can be bypassed, and the mini-batch method based on the central node can continue to operate.

Fig. 4 shows the Aggregation PE’s structure. The mini-batch structure neighbor node information is sent to the Aggregate Buffer, and the intermediate result is used as the input to continue participating in the operation. We use the redundancy eliminate method [15] by storing reusable intermediate results of aggregation to further improve system performance. Aggregation tasks for the same central node are assigned to the same aggregation engine unit, and aggregation workload from various central nodes is processed in parallel by different units.

C. Process of Execution

Fig. 5 illustrates how SaGNN supports various GNN computation pipelines. Fig. 5(a) and Fig. 5(b) represent workflows that require and do not require edge updates, respectively. The nodes removed during the sample operation don’t have to participate in the edge update operation, ensuring that no unused intermediate results are generated. In Fig. 5(a), Sample Engine divides the entire graph into mini-batch-based tasks, then performs edge update, aggregation, and combination on neighbor node data in turn.
TABLE II
SYSTEM CONFIGURATIONS OF COMPARED ACCELERATORS.

<table>
<thead>
<tr>
<th></th>
<th>HyGCN</th>
<th>AWG-GCN</th>
<th>HaGCN</th>
<th>SaGNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute</td>
<td>1GHz @ 32 SIMD</td>
<td>1GHz @ 4K PEs</td>
<td>1GHz @ 512 32-bit ALUs</td>
<td>1GHz @ 512 Aggregation PEs</td>
</tr>
<tr>
<td></td>
<td>16 cores and 32x128 arrays</td>
<td>32x128 PEs</td>
<td>32x128 PEs</td>
<td>32x128 PEs</td>
</tr>
<tr>
<td>On Chip Memory</td>
<td>22MB and 128KB</td>
<td>12MB</td>
<td>24MB</td>
<td>22MB</td>
</tr>
<tr>
<td>Off Chip Memory</td>
<td>256GB/s HBM 1.0</td>
<td>256GB/s HBM 1.0</td>
<td>256GB/s HBM 1.0</td>
<td>256GB/s HBM 1.0</td>
</tr>
</tbody>
</table>

Fig. 5. Process of Execution: (a) GNN Framework with Edge Update, (b) GNN Framework without Edge Update.

Fig. 5(b) depicts the operations performed during the GNN model execution process without edge updates: ❶ After sampling neighbor data based on the central node is directly sent to Aggregation PE, after which the combination task is distributed to two MMPEs units for operations. In this configuration, the two MMPEs computing components are all used, and no computing resources are idled.

IV. EXPERIMENTAL RESULTS AND EVALUATION

A. Experimental Setup

We implement SaGNN in Verilog and use the Synopsys Design Compiler (DC) with the TSMC 12 nm standard VT library for synthesis, and we estimate HBM timings using Ramulator [14]. GCN [6], GraphSage-Pool [7], and G-GCN [8] are three representative GNN models that we use to evaluate SaGNN, where GraphSage-Pool and G-GCN contain edge-update operations. Table I shows the datasets we use.

We compare ReGNN to the PyTorch Geometric [4] implemented on CPU and GPU, as well as the GNN accelerators HyGCN [10], AWB-GCN [11], and HaGNN [12]. The CPU platform includes two Intel Xeon(R) CPU E5-2680 v3 CPUs and 500GB DRAM, while the GPU platform includes an NVIDIA Tesla V100 GPU. Table II shows the system configurations of baseline accelerators.

B. Evaluation

We evaluate SaGNN with three GNN frameworks by comparing with PyG-CPU, PyG-GPU, HyGCN, AWB-GCN, and HaGNN. The GS-Pool and G-GCN models include edge update operations that HyGCN and AWB-GCN cannot support. Fig. 6 depicts SaGNN’s performance compared to those other baselines. All performance is normalized to PyG-CPU. On average, ReGNN is 6.83×, 2.32×, and 5.85× times faster than HyGCN, AWB-GCN, and HaGNN for different accelerator baselines. Fig. 7 depicts energy consumption. With the use of node-wise processing method, ReGNN saves 9.97×, 2.70×, and 8.82× energy compared to HyGCN, AWB-GCN, and HaGNN, respectively. Overall, SaGNN supports multiple working modes to provide acceleration solutions for different GNN frameworks while ensuring system performance.

V. CONCLUSION

We propose a node-wise, sampling-based acceleration method for GNN inference and training, then we design the hardware accelerating system SaGNN to support it. Compared to the method based on the sparse full-graph adjacent matrix, the node-wise method possesses dense operating characteristic, and it is scalable since the operations in the mini-batch data structure only use a portion of the nodes connected to the central node. SaGNN offers various of working modes to support different GNN structures and achieves 5.0X performance improvement over baseline on average.
REFERENCES


