HBM-based Hardware Accelerator for GNN Sampling and Aggregation

Yuchen Gui[1], Qizhe Wu[1], Wei Yuan[1], Huawen Liang[1], Xiaotian Wang[1], Xi Jin[1]
[1]*University of Science and Technology of China* Hefei, China

***Abstract*—The GNN computation process described by the message passing network mainly consists of two processes: combination and aggregation. Among them, the combination process (matrix multiplication) is consistent with the matrix multiplication computation process in traditional DNNs and does not have special characteristics of GNNs. The aggregation process, on the other hand, is unique to GNN and imposes a large random access pressure on memory. Most of the currently available accelerator solutions focus on the whole computational process of GNN or sample-aggregation on GPU/CPU, while there are fewer FPGA/ASIC-based accelerators dedicated to the sample-aggregation process. The hardware acceleration scheme proposed in this paper focuses on two aspects: First, an innovative streaming sampler is proposed to accelerate sampling from the power-law distribution characteristics of degree in GNN datasets. Second, the feature acquisition is accelerated by the high bandwidth advantage of HBM, and an innovative aggregation scheme is realized to match the readout data from HBM. The proposed hardware streaming sampler scheme is tested by FPGA and achieves a speedup of $2\times$ to $20\times$ relative to traditional FPGA-based node sampling, and the feature acquisition and aggregation accelerator achieves up to $300\times$ speedup relative to the GPU platform.**

***Index Terms*—GNN, sampling, aggregation, HBM, accelerator, FPGA**

I. Introduction

Graph neural networks (GNNs) [1]–[3] have achieved widespread application, including scenarios such as networks formed by social groups, protein relationship networks, paper citation networks, and relationships between natural language words [4], [5].

Message passing network is a generalized model for describing GNNs. A variety of existing GNN frameworks such as GCN [6], GraphSAGE [7], GAT [8], etc. can be described by message passing networks. Sampling and aggregation in this work is performed based on the message passing network.

In message passing network for GNN node classification task, each central node needs to aggregate the feature information of the nodes connected to it (neighbor nodes). For each central node to be classified, its neighbor nodes are randomly distributed over a large storage space, which brings about a large number of random accesses to the memory, making the whole system speed limited by the memory bandwidth and slowing down the system. Therefore this process needs to be optimized. In addition, the number of neighbor nodes (degree) of different central nodes of most GNN datasets varies greatly. Some of the central nodes have a large number of neighbor nodes, which will result in large memory access and time overhead, so it is necessary to select a part of a specific amount of neighbor nodes to participate in the aggregation (sampling), so as to reduce the computation and access overhead, making the data regularized. [9], [10].

Current accelerators (e.g., HyGCN [11], AWB-GCN [12], SmartSAGE [13], and GNNLab [14]) focus on optimizing the overall GNN arithmetic process or performing sampling using the CPU/GPU. There are fewer dedicated FPGA/ASIC-based sampling-aggregation accelerators optimized for random memory access.

In response to the above issues, the present work makes the following innovative efforts and contributions.

1) According to the characteristics of a typical GNN dataset, the vast majority of nodes have small degrees, while a small fraction of nodes have large degrees (power law distribution). Therefore, this work innovatively proposes a streaming sampler that shifts node sampling from the spatial domain to the temporal domain corresponding to the process of reading neighbor nodes, so that the sampling time is proportional to the number of neighbor nodes, thus achieving the acceleration of the node sampling process.
2) The acceleration of feature acquisition is realized by using the high bandwidth advantage of HBM. Meanwhile, the min-heap pipeline aggregation accelerator is innovatively proposed to solve the problems of response speed difference (time imbalance) and random distribution of the number of sample nodes (number imbalance) of different read channels of HBM to realize the aggregation task of feature vectors.

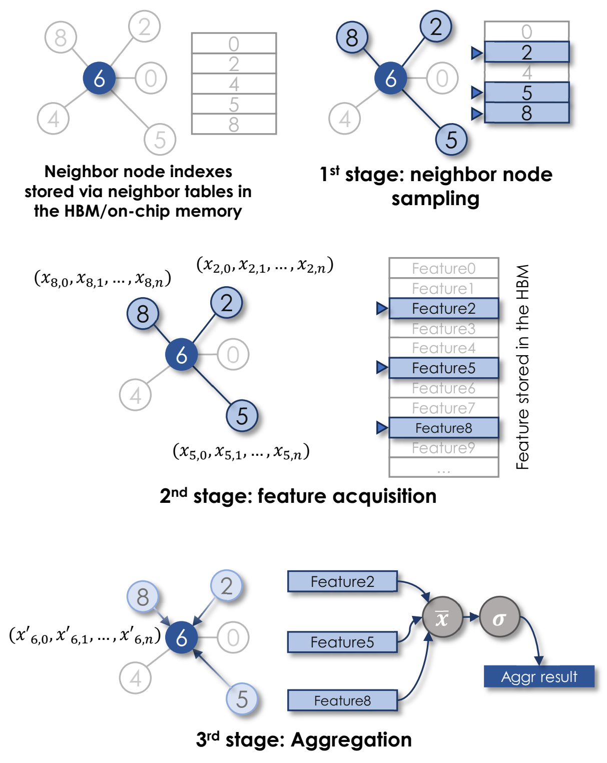
II. Background

A. Unified Representation of GNN - Message Passing Network

According to the existing research results [15]–[18], multiple GNN frameworks (e.g., GCN [6], GraphSAGE [7]) can be uniformly described as message passing networks. The message passing network can be described by Eq. 1 as follows.

$$\boldsymbol{h}_i' = f\left(\boldsymbol{h}_i, \bigcup_{j\in N(i)} g\left(\boldsymbol{h}_i, \boldsymbol{h}_j, \boldsymbol{e}_{j,i}\right)\right) \tag{1}$$

where $\boldsymbol{h}_i$ represents the feature vector of node i and $N(i)$ represents the set of neighbor nodes of node i. $\boldsymbol{e}_{j,i}$ represents the edges connected between node i and node j. The edge features can be empty. $\bigcup$ denotes an operation that satisfies specific conditions, such as average function,



Neighbor node indexes stored via neighbor tables in the HBM/on-chip memory
1st stage: neighbor node sampling
2nd stage: feature acquisition
Feature stored in the HBM
3rd stage: Aggregation
Aggr result

Fig. 1. Sampling and aggregation in GNNs.

maximum function, or minimum function. f and g denote differentiable functions according to the GNN structures. This equation represents the unified form of a layer of GNN. The repetition of multiple layers of message-passing networks constitutes different kinds of GNN frameworks.

For example if the network structure is GraphSAGE, its network can be represented in the form shown in Eq. 2.

$$\boldsymbol{h}_i' = \boldsymbol{W} \cdot \mathrm{aggr}_{j\in N(i)} \boldsymbol{h}_j \tag{2}$$

where $\boldsymbol{W}$ is the weight matrix and $aggr$ represents the aggregation function, the specific form of its implementation can be the average aggregation, the maximum value, with weight aggregation and other aggregation methods.

B. What is the process that this paper is concerned with - Sampling and Aggregation

In a message passing network expression, the nodes involved in the aggregation are specified by the expression $j \in N(i)$. Where N represents the set of neighboring nodes of node i. If the criterion for selecting $N(i)$ is to strategically choose a fraction of all nodes, then N represents sampling. Fig. 1 illustrates the process that is the concern of this work. As shown in the Fig. 1, in a specific algorithm implementation or hardware implementation, the index number of the sampled node needs to be selected first (node sampling), and then the feature vectors need to be read from the memory storing the feature vectors according to the index number (feature acquisition). This work addresses both of these aspects with innovative optimization and acceleration.

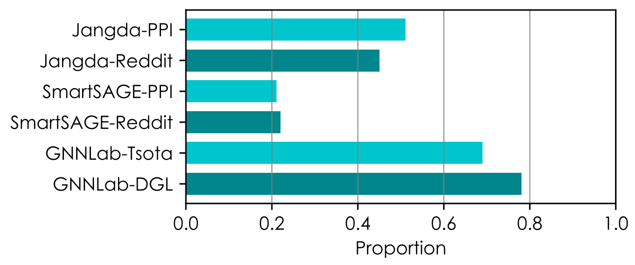


Fig. 2. Percentage of total elapsed time for sampling and feature acquisition in typical GNN architectures.

C. Application scenarios of this work - time consumption of sampling in GNN computation

In the GNN computation process, especially the GNN training process, sampling and the subsequent feature acquisition and feature aggregation usually account for a considerable proportion of the total elapsed time.

Based on the data measured or investigated in existing work such as GNNLab [14], Jangda [19] and SmartSAGE [13], the proportion of time taken by sampling and feature acquisition is shown in Fig. 2. Data based on the Reddit and PPI datasets were chosen for the figure, respectively. The test or research results shown in the figure show that sampling and feature acquisition take up a percentage of time between 20% and 80%, slowing down the system, so it is practical and necessary to speed them up.

III. Motivation

A. Motivation of Proposed Streaming Sampler

Traditional sampling generally takes place in the address space where the object to be sampled is located, i.e., it occurs in the **spatial domain**.

As shown in the Fig. 4, the traditional form of sampling generates an address signal by means of a random number generator and a random access memory by means of the generated address. The nodes to be sampled are stored in memory. In this way, one sampling result can be obtained per work cycle, and the time complexity of the sampling task is then $O(S)$, where S is the sample size. If a certain degree of parallel work is performed, the time consumed for sampling is reduced by a corresponding multiplier, but its time complexity is still $O(S)$.

Therefore, in order to accelerate the sampling process, improvements need to be made for the characteristics of the GNN dataset. The advantage of platforms such as FPGAs over software optimization on the CPU side is that they can utilize the transfer characteristics of the bus in a more direct and bottom-up manner at the hardware level.

Fig. 3 shows the number of nodes of different degrees in six typical GNN datasets in the form of histograms. According to

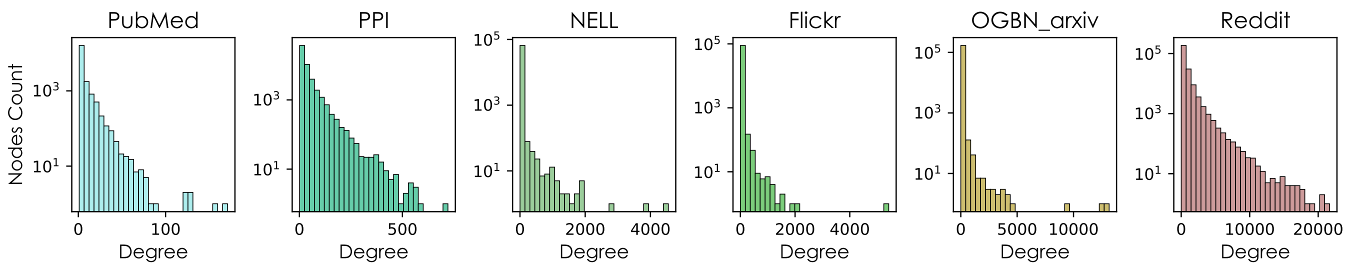
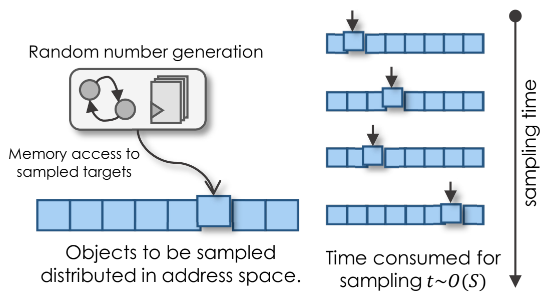
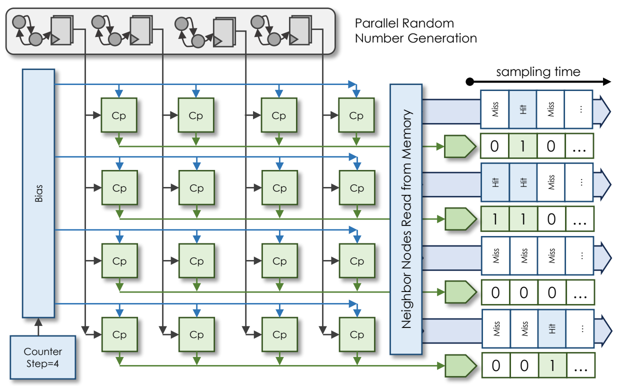


Fig. 3. The node degree distributions of the typical datasets shown in the figure all obey an approximate power law distribution, i.e., the vast majority of nodes possess very low degrees, while only a small number of nodes have large degrees. The vertical coordinates in the figure are logarithmic axes.



Random number generation
Memory access to sampled targets
Objects to be sampled distributed in address space.
Time consumed for sampling $t \sim O(S)$
sampling time

Fig. 4. Traditional sampling.



Parallel Random Number Generation
sampling time
Bias
Cp
Counter Step=4
Neighbor Nodes Read from Memory
Miss
Hit

Fig. 5. Structure of streaming sampler. As each node is sampled, the random number port is held constant and the counter is incremented each clock cycle. The comparator array outputs the sampling result after the OR operation. If the result after OR operation is 1, then it means that the neighbor node of the corresponding way is selected, otherwise it means that it is not selected.

this figure, the vast majority of the nodes have a small number of neighbors, and their distribution form shows a power law distribution, i.e., the vast majority of the nodes are low-degree nodes. This feature of the GNN dataset can be enlightening, and if the time complexity of the aforementioned sampling process can be shifted to $O(degree)$, the time of sampling can be reduced.

The sampler proposed in this paper samples the neighbor nodes while transferring them to the chip using burst transfer. Thus this sampler works directly during the bus data stream transfer and is a streaming sampler. The time it consumes is the time required to read the neighboring node to the chip without additional time overhead. According to this method, the time complexity of sampling is $O(degree)$. This proposed sampler accomplishes sampling at the same time as the bus burst transmission, occurring in the **time domain**.

B. Motivation of Feature Acquisition and Aggregation

After completing the node sampling, the index of the sample node can be obtained, and it is also necessary to obtain the feature vector of the sample node based on the index. This process will bring large-scale random memory access.

The feature vectors of the sample nodes are usually distributed irregularly and discontinuously scattered in the memory space. At the same time, unlike the node index values obtained from sampling, feature vectors usually contain data of several hundred or more dimensions, so that the length of data to be read for each random memory access increases.

FPGA with HBM can better cope with the above problem. First of all, the HBM has a high read bandwidth, which is suitable for the above task of intensive memory access. At the same time, it has more read ports, which can store the node features of different segments in the address segments corresponding to each port, so that multiple read ports can work in parallel when reading the feature vectors of different node segments, which greatly improves the access efficiency and the bandwidth utilization of the HBM.

However, for each central node, its multiple neighbor nodes are unevenly distributed across the different port segments of the HBM, and it is possible that multiple neighbor nodes are clustered at one port for reading, while other ports may have no neighbor nodes distributed. Meanwhile, the response time of different ports varies, and the neighbor nodes of the same central node distributed in different ports may be read out at different times. As a result, the multi-channel reads of the HBM bring about a temporal and spatial imbalance in the feature reads of neighbor nodes. Facing this imbalance, there is a need to propose an aggregation accelerator that is compatible with it.

IV. Proposed Hardware

A. Streaming Sampler

Based on the descriptions in the previous sections, this work proposes an innovative streaming sampler in the time domain.

As shown in Fig. 5 as an example, the width of the neighbor node input port is set to be wide enough to accommodate 4-way neighbor node indexes, and the sample size is set to be 4.

The four random numbers given by the previous random number generator module are used in parallel as input to the streaming sampler and are latched by the it. At the same time, at each clock cycle, the 4-way neighbor node index data is also input in parallel using the burst transmission of the data bus.

There is a counter set up in the module with a step size of 4 (the step size of the counter is the number of ways to input neighbor node indexes). Every time neighbor node indexes are input for one clock cycle, the value of this counter is increased. The first neighbor node input way in the array uses the value of the counter directly, the second neighbor node input way input uses the value of the counter after adding 1, and so on. This results in 4 counter offset values. Each counter offset value is compared to the four input random numbers respectively. In this way, each neighbor node index input way gets four comparator outputs. If one of the random numbers is equal to the counter value, it gets one bit of 1 in the four comparator outputs. The value obtained from the four comparator outputs is fed into a logical OR gate to get the result. The resulting value of the OR gate output identifies whether the neighbor node of that way is sampled or not and becomes a sample mask.

In this way, the sampler continues to provide sampling results as the data stream continues to enter the sampler. Thereby the sampling process does not consume dedicated time anymore, but is done while acquiring neighbor node indexes. This approach greatly saves time consumption and fits the characteristics of bus burst transmission.

At the same time, the streaming sampler has the advantage of being compatible with both playback and non-playback sampling: the 4 random number inputs are connected to 4 comparators that give their respective comparison results. If some of the random numbers give the same value (There are duplicate entries in the samples obtained with playback sampling.), then the number of 1's in the 4-bit comparison result is the result of playback sampling. If an OR operation is performed on the 4-bit comparison result, the value given is the result without playback sampling.

B. HBM-based Feature Acquisition & Aggregation Accelerator

The second part of the system is the HBM-based feature acquisition and aggregation accelerator. During the feature acquisition process, the address range of the random memory accesses is the entire feature table of the GNN dataset, rather than being limited to a segment as in the case of accessing the neighbor table. Therefore, it is reasonable to choose HBM as the memory for this task.

The accelerator uses up to 32 independent HBM ports, each port accesses one segment of HBM space, and the feature vectors of different nodes are uniformly distributed in each

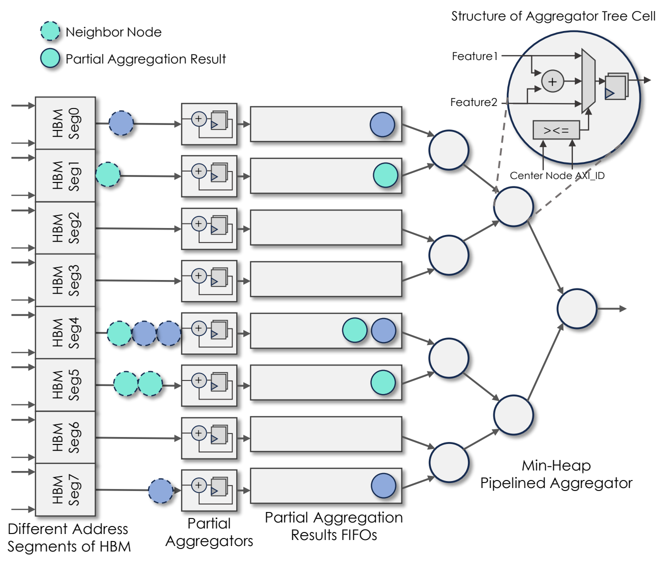

Neighbor Node
Partial Aggregation Result
Structure of Aggregator Tree Cell
Feature1
Feature2
>&lt;=
Center Node AXI_ID
HBM Seg0
HBM Seg1
HBM Seg2
HBM Seg3
HBM Seg4
HBM Seg5
HBM Seg6
HBM Seg7
Different Address Segments of HBM
Partial Aggregators
Partial Aggregation Results FIFOs
Min-Heap Pipelined Aggregator

Fig. 6. HBM-based feature loader array and aggregation accelerator

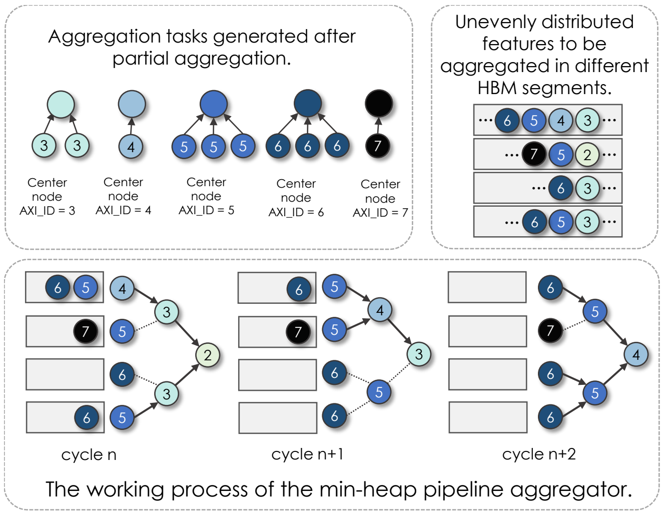

Aggregation tasks generated after partial aggregation.
Center node AXI_ID = 3
Center node AXI_ID = 4
Center node AXI_ID = 5
Center node AXI_ID = 6
Center node AXI_ID = 7
Unevenly distributed features to be aggregated in different HBM segments.
cycle n
cycle n+1
cycle n+2
The working process of the min-heap pipeline aggregator.

Fig. 7. Example of how the aggregation accelerator works.

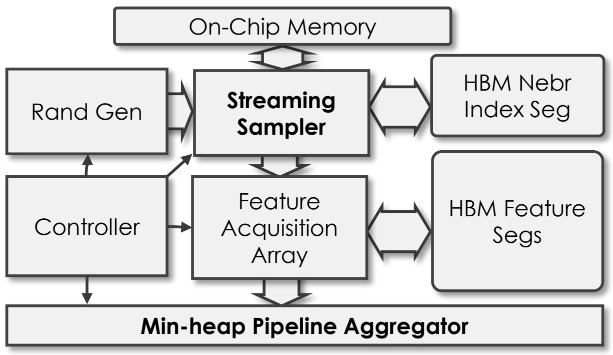


Fig. 8. Hardware system. The bolded parts are the two innovative modules presented in this work.

TABLE I
THE PARAMETERS OF THE DATASET USED.

Dataset	#Nodes	#Edges	#Features	Ave. Deg.
PubMed	$19,717$	$88,648$	500	4.5
PPI	$56,944$	$1,612,348$	50	28.3
NELL	$65,755$	$251,550$	61278	3.8
Flickr	$89,250$	$899,756$	500	10.1
OGBN_arxiv	$169,343$	$1,166,243$	128	6.9
Reddit	$232,965$	$114,615,892$	602	491.9

segment in order according to the node index, and the space size of each HBM segment is 256 MB. if the size of the dataset after segmentation is less than 256 MB, unutilized regions are left in each segment. The sample node indexes obtained from the sampler are fed to the HBM through the address port to obtain the feature vectors. The sample nodes corresponding to different central nodes are identified with each other using the AXI_ID of the AXI interface, so that the neighboring nodes read by different channels can be identified by the AXI_ID to which central node they belong to, and then aggregated to the correct central node.

The schematic is drawn using 8 segments as shown in the Fig. 6. When the neighbor nodes are read out, partial aggregation is first performed in the partial aggregator module to aggregate the features of neighbor nodes belonging to the same central node in each channel into one partial aggregation result. Then, the partial aggregation results in different segments enter the aggregator tree. Each cell in the aggregator tree receives the feature vectors from two upstream cells and the *central node AXI_ID* (CID) of the partial aggregation results. The cell works according to the following rules: if the CIDs of the two inputs are equal, then the two feature vectors are aggregated and passed downstream pass, if the CIDs of the two upstream child cells are different, the feature vector of the child cell with the smaller CID is selected to pass downstream while keeping the state of the child cell with the larger CID unchanged and also recursively locking the state of its upstream child cells.

The example in Fig. 7 shows the working process of the proposed aggregation accelerator. After each segment in the HBM is partially aggregated, the partial aggregation results are fed into the pipeline aggregation tree for aggregation between different segments in the HBM. The task to be accomplished is shown in the figure where features of partial aggregations belonging to different central nodes are distinguished by different CIDs. Whereas the distribution of some of the aggregation results is uneven, there are more features to be aggregated on some channels.

In the cycle n of the work of the aggregator in the figure, the two child cells of the root cell with $\text{CID} = 2$ have the same CID, so they are aggregated and passed downstream to the root cell. And the child cells of the cell with $\text{CID} = 3$ have different CIDs, so only the path with small CID is passed downstream to get the result for the cycle $(n + 1)$. About the two child cells of the root cell in the cycle $(n + 1)$, the path with $\text{CID} = 4$ is the path with the smaller CID and is therefore activated and transmitted downstream. The two sub-paths of the cell with $\text{CID} = 4$ have the same CID and both are aggregated. This gives the result for the cycle $(n + 2)$. And so on, the aggregator can continuously receive data from upstream and work as a pipeline. At each clock cycle, the root cell can output an aggregation result

The two aforementioned hardware modules are the main components of this work and they are placed in the hardware system (Fig. 8). In addition, there are other auxiliary modules: the Controller controls the whole system, the On-Chip Memory holds the degree of the dataset and the base address of the neighbor indexes of each node, the Rand Gen generates the random numbers used for sampling, the Feature Acquisition Array buffers the samples and reads the feature vectors, and the HBM is divided into two parts, storing the neighbor indexes and the node feature vectors, respectively.

V. EVALUATION

The graph datasets vary greatly in size, and datasets from small to large sizes were selected for testing separately in the experiments. The information of the selected datasets is shown in the Tab. I.

The number of nodes in these datasets ranges from around 19,000 to 230,000. The maximum feature vector dimension supported by the aforementioned feature vector acquisition module is 1024, where the PubMed [20], OGBN_arxiv [21] and Reddit [7] datasets satisfy the requirement, and the feature vectors of the NELL [22] dataset are sparse and can be compressed to within the required range.

The traditional FPGA-based node sampling module in the spatial domain is first implemented and the time consumed to perform node sampling is measured to serve as the node sampling time baseline, which in turn demonstrates the effectiveness of the streaming sampler's improvements to the FPGA sampler.

Then the proposed HBM and FPGA based feature acquisition and aggregation accelerator is used for comparison with GPU baseline. The data format in both the GPU baseline and the proposed hardware is INT8. The FPGA platform used for validation is the AMD/Xilinx Virtex UltraScale+ HBM VCU128 FPGA Evaluation Kit with 8GB of integrated HBM [23]. Testing on the GPU platform was done using an NVIDIA Tesla P100-PCIE-16GB as the device, in combination with PyTorch and the PyG framework.

The clock frequency at which the FPGA tests were performed was 100 MHz, and the sample size of the sampling module was set to 32, which meets the sample size requirements of commonly used networks. The overall power consumption of the system is 15.05 W, most of which is consumed by the HBM storage, which is 10.33 W. The power consumption occupied by the proposed scheme in this work is 4.72 W.

From the measurement results shown in Tab. II and Fig. 9, it can be seen that the FPGA-based streaming sampler consumes significantly less time than the traditional spatial-domain FPGA-based sampler, thus proving the effectiveness of the

TABLE II
EVALUATION RESULT

Dataset	Node Sampling Time (ms)			Full-Dim Acquisition Time (ms)			128-Dim Acquisition and Aggregation Time (ms)		
	Traditional Sampling	Streaming Sampler	Speedup	GPU	Ours	Speedup	GPU	Ours	Speedup
PubMed	20.54	**0.99**	20.7×	67.7	**4.11**	16.5×	79.82	**0.24**	332.6×
PPI	58.73	**2.95**	19.9×	15.3	34.13	0.4×	16.27	**2.89**	5.6×
NELL	68.71	**3.35**	20.5×	191.11	**21.12**	9.0×	208.05	**2.89**	72.0×
Flickr	92.53	**4.51**	20.5×	389.05	**40.09**	9.7×	453.22	**4.18**	108.4×
OGBN_arxiv	175.37	**8.53**	20.6×	865.71	**70.18**	12.3×	963.22	**4.17**	231.0×
Reddit	239.96	**118.77**	2.0×	8996.83	**282.42**	31.9×	9284.60	**148.55**	62.5×

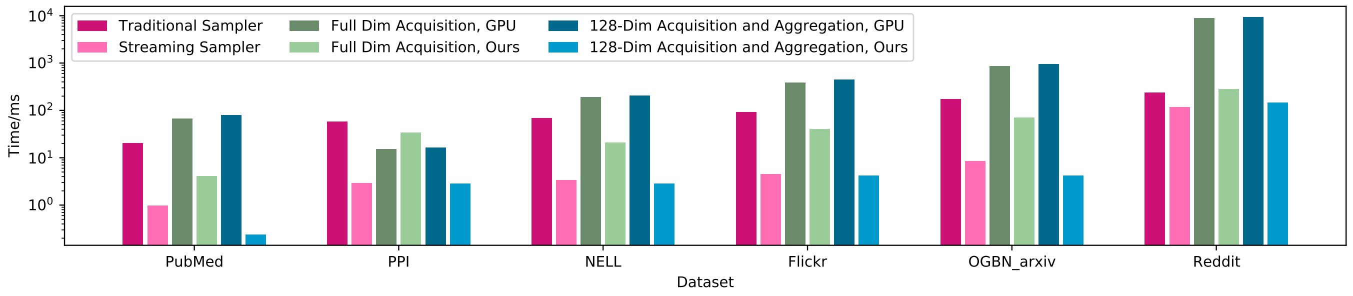


Fig. 9. The results of the evaluation of the proposed hardware, where the vertical coordinate is the time consumption in milliseconds, plotted on a logarithmic axis, and the horizontal coordinate is the different data sets. The dark data bars in the figure are the baseline and the light data bars are the evaluation results of the proposed hardware.

stream sampler. Among them, the acceleration of datasets with smaller average node degree (PubMed, NELL, OGBN_arxiv) is better, while the acceleration of the Reddit dataset is smaller, which is due to the fact that the Reddit dataset's average node degree is larger, the data transmission consumes more time, and the measured sampling time obtained is correspondingly longer.

The computational process of GNN is divided into two parts: combination (matrix multiplication or dimension transformation) and aggregation (the commonly used aggregation method is average aggregation). For the input feature vectors of original dimensions, they should generally be combined first to reduce the dimensions (dimension transformed to the dimension of the hidden layer), and then the resulting feature vectors can be aggregated. To address the above two aspects, this work conducts two tests for the acceleration of the feature vector acquisition and aggregation parts: 1. the acquisition of feature vectors with original full-dimensions (up to 1024 dimensions are supported). 2. the acquisition and aggregation of feature vectors of the hidden layer with up to 128 dimensions. The results are shown in the Tab. II and 9.

According to the test results, the proposed accelerator achieves acceleration relative to traditional samplers and GPU baseline in the majority of scenarios, validating the effectiveness of the proposed architecture. The proposed streaming sampler achieves a $2\times$ to $20\times$ speedup relative to the baseline, and the proposed feature acquisition and aggregation accelerator achieves up to $300\times$ speedup relative to the baseline.

VI. CONCOLUSION

This paper first describes the proportion of time that sampling and feature acquisition occupy in GNN training and inference, thus illustrating the need for acceleration of the process.

The node degree of the graph dataset is characterized by a power-law distribution, which brings some inspiration: Sampling can be performed in the process of reading neighbor nodes, thus shifting the time complexity of sampling to $O(degree)$, and speedup is achieved by power-law distribution of node degree. At the same time, the high bandwidth advantage of the HBM enables fast feature acquisition. In addition, a matching accelerator is required to aggregate the unbalanced data read from the HBM. A streaming sampler is proposed which has a large performance improvement over traditional hardware sampling and a high-performance feature acquisition-aggregation accelerator (min-heap pipeline aggregator) is proposed which has a faster speed than GPU baseline. Compared to traditional hardware sampling, the proposed sampler also has the ability to be compatible with both put-back and no-put-back sampling strategies.

Based on the experimental validation results, the proposed streaming sampler achieves a $2\times$ to $20\times$ speedup and the proposed feature acquisition and aggregation accelerator achieves up to $300\times$ speedup.

REFERENCES

[1] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural

networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[2] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[3] Fan Liang, Cheng Qian, Wei Yu, David Griffith, Nada Golmie, et al. Survey of graph neural networks and applications. *Wireless Communications and Mobile Computing*, 2022, 2022.

[4] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys*, 55(5):1–37, 2022.

[5] Weiwei Jiang. Graph-based deep learning for communication networks: A survey. *Computer Communications*, 185:40–54, 2022.

[6] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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

[8] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[9] Youhui Bai, Cheng Li, Zhiqi Lin, Yufei Wu, Youshan Miao, Yunxin Liu, and Yinlong Xu. Efficient data loader for fast sampling-based gnn training on large graphs. *IEEE Transactions on Parallel and Distributed Systems*, 32(10):2541–2556, 2021.

[10] Marco Serafini and Hui Guan. Scalable graph neural network training: The case for sampling. *ACM SIGOPS Operating Systems Review*, 55(1):68–76, 2021.

[11] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. Hygcn: A gcn accelerator with hybrid architecture. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 15–29. IEEE, 2020.

[12] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, et al. Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 922–936. IEEE, 2020.

[13] Yunjae Lee, Jinha Chung, and Minsoo Rhu. Smartsage: training large-scale graph neural networks using in-storage processing architectures. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 932–945, 2022.

[14] Jianbang Yang, Dahai Tang, Xiaoniu Song, Lei Wang, Qiang Yin, Rong Chen, Wenyuan Yu, and Jingren Zhou. Gnnlab: a factored system for sample-based gnn training over gpus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 417–434, 2022.

[15] Pyg, 2019. https://www.pyg.org/.

[16] Li Zhang, Dan Xu, Anurag Arnab, and Philip HS Torr. Dynamic graph message passing networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3726–3735, 2020.

[17] Xiaolong Fan, Maoguo Gong, Yue Wu, A Kai Qin, and Yu Xie. Propagation enhanced neural message passing for graph representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(2):1952–1964, 2021.

[18] Muhammet Balcilar, Pierre Héroux, Benoit Gauzere, Pascal Vasseur, Sébastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks. In *International Conference on Machine Learning*, pages 599–608. PMLR, 2021.

[19] Abhinav Jangda, Sandeep Polisetty, Arjun Guha, and Marco Serafini. Accelerating graph sampling for graph machine learning using gpus. In *Proceedings of the sixteenth European conference on computer systems*, pages 311–326, 2021.

[20] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

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

[22] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam Hruschka, and Tom Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24, pages 1306–1313, 2010.

[23] Virtex ultrascale+ hbm vcu128 fpga evaluation kit, 2019. https://www.xilinx.com/products/boards-and-kits/vcu128.html.