HolisticGNN: Geometric Deep Learning Engines for Computational SSDs

Miryeong Kwon   Donghyun Gouk   Sangwon Lee   Myoungsoo Jung
Computer Architecture and Memory Systems Laboratory
Korea Advanced Institute of Science and Technology (KAIST)
http://camelab.org

I. Introduction

Graph neural networks (GNNs) process large-scale graphs consisting of a hundred billion edges, which exhibit much higher accuracy in a variety of prediction tasks. However, as GNNs are engaged with a large set of graphs and embedding data on storage, they suffer from heavy I/O accesses and irregular computation.

We propose a novel deep learning framework on large graphs, *HolisticGNN*, that provides an easy-to-use, near-storage inference infrastructure for fast, energy-efficient GNN processing. To achieve the best end-to-end latency and high energy efficiency, HolisticGNN allows users to implement various GNN algorithms and directly executes them where the data exist in a holistic manner.

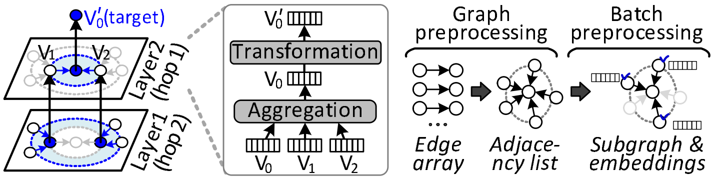
We fabricate HolisticGNN's hardware RTL and implement its software on an FPGA-based computational SSD (CSSD). Our empirical evaluations show that the inference time of HolisticGNN outperforms GNN inference services using high-performance GPU by 7.1× while reducing energy consumption by 33.2×, on average.

II. Graph Neural Networks and Its Challenge

GNNs generalize conventional deep learning (DL) to understand structural information in the graph data by incorporating feature vectors (i.e., *embeddings*) in the learning algorithms. GNNs can capture topological structures of the local neighborhood (per node) in parallel with a distribution of the neighborhood's node embeddings [1].

GNN algorithm. As shown in Figure 1a, GNNs process the structural information with node embeddings by employing multiple layers. Each computational layer of GNNs is composed of two primary execution phases, called neighborhood *aggregation* and node *transformation*. Based on a given target node, each layer performs the aggregation and transformation for a different hop of neighbors (connected to the target node). While the aggregation accumulates node embeddings of the target node's neighbors, the transformation converts the aggregated results to a new node embedding through a *multi-layer perceptron* (MLP). Aggregation, therefore, processes data relying on graph structures and mostly exhibits irregular, graph-natured execution patterns. In contrast, the computing procedure of transformation is very similar to that of conventional neural networks (e.g., CNNs and RNNs), but it does not require heavy computation.

GNN preprocessing. GNNs need to deal with real-world graphs consisting of billions of edges and node embeddings [2]. The graph information initially reside in storage and are regularly updated as raw-format data owing to their large size and persistence requirements (cf. Figure 1b). As GNNs need to understand the structural geometry and feature information of given graph(s), the raw-format data should be loaded into working memory and reformatted in the form of an adjacency list before the actual inference services begin. We refer to this task as *graph preprocessing*. Since the internal memory of GPUs is insufficient to accommodate all the inputs, it is essential to reduce the size of the graph and embeddings by preprocessing them. The modern GNN models in practice sample a set of subgraphs and embeddings from the target graph information and aggregate the sampled embeddings for inductive node inferences. This *batch preprocessing* can significantly reduce the amount of data to process, which can also decrease the computation complexity to infer the results without an accuracy loss.



(a) GNN algorithm. (b) GNN preprocessing.

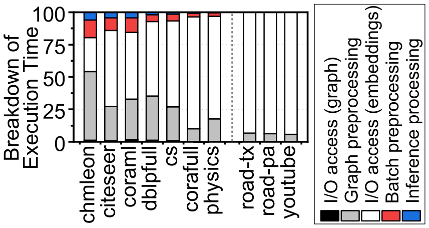
Fig. 1: End-to-end GNN inference.

Challenge analysis. While there is less system-level attention on the management of both graph preprocessing and batch preprocessing, their tasks introduce heavy storage accesses and frequent memory operations. To analyze these challenges, we use the most popular GNN model, GCN [3], and decompose the *"end-to-end GCN inference"* times across 13 real-world graph datasets (cf. Table 2b). As shown in Figure 2a, GCN inference processing only takes 2% of the end-to-end inference latency, on average. Specifically, storage accesses for embeddings account for 61% of the most end-to-end latency for the small graphs having less than 1 million edges. Since graph preprocessing includes a set of heavy computing processes such as a radix sort, it also consumes 28% of the end-to-end latency for the small graphs. As the graph size increases (> 3 million edges), storage accesses for embeddings become the dominant contributor of the end-to-end GNN inference time (94%, on average).

III. Near-Storage Inference Framework

As shown in Figure 3a, the new concept of computational SSDs (CSSDs) decouples the computing unit from the storage resources by locating *reconfigurable hardware* (FPGA) near SSD in the same PCIe subsystem (card) [4]. CSSD allows the hardware logic fabricated in FPGA to access the internal SSD via the internal PCIe switch. To this end, the host is responsible for writing/reading data on the SSD using the I/O region of NVMe protocol while giving the data's block address to the FPGA through its own I/O region, whose address is designated by PCIe's base address register. While CSSD is promising to realize near-data processing, it is non-trivial to automate all end-to-end procedures of GNN inference over hardware-only logic because of the variety of GNN model executions. For example, the aggregation and combination of GNNs can be accelerated with parallel hardware architecture, but GNN's graph traversing, GNN preprocessing, and embedding handling are impractical to be programmed into hardware because of their graph-natured computing irregularities.

Overview of HolisticGNN. HolisticGNN is a hardware and software co-programmable framework that leverages CSSD to accelerate the end-to-end GNN inference services near storage efficiently.

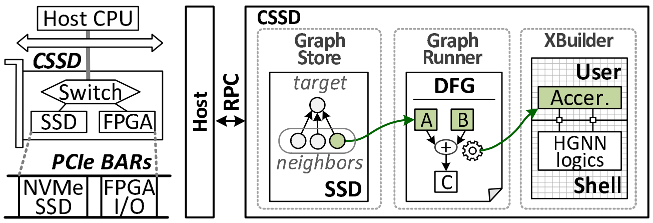


(a) Latency breakdown.

Legend	Edges	Feature
chmleon	65K	2326
citeseer	9K	3704
coraml	19K	2880
dblpfull	123K	1639
cs	182K	6805
corafull	147K	8710
physics	530K	8415
road-tx	3.84M	4353
road-pa	3.08M	4353
youtube	2.99M	4353

(b) Graph dataset.

Fig. 2: Challenge analysis of end-to-end GNN inference.



(a) CSSD. (b) Overview of HolisticGNN.

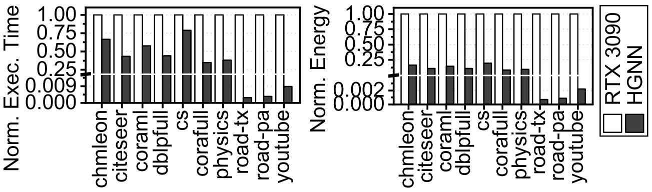
Fig. 3: Enabling CSSD for near storage GNN processing.

The software part of our framework offers easy-to-use programming/management interfaces and performs GNN preprocessing directly from where the data is stored, thereby minimizing the aforementioned storage access overhead. On the other hand, our framework's hardware logic and administration module provide a low-overhead bare-metal computing environment and reconfigurable hardware to accelerate GNN model executions. As shown in Figure 3b, our framework is specifically composed of three distinguishable components to support fast and energy-efficient GNN processing: i) graph-centric archiving system (*GraphStore*), ii) programmable inference client and server model (*GraphRunner*), and iii) accelerator building system (*XBuilder*).

Graph-centric archiving system. The main purpose of GraphStore is to bridge the semantic gap between the graph abstraction and its storage representation without having a storage stack. GraphStore manages the user data as a graph structure rather than exposing it directly as files. Generally speaking, GraphStore converts the incoming edge array to an adjacency list in parallel with transferring the embedding table, and it stores them to the internal SSD. This makes the conversion and computing latency overlapped with the heavy embedding table updates, which can deliver the maximum bandwidth of the internal storage. For the storage accesses, GraphStore uses VID to *logical page number* (LPN) mapping information by being aware of a long-tailed distribution of graph degree as well as flash page access granularity. The LPNs are used for accessing CSSD's internal storage through NVMe, which can minimize the write amplification caused by I/O access granularity differences.

Programmable inference model. GraphRunner processes a series of GNN inference tasks from the beginning to the end by allowing users to program the tasks using a computational graph, *dataflow graph* (DFG). The users can then simply transfer the DFG into the CSSD and manage its execution through a remote procedure call (RPC). This does not require cross-compilation or storage stack modification to program/run a user-defined GNN model. Once the DFG is downloaded, GraphRunner executes each node by checking the registered hardware codes in CSSD. The users may want to register more hardware codes because of adopting a new GNN model or hardware logic. To meet this requirement, GraphRunner also offers a Plugin mechanism registering hardware codes and a new device configuration as a shared object.

Accelerator builder. XBuilder manages the FPGA hardware infrastructure and accelerates diverse GNN algorithm executions near storage. It first divides the FPGA logic die into two regions, *Shell* and *User*, using the dynamic function exchange (DFX) technique [5]. XBuilder then secures hardware logic necessary to run GraphStore and GraphRunner at Shell while placing DL accelerator(s) to User. The Shell and User hardware are programmed to CSSD as two separate bitstreams, such that we can reprogram the User with a different bitstream at any time. To this end, XBuilder implements a hardware engine in Shell by using an internal configuration access port, which downloads a bitstream and programs it to User.



(a) End-to-end latency. (b) Energy consumption.

Fig. 4: End-to-end performance comparison.

IV. EVALUATIONS AND CONCLUSION

Evaluation setup. We prototype a customized CSSD that employs a 14nm 730MHz FPGA chip [6], 16GB DDR4-2400 DRAM, and a 4TB high-performance SSD together within the same PCIe 3.0×4 subsystem. For a fair performance comparison, we also prepare high-performance GPU, RTX 3090.

End-to-end latency. Figure 4a shows that our HolisticGNN (*HGNN*) exhibits 2× shorter end-to-end latency compared to RTX 3090 for the small graphs. This performance superiority of HGNN becomes higher when we infer large-scale graphs (e.g., `youtube`), which makes HGNN 222.8× faster than RTX 3090, on average. HGNN can preprocess graphs in parallel with the graph updates and prepare sampled graphs/embeddings directly from the internal SSD, thereby successfully reducing the overhead of GNN preprocessing and storage accesses.

Energy consumption. Figure 4b analyzes the energy consumption characteristics of HGNN. For the small graphs, HGNN exhibits 8.1× better energy consumption behaviors compared to RTX 3090, on average. This is because our CSSD consumes only 111 Watts at the system-level thanks to its low-power computing of FPGA (16.3 Watts) while significantly reducing the inference latency. This makes HGNN much more promising on GNN computing compared to other GPU-based acceleration approaches. HGNN's energy gains become more significant when analyzing the large-scale graphs; it consumes 453.2× less energy than the RTX 3090, on average.

In conclusion, we designed an easy-to-use, near-storage inference infrastructure for fast, energy-efficient GNN processing by allowing users to implement various GNN algorithms close to the data source and execute them directly near storage in a holistic manner.

V. DEMOSCENE AND ORIGINAL PUBLICATION

Demo video. *https://www.youtube.com/watch?v=b5fZBESH1TM*
Original publication. M. Kwon, D. Gouk, S. Lee, and M. Jung. USENIX FAST 2022. Hardware/Software Co-Programmable Framework for Computational SSDs to Accelerate Deep Learning Service on Large-Scale Graphs. *https://www.usenix.org/system/files/fast22-kwon.pdf*

VI. ACKNOWLEDGEMENT

This research is supported by Samsung Research Funding & Incubation Center of Samsung Electronics (SRFC-IT2101-04). Myoungsoo Jung is the corresponding author.

REFERENCES

[1] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation Learning on Graphs: Methods and Applications," *arXiv preprint arXiv:1709.05584*, 2017.

[2] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. L. Lee, "Billion-scale commodity embedding for e-commerce recommendation in alibaba," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[3] T. N. Kipf and M. Welling, "Semi-supervised Classification with Graph Convolutional Networks," *arXiv preprint arXiv:1609.02907*, 2016.

[4] S. Electronics, "Samsung SmartSSD," https://samsungsemiconductor-us.com/smartssd-archive/pdf/SmartSSD_ProductBrief_13.pdf.

[5] Xilinx, "Dynamic Function eXchange," https://www.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx2021_1/ug909-vivado-partial-reconfiguration.pdf.

[6] ——, "Virtex UltraScale+ FPGA," https://www.xilinx.com/content/dam/xilinx/support/documentation/product-briefs/virtex-ultrascale-product-brief.pdf.