A Generic FPGA Accelerator Framework for Ultra-Fast GNN Inference

Rishov Sarkar, Cong Hao
Georgia Institute of Technology, School of Electrical and Computer Engineering, Atlanta, GA, USA
rishov.sarkar@gatech.edu, callie.hao@ece.gatech.edu

1 Introduction

Graph structures are prevalent in real-world data and often uncover learnable tasks at the node level (e.g., presence of protein [1]), edge level (e.g., drug-drug interactions [2]), and graph level (e.g., molecular property prediction [3]). Graph neural networks (GNNs) are the solution to apply deep learning to graph-related problems such as analysis of social networks and citation networks, recommendation systems, traffic forecasting, LIDAR point cloud segmentation for autonomous driving, high-energy particle physics, and molecular representations [4].

Real-time applications such as high-energy physics demand high-throughput, low-latency GNN inference of thousands of graphs. Achieving state-of-the-art accuracy in such applications motivates the need for a *generic, extensible, and flexible acceleration framework* that can easily be adapted to any message-passing GNN without compromising on latency. Our previous work, **GenGNN** [5], presents our initial effort on a generic acceleration framework, but it overlooks more advanced optimization opportunities, the most promising of which is parallelization of node and edge computations.

Therefore, in this work, we introduce the generic architecture of GenGNN and present further optimizations upon it that reduce the latency of GenGNN by up to 42%.

2 Related Work

GNN acceleration is attracting intensive attention in the research community. Recent works are summarized by a survey [6], which includes both ASIC and FPGA accelerators.

The majority of existing accelerators target ASICs in simulation. For instance, HyGCN [7] is among the earliest of these, which introduces a hybrid architecture for GCN acceleration. EnGN [8] uses PEs connected in a ring and performs aggregations using a technique called Ring-Edge Reduce, while GRIP [9] uses the GReTA abstraction [10] to enable acceleration of any GNN variant. GCNAX [11] addresses the shortcomings of resource underutilization and excessive data movement using a flexible dataflow.

On the other hand, FPGA-based accelerators primarily focus on GCN acceleration. AWB-GCN [12] is an FPGA accelerator that aims to combat workload imbalance in graph processing. Zhang *et al.* [13] combine software preprocessing with hardware utilizing both node-level and feature-level parallelism, while BoostGCN [14] specifically optimizes GCN via sparsity analysis and graph partitioning. I-GCN [15] uses an islandization approach to de-duplicate redundant GCN computations. Auten *et al.* [16] propose an architecture that uses general-purpose CPUs connected by a network-on-chip to accelerate various GNNs. Rubik [17] and GraphACT [18] aim to accelerate GCN training using ASIC and FPGA, respectively.

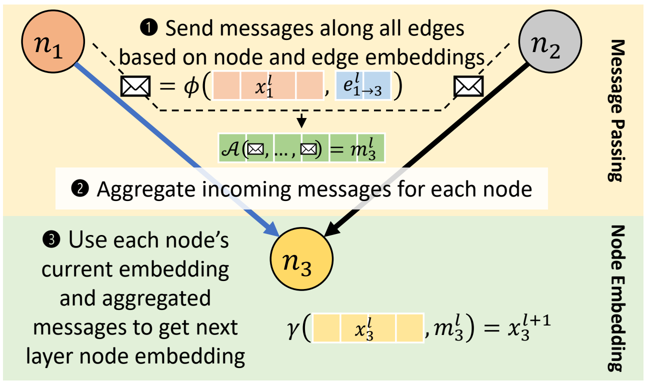
n_1
n_2
n_3
❶ Send messages along all edges based on node and edge embeddings
$✉ = \phi(x_1^l, e_{1\to3}^l)$
$\mathcal{A}(✉, \ldots, ✉) = m_3^l$
❷ Aggregate incoming messages for each node
❸ Use each node's current embedding and aggregated messages to get next layer node embedding
$\gamma(x_3^l, m_3^l) = x_3^{l+1}$
Message Passing
Node Embedding

Figure 1. The general computation pattern of a message-passing GNN (one layer).

2.1 Prior Art Limitations

Most prior FPGA-based GNN accelerators have been developed to accelerate models such as Graph Convolutional Networks (GCN) whose computation can be reduced to sparse-matrix/matrix multiplication (SpMM), but many GNN models are not reducible this way. For instance, the Principal Neighborhood Aggregation (PNA) model achieves state-of-the-art performance on many node-level and graph-level benchmarks [19], but it makes use of complex computations such as min, max, and standard deviation that cannot be computed using SpMM. Instead, the general behavior of GNN models can be represented as layers of message-passing operations, combined with transform and aggregate operations [20].

In addition, many of these accelerators also adopt off-chip preprocessing such as graph partitioning, which in real-time applications is not possible.

3 Generic and Parallelized Message-Passing

3.1 Generic Message Passing — GenGNN

The message-passing computation paradigm accommodates the widest range of GNN models. The general computation of each layer of a message-passing GNN is shown in Figure 1.

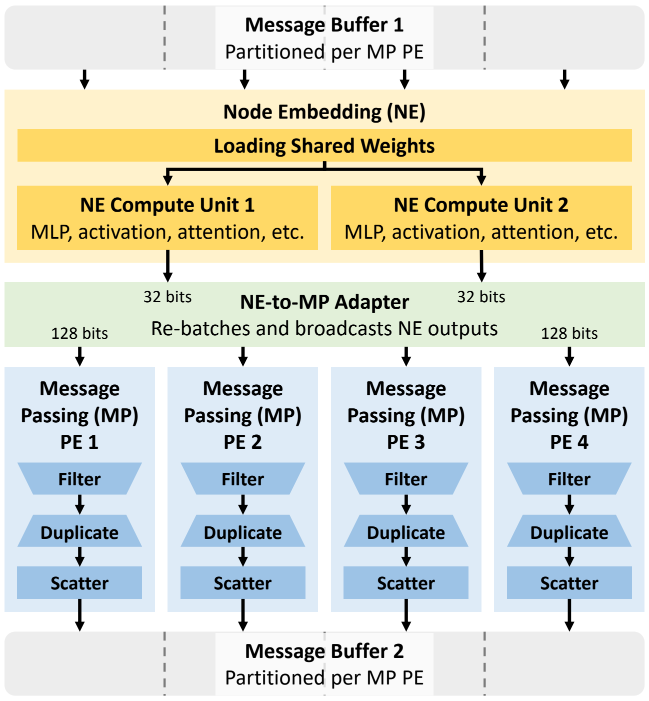

Figure 2. Our architecture for node-level and edge-level parallelism in each GNN layer. This example demonstrates 2× node-level parallelism and 4× edge-level parallelism.

Each layer can be interpreted as two interdependent steps: a Message-Passing (MP) step involving computations across all neighbors of each node and a Node Embedding (NE) step that updates each node's embedding using the aggregated messages computed in the MP step. Different GNN models are distinguished by different choices for the aggregation function $\mathcal{A}(\cdot)$ and the node transformation function $\gamma(\cdot)$.

GenGNN's architecture follows the message-passing style. It consists of two main Processing Elements (PEs), NE and MP, connected by a node queue. The node queue enables a novel streaming-based pipelined processing paradigm that overlaps NE and MP and minimizes idle time in both PEs.

However, although GenGNN overlaps the computation of the two stages, each is limited to sequential processing; NE can only perform one node transformation at any given time, and MP can only perform aggregations one edge at a time. Our latest work addresses this using both node-level parallelism in NE and edge-level parallelism in MP.

The architecture of our node-level and edge-level parallel PEs are shown in Figure 2. Section 3.2 describes the implementation of the parallel NE PE, and section 3.3 describes the implementation of the parallel MP PEs.

3.2 Node-Level Parallelism in Node Embedding PE

To implement node-level parallelism by a factor of P_N, we expand our Node Embedding PE to support simultaneous computation of P_N node transformations. Input messages are read P_N nodes at a time, and transformed node embeddings are output simultaneously via P_N parallel FIFO streams. Any weights needed for the node transformation are loaded only once and reused for the P_N nodes being processed in parallel.

Accelerator	DSP	LUT	FF	BRAM	URAM
Available	5,952	872K	1,743K	1,344 (47 Mb)	640 (180 Mb)
GenGNN	1,042	73,735	93,579	523	0
This work	1,559	202,763	167,044	462	0

Table 1. Resource utilization of the DGN model [21] on the Xilinx Alveo U50 FPGA. The clock frequency is 300 MHz.

3.3 Edge-Level Parallelism in Message Passing PEs

To implement edge-level parallelism by a factor of P_E, we create P_E separate Message Passing PEs, each handling a subset E_i of the edges of the entire graph—specifically, only edges directed to the corresponding partition of the message buffer.

Each MP PE is constructed as a dataflow accepting P_N input streams from the NE PE. The "filter" process discards nodes with no outgoing edges within E_i. Next, the "duplicate" process repeats each node's embedding once for each of its outgoing edges in E_i, resulting in a total of $|E_i|$ node embeddings. Finally, the "scatter" process performs the GNN-specific aggregation for each edge, taking as input the transformed node embedding for the source node of each edge.

This dataflow within each PE makes the loop behavior of each process predictable, thereby reducing loop overhead, and ensures each process is limited only by its input rate.

4 Evaluation and Results

We select the Directional Graph Network (DGN) [21] as a representative model for evaluation of our FPGA framework, and we evaluate its performance using the MolHIV dataset from the Open Graph Benchmark [22], a binary classification dataset with 41,127 graphs averaging 25.5 nodes and 27.5 edges each. We compare the inference latency of our DGN model on a Xilinx Alveo U50 FPGA against a PyTorch-based implementation running on CPU (Intel Xeon Gold 6226R)

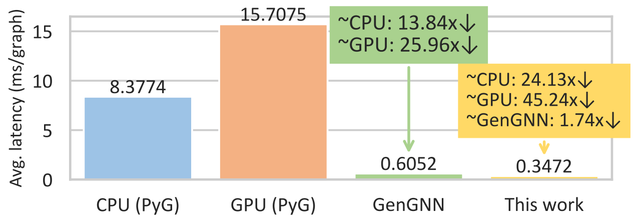

Figure 3. Average inference latency of the DGN model [21] on the MolHIV dataset [22] across several platforms.

and GPU (NVIDIA RTX A6000). FPGA resource utilization is shown in Table 1. All models are evaluated with batch size 1.

Results are shown in Figure 3. First, GenGNN achieves a speedup of 13.84× over the CPU and 25.96× over the GPU baseline; second, using the edge and node parallelization technique proposed in this work, we achieve another 1.74× speedup over GenGNN, with 2× node-level parallelism and 4× edge-level parallelism; it results in a speedup of 22.39× over CPU and 41.97× over GPU.

In this parallelized architecture, the vast majority of the inference latency can be attributed to pure overhead from starting and stopping the kernel on the FPGA. We evaluated the latency of a no-compute kernel, i.e., one which does nothing except return a fixed result, and found that it takes an average of 0.3232 ms per graph. Subtracting this latency from the overall inference latency, we find that in GenGNN, the time spent performing GNN computation averages 0.2820 ms, while our latest work with parallelization reduces this average to only 0.0240 ms per graph. That is, when comparing only the time spent on GNN computation, our current architecture achieves a speedup of nearly 12× over GenGNN.

5 Conclusion

In this work, we proposed a version of GenGNN enhanced with node-level and edge-level parallelism. The architecture is flexible enough to accommodate any message-passing GNN yet performant enough to beat state-of-the-art latency.

The most significant opportunity for future optimization is to reduce the kernel start/stop overhead, which will enable inference speeds as low as tens of microseconds per graph.

Future work includes design automation, design space exploration, and optimization for large graphs.

References

[1] D. Szklarczyk, A. L. Gable, D. Lyon, A. Junge, S. Wyder, J. Huerta-Cepas, M. Simonovic, N. T. Doncheva, J. H. Morris, P. Bork, L. J. Jensen, and C. v. Mering, "STRING v11: Protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets," *Nucleic Acids Research*, vol. 47, pp. D607–D613, Jan. 2019.

[2] D. S. Wishart, Y. D. Feunang, A. C. Guo, E. J. Lo, A. Marcu, J. R. Grant, T. Sajed, D. Johnson, C. Li, Z. Sayeeda, N. Assempour, I. Iynkkaran, Y. Liu, A. Maciejewski, N. Gale, A. Wilson, L. Chin, R. Cummings, D. Le, A. Pon, C. Knox, and M. Wilson, "DrugBank 5.0: A major update to the DrugBank database for 2018," *Nucleic Acids Research*, vol. 46, pp. D1074–D1082, Jan. 2018.

[3] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "MoleculeNet: A benchmark for molecular machine learning," *Chemical Science*, vol. 9, no. 2, pp. 513–530, 2018.

[4] K. Atz, F. Grisoni, and G. Schneider, "Geometric deep learning on molecular representations," *Nature Machine Intelligence*, vol. 3, pp. 1023–1032, Dec. 2021.

[5] S. Abi-Karam, Y. He, R. Sarkar, L. Sathidevi, Z. Qiao, and C. Hao, "GenGNN: A generic FPGA framework for graph neural network acceleration," *arXiv:2201.08475 [cs]*, Jan. 2022.

[6] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón, "Computing graph neural networks: A survey from algorithms to accelerators," *ACM Computing Surveys*, vol. 54, pp. 1–38, Dec. 2022.

[7] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "HyGCN: A GCN accelerator with hybrid architecture," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, (San Diego, CA, USA), pp. 15–29, IEEE, Feb. 2020.

[8] S. Liang, Y. Wang, C. Liu, L. He, H. Li, D. Xu, and X. Li, "EnGN: A high-throughput and energy-efficient accelerator for large graph neural networks," *IEEE Transactions on Computers*, vol. 70, pp. 1511–1525, Sept. 2021.

[9] K. Kiningham, C. Re, and P. Levis, "GRIP: A graph neural network accelerator architecture," *arXiv:2007.13828 [cs]*, July 2020.

[10] K. Kiningham, P. Levis, and C. Ré, "GReTA: Hardware optimized graph processing for GNNs," in *Proceedings of the Workshop on Resource-Constrained Machine Learning (ReCoML 2020)*, Mar. 2020.

[11] J. Li, A. Louri, A. Karanth, and R. Bunescu, "GCNAX: A flexible and energy-efficient accelerator for graph convolutional neural networks," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, (Seoul, Korea (South)), pp. 775–788, IEEE, Feb. 2021.

[12] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt, and M. C. Herbordt, "AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, (Athens, Greece), pp. 922–936, IEEE, Oct. 2020.

[13] B. Zhang, H. Zeng, and V. Prasanna, "Hardware acceleration of large scale GCN inference," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, (Manchester, United Kingdom), pp. 61–68, IEEE, July 2020.

[14] B. Zhang, R. Kannan, and V. Prasanna, "BoostGCN: A framework for optimizing GCN inference on FPGA," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, (Orlando, FL, USA), pp. 29–39, IEEE, May 2021.

[15] T. Geng, C. Wu, Y. Zhang, C. Tan, C. Xie, H. You, M. Herbordt, Y. Lin, and A. Li, "I-GCN: A graph convolutional network accelerator with runtime locality enhancement through islandization," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, (Virtual Event Greece), pp. 1051–1063, ACM, Oct. 2021.

[16] A. Auten, M. Tomei, and R. Kumar, "Hardware acceleration of graph neural networks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, (San Francisco, CA, USA), pp. 1–6, IEEE, July 2020.

[17] X. Chen, Y. Wang, X. Xie, X. Hu, A. Basak, L. Liang, M. Yan, L. Deng, Y. Ding, Z. Du, and Y. Xie, "Rubik: A hierarchical architecture for efficient graph neural network training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, pp. 936–949, Apr. 2022.

[18] H. Zeng and V. Prasanna, "GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, (Seaside CA USA), pp. 255–265, ACM, Feb. 2020.

[19] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 13260–13271, Curran Associates, Inc., 2020.

[20] P. Veličković, "Message passing all the way up," *arXiv:2202.11097 [cs, stat]*, Feb. 2022.

[21] D. Beaini, S. Passaro, V. Létourneau, W. Hamilton, G. Corso, and P. Lió, "Directional graph networks," in *Proceedings of the 38th International Conference on Machine Learning*, pp. 748–758, PMLR, July 2021.

[22] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 22118–22133, Curran Associates, Inc., 2020.