GNNHLS: Evaluating Graph Neural Network Inference via High-Level Synthesis

Chenfeng Zhao
Zehao Dong
Yixin Chen
Xuan Zhang
Roger D. Chamberlain

Chenfeng Zhao, Zehao Dong, Yixin Chen, Xuan Zhang, and Roger D. Chamberlain, "GNNHLS: Evaluating Graph Neural Network Inference via High-Level Synthesis," in *Proc. of IEEE 41st International Conference on Computer Design* (ICCD), November 2023, pp. 574-577.
DOI: 10.1109/ICCD58817.00092

Dept. of Computer Science and Engineering
Washington University in St. Louis

Dept. of Electrical and Systems Engineering
Washington University in St. Louis

GNNHLS: Evaluating Graph Neural Network Inference via High-Level Synthesis

Chenfeng Zhao, Zehao Dong, Yixin Chen, Xuan Zhang, Roger D. Chamberlain
McKelvey School of Engineering
Washington Univ. in St. Louis
{chenfeng.zhao,zehao.dong,ychen25,xuan.zhang,roger}@wustl.edu

***Abstract*—We present GNNHLS, an open-source framework to comprehensively evaluate GNN inference acceleration on FPGAs via HLS, containing a software stack for data generation and baseline deployment and FPGA implementations of 6 well-tuned GNN HLS kernels. Evaluating on 4 graph datasets with distinct topologies and scales, the results show that GNNHLS achieves up to $50.8\times$ speedup and $423\times$ energy reduction relative to the CPU baselines. Compared with the GPU baselines, GNNHLS achieves up to $5.16\times$ speedup and $74.5\times$ energy reduction.**

***Index Terms*—field-programmable gate arrays, graph neural networks, high-level synthesis**

I. Introduction

Machine learning (ML) on graphs has experienced a surge of popularity recently since traditional ML models, which are designed to process Euclidean data with regular structures, are ineffective at performing prediction tasks on graphs. Due to their simplicity and superior representation learning ability, Graph Neural Networks (GNNs) [4], [9], [13], [14], [16] have achieved impressive performance on various graph learning tasks, such as node classification, graph classification, etc.

To implement GNNs, a set of widespread libraries, such as PyTorch Geometric (PYG) and Deep Graph Library (DGL), are built upon ML frameworks (e.g., PyTorch) targeting both CPUs and GPUs. However, the performance and energy consumption of GNN execution is hindered by both hardware platforms and software frameworks: (1) Distinct from traditional NNs, GNNs combine the irregular communication-intensive patterns of graph processing and the regular computation-intensive patterns of NNs. (2) These frameworks execute functions sequentially, which leads to extra memory accesses and implicit barriers for intermediate results.

Field-Programmable Gate Arrays (FPGAs) are an attractive approach to GNN inference acceleration. FPGAs' massive fined-grained parallelism provides opportunities to exploit GNNs' inherent parallelism. They also deliver better performance per watt than general-purpose computing platforms. In addition, FPGAs' reconfigurability and concurrency provide great flexibility to solve the challenges of hybrid computing patterns. Most of the prior works investigating FPGAs focus on accelerating a specific GNN model implemented using Hardware Description Languages (HDL). AWB-GCN [6] proposes a GCN accelerator to solve the workload imbalance problem. BoostGCN [15] proposes a graph partition algorithm in a preproessing step to address workload imbalance issues. High-Level Synthesis (HLS) tools have also been used to create GNN kernels using popular languages such as C/C++. With the help of HLS, development time is substantially shortened relative to HDL designs. Lin et al. [11] proposes an HLS-based accelerator for GCN with separated sparse-dense matrix multiplication units and dense matrix multiplication units which are connected by shared memory and execute sequentially. GenGNN [1] proposes a framework to accelerate GNNs for real-time requirements where the whole graph and corresponding intermediate results are stored on the FPGA. Despite promising results, this work is limited to small-scale graphs with low edge-to-node ratio due to memory usage being proportional to graph scale and feature dimensions.

Distinct from pure software programming, HLS developers need to adopt multiple optimization pragmas and follow certain coding styles to achieve best performance and energy cost. The performance difference between a well-optimized version and a non-optimized version of the same kernel can be two to three orders of magnitude. This invites an open question: *how effectively can modern HLS tools accelerate GNN inference?*

In this paper, we introduce GNNHLS[1], an open-source framework for comprehensive evaluation of GNN kernels on FPGAs via HLS. GNNHLS contains a software stack extended from a prior GNN benchmark [5] based on PyTorch and DGL for input data generation and conventional platform baseline deployments (i.e., CPUs and GPUs). It also contains six well-optimized general-purpose GNN applications: Graph Convolutional Network (GCN) [9], GraphSage (GS) [7], Graph Isomorphism Network (GIN) [14], Graph Attention Network (GAT) [13], Mixture Model Networks (MoNet) [12], and Gated Graph ConvNet (GatedGCN)) [2]. These kernels can be classified into 2 classes: (1) isotropic GNNs (GCN, GS, and GIN) in which every neighbor contributes equally to the update of the target vertex, and (2) anisotropic GNNs (GAT, MoNet, and GatedGCN) in which edges and neighbors contribute differently to the update due to the adoption of operations such as attention and gating mechanisms. In this paper, we make the following contributions:

- We propose GNNHLS, a framework to evaluate GNN inference acceleration via HLS, containing: (a) a software stack based on PyTorch and DGL for data generation

[1]Released as a benchmark suite [17] and also available at https://github.com/ChenfengZhao/GNNHLS

2576-6996/23/$31.00 ©2023 IEEE
DOI 10.1109/ICCD58817.2023.00092

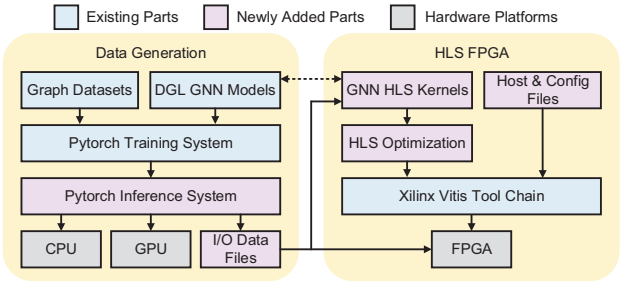

Fig. 1. Diagram of the GNNHLS framework.

and baseline deployment, and (b) FPGA implementation including 6 well-tuned GNN HLS kernels which can also be used as benchmarks.

- We provide a comprehensive evaluation of our GNN HLS implementations on 4 graph datasets, assessing both performance improvement and energy reduction.

Our results show that GNNHLS provides up to $50.8\times$ speedup and $423\times$ energy reduction relative to the multicore CPU baselines. Compared with the GPU baselines, GNNHLS achieves up to $5.16\times$ speedup and $74.5\times$ energy reduction.

II. Framework Description

A. GNNHLS Overview

The GNNHLS framework, as depicted in Figure 1, comprises two primary components: data generation and HLS FPGA. The former is designed to generate input and output files and measure baselines on a CPU and a GPU, while the latter is designed to implement the optimized HLS applications on an FPGA. The data generation component mainly consists of the training system and the inference system.

The HLS FPGA component implements the GNN kernels on the FPGA. These kenels match the functionality of the DGL baselines and are optimized with several optimization techniques [3]. The optimized HLS kernels are compiled by Vitis and executed on the FPGA. Detailed descriptions of the optimized GNN HLS kernels, experimental methodology, and computation characterization are all included in the supplementary material [17].

III. Experimental Methodology

Datasets: Table I shows the graph datasets used in our evaluation. All these graphs are from Open Graph Benchmark [8] and have a wide range of fields and scales. These graphs represent two classes of graphs with distinct topologies: MH and MT consist of multiple small dense graphs, while AX and PT each consist of one single sparse graph. The maximum and average degree shown in Table I indicates their varying distributions ranging from regular-like to powerlaw-like.

Evaluation methods: To perform evaluation, we use a Xilinx Alveo U280 FPGA card, provided by the Open Cloud Testbed [10], to execute the HLS kernels. We compare our HLS implementation with CPU and GPU baselines with PyTorch and the highly-optimized DGL library.

TABLE I
Graph datasets.

Dataset	Node #	Edge #	Max. Deg.	Avg. Deg.
OGBG-MOLTOX21 (MT)	145459	302190	6	2.1
OGBG-MOLHIV (MH)	1049163	2259376	10	2.2
OGBN-ARXIV (AX)	169343	1166243	13155	6.9
OGBN-PROTEINS (PT)	132534	79122504	7750	597.0

TABLE II
Resource Utilization of HLS GNN models.

	Frequency	LUT	FF	BRAM	DSP
GCN	250 MHz	264485	413197	41	2880
GS	204 MHz	253608	358722	33	2766
GIN	190 MHz	278251	421915	55	3264
GAT	255 MHz	168559	248424	81	1718
MN	250 MHz	289208	428917	212	2236
GGCN	270 MHz	151497	235484	124	1036

IV. Evaluation

A. Resource Utilization

We first examine the resource utilization and clock frequency after place & route. FPGA resources include look-up tables (LUT), flip-flops (FF), BRAM, and digital-signal-processors (DSP). Table II shows these results. Among these kernels, GraphSage achieves a low frequency due to some critical paths unresolvable by the tool. In addition, we observe that the resources on the FPGA are not over-utilized.

B. Performance

We next examine the performance improvement by showing the overall speedup, defined as the execution time of the GNN HLS kernels relative to CPU-DGL (using all 10 cores on the CPU), in Figure 2. Focusing on the first and third bar of each kernel, we observe that the speedup of our HLS kernels ranges from $0.47\times$ to $50.8\times$ over the multi-core CPU baselines.

Among isotropic GNN kernels, GCN achieves better performance than GraphSage and GIN, ranging from $1.08\times$ to $1.98\times$ because its simpler structure enables us to create two CUs. In contrast, we can only create one CU for GraphSage and GIN because of their complex structure and heavy resource usage. In addition, we observe that the performance of GraphSage and GIN are close. Thus, we conclude that the distinction on the structure of these two GNN models will not substantially affect HLS implementation results.

Among anisotropic kernels, MoNet achieves the highest performance improvement, ranging from $6.04\times$ to $50.8\times$ due to: (1) its single pipeline structure with computation order optimization where the node-wise operations are placed behind the edge-wise operations, and (2) well-designed MHVMM modules with lower II, reduced from $O(dk)$ to $O(d+k)$. In spite of the 2-pipeline structure of GAT, we observe that it still achieves $4.31\times$ to $6.61\times$ speedup. In addition, since the feature size of GatedGCN is smaller, leading to more performance improvement for CPU baselines with time complexity of $O(d^2)$, its speedup is not comparable to other anisotropic kernels, ranging from $0.5\times$ to $1.16\times$.

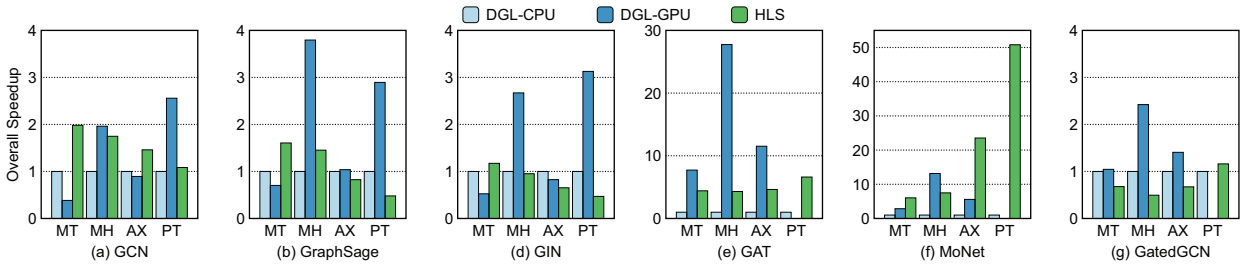

Fig. 2. Speedup of HLS kernels relative to DGL-CPU. Higher is better.

Turning our attention to how the performance benefit of HLS implementations varies across graph datasets, we observe that the speedup of isotropic kernels relative to DGL-CPU on regular-like graphs (i.e., MT and MH) is higher than powerlaw-like graphs (i.e., AX and PT) because (1) the edge-wise operations are less computation-intensive than node-wise operations in these kernels, making the baselines more computationally efficient on powerlaw-like graphs containing more edges than nodes; and (2) the edge-wise aggregation operations in HLS implementations are executed sequentially without leveraging edge-level parallelism, making these HLS kernels less computationally efficient for powerlaw-like graphs. Distinct from isotropic kernels, the speedup of anisotropic kernels on powerlaw-like graphs is higher than regular-like graphs because the edge-wise operations of these kernels are more computation-intensive than isotropic kernels, making baselines less efficient on powerlaw-like graphs.

Focusing on the second and the third bar, we observe that DGL-GPU outperforms HLS implementations in many cases, due to the high-performance fixed-function accelerators in the GPU. The speedup of HLS kernels relative to the GPU baselines ranges from $0.13\times$ to $5.16\times$. Note that GPU results for GAT, MN, and GGCN on PT cannot be obtained because of out of memory (OoM) limitations. In spite of the promising GPU performance, there are still some drawbacks of GPU compared with HLS implementations. For the execution of isotropic GNN models, DGL-GPU achieves lower speedup than HLS on small-scale graphs such as MT and AX. It is speculated that the GPU is designed to achieve high throughput in the cost of latency which plays a more important role for small-scale graphs than large-scale graphs. In addition, compared with HLS implementations on FPGA, GPU is also not suitable for the execution of anisotropic GNN models on large-scale, especially powerlaw-like graphs (e.g., PT) due to: (1) the non-trivial memory footprint caused by its sequential execution, storing intermediate results of edge-wise operations, and (2) insufficient memory capacity on the GPU. That is why we failed to execute anisotropic GNNs on PT with GPU. This is addressed by the HLS implementations' pipeline structure not storing the intermediate results.

Since GenGNN [1] also discusses 3 of the GNN models included in this paper (GCN, GIN, and GAT), we can make a limited comparison of our GNN HLS implementations with theirs. The two are not directly comparable for a number of reasons: (1) the feature dimensions of our GNN HLS kernels are higher, (2) we use off-chip memory instead of on-chip memory, (3) our general-purpose GNN HLS kernels focus more on throughput rather than real-time latency, and (4) the FPGAs are from the same family, but are not same part. The performance of our HLS kernels exceeds that of GenGNN, achieving overall speedup of $35\times$, $5\times$, and $6\times$ over GCN, GIN, and GAT, on MT, respectively.

TABLE III
EXECUTION TIME OF VARIOUS OPTIMIZATION TECHNIQUES FOR GRAPHSAGE ON MH.

Optimizations	Execution Time (s)	Speedup
No Pragmas	129.59	1.00×
Dataflow	65.11	1.99×
Loop Unroll	11.11	11.7×
Vectorization	4.44	29.2×
Split Loops	0.98	132×

C. Optimization Techniques

As described in Section II, we apply multiple optimization techniques to the HLS kernels. In order to evaluate the efficacy of these techniques, we use GraphSage on MT as a case study. Table III presents the execution time of GraphSage with the combined impact of optimization techniques applied. The reported execution time of each technique represents the effect of both the current technique and above techniques listed in the table. In the table, No Pragma means we don't intentionally apply any pragmas to the HLS code, except for those automatically applied by Vitis (i.e., Pipeline, Loop Merge, and Memory optimizations). Dataflow denotes that we apply dataflow pragma and FIFO streams to exploit the task-level parallelism of each application. Loop Unroll means we apply loop unroll pragmas to completely or partially unroll for loops, keeping II as low as possible while exploiting instruction parallelism. Vectorization means using vector data types to widen the width of FIFO streams and corresponding operations to decrease the cost of FIFO accesses. Split Loops means splitting the outer-most node loop and putting it inside each function connected by streams to further optimize FIFO properties inferred from loop indices.

We observe that Loop Unroll achieves the highest performance improvement. Therefore, exploiting instruction paral-

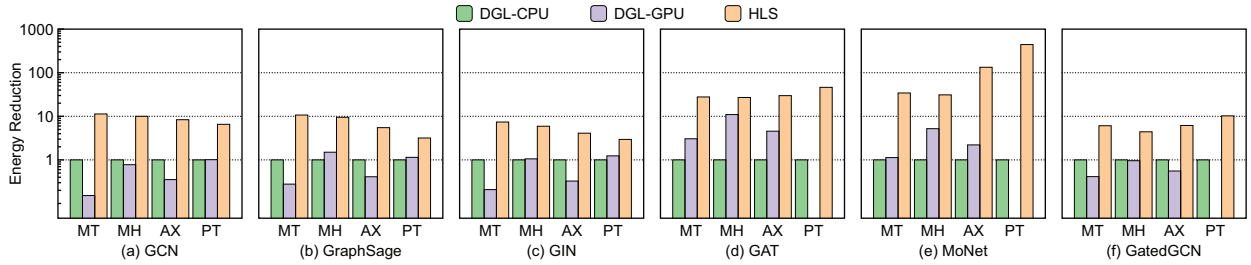

Fig. 3. Energy consumption reduction of HLS kernels relative to DGL-CPU (logarithmic scale). Higher is better.

lelism is still the primary choice for GNN HLS optimization. In order to further improve performance, exploiting task-level parallelism is necessary. Focusing on the first and second row in the table, we observe that only performing the dataflow pragma and streams in a naive way obtains $1.99\times$ performance improvement. By applying Vectorization and Split Loops as complementary techniques of Dataflow, performance is further improved by $2.5\times$ and $3.9\times$, respectively. After applying all the optimization techniques together we observe that the performance of GraphSage is improved by $132\times$.

D. Energy Consumption

We next present a quantitative analysis of the energy consumption. Figure 3 displays the energy reduction of both DGL-GPU and HLS implementations relative to DGL-CPU. Energy reduction is calculated as the energy consumption of DGL-GPU or HLS divided by that of DGL-CPU. Examining the final bar of each application and dataset, we observe that HLS implementations consume less energy than CPU and GPU baselines in all cases, the energy reduction ranging from $2.95\times$ to $423\times$ and from $2.38\times$ to $74.5\times$, respectively. It is because of the low power of FPGA logic, low clock frequency, and efficient pipeline structure of HLS implementations.

Focusing on the first and last bar, we observe a similar tendency in energy reduction as in performance: for isotropic GNN models, denser graphs result in lower energy reduction, whereas for anisotropic GNN models, denser graphs result in higher energy reduction. This leads us to conclude that improving GNN applications generally will require some degree of graph topology awareness.

V. Conclusions

We propose GNNHLS, an open-source framework to comprehensively evaluate GNN inference acceleration on FPGAs via HLS. GNNHLS consists of a software stack for data generation and baseline deployment, and 6 well-tuned GNN HLS kernels. We evaluate the HLS kernels on 4 graph datasets with various topologies and scales. Results show up to $50.8\times$ speedup and $423\times$ energy reduction relative to the multi-core CPU baselines. Compared with GPU baselines, GNNHLS achieves up to $5.16\times$ speedup and $74.5\times$ energy reduction. GNNHLS has been released as a benchmark suite [17].

Acknowledgment

This work is supported by NSF under grants CNS-1739643 and CNS-1763503 and a gift from BECS Technology, Inc. The authors are grateful for the use of the Open Cloud Testbed [10] as an experimentation platform.

References

[1] 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 preprint arXiv:2201.08475*, 2022.
[2] X. Bresson and T. Laurent, "Residual gated graph convnets," *arXiv preprint arXiv:1711.07553*, 2017.
[3] J. de Fine Licht, M. Besta, S. Meierhans, and T. Hoefler, "Transformations of high-level synthesis codes for high-performance computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1014–1029, 2020.
[4] Z. Dong, W. Cao, M. Zhang, D. Tao, Y. Chen, and X. Zhang, "CktGNN: Circuit graph neural network for electronic design automation," *arXiv preprint arXiv:2308.16406*, 2023.
[5] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *Journal of Machine Learning Research*, vol. 23, 2020.
[6] 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 *Proc. of 53rd Int'l Symp. on Microarchitecture*, 2020, pp. 922–936.
[7] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.
[8] 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," *Adv. Neural Inf. Process. Syst.*, vol. 33, 2020.
[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. of Int'l Conf. on Learning Rep.*, 2017.
[10] M. Leeser, S. Handagala, and M. Zink, "FPGAs in the cloud," *Computing in Science & Engineering*, vol. 23, no. 6, pp. 72–76, 2021.
[11] Y. C. Lin, B. Zhang, and V. Prasanna, "GCN inference acceleration using high-level synthesis," in *Proc. of IEEE High Performance Extreme Computing Conf.*, 2021.
[12] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, 2017, pp. 5115–5124.
[13] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. of Int'l Conf. on Learning Rep.*, 2017.
[14] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. of Int'l Conf. on Learning Rep.*, 2019.
[15] B. Zhang, R. Kannan, and V. Prasanna, "BoostGCN: A framework for optimizing GCN inference on FPGA," in *Proc. of 29th Int'l Symp. on Field-Programmable Custom Computing Machines*, 2021, pp. 29–39.
[16] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. of AAAI Conf. on Artificial Intelligence*, vol. 32, no. 1, 2018.
[17] C. Zhao, Z. Dong, Y. Chen, X. Zhang, and R. D. Chamberlain, "Graph Neural Network High-Level Synthesis Benchmark Suite V1," https://doi.org/10.7936/6RXS-103645, Sep. 2023.