Characterizing and Understanding GCNs on GPU

Mingyu Yan, Zhaodong Chen, Lei Deng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie

Abstract—Graph convolutional neural networks (GCNs) have achieved state-of-the-art performance on graph-structured data analysis. Like traditional neural networks, training and inference of GCNs are accelerated with GPUs. Therefore, characterizing and understanding the execution pattern of GCNs on GPU is important for both software and hardware optimization. Unfortunately, to the best of our knowledge, there is no detailed characterization effort of GCN workloads on GPU. In this letter, we characterize GCN workloads at inference stage and explore GCN models on NVIDIA V100 GPU. Given the characterization and exploration, we propose several useful guidelines for both software optimization and hardware optimization for the efficient execution of GCNs on GPU.

Index Terms—Graph convolutional neural networks, characterization, execution pattern, GPU

1 Introduction

In recent years, Graph Convolutional Neural Networks (GCNs) that operate on graph-structured data have achieved state-of-the-art performance on tasks like node classification, link prediction, and recommendations, etc. GCNs have become a new workload family member in data-centers [1], [2]. Like traditional neural networks, training and inference of GCN models are accelerated with Graphics Processing Units (GPUs) to achieve an order of magnitude lower latency [1]. Therefore, characterizing the execution pattern of GCNs on GPUs is important for both software and hardware optimization for GCNs.

To the best of our knowledge, there is no characterizing effort of GCNs on GPU. Popular GCN models usually contain two major execution phases with distinct execution pattern: *Aggregation* and *Combination*. The former phase aggregates the feature vectors of the neighbor nodes like graph processing, so it exhibits a similar irregular execution pattern. The latter phase updates the feature vectors with multi-layer perceptrons (MLPs), so it has alike regular patterns with traditional neural networks. Nevertheless, GCNs have shown several new features that make their execution patterns differ from traditional workloads, so conclusions in existing characterization studies on graph processing and neural networks cannot be directly inferred in GCNs.

To understand the computation and memory accessing pattern of GCNs, we profile and analyze the inference stage of several GCN models on popular benchmarks with NVIDIA GPU V100, and the results are compared with traditional graph processing and MLP workloads. Besides, we also conduct an exploration of how configurations like dimension size influence the execution time. Our key observations and insights toward architecture design are summarized below.

- *Comparison to Graph Processing:* 1) High-degree spatial data locality and parallelism exist intra vertex; 2) Only inter-warp atomic collision exists; 3) L2 cache hit ratio in *Aggregation* phase is extremely lower than graph processing due to the long reuse distance of vertex data.
- *Comparison to MLP-based Neural Network:* 1) The parameters of MLP exhibit extremely high reusability inter vertex; 2) High-degree parallelism exists inter vertex.
- *Overall Execution:* 1) Hybrid execution pattern exists in GCNs; 2) Execute *Combination* phase ahead of *Aggregation* phase helps reduce data access and computation of *Aggregation* phase; 3) A dataflow exists inter phase for each vertex.
- *Exploration:* 1) The execution time of *Combination* is almost proportion to input feature length, while the execution time of *Aggregation* phase in various length are almost the same since it is independent on the length of input feature vector; 2) Both the execution time for *Aggregation* phase and *Combination* phase are almost proportion to output feature length; 3) There are sweet spots for the execution of *Combination* phase in terms of the length of input and output feature.

Given the characterization and exploration, we propose useful guidelines as follows for both software framework optimization and hardware optimization for GCNs.

- *Software Optimization Guideline:* 1) A degree-aware feature access scheduling to reuse the vertex with high degree; 2) Vectorizing atomic operation to improve the efficiency of parallelism; 3) An adaptive execution granularity to leverage the inter-phase dataflow and hardware-optimized function.
- *Hardware Optimization Guideline:* 1) A degree- and length-aware replacement policy for Cache to reuse the feature of high-degree vertex and improve memory level parallelism.

2 Background of GCNs

In general, GCNs follow a neighborhood gather scheme. The feature vector of each vertex is updated by recursive aggregation of the feature vectors of neighbor nodes and combination of features via an MLP or a single fully-connected layer [3]. Let $h_v^{(k-1)}$ be the feature vector of vertex v at layer $k-1$, $N(v)$ be the neighbor list of vertex v, and σ be the activation function, we briefly summarize three popular GCN models as follows.

Graph Convolutional Network (GCN) [4]. The propagation rule of GCN at layer k is defined as follows:

$$h_v^{(k)} = \sigma\Big(mean\Big(W^{(k)} h_u^{(k-1)} | u \in \{N(v)\} \cup \{v\}\Big)\Big), \quad (1)$$

where the term $W^{(k)} h_u^{(k-1)}$ (*Combination*) multiplies the feature vector of each vertex by the weight matrix $W^{(k)}$ and then the term $mean$ (*Aggregation*) updates each feature vector with the average of its neighborhood.

Graph Isomorphism Network (GIN) [5]. In GIN-0 introduced in Xu *et al.* (2018) [5], the feature vector of each vertex is updated with

$$h_v^{(k)} = mlp\Big(sum\Big(h_u^{(k-1)} | u \in \{N(v)\} \cup \{v\}\Big)\Big), \quad (2)$$

in which the feature vectors are first aggregated by the summation of neighborhood and then updated with MLP.

GraphSAGE (SAG) [6]. In GraphSAGE, the feature vectors are updated with the same propagation rule of *GCN*, the difference is that while *GCN* updates the feature vectors of all vertexes in the

- *M. Yan is with the SKLCA, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing 100864, China, the University of Chinese Academy of Sciences (UCAS), Beijing 100049, China, and also with the University of California, Santa Barbara, Santa Barbara, CA 93106. E-mail: yanmingyu@ict.ac.cn.*
- *Z. Chen, L. Deng, Y. Xie are with the University of California, Santa Barbara, Santa Barbara, CA 93106. E-mail: {chenzd15thu, leideng, yuanxie}@ucsb.edu.*
- *X. Ye and Z. Zhang are with the SKLCA, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing 100864, China. E-mail: {yexiaochun, zzm}@ict.ac.cn.*
- *D. Fan is with the SKLCA, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing 100864, China, and also with the University of Chinese Academy of Sciences (UCAS), Beijing 100049, China. E-mail: fandr@ict.ac.cn.*

Manuscript received 31 Dec. 2019; accepted 26 Jan. 2020. Date of publication 30 Jan. 2020; date of current version 3 Apr. 2020.
(Corresponding author: Xiaochun Ye.)
Digital Object Identifier no. 10.1109/LCA.2020.2970395

TABLE 1
Configuration of Convolution Layers

	Aggregation Operator & Combination Operator
GCN (GCN)	Mean & MLP: $\|h_u^{(k-1)}\|$–128
GraphSage (SAG)	Mean & MLP: $\|h_u^{(k-1)}\|$–128
GINConv (GIN)	Add & MLP: $\|h_u^{(k-1)}\|$–128–128
PageRank (PGR)	Graph Processing
MLP-MNIST	MLP: 784–128 with batching size 1000

Here $|h_u^{(k-1)}|$ denotes the length of feature vector $h_u^{(k-1)}$.

graph in each iteration, GraphSAGE only update a batch of vertexes along with their 2-hop neighbors in an iteration.

There are three major different features between GCNs with traditional graph processing and neural networks:

1) *Large and Variable Feature Length*. The feature data in graph processing are small, usually one element for each vertex, while the feature vector of each vertex in the *Aggregation* phase of GCNs usually contains hundreds of entries and varies across layers and datasets.
2) *Parameters Shared by Vertices*. In traditional MLP-based neural network, to classify one sample, only one feature vector is forward through the MLP, and the parameters in the MLP are not shared. However, in node classification of GCNs with k layers, the feature vectors of all k-hop neighbours are forwarded; In graph classification, the feature vectors of all vertexes are required. As a result, in GCNs, the parameters in the MLP can be fully shared by each feature vector.
3) *Alternative Execution*, the two phases are executed alternatively until the final result is produced.

3 Evaluation Setup

Benchmark. Tables 2 and 1 provide the information of the benchmark GCN models and graph datasets used in our evaluation. For GCNs, we select three advanced models: Graph Convolutional Network (GCN) [4], Graph Isomorphism Network (GIN) [5], and GraphSAGE (SAG) [6]. For clarity, we evaluate the first graph-convolutional layer of each model on popular datasets including Cora, Citeseer, Pubmed, and Reddit. For classical graph processing, we run PageRank on the Reddit and LiveJournal dataset. For traditional MLP, we test a single fully-connected layer on MNIST. Notably, we mainly focus on the inference stage rather than training.

Profiling Platform. The GCN models are implemented with the state-of-the-art GPU-based software framework for GCNs: *PyTorch Geometric* [8]. The PageRank is implemented with Gunrock [9]. All the workloads are profiled on single NVIDIA GPU V100 with NVIDIA NVProf and averaged among 5 iteration.

4 Observation and Analysis

This section is organized as follows. First, we present an overview of our profiling result. Then, we characterize dominant kernels in *Aggregation* and *Combination* phase and compare them with traditional workloads. At last, we explore the impact of feature length on execution time.

TABLE 2
Datasets Information [4], [7]

Dataset	#Vertex	Feature Len.	#Edge
Cora (CR)	2,708	1,433	5,429
Citeseer (CS)	3,327	3,703	4,732
Pubmed (PB)	19,717	500	44,338
Reddit (RD)	232,965	602	11,606,919
LiveJournal (LJ)	4,847,571	1	68,993,773

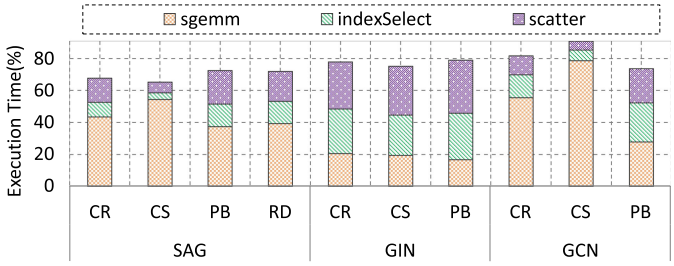

Fig. 1. Execution time breakdown on V100 GPU.

4.1 Overview of Profile

Execution Time Breakdown. Fig. 1 illustrates the execution time breakdown of major kernels that occupy most of execution time on GPU.

The *sgemm* kernels multiplies the weight matrix with feature vectors to perform *Combination*. The *indexSelect* kernel and *scatter* kernel execute *Aggregation* function for all vertices. Specifically, the *indexSelect* kernel uses the neighbor ID to select the neighbor's feature vector of each vertex, and then uses these feature vectors to build a dense feature matrix for the input of *scatter* kernel. Each thread in the *scatter* kernel executes aggregation operator for each element in a neighbor's feature vector.

As illustrated in Fig. 1, the above three kernels take up 65 to 90 percent execution time in different configurations. The portion of execution time that each kernel takes is determined by the sequence of *Combination* and *Aggregation* as well as the length of feature vectors. Specifically, GIN executes *Aggregation* phase first while the other two execute *Combination* phase first. While the scale of *Combination* is similar among the three models, the feature vector length in GCN and SAG are significantly reduce by *Combination*, so their *Aggregation* phase take much fewer time than GIN. In terms of dataset, the *Combination* takes more execution time in the datasets with longer feature length, i.e., CS.

4.2 Analysis of *Aggregation* Phase

Here, we provide detailed analysis of the *Aggregation* phase in SAG and compare it with PGR on the RD and LJ datasets.

High-degree spatial data locality exists in the access to feature data. This locality derives from the access to each long-length feature vector. Figs. 2a and 2b respectively illustrate that *L1 cache hit ratio* and *L1 cache to multiprocessor throughput(GB/sec)* of *Aggregation* phase are higher than that in graph processing on the same dataset (i.e., RD). Besides, Fig. 2c depicts that the value of *Memory Throttle* of PGR is extremely higher than *Aggregation* phase (39.27 percent versus 0.225 percent). It means a large number of pending memory operations prevent further forward progress in the micro-architecture pipeline on PGR due to the fine-grained and irregular data access to feature data. However, the accesses to long-length feature data in *Aggregation* phase can be reduced by combining several memory transactions into one. These three results demonstrate that *Aggregation* phase has an extra spatial locality in the access to feature data compared to graph processing.

High-degree parallelism exists intra vertex. Except for the inter-vertex and inter-edge parallelism as in graph processing, *Aggregation* phase possesses a new kind of parallelism, the intra-vertex parallelism. This parallelism comes from the element-wise aggregation of each neighboring vertex feature vector. As a result, the *Achieved Occupancy* and *Issue Slot Utilization* of *Aggregation* phase are higher than that in graph processing as shown in Figs. 2d and 2e.

Only inter-warp atomic collision exists in parallelism exploitation. Atomic collision refers to scenarios where multiple threads try to read-modify-write the same data word simultaneously. To guarantee the atomicity, these updates from different threads will be serialized.

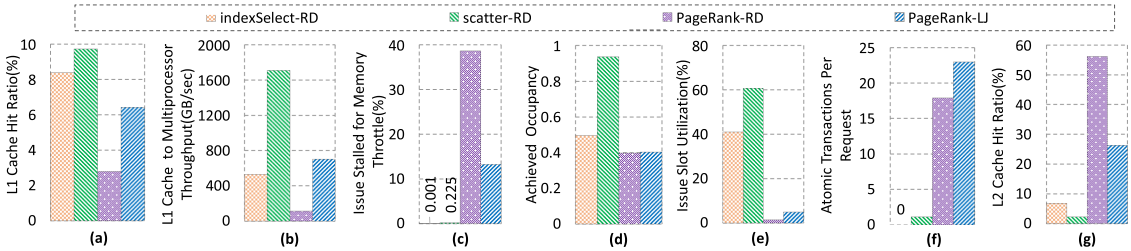

Fig. 2. The profiling result of *Aggregation* phase.

There are two collisions existing in GPUs: inter-warp collision and intra-warp collision. Since each thread inside a warp processes one of the consecutive feature elements of neighboring feature vector, there is almost no intra-warp collision in *Aggregation* phase. In contrary, in PGR, each thread processes a random vertex with a single feature element. It means that the threads intra or inter warp may update the same vertex, which causes inter-warp collision or intra-warp collision. Thus, *Atomic Transactions Per Request* is 1.1 in *Aggregation* phase (Fig. 2f), smaller than the 17.9 in PGR.

Aggregation phase exhibits lower reuse of neighbors' feature data than that in graph processing. Although the graph traversal in *Aggregation* phase is same to that in PGR, *L2 cache hit ratio* in *Aggregation* phase is extremely low. Fig. 2g shows *L2 cache hit ratio* of *Aggregation* phase is only 6.9 percent while that is 56.2 percent in PGR, even although they process the same graph. The reason is as follows. The vertex data in graph processing is only one element, while the feature vector contains hundreds element in *Aggregation* phase. As a result, L2 cache can hold many vertex data in PGR, which enlarges the opportunity to reuse the vertex data of the shared neighbor. However, L2 cache can only contain smaller amount of feature vectors in *Aggregation* phase, which results in longer data reuse distance of feature vector than that in graph processing. Therefore, *Aggregation* phase exhibits low reuse of neighbors' feature data.

4.3 Analysis of *Combination* Phase

Here, we provide detailed analysis of the *Combination* phase on SAG model with RD dataset and compare it with MLP-MNIST with batch size 1,000, such that both model has similar batch size and input feature vector length.

The parameters of neural network exhibits extremely high reusability inter vertex. To classify one handwritten number in MNIST with MLP, only a single feature vector need to be forwarded. On the contrary, in node classification tasks, the feature vectors of all the neighbors within k-hop of the target node need to be processed. In graph classification tasks, all the vertex in the graphs should be processed. When processing multiple feature vectors, the parameters in the model can be shared across all the features. As a result, *Combination* phase presents more reusability on the parameters of neural network.

High-degree parallelism exists inter vertex. As mentioned before, the amount of feature vectors processed in *Combination* phase is much larger than traditional neural networks, which introduces more parallelism. As a result, it is more capable of feeding up the thousands of parallel floating-point units in GPU and hiding the latency to memory. As shown in Figs. 3a and 3b, *Multiprocessor Activity(%)* and *Executed IPC* of *Combination* phase are 98.885 percent and 1.8, more than 76.829 percent and 1.3 respectively, the values of MLP-MNIST.

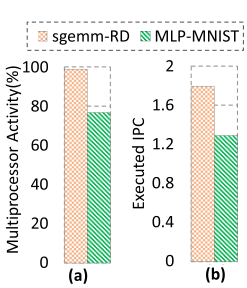

Fig. 3. The result of *Combination* phase.

4.4 Analysis of Overall Execution

Here, we analysis the overall execution on SAG model.

Hybrid execution patterns exist. In *Aggregation* phase, the *Computation Unit Utilization* is only 50 percent and the *Executed IPC* is only 1.78 on average as shown in Table 3. The aggregation heavily relies on the graph structure so that it is obstructed by irregularity [7] and load-load data dependency chain [10]. Therefore, it is mainly stalled for *Data Request* and *Execution Dependency* as depicted in Fig. 4. The irregularity also leads to low *L2 Cache Hit Rate* (6.87 percent) and high *DRAM Byte per Operation* (2.35). In contrary, the *Combination* phase achieves 90 percent *Computation Unit Utilization* and 2.49 *Executed IPC*. The intensive Float-Point calculations well hide the data access latency, and the stalls are issued majorly for *Pipe Busy* and *Not Selected*, which is due to the limited number of computation units. The regular execution pattern leads to high spatial and temporal data locality, the *L2 Cache Hit Rate* is 82.5 percent and *DRAM Byte per Operation* is as low as 0.01. As a result, while the *Aggregation* phase is memory bound with irregular data access pattern and low data reusability, the *Combination* phase is computation bound with regular data access pattern and high data reusability.

Execute Combination phase ahead of Aggregation phase helps reduce data access and computation of Aggregation phase. While the feature length in RD is 602, the *Combination* phase usually reduces the dimension to 128 by a factor of $4.7\times$. Therefore, in the *Aggregation* phase, the

TABLE 3
Characterization of Hybrid Execution Patterns on RD

	Aggregation	*Combination*
Computation Unit Utilization	50%	90%
Executed IPC	1.78	2.49
L2 Cache Hit Rate	6.87%	82.5%
DRAM Byte per Operation	2.35	0.01
Execution Bound	Memory	Compute
Data Access Pattern	Irregular	Regular
Data Reusability	Low	High

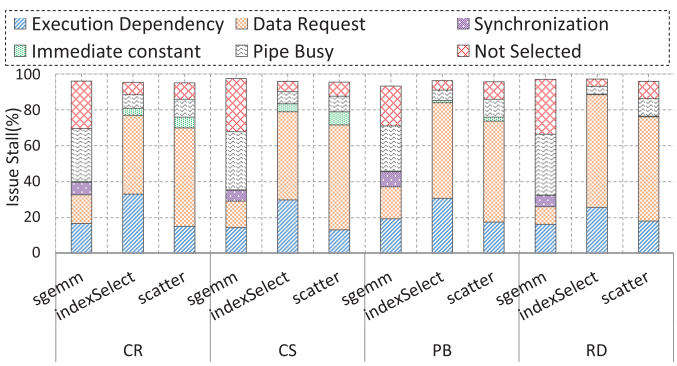

Fig. 4. Percentages of issue stall reasons on SAG model.

TABLE 4
Impact of the Execution Flow on *Aggregation* Phase

	Com → Agg	*Agg → Com*	*Reduction*
Data Accesses (bytes)	568,064,375	2,698,865,170	4.75×
Computations (Operations)	231,995,186	1,096,220,688	4.72×
Execution Time (ms)	1.12	5.34	4.76×

data access to neighbor's feature vector becomes less and the computation for the aggregation of each neighbor also becomes less. Table 4 illustrates the reduction of data accesses and computations in *Aggregation* phase, up to 4.75× and 4.72× respectively. Moreover, the performance achieves 4.76× improvement.

A dataflow exists inter phase in GCNs for each vertex. The result of each vertex in *Aggregation* phase is taken as the input of *Combination* phase for the transformation of each vertex. It indicates that a vertex is able to start the execution in *Combination* phase after this vertex completes its aggregation. Therefore, an inter-phase dataflow exists in GCNs for each vertex. However, to leverage the hardware-optimized functions, the implementation of GCNs on GPU misses this inter-phase dataflow. As a result, many unnecessary data accesses and data addressing computations are introduced.

4.5 Exploring GCN Model

Here, we explore the new features of GCNs on SAG model with RD dataset. As SAG executes *Combination* phase ahead of *Aggregation* phase, the execution time of *Combination* phase is determined by the length of both input and output feature vector, while *Aggregation* is only determined by the length of output feature vector.

Various Length of Input Feature Vector. As illustrated in Fig. 5a, the execution time of *Combination* phase is almost proportion to input feature length. An interesting observation is that there are sweet spots when the input dimension is the index of 2, i.e., 256.

Various Length of Output Feature Vector. As illustrated in Fig. 5b, the execution time of *Aggregation* phase increases linearly with the output feature length. On the other hand, the *Combination* phase is insensitive to the output feature size when the feature length is smaller than 64, which is due to the redundant computational resources. Besides, the sweet spot still exists when the output dimension is the index of 2.

5 Architectural Guidelines

5.1 Software Optimization Guideline

Degree-Aware Feature Access Scheduling. Real-world graphs possess well-connected regions where relatively few vertices share edges with many common neighbors. It indicates that the vertices with large degree exhibits high reusability on their feature data. Thus, an online data access scheduling can leverage that to shorten the reuse distance.

Vectorizing Atomic Operation. To improve the parallelism efficiency, vectorizing atomic operation is available for *Aggregation* to reduces the atomic overhead in GPU since only inter-warp collision exits in GCNs.

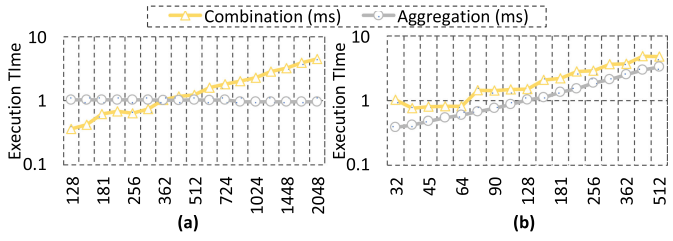

Fig. 5. Exploration on the length of input feature vector (a) and output feature vector (b).

Adaptive Execution Granularity. Leveraging inter-phase dataflow is an excellent opportunity to overlap the memory-bound *Aggregation* phase and computation-bound *Combination* phase. Meanwhile, it is also important to leverage the architecture's advantage. Therefore, an appropriate or adaptive granularity for execution can achieve a better trade-off.

5.2 Hardware Optimization Guideline

Degree- and Length-Aware Replacement Policy. To ease the programmer efforts and improve data reuse, L2 Cache can be modified to equip a degree- and length-aware replacement policy. This policy can replace the vertex feature by aware of its degree, which indicates its reusability. Besides, it can replace the whole vertex feature vector in a time since all the elements in vector are used together. This way helps fire many requests at the same time to exploit the high bandwidth memory.

6 Conclusion

In this work, we characterize and explore an emerging application GCNs on NVIDIA V100 GPU. The characterization results can help programmers understand the execution pattern of GCNs. We also believe the observations made in this paper will provide useful guidance to enable future architecture and system research for GCNs.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China 61732018 and US National Science Foundation 1725447.

References

[1] A. Lerer *et al.*, "PyTorch-BigGraph: A large-scale graph embedding system," in *Proc. 2nd SysML Conf.*, 2019.
[2] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," 2018, *arXiv:1806.01261*.
[3] Y. Hongxia, "AliGraph: A comprehensive graph neural network platform," in *Proc. 25th ACM SIGKDD Int. Conf. Know. Discovery Data Mining*, 2019, no. 2, pp. 3165–3166, doi: 10.1145/3292500.3340404.
[4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.
[5] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," 2018, *arXiv: 1810.00826*.
[6] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.
[7] M. Y. Yan *et al.*, "Alleviating irregularity in graph analytics acceleration: A hardware/software co-design approach," in *Proc. 52th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2019, pp. 615–628.
[8] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," 2019, *arXiv:1903.02428*.
[9] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the GPU," in *Proc. 21st ACM SIGPLAN Symp. Princ. Practice Parallel Program.*, 2016, pp. 11:1–11:12.
[10] A. Basak *et al.*, "Analysis and optimization of the memory hierarchy for graph processing workloads," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2019, pp. 373–386.