A Taxonomy for Classification and Comparison of Dataflows for GNN Accelerators

Preprint · March 2021

CITATIONS
0

READS
935

11 authors, including:

Raveesh Garg
Georgia Institute of Technology
4 PUBLICATIONS **1** CITATION
SEE PROFILE

Francisco Munoz Martinez
University of Murcia
8 PUBLICATIONS **12** CITATIONS
SEE PROFILE

Akshay Jain
29 PUBLICATIONS **143** CITATIONS
SEE PROFILE

Sergi Abadal
Universitat Politècnica de Catalunya
142 PUBLICATIONS **1,562** CITATIONS
SEE PROFILE

A Taxonomy for Classification and Comparison of Dataflows for GNN Accelerators

Raveesh Garg*, Eric Qin*, Francisco Muñoz Matrínez†, Robert Guirado‡, Akshay Jain‡, Sergi Abadal‡, José L. Abellán§, Manuel E. Acacio†, Eduard Alarcón‡, Sivasankaran Rajamanickam¶, Tushar Krishna*

Georgia Institute of Technology*, Universidad de Murcia†, Universitat Politecnica de Catalunya‡
Universidad Católica de Murcia§, Sandia National Laboratories¶

Email: *{raveesh.g, ecqin}@gatech.edu, †{francisco.munoz2, meacacio}@um.es, ‡robertguirado97@gmail.com, ‡{akshay.jain, eduard.alarcon}@upc.edu, ‡abadal@ac.upc.edu, §jlabellan@ucam.edu, ¶srajama@sandia.gov, *tushar@ece.gatech.edu

***Abstract*—Recently, Graph Neural Networks (GNNs) have received a lot of interest because of their success in learning representations from graph structured data. However, GNNs exhibit different compute and memory characteristics compared to traditional Deep Neural Networks (DNNs). Graph convolutions require feature aggregations from neighboring nodes (known as the *aggregation phase*), which leads to highly irregular data accesses. GNNs also have a very regular compute phase that can be broken down to matrix multiplications (known as the *combination phase*). All recently proposed GNN accelerators utilize different dataflows and microarchitecture optimizations for these two phases. Different communication strategies between the two phases have been also used. However, as more custom GNN accelerators are proposed, the harder it is to qualitatively classify them and quantitatively contrast them. In this work, we present a taxonomy to describe several diverse dataflows for running GNN inference on accelerators. This provides a structured way to describe and compare the design-space of GNN accelerators.**

I. Introduction

Graph Neural Networks (GNNs) are gaining attention because of their ability to accurately learn representations of graph structured data to solve graph/node classification, graph generation, and link prediction problems [38]. Example applications include item recommendation [9], molecular feature extraction [8], natural language processing [30], semantic segmentation [32], and fraud detection [17]. Computing GNN inference requires a mix of memory and compute intensive operations, which commodity CPUs, GPUs and traditional DNN accelerators do not exploit efficiently [27], [39], [40], [44]. This led to the development of many dedicated GNN accelerators, each with their own design methodology to extract as much performance as possible [3], [4], [11], [21], [27], [39], [41]–[43].

GNN accelerators mainly focus on accelerating two phases that dominate GNN inference: (1) aggregation and (2) combination. *Aggregation phase* requires irregular accesses of data. This is due to the irregular degree distribution in the graphs of interest and as a result irregularity in the neighbor locations of a particular vertex in a given dataset. For example, Figure 1a shows a directed graph, and Figure 1c shows its equivalent adjacency matrix. Since the number of vertices can

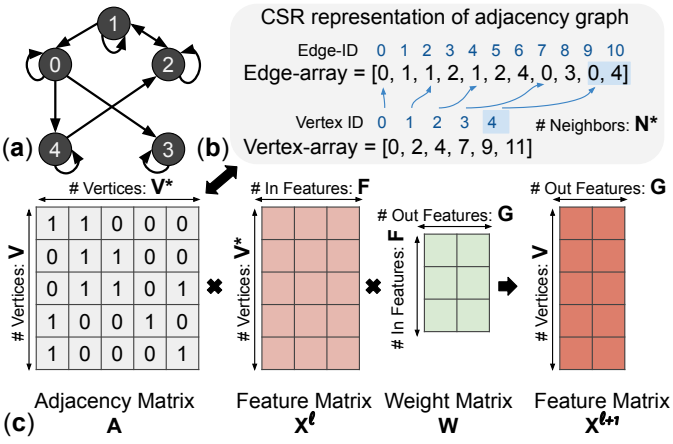


Fig. 1. (a) Example graph (self loops not shown), (b) CSR representation of the adjacency matrix, (c) main computations. (Note: V* can be represented as neighbors (N*) if it is represented in CSR format.) Any computation with **A** is the aggregation phase (e.g. **AX** or **A(XW)**), and any computation with **W** is the combination phase (e.g. **XW** or **(AX)W**).

be in the millions and the total number of edges can be in the billions [25], accessing data efficiently is necessary for reasonable execution time. *Aggregation phase* can be mapped as sparse matrix × dense matrix multiplication (SpMM) with the extremely large sparse matrix (adjacency matrix with number of vertices in each dimension) being the main source of costly irregular accesses. *Combination phase* computations are similar to that of traditional DNNs. Rather than being memory intensive like *Aggregation phase*, *Combination phase* is compute intensive. The main kernel for combination is dense matrix × dense matrix (GEMM). During this phase, not only the data accesses are regular, but also there are lots of opportunities for data reuse.

Each GNN accelerator has its own unique way to compute both *Aggregation* and *Combination phases*. This is due to three factors: the different loop transformation choices, mapping strategies across the processing elements, and the accelerator architecture itself. The factors determine the possible dataflow an accelerator can perform. For this paper, we label the

dataflow of one phase, either *Aggregation* or *Combination phase*, as *intra-phase dataflow*. Through our evaluations, we observe that different intra-phase dataflows perform better for different graph workloads. This stems from the fact that there are various graph categories (from social networks to citation networks to biochemical graphs), which leads to various graph properties (such as number of vertices, edges, features). Additionally, different partitions of a single large graph can present different structures. Unfortunately, many state-of-the-art GNN accelerators only support a fixed intra-phase dataflow, leading to inefficiencies for other graph workloads.

Efficient data movement from one phase to the next is important as it can reduce costly main memory or global buffer reads and writes. We refer to this as *inter-phase dataflow*. We classify three types of inter-phase dataflows: (1) sequential, (2) sequential pipeline, and (3) parallel pipeline. All three types are used in GNN accelerators, but similar to the intra-phase dataflows, each GNN accelerator has its own fixed inter-phase dataflow. This paper will qualitatively and quantitatively show how different inter-phase dataflows perform better for different graph workloads.

Different GNN accelerators have chosen their own unique intra- and inter-phase dataflows and implemented hardware to run those dataflows; however, no work has classified and evaluated which dataflows perform better across a suite of workloads. Additionally, no work has attempted to provide a comprehensive taxonomy for classification and comparison of the possible dataflows for GNN accelerators. In this paper, we propose the first dataflow taxonomy for GNN accelerators. Then, we contrast the performance of various dataflows (some already implemented by previous accelerator proposals and some that have never been studied) across both synthetic and real workloads.

The key contributions of this paper are:

- We evaluate the characteristics of GCN [22] and GraphSAGE [13], which are two representative state-of-the-art GNNs, on an NVIDIA Titan RTX GPU. We show how the time spent on different phases depends on the workload size and structure.
- We develop a novel dataflow taxonomy to characterize the design-space of GNN accelerator dataflows. This is the first work to do this, to the best of our knowledge. This taxonomy expresses: (1) *Aggregation intra-phase* dataflow, (2) *Combination intra-phase* dataflow, (3) *Inter-phase strategy*, and (4) computation ordering.
- Using the taxonomy, we contrast existing and hypothetical GNN dataflows on a cycle-accurate GNN accelerator simulator across synthetic and real graph datasets.
- We conclude that there is no "right" dataflow and prescribe a vision for how future GNN accelerators should be architected to handle different workloads.

Value of Proposed Taxonomy. This work provides insights into understanding the design-space of dataflows for GNN accelerators, which have so far picked specific dataflows [39], [27], [11] and classifying them with a common taxonomy. While there has been significant work on classifying dataflows for DNN accelerators via papers like Eyeriss [6] and MAESTRO [23], and there exist tools for DNN dataflow exploration [31], [23] their focus has been on dense GEMMs. GNNs comprise GEMM and SpMM, introducing additional optimization and reuse opportunities between these two phases that this work tries to capture in a succinct taxonomy. A systematic classification and understanding of GNN dataflows using our taxonomy will lead to more informed design space-exploration for GNN accelerator hardware architectures and dataflow choices. This work can also pave the way for interesting future work on mapping optimizers for GNN accelerators. Researchers targeting specific graph applications will be able to choose the appropriate dataflows with proper understanding of the design space. We also propose our vision for the design of a flexible GNN accelerator in Section V-D which can adapt based on the optimal dataflow which would be efficient for all graphs.

II. Background and Characterization

A. Graph Neural Networks

An input graph is represented by *G(V, E)* with *V* vertices/nodes and *E* edges. Figure 1a shows an example graph with five vertices and eleven edges, including self loops. Figure 1b is the Compressed Sparse Row (CSR) representation of the graph's adjacency matrix, as shown on the left of Figure 1c. They both represent the same information. The $get\_neighbor$ operation gathers attributes from neighboring nodes, and is essential for aggregation. The $get\_neighbor$ operation complexity is *O(n)* where *n* is the number of neighbors of a particular vertex. For CSR representation, shown in Figure 1b, the neighbors of a particular vertex is stored back-to-back. Because most graphs are extremely sparse ($> 99\%$ sparsity) [19], CSR is often used as the graph representation [11], [39]. We use CSR to represent the adjacency matrix in this paper. As shown in Figure 1b, CSR has an edge-array and vertex-array. The vertex-array determines the number of neighbors of a particular vertex and points to the corresponding edge-array location, which contains the specific neighbor vertices.

GNN accelerators mainly target aggregation and combination. Aggregation can be generalized as SpMM (i.e., sparse matrix multiplication) operation, while combination is typically generalized as GEMM operation. Geng et al. [11] describe two possible compute ordering for the forward propagation of a Graph Convolutional Network (GCN): $(\mathbf{A}\times\mathbf{X})\times\mathbf{W}$ and $\mathbf{A}\times(\mathbf{X}\times\mathbf{W})$. As labeled in Figure 1c, $\mathbf{A}$ represents the adjacency matrix, $\mathbf{X}$ represents the feature matrix, and $\mathbf{W}$ represents the weight matrix. Regardless of the compute order, we refer to any computation with $\mathbf{A}$ as part of the *Aggregation phase*, and any computation with $\mathbf{W}$ as part of the *Combination phase*. Other variables include: F (# input features), G (# output features), and N (# of neighbors of a vertex). In this work, *tile size* (T_<Dimension>) is used in the context of spatial loop tiling and it refers to the number of elements of a dimension mapped in parallel across PEs. For

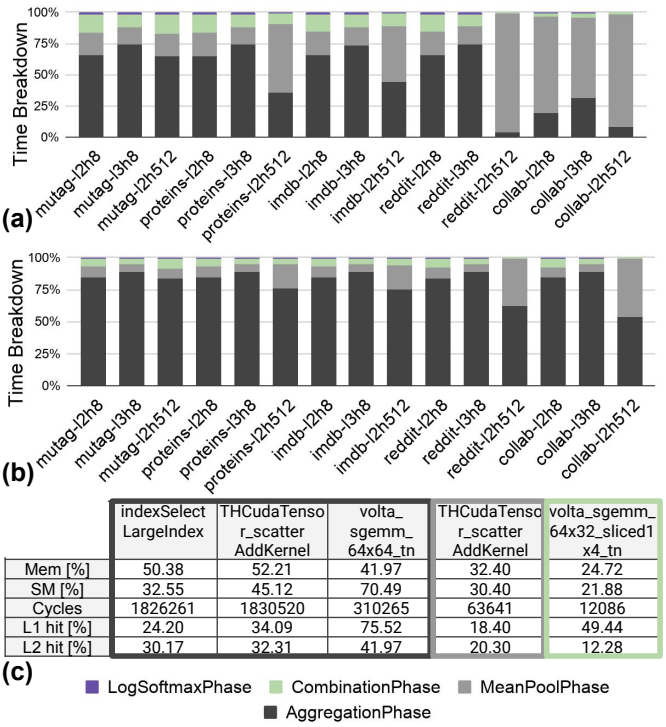
instance, T_F is the number of input features (F) processed in parallel across PEs.

B. PyTorch Geometric Characterization

We first characterize two state-of-the-art GNNs, GCN [22] and GraphSAGE [13] on an NVIDIA Titan RTX GPU. We use PyTorch Geometric and its default model implementations for evaluation. We separate inference into four phases: aggregation, pooling, combination, and softmax. Fig. 2a-b shows the computation breakdowns on GraphSAGE and GCN inference respectively. We use five datasets for graph classification evaluation: Mutag, Proteins, Imdb-bin, Reddit-bin, and Collab (details in Table IV). We also vary both the number of graph convolution layers (two or three), and the hidden dimension size (8 or 512). A network with two convolution layers will get information from nodes that are two hops away.

We observe that aggregations consume roughly ∼40%–∼80% of the total runtime. Mean Pool Phase consumes a good portion of runtime. Pooling phase for GNN accelerators can be implemented with (1) additional parallel lightweight logic for simple pooling operations such as max-pooling and min-pooling and (2) slight extensions to the aggregation hardware for mean-pooling. As shown in in Fig. 2c, mean pool uses many *ThCudaTensor_scatterAddKernels* which are also present in the Aggregation phase. Again, similar to previous GNN accelerators [3], [4], [11], [21], [27], [39], [41]–[43], this work will focus only on the *Aggregation* and *Combination* phases, as the main kernels in aggregation and combination consume a majority of the GNN inference runtime.

Fig. 2c shows memory and compute intensive kernels that were launched during each inference stage. Benchmarked with NVIDIA Nsight [1], the table shows the memory utilization (percentage of GPU resource utilized per kernel), compute (SM) utilization, number of cycles, L1/ Texture cache hit rate, and L2 cache hit rate. To highlight, *indexSelectLargeIndex* (gathers neighbor information) consumes a large percentage of the memory bandwidth and has low L1 hit rate. On the other hand, *volta_sgemm_64x64_tn* and *volta_sgemm_64x32_sliced1x4_tn* (GEMM operations) has a relatively high L1 hit rate. In summary, our key observation is that the workload size and structure determine how much time is spent on the different phases.



	indexSelect LargeIndex	THCudaTensor_scatter AddKernel	volta_ sgemm_ 64x64_tn	THCudaTensor_scatter AddKernel	volta_sgemm_ 64x32_sliced1 x4_tn
Mem [%]	50.38	52.21	41.97	32.40	24.72
SM [%]	32.55	45.12	70.49	30.40	21.88
Cycles	1826261	1830520	310265	63641	12086
L1 hit [%]	24.20	34.09	75.52	18.40	49.44
L2 hit [%]	30.17	32.31	41.97	20.30	12.28

(c)

Fig. 2. (a) GraphSAGE and (b) GCN inference runtime breakdown. The labels are: <dataset>_<layers>_<hidden dim>. (c) Select kernels from each computation phase on COLLAB dataset with hidden dimension size of 256.

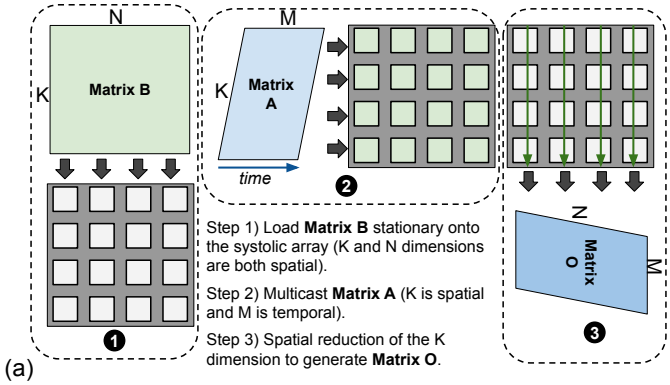
III. Dataflow Taxonomy and Exploration

A. Dataflow description

In the spatial accelerator domain, dataflow refers to the loop order and parallelism strategy employed when mapping the data and computation tiles over the PEs. The dataflow choice exposes reuse opportunities for data operands over space (i.e., over wires via multicasts and reductions) and time (i.e., via scratchpad buffers) [23]. Fig. 3a shows an example of a 2D systolic array using a weight stationary dataflow. The KN matrix is loaded spatially onto the PEs, and the elements of MK matrix are multicasted to the PEs. The output matrix is generated by spatial reductions across the K dimension. Fig. 3b shows typical 2D GEMM dataflows. In the following sections, we discuss GEMM/ SpMM dataflows within an individual phase *(Intra-phase dataflows)* and the overall dataflow with both the phases *(Inter-phase dataflows)*.

B. Intra-phase Dataflow

1) ***Aggregation Dataflow****:* Fig. 4a-b illustrate two examples of dataflows present in the aggregation operation. For the example, we assume an unweighted adjacency matrix; therefore, only adders are necessary. This is because the adjacency matrix only has values of "1", so multiplications are unnecessary. Both examples show V (vertex dimension) as the most outer loop, and both are marked as temporal. This means that the next vertex starts after the current vertex finishes along with their inner loops. In the next loop, Fig. 4a shows F (features) as spatial, while Fig. 4b shows N (neighbors) as spatial. In the most inner loop, Fig. 4a shows N as temporal and Fig. 4b shows F as temporal. Since the corresponding features of each neighbor must be accumulated, the dataflow defines which reduction strategy is required. Fig. 4a and Fig. 4b require temporal and spatial reductions, respectively. Note that spatial reduction can be done through a linear chain or adder tree. Finally, for our taxonomy, we define Fig. 4a and Fig. 4b aggregation dataflows as (V_t {F_s N_t}) and (V_t {N_s F_t}) respectively. Here s and t in the subscript represent whether a dimension *Dim* is spatial (T_Dim > 1) or temporal (T_Dim = 1) and x in the subscript implies that the dimension can be either temporal or spatial. The curly brackets show that the loop order is interchangeable; for example, Fig. 4a can



(b)

Dataflow	Implication of Loop Order	Implication of Spatial Dimensions
$\{M_s\ N_s\}\ K_t$	Output (MN) stationary, Inputs (MK) and Weights (KN) stream every cycle	Spatial multicast of Inputs and Weights every cycle. Temporal reduction of partial sums within each PE.
$\{N_s\ K_s\}\ M_t$	Weight (KN) stationary, Input (MK) streams every cycle	Spatial multicast of inputs every cycle. Spatial reduction of partial sums across PEs
$\{M_s\ K_s\}\ N_t$	Input (MK) stationary, Weight (KN) streams every cycle	Spatial multicast of weights every cycle. Spatial reduction of partial sums across PEs

Fig. 3. (a) Weight stationary systolic array GEMM dataflow. (b) Common 2D dataflows. The order of dimensions within {} can be interchanged. The subscript *s* on two dimensions represents a spatial mapping (i.e., unrolling) of those dimensions across the rows and columns of the accelerator (in a tiled manner), while *t* represents temporal. GEMM's standard M, N, K dimension notations correlate to V, F, N for aggregation and V, G, F for combination (assuming aggregation to combination). Note that the N in GEMM's notation of M, N, K is different than V, F, N, which represents neighbors.

be either (V_t $\{F_s$ $N_t\}$) or (V_t $\{N_t$ $F_s\}$). This is because the dataflow is determined by relative order of the temporal loops. In Figure 4, we assume that the spatial dimension fits on the PEs. However, if the dimensions do not fit, then they have to be completed in multiple iterations. There is a temporal loop nest for multiple iterations and the order of the spatial dimensions matters in that case. Note that we show only four dataflows. Other possible dataflows are different combinations of temporal and spatial loops orderings.

*2) **Combination Dataflow**:* Fig. 4c shows an example combination dataflow. Both the V × F and F × G matrices are streaming into the multiply-accumulate (MAC) units. The input features vary each cycle; and thus the dimension is temporally mapped. The other dimensions are spatial, resulting to ($\{V_s$ G_s $F_t\}$). Note that this dataflow is identical to an output stationary systolic array. Fig. 4d also shows an output stationary approach, but with the G dimension as temporal. This leads to each PE holding multiple outputs. Different dataflows require different hardware implementations (e.g. Fig. 4c and Fig. 4d). Even the same dataflow can have different implementations (e.g. Fig. 4c data can be sent through a store-and-forward manner or through a single cycle bus multicast).

Although each phase can use any intra-phase dataflow, the dataflow choice for one can affect the other. For e.g., an efficient aggregation dataflow should generate partial outputs that can readily be used as inputs for the combination dataflow.

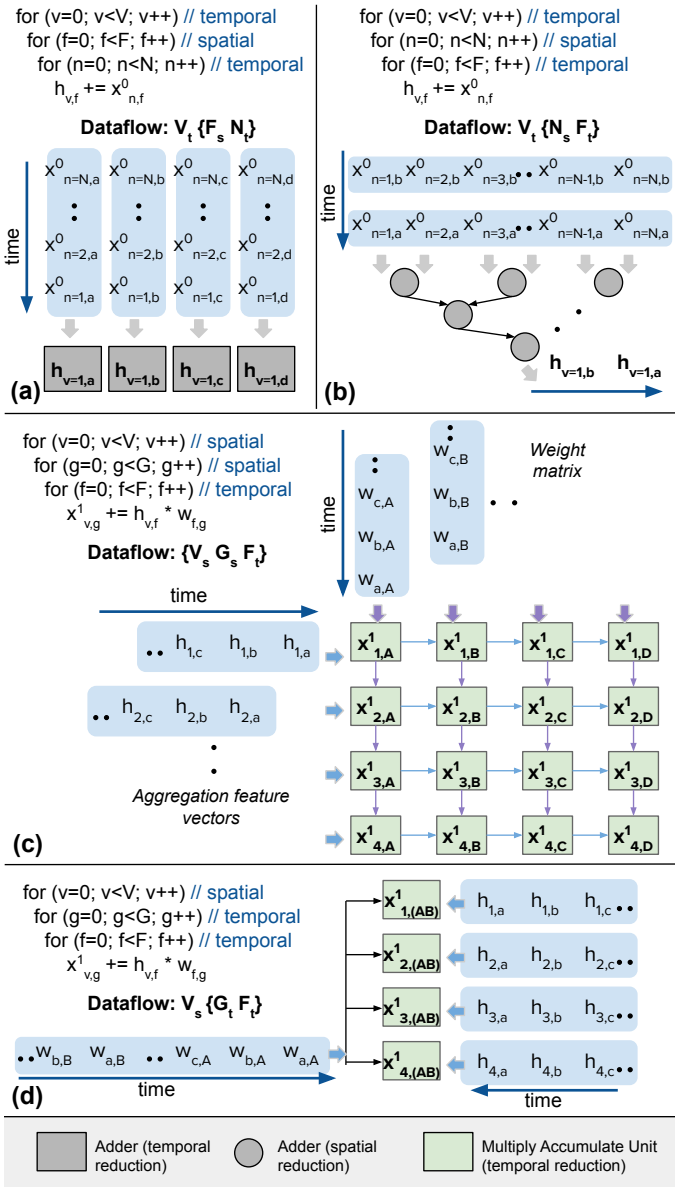


Fig. 4. (a,b) Aggregation dataflow example showing temporal and spatial maps. (c, d) Combination dataflow example. **Feature Notation**: $x^{layer}_{vertex,feature}$, **Weight Notation**: $w_{infeature,outfeature}$, **Intermediate Notation**: $h_{vertex,feature}$. Dataflow notion is described in Sec. III-B.

We term this bridge as inter-phase dataflow.

C. Inter-phase Dataflow

Each GNN accelerator has proposed its own unique dataflow that merges the data between aggregation and combination seamlessly, which we refer to as *inter-phase dataflow*. This is important because it determines the number of memory accesses required to move data from one phase to the next. Fig. 5 presents the types described below.

*1) **Sequential (Seq)**:* occurs when there is only one unit to compute both aggregation and combination phases. First, one phase (either aggregation or combination) allocates all the PEs in the accelerator for computation. After finishing the

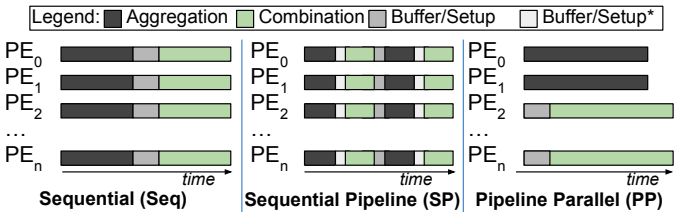


Fig. 5. Different inter-dataflow strategies between aggregation and combination phases. Buffer/setup* is optional depending on the dataflow.

full phase, the output goes into a global buffer. Then, for the next phase, all the necessary data are gathered from the global buffer and are allocated back onto the PEs.

*2) **Sequential Pipeline (SP)**:* is similar to sequential but splits aggregations and combinations into small stages, which are then interleaved over time on the same unit. Depending on the intra-phase dataflows, it is possible to reduce data movement between aggregation and combination stages. This is represented by the light gray region in Fig. 5. Specifically, the output data of one stage can be kept stationary within the PEs' local buffers (as EnGN does [27]) rather than going into an intermediate global buffer. An intermediate global buffer is placed between the aggregation and combination units. This is represented by the yellow rectangle in Fig. 7.

*3) **Parallel Pipeline (PP)**:* is when both phases are allocated onto units of an accelerator at the same time. A fixed ratio of PEs (like in HyGCN [39]) or a flexible ratio of PEs (like in AWB-GCN [11]) are allocated to different phases. An intermediate NoC is needed to send data from one stage to the next. For this methodology, it is critical to balance the production and consumption rate to reduce stalls.

D. Detailed Taxonomy and Example: HyGCN

Given the dataflow types described above, the taxonomy template that we use to describe GNN accelerators is:

<Inter>$_{\text{<order>}}$(<AggIntra>, <Cmb Intra>)

<Inter> represents the inter-phase dataflow, <order> represents the computation order (aggregation to combination is AC, combination to aggregation is CA). The computation order affects the data movement and buffer utilization of the accelerator. <AggIntra> represents the aggregation phase intra-phase dataflow, and <CmbIntra> represents the combination phase intra-phase dataflow. Using this template, we can describe HyGCN [39] as: $PP_{AC}(V_xF_sN_t, V_sG_sF_t)$

Note that HyGCN's aggregation and combination intra-dataflows correspond to Fig. 4a and Fig. 4c respectively. HyGCN gathers intermediate output data from the aggregation engine using an unidirectional high bandwidth bus, and then distributes the data into the combination engine; therefore, the computation order is AC. To summarize, Table I shows complete dataflow descriptions of multiple GNN accelerators using our proposed taxonomy fields. The table also shows what hardware structures are required.

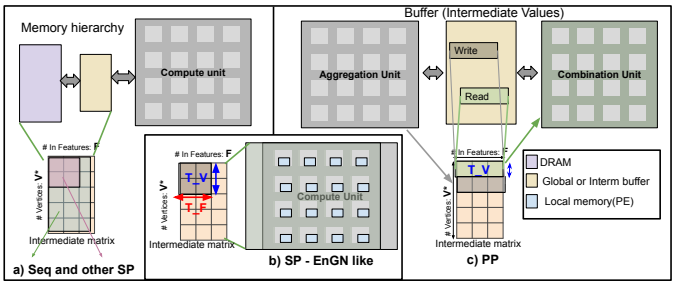


Fig. 6. Intermediate buffering in Inter-Phase dataflows

E. Analyzing Inter-Phase Dataflows

This section discusses Inter-phase dataflows in more detail. We mainly focus on phase order Aggregation to Combination but the concepts are similar for the reverse phase order. We analyze inter-phase dataflows in terms of runtime, intermediate global buffering support needed and feasible dataflows. Table II describes the runtime and intermediate buffering reqirements for intermediate dataflows.

1) Sequential Dataflow: In sequential dataflow, the phases simply run one after the other using any intra-phase dataflow. The overall latency is the sum of latencies of individual phases. The entire intermediate matrix is first written to the memory by first phase and then read from the memory by second phase. Hence the intermediate storage ($\text{Buffer}_{\text{inter}}$) used is simply the number of output elements which is V×F data elements as shown in Fig 6a. Such amount of data cannot be stored on-chip for large graphs and hence incurs high energy overhead.

2) Sequential Pipeline: In Sequential pipeline, a few elements of the first phase are computed and then second phase acts on those elements and the process is repeated in an interleaved manner. In general, the whole workload can be divided and processed in an interleaved manner using any dataflow. For example, one trivial way can be to partition the graph by vertices and process the partitions in an interleaved manner, any dataflow can be used for aggregation and combination for the selected vertices. For any arbitrary dataflow, the intermediate data can be written in the buffers and then read from the buffers as shown in Fig 6a. Thus, the intermediate storage required is equal to the tile size T_V×T_F. The runtime is same as that of sequential dataflow.

Sequential pipeline provides savings in latency and memory over sequential only when the full outputs produced in the processing elements are used as inputs without reading them from global buffer memory as it saves extra memory reads and loading latency. We focus our analysis on the corresponding specific dataflow $\text{SP}_{\text{AC}}(\{\text{V}_\text{x}\text{F}_\text{x}\}\text{N}_\text{t},\{\text{V}_\text{x}\text{F}_\text{x}\}\text{G}_\text{t})$ in second row of Table I and is used in EnGN [27].

In order to save memory and latency, the above SP-EnGN-like dataflow requires support to store intermediate outputs locally inside the processing element for the next phase to use it directly as input. The loop order pair for aggregation to combination phase should be ({VF}N, {VF}G). After, one iteration of temporal loop-nest of V,F is finished with all neighbors reduced, the accumulated data remains in the MAC units and

TABLE I

CHARACTERIZING THE DESIGN-SPACE OF DATAFLOWS FOR GNN ACCELERATORS (NOTE THAT ANY DIRECTION WILL AFFECT THE AGGREGATION AND COMBINATION DIMENSION VARIABLES, BUT SIMILAR CONCEPTS APPLY). SUBSCRIPT S, T, X MEANS SPATIAL, TEMPORAL, EITHER SPATIAL OR TEMPORAL RESPECTIVELY. CURLY BRACKETS '{}' SHOW THAT THE LOOP ORDER IS INTERCHANGEABLE.

Inter Phase	Aggregation	Combination	Intermediate Buffer	NoC / PE Support	Example & Order	Remarks
Sequential (Seq)	$\{V_x\ F_x\ N_x\}$	$\{V_x\ G_x\ F_x\}$	✓	Depends on Accelerator	TPU [18] Eyeriss [5] (any direction*)	Similar to running one full or tiled layer at a time. Outputs of one layer gets stored in scratchpad and rescheduled back into the PEs.
Sequential Pipeline (SP)	$\{V_x\ F_x\}\ N_t$	$\{V_x\ F_x\}\ G_t$	✗	Requires support to store phase 1 outputs locally inside PE, so that phase 2 can use them as inputs	EnGN [27] (any direction*)	No intermediate support required, as output data of one phase is stationary in the buffers, and can be used as input for the next phase. Order of V,F should be the same for both the phases. T_V_{AGG}= T_V_{CMB} and T_F_{AGG}=T_F_{CMB} and reduction is temporal as data is always in in-place buffers.
	$V_t\ \{F_s\ N_t\}$	$\{V_s\ G_s\ F_t\}$	✓	Requires support to store agg. phase outputs locally inside PE, so that cmb. phase can use them as inputs	Rubik [4] (agg->cmb)	Paper also focuses on reducing redundant aggregations by saving duplicate data onto a private cache. Similar to EnGN, no intermediate data reordering required as output data of one phase is stationary in PEs and is used as inputs for the next phase.
	$\{V_x\ F_x\ N_x\}$	$\{V_x\ G_x\ F_x\}$	✓	Agg:V_s,F_s require multicast distribution. $N_{s/t}$ requires spatial/temporal reduction. Cmb:V_s,G_s require multicast distribution. $F_{s/t}$ requires spatial/temporal reduction.	(any direction*)	There will be a buffer/ setup delay between aggregation and combination phases to remap output data to new location.
Parallel Pipeline (PP)	$\{V_x\ F_x\}\ N_x$	$\{V_x\ F_x\}\ G_x$	✓	Need support to connect agg and cmb units to intermediate buffer	(agg ->cmb)	**Element(s) wise granularity**: Element(s) of the intermediate matrix indexed by V,F can be pipelined. The order of V,F should be same for both phases.
	$V_x\ \{F_x\ N_x\}$	$V_x\ \{G_x\ F_x\}$	✓	Need support to connect agg and cmb units to intermediate buffer	HyGCN [39] DAC GNN [3] (agg ->cmb)	**Row(s) wise granularity**: Row(s) of intermediate matrix indexed by V can be pipelined. The order should not be (VFN,VFG) since that can be done element wise. HyGCN allocates fixed number of PEs per unit, which may lead to stalls. E.g., combination engine idle while waiting. HyGCN dataflow - $PP_{AC}(V_xF_sN_t, V_sG_sF_t)$
	$F_x\ \{V_x\ N_x\}$	$F_x\ \{G_x\ V_x\}$	✓	Need support to connect agg and cmb units to intermediate buffer	(agg ->cmb)	**Column(s) wise granularity**: Column(s) of the intermediate matrix indexed by F can be pipelined. The order should not be (FVN,FVG).
	$\{N_x\ F_x\}\ V_x$	$\{V_x\ G_x\}\ F_x$	✓	Need support to connect agg and cmb units to intermediate buffer	(cmb ->agg)	**Element(s) wise granularity**: The order should be (NFV, VGF) or (FNV, GVF). V×G matrix after cmb becomes N × F for agg.
	$N_x\ \{V_x\ F_x\}$	$V_x\ \{G_x\ F_x\}$	✓	Need support to connect agg and cmb units to intermediate buffer	(cmb ->agg)	**Row(s) wise granularity**: Order should not be (NFV, VGF) since it can be done element wise.
	$F_x\ \{V_x\ N_x\}$	$G_x\ \{F_x\ V_x\}$	✓	Need support to connect agg and cmb units to intermediate buffer	AWB-GCN [11] (cmb ->agg)	**Column(s) wise granularity**: The order should not be (FNV, GVF). AWB-GCN enables flexible allocation of PEs for different phases to match production and consumption rates. AWB-GCN dataflow- $PP_{CA}(F_sN_tV_s, G_tF_tV_s)$

then dimension G is streamed over it. Moreover, corresponding T_Dimensions for both Aggregation and Combination would be same since the same intermediate data stored in the PEs by the first phase is processed by the second phase. Thus for the phase order Aggregation to Combination, T_V_{AGG}=T_V_{CMB} and T_F_{AGG}=T_F_{CMB}. Also, since the data should be available for consumption locally in the processing elements, reduction must be temporal (T_N=1).

The advantage of SP-EnGN-like dataflow is reduced memory accesses, since the data to be used in the second phase is directly used from the PEs, there is no extra buffering required as also shown in Fig 6b. Moreover it saves the latency and memory read overhead of loading the data into PEs.

3) Parallel Pipeline Dataflow: Parallel pipeline dataflow consists of two compute engines with one engine feeding the other engine spatially. The advantage that this kind of dataflow has over the others is that computation is done parallely, so the latency of one phase is hidden. The intermediate data is broken down into small granularities and the data is processed in a pipelined manner. For example, if granularity is a single row of intermediate matrix, then the aggregation phase computes and writes the data corresponding to n^{th} row and in parallel, combination phase reads and processes data corresponding to $(n\text{-}1)^{th}$ row. To facilitate this, intermediate buffers are required and the amount of intermediate buffering required is twice number of elements of the intermdiate matrix *(PPel)* pipelined (2×*PPel*) as shown in Fig 6. The runtime of one pipeline iteration is equal to the runtime of the slower phase for producing *PPel* elements. The total runtime is the sum of runtimes of individual iterations sum(max(t_{AGG}, t_{CMB})). There are three broad categories of granularities at which intermediate matrix can be broken.

Element-wise granularity: Element(s) wise granularity involves tiles of few elements being processed in a pipelined

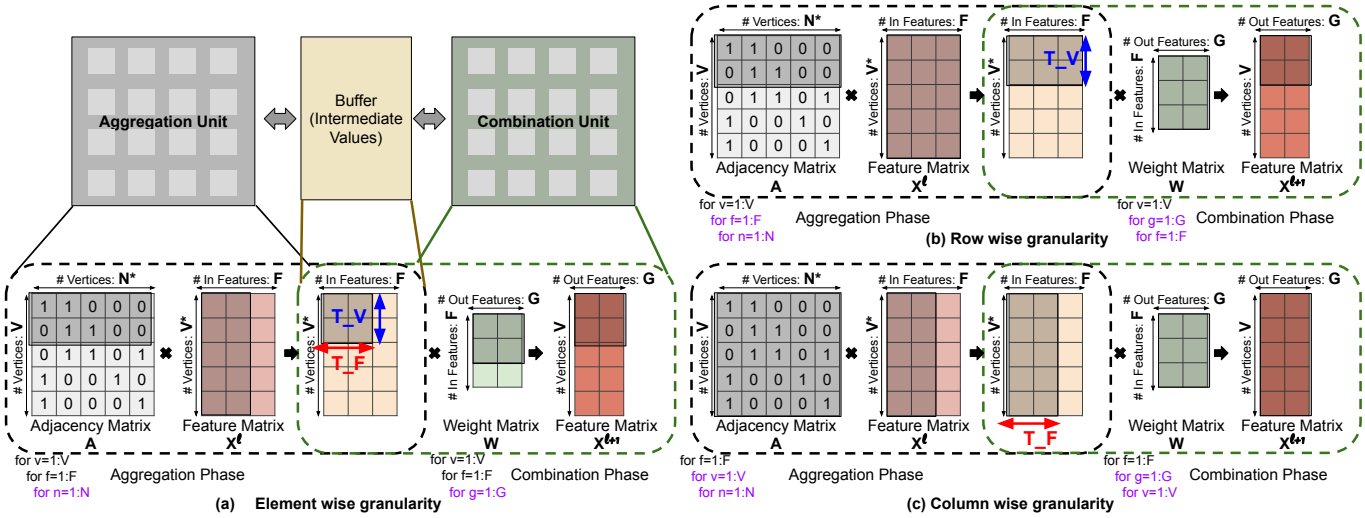


Fig. 7. Different parallel pipeline strategies with different granularities: (a) Element(s) wise (b) Row(s) wise and (c) Column(s) wise for the phase order Aggregation to Combination.

manner rather than the whole dimension. PPel is T_F×T_V.

Fig. 7a shows an example loop order (VFN,VFG) for Element(s) wise granularity for phase order Aggregation to Combination. For each element or a set of elements which is (are) indexed by V and F, the inner-most loop (N) of aggregation reduces all the neighbors for a given tile of vertices and features to be reduced and parallelly, the innermost loop of combination computes G for the previous tile of vertices and features. Table I shows all the feasible loop orders for all possible granularities and phase orders.

If the tile sizes of dimensions are imbalanced for multiple phases, the number of elements that would be written in the intermediate buffers would be the Least Common Multiple (LCM) of both the $T\_Dimension_{AGG}$ and $T\_Dimension_{CMB}$. For this work, in case of an imbalance, we only consider tiling strategies where higher tile size would be a multiple of lower one to avoid having large number of pipelined elements due to large LCM. So, it is equivalent to considering the max of two tile sizes T_F_{MAX} and T_V_{MAX}. For example, if T_F_{AGG}=1 and T_F_{CMB}=2, with same T_F, two iterations of aggregation are needed before pipelining step. Element-wise granularity becomes inefficient if the tile size of the outermost dimension (V or F) in the loop order is imbalanced because it requires completion of atleast one row/column in one phase. The amount of buffering is twice of *PPel* ($2\times T\_V_{max}\times T\_F_{max}$).

Row-wise granularity: In row(s)-wise granularity, whole row(s) of the intermediate matrix is (are) considered instead of a few elements in a row. The number of rows that are pipelined is T_V_{max}. So *PPel* = $T_V_{max}\times$F. Fig. 7b shows an example loop order (VFN, VGF) for Row(s) wise granularity for phase order Aggregation to Combination. Each row of the aggregated matrix is indexed by V. So for each set of rows indexed by V, the inner two loops compute for aggregation F and N in parallel, the two innermost loops for combination compute G and F for the previous row(s). The loop orders corresponding to row(s) wise granularity for both phase orders are described in Table I.

Column-wise granularity: In column(s)-wise granularity the whole column(s) of the intermediate matrix is (are) considered instead of a few elements in a column. The number of columns that are pipelined is T_F_{max}. So the *PPel* = $T_F_{max}\times$V. Fig. 7c shows an example loop order (FVN, FGV) for Column(s) wise granularity for phase order Aggregation to Combination. Each column of the aggregated matrix is indexed by F while the inner two loops compute the computations corresponding to columns in pipelined manner. Feasible loop orders are described in Table I.

Table II contains expressions for the runtime and buffering requirement for different inter-phase dataflows.

F. Scope of Taxonomy

The present taxonomy distinguishes the intra-phase dataflows that can be used for vertex aggregation and combination, as well as dataflows between both phases of a GCN. This is enough to characterize most accelerators [11], [27], [39], since they consider vertex updates only. However, we note that general computation of a GNN may also include edge updates (once per edge and layer), as well as a final graph-wide update [2]. Even though these updates may involve distinct aggregation and combination functions, our taxonomy is also applicable to their intra- and inter-phase dataflows. Our taxonomy also does not capture how the nodes of the graph are ordered and the graph is partitioned.

Some sparsity centric optimizations for Aggregation phase such as workload balancing in AWB-GCN [11] and window shrinking in HyGCN [39] are not captured in the dataflows. Moreover, the description does not capture specific Aggregation optimizations in Rubik [4] where repeated partial sums are reused, thus redundant computations are eliminated [16].

IV. Experimental Methodology

A. Cycle Accurate Simulation Framework

In order to evaluate various intra-phase dataflows, we built a cycle accurate simulation framework which can model various

TABLE II
RUNTIME AND BUFFERING REQUIREMENTS FOR INTER-PHASE DATAFLOWS

Inter-phase dataflow	Intermediate Buffering	Runtime
Seq	$V \times F$	$t_{AGG}+t_{CMB}$
SP-arbitrary	$T\_V \times T\_F$	$t_{AGG}+t_{CMB}$
SP-EnGN-like	0	$t_{AGG}+t_{CMB}-t_{load}$
PP-elements	$2 \times T\_V_{max} \times T\_F_{max}$	$sum(max(t_{AGG},t_{CMB})_{PPel})$
PP-Row	$2 \times T\_V_{max} \times F$	$sum(max(t_{AGG},t_{CMB})_{PPel})$
PP-Column	$2 \times V \times T\_F_{max}$	$sum(max(t_{AGG},t_{CMB})_{PPel})$

dataflows. We build our cycle accurate framework around STONNE simulator [29].

STONNE simulator models flexible accelerators MAERI [24] and SIGMA [33]. The underlying hardware model consists of a flexible distribution network with distribution latency of one cycle and the reduction is tree based. The simulator can support re-configurable spatial reduction sizes. The engines can therefore handle different tile sizes. We add support for SpMM (Sparse×Dense) memory controller in STONNE. The simulator can be used in both SpMM configuration and Dense configuration and is used to compute the complete phases aggregation and combination respectively. To implement inter-phase dataflows, we built an analytical model around the simulator that computes the runtime and energy of a GNN layer. We input the sparse graphs of the workloads using CSR format and use the dimensions V, F and G as the inputs to the framework

In this work, since our goal is to compare the impact of the dataflows, we assume that there is sufficient buffering to run any dataflow to ensure that there are no stalls due to data fetches from the memory. Any performance difference is purely due to under-utilization of a dataflow, loading and streaming delays, and stalls in parallel pipeline dataflow.

B. Dataflow Evaluator

For evaluation, we choose 8 representative dataflows. All the configurations have phase order from Aggregation to Combination and vertex is always the outermost loop. Parallel pipeline dataflow assumes row-wise granularity and since the number of vertices is high, it is possible to have different number of rows to have different impact. Sequential pipeline has vertex as the outer loop and feature as the inner loop because of the huge amount of memory saving in that order. The dataflow configurations have been named based on the taxonomy in Section III-D. Since we select a subset of mapping choices, we use a compact dataflow name for the evaluation section. The compact name consists of three parts separated by '-'. First part represents Inter-phase dataflow, second part indicates dimensions which are necessarily spatial or temporal for aggregation and the third part (if any) indicates dimensions which are necessarily spatial or temporal for combination. In third part of compact name for PP the subscripts corresponding to the vertex are t/sl for lower(l) granularity and sh for higher(h) granularity. We also introduce High-V_s-SP dataflow to highlight the problem of parallelizing sparse dimensions for all datasets except proteins and Mutag where T_V is already high due to smaller T_F limited by F. Table III[1] describes all the configurations that we use for evaluations.

TABLE III
REPRESENTATIVE DATAFLOW CONFIGURATIONS FOR EVALUATION.

Dataflow Configuration	Compact	Description
$Seq_{AC}(V_xF_xN_t,V_xG_xF_x)$	Seq-N_t	Sequential dataflow Temporal Aggregation
$Seq_{AC}(V_xF_xN_s,V_xG_xF_x)$	Seq-N_s	Sequential dataflow Spatial Aggregation
$SP_{AC}(V_xF_sN_t,V_xF_sG_x)$	SP-F_sN_t-F_s	Sequential Pipeline dataflow EnGN-like Temporal Aggregation Relatively high T_F
$SP_{AC}(V_sF_xN_t,V_sF_xG_x)$	SP-V_sN_t-V_s	Sequential Pipeline dataflow EnGN-like Temporal Aggregation Relatively high T_V
$PP_{AC}(V_xF_xN_t,V_xG_xF_x)$	PP-N_t-$V_{t/sl}$	Parallel Pipeline dataflow Temporal Aggregation HyGCN-like Granularity of lower rows
$PP_{AC}(V_xF_xN_s,V_xG_xF_x)$	PP-N_s-$V_{t/sl}$	Parallel Pipeline dataflow Spatial Aggregation Granularity of lower rows
$PP_{AC}(V_xF_xN_t,V_sG_xF_x)$	PP-N_t-V_{sh}	Parallel Pipeline dataflow Temporal Aggregation HyGCN-like Granularity of higher rows
$PP_{AC}(V_xF_xN_s,V_sG_xF_x)$	PP-N_s-V_{sh}	Parallel Pipeline dataflow Spatial Aggregation Granularity of higher rows
$SP_{AC}(V_sF_xN_t,V_sF_xG_x)$	High-V_s-SP	SP dataflow with high T_V

TABLE IV
DATASETS INFORMATION. FIRST HALF FOR GRAPH CLASSIFICATION (WE USE BATCH SIZE OF 64), BOTTOM HALF FOR NODE CLASSIFICATION. (* MEANS THAT INDICATOR VECTORS WERE USED IN PLACE OF FEATURES).

Name	# Graphs	# Nodes (avg)	# Edges (avg)	# Features
Mutag (MU)	188	17.93	19.79	28*
Proteins (PR)	1113	39.06	72.82	1(reg)/29(full)
Imdb-bin (IB)	1000	19.77	96.53	136*
Reddit-bin (RB)	2000	429.63	497.75	3782*
Collab (CL)	5000	74.49	2457.78	492*
Citeseer (CS)	1	3327	9464	3703
Cora (CR)	1	2708	10858	1433

C. GNN Algorithms and Datasets

We evaluate the dataflows described in Table III for GCN [22] for the target datasets described in Table IV. For each dataflow, we also selected appropriate tile sizes as described below to create a complete mapping.

D. Choosing Tile Sizes

We pick a few representative tile sizes for our analysis, based on the following heuristics.

- Number of PEs=512 and Mapping Efficiency (Product of Tile sizes/Number of PEs) is close to 100%. Since Parallel

[1]Our purpose here is to not compare HyGCN and EnGN's underlying implementation. We evaluate HyGCN-like (PP) and EnGN like (SP) dataflows to understand the factors affecting the runtime associated with the dataflows.

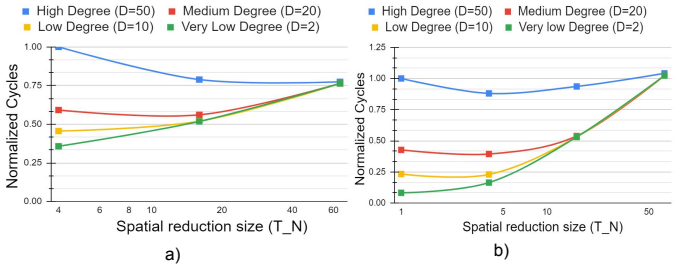


Fig. 8. Aggregation dataflow- $V_tF_xN_x$. Impact of tile size T_N on performance for graphs with (a) V=10000 (High) and F=32 (low) and (b) F=512 (High) and V=1000 (Low) for varying average degrees (D). Number of PEs = 128. T_V=1 and T_F=128/T_N. Runtime is normalized by runtime correspoding to highest D and lowest T_N

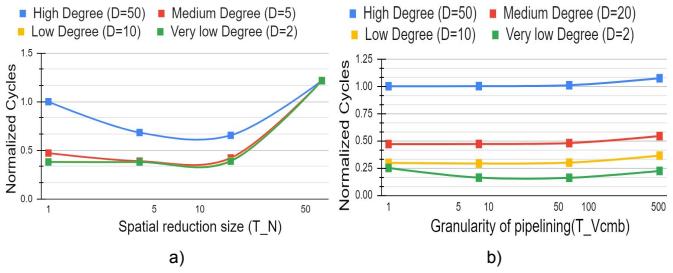


Fig. 9. Impact of factors affecting pipelining for Aggregation followed by Combination. V=1024, F=512 and G=16 and number of PEs=512 for each phase for different average degrees D. (a) shows the runtime for different tile sizes. Dataflow - $PP_{AC}(V_tF_xN_x, V_tG_sF_s)$. T_N in the inter-phase pipeline. T_V=1, T_F_{CMB}=32, T_G=16, T_F_{AGG} = 512/T_N. (b) shows the variation of performance with the increase in granularity of pipelining. Dataflow- $PP_{AC}(V_tF_sN_t, V_xG_tF_x)$. V=1024, T_V_{CMB} is varied and it represents the number of rows being pipelined. T_F_{CMB}=512/T_V_{CMB}.

Pipeline has two phases operating in parallel, to have same number of total PEs, each phase has 256 PEs.

- For SP, we use EnGN-like dataflow. The buffer setup time between aggregation and combination is 0, that is the intermediate outputs are directly in PEs for next phase. T_V_{Agg}=T_V_{Cmb} and T_F_{Agg}=T_F_{Cmb}. T_N=1 since the intermediate matrix should be accumulated inside PE.

V. Evaluation

A. Runtime Analysis using Synthetic Matrices

In this subsection we evaluate GCN [22] on synthetic matrices to understand the impact of tile sizes for some of the dataflows in Table III. Synthetic workload used comprises of GEMMs. Each of the synthetic adjacency matrices used have varying number of edges (i.e., sparsity ratio).

1) Intra-phase dataflows: Here, we evaluate the factors that influence the performance of an individual phase. For, combination (Dense) phase, there is a vast literature describing Dense CNNs and Dense GEMMs and we refer the reader to Kwon et al. [23].

For Aggregation (SpMM), two primary factors that influence the run-time is the set of tile sizes for various dimensions and the amount of sparsity. Large T_N can lead to under-utilization of PEs. However if T_N=1, latency can be high specially when the second phase needs to use the output soon, for example PP. The impact of tile sizes on performance for different graphs is shown in Fig. 8. These plots show that as the graph becomes more densely connected, higher T_N is more optimal. Thus, optimal tile size for reduction in SpMM depends on the graph sparsity.

2) Inter-phase dataflows: The factors affecting the performance of Sequential dataflow are same as the factors affecting dataflows of individual phases as the number of cycles is simply the sum of number of cycles of individual phases.

In Parallel Pipeline dataflow, the delay is governed by the slower phase. Therefore intra-phase performance is critical to parallel pipeline. Parallel pipeline is also affected by granularity as increase in granularity increases the delay of the non-overlapping part. Fig. 9a shows the impact of temporal or spatial reduction in Aggregation on overall pipelining. For lower average degrees, combination delay dominates the

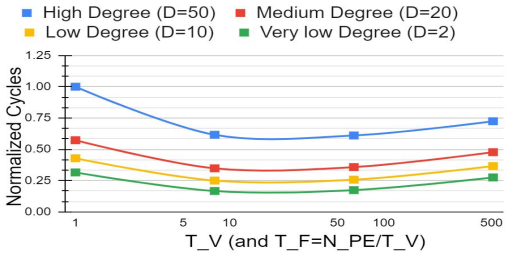


Fig. 10. Impact of factors affecting interleaving for Aggregation followed by Combination. Dataflow - $SP_{AC}(V_xF_xN_t,V_xF_xG_t)$. V=1024, F=512, G=16, Number of PEs = 512. These figures show the impact of different tile size combinations T_V and T_F. Sequential pipelining for T_V = {1,8,64,512}. T_V = 512/T_V with T_N = T_G = 1. Note that this example does not aim to model exact EnGN implementation. Here, inter-phase dataflow is EnGN like.

Aggregation and thus change in T_N does not impact the net delay. But as average degree increases, Aggregation delay starts to dominate and affect the net delay. Fig. 9b shows the impact of having high granularity on performance. On increasing T_V_{CMB}, the parallel pipeline delay first decreases due to parallelization of the stationary dimension V but increases because of the increase in the non-overlapping delay.

For SP, we use EnGN-like dataflow. We evaluate this dataflow for phase order aggregation to combination for tile sizes T_F and T_V.

Fig. 10 shows the impact of different tile sizes T_V and T_F on interleaving. In this example, aggregation and combination have vertex in the outermost loop. Thus, tiling T_V spatially reduces the number of cycles needed to load vertices in the PEs, however, beyond a point the latency becomes higher. The main reason for that is that V is an irregular dimension and parallelizing V implies that the computation is limited by the vertex with most neighbors in the tile which is referred to as "evil row" in AWB-GCN [11]. The impact of evil row is more significant in real world graphs. AWB-GCN has a load balancing engine that schedules the evil row on multiple PEs. Alternatively, one can also choose lower T_V.

Thus the choice of tile sizes has a huge impact on each dataflow and the optimal tile sizes can vary with the graph.

B. Runtime Evaluation on Real Graph Structures

We evaluate dataflow configurations in Table III for datasets in IV. Figure 11 shows the GCN runtimes normalized by the delay of Seq-N_t dataflow. Our observations are as follows-

- **For Collab, spatial reduction (N_s) in general performs much better than temporal reduction (N_t), since Collab is densely connected,** for Imdb, the performance is similar which means optimal is between 1 and 10, for other datasets, since they are sparse, optimal T_N is low.
- **SP-$V_s N_t$-V_s performs well in most of the cases since parallelization of the vertices leads to reduced loading overhead**, however, as observed in most of the datasets, specially in Reddit-bin, Citeseer and Cora, **extremely high T_V can lead to delay as the performance is limited by the evil row as demonstrated in High-V_s-SP dataflow**
- **For PP dataflow, spatial or temporal reduction has more impact on runtime compared to the granularity of pipelining.** This is because pipelining delay is dominated by the slower phase and the non-overlapping extra delay is that of a faster phase.
- For all datasets SP-$F_s N_t$-F_s and PP-N_t-$V_{t/sl}$ have tilings such that PP-N_t-$V_{t/sl}$ computes both phases in parallel with half the feature tile size compared to SP-$F_s N_t$-F_s. **Yet both SP-$F_s N_t$-F_s and PP-N_t-$V_{t/sl}$ display different characteristics across different workloads.**

The impact of a mapping or a tile size is highly dependent on the graph structure.

C. Energy of GNN Dataflows

This subsection discusses the on-chip energy for GNN dataflows. Fig 12 shows the Energy consumed by various dataflows for various workloads. We assume the energy of a global buffer (GB) access to be 1.046pJ and the energy of a local buffer (L1) access to be 0.053pJ based on the energy model from Dally et al. [7]. Some of the observations related to energy are as follows:

- Energy is dominated by GB reads followed by L1 reads.
- Energy trends for T_N are similar to performance across applications. Collab favors high T_N while most of the others favor low T_N.
- SP usually exhibits low energy but that is not true for High-V_s-SP.
- Fig 13 shows the GB access breakdown for two examples, since GB energy is dominant. SP dataflow does not have Global Buffer Intermediate accesses, since the second phase uses the data locally in PE. SP dataflow has partial sum (psum) accesses because T_F features are processed at once and then phases are switched. When T_F is low in High-V_s-SP, increase in psum accesses compensates for the reduction in Intermediate data accesses.
- In Mutag, where the number of features is low, input feature map reads in the Agg phase dominate but in Citeseer where the number of features is even higher than the number of vertices, weight reads in the Cmb phase dominate.
- High T_V_{CMB} reduces the Weight Read energy considerably due to spatial reuse of vertices in DenseGEMM.

D. Takeaways, Insights and Architectural Vision

Graph datasets have highly irregular structures which even amongst datasets are different. We have observed in the evaluations using both synthetic and real workloads that tile sizes are critical to performance and energy. In terms of energy, SP-$V_s N_t$-V_s works well, however there is an optimal T_V for which the energy is the best. In terms of runtime, SP works well for low F (but there is an optimal T_V) but PP works well for high F (RedditBIN, Cora, Citeseer) with high T_V_{CMB}. For runtime and energy, there is an optimal T_V in SP dataflow. Parallelizing V dimension improves latency, however the performance can be limited by evil row if T_V is too large. From energy perspective, high T_V reduces the weight reads but also increases the partial sum accesses (due to low T_F) as shown in Fig 13.

The choice of desired metric influences the design choices. To achieve optimal energy, SP dataflow with flexible distribution network for variable T_V, T_F is a good choice. For performance, the optimal inter-phase dataflow is workload dependent. For acceleration of all GNN workloads, we need a flexible accelerator substrate for GNNs with an ability to choose various intra-phase parameters like tile sizes and also with the ability to model inter-phase dataflows - sequential pipeline and parallel pipeline. Note-By "flexibility" we mean configurability of tile-sizes and partition sizes, not Turing-completeness like CPUs/GPUs.

Our vision for the flexible accelerator is as follows:

- A flexible Network-On-Chip (NoC) for Intra-phase dataflows which adds the ability to support reduction and distribution of variable tile sizes for each dimension like in MAERI [24] and SIGMA [33]. This will always be helpful whatever the inter-phase dataflow since intra-phase metrics depend on tile sizes.
- Supporting variable sized splitting of a substrate into two phases for parallel pipelining which requires a flexible NoC and homogeneous substrate.
- The ability to support different inter-phase dataflows, a high-level architecture of which is shown in Fig. 14
- The ability to turn off intermediate buffers when not needed for EnGN like SP dataflow to save energy.

VI. Related Work

The acceleration of GNN workloads is an active area of research that distinguishes between software and hardware acceleration [2]. On the one hand, software acceleration for GNNs aims at exploiting the knowledge of the graph properties to better adapt the workload to the underlying hardware [4], [10], [14], [16], [20], [28], [34]–[37], [45]. This includes

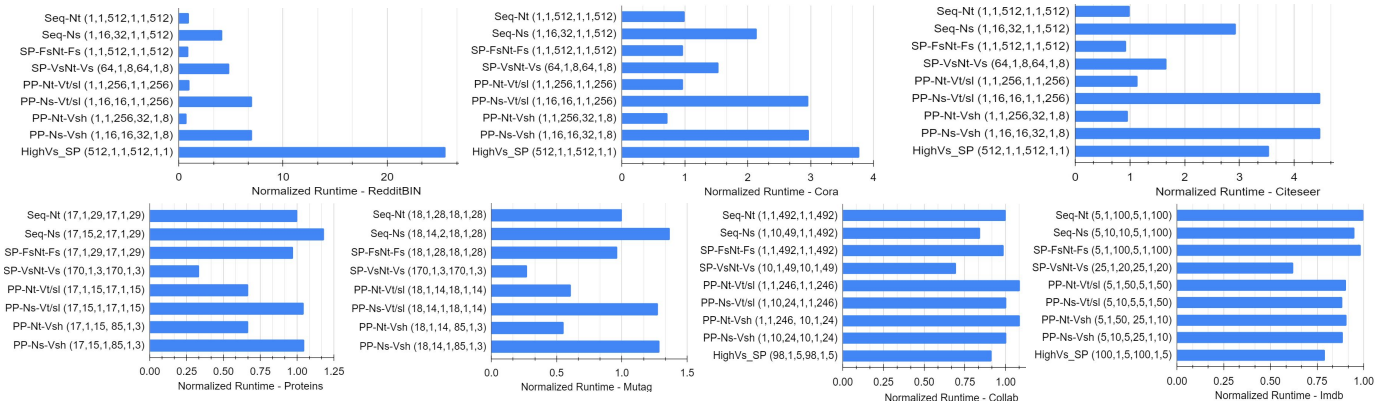


Fig. 11. Runtimes of Dataflows normalized to Seq-Nt for GCN algorithm. Tile sizes are represented as (T_V_{AGG},T_N,T_F_{AGG},T_V_{CMB},T_G,T_F_{CMB})

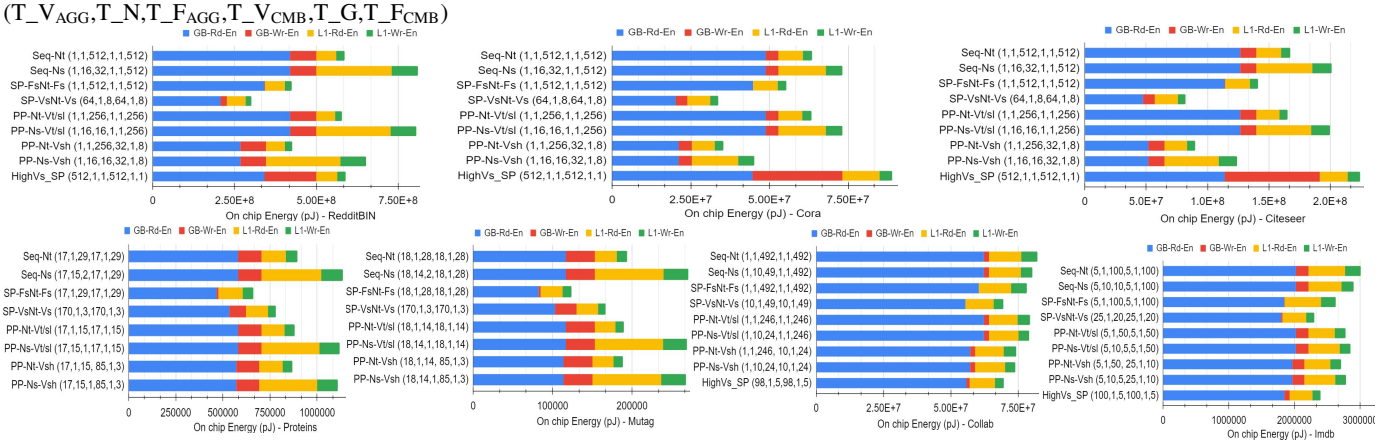


Fig. 12. On-Chip Buffer access energies of Dataflows, GB is Global Buffer and L1 is local (PE) buffer

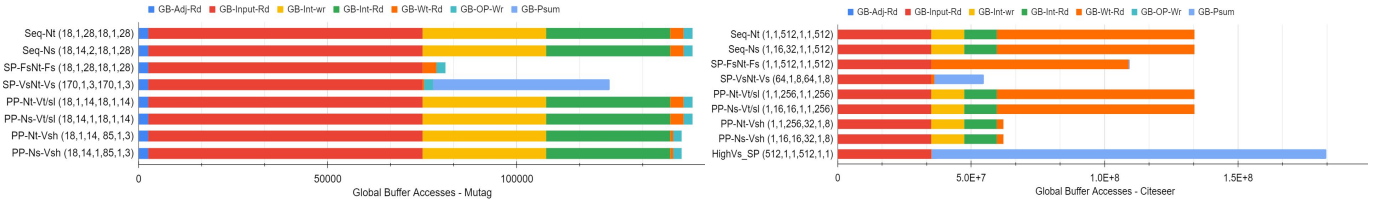


Fig. 13. Global Buffer breakdown for Mutag and Citeseer, Adj-Adjacency matrix, Inp-Input matrix, Int-Intermediate matrix, Wt-weight matrix and Op-output matrix, Psum-partial sum accesses

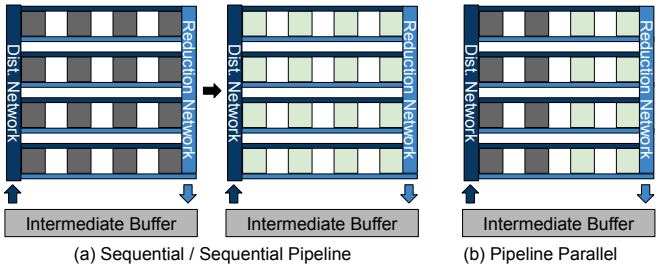


Fig. 14. Supporting multiple inter-phase dataflows in an accelerator.

techniques such as intelligent partitioning [34], sparsity-aware workload management [37], vertex reordering [4], or the caching of partial aggregations to avoid redundant sums [16]. These techniques are either specific for GPUs, such as the dataflow constructs in Neugraph [28], or orthogonal to the dataflow approach.

On the other hand, hardware accelerators employ specific architectures to combat the alternating dense–sparse phases of GNNs and to adapt to their time-varying needs [3], [4], [11], [21], [27], [39], [42]. The existing accelerators tackle the challenge in various ways. For instance, EnGN [27] relies on a unified architecture to perform both aggregation and combination efficiently, whereas HyGCN [39] utilizes two separate engines to that end. Centaur [15] tackles both dense and sparse computations, although not aiming directly at GNNs. Finally, we highlight GRIP [21] as being among the first accelerators to explicitly consider edge updates. Each of these hardware accelerators inherently implements a given dataflow, but the choice is generally fixed and not formally stated. This is, to the best of our knowledge, none of the listed works formalize the taxonomy for the dataflows for

GNN accelerators. Recently, GCNAX [26] proposed a flexible accelerator with Design Space Exploration to choose the ideal dataflow for a workload. Recently, there has also been work on analytical modelling of data movement for GNN accelerators [12].

VII. Conclusion

GNN accelerators are recently becoming extremely popular. Each accelerator has picked a specific dataflow to stage the GNN computations on the accelerator's compute and memory resources. To date, there exists no systematic mechanism to qualitatively and quantitatively compare different accelerators. This is the first work to propose a taxonomy to describe the space of dataflow choices for GNN accelerators. The taxonomy captures intra- and inter- phase dataflows and then accurately analyzes the hardware requirements and performance characteristics. We envision this work paving the way for systematic study and design of future GNN accelerators.

Acknowledgements

Support for this work was provided through the ARIAA co-design center funded by the U.S. Department of Energy (DOE) Office of Science, Advanced Scientific Computing Research program. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

[1] "Nvidia nsight," in *https://developer.nvidia.com/nsight-visual-studio-edition*, 2018.

[2] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón, "Computing graph neural networks: A survey from algorithms to accelerators," *arXiv preprint arXiv:2010.00130*, 2020.

[3] A. Auten, M. Tomei, and R. Kumar, "Hardware Acceleration of Graph Neural Networks," *DAC*, 2020.

[4] X. Chen, Y. Wang, X. Xie, X. Hu, A. Basak, L. Liang, M. Yan, L. Deng, Y. Ding, Z. Du *et al.*, "Rubik: A hierarchical architecture for efficient graph learning," *arXiv preprint arXiv:2009.12495*, 2020.

[5] Y. Chen, T. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[6] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 367–379. [Online]. Available: https://doi.org/10.1109/ISCA.2016.40

[7] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Commun. ACM*, vol. 63, no. 7, p. 48–57, Jun. 2020. [Online]. Available: https://doi.org/10.1145/3361682

[8] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[9] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The World Wide Web Conference*, 2019, pp. 417–426.

[10] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

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

[12] R. Guirado, A. Jain, S. Abadal, and E. Alarcón, "Characterizing the communication requirements of gnn accelerators: A model-based approach," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021.

[13] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[14] Y. Hu, Z. Ye, M. Wang, J. Yu, D. Zheng, M. Li, Z. Zhang, Z. Zhang, and Y. Wang, "Featgraph: A flexible and efficient backend for graph neural network systems," *arXiv preprint arXiv:2008.11359*, 2020.

[15] R. Hwang, T. Kim, Y. Kwon, and M. Rhu, "Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations," *ISCA*, 2020.

[16] Z. Jia, S. Lin, R. Ying, J. You, J. Leskovec, and A. Aiken, "Redundancy-free computation graphs for graph neural networks," *KDD*, 2020.

[17] J. Jiang, J. Chen, T. Gu, K.-K. R. Choo, C. Liu, M. Yu, W. Huang, and P. Mohapatra, "Anomaly detection with graph convolutional networks for insider threat and fraud detection," in *MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM)*. IEEE, 2019, pp. 109–114.

[18] N. P. Jouppi, , C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[19] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann, "Benchmark data sets for graph kernels, 2016," *URL http://graphkernels.cs.tu-dortmund.de*, vol. 795, 2016.

[20] K. Kiningham, P. Levis, and C. Re, "GReTA: Hardware Optimized Graph Processing for GNNs," in *Proceedings of the Workshop on Resource-Constrained Machine Learning (ReCoML 2020)*, March 2020.

[21] K. Kiningham, C. Re, and P. Levis, "GRIP: A Graph Neural Network Accelerator Architecture," pp. 1–14, 2020. [Online]. Available: http://arxiv.org/abs/2007.13828

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR*, 2017.

[23] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2019, pp. 754–768.

[24] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: enabling flexible dataflow mapping over dnn accelerators via programmable interconnects," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2018, p. 461–475.

[25] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[26] 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)*, 2021, pp. 775–788.

[27] S. Liang, Y. Wang, C. Liu, L. He, L. Huawei, D. Xu, and X. Li, "Engn: A high-throughput and energy-efficient accelerator for large graph neural networks," *IEEE Transactions on Computers*, 2020.

[28] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai, "Neugraph: Parallel deep neural network computation on large graphs," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 443–458.

[29] F. M. Matrínez, J. L. Abellán, M. E. Acacio, and T. Krishna, "Stonne: A detailed architectural simulator for flexible neural network accelerators," *arXiv preprint arXiv:2006.07137v1*, Jun. 2020.
[30] M. Miwa and M. Bansal, "End-to-end relation extraction using lstms on sequences and tree structures," *arXiv preprint arXiv:1601.00770*, 2016.
[31] A. Parashar, P. Raina, Y. S. Shao, Y. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.
[32] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun, "3d graph neural networks for RGBD semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5199–5208.
[33] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 58–70.
[34] C. Tian, L. Ma, Z. Yang, and Y. Dai, "PCGCN: Partition-Centric Processing for Accelerating Graph Convolutional Network," *Proceedings of IPDPS 2020*, pp. 936–945, 2020.
[35] A. Tripathy, K. Yelick, and A. Buluc, "Reducing communication in graph neural network training," *arXiv preprint arXiv:2005.03300*, 2020.
[36] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *arXiv preprint arXiv:1909.01315*, 2019.
[37] Y. Wang, B. Feng, G. Li, S. Li, L. Deng, Y. Xie, and Y. Ding, "GNNAdvisor: an efficient runtime system for gnn acceleration on gpus," *arXiv preprint arXiv:2006.06608*, 2020.
[38] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.
[39] 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)*, 2020, pp. 15–29.
[40] M. Yan, Z. Chen, L. Deng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Characterizing and understanding GCNs on GPU," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 22–25, 2020.
[41] W. Yan, W. Tong, and X. Zhi, "FPGAN: an FPGA accelerator for graph attention networks with software and hardware co-optimization," *IEEE Access*, vol. 8, pp. 171 608–171 620, 2020.
[42] H. Zeng and V. Prasanna, "GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms," *FPGA 2020 - 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 255–265, 2020.
[43] B. Zhang, H. Zeng, and V. Prasanna, "Hardware Acceleration of Large Scale GCN Inference," in *Proceedings of the ASAP '20*, 2020, pp. 61–68.
[44] Z. Zhang, P. Cui, and W. Zhu, "Deep Learning on Graphs: A Survey," *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 8, pp. 1–1, 2020.
[45] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "AliGraph: A comprehensive graph neural network platform," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2094–2105, 2018.