Understanding the Design Space of Sparse/Dense Multiphase Dataflows for Mapping Graph Neural Networks on Spatial Accelerators

Raveesh Garg¹, Eric Qin¹, Francisco Muñoz-Matrínez², Robert Guirado³, Akshay Jain⁴, Sergi Abadal⁵, José L. Abellán⁵, Manuel E. Acacio⁶, Eduard Alarcón⁶, Sivasankaran Rajamanickam⁶, and Tushar Krishna¹

¹Georgia Tech, {raveesh.g, ecqin}@gatech.edu, tushar@ece.gatech.edu
²Universidad de Murcia, {francisco.munoz2, meacacio}@um.es
³Universitat Politècnica de Catalunya, {rguirado, abadal}@ac.upc.edu, eduard.alarcon@upc.edu
⁴Neutroon, akshay.jain@neutroon.com
⁵Universidad Católica de Murcia, jlabellan@ucam.es
⁶Sandia National Laboratories, srajama@sandia.gov

Abstract

Graph Neural Networks (GNNs) have garnered a lot of recent interest because of their success in learning representations from graph-structured data across several critical applications in cloud and HPC. Owing to their unique compute and memory characteristics that come from an interplay between dense and sparse phases of computations, the emergence of reconfigurable dataflow (aka spatial) accelerators offers promise for acceleration by mapping optimized dataflows (i.e., computation order and parallelism) for both phases. The goal of this work is to characterize and understand the design-space of dataflow choices for running GNNs on spatial accelerators in order for the compilers to optimize the dataflow based on the workload. Specifically, we propose a taxonomy to describe all possible choices for mapping the dense and sparse phases of GNNs spatially and temporally over a spatial accelerator, capturing both the intra-phase dataflow and the inter-phase (pipelined) dataflow. Using this taxonomy, we do deep-dives into the cost and benefits of several dataflows and perform case studies on implications of hardware parameters for dataflows and value of flexibility to support pipelined execution.

1. Introduction

Recently, there has been an emergence of several general-purpose programmable spatial accelerators (aka Reconfigurable or Flexible Dataflow Accelerators [59]) targeting machine learning and HPC. Commercial examples include Cerebras [45], Graphcore [1], and SambaNova [14], and academic prototypes include Plasticine [42] and MAERI [33]. While low-level technological and microarchitectural details vary, at a high-level these accelerators are comprised of a "spatial" array of processing elements (PEs), a private register file (RF) in each PE, a programmable scratchpad-based memory hierarchy, and specialized networks-on-chip (NoC) for operand distribution and output collection to/from the PEs. Figure 1 shows an example.

A key distinguishing feature of spatial accelerators from conventional CPUs and GPUs is the ability to support dataflow execution [42]. In the spatial accelerator domain, the term dataflow is used to refer to the mechanisms used by accelerators to stage the computation across space and time [7, 32]. A "good" dataflow is one that can maintain high utilization via efficient parallelization, and minimize data movement through the memory hierarchy via data reuse. The dataflow along with tile sizes is known as a mapping [32]. Flexible accelerators support diverse mappings which can be statically configured by a compiler. The space of dataflows for executing dense DNN layers sequentially has been explored heavily [7, 32, 41, 55]. However, several ML and HPC kernels employ multiple phases of sparse-dense computations [9, 11, 31, 40], offering opportunities for pipelining and reuse across phases. The focus of this work is to characterize and understand the dataflow choices for such multiphase kernels that can be leveraged by a mapping optimizer. Specifically, we focus on understanding dataflow choices for Graph Neural Networks (GNNs), though our analysis can extend to other dependent multiphase computations as well.

GNNs are becoming increasingly popular because of their ability to accurately learn representations from graph structured data to solve graph/node classification, graph generation, and link prediction problems [50]. Example applications include item recommendation [15], molecular feature extraction [13], 3D classification tasks [17], natural language processing [37], semantic segmentation [43], and fraud detection [25]. The diversity of these problem domains results in very different workload characteristics.

GNN inference runtime is dominated by two phases [52]: (1) aggregation and (2) combination. Aggregation is an SpMM computation with irregular, workload dependent accesses of data. Combination computations can be cast as GEMMs, similar to traditional dense DNNs. Several recent works on GNN
acceleration [4,6,18,30,35,52,53,57,58] have demonstrated the challenges with getting high performance for GNNs from commodity CPUs, GPUs and DNN accelerators. These challenges arise from (i) extremely high amounts of sparsity in the aggregation phase (over 99% as compared to 70-80% in modern DNNs), and (ii) diverse data reuse opportunities within and across these phases. These works have also demonstrated that GNNs offer opportunities for acceleration by crafting specialized dataflows for extracting reuse both within and across the phases. Unfortunately, most of these prior works propose building specialized GNN accelerator ASICs with heavy co-design of a specific GNN dataflow and its hardware microarchitecture which limits their applicability as Graph datasets and GNN algorithms evolve. In contrast, this is the first work, to the best of our knowledge, that provides a qualitative and quantitative analysis of the design-space of various inter-phase and intra-phase dataflow strategies for mapping GNNs over flexible hardware accelerators.

We start by proposing a succinct taxonomy to classify various inter-phase and intra-phase dataflows to describe the possible pipelining strategies employed by GNN accelerators. This taxonomy expresses: (1) Aggregation intra-phase dataflow, (2) Combination intra-phase dataflow, (3) Inter-phase strategy, and (4) phase ordering. Rather than target a specific GNN accelerator where the dataflow choices might be limited due to its microarchitecture [18, 35, 52] or directly compare two different GNN accelerators where it would become hard to tease apart the performance difference due to dataflow versus microarchitecture, we target a templated flexible spatial accelerator substrate [33, 42] over which any dense/sparse dataflow that our taxonomy describes can be run. Using our taxonomy, we explore various factors that affect the runtime and energy of the dataflow strategies such as spatial/temporal mapping choices of various dimensions in the intra-phase, the pipelining granularity in the inter-phase, and tile sizes. We study the impact of graph properties (such as number of vertices, edges, features) on dataflow choice. We also analyze the impact of hardware parameters, for example low distribution and reduction bandwidth and issue of load balancing in pipelining. We also make a case for flexible accelerators as opposed to GNN accelerator ASICs given the interdependence of dataflows.

A summary of the key contributions of this paper is:

- We develop a taxonomy to characterize and enumerate the design space of dataflow choices for mapping multi-

---

Note: By "flexibility" we mean configurability of tile-sizes and PE-partition sizes, not Turing-complete programmability like CPUs/GPUs.

---

2. Background

2.1. GNN Computation

An input graph is represented by $G(V, E)$ with $V$ vertices and $E$ edges. Graph can also be represented as an adjacency matrix which represents the connectivity of the graph. Matrix representations of graphs are common, since dense/sparse matrix

Figure 2: Map space of our work on GNN dataflow studies is a super-set of that of prior works on DNN dataflow studies - Kwon et al. [32], Eyeriss [7], Timeloop [41] and Interstellar [55].
multiplication have been prime targets of spatial accelerators.

Recent works show that the main computation bottlenecks of various GNN algorithms like GCN [31], GraphSage [20], GINConv [51], and DiffPool [56] can be broken down into two phases - Aggregation and Combination [52]. GNNs allow either phase to precede the other while some algorithms like GraphSAGE perform aggregation before combination. Aggregation phase involves addition of the feature vectors of the neighboring nodes in the graph. This leads to irregular accesses of data 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. Aggregation phase can be mapped as sparse matrix - dense matrix multiplication (SpMM) with the highly sparse adjacency matrix being the main source of costly irregular accesses. Combination phase is a feature reduction phase. Unlike Aggregation, it is dense with regular GEMM computation.

Figure 3a shows an example graph with five vertices and eleven edges, including self loops. Figure 3b is the Compressed Sparse Row (CSR) representation of the graph’s adjacency matrix, as shown on the left of Figure 3c. For CSR representation, the neighbors of a particular vertex are stored back-to-back. Because most graphs are extremely sparse (> 99% sparsity) [28], CSR is often used as the graph representation [18, 52], and is what we assume in this work. As labeled in Figure 3c, A represents the adjacency matrix, X represents the feature matrix, and W represents the weight matrix. Any computation with A is a part of the Aggregation phase, and any computation with W is a part of the Combination phase. Other variables include: \( V \) (# vertices), \( F \) (# input features), \( G \) (# output features), and \( N \) (# of neighbors of a vertex).

2.2. Spatial Accelerators

Microarchitecture. Spatial accelerators are built as arrays of processing elements (PE). Each PE houses a multiply-and-accumulate (MAC) unit and a private register file (RF). The PE array is backed by a shared scratchpad memory [7, 26, 33], which we refer to as Global Buffer in this work. While early accelerators targeted dense GEMM/CONV2D operations [2, 7, 12, 26, 33], recent accelerators also target sparse convolutions, SpGEMM and SpMM [8, 21, 22, 44]. All accelerators provide efficiency by enabling reuse of data either spatially (i.e., via multicast) or temporally (i.e., via scratchpad memories).

Dataflows. 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 [7, 32]. 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) [32]. Figure 4 shows an example of a 2D spatial array running a “weight stationary” [7] dataflow for GEMM. Tiles (portion of matrix mapped onto the PEs at a time) of the weight (KN) matrix are mapped spatially onto the PE array and remain stationary. Tiles of the input (MK) matrix are streamed over the PEs, reusing the weight tiles. Tiles of the output matrix are generated by spatial reductions across the K dimension. In Section 3 we discuss all possible dataflows for GEMM which forms the combination phase for GNNs.

3. GNN Dataflow Taxonomy

In this section, we discuss GEMM/ SpMM dataflows within an individual phase (Intra-phase dataflows) and the overall dataflow with both the phases (Inter-phase dataflows).

Notation. We use the notations from Figure 3 for the aggregation and combination matrices for the rest of the paper. Table 1 presents the succinct notations we will use for describing dataflows, without writing out its full loop-nest. It also discusses the implication of the loop order and spatial (i.e., parallelism dimension) on data movement. Further, in the paper, 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 e.g., \( T_F \) is the number of input features (F) processed in parallel across PEs.

3.1. Intra-phase Dataflow

3.1.1. Combination Dataflow. Figure 5 (right side in (a) and bottom in (b)) shows examples of combination dataflow. In Figure 5(a), \( V \times F \) matrix is loaded into the RFs from the buffer and streamed from the RFs. \( F \times G \) matrix is streaming from

Figure 4: “Weight stationary” dataflow for GEMM.

Table 1: Intra-phase Dataflow choices and their hardware implication. The order of dimensions within {} can be interchanged. The subscript s on two dimensions represents spatial mapping (i.e., unrolling) of those dimensions across the rows and columns of the accelerator (in a tiled manner), while t represents temporal mapping. We show the choices for combination (VGF). The same choices exist for aggregation (VFN).

| Dataflow          | Implication of Loop Order | Implication of Spatial Dimensions |
|-------------------|---------------------------|-----------------------------------|
| \( V, G, F \)      | Output (VG) stationary, Intermediate matrix (VF) and Weights (FG) stream every cycle | Spatial multicast of VF and FG every cycle. Temporary reduction of partial sums within each PE |
| \( G, F, V \)      | Weight (FG) stationary, Intermediate matrix (VF) streams every cycle | Spatial multicast of VF every cycle. Spatial reduction of partial sums across PEs |
| \( V, F, G \)      | Intermediate matrix (VF) stationary, Weight (FG) streams every cycle | Spatial multicast of FG every cycle. Spatial reduction of partial sums across PEs |
Intermediate storage

the buffer into the PEs. The input features vary each cycle; and thus the dimension is temporally mapped. The other dimensions are spatial since they have parallelism, resulting to $V_sG_tF_t$. 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 reduction is temporal which can be achieved by accumulators inside the MACs. As another example, Figure 5(b) shows combination dataflow in which different features are mapped in parallel. $V \times F$ matrix is present in the RF. $F \times G$ matrix is streamed from the buffer into the PEs. Different partial sums resulting from different features “a-h” are reduced spatially. Note that spatial reduction can be done through a linear chain or adder tree [32]. This results in $V_sF_sG_s$.  

3.1.2. Aggregation Dataflow. Figure 5 (left side in (a) and on top in (b)) illustrate an example of aggregation dataflow (same example in both sub-figures). Sparsity gets encoded as N in

taxonomy. As shown in Figure 3b, N can be encoded in CSR, in which all of the neighbors are stored back-to-back. The taxonomy is agnostic to the amount of sparsity. Encoding the amount of sparsity will lose the generalizability on different graphs, as the amount of sparsity is workload dependent, not taxonomy dependent. The example shows V (vertex dimension) as the outermost loop, and is temporal. This means that the next vertex starts after the current vertex finishes along with their inner loops. In the next loop, F (features) is spatial. In the innermost loop, N is temporal, thus temporal reduction is required. Finally, for our taxonomy, we define Aggregation dataflow Figure 5 as $V_sF_sN_s$. In Figure 5, 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. In addition to the examples shown in Figure 5, other dataflows are possible with different combinations of temporal and spatial loops orderings as shown in Table 1.

3.2. Inter-phase Dataflow

Although each phase can use any intra-phase dataflow, as described above, the dataflow choice for one can affect the other. This opens up a design-space for inter-phase dataflow. This part of the dataflow is important as it determines the number of memory accesses required to move data from one phase to the next. Figure 6 presents the types described below. For the description, we focus on an Aggregation-to-Combination computation order.

2.1. Sequential (Seq). Seq is the simplest inter-phase dataflow where the two phases are run sequentially. All outputs of the first phase are stored in the global buffer (and moved to DRAM if the size of the global buffer is insufficient)
and then loaded back to the PEs for the next phase.

3.2.2. Sequential Pipeline (SP). SP is similar to sequential but splits aggregations and combinations into small stages, which are then interleaved over time on the accelerator’s PEs. Figure 5(b) shows SP inter-phase dataflow. Depending on the intra-phase dataflows, it is possible to reduce data movement between aggregation and combination stages as shown in Figure 6. Specifically, the output data of one stage can be kept stationary within PEs’ local registers rather than going into a global buffer or DRAM as shown in Figure 5(b). We discuss this in more detail in Section 4.2.

3.2.3. Parallel Pipeline (PP). PP is when both phases are allocated onto parallel units (i.e., group of PEs) within an accelerator at the same time. The size of these parallel units can be same (e.g., HyGCN [52]) or flexible (e.g., AWB-GCN [18]), depending on the flexibility within the accelerator substrate. For this dataflow, it is critical to balance the production and consumption rate to reduce stalls. An intermediate ping-pong buffer and network-on-chip (NoC) is needed to send data from one stage to the next. This is similar to the typical workflow for a more general accelerator such as GraphCore. Figure 5(a) shows an example of PP dataflow with left half performing Aggregation and right half performing combination.

3.3. Complete Description of GNN Dataflow

Given the dataflow types described above, we use the following template to describe a complete dataflow:

\[ \text{<Inter>}_{\text{order}} \cdot \text{<AggIntra>} \cdot \text{<CmbIntra>} \]

- \( \text{<Inter>} \) represents the inter-phase dataflow, \( \text{<order>} \) represents the computation order (aggregation to combination is AC, combination to aggregation is CA), \( \text{<AggIntra>} \) represents the aggregation phase intra-phase dataflow, and \( \text{<CmbIntra>} \) represents the combination phase intra-phase dataflow.

Table 2 uses our taxonomy to enumerate and characterize the space of dataflow choices for mapping the aggregation and combination loops, including the hardware structures required for supporting that dataflow. To keep the table compact, we add a subscript \( \alpha \) to indicate that the dimension could be spatial or temporal while curly brackets \{ ... \} indicate that the dimensions can be interchanged for different loop orders. This leads to a total 6,656 choices purely from the product of all feasible loop orders, parallelism choices, and phase order across the three inter-phase choices. Note that for each dataflow choice, the tile sizes are also parameters as Section 5 discusses. This can put the actual number of possible mappings in the trillions [41]. We observe in Table 2 and Section 4 that loop orders and tile-sizes of two phases are interdependent. Thus we formalize the non-trivial space of multiphase GNN dataflows.

For the purposes of analysis, in Table 2 we explicitly list sets of interesting dataflows (some of which existing accelerators have leveraged) with the remarks highlighting their key characteristics. As an example, the HyGCN [52] accelerator’s dataflow can be succinctly described as: \( PP_{AC}(V, F, N_t, V, G, F_t) \) In fact, the dataflow shown in Figure 5(a) is same as that used by HyGCN [52]. However, the HyGCN microarchitecture employs separate dedicated SIMD and Systolic engines respectively for Aggregation and Combination, along with a dedicated buffer between them for intermediate values. The dataflow is tied to the microarchitecture; in contrast, Figure 5(a) runs the HyGCN dataflow on a programmable spatial accelerator by configuring different sets of PEs to run the aggregation and combination dataflows described earlier in Section 3.1 and staging the intermediate values through the partitions in scratchpads. Programmable spatial accelerator provides flexibility as opposed to original HyGCN microarchitecture with fixed dataflow.

Hardware support for dataflows. We would like to note that each dataflow may require different levels of support from the hardware. For the intra-phase dataflows, support for multicasts and reductions (either spatially via a store-and-forward or temporally via a read-modify-write register within PEs may be needed [32]) as Table 1 discusses. For the inter-phase dataflows, Table 2 lists the NoC and PE support needed for each dataflow. We discuss the implications of hardware parameters in detail in Section 5.

Scope of Taxonomy and Extensions. In the interest of space, we apply our taxonomy to study GCNs. Sec 6 discusses scope and applicability to other GNNs and HPC kernels.

4. Qualitative and Analytical Analysis

In this section, we present a qualitative analysis of the trade-offs of various dataflow choices and their runtime and storage implications. We analyze inter-phase dataflows in terms of runtime, intermediate global buffering support needed and feasible dataflows. We also focus on the intra-phase dataflow implications on inter-phase dataflows. Table 3 describes the runtime and intermediate buffering requirements for inter-phase dataflows that we derive in this section. Figure 6 shows the space-time representation of Inter-phase dataflows and the impact of Inter-phase dataflows on energy of intermediate data. We observe that dataflow parameters for both phases can be interdependent thus multiphase dataflow space is non-trivial. For the analysis, we focus on AC computation order.

4.1. Sequential Dataflow

Recall that 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 is simply the number of output elements which is \( V \times F \) data elements as shown in Figure 7a. Such amount of data cannot be stored on-chip for

\[ \text{For instance, to implement its PP dataflow (Row 5), HyGCN [52] uses several SIMD units for the aggregation phase, followed by a unidirectional high bandwidth bus to feed it to a 2D systolic array for the combination phase.} \]
Table 2: 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. Subscripts $s$, $t$, $x$ mean spatial, temporal, either spatial or temporal respectively. Curly brackets ‘{}’ show that the loop order is interchangeable.

| Row | Inter Phase | Order | Intermediate Global Buffer | No/CPE support | Example & Order | Remarks |
|-----|-------------|-------|-----------------------------|----------------|----------------|---------|
| 1   | Sequential (Seq) | ANY- $\{V_t, F_t, N_t\}$, $\{V_s, G_s, F_s\}$ | ✓ | Intra-phase | TPU [26] Eyeriss [8] | Similar to running one layer at a time. Outputs of one layer gets stored in scratchpad and rescheduled back onto PEs. |
| 2   | Sequential Pipeline (SP) | AC - $\{V_t, F_t\}$, $\{V_s, G_s\}$, $\{V_t, G_t\}$ F | ✓ | Local buffer inside PEs to accumulate data | EnGN [35] (any direction*) | No intermediate global buffer, as output data of one phase is stationary in the buffers, and can be used as input for the next phase. For agg -> cmb, order of VF should be same. $T_{V,AGG} = T_{V,CMB}$ and $T_{F,AGG} = T_{F,CMB}$ and reduction is temporal as data is always in-place buffers. |
| 3   | Parallel Pipeline (PP) | AC - $\{V_t, F_t\}$, $\{V_s, G_s\}$ | ✓ | No intermediate support needed | (any direction*) | There will be a buffer setup delay between aggregation and combination phases to remap output data to new location. |
| 4   | Parallel Pipeline (PP) | AC - $\{V_t, F_t\}$, $\{V_s, F_s\}$, $\{G_s\}$ | ✓ | NoC connecting Agg and Cmb units to intermediate buffer. | (agg -> cmb) | Element(s) wise granularity: Element(s) of the intermediate matrix indexed by VF can be pipelined. The order of VF should be same for both phases. |
| 5   | Parallel Pipeline (PP) | AC - $\{V_t, F_t\}$, $\{V_s, G_s\}$ | ✓ | NoC connecting Agg and Cmb units to intermediate buffer. | (agg -> cmb) | Row(s) wise granularity: Row(s) of intermediate matrix indexed by V can be pipelined. The order should not be (VFN, VF). HyGCN allocates fixed number of PEs per unit, which may lead to stalls. E.g., combination engine idle while waiting. HyGCN dataflow: $PP_{HyGCN}(V,F,N,G)$ |
| 6   | Parallel Pipeline (PP) | AC - $\{V_t, F_t\}$, $\{V_s, G_s\}$ | ✓ | NoC connecting 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). |
| 7   | Parallel Pipeline (PP) | CA - $\{V_t, F_t\}$, $\{V_s, G_s\}$ | ✓ | NoC connecting Agg and Cmb units to intermediate buffer. | (cmb -> agg) | Element(s) wise granularity: The order should be (NFV, VGF) or (ENV, GF). $V$ -> G matrix after cmb becomes $N$ x $F$ for agg. |
| 8   | Parallel Pipeline (PP) | CA - $\{V_t, F_t\}$, $\{V_s, G_s\}$ | ✓ | NoC connecting 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. |
| 9   | Parallel Pipeline (PP) | CA - $\{V_t, F_t\}$, $\{V_s, G_s\}$ | ✓ | NoC connecting Agg and Cmb units to intermediate buffer. | (cmb -> agg) | Column(s) wise granularity: The order should not be (ENV, GFV). AWB-GCN enables flexible allocation of PEs for different phases to match production and consumption rates. AWB-GCN dataflow: $PP_{AWB-GCN}(F,N,V,G,F,V)$ |

**Figure 7:** Intermediate buffering in Inter-Phase dataflows – AC computation order is assumed.

Large graphs, as Figure 6 shows and would incur higher energy than the other inter-phase dataflows.

### 4.2. Sequential Pipeline (SP)

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 over time. As Figure 6 shows, this reduces the intermediate matrix footprint and naturally gives an energy advantage over Seq by avoiding expensive memory accesses down in the memory hierarchy (including DRAM).

**SP-Generic.** This can be considered a naïve implementation for the SP dataflow. For this dataflow, as shown in Figure 7a, the intermediate data produced in the aggregation phase is written by the PEs to the global buffer, and then read from there for computing the combination phase. Thus, the intermediate storage required is equal to the size of the number of elements pipelined which we define as $Pel$. The intermediate data is broken down into granularities which we describe in detail in Section 4.4. The total feasible number of loop orders is shown in row 3 of Table 2, which is equivalent to the number of PP loop orders shown in rows 4-9. The total runtime of SP-generic would be similar to the sequential dataflow, although the energy consumption would be lower by avoiding the transfer of the aggregation outputs to memory.

**SP-Optimized.** In order to save both energy and latency, we can select a subset of intra-phase dataflows for aggregation and combination such that the outputs generated by aggregation can stay local within the PEs’ registers and reused directly by the combination phase. This avoids writing this intermediate data up the memory hierarchy to the buffer and reading it again. These dataflows are $SP_{AC}(\{V_t, F_t\}N_t, \{V_t, F_t\}G_t)$ for Aggregation to Combination order and are shown in Row 2 of Table 2. In this dataflow, the requirement is that the loop order pair for
aggregation to combination phase should be \((\{VF\}N, \{VF\}G)\). After, one iteration of temporal loop-nest of VF, is finished with all neighbors reduced, the accumulated data remains in the MAC units and 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 (\(T_{N}=1\)). Recall from our notation that the order between V and F is interchangeable and either of them (or both) can be mapped spatially or temporally depending on the underlying accelerator. The advantage of SP-optimized is reduced buffer accesses and reduced buffer requirement, since the data to be used in the second phase is directly used inside the PEs, as also shown in Figure 7b. Moreover it saves the latency and memory read overhead of loading the data into PEs. An instance of this dataflow is used by the EnGN [35] accelerator. Rubik [6] also uses the SP-Optimized inter-phase dataflow. For Aggregation, it memoizes the partial aggregations in private cache and reuses them eliminating redundant computations.

### 4.3. Parallel Pipeline Dataflow

Parallel pipeline dataflow divides the accelerator into two engines, with one engine feeding the other engine spatially. The intermediate data is broken down into small granularities and the data is processed in a pipelined manner as shown in Figure 6. For example, if granularity is a single row of intermediate matrix, then the aggregation phase computes and writes the data corresponding to \(n^0\) row and in parallel, the combination phase reads and processes data corresponding to \((n-1)^0\) row. To facilitate this, intermediate buffers are required and the amount of intermediate buffering required is twice number of elements of the intermediate matrix pipelined. We refer to the number of elements being pipelined as \(Pel\). Thus the amount of intermediate storage used is \(2 \times Pel\) per each phase. The runtime of one pipeline iteration is equal to the runtime of the slower phase for producing \(Pel\) elements. The total runtime is the sum of runtimes of individual iterations \(\text{sum}(\max(T_{AGG}, T_{CMB}))\).

### 4.4. Granularities for SP-Generic and PP

We discuss three types of granularities at which the intermediate matrix can be broken into pipeline stages for PP and SP-Generic dataflows. Rows 4-9 in Table 2 shows all feasible loop orders for all possible granularities and phase orders.

**Element.** Element(s)-wise granularity involves tiles of few elements (\(Pel = T_{F} \times T_{V}\)) being processed in a pipelined manner rather than the whole dimensions.

Figure 8a shows an example of element(s)-wise granularity \(PP_{AC}(V_{f_{1}}F_{x}, V_{x}F_{G})\). 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, in parallel, the innermost loop of combination computes \(G\) for the previous tile of vertices and features.

If the tile sizes of dimensions are imbalanced for multiple phases, the number of elements that would be written in the intermediate buffer 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. The amount of buffering is \(2 \times Pel\) for PP datalow and \(Pel\) for SP-Generic.

**Row.** In row(s)-wise granularity, whole row(s) of the inter-
Table 3: Runtime and buffering requirements for Dataflows

| Inter-phase dataflow | Intermediate Buffering | Runtime |
|----------------------|-------------------------|---------|
| Seq                  | V×F                     | AGG+CBM |
| SP Generic           | V×F                     | AGG+CBM |
| SP-Optimized         | 0                       | AGG+CBM+PMB |
| PP-Element           | 2×V×F×max×F             | sum(max(AGG+CBM, F)) |
| PP-Row               | 2×V×F×max×F             | sum(max(AGG+CBM, F)) |
| PP-Column            | 2×V×F×max              | sum(max(AGG+CBM, F)) |

Intermediate matrix is (are) considered instead of a few elements in a row. The number of rows that are pipelined is \( T_{V_{\text{max}}} \). So \( Pel = T_{V_{\text{max}}} \times F \). Figure 8b shows an example loop order (VFN, VGF) for Row(s) wise granularity for PP dataflow 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).

**Column.** 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 \( T_{F_{\text{max}}} \). So \( Pel = T_{F_{\text{max}}} \times F \). Figure 8c shows an example loop order (FVN, FGV) for Column(s) wise granularity for PP dataflow 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.

Table 3 summarizes the runtime and buffering requirements for the different inter-phase dataflows discussed before.

## 5. Quantitative Analysis and Case Studies

In this section, we do a deep dive into the performance and energy of different GNN dataflows. We also perform case studies on implications of pipelining. We also discuss the costs and benefits of flexible accelerators for pipelined dataflows.

### 5.1. Experimental Methodology

#### 5.1.1. Cycle-Accurate Simulation Framework

In order to carry out detailed evaluation of various dataflows, we built a cycle-accurate simulation framework around STONNE simulator [39]. STONNE simulator models the flexible accelerators MAERI [33] and SIGMA [44], and the hardware models have been extensively validated against their RTLs. It consists of reconfigurable networks-on-chip for operands (e.g., inputs, partial sums and weights) distribution, operands multiplexation, and output reduction (e.g., addition) allowing us to study different tile sizes. The STONNE framework also supports CSR decoding and indexing logic to run SpMM in addition to GEMM to compute both phases. To implement inter-phase dataflows, we built an analytical model around the simulator that computes the runtime, buffering and energy statistics of a GNN layer based on the analysis in Section 4.

#### 5.1.2. Datasets

We evaluate the GNN dataflows for the target datasets described in Table 4. These are standard datasets representing workloads from multiple domains like biochemistry, citation networks and social networks [28, 54]. We evaluate one batch of 64 graphs for graph classification workloads (batch of 32 graphs for RedditBIN) and we evaluate node classification datasets Citeseeer and Cora. The large graph datasets are generally sliced to fit on-chip [35, 52]. Large graph classification datasets can be batched such that the graphs fit on-chip. For this work, there is sufficient on-chip buffering for a batch of graph classification datasets and for node classification datasets. We characterize on-chip data movement and runtime of the GNN dataflows since our aim is to study the behaviour of these dataflows on spatial accelerators. Based on matrix dimensions and sparsity, we divide the workloads into 3 categories: high number of edges/vertices and relatively low features/vertices (HE), high features/vertices and relatively lower edges/vertices (HF), and low edges/vertices and low number of features/vertices (LEF).

### 5.2. Comparison of dataflows

In this subsection, we compare the performance and energy of GNN dataflows. In the interest of space, we evaluate the representative dataflow configurations shown in Table 5. The inter-phase dataflows here are similar to EnGN and HyGCN (SP-Optimized and PP at row-wise granularity) respectively. For SP-Optimized dataflow, \( T_N=1 \), \( T_{V_{\text{AGG}}}=T_{V_{\text{CMB}}} \) and \( T_{F_{\text{AGG}}}=T_{F_{\text{CMB}}} \).

Table 4: Datasets information. First part for graph classification, bottom part for node classification – ‘*’ means that indicator vectors were used in place of features.

| Name  | #Graphs | #Nodes(avg) | #Edges(avg) | #Features | Category |
|-------|---------|-------------|-------------|-----------|----------|
| Mutag | 188     | 17.93       | 19.79       | 28*       | LEF      |
| Proteins | 1113 | 39.06       | 72.82       | 29 (full) | LEF      |
| Imdb-bin | 1000 | 19.77       | 96.53       | 136*      | HE       |
| Collab | 5000    | 74.49       | 2457.78     | 493*      | HE       |
| Reddits-bin | 2000 | 429.63      | 497.75      | 3782*     | HF       |
| Citeseer | 1    | 3327        | 9464        | 3703      | HF       |
| Cora   | 1       | 2708        | 10858       | 1433      | HF       |

### 5.2.1. Performance

Figure 9 shows the runtimes of various dataflows. Our observations are as follows:

- For Collab and Imdb (HE category) spatial reduction (\( N_s \)) in general performs much better than \( (N_t) \), because they
are densely connected. For other datasets, since they are sparse, optimal \( T_N \) is low.

- \( SP-V_N \) performs well in most of the cases since parallelization of the vertices leads to reduced redistribution overhead. However, for HF datasets extremely high \( T_V \) can lead to delays since the performance is limited by the dense row (with large number of non-zeros) as demonstrated in High-\( V_s \)-SP dataflow. Such dense row is referred to as "evil row" in AWB-GCN [18].

- Mutag and proteins have great performance despite extremely high \( V \), which shows that these workloads don’t have evil rows.

- For the Collab dataset, PP performs worse than Seq dataflow due to poor load balancing between Aggregation and Combination.

**Best Performance:** For HF category datasets, PP-\( N_t \) is best while for other datasets SP-\( V \) performs the best, although there is an optimal \( T_V \) for best performance.

### 5.2.2. Energy

Figure 10 shows the energy consumed by various dataflows across workloads. The energy difference comes due to differences in number of accesses across the memory hierarchy. We assume the energy of a global buffer (GB) access to be 1.046pJ (1MB/bank) and the energy of a local PE register file (RF) access to be 0.053pJ based on the energy model from Dally et al. [10]. For PP dataflow, we assume that there is a separate ping-pong buffer for intermediate data and the size of that depends on the capacity required based on Table 3. Our observations related to energy are as follows:

- Energy is dominated by GB reads followed by RF reads.
- SP and PP have lower energy compared to Seq dataflow.
since these dataflows do not incur high intermediate global buffer accesses.

- Figure 11 shows global buffer accesses. In Collab (HE category), input accesses dominate, in Cora (HF category), weight accesses dominate. Mutag (LEF category) extracts best reuse due to pipelining.
- In PP dataflow, even though the GB accesses are same as Seq dataflow, we observe lower energy since the energy of memory accesses from smaller intermediate buffer is less.
- SP-V SH-V has low energy but High-V SP has higher energy due to the partial sum accesses due to low T_F. Therefore there is a cost associated with SP-Optimized which is the overhead of partial sums. High T_F can help reduce the energy due to partial sums.

**Best Energy:** For HF category datasets, PP-N_V and SP-V has best energies. For HE workloads, SP-V has best energy, however energy saving by pipelining is not significant. For LEF workloads, SP-F has best energy due to low T_F.

**5.2.3. Costs associated with pipelining.** We observed the energy and performance benefits of inter-phase pipelining in Section 5.2.1 and Section 5.2.2. However, we also observed some dataflows become worse due to pipelining, thus the main costs of pipelining are as follows:

- **SP-Optimized:** While SP optimized dataflow re-uses the outputs accumulated locally in the PEs, since it does not always process all features, partial sums are generated instead of complete sums and have to be re-read once all the partial sums corresponding to an output are generated. This increases the energy as seen in Figure 11.
- **PP:** A major performance implication of PP is load balancing. Since this dataflow is over space, the delay is dictated by the worse stage of pipelining, therefore in some datasets like Collab, it is actually worse than Seq (Figure 9). Moreover, optimal allocation does not always lead to perfect load balancing due to unstructured sparsity.

**5.3. Case Study: Hardware Parameter Implications**

**5.3.1. PP: Load balancing.** In PP dataflow, the delay is decided by the slower phase. Therefore load balancing is critical. Figure 12 shows the performance of different allocations of PEs to phases for different pipeline granularities. Collab has higher density (HE category) hence slow Aggregation, therefore 25-75 performs poorly. For Mutag (LEF category), 50-50 is best allocation scheme amongst the three allocation schemes. Since, citeseer is sparse and has high number of features (HF category), combination phase is slower, therefore 75-25 allocation performs poorly.

**5.3.2. Implications of Low bandwidth.** For Section 5.2, we assumed sufficient on-chip distribution and reduction bandwidth to ensure that the data is received/sent without stalls. However, lower Global Buffer distribution and reduction bandwidth will lead to stalls and hence would affect the performance of dataflows where spatial reuse becomes less. Figure 13 shows the performance implications of reducing the bandwidth for different inter-phase dataflows. Runtime reduces with decrease in bandwidth and PP dataflow suffers the most since the bandwidth is shared between the two phases as shown in Figure 13.

**5.4. Case Study: Rigid vs Flexible Dataflow Accelerators**

In this section, we make a case for flexible dataflow accelerators as opposed to highly specialized GNN ASICs. These ASICs have fixed dataflows and tile sizes and their microarchitecture is tailored them. SP and PP dataflows are interdependent since the portion of the intermediate matrix produced by the first phase governs the second phase. As discussed in Section 4, there are certain loop orders and tile sizes which can take advantage of pipelining. Next, we discuss the flexibility features needed to support efficient pipelining.

**SP-Optimized:** For SP-Optimized dataflow, T_VAGG = T_CBCM and T_FAGG = T_CBCM. Also, in order to retain intermediate matrix locally in PE, T_N = 1. Therefore the Aggregation phase should have temporal reduction. A rigid architecture would have the same reduction tile size. For combination, T_CBCM determines spatial reduction. A rigid architecture with only spatial reduction cannot map SP-Optimized. A rigid architecture with only temporal reduction can map only one instance with tile sizes T_F = T_N = 1 which implies that only V is distributed per row. This dataflow is High-V SP. We observe from Figure 9 and 10 that this dataflow has a huge runtime due to performance being limited by the sparse rows, and a huge energy due to the overhead of writing and reading partial sums. Therefore, configurability of tile sizes is essential given the interdependence of phases. EnGN [35] uses two networks, however flexibility is more robust to workload changes.

**PP dataflow:** In case of PP dataflow, a critical factor for pipelining is the load balancing of phases. We observed in Figure 12 that depending on the workload, optimal allocation to a phase can change depending on workload sparsity and dimensions. This requires a flexible substrate. Using a rigid substrate with two distinct sub-accelerators like HyGCN [52] can lead to load imbalance for certain workloads. Load balancing also requires an appropriate choice of tile sizes for both the phases to minimize stalls due to the slower phase. Therefore flexible resource allocation and configurability of tile sizes are essential for mapping PP efficiently.

**Cost/benefit of flexibility:** Mapping SP and PP efficiently dataflow require configurable tile sizes and flexible PE allocation. Moreover as shown in Section 5.2, flexibility to choose from SP and PP according to the workload leads to optimal dataflow. All these features add up to a programmable spatial accelerator. However, there is no additional cost for a programmable spatial accelerator running pipelined dataflows compared to running single phase dataflows. Therefore flexibility provides more benefit per cost for mapping multiphase kernels than independent kernels.
Qualitative Analysis of GraphSage dataflows

Our taxonomy in Section 3 focuses on GCN [31]. We briefly describe the design space of GraphSage [20] dataflows. The key differences between GraphSage and GCN are:

1. GraphSage samples the neighbourhood of a vertex for fixed number of neighbours as a result, as a result, adjacency matrix follows structured sparsity. While this does not affect the design space description, perfect load balancing can be achieved statically in PP and fixed $T_N$ would not incur underutilization, since number of neighbours is same.

2. GraphSage combination phase involves matrix multiplication of weight ($W$) with concatenated input feature map ($X$) and intermediate matrix (A.$X'$) ($V \times 2F$). Thus dimensions of $W$ are $2F \times X$. GraphSage combination can be described as: $X'_{l+1} = (concat(X,A,X'))$, $W = X.W_{top} + (A,X).W_{bottom}$

The concat operation is reading the $X^l$ for the top half $F \times G$ features of $W$ ($W_{top}$) and reading $A.X^l$ for reading the bottom half $F \times G$ features of $W$ ($W_{bottom}$). We can either concatenate the intermediate and input matrices first and compute a large GEMM or we could consider two GEMMs separately and add the partial sums later. Thus there is an additional knob in the combination phase, order of concatenation and GEMMs. Performing concatenation first constrains the dataflow since execution of the whole combination depends on the aggregation phase. Performing GEMM first on the contrary is extremely flexible since $X.W_{top}$ can be scheduled independent from Aggregation thus transforming the design-space to three operations. This flexibility would be beneficial. For example, $X.W_{top}$ can be allocated flexibly to balance the load in PP or computations of Aggregation and both GEMMs can be interleaved taking full advantage of SP dataflow reuse between Aggregation and Combination and between the two GEMMs of combination. Thus GraphSage opens up another optimization knob compared to GCNs.

6. Extensions and Future work

Scope of the Taxonomy

The present representation distinguishes the intra-phase dataflows that can be used for vertex aggregation and combination, as well as dataflows between both phases of a GNN. However, the loop transformations and inter-phase pipelining in our taxonomy also does not capture order of nodes and graph partitioning. Further, our taxonomy does not capture accelerator-specific microarchitecture optimizations for GNNs such as load balancing [18], window shrinking [52], computation elimination via memoizing [6, 24]. Capturing all this within the compiler is exciting future work.

Application of Taxonomy to Other Kernels beyond GNNs

- Though this work focuses on GNNs, the taxonomy and inter-phase analysis proposed in Sections 3 and 4 can be generalized to dataflows for multiphase computations (GEMM-GEMM/SPMM/SpMM-SPMM). One immediate example is Deep Learning Recommendation Models [40] that is built of a SpMM and DenseGEMM in parallel followed by concatenation followed by DenseGEMMs.

Mapping Optimizer

In this work, we performed case studies on select dataflows to demonstrate interesting insights. A mapping optimizer can be built on top of the inter-phase cost model and STONNE simulator [39] which automatically searches the search space of the dataflows. There are existing mapping optimizers for DNN accelerators [5, 27, 55]. Since GNN dataflows add an additional inter-phase optimizations knob, dataflow search is important.

7. Related Work

Dataflow Analysis

Prior dataflow works (such as Timeloop [41], Interstellar [55], and Kwon et al [32]) only analyze single-phase dense dataflows and are a subset of dataflows that we study as shown in Figure 2.

GNN Acceleration: Software Techniques

Software acceleration for GNNs aims at exploiting the knowledge of the graph properties to better adapt the workload to the underlying hardware [3, 6, 16, 23, 24, 29, 36, 46–49, 60]. This includes techniques such as intelligent partitioning [46], sparsity-aware workload management [49], vertex reordering [6], or the caching of partial aggregations to avoid redundant sums [24]. These techniques are either specific for GPUs, such as the dataflow constructs in Neugraph [36], or orthogonal to the approach of this paper.

GNN Acceleration: Hardware Techniques

Each GNN accelerator implements a specific dataflow, which is generally fixed and not formally defined [3, 4, 6, 18, 19, 30, 34, 35, 52, 57]. Recently, GCNAX [34] proposed a flexible accelerator with Design Space Exploration to choose the most energy efficient...
dataflow for a workload. GCNAX primarily targets off-chip dataflows and considers small Global buffers and 16 PEs, while our work primarily focuses on the on-chip dataflow strategies for large programmable spatial accelerators.

8. Conclusion
With increasing popularity of multiphase workloads like GNNs, dataflow strategies to map them on accelerators and extract reuse both across and within the phases is crucial. While there has been prior work on dataflow exploration for dense DNNs, GNNs are a wider generalization of DNNs since they consist of sparse and dense phases. We capture the design space of GNN dataflows in a succinct taxonomy template. Using this taxonomy, we contrast various GNN dataflows and perform various case studies on pipelined dataflows.

Acknowledgements
We thank Prasanth Chatarasi for his insightful feedback. 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] Graphcore IPU. https://www.graphcore.ai/products/ipu.
[2] NVidia deep learning accelerator. http://nvidia.org, 2017.
[3] Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. Computing graph neural networks: A survey from algorithms to accelerators. ACM Computing Surveys, 2021.
[4] Adam Auten, Matthew Tomei, and Rakesh Kumar. Hardware Acceleration of Graph Neural Networks. DAC, 2020.
[5] Prasanth Chatarasi, Hyoukjun Kwon, Natesh Raina, Saurabh Malik, Vaisakh Haridas, Anshuman Parashar, Michael Pellauer, Tushar Krishna, and Vivek Sarkar. Marvel: A data-centric compiler for dnn operators on spatial accelerators. arXiv preprint arXiv:2002.07752, 2020.
[6] Xiaoqiang Chen, Yueh Wang, Xinfeng Xie, Xing Hu, Abantí Basak, Ling Liang, Mingyu Yan, Lei Deng, Yufei Ding, Zidong Du, and Yuan Xie. Rubik: A hierarchical architecture for efficient graph neural network training. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., 2021.
[7] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In Proceedings of the 43rd International Symposium on Computer Architecture, ISCA’16, page 367–379. IEEE Press, 2016.
[8] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne 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, 9(2):292–308, 2019.
[9] Siegfried Cools and Wim Vanroose. The communication-hiding pipelined biegstmp method for the parallel solution of large symmetric linear systems. Parallel Computing, 65:1–20, 2017.
[10] William J. Dally, Yatish Turakhia, and Song Han. Domain-specific hardware accelerators. Commun. ACM, 63(7):48–57, June 2020.
[11] Jack Dongarra, Michael A Heroux, and Piotr Luszczek. Hpcg benchmark: a new metric for ranking high performance computing systems. Knoxville, Tennessee, page 42, 2015.
[12] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaoqiang Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the data. In Proceedings of the 42nd Annual International Symposium on Computer Architecture, ISCA’15, page 92–104, New York, NY, USA, 2015. Association for Computing Machinery.
[13] David K Duvenaud, Dougall Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alin Asapu-Gruzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In Advances in neural information processing systems, pages 2224–2232, 2015.
[14] Murali Emani, Venkatram Vishwanath, Corey Adams, Michael E. Papka, Rick Stevens, Laura Florescu, Sumit Jairath, William Liu, TejasNama, and Arvind Sujeth. Accelerating scientific applications with sambanova reconfigurable dataflow architecture. Computing in Science Engineering, 23(2):114–119, 2021.
[15] Wensan Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In The World Wide Web Conference, pages 417–426, 2019.
[16] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. arXiv preprint arXiv:1903.02428, 2019.
[17] James Fox, Bo Zhao, Sivasankaran Rajamaniakam, Rampi Ramprasad, and Le Song. Concentric spherical gnn for 3d representation learning. arXiv preprint arXiv:2103.10494, 2021.
[18] Tong Geng, Ang Li, Tianjiao Wang, Chunshu Wu, Yanfei Li, Runbin Shi, Antonio Tumeo, Shuai Che, Steve Reinhardt, and Martin Horberdt. AFW-CCN: a graph convolutional network accelerator with runtime workload rebalancing. MICRO, 2020.
[19] Robert Guirado, Akshay Jain, Sergi Abadal, and Eduard Alarcón. Characterizing the communication requirements of gnn models: A model-based approach. In 2021 IEEE International Symposium on Circuits and Systems (ISCAS), 2021.
[20] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Advances in neural information processing systems, pages 1024–1034, 2017.
[21] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J. Dally. Eie: Efficient inference engine on compressed deep neural network. In Proceedings of the 43rd International Symposium on Computer Architecture, ISCA’16, page 243–254. IEEE Press, 2016.
[22] Kartik Hegde, Hadi Asghari-Moghaddam, Michael Pellauer, Neal Crago, Aamer Jaleel, Edgar Solomonik, Joel Emer, and Christopher W. Fletcher. Extensor: An accelerator for sparse tensor algebra. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO ’52, page 319–333, New York, NY, USA, 2019. Association for Computing Machinery.
[23] Yuwei Hu, Zihao Ye, Minjie Wang, Jiali Yu, Da Zheng, Mu Zheng, Zhiru Zhang, and Yida Wang. Featgraph: A flexible and efficient backend for graph neural network systems. arXiv preprint arXiv:2008.11359, 2020.
[24] Zhihao Jia, Sina Lin, Rex Ying, Jiuxuan You, Jure Leskovec, and Alex Aiken. Redundancy-free computation graphs for graph neural networks. KDD, 2020.
[25] Jiuxuan You, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and William Liu. Eie: Efficient inference engine on compressed deep neural network. In Proceedings of the 43rd International Symposium on Computer Architecture, ISCA’16, page 243–254. IEEE Press, 2016.
