Crossbar based Processing in Memory Accelerator Architecture for Graph Convolutional Networks

Nagadastagiri Challapalle[1] Karthik Swaminathan[2] Nandhini Chandramoorthy[2] Vijaykrishnan Narayanan[1]
(1) The Pennsylvania State University (2) IBM T.J. Watson Research Center
{nrc53, vxn9}@psu.edu {kvswamin, Nandhini.Chandramoorthy}@us.ibm.com

2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 978-1-6654-4507-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCAD51958.2021.9643465

***Abstract*—Graph data structures are central to many applications such as social networks, citation networks, molecular interactions, and navigation systems. Graph Convolutional Networks (GCNs) are used to process and learn insights from the graph data for tasks such as link prediction, node classification, and learning node embeddings. The compute and memory access characteristics of GCNs differ, both from conventional graph analytics algorithms and from convolutional neural networks, rendering the existing accelerators for graph analytics as well as deep learning, inefficient. In this work, we propose PIM-GCN, a crossbar-based processing-in-memory (PIM) accelerator architecture for GCNs. PIM-GCN incorporates a *node-stationary* dataflow with support for both Compressed Sparse Row (CSR) and Compressed Sparse Column (CSC) graph data representations. We propose techniques for graph traversal in the compressed sparse domain, feature aggregation, and feature transformation operations in GCNs mapped to in-situ analog compute functions of crossbar memory, and present the trade-offs in performance, energy, and scalability aspects of the PIM-GCN architecture for CSR, and CSC graph data representations. PIM-GCN shows an average speedup of over 3–16× and an average energy reduction of 4–12× compared to the existing accelerator architectures.**

***Index Terms*—processing in memory, graph convolution network, aggregation, transformation, node features**

I. Introduction

In recent years, graph convolutional networks (GCNs) have been widely adopted to process, analyze, and extract insights from graph data [17]. GCNs reduce the dimensionality of the feature representations of graph data by aggregating the features of connected nodes and transforming them using shared machine learning layers. They incorporate the structure and connectivity of various nodes in the graph into the training process for performing various graph analytics tasks such as node classification [17, 16], link prediction [10, 11, 31], and clustering [29].

Graph convolutional neural network characteristics are significantly different from conventional DNNs such as CNNs and RNNs. Existing dense DNN accelerators such as [7, 24, 20] optimize the hardware architecture and memory hierarchy to support dense computations in DNNs and may incorporate support to leverage sparsity in weight matrices. Sparse DNN accelerators presented in [22, 9, 19, 1] propose software/hardware co-design approaches to leverage sparsity in both weights and activations. These accelerators are primarily equipped to leverage *structured sparsity* that are characteristic of most DNN models. On the other hand, GCNs exhibit random memory accesses and irregular/unstructured computations due to graph traversal during the aggregation phase, and sequential and structured computations during the transformation phase [32]. Consequently, we observe *hybrid* compute and memory access patterns during the inference operation in GCNs, which makes mapping of GCNs to existing DNN accelerators highly non-trivial and inefficient.

Crossbar based processing-in-memory (PIM) architectures have achieved significant performance and energy savings over conventional ASIC accelerators for DNN [23, 8, 2, 15, 14, 35] and graph analytics [25, 5, 36] applications. Crossbar memory's in-situ matrix vector multiplication (MVM) capability and density makes it favourable over in-DRAM and in-SRAM technologies which perform either bit-level operations or incorporate compute units near the arrays increasing the design complexity, given their intricate timing constraints. For DNN applications, crossbar based PIM architectures map the dense matrix vector multiplication (MVM) operations onto the in-situ parallel MAC operations in crossbar arrays. However, graph data is sparser than weight matrices of DNNs by several orders of magnitude and mapping operations in GCNs to a dense MVM compute model would result in a significant number of additional computations and memory accesses. Sparsity in DNN weight matrices ranges from 10%-95% [34, 13], whereas sparsity in widely used graph datasets for GCNs is close to 99.9%. Although crossbar-based architectures have been proposed to leverage structured sparsity in DNNs in works such as SparseReRAM [34], they are not effective for the irregular sparsity observed in GCNs. Further, the input features used in graph datasets are significantly larger in size than weights, unlike in DNNs where weights may be larger in size by 2-3 orders of magnitude. As a result, mapping GCNs onto either conventional or PIM-based accelerators with dataflow and memory access optimizations tailored for DNNs would incur significantly higher memory accesses and computations due to the hybrid execution pattern and workload characteristics of GCNs [3].

In conventional graph analytics applications, such as page rank, breadth-first search, and routing algorithms, the edge feature data (weights, degree etc.) is larger than node feature data (rank, distance etc.). Crossbar-based PIM architectures for graph analytics applications such as [5, 25] leverage the in-situ parallel multiply-and-accumulate (MAC) and parallel addition operations to perform graph computations. They map the graph

This work was supported in part by Semiconductor Research Corporation Center for Research in Intelligent Storage and Processing in Memory.

adjacency matrix onto the crossbars either in dense or sparse formats to perform computations on edge data incorporating an *edge-stationary* dataflow. However, in case of GCNs, node feature data (embedding vectors etc.) is typically much larger than the edge feature data (weights, degree etc.). This makes GCNs more suitable for a *node-stationary* dataflow, where node features remain stationary in crossbar PEs resulting in maximal re-use of node feature data, overcoming the inefficiencies of mapping them using *edge-stationary* dataflow on conventional PIM architectures.

Recently proposed GCN accelerator architectures such as GNNA [3], HyGCN [32], AWB-GCN [13], and GCNAX [18] incorporate custom hardware pipelines to support the hybrid compute and memory access patterns of GCNs and achieve significant performance and energy benefits over baseline CPU and GPU architectures. HyGCN incorporates an edge-stationary dataflow and employs dedicated engines for aggregation and transformation. AWB-GCN and GCNAX formulate the GCN inference operation as a series of sparse-dense matrix multiplications and incorporates dynamic tuning and loop tiling/ordering techniques to increase hardware utilization. However, adapting these architectures for node-stationary dataflows is challenging, even with large on-chip buffers, as the node feature data must be moved in and out of the compute units.

In this work, we introduce PIM-GCN, a crossbar-based accelerator for GCNs with support for both row and column-based compressed sparse graph data representations, namely CSR and CSC respectively, by incorporating a *node-stationary* dataflow for inference operation. The PIM-GCN accelerator maps the hybrid computations in aggregation and transformation phases of GCN inference onto crossbar-based MAC computations by incorporating efficient graph traversal mechanism and compute partitioning across graph data and node feature data. Accelerators such as GaaS-X [5] for graph applications have been shown to be highly efficient for dense and sparse graphs; however, this is limited to relatively small graphs that do not require large data transfers on and off-chip. Further, most of the applications evaluated on GaaS-X are *edge-stationary*, which is not the case for GCNs. Similarly, PIM DNN accelerators such as ISAAC [23] are restricted to dense and structured sparse models with weight-stationary dataflows. Running GCNs with hybrid execution patterns and highly varying sparsity across models on these architectures would be sub-optimal. In contrast, PIM-GCN can effectively run *node-stationary* models of different sizes, sparsities and dataflows, which makes it an ideal choice for GCNs. *To the best of our knowledge, this is the first work to propose an in-memory accelerator design for GCNs and demonstrate the mapping of GCN inference to crossbar architectures.*

The key contributions of this paper include:

- PIM-GCN, a crossbar-based accelerator architecture for GCN inference operations. PIM-GCN has flexible support for graph data represented in both compressed sparse row and column formats.
- A *node-stationary* dataflow for reducing the off-chip memory accesses and increasing the re-use of node feature data in aggregation operation. PIM-GCN leverages in-situ MAC capabilities of crossbar memories for the aggregation and feature transformation phases in GCN inference.
- An in-memory traversal mechanism for CSR graph data representation using the content addressable and parallel comparison operations in crossbar memory organizations.

Based on a comprehensive analysis of performance and energy trade-offs of the PIM-GCN architecture with respect to state-of-the-art implementations, we demonstrate an average speed up of 16×, 4×, 3× and energy savings of 12×, 8×, 4× over HyGCN [32], AWB-GCN [13], and GCNAX [18] implementations respectively.

II. Background

A. Graph Convolutional Neural Networks

Graph Convolutional Networks (GCNs) are widely used for node classification, link prediction and clustering tasks across scientific, data analytic, medical and other application domains. GCNs consist of a sequence of layers each with two phases - a) *Aggregation Phase* and b) *Transformation Phase*. In the aggregation phase, each layer propagates the input graph features along with the edges as per the underlying graph structure and aggregates the incoming features at each node. The aggregated features are projected onto a new subspace in the transformation phase, using a learned fully connected layer and a non-linear layer such as ReLu for each transformation step. The aggregation and transformation operations in a feed-forward layer can be represented using Equation 1. F is the input feature matrix, F^A is the aggregated feature matrix, and F^T is the transformed feature matrix for a given layer. A is the adjacency matrix of the graph, and W is the weight matrix of the transformation/fully-connected operation (for a given layer).

$$F^A = AF, \quad F^T = \sigma(F^A W) \tag{1}$$

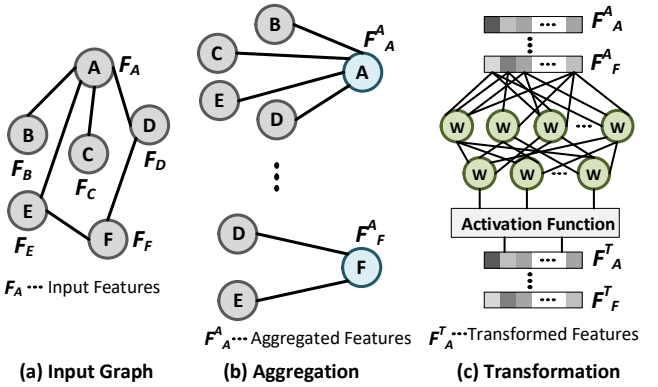

Fig. 1: (a) Sample graph along with node features (b) GCN aggregation operation and (c) GCN transformation operation

Figure 1(a) depicts a sample graph. The corresponding aggregation and transformation operations are illustrated in Figures 1(b) and (c) respectively. In the aggregation phase, node features incoming onto each node are aggregated through

the edges as shown in Figure 1(b). At each node, node features of all the incoming nodes are multiplied by their corresponding edge weights and summed together to form the aggregated node feature. Based on the underlying graph structure, the computation graph for each node varies in the aggregation phase, and also incurs random accesses to the feature data. The aggregated features are transformed onto another space using a fully connected layer followed by an optional activation function as shown in Figure 1(c). This transformation phase involves dense matrix-vector multiplication operation and memory access patterns are mostly sequential. The diverse compute and memory access patterns observed in GCNs during aggregation and transformation phases make the acceleration of GCNs challenging and opens up opportunities for novel dataflow optimizations.

B. Crossbar-based Compute Support

Resistive crossbar memory technologies use varying cell resistance to store information. A ReRAM-based memory element uses metal oxide as the storage medium and its resistance varies based on the information stored in the cell and multiple bits can be stored in the same cell using various resistance levels unlike conventional CMOS-based memory organizations [28]. ReRAM-based crossbar memory arrays are widely used for performing in-situ operations such as matrix-vector-multiplication (MVM) [23, 8, 2], scalar arithmetic [12], and pattern matching. For performing the MVM operation, the crossbar memory cells are programmed with matrix elements and bits of the input vector are fed sequentially to the wordlines through digital-to-analog converters (DACs) (Figure 2(a)). The applied input voltage is converted into the appropriate current as per the cell resistance and Kirchoff's law. The currents from the cells sharing a bitline are accumulated resulting in an in-situ sum-of-products operation. The accumulated values are stored in sample and hold (S&H) circuits and converted to digital/binary representation using analog-to-digital (ADC) converters. The partial products from each bitline are further accumulated using shift and add (S&A) units for the final result as shown in Figure 2(a).

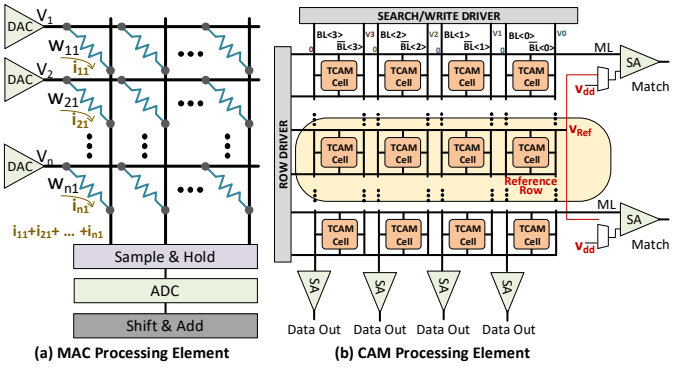

Fig. 2: ReRAM crossbar-based matrix-vector multiplication operation and search/compare operations

Crossbar memory arrays can also be used to perform the in-situ compare operations by means of content-addressable memories (CAMs). Figure 2(b) shows the array organization with the peripherals needed for the mentioned operations. The ternary CAM (TCAM) cell consists of two ReRAMs cells storing the data in complementary fashion similar to SRAM nodes storing Q and $\overline{Q}$. These complementary nodes are needed in order to perform CAM functionality in the array. In the array, the bitlines ($BL/\overline{BL}$) are connected to the nodes ($Q/\overline{Q}$) respectively. The match line (ML) is initially precharged to Vdd using the row drivers.

In case of the conventional CAM operation, the bitlines (BL <3:0> & $\overline{BL}$ <3:0>) are written with the search value using the search/write drivers. If there is a mismatch between any of the bits in the row and the search value, then the ML of that row is discharged. The sense amplifier (SA) connected to ML senses whether the row is a match or mismatch with SA reference connected to Vdd. During the compare operation, the BL <3:0> are always connected to the ground and the $\overline{BL}$ <3:0> are connected to increasing voltage from the least significant bit (LSB : $\overline{BL}$ <0>) to the most significant bit (MSB : BL <3>). In Figure 2(b), the voltages are increasing in magnitude as V0<V1<V2<V3. These voltages are calibrated according to the current characteristics of the ReRAM device, preferably the current discharge is increased in the powers of 2 from LSB to MSB. Before the compare operation is executed, the value which is to be compared is written into the reference row, and this row's ML is connected as the reference input to the SAs (through the MUX) of the other rows. Once the compare operation is initiated by connecting the bitlines appropriately, all the rows are compared against the reference row, thereby indicating the rows greater than the reference value.

III. PIM-GCN Architecture

PIM-GCN is a crossbar-based accelerator architecture for GCN inference operation that reduces memory access overheads of large feature data in GCNs. PIM-GCN introduces a novel graph traversal mechanism for CSR data representation to support gather operations at each destination node using CAM operations. PIM-GCN introduces compute partitioning across feature dimensions to support feature data reuse in GCN inference operation for graph data represented in CSC format.

A. Overview

The PIM-GCN architecture consists of the following: a) Traversal engine, b) Aggregation engine and c) Transformation engine, as illustrated in Figure 3. The traversal and aggregation engines perform operations for aggregating incoming node features at each node, as shown in Figure 1(b). The traversal engine consists of crossbar processing elements (PEs) capable of performing CAM based search (S-CAM) and compare (C-CAM) operations. The traversal engine processes the graph structure stored in compressed sparse representation for graph traversal using the CAM PEs and generates input control vectors for the aggregation engine. The aggregation engine consists of crossbar processing PEs (MAC PEs) capable of performing in-situ multiply-and-accumulate (MAC) operations for the feature aggregation and a global reduce unit to further aggregate results of various crossbar PEs. The transformation

engine implements the dense matrix-vector multiplications in the transformation or fully-collected layer operation in GCN inference (Figure 1(c)) using the crossbar MAC PEs and scalar Arithmetic and Logic Units (sALU).

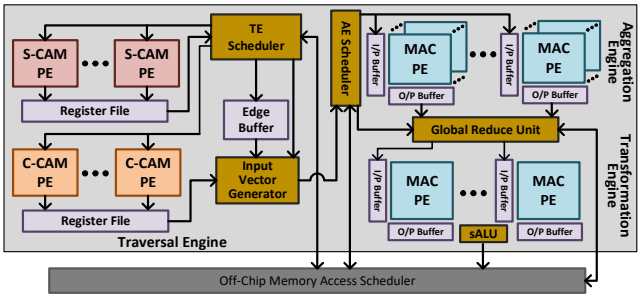

Fig. 3: PIM-GCN architecture overview

B. Traversal and Aggregation Engines

Traversal and aggregation engines implement an efficient node-stationary dataflow for GCN inference for both compressed sparse row (CSR) and compressed sparse column (CSC) graph representations. In a node-stationary dataflow, a set of node features are fetched from off-chip memory and all operations requiring these set of features are performed in a single iteration. These node features remain stationary in the crossbar PEs in the aggregation engine until a new set of node features are fetched for the next iteration, resulting in maximum reuse of feature data. The node features are stored in the crossbar arrays in the aggregation engine. The traversal engine consists of two sets of CAM crossbar arrays to perform in-situ parallel search (S-CAM) and compare (C-CAM) operations on the graph structure data for traversal. Input vector generator unit uses the result of C-CAM operation and edge data to generate input control vectors for aggregation engine which are used to activate specific rows of aggregation engine PEs, corresponding to incoming edges. The traversal engine scheduler (TE Scheduler) is responsible for the control flow of operations in the traversal engine. It handles the off-chip memory requests for reading graph data, and generates the inputs for CAM operations. The aggregation engine scheduler (AE Scheduler) is responsible for reading the node feature data, programming the crossbars, reading and broadcasting traversal engine outputs to crossbar PEs etc. The crossbar PEs in the aggregation engine perform in-situ multiply-and-accumulate operations, and the global reduce unit in the engine further processes the results from MAC PEs. *Together, the engines work to efficiently traverse graph nodes and aggregate incoming node feature data along edges to each node.*

1) CSR Dataflow: In CSR data representation, graph data is represented using three arrays $< I, E, P >$ for a given adjacency matrix, a) a Column Index Array (I) consisting of column indices of all non-zero edges scanned in a row-major order, B) an Edge Weight Array (E) that stores the corresponding edge weight values, and c) a Row Index Pointer Array (P) that stores the starting index of each node/row in I or E arrays. In each iteration, the AE scheduler loads feature data (F) of a subset of nodes onto the crossbar PEs in the aggregation engine. The TE scheduler loads the graph data in the form of column indices of edges (I) into the search crossbar PEs (S-CAM), and row index pointers (P) into compare crossbar PEs (C-CAM) in the traversal engine. It loads the edge weights (E) into the edge buffer in the traversal engine. A given destination node serves as search input to the S-CAM operation and crossbar rows that match the search input are enabled. Matching rows are reference inputs for comparison in the C-CAM operation, that identifies the source nodes with edges to the destination node by comparing the input row with P. In the aggregation engine the rows corresponding to the identified source nodes are enabled and an in-situ multiply-accumulate operation is computed. In each iteration, all the feature dimensions of a set of nodes are loaded onto the crossbar PEs in the aggregation engine, and all the required computations on that set of node features are performed, thus incorporating the node stationary dataflow to maximize the node feature reuse. *In summary, the traversal engine converts random memory accesses in CSR into efficient search and compare CAM operations and the aggregation engine aggregates the feature data of the identified source nodes for each destination node.*

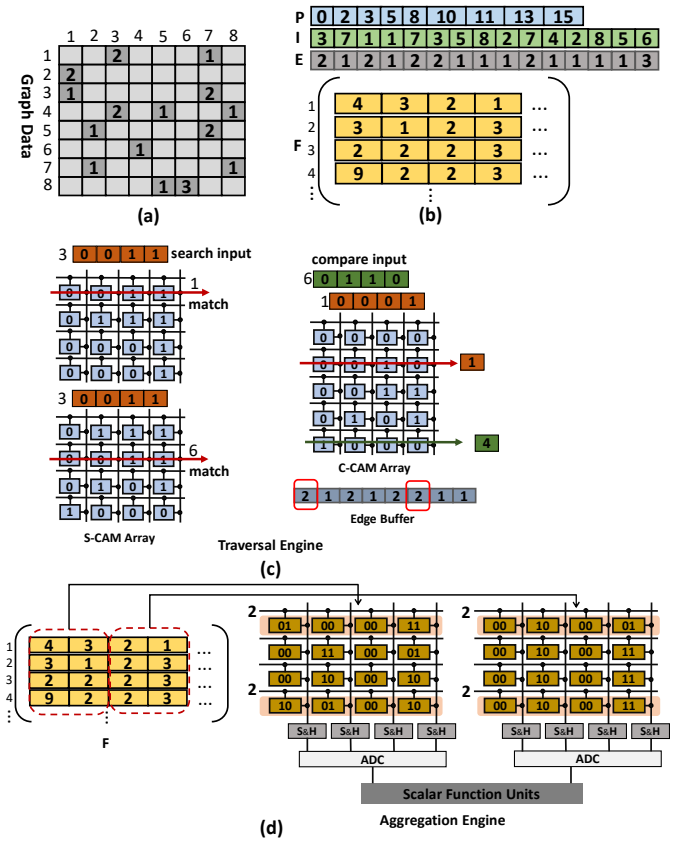

Fig. 4: (a) Sample graph data represented in adjacency matrix format (b) Graph data in CSR representation and node feature data (c) crossbar-based CAM and parallel comparison operations in traversal engine (d) crossbar-based MAC operation in aggregation engine

Consider the graph represented in adjacency matrix format in Figure 4(a) and its corresponding CSR representation $< I, E, P >$ and node features (F) depicted in Figure 4(b). In the first iteration, the node features of a subset of nodes (nodes

1 to 4 in this example) are loaded onto the crossbar arrays in aggregation engine (Figure 4(d)). For a given destination node (3 here) as shown in Figure 4(c), S-CAM searches for the destination node in I and generates row hits (1 and 6). Each matching row ID is compared with P in the C-CAM to retrieve corresponding source node ids (1 and 4) with edges to the destination node (3) using parallel compare operation. In aggregation operation for destination node 3, the rows 1 and 4 of crossbar arrays in aggregation engine are enabled and corresponding edge values (2, 2) are fed as inputs from the edge buffer as shown in Figure 4(d). The product of enabled feature data and the corresponding edge weights is then aggregated.

In the CSR dataflow, destination node features are updated partially as only a subset of node features are loaded onto the aggregation engine in each iteration. Therefore the partially aggregated results need to be stored into the off-chip memory and fetched back in the subsequent iterations when the corresponding destination node is processed again. However, as the set of destination vertices are known from I, these intermediate results are prefetched and fed to the global reduce unit for aggregating the results.

2) CSC Dataflow: In the CSC data representation, graph data is represented using three arrays $< I, E, P >$ for a given adjacency matrix - a) a Row Index Array (I) consisting of row indices of all non-zero edges scanned in a column-major order, b) an Edge Weight Array (E) that stores the corresponding edge weight values, and c) a Column Index Pointer Array (P) that stores the starting index of each node/column in I or E arrays. In CSC representation, neighboring nodes of each destination node are grouped together in I, therefore traversal operation is simple. Source nodes are retrieved by reading the row indices from I based on the offsets in P.

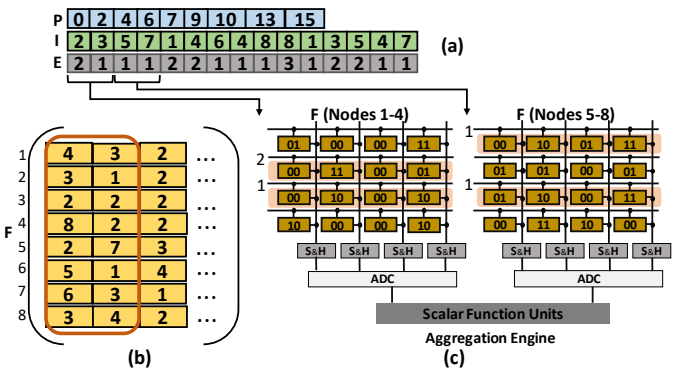

Fig. 5: (a) Graph data represented in CSC format (b) Node feature data (c) Crossbar-based MAC operation in aggregation engine

As described in sub-section III-B1, CSR dataflow partitions node feature data (F) by loading features of a subset of nodes during the aggregation operation. However, loading a subset of feature data is inefficient for CSC representation and leads to minimal reuse of the features loaded onto the crossbars. Loading all the node features may not be feasible for large graph datasets that do not fit into the crossbar memory. Instead, PIM-GCN incorporates a node-stationary dataflow for CSC representation by partitioning the aggregation operation across feature dimensions. Graph datasets in GCNs have large feature lengths or dimensions and there is no dependency across feature dimensions during aggregation. In CSC dataflow, in each iteration, a set of feature dimensions of maximum possible nodes (depending on the crossbar memory size) are loaded onto the crossbar arrays in the aggregation engine. In each iteration, all the edges in graph data are traversed and the aggregation operation is performed over a set of feature dimensions for all the nodes. Even though, graph data needs to be read/traversed several times, CSC dataflow reduces the memory accesses for intermediate feature results and result in overall reduced memory accesses, since for a majority of widely used graph datasets, the node feature data is much larger than the graph data.

Consider the graph represented in adjacency matrix format in Figure 4 (a) and its corresponding CSC representation and node features depicted in Figures 5 (a) and (b). In the first iteration, two feature dimensions of all the source nodes are loaded onto the crossbars in aggregation engine as shown in Figure 5 (c). Input vectors to the crossbar arrays in aggregation engine are generated by reading I and E array entries for each destination node for the aggregation operation as shown in Figure 5(c).

C. Transformation Engine

The transformation operation in GCNs takes node features and transforms them using a fully connected layer. In each GCN layer, the weights of the fully connected layer are shared across all the nodes of the graph. The primary kernel in transformation operation is dense matrix-vector multiplication followed by activation functions.

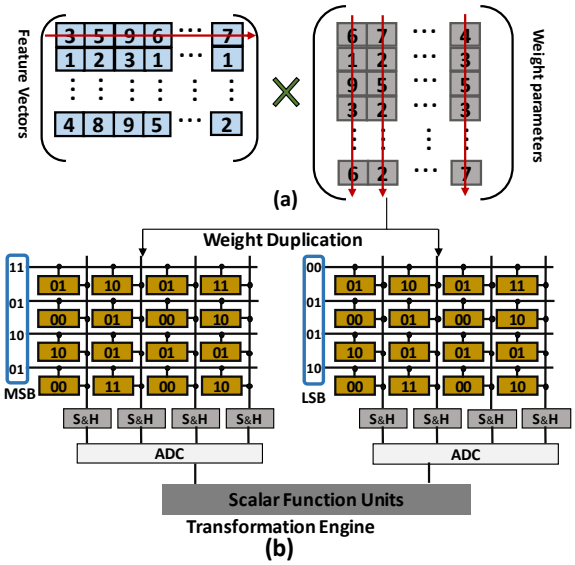

Fig. 6: (a) Dense Matrix-Vector-Multiplication (MVM) operation in transformation phase (b) Crossbar-based MAC operation with weight duplication and input splicing in transformation engine

The memory requirements of weight parameters are minimal and they can be programmed onto the crossbar arrays prior to the inference operation. The dataflow in the transformation engine is similar to existing crossbar-based accelerators for CNNs such as ISAAC [23], PRIME [8] and PipeLayer [26].

A node-stationary dataflow is implemented by storing the node weight matrix in the crossbar arrays and feeding feature vectors as inputs for performing dense matrix vector multiplication. Transformation engine also duplicates the weights across different crossbar arrays to increase the throughput of the MVM operation. Further, weight duplication decreases the latency of crossbar-based MVM operation as illustrated in Figure 6. In typical crossbar-based MVM operation, input bits are fed sequentially to the crossbar wordlines (depending on the precision of DAC unit, 2-bits in our case). PIM-GCN transformation engine feeds different bits to different crossbar arrays which are programmed with same weight parameters and accumulates the partial products using scalar functional units. As the storage requirements of weights are minimal and weight duplication helps in latency improvements and increasing throughput of the transformation engine. For the CSR-based graph data representation, transformation operation is performed in the last iteration of aggregation as the final aggregated features are calculated in the last iteration. For the CSC-based graph data representation, transformation operation is pipelined with aggregation as the final aggregated features across each dimension are available in the end of each aggregation iteration.

D. Execution Flow

The aggregation and transformation operations described above, together constitute the GCN inference operation. The metadata of the GCN inference such as the number of nodes, number of edges, feature dimensions, and dataflow are programmed onto the control registers in the TE Scheduler. PIM-GCN can be configured dynamically using the control registers in TE Scheduler to adapt either CSR or CSC dataflow for the inference based workload characteristics such as graph size, feature dimensions and sparsity. Figure 7 shows the overview of PIM-GCN execution flow. In each iteration, aggregation operation is performed on a subset of source node features or a subset of feature dimensions across the source nodes depending on the graph data representation (CSR or CSC). PIM-GCN incorporates double buffering for feature data and graph data to enable overlapping of compute, traversal, and crossbar writing/programming phases. The aggregation and transformation engines work in parallel to increase the crossbar PE utilization in aggregation engine. In the CSR-based dataflow, the traversal engine starts processing edges once the graph data is loaded onto crossbar arrays and edge buffer. For each destination node, corresponding input vectors along with destination node are sent to the input buffers of crossbar PEs in aggregation engine. The partially updated features are written to the off-chip memory in each iteration and fetched for accumulating with updated features in later iterations. The transformation operation is performed once the destination nodes features are updated (in the last iteration of aggregation) by feeding the updated features as inputs to the crossbar PEs programmed with weights in the transformation engine.

In the CSC-based dataflow, in each iteration a set of feature dimensions of the nodes are loaded/programmed onto the crossbar PEs in aggregation engine. Traversal engine reads the graph data in each iteration and the input vectors for aggregation operation of each destination node are sent to the input buffers of crossbar PEs in aggregation engine. Similar to the CSR-based dataflow, the aggregation, and programming of crossbar PEs in aggregation engine are overlapped by using double buffering. The transformation operation is pipelined with aggregation operation as in each iteration, the features of destination nodes are updated across few dimensions and they are transformed using the MVM operation with weight parameters.

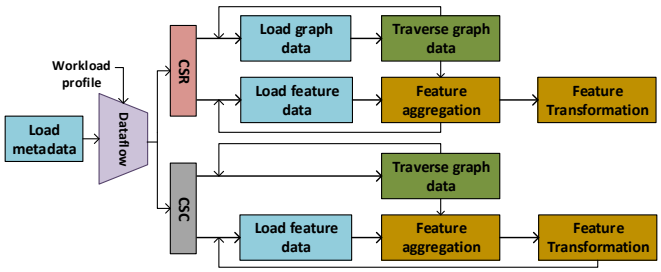

Fig. 7: PIM-GCN execution flow for CSC and CSR dataflows

IV. Evaluation

A. Evaluation Methodology

In this section, we discuss the evaluation methodology, benchmarks and comparison baseline architectures used for performance/energy analysis of PIM-GCN architecture.

System Design and Evaluation

We implemented the digital compute and control elements in PIM-GCN architecture such as controller/scheduler, scalar function units and activation function units in SystemVerilog. The power consumption and area metrics are obtained from RTL synthesis in 32nm technology [27] for operation at 1GHz. The latency and power consumption parameters of the CAM crossbar and MAC crossbar arrays are obtained by performing SPICE simulations using the 32nm resistive random access memory (ReRAM) model from [30]. We activate only up to 4 wordlines in the aggregation engine and 16 wordlines in the transformation engine for the crossbar-based MAC operation to reduce the peripheral overheads. The ADC and DAC models in our design are adapted from [6, 12]. The overall latencies of the MAC operation in transformation and aggregation engines are 15ns and 3ns respectively. Due to the latency mismatch between these operations, we also model the buffering requirements for pipelining. The latency of CAM operation and compare operation in the traversal engine are 1ns and 4ns respectively. Table I lists the various components in PIM-GCN architecture and their corresponding power and area metrics.

We designed a custom cycle-accurate simulator for performance and energy efficiency evaluation of PIM-GCN. The simulator models all micro-architectural characteristics of the PIM-GCN architecture including the on-chip storage buffers, processing-in-memory (PIM) storage and compute elements and other peripheral circuits. The latency and power consumption parameters of the on-chip storage buffers (input

TABLE I: Parameters of the PIM-GCN architecture

Component	Configuration	Area (mm^2)	Power (W)
Traversal Engine			
Crossbar PEs	512×32, 1-bit/cell number: 2K	8E-2	61E-2
Edge Buffer	512 KB	2E-1	15E-3
Register File	2048x12	66E-3	9E-2
Input Vector Generator		89E-4	35E-4
TE Scheduler		25E-2	35E-4
Aggregation Engine			
Crossbar PEs	512×512, 2-bits/cell number: 1K	13	10
Global Reduce Unit		35E-3	4E-2
Interconnect		21E-1	23E-2
Buffer	1.5 MB	12E-1	56E-2
AE Scheduler		11E-2	6E-3
Transformation Engine			
Crossbar PEs	128×128, 2-bit/cell number: 256	48E-2	2E-1
sALU	number: 64	33E-3	32E-4
Buffer	80 KB	64E-3	44E-3

buffer, output buffer, etc.) are modeled using a 32nm CACTI model [4], and HBM parameters are adapted from [33, 21]. The latency and power consumption parameters of the PIM compute elements and other digital compute elements are fed into the simulator for obtaining the overall system level execution time and energy analysis.

Comparison Baselines We compare PIM-GCN with the state-of-the-art ASIC accelerators HyGCN [32] and GCNAX [18], FPGA-based accelerator AWB-GCN [13](4096 PE configuration). We modeled the microarchitectural characteristics of HyGCN dataflow for performance comparison, off-chip memory accesses and used the power/energy metrics published in the reference manuscript scaled to 32nm. For AWGB-GCN and GCNAX, we use the absolute performance and energy metrics published in the reference implementations and modelled the dataflow for measuring off-chip memory accesses and corresponding access energy.

Graph Datasets The graph datasets used for evaluation are listed in Table II.

TABLE II: Graph datasets and characteristics

Dataset	Reddit	Pubmed	Citeseer	Cora	Collab
Vertices	232965	19717	3327	2707	372475
Edges	114615892	88651	9226	5429	24574995
Feature Length	602	500	3703	1433	496

B. PIM-GCN performance and energy evaluation

The latency and energy consumption for GCN inference on various graph datasets using PIM-GCN CSR and CSC dataflows are listed in Table III. CSR dataflow shows better performance over CSC dataflow for large graph datasets *Reddit* and *Collab*. Aggregation phase is partitioned across node feature dimensions in CSC dataflow and across source nodes in CSR dataflow based on the number of PEs in aggregation engine. For large graph datasets, CSC dataflow requires more aggregation steps due to large feature data and incurs high latency compared to CSR dataflow. CSC dataflow shows better performance than CSR dataflow for smaller graph datasets such as *Pubmed, Citeseer, Cora* as the number of aggregation steps are minimal due to the less number of nodes and overlap of transformation operation with aggregation is lesser in CSR dataflow for smaller graphs.

The energy consumption depends on the total number of crossbar operations, distribution of graph edges across nodes, off-chip data accesses for graph data, feature data, and intermediate data storage requirements. CSR dataflow reads both graph data and node feature data only once per inference, however incurs additional memory accesses for intermediate data. CSC dataflow reads the node feature data once, however reads the graph data several times. CSR dataflow also incurs more energy compared to CSC dataflow in traversal phase due to CAM operations. The number of computations in aggregation engine in each dataflow depends on the distribution of graph edges across the nodes. Overall, CSR dataflow incurs more or similar energy consumption than the CSC dataflow across all the datasets. Figure 8 shows the percentage of compute energy and memory access energy (stacked bar graph), and the graph data (graph structure data, edge attributes, and node feature data) size (line graph) . In both the dataflows for larger datasets, the memory access energy contributes significantly (95%-60%) to the overall energy consumption due to large graph data and node feature data. Relative to the graph data size, *Collab's* memory consumption energy percentage is significantly lesser than the other large datasets in CSR dataflow. We observed that most of the edges in *Collab* are concentrated around only a few nodes which leads to significantly lesser memory accesses for storing/retrieving intermediate data thereby reducing overall memory access energy consumption. For the smaller datasets, compute energy contributes significantly (85%-75%) to the overall energy consumption due to the small number of memory accesses for graph and feature data.

TABLE III: Latency and energy consumption of PIM-GCN

	CSR Dataflow		CSC Dataflow	
Dataset	**Latency (s)**	**Energy (J)**	**Latency (s)**	**Energy (J)**
Reddit	7.6E-3	1.26	1.07E-2	9.93E-1
Pubmed	3.34E-5	3.76E-3	4.93E-5	3.1E-3
Citeseer	9.54E-6	9.16E-4	9.99E-6	4.11E-4
Cora	5.56E-5	2.81E-4	9.32E-6	1.25E-4
Collab	1.08E-3	3.28E-1	3.02E-3	3.27E-1

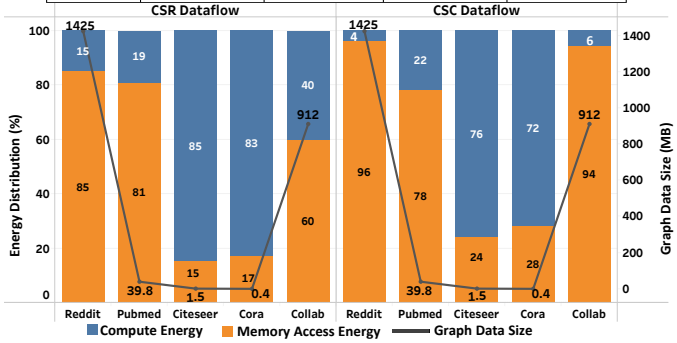

Fig. 8: Distribution of memory and compute energy for PIM-GCN CSR and CSC dataflows during inference operation, and graph data size for various graph datasets evaluated

We also observed that by increasing the number of crossbar PEs in the PIM-GCN architecture, performance and energy efficiency increases linearly for most of the datasets and saturates once the entire node feature data fits onto the crossbar PEs. However, for *Collab* dataset, performance and energy efficiency improvements were minimal in CSR dataflow due to the concentration of edges around only a few nodes.

C. Performance Comparison

Figure 9 shows the speedup of GCN inference using PIM-GCN CSR and CSC dataflows over accelerators HyGCN [32], AWB-GCN [13], and GCNAX [18]. AWB-GCN and GCNAX do not evaluate on *Collab* dataset, hence we omit them from the comparison on *Collab* dataset throughout our analysis. PIM-GCN CSR and CSC dataflows achieve geometric mean speedup of 20.3, 4.6, 3.1 and 12.7, 3.3, 2.3 over HyGCN, AWB-GCN, and GCNAX respectively. The performance improvements are due to the parallel computations in crossbar arrays for both aggregation and transformation operations and efficient traversal mechanisms. PIM-GCN's leverages the parallel compute capabilities of crossbar arrays for GCN inference by incorporating node stationary dataflow along with the custom traversal mechanisms. The performance improvements over AWB-GCN and GCNAX are lesser than that of HyGCN because they incorporate dynamic load balancing and loop tiling/ordering techniques. As PIM-GCN is a processing-in-memory technique, moving the feature data across crossbar PEs for dynamic load balancing is prohibitively expensive due to write amplification and interconnect overheads. Despite these challenges, with efficient dataflow PIM-GCN shows performance improvements and has versatile support for both CSR and CSC compressed representations whereas AWB-GCN and GCNAX support only CSC representation.

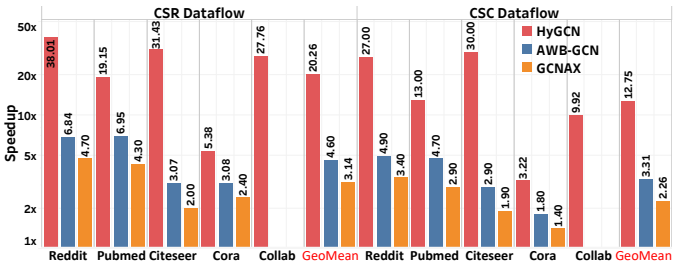

Fig. 9: Speedup of PIM-GCN CSR and CSC dataflows w.r.t HyGCN, AWB-GCN and GCNAX accelerators for GCN inference

D. Energy Comparison

Figure 10 shows the ratio of energy consumption of HyGCN, AWB-GCN, and GCNAX with PIM-GCN architecture for both the CSR and CSC dataflows. PIM-GCN CSR and CSC dataflows achieve geometric mean energy savings of 9.2, 6.2, 2.9 and 13.8, 10.4, 4.8 over HyGCN, AWB-GCN, and GCNAX respectively. Energy savings can be attributed to the low power crossbar operations and also reducing the overall memory accesses with efficient traversal mechanisms.

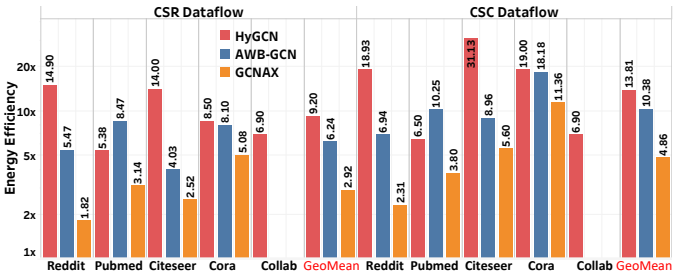

Fig. 10: Energy savings of PIM-GCN CSR and CSC dataflows w.r.t HyGCN, AWB-GCN, and GCNAX accelerators for GCN inference

Figure 11 shows the ratio of compute energy of HyGCN, AWB-GCN, and GCNAX with PIM-GCN architecture. PIM-GCN maps the computations in the aggregation and transformation operations to low power in-situ analog MAC operations by incorporating graph traversal in sparse compressed representation. PIM-GCN CSR and CSC dataflows achieve geometric mean compute energy savings of 4.2, 13.4, 5.8 and 13.1, 33.5, 14.5 over AWB-GCN and HyGCN respectively. Figure 12 shows the ratio of memory access energy of AWB-GCN and HyGCN with PIM-GCN architecture. PIM-GCN CSR and CSC dataflows achieve geometric mean memory access energy savings of 18.8, 2.7, 1.5 and 19.5, 3.2, 1.7 over HyGCN, AWB-GCN, and GCNAX. PIM-GCN architecture incurs lesser memory accesses for most of the datasets due to efficient reuse of node feature data by incorporating node stationary dataflow in the aggregation engine. PIM-GCN shows meagre improvements (lesser in case of *Pubmed*) in memory accesses over GCNAX due to the dynamic loop tiling/fusion/ordering techniques employed in GCNAX.

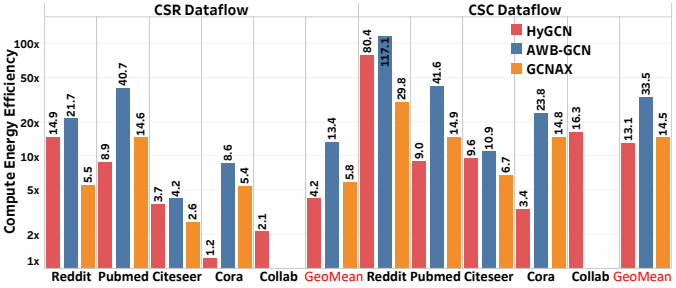

Fig. 11: Compute energy savings of PIM-GCN CSR and CSC dataflows w.r.t HyGCN, AWB-GCN, and GCNAX accelerators for GCN inference

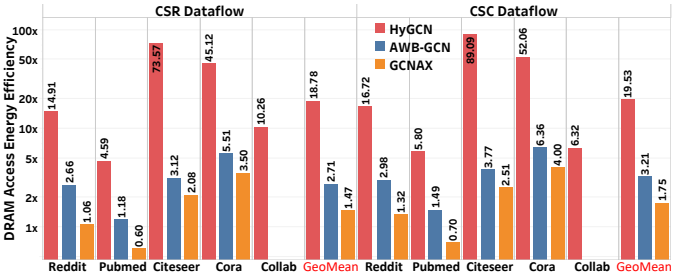

Fig. 12: DRAM access energy savings of PIM-GCN CSR and CSC dataflows w.r.t HyGCN, AWB-GCN, and GCNAX accelerators for GCN inference

V. Conclusion

In this paper, we present the PIM-GCN accelerator architecture for GCN inference operation. PIM-GCN incorporates a *node-stationary* dataflow, leveraging the in-situ MAC, CAM and parallel compare operations of crossbar arrays for graph traversal and compute operations in GCN inference. It has flexible dataflow and traversal mechanisms to support compressed sparse row and column graph data representations and compute partitioning across graph nodes and feature dimensions. In comparison to the existing accelerators, PIM-GCN reduces overall off-chip memory accesses and leverages in-situ compute capabilities of crossbar arrays to improve the performance and energy efficiency of GCN acceleration. PIM-GCN shows an average speedup of over 3–16× and an average energy reduction of 4–12× compared to the existing accelerator architectures.

REFERENCES

[1] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 1–13.

[2] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy, and D. S. Milojicic, "PUMA: A Programmable Ultra-Efficient Memristor-Based Accelerator for Machine Learning Inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, p. 715–731.

[3] A. Auten, M. Tomei, and R. Kumar, "Hardware Acceleration of Graph Neural Networks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[4] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, pp. 14:1–14:25, 2017.

[5] N. Challapalle, S. Rampalli, L. Song, N. Chandramoorthy, K. Swaminathan, J. Sampson, Y. Chen, and V. Narayanan, "GaaS-X: Graph Analytics Accelerator Supporting Sparse Data Representation using Crossbar Architectures," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 433–445.

[6] V. H. . Chen and L. Pileggi, "An 8.5mW 5GS/s 6b flash ADC with dynamic offset calibration in 32nm CMOS SOI," in *2013 Symposium on VLSI Circuits*, 2013, pp. C264–C265.

[7] 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.

[8] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 27–39.

[9] A. Delmas Lascorz, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, K. Siu, and A. Moshovos, "Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, p. 749–763.

[10] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional Networks on Graphs for Learning Molecular Fingerprints," in *Proceedings of the 28th International Conference on Neural Information Processing Systems*. MIT Press, 2015, p. 2224–2232.

[11] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein Interface Prediction Using Graph Convolutional Networks," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2017, p. 6533–6542.

[12] D. Fujiki, S. Mahlke, and R. Das, "In-Memory Data Parallel Processor," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS, 2018, p. 1–14.

[13] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt, and M. C. Herbordt, "AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 922–936.

[14] Y. He, Y. Wang, Y. Wang, H. Li, and X. Li, "An agile precision-tunable cnn accelerator based on reram," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–7.

[15] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM: In-Memory Acceleration of Deep Neural Network Training with High Precision," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 802–815.

[16] N. Jia, X. Tian, Y. Zhang, and F. Wang, "Semi-supervised node classification with discriminable squeeze excitation graph convolutional networks," *IEEE Access*, vol. 8, pp. 148 226–148 236, 2020.

[17] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[18] 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.

[19] M. Mahmoud, I. Edo, A. H. Zadeh, O. Mohamed Awad, G. Pekhimenko, J. Albericio, and A. Moshovos, "Tensordash: Exploiting sparsity to accelerate deep neural network training," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 781–795.

[20] E. Medina, "Habana Labs presentation," in *2019 IEEE Hot Chips 31 Symposium (HCS)*, 2019, pp. 1–29.

[21] M. O'Conner, "Highlights of the high-bandwidth memory (hbm) standard," in *Memory Forum Workshop*, 2014.

[22] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: an accelerator for compressed-sparse convolutional neural networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, p. 27–40.

[23] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 14–26.

[24] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, p. 14–27.

[25] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating Graph Processing Using ReRAM," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2018, pp. 531–543.

[26] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 541–552.

[27] Synopsys, [online]. Available: https://www.synopsys.com/community/university-program/teaching-resources.html, [Accessed: 24- Nov- 2020].

[28] Q. Wang, X. Wang, S. H. Lee, F.-H. Meng, and W. D. Lu, "A deep neural network accelerator based on tiled rram architecture," in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 14.4.1–14.4.4.

[29] Z. Wang, L. Zheng, and Y. Li, "Linkage based face clustering via graph convolution network," in *IEEE Conference on Computer Vision and Pattern Recognition*, 06 2019, pp. 1117–1125.

[30] Wei Zhao and Yu Cao, "New generation of predictive technology model for sub-45nm design exploration," in *7th International Symposium on Quality Electronic Design (ISQED'06)*, 2006, pp. 6 pp.–590.

[31] N. Yadati, V. Nitin, M. Nimishakavi, P. Yadav, A. Louis, and P. Talukdar, "NHP: Neural Hypergraph Link Prediction," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. Association for Computing Machinery, 2020, p. 1705–1714.

[32] 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.

[33] M. Yan, X. Hu, S. Li, A. Basak, H. Li, X. Ma, I. Akgun, Y. Feng, P. Gu, L. Deng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Alleviating irregularity in graph analytics acceleration: A hardware/software co-design approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, p. 615–628.

[34] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, "Sparse ReRAM Engine: Joint Exploration of Activation and Weight Sparsity in Compressed Neural Networks," in *ISCA*, 2019, pp. 236–249.

[35] X. Yang, B. Yan, H. Li, and Y. Chen, "Retransformer: Reram-based processing-in-memory architecture for transformer acceleration," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20, 2020.

[36] M. Zhou, M. Imani, S. Gupta, Y. Kim, and T. Rosing, "GRAM: Graph Processing in a ReRAM-based Computational Memory," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 591–596.