2023 IEEE 30th International Conference on High Performance Computing, Data, and Analytics (HiPC) | 979-8-3503-8322-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/HIPC58850.2023.00016

PARAG: PIM Architecture for Real-time Acceleration of GCNs

Gian Singh
Arizona State University
gsingh58@asu.edu

Sanmukh R. Kuppannagari
Case Western Reserve University
sanmukh.kuppannagari@case.edu

Sarma Vrudhula
Arizona State University
svrudhul@asu.edu

***Abstract*—Graph Convolutional Networks (GCNs) have successfully incorporated deep learning to graph structures for social network analysis, bio-informatics, etc. The execution pattern of GCNs is a hybrid of graph processing and neural networks which poses unique and significant challenges for hardware implementation. Graph processing involves a large amount of irregular memory access with little computation whereas processing of neural networks involves a large number of operations with regular memory access. Existing graph processing and neural network accelerators are therefore inefficient for computing GCNs. This paper presents PARAG, processing in memory (PIM) architecture for GCN computation. It consists of customized logic with minuscule computing units called *Neural Processing Elements (NPEs)* interfaced to each bank of the DRAM to support parallel graph processing and neural network computation. It utilizes the massive internal parallelism of DRAM to accelerate the GCN execution with high energy efficiency. Simulation results for inference of GCN over standard datasets show a latency and energy reduction by *three orders* of magnitude over a CPU implementation. When compared to a state-of-the-art PIM architecture, PARAG achieves on an average 4× reduction in latency and 4.23× reduction in the energy-delay-product (EDP).**

***Index Terms*—Graph Convolutional Networks, Memory Bottleneck, Processing In-Memory, DRAM**

I. Introduction

Deep learning on structured data, such as images and text, has found widespread success for various tasks in computer vision [1], [2], and natural language processing (NLP) [3]. However, these models are unsuitable for unstructured, relational data represented as graphs. Graph Convolutional Networks (GCNs) [4], which combine deep learning with graph data, are effective for node classification, and other such graph-based tasks. Among the various graph representation learning models [5], GCNs have been particularly successful when applied to domains such as bio-informatics [6], social network analysis [7], e-commerce [8], recommendation systems [9] etc.

GCN computation has two phases: *aggregation* and *combination*. The aggregation phase is primarily graph processing in which while visiting every node v, the feature vectors of its neighbors $N(v)$ are processed to generate a new aggregated feature vector of v. The combination phase takes the aggregated vector and transforms it into a lower-dimensional vector using a convolutional neural network (CNN). Inference of a GCN [4] model requires computing the aggregation and combination phases sequentially over a few hundred of thousand nodes while processing millions of edges [10].

The *aggregation* and *combination* present unique challenges for hardware due to their contrasting computational characteristics. The aggregation phase, with irregular memory accesses due to graph processing, is memory-bound while the combination phase, with CNN computation and regular memory accesses, is compute-bound [11]. While several CPU+accelerator (GPU, FPGA) architectures have been developed for GCNs, they have been able to only address the compute-bound combination phase and have had limited success in reducing the memory bottleneck of the aggregation phase [12].

PIM architectures have demonstrated significant success in improving the performance and energy efficiency of memory-bound applications by orders of magnitude compared to CPUs, and GPUs [13]. However, existing PIM architectures are optimized for either graph processing [14] or CNNs [15], but not GCNs which includes both. Graph processing PIM architectures are based on near data/memory processing (NDP) architectures in 3D memories such as hybrid memory cube (HMC) [16] or high bandwidth memory (HBM) [17] (see Section II-D1). Such NDP architectures do not have sufficient parallelism for computing CNNs. On the other hand, CNN-specific PIMs are based on near bank processing (NBP) or near array processing (NAP) (see Section II-D2 and II-D3). These architectures include a high degree of parallelism for regular data accesses but have low performance for graph processing. Therefore, existing PIM architectures do not achieve high performance and high energy efficiency for GCNs. Additionally, they also have high area and power overhead that can lead to reduced DRAM capacity and thermal issues [15].

To address these issues, this paper introduces a new PIM architecture, named PARAG, that simultaneously improves the GCN's throughput and energy efficiency. PARAG features an array of Neural Processing Elements (NPEs) [18] that interface with the DRAM bank's I/O block. The NPE-Array achieves a high degree of parallelism through SIMD computation and acts as a near-bank processing (NBP) architecture to meet parallel computing requirements of the compute-intensive combination phase. The NPEs are composed of digital artificial neurons (ANs) (a.k.a. threshold logic gates [19]), and local registers, whose area and power are substantially lower than a functionally equivalent CMOS implementation [20]. Additionally, to

The research was supported in part by NSF grant #2008244. The authors also gratefully acknowledge Qualcomm Technologies Inc. for the support.

2640-0316/23/$31.00 ©2023 IEEE
DOI 10.1109/HiPC58850.2023.00016

TABLE I: Notations used in GCN.

Term	Meaning
$\mathcal{G}$	graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
$\mathcal{V}$	vertices of $\mathcal{G}$
$\mathcal{E}$	edges of $\mathcal{G}$
D_v	degree of vertex v
$\phi(.)$	activation function
$e_{(i,j)}$	edge between vertex i and j
$N(v)$	neighbor set of v
$A(A_{ij})$	(element of) adjacency matrix
a_v	aggregation feature vector of v
h_v	feature vector of v
$\boldsymbol{X}$	feature matrix composed by feature vectors

execute the aggregation phase, PARAG includes a customized *aggregation engine* using a small controller and an array of NPEs, interfaced with each DRAM bank. In this way, PARAG also mimics a near data/memory architecture (NDP) and computes the aggregation phase with high throughput and energy efficiency. With these two innovations, PARAG executes the entire GCN in the DRAM and achieves high throughput and high energy efficiency within the **tight** area and power constraints of the DRAM.

The main contributions of this paper are summarized below:

- PARAG is a new PIM architecture for GCN inference, which integrates NPE-Array and a custom aggregation engine with DRAM within the stringent area, power, and timing constraints.
- PARAG can compute operands with bit-widths ranging from 1 to 8 with high throughput and energy efficiency.
- PARAG maximally utilizes the large internal parallelism in a DRAM to perform computation on the NPEs and also handle the irregular memory accesses to execute the entire GCN within the DRAM.
- This paper explores and evaluates two different configurations of PARAG with varying compute parallelism, on the 2D DRAM and 3D DRAM organizations.

II. BACKGROUND

A. Graph Convolution Network (GCN) models

The notation used to describe GCN computations is shown in Table I. Graph Convolutional Networks (GCNs) are based on convolutional neural networks (CNNs) but are adapted to work with graph-structured data. The input is a node-attributed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ denote the set of nodes (vertices) and edges, respectively. Each node $v \in \mathcal{V}$ has an associated feature vector (h_v) of length f and $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times f}$ represents the feature matrix where each row contains the feature vector h_v of a node.

GCN inference for a single layer is illustrated in Fig. 1. For a node v in a graph, the aggregation phase transforms the feature vectors (of size f) of the neighbors $N(v)$ to generate an aggregated feature vector a_v (of size f). In the combination phase, the aggregated feature vector of every node is transformed into a feature vector of a different dimension

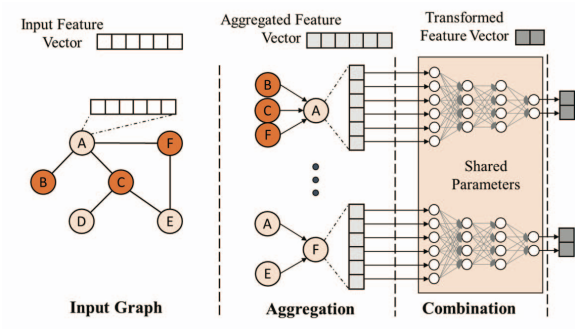

Fig. 1: GCN inference model.

using a multi-layer perceptron (MLP) network. A layer l in GCN inference can be written as:

$$\begin{aligned} a_v^l &= \textbf{Aggregate}(h_u^{l-1} : u \in \{N(v)\} \cup \{v\}), \\ h_v^l &= \textbf{Combine}(a_v^l) \end{aligned} \tag{1}$$

There are several types of GCN models such as Vanilla-GCN [4], GraphSage [21], GINConv (GIN) [22], etc., that use different aggregation and combination functions. Aggregation functions include element-wise addition, mean, maximum, etc., and the combination phase commonly uses an MLP of varying number of hidden layers of different sizes.

B. Quantized GCN

GCNs have demonstrated excellent performance in graph-based tasks in commercial applications, but large graph sizes have huge memory requirements. To this end, the low-precision computation on quantized (8-bit, 4-bit, and 1-bit) data has been explored for GCN inference with a small reduction in the inference accuracy [23], [24]. Binarized GCN (Bi-GCN) [24] is a special case of quantization the FP-multiplications are replaced by bitwise XNOR, and the accumulation is replaced by the bitcount operation with an overall reduction of the memory requirements by about 30×. PARAG supports the quantization GCN model of up to 8 bits though in this work we focus on 1-bit, 4-bit, and 8-bit quantization for thorough evaluations.

C. DRAM: 2D DIMM and 3D organization

The organization of a DRAM consists of several levels of hierarchy. Logically, the lowest level building block is called a *bank*. A bank contains multiple 2D arrays of memory cells, a row of sense amplifiers, a row, and a column decoder, collectively referred to as a *subarray*. In a conventional 2D DRAM organization called *Dual-in-line memory module (DIMM)*, the DRAM chips are packaged separately and placed on a printed circuit board (PCB) as shown in Fig. 2. A DIMM has two sides and the set of chips connected to the same chip-select signal on one side of a DIMM is called a *rank*. The chips on one rank are simultaneously accessed to supply data to the *memory channel* connecting the DRAM and CPU.

Lately, DRAM chips have been integrated in 3D using through silicon vias (TSVs) that serve as mechanical support, as well as the communication and data links between the

DRAM chips as shown in Fig. 2. All the DRAM chips are connected to a *logic layer* at the base which houses the memory control logic. The DRAM chips and the logic layer are packaged together and are commercially available as high bandwidth memory (HBM) [25] and hybrid memory cubes (HMC) [26]. Compared to the 2D DDR-DIMM, the 3D HMC and HBM have substantially higher bandwidth (4× to 5×) at much lower energy consumption (3× to 4×) [27].

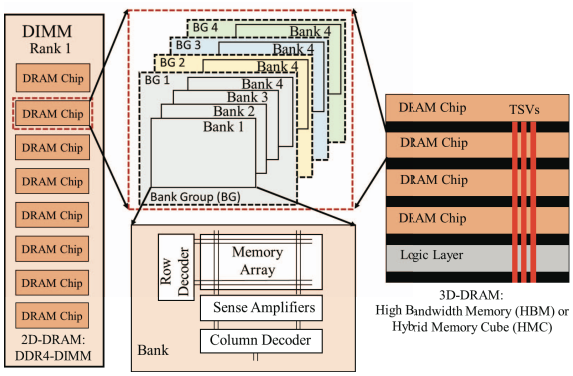

Fig. 2: 2D and 3D DRAM organization.

D. Existing PIM architectures

Placing computation near or within a conventional digital system's memory is a popular approach to improve the performance and energy efficiency of memory-bound applications. Computation can be added at different levels of the memory hierarchy (rank, chip, banks, arrays) with varying complexity catering to the specific type of memory-bound applications. The classification of PIM architectures is shown in Table II.

TABLE II: Classification of PIM architectures.

PIM Architecture	DRAM Hierarchy level	Operations Enabled	Applications
Near Data Processing (NDP)	Rank/Chip (DDR) or Logic Layer (HMC/HBM)	32-bit INT/FP ops, Data movement	General Purpose, Graph Processing [16], [28]
Near Bank Processing (NBP)	Memory Banks	Quantized bits (≤16) INT/FP specific ops	Neural Networks (NN), Machine Learning [15], [29]
Near/In Array Processing (NAP/IAP)	Memory Arrays	Bitwise Boolean ops	Quantized NN (QNN), Database, Encryption [13], [30]

1) Near Data Processing (NDP)

The NDP architectures include general-purpose compute elements on the same DRAM chip in the DIMM organization [28] or on the logic layer of the 3D-DRAM memory (HBM/HMC) [16], [25], and thus avoid inefficient data transfers on the memory channel. Other system-level innovations in NDP architectures include the design of instruction-level general-purpose offloading scheme [31] and distributed GNN processing framework with optimized graph partitioning and scheduling [32]. In General, the NDP architectures have low compute parallelism as the compute elements are interfaced with the global I/O block and are therefore, inefficient when executing the compute-bound combination phase of GCNs.

2) Near Bank Processing (NBP)

The NBP architectures host the compute logic between the local I/O and global I/O of the memory. The compute logic is interfaced with the data at the output of each memory bank. All the banks are operated simultaneously to maximize bank-level parallelism. The examples of near-bank architectures are Samsung Aquabolt-XL [15], SK-Hynix GDDR6-based accelerator in memory (AIM) [29] etc. These architectures utilize higher parallelism than the NDP architectures but have compute elements with limited functionality and lack the capabilities to handle the large number of data movements required in GCNs within the memory. These architectures have reported a reduction in memory capacity and thermal problems despite a primitive design of compute elements [15].

3) Near/In Array Processing (NAP/IAP)

These architectures exploit the bit-level parallelism in the memory by operating on the row-wide operands in parallel using a single command. They perform bit-wise Boolean operations on multiple rows of the memory array by issuing only a few DRAM commands. They rely on analog charge-sharing operation [13] which is unreliable and violates the memory access protocol and timing constraints. Near-array architecture such as [30] adds logic gates interfaced directly with sense amplifiers of the memory array. They exploit the maximum available bandwidth in memory but are limited to operations on bit-vectors and incur a large performance penalty for the complex operations required in GCNs.

4) Bandwidth and Parallelism in PIM Architectures

The available *Bandwidth* (BW) decreases as the data is transferred through the DRAM hierarchy. For example in a DDR4-DIMM with 8Kb of row buffer operating at 1 GHz and $t_{rcd} = 16$, 64 GB/s of BW is available per bank. Though from the row buffer, only 8 bits are accessed in time $t_{ccd} = 4$ cycles which results in the BW of 256 MB/s available external to the bank. Therefore, the NAP/IAP architectures utilize much higher bandwidth than the NBP and NDP architectures.

Summary: The NDP architectures are suitable for graph processing as they reduce the data transactions on the external memory bus. On the other hand, the NBP, NAP, or IAP architectures can utilize internal memory parallelism and add adequate computation units inside the DRAM to meet the computation demands of either full precision or quantized neural networks. Therefore, a combination of NDP with either NBP or NAP/IAP architecture is ideal for computing GCNs.

GCIM [33] is the state-of-the-art custom GCN PIM architecture using HMC. GCIM uses MAC PEs with a controller and buffers in the DRAM layers of HMC for the aggregation and a larger systolic array of MAC-PEs in the logic layer of the HMC for combination. GCIM does not have support for functions such as mean, maximum, and activation, which are common in the aggregation phase for different GCN models.

PARAG eliminates the memory bottleneck in both phases of the GCN workload by computing them within the DRAM and adhering to the area, power, and timing constraints without modifying the memory arrays. In addition, satisfying

the DRAM constraints is absolutely critical for adoption in commercial and practical systems.

The key advantages of PARAG are summarized below.

- NDP and NAP processing within the DRAM is enabled without violating timing or modifying the DRAM array.
- Utilize maximum available bandwidth inside the DRAM.
- Unlike existing NAP architectures, a complete digital processing of the workload is performed, which increases the reliability of operation and does not lead to a drop in the software accuracy for GCN.
- Unlike existing NBP architecture, there is no reduction in DRAM capacity in PARAG.
- The NPEs in PARAG are not limited to the MAC operation and can be configured to different operations including the activation function. This enables support for different GCN models without using any other hardware for computation, unlike existing PIM architectures.

III. PARAG ARCHITECTURE

Fig. 3 shows the major hardware components of PARAG. At the lowest level (Fig. 3c), PARAG is composed of *artificial neurons* (AN, Section III-A). A network of ANs (Fig. 3b) serves as a *neuron processing element* (NPE) that can be instantly configured to perform different arithmetic, logic, and other operations common to DNNs. It is operated to ensure that overprovisioning of hardware the given operand size. This reduces both power and area – the latter because the same basic cell is repeatedly used, thus avoiding the need for function-specific hardware units. A collection of independent NPEs, referred to as an NPE-Array in Fig. 3a are integrated with memory arrays without interfering with the timing constraints or access protocols of the memory. In addition, there is an *aggregation engine* (AE) (Section III-D) that interfaces with the output of the column decoder of each bank. The AE also uses a small array of NPEs for computation. A memory bank with the NPE-Array and the aggregation engine form a single processing in memory unit, referred to as a *PIM-unit*. The PIM-unit is unrelated to the DRAM organization which allows PARAG to be adaptable to any variants of a DRAM such as 2D-DIMM (DDR, GDDR, LPDDR, etc.) and 3D DRAM (HMC and HBM). PARAG is fully scalable with DRAM capacity, organization, and the DRAM's interface with the host CPU. A GCN application can be mapped to different levels of the DRAM hierarchy with the input graph being partitioned to the number of banks (PIM-units) in the DRAM operating in parallel. Thus, PARAG can be operated as a low-power edge device as well as a high-performance data center accelerator.

A. Threshold logic functions and Artificial Neurons

A Boolean function $f(x_1, x_2, \cdots, x_n)$ is called a threshold function if there exists a set of weights $W = (w_1, w_2, \cdots, w_n)$, and a threshold T such that[1]

$$f(x_1, x_2, \cdots x_n) = 1 \;\Leftrightarrow\; \sum_{i=1}^{n} w_i x_i \geq T, \tag{2}$$

[1]W.L.O.G. the weights w_i and threshold T can be integers [19].

where $\sum$ denotes the arithmetic sum. Many analog and digital implementations of threshold functions exist in literature. These implementations can be called *artificial neuron* (AN). Fig. 3c shows the design of the AN that is used in PARAG.

The advantages of a single AN over the CMOS equivalent are demonstrated in [20]. For instance, *a 5-input AN in 40nm, which is about the size of a high drive strength D-FF* can replace a complex function such as a 3-out-of-5 majority function $f(x_1, x_2, x_3, x_4, x_5) = x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4 + x_1x_2x_5 + x_1x_3x_5 + x_2x_3x_5 + x_1x_4x_5 + x_2x_4x_5 + x_3x_4x_5$ *and* the D-FF that f drives. Many other functions that would normally require several levels of logic can be replaced by a single AN.[2] Overall, at the individual cell level, [20] shows that a 5-input AN results in improvements in area, power, and delay of $[80\%, 60\%, 40\%]$ respectively, over the performance optimized, functionally equivalent CMOS circuit.

The AN shown in Fig. 3c has four main components:[3] the left input network (LIN), the right input network (RIN), a sense amplifier (SA), and an output latch (LA). The sense amplifier outputs are differential digital signals $(N1, N2)$, with $(1, 0)$ and $(0, 1)$ setting and resetting the latch. The LIN and RIN consist of a set of branches with two devices in series in each branch. One provides a *programmable* conductance between its two terminals, and the other device, a MOSFET is driven by an input signal x_i. The conductance of a branch controlled by x_i serves as a proxy of the weight w_i in Equation 2. When the clock is enabled, LIN and RIN compute the weighted sum of the inputs in the form of a cumulative current and connect to the sense amplifier as two differential signals, as shown in Fig. 3c. The sense amplifier evaluates to 1 (0) if the LIN current is greater (lesser) than the RIN and stores the result in the SR latch.

B. Neuron Processing Element (NPE)

NPE is the basic compute element used for both the aggregation and the combination phase. The architecture of the NPE is shown in Fig. 3b as presented in [18]. It consists of k (=4) fully connected ANs, denoted by N_k, where k is the index of the neuron. Each AN is connected to a 16-bit local register, and ANs communicate with each other using multiplexers. Each AN implements the threshold function $[2, 1, 1, 1; T]$ where T can take values $[1, 2, 3]$. This paper uses the physical implementation of an AN described in [20] and shown in Fig. 3c. A threshold function can be represented graphically as shown in Fig. 4a. Other functional equivalent implementations of the ANs are also possible [34], [35]. The implementation in [20] was chosen because the area of each AN is extremely small (about the size of DFF). An NPE performs a bit-serial operation on the n-bit operands by processing one bit in one cycle using the same AN regardless of the bit-width of the operands. A four-input AN $[2, 1, 1, 1; T]$ is used to implement

[2]Includes all 117 positive threshold functions of ≤ 5 variables and their NPN equivalents, which number in the thousands.

[3]This is a simplified version of the design shown in [20], with *programming* circuitry not shown.

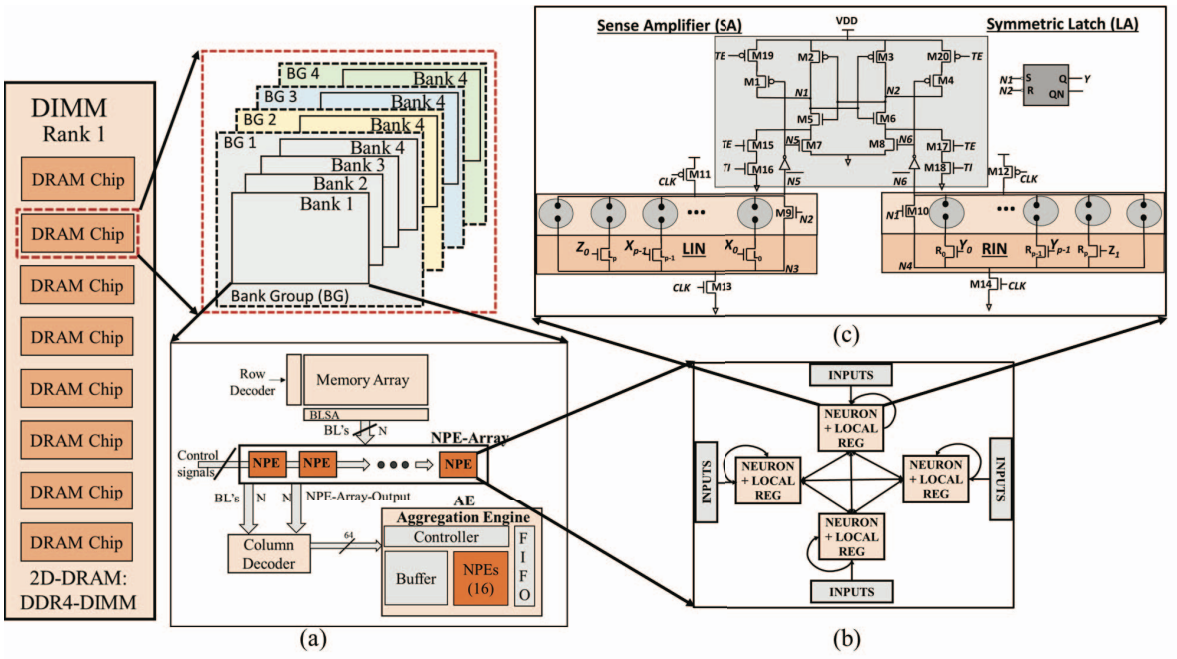

Fig. 3: (a) Top-level architecture and integration of NPEs and aggregation engine (AE) with DDR-DIMM DRAM (b) Architecture of neuron processing element (NPE) [18] (c) Structure and of the artificial neuron (AN) [20].

different functions (logic, addition and comparison) using the same physical implementation.

The value of T and the inputs are selected during run-time using control signals. The bit operands (x, y, z) to the NPE can be tied to the binary inputs a, b, c, and d of a N_k AN to perform **logic operations** such as (N)AND, (N)OR, NOT, and 3-input majority in a single clock cycle as shown in Table III. The NPE can also perform single-bit addition using threshold function evaluation on two ANs in consecutive clock cycles. Fig. 4b shows 1-bit addition using XOR and majority.

TABLE III: Inputs and threshold (T) selection for various bitwise operations on a single AN executed in a single cycle. n-bit operations execute in n+1 cycles. x, y and z are input operand bits [20].

Operation	a	b	c	d	T
NOT (a)	x	0	0	0	1
AND (a,b)	x	y	0	0	2
OR (a,b)	x	y	0	0	1
MAJ (a,b,c)	x	y	z	0	2

An NPE computes a **multi-bit addition** operation by decomposing it to a ripple carry adder, where each full-adder computes the i^{th} sum (S_i) and carry (C_i) bit using AN as shown in Fig. 4a. An NPE either receives the operand bits from external inputs or fetches from the local register of the neurons in each clock cycle. The output of the addition is stored in the local register of the neurons. An NPE can individually address each bit in the local register, which allows it to choose the specific bits from the local register.

Similar to addition, an NPE computes a **multi-bit comparison** operation sequentially using a single neuron, as shown in Fig. 4b. The n-bit operands are evaluated on the neuron from LSB to MSB. The neuron evaluates the result of $x_i > y_i$

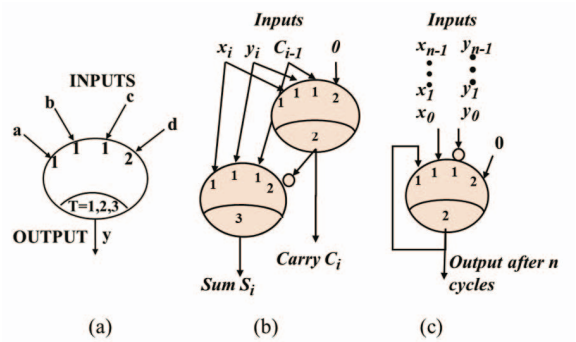

Fig. 4: (a) Artificial Neuron (AN) used in the NPE, (b) 1-bit Addition operation using sequence two threshold functions. n-bit addition takes n+1 cycles on NPE. (c) n-bit comparison using sequential execution on AN [18] takes n+1 cycles.

in each cycle and overrides the result of the evaluation of all previous cycle operations by comparing all lower significant bits. Consequently, at the end of n cycles, the output of the comparison is 1 if $X > Y$, else it is 0.

The **multiplication operation** on an NPE is divided into a series of bitwise AND and addition operations. Consider the example of multiplying two 4-bit operands, X = *x3,x2,x1,x0* and Y = *y3,y2,y1,y0*. Using the shift and add multiplication algorithm, the NPE generates partial products *P0 = X.y0, P1 = X.y1, P2 = X.y2, P3 = X.y3* over four steps and multiple clock cycles. The NPE then performs a series of multi-bit additions on the partial products to obtain the final result, which is stored in the local register of one of the neurons. Overall, 4-bit multiplication takes 21 cycles on NPE. For operands of larger bit-widths, the operands are divided into 4-bit segments,

and the same process is repeated.

An NPE consumes, on average, substantially less (59×) power and has about 23× lower area as compared to a functionally equivalent CMOS implementation. The readers can refer to [18] for a detailed explanation of the design of NPE. The logical operations, addition (XOR/XNOR), comparison, and multiplication are all used to compute the operations of the aggregation and combination phase of the GCN.

C. NPE-Array

The NPE-Array is placed between the bit-line sense amplifier (BLSA) output and the column decoder of a bank as shown in Fig. 3a. The BLSA latches the entire row data (N-bits) of the memory array and delivers the maximum amount of data in parallel inside DDR-DRAM to the compute elements. Each NPE is connected to four BLSA outputs, and therefore, there are $N/4$ NPEs in each bank. The NPEs operate in a SIMD fashion by sharing the control signals from an external controller. Each NPE performs operations on the operands local to the bank to which they are connected. Multiple operands are placed in different rows and share the same columns as shown in Fig. 5b. Therefore, a sequence of row activation (ACT) and precharge (PRE) commands can be used to supply operands to all the NPEs for parallel operations. NPEs being connected to four BLSA outputs can receive up to four bits of an operand with a single ACT command. Therefore, larger bit-width operands must be split into multiple rows sharing the same columns. After receiving the operands, the NPEs can perform logical, arithmetic, relational, and predicate operations in single or multiple cycles depending upon the operation. The NPEs write back the results (N-bits) to the rows reserved in the memory array for the outputs by driving the BLSA outputs.

D. Aggregation Engine (AE)

The main components of the aggregation engine of PARAG are shown in Fig. 3a. The AE is interfaced with the output of the column decoder (64 bits in DDR4) in each bank of the DRAM. The AE consists of a *controller* which reads the edge list $\mathcal{E}$ from the *FIFO* and generates the address for the feature vectors of the source and destination vertices. The NPEs perform the aggregation operation on the feature vector and store the partially updated feature vector for a node in the *Buffer*. There are 16 NPEs in the AE, where each NPE operates on 4 bits in parallel. A completed aggregated feature of a node is then written back to the memory bank.

E. Data layout for Aggregation and Combination Phase

To maximize performance by enabling all PIM units to operate in parallel, the operands must be local to the memory bank of a PIM-unit. Any inter-bank communication through shared buffers must be initiated by an external CPU which may require multiple data transactions on the memory bus leading to a drop in performance and energy efficiency.

To extract the maximum compute parallelism, PARAG uses a horizontal data layout for the aggregation and a vertical data layout for the combination phase. This is shown in Fig. 5. The

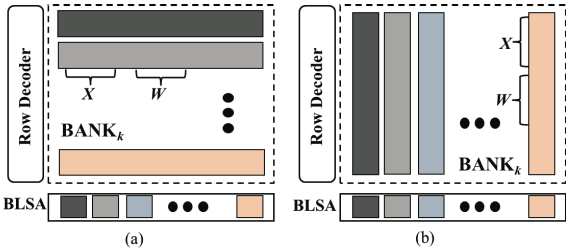

Fig. 5: (a) Data layout for computation using aggregation engine (AE), (b) Data layout for computation using NPE-Array.

main operation in the aggregation phase involves generating and accessing the random addresses to read the feature vectors of the neighborhood of a node. Therefore for the aggregation phase, the data is stored in a horizontal layout and is byte addressable as in conventional CPU memory access. Fig. 5a shows two vectors **X** and **W** mapped to consecutive addresses in a row. Vector operands with multi-bit (8-bit, 4-bit) elements can flow into multiple rows and still be computed by the aggregation engine.

In the combination phase, the multi-layer perceptron computation involves regular memory accesses for MLP of a fixed size determined by the size of the feature vector, the size of the hidden layers, and the number of classes in the dataset. MLP computation in GCN involves high compute parallelism as the same MLP is computed for all the vertices of the graph dataset and further each node of an MLP layer involves an equal number of operations with independent weights and shared input activations. Hence, the MLP can be extensively accelerated by computing each node of MLP in parallel on processing units operating in a SIMD fashion. Therefore, it is ideal to use the NPE-Array for the combination phase. The NPE-Array requires that the data associated with a single operation must be mapped to the same columns and follow a vertical layout as shown in Fig. 5b.

F. Dataflow for GCN processing

The pseudo-code labeled as Algorithm 1 shows the overall dataflow for computing a GCN on the PARAG platform. This paper focuses on the inference of the GCN, which involves transforming the features of all the nodes of the dataset through an aggregation and combination phase. The input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is subjected to one-time pre-processing steps of partitioning and storage with the appropriate data layout into the memory banks to enable maximum parallelism. The set of vertices $\mathcal{V}$ of the graph are partitioned to K subsets, where K is equal to the number of banks in the DRAM. The $PartitionAlgorithm$ stated in line 2 of Algorithm 1 is based on METIS [36] and is common in several GNN implementations [37]. METIS uses multiple phases and multi-level algorithms to perform the partitioning. The graph pre-processing steps are written on lines 1 and 2 in Algorithm 1. Lines 3-8 in Algorithm 1 shows the execution of the aggregation and combination phases. The order of execution depends upon the GCN model. For GCN model [4], the combination

phase is executed before the aggregation phase, whereas it is reversed for GIN [22]. The second loop in the Algorithm 1 (line 4) is executed in parallel across all banks.

Algorithm 1 GCN Processing dataflow on PARAG

Input: $\mathcal{G}(\mathcal{V}, \mathcal{E})$, $\boldsymbol{X}$, $\boldsymbol{W}$ and K
{$\mathcal{G}(\mathcal{V}, \mathcal{E})$ is input graph with a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$, $\boldsymbol{X}$ and $\boldsymbol{W}$ are feature and weight matrix, and K is the number of bank in the DRAM}
Output: Feature vector h_v, $\forall v \in \mathcal{V}$ **for layer** l
1: Create vertices $\mathcal{V}_k$, feature vector matrix $\boldsymbol{X}_k$, and edge list partitions $\mathcal{E}_k$ = PartitionAlgorithm($\mathcal{G}$, $\boldsymbol{X}$, K); $\forall k \in K$ and store partitions $\mathcal{V}_k$, $\boldsymbol{X}_k$ and $\mathcal{E}_k$ in DRAM bank $k, \forall k \in K$
2: Map $\boldsymbol{X}_k$ and $\boldsymbol{W}$ to memory banks as described in Section III-E
3: **for** GCN layer $l \in L$, **do**
4: **for** $k = 0$ to $k < K$ **do**
5: h_v^l = **Combine**(a_v^l, $\boldsymbol{W}$), $\forall v \in \mathcal{V}_k$, {using NPE-Array}
6: a_v^l = **Aggregate**(h_u^{l-1}: $u \in \{N(v)\} \cup \{v\}$), $\forall v \in \mathcal{V}_k$, {using Aggregation Engine}
7: **end for**
8: **end for**

IV. EXPERIMENTS AND RESULTS

Design and evaluation methodology: PARAG consists of three major architecture components: the NPEs, the aggregation engine (AE), and the DRAM. The DRAM organizations used in this paper operate at 1.2 GHz. The energy, performance, and area models of the NPE were used from [18] which used well-established commercial flows to obtain the post-layout estimates. The controller and the FIFO in the AE are designed using Verilog and synthesized at 1.2 GHz using commercial tools in TSMC 40nm LP technology to obtain its power and area. The area and power of the buffer are obtained using CACTI [27] memory model at 40nm technology. The area, power, and energy numbers of the NPEs and the custom logic as shown in Table IV are used in the custom-designed behavioral simulator written in concert with DRAMPower [38]. This simulator takes a workload description and DRAM specifications as the input to characterize the latency and energy consumption of the PARAG architecture.

TABLE IV: Power and area of logic components added in PARAG.

Component	Power (mW)	Area (mm^2)
Controller	2.2	0.001
Feature Buffer	18.19	0.015
FIFO	4.84	0.005
NPE [18]	0.057	0.001

Hardware Configuration: This paper uses a DDR4-2400 4-Gb DRAM chip in DIMM configuration with 8 chips per Rank and a total of two Ranks with a total capacity of 8 GB. Each Chip is composed of 4 bank groups (BG) with 4 banks in each BG. The row buffer[4] size of each bank is 1 KB. As the NPE-array is interfaced with the row buffer, with each NPE operating on 4 bits, therefore, the NPE-Array has 2048 NPEs. The aggregation engine (AE) has 16-NPEs, 1 KB FIFO, and an output buffer of 1 KB.

This paper studies two configurations of PARAG using DDR-DIMM. (1) PARAG-2D-Bank-Group: this configuration has one NPE-Array per bank group, (2) PARAG-2D-Bank: which uses an NPE-Array for every bank in the memory. Further, each of PARAG configurations is also scaled for the 3D *hybrid memory cube (HMC)* with 8 GB total capacity. The 3D configurations of PARAG are referred to as PARAG-3D-Bank-Group and PARAG-3D-Bank and these configurations are used for comparison against the baselines which use 3D DRAM architectures with the same capacity.

Workloads: Evaluation of two types of GCN models, GCN [4] and GIN [22], and three commonly used graph datasets[5], Citeseer, DBLP, and Pubmed is presented. These datasets are chosen to have different characteristics (vertices, density, and size of feature vectors) resulting in the dominance of either the aggregation phase or the combination phase or having a balanced effect on the overall latency. For instance, the Citeseer dataset has a small number of vertices but large feature vectors, hence its combination phase is dominant. The Pubmed dataset has the opposite characteristics, whereas, in the case of the DBLP dataset, both phases contribute almost equally to the overall execution latency.

The PARAG architecture can be easily scaled with the DRAM capacity to implement even larger graphs. Both GCN and GIN use addition as the aggregate function and an MLP with a hidden layer size of 128 for the combination. GCN and GIN have one and two hidden layers respectively. In GCN, the combination phase is performed before aggregation, whereas in GIN, the combination follows the aggregation phase. The parameters of the datasets are shown in Table V.

TABLE V: Description of the datasets evaluated.

Dataset	Citeseer (CS)	DBLP (DB)	Pubmed (PB)
Vertices	3264	17716	19717
Edges	9430	105734	88648
Features	3703	1639	500
Classes	6	4	3
Density of *A*	0.09%	0.03%	0.02%

Baselines: PARAG is compared against three types of state-of-the-art platforms used for GCN acceleration.

1) GCIM (3D) [33] is a PIM architecture that adds custom logic for aggregation and combination on DRAM and logic layers of 3D hybrid memory cube (HMC). The 3D architecture has high memory bandwidth and parallelism and consumes the least amount of energy for internal data transfers due to short wiring distances among banks in the DRAM layers and the logic layer.
2) HyGCN (2.5D) [39] is a GCN accelerator synthesized as an ASIC using commercial tools and uses a HBM. The ASIC and the HBM communicate through an interposer. This is popularly known as 2.5D architecture.

[4]An array of bit-line sense amplifiers (BLSA) is also called row buffer in DRAM as it latches the entire row of data when a row is activated in a bank until the next activation command is issued to the same bank.

[5]Dataset source: https://github.com/EdisonLeeeee/GraphData

3) PyG [40] is a library optimized for deep learning for graphs. This is a software implementation for a CPU (PyG-CPU) or a GPU (PyG-GPU) with external DRAM.

The system configurations for the baseline architectures are shown in Table VI [33]. NVIDIA V100 GPU [39] is used.

TABLE VI: Configuration of the baseline platforms used.

	PyG-CPU	HyGCN	GCIM
Compute Elements	2.4 GHz @ 20 Cores	500 MHz @ 16 SIMD units with 16 cores and 4 (8 x 64 MACs) systolic arrays	500 MHz @ 64 LLUs with 4 MACs and 16 (8x16 MACs) systolic arrays
On-Chip Memory	27.5 MB	4.25 MB Buffers	2.125 MB
Off-Chip Memory	256 GB DDR4	8 GB HBM 1.0	8 GB HMC

Note that the results reported for HyGCN [39] and GCIM [33] are based on cycle accurate simulation similar to PARAG, whereas the results for PyG-CPU and PyG-GPU [39] are obtained from running software on commerical chips. PARAG and GCIM [33] are both PIM architectures and their computational capabilities are limited by the type of DRAM and its capacity. Hence, for a *fair* comparison, PARAG uses HMC of equal capacity as in GCIM [33], and cross-platform comparisons are made with the other baseline designs. If HBM is used instead of HMC for PIM architectures, the results are expected to be similar, given that for an equal capacity both, HBM and HMC have similar internal bandwidth.

Results and discussion: The evaluation of PARAG is carried out at three quantization levels of the GCN models: 8, 4, and 1 bit, and compared against the baseline architectures for latency, energy, and energy-delay product. PARAG performs full graph inference, i.e., the labels of all the vertices are being predicted.

1) Latency and Speedup:

Table VII presents the latency and energy consumption of the 2D configurations of PARAG for all six workloads. It shows that PARAG-2D-Bank has a lower latency than the PARAG-2D-Bank-Group for each of the corresponding bit precision. Since PARAG-2D-Bank has an NPE-Array attached to every bank, it can utilize the maximum bank-level parallelism in the combination phase by operating simultaneously on each graph partition. On the other hand, in PARAG-2D-Bank-Group an NPE-Array performs the combination phase serially on partitions stored in the banks in a bank group. Therefore in PARAG-2D-Bank-Group, the compute parallelism is reduced by a factor equal to the number of banks in a bank group. For any PARAG configuration, the latency of computation increases (decreases) with increasing (decreasing) bit-precision of the workload as the number of cycles for an operation on an NPE increases with increasing precision. Having flexibility without incurring any overhead is a significant advantage of PARAG.

Fig. 6 shows the latency comparison of the different baseline architectures against the various configurations of PARAG, with PARAG computing at 8 bits (highest latency). The comparison shows that PARAG-3D-Bank has the least latency among all the PARAG configurations by virtue of maximizing the bank-level parallelism in high bandwidth 3D HMC. As shown in Fig. 6, excluding one workload GCN on Citeseer dataset (GCN-CS) using PARAG-2D-Bank-Group with only 3.3% higher latency, all PARAG configurations have lower latency than the GCIM (minimum latency baseline) for all the workloads. On average over all workloads, GCIM, HyGCN, PyG-GPU, and PyG-CPU have **4×, 8.76×, 12.85×, and 1025×** higher latency than PARAG-3D-Bank configuration.

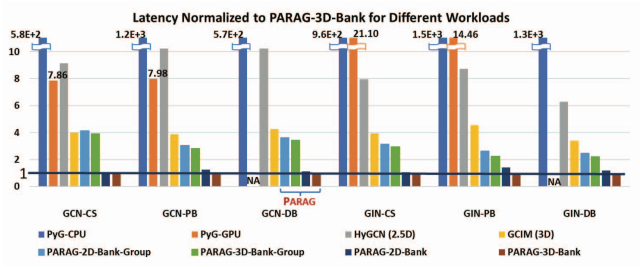

Fig. 6: Latency comparison of the baseline architectures to 8-bit computation on different configurations of PARAG. PARAG-3D-Bank has **4×, 8.76×, 12.85×, and 1025×** lower latency than on average than GCIM, HyGCN, PyG-GPU and PyG-CPU respectively.

How PARAG speeds up the computation? The acceleration of the aggregation phase is attributed to the three main factors facilitated by PARAG: (i) execution of the workload near the source of data leading to the reduction in the number of data transactions on the memory bus, (ii) parallel execution of the graph partitions in all the banks, and (iii) parallel execution of the aggregate function on the elements of the feature vector using NPE array in the aggregation engine (AE). The combination phase computation involves parallel MLP computation for all the nodes. The highest degree of parallelism within the DRAM is at the array level, where a single DRAM ACT command can supply Kbits of row-wide data to associated compute units. PARAG efficiently utilizes the array level parallelism with its NPE-Array composed of highly area-efficient and flexible NPEs. The ANs in NPEs enable the run-time reconfigurability of different bit-wise operations with zero overhead in latency and power. PARAG with its near array processing (NAP) through the NPE-array does not require the operand duplication to avoid overwriting the source data as is the case in the destructive operations in prior NAP architectures [13]. Therefore, PARAG results in a substantial decrease in the latency of the quantized combination phase.

2) Energy Consumption and Energy-Delay-Product:

Table VII shows the energy consumption of PARAG configurations for different workloads at 8, 4, and 1 bit of precision. Fig. 7 shows the energy consumption of baseline architectures and PARAG configurations normalized to the most energy efficient PARAG-3D-Bank-Group architecture for different workloads at 8 bits of precision. Therefore, PARAG-3D-Bank-Group has a relative value of 1. On average over all the workloads PARAG-3D-Bank-Group computing at 8-bit precision has **1.24×**, **3.41×**, **35.17×**, and **7904×** lower energy than GCIM, HyGCN, PyG-GPU and PyG-CPU. As shown in Table VII, the energy consumption can be further reduced up to **5×** to **6×** by reducing the bit precision of the workload.

Energy-delay-product (EDP) is an important metric for inference accelerators. It characterizes the energy efficiency

TABLE VII: Latency and energy consumption of PARAG-2D-Bank and PARAG-2D-Bank-Group.

Workload	Latency Normalized to 1 bit computation						Energy to 1 bit computation					
	PARAG-2D-Bank			PARAG-2D-Bank-Group			PARAG-2D-Bank			PARAG-2D-Bank-Group		
Bit Prec	**8 bit**	**4 bit**	**1 bit**	**8 bit**	**4 bit**	**1 bit**	**8 bit**	**4 bit**	**1 bit**	**8 bit**	**4 bit**	**1 bit**
GCN-CS	4.59	1.55	1.00	5.05	1.62	1.00	5.67	1.81	1.00	7.48	2.26	1.00
GCN-DB	2.58	1.24	1.00	4.02	1.46	1.00	4.94	1.68	1.00	5.66	1.91	1.00
GCN-PB	1.66	1.10	1.00	2.84	1.28	1.00	3.96	1.51	1.00	3.93	1.57	1.00
GIN-CS	5.52	2.09	1.00	5.33	1.79	1.00	5.87	1.98	1.00	7.62	2.52	1.00
GIN-DB	4.13	1.96	1.00	4.68	1.82	1.00	5.31	2.13	1.00	6.02	2.53	1.00
GIN-PB	2.49	1.48	1.00	3.35	1.54	1.00	4.62	1.90	1.00	4.83	2.09	1.00

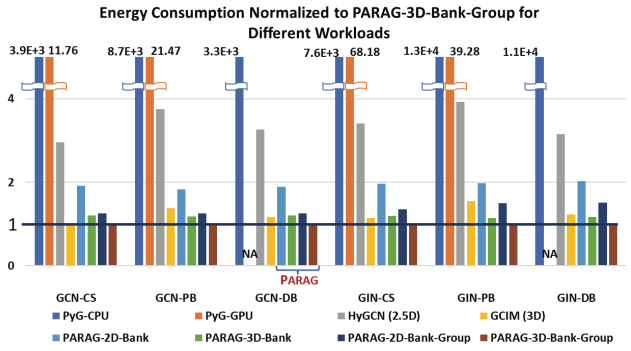

Fig. 7: Energy consumption comparison of the baseline architectures to 8-bit computation on PARAG configurations. PARAG-3D-Bank-Group has **1.24**×, **3.41**×, **35.17**×, and **7904**× lower energy than on average than GCIM, HyGCN, PyG-GPU and PyG-CPU respectively.

of the hardware to process multiple inference requests within a deadline. An accelerator with a low EDP is desirable in both battery-powered edge devices (to prolong battery life) and high-end servers. The fast response time and low energy consumption in the servers help to reduce operational costs. Fig. 8 shows the EDP comparison of the different PARAG configurations and PIM GCN accelerator GCIM [33]. For all the workloads, all PARAG configurations have a lower EDP except for GCN-CS workload, where PARAG-2D-Bank-Group has a higher EDP than GCIM. On average, PARAG-3D-Bank, has **4.2**×, **25.4**×, **460.57**× and **8x10**^{6}× lower EDP than GCIM, HyGCN, PyG-GPU, and PyG-CPU respectively.

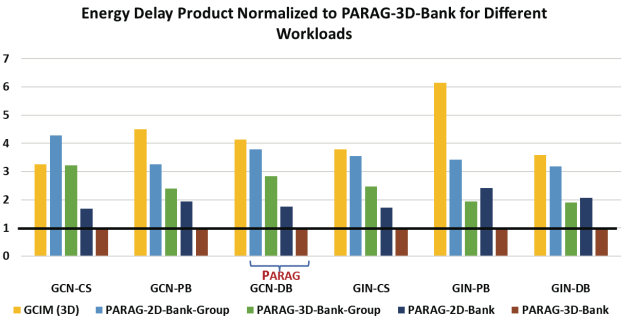

Fig. 8: Energy-Delay-Product (EDP) comparison of GCIM [33] to 8-bit computation on different configurations of PARAG. PARAG-3D-Bank has **4.23**×**, 25.37**×**, 460.57**×**, and 8x10**^{6}× lower EDP than on average than GCIM, HyGCN, PyG-GPU, and PyG-CPU respectively.

Key takeaway: PARAG achieves all the improvements by utilizing the maximum parallelism and bandwidth in DRAM. This is possible in PARAG due to the low area and power and run-time reconfigurable design of the NPE [18]. Large arrays of NPEs can be added to each bank of the DRAM within its area overhead constraints and perform all the GCN operations.

V. Related Work

Software frameworks: GCN execution on CPUs and GPUs is performed using various programming frameworks and algorithms. Deep graph library (DGL) [41], Neugraph [42], etc. are popular GNN frameworks. These frameworks running on CPUs and GPUs are energy inefficient due to memory bottlenecks and GPU under-utilization for irregular data.

Graph processing accelerators: Graph processing requires high memory bandwidth due to the irregular and sparse structure of real-world graphs. For these, near data processing (NDP) architectures such as GraphH [43], and GraphQ [44] have been proposed. These NDP architectures use HMC or HBM with high internal memory bandwidth. Other PIM graph processing architectures based on DRAM [45], and ReRAM [46] have also been proposed. Although these architectures provide high energy efficiency and performance, they are unsuitable for GCN as they cannot leverage the high parallelism and data regularity of the combination phase.

Neural network accelerators: Neural network accelerators like Eyeriss [47] use systolic arrays of MAC processing elements to perform convolution. They optimize memory interfaces (DianNao [48]) or adopt hardware compression techniques (EIE [49]) to reduce memory usage. Some accelerators such as Sparten [50]) handle sparsity in convolutions and can perform sparse matrix multiplication (SpMM) in GCN but lose their advantages for convolutions. CNN accelerators are highly optimized only for regular memory accesses and high computation intensity and, therefore, are not ideal for GCN.

GCN accelerators: Several specialized hardware for GCN have been proposed, including high-level synthesis [51], FPGA architectures [52], [53], hardware-software co-design frameworks [54], and ASIC architectures [55]. LL-GNN [53] with novel matrix multiplication implementation improved the latency of GNN execution on an FPGA.HyGCN [39] was the first ASIC with separate aggregation and combination engines for GCN. AWB-GCN [56] proposes a runtime technique to balance the workload to increase the utilization of the PEs and achieve about 5.1× speedup over the HyGCN. EnGN [57] has a unified architecture and uses the ring edge reduce (RER) technique. GCNAX [55] was later proposed to optimize data flow. Custom ASICs have fixed data flow and do not

perform well with larger datasets, as performance is tied to the size of feature vectors, the graph structure, and the CNN parameters. Both the FPGAs and ASICs have resource constraints and due to external DRAM accesses, they suffer from memory bottlenecks and have reduced performance and energy efficiency for larger workloads.

VI. Conclusion

This paper presents PARAG a PIM architecture that integrates the functional capabilities of near data processing (NDP) and near array processing (NAP) architectures in a DRAM resulting in an efficient architecture for GCN. PARAG has an aggregation engine (AE) integrated with each DRAM bank to circumvent the memory bottleneck of the aggregation phase and NPE-Array interfaced with the memory arrays to compute the neural network of the combination phase. This paper evaluates two models of GCN over three graph datasets. Simulation results show that the PARAG-3D-Bank configuration achieves on an average 4× latency and 4.23× EDP reduction against the state-of-the-art PIM GCN accelerator.

References

[1] J. Redmon et.al. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE CVPR*. IEEE.

[2] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *CoRR*, abs/1602.07261, 2016.

[3] T. B. Brown et. al. Language Models are Few-Shot Learners. *CoRR*, abs/2005.14165, 2020.

[4] T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR*, abs/1609.02907, 2016.

[5] Z. Zhang et.al. Deep learning on graphs: A survey. *IEEE TKDE*, 34(1), Jan 2022.

[6] N. De Cao and T. Kipf. MolGAN: An implicit generative model for small molecular graphs. 2018.

[7] A. Lerer et. al. PyTorch-BigGraph: A Large-scale Graph Embedding System. *CoRR*, abs/1903.12287, 2019.

[8] R. Zhu et. al. AliGraph: a comprehensive graph neural network platform. *Proceedings of the VLDB Endowment*, 12(12), Aug 2019.

[9] R. Ying et. al. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *2018 ACM KDD*.

[10] J. Yang and J. Leskovec. Defining and Evaluating Network Communities based on Ground-truth, 2012.

[11] B. Zhang et al. Efficient neighbor-sampling-based gnn training on cpu-fpga heterogeneous platform. In *2021 IEEE HPEC*.

[12] S. Abadal et. al. Computing Graph Neural Networks: A Survey from Algorithms to Accelerators. *ACM Comput. Surv.*, 54(9), oct 2021.

[13] N. Hajinazar et al. SIMDRAM: a framework for bit-serial SIMD processing using DRAM. In *ASPLOS'21*.

[14] M. Zhou et. al. HyGraph: Accelerating Graph Processing with Hybrid Memory-centric Computing. In *2021 DATE*.

[15] J. Kim et al. Aquabolt-XL HBM2-PIM, LPDDR5-PIM With In-Memory Processing, and AXDIMM With Acceleration Buffer. *2022 IEEE Micro*, 42(3):20–30, May.

[16] X. Xie et. al. MPU: Towards Bandwidth-abundant SIMT Processor via Near-bank Computing. *CoRR*, abs/2103.06653, 2021.

[17] G. Singh et. al. NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling. In *2020 FPL*.

[18] A. Wagle et al. A configurable BNN ASIC using a network of programmable threshold logic standard cells. In *2020 IEEE ICCD*.

[19] S. Muroga. *Threshold logic and its applications*. Wiley-Interscience, New York, 1971.

[20] A. Wagle et al. A Novel ASIC Design Flow Using Weight-Tunable Binary Neurons as Standard Cells. *IEEE TCAS I: Regular Papers*, 2022.

[21] W. Hamilton et al. Inductive Representation Learning on Large Graphs, 2018.

[22] K. Xu et al. How Powerful are Graph Neural Networks?, 2019.

[23] S. A. Tailor et al. Degree-Quant: Quantization-Aware Training for Graph Neural Networks, 2021.

[24] J. Wang et. al. Bi-GCN: Binary Graph Convolutional Network. In *2021 IEEE/CVF CVPR*. IEEE Computer Society, jun.

[25] High Bandwidth Memory (HBM). https://www.jedec.org/document_search?search_api_views_fulltext=jesd235/, 2013.

[26] J. Jeddeloh and B. Keeth. Hybrid memory cube new DRAM architecture increases density and performance. In *2012 VLSIT*.

[27] N. P. Jouppi et al. CACTI-IO: CACTI with off-chip power-area-timing models. In *2012 IEEE/ACM ICCAD*.

[28] J. Gomez-Luna et al. Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access*, 10:52565–52608, 2022.

[29] S. Lee et al. A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications. In *2022 IEEE ISSCC*.

[30] Q. Deng et al. DrAcc: a DRAM based accelerator for accurate CNN inference. In *DAC'18*.

[31] D. Chen et al. A General Offloading Approach for Near-DRAM Processing-In-Memory Architectures. In *IEEE IPDPS*, 2022.

[32] L. Huang et al. Practical Near-Data-Processing Architecture for Large-Scale Distributed Graph Neural Network. *IEEE Access*, 2022.

[33] J. Chen et al. GCIM: Toward Efficient Processing of Graph Convolutional Networks in 3D-Stacked Memory. *IEEE TCAD*, 2022.

[34] J. Yang et al. Integration of Threshold Logic Gate Circuit with RRAM Devices for Low Power, and Robust Operation. In *Nanoarch'14*.

[35] N. Kulkarni et al. Reducing Power, Leakage, and Area of Standard-Cell ASICs Using Threshold Logic Flip-Flops. *TVLSI'16*.

[36] G. Karypis and V. Kumar. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1997.

[37] Zhenkun Cai et al. DGCL: An Efficient Communication Library for Distributed GNN Training. In *ACM EuroSys '21*, EuroSys '21.

[38] K. Chandrasekar et al. DRAMPower: Open-source DRAM Power and Energy Estimation Tool,. http://www.drampower.info/.

[39] M. Yan et. al. HyGCN: A GCN Accelerator with Hybrid Architecture. In *2020 IEEE HPCA*. IEEE Computer Society.

[40] M. Fey and J. E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. *CoRR*, abs/1903.02428, 2019.

[41] M. Wang et. al. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *CoRR*, abs/1909.01315, 2019.

[42] L. Ma et. al. Neugraph: Parallel Deep Neural Network Computation on Large Graphs. In *2019 USENIX ATC '19*, USA.

[43] G. Dai et al. GraphH: A Processing-in-Memory Architecture for Large-Scale Graph Processing. *IEEE TCAD*, 2019.

[44] Y. Zhuo et. al. GraphQ: Scalable PIM-Based Graph Processing. In *IEEE/ACM MICRO '52*, 2019.

[45] S. Angizi et al. GraphiDe: A Graph Processing Accelerator leveraging In-DRAM-Computing. In *GLSVLSI'19*, 2019.

[46] L. Song et. al. GraphR: Accelerating Graph Processing Using ReRAM. In *2018 IEEE HPCA*.

[47] Y. Chen et. al. Eyeriss v2: A flexible and high-performance accelerator for emerging deep neural networks. *CoRR*, abs/1807.07928, 2018.

[48] T. Chen et. al. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *ASPLOS '14*, page 269–284.

[49] S. Han et. al. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *IEEE ISCA '16*.

[50] A. Gondimalla et. al. SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks. In *IEEE/ACM MICRO '52*.

[51] B. Zhang et. al. Hardware Acceleration of Large Scale GCN Inference. In *2020 IEEE ASAP*, pages 61–68.

[52] S. Abi-Karam et. al. GenGNN: A Generic FPGA Framework for Graph Neural Network Acceleration. *CoRR*, abs/2201.08475, 2022.

[53] Z. Que et al. LL-GNN: Low Latency Graph Neural Networks on FPGAs for High Energy Physics, 2023.

[54] J. R. Stevens et. al. GNNerator: A Hardware/Software Framework for Accelerating Graph Neural Networks. In *2021 ACM/IEEE DAC*.

[55] J. Li et. al. GCNAX: A Flexible and Energy-efficient Accelerator for Graph Convolutional Neural Networks. In *2021 IEEE HPCA*.

[56] T. Geng et al. AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing. In *IEEE/ACM MICRO*, 2020.

[57] S. Liang et. al. EnGN: A High-Throughput and Energy-Efficient Accelerator for Large Graph Neural Networks. *IEEE TC*, 2021.