DARe: DropLayer-Aware Manycore ReRAM architecture for Training Graph Neural Networks

Aqeeb Iqbal Arka*, Biresh Kumar Joardar†, Janardhan Rao Doppa*, Partha Pratim Pande*, and Krishnendu Chakrabarty†

*School of EECS, Washington State University
Pullman, WA 99164, USA.
{aqeebiqbal.arka, jana.doppa, pande}@wsu.edu

†Department of ECE, Duke University
Durham, NC 27708, USA.
{bireshkumar.joardar, krish}@duke.edu

Abstract— Graph Neural Networks (GNNs) are a variant of Deep Neural Networks (DNNs) operating on graphs. GNNs have attributes of both DNNs and graph computation. However, training GNNs on manycore architectures is a challenging task because it involves heavy communication that bottlenecks performance. DropEdge and Dropout, which we collectively refer to as DropLayer, are regularization techniques that can improve the predictive accuracy of GNNs. Moreover, when implemented on a manycore architecture, DropEdge and Dropout are capable of reducing the on-chip traffic. In this paper, we present a ReRAM-based 3D manycore architecture called *DARe*, tailored for accelerating on-chip training of GNNs. The key component of the *DARe* architecture is a Network-on-Chip (NoC) that reduces the amount of communication using DropLayer. The reduced traffic prevents communication hotspots and leads to better performance. We demonstrate that *DARe* outperforms conventional GPUs by up to 6.7X (5.6X on average) in terms of execution time, while being up to 30X (23X on average) more energy efficient for GNN training.

Keywords—3D architectures, GNNs, NoC, ReRAM, DropEdge, Dropout

I. INTRODUCTION

Graph Neural Networks (GNNs) are used for predictive analytics using graph-structured data. This makes them different from traditional Deep Neural Networks (DNNs) that operate on regular data structures such as images or sequences. GNNs have various real-life applications such as recommendation systems [1], quantum chemistry [2], social networks [3] [4] etc. To learn representation using the relational structure of graphs, GNNs perform iterative neighborhood aggregation, where each node aggregates features of its neighbors to compute new features [5]. This gives rise to repeated message-passing operations. GNNs exhibit characteristics of both DNN training (involving trainable weights) and graph analytics (accumulating neighboring vertices' information along graph edges). Hence, GNN training is both compute- and communication-intensive.

Project.All the computations associated with GNN training can be represented as multiply-and-accumulate (MAC) operations. Resistive random-access memory (ReRAM) can implement such MAC operations efficiently [6]. Moreover, ReRAMs allow for processing-in-memory (PIM), which greatly reduces the communication between the computing cores and the main memory. However, GNN training involves repeated message-passing operations to accumulate neighbor information in a recursive manner [7]. When implemented on a ReRAM-based manycore architecture, this can give rise to a substantial amount of on-chip traffic that creates performance bottlenecks if not addressed appropriately.

DropEdge and Dropout are regularization techniques that can help in reducing the amount of traffic during GNN training; normally, these layers are used to prevent over-smoothing and over-fitting during GNN training. Over-fitting limits the generalization ability on small datasets, while over-smoothing isolates output representations from the input features as network depth increases; both these phenomena lead to poor model accuracy [8]. As the name suggests, DropEdge involves randomly dropping a few edges of the input graph during training. In practice, this is realized when information from a few randomly selected neighbors of each node of the input graph is not accumulated during the message-passing stage of GNN training. Thus, DropEdge reduces the number of messages communicated among adjacent nodes, which in turn improves performance. The Dropout operation in DNNs randomly skips a neural unit in the network (along with the associated connections) temporarily [9]. This process simulates the creation of a large ensemble of sparse DNNs resulting in a single trained GNN with small weights to approximate the effect of average predictions from this ensemble. Overall, both Dropout and DropEdge reduce the generalization error (error on testing instances drawn from the target data distribution) in GNN training, while also reducing the amount of on-chip communication. Note that the overall goal of learning is to achieve low error on unseen data (i.e., low generalization error). In the rest of this paper, we collectively refer to DropEdge and Dropout as "DropLayer".

Despite these advantages, implementing DropLayer in manycore ReRAM-based architectures is a challenging task. It is well-known that the output of one GNN layer is the input to the next GNN layer, and so on [10]. However, due to the inherently random nature of DropLayer, different edges of the graph and neural units are dropped in each iteration. This results in a randomly varying traffic pattern where the data exchanged between two adjacent GNN layers keeps changing in every iteration; the change can be in terms of the data size, content, and/or the source-destination pair. Hence reliable communication is impossible without an appropriate control mechanism. In this work, we enable DropLayer in the NoC-enabled manycore architecture using variable-sized packets and a suitable control mechanism. We call the NoC incorporating DropLayer as the Drop-aware NoC architecture. Overall, the Drop-aware NoC reduces the communication, thereby leading to better performance.

Reducing the amount of on-chip communication through the Drop-aware NoC is one aspect of reducing the traffic bottleneck. The other desirable characteristics of the NoC should be to support the randomly varying traffic pattern exhibited by GNN training. Moreover, GNN training exhibits heavy many-to-few and multicast traffic [11]. Traditional planar (2D) architectures are not well-suited for such traffic patterns. As the large physical separation between processing elements (PEs) causes high-latency and low-throughput, it is not ideal for high-performance GNN training. Prior work has shown that three-dimensional (3D) NoC enables the design of a high-throughput and low-latency communication backbone for manycore chips by lowering the physical distance among

This work was supported in part by the US National Science Foundation (NSF) under grants CNS-1955353, CNS-1955196, and by the USA Army Research Office grant W911NF-17-1-0485. Biresh Kumar Joardar was also supported in part by NSF Grant # 2030859 to the Computing Research Association for the CIFellows Project.

2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 978-1-6654-4507-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCAD51958.2021.9643511

the PEs [12]. Moreover, 3D NoCs can support high-throughput multicast, which is essential for GNN training. In this paper, we present the design of a ReRAM-based manycore architecture integrated via a Drop-aware 3D NoC. The NoC architecture helps to alleviate the communication bottleneck in GNN training. We call this manycore architecture as "*DARe*". The main contributions of this paper include:

- We design a manycore architecture (*DARe)* for accelerating GNN training. *DARe* leverages the benefits introduced by the ReRAM-based PEs and an efficient on-chip communication infrastructure enabled by a Drop-aware 3D NoC.
- To maximize performance gain and energy efficiency, we balance the computation and communication latencies. The communication latency is reduced via the Drop-aware NoC, whereas the computation latency is reduced by allocating adequate number of ReRAM PEs to each GNN layer.
- Through extensive performance evaluation, we demonstrate that *DARe* outperforms both traditional GPU-based designs and state-of-the-art ReRAM-based architectures for training GNNs using diverse real-world graphs with millions of nodes and edges.

To the best of our knowledge, this is the first work that proposes a Drop-aware NoC architecture to accelerate GNN training in a ReRAM-based manycore system. The rest of the paper is organized as follows. Section II describes relevant prior work. In Section III, we discuss GNN characteristics and the effect of DropLayer on GNN traffic. In Section IV, we introduce the proposed *DARe* architecture, the design methodology for the Drop-aware 3D NoC, and the policy for mapping the GNN layers to the *DARe* architecture. Section V presents experimental results. We conclude the paper in Section VI by summarizing the findings of this work.

II. Related Prior Work

A. GNN training algorithms

The primary idea behind GNNs is to aid learning on graph structured data by using a neural network that can operate directly on graphs [10]. However, GNN training on large-scale graphs is very memory intensive, which necessitates the use of efficient graph partitioning. Using graph partitioning, the Cluster-GCN approach enables scalable GNN training over large graphs with high accuracy and speed [13]. However, even with graph partitioning, GNN training is communication intensive due to recursive message-passing operations. DropEdge is a regularization technique similar to Dropout in DNNs that reduces the number of times the neighboring node features are aggregated in GNN training while also reducing over-fitting and over-smoothing, thereby resulting in improved accuracy [8]. However, as mentioned earlier, the hardware implementation of DropLayer (DropEdge and Dropout) requires a suitable control mechanism to ensure proper synchronization between the communicating PEs. Hence, a Drop-aware communication protocol is necessary for high-performance GNN training in a manycore architecture, without sacrificing accuracy.

B. Hardware for GNN computation

The design of hardware architectures for GNN computation using commodity processors, FPGAs and custom ASICs has been considered in recent work [7], [11], [14] [15]. However, all these architectures primarily focus on GNN inference but not on training. The training of GNN is considerably more challenging due to the additional data exchange between the forward and backward phases. GNN training is primarily implemented using GPUs. It is well known that GPUs are not optimized for GNN training, which leads to sub-optimal performance [14]. Moreover, all these custom architectures (e.g., [7], [11], [15]) are limited to relatively small graph structures, which do not require large amounts of memory and computation. In contrast, this paper focuses on an on-chip architecture enabled by ReRAM-based PEs for GNN training with large graphs.

C. ReRAM-based architectures

ReRAMs enable processing-in-memory (PIM) that allows for in-situ MAC (IMA) operations [6]. Computations in both DNN and graph analytics can be decomposed into simple MAC operations [11]. ReRAM-based accelerators for graph analytics have been shown to significantly outperform CPU- or GPU-based systems, both in terms of execution time and energy [16] [17] [18]. Moreover, ReRAMs have been used extensively to accelerate both DNN training and inference [19] [20] [21] [22]. However, these solutions focus mainly on accelerating only the computation [19] [20]. The maximum achievable performance will be bottlenecked unless the communication is also optimized [23]. In addition, all these ReRAM-based accelerators are fine-tuned for either DNN or graph analytics and are not suitable for GNN training; GNN training exhibits features of both DNN and graph computation simultaneously. An NoC-enabled ReRAM-based architecture for high-performance training of GNNs, referred as ReGraphX, has been proposed in [24]. However, as we show in this work, ReGraphX does not realize the full potential of ReRAMs due to communication bottlenecks inherent in GNN training. Our work addresses a key shortcoming of the state-of-the art by proposing an architecture that focuses on inherent communication bottleneck of GNN training. The *DARe* architecture incorporates a Drop-aware 3D NoC and ReRAM-based PEs that enable high-performance training of GNNs.

III. GNN Kernel and Communication Traffic

In this section, we discuss the GNN training process. Next, we explain how DropLayer reduces GNN traffic when implemented on the *DARe* architecture.

A. The GNN Kernel

Computation: A graph consists of (a) vertices: each vertex is represented using a feature vector that characterizes the node; and (b) edges: the edges are represented by an adjacency matrix indicating the vertex connectivity (α), where α is a sparse matrix. Together, vertices and edges define a graph and are crucial for the computational kernel of GNNs. A GNN consists of multiple back-to-back neural layers. Each neural layer further consists of two sub-layers that perform two different types of computations: (a) Vertex sub-layer: the computations associated with this layer are MAC operations similar to conventional DNNs, and (b) Edge sub-layer: this resembles the message passing operation in graph analytics.

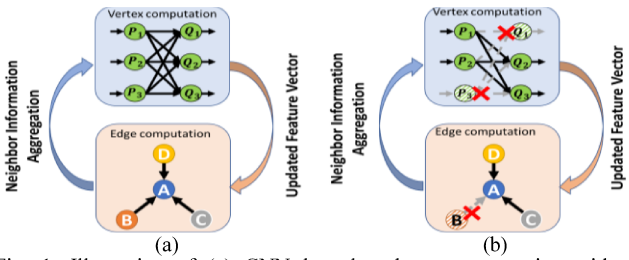
Fig. 1(a) depicts the two types of computations in a GNN for an example graph with four nodes (A, B, C and D) and three edges (for forward phase only). The vertex sub-layer of the GNN resembles a fully connected DNN layer with input and output nodes denoted as P_i and Q_i respectively. Computations associated with the vertex sub-layer involve multiplying the node features with weights (ω_l for layer l) to

compute the updated feature vectors of each node in the graph. The updated feature vector is then accumulated by each node in the graph during the computation of edge sub-layer as shown in Fig. 1. For instance, node A accumulates the information of its neighbors (nodes B, C and D) as shown in Fig. 1(a); other nodes perform the same task in parallel as well. This neighbor information aggregation is similar to message passing in graphs and can be represented as multiplication of the updated node feature vectors with the graph adjacency matrix (α). Hence, the edge computation can also be decomposed as MAC operations that can benefit from a ReRAM-based architecture such as *DARe*. These two sub-layers (vertex and edge sub-layers) are executed repeatedly one-after-another to get the final output as shown in Fig. 1(a).

Similar to traditional DNNs, the backward phase computations of GNN consist of error/gradient calculation and weight updates, both of which are also predominantly MAC operations [20]. As a result, the backward phase computations can also be efficiently implemented using ReRAM-based PEs similar to the forward phase. The error gradients of one layer can then be communicated to its preceding layer via the NoC. Hence, both forward and backward phases of GNN training are executed using ReRAM-based PEs in the *DARe* architecture.

Communication: To train a GNN using ReRAMs, the weights of each vertex sub-layer (ω_l) and the graph adjacency matrices (α) must be first mapped to the ReRAM-based PEs [19]. Let us consider a GNN with three neural layers as an example. As each layer consists of two sub-layers, there will be three vertex sub-layers (V_1, V_2 and V_3,) and three edge sub-layers (E_1, E_2,and E_3). The vertex sub-layer (V_l) and edge sub-layer (E_l) of a neural layer l are executed alternately as shown in Fig. 1(a). The loop in Fig. 1(a) is repeated three times (as there are three neural layers). The output of a vertex sub-layer V_l is used as an input for the edge sub-layer, E_l; the output of E_l is then used as an input to the next vertex sub-layer, V_{l+1} and so on. Each vertex sub-layer involves a unique set of weights, i.e., the vertex sub-layers V_1, V_2,and V_3 have weights ω_1, ω_2 and ω_3 associated with them respectively and $\omega_1 \neq \omega_2 \neq \omega_3$ (similar to conventional DNN). Hence, each set of weights need to be mapped/assigned to a unique set of PEs for computation. These PEs compute the updated feature vectors (vertex sub-layer), which is then used by the edge sub-layer.

The computation in the edge sub-layer involves the graph adjacency matrix that must also be stored in ReRAM crossbars [16]. The adjacency matrix α is mapped to a separate set of PEs than weights ω_1, ω_2, and ω_3. However, unlike the vertex sub-layers, the adjacency matrix is constant across different edge sub-layers, i.e., E_1, E_2,and E_3 share the same α. Hence, the computation of edge sub-layers associated with different neural layers must be done on the same PE that stores the adjacency matrix i.e., the PEs storing the adjacency matrix will be shared by all the neural layers in the GNN.



(a) (b)

Fig. 1. Illustration of (a) GNN kernel and message passing without DropLayer (b) GNN training with DropLayer.

As shown in Fig. 1, the output of one computation is used as the input for the next set of computations. Hence, the PEs storing ω_1, ω_2, and ω_3 need to communicate their outputs to the PEs storing α, and vice-versa. However, as the PEs storing α is shared by all neural layers, the data transfer will result in a many-to-few communication pattern. Without a suitable NoC, such a traffic pattern can result in traffic hotspots, resulting in poor performance [23]. Hence, the manycore architecture needs to be supported by a suitable NoC for accelerating GNN training.

B. *Effect of* DropLayer *on GNN Traffic*

Both DropEdge and Dropout (recall that they are collectively referred as DropLayer in this paper) are data-augmentation techniques that address over-fitting and over-smoothing in GNN training [8]. In this sub-section, we explain how DropLayer can reduce communication during GNN training. As shown in Fig. 1(a), without DropEdge, all the nodes (A, B, C and D) aggregate neighbor information from all their neighbors in every epoch during the edge sub-layer. For instance, node A aggregates information from all three neighbors (B, C and D). Hence, three sets of messages are passed in every epoch; here a message represents the data that is gathered from each neighboring node. Similarly, without Dropout in the vertex sub-layer, all the weights are used for MAC operations. Hence, all the inputs and outputs associated with the vertex sub-layer must be communicated.

Fig.1(b) shows how DropLayer can reduce communication during GNN training. DropEdge temporarily removes few edges randomly at each epoch during computation in the edge sub-layer while Dropout temporarily removes a few nodes in the vertex layer. For instance, as shown in Fig. 1(b), DropEdge omits the edge between node "A" & "B"; hence, the data from node "B" is not accumulated by node "A". This leads to one fewer message during the message passing stage of GNN training. Similarly, Dropout omits the node P_3 and Q_1 in Fig. 1(b). As a result, the computations associated with these nodes are omitted. Consequently, the associated inputs and outputs to these nodes need not be communicated between the PEs as well. As a result, collectively DropLayer reduces the amount of data that need to be communicated between the PEs. We utilize this property in the proposed *DARe* architecture to achieve high-performance, without sacrificing accuracy.

IV. Overall *DARe* Architecture

In this section, we first present the ReRAM crossbar configuration of the *DARe* architecture. Then, we present the design of a control mechanism that enables the DropLayer operations. Next, we discuss the overall GNN training strategy on *DARe*.

A. *ReRAM Crossbar Configuration in DARe*

Fig. 2 shows the overall *DARe* architecture. As shown in Fig. 2, each ReRAM-based PE contains multiple crossbars for executing MAC operations and a router for data exchange.

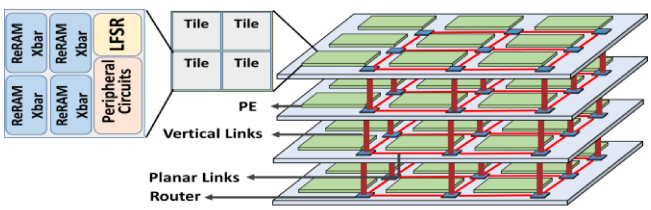


Fig. 2. *DARe* architecture. This figure is for illustration purposes only.

First, we must determine the most suitable crossbar configuration for GNN training. Conventionally, graph computations involve sparse adjacency matrices. Hence, relatively smaller ReRAM crossbars are used as they avoid/reduce the storage of zeros (zeros are redundant in MAC operations) [17]. On the other hand, DNNs are implemented using relatively larger ReRAM crossbars (e.g., 128x128 as shown in [19]) because weight matrices tend to be relatively dense. As elaborated in Section III, the training process of GNNs exhibits attributes of both DNNs and graph computations. This presents a challenge while choosing the suitable crossbar size for training GNNs.

From prior work, it is well-known that peripheral circuits dominate the ReRAM tile area [19]. Hence, the ReRAM tile's area and power consumption do not vary significantly with the crossbar size. However, smaller crossbars have significantly lower storage capacity or density (bits stored per unit area). A crossbar of size $N \times N$ can store up to N^2 values. Hence, to match the storage capability of one 128x128 crossbar, we will need up to 256 (=$128^2/8^2$) 8x8 crossbars [19]. Higher crossbar requirement leads to more tiles, which necessitate more area (for more peripherals such ADC, DAC, etc.). Moreover, more tiles consume more energy overall despite having lower energy consumption per tile due to more efficient zero storage and low-resolution ADCs.

To determine a suitable ReRAM crossbar configuration for training GNNs, we consider different ReRAM crossbar sizes varying from 8x8 to 256x256 for *DARe*. The choice of crossbar size should be such that the overall area (and power consumption) is minimized. We observe that smaller crossbars lead to many tiles but store fewer zeros, and larger crossbars necessitate fewer tiles but store many zeros. Our analysis, as shown in Section V, indicates that for all input graphs considered in this work for GNN training, 128x128 sized crossbars are the best choice in terms of storage-power-area trade-offs. Hence, we use 128x128 sized crossbars as the ReRAM configuration of choice for *DARe*.

B. *Implementing* DropLayer *in DARe*

As discussed earlier, DropLayer translates to reduced inter-PE traffic in the manycore architecture. However, the randomness inherent in DropLayer makes it challenging to implement on conventional manycore architectures. Randomly dropping data results in a dynamically changing traffic pattern, where the change can be with respect to the size or content of data that need to be communicated. This makes communication especially challenging because the destination PE must be able to correctly decode the received data every time. Fig. 3 highlights the control mechanism needed to implement DropLayer on ReRAM-based manycore architectures. For instance, let us assume that the ReRAM computations of a neural layer yields the output: (d_1, d_2, d_3, d_4); this needs to be communicated to the ReRAM PEs responsible for the next neural layer, as shown in Fig. 1. It should be noted that, d_i is a 16-bit fixed point number [19]. The data must then be communicated via the NoC using packets. In NoC packet structure, d_i represents a flit and multiple such flits together constitute a packet. Without DropLayer, none of this data is omitted and we would communicate all the flits (d_1, d_2, d_3,and d_4) in a single packet, to the appropriate destination PEs for further processing. However, due to the incorporation of DropLayer, some flits will be omitted. For instance, if we omit d_2 and d_4, the destination PE will receive a packet with the following flits only: (d_1, d_3); the receiver PE must ideally interpret the data packet as: (d_1, 0, d_3, 0) i.e., d_2 and d_4 are omitted. However, without additional information, the receiver PE will not be able to correctly identify the missing flits, thereby leading to erroneous interpretations. In addition, the position and number of the missing flit keeps changing every epoch due to the inherent randomness within DropLayer. Hence, this dynamically changing packet structure and content requires a control mechanism to ensure seamless data exchange between the PEs. Here, it should be noted that, the omitted flits in each packet are determined in a random manner. Therefore, it is difficult to merge flits from different packets to create a new uniform packet structure.

In this paper, we solve this challenging problem and enable DropLayer using a dynamically varying packet structure in the manycore architecture [25]. In wormhole routing [26], this implies that the data packets will contain different number of flits. In order to implement DropLayer in the NoC-based architecture, we use a reconfigurable Linear Feedback Shift Register (LFSR) based control mechanism to decide which data to omit in each epoch.

Fig. 3(a) and Fig. 3(b) explain the proposed methodology in more detail using the example of four data flits in a packet. The LFSR-based control mechanism in Fig. 3(b) generates a 4-bit binary pattern, referred as the key (each bit of the key is represented by k_i) to capture the effects of DropLayer in the NoC. Each bit of the LFSR is used to decide whether to keep/discard a flit represented as d_i in Fig. 3, i.e., d_i will be omitted if $k_i = 0$, and vice versa (as shown in Fig. 3(b)). For instance, for the 4-flit packet structure, when the LFSR bit pattern (key) is "1010", outputs d_2 & d_4 are omitted while d_1 & d_3 are included in the packet following the proposed control mechanism shown in Fig. 3(a). The LFSR pattern (key) is then added to the header flit and sent to the destination PE. The key is used to decode the received data packet at the destination PE. For instance, the LFSR pattern ("1010") indicates that rows R_1 and R_3 of the destination crossbar will receive the information contained in d_1 and d_3, respectively. The size of the key is always equal to the bit-width of the LFSR. The number of 1's in the key is used by the NoC router to determine the packet size i.e., the number of body flits. For instance, the key 1010 indicates that there will be two body flits in the packet.

As mentioned in Section III, DropLayer results in some messages being omitted from the packet due to the randomly dropped edges (or neural units). As a result, each message generated by the neural layers need to be associated with one

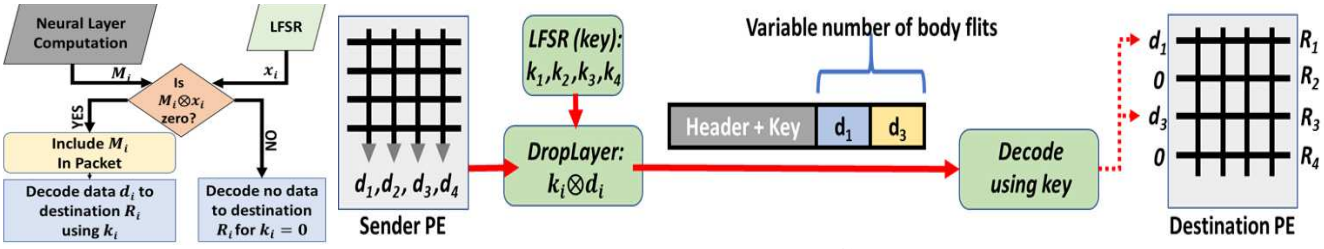


Fig. 3. (a) The control mechanism to enable DropLayer in *DARe*. (b) Hardware block emulating DropLayer. The figure is for illustration purposes only.

bit from the LFSR bit pattern. The bit width of the LFSR is determined considering the throughput (number of outputs generated in one cycle) of the ReRAM crossbar. For instance, if the ReRAM crossbar generates N outputs in one cycle, we must have an L-bit LFSR where,

$$L = N * (f_{ReRAM}/f_{LFSR}) \quad (1)$$

Here, f_{LFSR} and f_{ReRAM} are the clock frequencies of the LFSR and ReRAM crossbars, respectively. Note that, ReRAM crossbars and LFSRs have different clock frequencies. Generally, ReRAMs operate with a significantly slower clock frequency than the CMOS peripherals [19]. Using Equation (1), we determine the bit width of the specific LFSR configuration necessary for *DARe.*

C. Overall GNN training on DARe

In addition to the variable sized packets due to DropLayer, GNN training exhibits many-to-few (discussed in Section III) and multicast traffic patterns as shown in Fig. 4. Fig. 4(a) illustrates the communication pattern of GNN training for a 3-layer GNN. Fig. 4(b) and Fig. 4(c) show the resulting traffic pattern when the aforementioned GNN is trained on a manycore architecture. Note that the mappings shown in Fig. 4(b) and Fig. 4(c) are only examples. The many-to-few traffic pattern can create traffic hotspots (shown in Fig. 4(b)), which affects the achievable performance of the *DARe* architecture. Moreover, the multicast communication primarily arises as data from PEs executing layer l is shared with PEs responsible for layer $l+1$ (next layer) and PEs executing the backward phase of layer l. Note that the forward and backward phases of training are often implemented on separate sets of PEs for high throughput [19]. Also, the same neural layer is often distributed across multiple PEs, creating even more multicast traffic. As an example, if layer $l+1$ is mapped to three different PEs, the same output from layer l needs to be communicated to all three PEs (multicast traffic) as shown in Fig. 4(c). Traditional planar architectures are not suited for such traffic patterns, which result in a significant amount of long-range communication and multi-hop traffic due to the larger physical separation between communicating PEs. Hence, the overall system throughput is affected [12].

3D integration alleviates this problem by stacking planar tiers on top of each other. This reduces the physical separation between the PEs, resulting in faster communication. By adding an extra z-dimension to the NoC, path diversity during communication is improved [12]. This results in higher throughput, a key requirement for training GNNs. In addition, 3D NoC architectures can enable high-performance multicast support, which is essential for GNN training [23]. Fig. 2 shows the overall 3D architecture. In *DARe,* each planar tier consists of an equal number of ReRAM-based PEs connected by a 3D mesh-based NoC with tree-based multicast.

Finally, the mapping of the GNN weights and adjacency matrix to the ReRAM PEs also play a big role in determining the amount of communication during GNN training. ReRAM-based architectures adopt a pipelined training strategy to avoid repeated writing of weights to the ReRAM crossbars [20]. However, pipelining cannot be implemented with GNNs that operate on one large monolithic input graph [10]. Following [13], we use graph-partitioning to enable pipelining for training GNNs on *DARe*. However, pipelining requires that all neural layers be computed simultaneously.

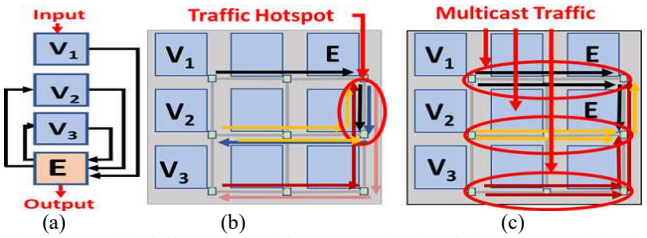


Fig. 4. Simplified illustration of the communication during GNN training in *DARe.* (a) communication pattern in GNN training for a 3-layer GNN, (b) many-to-few communication pattern and resulting traffic hotspot, and (c) Multicast communication between different PEs.

Hence, all the weights and sub-graphs need to be present on-chip to fill up the computation pipeline at a certain instance of time and minimize the stall [20]. The weights and graph adjacency matrices are mapped suitably to PEs to reduce communication. For instance, if two highly communicating neural layers are mapped far from each other, there will be a significant amount of multi-hop communication, which will create performance bottlenecks as shown in Fig. 4(b). As an example, in Fig. 4(b), if layers V_1, V_2, V_3 and E are mapped to PEs physically closer to each other, it is possible to reduce hop count and therefore reduce latency. Furthermore, by mapping the neural layers to PEs appropriately, it is possible to avoid traffic hotspots, including the scenario shown in Fig. 4(b). Hence, a suitable mapping strategy is necessary to further reduce the amount of multi-hop communication and complement the advantages introduced by the Drop-aware control mechanism and multicast in the NoC.

For *DARe*, we employ an efficient Simulated Annealing (SA) based mapping strategy as proposed in [23], as SA can uncover high-quality solutions in a reasonable amount of time. The mapping of weights and the adjacency matrix to the PEs can be seen as a combinatorial optimization problem: Given a total of P PEs, L layers and A adjacency matrices, we need to distribute all the weights of the L layers and A adjacency matrices to P PEs such that the highly communicating layers are mapped to nearby PEs. The optimized mapping enables high-throughput communication, reduces long-range traffic and promotes efficient multicast. Taken together, Drop-aware 3D NoC and optimized mapping policy enable high-performance GNN training on *DARe*.

V. Experimental Results

In this section, we present a comprehensive performance evaluation of the proposed *DARe* architecture and compare it with appropriate baseline solutions.

A. Experimental Setup

The specific embodiment of the *DARe* architecture considered in our performance assessment consists of 36 homogeneous ReRAM-based PEs. The PEs are distributed evenly across four planar tiers and every planar tier is connected to each other using Through Silicon Vias (TSVs). The ReRAM crossbar configuration is chosen based on the storage-power-area trade-offs (shown in Fig. 5). The selected crossbar and tile configurations are shown in Table I [19]. The ReRAM crossbars operate at 10 MHz, which is typical [19] [27]. Each PE has four tiles as shown in Table I, and each tile contains one 16-bit reconfigurable LFSR operating at 1GHz. The default packet structure with no DropLayer has 16 flits, where each flit is a 16-bit fixed-point number generated by ReRAM crossbars. We follow the Garnet packet structure in the NoC [28]. The LFSR bit width is determined

TABLE I. *DARE* PARAMETERS FOR PERFORMANCE EVALUATION

4 planar tiers, 9 cores per tier, 4 tiles per core	
ReRAM Tile (12 IMAs, In-situ Multiply-Accumulate Units)	1 IMA has: 8-ADCs (8-bits), 128x8 DACs (1-bit), 8 crossbars, 128x128 crossbar size, 10MHz, 2-bit resolution, 1 Programmable LFSR (16-bit)

TABLE II. GRAPH DATA STATISTICS

Datasets	**No. of Nodes**	**No. of Edges**	**No. of Edges per Node**	**No. of Features**	**No. of Labels**
PPI	56,944	818,716	~14	50	121
Reddit	232,965	11,606,919	~49	602	41
Amazon2M	2,449,029	61,859,140	~25	100	47

using (1). The LFSR is designed to be reconfigurable (using weighted outputs, e.g., as in [29]) so that different DropLayer probabilities can be implemented. The LFSR is synthesized from a register-transfer level (RTL) design using a TSMC 28 nm CMOS process in Synopsys Design Vision. It adds negligible area and power overheads (less than 1% of an ReRAM PE). The PEs communicate with each other via the 3D Mesh NoC with tree-based multicast. ReRAM arrays always execute instructions in-order, and the instruction latencies are deterministic [19]. Hence, deterministic models are used to evaluate execution time, on-chip traffic, etc. The mapping of GNN weights and adjacency matrices on the tiles are determined offline. We do not discuss the ReRAM execution models in detail for the sake of brevity as it has been elaborated in previous work [20].

The popular graph convolutional network (GCN) algorithm Cluster-GCN from [13] (implemented in TensorFlow) is used as a representative GNN for the performance evaluation. The GCN configuration in our experiments employs the METIS graph partitioning tool [30] to reduce memory overhead and enable pipelined training. This allows us to evaluate GNNs for large-scale graphs, which are otherwise impossible to process in an on-chip environment. Note that, our findings and the proposed architecture are equally applicable to other GNNs relying on the recursive message-passing scheme for neighbor information aggregation. For the evaluation of training GNNs on *DARe*, we choose three popular graph datasets – PPI, Reddit, and Amazon2M (details provided in Table II). The GCN for each dataset consists of four neural layers. As Table II shows, the graph datasets are diverse in nature (in terms of size, partitioning, etc.). This allows for comprehensive performance evaluation of *DARe*.

B. *Crossbar Configuration*

In this sub-section, we first determine the appropriate crossbar configuration for the *DARe* architecture. As discussed earlier, smaller crossbars are preferred for graph computations while larger crossbars are used for DNN

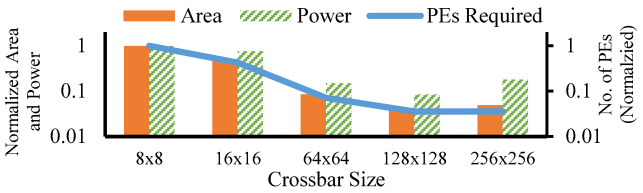


Fig. 5. Storage Efficiency-Power-Area trade-offs for different ReRAM crossbar configurations, normalized w.r.t. 8x8 crossbar configuration

operations. As GNN training involves both types of operations, we must first determine the appropriate crossbar configuration based on the storage efficiency, power, and area trade-offs by varying the crossbar sizes from 8x8 to 256x256. Fig. 5 shows the PE requirement, area and power needed to store one input sub-graph of the Reddit dataset. Similar trends were seen for other datasets. As shown in Fig. 5, larger crossbars (e.g., 128x128) require 28X times fewer PEs than the smaller (8x8) configuration to store the same amount of information. This happens because smaller crossbars are able to store much less information in each crossbar. Hence, more PEs are needed, necessitating more peripheral circuits (ADC, DAC etc.). On the other hand, larger crossbars require fewer PEs, eliminating extra overhead required by peripheral circuits. Note that larger crossbars store more redundant zeros [17]. Despite that, they are more efficient in terms of area and power. However, extremely large crossbars (beyond 128x128) are relatively hard to design and require more area and power. This happens as peripheral circuits for such large crossbars are big. For instance, a 256x256 crossbar requires a 9-bit ADC [19], which is not only difficult to design but is extremely area- and power-hungry, overshadowing any benefit of the larger crossbars (note that ADC/DAC power and area grow with increasing resolution). Thus, from Fig. 5 we see that the most suitable crossbar configuration in terms of power, area and storage efficiency is the 128x128 configuration, even though it stores more zeros compared to smaller crossbar configurations.

C. *Effect of DropLayer on accuracy and traffic*

In this subsection, we explore the effect of DropLayer on GNN accuracy and inter-PE traffic in a manycore system. Fig. 6 shows the accuracy of GNN training on unseen validation data for the (a) PPI, (b) Reddit, and (c) Amazon2M datasets. In practice, the DropLayer probabilities are hyper-parameters determined by the user. In this work, we undertake a grid search to determine the probability of DropLayer that leads to best accuracy. It can be seen that both DropEdge and Dropout help improve the accuracy of the GNN model as expected. In Fig. 6, DropEdge (or Dropout) of 0.1 indicates that 10% of the graph edges (or neural layer weights) are temporarily removed randomly in each epoch, and so on. Increasing the probability of DropEdge (represented as Prob. of DropEdge in Fig. 6 and Fig. 7) initially causes a slight increase in accuracy for all datasets as it reduces over-fitting and over-smoothing [8]. However, increasing the probability of DropEdge excessively

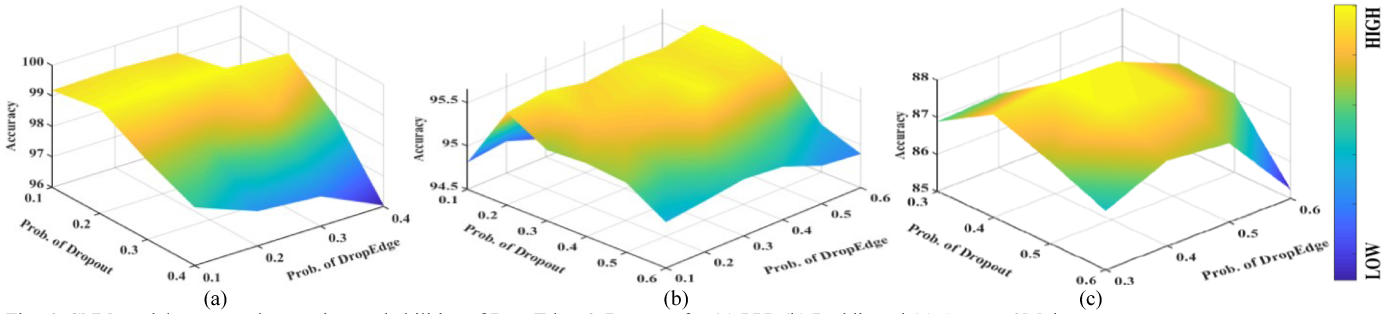


Fig. 6. GNN model accuracy by varying probabilities of DropEdge & Dropout for (a) PPI, (b) Reddit and (c) Amazon2M datasets

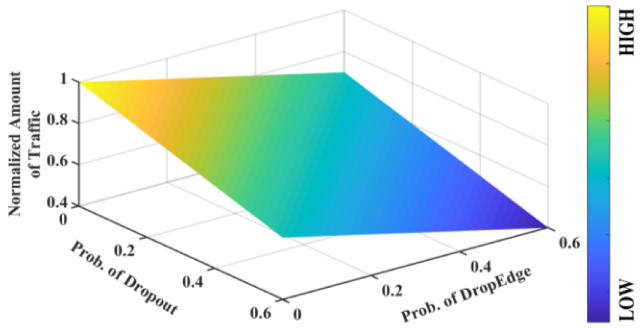


Fig. 7. Effect of varying probabilities of DropEdge & Dropout on total inter-PE traffic during GNN training for Reddit dataset.

(greater than 0.3 for PPI and 0.5 for both Reddit and Amazon2M) causes loss in accuracy. This happens as relatively higher probability of DropEdge causes information loss as multiple graph edges are omitted every epoch during GNN training. Hence, omitting too many edges has an adverse impact on accuracy.

It should also be noted that the probability of DropEdge to achieve best accuracy is dataset dependent. As seen from Table II, PPI dataset has relatively lower degree distribution (number of nodes per graph edge) when compared to Reddit and Amazon2M. As a result, even a relatively lower probability of DropEdge can cause a large number of edges being omitted per node, which leads to a significant accuracy loss in case of PPI. Reddit and Amazon2M both have relatively higher number of edges per node. Hence, we can achieve higher accuracy even with greater probability of DropEdge for both Reddit and Amazon2M datasets.

Similar to DropEdge, increasing the probability of Dropout initially improves accuracy due to reduced over-fitting. However, as is the case for DropEdge, the probability of Dropout for the best possible accuracy is different for each dataset. The Dropout probability for Reddit and Amazon2M can be as high as 0.5, whereas for PPI the accuracy falls significantly with a Dropout probability higher than 0.3. The probability of Dropout for a dataset depends on the number of features used for training. In case of PPI, each node has fewer features but a larger number of labels to classify from. With high Dropout, the classification becomes even harder with the lack of information as more features are omitted due to Dropout. In contrast, Reddit and Amazon2M both have more features for each node and a smaller number of labels. Hence, it allows for relatively higher probability of Dropout as the GNN has many features to work with even at relatively larger probability of Dropout. Overall, Reddit and Amazon2M have similar accuracy trend with varying probability of DropEdge and Dropout as shown in Fig. 6(b) and Fig.6(c). On the other hand, PPI (Fig. 6(a)) allows for a relatively smaller probability of DropEdge and Dropout without loss of accuracy.

Next, we investigate how varying DropEdge and Dropout affects communication during GNN training. As mentioned in Section III, DropEdge and Dropout reduce on-chip traffic during GNN training. Fig. 7 shows inter-PE traffic during GNN training for the Reddit dataset as an example with different probabilities of DropEdge and Dropout. As shown in Fig. 7, the amount of traffic monotonically decreases as the probabilities of DropEdge and Dropout are increased. As an example, the amount of traffic is highest when the probabilities of both DropEdge and Dropout are zero, while the traffic decreases as the probabilities of DropEdge and Dropout are reduced. For all further experiments, we choose the value of Dropout and DropLayer for each dataset that achieves less than 1% drop in accuracy (relative to the best-case accuracy observed in our experiments) and leads to the highest reduction in traffic. Interestingly, we found that this condition is satisfied when the probabilities of both DropEdge and Dropout are the same. Hence, for all further analysis, we use DropLayer probability of 0.3, 0.5 and 0.5 for PPI, Reddit and Amazon dataset respectively. Recall that we refer to DropEdge and Dropout collectively as DropLayer.

D. Performance Evaluation of the Drop-aware NoC

In this subsection, we evaluate the performance of the Drop-aware 3D NoC, which serves as the communication backbone for *DARe*. As GNN training on *DARe* is implemented in a pipelined fashion, we examine the computation and communication pipeline stage delays. The communication delay includes the timing overhead introduced by the LFSR-based control mechanism. It is well-known that the slowest stage in a pipeline determines the overall stage delay. In turn, the overall stage delay governs the latency of the system. Hence, to maximize the overall performance, our aim is to balance the pipeline stage delays. In order to establish the efficiency of the 3D Drop-aware NoC used in *DARe*, we consider four ReRAM-based manycore architectures with different NoC configurations as follows: (a) 2D Mesh without DropLayer (2D NoDrop), (b) 2D Mesh with DropLayer (2D Drop), (c) 3D Mesh without DropLayer (3D NoDrop) and finally, (d) proposed Drop-aware 3D NoC architecture used in *DARe*. Note that all the NoCs adopt tree-based multicast to handle the GNN traffic.

Each GNN layer involves both computation (MAC operations) and communication (the data of one layer must be sent to the next layer as shown in Fig. 1 and Fig. 4(a)). Hence, the overall stage delay for GNN training is determined by the worst-case delay among the computation and communication stages. Fig. 8 shows the worst-case computation and communication delays when a GNN is trained on ReRAM-based architectures with 2D NoDrop, 2D Drop, 3D NoDrop NoCs, and the *DARe* architecture. The computation delay is the maximum time needed to finish all the MAC operations of any given layer. Similarly, the communication delay represents the maximum time needed to communicate all required data from one neural layer to another. It should be noted that there is no change in computation time for different NoC architectures because the number of ReRAM-based PEs for computation does not vary with the choice of the NoC.

Fig. 8 compares the computation, communication, and overall stage delays between 2D NoDrop, 2D Drop, 3D

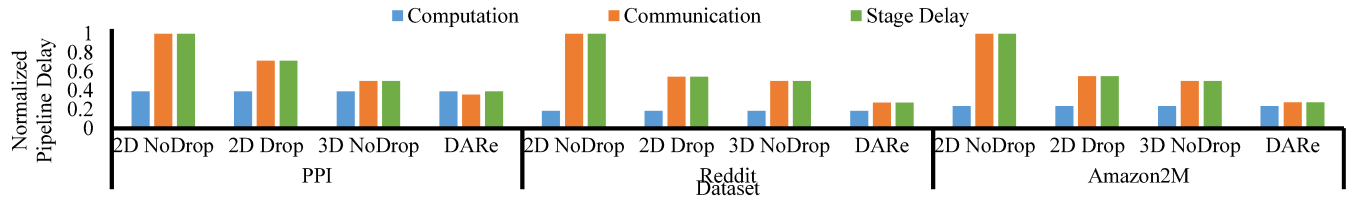


Fig. 8. Computation and communication delay for 2D NoDrop, 2D Drop, 3D NoDrop and *DARe*; all delays are normalized with respect to the communication delay of 2D NoDrop.

NoDrop and *DARe*. Here, we consider 2D NoDrop as our baseline. In order to utilize all (or most) of the available ReRAM resources, we replicate the parameters of the GNN layers (such as weights) on different sets of ReRAM tiles following [20]. Replicating the weights of a neural layer results in faster execution as more data can be processed in parallel [20]. However, the replication of weights requires additional ReRAM resources. Hence, the number of times we can replicate the GNN parameters is limited by the total number of available ReRAM crossbars. In Fig. 8, we show the computation delay of each architecture with the maximum amount of weight duplication possible for all the datasets. This duplication ensures maximum ReRAM utilization and results in the best possible computation delay.

Without DropLayer, GNN training is bottlenecked by the heavy data traffic. As a result, the communication delay dominates the overall performance (as seen for 2D NoDrop in Fig. 8). The slowest stage determines the overall execution time in a pipelined implementation; therefore, the high communication latency will bottleneck performance. Even though the computation latency is improved by using ReRAMs, the overall latency will be limited by the time to communicate. This will affect the full-system execution time as we show later. As mentioned in Section III and IV, implementing DropLayer during GNN training reduces the inter-PE traffic (as shown in Fig. 6). Implementing DropLayer in the 2D NoC (2D Drop) reduces the communication delay by 40% on an average compared to 2D NoDrop. However, despite the reduction in latency, communication still remains the bottleneck in both of the 2D architectures. The 3D NoDrop NoC reduces the communication delay by 9% compared to the 2D Drop architecture. In spite of not incorporating DropLayer, the 3D NoC (3D NoDrop) achieves better overall performance compared to the 2D Drop counterpart. This happens due to the inherently lower average hop count of 3D NoC compared to its 2D counterpart. Moreover, as shown in Fig. 8, in case of the architectures with 2D NoDrop, 2D Drop and 3D NoDrop NoC, the stage delay is always bottlenecked by the communication. As a result, the stage delay is the same as the communication delay.

Implementing DropLayer in the 3D NoC enables reduction in traffic and communication stage latency noticeably. Compared to the 2D NoDrop baseline, *DARe* improves communication delay by 65%, 73% and 72% for PPI, Reddit and Amazon2M datasets, respectively. The reduction in communication stage delay compared to the NoDrop NoC architectures can be attributed to the traffic reduction enabled by DropLayer. The variable packet sizing-based control scheme creates packets with lower number of flits (some flits are omitted), which reduces the overall time needed to communicate all the packets. By combining the low hop count feature of 3D NoC and the traffic reduction using DropLayer, we reduce the communication stage delay significantly in *DARe*. For PPI, the overall stage delay is bottlenecked by the computation as it cannot be further accelerated by duplicating weights with the available ReRAM resources considered in this work. As a result, the overall improvement for PPI is governed by the computational stage delay. On the other hand, the communication delay improves significantly for Reddit and Amazon2M, but it still remains the bottleneck due to the accelerated computation. We can conclude that for all the datasets, *DARe* achieves significant performance improvement with respect to the baseline 2D NoDrop architecture.

E. *Full System Performance Evaluation*

Next, we undertake a full-system performance evaluation of *DARe*. We consider the baseline architecture (2D NoDrop), a state-of-the-art ReRAM-based GNN accelerator, ReGraphX [24] and a conventional GPU (Nvidia V100 in this case) while benchmarking the performance of *DARe*. ReGraphX is also a 3D NoC-enabled ReRAM-based manycore architecture. However, it does not incorporate any DropLayer feature. Fig. 9(a) and Fig. 9(b) show the execution time and energy consumption (normalized with respect to the GPU) for GNN training, respectively, on *DARe*, 2D NoDrop and ReGraphX. *DARe* achieves 5.6X, 3.2X and 1.9X lower overall execution time on an average compared to GPU, 2D NoDrop and ReGraphX respectively. The lower execution time achieved by *DARe* compared to 2D NoDrop and ReGraphX is attributed to the Drop-aware 3D NoC. The Drop-aware 3D NoC in *DARe* reduces the inter-PE traffic by enabling the DropLayer feature and thus improves the overall performance by reducing communication latency. Note that all the ReRAM-based architectures (even 2D No Drop) outperform the GPU. The inherent advantages of ReRAM-based architectures in implementing high-throughput MAC is the reason behind this. Fig. 9(b) shows that the *DARe* architecture on an average consumes 23X, 3.3X and 2.5X less energy than conventional GPUs, 2D NoDrop and ReGraphX, respectively. This is also enabled by the lower latency and higher throughput of the Drop-aware 3D NoC in *DARe*. Overall, *DARe* is up to 6.7X faster while consuming up to 30X less energy for GNN training than the traditional GPU-based implementation.

VI. CONCLUSION

Graph neural networks (GNNs) have a multitude of real-world applications such as social media, drug discovery, and recommendation systems. Software techniques such as – DropEdge and Dropout (together referred as DropLayer in this work) are regularization techniques that help improve GNN accuracy. When incorporated in a manycore architecture, DropLayer reduces the inter-PE traffic. However, this traffic reduction is dynamic in nature due to randomness of the DropLayer mechanism. In this work, We have presented a Drop-aware ReRAM-based manycore architecture for training GNNs called "*DARe*". The proposed *DARe* architecture is able to achieve high performance by combining the efficient MAC operations of ReRAM-based PEs and a high-throughput, low-latency Drop-aware 3D NoC. The Drop-aware 3D NoC incorporates a control mechanism that emulates the effects of DropLayer in the hardware to reduce heavy data traffic inherent in GNN training. The control mechanism enables handshaking between the source and destination PEs in presence of the dynamically varying traffic. *DARe* outperforms conventional GPUs by up to 6.7X in terms of execution time and is up to 30X more energy efficient.

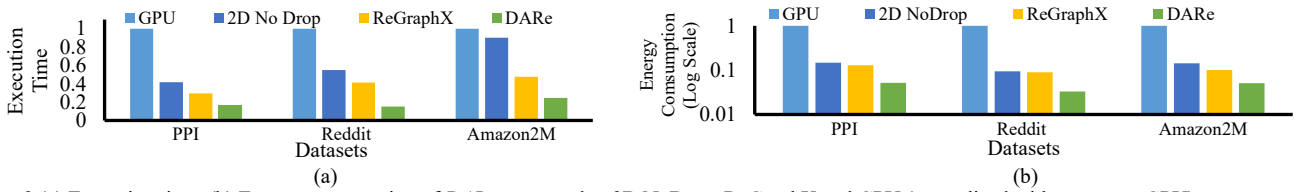


Fig. 9 (a) Execution time, (b) Energy consumption of *DARe* compared to 2D NoDrop, ReGraphX and GPU (normalized with respect to GPU)

REFERENCES

[1] R. Ying et al., "Graph Convolutional Neural Networks for Web-Scale Recommender Systems," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, London, 2018.

[2] F. Ding, "Graph Neural Networks for Quantum Chemistry," July 2019. [Online]. Available: https://github.com/ifding/graph-neural-networks.

[3] W. Fan et al., "Graph Neural Networks for Social Recommendation," in *The World Wide Web Conference*, San Fransisco, CA, 2019.

[4] J. Zhou et al., "Graph Neural Networks: A Review of Methods," in *arXiv:1812.08434*, 2018.

[5] M. Zhang et al., "An End-to-End Deep Learning Architecture for Graph Classification," in *AAAI*, New Orleans, LA, 2018.

[6] D. Fujiki, S. Mahlke and R. Das, "In-memory Data Flow Processor," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Portland, OR, 2017.

[7] K. Kiningham, C. Re and P. Levis, "GRIP: A Graph Neural Network Accelerator Architecture," in *arXiv eprint 2007.13828*, 2020.

[8] Y. Rong, W. Huang, T. Xu and J. Huang, "DropEdge: Towards Deep Graph Convolutional Networks on Node Classification," in *International Conference on Learning Representations (ICLR)*, 2020.

[9] N. Srivastava et al., "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research,* vol. 15, no. 1, pp. 1929-1958, 2014.

[10] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *International Conference on Learning Representations (ICLR)*, Toulon, 2017.

[11] T. Geng et al, "AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020.

[12] B. Feero and P. P. Pande, "Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation," *IEEE Transactions on Computers,* vol. 58, no. 1, pp. 32-45, 2009.

[13] W. L. Chiang et al., "Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Anchorage, AK, 2019.

[14] A. Auten, M. Tomei and R. Kumar, "Hardware Acceleration of Graph Neural Networks," in *IEEE/ACM Design Automation Conference (DAC)*, 2020.

[15] M. Yan et al., "HyGCN: A GCN Accelerator with Hybrid Architecture," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[16] L. Song et al, "GraphR: Accelerating Graph Processing Using ReRAM," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Vienna, 2018.

[17] G. Dai et al, "GraphSAR: a sparsity-aware processing-in-memory architecture for large-scale graph processing on ReRAMs," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, New York, NY, 2019.

[18] L. Zheng et al, "Spara: An Energy-Efficient ReRAM-Based Accelerator for Sparse Graph Analytics Applications," in *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, New Orleans, LA, 2020.

[19] A. Shafiee et al, "ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars.," in *International Symposium on Computer Architecture (ISCA)*, 2016.

[20] L. Song et al, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Austin, TX, 2017.

[21] P. Chi et al., "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *International Symposium on Computer Architecture (ISCA)*, South Korea, 2016.

[22] Z. He, J. Lin, R. Ewetz, J. Yuan and D. Fan, "Noise Injection Adaption: End-to-End ReRAM Crossbar Non-ideal Effect Adaption for Neural Network Mapping," in *IEEE/ACM Design Automation Conference (DAC)*, 2019.

[23] B. K. Joardar et al, "AccuReD: High Accuracy Training of CNNs on ReRAM/GPU Heterogeneous 3-D Architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 40, no. 5, pp. 971-984, 2020.

[24] A. I. Arka et al., "ReGraphX: NoC-enabled 3D Heterogeneous ReRAM Architecture for Training Graph Neural Networks," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2021.

[25] W. T. Flynn, D. A. Shedivy and K. M. Valk, "Using variable length packets to embed extra network control information". USA Patent US8514885B2, 2010 .

[26] J. Duato, S. Yalamanchil and N. Lionel, Interconnection Networks: An Engineering Approach, vol. 54, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 2002, pp. 1025-1040.

[27] Y. Long, T. Na and S. Mukhopadhyay, "ReRAM-Based Processing-in-Memory Architecture for Recurrent Neural Network Acceleration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 26, no. 12, pp. 2781-2794, 2018.

[28] N. Agarwal, T. Krishna, L. Peh and N. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Proc. of the ISPASS*, Boston, MA, 2009.

[29] M. Bushnell and V. Agrawal, Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits, Springer US, 2006.

[30] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing,* vol. 20, no. 6, p. 359–392, 1998.