Heterogeneous Manycore Architectures Enabled by Processing-in-Memory for Deep Learning: From CNNs to GNNs

(ICCAD Special Session Paper)

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

*Department of ECE
Duke University
Durham, NC, US
{bireshkumar.joardar, hai.li, krish}@duke.edu

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

2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 978-1-6654-4507-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCAD51958.2021.9643559

Abstract—**Resistive random-access memory (ReRAM)-based processing-in-memory (PIM) architectures have recently become a popular architectural choice for deep-learning applications. ReRAM-based architectures can accelerate inferencing and training of deep learning algorithms and are more energy efficient compared to traditional GPUs. However, these architectures have various limitations that affect the model accuracy and performance. Moreover, the choice of the deep-learning application also imposes new design challenges that must be addressed to achieve high performance. In this paper, we present the advantages and challenges associated with ReRAM-based PIM architectures by considering Convolutional Neural Networks (CNNs) and Graph Neural Networks (GNNs) as important application domains. We also outline methods that can be used to address these challenges.**

Keywords—Deep Learning, Processing-in-memory, ReRAM, 3D

I. Introduction

Deep learning (DL) has revolutionized application domains such as image processing, autonomous driving, and remote healthcare. However, DL is both compute- and data-intensive in nature. As a result, a major portion of the computations (e.g., training for DL) have remained traditionally confined to datacenters. CPU and GPU-based manycore architectures are the most common choice of hardware for deep learning applications. However, general purpose CPU- and GPU-based systems are not customized for deep learning and often suffer from: (a) high area and power overheads, and (b) memory bottleneck. These limitations of traditional manycore systems have resulted in many studies aimed at developing the next generation of DL accelerators.

Resistive Random-Access Memory (ReRAM)-based Processing-in-Memory (PIM) architecture is one of the most promising technologies in this direction. ReRAM crossbars can efficiently perform matrix-vector multiplications, which form the backbone of most deep learning algorithms [1]. Prior work, e.g., Pipelayer [1] and AccuReD [2], has shown that ReRAM-based architectures can outperform GPUs for training Convolutional Neural Networks (CNNs) while consuming considerably less energy. ReRAM-based PIM architectures have also been used to accelerate other DL models, such as Graph Neural Networks [3], Recurrent Neural Networks [4] and Generative Adversarial Networks [5]. In addition, ReRAM-based systems are more area-efficient compared to their GPU-based counterparts and do not require expensive off-chip memory access due to their "in-memory" nature of computation [6] [7].

Despite the above-mentioned advantages, ReRAM-based PIM architectures have several shortcomings, which can lead to sub-optimal power-performance-accuracy trade-offs. In addition, the choice of deep learning algorithm also imposes new design challenges that must be solved. For instance, CNN training often involves the use of Batch Normalization (BN) and SoftMax layers. Both these layers are precision critical and must be implemented using high precision arithmetic [8] [9]. Moreover, the backward phase during CNN training is also sensitive to precision [10]. ReRAMs have limited representation capability (16-bit fixed point) compared to conventional GPUs (which use 32-bit floating-point precision). This aggressive reduction of precision can make the training process unstable, thereby compromising prediction accuracy [2]. These limitations present a barrier towards the widespread adoption of ReRAM-based architectures for CNN training. Hence, it is of utmost importance to address these shortcomings of current ReRAM-based implementations.

Similarly, other DL techniques such as GNNs introduce new sets of design challenges that cannot be solved using conventional ReRAM-based architectures. For instance, GNNs involve sparse matrices and heavy data exchange due to message-passing operations to accumulate neighbor information in a recursive manner [11]. Storing sparse matrices on ReRAMs is challenging as they contain many zeros. Computations with zeros are redundant and hence, we must avoid storing zeros on ReRAM cells. However, the inherent crossbar structure of ReRAMs is not suited for storing irregular sparse matrices [12] [13]. Similarly, the iterative message passing in GNNs gives rise to significant data exchange among the ReRAM tiles, which limits the overall achievable performance [3]. The data exchange during GNN training exhibits many-to-one, and multicast patterns, both of which can cause performance bottleneck without a suitable communication backbone [14].

Apart from the unique design requirements imposed by the choice of DL techniques, ReRAMs also suffer from numerous nonidealities. ReRAMs are susceptible to noise, "hard faults" such as stuck-at-faults (SAFs), process variations, and wear out

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.

due to limited write endurance [15] [16]. These non-idealities have a deleterious impact on implementing large-scale deep learning algorithms on ReRAM-based accelerators [15] [17]. These faults arise due to the immature fabrication process and limited endurance of ReRAMs. Moreover, the effect of these faults can either be static (present at $t = 0^-$) or dynamic (appears during use, i.e., at $t > 0$) in nature. On the other hand, noise is random and can be introduced due to high temperature, write variations, etc. Existing work has confirmed that training and inferencing in the presence of these non-idealities lead to unacceptably low accuracy of the CNN models [15] [17].

In this paper, we first present the state-of-the-art in ReRAM-based manycore PIM architectures. Next, we discuss the shortcomings of the existing architectural solutions. We elaborate the specific challenges introduced by different deep learning algorithms, such as precision sensitivity of CNNs and the communication intensive nature of GNNs. Finally, we present ReRAM-based heterogeneous manycore PIM designs as a solution to address these shortcomings.

II. ReRAM-based Accelerators and Challenges

In this section, we present the salient features of ReRAM-based PIM architectures. Next, we demonstrate how ReRAMs accelerate deep learning training/inferencing using CNNs as a representative example. Finally, we outline the shortcomings of existing architectures that must be addressed.

A. *ReRAM crossbars*

ReRAMs are typically implemented with a crossbar architecture [18] [19]. As shown in Fig. 1, every bitline is connected to every word line via resistive memory cells. Each ReRAM cell stores data as resistance (or conductance) values. The crossbar operates following Ohm's law and Kirchhoff's current laws. The total current emerging from the *j*th bitline I_j, represents a dot product operation and can be expressed mathematically as:

$$I_j = \sum_{i=1}^{N} G_{ij} * V_i \qquad (1)$$

Here, G_{ij} is the conductance of the ReRAM cell connecting the word line i and the bit line j and V_i is the input voltage to the word line i. The ReRAM crossbar shown in Fig. 1 can perform multiple multiplication and addition operations in *O(1)* time. An $M \times N$ crossbar array performs dot products on M-entry vectors for N different neurons in a single step. A sample-and-hold (S&H) circuit receives the bitline current and feeds it to a shared ADC unit (as shown in Fig.1). This conversion of analog currents to digital values is necessary before communicating the results to other digital units. Similarly, a DAC unit converts digital input values into appropriate voltage levels that are applied to each row. Many such crossbars form a ReRAM tile (core). Each tile is composed of eDRAM buffers to store input

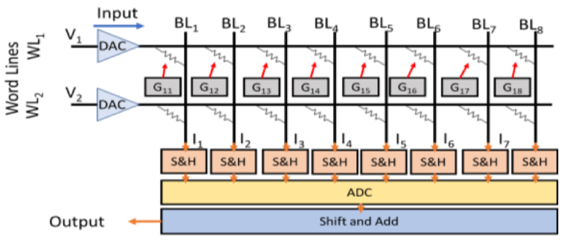


Fig. 1: Illustration of matrix dot product in ReRAM crossbar [6]

values, several in-situ multiply-accumulate (IMA) units, and output registers to aggregate results, all connected with a shared bus. The tile also has shift-and-add, sigmoid, and max-pool units. Many such tiles are connected using a network-on-chip (NoC), such as mesh [10] or H-tree [20], to create a ReRAM-based manycore PIM system.

B. *Accelerating CNNs on ReRAMs*

CNN computations predominantly involve two types of layers: convolution (Conv) layer and fully connected (FC) layer. In a convolution layer, a set of kernels (also known as weights) are multiplied with the inputs (which are also the outputs from the previous layer *l*) to generate intermediate data (also known as activations) for next layer (layer *l* + *1*). Unlike CNN inferencing, training involves an additional backward phase where the data moves backwards (i.e., from layer *l*+*1* to layer *l)* to update weights based on the errors. The computations of the forward and backward phases of the Conv and FC layers can be summarized as follows:

$$Forward: \quad y_{l+1} = w_l * y_l + b_l \qquad (2)$$

$$Backward: \begin{cases} \partial_{l-1} = (w_l)^t * \partial_l & (3) \\ \nabla w_l = y_{l-1} * (\partial_l)^t & (4) \\ \nabla b_l = \partial_l & (5) \end{cases}$$

Here, y_l, w_l, b_l, ∂_l, and ∇w_l represents the activations, weights, bias, error gradients and the weight gradients of layer l respectively. Note that, other layers such as pooling, ReLU, Batch Normalization and SoftMax are also present in CNN training and inferencing. However, Conv and FC layers represent the most compute intensive layers, which are accelerated using ReRAM-based engines. Hence, we do not show these other layers in Eqns. 2-5. From Eqns. 2-5, we note that both the forward and backward phases of Conv and FC layers are primarily matrix multiplications, which can be implemented using the ReRAM crossbars.

Fig. 2 shows an illustration of how the CNN weights (for a convolution layer in forward pass) are mapped on the ReRAM crossbars. Here, we assume a convolution layer with $K \times K \times IC \times OC$ weights (K: dimension of filter, IC: number of input channels, and OC: number of output channels). The CNN layer considered in Fig. 2 operates on an $I \times I$ input to generate an output of size $O \times O$. The weights are stored on ReRAMs in 16-bit fixed-point format [20]. However, all 16 bits (2^{16} states) are not mapped on a single ReRAM cell mainly due to area (required ADC size will be astronomical [20]) and noise concerns [15]. As shown in Fig. 2, the 16 bits of each weight are distributed across multiple arrays/cells. With a 4-bit resolution of each ReRAM cell, a 16-bit weight would require four separate ReRAM cells for complete representation [1]. Similarly, if each

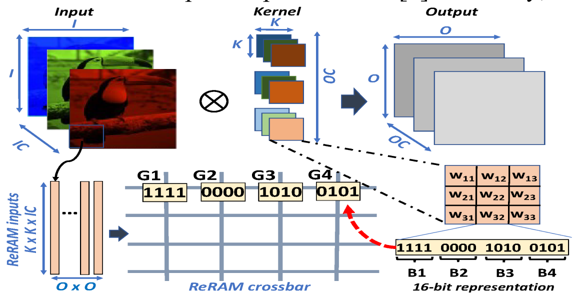


Fig. 2: Illustration of CNN computations on ReRAM crossbar [2]

ReRAM cell has a 2-bit resolution, we will require eight different ReRAM cells to represent the same 16-bit weight value [20]. The input to ReRAM crossbar is generated by sliding a window of dimension $K \times K \times IC$ across the $I \times I$ input matrix, which is then fed to the ReRAM array in a sequential manner. Overall, $O \times O$ inputs need to be processed to complete the convolution layer (forward phase). Importantly, other layers, e.g., fully connected, as well as the backward phase of training also involve similar operations. Hence, they can be similarly mapped to ReRAM crossbars with minimal adjustments [1].

C. Limitations of existing ReRAM-based architectures

Despite the advantages of an ReRAM-based accelerator for deep learning algorithms, there are numerous challenges that must be addressed. In this sub-section, we enumerate and explain some of these challenges.

Lack of BN support: Exploding/Vanishing gradients is one of the primary challenges associated with training deep CNNs (i.e., CNNs with many Conv/FC layers). BN layers are commonly used to address this problem. However, BN layers are precision sensitive and involves complex math operations (such as division and square root); both operations are difficult to implement using ReRAMs. As a result, training deep CNNs is challenging using sole-ReRAM based architectures. Specialized initialization schemes such as Xavier [21] or Kaiming [22] are alternative ways to train deep CNNs in the absence of BN. By carefully setting the weights, these techniques can prevent exploding/vanishing gradients. However, these initialization schemes require careful hyper-parameter tuning (i.e., expert domain knowledge) and yet, do not work all the time. To demonstrate the sensitivity of these methods to the choice of hyper-parameters, we consider two cases: (a) All-BN: BN layers are placed after every Conv layer as is usually done in traditional CNNs, and (b) No-BN: BN layers are not used at all, and varied five hyper-parameters: learning-rate (LR), number of epochs, LR schedule, batch size, and the initialization scheme (Xavier and Kaiming only; other initialization methods resulted in significantly low training accuracy in the absence of BN). The results are shown in Fig. 3. Overall, CNN training with 150 different hyper-parameter settings were performed and the best accuracy achieved at the end of each training instance is noted. Fig. 3 shows the range of observed accuracy (represented by the blue line, whose ends indicate the minimum and maximum accuracy; the red line represents the average accuracy) considering all 150 experiments with VGG-11/16/19 and ResNet-18.

Fig. 3 clearly shows that it is possible to train CNNs in the absence of BN sometimes, which is in line with previous findings [21] [22]. However, it is not reliable and fails to train most of the time. For instance, the average accuracy of No-BN with Xavier/Kaiming initialization considering all 150 instances of training is a mere 57.8% for VGG-19. This happens as multiple combinations of hyper-parameters (among the 150 chosen here), either failed to train or resulted in unacceptable accuracies. This is problematic as an ML-practitioner (or user) will have to repeatedly train a CNN to find out the valid hyper-parameter combination(s) for a successful training. This process can be time consuming, particularly for deep CNNs and larger datasets, which require more time to train. On the other hand, CNN training with BN is more robust to the choice of hyper-parameters and is effective in all cases considered here. Unlike No-BN, All-BN achieves an average accuracy (considering all 150 All-BN training instances) of 85.1% for VGG-19 indicating that all these 150 experiments succeeded. Hence, hardware support for BN is important to train deep CNNs and should be incorporated in ReRAM-based architectures.

Low precision computations: ReRAM-based architectures typically utilize 16-bit fixed point precision as opposed to 32-bit floating point precision in GPUs. However, CNNs often fail to train or reach unacceptable accuracy when trained using low precision. This happens as the backward phase of training is precision sensitive. Basic rounding schemes that are used to convert higher precision data to lower precision representation often lose important information (round-off error). This results in erroneous gradients, which get accumulated after each weight update. Eventually, the training becomes unstable and reaches unacceptable accuracy. This problem must be addressed for successful training and inferencing.

Fig. 4 shows the accuracy after 50 epochs of training for VGG-19. Here, we consider two different cases: 1) GPU-based training with 32-bit floating-point scheme (GPU); 2) ReRAM-based training (16-bit fixed point) with the default rounding scheme in Pytorch. Normalization layers are used in both cases to rule out exploding/vanishing gradients scenario. From Fig. 4, we note that even with Normalization, VGG-19 fails to train successfully on ReRAM-based architectures. VGG-11 and ResNet-18 exhibit similar behavior as VGG-19 (i.e., failure to train meaningfully) due to low precision. However, smaller CNNs, such as LeNet, train successfully and only experience a slight drop in model accuracy despite low precision. From Fig. 4, we can conclude that for training deep CNNs on ReRAM-based architectures, we must address the problem of accuracy loss due to low precision.

Inter-tile communication: PIM architectures reduce the amount of off-chip data communication that happens between processor and memory in a typical Von-Neumann architecture. Hence, it is often assumed that communication is not an issue for PIM-based architectures. However, inter-tile communication in PIM can be significant for some deep learning applications. For instance, GNN computation requires a neighborhood aggregation operation, where each node aggregates the features of its *k-hop* neighbors to learn node representations [11]. This in turn gives rise to repeated message passing that can lead to high

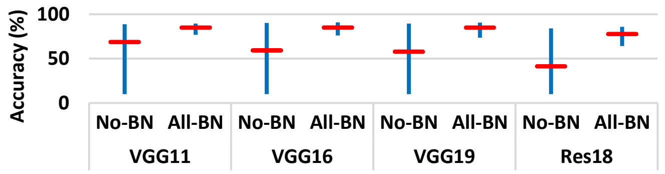


Fig. 3: Accuracy of No-BN and All-BN CNNs trained with various hyperparameter settings. [27]

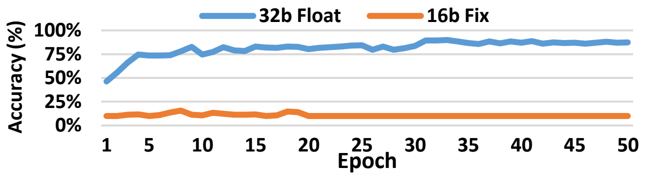


Fig. 4: Accuracy of VGG19 trained with 32-bit floating point (32b Float) and 16-bit fixed point (16b fixed) representations. [2]

volume of data exchange. Traffic associated with GNN training exhibits many-to-one, multicast, and long-range characteristics (explained later in Section IV). This can lead to performance bottleneck.

Traditional planar architectures with mesh network-on-chip (NoC) are not suited for such traffic patterns. The heavy many-to-one and multicast traffic, as well as the large physical separation between tiles, impose a significant amount of long-range communication requirement in planar architectures. As a result, the communication backbone quickly becomes a performance bottleneck. In addition, the inherent multi-hop nature of a planar mesh NoC leads to higher communication latency [23], which is not desirable for training GNNs. Hence, an appropriate NoC design is necessary for ReRAM-based PIM architectures, despite the "in memory" nature of computing.

Storing sparse data: GNN computation involves adjacency matrices of input graph data, which are sparse and contain large number of zeros. Zeros are redundant in computations and must be omitted to reduce hardware and energy requirements. Hence, reduction in zero storage is key to improve GNN computation using ReRAM-based architectures. This is commonly achieved using smaller ReRAM crossbars (e.g., 4×4, 8×8 [13]); these crossbars can store sparse matrices more efficiently. To store sparse matrices, a non-overlapping sliding window operation is performed over the sparse adjacency matrix to generate $N \times N$ segments. Each of these segments is then mapped on to distinct $N \times N$ shaped ReRAM crossbars. This process is illustrated in Fig. 5. Fig. 5(a) and Fig. 5(b) show an input graph and its corresponding adjacency matrix (which is sparse), respectively. Any $N \times N$ segment with N^2 zero-entries (referred as 'invalid segment' in Fig. 5(b) in red) is discarded [12]. The remaining segments that include at least one graph edge (referred as 'valid segment' in Fig. 5(b) in green) are stored on ReRAM cells. This technique reduces the number of zeros that need to be stored on-chip. For instance, the adjacency matrix in Fig. 5(b) has fourteen valid segments and two invalid segments; the invalid segments can be safely discarded, which reduces redundant matrix operations involving zeros. However, as we show later, smaller crossbars are inefficient in terms of the full-system area and power requirements. Smaller crossbars provide lower storage density (number of bits per unit area) while dissipating more power. Hence, the traditional method of using smaller crossbars for storing sparse matrices has high power and area overhead despite storing fewer zeros. A suitable GNN accelerator must address this problem.

Reliability of ReRAMs: The ReRAM fabrication process is not as mature as conventional CMOS fabrication. As a result, ReRAMs are prone to many types of hardware faults and noise. For instance, hard faults prevent the resistance of a ReRAM cell from being updated, resulting in write failures. Fig. 6 shows the different types of hard faults that can happen in ReRAM cells. As shown in Fig. 6, hard faults can be further classified into static and dynamic faults based on when the fault appears for the first time. Static faults appear before use, i.e., at $t = 0^-$ and are caused by manufacturing defects such as short defects, over-forming defects or reset failure. Even if a crossbar passes manufacturing test, faults can appear over time as the crossbar is utilized (at $t > 0$) [24]. These faults, referred as dynamic faults in Fig. 6, can be attributed to the limited write endurance of ReRAMs or due to the application of multiple consecutive write-0 (or write-1) pulses. Dynamic faults can be further sub-divided into permanent and limited-duration faults depending on how long the associated fault persists.

Moreover, ReRAM cells also suffer from different types of noise, such as thermal noise, shot noise, etc. Unlike hard faults, where the cell resistance is stuck at either zero or one, noise is random and is therefore difficult to address. From Fig. 1, we know that data is stored on ReRAM cells as resistance (or conductance) values [25]. However, at higher on-chip temperature, thermal noise causes a shift in resistance (and thereby the stored value). Without appropriate measures, this will lead to erroneous interpretation of stored data, which in turn affects prediction accuracy [25]. In addition, higher ReRAM temperature reduces noise margin. As a result, the computed outputs are more susceptible to errors due to thermal noise (and other nonidealities) [15]. This presents a significant barrier toward scalability in terms of both the size of CNN that can be trained using ReRAMs and the operating frequency.

III. ReRAM-BASED PIM FOR CNNS

In this section, we present a heterogeneous architecture to address the above mentioned shortcomings of sole ReRAM-based PIMs for CNN training. Fig. 7 shows the heterogeneous 3D architecture that consists of both ReRAMs and GPUs [26]. Some of the distinguishing features of this architecture are as follows: (i) It can implement both CNN training and inference, (ii) As shown in Fig. 7(a), it consists of both ReRAM and GPU layers. The GPU layer(s) provide a full-precision computing platform necessary for Normalization support, (iii) There are multiple layers of both ReRAM and GPU tiles stacked vertically using 3D integration. Having multiple layers of ReRAMs and GPUs provide higher computational capability that can support the training of deep CNNs, (iv) The ReRAM tiles include additional peripheral circuits to implement stochastic rounding [Fig. 7(b)], which is necessary for training CNNs at low precision, (v) Each ReRAM tile has dedicated reference cells to track any change in output current due to temperature [Fig. 7(c)], and (vi) The ReRAM and GPU tiles are connected by an efficient NoC for high-performance CNN training. We discuss the salient features of this architecture in more detail next.

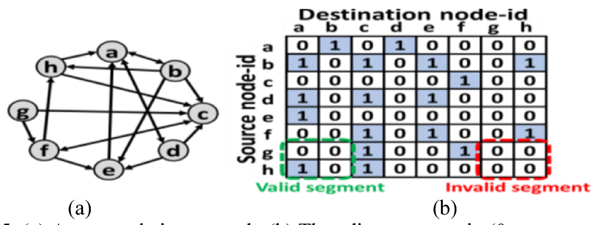
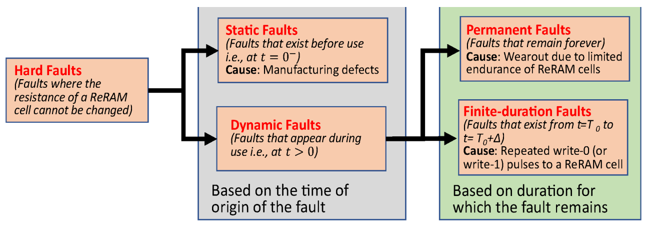


Fig. 5. (a) An example input graph, (b) The adjacency matrix (0 means no edge and 1 means presence of an edge) [12]


Hard Faults
(Faults where the resistance of a ReRAM cell cannot be changed)
Static Faults
(Faults that exist before use i.e., at $t = 0^-$)
Cause: Manufacturing defects
Dynamic Faults
(Faults that appear during use i.e., at $t > 0$)
Based on the time of origin of the fault
Permanent Faults
(Faults that remain forever)
Cause: Wearout due to limited endurance of ReRAM cells
Finite-duration Faults
(Faults that exist from $t=T_0$ to $t=T_0+\Delta$)
Cause: Repeated write-0 (or write-1) pulses to a ReRAM cell
Based on duration for which the fault remains
Fig. 6: Taxonomy of the different faults considered in this work [31]

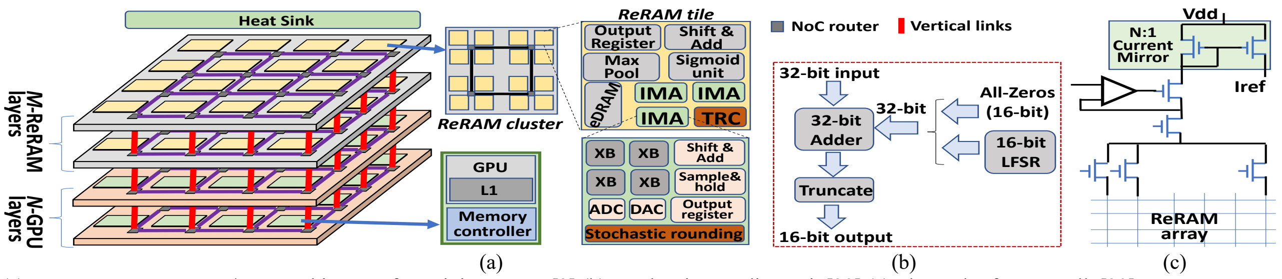


Fig. 7: (a)Heterogeneous ReRAM/GPU architecture for training CNNs [2] (b) Stochastic rounding unit [29] (c) Thermal reference cell [30]

Addressing lack of BN: As mentioned earlier, BN layers require high precision computing support due to its precision critical nature and it also involves more complex arithmetic operations. As a result, BN layers are difficult to implement using ReRAMs only. The GPUs can be used to implement the BN layers. GPUs provide a full-precision computing platform and can perform more complex arithmetic. However, the naïve methodology of using BN layers after every Conv layer, leads to the use of too many BN layers, which overloads the GPUs. On further analysis, we found that we can reduce the number BN layers necessary for training CNNs significantly [27]; this will necessitate fewer GPUs. We can achieve this by using a Bayesian Optimization formulation which can quickly determine the most suitable locations for BN layers. The BO optimized CNNs prevent exploding/vanishing gradients in deep CNNs, while using significantly fewer BN layers than their traditional counterparts; hence, fewer GPUs are needed.

Handling accuracy drop due to low precision: As mentioned earlier, ReRAMs use 16-bit fixed point precision. Lower precision reduces power and area requirements and leads to better performance. However, it can lead to accuracy drop during training. In the architecture shown in Fig. 7, the problem of accuracy loss due to low precision is solved by using stochastic rounding. Stochastic rounding is a probabilistic rounding scheme with a zero expected rounding error and is crucial for low-precision training [28]. The stochastic rounding circuit is shown in Fig. 7(b) [29]. It consists of three parts: 1) LFSR: generates pseudorandom 16-bit number; 2) Adder: adds the 32-bit ReRAM output with the 16-bit number generated by the LFSR; and 3) Truncate: this truncates the data to 16 bits after addressing over-/under-flow conditions. Stochastic rounding enables high accuracy CNN training under low precision.

Addressing resistance change due to temperature and thermal noise: The resistance instability of ReRAMs can affect the reference and output currents. This would lead to misjudging the stored data, which can corrupt the subsequent computed CNN outputs. This can lead to accuracy loss for CNN training. Therefore, a thermal reference cell (TRC) that averages out these fluctuations in the architecture is necessary [as shown in Fig. 7(c)] [30]. The inclusion of a TRC in each ReRAM tile ensures that any change in output current (due to change in temperature) will be tracked. Next, to address thermal noise, the CNN layers can be mapped to the tiles following a joint performance-thermal aware multi-objective optimization algorithm. The more compute intensive layers should be mapped to ReRAMs closer to the sinks (considering the 3D architecture). As a result, heat generated in these layers will be dissipated fast, which reduces thermal noise in ReRAMs; this leads to better training accuracy.

Solving accuracy loss due to faults: Fig. 8 demonstrates the effects of faults on CNN weights. Fig. 8 shows the maximum and average weight values observed for Conv-2 layer in VGG-11 during training considering both faulty and ideal hardware. The maximum (M) and the average (A) of all the weights in the chosen CNN layer (Conv-2) for the ideal training, are referred as M-Ideal and A-Ideal in Fig. 8. Similarly, the average and maximum weight values for training using faulty hardware is referred as A-Fault and M-Fault respectively; here we assume fault density of 2% as an example. Fault density is defined as the fraction of cells that are faulty in an ReRAM tile. Fig. 8 shows that on an average, weights have identical values, irrespective of whether they are trained with ideal or faulty ReRAMs. However, the maximum weight values (M-Fault and M-Ideal) show a stark difference. It is clear from Fig. 8 that a handful of weights explode (increase very fast) in the presence of faults. This increase at a rapid pace makes the CNN training unstable after a few iterations. Compared to M-Fault, M-Ideal increases gradually over time (number of iterations), leading to successful training. This suggests that this rapid explosion is an anomalous behavior exhibited by only a handful of weights [31].

From the observations in Fig. 8, we conclude that ReRAM faults result in exploding weights, which in turn leads to unstable training and poor accuracy. Hence, clipping these exploding weights to a relatively lower value ϵ, where $\epsilon > 0$, will enable the CNNs to train successfully. The clipping operation can be mathematically expressed using the following equation:

$$|w| = \begin{cases} |w|, \text{if } |w| < \epsilon \\ \epsilon, \text{otherwise} \end{cases} \tag{6}$$

Weight clipping prevents accuracy loss in the presence of static and dynamic faults. This happens as clipping the unrealistically large weights prevents the CNN training from becoming unstable. As a result, the backpropagation algorithm has a much better chance to train the remaining weights and compensate for the ones mapped to the faulty cells. Hence, we can use weight clipping as a solution for reliable CNN training on faulty ReRAM-based PIM architectures. The operations associated with weight clipping are simple and can be implemented using the GPUs without additional overhead.

IV. ReRAM-based PIM for GNNs

In this section, we present a ReRAM-based architecture for accelerating GNN training and inferencing. Unlike CNNs, GNNs present a unique set of problems for ReRAM-based architectures that must be addressed. For instance, GNNs are typically only a few layers deep. Hence, low precision and BN layers are not important for GNN training. However, the amount of data communicated during GNN training is extraordinarily high. The amount of traffic is proportional to the total number of nodes, which considering all the features in the input graph, is often very high. For example, each image in the popular ImageNet dataset (often used for evaluating DNNs) consists of 150K entries (image size: 224x224x3) which is an order of magnitude smaller than some of the smallest graph datasets, such as the PPI dataset (1.6 million entries) [32]. GNNs perform an iterative neighborhood aggregation operation, where each node aggregates features of its neighbors to compute the new features [11] [33]. After k iterations, the transformed feature vector of a node captures the relational structure information within this node's *k-hop* neighborhood. The message passing operation gives rise to significant data exchange among the ReRAM tiles, which limits the overall achievable performance. The graph datasets can be even larger with millions of nodes. This results in significant amount of data exchange during GNN training. Simply accelerating the computations of GNNs using ReRAMs is not sufficient as the communication will create performance bottlenecks. The computations will be repeatedly stalled as the processing units has to wait for the data to arrive.

Reducing traffic hotspots during GNN training: Fig. 9 shows an example GNN with three neural layers. Each neural layer in a GNN consists of two sub-layers: (a) V-layer: this layer is like a FC layer in CNN (V1, V2 and V3), and (b) E-layer: this layer resembles message passing in graph computations (E). Each V-layer has a unique set of weights which need to be mapped/assigned to different sets of ReRAM tiles (V-tiles) for computation. The E-layer requires only the graph adjacency matrix (Adj). The Adj matrix is fixed for a given graph and is mapped to another set of ReRAM tiles (E-tiles). The E-tiles are shared by all the neural layers in a GNN. As shown in Fig. 9, the output of the V-layer is used as input for the next E-layer and so on. This results in a many-to-one communication pattern as multiple sets of V-tiles communicate with the same set of E-tiles. Without a suitable interconnection backbone, the many-to-one communications can overwhelm the training process, resulting in a performance bottleneck. Moreover, training involves data sharing between the forward- and backward-phase computations of each layer; often the backward-phase computations are implemented on separate set of ReRAMs, as described in [1]. Overall, this results in the output of layer L_i being sent to: (a) PEs responsible for the next layer L_{i+1}, and (b) the PEs responsible for the backward phase of layer L_i. Therefore, there is a significant amount of multicast traffic on top of the many-to-one traffic. Similalr to CNN, in this case also a 3D architecture can alleviate the communication bottleneck problem. Fig. 10 shows a ReRAM-based 3D manycore architecture for training GNN. Overall, this 3D manycore architecture has four planar tiers stacked on top of each other, each planar tier consisting of multiple ReRAM-based PEs with the same crossbar configuration. The PEs in different planar layers are connected with each other using TSV-based vertical links as shown in Fig. 10. This 3D manycore architecture outperforms conventional GPU-based designs significantly.

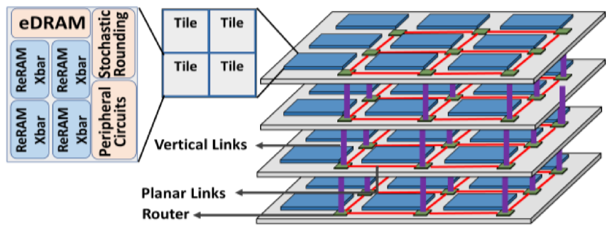


Fig. 10. ReRAM-based manycore (many tiled) architecture for GNN training.

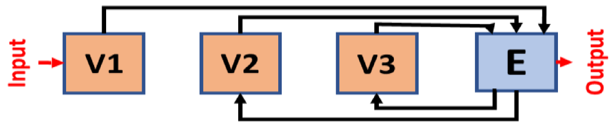


Fig. 9: GNN structure with three neural layers as an example; The arrows indicate the data communication pattern in a GNN. [3]

Efficient storage of graph structured (sparse) data: As discussed earlier, GNNs involve characteristics of both DNN and graph computations. Therefore, it is essential to efficiently store graph data to accelerate GNN training using ReRAM crossbars. This is typically achieved using smaller crossbars [12] [13]. However, these architectures are inefficient in terms of power, performance, and area overheads. To find the most effective solution for storing sparse data on ReRAM crossbars, it is important to 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. 11 shows the number of PEs required, area and power needed to store one input sub-graph of the Reddit dataset [32]. Note that, similar trends were seen for other datasets as well. As shown in Fig. 11, 8x8 sized crossbars require 28X times more PEs than the 128x128 configuration to store the same amount of information. This happens as smaller crossbars need more PEs to store the same adjacency matrix, necessitating a larger number of peripheral circuits such as ADC, DAC, and routers. Larger crossbars store more redundant zeros [12]. Despite that, they are more efficient in terms of area and power, resulting from the lower PE requirements. However, extremely large crossbars (beyond 128x128) 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 [20] [34], which is not only difficult to design but is extremely area- and power-hungry, overshadowing any benefit of larger crossbars. Thus, from Fig. 11, 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. Hence, 128×128 crossbars must be used for ReRAM-based GNN accelerators.

V. Conclusion

Processing-in-memory (PIM) is an enabling technology to break the memory wall by processing the data in-situ (i.e., at the location where the data is stored). Resistive random-access memory (ReRAM)-based PIM architectures are popular for accelerating various deep learning algorithms. However, when multiple PIM-based computing cores are integrated to design a manycore architecture then we need to address various challenges to establish suitable power-performance-accuracy trade-offs. This paper highlights advantages and challenges associated with PIM-based manycore computing targeted towards CNNs and GNNs.

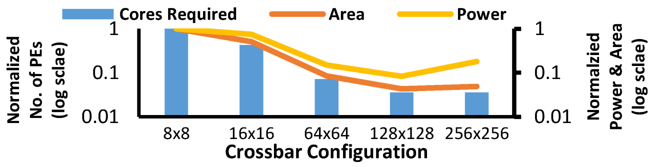


Fig. 11. Storage Efficiency-Power-Area trade-offs for different ReRAM crossbar configurations, normalized w.r.t. 8x8 crossbar configuration

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] F. Chen, L. Song and Y. Chen, "ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 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] M. Hu et al., "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX, USA, 2016.

[8] P. Micikevicius et. al., "Mixed precision training," in *International Conference on Learning Representations (ICLR)*, Vancouver, Canada, 2018.

[9] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,," in *Internation Conference on Machine Learnig (ICML)*, 2015.

[10] B. K. Joardar, B. Li, J. R. Doppa, H. Li, P. P. Pande and K. Chakrabarty, "REGENT: A Heterogeneous ReRAM/GPU-based Architecture Enabled by NoC for Training CNNs," in *IEEE/ ACM Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, 2019.

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

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

[13] 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.

[14] B. K. Joardar et al., "GRAMARCH: A GPU-ReRAM based Heterogeneous Architecture for Neural Image Segmentation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020 .

[15] 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.

[16] A. Chaudhuri and K. Chakrabarty, "Analysis of Process Variations, Defects, and Design-Induced Coupling in Memristors," in *IEEE International Test Conference (ITC)*, Phoenix, AZ, USA, 2018.

[17] L. Xia, M. Liu, X. Ning, K. Chakrabarty and Y. Wang, "Fault-Tolerant Training Enabled by On-Line Fault Detection for RRAM-Based Neural Computing Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 38, no. 9, pp. 1611-1624, Sept. 2019.

[18] P. O. Vontobel, W. Robinett, P. J. Kuekes, D. R. Stewart, J. Straznicky and R. S. Williams, "Writing to and reading from a nano-scale crossbar memory based on memristors," *Nanotechnology,* vol. 20, 2009.

[19] C. Xu et al., "Overcoming the Challenges of Crossbar Resistive Memory Architectures," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2015.

[20] 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.

[21] X. Glorot and Y. Bengio, "Understanding the difficulty of training deepfeedforward neural networks," in *AISTATS*, 2010.

[22] K. He, X. Zhang, S. Ren and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015.

[23] B. K. Joardar et al., "Learning-Based Application-Agnostic 3D NoC Design for Heterogeneous Manycore Systems," *IEEE Transactions on Computers,* vol. 68, no. 6, pp. 852-866, 2019.

[24] E. Esmanhotto et al., "High-Density 3D Monolithically Integrated Multiple 1T1R Multi-Level-Cell for Neural Networks," in *IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA, USA, 2020.

[25] M. V. Beigi and G. Memik, "Thermal-aware Optimizations of ReRAM based Neuromorphic Computing Systems," in *Design Automation Conference (DAC)*, 2018.

[26] Y. Yu and N. K. Jha, "A Monolithic 3D Hybrid Architecture for Energy Efficient Computation," *IEEE Transactions on Multi-Scale Computing Systems,* vol. 4, no. 4, pp. 533-547, 2018.

[27] B. K. Joardar et al, "High-Throughput Training of Deep CNNs on ReRAM-based Heterogeneous Architectures via Optimized Normalization Layers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 2021.

[28] S. Gupta, A. Agrawal, K. Gopalakrishnan and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015.

[29] T. Na et. al., "On-chip training of recurrent neural networks with limited numerical precision," in *ICJNN*, 2017.

[30] Y. H. Lin et al., "Device Instability of ReRAM and a Novel Reference Cell Design for Wide Temperature Range Operation," *IEEE Electron Device Letters,* vol. 38, no. 9, pp. 1224-1227.

[31] B. K. Joardar et al., "Learning to Train CNNs on Faulty ReRAM-based Manycore Accelerators," *ACM Transactions on Embedded Computing Systems (TECS), as part of ESWEEK,* 2021.

[32] 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.

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

[34] B. Murmann, "ADC Performance Survey 1997-2020," 2020. [Online]. Available: https://web.stanford.edu/~murmann/publications/ADCsurvey_rev20200802.xls.