GRAMARCH: A GPU-ReRAM based Heterogeneous Architecture for Neural Image Segmentation

Biresh Kumar Joardar*, Nitthilan Kannappan Jayakodi*, Janardhan Rao Doppa*, Hai Li†, Partha Pratim Pande*, Krishnendu Chakrabarty†

*School of EECS, Washington State University
Pullman, WA 99164, U.S.A.
{biresh.joardar, n.kannappanjayakodi, jana.doppa, pande}@wsu.edu

†Department of ECE, Duke University
Durham, NC, USA.
{hai.li, krishnendu.chakrabarty}@duke.edu

***Abstract*— Deep Neural Networks (DNNs) employed for image segmentation are computationally more expensive and complex compared to the ones used for classification. However, manycore architectures to accelerate the training of these DNNs are relatively unexplored. Resistive random-access memory (ReRAM)-based architectures offer a promising alternative to commonly used GPU-based platforms for training DNNs. However, due to their low-precision storage capability, these architectures cannot support all DNN layers and suffer from accuracy loss of the learned models. To address these challenges, we propose *GRAMARCH*, a heterogeneous architecture that combines the benefits of ReRAM and GPUs simultaneously by using a high-throughput 3D Network-on-Chip. Experimental results indicate that by suitably mapping DNN layers to processing elements, it is possible to achieve up to 53X better performance compared to conventional GPUs for image segmentation.**

***Keywords*— Heterogenous, 3D, ReRAM, NoC, DNNs**

I. Introduction

Many advanced image analysis tasks require predicting a categorical label for *each* pixel in the image, also known as image segmentation. DNNs employed for segmentation, e.g., U-Net [1], FCN [2], are usually more complex and computationally expensive than their classification counterparts. However, the design and optimization of suitable hardware architectures targeting these DNNs remain relatively unexplored. From a hardware perspective, DNNs for image segmentation present two major challenges: (a) DNNs for segmentation introduce additional layers, e.g., Deconvolution. Conventional hardware architectures for DNNs are not optimized for deconvolution; and (b) Segmentation introduces *skip connections* between layers for feature sharing to improve prediction accuracy [1][2]. This results in a significant increase in data communication when executing training/inference on a manycore platform. Due to their inherent parallelism, GPUs are often the first choice to train and execute any kind of DNN architectures. However, despite their popularity, GPUs have some major limitations: (a) high area, (b) high power consumption, and (c) limited memory bandwidth [3]. The additional computation involved with these DNNs (discussed above) will lead to higher power consumption and execution time on GPUs, while the additional data traffic will put even more stress on the already limited memory bandwidth. A good hardware design for image segmentation should optimize for both deconvolution and skip connections.

Metal-oxide resistive random-access memory (ReRAM) has been used to design high-performance hardware architectures for DNN training and inference due to their ability to perform matrix and vector operations [4][5]. ReRAMs are more area and energy efficient compared to their GPU counterparts [3]. Furthermore, ReRAMs compute "*in-memory*" without the need for expensive off-chip memory access. Hence, ReRAMs provide a promising solution to implement DNN training [5]. However, existing ReRAM-*only* implementations for DNNs have some notable limitations: (a) ReRAMs have low precision, which affects prediction accuracy as backpropagation algorithm used to train DNNs, is precision sensitive [6]. Despite all the advantages, training DNNs on ReRAMs can potentially lead to no convergence or a loss of accuracy in the trained model [6][7]; and (b) ReRAMs are not suited for implementing all DNN layers, e.g., Batch-Normalization and Soft-Max. Both these layers require high precision to prevent overflows [8][9]. Existing ReRAM-only architectures are not able to support these layers due to the absence of any high-precision computing units. Hence, such ReRAM-only solutions have limited capabilities and require additional external resources to enable these precision-critical layers such as normalization. Therefore, we need to address these shortcomings of ReRAM-only solutions. Furthermore, the training of a DNN involves repeated memory access to fetch data and weights along with data sharing between DNN layers. In DNNs for image segmentation, communication is a major concern as the *skip connections* introduce additional traffic. Without appropriate Network-on-Chip (NoC) support, the high amount of traffic can create performance bottlenecks resulting in an increase in execution time.

In this paper, we address all these challenges by proposing *GRAMARCH*: a GPU and ReRAM-based architecture for image segmentation that has the following capabilities: (a) Accelerate the matrix and vector operations necessary for training, (b) Appropriately map DNN layers on *GRAMARCH* to minimize the overall communication, (c) Design a 3D NoC for high-throughput data movement (d) Execute any DNN layer including Normalization and Soft-Max, (e) Solve the problem of accuracy loss due to low precision computation with ReRAMs.

II. Related Work

Floating-Point (FP32) precision is the *de-facto* representation for variables while training DNNs. However, low-precision training offers a more compute- and memory-friendly alternative [11]. Prior works, e.g., [8][11][12], have successfully employed low-precision for training/inference. However, training with low precision is challenging as weight updates are often too small and get rounded to zero, resulting in accuracy loss [8]. Moreover, DNN layers such as Normalization and Soft-Max are prone to overflow errors and require full-precision support [7][8]. To enable these layers and preserve the quality of weight updates, mixed-precision training for DNNs was proposed in [8][9]. However, this methodology requires storing an extra copy of weights and gradients in FP32, which is expensive in terms of memory. In [11], the authors propose

This work was supported, in part by the US National Science Foundation (NSF) grants CNS-1564014, CCF 1514269, CSR-1717885 and USA Army Research Office grant W911NF-17-1-0485.

stochastic rounding to train DNNs in low precision, which eliminates the excess memory overhead. However, they only focus on the software implementation aspect of this problem.

ReRAMs can be employed as memory as well as to perform *in-situ* multiply-and-accumulate (IMA) [5]. Hence, ReRAM-based PIM architectures are popular for accelerating both inference [10][13] and training of DNNs [5][14]. However, as mentioned above, ReRAM-based designs have two main drawbacks: (a) trained models suffer from accuracy loss [6][7], and (b) they do not support layers like normalization [10]. Existing sole ReRAM-based solutions, e.g., PRIME [13], TIME [14], ISAAC [10], and PipeLayer [5] do not address these challenges. REGENT [15] solves the accuracy loss problem using a combination of ReRAM and GPUs. However, it does not support normalization/soft-max layers. Also, existing works mostly target DNNs for classification and are not optimized for image segmentation. In [16], the authors have designed an ReRAM-based architecture for segmentation. However, it only supports inference and does not consider the effect of skip-connections. Skip-connections introduce high traffic with multicasting. Without suitable hardware support, this can repeatedly stall execution, leading to higher execution time.

In this work, we propose the design of *GRAMARCH*: a high-throughput, NoC-enabled, heterogeneous architecture with GPU and ReRAMs, targeted specifically to train DNNs for image segmentation. *GRAMARCH* advances the state-of-the art by addressing the shortcomings of existing ReRAM-only solutions.

III. The *GRAMARCH* Architecture

A. Training on ReRAMs: Challenges

Image segmentation involves predicting a class label for every pixel in an image. Typical DNN architectures employed for segmentation consists of convolution and deconvolution layers (followed by pooling and/or normalization) with Skip connections [1][2]. Normalization layers are often used with these DNNs for better prediction accuracy and improvement in training time [17]. Fig. 1 illustrates a typical DNN used for image segmentation. Deconvolution involves sparse matrices that introduce significant amount of redundant multiplications with zeros whereas skip connections introduces high amount of traffic with multicasting. Conventional ReRAM-based architectures intended for classification, are not optimized for either of them, which can lead to sub-optimal performance.

Moreover, ReRAM-based implementations can only support limited precisions (2 to 6 bits of storage per cell [5]), which presents a major hurdle for reliable DNN training. Complex DNNs including CNNs, are sensitive to precision of data. Low precision leads to poor convergence behavior and/or accuracy loss for the trained model [6][7]. This happens as the weight and activation gradients are often too small to represent in low precision for many DNNs [8]. Moreover, layers that perform large reductions (sums across the elements of a vector), e.g., normalization and softmax, are prone to data overflow and should be carried out in full precision [8]. ReRAM-based implementations cannot implement these layers due to limited precision. This enforces unnecessary constraints on the choice of DNNs that can be trained (e.g., no normalization layers) on existing ReRAM-only architectures, which can lead to lower accuracy and/or higher training time. Existing solutions for reliable low-precision training e.g. [8], have high memory overhead (~1.5X higher than FP32). Stochastic rounding is a light-weight alternative, but not realizable with ReRAMs. Also, low-precision training with stochastic rounding requires normalization to be effective [7]. Existing ReRAM-only architectures cannot execute normalization due to low-precision and will require additional hardware to execute this operation.

Moreover, the role of NoC has not received significant attention in ReRAM-based architectures. For instance, Pipelayer [5] achieves significant speed-up by using a clever pipelining of DNN layers. Due to the higher SIMD capability of ReRAMs, it is possible to achieve extra-ordinary amount of parallelism in each layer resulting in such speed-up. However, communication is one of the major bottlenecks in a pipelined implementation. Without an appropriate on-chip communication infrastructure, the pipeline stages will be repeatedly stalled waiting for data to arrive, leading to higher execution time. Hence, *allocating more resources to speed-up computation only is not effective if communication cannot keep up with it*. The overall execution time will depend on both the time to compute and time to communicate. The problem is more serious in DNNs for segmentation due to their use of skip connections that results in increased memory access and on-chip traffic. This exerts more stress on the already limited memory bandwidth and the on-chip network. A suitable hardware architecture to accelerate DNNs used for segmentation must address all these challenges.

B. Overall Architecture

To overcome these challenges, we propose *GRAMARCH*: a heterogeneous, 3D NoC enabled GPU and ReRAM architecture with stochastic rounding to accelerate DNN training for image segmentation. Fig. 2 shows the *GRAMARCH* architecture,

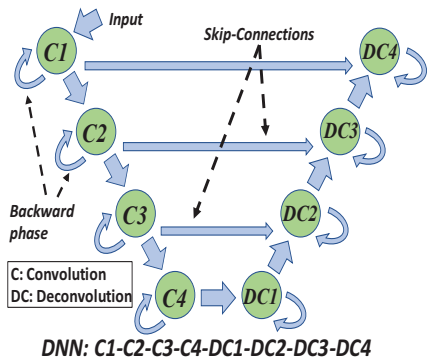


Fig. 1: Communications in a U-Net *like* architecture (ReLU, Pooling are included after each layer)

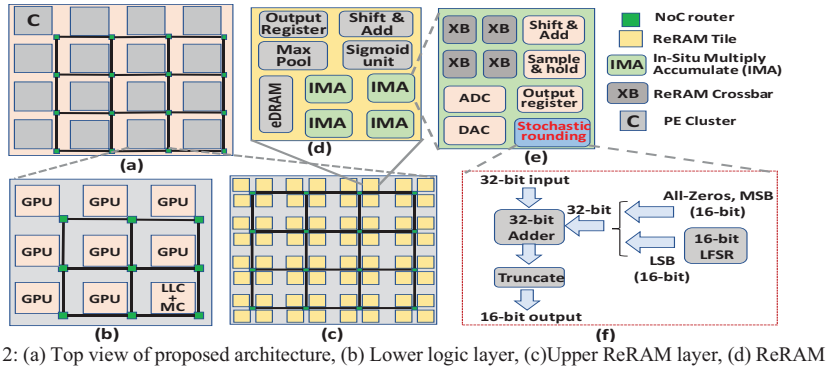


Fig. 2: (a) Top view of proposed architecture, (b) Lower logic layer, (c)Upper ReRAM layer, (d) ReRAM tile, (e) In-situ Multiply Accumulate (IMA) and (f) Stochastic rounding implementation. (This figure is for illustration purpose only; Actual Implementation details of *GRAMARCH* is mentioned in Sec. IV(A))

which consists of two layers: the bottom layer consists of GPU and Last Level Cache (LLC) tiles (Fig. 2(b)) while the top layer consists of ReRAM (Fig. 2(c)). The overall architecture is divided into multiple clusters for a two-level NoC design. The clusters are identical in terms of the number of Processing Elements *a.k.a*. PEs (ReRAM, GPU, and LLCs).

The ReRAM layer consists of *morphable subarrays* that can be configured for both storage and computation. Therefore, the ReRAM layer performs dual roles of both computing and storage. The ReRAM tiles (Fig. 2(d)) are designed following [10]. Each tile includes eDRAM buffers, in-situ multiply-accumulate (IMA) units, output registers, along with shift-and-add, sigmoid, and max-pool units. The IMAs (Fig. 2(e)) have multiple crossbar arrays along with other peripheral circuitry connected with a shared bus. However, unlike [10], each ReRAM tile in the *GRAMARCH* architecture is equipped with additional peripheral circuitry to implement stochastic rounding (Fig. 2(e)) to address the accuracy loss due to low precision of ReRAMs. The lower (logic) layer consists of GPU and LLC tiles to support the precision-critical portions of DNN training. The GPU tiles include a GPU core, while the LLC tiles include a portion of the Last-Level cache and a memory controller (MC) that can access data from the ReRAM (memory) layer. In addition, each tile also includes a router for communication. The LLC tiles are connected to the ReRAM layer using Through-Silicon Vias (TSVs). This 3D integration enables low-latency and high-bandwidth access to ReRAM-based memory, which alleviates the memory bottleneck problem in conventional GPUs. The PEs in both layers are connected via a high-throughput NoC (discussed in a later section). Next, we discuss two key features of the *GRAMARCH* architecture.

Stochastic rounding: Stochastic rounding *is an unbiased rounding scheme that makes a probabilistic decision of where to round and has the desirable property that the expected rounding error is zero*. This property of stochastic rounding makes it attractive for low-precision DNN training and prior work have shown good results for training classification models [11]. However, DNNs for segmentation can also benefit from it. Fig. 3 shows the training error when U-Net [1] (a popular DNN for image segmentation) is trained with both floating point (FP) and low-precision + stochastic rounding (LP+SR) using images from the *CARVANA* dataset [24]. It is clear from Fig. 3 that there is negligible difference in accuracy between U-Net trained with low-precision + stochastic rounding and a conventional full floating-point representation. Similar behavior is observed with the other DNN architectures considered in this work.

This shows that stochastic rounding support in *GRAMARCH* enables it to retain the efficiency of ReRAMs (high-throughput and low-energy matrix multiplications) while achieving near-GPU accuracy. However, ReRAMs cannot implement stochastic rounding; additional peripheral circuits are required to enable stochastic rounding in ReRAMs. The stochastic rounding circuit used in this work is based on the implementation in [7]. It consists of three parts: (a) LFSR: It generates pseudo-random 16-bit sequences, (b) Adder: It adds the 32-bit number (that is to be rounded) with the LFSR output, and (c) Truncate: This truncates the result of addition to 16-bits after addressing over/under-flow conditions (as shown in Fig. 2(f)). Here, we adopt the pipelined DNN implementation commonly used in ReRAM-based architectures [5]. Hence, it is important that the additional circuitry must match the throughput of ReRAM crossbars to ensure that the pipeline of DNN execution is not stalled. For example, an $N \times N$ ReRAM crossbar will produce N bitline currents in one ReRAM cycle (100ns, which is the read latency for the crossbar array [10]). In a pipelined implementation, outputs are generated in every ReRAM cycle. Thus, all N bitline currents need to be processed in one cycle, before the next set of N bitline currents are generated in the next cycle to avoid pipeline stalls. We design the stochastic rounding circuit following this timing constraint for a 16-bit output using the Cadence Genus synthesis tool and Nangate 45nm library. Synthesis results indicate that stochastic rounding can be enabled in ReRAM crossbars with negligible area overheads (each circuit adds <1% of IMA [10] area).

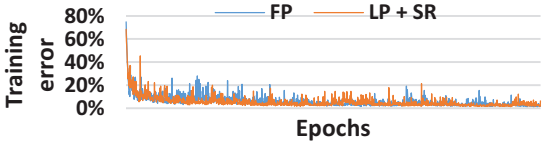


Fig. 3: Training error for U-Net with Full Precision (FP) and Low precision with stochastic rounding (LP + SR) on the *CARVANA* dataset [24]

Deconvolution: Deconvolution can be implemented by inserting zeros to smaller inputs, followed by a convolution operation on the upsized input data [18]. Existing ReRAM architectures, are not optimized for such an operation and will spend significant amount of time performing multiplications with zeros wasting both computing resources and communication bandwidth. In [18], the authors address this by using a zero-skipping methodology. For brevity, we do not repeat the details of zero-skipping here. By leveraging the knowledge of predetermined sparseness patterns, it is possible to skip multiplications by zeros to reduce execution time and on-chip traffic. On-chip traffic has a major impact on overall performance. Hence, we include zero skipping in *GRAMARCH* to reduce on-chip traffic and enable better mapping of DNNs to PEs. However, only adopting zero-skipping optimizations is not sufficient. Skip-connection, which is another key feature in DNNs used for segmentation, also results in an increase in on-chip traffic. This can cause performance bottleneck and needs extra architectural considerations. We address this problem next.

C. NoC design and DNN mapping

In this subsection, we first discuss the NoC design used in *GRAMARCH*. Next, we discuss the importance of mapping the DNN layers on appropriate PEs for achieving best performance.

NoC design: As shown in Fig. 2, the proposed architecture consists of two layers: the upper layer consists of ReRAM tiles while the bottom layer consists of GPUs and LLCs. Due to heterogeneity in the proposed architecture, there are two distinctly unique execution platforms: (a) ReRAMs provide high-throughput and energy-efficiency, but low precision execution; (b) GPUs provide relatively slower but a full-precision computing platform. For the best performance and negligible loss of accuracy, we should map the compute-intensive layers, i.e., convolution and deconvolution on the faster ReRAMs. The precision-critical yet less time-consuming Normalization and Soft-Max layers should be mapped on to the GPUs. In the absence of any precision sensitive layers, the output of one ReRAM crossbar is directly communicated to other ReRAM(s) for further processing in the upper layer only (without the involvement of the lower layer). However, in the

case of normalization (or softmax), there is vertical traffic between the ReRAM and GPU layers. These traffic patterns must be considered to design the NoC for optimal performance.

It is well known that mesh is widely adopted as the NoC architecture of choice for manycore architectures. However, the communication in DNNs is limited to (a) data sharing between adjacent layers in the DNN, and layers connected using the skip connections, and (b) Memory traffic to fetch/store data while performing the computations in each DNN layer. As a result, only a limited number of links in a mesh NoC are utilized [19]. The majority of the links remain unused (or under-used), leading to load imbalance. Hence, we can concentrate several of these PEs that inject less traffic, into clusters, to create a *hierarchical two-level NoC*, which can deliver similar level of performance as a mesh with fewer links. It also improves load balancing and enables the design of more scalable architectures [19][22].

Hence, we adopt a two level NoC design (Fig. 2). To design the two-level NoC, all available PEs are divided among *C* clusters. The PEs in each cluster are interconnected by the first level mesh NoC while each cluster is connected to other clusters via the second level mesh NoC. Note that due to its multi-hop nature, mesh NoC is not effective for long-range communication. However, as each DNN layer communicates with a limited number of other layers only, it is possible to reduce long-range traffic by mapping these communicating DNN layers to nearby PEs. Hence, a mesh NoC suffices for both levels in the two-level NoC; spatial nearness among communicating DNN layers is ensured by the DNN mapping to PEs (hence traffic is mostly short-range). We discuss the role of mapping in next subsection. Also, mesh NoC can implement efficient multicast techniques, e.g., tree multicast, that leads to further performance improvements. Hence, we design a two-level mesh-mesh NoC in both layers of the proposed architecture as depicted in Fig. 2. On the other hand, the vertical traffic from ReRAM to GPUs is facilitated by the TSVs. The GPUs access data from ReRAMs via the LLC (and MC) tiles which connects to the ReRAM cluster directly above it via the vertical links. The 3D integration allows high bandwidth and low latency data transfer from ReRAM to the GPU cores. Due to lack of long-range traffic (ensured by the mapping policy), similar two-level NoC is used in the logic layers as well i.e. GPUs and LLCs in same cluster are connected via the first level NoC while individual clusters communicate via the second level mesh NoC.

Interestingly, we note from Fig. 1, that there is a significant amount of multicast communication during DNN training. For example, in a U-Net like architecture (as shown in Fig. 1), PEs executing the forward phase of C2 must share the output with other PEs responsible for (a) DNN layer C3, (b) DC3, and (c) backward phase of C2. In conventional unicast-based hardware architectures, this would result in a ~3X increase in the number of messages originating from PEs executing C2. In a pipelined DNN implementation, where communication is a major bottleneck, this is undesirable. Hence, multicast support by the NoC is crucial. The mapping policy plays a significant role as well. Since output of previous layer(s) is the input to the current layer, mapping the current layer across *N* PE clusters also results in a *N*-fold increase in traffic (i.e., multicast due to mapping). For example, in Fig. 2, if C1 (Convolution-1) is mapped to Cluster-1 and C2 is mapped to Cluster-2 and Cluster-3, then the output of C1 needs to be communicated from Cluster-1 to both

Algorithm 1: Mapping DNN layers to PEs

Input: DNN architecture, No. of available PEs
Output: *Map* (Best DNN Layer mapping to PEs)
Algorithm:

```
1  numPE [.] = No. of ReRAMs/GPUs for given DNN
2  Map [.] = Random allocation of numPE [.]
3  Simulated Annealing (Repeat for MAX iterations):
4      MapNew [.] = Perturb (Map [.])
5      TrafficOnEachLink [.] = Traffic (MapNew [.])
6      Cost (MapNew [.]) = max (TrafficOnEachLink [.])
7      If P(Cost(MapNew [.]), Annealing Temperature):
8          Map [.] = MapNew [.]
9  return Map [.] with best Cost
```

Cluster-2 and Cluster-3 (resulting in multicast). Therefore, multicast communication not only depends on the connectivity between the layers of DNNs, but it also arises due to mapping of layers across multiple clusters. Unnecessary/random splitting across multiple clusters will only increase the amount of multicast. To handle the multicast communication and reduce redundant traffic in NoC, we implement the tree multicast technique. Unlike unicast communication, where each destination cluster(s) receives a distinct message originating from the source cluster, tree multicast sends a single copy of the message to all destination cluster(s) along a common path. This reduces the number of messages flowing through the NoC, which accelerates training. Tree multicast is only used as an example here. Other multicast techniques can also be employed here. Next, we discuss mapping of DNN layers to *GRAMARCH*.

Mapping DNN layers to PEs: Algorithm 1 shows a high-level description of the DNN mapping strategy adopted in this work. For the pipelined DNN implementation, all DNN layers need to be executed simultaneously. Hence, we need to first allocate adequate resources (ReRAMs and GPUs) to each DNN layer based on its individual requirement. Note also that in a pipelined implementation, the overall delay is dominated by the slowest stage. Hence, we need to balance all the pipeline stages, i.e., execution time of the DNN layers should be similar to each other. As each layer has different number of operations to perform, some layers take longer than others. In pipelined DNN training, the overall execution time will be bottlenecked by these slower layers. In order to balance the pipeline stages and improve the execution time, the slower layers must be accelerated by increasing the amount of parallelism in computation, i.e., allocate more ReRAM crossbars/GPUs (Line 1, Algorithm-1). This ensures that each pipeline stage in DNN training is balanced, thereby leading to high performance. Considering the above constraints, we can formulate the mapping as a combinatorial optimization problem: given *N* number of total PEs (GPUs and ReRAMs) and *L* number of layers in the DNN, the goal is to distribute the computations of all *L* layers (both forward and backward phases) across the *N* number of PEs to ensure spatial nearness among communicating layers, reduce long-range traffic, and handle multicast efficiently for better performance. There are two key challenge in solving the above optimization problem: (a) Large space of combinatorial solutions prohibiting an exhaustive search, and (b) The objective function to evaluate the quality of a solution depends on experimental analysis to measure the impact of mapping on on-chip traffic. Together, they make the problem

intractable. Therefore, we solve this optimization problem using simulated annealing (SA)-based heuristics as it can uncover high-quality solutions in a reasonable amount of time.

First, we start with a randomly chosen mapping of DNN layers to PEs (Algorithm-1, Line-2). Next, we **Perturb** the candidate mapping solution to get a new mapping (Algorithm-1, Line-3). Here, a valid **Perturb** is defined as allocating whole or part of the resources (PEs) required by a randomly chosen DNN layer, to a different cluster than its current location. For instance, if DNN layer L1 is initially mapped to P PEs in Cluster-1 (Fig. 2), then a valid perturbation can be one of the following: (a) Allocate all P PEs required by L1 to any other cluster (not Cluster-1) that has enough unmapped PEs to accommodate L1, or (b) Allocate P1 PEs to any other cluster (with enough unmapped PEs) while leaving P2 PEs on cluster-1 (Here, P1 + P2 = P). Next, to evaluate the quality of the new mapping, we first determine the on-chip **Traffic** patterns corresponding to a given mapping (Line-5 of Algorithm-1). We discuss the experimental procedure for this in the next section. The optimization tries to reduce the **Cost** which is defined as the maximum amount of traffic flowing through any link in the NoC. This is crucial to ensure spatial nearness between communicating layers as multiple long-range traffic via a link will cumulatively add up, resulting in high traffic. This affects performance as the associated DNN layer(s) will have to wait longer for data to arrive (via the congested link) before it can start computation, leading to higher execution time. Overall, the mapping strategy complements the NoC in delivering best performance. We decide whether to discard/keep the new mapping based on the annealing temperature and **Cost** of both current and previous mapping solutions (Line-8). Finally, we obtain the best mapping (Line-9). We repeat the entire procedure (Algorithm-1) multiple times with different initial solutions and annealing schedules for thorough exploration of design space.

IV. Experimental Results and Analysis

A. Experimental Setup

We employ Gem5-GPU [21] to obtain network- and processor-level information for the logic layer (GPUs and LLCs). We modified the Garnet network within Gem5-GPU to implement the NoC. The LLC in each cluster is shared among all the GPU cores in that cluster. The ReRAM crossbar and tile configurations/area are based on information available in [10]. To the best of our knowledge, Gem5-GPU (or other widely used full system, cycle-accurate simulators) currently do not model DNN training on ReRAM crossbars. However, DNN execution on ReRAM tiles exhibits deterministic behavior without any run-time dependences. Moreover, ReRAM arrays execute instructions in-order and instruction latencies are deterministic [3]. As shown in [10], deterministic execution models suffice to reliably capture ReRAM performance parameters, e.g., execution time, on-chip traffic, etc. Hence, we use the ReRAM execution models from [10] to simulate the ReRAM layer of *GRAMARCH*. The mapping of layer weights to ReRAM tiles and the resulting pipeline structure are determined off-line. Data transfers over the NoC are then statically determined to ensure conflict-free routing at each cycle based on the ReRAM execution models and DNN mapping to PEs. We do not discuss more details about the ReRAM execution models here for the sake of brevity; they are already elaborated in [10]. The ReRAM execution model and Gem5-GPU are used together to simulate all the DNN layers. Some of the important parameters for the *GRAMARCH* architecture are listed in Table-1. The number of ReRAMs in each cluster is obtained based on the area estimates reported in [10] including the synthesis results for stochastic rounding. ReRAM crossbars in the same tile are connected to a bus. Each ReRAM tile communicates with other cores (GPUs and LLCs) and other ReRAM tiles via the NoC.

Table-1: Relevant parameters for the *GRAMARCH* architecture

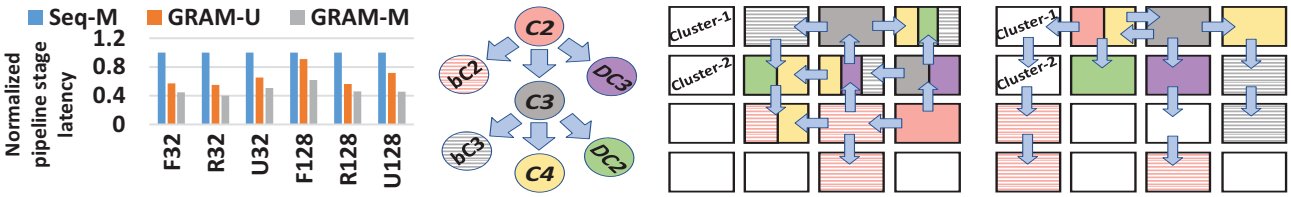
Logic layer: 64-tiles (48-GPUs + 16-LLCs)	
GPUs	Fermi architecture, 1400 MHz, Private L1 (64kB)
LLCs	Shared L2 (1MB) per cluster
ReRAM layer: 64-tiles per cluster, C=16 clusters	
ReRAM tile	8-ADCs(8-bits), 128x8-DACs (1-bit), 12-crossbars, 128x128 memristor size, 10MHz, 2-bit resolution
Stochastic rounding (SR)	8-SRs per tile, 0.0032 mm^2 (~1% of ReRAM tile area)

For experimental evaluation, we choose variants of three commonly employed DNN architectures used for segmentation: (a) U-Net [1]: Fig. 1 illustrates this architecture. *Normalization* is performed after every *convolution*/*deconvolution* layer; and (b) FCN: This is similar to U-Net architecture, with fewer *deconvolutions* and *skip-connections* similar to *FCN-8* architecture [2], and (c) ResNet-FCN [23]: This resembles a ResNet architecture followed by deconvolution layers and has more skip-connections than FCN and U-Net. Both architectures are evaluated for two input sizes: 32×32 and 128×128. We refer the U-Net architecture with input size 32 and 128 as U32 and U128, respectively. Similarly, the FCN, and ResNet equivalents are referred as F32, F128 and R32, R128 respectively.

B. Performance Evaluation

First, we show the importance of appropriately mapping DNN layers on the *GRAMARCH* architecture. Fig. 4(a) shows the effect of multicast-aware placement on pipeline-stage latency with three different candidates: (a) Seq-M: Sequential mapping, i.e., *Layer-i* mapped to *Cluster-i* with multicast support (if a layer needs PEs from more than a single cluster, the excess is allocated among remaining free PEs in round-robin fashion); (b) GRAM-U: Conventional unicast enabled DNN-aware mapping (Algorithm-1) on *GRAMARCH*, and (c) GRAM-M: DNN-aware mapping with multicast enabled on *GRAMARCH*. We observe that, on average, Seq-M performs 34% worse than even GRAM-U. This indicates that without a suitable mapping strategy, only multicast support is not sufficient. On the other hand, *GRAMARCH* with multicast (GRAM-M) consistently outperforms its unicast counterpart (GRAM-U), by 28% on an average. Repeated unicasts in GRAM-U introduce duplicate traffic, which contributes to higher pipeline-stage latency. These results shows that both multicast support and a DNN-aware mapping in the NoC are crucial for achieving high-performance.

To analyze deeper, we also show the actual mapping for F128 on *GRAMARCH* clusters (ReRAM layer only). Fig. 4(b) shows a communication sub-graph for the F128 architecture. For the sake of clarity, we only show a subset of all communications in F128. Fig. 4(c) and Fig. 4(d) show the corresponding mapping of DNN layers on *GRAMARCH* for GRAM-M and Seq-M, respectively. It should be noted that as we only consider the layers in Fig. 4(b) for illustration, some of



(a) (b) (c) (d)

Fig 4: (a) Effect of mapping and multicast on pipeline stage latency, (b) F128 Communication sub-graph, and Corresponding mapping to *GRAMARCH* clusters (ReRAM layer only) in (c) GRAM-M, and (d) Seq-M configurations (C: Convolution, DC: Deconvolution, bC: Backward phase of Convolution)

the clusters in Fig. 4(c) and Fig. 4(d) are empty (the white clusters). The arrows indicate the direction of traffic flow. The precision-critical layers e.g. Normalization, are mapped to the GPUs directly below the ReRAM cluster executing the prior convolution/deconvolution layer. It is clear that *GRAMARCH* with multicast (GRAM-M) ensures spatial nearness and takes advantage of tree multicast during mapping. The communicating layers are all placed nearby within a few hops of each other. Moreover, to take advantage of tree multicast's behavior (single copy of message in each direction), the destination layers have been mostly placed in the same row/column to reduce traffic. On the other hand, sequential mapping results in a more sporadic distribution of communicating layers which results in long-range traffic. Moreover, tree multicast is not as effective due to the sporadic nature of mapping. Therefore, both multicast support and a DNN-aware mapping are crucial for best performance.

Finally, we compare the full-system execution time with a conventional GPU in Fig.5 (shown in log-scale). For this experiment, we implement all the DNN architectures described previously, in Caffe and execute them on an Nvidia GTX 1080Ti GPU. Overall, as is evident from Fig. 5, GRAM-M outperforms conventional GPU architectures by 33.4X on average and by 53X in the best case. This speed-up is mainly achieved due to the accelerations provided by ReRAMs. Also, the mapping strategy ensures that pipeline stages remain balanced and communication does not become a bottleneck. Overall, GRAM-M significantly outperforms GPU-based architectures in training DNNs for image segmentation.

V. Conclusion

ReRAMs provide a high-performance platform for training DNNs. However, their low precision representation capability poses a significant challenge. In this work, we propose *GRAMARCH*, a heterogeneous ReRAM and GPU-based architecture to address this challenge and train DNNs used for image segmentation. Taking advantage of the heterogeneity in architecture involving ReRAMs and GPUs, we map the DNN layers appropriately, e.g., compute-intensive layers on ReRAM and precision-critical layers on GPUs. Furthermore, to ensure that communication does not become a bottleneck, we propose a multicast-aware placement strategy to map DNNs on *GRAMARCH*. Overall, complemented with a high-throughput NoC and optimized layer mapping, *GRAMARCH* with multicast (GRAM-M) can outperform conventional GPUs by up to 53X.

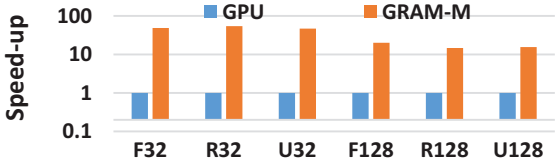


Fig 5: GRAMRACH speed-up in execution time compared to GPU

References

[1] O. Ronneberger, P. Fischer, T. Brox, "U-net: Convolutional networks for biomedical image segmentation", in MICCAI, pp. 234-241, 2015.

[2] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in CVPR, MA, 2015, pp. 3431-3440.

[3] D. Fujiki, S. Mahlke, R. Das, "In-memory data parallel processor", in ASPLOS, pp. 1-14, 2018.

[4] M. Hu et. al., "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in DAC, 2016, pp. 1-6.

[5] L. Song, X. Qian, H. Li and Y. Chen, "PipeLayer: A Pipelined ReRAM Based Accelerator for Deep Learning," in HPCA, 2017, pp. 541-552

[6] Y. Chen et. al., "DaDianNao: A Machine-Learning Supercomputer," in MICRO, 2014, pp. 609-622.

[7] T. Na, et.al , "On-chip training of recurrent neural networks with limited numerical precision," in IJCNN, 2017, pp. 3716-3723

[8] P. Micikevicius et. al., "Mixed precision training," CoRR,abs/1710.03740, 2017

[9] D. Das et al., "Mixed precision training of convolutional neural networks using integer operations," in arXiv preprint arXiv:1802.00930, 2018

[10] A. Shafiee et. al., "ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in SIGARCH., 2016, 14-26.

[11] N. Wang,et. al., "Training deep neural networks with 8-bit floating point numbers," In NIPS, 2018

[12] M. Courbariaux, Y. Bengio, and J. P. David, "Binaryconnect: Training deep neural networks with binaryweights during propagations", in ArXiv e-prints, abs/1511.00363, November 2015

[13] P. Chi et al., "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in ISCA, 2016, pp. 27-39

[14] M. Cheng et al., "TIME:A Training-in-memory Architecture for RRAM-based Deep Neural Networks," in TCAD, 2019, vol. 38, no.5, pp.834-847

[15] B. K. Joardar et. al, "REGENT: A Heterogeneous ReRAM/GPU-based Architecture Enabled by NoC for Training CNNs," in DATE, 2019, pp. 522-527

[16] S. Wen, H. Wei, Z. Zeng, and T. Huang, "Memristive fully convolutional network: An accurate hardware image-segmentor in deep learning," in IEEE TETCI, vol. 2, no. 5, pp. 324–334, 2018

[17] X. Y. Zhou and G. Z. Yang, "Normalization in training u-net for 2D biomedical semantic segmentation," in IEEE RA-L, 2019

[18] Z. Fanet al., "Red: A reram-based deconvolution accelerator," in DATE, 2019, pp. 1763-1768

[19] X. Liu et. al., "Neu-NoC: a high-efficient interconnection network for accelerated neuromorphic systems," in ASPDAC, 2018, 141-146

[20] X. Wang et al., "Low latency and energy efficient multicasting schemes for 3D NoC-based SoCs," in Conf. on VLSI and SoC, pp. 337-342, 2011.

[21] J. Power et.al., "gem5-gpu: A Heterogeneous CPU-GPU Simulator," in IEEE Computer Architecture Letters, vol. 14, no. 1, pp. 34-36, 2015

[22] B. Zimmer et al., "A 0.11 pJ/Op, 0.32-128 TOPS, Scalable Multi-Chip-Module-based Deep Neural Network Accelerator with Ground-Reference Signaling in 16nm," in Symp. on VLSI Circuits, 2019, pp. C300-C301.

[23] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in CVPR, 2016, pp. 770-77

[24] https://www.kaggle.com/c/carvana-image-masking-challenge