ApGAN: Approximate GAN for Robust Low Energy Learning from Imprecise Components

Arman Roohi, *Student Member, IEEE,* Shadi Sheikhfaal, *Student Member, IEEE,* Shaahin Angizi, *Student Member, IEEE,* Deliang Fan, *Member, IEEE,* and Ronald F DeMara, *Senior Member, IEEE*

Abstract—A Generative Adversarial Network (GAN) is an adversarial learning approach which empowers conventional deep learning methods by alleviating the demands of massive labeled datasets. However, GAN training can be computationally-intensive limiting its feasibility in resource-limited edge devices. In this paper, we propose an approximate GAN (ApGAN) for accelerating GANs from both algorithm and hardware implementation perspectives. First, inspired by the binary pattern feature extraction method along with binarized representation entropy, the existing Deep Convolutional GAN (DCGAN) algorithm is modified by binarizing the weights for a specific portion of layers within both the generator and discriminator models. Further reduction in storage and computation resources is achieved by leveraging a novel hardware-configurable in-memory addition scheme, which can operate in the accurate and approximate modes. Finally, a memristor-based processing-in-memory accelerator for ApGAN is developed. The performance of the ApGAN accelerator on different data-sets such as Fashion-MNIST, CIFAR-10, STL-10, and celeb-A is evaluated and compared with recent GAN accelerator designs. With almost the same Inception Score (IS) to the baseline GAN, the ApGAN accelerator can increase the energy-efficiency by $\sim$28.6$\times$ achieving 35-fold speedup compared with a baseline GPU platform. Additionally, it shows 2.5$\times$ and 5.8$\times$ higher energy-efficiency and speedup over CMOS-ASIC accelerator subject to an 11% reduction in IS.

Index Terms—Generative adversarial network, in-memory processing platform, neural network acceleration, hardware mapping.

1 Introduction

Recently, deep Convolutional Neural Networks (CNNs) [1] have shown impressive performance for computer vision, e.g. image recognition tasks, achieving close to human-level perception rates. These neural network models are usually trained using a supervised approach, which limits scalability due to the requirement for large-scale labeled data-sets. The processing demands of high-depth CNNs spanning hundreds of layers face serious challenges for their tractability in terms of memory and computation resources and because of so-called "CNN *power* and *memory wall*" phenomena, conventional processing platforms such as CPU cannot perform this training step. This has been motivating the development of alternative approaches in both SW/HW domains to improve conventional CNN efficiency.

In algorithm-based approaches, use of quantizing parameters [2], and network binarization [3] have been explored extensively to eliminate the need for intensive Multiplication-And-Accumulate (MAC) operations. Recently, utilizing weights with low bit-width and activations reduces both model size and computing complexity [3]. For instance, performing bit-wise convolution between the inputs and low bit-width weights has been demonstrated in [3] by converting conventional MAC operations into their corresponding AND-bitcount operations. Meanwhile to improve computing efficiency of CNNs from the hardware point of view extensive studies for developing deep learning accelerators using GPUs and FPGAs have been researched. However, within conventional isolated computing units and memory elements interconnected via buses, there are serious challenges, such as limited memory bandwidth channels, long memory access latency, significant congestion at I/O chokepoints, and high leakage power consumption [4], [5].

Processing-in-Memory (PIM) paradigms built on top of non-volatile devices, such as Resistive Random Access Memory (ReRAM) [6], [7], Magnetic RAM (MRAM) [8], [9], and Phase Change Memory (PCM) [10] have been introduced to address the aforementioned concerns, such as memory bottlenecks and high leakage power dissipation that has become increasingly prominent with technology scaling. Due to the interesting features of Non-Volatile Memory (NVM) technology such as near-zero standby power, high integration density, compatibility with CMOS fabrication processes, and radiation-hardness, they offer some promising attributes for in-memory processing implementations including the realization of logic functions along with an inherent state-holding capability.

Due to the abovementioned challenges, semi-supervised and unsupervised learning models, such as the Generative Adversarial Network (GAN) algorithm [11], especially Deep convolutional GANs (DCGANs) [12], are of increasing interest. The DCGAN architecture is composed of two separate models. A discriminator model (D) that estimates the probability of a given sample being legitimate or counterfeit. It is trained as a detective to discern between fake samples and real ones. Whereas, the other model, known as the generator (G), samples a uniform random noise input and also captures the real data distribution to generate images as real as possible to deceive the discriminator, as shown in Fig. 1. Basically, this realizes a zero-sum game between the two models. Based on the GAN structure, two training

• *The authors are with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, 32816.*
E-mail: aroohi@knights.ucf.edu

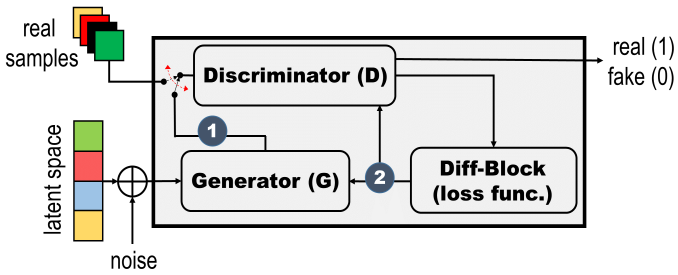

Fig. 1: GAN structure. D downsamples the input data, while G is given a uniform noise distribution to generate fake samples ①. In ②, fine-tuning of training is performed.

processes, i.e. consisting of four forward and four backward passes are required, which are more sophisticated than CNN training with one forward pass and one backward pass. Therefore, implementing an efficient accelerator for GAN using the existing designs for energy and area -constrained IoT nodes, is vital but challenging.

In this paper, to make GAN suitable for resource-limited edge devices, the advancements from both algorithm and hardware architecture perspectives to efficiently accelerate GAN training are deployed. The existing GAN algorithm is modified by replacing the multiplications in convolution layers in the generator (G) model and in the discriminator (D) model, with less complex and more efficient subtraction and addition. In summary, our major contributions in this paper can be listed as follows:

1) We introduce a partial replacement approach which can find the locations of layers in both G and D networks to be quantized in a way to achieve the best performance, a maximum number of quantized layers and lowest accuracy loss. It can massively reduce the required storage and computational resources in the inference paths with the minimum performance degradation compared to the full-precision model.

2) Further improvement in the performance efficiency of systems such as energy and area reduction is achieved by developing a new approximate arithmetic unit. To avoid unacceptable error in output behaviors a partial approximate computing datapath consisting of a precise adder and an approximate adder is developed.

3) We propose a PIM accelerator for GAN, namely *ApGAN*, based on memristor computational sub-arrays and ultra-low power activation function to efficiently accelerate its training within the non-volatile memory. Moreover, we present a pipeline computation optimization approach to further enhance the training efficiency of ApGAN in hardware level.

4) Finally, the evaluation of system accuracy in different data precision and the system performance in speed and energy are carried out. Applying steps 1 to 3 causes an extensive reduction in energy and area as well, whereas an acceptable accuracy is achieved. Our experimental results show that it improves the energy-efficiency and speed by ∼21× and 35.5× speedup compared with GPU platform.

2 DCGAN Review

Compared to conventional CNN topologies, realization of Deep convolutional GAN (DCGAN) [12] implementations have several constraints: a) the strided convolutions and fractional-strided convolutions on D and G, respectively, are utilized instead of the pooling layers; b) Although in the last layer of both D and G models, Sigmoid and Tanh activations are highly used, in the other layers of G and D models, *ReLU* and *LeakyReLU* activations are utilized, respectively; and c) batch normalization is leveraged on both D and G models to stabilize the training process.

DCGANs are composed of two learning subnetworks, a generator (G) as a deconvolutional neural network and a discriminator (D) as a CNN. Usually, these are developed as Deep Neural Networks (DNNs), which are trained simultaneously. Despite traditional unsupervised learning techniques, in GAN, feature representations can be learned from raw data, which results in higher accuracy. The generator learning model can be optimized to produce deceptive samples to fool the discriminator, whereas the discriminator learning model is trained in a way to distinguish the real samples from the artificial ones. The entire process is similar to a 2-player minimax game, which is expressed by:

$$min_G\ max_D V(D,G) = \mathrm{E}_{x\sim p_{data}(x)}[log\ \mathrm{D}(x)] + \mathrm{E}_{z\sim p_z(z)}[log\ (1-\mathrm{D}(G(z)))] \quad (1)$$

where $p_{data}(x)$ is the distribution of data and z is the noise vector. By leveraging minibatch of data samples from D and fake images from G, we minimize $V(D,G)$ regarding G by assuming fixed D and maximize it regarding elements of D by assuming fixed G. Due to the nature of zero-sum game, each of D and G models try to improve their performance, finding a Nash equilibrium point [13], in a non-cooperative manner, which in turn causes several issues like no guarantee for convergence. Some of the most recent and promising advancements in GAN training algorithms are Wasserstein GAN (WGAN) [14], WGAN with weight clipping (WGAN-CP) [15], and WGAN with gradient penalty (WGAN-GP) [15] leveraging modified loss functions. WGAN algorithm uses Wasserstein distance as a quantitative scheme to measure the distance between two probability distributions. Further improvements can be achieved by limiting the trained weights of D in a certain range in WGAN-CP and utilizing gradient penalty in WGAN-GP training algorithms.

Although GAN, particularly DCGAN, can be considered as a dominant algorithm for unsupervised learning technique, which is useful for self-learning IoT nodes [16], its deconvolution/convolutional layers occupy the largest portion of running time and consume significant computational resources, which is crucial for IoT nodes. Therefore herein we focus on developing an optimized in-memory accelerator for both types of layers via algorithm and hardware co-design approach.

3 Approximate GAN (ApGAN) Architecture

Figure 2 depicts the general architecture for our deep convolutional-based Approximate GAN (ApGAN), which consists of four deconvolution and four convolution layers for generator (G) and discriminator (D), respectively. In this section, first, the training procedure of ApGAN is analyzed with respect to the partially-quantized layers. Afterwards, we introduce the method of partial approximate computing to further improvement at the cost of lower accuracy.

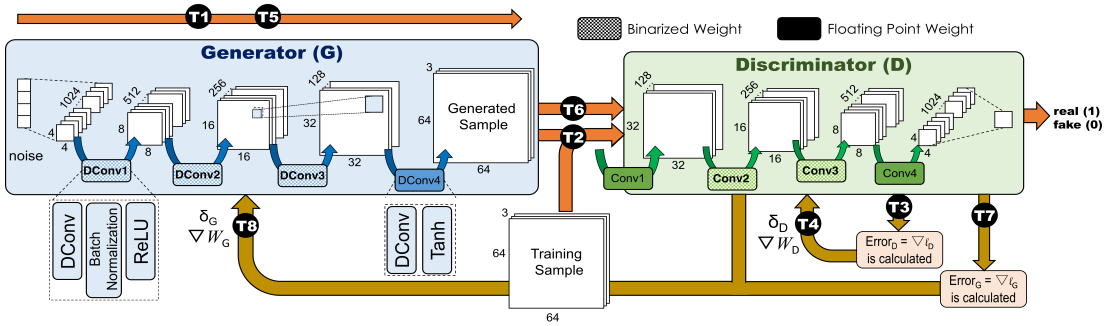

Fig. 2: Approximate GAN system and its training loop from T1 to T8.

3.1 ApGAN Training

Since discriminator units are developed similar to conventional CNNs, all the proposed compression techniques such as quantization and pruning can be applied in the same way. However, due to the deconvolution process in G, local to global mapping instead of the global to local mapping process in D, leveraging these techniques have negative effects on the developed compression methods. On the other hand, as mentioned previously, GAN consumes massive computational power for the training phase, in which two distinct D and G models should be trained separately but simultaneously.

Therefore, to enhance the efficiency of training and facilitate hardware mapping, a novel training approach including partially-quantized layers, i.e. wight binarization, and modification of the loss function presented in [17], is introduced. Herein, both D and G networks are trained using binarized weights (-1, +1), which results in the elimination of the computationally expensive multiplication operations. ApGAN training includes a) forward computation, **computation phase**, and b) backpropagation, **update phase**. After producing a series of fake samples by generator (T1), both real and fake samples are imported into the D network(T2). Next, regarding the output layer of D, the error is calculated based on the gradient of the loss function (T3). Then T4 starts by feeding the error back into D. After passing the error to each layer of D, the weight of D are updated. Updating the G network starts by importing artificial sample (T5) into D (T6). The loss for training G is then computed (T7) and back-propagated to G (T8) to update its weights.

The eight-step training process can be summarized into three main phases, which are operating sequentially in an iterative manner: (I) weight binarization and statistical weight scaling, (II) binary weight-based inference to compute the loss function and (III) back propagation to update full precision weights. In (I), current full precision weights are binarized by only taking the sign function, expressed in Equation 2 and then the corresponding scaling factor will be computed based on the current statistical distribution of full precision weight.

$$\text{Forward:} \quad b = sgn(y) = \begin{cases} +1, & if\, y \geq 0 \\ -1, & otherwise \end{cases} \tag{2}$$

In this case, the sign function is non-convex, which results in the gradient becoming zero. Thus, a standard back-propagation approach will be impractical due to the vanishing gradient problem. Several studies have performed to make the sign function smooth by developing continuation methods such as *softsign* [18] and *appsign* [19], in which the original complex problem is split into several problems that can be optimized easier by reducing the smoothing rate steadily. Herein, due to similar observations between *appsign(.)* and *tanh(.)* functions and also ease of implementation of *tanh* activation function in hardware perspective, Equation 3 is considered in the forward path.

$$appsign(y) = \begin{cases} +1, & if\, y \geq 0 \\ y, & if 1 \geq y \geq -1 \\ -1, & if\, y \leq -1 \end{cases}$$
$$\text{Forward:} \quad sign(y) = \lim_{\beta \to \infty} appsign\,(\beta y) \approx \lim_{\beta \to \infty} \tanh\,(\beta y) \tag{3}$$

In order to achieve a good binary representation, we use the modified Binarized Representation Entropy (BRE) regularization [17] to boost the variety of binary columns in the low-dimensional layer [18]. The BRE is calculated over a mini-batch of X = $\{x_1, ..., x_K\}$ including two terms, marginal entropy (ME), and modified activation correlation (MAC) [17].

$$\ell_{ME} = \frac{1}{d}\sum_{j=1}^{d}\left(\frac{1}{K}\sum_{k=1}^{K}(s_k, j)\right)^2$$
$$\ell_{MAC} = \sum_{j,k=1, j\neq k}^{N} \frac{\alpha_{k,j}}{\sum_{j,k=1,j\neq k}^{N}\alpha_{k,j}} . \frac{|\mathrm{S}_{f,j}^{T} \,.\, \mathrm{S}_{f,k}|}{d} \tag{4}$$

where s_k is the activation vector of $x \in X$, while the large parenthesis denotes the average of jth element of the s_k. Letter $\alpha_{k,j}$ are weights regarding $\mathrm{S}_{f,j}^{T}$. $\mathrm{S}_{f,k}$ pairs, and the sum in the denominator is defined as a normalization constant. Therefore in (II), the input mini-batch takes the binarized model for inference and the loss function of the discriminator will be calculated, which can be expressed as follow:

$$L = \lambda_1.\ell_D + \lambda_2.\ell_{ME} + \lambda_3.\ell_{MAC} \tag{5}$$

where, ℓ_D as adversarial loss is computed by Equation 1 and $\boldsymbol{\lambda}$s (λ_1-λ_3) are regularization constants. Whereas training D is performed by Equation 5, the G model is trained by $\ell_G = ||\mathrm{E}_{x\sim p_{data}(x)}\mathrm{f}(x) - \mathrm{E}_{x\sim p_z(z)}\mathrm{f}(G(z))||_2^2$, where the intermediate layer of D, penultimate layer, defines f(x). In (III), the weights will be updated during back-propagation and stochastic gradient descent is utilized to minimize the loss.

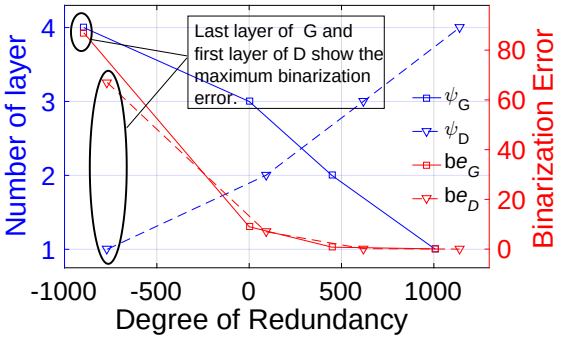

Fig. 3: Number of layers and binarization error (be) w.r.t degree of redundancy (ψ).

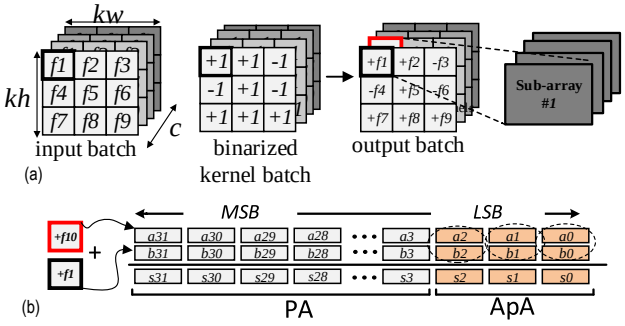

Fig. 4: (a) ApGAN's binary convolution, and (b) partial approximate computing on three LSBs.

The next iteration starts to recompute the weight scaling factor and binarize weights as described in step ①.

To realize the possible layers to be quantized, in both G and D networks, *degree of redundancy* parameter [20], $(\psi) \approx (c_i - h_i w_i)$, is utilized. This term is defined and computed based on the input matrix dimension, where c_i is the number of channels, h_i and w_i are the number of height and width, respectively. It has been proven that the deconvolution layer with the negative value of ψ, which indicates that the dimension of the input space is lower than the dimension of the output space, is more susceptible to binarization errors. The obtained results regarding ApGAN, as shown in Fig. 3, depict deeper layers, i.e. layer 1 (4) in D (G), generate the lowest values for degree of redundancy, means biggest negative number, which causes the maximum binarization errors. As a result, the shallower layers, layers with a higher degree of redundancy, will be binarized. To avoid further accuracy degradation in our ApGAN, all the deconvolution layers in G except the last one and all the convolution layers in D except the first and last layers[1] are quantized.

3.2 Partial Approximate Computing Unit

Approximate computing paradigms can improve metrics such as energy, delay, and area at the cost of lower accuracy [21]. However, the technique needs to be applied judiciously to avoid unacceptable error in output behaviors. Nowadays, approximate computing paradigms have been studied extensively to improve the performance efficiency of systems such as energy and area reduction at the cost of lower accuracy.

Figure 4(a) depicts the simplified computation of ApGAN's binarized convolutional layers. Initially, c channels (herein $c = 4$) in the size of $kh \times kw$ (herein 3×3 has been used) are selected from input batch and accordingly generates a combined batch w.r.t. the corresponding {-1,+1} kernel batch. The combined batch is then mapped to the designated computational sub-arrays of ApGAN accelerator (detailed in Section 4). After this step, the main computation is to perform full-precision addition/subtraction between 32-bit output feature maps. Since, implementation of the whole design using approximate adders results in large errors in outputs, herein, a partial approximate computing unit consisting of a Precise Adder (PA) and an Approximate Adder (ApA) is developed. As shown in Fig. 4(b), the PA and ApA are used for the most significant bits (MSBs) and the least significant bits (LSBs), respectively, in a manner to maximize the accuracy and minimize energy consumption. In order to find the optimal number of LSBs for ApA, regarding accuracy and energy trade-off, PyTorch implementation of ApGAN inspired by BGAN [19] and BRE regularization [17] method combined with depthwise separable convolution is developed and evaluated.

1. These layers are kept in floating point, un-binarized, format.

4 ApGAN Accelerator

4.1 Architecture

In order to address data transfer and computation limitations of various GAN architectures, we develop an in-memory accelerator for approximate GAN, based on memristive computational sub-array. In comparison to well-trained GANs using floating point operations on CPUs and GPUs, ApGAN has the least computational complexity on the underlying hardware, due to the binarization of weights in the forward path. The proposed accelerator can execute the entire GAN training step discussed in the previous sections and the forward path of training in both discriminator and generator units is focused upon herein. The architecture of ApGAN accelerator is shown in Fig. 5(a). It includes Image and Kernel sub-arrays, distributed across the memory banks, which are storing the original values of input feature-maps and weights, respectively. It also contains the memristive computational sub-arrays and an External Processing Unit (EPU) with five computational components (i.e. Binarizer, Activation Function, Batch Normalization, Loss Functions 1 and 2). Mathematically, a $DConv$ can be implemented with a direct $Conv$ [7]. This step is achieved by adding zeros, using zero padding between inputs in the feature maps, and then computing the convolution phase between the kernels and extended input feature maps. Since, in the forward path the binarized weights are utilized, all the $DConv$ and $Conv$ operations are converted to subtraction/addition (sub/add). Here, we give an overview of ApGAN accelerator's functionality. Initially, for each ApGAN layer, c channels in the size of $kh \times kw$ are selected from input batch and accordingly produce a combined batch to which is the corresponding binary {-1,+1} kernel processing ❶ performed by the EPU's

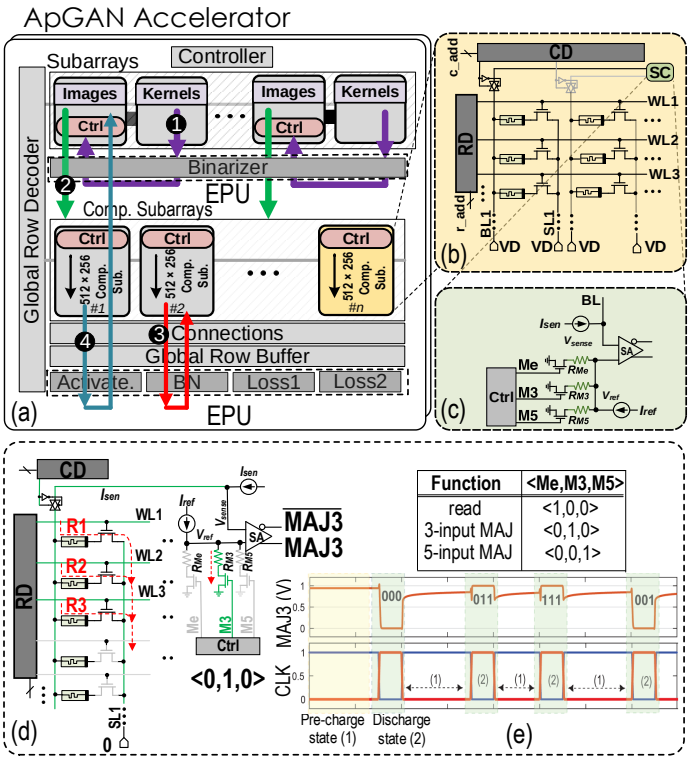

Fig. 5: (a) The *ApGAN* accelerator, (b) memristive computational sub-array architecture, (c) configurable memory sense amplifier, (d) 3-input majority functions realization using resistive references, and (e) MAJ3's transient response for four different inputs.

binarizer. This step is readily accomplished by changing the sign-bit of input data w.r.t. kernel data. After this step, the channels of a combined batch are transposed and mapped to the designated computational sub-arrays of ApGAN ❷. The presented computational array architecture can support massively-parallel and flexible bit-width add/sub operations required in forward path of ApGAN's training as elaborated in the next part. After parallel processing over combined batches, EPU's shared components are employed to process ❸ the batches (i.e. calculating the losses, etc.) and eventually generate output feature-maps ❹ required for next layer.

4.2 Resistive Computational Sub-array

The memristive sub-array architecture is shown in Fig. 5(b). This architecture includes one modified Row Decoder (*RD*), Column Decoder (*CD*), and Sense Circuitry (*SC*). *SC* includes one configurable sense amplifier per bit-line to maximize the throughput (Fig. 5(c)), and can be adjusted by *Ctrl* unit to morph between write operation and 3 possible read-based in-memory operations. *Write* is accomplished by activating the corresponding Word-Line (WL) using *RD* and then applying the differential voltage to the corresponding Bit-Line (BL) and Source-Line (SL) by voltage driver leading to a change in memristor resistivity to either High-R_H (/Low-R_L). *Read* operation is performed by activating the corresponding WL. The corresponding BL activated through *CD* is connected to the *SC*. The *SC*'s sense amplifier generates a read current passing through the resistive device to the grounded SL to generate a sense voltage (V_{sen}), which is then compared with memory reference voltage activated by Me signal ($V_{sen,Low} < V_{Me} < V_{sen,High}$). Accordingly, the sense amplifier outputs Low-'0' (/High-'1') voltage if the path resistance is lower (/higher) than R_{Me}, memory reference resistance.

We propose to extend the existing *SC* unit only by adding two low-overhead reference resistances per sense amplifier to enable required in-memory computing within ApGAN's sub-arrays. The proposed configurable memory sense amplifier (Fig. 5(c)) now consists of three reference-resistance branches that can be selected by control bits (Me, $M3$, $M5$) by the sub-array's *Ctrl* to carry out one-threshold memory, 3-input (MAJ3), and 5-input (MAJ5) majority functions and their complement in a single memory cycle, respectively. To perform such in-memory computation, every three (/five) resistive cells located in the same bit-line could be activated by *RD* and sensed to implement MAJ3/MAJ5. To realize MAJ3 operation, as shown in Fig. 5(d), R_{M3} is set between $R_L//R_L//R_H$ ('0','0','1') and $R_L//R_H//R_H$ ('0','1', '1'). For MAJ5, such reference is set between $R_L//R_L//\ R_L//R_H//R_H$ ('0','0','0','1','1') and $R_L//R_L//R_H//R_H//R_H$ ('0','0','1', '1','1'). Now, parallel resistances of selected three (/five) cells will be compared with the corresponding reference resistances to produce desired output.

4.3 Configurable In-Memory Addition Scheme

As the main operation of ApGAN, add/sub is widely used to process most iterative layers which consume the vast majority of the run-time in the network. Therefore, we present a parallel in-memory computation and mapping method for add/sub based on ApGAN's resistive computational sub-arrays to accelerate multi-bit operations. A close observation on Full-Adder (FA) truth table clarifies that an approximate FA (25%-ER on Sum) could be implemented through making approximate sum like $Sum_{app} = \overline{C_{out}}$. Based on this, a streamlined and cost-effective approximate in-memory FA circuit can be designed by storing three input operands (R_i, R_j, R_k) as resistances in the same memory bit-line and then using the MAJ3 scheme (Fig. 5(d)). The C_{out} and Sum_{app} of such adder are generated through $MAJ3(R_i, R_j, R_k)$ and $\overline{MAJ3(R_i, R_j, R_k)}$, respectively, in a single memory cycle. Moreover, the accurate sum (Sum_{Acc}) can be carried out through $MAJ5(R_i, R_j, R_k, \overline{C_{out}}, \overline{C_{out}})$ with only writing back the $\overline{C_{out}}$ into memory and then applying MAJ5 scheme. In addition to transient response for MAJ3, Fig. 5(e) illustrates all possible functional modes.

4.4 Instructions

While ApGAN is designed to be an independent energy-efficient and high-performance accelerator, we need to expose it to programmers and system-level libraries to use it. From a programmer perspective, ApGAN is a third party accelerator that can be connected directly to the memory bus or through PCI-Express lanes rather than a memory unit, thus it is integrated similar to that of GPUs. Therefore, a virtual machine and ISA for general-purpose parallel thread execution need to be defined similar to PTX [22] for NVIDIA. In this way, the programs will be translated to the

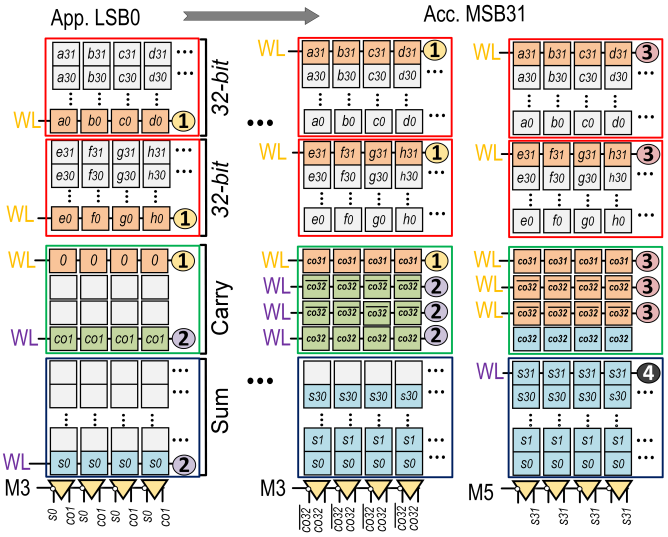

Fig. 6: Mapping and parallel in-memory addition within the resistive computational sub-array of ApGAN.

ApGAN hardware instruction set at install time. ApGAN basically supports three main instructions of in-memory copy (consecutive read and write), MAJ3 and MAJ5. The in-memory copy takes two operands corresponding to destination and source row addresses. MAJ3 and MAJ5 takes the address of input operands and write back the result on a destination row. Such instructions is directly copied/written to a predefined memory-mapped address ranges, for example, in the memory type range registers (MTRRs), or by programming to Memory-Mapped I/O regions that are allocated through a simple device driver to do initialization/cleanup for required software memory structures. We allotted the subsection 4.4 to the aforementioned explanation as highlighted in the manuscript.

4.5 Hardware Mapping

Figure 6 elaborates the required data organization and computation steps of ApGAN with a straightforward and intuitive example only considering the add operation. Clearly, sub can be implemented based on add.

Considering n-activated sub-arrays with the size of $x \times y$, each sub-array can handle the parallel add/sub of up to x elements of m-bit ($3m + 4 \leq y$) and so ApGAN could process $n \times x$ elements simultaneously within computational sub-arrays to maximize the throughput. After the mapping step ❷ shown in Fig. 5(a), the parallel in-memory adder of ApGAN accelerator operates to produce the output feature maps. The memory sub-array organization for such parallel computation is delineated in Fig. 6. Four reserved rows for Carry results initialized by zero and 32 reserved rows are considered for Sum results. Every pair of corresponding elements to be added together have to be aligned in the same bit-line. Herein, channel 1 (Ch1) and Ch2 should be aligned in the same sub-array. Ch1 elements occupy the first 32 rows of the sub-array followed by Ch2 in the next 32 rows.

The addition algorithm starts bit-by-bit from the LSBs of the two words and continues towards MSBs. We consider approximate computation for LSBs and accurate computation for MSBs based on conclusion drawn from algorithm-level evaluations in Section 3.2. Figure 6 L.H.S. shows App. LSB computation. There are 2 cycles for every bit-position to perform such computation. In step one (❶ in Fig. 6), two WLs (accessing to LSBs of elements) and one reserved carry row are enabled to generate C_{out} and Sum_{app} in parallel for whole memory sub-array with $Ctrl$'s M3 command. During step ❷, two WLs are activated to save back the results to the designated locations. This carry-out bit overwrites the data in the carry latch and becomes the carry-in of the next cycle. This process is concluded after $2 \times m$ cycles, where m is a number of bits in its elements. Figure 6 R.H.S. shows an Acc. MSB computation as a 4-cycle operation. In step ❶, two WLs (accessing to LSBs of elements) and one reserved carry row are enabled to generate C_{out} in parallel. During step ❷, three WLs are activated to store back the results of C_{out} and $\overline{C_{out}}$ to the designated locations. Now, five WLs are selected (step ❸) to generate the Sum_{acc} with $Ctrl$'s M5 command and write it back (step ❹) to the sub-array. The Acc. MCB computation is concluded after $4 \times m$ cycles, where m is a number of bits in its elements.

4.6 Parallelism

Here, we design a Fully-Pipelined Computation mechanism named *FPC* on top of the presented spatial parallelism (*SP*) method in [7] to boost ApGAN performance. The input data are usually processed in 8/32/64 batch size-**b** during the training phase. For the sake of simplicity, Figure 7 depicts *FPC* method with a batch size of 2. Obviously, regardless of pipelining, GAN training takes b×(D1+D2+G) cycles. Typically, if all inputs in the prior batch are processed, a new batch can come into the pipeline. The key idea behind *FPC* is to duplicate the data for intermediate layers such that pipelining can be readily achieved in ApGAN. Fig. 7 shows such pipeline for**b1** and **b2**. Consider DL as the discriminator's layers, D1 needs DL+1+DL+(b-1) cycles, where b-1 cycles is needed for draining a batch from a pipeline. The (*SP*) method [7] proves that for each input batch, as there is no data-dependency between the training phases of discriminator, they can perform simultaneously. We exploit the *SP* method in *FPC*, as shown Figure 7; D1 and D2 training phases occupy different computational sub-arrays and both $Conv$ and $DConv$ layers can be run at a same time. Consider GL as the generator's layers, D2 takes GL+DL+1+DL+1+(b-1) latency for updating the D. Besides, *FPC* takes advantage of this observation that after D2's loss function computation and back-propagation to GD4 layer, the training of generator for different batches can be started while the corresponding GD3 is being processed in D2. This phase takes 2DL+2GL+2+(b-1).

5 Performance Evaluation

5.1 Experimental Setup and Results

In order to perform a fair comparison between our design and the well-known GAN models, DCGAN, WGAN-CP, and WGAN-GP, the same architecture including four convolution and deconvolution layers for D and G, respectively, is leveraged.

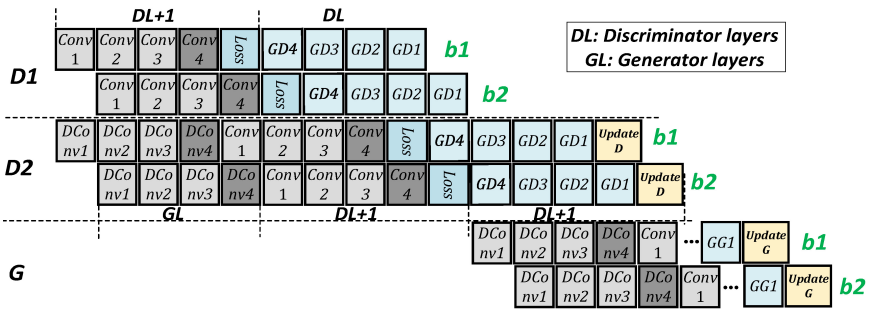

Fig. 7: Fully-paralleled training method for ApGAN.

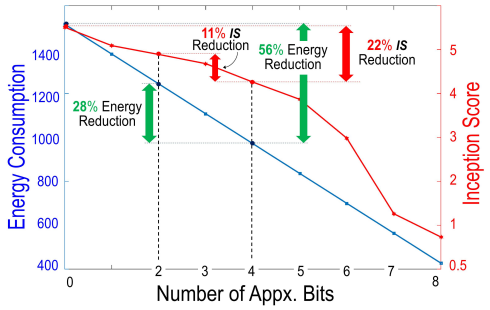

Fig. 8: Energy consumption vs. *IS* regarding number of approximated bits.

Datasets: We conduct experiments of ApGAN on several datasets to evaluate the performance of the proposed algorithm, including MNIST [23], Fashion-MNIST [24], CIFAR-10 [25], STL-10 [26], and celeb-A [27]. MNIST is leveraged as a gray-scale dataset which contains 70,000 28×28 images of handwritten digits from 0 to 9, 60,000 images for training and 10,000 images for testing sets. Similar to MNIST, Fashion-MNIST consists of 28×28 gray-scale images but it includes 10,000 images for each of training and testing sets to form ten fashion categories. We use CIFAR-10 for RGB images of size 32×32. It has 60,000 images evenly distributed in ten distinct classes, in which 50,000 and 10,000 examples are used for training and testing, respectively. In addition to CIFAR-10, STL-10 is used, which is similar to CIFAR-10 dataset except that it has 100,000 unlabeled images for unsupervised learning and only 500 labeled images for training. Finally, we also exploit celeb-A to evaluate performance quantitatively. It includes 202,559 images of celebrity faces labeled with 40 different face attributes and because each image consists of only one face, the quality of the generated images is readily evaluated.

Evaluation Metrics: According to [30], which includes extensive studies for highly-used metrics i.e. log-likelihood to evaluate the performance of NN models, authors showed there is not necessarily a direct relationship between the good performance of GANs and the metric(s). Therefore herein, we use Inception Score (IS) [31] as an evaluation metric in our experiments, which is leveraged to measure information on the quality and variation of the generated images by using a pre-trained inception V3 [32] network. The IS's of generators is calculated by

$$IS_G = exp\big(E_{x\sim P_g} D_{KL}(p(y|x)||p(y))\big) \quad (6)$$

where x is an image, y is the output label which will be predicted, and $D_{KL}(p|q)$ is the KL divergence between two distributions, p and q. A high IS[2] illustrates diversity and clarity among generated images and it is achieved if $p(y|x)$ is low entropy, means that the generated image includes clear objects, and $p(y)$ is high entropy, which indicates a high diversity of images from all categories.

Results and Analysis: Herein, several sets of experiments on both CIFAR-10 and STL-10 using DCGAN, WGAN-CP, and WGAN-GP are conducted. First, GAN networks are trained using 32-bit floating point number weights as the baseline. Next, several variant GANs are trained from scratch. Since the GAN training phase usually suffers from training instability and convergence problems, the change of IS is monitored after each epoch, which helps us to observe the stability of the proposed method compared to the full precision models.

Figure 8 depicts the IS results for ApGAN on CIFAR-10 with respect to the number of approximated LSBs, and energy consumption of the convolution layers. The optimal condition occurs when 2 to 4 LSBs are approximated, which leads to a relatively high reduction in energy whereas IS is slightly decreased. Table 1 summarizes ISs of DCGAN, WGAN-CP, and WGAN-GP on CIFAR-10 and STL-

2. The higher score is better.

TABLE 1: IS values on CIFAR-10 and STL10 Datasets.

Model		CIFAR-10	STL-10
DCGAN	32-bit	5.46±0.2	2.93±0.2
	DoReFa-Net [2]	1.2±0.003	1.39±0.007
	TWN [28]	1.09±0.003	1.45±0.008
	TGAN [29]	4.52±0.1	2.91±0.3
	ApGAN	**5.01±0.08**	**2.46±0.07**
WGAN-CP	32-bit	4.69±0.15	3.13±0.1
	DoReFa-Net [2]	3.84±0.09	2.37±0.05
	TWN [28]	4.26±0.07	2.78±0.06
	TGAN [29]	3.76±0.07	2.31±0.09
	ApGAN	**4.46±0.15**	**2.93±0.1**
WGAN-GP	32-bit	5.51±0.008	3.04±0.09
	DoReFa-Net [2]	4.70±0.05	2.31±0.012
	TWN [28]	4.45±0.05	2.68±0.015
	TGAN [29]	4.98±0.01	2.81±0.05
	ApGAN	**5.08±0.05**	**2.61±0.09**

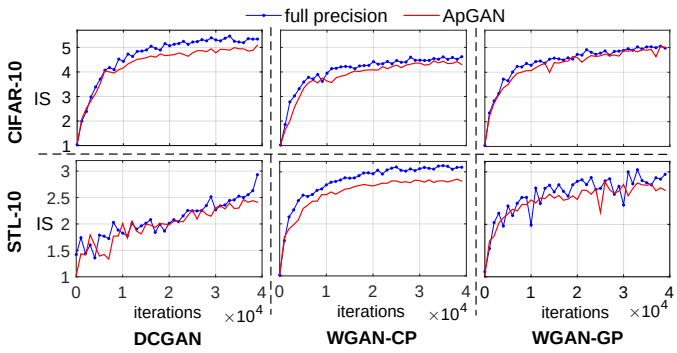

Fig. 9: Inception score on CIFAR-10 and STL-10 datasets leveraging full precision and ApGAN for different GANs.

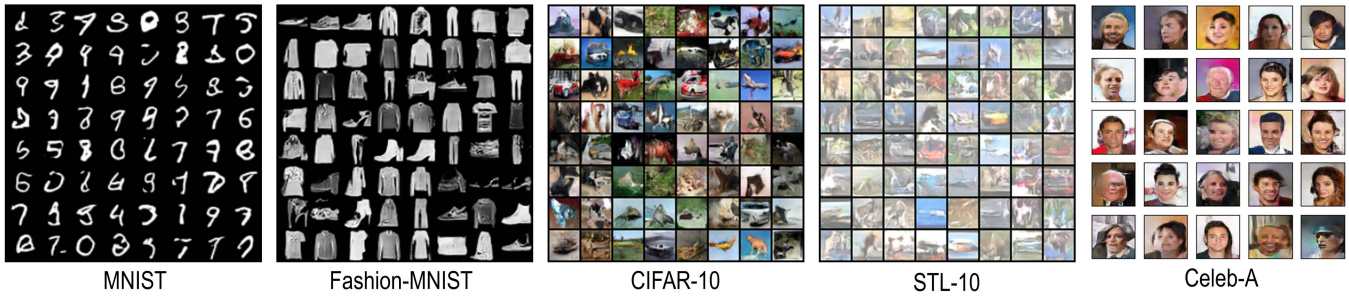

Fig. 10: Generated images for various datasets by ApGAN.

10 datasets. In addition to the 32-bit full-precision as the baseline, ApGAN and three other GANs including ternarized and binarized -weight training are examined. Based on the obtained results, the full precision WGAN-GP and WGAN-CP show the best ISs for CIFAR-10 (5.51) and STL-10 (3.13) datasets, respectively. Although the IS of our proposed ApGAN degrades roughly by 0.37 (in both examined datasets) compared to the best results, it shows better scores than 32-bit WGAN-CP and almost all of the proposed fully-quantized training approaches. Moreover, the training convergence behaviors for all the examined GANs are shown in Fig. 9. Although the baseline full-precision training has a faster convergence, our ApGAN achieves comparable IS results for CIFAR-10 and STL-10.

In addition to the quantitative comparison, Fig. 10 depicts the generated images by ApGAN architectures for five different datasets as qualitative evidence. The generated images which look similar to the full-precision DCGAN's results verify the performance and functionality of ApGAN. Figure 11 depicts loss values for both discriminator (D) and generator (G) networks in full precision, fully-binarized and ApGAN in Celeb-A dataset. The **y** and **x** axes indicate the loss values and the number of epochs, respectively. As depicted in the fully-binarized network shown in Fig.11 (b), after a few epochs for initializing and competition steps, the convergence process and consequently improvement in the generated images stop. Nonetheless, for ApGAN, after initial state, competition starts quickly to improves the quality of the generated images and due to the semi-balanced binarized structures for D and G, the competition continues for a sufficient number of epochs. The ApGAN actually converges in an almost similar manner as the original 32-bit full-precision training.

5.2 Hardware Setup and Results

In this section, we estimate ApGAN's energy-efficiency and performance and compare it with other feasible GAN accelerators (based on ASIC, SOT-MRAM, ReRAM, and GPU) based on three GAN architectures (DCGAN, WGAN-CP, and WGAN-GP). It is clear that the larger chip area is, then the higher performance for ApGAN and other accelerators are achieved due to having additional sub-arrays or computational units, albeit the memory die size impacts the area cost. To have a fair comparison in this work, we report the area-normalized results (performance/energy per area), henceforth.

Experiment Setup: To assess the performance of the proposed accelerator at the circuit-level, we use the SPICE model for memristors with the Ag-Si memristor device parameters from [33]. We then combine the SPICE models of CMOS transistors and memristors under NCSU 45nm CMOS PDK [34]. To perform the system-level evaluations, we modified the memory evaluation tool NVSim [35] to co-simulate with our developed in-house C++ code based

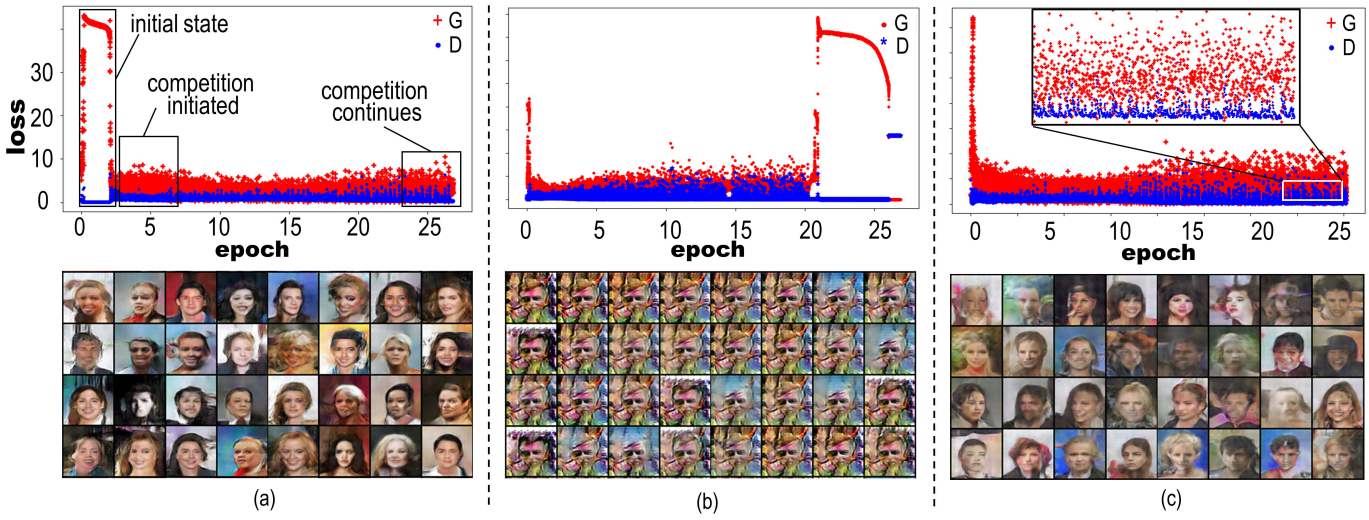

Fig. 11: Value of losses in (a) 32-bit (full precision) DCGAN, (b) fully-binarized DCGAN, and (c) proposed ApGAN.

on circuit-level results. We configure the memory organization of the sub-arrays with 512 rows and 256 columns per memory matrix (mat) considering an H-tree routing method, 2×2 mats per bank, 8×8 banks per group; in total 16 groups leading to a 512Mb total capacity. For comparison, a ReRAM-based in-memory accelerator based on [7] was developed with 256 fully-functional sub-arrays with the size of 256×256 and eight-bit configurable SAs. To perform the evaluations, NVSim was extensively modified to estimate the system energy and performance adopting its default ReRAM cell file (.cell). We developed a SOT-MRAM-based accelerator based on PIM-TGAN [29]. For the circuit level simulation, a Verilog-A model of 2T1R SOT-MRAM device is developed to co-simulate with the interface CMOS circuits in SPICE. Finally, an architectural-level simulator was built on top of NVSim. To compare the result with ASIC accelerators, we developed a YodaNN-like [36] design with two 8×8 tiles configuration. Then, the designs were synthesized using Design Compiler [37] with 45nm technology. The SRAM and eDRAM performances were calculated using CACTI [38]. We created a comprehensive Verilog model for EPU to interact with our SPICE circuit code to perform the evaluation. Activation functions were developed based on lookup-table-based transformations [39] with case-statement codes. Batch normalization unit generally performs an affine function ($y = kx + h$) [40], where y and x represent the corresponding output and input feature map pixels, respectively. During inference mode, all the other parameters (k,h) are pre-computed and stored in ApGAN sub-arrays, therefore, Batch normalization unit can readily fetch each pixel of input feature map, fed forward to the batch-norm layer, and write back the corresponding normalized pixel employing an internal, multiplexed CMOS adder and multiplier to perform this computation efficiently.

Energy Efficiency: Figure 12 shows ApGAN's energy-efficiency (frames per joule) results implemented by FPC method for three possible approximation degree (i.e., 2-, 3-, and 4-bit) compared with other designs, running a similar task under two batch size configurations, i.e. 8 and 32. Here, as the batch size gets larger, higher energy-efficiency is obtained. We can see that ApGAN-4b has the highest energy-efficiency normalized to the area, related to other methods, as a result of its 4-bit approximated, parallel, energy-efficient operations. ApGAN-3b shows ∼2.5×, 13.1×, and 28.6× higher energy-efficiency than that of the leading ASIC, ReRAM, and GPU-based solutions. This energy reduction arises from three sources: 1) standard $Conv$ and $DeConv$ operations in the forward path are replaced with energy-efficient add/sub operations due to binarization, 2) ApGAN's inter-layer parallelism which massively reduces the latency of operations and 3) bulk and energy-efficient approximated in-memory operations of ApGAN. Compared to the recent processing-in-MRAM platform in [29], ApGAN reduces energy consumption by ∼2.3×.

Throughput: Figure 13 compares the ApGAN throughput (frames per second) results for three possible approximation degree (i.e., 2-, 3-, and 4-bit), normalized with the area, for different accelerators. Based on the results, ApGAN-3b is 35× and 5.8× faster on average than GPU and ASIC-64 methods. This efficiency can be related to parallel and ultra-fast in-memory operations of ApGAN compared

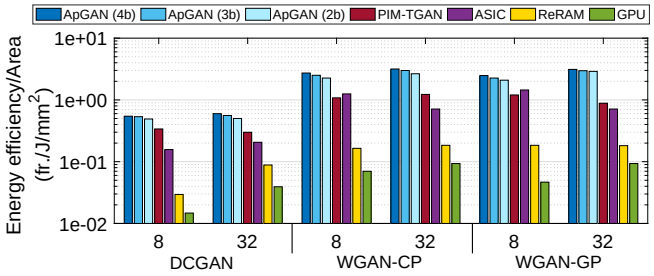

Fig. 12: Energy-efficiency evaluation of various platforms normalized to the area (Y-axis: log scale).

to multi-cycle ASIC and GPU operations as well as the potential mismatch between data movement and computation in ASIC and GPU methods. Additionally, ApGAN is 1.9× faster than ReRAM method. It is worth pointing out that ReRAM accelerators suffer matrix splitting owning to intrinsically-limited bit levels of ReRAM device, thus more sub-arrays need to be occupied. This can further limit parallelism methods. Additionally, a ReRAM crossbar imposes a large peripheral circuit overhead due to existing DAC/ADC and buffers occupying roughly 85% of area [4], [41]. We also observe that ApGAN achieves ∼ 40% better performance compared to that of PIM-TGAN platform [29].

Area Overhead: To assess the area overhead of ApGAN on top of commodity RRAM chip, three main hardware cost sources must be taken into consideration as shown in Fig. 14(a). First, add-on transistors to SAs; in our design, each SA requires 2 additional transistors connected to each BL (Fig. 5(c)) to enable in-memory computing; Second, the modified MRD overhead; we modify each WL driver by adding two more transistors in the typical buffer chain based on the method used in [42]. Third, the ctrl's overhead to control enable bits; ctrl generates the activation bits with MUX units with 6 transistors. To sum it up, ApGAN roughly imposes 3 additional rows per sub-array, which can be interpreted as ∼ 2% of memory chip area. The detailed breakdown of area overhead is shown in Fig. 14(b).

Resource Utilization: We estimated the time fraction at which the computation has to wait for data and on-/off-chip data transfer limits the performance referred to as memory bottleneck ratio for different platforms, as depicted in Fig. 15. This evaluation is done through the peak performance and experimentally extracted results for each platform considering a number of memory access. We observe that

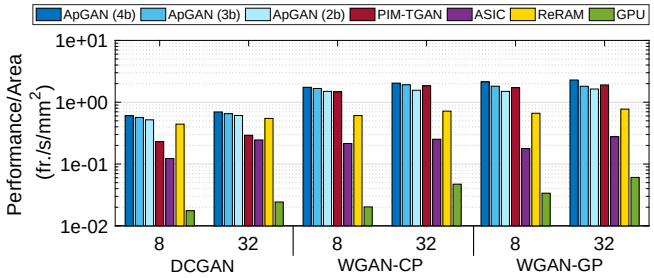

Fig. 13: Performance evaluation of various platforms normalized to the area (Y-axis: log scale).

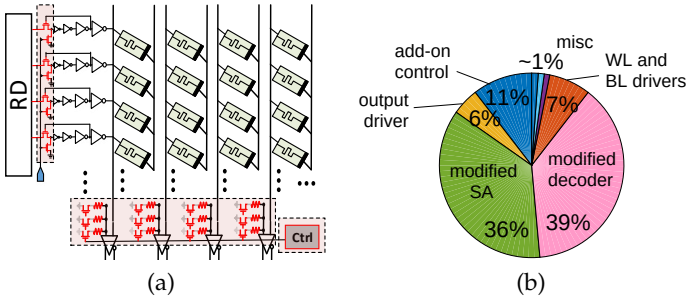

Fig. 14: (a) Three main hardware cost sources in ApGAN's sub-array. Note: Access transistors and CD are not shown for simplicity, and (b) Area overhead breakdown of ApGAN.

processing-in-memory solutions i.e. ApGAN, PIM-TGAN, and ReRAM spend less than 30% time for data transfer and memory access. But, ASIC and GPU spend over 50% and 90% time, respectively, waiting for the loading data from the memory. In this way, we can define a resource utilization ratio for different platforms. We observe that ApGAN-4b achieves the highest ratio by efficiently utilizing up to 88% of its computation resources. This number is limited to 5% for GPUs performing the similar task.

6 Conclusion

In this paper, we presented a novel hardware-optimized GAN training algorithm using binary weights for three and two layers of generator and discriminator networks, respectively. Moreover, we developed a reconfigurable addition approach in which both approximate and accurate add operations are performed. In order to further accelerate the ApGAN training process, a new PIM accelerator based on memristor was implemented. Finally, in addition to focus on the computational performance of ApGAN exploiting its intrinsic in-memory parallelism to increase the throughput of the system, we developed FPC optimization as a spatial parallelism method. The performance of the ApGAN in both quantitative and qualitative approaches have been evaluated on different data-sets including Fashion-MNIST, CIFAR-10, STL-10, and celeb-A. The generated images by ApGAN look similar to the full-precision DCGAN's result. Moreover, the obtained simulation results showed that our PIM-ApGAN can achieve ∼2.5× better energy-efficiency and 5.1× speedup compared to CMOS-ASIC accelerator, whereas IS is degraded by 11%. Hence, due to the small IS degradation and a significant reduction in the hardware aspect, the ApGAN can be a promising weight training scheme for resource-limited IoT devices. Since in an environment tens to hundreds of IoT nodes are distributed, similar approaches which are used in random forest methods, majority voters and mean prediction methods, can be leveraged.

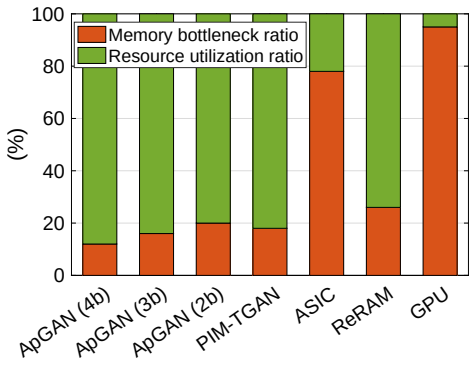

Fig. 15: Memory bottleneck ratio for different platforms.

Acknowledgments

We would like to acknowledge and thank the Advanced Research Computing Center at the University of Central Florida for provision of computing resources used herein.

This work was supported in part by the Center for Probabilistic Spin Logic for Low-Energy Boolean and Non-Boolean Computing (CAPSL), one of the Nanoelectronic Computing Research (nCORE) Centers as task 2759.006, a Semiconductor Research Corporation (SRC) program sponsored by the NSF through CCF-1739635.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems 25 (NIPS)*, 2012, pp. 1097–1105.

[2] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[3] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*. Springer, 2016, pp. 525–542.

[4] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*. IEEE Press, 2016, pp. 27–39.

[5] M. Imani, Y. Kim, and T. Rosing, "Mpim: Multi-purpose in-memory processing using configurable resistive memory," in *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 757–763.

[6] B. Li, L. Song, F. Chen, X. Qian, Y. Chen, and H. H. Li, "Reram-based accelerator for deep learning," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, March 2018, pp. 815–820.

[7] F. Chen, L. Song, and Y. Chen, "Regan: A pipelined reram-based accelerator for generative adversarial networks," in *Proceedings of the 23rd ASP-DAC*. IEEE, 2018, pp. 178–183.

[8] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proceedings of the 55th DAC*. ACM, 2018, p. 105.

[9] A. Roohi and R. F. DeMara, "Nv-clustering: Normally-off computing using non-volatile datapaths," *IEEE Transactions on Computers*, vol. 67, no. 7, pp. 949–959, 2018.

[10] G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. N. Kurdi, and H. Hwang, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3498–3507, November 2015.

[11] I. J. Goodfellow, J. Pouget-Abadie, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27 (NIPS)*, 2014, pp. 2672–2680.

[12] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[13] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in neural information processing systems 29 (NIPS)*, 2016, pp. 2234–2242.

[14] M. Arjovsky, C. Soumith, and B. Leon, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.
[15] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017, pp. 5767–5777.
[16] M. Song, Z. Jiaqi, C. Huixiang, and L. Tao, "Towards efficient microarchitectural design for accelerating unsupervised gan-based deep learning," in *HPCA*, Feb 2018, pp. 66–77.
[17] M. Zieba, P. Semberecki, T. El-Gaaly, and T. Trzcinski, "Bingan: Learning compact binary descriptors with a regularized gan," *Advances in Neural Information Processing Systems 31 (NIPS)*, 2018.
[18] Y. Cao, W. D. Gavin, K. Y.-C. Lui, and R. Huang, "Improving gan training via binarized representation entropy (BRE) regularization," *arXiv preprint arXiv:1805.03644*, 2018.
[19] J. Song, T. He, L. Gao, X. Xu, A. Hanjalic, and H. T. Shen, "Binary generative adversarial networks for image retrieval," in *Proceedings of Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
[20] J. Liu, J. Zhang, Y. Ding, X. Xu, M. Jiang, and Y. Shi, "Pbgan: Partial binarization of deconvolution based generators," *arXiv preprint arXiv:1802.09153*, 2018.
[21] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE TCAD*, vol. 32, no. 1, pp. 124–137, January 2013.
[22] (2018) Parallel thread execution isa version 6.1. [Online]. Available: http://docs.nvidia.com/cuda/parallel-thread-execution/index.html
[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[24] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
[25] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, no. 7, 2010.
[26] A. Coates, H. Lee, and A. Y. Ng, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215–223.
[27] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of the IEEE ICCV*, 2015, pp. 3730–3738.
[28] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv: 1605.04711*, 2016.
[29] A. S. Rakin, S. Angizi, Z. He, and D. Fan, "Pim-tgan: A processing-in-memory accelerator for ternary generative adversarial networks," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, Oct 2018, pp. 266–273.
[30] L. Theis, A. v. d. Oord, and M. Bethge, "A note on the evaluation of generative models," *arXiv preprint arXiv:1511.01844*, 2015.
[31] S. Barratt and R. Sharma, "A note on the inception score," *arXiv preprint arXiv:1801.01973*, 2018.
[32] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and W. Zbigniew, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE CVPR*, 2016, pp. 2818–2826.
[33] L. Gao, F. Alibart, and D. B. Strukov, "Analog-input analog-weight dot-product operation with ag/a-si/pt memristive devices," in *Proceedings of the IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC)*. IEEE, 2012, pp. 88–93.
[34] (2011) Ncsu eda freepdk45. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents
[35] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*, no. 7, pp. 994–1007, July 2012.
[36] R. Andri, L. Cavigelli, D. Rossiand, and L. Benini, "Yodann: An ultra-low power convolutional neural network accelerator based on binary weights," in *Proceedings of the 2016 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2016, pp. 236–241.
[37] Synopsys, Inc., "Synopsys design compiler, product version 14.9.2014," 2014.
[38] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, March 2012, pp. 33–38.
[39] M. Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," *IEE Proceedings-Computers and Digital Techniques*, vol. 150, no. 6, pp. 403–411, 2003.
[40] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 15–24.
[41] S. Angizi, Z. He, F. Parveen, and D. Fan, "IMCE: energy-efficient bit-wise in-memory convolution engine for deep neural network," in *Proceedings of the 23rd ASP-DAC*. IEEE, January 2018, pp. 111–116.
[42] S. Angizi, Z. He, N. Bagherzadeh, and D. Fan, "Design and evaluation of a spintronic in-memory processing platform for nonvolatile data encryption," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1788–1801, 2018.

Arman Roohi (S'14) received his B.Sc. degree in Computer Engineering in 2008 from Shiraz University, Shiraz, Iran. He also received his M.Sc. degree in Computer Architecture at Department of Computer Engineering, Science and Research Branch of Azad University, Tehran, Iran, in 2011. He is completing the Ph.D. degree in Computer Engineering at the University of Central Florida, Orlando, USA. His research interests span neuromorphic computing, deep learning hardware acceleration and deep learning security, reconfigurable and adaptive computer architectures, beyond CMOS computing, with emphasis on QCA and Spintronics.

Shadi Sheikhfaal (S'19) received her B.Sc. degree in computer engineering from Azad University, Ardebil, Iran, in 2012 and her M.Sc. degree in computer engineering and computer systems architecture from Science and Research Branch of Azad University, Tehran, Iran, in 2014. She is currently pursuing her Ph.D. degree in computer engineering at University of Central Florida, Orlando, FL, USA. Her current research interests include biologically inspired computing and spin-based computing.

Shaahin Angizi (S'15) received his B.Sc. in Computer Engineering, Hardware from South Tehran Branch of Azad University, Tehran, Iran in 2012 and his M.Sc. in Computer Engineering, Computer Systems Architecture from Science and Research Branch of Azad University, Tabriz, Iran in 2014. He is currently working toward the Ph.D. degree in Computer Engineering at University of Central Florida, Orlando, USA. His research interests include in-memory computing, deep learning, low power VLSI designs, Spin-based computing and Quantum-dot Cellular Automata (QCA).

Deliang Fan (M'15) received his B.S. degree in Electronic Information Engineering from Zhejiang University, China, in 2010. He received M.S. and Ph.D. degree in Electrical and Computer Engineering from Purdue University, West Lafayette, IN, USA, in 2012 and 2015, respectively. He joined the Department of Electrical and Computer Engineering at University of Central Florida, Orlando, FL, as an Assistant Professor in 2015. His primary research interest lies in Ultra-low Power Brain-inspired (Neuromorphic), Non-Boolean and Boolean Computing Using Emerging Nanoscale Devices like Spin-Transfer Torque Devices and Memristors. His other research interests include nanoscale physics based spintronic device modeling and simulation, low power digital and mixed-signal CMOS circuit design.

Ronald F. DeMara (SM'10) received the Ph.D. degree in Computer Engineering from the University of Southern California in 1992. Since 1993, he has been a full-time faculty member at the University of Central Florida where he is a Professor of Electrical and Computer Engineering, and joint faculty of Computer Science, and has served as Associate Chair, ECE Graduate Coordinator, and Computer Engineering Program Coordinator. His research interests are in adaptive computer architectures with emphasis on reconfigurable and post-CMOS devices, evolvable and intelligent hardware, resilient and energy-aware logic design, and the digitization of STEM education. On these topics, he has completed over 275 publications, 47 funded projects as PI or Co-PI including sponsorship of NSF, NASA, Army, Navy, Air Force, DARPA, and NSA, with one patent granted and one provisional patent. He has completed 48 graduates as Ph.D. dissertation or M.S. thesis advisor and was previously an Associate Engineer at IBM and a Research Scientist at NASA Ames, in total for four years. He is an Associate Editor of IEEE Transactions on Emerging Topics in Computing. He has served as Topical Editor of IEEE Transactions on Computers and as Associate Editor of IEEE Transactions on VLSI Systems, Microprocessors and Microsystems, and as Guest Editor of various Transactions, and serves on various IEEE conference program committees including ISVLSI, NVMSA, SSCI, etc. He received the IEEE Joseph M. Bidenbach Outstanding Engineering Educator Award in 2008.