Memory-Efficient Architecture for Accelerating Generative Networks on FPGA

Shuanglong Liu*, Chenglong Zeng†, Hongxiang Fan*, Ho-Cheung Ng*, Jiuxi Meng*, Zhiqiang Que*, Xinyu Niu‡ and Wayne Luk*
*Dept. of Computing, Imperial College London, London, United Kingdom
{*s.liu13, h.fan17, h.ng16, jiuxi.meng16, z.que, w.luk*}*@imperial.ac.uk*
†School of Microelectronics, Tianjin University, Tianjin, China, *zengchenglong@tju.edu.cn*
‡Corerain Technologies Ltd., Shenzhen, China, *xinyu.niu@corerain.com*

***Abstract*—Generative adversarial networks (GANs) are a class of artificial intelligence algorithms used in unsupervised machine learning, implemented by a system of two neural networks: a generative network (*generator*) and a discriminative network (*discriminator*). These two networks compete with each other to perform better at their respective tasks. The generator is typically a deconvolutional neural network and the discriminator is a convolutional neural network (CNN). Deconvolution performs a fundamentally new type of mathematical operation which differs from convolution. While the FPGA-based CNN accelerators have been widely studied in prior work, the acceleration of deconvolutional networks on FPGA is rarely explored. This paper proposes a novel parametrized deconvolutional architecture based on an FPGA-friendly method, in contrast to the transposed convolution implementation in CPUs and GPUs. Hardware design templates which map this architecture to FPGAs are provided with configurable deconvolutional layer parameters. Furthermore, a memory-efficient architecture with a new tiling method is proposed to accelerate the generator of GANs, by storing all intermediate data in on-chip memories to significantly reduce off-chip data transfers. The performance of the proposed accelerator is evaluated using a variety of GANs on a Xilinx Zynq 706 board, which shows 2.3x higher speed and 8.2x off-chip memory access reduction than an optimized Vanilla FPGA design. Compared to the respective implementations on CPUs and GPUs, the achieved improvements are in the range of 30x-92x in speed over an Intel 8-core i7-950 CPU, and 8x-108x in terms of Performance-per-Watt over an NVIDIA Titan X GPU.**

I. Introduction

Generative adversarial networks (GANs) [1] have recently received an increasing amount of attention and produced promising results on performing human tasks, especially in image generation [2], [3] and video generation [4], [5]. With the capability of learning the underlying distribution of data, these networks are so powerful that they can produce visually impressive new data that resemble the real world data.

A typical GAN consists of two competing neural network models, i.e., the generative model (***generator***) and the discriminative model (***discriminator***). The generator is often to be a deconvolutional neural network [2] and it aims to produce synthetic samples from the same distribution that the training data follow. Meanwhile, the discriminator, which is normally a convolutional neural network, aims to distinguish whether the sample produced by the generator is synthetic or original. The framework of GANs is shown in Figure 1. The generator takes noise as input and tries to fool the discriminator by creating samples as if they come from the real distribution of data. The discriminator receives samples from both the generator and the training data, and uses traditional supervised learning techniques such as CNNs to classify the images as real or fake. The corresponding results can then be back propagated to the generator in order to change its parameters to perform better. As training goes on, the discriminator performs better and better at dividing real and fake samples, so the generator has to produce more realistic samples to deceive the discriminator. This competition improves both networks and finally leads to a generator which can produce indistinguishable samples from the real data. After training, the discriminator can be discarded and the generator can be used for different applications.

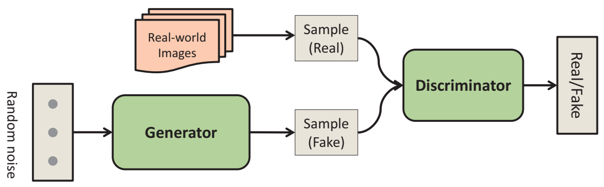


Fig. 1. The framework of generative adversarial networks (GANs).

Despite its popularity in the community of deep learning, hardware accelerators of the generator or deconvolution networks have rarely been explored while FPGA-based CNN accelerators were extensively studied previously. The generator depends on a new type of operator, i.e., deconvolution (DeConv), which performs a fundamentally new type of mathematical operation compared with convolution (Conv). Although DeConv can be implemented as a convolution operator (which is the case in CPU and GPU platforms), this method needs to insert a large number of zeros in its input data. As such, it is inherently inefficient for the FPGA because of the additional unproductive arithmetic operations, and performing multiply-add operations on the inserted zeros results in much less efficient computations [6]. An FPGA-efficient DeConv algorithm is proposed in [7] to avoid inserting zeros. This

The support of the UK EPSRC (EP/I012036/1, EP/L00058X/1, EP/L016796/1 and EP/N031768/1) and the European Union Horizon 2020 Research and Innovation Programme under grant agreement number 671653 is gratefully acknowledged. Shuanglong Liu and Hongxiang Fan are the corresponding authors.

method, however, suffers from an overlapping sum problem [7], [8]. More significantly, FPGA-based accelerators of both CNNs and GANs often require frequent data transfers between on-chip and off-chip memories due to the enormous size of intermediate data and limited memory resource on FPGAs [9], [10]. It imposes a large overhead on both latency and energy consumption. This work focuses on the acceleration of the generator of GANs for fast inference on FPGAs, since the discriminator is discarded after training and only the generator is performed during inference. We tackle both of the challenges mentioned above in this work.

In this paper, a novel and fully customized architecture is proposed to efficiently map the hardware-friendly DeConv method on an FPGA. The overlapping sum problem is dealt with in this architecture by introducing additional hardware blocks with little overhead on resource and latency. Hardware templates are designed to implement this architecture with configurable parameters in Verilog HDL to support different DeConv layers in GANs. A new tiling method together with a memory-efficient architecture is then proposed to accelerate the generator, where all intermediate data are stored in on-chip memory thus eliminating most of the off-chip data transfers. The main contributions of this work are:

1) A deeply customized and optimized DeConv architecture with a crop module to support output shapes of arbitrary size, while the overlapping sum is processed in hardware with small resource overhead; Hardware design templates which can be reconfigured on FPGA for a variety of GANs and applications;

2) A memory-efficient architecture based on the novel tiling method to accelerate the generator for fast inference. The proposed accelerator takes into account the unique data access pattern of DeConv, and therefore achieves higher throughput compared to a vanilla design on FPGA;

3) Evaluation of the hardware accelerators on state-of-the-art GANs with a set of DeConv kernel configurations. The proposed accelerator achieves significant performance improvement compared to the vanilla accelerator and the respective CPU and GPU implementations.

II. Background and Related Work

A. The Generator and Deconvolution

The architecture of the generator is a sequence of deconvolutional layers stacked together, as shown in Figure 2. DeConv is the core and basic unit of the generator and it consumes the majority of the computation. DeConv, also known as upsampling or transposed convolution, aims to extrapolate new information from the input feature maps. This contrasts with Conv that aims to interpolate the most relevant information from the input. DeConv layer takes feature maps of size $N_C * H * W$ and a group of coefficient matrix of shape $N_F * N_C * k * k$ as inputs, and produces output feature maps of size $N_F * H_O * W_O$. The input and output size in height and width dimensions are related as follows:

$$H_O = s * (H - 1) + k - 2 * p \tag{1}$$

$$W_O = s * (W - 1) + k - 2 * p \tag{2}$$

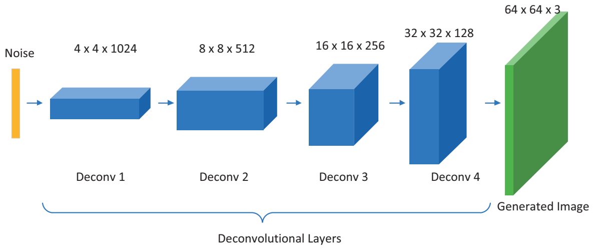


Fig. 2. A typical architecture of the generator of GANs, which is used in DCGAN [2] for scene modelling.

where k, s and p denote the values of kernel size, stride and padding for a given layer respectively. Please note that different DeConv parameters (such as $k = 4, s = 2$ or $k = 5, s = 2$) can be configured to implement the same generative network shown in Figure 2 for different applications. This is exactly the motivation for us to provide parametrized hardware design templates to support different layer configurations.

Algorithm 1 describes the DeConv layer of the generator in a high level which consists of the filter loop and channel loop. For each filter of output, at first one channel of input with shape of $H * W$ ($I[c]$) is taken out to be deconvolved with the kernel matrix ($K[f, c]$); then the results of each channel of input maps are accumulated to produce one filter of output ($O[f]$). This process is repeated for N_F times to produce all filters of the output feature maps. The implementation of the *deconv* operation on the two 2-D matrices shown in line 3 of Algorithm 1 will be covered in detail in Section II-B.

Algorithm 1 Deconvolution Algorithm of the Generator.

Input: input feature map I of shape $N_C * H * W$;
Input: A coefficient matrix K of shape $N_F * N_C * k * k$;
Output: output feature map of shape $N_F * H_O * W_O$;

```
1: for $f = 1$ to $N_F$ do
2:     for $c = 1$ to $N_C$ do
3:         $O[f] += deconv(I[c], K[f, c])$
4:     end for
5: end for
```

B. 2-D Deconvolution Algorithms

Software-based: The CPU- and GPU-based 2-D DeConv are implemented as transposed convolution, by adding appropriate amount of zero padding around and/or between the input feature maps [11]. A deconvolution described by kernel size k, stride s and padding p is performed as a convolution with $k' = k$, $s' = 1$, $p' = k - p - 1$ and adding $s - 1$ zero padding between each input data.

Hardware-based: An FPGA-friendly method was proposed in [7], [12] by multiplying the coefficient kernel with each input pixel and summing the overlapping area in output maps. This method is demonstrated in Figure 3. It proceeds through four steps: (1) multiply a single input pixel by the 2-D coefficient matrix; (2) sum the results of step (1) where the outputs overlap; (3) repeat (1) and (2) for all input pixels; (4)

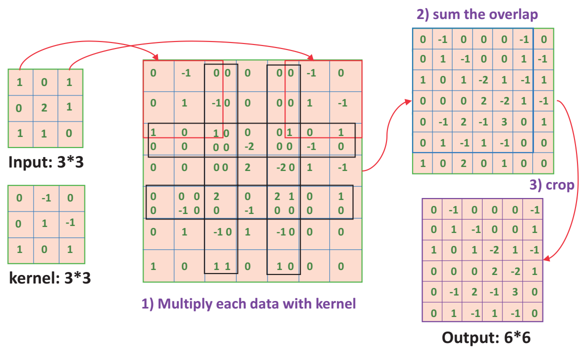


Fig. 3. An illustration of the hardware-based 2-D DeConv method. Raw input size is 3x3 and output size is 6x6, with DeConv configuration: $k = 3$, $s = 2$, $p = 1$. The black boxes indicate the overlapped rows and columns.

remove the elements from output feature maps in the border of size p. The last step which crops the output feature maps is also needed by the software-based method and it can be different for different deep learning frameworks such as Tensorflow [13] or Caffe [14].

Comparison: Please note that these two methods produce identical results for the same input. However, when targeting FPGA, the key disadvantages of the software-based method are:

- The zero-padding operation is inherently inefficient when implemented in an FPGA, and it involves a non-uniform data access pattern when DeConv window slides over the zero-inserted input;
- Performing multiply-accumulate on the inserted zeros causes computational inefficiency due to multiplications with zero.

The second method, by contrast, is more FPGA friendly by avoiding zeros insertion and it thus improves the computational efficiency. Besides, it is more extensible and can adapt to different layer configurations. Therefore, we propose an optimized DeConv architecture based on this method in this work. In contrast to Conv which needs padding its input, DeConv requires a crop step for output to produce the required output shapes. The main challenge of the hardware implementation of DeConv lies in dealing with the overlapping sum problem in the outputs.

C. Related Work

Previous FPGA-based CNN accelerators aim to optimize the convolution from computation reduction such as using fast Fourier transform (FFT) [15] and winograd algorithm [16], to data quanization [17] and hardware level optimizations [18] such as memory hierarchy. However, some of these optimizations such as FFT method are only suitable for Conv, and cannot be directly applied to DeConv and GANs.

Recent FPGA-based accelerators for DeConv networks are presented in [6]–[8]. Yazdanbakhsh *et al.* [6] proposed an end-to-end FPGA accelerator for GANs that combined MIMD and SIMD models while separating data retrieval and data processing units at the finest granularity. Chang *et al.* [8] proposed a design methodology for FPGA-based CNN accelerator for image super-resolution algorithm, a combination of multiple Conv layers and a single DeConv layer with efficient parallelization. However, both designs in [6], [8] are based on the software-based method and thus computationally inefficient compared to the hardware-based method in this work as we mentioned earlier. Zhang *et al.* [7] proposed an accelerator using Vivado HLS tool based on the second method which is the same as our work. They proposed a reverse looping method that determines which inputs are needed to get the desired output, in order to process the overlapping sum problem. However, with their method, the positions of input pixels should be calculated through the formulas for each loop iteration, and loop dimensions are increased as large as the output image, which imposed overhead on communication with the host processor and increased system latency thereby precluding real-time applications [8].

In terms of the data load and transfer problem which is often the *de facto* bottleneck for FPGA-based acceleration of very deep CNNs, loop tiling is performed to split the original workload into smaller ones for hardware execution. The tiling method (also named as blocking) is to divide the input feature maps into several parts so that they can be stored in on-chip buffers for data reuse and thus suited for memory-limited FPGA devices. Previous researches mainly focused on optimizing the tiling factors in the design space exploration process [19], but the system's performance still suffers from the data transfer overheads among memories. Aydonat *et al.* [20] proposed an OpenCL-based accelerator for CNNs by caching all intermediate feature maps on-chip in stream-buffers. However, their methodology requires very large on-chip memory capacity. Li *et al.* [21] studied efficient tiling method for convolution but its accuracy is sacrificed due to replacing the boundary pixels of tiles with zero values to remove the data dependencies in adjacent tiles.

However, none of the above efforts that target FPGAs have considered to reduce the memory accesses produced by the generative networks. To the best of the authors' knowledge, this is the first work that proposes hardware architecture of GANs' generator with memory-efficient tiling method, in order to address the memory-bound problem to provide effective FPGA-based acceleration.

III. Proposed Deconvolutional Architecture

This section presents the optimized architecture for the hardware-based DeConv algorithm. A crop module is designed to support removing the border of any size in the output blocks. To automate the hardware generation process, hardware templates are provided with configurable DeConv parameters and adjustable parallelism to accommodate a range of networks and devices with constrained resources.

A. Parametrized DeConv Architecture

The proposed DeConv architecture is shown in Figure 4. The input data are stored in on-chip memory for data reuse between filter processing as shown in the filter loop in line 1

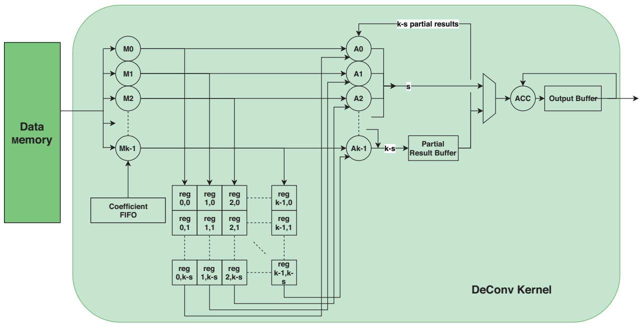


Fig. 4. The proposed DeConv architecture with configured parameters to supports different layers of GANs.

of Algorithm 1. For $s < k$, there are $k - s$ columns or/and rows overlapped between the output blocks for the adjacent input pixels. Therefore, a register array with size $k * (k - s)$ is instantiated to buffer these overlapped data for summation. In each cycle, one data is read from the memory and sent to k multipliers where the data is multiplied by one column of the coefficient matrix. The results are then buffered in one row of the $k * (k - s)$ registers and shifted every cycle. To deal with the column overlap, the results of the multipliers are added by the data in the last column of the registers before they are fed into the partial result buffer or output buffer. To deal with the row overlap, the outputs of the first $k - s$ adders are also added to the partial results before they are accumulated and stored in the output buffer.

Compared to the Conv architecture, the difference is that beside the multiply-accumulate (MAC) operators in the kernel, the register array is introduced between the multipliers and adders to sum the column overlaps, and partial result buffers are inserted between the adders and the accumulators to sum the row overlaps. We ensure the correctness of this hardware module using a control logic that places the required data at the right clock cycles. The proposed architecture can support different parameters of DeConv layer, i.e., (k, s, p) and therefore can be reused for a variety of GANs. It is further optimized for parallel processing by instantiating multiple kernels and storing multiple data in one memory address.

B. *Crop Module*

The crop function in DeConv is necessary for generating the right shape of output by removing the unwanted data in the border of results from multiply-accumulator. We implement the crop module in hardware directly taking the output from the DeConv kernel as input. The design of this module is shown in Figure 5. Two counters: line counter and row counter, are used to obtain the position of the pixels in the output shape and then the data is determined to be valid or disabled (cropped) to the next module or memories. By taking two parameters (x and y shown in Figure 5) which can be configured at runtime as inputs, it is capable of removing the border of any size in the output blocks, in order to support different deep learning frameworks. When $p = 0$, the crop module is bypassed and the entire output results from DeConv kernel are passed to the next stage.

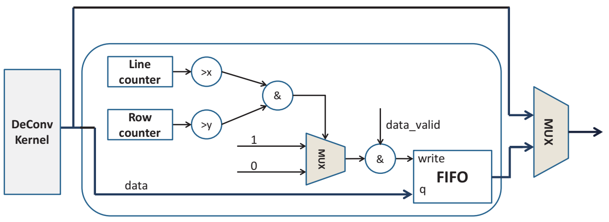


Fig. 5. The block diagram of the crop module to produce output shape of arbitrary size for different deep learning frameworks.

C. *Hardware Design Template*

A hardware template is a generic implementation with configurable parameters and described in a hardware description language (HDL). To improve the design productivity and efficiency, we implement the DeConv architecture as a hardware template design using Verilog HDL, which can be generalised and deployed onto multiple devices. In order to accelerate different GANs, the above architecture is customized to support different DeConv layers with multiple (k, s, p) configurations. The design template with supported parameters covers three cases of processing: 1) when $k = s$, there are no overlapping sum computations; 2) when $s < k$, the design sums the overlapping regions between output blocks; 3) when $p = 0$, the crop module is bypassed. As such, without extra hardware design work, our template design can be used for a variety of generators in GANs for different applications. In Section V, we evaluate the template designed at three configurations: $(k, s, p) = (2, 2, 0), (4, 2, 1), (5, 2, 2)$ which cover all the cases to be considered. In addition, the DeConv kernels are optimized for parallel processing to improve resource utilization and performance. The degree of parallelism is also one of the configurable parameters to satisfy the resource constraints of large or small scale chips.

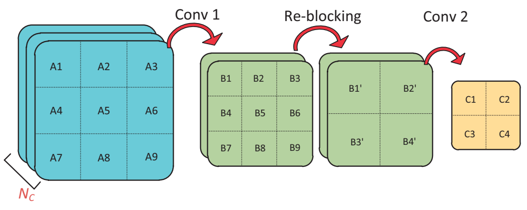
IV. Memory-Efficient Generator Accelerator

In this section, we propose a new tiling method to accelerate the generator of GANs so as to reduce off-chip memory accesses for FPGA-based accelerators. A vanilla design is also proposed as a baseline for comparison.

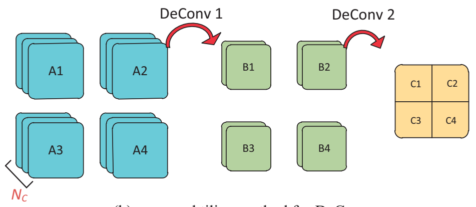
A. *Proposed Tiling Method*

Existing FPGAs are memory limited and the capacity of on-chip memory is not large enough to store all the coefficients and intermediate data in large-scale networks during inference. As a consequence, FPGA-based accelerators resort to external DRAMs to store these data. However, the massive amount of off-chip memory accesses are costly in terms of energy and latency [22], [23], and input data cache must be implemented by means of on-chip buffers for data reuse.

This problem is often addressed with *tiling* or *blocking*, which divides the input maps of each layer into multiple blocks that fit into the on-chip buffers. Loop tiling is performed in the height and width dimensions of input maps rather than the channel dimension. That is to say, the original inputs of shape $N_C * H * W$ are divided into small blocks of shape



(a) traditional tiling method for Conv or DeConv.



(b) proposed tiling method for DeConv.

Fig. 6. An illustration of the traditional (**up**) and proposed (**down**) tiling methods.

$N_C{*}T_H{*}T_W$, where $T_H < H$ and $T_W < W$. An illustration of the standard tiling method for two successive layers is shown in Figure 6(a). The input map A is firstly divided into 9 blocks and each block is processed sequentially to produce the output map B. Then for the second layer, the input B is divided into 4 blocks due to the reduced channel value compared to A. For each layer, the tile or block size [1] is re-selected to avoid very large loop dimensions due to relatively small tile size for layers with large N_C. This blocking method is required by convolution, since the second convolutional layer's result relies on the output data of the adjacent tiles. For example, $C1$ not only depends on $B1$ but also partial data of $B2$, $B3$ and $B4$. These data dependencies require the implementation resorting to off-chip memory to buffer the intermediate data. This method can also be used to implement DeConv networks in hardware. Its advantage is that it can fully utilize the on-chip buffers by adjusting tile size for different layers. However, the intermediate data has to be stored in off-chip memories and thus increasing the extra DRAM access latency.

Nevertheless, there is no data dependency in the input maps between successive DeConv layers. For this reason, we propose a memory-efficient tiling approach for DeConv layers that aims to take the advantages of multi-layer fusion and to buffer the intermediate data without resorting to off-chip memory. The proposed method is shown in Figure 6(b). Based on the memory resource of the target device and the maximum number of N_C in the generator, the largest tile size is first determined. Then the number of blocks are fixed for each layer and each input block is processed until the final layer to obtain the corresponding output block. As such, the entire network can be seen as divided into independent sub-networks which are processed sequentially. Compared to the traditional method, it has the cost of larger on-chip memory since both the input and output data need to be buffered. The memory utilization can be relatively low for some layers with small

[1]In this paper, the tile size refers to the height and width dimensions, i.e., $T_H * T_W$ but excludes the channel dimension.

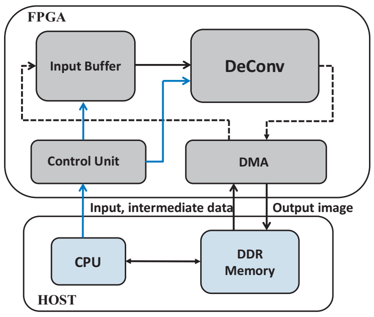


Fig. 7. The block diagram of the vanilla generator accelerator.

N_C, as the tile size is fixed between layers. Nevertheless, it only requires data transfers of input data and output image, and all intermediate data are processed on-chip. As a consequence, it reduces most of the off-chip memory accesses.

B. *Vanilla Generator Accelerator*

We propose the vanilla FPGA-based accelerator of the generator based on the conventional tiling method as the baseline. The overall system is shown in Figure 7. It is composed of several major components, which are the computation unit (DeConv), on-chip buffers, external memory and on-/off-chip interconnect (DMA). Deconv is the basic computation unit of the generator and its architecture and implementation details have been described in Section III. All data for processing are stored in external memory. Due to the limitation of the on-chip memory size, data are first cached in on-chip buffers before being fed to the computation unit. For each layer, the input feature maps are first read from DDR by block and stored in on-chip buffers; then after processed through DeConv unit, the corresponding block of the output feature maps is sent out to the DDR memory; for the successive layer execution, the whole input maps are divided to new block size based on the on-chip buffer size and N_C. The control of the hardware execution and the configurations of each layer are performed in the host CPU.

C. *Memory-Efficient Generator Accelerator*

The architecture of the proposed memory-efficient accelerator is shown in Figure 8, which is based on a single run of the network in on-chip. The overall architecture is similar to the baseline, but two on-chip buffers (let's say A and B) are utilized to store the input and output maps of each layer. The feature maps of all the layers are divided into a fixed number of blocks based on the maximum channel of the network to fit into the on-chip memories on the target FPGA. One input block is first cached on chip and processed until the final block of output image is produced and transferred to DDR memory. This process is repeated for all blocks to obtain the final output image. A detailed data flow of two successive layers is as follows: for the first layer, one block of input feature maps are cached in buffer A and processed through DeConv, while the results are cached in buffer B; To run the second layer, the data in buffer B performs as input and are processed through

TABLE I

A SUMMARY OF THE EVALUATED GENERATORS OF GANS

Name	Year	Application	Input Shape	Output Shape	MAX N_C	# DeConv
DCGAN [2]	2016	Generate indoor scenes	4*4*1024	64*64*3	1024	4
C-GAN [5], [24]	2018	Generate photo-realistic images	4*4*256	256*256*3	256	6
UP-GAN [25]	2017	Generate Chairs, Tables and Cars	8*8*256	128*128*3	256	4
DN-GAN [25]	2017	Generate Chairs, Tables and Cars	8*8*128	128*128*1	128	4

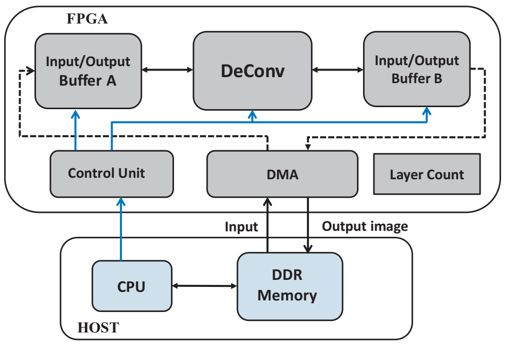


Fig. 8. The block diagram of the proposed generator accelerator.

DeConv. Correspondingly, the result of the second layer are cached in buffer A. This process works like ping-pong until one input block is finished to the last layer, then the second input block is continued. The direction of the data flow is controlled by a layer counter.

Compared to the baseline, the proposed architecture is at the expense of memory resource on FPGA by doubling the data buffers. The benefit comes from the fact that it doesn't require the DDR memory to buffer the intermediate data any more, and off-chip transfers are only needed for the input data, network parameters and output images. Therefore, it is much more memory-efficient and energy-efficient. Data parallelism concerning the DeConv kernel is further explored to fully utilize the resources for both accelerators.

V. Evaluation and Experiments

We evaluate the performance of the two accelerators for the generator of several state-of-the-art GANs with different configurations using an FPGA device (Xilinx XC7Z045), with comparison to the respective implementations on an Intel i7-950 CPU and NVIDIA Titan X GPU.

A. Benchmarks

A variety of GANs proposed in recent literature for different applications are used as case studies. Table I summarizes the evaluated generators of the GAN models. All of these generator models are configured with three DeConv kernels, i.e., (k, s, p) = (2,2,0), (4,2,1) and (5,2,2), by utilizing the provided hardware templates. These kernels can be used for any specific GAN model to provide trade-off between the accuracy and processing speed to meet different user's requirement on their respective tasks or datasets. Overall, these GAN models can be used for various applications including high-resolution image generation [2], audio to video mapping [5] and 3D object generation [25].

B. Experiment Setup

We implement the two accelerators on a Xilinx Zynq ZC706 board which consists of a Xilinx XC7Z045 FPGA chip, dual ARM Cortex-A9 Processor and 1 GB DDR3 memory. Both accelerators are designed using 16-bit fixed-point precision and developed using Verilog HDL. The whole system is implemented with Vivado Design Suite 2016.2 which performs synthesis and implementation. All designs run on a single 150 MHz clock frequency. The DDR3 memory in our design has a datapath width of 64 bits and it operates at the same clock frequency (150 MHz) as the FPGA engine. All results are post place and route except if it is stated otherwise.

For comparison, the respective software implementations run on CPUs and GPUs use the deep learning software framework Tensorflow [13] on CentOS 7.2 operating system. The CPU platform is Intel Core i7-950 CPU@3.07GHz (8 cores). The GPU platform is Nvidia TITAN X (Pascal) (3840 CUDA cores with 12GB GDDR5 384-bit memory).

C. Resource Utilization

Table II gives the resource utilization (LUTs, FFs, etc.) of a single DeConv kernel (without parallelism) for each configuration in the target device. The number of DSPs is linear to the square of kernel size, since only multipliers are implemented in DSPs and the adders are implemented using LUTs to save DSPs. The last two kernels use more LUTs and FFs since they need additional logic to process the overlapping sum which is not required by the first configuration.

TABLE II

RESOURCE UTILIZATION OF A SINGLE DECONV KERNEL UNDER THE THREE EVALUATED CONFIGURATIONS.

Configuration (k, s, p)	LUTs	FFs	DSPs
(2,2,0)	1444	1570	4
(4,2,1)	3779	2246	16
(5,2,2)	5080	2471	25

For both accelerators, the DeConv kernels are instantiated multiple times to take advantage of parallel processing, in order to fully utilize the resources. The resource usage of the two accelerators with each configuration is shown in Figure 9. The computational resources such as LUTs, FFs

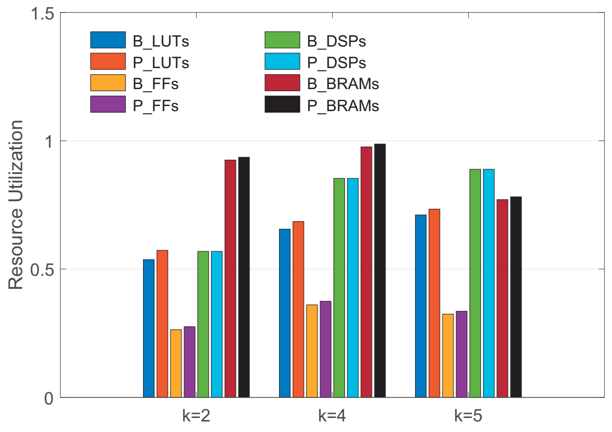


Fig. 9. The resource utilization of the two accelerators on the target FPGA (XC7Z045), where B_LUTs and P_LUTs represent the LUTs utilization of the baseline and proposed accelerator respectively.

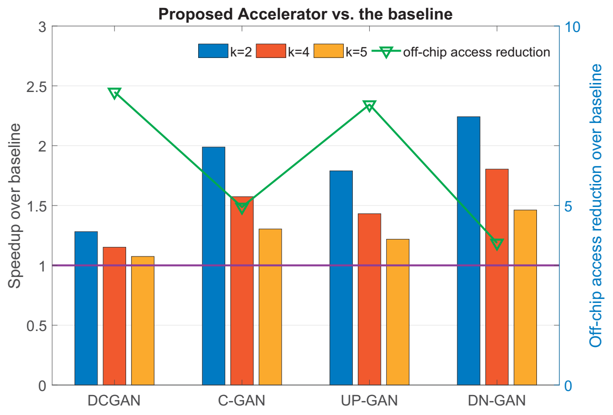


Fig. 10. The speedup with the proposed accelerator over the baseline. The green line indicates the total off-chip data transfer reductions across the benchmark GANs.

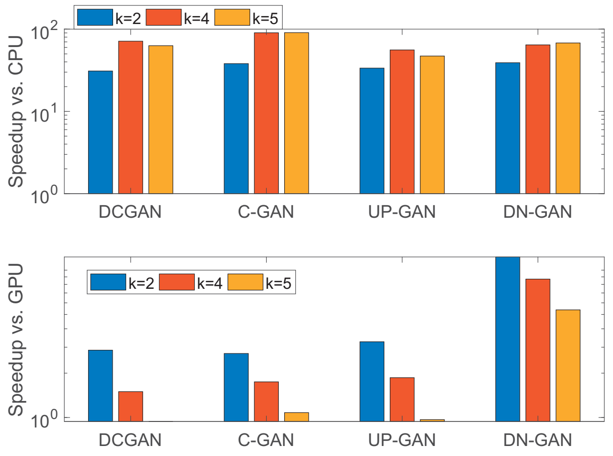


Fig. 11. The speedup of the proposed accelerator compared to the respective CPU (**up**) and GPU (**down**) implementations.

and DSPs are mainly spent on the DeConv kernel module. Since both accelerators have the DeConv module with the same parallelism under each configuration, they have the same DSP usage. However, the memory-efficient accelerator has a slightly increased LUTs and FFs usage compared to the baseline due to the additional control logic for the dataflow between two data buffers and DeConv kernel. Both accelerators have more than 80% of BRAM utilization due to the limited memory size on the target FPGA chip.

D. Speedup and Energy Efficiency

We first evaluate the off-chip memory access reduction and speedup for inference of the memory-efficient accelerator compared to the baseline for the evaluated generators of GANs in Table I under each configuration. The results are shown in Figure 10. In average, it achieves 6.2x of off-chip data transfer reduction across the benchmark GANs, and up to 8.2x reduction for DCGAN model. Since the total size of the intermediate data is independent of the kernel size and thus they are the same among the three configurations, the achieved improvements of each model are the same across all configurations. The reduction will be more significant for large-scale networks with deep layers and large channels as they need more data transfers by the baseline accelerator.

In terms of the processing speed, the proposed accelerator provides up to 2.3x speedup compared to the baseline. The speedup is less notable for large DeConv kernel size such as $k = 5$ compared to that of $k = 2$. This is because the degree of parallelism is limited on the target device for large DeConv kernel size, due to the limited computation and memory resources. Therefore, for networks such as DCGAN and/or when utilizing large kernel size, the computation time is dominant and communication time is a minor part. In such situation, the baseline can provide comparable performance to the proposed one. Nevertheless, the processing speed of the memory-efficient accelerator outperforms the baseline in all configurations of the evaluated GAN models.

Finally, we compare the speedup of our accelerator against the respective CPU and GPU implementations. The results are shown in Figure 11. The speedup is in the range of 30x-92x and 58x in average compared to the designs on an Intel 8-core i7-950 CPU. Compared to the respective GPU design, the speedup is in the range of 0.8x-12x and 3.6x in average. For DCGAN and UP-GAN models with $k = 5$, our accelerator is a bit slower (around 0.8x) than the GPU. Nevertheless, in all other cases, the proposed accelerator outperforms the GPU design. From the results in Figure 11, we observe that the FPGA-based accelerators provide higher speedups over GPUs for small kernels (such as $k = 2$) and small networks (such as DN-GAN). This happens because the Titan GPU cannot achieve full thread utilization, while the FPGA is able to utilize a high percentage of its resources even for small networks.

Figure 12 shows the energy efficiency improvement in terms of Performance-per-Watt of the proposed accelerator compared to the respective CPU and GPU implementations. The power consumption of each device is measured using a wall plug power meter during the execution of the application. Compared to the respective designs on CPUs and GPUs, the achieved improvements are more than 400x over the CPU, and in the range of 8.3x-108x over the NVIDIA Titan X GPU. The results confirm that our accelerator yields much lower power consumption and significant energy efficiency compared to

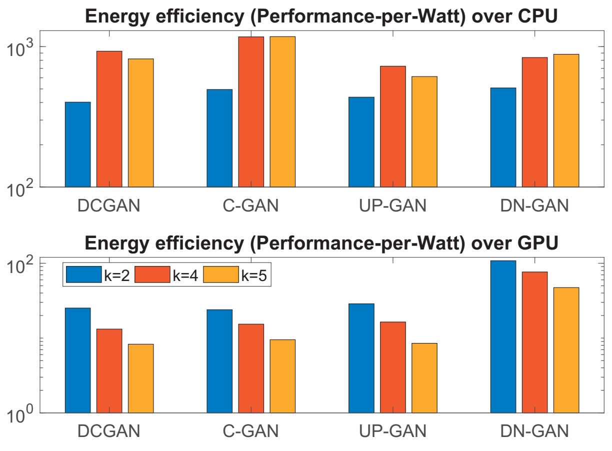


Fig. 12. The energy efficiency in terms of Performance-per-Watt of the proposed accelerator compared to the respective CPU (**up**) and GPU (**down**) implementations.

GPUs. As such, it is more promising for embedded and mobile applications.

E. Discussion

In summary, the proposed accelerator achieves better performance compared to the vanilla design, by storing all the intermediate data in on-chip buffers at the cost of memory resource. It also provides significant speedups compared to the CPU design and energy-efficiency improvement over GPUs. Based on the results we obtained for the evaluated GAN models, we provide some directions on the design choices for accelerating the generator of GANs on FPGAs:

1) For *non-deep* networks which have a small number of layers and consist of layers with large channels and filters (such as DCGAN), they are often computation bound. When accelerating these networks on FPGA, the vanilla accelerator provides comparable performance to the memory-efficient one. Therefore, the vanilla design is preferred as it has lower memory overhead and is more suited for memory-limited devices when targeting these networks;

2) The proposed memory-efficient accelerator is preferred for *deep* or *wide* networks. In particular, when small DeConv kernel is used or large-scale FPGA chip is targeted, high degree of parallelism can be achieved within the accelerator and the communication time becomes the bottleneck. In this case, the memory-efficient accelerator provides notable performance improvement compared to the vanilla design.

VI. Conclusion

This paper focuses on the acceleration of the generators of GANs for fast inference on FPGAs. We propose a parametrized and deeply optimized hardware architecture for deconvolutional layers, and provide hardware templates with configurable parameters to support a variety of GAN models in order to improve designer productivity. To efficiently map the generator on memory-limited FPGAs, we introduce a novel tiling method. The proposed method comes with a memory-efficient architecture that eliminates the requirement of off-chip memory to buffer the intermediate data and thus reduces off-chip data transfers largely. Experimental results show that our memory-efficient accelerator achieves notable speedups over the vanilla accelerator. Compared to the respective CPU and GPU designs, the proposed accelerator provides significant energy efficiency improvement. Future work includes targeting the proposed accelerator for various applications, automating their development and evaluating their implementations against various technologies including embedded GPUs.

References

[1] I. Goodfellow *et al.*, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

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

[3] S. Liu *et al.*, "Optimizing cnn-based segmentation with deeply customized convolutional and deconvolutional architectures on fpga," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2018.

[4] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz, "Mocogan: Decomposing motion and content for video generation," *arXiv:1707.04993*, 2017.

[5] S. A. Jalalifar, H. Hasani, and H. Aghajan, "Speech-driven facial reenactment using conditional generative adversarial networks," *arXiv:1803.07461*, 2018.

[6] A. Yazdanbakhsh *et al.*, "FlexiGAN: An End-to-End Solution for FPGA Acceleration of Generative Adversarial Networks," in *FCCM*, 2018.

[7] X. Zhang, S. Das, O. Neopane, and K. Kreutz-Delgado, "A design methodology for efficient implementation of deconvolutional neural networks on an fpga," *arXiv:1705.02583*, 2017.

[8] J.-W. Chang and S.-J. Kang, "Optimizing FPGA-based convolutional neural networks accelerator for image super-resolution," in *ASP-DAC*, 2018, pp. 343–348.

[9] S. Liu, G. Mingas, and C.-S. Bouganis, "Parallel resampling for particle filters on FPGAs," in *FPT*, 2014, pp. 191–198.

[10] ——, "An exact MCMC accelerator under custom precision regimes," in *FPT*, 2015, pp. 120–127.

[11] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv:1603.07285*, 2016.

[12] S. Liu, X. Niu, and W. Luk, "A low-power deconvolutional accelerator for convolutional neural network based segmentation on fpga," in *FPGA*, 2018, pp. 293–293.

[13] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. [Online]. Available: https://www.tensorflow.org/

[14] Y. Jia *et al.*, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, 2014, pp. 675–678.

[15] C. Zhang and V. Prasanna, "Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system," in *FPGA*, 2017, pp. 35–44.

[16] L. Lu *et al.*, "Evaluating fast algorithms for convolutional neural networks on fpgas," in *FCCM*, 2017, pp. 101–108.

[17] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "Rebnet: Residual binarized neural network," in *FCCM*, 2018.

[18] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA*, 2016, pp. 26–35.

[19] R. Zhao *et al.*, "Hardware compilation of deep neural networks: An overview," in *ASAP*, 2018.

[20] U. Aydonat *et al.*, "An OpenCL$^{\text{TM}}$Deep Learning Accelerator on Arria 10," in *FPGA*, 2017, pp. 55–64.

[21] G. Li *et al.*, "Block convolution: Towards memory-efficient inference of large-scale CNNs on FPGA," in *DATE*, 2018, pp. 1163–1166.

[22] S. Liu and C.-S. Bouganis, "Communication-Aware MCMC Method for Big Data Applications on FPGAs," in *FCCM*, 2017, pp. 9–16.

[23] S. Liu, G. Mingas, and C.-S. Bouganis, "An unbiased MCMC FPGA-based accelerator in the land of custom precision arithmetic," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 745–758, 2017.

[24] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv:1411.1784*, 2014.

[25] A. Dosovitskiy *et al.*, "Learning to generate chairs, tables and cars with convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 4, pp. 692–705, 2017.