A Reconfigurable Parallelization of Generative Adversarial Networks based on Array Processor

Xiaoyan Xie*, Miaomiao Chai *, Zhuolin Du *, Kun Yang*, Shaorun Yin*

*Xi'an University of Posts and Telecommunications, Xi'an, China
E-mail: chaimiaomiao@stu.xupt.edu.cn Tel: +86-29-18232726200

Abstract— **Aiming at the intensive calculations of convolution and the invalid calculations caused by "zero" inserted of deconvolution in Generative Adversarial Network (GAN), which makes difficulties of accelerated by hardware. Through analyzing of network structure and calculation flows of GAN, a paralleling scheme of reconfiguration for convolution and deconvolution is proposed in this paper. Based on the Dynamic Programmable Reconfigurable Array Processor (DPRAP), on a 4×4 processing elements (PEs) array, the flexible switching of the two convolution modes are driven by a H-tree controlled reconfiguration mechanism. The proposed scheme is verified based on the DPRAP. The experimental results show that, compared with other FPGA schemes, the resource occupation can be reduced by up to 90% at a working frequency of 150MHz. Performance has been significantly improved.**

Keywords— Generative Adversarial Network; Parallelization; Array Processor; Reconfigurable

I. Introduction

GAN [1] are widely used in deep learning and have made great progress in image generation, such as style transfer, image super-resolution generation, text-to-image synthesis etc. Deep neural networks usually have the characteristics of high throughput and require a higher calculation amount and memory access frequency. To meet the growing demands for real-time applications, there requires running neural network with hardware accelerator strongly. Recently, various high-performance hardware schemes of accelerating of convolutional neural network (CNN) have been discussed, including the distributed GPUs and special accelerators based on FPGAs or ASIC. Among them, FPGA-based accelerators have lower latency and lower power consumption than GPUs, more flexible and configurable than ASICs [2].

Reconfigurable accelerators based on Field Programmable Gate Array (FPGA) have attracted the attention of more and more researchers due to their better performance, high energy efficiency, rapid development cycle and reconfigurability. In [3], a reconfigurable and efficient accelerator is proposed. Which achieve dual mapping of convolution and deconvolution layers but needs to introduce additional hardware to handle deconvolution. Nowadays, the research on hardware acceleration of pure convolution has been very deep, but the effective solutions to the cooperative design of convolution and deconvolution is still lack of mention. To map the deconvolution algorithm on a unified architecture, it is usually necessary to insert "zero" between the input feature maps, and then treat it as a convolution operation for calculation. However, these cause more than 75% of invalid calculations [4]. For this, reference [5] proposed an end-to-end FPGA accelerator for GAN, which combines MIMD and SIMD models, and separates the data retrieval and data processing units with the best computational granularity, but the MIMD control adds locally buffered instruction storage, resulting in additional resource overhead and area cost. Reference [6] proposed an FPGA-based deconvolution accelerator, but it treated the convolution and deconvolution with separate functional unit, resulting in the larger consumption of area overhead, and the lower utilization of on-chip resources. Reference [7] proposed a reverse loop method that supports convolution and deconvolution, but the pixel address needs to be recalculated in each iteration, which increases the communication overhead with the main processor.

In summary, although there has solved some difficulties, there are still challenges to high resource overhead and inflexible switching of convolution and deconvolution in accelerating GAN with hardware. Within the convolution and deconvolution, the calculations in each individual layer are independent of each other. In the process of convolution, the calculation of the innermost layer only involves intra-block addition, but in the process of deconvolution, the calculation of the innermost layer involves the addition among blocks. Compared to the convolution, the parallelization is relatively complicated. The challenge to balance the isomeric structure of deconvolution and the cost of area and power consumption is not to be ignored. In this paper, we focus on the latent parallelism of DCGAN, based on the DPRAP, a dynamic programmable reconfigurable array processor developed by author's team. Which has a H-Tree based reconfigurable controlled communication network on chip, can effectively support functional switching between convolution and deconvolution on the same PE group. By analyzing the parallelism of the convolution and deconvolution flows, a reconfigurable paralleling scheme of reconfiguration for convolution and deconvolution is given, based on the 4×4 PEs array. By verified with DPRAP, the experimental results show that the resource occupation can be reduced by up to 90% at a working frequency of 150MHz.

II. Related Works

A. Structure of GAN

GAN are composed of two parts, a generative model and a discriminant model, as shown in Figure 1. The generator includes a deconvolutional layers, which is used to capture the real data distribution. The discriminator is usually a convolutional layer, which aims to distinguish whether it is a sample synthesized by the generator or an original sample. The generator and discriminator compete to each other to produce a more powerful pair, to generates more realistic samples.

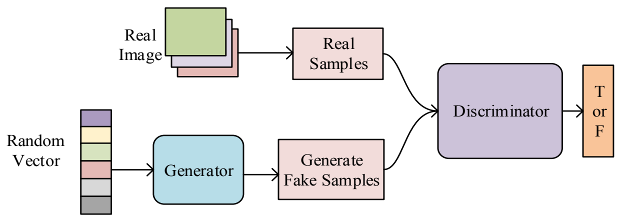


Figure 1: Schematic diagram of Generative Adversarial Network (GAN)

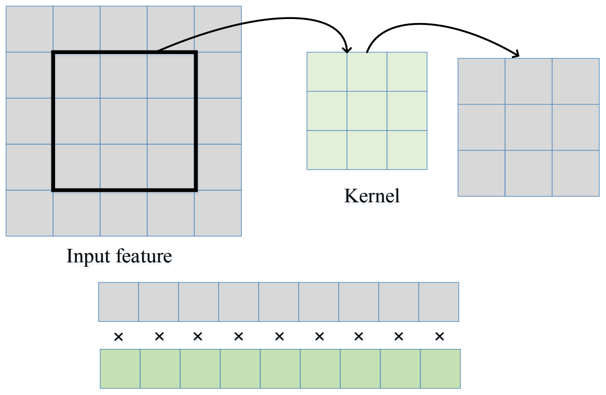
B. Parallelism of Convolutional

In the GAN, the convolutional layer is used for feature extraction. The operation flow is shown in Algorithm 1, which contains four cycles. Each of cycles can be parallelized according to its data correlation to improve the parallelism of its execution.

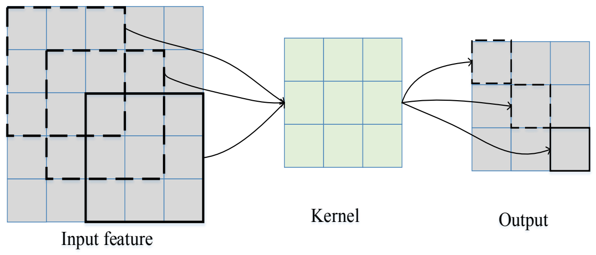
algorithm 1 Convolution algorithm

```
Input:input feature map I of shape $N_{if} *X_i *Y_i$;
Input:A coefficient matrix K of shape K * K;
Output: output feature map of shape $N_{of} *X_0 *Y_0$;
1: while i<=n do
2: LOOP1: Traverse $N_{of}$ output feature maps;
3:  LOOP2: Traverse $N_{if}$ input feature maps;
4:   LOOP3: The convolution window slides through the input feature
       graph of size XI * YI
5:    LOOP4: K*K multiplication and accumulation MAC operations in
a convolution window
7: end while
10: return output
```

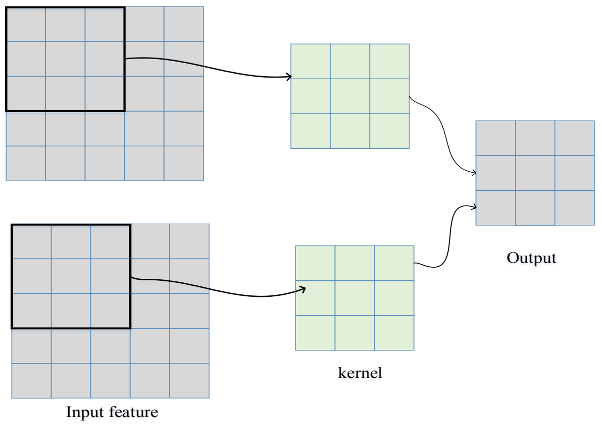
According to the above algorithm, we analyze the parallelism of each cycle. Figure 2 (a) shows the expansion of the innermost loop LOOP4. The internal parallelism of the convolution window is the parallelism of the operation between the single convolution window of the feature map and the single convolution kernel. The size of the convolution kernel is 3×3, and the convolution calculation with the convolution window of the feature graph requires a total of 9 multiplication calculations, which can be carried out in parallel. Figure 2(b) shows the parallel process of LOOP3, the same convolution kernel interacts at different positions on the same feature graph. Because the data of each convolution window is independent, these convolution operations can be calculated in parallel. Figure 2(c) is an extension of LOOP2. The output feature map is the result of superimposing multiple input feature maps and corresponding convolution kernels after convolution calculation. Because each feature map is independent, the convolution calculation of each feature map can be performed independently. The feature mapping is obtained by the accumulation and addition of two feature maps. The convolution calculation of the two feature maps is performed in parallel, and then the final result is accumulated by accumulation. Figure 2(d) expand LOOP1. Each input feature map will participate in the generation of all output feature maps. The convolution calculation of the convolution window of the feature map and the convolution kernel of different output feature maps is independent, so the calculation can be performed in parallel.



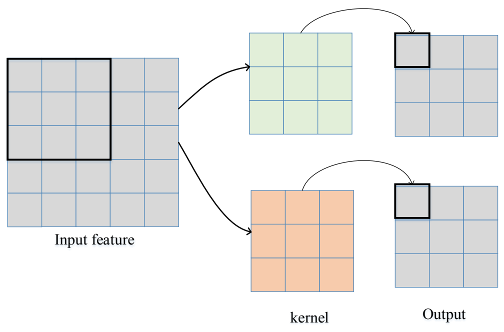
(a) Internal parallelism of the convolution window



(b) Parallelism between convolution Windows



(c) Internal parallelism of the input feature map



(d) Internal parallelism of the output feature map

Figure 2 Convolution calculation parallel graph

C. *Parallelism of Deconvolution*

It is the most basic unit of the generation network, and used for up-sampling the image from low resolution to high resolution. Algorithm 2 describes the deconvolution algorithm.

```
algorithm 2 Deconvolution algorithm
1.Input: input feature map I of shape NC ∗ H ∗W;
2. Input: A coefficient matrix K of shape k ∗ k;
3. Output: output feature map of shape NF ∗ HO ∗WO;
4. while $i<=n$ do
5. LOOP1: Traverse NFoutput feature maps
6.   LOOP2: Traverse $N_{if}$ input feature maps
       LOOP3: Implement deconvolution operation
7. end while
8. return output
```

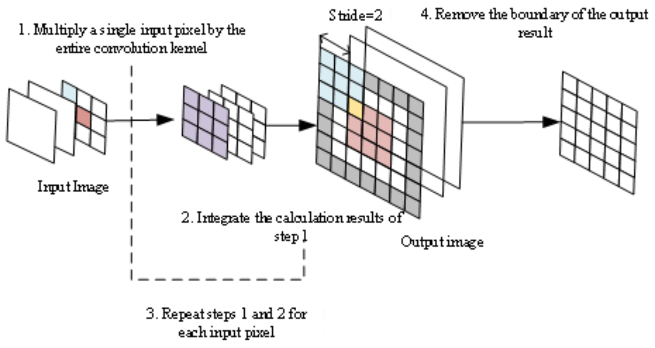
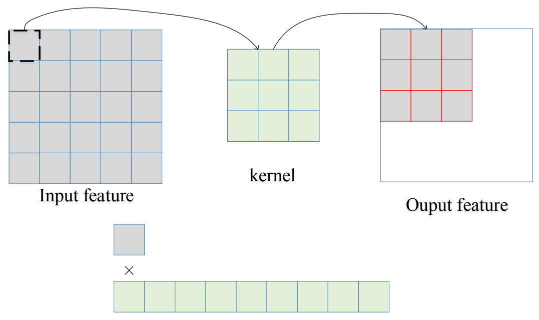

1. Multiply a single input pixel by the entire convolution kernel
Stride=2
4. Remove the boundary of the output result
Input Image
2. Integrate the calculation results of step 1
Output image
3. Repeat steps 1 and 2 for each input pixel

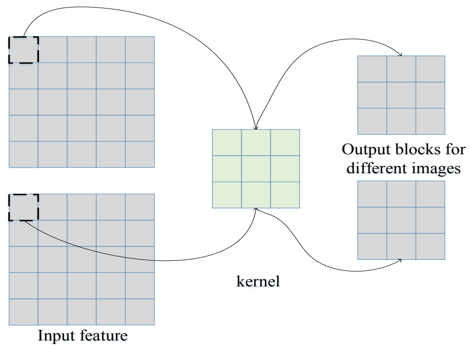
Figure 3: Schematic diagram of deconvolution process

The specific implementation process of the deconvolution algorithm is shown in Figure3.There are four steps to achieve deconvolution. Step 1 is to multiply the single input by the (K×K) convolution kernel, step 2 is to add the results of step 1 to the local area in the output feature maps. As shown in the Figure2, if the first pixel and the second pixel are multiplied by the weight, the multiplication result will be divided into two parts, one is the blue part, the other is the red part, the light-yellow part is overlaps. Step 3 is to repeat steps 1 and 2 until calculating all input pixels. The fourth step is to remove the border of the output image, the gray part in the figure. For the deconvolution layer with a span of s, when s=k, the output of the multiplication results of each pixel does not overlap. Step 2 is not needed in this case. and only multiply each input pixel by the k*k convolution kernel. When s <k, this computing manner needs to deal with the overlap of columns and rows in the output.

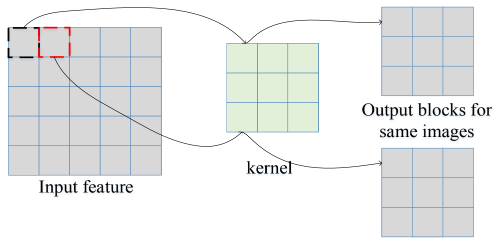
According to the above deconvolution algorithm, analyze the parallelism of the deconvolution algorithm. Figure4(a)(b) is the expansion of LOOP3. The internal parallelism of the deconvolution means that each pixel is multiplied by the entire convolution kernel. Each multiplication operation is independent of each other, so it is the parallelism of the internal deconvolution window. Each pixel in the input feature map is independent of each other, so their multiplication with the deconvolution window is also independent of each other. These operations can be implemented in parallel. Figure4 (c) is the expansion of LOOP2. Since each feature map is independent of each other, the deconvolution calculation of each feature map is also independent of each other. Figure4 (d) is the expansion of LOOP1.The deconvolution window and the convolution calculation of different output feature maps are independent.



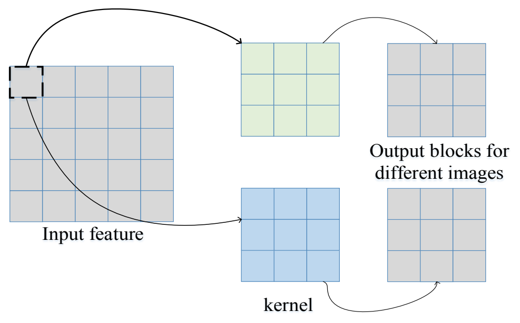
(a) parallelism of the internal deconvolution window



(b) Parallel between deconvolution windows



(c) Input feature map parallelism



(d) Ouput feature map parallelism

Figure 4 Parallel graph for deconvolution calculation

III. Dynamic Reconfigure Design of DCGAN

According to the conclusion of Section II, the DCGAN mapping can be reconfigurable paralleled with DPRAP. Which is indicated as shown in Fig.5. The global controller is the core of the reconfigurable mechanism. the upper layer is the host interface. the lower layer is multiple process element group (PEG). Each PEG contains 16 PEs. The global controller forms an H-Tree hierarchical communication network between the host interface and the array. While ensuring that every instruction can reach the PE at the same time, it realizes the control and management of array resources. When the host interface accesses the array, the global controller receives bus information from the host interface. The bus information includes address, flag, and instruction. They are used to determine the specific instruction executed by each PE at a certain time.

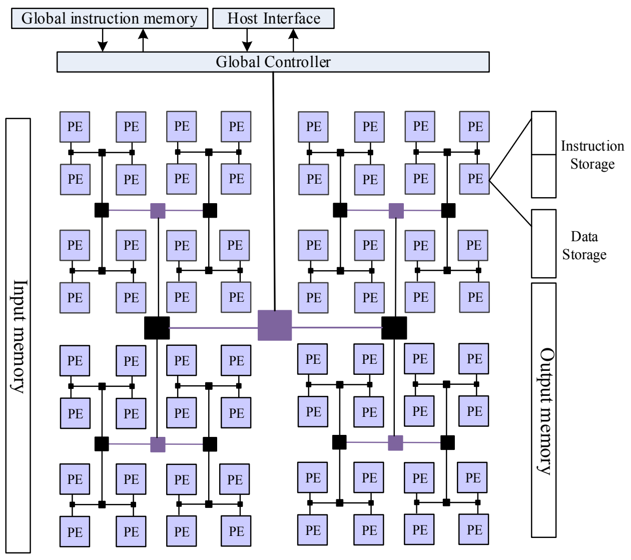


Figure 5 Dynamic reconfigurable array

When the processor resources are limited but multiple type of algorithms need to be mapped, instruction flows of each algorithm are initialized to different instruction stores of the same PE, the function switching between algorithms is controlled by configuration information derived from H-Tree and global controller. This reconfiguration mechanism based on context switching can switch between multiple algorithms according to application’s needs.

With the DPRAP, a reconfigurable DCGAN scheme can be planed as follows. Firstly, the data is preprocessed, the original data and the convolution kernel are stored in DIM in columns, and the generated intermediate data is stored in DOM. Secondly, the different algorithm instructions are initialized in different instruction stores of the same PE, and the call instructions are issued through the H-Tree network to complete the flexible switching between the two parts of the configuration. When it is necessary to update the information of a certain configuration store, the H-tree configuration network can directly send the required configuration information to the corresponding configuration store without affecting the normal execution of the PE. Through the H-Tree configuration network, the instructions flow of DCGAN issues different configuration instructions to the PEs, the switching is realized continually between convolution and deconvolution. As shown in Figure 6.

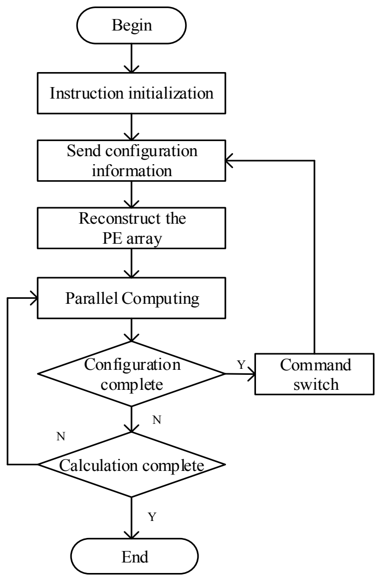


Figure 6 The reconfigure flows of DCGAN

The reconfiguration of DCGAN shown in Figure 4 can be deployed on a PEG. The instructions of deconvolution and convolution are all initialized in different configuration storage positions of the same PE. Different configurations are called in the same PEG to realize the switch between the two algorithms of convolution and deconvolution.

The mapping scheme is shown as in Figure7. Different calculation configurations are set for different algorithms. The default configuration is in PC1, which performs deconvolution calculation. After the deconvolution calculation is completed, and the reconfiguration instruction is to be called, performing the convolution calculation configured in PC2.

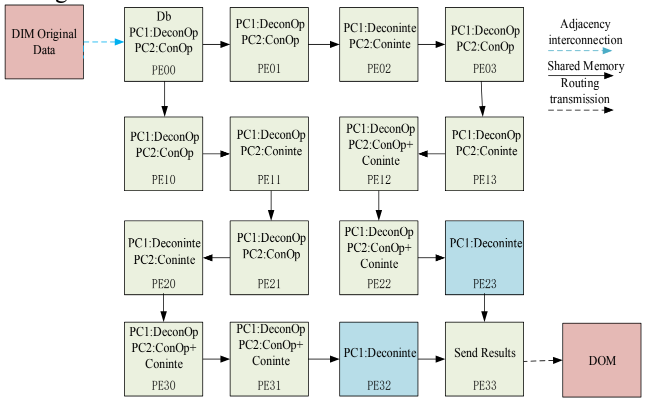

DIM Original Data
Db PC1:DeconOp PC2:ConOp PE00
PC1:DeconOp PC2:ConOp PE01
PC1:Deconinte PC2:Coninte PE02
PC1:DeconOp PC2:ConOp PE03
Adjacency interconnection
Shared Memory Routing transmission
PC1:DeconOp PC2:ConOp PE10
PC1:DeconOp PC2:Coninte PE11
PC1:DeconOp PC2:ConOp+ Coninte PE12
PC1:DeconOp PC2:Coninte PE13
PC1:Deconinte PC2:Coninte PE20
PC1:DeconOp PC2:ConOp PE21
PC1:DeconOp PC2:ConOp+ Coninte PE22
PC1:Deconinte PE23
PC1:DeconOp PC2:ConOp+ Coninte PE30
PC1:DeconOp PC2:ConOp+ Coninte PE31
PC1:Deconinte PE32
Send Results PE33
DOM

Figure 7: The reconfigure flow chart of DCGAN. Db: Data distribution; ConOp: Convolution operation; Coninte: Convolutional integration; DeconOp: Deconvolution operation; Deconinte: Deconvolution integration;

The deconvolution operation multiplies each pixel by a k×k convolution kernel, where the input data can be reused for filter. The deconvolution integration sums the results of deconvolution operation where the outputs overlap. The convolution operation multiplies input image by convolution kernel. The convolution integration accumulates the result of the multiplication. The reconfiguration process is shown as in Algorithm 3.

Algorithm 3 DCGAN reconfigurable parallelization algorithm

```
Input: Preprocessed data: $deconv\_x^{l-1}$; Deconvolution kernel: K×K
Output: Convolution calculation result: $conv\_x^{l}$
1: PE00 loads data in blocks from DIM
2: while i<=n do
3:     LOOP1:
4:     Data distribution: After PE00 loads the data, it sends data to PE01 and PE10 in blocks. After sending the data, PE00 sends handshake signals to the two PEs.
5:     Deconvolution calculation process: After PE01 and PE10 receive the handshake signal, each PE starts to perform deconvolution calculations at the same time.
6:     Results saving process: The deconvolution calculation result is passed into PE33, and PE33 finally writes all the results into the DOM.
7:     end LOOP1
8: end while
9: return Store the deconvolution calculation result in the DOM
10: A configuration call instruction is issued to switch between deconvolution and convolution calculation reconfiguration.
11: PE33 loads the convolution result data from the DOM and blocks the result.
12: while i<=n do
13:     LOOP2:
14:     Data distribution: PE33 loads the result data of the deconvolution calculation from the DOM and divides the result into blocks. PE33 sends handshake signals to PE01 and PE10 respectively
15:     Convolution calculation process: After PE01 and PE10 receive the handshake signal, each PE starts to perform convolution calculation at the same time.
16:     Results saving process: The convolution calculation result is transferred from PE23 and PE32 to PE33, and PE33 finally writes all the results into the DOM.
17:     end LOOP2
18: end while
19: return Store convolution calculation results in DOM
```

IV. Experimental Results and Performance Analysis

To verify the feasibility of the reconfigurable scheme proposed in Section 3, the verification is based on the DPRAP shown as in Figure 5. Firstly, the pictures being tested are converted into binary sequences and stored in off-chip storage. Secondly, the instructions to initialize of the convolution and deconvolution algorithms are stored into the instruction memory. Finally, functional simulation verification through Questasim10.1d are performed on the DPRAP. The design is synthesized through Xilinx's ISE14.7. The FPGA platform is Vertex-6 series XC6VLX760 devices, and the operating frequency can reach 150 MHz.

Table 1 Comparison of resource occupation

	Chip	Freq (MHZ)	LUTs （K）	FFs (K)
[5]	XCVU13P	190	1325	——
[8]	Xilinx Virtex7 485T	100	142	151
[9]	Kintex-7 XC7K410T	130	94	107
[10]	Kintex-7 XC7K410T	130	167	158
This work	Vertex-6	150	126	34

Use Xilinx's ISE14.7 development environment is used to synthesize the design. Table 1 list the frequency and resource utilization, including LUTs and FFs. Literature [5] aims to separate data retrieval and data processing units to greatly reduce the on-chip cache. Although its frequency is higher than that of ours, we reduce hardware resource consumption (LUTs) by 90%. Literature [8] Propose a method of using fast algorithm to implement deconvolution on hardware, its frequency is lower than ours, and the resource consumption (LUTs and FFs) is 45% higher than ours. Literature [9] proposes a method to convert the deconvolution layer into a convolution layer. Literature [8] adopted the method of literature [9], and its frequency was slightly lower than this paper, and the resource consumption (LTS and FFS) was 20.3% higher than our study. literature [10] proposes a method to optimize data flow to improve performance, but its resource consumption (LUTs and FFs) has increased by 50.7% compared to this paper.

V. Conclusions

In the process of GAN operation, the calculation process of deconvolution is more complicated than that of convolution. This paper proposes a reconfigurable implementation scheme that takes into account high parallelism and low on-chip resource overhead. By using a reconfigurable mechanism, two type of convolution functions can be flexibly switched on the same PEG, and the utilization of on-chip resources can be improved. The results show that at 150MHz operating frequency, it has obvious advantages compared with other FPGA platform implementations.

ACKNOWLEDGMENT

This research is supported by the National Natural Science Foundation of China (No.61834005, 61772417, 61802304, 61602377, 61874087, 61634004), and the Shaanxi province key R&D plan (NO.2021GY-029, 2021KW-16).

REFERENCES

[1] I.Goodfellow, J.Pouget-Abadie, M.Mirza, B.Xu, D.Warde-Farley, and S.Ozair, "Generative Adversarial Nets," In Advances in Neural Information Processing Systems 27, no.27, 2014, pp.2672–2680.

[2] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2015, pp. 161–170.

[3] J.L.Yan, S.Y.Yin, F.B.Tu, L.B.Liu, and S.J.Wei, "GNA: Reconfigurable and efficient architecture for generative network acceleration," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.37, no.11, 2018, pp. 2519-2529.

[4] D.W.Xu, K.J.Tu, Y.Wang, C.Liu, B.S.He, and H.W.Li, "FCN-engine: accelerating deconvolutional layers in classic CNN processors," the International Conference. IEEE, 2018.

[5] A.Yazdanbakhsh, M.Brzozowski, B.Khaleghi, S.Ghodrati, and K.Samadi, "Flexigan: An end-to-end solution for FPGA acceleration of generative adversarial networks," 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE,2018, pp.65-72.

[6] S.L.Liu, H.X.Fan, X.Y.Niu, and H.C.Ng, "Optimizing CNN-based segmentation with deeply customized convolutional and deconvolutional architectures on FPGA," ACM Transactions on Reconfigurable Technology and Systems, vol.11,no. 3, 2018, pp.1-22.

[7] X.Y.Zhang, S.Das, O.Neopane, and K.Kreutz-Delgado, "A Design Methodology for Efficient Implementation of Deconvolutional Neural Networks on an FPGA," ArXiv Preprint ArXiv:1705.02583, 2017.

[8] J.W.Chang, S.Ahn, K.W.Kang, and S.J.Kang, "Towards design methodology of efficient fast algorithms for accelerating generative adversarial networks on FPGAs," 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2020, pp.283-288.

[9] J.W.Chang, and S.J.Kang, "Optimizing FPGA-based convolutional neural networks accelerator for image super-resolution," 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2018, pp.343-348.

[10] J.W.Chang, K.W.Kang, and S.J.Kang, "An energy-efficient FPGA-based deconvolutional neural networks accelerator for single image super-resolution," IEEE Transactions on Circuits and Systems for Video Technology, vol.30, no.1, 2018, pp. 281-295.