Block Convolution: Towards Memory-Efficient Inference of Large-Scale CNNs on FPGA

Gang Li*[1,2], Fanrong Li*[1,2], Tianli Zhao[1], Jian Cheng[1,2,3]
[1]National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences
[2]University of Chinese Academy of Sciences
[3]CAS Center for Excellence in Brain Science and Intelligence Technology
Beijing, China
Email: {gang.li, jcheng}@nlpr.ia.ac.cn

***Abstract*—FPGA-based CNN accelerators are gaining popularity due to high energy efficiency and great flexibility in recent years. However, as the networks grow in depth and width, the great volume of intermediate data is too large to store on chip, data transfers between on-chip memory and off-chip memory should be frequently executed, which leads to unexpected off-chip memory access latency and energy consumption. In this paper, we propose a block convolution approach, which is a memory-efficient, simple yet effective block-based convolution to completely avoid intermediate data from streaming out to off-chip memory during network inference. Experiments on the very large VGG-16 network show that the improved top-1/top-5 accuracy of 72.60%/91.10% can be achieved on the ImageNet classification task with the proposed approach. As a case study, we implement the VGG-16 network with block convolution on Xilinx Zynq ZC706 board, achieving a frame rate of 12.19fps under 150MHz working frequency, with all intermediate data staying on chip.**

I. Introduction

Recently, deep convolutional neural networks (CNNs) have shown impressive performance on a large variety of tasks while suffering from huge computational complexity. Because of the high power consumption and low flexibility, GPU is not the ideal solution in embedded settings such as wearable devices, unmanned drones, etc. Consequently, FPGA-based CNN accelerators have attracted enormous research interests in both industrial and academic communities.

Since FPGA often offers limited on-chip memory and off-chip bandwidth, there are three main characteristics of modern large-scale CNN architectures that challenge FPGA-based accelerator design: (1) enormous computation operations, (2) massive network parameters, (3) huge volume of intermediate data. The first issue can be alleviated by maximizing resource utilization and computing parallelism to achieve higher throughput. As for the large number of parameters, techniques such as low-bit quantization [1], network pruning [2] and low-rank method [3] are widely adopted to reduce the memory footprint and transfer size.

However, in terms of the huge bulk of intermediate data, there is a lack of efficient solutions structured for FPGAs from the algorithm side. Prior works either resort to smaller networks at the cost of performance and scalability, or simply conduct frequent off-chip transfers [4]–[9], which is inefficient with respect to latency. We observe that the bottleneck lies in the inherent property of data dependencies in CNN structures, that is, a current layer's computing cannot be executed until all its input information is available. Most recently, fusion-based method trying to fuse multiple layers to save memory transfers becomes a promising solution [10], [11]. However, it often leads to unexpected design overhead with the network structure untouched.

In this paper, we investigate the opportunity of approximating the convolution operations in traditional CNNs, which can be efficiently structured for memory-limited FPGAs while maintaining or improving accuracy. The major contributions in this work are threefold.

- We propose a novel and simple algorithm named block convolution, which is hardware-friendly convolution that eliminates data dependencies between adjacent tiles. Extensive results show that comparable or superior accuracy can be achieved on the ImageNet classification task.
- We propose a class of layer-wise fusion structures based on block convolution, which further extends the design space of FPGA-based CNN accelerators.
- To our best knowledge, we are the first to implement the very large VGG-16 network [12] on memory-limited FPGA with all intermediate data storing on chip while achieving a frame rate of 12.19fps, which is the best record among 28nm FPGAs.

II. Block Convolution

A. Motivation

The memory footprint of CNNs is a big challenge in accelerator design. For memory-limited FPGAs, it is not possible to store all the intermediate data on chip during inference, even for the first few layers in large-scale networks such as VGG-16. Thus, external DRAM should be used to buffer these data. However, the massive off-chip transfers are inefficient in terms of latency and power consumption, since the cost on DRAM is several orders of magnitude of SRAM [2]. In this scenario, it is necessary to reduce the workloads on external DRAM.

For most CNN structures, the first few layers cause the bottleneck as a result of dominant intermediate data size. If

*The first two authors contribute equally to this paper.

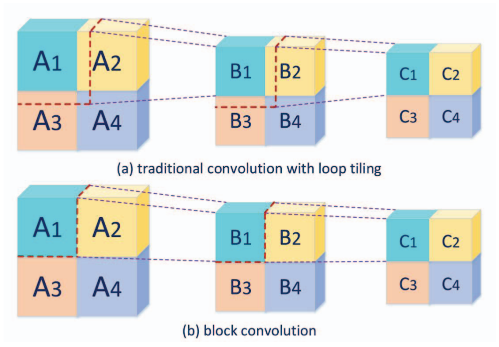


Fig. 1. Comparison between traditional convolution with loop tiling and block convolution.

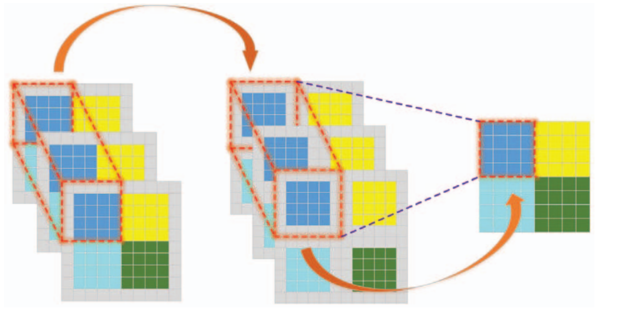


Fig. 2. An example of block convolution. Each batch of tiles is implicitly padded at the boundary, then convolution is executed to derive the corresponding tiles to form an output feature map.

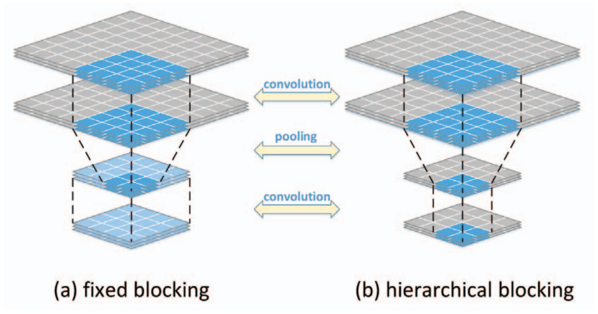


Fig. 3. Blocking styles based on block convolution.

multiple successive layers can be fused by skipping several bottleneck layers, the problem then can be alleviated [10], [11]. However, in traditional CNN structures, a second convolutional layer's computing relies on all the output data of the first one, it is difficult to buffer the intermediate data without resorting to off-chip memory. Though loop tiling can be introduced to partition one layer into several groups and do computation in batches, data dependencies caused by convolution in adjacent tiles still remain, which also poses a big pressure on on-chip storage.

Fig. 1(a) illustrates traditional convolution with loop tiling in three successive convolutional layers. It can be seen that when tiles B_1 are obtained, they can not be used immediately for calculating C_1, as C_1 also depends on partial data of B_2, B_3 and B_4. These data dependencies will either increase the overhead of on-chip storage or increase the extra DRAM access latency caused by physical non-contiguous data rearrangement.

In this paper, we propose a block convolution approach, which is a memory-efficient approximation of original convolution, as shown in Fig. 1(b). Block convolution aims to maximize the advantages of multi-layer fusion and loop tiling, thus cross-layer convolution can be efficiently calculated in a straightforward dataflow to reduce the on-chip buffer size in memory-limited FPGAs.

B. Algorithm

We use the example in Fig. 2 to illustrate how block convolution works. In this example, the input tensor consists of three separated channels of size 8×8 with additional zero padding, the filter size is 3×3, and each input feature map is divided into 4 independent tiles. In original convolution, the size of output feature map can be determined as:

$$O_i = (I_i + p_i - k_i)/s_i + 1, i \in \{h, w\} \quad (1)$$

which is 8×8 in this example. Since inter-tile dependencies are eliminated in block convolution, it is not possible to obtain an output tile of size 4×4 directly from three input tiles at the corresponding position. Therefore, we first do *block padding* to provide supplementary data at the boundary in each individual tiles. Specifically, the original size of top-left tiles is 5×5, we simply pad the rightmost and bottommost boundary to make them of size 6×6. Then, convolution in these tiles can obtain an output tile of size 4×4. In this way, a batch of tiles in the input layer provides all the information to derive the output tile in the successive layer.

In terms of block padding, there are two proper methods: *zero padding* and *repeated padding*. Zero padding simply pads the boundary pixels with zeros while repeated padding duplicating the boundary pixels outwards based on the coherency of the pixels in an image. Note that we need neither explicitly pad each tiles nor rearrange memory pattern on hardware, because in both methods, block padding can be merged into the process of convolution either by on-power initialization or manipulating BRAM addressing, which is memory-efficient compared to previous work [4]–[9].

C. Blocking Styles

Based on block convolution, there are two blocking styles that can further extend the design space.

Fixed Blocking: Fixed blocking ensures the input tiles of each convolutional layer are of the same size, as can be seen from Fig. 3(a). This blocking strategy allows the accelerator's architecture to be relatively fixed, all calculations can share the same architecture at the cost of larger on-chip buffer size.

Hierarchical Blocking: Hierarchical Blocking aims to minimize data dependencies between successive layers, as shown in Fig. 3(b). In this way, the entire network is divided into completely independent subnetworks which can be processed either in parallel or serial manner. Besides, hierarchical

TABLE I

Top-1/top-5 accuracy(%) of various blocking configurations on VGG-16. The accuracy of pre-trained model is 71.12/89.94 (%).

	$H_{2\times2}$	$H_{4\times4}$	$H_{8\times8}$	$H_{16\times16}$	F_{112}	F_{56}	F_{28}	F_{14}
Zero padding	69.91 / 89.54	67.92 / 88.38	66.23 / 87.18	63.12 / 84.95	**72.57 / 91.01**	**72.43 / 90.87**	**71.50 / 90.44**	69.06 / 88.79
Repeated padding	68.67 / 88.53	66.49 / 87.33	65.19 / 86.43	64.06 / 85.72	**72.60 / 91.10**	**72.53 / 90.98**	**71.99 / 90.63**	70.29 / 89.55

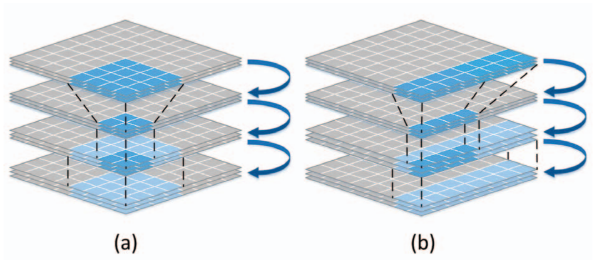
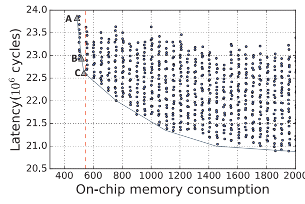
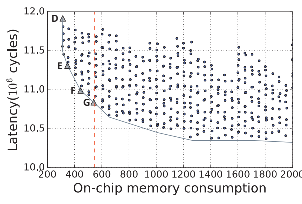


Fig. 4. Fused-layer structures: (a) multi-layer fusion based on square tiling, (b) multi-layer fusion based on rectangular tiling.



(a) 16-bit quantization, 2 PEs



(b) 8-bit quantization, 4 PEs

Fig. 5. Design space exploration according to inference latency and on-chip memory consumption. The x-axis stands for BRAM consumption of a specific fusing configuration. The red dotted line indicates the total amount of available on-chip BRAMs in Xilinx Zynq 7Z045 FPGA used in this work. Points on the left side of the dotted line stand for solutions that can fit all intermediate data on chip.

blocking can significantly reduce the on-chip buffer size, which is well suited for memory-limited settings.

D. Experiments

In this section, we use the very large VGG-16 network to verify the effectiveness of our algorithm. We adopt the Caffe framework [13] to implement the block convolution layer. The top-1/top-5 classification accuracy on the ILSVRC12 ImageNet dataset is shown in Table I.

In our experiments, we test two blocking styles illustrated in section II-C and their combinations. Meanwhile, two padding strategies are also evaluated. Specifically, $H_{i\times i}$ indicates that feature maps in each layer are partitioned into $i \times i$ independent batches of tiles using hierarchical blocking, while in F_i, the tiling size of each layer is fixed to $[i, i]$ by fixed blocking.

From Table I, it can be seen that higher accuracy than the original model can be achieved. Among most configurations, fixed blocking can achieve slightly higher accuracy than hierarchical blocking. We can also notice that zero padding is almost as accurate as repeated padding, which can be efficiently implemented on FPGAs.

III. Accelerator Design and Evaluation

To verify the efficacy of block convolution in accelerator design, as a case study, we implement the VGG-16 network on memory-limited FPGA. In our approach, we fuse several adjacent layers into a single unit. Off-chip transfers are only needed for the input image, network parameters and output predictions. We use the design in [4] as the baseline and build our accelerator upon it.

A. Multi-Layer Fusion Based on Block Convolution

Tiling sizes in height and width dimension are usually set to $T_r = T_c$ in existing work. However, we observe that square tiling is not always the best choice considering the utilization of memory resources, we can simply demonstrate this in the following example.

Suppose a single channel feature map of size 128×128, when square tiling is utilized, it can be partitioned into the size of $128 \times 128, 64 \times 64, 32 \times 32, 16 \times 16$, etc. If the on-chip memory enables to accept a tile of 100×100, the maximal size that can fit on chip, in this scenario, is 64×64, thus the memory utilization is only 40.96%. However, if we use rectangular tiles like 128×64, then memory utilization can be easily doubled. Consequently, we take rectangular tiling into consideration in our multi-layer fusion design, which is shown in Fig. 4.

B. Design Space Exploration

For a given network structure, there are a number of ways to fuse multiple layers, thus it is critical to search the fusing space. Fig. 5 shows the results for VGG-16 with different fusing configurations. Each point in the figure represents a specific fusing strategy with corresponding theoretical inference latency and on-chip memory consumption. For an individual design, we take blocking style, tiling size into consideration to derive latency, and the on-chip BRAM consumption is based on the estimation from Vivado reports. We also implement the accelerators based on different numbers of PEs and quantization bits to further extend the design space.

As illustrated in Fig. 5, for both 8-bit and 16-bit quantization, there are many ways to implement VGG-16 with a high speed and keep intermediate data staying on chip, which is not possible in previous work [4]–[11]. The tiling size (T_r, T_c) in A and D is always $[14, 14]$ through the entire network, which allows them to have a smaller data buffer, and thus reduce the cost of on-chip storage. On the contrary, C and G achieve highest inference speed either by using rectangular tiling or by using hierarchical blocking to fuse more layers into one group, which significantly improves the on-chip memory utilization.

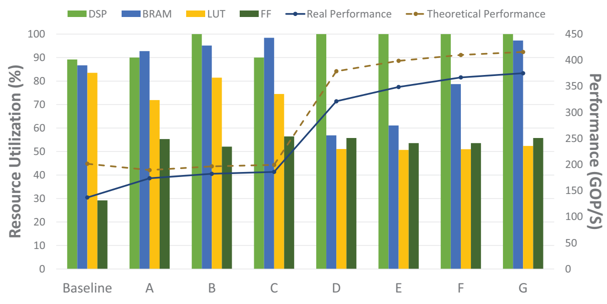


Fig. 6. Comparison of our variant designs with baseline in terms of on-chip resource utilization and performance. The design of the baseline, A, B, C are based on 16-bit precision with 2 PEs, and D, E, F, G are based on 8-bit precision with 4 PEs.

TABLE II
COMPARISON WITH OTHER FPGA ACCELERATORS ON VGG-16

	[4] FPGA'16	**[7] FPGA'16**	**[8] ICCAD'16**	**[9] FPGA'17**	**[6] FPGA'17**	**Ours**
Platform	Zynq ZC706	Stratix-V GSD8	Virtex-7 VX690t	Intel QuickAssist QPI FPGA	Arria-10 GX1150	Zynq ZC706
Precision	16bits fixed	8-16bits fixed	16bits fixed	32bits float	8-16bits fixed	8bits fixed
Technology	28nm	28nm	28nm	28nm	20nm	28nm
Frequency (MHz)	150	120	150	200	150	150
BRAMs	1090×18k	2567×20k	2940×18k	2560×20k	2713×20k	1090×18k
DSPs	900	1963	3600	512	1518	900
Performance (GOP/s)	136.97	117.8	354	123.48[a]	645.25	374.98
Latency/Image (ms)	224.6	262.9	87.29	263.27[a]	47.97	82.03
Intermediate data transfer	Yes	Yes	Yes	Yes	Yes	No

[a]The original paper only report results in convolutional layers.

C. Evaluation

Our implementation of VGG-16 is built on the Xilinx Zynq ZC706 SoC development board, which consists of an ARM processor and a Xilinx Kintex-7 FPGA with 19.1 Mbits on-chip memory.

We first evaluate the resource utilization and inference latency of our variant designs, as shown in Fig. 6. Compared to the baseline, although we move the intermediate data buffer completely from external DRAM to on-chip memory, there is only about 10% increase on BRAM consumption. This is mainly due to the efficient block convolution involved in the fusing architectures. Moreover, the real speed of our designs is also higher than the baseline. We attribute this to the following reasons: (1) With block convolution, all the network intermediate data can stay on chip, which significantly reduces the DRAM access latency as well as CPU interrupts. (2) Rectangular tiling can further reduce the latency introduced by CPU interrupts compared to square tiling; (3) CPU interrupts caused by filter transfers can be significantly decreased by loading more filters at a time.

From Fig. 6, we can also notice that there is a gap between theoretical performance and real performance in our design. This is mainly caused by frequent CPU interrupts in the process of filter transfers. If we can enlarge the on-chip filter buffers to reduce these transfers, this gap is likely to be narrowed. However, this happens at the cost of extra on-chip memory requirements.

Finally, we compare our design G in Fig. 6 against a few existing works, as shown in Table II. Our solution achieves the highest performance (GOP/s) among 28nm FPGAs with the minimum amount of on-chip memory. Although better performance on more advanced FPGA can be obtained by utilizing more memory and DSPs [6], we are the first to implement the very large VGG-16 network on low-cost FPGA without intermediate data caching on external DRAM. Our solution is memory-efficient, and can achieve high performance under a lower budget, which is more promising in mobile applications and emerging technologies such as IoT.

IV. Conclusion and Future work

In this paper, we propose a simple yet effective algorithm named block convolution, which aims at facilitating the deployment of large-scale CNNs on memory-constraint FPGAs. Block convolution eliminates inefficient inter-tile dependencies so that convolution for each individual tiles can be calculated independently. Experiments on the ImageNet classification task demonstrate that the proposed approach can gain even higher accuracy. As a case study, we implement the very large VGG-16 network on Xilinx Zynq ZC706 development board, achieving a frame rate of 12.19fps with all the intermediate data staying on chip throughout the whole process of network inference.

V. Acknowledgement

This work was supported in part by National Natural Science Foundation of China (No.61332016), the Scientific Research Key Program of Beijing Municipal Commission of Education (KZ201610005012) and Jiangsu Key Laboratory of Big Data Analysis Technology.

References

[1] P. Gysel, M. Motamedi and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," *ICLR*, 2016.
[2] S. Han, J. Pool, J. Tran and W. J. Dally, "Learning both weights and connections for efficient neural networks, " *NIPS*, 2015.
[3] P. Wang, J. Cheng, "Accelerating convolutional neural networks for mobile applications, " *MM*, 2016, pp. 541-545.
[4] J. Qiu et al., "Going deeper with embedded fpga platform for convolutional neural network," *FPGA*, 2016.
[5] C. Zhang et al., "Optimizing fpga-based accelerator design for deep convolutional neural networks, " *FPGA*, 2015.
[6] Y. Ma, Y. Cao, S. Vrudhula, J. Seo, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," *FPGA*, 2017.
[7] N. Suda et al., "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," *FPGA*, 2016.
[8] C. Zhang, Z. Fang, P. Zhou, P. Pan, J. Cong, "Caffeine: Towards uniformed representation and accelerator for convolutional neural networks,," *ICCAD*, 2016.
[9] C. zhang, V.Prasanna, "Frequency domain acceleration of convolutional neural networks on cpu-fpga shared memory system," *FPGA*, 2017.
[10] M. Alwani, H. Chen, M. Ferdman, P. Milder, "Fused-layer cnn accelerators," *MICRO*, 2016.
[11] Q. Xiao, Y. Liang, L. Lu, S. Yan, Y. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas," *DAC*, 2017.
[12] K. Simonyan, A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.
[13] Y. Jia et al., "Caffe: convolutional architecture for fast feature embedding," *MM*, 2014.