IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 30, 2019, accepted May 16, 2019, date of publication May 28, 2019, date of current version June 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2919527

A Resources-Efficient Configurable Accelerator for Deep Convolutional Neural Networks

XIANGHONG HU[1], YUHANG ZENG[1], ZICONG LI[1], XIN ZHENG[1], SHUTING CAI[1], AND XIAOMING XIONG[1,2]

[1]School of Automation, Guangdong University of Technology, Guangzhou 510006, China
[2]Company of Chipeye Microelectronics Foshan Ltd., Foshan 528225, China

Corresponding author: Xiaoming Xiong (xmxiong@gdut.edu.cn)

This work was supported by the Science and Technology Planning Project of Guangdong Province of China under Grant 2017B090909004 and Grant 2017B010124003.

ABSTRACT Deep convolutional neural networks (DCNNs) have become one of the most popular approaches to many visual processing tasks. The majority of existing works on the accelerating DCNNs focus on high performance while neglecting the hardware resource utilization, like on-chip memory and DSP. In this paper, we propose a resources-efficient and configurable DCNN accelerator. A four-level processing-element (PE)-array-based structure is presented to realize high parallelism calculation of convolutional operation, and a new storage pattern named hybrid stationary (HS) is proposed to take full advantage of the used on-chip memory footprint and limited off-chip memory bandwidth. Moreover, roofline model is adopted to explore the design space of the given hardware resources. The proposed architecture achieves 113 G-ops/s at 100 MHz and consumes 784 DSP48 modules and 211.5 Block RAM modules on ZYNQ-7 ZC706 evaluation board. To the best of our knowledge, the proposed accelerator is the only implemented system on FGPA platform that can achieve multiple advantages: high-performanced, configurable, and efficient in power and resources utilization. It shows significant utilization improvement compared with the other available architectures.

INDEX TERMS Deep convolutional neural networks, resources-efficient, configurable, accelerator.

I. INTRODUCTION

Deep Convolutional Neural Networks (DCNNs) have been successfully applied in various modern AI applications from object detection, image classification [1]–[4], scene understanding to natural language processing [5]. However, the high computational complexity of those available DCNNs brings huge challenges to the embedded hardware designers. Recently, DCNNs have been accelerated on computational platforms including Graphical Processing Unit (GPU), Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC). The hardware and power consumption of GPU are too high for embedded devices. ASICs can efficiently reduce DCNNs' power consumption and improve hardware utilization, but it has significant fabrication cost and limited flexibility. FPGA accelerators have the advantages of low power consumption, high configurability and reasonable price. These advantages make FPGA attractive for DCNN implementations on embedded systems.

The associate editor coordinating the review of this manuscript and approving it for publication was Junxiu Liu.

Many DCNN accelerators have been proposed [6]–[25]. However, while pursuing higher performance, most of FPGA accelerators do not pay attention to hardware cost, especially like DSP and on-chip memory resource. Zhang *et al.* [6] and Motamedi *et al.* [14] identify the best system performance with roofline model based on the peak performance and off-chip memory traffic provided by the hardware platform. With roofline model, the parallelism of convolution is explored and the optimal tiling parameters are obtained. Zheng *et al.* [7] introduce an accelerator, named Caffeine, with uniformed convolutional matrix-multiplication for convolutional layers and fully connected layers by High Level Synthesis. Rahman *et al.* [8] introduce an architecture ICAN, which is a 3D neuron array architecture. Those accelerators [6]–[8], [14], [18] consume a large number of DSPs or specialized processing elements (PEs) and on-chip memory to enhance performance.

A few works focus on the utilization of DPS resources, such as [9], [15], and [16]. However, they do not make much efforts to reduce on-chip memory usage. Moini *et al.* [9] take most of on-chip memory to load the necessary data only

once and store it on on-chip memory. NullHop, proposed by Aimar *et al.* [15], exploits the sparsity of neuron activations in DCNNs to accelerate the computation. Snowflake, presented by Gokhale *et al.* [16], is a scalable accelerator that is designed to always perform at near-peak hardware utilization. NullHop and Snowflake costs 1280 kB and 768 kB on-chip memory for feature map and kernel data, respectively. When on-chip memory is limited, optimizing the tiling strategy and data storage on chip has great significance.

Storage pattern is often exploited to minimize the energy cost of data transfers between off-chip and on-chip memory. Output stationary storage pattern is adopted through the whole network in [6], [9], and [14]–[16], while input stationary storage pattern is employed in [8] and weight stationary storage pattern is used in [7]. All the works above choose only one storage pattern for all convolutional layers to decrease data transfers. In fact, there is always an optimal storage pattern for different layers. Therefore, storage pattern in each layer should be configurable in an energy-efficient DCNN accelerator.

In this paper, we present a resources-efficient and configurable DCNN accelerator. The accelerator allows high utilization of the used computing and storage resources across kernel sizes ranging from 1×1 to 11×11 and various kernel strides. We design a four-level structure based on a 14×14 PE array, which is different from 16×16 PE array [10]. We improve the PE utilization by simply adjusting the PE array size to 14×14. Different from to the four-level structure in [10], we place weight register (WREG) and partial sum register (PREG) in second level and simplify output register (OREG) in the fourth level. A new storage pattern, called hybrid stationary (HS), reconfigures the data storage pattern in on-chip memory to reduce off-chip memory access and fully exploits limited off-chip bandwidth. In order to improve the performance, roofline model is applied to explore the design space of the proposed accelerator to find the optimal tiling parameter combination. A two-step strategy is proposed to improve the performance and the efficiency of resource utilization. The first step is to find the optimal tiling parameter combination of each storage pattern and the second step is to select the storage pattern.

In the following: Section II provides a DCNN's preliminary and tiling methodology; Section III presents HS storage pattern and corresponding resources-efficient architecture; Section IV shows the experimental results and comparison with other accelerators; Section V concludes this paper.

II. PRELIMINARY

A. DEEP CONVOLUTIONAL NEURAL NETWORKS

Figure 1 shows a typical convolutional layer in DCNN. It has $N \times H \times L$ input feature maps (Imap) and M 3-D convolutional kernels ($K \times K \times N$). Each kernel is applied to all the Imap with a 3-D convolution operation and the $R \times C$ output feature maps (Omap) are generated.

AlexNet, a well-know DCNN, has been accelerated in many researches [6], [7]–[12], and [16]. AlexNet is always

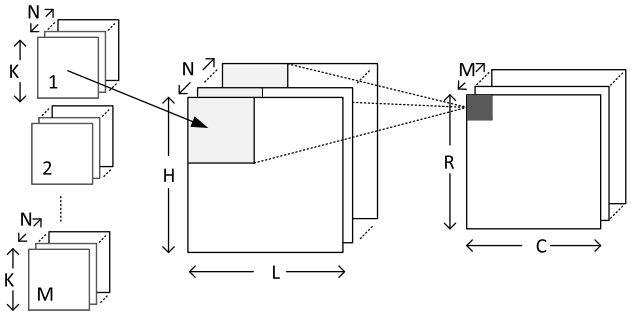


FIGURE 1. Illustration of a convolutional layer.

TABLE 1. Parameters of each convolutional layers.

Layer	1	2	3	4	5	total
H, L	224	27	13	13	13	-
N	3	96	256	384	384	-
R, C	11	5	3	3	3	-
M	96	256	384	384	256	-
K	11	5	3	3	3	-
$Stride$	4	1	1	1	1	-
$\#Weight$	34.8 k	307.2 k	884.7 k	663.6 k	442.4 k	2332.7 k
$\#Imap$	150.5 k	70.0 k	43.3 k	64.9 k	94.9 k	393.6 k
$\#Omap$	290.4 k	186.7 k	64.9 k	54.9 k	43.3 k	650.1 k
$\#MAC$	105.4 M	223.9 M	149.5 M	112.1 M	74.8 M	665.8 M

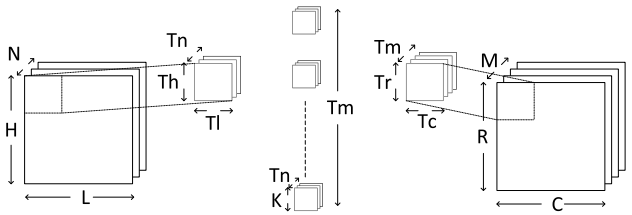


FIGURE 2. Tiling methodology in convolutional layer.

used as a benchmark to test configurability because of its various kernel sizes and convolution strides. Like other DCNN models, AlexNet consists of four types of layer: convolutional layers, pooling layers, activation function layers and fully-connected layers. Our research concentrates on how to accelerate the convolutional layers because it dominates the runtime of up to 90.7% [26].

The parameters of each of AlexNet's convolutional layers are listed in Table 1. AlexNet has five convolutional layers with convolution strides ranging from 1 to 4 and kernel sizes ranging from 3×3 to 11×11. The last four rows of the table list the total number of kernel weight, Imap, Omap, and Multiply-and-Accumulate (MAC) for each layer. There are more than 665 million MAC operations totally in five convolutional layers.

B. TILING METHODOLOGY

In general, hardware resources in embedded system, such as DSP and on-chip memory, are very limited. Therefore, it is difficult to compute and save an entire convolutional layer simultaneously. The tiling methodology is a common method to resolve the problem of the limited on-chip resources. Figure 2 shows a tiling strategy in a convolutional layer.

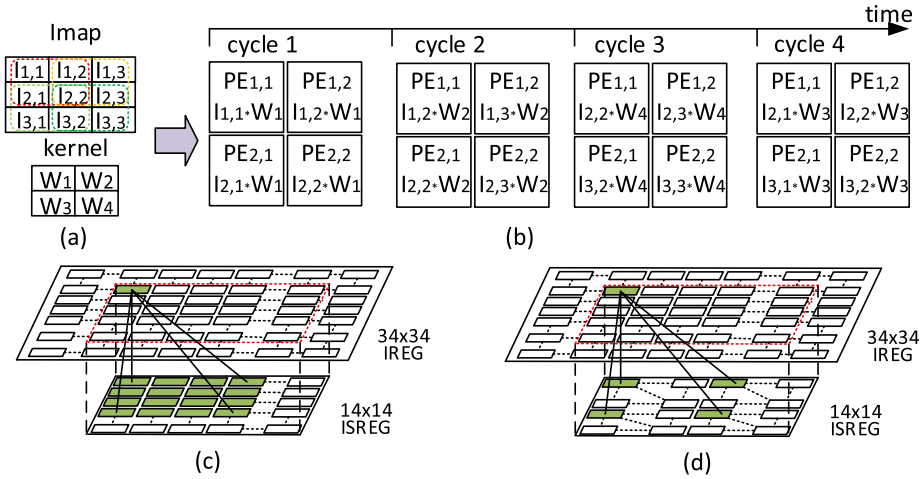


FIGURE 3. Output-oriented mapping architecture (a) and (b) Convolution on a 2 × 2 PE array in the case $Th = Tl = 3$, $K = 2$, and $S = 1$. (c) $S = 4$, IREG broadcasts significant pixels to ISREG. (d) $R = 7$ and $S = 1$, IREG broadcasts four pixels to ISREG.

Each tiling convolution operation is that Tm 3-D kernel ($K \times K \times Tn$) convolves on $Tn \times Th \times Tl$ Imap and generates $Tm \times Tr \times Tc$ partial sum (Psum) or Omap.

III. ARCHITECTURE DESIGN AND ACCELERATION STRATEGY

A. CONVOLUTION MAPPING STRUCTURE

In this part, we first introduce a mapping method called Parallel Output Oriented Mapping (POOM), which was designed to improve PE utilization in large stride case and irregular map size case [10]. Then, we give a four-level convolution mapping structure based on PE array, which takes advantage of the design of POOM for large stride case but not for irregular map size case.

1) POOM

Each PE computes one output pixel by accumulation. As shown in Figure 3 (a), a 2 × 2 kernel convolves a 3 × 3 Imap with stride 1 on a given 2 × 2 PE array. In Figure 3 (b), the $PE_{1,1}$ performs $O_{1,1} = I_{1,1} \times W_1 + I_{1,2} \times W_2 + I_{2,2} \times W_4 + I_{2,1} \times W_3$ with 4 cycles. In cycle 1, the 2 × 2 PE array loads pixel data $\{I_{1,1}, I_{1,2}, I_{2,1}, I_{2,2}\}$ of Imap and the first kernel weight W_1 to generate product. In cycle 2, the PEs load pixel data $\{I_{1,2}, I_{1,3}, I_{2,2}, I_{2,3}\}$ and the second weight W_2 to accumulate product. In cycle 3, the PEs load pixel data $\{I_{2,2}, I_{2,3}, I_{3,2}, I_{3,3}\}$ and the fourth weight W_4 to accumulate product. In cycle 4, the PEs load pixel data $\{I_{2,1}, I_{2,2}, I_{3,1}, I_{3,2}\}$ and the third weight W_3 to accumulate product. Since the Imap are stored in Imap array register (IREG) and the whole Imap shift in S-shape during convolution, it is convenient to bring needed pixels into PEs at each cycle. In a large stride case [10], such as when stride = 4, one input pixel in IREG is broadcasted to sixteen pixels in Imap share array register (ISREG), as shown in Figure 3 (c). The different weights of sixteen output channels are also transferred to WREG, and Omap of sixteen output channels are generated. When Omap's size (R) is equal to 7 × 7 and kernel stride is equal to 1, like the "conv5_x" layer of ResNet [4], one input pixel in IREG is broadcasted to four pixels in ISREG and Omap of four channels are generated at the same time, as shown in Figure 3 (d).

2) CONVOLUTIONAL STRUCTURE

In this work, the size of PE array is cut from 16 × 16 to 14 × 14, which can reduce design complexity and remain a good PE utilization in irregular map size case. Compared to 16 × 16, the 14 × 14 PE array is more suitable for modern DCNNs, like AlexNet or VGG. That is because the row and column number of feature maps in modern DCNN are mostly a multiple of 14 and our tiling Omap will fit for the PE array leaving few PEs idle. If the PE array size is larger than 14 × 14, the computation of tiling Omap on the margin of the whole Omap will not make full use of all PEs, causing low PE usage. On AlexNet, our PE utilization is 86% of layer 3 ∼ 5 while 84% of that in [10], 68% of layer 1 and 93% of layer 2. On VGG-16, our PE utilization are 100% of all convolutional layers.

Figure 4 shows the four-level structure of ConvCore. The first level is two IREGs which take pixels from Input data Buffer (InBuf) in ping-pong mode. The second level consists of PREG, WREG and ISREG. ISREG and WREG are placed to improve PE utilization ratio. PREG is designed for Psum data and bias data in the subsequent convolution operation. The third level is a PE array. The fourth level is OREG. The MAC block consists of PREG, WREG, OREG and PE array. The following example illustrates how it works: $Tm = 1$, $Tn = 1$, $Th = Tl = 15$, $Tr = Tc = 14$, $K = 2$, $S = 1$. Cycle 1: The Imap of $Tn = 1$ reaches to IREG; Cycle 2: IREG broadcasts corresponding pixels to ISREG and shifts left, meanwhile Weight Buffer (WBuf) broadcasts the first weight to WREG. Cycle 3∼6: PE array performs the multiply-and-accumulate operation on the pixels and the weight. Cycle 7:

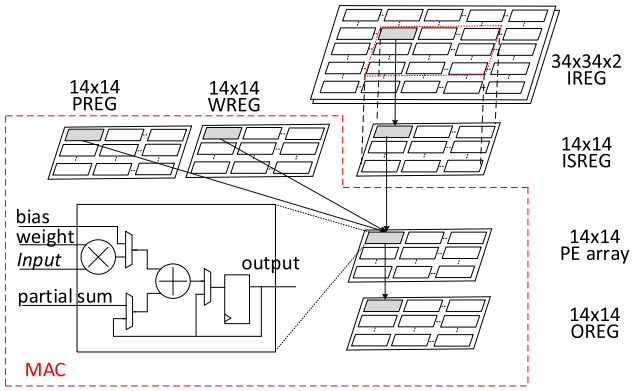


FIGURE 4. Four-level structure of ConvCore.

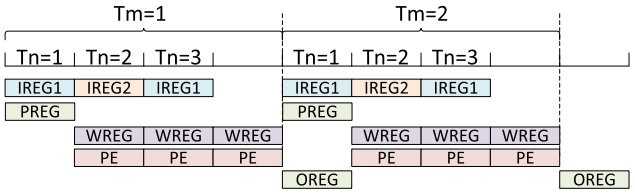


FIGURE 5. Pipelining design.

OREG receives output pixels from PEs. Finally, Omap are stored to Output data Buffer (OutBuf). If $Tn \geq 1$, the Psum are kept in PE until Tn loop finishes.

3) PIPELINING DESIGN

The four-level structure is built on a basic idea of parallelization, in which double IREGs operate in a ping-pong manner to prepare data for convolution computation. It is noted that loading Imap into IREG, loading Psum into PREG and outputting Omap from OREG usually take more than one clock cycle due to the limited on-chip network bandwidth. However, only IREG need to load Imap continuously. PREG and OREG are designed with one register array while IREG are two-fold. WREG can load weight in one clock cycle because all PEs receive the same weights at the same moment or the on-chip network bandwidth supports simultaneous sixteen weights when stride is 4. Therefore, the transfer time of WREG is overlapped with convolution computation time, which means it takes no extra clocks to preload.

Figure 5 gives an example that demonstrates how convolution of a tiling block runs: $Tm = 2$, $Tn = 3$. IREG1 and IREG2 alternately load Imap of different channels from InBuf. Once IREG1 finishes loading data, PE array performs calculation; meanwhile IREG2 prepares data of next channel. After Omap of a channel is generated and stored in OREG, OREG writes Omap to OutBuf while IREG1 loads Imap again.

The run time T_{tiling} of one tiling convolution operation is given as follows.

$$T_{tiling} = (\lceil\frac{Th}{4}\rceil\lceil\frac{Tl}{8}\rceil + max(\lceil\frac{Th}{4}\rceil\lceil\frac{Tl}{8}\rceil, K\times K)\times Tn)\times Tm + T_{pipeline} \quad (1)$$

where $\lceil\frac{Th}{4}\rceil\lceil\frac{Tl}{8}\rceil$ gives the number of transfer cycle from InBuf to IREG in Convolution Core (ConvCore). $max(\lceil\frac{Th}{4}\rceil\lceil\frac{Tl}{8}\rceil, K \times K) \times Tn$ stands for the number of clock cycle for convolution of Tn input channels and is decided by the max time of between loading data and convolution operation. When kernel size is very small ($K \leq 2$), it takes longer to load data from InBuf to IREG than convolution operation because of limited on-chip bandwidth. $T_{pipeline}$ denotes the number of transfer cycle using pipeline.

B. STORAGE PATTERN

There are some works not only focusing on high performance but also on reducing the usage of on-chip memory, like [10], [12], [19], and [20]. They are implemented on ASIC platform and consume less than 300k of on-chip memory for convolutional data (Imap, Omap, Psum and weight). To our knowledge, no FGPA-based accelerator running large scale DCNN has considered optimizing the on-chip memory efficiency. Limiting the use of on-chip memory results in increased data transfer between on-chip memory and off-chip memory. According to Tu *et al.* [10], hybrid data reuse is an advanced data reuse model to reduce off-chip memory access. In this work, this method has been improved to make it suitable for increasing the on-chip memory efficiency. The on-chip memory here consists of WBuf, InBuf, Partial sum Buffer (PBuf) and OutBuf. They store only two tiling data, one for the current tiling convolution operation and the other for the next tiling convolution operation. The total number of off-chip memory access can be handled by equations as follows.

$$A_{total} = \alpha_{in}\beta_{in} + \alpha_{wght}\beta_{wght} + \alpha_{out}\beta_{out} \quad (2)$$

where α_{in}, α_{wght}, α_{out} and β_{in}, β_{wght}, β_{out} denote the trip counts and tiling data size of Imap, weight and Omap (or Psum), respectively. α_{in}, α_{wght}, α_{out} are determined by storage pattern and will be given in below. β_{in}, β_{wght}, β_{out} are calculated by equations as follows.

$$\begin{cases} \beta_{in} = Tn \times 4\lceil\frac{Th}{4}\rceil 8\lceil\frac{Tl}{8}\rceil \\ \beta_{wght} = N_{MAC}TmTnKK \\ \beta_{out} = N_{MAC}Tm \times 4\lceil\frac{Tr}{4}\rceil 8\lceil\frac{Tc}{8}\rceil \end{cases} \quad (3)$$

where N_{MAC} represents the number of MAC block, $4\lceil\frac{Th}{4}\rceil 8\lceil\frac{Tl}{8}\rceil$ and $4\lceil\frac{Tr}{4}\rceil 8\lceil\frac{Tc}{8}\rceil$ are the number of transfer data.

Let us consider one convolutional layer. There are M output channels and N input channels in the given convolutional layer. The row and column of Omap are denoted by R and C. K and S are the kernel size and convolution stride. A complete data operation of the convolutional layer can be expressed by the combination of the patterns given in Figure 6. The first three loops (a, b and c) show three tiling techniques which determines storage pattern on chip. The fourth loop (d) demonstrates the convolution of tiling data on chip.

1) OUTPUT STATIONARY

The first common storage pattern is called output stationary (OS), which minimizes the output data access to off-chip memory. OS pattern has three stages: 1) all the tiling Imap in the same spatial position of different channel (N) and related weight are sequentially loaded into InBuf and WBuf; 2) the corresponding Psum are updated and kept within OBuf for reuse; and 3) the tiling Omap are generated and written to off-chip memory. OS keeps Psum in OutBuf and does not writes Psum to off-chip memory until Omap are generated and written to off-chip memory. As shown in Figure 6 (a), the stage 1 and 2 correspond to innermost loop N. The trip counts can be expressed as follows.

$$\begin{cases} \alpha_{in} = \lceil \dfrac{M}{N_{MAC}Tm} \rceil \dfrac{N}{Tn} \dfrac{H}{Th} \dfrac{L}{Tl} \\ \alpha_{wght} = \dfrac{M}{N_{MAC}Tm} \dfrac{N}{Tn} \dfrac{R}{Tr} \dfrac{C}{Tc} \\ \alpha_{out} = \dfrac{M}{N_{MAC}Tm} \dfrac{R}{Tr} \dfrac{C}{Tc} \end{cases} \tag{4}$$

2) INPUT STATIONARY

Input stationary (IS) storage pattern minimizes the input data access to off-chip memory. IS pattern also has three stages: 1) the tiling Imap are loaded and kept within InBuf for reuse, and the related weight are sequentially loaded into WBuf while the related Psum are loaded in PBuf if necessary; 2) the corresponding Psum in the same spatial position of different channel (M) are computed and written to off-chip memory; 3) the tiling Omap are generated and written to off-chip memory. Note that each tiling Imap will be loaded only once from off-chip memory. In Figure 6 (b), the stage 1 and 2 correspond to innermost loop M. The trip counts can be expressed in equations.

$$\begin{cases} \alpha_{in} = \dfrac{N}{Tn} \dfrac{H}{Th} \dfrac{L}{Tl} \\ \alpha_{wght} = \dfrac{M}{N_{MAC}Tm} \dfrac{N}{Tn} \dfrac{R}{Tr} \dfrac{C}{Tc} \\ \alpha_{out} = \begin{cases} 2(\dfrac{N}{Tn} - 1) \dfrac{M}{N_{MAC}Tm} \dfrac{R}{Tr} \dfrac{C}{Tc}, & Tn < N \\ \dfrac{M}{N_{MAC}Tm} \dfrac{R}{Tr} \dfrac{C}{Tc}, & Tn = N \end{cases} \end{cases} \tag{5}$$

3) WEIGHT STATIONARY

Weight stationary (WS) storage pattern decreases the weight accesses to off-chip memory. WS pattern has four stages: 1) the tiling weights are loaded and kept within WBuf for reuse, and relevant tiling Imap are loaded in InBuf while the related Psum are loaded in PBuf if necessary; 2) the corresponding Psum are updated and written to off-chip memory; 3) all the Psum in the different spatial position ($R \times C$) of same channel are generated and written to off-chip memory; 4) all the tiling Omap are generated and written to off-chip memory. Note that each tiling weight will be loaded only once from off-chip memory. Three stage are the explanation of the innermost two loops R and C in Figure 6 (c). The trip counts can be expressed as follows.

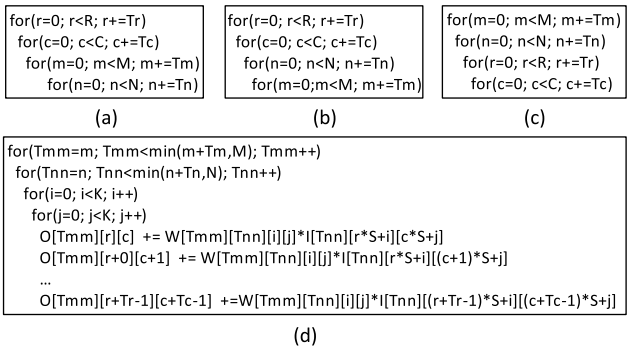


FIGURE 6. Parallelization schemes in tiling methodology. (a) OS storage pattern. (b) IS storage pattern. (c) WS storage pattern. (d) Convolution of tiling data.

$$\begin{cases} \alpha_{in} = \lceil \dfrac{M}{N_{MAC}Tm} \rceil \dfrac{N}{Tn} \dfrac{H}{Th} \dfrac{L}{Tl} \\ \alpha_{wght} = \dfrac{M}{N_{MAC}Tm} \dfrac{N}{Tn} \\ \alpha_{out} = \begin{cases} 2(\dfrac{N}{Tn} - 1) \dfrac{M}{N_{MAC}Tm} \dfrac{R}{Tr} \dfrac{C}{Tc}, & Tn < N \\ \dfrac{M}{N_{MAC}Tm} \dfrac{R}{Tr} \dfrac{C}{Tc}, & Tn = N \end{cases} \end{cases} \tag{6}$$

4) HYBRID STATIONARY

Hybrid stationary (HS) storage pattern combines OS, IS and WS and selects the optimal pattern among them for each layer. Given a system based on the used DSP and on-chip memory resources with limited off-chip memory bandwidth, we propose a two-step strategy, which yields the best performance and the least off-chip memory accesses. We find the optimum tiling parameter combinations $\{Tr, Tc, Tm, Tn\}$ for three storage patterns and then choose the best one. In step 1, given a DCNN layer, all combinations of tiling parameters $\{Tr, Tc, Tm, Tn\}$ are explored to find the optimal one that has the best performance and the least off-chip memory accesses for each storage pattern. In step 2, if the three sets of parameters have the same performance, the storage pattern whose parameter brings the lowest off-chip memory accesses will be chosen. Otherwise, the parameters that have the best performance and the corresponding storage pattern will be selected. The next subsection will introduce in detail how to determine the optimum tiling parameters $\{Tr, Tc, Tm, Tn\}$ and storage pattern.

C. DESIGN SPACE EXPLORATION

In this subsection, we first introduce the roofline model and then explain how to select tiling parameters. Our scheme is to achieve the best performance and the least off-chip memory accesses on the used DSP and on-chip memory resources and the limited off-chip bandwidth.

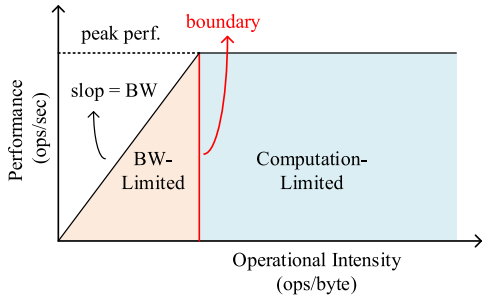


FIGURE 7. Roofline model.

1) ROOFLINE MODEL

To find the optimal parameters of mapping a DCNN into our accelerator, we adopt the well-known roofline model [27] for quantifying the impact of insufficient bandwidth on performance. The roofline model, as shown in Figure 7, is developed to relate attainable performance to off-chip memory bandwidth and the peak performance provided by the hardware platform. When the operational intensity is on the right side of the boundary, the performance will be computation-limited; otherwise, it is bandwidth-limited. Equation (7) formulates the attainable performance on a specific hardware platform.

$$Attainable\ Performance = min \begin{cases} Peak\ Performance \\ Operational\ Intensity \times BW \end{cases} \tag{7}$$

where BW stands for off-chip memory bandwidth.

a: PEAK PERFORMANCE

The proposed convolution mapping structure has been explained in the subsection III.A. Given a specific tiling parameter combination $\{Tr, Tc, Tm, Tn\}$, the computational performance (or peak computation in the roofline model) of this convolution mapping structure can be calculated by Equation (8).

$$\begin{aligned} Peak\ Performance &= \frac{total\ number\ of\ operations}{number\ of\ execution\ cycles} \\ &= \frac{2 \times R \times C \times M \times N \times K \times K}{\frac{1}{N_{MAC}} \lceil \frac{M}{T_m} \rceil \times \lceil \frac{N}{T_n} \rceil \times \lceil \frac{R}{T_r} \rceil \times \lceil \frac{C}{T_c} \rceil \times T_{tiling}} \end{aligned} \tag{8}$$

where T_{tiling} is the run time of one tiling convolution operation and is given in Equation (1).

b: OPERATIONAL INTENSITY

Operational intensity is used to describe the computation operations per byte of off-chip memory traffic. Data reuse optimization will decrease the total number of off-chip memory accesses, thus increase the operational intensity. The operational intensity of the loop unrolling shown in Figure 6 can be calculated by Equation (9).

$$\begin{aligned} Operational\ Intensity &= \frac{total\ number\ of\ operations}{total\ amount\ of\ external\ data\ access} \\ &= \frac{2 \times R \times C \times M \times N \times K \times K}{\alpha_{in}\beta_{in} + \alpha_{wght}\beta_{wght} + \alpha_{out}\beta_{out}} \end{aligned} \tag{9}$$

2) TILING PARAMETERS

Loop unrolling can be applied to increase the utilization of computing resources in FPGA devices. Unrolling along different loop dimensions results in different implementation variables. Without limitation of on-chip memory resources, the space of all legal tiling parameters for architecture we propose are illustrated in Equation (10). The size of PE array is 14×14, therefore Tr and Tc can not be more than 14.

$$\begin{cases} 0 < Tm \le M \\ 0 < Tn \le N \\ 0 < Tr \le min(R, 14) \\ 0 < Tc \le min(C, 14) \end{cases} \tag{10}$$

The on-chip memory, WBuf, InBuf, PBuf and OutBuf, store two tiling data for pipelining. Taking into account the effect of the used on-chip memory, the space of all legal tiling parameters is also limited by Equation (11).

$$\begin{cases} 0 < TnThTl \le \frac{1}{2} Size(InBuf) \\ 0 < N_{MAC} TmTnKK \le \frac{1}{2} Size(WBuf) \\ 0 < N_{MAC} TmTrTc \le \frac{1}{2} Size(OutBuf) \end{cases} \tag{11}$$

3) DESIGN SPACE EXPLORATION

As mentioned above, given a specific data storage pattern and tiling parameter tuple $\{Tr, Tc, Tm, Tn\}$, the peak performance and operational intensity of the architecture can be calculated. Enumerating all possible storage patterns and tiling parameter combinations will generate a series of performance and operational intensity pairs.

Figure 8(a)~8(c) depict all legal solutions of three data storage patterns for layer 3 of the AlexNet in roofline model. The "x" axis denotes the operational intensity (ops/byte) and the "y" axis denotes the performance (G-ops/sec). The slope of the line between any point and the origin point (0, 0) denotes the minimal bandwidth requirement for this implementation. For example, in Figure 8(a), the minimal bandwidth requirement of the implementation of the point D is equal to the slope of the line D'. The flatter slope, the less bandwidth requirement and the off-chip memory accesses.

Step 1 (Parameters): In each data storage pattern of a layer, we explore the architecture-supported design space. And a set of tiling parameter combinations with the highest performance can be collected. If this set only has one implementation combination, then this combination will be the optimal result of design space exploration of this data storage pattern of this layer. However, if there are several counterparts within this set, the one with the highest operation intensity

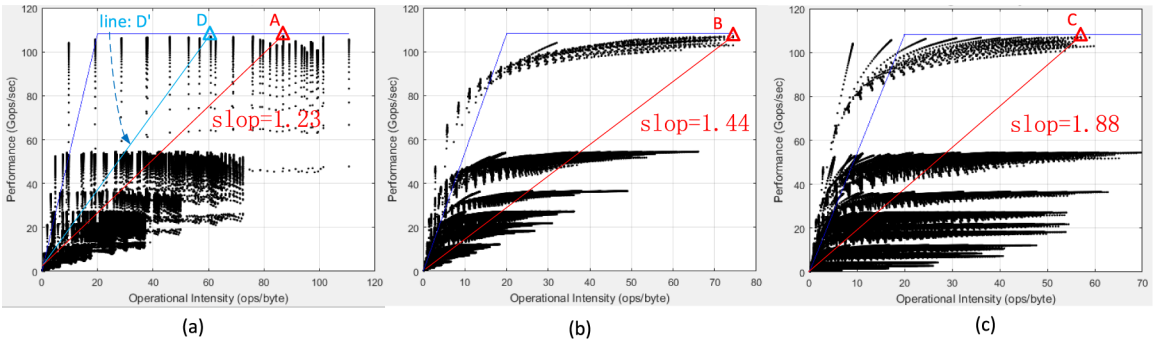


FIGURE 8. Design space of architecture-supported designs. (a) Output Stationary. (b) Input Stationary. (c) Weight Stationary.

value will be picked because this implementation requires the least off-chip memory accesses. For example, in Figure 8(a), the point A and D have the same highest performance and the point A is selected.

Step 2 (Storage Pattern): From step 1 we get the optimal tiling parameter combinations for three storage patterns. In this step, the best storage pattern will be chosen. If there are one storage pattern with a highest performance, this storage pattern will be selected. If two or three storage patterns share the same highest performance, we will choose the one with the highest operation intensity value. For example, in Figure 8, the point A, B and C of three storage patterns share the same highest performance and finally the point A of OS pattern is selected.

D. PROPOSED ACCELERATOR

The main controller of the entire system is the ARM processor on the Processing System (PS) side. As shown in Figure 9, the ARM processor transfers the tiling operation instructions to the Instruction Memory block via an AXI Direct Memory Access (DMA). These instructions consist of convolution parameters, data transfer configurations and off-chip memory address references. The convolution parameters are used by Convolution (Conv) Controller to control the tiling convolution operation and data transfer between the on-chip memory and the Convolution Core (ConvCore). The data transfer configurations are used to configure the data access to off-chip memory for tiling data (all or some of input data, output data and kernel weight). The off-chip memory address references are applied to off-chip memory address generation. The convolutional layer is subdivided into tiling convolution operations and each instruction only contains configuration information about the tiling operation.

On the PS side, ARM processor provides configuration instructions based on different layer characteristics and the parameters of outermost four loops in Figure 6 (a)∼Figure 6 (c). On the Programmable Logic (PL) side, the Top-Level Controller block is a state machine that controls the four loops in Figure 6 (d). The Top-Level Controller block reads instructions from the Instruction Memory block, then coordinates the Transfer (Tran)

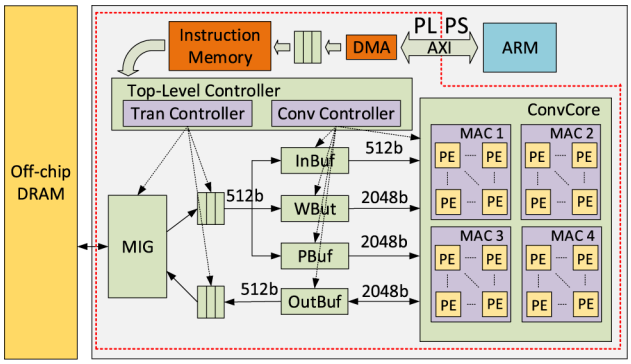


FIGURE 9. The proposed DCNN accelerator with 4 MACs.

Controller block and the Conv Controller block. The Conv Controller block controls the current tiling computations, while the Tran Controller block is responsible for the data preparation of the next tiling convolution.

The total on-chip memory for convolution data, called Data Buffer, consists of InBuf, WBuf, PBuf and OutBuf. To eliminate data loading and reading latency, all buffer can work in the "ping-pong" mode to exchange data with the Memory Interface Generator (MIG) and the ConvCore block. In design with 4 MACs, the InBuf size is 64 kB and four MACs share the same InBuf. The WBuf size and OutBuf size are 128 kB, and each MAC has 32 kB. Each MAC has 36 kB PBuf (32 kB for partial sum and 4 kB for bias) and the total PBuf size is 144 kB. The ConvCore is made up of four MACs, and each MAC contains 14 $\times$ 14 PEs. Four MACs share the same Imap but different weights to generate Omap of different channels. To get high data transferring rate between off-chip memory and on-chip memory, MIG is chosen. It can provide a bandwidth of 5.4GB/s at 100MHz in our experiment.

IV. EXPERIMENTAL RESULT

A. IMPLEMENTATION

The accelerators with 1 MAC to 4 MACs described above are implemented with Verilog-HDL language. To validate these accelerators, we implemented them on the Xilinx

TABLE 2. Resource utilization of the entire system.

Resource	Design	frequency	LUT	LUT (ConvCore)	FF	DSP	BRAM
Utilization/ Percent	4 MAC	100 MHz	149037 (68.18%)	101471 (46.42%)	95284 (21.79%)	784 (87.11%)	211.5 (38.81%)
	3 MAC	112.5 MHz	130330 (59.62%)	82559 (37.77%)	84732 (19.38%)	588 (65.33%)	171.5(31.47%)
	2 MAC	145.45 MHz	112469 (51.45%)	69444 (31.77%)	70519 (16.13%)	392(43.56%)	131.5(24.13%)
	1 MAC	163.64 MHz	93977 (42.99%)	56152 (25.69%)	57033 (13.05%)	196 (21.78%)	99.5(18.26%)

TABLE 3. RAM utilization details.

BRAM	Design	Data Buffer	FIFO	Instruction Memory	DMA	Total
Utilization	4 MAC	160	38	12	1.5	211.5
	3 MAC	120	38	12	1.5	171.5
	2 MAC	80	38	12	1.5	131.5
	1 MAC	48	38	12	1.5	99.5

Zynq XC7Z045 device, respectively. The actual performance results are attained on Mentor ModelSim simulator and the theoretical performance results are from Matlab.

The implementation results show that the accelerator using 1 MAC, 2MACs and 3 MACs can run at the max clock frequency of 163.64 MHz, 145.45 MHz and 112.5 MHz, respectively. These accelerators can support kernel size ranging from 1×1 to 11×11, kernel stride 1/2/4. Since high configurability needs too much wiring resources for multiplexers and results in fail implementation for high congestion during routing design process, the kernel configuration of stride 2 is not supported in the accelerator with 4 MACs. The accelerator containing 4 MACs can run at a frequency of 100 MHz.

Table 2 shows the required resources of the entire system with 1∼4 MACs. The design with 4 MACs costs 784 DSP48 modules and 68.18% LUT resources of the device, and its ConvCore block requires 46.42% LUT resources. The on-chip memory Block RAM (BRAM) utilization details are shown at Table 3. At 4-MACs design, 160 BRAM modules are used for Data Buffer, and others are used for FIFO, Instruction Memory and DMA. The data precision is 16-bit fixed point.

B. EXPERIMENT RESULT AND ANALYSIS

We measure the proposed accelerator's performance on a benchmark suite consisting of AlexNet [1], VGG-16 [2] and ResNet-34 [4]. We choose AlexNet and VGG-16, which are entirely made up of conventional layers, because both models are widely used in other accelerators [6], [7]–[12], [16], [18], [21]–[25] for testing performance. ResNet-34 is selected due to its special residual modules. For better comparison, the performance results below of four designs using 1∼4 MACs are tested at the frequency of 100 MHz. We employ the two-step strategy introduced in the Section III.C to get the optimal parameter combination $\{Tr, Tc, Tm, Tn\}$ and storage pattern for four designs.

1) ALEXNET

Table 4 shows layer-wise storage pattern with the optimal parameter combination and performance for AlexNet among designs using 1∼4 MACs. It is worth noted that the tiling parameter combination $\{Tr, Tc, Tm, Tn\}$ stands for one MAC. That is to say, 4 MACs employ channel parameters Tm and Tn and generate $4Tm$ output channels at the same time but use Tn input channels since all MACs share the same IREG. Table 4 shows four designs have same parameter combination but different storage patterns for each layer except layer 1. When exploring design space in a layer on one of four designs, it's interesting that the optimal parameter combinations with the highest performance are the same among three storage patterns. In all cases of 1∼4 MACs, since the PE utilization of layer 1, layer 2 and layer 3∼5 are 68%, 93% and 86%, respectively, the performance of layer 1 is the lowest while the layer 2 achieves the highest performance. In layer 1, because the stride $S = 4$, the minimum tiling output channel Tm of each MAC is equal to 16. And at 4 MACs, each tiling operation generates 64 tiling output channels. Considering the output channel $M = 96$, there are 64 tiling output channels in the first tiling operation but only 32 tiling output channels in the second tiling operation. At 3 MACs, there are 48 tiling output channels in both tiling operations. Therefore, in layer 1, the average performance of 4 MACs is similar to that of 3 MACs.

Figure 10 shows the off-chip memory access on AlexNet with optimal parameters but different storage patterns across design with 1∼4 MACs. In the first layer, WS can reduce much more off-chip memory access than OS and IS because of the large kernel size of 11x11. As the number of MAC decreases, the total number of off-chip memory access per layer goes up.

2) VGG-16

Table 5 shows layer-wise parameter combination and performance for VGG-16. Four designs have the same parameter combinations except layer 1. In layer 1, the tiling output channel Tm is 16 for 3 MACs and 4 MACs, while 32 for 1 MAC and 2 MACs. The reason is that the output channel $M = 64$ is divided by 4 (4 MACs) to get 16 and is divided by 2 (2 MACs) to get 32. The tiling output channel Tm of layer 1 is not 64 for 1 MAC because of the limited capacity of OutBuf. From Equation (1), we can see that performance drops with Tn. Therefore, low performance in layer 1 comes from the small input channel number N, equaling to 3. Different from VGG-16, small N does not affect performance in layer 1 of AlexNet greatly, since its kernel size is 11×11 and the first term of $\lceil \frac{Th}{4} \rceil \lceil \frac{Tl}{8} \rceil$ can be ignored comparing with the second part of $max(\lceil \frac{Th}{4} \rceil \lceil \frac{Tl}{8} \rceil, K \times K) \times Tn$. For the same kernel size, the performance results of layer 2∼13 of VGG-16 are higher

TABLE 4. Layer-wise parameter and performance on AlexNet.

layer	{Tm,Tn,Tr,Tc}	4 MAC			3 MAC			2 MAC			1 MAC		
		Pattern	Theore. Perf. (G-ops/s)	Actual Perf. (G-ops/s)	Pattern	Theore. Perf. (G-ops/s)	Actual Perf. (G-ops/s)	Pattern	Theore. Perf. (G-ops/s)	Actual Perf. (G-ops/s)	Pattern	Theore. Perf. (G-ops/s)	Actual Perf. (G-ops/s)
1	{16, 3, 3 , 3}	WS	72.33	72.28	WS	72.29	72.28	WS	48.19	48.19	WS	24.10	24.09
2	{8, 24, 14, 14 }	WS	130.38	130.42	WS	97.79	96.03	IS	65.19	65.21	IS	32.60	32.61
3	{24, 32, 13, 13}	OS	106.20	97.94	OS	79.65	73.44	OS	53.10	48.97	IS	26.55	24.48
4	{24, 32, 13, 13}	OS	106.20	97.94	OS	79.65	73.44	OS	53.10	48.97	IS	26.55	24.48
5	{16, 32, 13, 13}	OS	106.11	97.86	OS	79.58	72.81	IS	53.06	48.93	IS	26.53	24.47
Average			104.96	100.70		83.51	79.45		55.67	53.29		27.84	26.65

TABLE 5. Layer-wise parameter and performance on VGG-16.

layer	{Tm,Tn,Tr,Tc}	4 MAC			3 MAC			2 MAC			1 MAC		
		Pattern	Theore. Perf. (G-ops/s)	Actual Perf. (G-ops/s)	Pattern	Theore. Perf. (G-ops/s)	Actual Perf. (G-ops/s)	Pattern	Theore. Perf. (G-ops/s)	Actual Perf. (G-ops/s)	Pattern	Theore. Perf. (G-ops/s)	Actual Perf. (G-ops/s)
1	{16, 3 , 14, 14}	WS	88.20	84.88	WS	66.15	61.25	-	-	-	-	-	-
1	{32, 3 , 14, 14}	-	-	-	-	-	-	WS	44.56	42.82	WS	22.28	21.41
2	{16, 32, 14, 14}	WS	123.06	113.50	WS	92.29	82.46	WS	61.53	56.75	IS	30.76	28.37
3	{16, 32, 14, 14}	WS	123.06	113.50	WS	92.29	84.44	IS	61.53	56.75	IS	30.76	28.37
4	{16, 32, 14, 14}	OS	123.06	113.50	OS	92.30	84.44	IS	61.53	56.75	IS	30.77	28.37
5	{16, 32, 14, 14}	OS	123.06	113.50	OS	92.30	84.44	IS	61.53	56.75	IS	30.77	28.37
6,7	{16, 32, 14, 14}	OS	123.07	113.50	OS	92.30	84.44	IS	61.53	56.75	IS	30.77	28.37
8	{16, 32, 14, 14}	OS	123.07	113.50	OS	92.30	84.95	IS	61.53	56.75	IS	30.77	28.37
9,10	{16, 32, 14, 14}	OS	123.07	113.50	OS	92.30	84.95	OS	61.53	56.75	IS	30.77	28.37
11,12,13	{16, 32, 14, 14}	OS	123.07	113.50	OS	92.30	84.95	OS	61.53	56.75	OS	30.77	28.37
Average			122.79	113.28		92.09	84.22		61.40	56.65		30.70	28.32

than that of layer 3∼5 of AlexNet because of the different PE utilization ratios, 100% of VGG-16 and 86.22% of AlexNet.

3) RESNET-34

Table 6 gives layer-wise and module-wise parameter combination and performance for ResNet-34. Table 6 does not contain the performance of the design with 4 MACs since the design deletes the configuration of kernel stride 2 to prevent from congestion on routing design process. The first layer of ResNet-34 are conventional layer and the rest are the bottleneck modules. Each "conv_x" module in the table has a bottleneck module replicated multiple times. "conv_x/2" stands for the convolution operation with a stride of 2. There are two reasons why high performance is attained in layer 1. On one hand, the kernel size is so large (7×7) that performance is not even affected by small N. On the other hand, PE utilization achieves 100%, because the kernel stride is 2 and each 14×14 PE array generates four 7×7 Omap of different output channels simultaneously. It is shown that performance of the remaining layers is lower than that of layer 1 since the kernel sizes of the remaining layers are all 3×3.

Ratios of the total off-chip memory access of single stationary storage pattern and of HS storage pattern are defined for comparison in Figure 11. It is shown that HS saves off-chip memory access by $0.33 \times \sim 1.89 \times$ over single stationary storage pattern in most cases on AlexNet, $0.02 \times \sim 0.72 \times$ on VGG-16 and $0.09 \times \sim 1.41 \times$ on ResNet-34.

C. PERFORMANCE COMPARISON

Table 7 provides a summary of the implementation result and comparison with prior DCNN accelerators. Perf./DSP and Perf./power are defined for the mean share of each

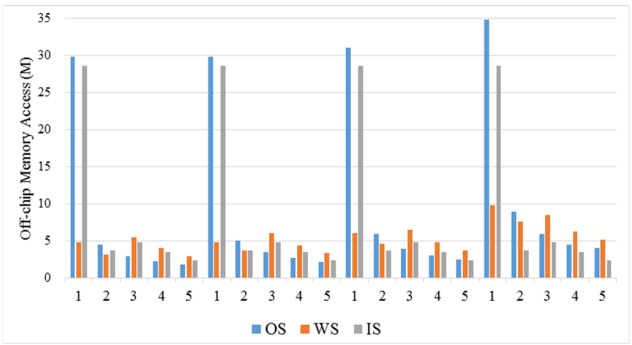


FIGURE 10. Layer-wise off-chip memory access on AlexNet.

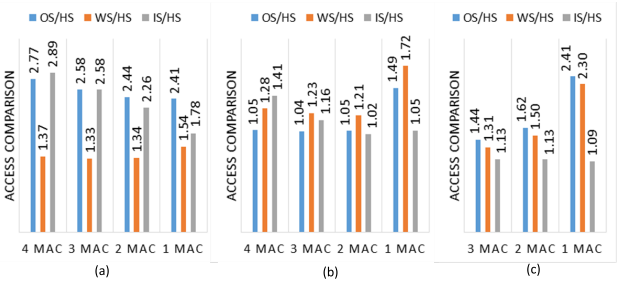


FIGURE 11. Total off-chip memory access analysis. (a) AlexNet. (b) VGG-16. (c) ResNet-34.

DSP48 block and for energy efficiency, respectively. Both are used for fair comparison with other works using a different number of resources.

Large amounts of BRAM resources will be needed if all data are loaded only once without tiling technique. To achieve that goal, the accelerator in [9] consumes all on-chip memory and a part of distributed memory. Compared with the design

TABLE 6. Layer-wise and module-wise parameter and performance on ResNet-34.

layer	{Tm,Tn,Tr,Tc}	3 MAC			2 MAC			1 MAC		
conv_1/2	{16, 3, 7 , 7 }	WS	95.39	84.48	-	-	-	-	-	-
conv_1/2	{32, 3, 7 , 7 }	-	-	-	WS	64.32	64.41	WS	32.16	32.20
conv_2	{16, 32, 14, 14}	WS	92.29	82.46	WS	61.53	56.75	IS	30.76	28.37
conv_3/2	{16, 32, 7 , 7 }	WS	91.57	81.94	IS	61.04	56.38	IS	30.52	28.19
conv_3	{16, 32, 14, 14}	OS	92.30	82.28	IS	61.53	56.75	IS	30.77	28.37
conv_4/2	{16, 32, 7 , 7 }	IS	91.59	81.94	IS	61.06	56.38	IS	30.53	28.19
conv_4	{16, 32, 14, 14}	OS	92.30	84.44	IS	61.53	56.75	IS	30.77	28.37
conv_5/2	{16, 32, 7 , 7 }	OS	91.59	83.89	OS	61.06	56.38	IS	30.53	28.19
conv_5	{8 , 64, 7 , 7 }	OS	94.01	86.25	OS	62.67	57.31	OS	31.34	28.65
Average			92.59	83.73		61.75	57.02		30.87	28.51

TABLE 7. Comparison with other designs.

Architecture	Precision	Platform	Benchmark	Frequency (MHz)	Performance (G-ops/s)	DSP	Perf./DSP	BRAM	LUT	Power(w)	Perf./power (G-ops/s/W)
This work (4 MACs)	16bit fixed	Zynq 7045	AlexNet VGG-16	100	100.7 113.28	784	0.128 0.144	211.5(464k)	149.037k	5.905	17.05 19.18
This work (2 MACs)	16bit fixed	Zynq 7045	AlexNet VGG-16 ResNet-34	145.45	77.51 82.39 82.93	392	0.198 0.210 0.212	131.5(264k)	112.469k	5.433	14.27 15.16 15.26
NullHop[15]	16bit fixed	Zynq 7100	VGG-16	60	17.2	128	0.134	386(1280k)	Logic 229k	2.339	27.4
Moini[9]	16bit fixed	Zynq 7045	AlexNet	150	38.4	391	0.098	545	77.442k	<10	3.8
Snowflake[16]	16bit fixed	Zynq 7045	AlexNet	250	120.3	256	0.47	-(768 k)	-	9.48	12.7
HWCE[25]	16bit fixed	Zynq 7045	AlexNet	100	130	800	0.163	-	-	-	-
Qiu[18]	16bit fixed	Zynq 7045	VGG-16	150	187.8	780	0.241	486	182.616k	9.63	19.5
Guo[23]	$\leq$10bit fixed	Virtex-7 485T	AlexNet	100	337.73	1712	0.197	-	-	-	-

in [9], the design with 2 MACs here achieves a speedup of 2.02 $\times$ with almost the same DSP resources, 24% BRAM resources and a little more off-chip memory access. Compared to [9], our accelerators have higher DSP and power efficiency.

The architecture named NullHop in [15] is designed for the ASIC platform and has been implemented in FPGA. They only presented the resources of the NullHop block in paper. Although NullHop's 17.2 G-ops/s is slow due to its limited DSP resources (128) and low frequency (60MHz), it is able to achieve a good DSP efficiency of 0.134 and low power consumption of 27.4 because of its zero-skipping pipeline and high MAC utilization. And NullHop has great efficiency of 471 G-ops/s on 28 nm ASIC platform. On VGG-16 benchmark, our design with 4 MACs has a slightly higher DSP efficiency of 0.144 than that of NullHop.

Snowflake in [16] attains 120.3 G-ops/s running at 150 MHz. Compared to our work, it has higher DSP efficiency but consumes more power. The hardware convolution engine (HWCE) presented in [25] is implemented on the same Zynq device as Snowflake. Its second layer performs at 130 G-ops/s while our design with 4 MACs has similar performance at 130.38 G-ops/s with lower DSP consumption. Therefore, our design has higher performance and DSP efficiency than HWCE. The accelerator in [18] supports dynamic-precision data quantization (16/8/4-bit) and obtains 187.8 G-OPS/S on 16-bit precision, costing 780 DSP and 486 BRAM. Although better performance and power efficiency are achieved, it can only support limited kernel sizes like 3 $\times$ 3, 5 $\times$ 5 and 7 $\times$ 7 and kernel stride 1.

The accelerator in [23] is a bit-width adaptive accelerator which can adapt to the DCNN layers with various bit-width requirements in a same network. A DSP can perform a multiply-accumulate operation adaptive to one 16-bit $\times$ 16-bit, or two 8-bit $\times$ 8-bit, or three 4-bit $\times$ 4-bit. The implementation of accelerator in [23] are based on the adaptive data width ranging from 4-bit to 9-bit. Therefore, it attains a high performance of 337.73 G-ops/s and a high DSP ratio of 0.197.

There are lots of works implemented on ASIC platform, such as accelerator in [10] with performance of 194.4 G-ops/s, processor in [24] with 368.4 G-ops/s and accelerator in [15] achieving 471 G-ops/s. Compared to FPGA platform, higher performance and lower power consumption can be realized on ASIC platform. Both designs in [10] and [24] are implemented with the TSMC 65-nm technology and have energy efficiency of 406.2 G-ops/s/W and 2.27 T-ops/s/W, respectively. The design in [15] is implemented with GF 28-nm and its energy efficiency is 3.042 T-ops/s/W. They achieve a great performance and energy efficiency.

The accelerators we propose apply roofline model to explore all design space to find the best performance with the used hardware resources. With the optimal tiling parameters and storage pattern, the design with 4 MACs achieves 100.7 G-OPS/S on AlexNet and 113.28 G-OPS/S on VGG-16. It has a DSP ratio of 0.144 and energy efficiency of 19.18 G-ops/s/w on VGG-16.

V. CONCLUSION

In this paper, we propose a resources-efficient and configurable accelerator, which can efficiently process a wide range of DCNN. The accelerator applies hybrid stationary storage pattern to improve the data reuse and to minimize

off-chip memory access. To get the best performance, roofline model is employed to explore the design space with the used limited resource in system. The accelerator is implemented on a ZYNQ-7 ZC706 evaluation board and the design with 4 MACs costs 784 DSP48 modules and 211.5 BRAM modules. It achieves average performance of 100.7 G-OPS/S on AlexNet and 113.28 G-OPS/S on VGG-16 running at the frequency of 100 MHz and consumes power of 5.905 watts. In the future, we will explore to extend the accelerator to run the complete flow of DCNN on embedded systems.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Sep. 2014, *arXiv:1409.1556*. [Online]. Available: https://arxiv.org/abs/1409.1556

[3] C. Szegedy, W. Liu, and Y. Jia, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[4] K. He, X. Zhang, and S. Ren, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2005, pp. 770–778.

[5] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very deep convolutional networks for text classification," Jun. 2016, *arXiv:1606.01781*. [Online]. Available: https://arxiv.org/abs/1606.01781

[6] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2015, pp. 161–170.

[7] C. Zhang, Z. Fang, P. Pan, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *Proc. Int. Conf. Comput. Aided Design*, Nov. 2016, p. 12.

[8] A. Rahman, J. Lee, and K. Choi, "Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2016, pp. 1393–1398.

[9] S. Moini, B. Alizadeh, and M. Emad, R. Ebrahimpour, "A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications," *IEEE IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 10, pp. 1217–1221, Oct. 2017.

[10] F. Tu, S. P. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 8, pp. 2220–2233, Aug. 2017.

[11] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 367–379.

[12] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[13] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Proc. Int. Conf. Artif. Neural Netw.*, Sep. 2014, pp. 281–290.

[14] M. Motamedi, P. Gysel, V. Akella, S. Ghiasi, "Design space exploration of FPGA-based deep convolutional neural networks," in *Proc. 21st Asia South Pacific Design Automat. Conf.*, Jan. 2016, pp. 575–580.

[15] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu, and T. Delbruck, "NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 3, pp. 644–656, Mar. 2018.

[16] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello, "Snowflake: A model agnostic accelerator for deep convolutional neural networks," 2017, *arXiv:1708.02579*. [Online]. Available: https://arxiv.org/abs/1708.02579

[17] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "DLAU: A scalable deep learning accelerator unit on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 3, pp. 513–517, Mar. 2017.

[18] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, and E. Zhou, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2016, pp. 26–35.

[19] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 Envision: A 0.26-to-10 TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 246–247.

[20] D. Shin, J. Lee, J. Lee, and H,-J. Yoo, "14.2 DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 240–241.

[21] Y.-H. Chen, T.-J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," 2018, *arXiv:1807.07928*. [Online]. Available: https://arxiv.org/abs/1807.07928

[22] J. Guo, S. Yin, P. Ouyang, L. Liu, and S. Wei, "Bit-width based resource partitioning for CNN acceleration on FPGA," in *Proc. IEEE 25th Int. Symp. Field-Program. Custom Comput. Mach.*, Apr. 2017, p. 31.

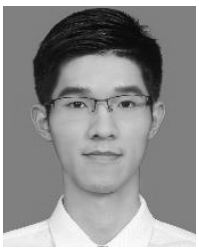
[23] J. Guo, S. Yin, P. Ouyang, F. Tu, S. Tang, L. Liu, and S. Wei, "Bit-width adaptive accelerator design for convolution neural network," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Florence, Italy, May 2018, pp. 1–5.

[24] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, L. Liu, and S. Wei "A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications," in *Proc. IEEE Symp. VLSI Circuits*, Kyoto, Japan, Jun. 2017, pp. C26–C27.

[25] P. Meloni, G. Deriu, F. Conti, I. Loi, L. Raffo, and L. Benini, "Curbing the roofline: A scalable and flexible architecture for cnns on fpga," in *Proc. ACM Int. Conf. Comput. Frontiers*, May 2016, pp. 376–383.

[26] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

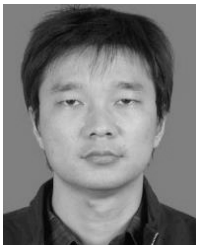
XIANGHONG HU received the B.Sc. degree in communication engineering from the Guangdong University of Technology, Guangdong, China, in 2015, where he is currently pursuing the Ph.D. degree with the Department of Automation. His current research interests include computer architecture, deep learning, information security, and VLSI design.



YUHANG ZENG received the B.Sc. degree in electronic information science and technology from the Guangdong University of Technology, Guangdong, China, in 2016, where he is currently pursuing the M.Sc. degree with the Department of Automation. His current research interests include computer architecture and deep learning.

ZICONG LI received the B.Sc. degree in electronic information science and technology from the Guangdong University of Technology, Guangdong, China, in 2016, where he is currently pursuing the M.Sc. degree with the Department of Automation. His current research interests include hardware architectures and deep learning.

XIN ZHENG received the M.Sc. degree in control engineering from the Guangdong University of Technology, Guangdong, China, in 2018, where she is currently pursuing the Ph.D. degree with the Department of Automation. Her research interests include software/hardware co-design and machine learning.



SHUTING CAI received the B.Sc. and M.Sc. degrees in computer science from Central South University, Changsha, China, in 2001 and 2004, respectively, and the Ph.D. degree in control science and engineering from the Guangdong University of Technology, Guangzhou, China, in 2011. He was a Visiting Scholar with the École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, in 2012. He is currently an Associate Professor with the Guangdong University of Technology. His current research interests include hardware architectures, multimedia signal processing, and computer vision.

XIAOMING XIONG received the B.Sc. degree from the Department of Radio, South China University of Technology, Guangzhou, China, in 1982, and the Ph.D. degree from the Department of Electrical Engineering and Communication, University of California at Berkeley, Berkeley, CA, USA, in 1988. He is currently a Professor with the Guangdong University of Technology. He is also the Chief Technical Advisor of the Company of Chipeye Microelectronics Foshan Ltd. His current research interests include computer vision, hardware/software cryptographic, SoC architectures, and VLSI design.