ABM-SpConv: A Novel Approach to FPGA-Based Acceleration of Convolutional Neural Network Inference

Dong Wang, Ke Xu, Qun Jia
Beijing Jiaotong University
Beijing, China
{wangdong,17112071,16120347}@bjtu.edu.cn

Soheil Ghiasi
University of California, Davis
Davis, CA, USA
ghiasi@ucdavis.edu

ABSTRACT

Hardware accelerators for convolutional neural network (CNN) inference have been extensively studied in recent years. The reported designs tend to utilize a similar underlying architecture based on multiplier-accumulator (MAC) arrays, which has the practical consequence of limiting the FPGA-based accelerator performance by the number of available on-chip DSP blocks, while leaving other resource under-utilized. To address this problem, we consider a transformation to the convolution computation, which leads to transformation of the accelerator design space and relaxes the pressure on the required DSP resources. We demonstrate that our approach enables us to strike a judicious balance between utilization of the on-chip memory, logic, and DSP resources, due to which, our accelerator considerably outperforms state of the art. We report the effectiveness of our approach on a Stratix-V GXA7 FPGA, which shows 55% throughput improvement, while using 6.25% less DSP blocks, compared to the best reported CNN accelerator on the same device.

1 INTRODUCTION

Convolutional Neural Network (CNN) has become the dominate approach in many artificial intelligence (AI) applications, such as computer vision, speech recognition and robotics. Many studies [11] have been carried out to design various CNN hardware accelerators for real-time processing. Being able to provide massive computational resources with flexible data precision, lower power dissipation and shorter deployment cycle, FPGA-based CNN inference accelerator has received great attention in various application fields from large-scale data-centers to energy-constrained IoT devices.

One of the key challenges in designing FPGA-based CNN accelerator is to take full advantage of the on-chip computing resource to speedup multiply-and-accumulate (MAC) operations in the convolution (CONV) and fully-connected (FC) layers, which account for over 99% [9] of the total operations for most CNNs. According to the way in which convolution computation is implemented, existing designs can be divided into three major categories. The first category of designs, such as [4, 12, 13], directly exploit parallelism

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317753

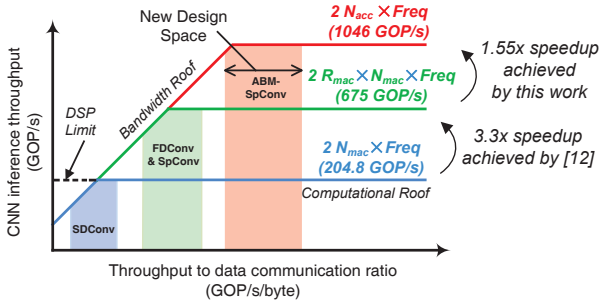

Figure 1: Comparing the design space of traditional MAC-based accelerator with the proposed architecture in a roofline model for Stratix-V-A7 FPGA.

of the convolution computation in *Spatial Domain* (referred to as SDConv) by using large number of DSP blocks performing massive number of MAC operations in every cycle. As shown in Figure 1, the design space of SDConv-based accelerator has a computational roof of $2 \times N_{mac} \times Freq$, where N_{mac} and $Freq$ denote the number of on-chip MAC units and the operating frequency, respectively. For instance, on a Stratix-V-A7 FPGA which has 256 DSPs, the maximum attainable inference throughput is 204.8 GOP/s under frequency of 200 MHz. (each DSP can perform two 16/8-bit fixed-point MACs).

The second category of designs [3, 10] perform convolution in the *Frequency Domain* (referred to as FDConv). By reducing the number of MAC operations required for convolution, FDConv-based accelerator raises the computational roof over SDConv-based design by a factor of R_{mac}, where R_{mac} is the reduction rate in MAC operation. For instance, up to 69.2% of the MAC operation is saved in [3], resulting in a theoretical speedup of 3.3× in peak performance. The throughput achieved by [3] on a Intel HARP platform is 669.1 GOP/s, which is very close to this roof.

The third type of designs reduce the number of MAC operations by directly pruning the CNN model [1, 2, 8]. Unimportant weights are forced to zero during the training (or fine-tuning) stage so that they will not contribute to computational workload and memory bandwidth in inference. Convolution performed on a pruned CNN model is referred to as *Sparse Convolution* (SpConv). The design space of SpConv-based accelerators share a similar computational roof of $2 \times R_{mac} \times N_{mac} \times Freq$ with FDConv-based ones. The performance of the reported SpConv-based FPGA accelerators have not exceed that of [3].

From an architecture view, existing FPGA accelerators tend to utilize a similar underlying architecture based on MAC arrays. The practical consequence is that the on-chip DSPs are exhausted, while

Table 1: Comparison of the #OP required by different convolution approaches for selected layers and the entire VGG16 model.

	Layer Parameters						#OP (MOP)					
						Pruning				ABM-SpConv		
Layer	C	R	N	K×K	M	Ratio	SDConv	FDConv[3]	SpConv[7]	Acc.	Mult.	Acc./Mult.
CONV1_1	224	224	3	3×3	64	42%	173	52.5	100	50.3	12.1	4.1
CONV1_2	224	224	64	3×3	64	78%	3,699	1,119	814	407	119	3.4
CONV4_1	28	28	256	3×3	512	68%	1,849	559	592	296	9.23	32.0
CONV4_2	28	28	512	3×3	512	73%	3,699	1,119	998	499	7.95	62.7
FC6	1	1	25088	1×1	25088	96%	205	205	8.23	4.11	0.037	111
FC7	1	1	4096	1×1	4096	96%	33.6	33.6	1.34	0.67	0.021	31.9
Entire CNN							30,941	9,531	10,082	5,040		
#OP Saved							0%	69.2%	67.4%	83.6%		

leaving other resource under-utilized. In this paper, we improve the performance of FPGA-based CNN inference accelerator *beyond* MAC-based designs by transforming the accelerator design space in which the computational roof is bound by accumulator resource rather than MAC. The insight enabling this idea is that *quantized CNN models only have a fixed number of possible values (For instance, 16 values for a 4-bit fixed-point number), so many multiplications performed in convolution can be avoided by factorization.* To this end, we first develop a new approach which performs the multiplication and accumulation operations of convolution in separated stages. Then, we design a heterogeneous hardware architecture comprised of a "big" accumulator array and a "small" multiplier array on which the two stages of convolution can be efficiently mapped. Our scheme transforms the design space of FPGA-based CNN accelerators to one with a raised computational roof of $2 \times N_{acc} \times Freq$, where N_{acc} is the number of accumulators used and N_{acc} is much larger than N_{mac} as shown in Figure 1

In summary, the contributions of this work are:

- We propose ABM-SpConv, a new sparse convolution scheme which achieves a higher arithmetic intensity for accumulation than multiplication so that the computational roof of the accelerator design space is transformed as accumulator-bound.
- We propose an FPGA accelerator architecture which consists of heterogeneous arrays of accumulators and multipliers to match the distinct computation flow of ABM-SpConv. Several optimization schemes, including semi-synchronous parallel processing and index-based weight encoding, are developed to ensure highly efficient data-path utilization and low external memory bandwidth requirement.
- A complete flow for design space exploration is introduced, and key design steps for finding the optimal hardware parameters are developed. The implemented accelerator achieves 1.55× speedup in inference throughput compared to the state-of-the-art design on a Stratix-V GXA7 FPGA.

2 PRELIMINARIES

CNNs are composed of multiple functional layers [11], each of which performs certain type of arithmetic operations on the input image or feature map. As the most compute-intensive layer, convolution computation is comprised of iterative 3-dimensional (3-D) MAC operations as follow:

$$FO_{m,r',c'} = \sum_{n=0}^{N-1} \sum_{k=0}^{K-1} \sum_{k'=0}^{K-1} FI_{n,r' \cdot S+k,c' \cdot S+k'} \cdot W_{m,n,k,k'} \quad (1)$$

where S denotes the convolution stride. $FI_{n,r,c}$, $FO_{m,r',c'}$ and $W_{m,n,k,k'}$ denote the input and output feature maps and the weight, which are of the size $N \times R \times C$, $M \times R' \times C'$, and $M \times N \times K \times K$, respectively. A convolution kernel is defined as the 3-D MAC operation that generates one feature map pixel, while channels refer to the feature matrices of size $R \times C$ or $R' \times C'$. For FC computation, Equation (1) can be reused by setting $R = 1, C = 1$ and $K = 1$, which becomes a 1-D inner-product operation. Table 1 gives an example of the dimensional parameters and the number of operations (#*OP*) for the VGG16 [5] model that has been widely used as a performance benchmark in the literature. Due to limited space, only the numbers for a few selected layers and the entire CNN are shown.

3 THE ABM-SPARSE CONVOLUTION

In this section, we introduce the *Accumulate-Before-Multiply Sparse Convolution* (ABM-SpConv) scheme. The key idea is to perform accumulate and multiply operations in separated steps so that the arithmetic intensity of multiplication can be reduced by removing redundant operations, which consequently relaxes the demand for DSP units when implemented on FPGA.

Assuming that weight W is quantized and kept in fixed-point format with q-bit precision, there exists at most $Q = 2^q$ different values for W. By denoting these fixed-point value as $\bar{W}_p$, where $p = 0, \cdots, Q-1$, we perform factorization on Equation 1 as follow:

$$\begin{aligned} FO_{m,r',c'} &= \sum_{w} FI_0(w) \cdot \bar{W}_0 + \cdots + \sum_{w} FI_{Q-1}(w) \cdot \bar{W}_{Q-1} \\ &= \sum_{p=0}^{Q-1} \left(\bar{W}_p \cdot \sum_{w} FI_p(w) \right) \end{aligned} \quad (2)$$

where $FI_p(w)$ represents the input feature pixels that are multiplied by the same weight $\bar{W}_p$ in a convolution kernel. Based on this new equation, we propose to conduct the convolution computation in a two-stage flow as follow:

(1) For each non-zero $\bar{W}_p$, find and ***Accumulate*** all the feature points $FI_p(w)$ in the convolution kernel, producing $Q-1$ partial products in total;

(2) ***Multiply*** each partial product with the corresponding $\bar{W}_p$ and do a final accumulation to obtain the output value for the current convolution kernel.

Iteratively repeating step (1) and (2) for all convolution channels will generate all $M \times R' \times C'$ output feature map pixels.

Studies [6] have shown that the weight can be quantized with 8-bit (or less) precision with less than 1% decrease in inference accuracy. Thus, with a 8-bit quantized weight, our scheme requires at most 256 rather than $N \times K \times K$ multiplications for each convolution kernel (For pruned weight, the actual number is much smaller than 256). On the other hand, accumulation of $FI_p(w)$ for $\bar{W}_p = 0$ can be easily skipped in the computation flow, which means that hardware implementation can also exploit sparsity to reduce computation complexity and memory footprint for weight storage. In Table 1, we report the number of addition and multiplication required when performing ABM-SpConv on a pruned VGG16 model [7]. 83.6% of the total operations (accumulate and multiply) is saved compared to SDConv, while the reduction over FDConv [3] and SpConv [7] are 47.1% and 50%, respectively.

4 HARDWARE ARCHITECTURE

4.1 Design Challenges

The new convolution scheme introduces the following novel design challenges: (i) The irregular sparsity pattern among different convolution kernels introduces imbalanced workload, which prevent us from achieving full parallelism of convolution on an accelerator architecture consisting of tightly-synchronized executed processing elements, such as MAC array. (ii) The accumulate and multiply operations have different arithmetic intensity and are carried out in two distinct computation stages. Mapping both operations on a homogeneous hardware architecture causes low utilization of computational resources. (iii) The algorithm also requires random memory access to the feature map data in memory accordingly to the irregular locations of the weights, which degrades the efficiency of external memory bandwidth and requires complicated data flow control in data path design. In the following section, we discuss our proposed hardware architecture, which addresses the aforementioned design challenges.

4.2 Architecture Design

Figure 2-(a) shows the overall architecture of the proposed ABM-SpConv accelerator, including a task scheduling unit, a fetch/store unit and multiple convolution units (CUs). The task scheduler detects the status of each CU and, whenever there is an idle CU, it launches a new *computation task* on that CU. As depicted in Figure 3, a computation task is defined as a group of convolution operations that are performed on a prefetch window of the input feature map. Each CU has its own loop counter so that it can independently execute tasks with varying workload. Synchronization of the CUs is infrequently conducted, only when feature map buffers are updated with new data. Measured CU utilization data (see Section 6) show that this semi-synchronous CU architecture successfully solves the first design challenge.

Convolution Unit. To address the second design issue, we propose a heterogeneous CU architecture of two independent arrays

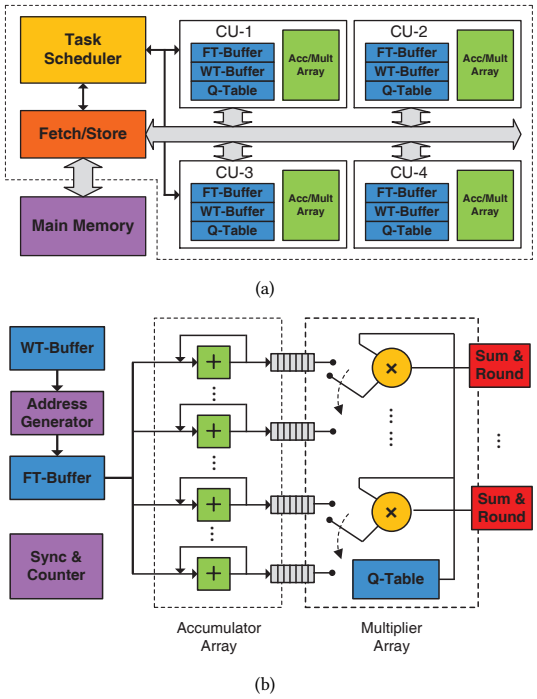

Figure 2: The proposed hardware architecture. (a) high-level architecture. (b) architecture of the convolution unit.

of accumulators and multipliers as shown in Figure 2-(b). The accumulator array accumulates the input feature map FI_p pixels which share the same weight $\bar{W}_p$ and stores the partial results into following FIFOs. The multipliers then read the partial result, multiply it by the fixed-point $\bar{W}_p$ and send the product to the Sum/Round logic for final processing. We further separate the accumulators into groups such that every N_g accumulators share one multiplier in the data-path. During convolution, the multiplier dynamically chooses the outputs of upstream FIFOs as its input operand in a round-robin manner. With a proper setting of FIFO depth, the two-stage convolution computation can be efficiently pipelined. This hardware structure enables the accelerator to use N_g times more accumulators than DSPs to speedup CNN inference computation. In the final design, 16-bit accumulator and 16b-by-16b multiplier are adopted to ensure full-precision fixed-point computation and no information loss during convolution, which guarantees the validity of Equation (2). Rounding is performed only once before writing feature map data back to main memory.

On-Chip Buffer. Figure 4 illustrates how the pruned CNN model is encoded and stored in a pair of local buffers. To alleviate the overhead of data-path control, we propose to encode the indexes (n, k, k') of the non-zero weights according to the order of corresponding $\bar{W}_p$ in the weight buffer (WT-Buffer). Another small buffer (Q-Table) is designed to store extra information, including the fixed-point value (VAL) of $\bar{W}_p$, the corresponding number (NUM) of occurrence of indexes and the total number of occurrence of the encoded weights, which are used by the loop counter and multipliers. A dedicated Address Generator is designed to decode the weight on-the-fly, map the indexes onto the feature map domain

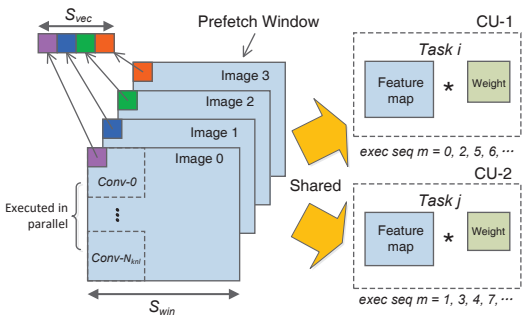

Figure 3: Computation task mapping and batched input images/feature prefetching schemes.

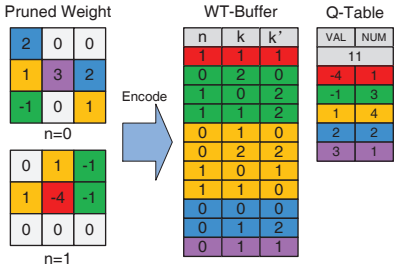

Figure 4: The proposed sparse weight encoding scheme. A simplified case for $M = 1, N = 2, K = 3$. Weighs are quantized in 3-bit, i.e., 1 sign bit and 2 integer bits.

and load continuous data stream from the feature map buffer (FT-Buffer). As illustrated in Figure 3, batched feature maps are cached in terms of prefetch window and feature matrices are vectorized so that data-level parallelism is exploited to further increase the throughput of the accelerator. The width of the local buffers FT-Buffer, WT-Buffer and Q-Table are $8 \cdot S_{vec}$, 16 and 16 bits, respectively.

Design Parameters. The proposed architecture can be configured by the following design parameters to achieve flexible performance and hardware cost:

(1) N_{cu} – the number of parallel CUs; N_{knl} – the maximum number of convolution operations that can be executed in parallel on one CU; N_g – the number of accumulators that share the same multiplier.
(2) S_{vec} – the width of the vectorized input data.
(3) D_f, D_w and D_q – the depth of the local feature, weight and Q-Table buffers, respectively.

5 DESIGN SPACE EXPLORATION

In this section, we define the mathematical models for design space exploration and then, present the complete flow and show how the models are applied.

5.1 Performance and Resource Estimation Models

The *Performance Model*, *Bandwidth Model* and *Resource Requirement Model* used in the proposed design space exploration flow are defined as follow:

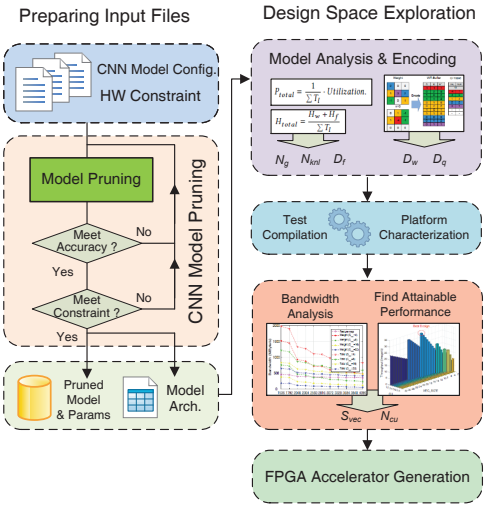

Figure 5: Overview of the design space exploration flow.

Performance Model. The theoretical execution time for the l-th convolution layer is calculated by:

$$T_l = \frac{\#OP_l}{S_{vec} \cdot N_{cu}} \cdot \frac{CEIL(R'/N_{knl})}{R'} \cdot \frac{1}{Freq} \tag{3}$$

where $\#OP_l$ is the number of accumulations performed in layer l. Then, the average performance (image/s) can be estimated by

$$P_{total} = \frac{1}{\sum T_l} \cdot Utilization. \tag{4}$$

Bandwidth Model. As show in Figure 3, the whole input feature map for the l-th layer is processed after $G_l^r \cdot G_l^c$ times of prefeching, where G_l^c and G_l^r equal

$$G_l^c = CEIL\left(\frac{C}{FLOOR[(S_{win} - K)/S] + 1}\right), \; G_l^r = CEIL\left(\frac{R'}{N_{knl}}\right)$$

Consequently, the total amount (Byte) of feature map data that are transmitted from external memory per image is

$$H_f = \sum_l \left(G_l^c \cdot G_l^r \cdot S_{win} \cdot [(N_{knl} - 1) \cdot S + K] \cdot N\right) \tag{5}$$

The corresponding amount of encoded weight that is fetched per image (assuming a minimum batch size of S_{vec}) is

$$H_w = \sum_l \left[2 \cdot G_l^c \cdot G_l^r \cdot M \cdot N \cdot K^2 \cdot (1 - P_l)\right] / S_{vec} \tag{6}$$

where P_l denotes the pruning rate of the l-th layer. In total, the average external memory bandwidth can be estimated by

$$H_{total} = \frac{H_w + H_f}{\sum T_l} \tag{7}$$

Resource Requirement Model. The following equations are used to estimate the required hardware resources, including logic, DSP and on-chip memory, respectively.

$$C_{logic} = C_0 + (C_1 + C_2 \cdot S_{vec} \cdot N_{knl}) \cdot N_{cu} \tag{8}$$

$$C_{dsp} = C_3 + [C_4 + (S_{vec} \cdot N_{knl})/4] \cdot N_{cu} \tag{9}$$

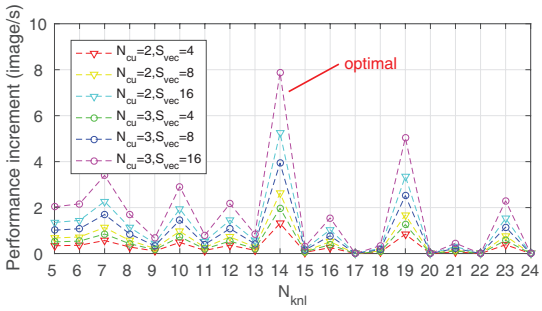

Figure 6: Exploration for the optimal value of N_{knl}. An operating frequency of 200 MHz is assumed.

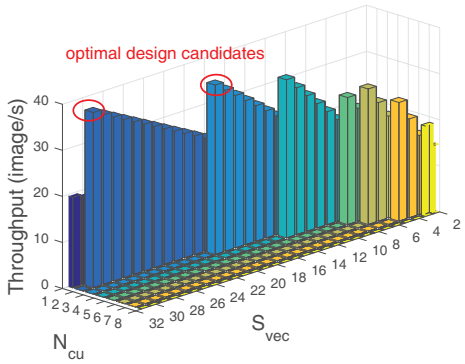

Figure 7: Exploration for the attainable throughput. The example shown is for VGG16 with $N_{knl} = 14$, $N_g = 4$ and $Freq = 200$ MHz. The constraint for logic utilization is 75%.

$$C_{mem} = C_5 + (C_6 \cdot S_{vec} + C_7 \cdot N_{knl}) \cdot N_{cu} \quad (10)$$

where C_0 to C_7 are platform-dependent constants which can be determined by characterizing the target FPGA.

5.2 Exploration Flow

The proposed design space exploration flow is illustrated in Figure 5. The flow first analyzes the network structure of the target CNN, encodes the pruned model layer-by-layer according to the proposed weight encoding scheme and determines the buffer sizes of D_w and D_q. At the same, the ratio of the arithmetic intensity between accumulate and multiply operations is analyzed and N_g is determined to fit the minimum ratio (see last column in Table 1). The *Performance Model* is then used to estimate the inference throughput for different values of design parameter N_{knl}. During this stage, preset values are assumed for parameters S_{vec} and N_{cu}. The optimal N_{knl} is selected in a way that normalized performance boost is maximized as shown in Figure 6.

In the following stage, several rounds of fast compilation of the design code (OpenCL kernels) are carried out for the target FPGA device, and hardware resource utilization information, including logic, DSP and on-chip memory, are collected. Design constants which characterize the hardware cost of the accelerator are then solved based on the resource information and the *Resource Requirement Model*.

In the final stage, the attainable performance is explored in a S_{vec}-N_{cu} design space by using the *Performance Model* as depicted in Figure 7. Constraints of full utilization of the DSP and memory resources are applied during exploration. However, a strict budget on logic resource (such as 70%) may leads to failure in FPGA compilation or large degradation in operating frequency. Therefore, several design candidates with close logic utilization ratio are selected for final implementation. Moreover, because pruned CNN models are adopted in the design flow, the size of the encoded weight is much smaller than original model (see Table 3). The external memory bandwidth spent on weight transmission is significantly reduced compared to previous works. By using the *Bandwidth Model*, we have verified that our design is compute-bound for most FPGA devices.

6 RESULTS

6.1 Experimental Setup

We use the DE5-Net platform for performance evaluation. The on-board FPGA is an Intel Stratix-V GXA7 device, which has 234,720 ALMs, 256 DSP blocks and 2560 M20K memory resources. DDR3 SDRAM is attached to the FPGA providing 12.8 GB/s external memory bandwidth.

A high-level-synthesis (HLS)-based FPGA design methodology was adopted in hardware implementation. The proposed architecture was modeled in OpenCL kernels and compiled by using the Intel FPGA OpenCL SDK v17.1. FPGA executes all convolution and FC layers, while the remaining layers, such as pooling, LRN and softmax, are executed by the host program on CPU. By adopting pipelined processing, the execution time of CPU were hidden by FPGA. The proposed accelerator is evaluated by running inference computation of two CNNs (AlexNet and VGG16) and the design parameters configured are summarized in Table 3. Both models were pruned by the scheme proposed by Han et al. [7] and quantized with 8-bit precision [6] with less than 1% accuracy drop compared to the original model.

6.2 Comparison with State-of-the-Art

Table 2 summarizes the comparison with state-of-the-art FPGA accelerators. As in other studies, throughput is calculated as the total #*OP* for spatial convolution of the original model divided by the average inference time. Moreover, in order to make a fair comparison, we only use the number achieved by FPGA accelerator rather than the whole system in the following discussion. Designs of [4, 12, 13] are based on spatial convolution, while the works of [3, 10] use frequency domain convolution.

The latest work of [3] uses a frequency domain convolution scheme which gains 3.3× reduction in MAC operations for both CNN models. For VGG16, the model pruning scheme adopted in our design maintains a similar reduction rate of 3.06×. The implemented accelerator achieves 1.55× speedup in throughput compared to [3] as a result of being able to utilize 1.6× accumulators to accelerate the convolution computation. Note that, although our scheme quantizes the CNN model in 8-bit, the precision of the datapath is of the same (16-bit) as [3]. For AlexNet, the pruning scheme adopted by us only reduces the total MAC operations by 2.3× (30%

Table 2: Comparison with state-of-the-art FPGA CNN accelerators.

	[13] SDConv	[12] SDConv	[4] SDConv	[10] FDConv	[3] FDConv	[3] FDConv	Proposed ABM-SpConv	
CNN Model	AlexNet	VGG16	VGG16	AlexNet	AlexNet	VGG16	AlexNet	VGG16
FPGA	Stratix-V GXA7	Arria-10 GT1150	Arria-10 GX1150	Arria-10 GX1150	Stratix-V GXA7	Stratix-V GXA7	Stratix-V GXA7	Stratix-V GXA7
Freq. (MHz)	100	231	385	303	200	200	202	204
Model Precision (bit)	8-16 (fixed)	8-16 (fixed)	16 (fixed)	16 (float)	16 (fixed)	16 (fixed)	8 (fixed)	8 (fixed)
Logic Usage (ALM)	121K (52%)	313K (73%)	–	246K (58%)	107K (46%)	107K (46%)	170K (73%)	160K (68%)
DSP Usage	256 (100%)	1500 (98%)	1378 (91%)	1476 (97%)	256 (100%)	256 (100%)	243 (95%)	240 (94%)
On-chip Memory (M20K)	1552 (61%)	1668 (61%)	1450 (53%)	2487 (92%)	1377 (73%)	1377 (73%)	2460 (96%)	2435 (95%)
Design Methodology	RTL	RTL	RTL+OpenCL	OpenCL	RTL	RTL	OpenCL	OpenCL
Throughput (GOP/s)	134.1	1171	1790	1382	663.5[1](780.6[2])	662.3(669.1)	699	1029
Perf. Density (GOP/s/DSP)	0.52	0.78	1.29	0.94	2.59	2.58	2.87	4.29

[1] Performance achieved by FPGA accelerator.
[2] Overall system performance.

Table 3: Design parameters and size of encoded weights.

	Design Parameter							Weight Size (MB)	
	N_{knl}	N_{cu}	N_g	S_{vec}	D_f	D_w	D_q	Original	Encoded
AlexNet	14	3	4	20	1152	1024	128	61	11.9
VGG16	14	3	4	20	1568	2048	128	138	26.4

lower than that of [3]), but our scheme still improves the inference throughput by 5.4%.

When compared with design that implements spatial convolution [13] on the same device, we achieve considerably 3.8× improvement in throughput when normalized by frequency. To compare with [4], [12] and [10] that are implemented on a different type of FPGA device, we further normalize the performance by the number of DSPs used. It is clear that our design shows over 3× advantage in performance density than all three designs.

7 RELATED WORK

Study of [1] presented an energy-efficient accelerator that deployed sparse convolutional neural networks on a Artix-7 FPGA. Techniques presented in this work were resource utilization and power optimization orientated. The CNN model evaluated was of low complexity (0.44 GOP) and the performance achieved was only 31.79 GOP/s. In [8], the authors reported an design framework which mapped sparse CNNs onto FPGA accelerator. The reported performance was 271.6 GOP/s on a Zynq VC706 FPGA without further information disclosed on resource utilization and working frequency. The work of [2] presented an algorithm-hardware co-design scheme to improve the efficiency of sparse convolutional layers executed in hardware. A structurally pruned AlexNet model was accelerated on a Virtex VC707 board and only the execution efficiency (64.5%) was disclosed, which was also lower than ours (87% for VGG16 and 81% for AlexNet). For the first time, we present a sparse-convolution-based FPGA accelerator for high-throughput CNN inference that surpasses the performance of state-of-the-art design on the same device.

8 CONCLUSIONS

This work presented the first high-throughput FPGA accelerator design which targeted efficient implementation of sparse convolutional neural network. A new sparse convolution scheme along with an efficient hardware architecture were developed. By transforming the design space of FPGA-based accelerator from MAC-bound to accumulator-bound, our design successfully achieved 1.55× throughput improvement over the best frequency-domain-convolution-based accelerator on the same FPGA device.

ACKNOWLEDGMENTS

This work was partially supported by NNSF of China Grants NO.61-574013, 61532005, 61702286, 61503300 and the NSF of Tianjin NO.18-JCYBJC15600.

REFERENCES

[1] A. Page et al. 2017. SPARCNet: A Hardware Accelerator for Efficient Deployment of Sparse Convolutional Networks. *J. Emerg. Technol. Comput. Syst.* (2017).
[2] B. Li et al. 2018. Running sparse and low-precision neural network: When algorithm meets hardware. In *ASP-DAC 2018*.
[3] H. Zeng et al. 2018. A Framework for Generating High Throughput CNN Implementations on FPGAs. In *FPGA 2018*.
[4] J. Zhang et al. 2017. Improving the Performance of OpenCL-based FPGA Accelerator for Convolutional Neural Network. In *FPGA 2017*.
[5] K. Simonyan et al. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556
[6] P. Gysel et al. 2018. Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks. *IEEE Trans. Neural Networks and Learning Systems* 29, 11 (Nov 2018), 5784–5789.
[7] S. Han et al. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR* abs/1510.00149 (2015). http://arxiv.org/abs/1510.00149
[8] S. Li et al. 2017. An FPGA Design Framework for CNN Sparsification and Acceleration. In *FCCM 2017*.
[9] T. Yang et al. 2017. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. In *CVPR 2017*.
[10] U. Aydonat et al. 2017. An OpenCL™Deep Learning Accelerator on Arria 10. In *FPGA 2017*.
[11] V. Sze et al. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* (2017).
[12] X. Wei et al. 2017. Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs. In *DAC 2017*.
[13] Y. Ma et al. 2016. Scalable and modularized RTL compilation of Convolutional Neural Networks onto FPGA. In *FPL 2016*.