LETTER

Efficient Hardware Accelerator for Compressed Sparse Deep Neural Network*

Hao XIAO^{†a)}, Member, Kaikai ZHAO[†], and Guangzhu LIU[†], Nonmembers

SUMMARY This work presents a DNN accelerator architecture specifically designed for performing efficient inference on compressed and sparse DNN models. Leveraging the data sparsity, a runtime processing scheme is proposed to deal with the encoded weights and activations directly in the compressed domain without decompressing. Furthermore, a new data flow is proposed to facilitate the reuse of input activations across the fully-connected (FC) layers. The proposed design is implemented and verified using the Xilinx Virtex-7 FPGA. Experimental results show it achieves 1.99×, 1.95× faster and 20.38×, 3.04× more energy efficient than CPU and mGPU platforms, respectively, running AlexNet.

key words: deep neural networks, filed programmable gate array, run-length compression, sparse data

1. Introduction

Recently, deep neural networks (DNNs) have been adopted by many modern AI applications as a means of achieving superior accuracy. This competitive advantage of DNNs, however, comes at the cost of high computational complexity. General-purpose computation engines, especially GPUs, have been employed to address these large computations of DNNs, but there is still an emerging trend toward embedded processing of DNNs in edge devices, such as mobiles and wearables. Thus, specialized hardware acceleration such as ASICs or FPGAs gain increasingly interest in providing more energy-efficient processing of DNNs.

It is known that convolution operations account for more than 90% of the overall computational workload in DNNs. Although these operations can leverage massively parallel computational paradigms, such as systolic arrays, throughput may not scale accordingly due to the accompanying bandwidth requirement, and the energy consumption remains high as data movement can be more expensive than computation. Thus, in order to achieve energy-efficient DNN processing without compromising throughput, it is essential to optimize the energy cost of data movement from both on-chip and off-chip. There has been an increasing effort to develop various different dataflows that support parallel processing with minimal data movement. However, unlike convolution (CONV) layers that contain lots of data

reuse, fully-connected (FC) layers are memory-intensive and have no parameter reuse, which results in significant bandwidth limitations on DNN processing [1]. Thus, in this work, we focus on the architecture support for accelerating FC processing, especially the techniques to efficiently minimize their data movement.

Previous works have explored the sparsity in weights and activations to improve the energy efficiency and the processing speed of DNN [2]–[6]. In [2], Cnvlutin is proposed to skip the cycles of processing MACs that have zero weights or activations. However, it only supports skipping cycles for zero value and does not compress the data for saving data movement. In [3], it explored the weight sparsity to reduced bandwidth requirement and locally stored compressed weights to improve the computation efficiency. However, it does not keep activations in compressed form. In Eyeriss [4], the dynamic sparsity of activations is exploited to improve energy efficiency by gating the switching of logic and data accesses. And the compression format is also used for saving off-chip DRAM bandwidth. However, it only uses compression between the on-chip memory and off-chip DRAM, leaving the sparsity of on-chip data unexplored. In [5], EIE exploits the sparsity in both dynamic activations and static weights to save computation. However, with the computation skipping for zero activations, the energy of on-chip and off-chip data movement caused by sparse activations is still not reduced. In [6], SCNN employs an algorithmic dataflow to eliminate all multiplications with a zero and maintain both weights and activations in a compressed representation through the CNN computation. However, for dealing with FC layers where the weight connection is not reused across input activations, the algorithmic dataflow approach may lose a lot of throughput, and thus, additional optimization dedicated for FC layer is still necessary.

Therefore, this paper aims at the acceleration of FC layers and presents a new architecture that can exploit the data sparsity in both weights and activations to improve their performance and energy efficiency. The contribution of this work comes twofold. (i) To take advantage of the sparsity of both activations and weights, a novel *circuit-level* processing scheme is proposed to process the sparse data directly in the compressed domain. Unlike the *dataflow-level* approach in [6], it is dataflow-independent, and thus, being applicable to both CNN and FC layers. (ii) To facilitate the reuse of input activations across the FC layers, a new dataflow is proposed to maximize the parallelism and the utilization of

Manuscript received December 6, 2020.

Manuscript publicized February 19, 2021.

[†]The authors are with the School of Microelectronics, HeFei University of Technology, China.

*This work is supported in part by National Natural Science Foundation of China 61974039, 61834006, and the Aeronautical Science Foundation of China 2018ZCP4003.

a) E-mail: xiaohao@hfut.edu.cn

DOI: 10.1587/transinf.2020EDL8153

processing elements. We implement and verify the proposed design using the Xilinx Virtex-7 FPGA. Experimental results show that the proposed design achieves $1.99\times$, $1.95\times$ faster and $20.38\times$, $3.04\times$ more energy efficient than CPU and mGPU platforms, respectively, running AlexNet.

2. Proposed Method

To exploit the sparsity of activations and weights, we store our encoded sparse data in compressed RLC format, which is shown in Fig. 1. For each non-zero value (in red), there is a run-length value (in black) representing the number of leading zeros in the uncompressed data stream. The next value is inserted directly as a 16-bit non-zero data, and the count for run-length starts again. To get the most efficient compression rate, we evaluate the compression rate with various bitwidth of run-length for both weights and activations. From our experimental results, setting the run-length to be 4 bits for activations and 5 bits for weights yields the best compression rate. However, to simplify the data interconnection of the hardware implementation, a unified data format with 5-bit run-length is adopted for both activations and weights. As shown in Fig. 1, every three pairs of run-length and non-zero data are packed into a 64-bit word, with the last bit (in blue) indicating if the word is the last one in the code.

To speedup the processing, we propose to carry out the MAC computation in the compressed domain directly. The idea is to use the run-length to calculate the position changed between the adjacent non-zero data. As shown in Eq. (1), the position of the first non-zero member of the activation or weight matrix (P_0) can be assumed to be 0.

$$P_0 = 0. \quad (1)$$

Then, with the position of the n -th non-zero member P_n and the run length between the adjacent non-zero data, the position of the $(n+1)$ -th non-zero member P_{n+1} could be represented by Eq. (2).

$$P_{n+1} = P_n + L_n + 1. \quad (2)$$

Thus, by using mathematical induction method, the position of the n -th non-zero member P_n can be further represented by Eq. (3).

$$P_n = n - 1 + \sum_{i=1}^{n-1} L_i. \quad (3)$$

After getting the position of each non-zero activation and weight member using Eq. (3), the MAC computation can be carried out in the compressed domain directly.

Fig. 1 Encoding of the RLC.

Depending on the position of the non-zero activation and weight, there are three possible scenarios: (1) If the position of the non-zero weights is corresponding to the one of the non-zero activation, they will be passed down the pipeline for computation. If the positions of the non-zero weights and activations are different, their positions will be further compared. (2) If the position of activation is bigger, this non-zero activation is skipped and the position of the next one is calculated. (3) Otherwise, this non-zero weight is skipped and the position of the next one is calculated.

Figure 2 shows an example of carrying out the MAC operation in RLC compressed format. There are two sparse vector a and b which are shown in Fig. 2(a). After RLC compression, as shown in Fig. 2(b), the position of the non-zero element can be calculated using Eq. (3). To read the non-zero weight and activation pairs for MAC computation, the PE reads a RLC block of both weights and activations, and then, calculates the position of the first non-zero member, i.e., 2 and 1 in Fig. 2(b). Since their position (shown in dashed circles) are different, it skips the smaller one, i.e., vector b , and calculates the position of its next non-zero member, i.e., 2. This process is repeated until the position of any active weight and activation pair matches. As shown in this example, the position of non-zero members does not match in cycle 1, 2, 3 and 4, and thus, these non-zero member are skipped and the address are updated alternately. Once the position matches, i.e., in cycle 5, the MAC operation can be carried out using the non-zero pair.

3. Hardware

3.1 Architecture Overview

Figure 3 shows the architecture overview of the proposed DNN accelerator, which consists of a PE array, a top controller, a RLC encoder and on-chip buffers. The off-chip DRAM saves all the static weights and dynamic activations,

Fig. 2 An example of MAC computation in the RLC compressed domain, (a) before RLC compression, (b) after RLC compression.

Fig. 3 Architecture overview of the proposed DNN accelerator.

Fig. 4 Dataflow for processing FC layers, (a) processing pass, (b) dataflow of PEs for a processing pass.

which are kept in the RLC compressed form. The PE array that consists of 64 PEs is adopted to carry out the MAC operation of the FC layer. Each PE calculates the 1-D convolution of a compressed filter vector with an activation vector, independently. The top controller coordinates the off-chip data accesses and on-chip data traffic between the buffers and PEs, and also, controls the working state of the PE array. Once the computation of one FC layer finishes, the top controller sends the instructions to switch the state of the PE array. There are two on-chip buffers, which work as the input activation (iact) buffer and the output feature map (ofmap) buffer in turn. They work as a ping-pong manner, in which one prepares iacts to be consumed and the other stores the encoded ofmap blocks. Each of them has a capacity of 72 KB.

3.2 Dataflow

The proposed dataflow for processing FC layers is shown in Fig. 4. Taking the AlexNet as an example, there are three FC layers, FC6, FC7 and FC8. Both FC6 and FC7 layers have 4096 filters, the size of which, before compression, is 9216 and 4096, respectively. The FC8 layer has 1000 filters, the size of which is 4096. These filters will be compressed into a set of RLC blocks. As shown in Fig. 4(a), we call the process of performing multiple 1-D convolutions of an iact vector with 64 filter vectors as a Processing Pass. After the process starts, the top controller reads the compressed RLC blocks of the one iact vector from the DRAM and sends it to the iact buffer. As shown in Fig. 4(b), 64 compressed filter vectors are read from the external memory in sequence and each of them is distributed to a given PE. Thereafter, the PE array reads 16 encoded iact packets from the iact buffer in one burst. And the next 16 iact packets will be read until the previous ones are consumed. Each iact packets is read only once from the iact buffer and ofmaps are generated when the processing is finished. These ofmaps are then sent to the encoder for packing into compressed RLC blocks. These encoded ofmaps are stored in the ofmap buffer, which becomes the iact buffer of the next FC layer.

According to this dataflow, all the data including both static weights and dynamic activations of the network are stored in the RLC compressed form. And during the processing pass, they are still kept in the compressed domain.

*Note: Filter, Ifmap, Ofmap denote the Num. of DRAM access before RLC compression, Filter_RLC, Ifmap_RLC, Ofmap_RLC denote the Num. of DRAM access after RLC compression.

Fig. 5 Number of DRAM access before and after RLC compression.

Fig. 6 Structure of the processing element.

This saves the space and the bandwidth of both on-chip SRAM and off-chip DRAM. Figure 5 shows the overall DRAM accesses in AlexNet before and after RLC encoding, respectively. The DRAM accesses including filters, ifmaps, and ofmaps, can be saved by nearly 87% in FC6, nearly 87% in FC7 and nearly 70% in FC8.

3.3 Processing Element

Figure 6 shows the hardware structure of the PE that can directly perform MACs on RLC encoded iacts and filters using the proposed scheme in Sect. 2. The proposed PE consists of six pipeline stages. In the first stage, two FIFOs are used to buffer iact and filter blocks before the computation, respectively. The filter FIFO is implemented using a 512x64-bit SRAM due to the large size of one encoded filter vector. And the size of the iacts FIFO is 16x64-bit, where 64-bit is the size of one RLC data package. After fetching the data, the second stage analyses the position of each non-zero member of the filter and iact vector using Eq. (3). And the following stage further compares the position of each non-zero iact and filter pairs. As described in Sect. 2.2, if the position of the iact does not equal the one of the filter, the pipeline is blocked and the second stage updates the position of the next non-zero member. If the position of the current iact is greater, the position of filter is updated; otherwise, the position of iact is updated. If the current iact and filter members have the same position, the data will be passed to the following two stages, where the MAC operation on the non-zero iact and filter is carried out. And after

Table 1 Comparison of execution time and energy efficiency among CPU, mobile GPU and the proposed design

Platform	CPU	mGPU	Proposed
Execution Time (ms)	1.2821	1.2565	0.6452
Power (W)	39.16	5.96	3.82
Throughput (Frames/s)	780	796	1550
Energy Efficiency (Frames/J)	19.91	133.56	405.76

getting the psum, the final stage may activate it by a ReLU operation optionally, and then, send the result out of the PE.

4. Implementation and Results

4.1 Hardware Performance

The proposed design is implemented and verified on the Xilinx Virtex-7 FPGA using Vivado 2017.4. The 64 PEs of the proposed design work at 200 MHz yielding a FC performance of 1.34 GOP/s. For comparison, we also evaluate the performance of the Intel Core i7-5930k CPU and the Nvidia TK1 Mobile GPU, respectively. The experiment on CPU and GPU are carried out using Ubuntu 14.04 and Tensorflow framework. As the benchmark, FC layers of the AlexNet is employed to evaluate the performance and energy efficiency.

Table 1 shows the performance comparison among CPU, mobile GPU and the proposed design when running the FC7 layer of the AlexNet. The results show the proposed design achieves 1.99× and 1.95× faster than the CPU and mobile GPU platform, respectively. The power consumptions of the DNNs accelerator when running the AlexNet is measured as 3.818 W, where the FPGA chip consumes 2.487 W and the off-chip DRAM consumes 1.331 W. Comparing with the CPU and mobile GPU, the proposed design achieves 20.38× and 3.04× more energy efficient, respectively.

Table 2 further compares our DNN accelerator with prior FPGA-based designs. The performance of the proposed design is 1.34 GOP/s. Considering it exploits the sparsity to eliminate 97% of the operations performed by dense DNNs, this requires a dense DNN accelerator with a processing throughput of 40.2 GOP/s to have the equivalent performance. Although all the listed works adopt various methods for acceleration, our proposed design achieves the best performance. This performance improvement comes twofold: (1) keeping data in RLC compressed format reduces the amount of the data; (2) special architecture supporting data processing in compressed domain skips the cycles of performing MACs that have zero weights or iacts.

5. Conclusion

This paper presents a DNN accelerator optimized for compressed sparse DNNs. By leveraging the sparsity in both the activations and the weights, a runtime processing scheme is proposed to ensure the compressed weights and activa-

Table 2 Comparison of FPGA-based DNN accelerators

	[1]	[7]	[8]	Proposed
Platform	Zynq XC7Z045	Stratix-V	Zynq XC7Z045	Virtex-7
Clock (MHz)	150	193.6	200	200
CNN Models	VGG16	AlexNet	AlexNet	AlexNet/VGG16
Data Format	16 bit	8 bit	8 bit	16 bit
Execution Time (ms)	60.18	4.40	5.07	2.92/6.26
Performance (GOP/s)	1.20	26.59	25.60	1.34
Throughput (Frames/s)	17	227	197	342/160

tions can be processed directly in their compressed format without decompressing, and therefore, leading to improved performance and energy efficiency. Furthermore, a new dataflow is also proposed to exploit the reuse of input activations across FC layers, and therefore, leading to the maximized parallelism and utilization of processing elements. Our implementation on the Xilinx Virtex-7 FPGA using AlexNet achieves 1.99×, 1.95× faster and 20.38×, 3.04× more energy efficient than CPU and mGPU platforms, respectively.

References

- [1] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded FPGA platform for convolutional neural network," *Proc. 2016 ACM/SIGDA Int. Symp. FPGA*, pp.26–35, Monterey, California, USA, Feb. 2016.
- [2] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N.E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-Neuron-Free deep neural network computing," *Proc. ACM/IEEE 43rd Int. Symp. Computer Architecture*, pp.1–13, Seoul, South Korea, June 2016.
- [3] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," *Proc. 49th Annual IEEE/ACM Int. Symp. Microarchitecture*, Taipei, Taiwan, pp.1–12, Oct. 2016.
- [4] Y. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *Proc. 2016 IEEE International Solid-State Circuits Conference*, pp.262–263, San Francisco, USA, Feb. 2016.
- [5] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W.J. Dally, "EIE: efficient inference engine on compressed deep neural network," *Proc. ACM/IEEE 43rd Int. Symp. Computer Architecture*, pp.243–254, Seoul, South Korea, June 2016.
- [6] A. Parashar, M. Rhu, A. Mukkarar, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S.W. Keckler, and W.J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," *Proc. ACM/IEEE 44th Int. Symp. Computer Architecture*, pp.27–40, Toronto, Canada, June 2017.
- [7] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-S. Seo, and Y. Cao, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," *Proc. 2016 ACM/SIGDA Int. Symp. FPGA*, pp.16–25, Monterey, California, USA, June 2016.
- [8] A. Azarmi Gilan, M. Emad, and B. Alizadeh, "FPGA-Based Implementation of a real-time object recognition system using convolutional neural network," *IEEE Trans. Circuits and Syst II: Express Briefs*, vol.67, no.4, pp.755–759, April 2020.