Work-in-Progress: A High-performance FPGA Accelerator for Sparse Neural Networks

Yuntao Lu, Lei Gong, Chongchong Xu, Fan Sun, Yiwei Zhang, Chao Wang, Xuehai Zhou
School of Computer Science and Technology
University of Science and Technology of China
luyuntao@mail.ustc.edu.cn

ABSTRACT

Neural networks have been widely used in a large range of domains, researchers tune numbers of layrs, neurons and synapses to adapt various applications. As a consequence, computations and memory of neural networks models are both intensive. As large requirements of memory and computing resources, it is difficult to deploy neural networks on resource-limited platforms. Sparse neural networks, which prune redundant neurons and synapses, alleviate computation and memory pressure. However, conventional accelerators cannot benefit from the sparse feature.

In this paper, we propose a high-performance FPGA accelerator for sparse neural networks which utilizes eliminate computations and storage space. This work compresses sparse weights and processes compressed data directly. Experimental results demonstrate that our accelerator will reduce 50% and 10% storage of convolutional and full-connected layers, and achieve 3x speedup of performance over an optimized conventional FPGA accelerator.

1 INTRODUCTION

Neural networks (NNs) are widely used in a large range of domains, such as Image Classification, Speech Recognition, Natural Language Processing et. al. The efficient NNs, which are fine-tuned of neurons and synapses, are computation and memory intensive. An advanced method Song et al. [2] can reduce the size of NNs by pruning redundant synapses and neurons. By means of [2], the sparsity (i.e, the sparsity is a fraction of remaining weights over the totals as 5.) of NNs is 10% on average. However, conventional accelerators, such as our previous work [3], can not benefit from the sparse feature. Thus, with the emerging of accelerators for sparse neural networks (SpNNs), in 2016, Song et al. [1] focus on sparse fully-connected layers, and first propose an engine to compress weights as well as coping with the load balance. Shijin et al. [5] employs a step-index module to eliminate zero multiplications to construct an accelerator for NNs with different sparsity.

In this paper, aiming to benefit from SpNNs, we propose an FPGA accelerator which achieves computation and storage reduction by compressing sparse weights and processing compressed data. In section 2, we describe the overview architecture of our accelerator which feature the compression and processing unit.In section 3, we demonstrate the experimental results. Through discussions, we conclude in section 4. Contributions made by this paper are as follows:

(1) We utilize a compression unit to compress sparse weights by coordinate and compressed sparse row formats, which reduces around 50% and 10% storage in different layers.

(2) We design a processing unit which has ability of performing computations with compressed data directly, and optimize computations in a pipeline with lower intervals.

CASES '17 Companion, Seoul, Republic of Korea

DOI: 10.1145/3125501.3125510

2 ACCELERATOR DESIGN

2.1 Overview

Figure 1 shows the architecture of our accelerator, which is composed of a direct memory access (DMA), a compression unit (CU) and processing elements (PEs). DMA transfers data from the external memory to accelerator block RAMs. CU converts sparse weights to compressed formats. PE processes computations for each layer and there are three data buffers (DBs) in PE to keep computing data, and function units (FUs) process input feature maps and compressed weights by multiplier-adders to work out intermediate results. Through activation functions, PE gain outputs, buffer them in DB, and then transfer data back to the external ultimately. In our design, we use 32-bit float point arithmetic units to align data transferred by DMA.

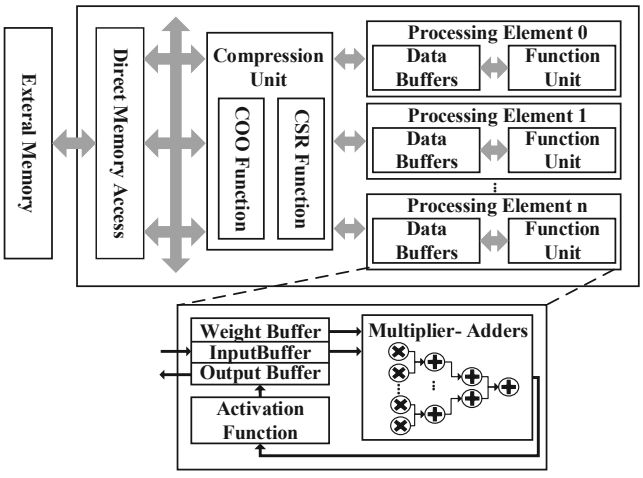

Figure 1: Architecture of our Accelerator.

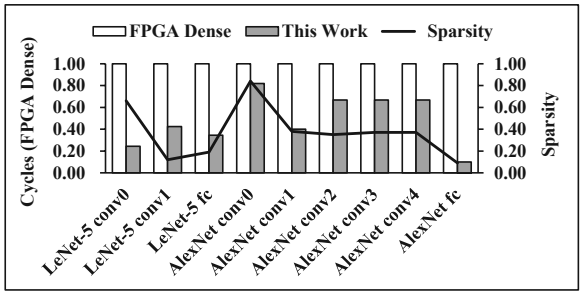

Figure 2: The comparison of this work with a conventional FPGA accelerator.

2.2 Compression unit

In order to save more storage space, CU uses two formats, coordinate (COO) and compressed sparse row (CSR), to convert weight data structures to compressed formats, which consist of three arrays to keep non-zero elements of weights, corresponding row and column index respectively. The difference between COO and CSR is in the row index array, which keeps the row index of non-zero data for COO while the index of the first non-zero element in each row for CSR.According to the step/direct index methods 5, we implement the two compressed formats to initialize weights in software, therefore, we don't need to the complexity of hardware implement. As a result, we achieve 50% and 10% compression ratio (i.e, compression ratio is the fraction of storage space of compressed weights over the originals.) of convolutional (CONV) and fully-connected (FC) layers.

2.3 Processing element

PE employs computations of CONV and FC layers, i.e., matrix-vector (MV) and vector-vector (VV) multiplications of compressed weights. In the bottom of Figure 1 illustrates the architecture of PE, which contains three DBs and an FU. FU takes compressed weights and feature maps from DB as inputs and produces an output transmitted back to DB.

FU. FU processes multiplications and additions for compressed data of each layer. Each FU contains three components, including $M \times M$-input multipliers, an M-input adder tree and an activation function, the value of M depends on hardware resources. To achieve an efficient performance, we pipeline function operations by five stages: data fetching, multiplying, adding, activation and outputting. Thus, FU can process $M \times M$ weights almost every cycle. As compressed formats convert data structures of weights, reading indexes indirectly incur more overheads. Computations change as well, in CONV layers, CONV multiplications change to MV multiplications, which take non-zero weight vectors and feature matrices as inputs, then produce new matrices. In FC layers, MV multiplications change to VV multiplications. Inputs are weight vectors and activation vectors, and the outputs are vectors for next iterations.

DB. A weight buffer (WB), an input buffer (IB) and an output buffer (OB) compose DB, which keeps input vectors (or feature maps), compressed weights and output feature maps respectively. Du to the weight size of large-scale sparse NNs, i.e., AlexNet, is still around 6M, we do not buffer total weights. For memory-limited hardware platforms, we can deploy an optimal 2KB WB in each PE, according to [5], a WB can offer W weights to an FU. For N PEs, the total size of WB will take $N \times 2KB$ buffer space. As the IB and OB keep inputs and outputs of an FU, their sizes depend on the number of multipliers, which is $W \times W \times 32$ bits. An optimal buffer size can hide memory access overhead of transferring data.

3 EXPERIMENT RESULTS

To evaluate the performance, we employ each layer core computations of sparse neural networks utilizing Vivado High-Level Synthesis to simulate RTL behaviors on Xilinx FPGA xc7z020 with the frequency of 100MHz.The COO and CSR compressed formats can reduce almost 50% and 10% storage space of a sparse CONV and FC layers.Figure2 illustrates the comparison of our design with an FPGA accelerator which we employed using the same method of [4]. The benchmark are sparse LeNet-5 and AlexNet models trained by Caffe framework. Execution cycles are normalized by the FPGA-dense accelerator cycles. Results demonstrate that our accelerator achieves 30%-40% and 70%-90% reduction for CONV and FC operations. The sparsity will be about 40% and 10% on average of CONV and FC layers, which means that remain weights over the totals of layers. However, this work cannot achieve the best performance in theory. The reason is two aspects. One is COO and CSR formats incur overhead for searching index in arrays. Another is that we cannot find a suitable input size of multipliers in our hardware implementation for variable convolutinal kernels, whose sparsity is range from 0 to 100%.

4 CONCLUSIONS

In this work, we propose an FPGA accelerator for sparse neural networks, which reduce computations as well as storage space. A compression unit and processing elements can compress and process weights of NNs. Experiment results by simulation demonstrate that this work achieves higher performance over a conventional accelerator. However, our accelerator cannot achieve the theoretical performance, due to compressed formats and optimizations.

Acknowledgment. This work was supported by the NSFC (No. 61379040), Anhui Provincial NSF (No.1608085QF12), CCF-Venustech Hongyan Research Initiative (No.CCF-VenustechRP1026002), Suzhou Research Foundation (No.SYG201625), Youth Innovation Promotion Association CAS (No. 2017497), and Fundamental Research Funds for the Central Universities (WK2150110003).

REFERENCES

[1] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *ISCA*. 243–254.
[2] Song Han, Jeff Pool, John Tran, and William Dally. 2016. Learning both weights and connections for efficient neural network. In *NIPS*. 1135–1143.
[3] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. 2017. DLAU: A scalable deep learning accelerator unit on FPGA. *TCAD* 36, 3 (2017), 513–517.
[4] Qi Yu, Chao Wang, Xiang Ma, Xi Li, and Xuehai Zhou. 2015. A Deep Learning prediction process accelerator based FPGA. In *CCGrid*. IEEE, 1159–1162.
[5] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *MICRO*. 1–12.