An Energy-Efficient Accelerator with Relative-Indexing Memory for Sparse Compressed Convolutional Neural Network

I-Chen Wu¹, Po-Tsang Huang², Chin-Yang Lo¹, and Wei Hwang¹,²

¹Department of Electronics Engineering, National Chiao Tung University, Taiwan
²International College of Semiconductor Technology, National Chiao Tung University, Taiwan

Abstract — Deep convolutional neural networks (CNNs) are widely used in image recognition and feature classification. However, deep CNNs are hard to be fully deployed for edge devices due to both computation-intensive and memory-intensive workloads. The energy efficiency of CNNs is dominated by off-chip memory accesses and convolution computation. In this paper, an energy-efficient accelerator is proposed for sparse compressed CNNs by reducing DRAM accesses and eliminating zero-op rand computation. Weight compression is utilized for sparse compressed CNNs to reduce the required memory capacity/bandwidth and a large portion of connections. Thus, ReLU function produces zero-valued activations. Additionally, the workloads are distributed based on channels to increase the degree of task parallelism, and all-row-to-all-row non-zero element multiplication is adopted for skipping redundant computation. The simulation results over the dense accelerator show that the proposed accelerator achieves 1.79x speedup and reduces 23.51%, 69.53%, 88.67% on-chip memory size, energy, and DRAM accesses of VGG-16.

I. INTRODUCTION

Over the past few years, convolutional neural networks (CNNs) are used to deal with many critical machine learning problems and gaining popularity in numerous computer-vision applications. Moreover, CNN is one of the most important AI techniques, which has overwhelming performance in object recognition and classification. However, state-of-the-art CNNs are both computationally and memory intensive based on deep network models, and hard to be fully deployed and implemented on edge devices. The most critical design challenge is that deep CNNs contain great amount of data, including weights and feature maps (fmaps), for large network models. All weights and fmaps have to be accessed from and restored to off-chip DRAM, respectively, that leading to large latency and energy costs. Therefore, weight compression techniques have been proposed to reduce the model size [1-3].

Based on the compressed CNNs, we can observe that lots of zero-valued weight stemmed from compression and zero-valued activation results from ReLU function which remains positive inputs, but makes negative values to zero. Thus, redundant arithmetic operation and storage can be eliminated but resulting in irregular execution and data movement. In this paper, an energy-efficient accelerator architecture is proposed for sparse neural networks to eliminate redundant computation and to reduce on-chip SRAM. We optimize the convolutional layer which occupies most computations and are the major layers in CNN [4]. Furthermore, the relative-indexing memory is realized to store non-zero (NZ) activations and weights for irregular data accesses, and the channel-based workload partition is adopted for an 8 x 8 processing element (PE) array to increase the overall resource utilization. Accordingly, all-row-to-all-row element multiplication is utilized to skip unnecessary computation.

II. ACCELERATOR ARCHITECTURE FOR SPARSE CNN

Deep Compression [2] is utilized to reduce the required memory capacity and bandwidth by compressing weight, and composed of three stages, including pruning, quantization and Huffman coding. First, pruning is widely used in neural networks to reduce model size and avoid over-fitting [1]. Removing small-value weights is the most important step to reduce a large amount of connections and partial neurons. The weights below a threshold value are set to zero, and then the remaining connections would be restrained to compensate for the accuracy. Second, each weight is represented in fewer bits to be further compressed by quantization. Weights are shared by k-means clustering, and then retrain the model. The final step is Huffman coding, which is a kind of optimal prefix code and widely used for lossless data compression. The Huffman coding is utilized to reduce the model size since the non-uniform distribution.

According to the sparse features of deep compressed CNNs, an energy-efficient accelerator is designed for AlexNet [5] and VGG-16 [6]. The proposed accelerator achieves lower energy dissipation by decreasing the total on-chip SRAM, the number of arithmetic operations and memory accesses. The details of the accelerator are described in the following sections.

A. Baseline Architecture

The overall architecture of the proposed accelerator is as Fig. 1 and consists of an 8x8 PE array, 8 Huffman decoders, 3-level accumulators, a pooling module and an activation encoder to form the NZ activations. The 8 Huffman decoders is implemented to get the 16-bit NZ fixed-point weights for 8 rows. And each PE receives partial and unique channels of weight and activations through the 1:8 de-multiplexers and relay stations (RSs). Every cycle, the PE array receives 8 NZ real weights and 1 NZ activation, and these data are transferred to 64 PEs through RSs. The RS is designed to increase the frequency of the accelerator by avoiding long routing wires and large loading. The outputs of PEs are partial sums (psums). Thus, all the outputs of PEs have to be accumulated as the completed output neurons for the ReLU pooling module. Based on the tradeoff between the accumulation cycles and the delay of interconnects, the outputs of PE array are accumulated through a 3-level tree topology of accumulators as shown in Fig. 1. The outputs of 4 PEs or 4 accumulators are connected to the next-level accumulators.

This work was partially supported under the Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE), and partially supported under Ministry of Science and Technology (MOST) in Taiwan.
B. Workload Distribution

The workload distribution is the key issue for the data flow of the accelerator and the efficiency of resource utilization. As the convolutional layer increases, the numbers of channels (C) dramatically increases, and the X and Y dimensions of the fmaps decreases. Therefore, the convolution workloads are distributed to each PE according to partial channel (C_t) of weight and activations as shown in Fig. 2. Additionally, since the channels of fmaps are usually multiple of 64, 64 PEs are implemented in this accelerator to balance the workload for increasing the resource utilization. For instance, in CONV1_2 of VGG-16, 1 channel of weight and activation are distributed to each PE. In CONV2_2, each PE receives 2 channels.

For decreasing the memory size in each PE, tiling the ifmaps is realized for the first few layers of CNNs. The size of ifmaps are usually big, and the input density is almost 100% for the first few layers. On the other word, a great disparity in the memory requirements exists between different layers of compressed CNNs. In view of this, tiling the ifmaps is essential for the first few layers to increase the usage of on-chip memory among deep networks. For example, in the first and second layer of VGG-16, the ifmaps are tiled into 4 slices. Nevertheless, the tiling procedure results in affecting the buffer size of the activation encoder module.

C. Relative-Indexing Memory

The zero-valued elements are not necessary to be stored no whether in off-chip DRAM or on-chip SRAM. Thus, only the activation and weight adopting relative-indexing (RI) are stored in memory. Fig. 3 presents an example of the relative-indexing compressed form in weight with 6 NZ elements. The original weights are illustrated on the left, and the NZ value, relative index and NZ num index vector are stored in the memory illustrated on the right. The value field only stores NZ values except for overflow, and the relative index indicates the distance of two NZ elements in a row (r), and “NZ num index” represents the number of NZ values in a row. Since only NZ elements are stored in the memory, an activation encoder is designed in the proposed accelerator to obtain the relative index and NZ num index for the activation data.

A tradeoff exists between the compression ratio and the execution latency. For improving the performance, “NZ num index” is utilized to provide row-independent indexing for each tile. For an example as shown in Fig. 3, “c” and its relative index, set as 0, can be stored back to memory immediately while computing the first tile. Otherwise, the indexing information is related to other tiles that increases the latency.
III. PE ARRAY FOR CONVOLUTION

The proposed PE architecture is designed to skip zero-element multiplication. Consequently, the PE array is composed of 8x8 PEs based on the workload distribution. Fig. 4 shows the overall PE architecture which is composed of activation SRAM, weight SRAM, index memory of activation and weight, indexing decoders, 3 multipliers, and 16 accumulators, which is based on the number of ofmaps in X-dimension, and output FIFOs. Both Activation SRAM and weight SRAM are implemented as ping-pong buffers to hide the latency. Thus, one activation SRAM is updated for the next layer or the next tiled ifmaps while the other is ready to process the current layer. Similarly, one weight SRAM is updated for the next set of parameters, and the other weight SRAM is accessed to calculate the current ofmaps simultaneously.

Each PE fetches data from memory which only stores NZ activations and NZ weights to perform multiplication. Activations and weights are divided in layers. Thus, every multiplier receives the specific layer of activations and the corresponding layer of weights to form psums. The decoders receive the index of activations and weights to decode the correct position in a row, and the location mapping module indicates which accumulator would receive the result of the multiplier. After computing one row, the psums is stored in output FIFOs, and the PE continues processing the next row until the output FIFO is full.

Every PE contains 3 multipliers, and the workload for each PE is divided into 3 layers to perform the convolution operation with 3x3 kernel size [7] as shown in Fig. 5. Each layer is represented by different colors. As the output of a multiplier is partial sum only, all the partial sums are accumulated by the following adders.

SCNN [8] performs all-to-all multiplication of NZ weight and activation vector element to avoid unnecessary arithmetic operations. A similar methodology is adopted for PEs using all-row-to-all-row NZ element multiplication. Fig. 6 presents the method of zero-skipping computation with 3x3 kernel size and stride 1. The first NZ weight “5” of the first channel is in the first row, and multiplied with all NZ input values in the first row of the first channel. The results are transferred to the corresponding accumulator. After searching NZ weights in the first row of the first channel, the next step is to search the next NZ weight “7” in the first row of the second channel, and “7” is multiplied with all NZ input values in the first row of the second channel.

The complete output neurons are finally calculated by a 3-level adder tree based on the output psums of 64 PEs. The execution cycles of generating psums among different PEs are not synchronous due to non-uniform sparsity. Thus, the FIFOs between different levels of tree-adders are applied as the interface to speed up the total execution time. Level-1 (L1) accumulator is illustrated as Fig. 7. Four 1-bit FIFOs are designed to store the corresponding ready signals from the previous stages. As all FIFOs are not empty, the adder tree is activated, and the pop signal is broadcasted to the 4 PEs. The output of the adder tree is stored in the output FIFOs temporally. Meanwhile, the output ready signal is transferred to L2 accumulator quad-tree. The design of L2 accumulator is the same as that of L1 accumulator. For L3 accumulator, the output FIFO can be removed since the outputs are complete output neurons for the ReLU.

IV. IMPLEMENTATION AND SIMULATION RESULTS

The proposed energy-efficient accelerator is evaluated using 2 popular CNN benchmarks, including AlexNet on ImageNet [9] which contains 256x256 images across 1000 classes and VGG-16 on CIFAR-10 [10] which contains 32x32 images across 10 classes. The results are simulated...
by CONV3–CONV5 on AlexNet and all convolution layers on VGG-16. The proposed accelerator is designed and implemented using TSMC 40nm CMOS technology. The operation voltage and frequency are 0.9V and 750 MHz, respectively.

The energy-efficiency of the overall system is dominated by off-chip DRAM accesses. Fig. 8 illustrates the number of DRAM accesses for each layer of VGG-16 via a dense CNN and a sparse compressed CNN, respectively. Based on the proposed accelerator for the sparse compressed CNN, the computed activations of most layers can be directly transferred back to the PE array, except for CONV1_1 and CONV1_2 in VGG-16 due to the tiling process. Consequently, the weights of CONV1_1 and CONV1_2 have to be accessed 4 times for 4 tiles also. Thus, the DRAM accesses of the sparse CNN are larger than those of dense CNN in the first 2 layers. Thanks to the contribution of deep compression scheme, the total number of DRAM accesses can be reduced by 88.67% over the dense CNN. Table I lists the memory size of on-chip SRAM for the dense and sparse CNN accelerators. The total memory size of the sparse accelerator can be reduced by 23.5% by reducing the activation memory.

For the first few layers, the speedup is not obvious since the zero values almost concentrate at the same row. Overall, the sparse architecture outperforms by 1.79x speedup by skipping unnecessary computation. The energy consumption of the dense CNN accelerator and sparse CNN accelerator is analyzed as shown in Fig. 9, excluding DRAM energy. By decreasing the latency and power consumption of the sparse accelerator, almost 70% energy reduction is achieved for VGG-16. Table II lists the summarized results of the dense and sparse compressed CNN accelerators on AlexNet and VGG-16. The sparse accelerator is employed. Compared to the dense accelerator, the proposed sparse compressed CNN accelerator achieves lower DRAM accesses, lower bandwidth requirement, higher performance and lower energy consumption by compressing weights/activations and skipping redundant computation.

V. CONCLUSIONS

Deep CNNs are rapidly rising in popularity across a broad range of applications. However, deep CNNs are both memory intensive and computationally intensive. Hence, a sparse compressed CNN accelerator is presented to improve the overall performance and energy efficiency in this paper. The relative-indexing memory is realized to store NZ activations and weights for reducing the number of memory accesses and the size of on-chip memory. For increasing the resource utilization of 64 PEs, the CNN workload is distributed based on channels. Additionally, all-row-to-all-row NZ element multiplication is adopted to skip redundant computation. Compared with a dense CNN accelerator, the total on-chip memory size is reduced by 23.51%. Overall, this sparse accelerator achieves 1.79x speedup, 69.53% energy reduction, and 88.67% reduction in DRAM accesses of VGG-16.

REFERENCES