2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture

Energy-efficient Neural Network Accelerator Based on Outlier-aware Low-precision Computation

Eunhyeok Park, Dongyoung Kim, Sungjoo Yoo
Department of Computer Science and Engineering
Neural Processing Research Center (NPRC)
Seoul National University, Seoul, Korea
{*eunhyeok.park, dongyoungkim42, sungjoo.yoo*}*@gmail.com*

***Abstract*—Owing to the presence of large values, which we call outliers, conventional methods of quantization fail to achieve significantly low precision, e.g., four bits, for very deep neural networks, such as ResNet-101. In this study, we propose a hardware accelerator, called the *outlier-aware accelerator* (*OLAccel*). It performs dense and low-precision computations for a majority of data (weights and activations) while efficiently handling a small number of sparse and high-precision outliers (e.g., amounting to 3% of total data). The OLAccel is based on 4-bit multiply-accumulate (MAC) units and handles outlier weights and activations in a different manner. For outlier weights, it equips SIMD lanes of MAC units with an additional MAC unit, which helps avoid cycle overhead for the majority of outlier occurrences, i.e., a single occurrence in the SIMD lanes. The OLAccel performs computations using outlier activation on dedicated, high-precision MAC units. In order to avoid coherence problem due to updates from low- and high-precision computation units, both units update partial sums in a pipelined manner. Our experiments show that the OLAccel can reduce by 43.5% (27.0%), 56.7% (36.3%), and 62.2% (49.5%) energy consumption for AlexNet, VGG-16, and ResNet-18, respectively, compared with a 16-bit (8-bit) state-of-the-art zero-aware accelerator. The energy gain mostly comes from the memory components, the DRAM, and on-chip memory due to reduced precision.**

***Keywords*-neural network; accelerator; quantization;**

I. Introduction

Hardware accelerators for neural networks have been extensively studied in recent years [1]–[6]. Of their benefits, such as low latency and low cost (in case of mass production), energy efficiency is critical to determining the suitability of neural networks for use in servers and mobile devices.

Reducing precision is a viable means of improving energy efficiency. Migacz recently reported that 8-bit quantization is possible without loss of accuracy in deep neural networks [7]. Park et al. proposed weighted entropy-based quantization that enables 5-bit weight/6-bit activation for such deep neural networks as ResNet-101 [8]. In our study, we aim to further reduce bitwidth. We propose a hardware accelerator that allows for 4-bit implementations in very deep models with a negligible loss of accuracy while yielding the benefits obtained by reducing precision, i.e., higher energy efficiency.

Note that we target multi-bit quantization for the weights and activations of deep neural networks. Although binary- or ternary-weighted networks [9], [10] are useful for medium-sized neural networks, e.g., AlexNet, it is known that very deep neural networks, such as ResNet-101, require more bits for weights and activations to maintain the quality of the output.

Our proposed accelerator is based on a novel quantization method called *outlier-aware quantization* [11], which divides the distribution of data (weights or activations) into two regions, of low and high precision. It applies reduced precision, e.g., 4-bit representation, to the low-precision region that contains a majority of the data. The high-precision region contains only a small portion (e.g., 3%) of the total data, and maintains the original, high precision, e.g., 16-bit representation. We call the data in the high-precision region *outliers*, as they are far fewer in number and larger in size than data in the low-precision region. Outlier-aware quantization enables low precision, e.g., four bits, for very deep models, such as ResNet-101 and DenseNet-121, with a very small (3%) ratio of outliers at a negligible ($<$1%) loss of accuracy.

In this study, we propose a hardware accelerator that performs dense and reduced precision computations on a majority of data in the low-precision region while performing sparse and high-precision computation on outliers. The proposed accelerator, called the outlier-aware accelerator (*OLAccel*), achieves a significant reduction in energy consumption by reducing the amount of memory access, and by using smaller units of computation.

The contributions of this paper are as follows:

- We propose an accelerator called OLAccel that implements 4-bit computations on very deep neural networks, e.g., ResNet-101 and DenseNet-121.
- OLAccel differently handles outlier activations and weights for computational efficiency.
- OLAccel performs sparse high-precision computation for outlier activations in parallel with dense low-precision computation for a majority of activations.

2575-713X/18/$31.00 ©2018 IEEE
DOI 10.1109/ISCA.2018.00063

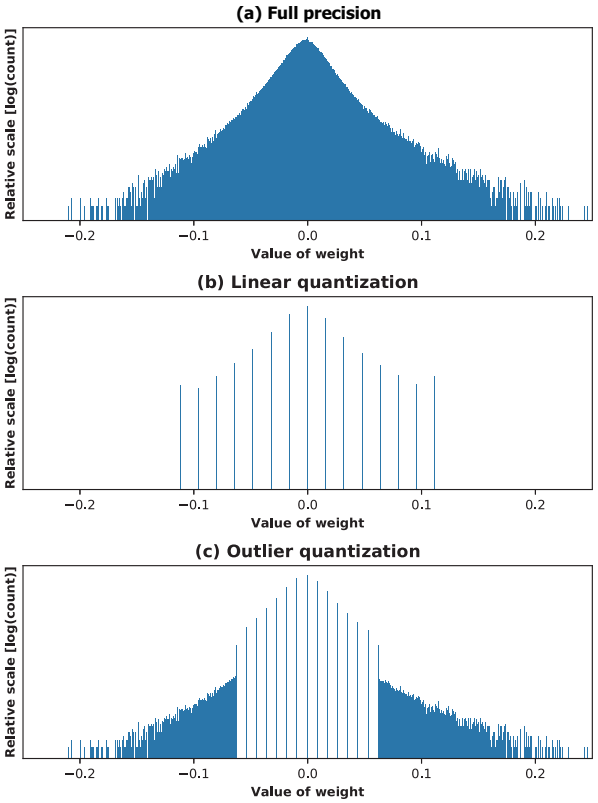

Figure 1: Weight distributions of full precision (a), linear quantization (b), and outlier quantization (c).

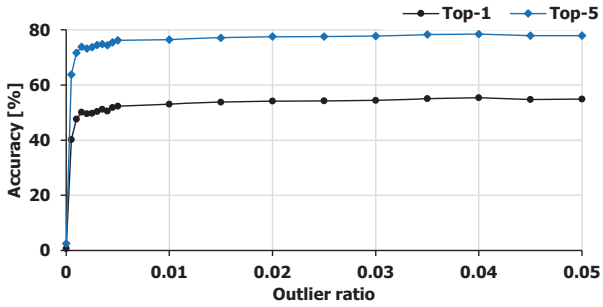

Figure 2: Accuracy vs. outlier ratio: AlexNet.

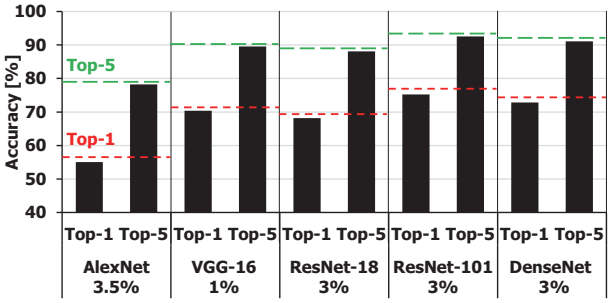

Figure 3: Accuracy of AlexNet, VGG-16, ResNet-18, ResNet-101, and DenseNet-121 with 4-bit outlier-aware quantization. The dashed lines represent the top-5 accuracy of a full-precision network.

- OLAccel reduces the number of additional execution cycles due to outlier weights by equipping SIMD lanes with an outlier MAC unit that runs in parallel with them to reduce latency in most cases of outlier weight occurrence.
- OLAccel skips computations with zero-input activation, thereby further improving energy efficiency.
- We designed OLAccel in Verilog and compare it with prevalent accelerators, Eyeriss [12] and ZeNA [5] at reduced precision.

II. Motivation: Outlier-aware Quantization

Figure 1 shows the distribution (in log scale) of weights in the second convolutional layer of a trained AlexNet. As the figure shows, the distribution is wide due to a small number of outliers with large absolute values. Most existing quantization methods divide the distribution into multiple levels with uniform inter-level spacing while often truncating large values [7], [13], [14]. As Figure 1 (b) shows, they waste most levels due to those outliers while assigning only a small number of levels to the majority of small data items. In case of very low precision, e.g., four bits, such an inefficient allocation of quantization levels incurs prohibitively large quantization errors, which makes quantization fail at such low precision.

The basic idea underlying outlier-aware quantization [11] is to divide a given distribution into two regions, low- and high-precision ones, as shown in Figure 1 (c). The low-precision region covers a majority (e.g., 97%) of data, and represents them with reduced precision, e.g., four bits. Compared with conventional quantization methods [7], [13], [14], given the same bitwidth, the low-precision region has more fine-grained levels with small inter-level spacing, which contributes to reducing the quantization errors in a majority of data. The high-precision region contains outliers with large values that are, thus, considered important. The outliers maintain their original high precision, e.g., 16 bits. Thus, outlier-aware quantization reduces the quantization errors in a majority of small data items by using fine-grained quantization levels while preserving the precision of important, large values, i.e., the values of the outliers, without additional retraining or fine-tuning.

Handling outliers is typically more expensive than normal dense data in terms of memory usage and computation. To reduce the cost of outlier handling, it is important to keep the outlier ratio as small as possible. Figure 2 shows the relationship between accuracy and outlier ratio (for 16-

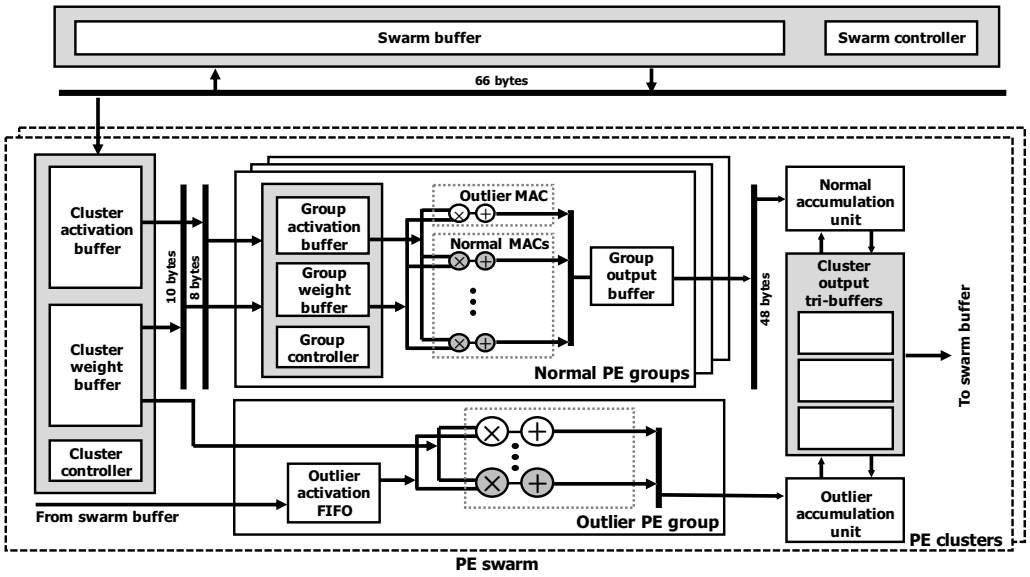

Figure 4: Overall structure of OLAccel.

bit outliers) for a 4-bit AlexNet. As the figure shows, an outlier ratio of 3.5% (3.5% of total weights and non-zero activations are outliers) yields less than a 1% loss in top-5 accuracy. Note that an outlier ratio of 0% corresponds to conventional linear quantization without truncation and retraining, i.e., fine-tuning. As the figure shows, conventional 4-bit quantization without fine-tuning suffers from significant loss (76.57%) of accuracy.

In the case of activation, we calculate the outlier ratio for non-zero activations. Thus, when rectified linear unit (ReLU) is used as activation function, as it transforms negative values into zero, the effective outlier ratio can decrease. For instance, when an outlier ratio of 3% is used, the effective outlier ratio (= number of non-zero outliers/total number of activations) can be lower than 3%.

When applying the outlier ratio during neural network execution, we need to compute the histogram of activations, which is expensive. Thus, for a given neural network, we apply sample inputs (100 randomly sampled images in our experiments) to the network and obtain a threshold value of activation for each layer, which corresponds to the magnitude of activation along the boundary of the outlier, i.e., the high-precision region. We then compare the activations with a threshold value to determine outliers at runtime.

Figure 3 shows the accuracy and outlier ratio of deep neural networks. As the figure shows, 4-bit quantization with an approximately 3% outlier ratio maintains the accuracy of the deep neural networks. Note that, in all the networks, the first convolutional layer takes the raw input activations in 16 or 8 bits while 4-bit input activations are used in the other layers. In addition, ResNet-18 and ResNet-101 require 8-bit weights at the first convolutional layer to meet the accuracy constraint while 4-bit weights are used for the other layers. It is often the case that a larger bitwidth is needed for the weights of the first layer(s), as it is more sensitive to such optimizations as quantization than the other layers [14], [15].[1] Note also that our proposed 4-bit accelerator supports 4-/8-/16-bit data, as is explained in the next section.

III. Proposed Architecture

A. Overall Structure

The computational micro-architecture is important, especially for performance. We use 4-bit data that reduce buffer size and MAC unit overhead significantly. This emphasizes the control overhead of the accelerator, and thus an SIMD structure is needed to minimize overhead. However, the conventional SIMD structure is not appropriate for outlier-aware quantization. For instance, if we use 4-bit precision in 16-way SIMD accelerators like DianNao [1] and SCNN [6], the performance overhead for outlier handling can be significant owing to the high probability of outlier occurrence, e.g., 27.5% $(= 1 - 0.99^{32})$, even with 1% of outliers. OLAccel addresses this with two following novel solutions: outlier PEs and outlier PE groups, and by supporting data structures for sparse and dense data.

Figure 4 shows the overall structure of OLAccel. At the top of the hierarchy is a PE swarm consisting of multiple PE clusters, a swarm buffer, and a controller. As the figure shows, each PE cluster has cluster activation/weight buffers

[1] Although we apply retraining-free quantization in this paper, fine-tuning can reduce the bitwidth of weights from 8 to 4 bits for the first convolutional layer.

(left), PE groups (center), and a cluster output buffer (right). Each PE group consists of multiple MAC units and group activation/weight/output buffers. OLAccel is based on 4-bit MAC units[2] (and a small number of full precision MAC units), and supports computation with 4-/8-/16-bit activations and 4-/8-bit weights.

As the figure shows, there are two types of PE groups, normal and outlier. The normal PE group contains 17 (16 normal and one outlier) 4-bit MAC units while the outlier PE group contains 17 mixed-precision MAC units that support 16-bit $\times$ 4-bit operations. Note that each normal 4-bit or 16-bit MAC unit is in charge of producing a partial sum of its associated output channel. Thus, on each cycle, it receives its associated weight and a broadcast activation and produces a partial sum. As will be explained in Section III-D, the additional outlier MAC unit in the PE group is involved in multiplication using outlier weights. Thus, the group weight buffer contains both normal (4-bit) and outlier (8-bit) weights while the group activation buffer contains only 4-bit activations. As the figure shows, the outlier PE group reads 4-bit or 8-bit weights from the cluster weight buffer, and 8-bit or 16-bit outlier activations from the swarm buffer. It then performs computations using outlier activations. Note that the outlier activations are stored only in the swarm buffer while outlier weights can be stored in the swarm buffer and the cluster/group weight buffers.

The figure shows two (normal and outlier) accumulation units connected to the tri-buffer containing the output of the cluster. Each accumulation unit adds the results of the associated (normal or outlier) PE group with the associated partial sum stored in the tri-buffer. To resolve the issue of coherence between the accumulation units, their execution is pipelined. The outlier accumulation unit accesses partial sums only once the normal accumulation unit finishes adding the partial sums. Thus, both accumulation units run in parallel in a pipeline. To support the required bandwidth for the normal and outlier accumulation units, the tri-buffer containing the cluster output consists of three buffers, two for the normal accumulation unit and one for the outlier accumulation unit.

B. Dataflow

Figure 5 shows the data structures of the normal/outlier weights and normal activations. In the figure, we use the following notation for the sake of a clear and concise description. In the case of activation, the width, height, and the number of channels of the 3D tensor are expressed as $A_{w\times h\times c}$. The kernel weight uses a similar notation, $K_{w\times h\times i\times o}$, which represents the width, height, and the numbers of input and output channels of a 4D tensor. As the figure shows, the cluster weight buffer stores a subset of

[2]In this paper, we explain OLAccel based on 4-bit MAC units. However, it is not limited to 4-bit precision, and can be easily extended to other base bitwidths, e.g., 2 or 8 bits.

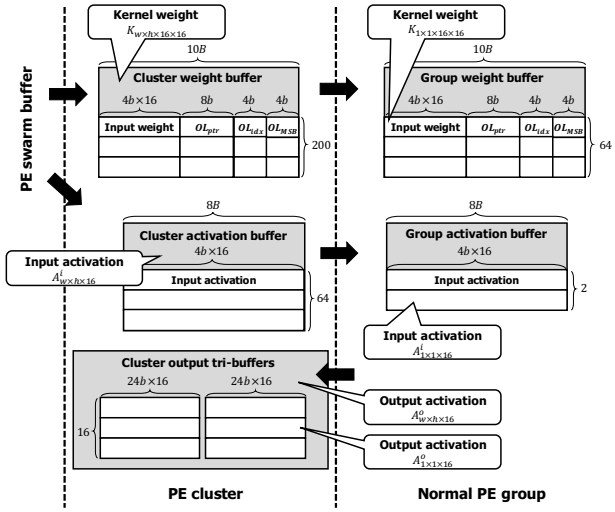

Figure 5: Data structure associated with normal/outlier weight and normal activation.

kernel weights, e.g., $K_{w\times h\times 16\times 16}$. As shown in the box of the cluster weight buffer in the figure, the weights are stored at a granularity of 80-bit weight chunks (entries in the table). The weight chunk consists of 16 4-bit weights (= $4b\times 16$), an 8-bit pointer (OL_{ptr}), a 4-bit pointer (OL_{idx}), and the most significant four bits of an outlier weight (OL_{MSB}). As the figure shows, the cluster weight buffer contains 200 weight chunks. The PE group receives these chunks and stores them in the group weight buffer for its execution.

The weight chunk represents a set of 16 kernel weights, each of which belongs to one of 16 output channels. In case there is no outlier weight among the 16 weights, the two pointers, OL_{ptr}, and OL_{idx}, and the 4-bit field OL_{MSB} are all zero. When an outlier weight exists, OL_{idx} points to the index (of the 16) of the weight in the given chunk and OL_{MSB} contains the most significant four bits of the 8-bit outlier weight. The remaining least significant three bits and a sign bit of the outlier weight are stored in the associated position of 4×16 weight bits in the chunk. In case there is more than one outlier weight among the 16 weights, their most significant four bits are stored in (the 64b field of the input weight of) another weight chunk, and OL_{ptr} points to that chunk in the cluster weight buffer (or group weight buffer). Thus, the PE group checks OL_{ptr} to see if there are more than one outlier weights in the weight chunk. We explain this in detail in Section III-D.

Figure 5 also shows how low-precision input activations are stored in the PE cluster and group. The cluster activation buffer stores a subset of input activations of $A^i_{w\times h\times 16}$, which consist of activation chunks each of which has 16×4-bit input activations. The PE group receives input activations at the granularity of the chunk. As the figure shows, the cluster activation buffer stores 64 activation chunks while the group

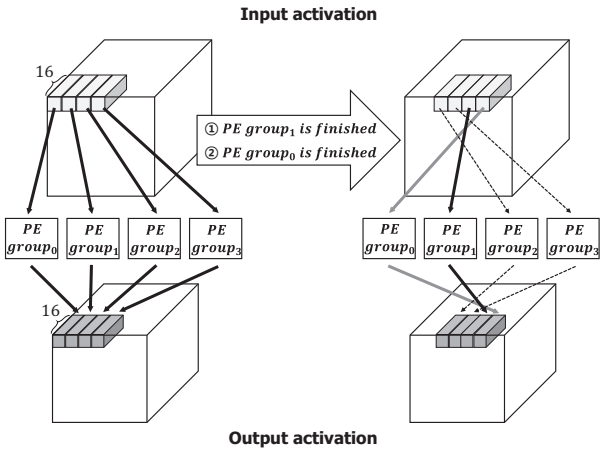

Figure 6: Operation of PE cluster.

activation buffer stores two chunks. As shown at the bottom of the figure, the cluster output tri-buffer manages output partial sums with a larger chunk size, $24b \times 16$, as a partial sum requires higher precision (24 bits) than that of the input (4/8/16 bits).

C. PE Cluster

Figure 6 shows how the PE cluster controls the execution of its PE groups. As the figure shows, a PE group takes as input an activation chunk $A^i_{1\times1\times16}$ and generates as output a partial sum chunk of $A^o_{1\times1\times16}$. The figure illustrates a scenario where four PE groups run in parallel.

The PE group skips computations with zero-input activations. Thus, owing to the different numbers of non-zero values in the input activation chunks, some PE groups can finish earlier than others. In the figure, we assume that PE group $_1$ finishes followed by group $_0$, before the other PE groups. In such a case, as shown on the right-hand side of the figure, the PE cluster allocates new input activation chunks to the PE groups that are ready, which keeps them busy as far as there are available input activations in the cluster activation buffer. This mechanism is important in OLAccel as it enables high utilization of MAC units.

D. Normal PE Group

Figure 7 shows how the PE group functions when there is one outlier among the 16 4-bit weights. The group activation buffer shows that the input activation chunk has only three non-zero activations (a_0, a_3, and a_{15}) among the 16 activations. The PE group selects, from the input activation chunk, a non-zero activation, e.g., a_0, and broadcasts it to the 16 normal and one outlier MAC units. The PE group also selects a weight chunk from the group weight buffer and provides the 16 normal MAC units with their associated kernel weights. The figure shows that the selected weight chunk contains an outlier weight, the index of which, OL_{idx}

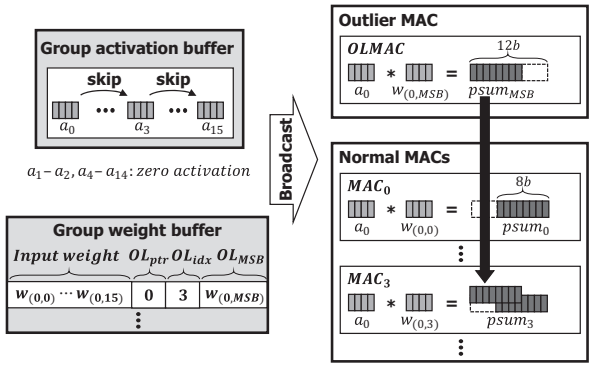

Figure 7: PE group operation in case that there is an outlier weight among 16 weights.

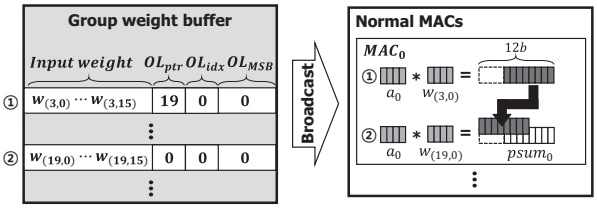

Figure 8: PE group in the case of more than one outlier weight.

is three. Thus, the outlier MAC unit takes as input the most significant four bits of the outlier weight OL_{MSB} from the weight chunk and multiplies it with the broadcast activation a_0. Then, the result is broadcast to all normal MAC units and the normal MAC unit pointed by OL_{idx} receives it. As the figure shows, the MAC unit, MAC_3, which performs computation with the least significant four bits of the outlier weight (in the input weight field of the weight chunk), receives and adds it to its partial sum. The entire operation of 17 MAC units can be carried out in one clock cycle. When there is no outlier weight, the PE group runs in the same manner as above, but OL_{MSB} stores the value zero. Thus, the outlier MAC unit generates a zero result, which does not affect the partial sum of the normal MAC units.

Figure 8 shows how the PE group performs computation when there are more than one outlier weights in the weight chunk. As mentioned above, in such a case, two weight chunks are used. In the figure, the first weight chunk sets the pointer, OL_{ptr}, to the index of the second weight chunk, e.g., 19. In this case, the MAC operation takes two clock cycles to perform an 8-bit weight $\times$ 4-bit activation. Thus, in the first clock cycle, the 16 4-bit weights (the least significant four bits) of the first chunk are provided to the 16 normal MAC units. In the next clock cycle, the 4-bit weights of the second weight chunk are provided. Each normal MAC units adds the two results. Note that the outlier MAC is not used

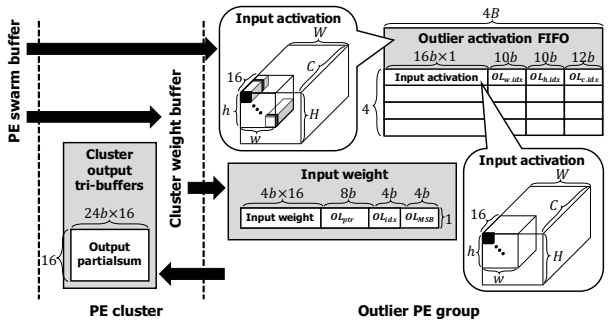

Figure 9: Data structure associated with outlier activation.

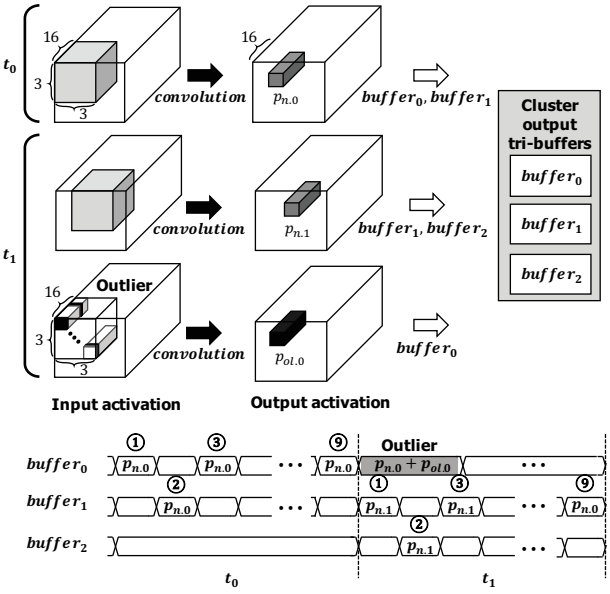

Figure 10: Cluster output tri-buffer.

in this scenario. Moreover, MAC units with normal 4-bit weights become idle in the second clock cycle.

E. Outlier PE Group and Cluster Output Tri-buffer

Figure 9 shows the data structure and data flow in the outlier PE group. As the figure shows, input outlier activations are fetched from the swarm buffer to the outlier PE group. The activations are sparse data. Thus, as shown in the figure (Outlier activation FIFO), each outlier activation is represented by an outlier chunk consisting of a 16-bit activation and the three coordinates ($OL_{w.idx}$, $OL_{h.idx}$, and $OL_{c.idx}$) of the outlier in the tensor of the input activation. The outlier PE group also fetches weight chunks from the cluster weight buffer. As mentioned above, the outlier PE group consists of 17 mixed-precision MAC units and runs in the same way as the normal PE group. Thus, the non-zero activation is broadcast to the 17 MAC units and multiplied by the associated weights to produce partial sums for the 16 output channels. The partial sums (16 24-bit data items) are stored in the cluster output tri-buffer as shown in the figure.

Figure 10 shows how the normal and outlier accumulation units accumulate the partial sums of the normal and outlier PE groups in a pipelined manner while accessing the three buffers in the cluster output tri-buffer. The figure illustrates a scenario of the pipeline at times t_0 and t_1. At t_0, the normal accumulation unit calculates 16 partial sums by accessing two buffers, $buffer_0$ and $buffer_1$, in the cluster output tri-buffers. Then, at t_1, the normal accumulation unit accesses two buffers, $buffer_1$ and $buffer_2$, while the outlier accumulation unit accesses $buffer_0$. Thus, there is no problem of coherence when the outlier accumulation unit accesses the buffer. To complete partial sum calculations for a convolution, e.g., a 3×3 convolution in the figure, multiple stages of the pipeline operation are needed.

IV. Evaluation Methodology

We implemented outlier-aware quantization in the PyTorch and Caffe environments [16], [17]. We used randomly selected data in TorchVision [18] and Caffe [17] to calculate average energy consumption and execution cycles by running neural networks on our hardware accelerator models. In case of AlexNet and VGG-16, we used pruned models [15] to evaluate the capability of zero skipping while we pruned ResNet-18 on our own.

For the baseline and our architectures, we developed cycle-accurate models. We did a Verilog design and synthesized them using Design Compiler with a commercial 65-nm LP library (typical) at 250 MHz and 1.0 V. We used CACTI to estimate SRAM area/power/latency [19] and the DRAM power model from Micron [20].

We compare our 4-bit OLAccel with the following baselines:

- 16-bit and 8-bit Eyeriss (denoted by Eyeriss16 and Eyeriss8, respectively), which, for zero input, do not reduce the number of execution cycles but clock-gate computations [12].
- 16-bit and 8-bit ZeNAs that reduce the number of execution cycles by skipping computations with zero input weights and activations [5].[3]

In 16/8-bit comparison, we utilize 16/8-bit raw input activation for all the networks. In [7], Migacz reported that the minimum bitwidth for linear quantization (without loss of accuracy) is eight bits for deep networks like ResNet-101/152 and GoogLeNet. In case of 8-bit Eyeriss and ZeNA, we assume that existing accelerators support eight bits and there is no degradation in accuracy by using the aforementioned quantization. In 8-bit comparison, we use

[3]We chose ZeNA as the baseline for the zero-skipping accelerator as it provides the best speedup for AlexNet by skipping both zero weights and activations, compared with other zero-skipping baselines.

8-bit outlier activations for OLAccel based on the same assumption.

	Eyeriss		ZeNA		OLAccel	
	8-bit	16-bit	8-bit	16-bit	8-bit	16-bit
# PEs	165	165	168	168	576	768
area (mm^2)	0.96	1.53	1.01	1.66	0.93	1.67
On-chip memory (AlexNet)	Act	393 kB (16-bit) / 196 kB (8-bit)				
	Weight	16 kB (16-bit) / 8 kB (8-bit)				
On-chip memory (VGG-16 & ResNet-18)	Act	4.8 MB (16-bit) / 2.4 MB (8-bit)				
	Weight	16 kB (16-bit) / 8 kB(8-bit)				

Table I: Configurations of Eyeriss, ZeNA, and OLAccel.

Table I shows the architectural configurations used in our experiments. In case of 16 (8)-bit comparison, OLAccel has 16 (8)-bit outlier activations with 4-bit MAC units and 8-bit outlier weights. We perform an ISO-area comparison between the baselines and OLAccel. Thus, given the same amount of on-chip memory, which keeps all the data required for a layer to avoid off-chip memory accesses, each architecture is allocated the same chip area for logic and buffer implementations as Eyeriss. In case of 8-bit Eyeriss, we first obtain the area of 165 8-bit processing elements (PEs) [12][4]. We then determine the configuration (number of MAC units) of OLAccel to meet the given area target. In 16 (8)-bit comparisons, Eyeriss, ZeNA, and OLAccel occupy 1.53 (0.96) mm^2, 1.66 (1.01) mm^2, and 1.67 (0.93) mm^2, respectively. ZeNA has 168 PEs in both the 16- and 8-bit cases. In these cases, we use 393 kB (16-bit comparison) and 196 kB (8-bit) of on-chip memory for AlexNet, and 4.8 MB and 2.4 MB for VGG-16 and ResNet-18, respectively. We use the same memory size for the three accelerators on each network for fair comparison. A single large memory for different networks is advantageous for OLAccel, especially on AlexNet as OLAccel benefits from reduced memory access, and the larger memory renders the memory more dominant in terms of energy consumption. In case of OLAccel, the on-chip memory is used as the swarm buffer.

In case of 16-bit comparison, OLAccel is equipped with eight PE clusters, six PE groups/cluster, and a total of 768 (= $8 \times 6 \times 16$) 4-bit MAC units. However, in case of 8-bit comparison, OLAccel has a smaller number of 4-bit MAC units, 576 (in six PE clusters), to satisfy the constraint of the reduced area, 0.96 mm^2.

V. Experimental Results

Figure 11 compares the number of execution cycles and energy consumption (normalized to Eyeriss16) on AlexNet.

[4]In the case of Eyeriss and ZeNA, the PE consists of a MAC unit and internal buffers.

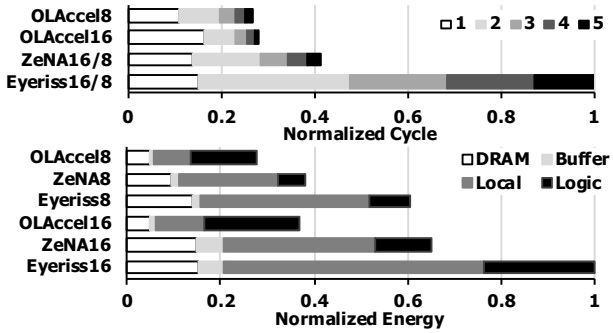

Figure 11: AlexNet cycle and energy breakdown.

Compared with Eyeriss16, OLAccel16 yields a 71.8% reduction in execution cycles.[5] This is mainly due to a large number of MAC units (768 in OLAccel versus 165 in Eyeriss), which is enabled by reduced precision (from 16 to 4 bits). Compared with ZeNA16 containing 168 PEs, OLAccel16 gives a 31.5% reduction in execution cycles. The reduction is smaller than in Eyeriss because ZeNA skips computations for zero weights and activations, whereas OLAccel skips computations for only zero activations. In addition, OLAccel16 spends a long execution cycle for the first convolutional layer due to the handling of 16-bit raw input activation on 4-bit MAC units.

OLAccel8 gives an 35.1% reduction in execution cycles compared with ZeNA8. Due to the smaller number (576) of 4-bit MAC units, OLAccel8 slows down from the second to fifth convolutional layer compared to OLAccel16. However, the input activation of the first layer has 8 bits, which reduces the execution cycle of the first convolutional layer significantly compared with OLAccel16 thereby enabling OLAccel8 to outperform OLAccel16.

Figure 11 also compares energy consumption (normalized to Eyeriss16) decomposed into DRAM, on-chip memory or swarm buffer (Buffer), the local buffer of the PE or PE clusters/groups (local), and logic circuits, e.g., MAC unit and bus (logic). OLAccel16 (OLAccel8) yields a 43.5% (27.0%) reduction compared with ZeNA16 (ZeNA8). As the figure shows, the reduction is mainly due to energy reduction in the memory components (DRAM and SRAM buffers) enabled by reduced precision.

Figure 12 shows comparisons for VGG-16. Compared with ZeNA8/16, OLAccel16 and OLAccel8 reduce by 45.3% and 28.3% execution cycles, respectively. OLAccel16 delivers the best performance owing to a large number (768) of MAC units. The performance improvement of OLAccel16 is greater than in the case of AlexNet because the effect of the first convolutional layer is mitigated due to the speedup of the other layers. Meanwhile, OLAccel8

[5]Note that both 16-bit and 8-bit Eyeriss (ZeNAs) yield the same number of execution cycles as we maintain the same number of PEs, 165 (168 in ZeNA), in both cases.

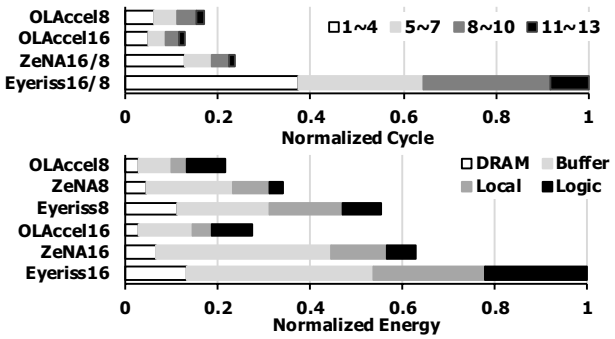

Figure 12: VGG-16 cycle and energy breakdown.

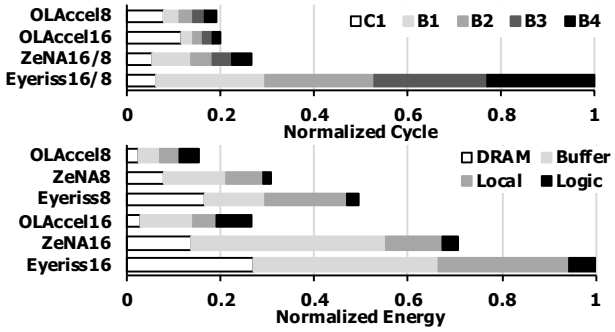

Figure 13: ResNet-18 cycle and energy breakdown.

shows worse performance than OLAccel16 because of the small number (576) of PEs. The comparison of energy consumption shows that OLAccel16 and OLAccel8 give a reduction of 56.7% and 36.3% compared with ZeNA16 and ZeNA8, respectively. As in the case of AlexNet, the reduced precision enables OLAccel to significantly reduce energy consumption in the memory components.

In the case of ResNet-18, as shown in Figure 13, OLAccel significantly reduces the number of execution cycles compared with Eyeriss. As in the case of AlexNet, ResNet-18 has a long execution cycle for the first convolutional layer. Especially, as shown in Figure 3, the first convolutional layer of ResNet-18 requires dense computation with 8-bit weights and 16/8-bit input activations in 16/8-bit comparison. Thus, in 16 (8)-bit comparison, the first convolutional layer takes 8 (4) times longer execution cycle than a simple 4-bit dense computation, which makes the first convolutional layer (C1 in Figure 13) occupy half the total execution cycle of OLAccel16. However, owing to the speedups of the other layers where 4-bit dense computation is performed, OLAccel is able to reduce execution cycles more significantly than in the case of AlexNet. Specifically, compared with Eyeriss16 (Eyeriss8), OLAccel16 (OLAccel8) reduces the numbers of cycles by 80.1% (81.1%) in ResNet-18 while it gives a reduction of 71.8% (73.2%) in AlexNet.

Compared with ZeNA, OLAccel16 and OLAccel8 reduce

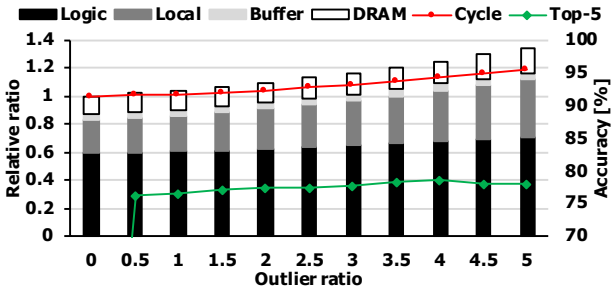

Figure 14: Normalized energy and cycle vs. outlier ratio: AlexNet on OLAccel16.

by 25.3% and 29.0% execution cycles, respectively. Considering the fact that, as the decomposition of execution cycle in OLAccel and ZeNA shows in Figure 13, OLAccel is superior to ZeNA in the other layers except the first one, we expect that OLAccel can give much better performance than ZeNA in deeper networks, e.g., ResNet-101.[6]

OLAccel significantly reduces energy consumption in ResNet-18. Compared with ZeNA, OLAccel16 and OLAccel8 give energy reductions of 62.2% and 49.5%, respectively. Note that, aside from the effect of the first layer, OLAccel gives more reduction in energy consumption for VGG-16 and ResNet-18 than AlexNet because the larger on-chip memory used for VGG-16 and ResNet-18 (Table I) makes the energy consumption of on-chip memory more dominant thereby amplifying the effects of reduced precision, i.e., reduction in on-chip memory traffics.

Figure 14 shows the impact of outliers on the number of execution cycles, energy consumption, and accuracy when running AlexNet on OLAccel16. As the figure shows, as outlier ratio increases, both energy consumption and the number of execution cycles increase while accuracy improves. In the case where there are 3.5% outliers, compared with the case of 0% outlier, the total energy consumption and the number of execution cycles increase by 20.6% and 10.6%, respectively, while yielding much better accuracy, only 0.8% drop from the top-5 accuracy of full precision.

Figure 15 shows scalability on AlexNet. One instance of neural processing unit (NPU) is equipped with 768 4-bit MAC units in OLAccel (16-bit outliers) and 168 16-bit PEs in ZeNA. As the figure shows, we increase both the number of NPUs and batch size. The speedup (y-axis) is normalized to ZeNA with a batch size of one. The figure shows that both OLAccel and ZeNA exhibit good scalability when the batch size is 4 and 16. In case of a single batch, speedup tends to saturate in both cases when the number of NPUs reaches 16. This is due to low resource utilization. OLAccel

[6]When fine-tuning is adopted, OLAccel can give further speedup in ResNet-18 since fine-tuning can reduce the bitwidth of weights from 8 to 4 bits for the dense computation of the first convolutional layer as mentioned in Section II.

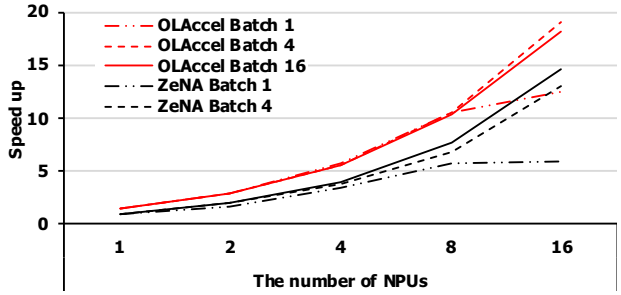

Figure 15: Scalability analysis on AlexNet: speedup (y-axis) vs. number of NPUs.

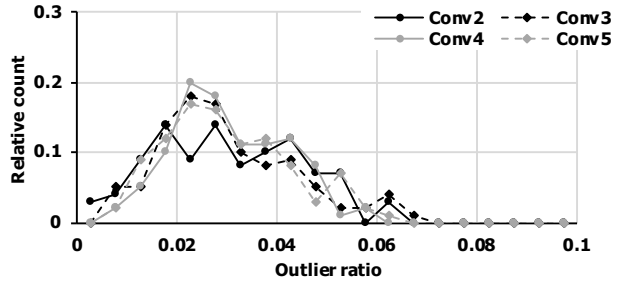

Figure 16: Histogram of outlier activation: outlier ratio of 3%.

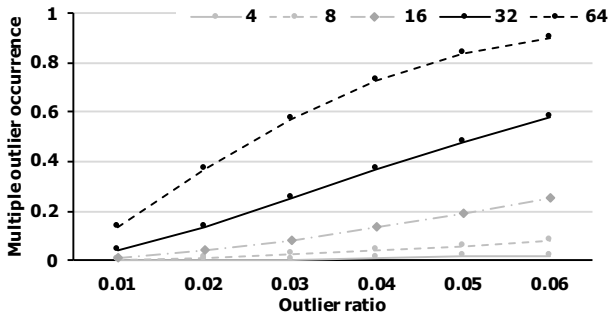

Figure 17: Probability of multiple outlier weights (y-axis) vs. outlier ratio.

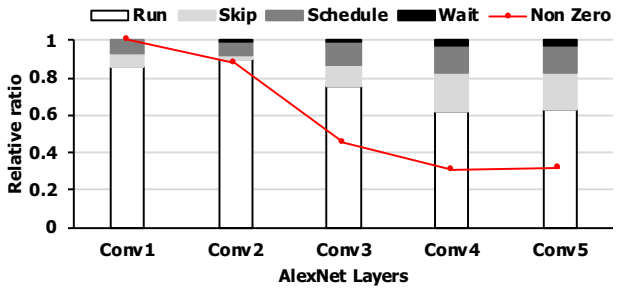

Figure 18: Utilization breakdown: AlexNet.

yields a slightly better speedup in batch 4 than batch 16, mainly due to the off-chip bandwidth limit because batch 16 requires more off-chip memory bandwidth than batch 4.

Figure 16 shows the histogram of outlier activations in AlexNet when the outlier ratio is 3%. As explained above, to avoid computing histograms at runtime, we obtain a static threshold for each layer by utilizing sample inputs during design time and compare the activation with the threshold at runtime to identify outlier activations. The figure shows that the distributions have mean values near 0.03, meaning that our implementation works relatively well. Although not shown in the paper, the neural networks adopting batch normalization layers, e.g., ResNet-101, tend to provide better distributions with sharp peaks near the target outlier ratio.

Figure 17 shows how we determined 16 MAC units in a single PE group. A large number of MAC units in a PE group can improve performance by better exploiting activation broadcast. However, as the figure shows, in case of 32 and 64 MAC units, the probability of multiple outlier occurrences on 32 and 64 weights is higher than 50% at an outlier ratio of 5%. In case of multiple outlier weights in a PE group, OLAccel suffers from long execution cycles. Thus, we set the number of MAC units in a PE group to 16, which yields a smaller probability, e.g., 20%, even at an outlier ratio of 5%. Another reason for our choice is recent trends in neural network architectures. In state-of-the-art architectures like ResNext [21] and Deep Roots [22], the number of channels (on a branch) tends to decrease, e.g., to 32 or 16. Thus, channel-level parallelism can be limited. In such a case, a PE group with a large number of MAC units can suffer from low resource utilization.

Figure 18 shows the utilization breakdown of the convolutional layers in AlexNet. As the figure shows, the active period (Run) is relatively proportional to the ratio of non-zero activations (Non-zero). The figure also shows that the overhead of zero-skip operation (Skip) increases in proportion to the ratio. This is because in the PE group, the zero-activation skip operation is performed every four activations while consuming a constant overhead of a clock cycle. Thus, when there is a large number of zero activations, as in Conv4 or Conv5, it is often the case that zero skipping, without computations, spends cycles only to skip four consecutive zero activations. As the figure shows, the cycle overhead can amount to approximately 20%. In future work, we will work to reduce these overhead cycles.

Figure 19 shows the average number of cycles a PE group spends to process a $A_{1\times1\times16}$ input activation chunk in AlexNet. Due to the difference in the number of non-zero activations, convolutional layers yield different distributions. For instance, Conv2 has a peak near 15 and 16 cycles owing to the high ratio of non-zero activations shown in Figure 18. On the contrary, Conv4 and Conv5 show similar distributions with peaks near five cycles, which can also be explained by the low ratio of non-zero activations in Figure 18.

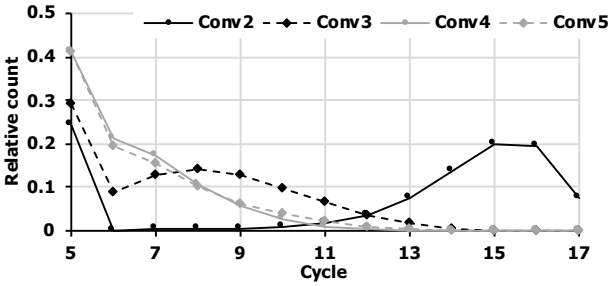

Figure 19: Execution cycles when processing a chunk of $A_{1\times1\times16}$ input activations.

VI. Related Work

Quantization has been extensively studied in recent years. Rastegari et al. showed that a binary-weight AlexNet maintains the same accuracy as a full precision one [9]. However, the binarization of weights and activations incurs a significant loss in accuracy. In [10], Zhu et al. proposed a ternary-weight quantization that can generate a ternary-weight model of ResNet-18 (with full precision activations) at a 2% loss in top-5 accuracy. Zhou et al. proposed a DoReFa network that clips activation to reduce the range of values, thereby improving the utility of quantization levels [14]. They reported that 1-bit weight and 2-bit activation incur only a 6% loss in top-1 accuracy in AlexNet. However, no study to date has examined very deep models like ResNet-101. Miyashita et al. proposed applying logarithm-based quantization and showed that AlexNet can be quantized with 4-bit weights and 5-bit activations at a 1.7% loss in top-5 accuracy [23]. No report has examined the results of logarithm-based quantization in deeper neural networks either. In [24], Zhou et al. proposed balanced quantization that maintains a uniform population of quantization levels, i.e., keeps it balanced, which contributes to improving the utility of quantization levels. However, their method yields a large loss in accuracy for 4-bit deep networks, e.g., 3.8% on GoogLeNet.

Migacz recently proposed a method of quantization that minimizes cross-entropy between the distributions of the original and quantized values, and showed that 8-bit quantization is possible for deep neural networks [7]. Park et al. proposed a more aggressive quantization method that allows 5-bit weights and 6-bit activations for deep neural networks like ResNet-101 at a very small loss of accuracy. To this end, their method maximizes the weighted entropy in weight clustering and logarithm-based quantization of activations. They reduced memory and bandwidth consumption, but the method requires full-precision computation units as well as additional LUTs, as the weights are clustered and a full-precision weight represents each cluster.

Zero skipping is crucial for performance as well as energy efficiency. In [2], Albericio et al. proposed Cnvlutin that skips multiplications with zero-input activations. Only non-zero input activations are broadcast to MAC units while skipping zero-input activations. In [3], Zhang et al. proposed Cambricon-X, which skips multiplications with zero weights obtained by pruning [15]. In [6], Parashar et al. proposed a sparse CNN (SCNN) that exploits both zero weights and activations by calculating the Cartesian products of non-zero weights and activations, and adding the results to the corresponding partial sums. SCNN suffers from low resource utilization in case of high sparsity and high area/power overhead due to accumulator buffers and the crossbar. In [5], Kim et al. proposed a zero-aware neural network accelerator (ZeNA) that also skips computation with both zero weights and activations. They reported that ZeNA gives a speedup of 4.4x on AlexNet. In our experiments, we use ZeNA as a baseline.

VII. Conclusion

In this paper, we proposed a hardware accelerator called OLAccel. It implements outlier-aware quantization, which provides a majority of data with fine-grained quantization while maintaining the precision of important outliers. OLAccel is based on 4-bit MAC units, and performs 4-bit dense computations on a majority of the data. To efficiently handle high-precision outliers, it has two mechanisms at the levels of the PE group and cluster. The PE group, equipped with an outlier MAC unit, performs computations with a single outlier weight without additional cycles, incurring a cycle overhead only when multiple outlier weights need to be handled. The outlier PE group performs computations with outlier activations using high-precision MAC units. The accumulation of partial sums from the normal and outlier PE groups is performed in a pipelined manner to avoid a coherence problem. Our experiments show that OLAccel reduces energy consumption by 43.5% and 27.0% on AlexNet compared with the state-of-the-art 16-bit and 8-bit zero-aware accelerators, respectively. It provides further energy reduction in VGG-16 (56.7%/36.3% with 16/8 bits) and ResNet-18 (62.2%/49.5%), where the performance degradation due to the first convolutional layer is mitigated and the energy benefit of reduced precision is amplified due to large on-chip memory. The experiments also show that OLAccel has the potential for scalability on large-scale problems.

Acknowledgments

This work was supported by the National Research Foundation of Korea grant funded by the Ministry of Science, ICT & Future Planning (PF Class Heterogeneous High Performance Computer Development, NRF-2016M3C4A7952587), Samsung Research Funding Center of Samsung Electronics (SRFC-TC1603-04), and Samsung Advanced Institute of Technology.

REFERENCES

[1] T. Chen *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM Sigplan Notices*, 2014.

[2] J. Albericio *et al.*, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *International Symposium on Computer Architecture*, 2016.

[3] S. Zhang *et al.*, "Cambricon-x: An accelerator for sparse neural networks," *International Symposium on Microarchitecture*, 2016.

[4] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," *International Symposium on Computer Architecture*, 2017.

[5] D. Kim, J. Ahn, and S. Yoo, "Zena: Zero-aware neural network accelerator," *Design & Test*, 2018.

[6] A. Parashar *et al.*, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *International Symposium on Computer Architecture*, 2017.

[7] S. Migacz, "NVIDIA 8-bit inference width TensorRT," *GPU Technology Conference*, 2017.

[8] E. Park, J. Ahn, and S. Yoo, "Weighted-entropy-based quantization for deep neural networks," *Computer Vision and Pattern Recognition*, 2017.

[9] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," *European Conference on Computer Vision*, 2016.

[10] C. Zhu *et al.*, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[11] E. Park, S. Yoo, and P. Vajda, "Value-aware quantization for training and inference of neural networks," *arXiv preprint*, 2018.

[12] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *International Symposium on Computer Architecture*, 2016.

[13] I. Hubara *et al.*, "Quantized neural networks: Training neural networks with low precision weights and activations," *arXiv preprint arXiv:1609.07061*, 2016.

[14] S. Zhou *et al.*, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[15] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *International Conference on Learning Representation*, 2016.

[16] "Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration," https://github.com/pytorch, accessed: 2018-04-15.

[17] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *International Conference on Multimedia*, 2014.

[18] "Pytorch torchvision," https://github.com/pytorch/vision, accessed: 2018-04-15.

[19] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP Laboratories, HPL-2009-85*, 2009.

[20] "Micron dram calculator," http://www.micron.com/support/tools-and-utilities/power-calc, accessed: 2018-14-15.

[21] S. Xie *et al.*, "Aggregated residual transformations for deep neural networks," in *Computer Vision and Pattern Recognition*, 2017.

[22] Y. Ioannou *et al.*, "Deep roots: Improving cnn efficiency with hierarchical filter groups," *arXiv preprint arXiv:1605.06489*, 2016.

[23] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *arXiv preprint arXiv:1603.01025*, 2016.

[24] S.-C. Zhou *et al.*, "Balanced quantization: An effective and efficient approach to quantized neural networks," *Journal of Computer Science and Technology*, 2017.