An Energy-Efficient Deep Convolutional Neural Network Accelerator Featuring Conditional Computing and Low External Memory Access

Minkyu Kim, *Member, IEEE*, and Jae-sun Seo, *Senior Member, IEEE*

***Abstract*— With its algorithmic success in many machine learning tasks and applications, deep convolutional neural networks (DCNNs) have been implemented with custom hardware in a number of prior works. However, such works have not exploited conditional/approximate computing to the utmost toward eliminating redundant computations of CNNs. This article presents a DCNN accelerator featuring a novel conditional computing scheme that synergistically combines precision cascading (PC) with zero skipping (ZS). To reduce many redundant convolutions that are followed by max-pooling operations, we propose precision cascading, where the input features are divided into a number of low-precision groups and approximate convolutions with only the most significant bits (MSBs) are performed first. Based on this approximate computation, the full-precision convolution is performed only on the maximum pooling output that is found. This way, the total number of bit-wise convolutions can be reduced by ∼2× with <0.8% degradation in ImageNet accuracy. PC provides the added benefit of increased sparsity per low-precision group, which we exploit with ZS to eliminate the clock cycles and external memory accesses. The proposed conditional computing scheme has been implemented with custom architecture in a 40-nm prototype chip, which achieves a peak energy efficiency of 24.97 TOPS/W at 0.6-V supply and a low external memory access of 0.0018 access/MAC with VGG-16 CNN for ImageNet classification and a peak energy efficiency of 28.51 TOPS/W at 0.9-V supply with FlowNet for Flying Chair data set.**

***Index Terms*— Application-specific integrated circuit (ASIC), approximate computing, conditional computing, deep convolutional neural network (DCNN), deep learning, energy-efficient accelerator.**

Manuscript received June 4, 2020; revised August 28, 2020; accepted October 2, 2020. Date of publication October 19, 2020; date of current version February 24, 2021. This article was approved by Guest Editor Mark Oude Alink. This work was supported in part by NSF under Grant 1652866; in part by Samsung Electronics; and in part by the Center for Brain-Inspired Computing (C-BRIC), one of the six centers in Joint University Microelectronics Program (JUMP), a Semiconductor Research Corporation (SRC) Program sponsored by the Defense Advanced Research Projects Agency (DARPA). *(Corresponding author: Minkyu Kim.)*

Minkyu Kim was with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA. He is now with Qualcomm, San Diego, CA 92121 USA (e-mail: minkkim@qti.qualcomm.com).

Jae-sun Seo is with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA (e-mail: jaesun.seo@asu.edu).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2020.3029235

I. Introduction

DEEP convolutional neural networks (DCNNs) have been successfully applied for many real-life applications ranging from computer vision [1], speech recognition [2] to medical diagnosis [3]. However, accurate, high-throughput, and low-power hardware implementations of DCNNs is still a challenging task, especially for portable systems such as smart drones, robots, and mobile/wearable devices [4]–[6]. This is due to high computational complexity, large storage requirement, and costly off-chip communication. For energy-efficient DCNN accelerators, it is crucial to design a computing scheme that can suppress redundant computations, exploit data sparsity statistics, and optimize system-level memory hierarchy that can reduce the amount of off-chip memory accesses [7].

A number of previous works presented application-specific integrated circuits (ASICs) hardware accelerator designs for DCNNs [8]–[16]. Envision [8] presented a voltage-accuracy-frequency-scaling scheme and variable precision technique with body bias modulation, which achieved an energy efficiency of 2 TOPS/W for VGG-16 CNN inference. QUEST [9] proposed a programmable multiple instruction, multiple data (MIMD) parallel accelerator, which achieved 0.877 TOPS/W for AlexNet CNN inference. However, most prior works [6]–[12] do not evaluate or optimize off-chip memory communication, which can largely degrade the system-level energy efficiency.

A few works have analyzed and demonstrated the off-chip memory access [13]–[16]. Eyeriss [13] employed run-length compression to reduce the DRAM communication with zero data. UNPU [15] proposed bit-serial processing with variable weight precision and achieved an energy efficiency of 4.71 TOPS/W with 87.02-MB DRAM access for VGG-16. Sim *et al.* [16] proposed a near-threshold voltage CNN inference processor based on an enhanced output stationary dataflow and achieved 1.15 TOPS/W with 45-MB DRAM access of activations for VGG-16 CNN. Note that the chip designs in [13]–[16] do not include an on-chip DRAM controller, thereby does not include the energy cost for real-time off-chip DRAM communication.

In this article, we present an energy-efficient DCNN accelerator [17] with a new conditional computing scheme that enables significant reduction in the amount of unnecessary

computations and external memory accesses. We prototyped the proposed DCNN accelerator in 40-nm CMOS, which integrated a custom off-chip DRAM controller. We demonstrate high-throughput DCNN inference and enhanced system-level energy efficiency, including the accelerator chip and off-chip memory. The main contributions of our work are as follows.

1) A new conditional computing scheme termed precision cascading (PC) reduces the redundant computations in convolution layers by integrating the subsequent max-pooling layers.
2) An energy-efficient architecture that performs full zero skipping (ZS) of zero activations for both convolution operations and on-/off-chip memory communication.
3) We integrated a custom off-chip DRAM controller in our DCNN accelerator, thereby demonstrated real-time off-chip communication and evaluated the associated energy cost.

Wu *et al.* [18] presented a deep learning processor supporting dynamic executions of conditional neural networks, but this requires generation of new CNNs and has not been demonstrated for large-scale data sets such as ImageNet. Adaptive precision scheme has been presented in [19] only for k-nearest neighbor algorithms, not for large-scale DCNNs. Our proposed conditional computing scheme can be applied to any deep CNN that includes max pooling, without any new CNN generation. We have benchmarked our DCNN accelerator, including both the chip energy efficiency and off-chip memory accesses with VGG-16 CNN for ImageNet data set [20] and FlowNet for Flying Chair data set [28]. In addition to VGG and FlowNet, DCNNs such as SegNet for image segmentation [31] and a new DCNN for high-resolution medical imaging [27] applications feature a relatively large number of max-pooling layers between convolution layers and will be best suited for the proposed accelerator with PC and ZS schemes, enabling large improvements in system-level energy efficiency.

However, many modern DCNNs such as ResNet [32] and MobileNet-V2 [33] only include a small number of max-pooling layers and will not be able to fully exploit the benefits of the proposed PC scheme. On the other hand, recent works on adversarial bit flip attacks [34] and noise in processing-in-memory accelerators [35] have shown that large models such as VGG family show higher resilience to such attacks and noise, compared to compact models such as MobileNet family. For example, Rakin *et al.* [34] reported that, by flipping just two (targeted) bits in the MobileNet-V2 model, the baseline accuracy of 72.01% severely degraded to 0.19% for ImageNet. To that end, while VGG model has redundancy with a large number of parameters, the resilience to adversarial attacks and noise remains an advantage, compared to more compact models.

The remainder of this article is organized as follows. Section II introduces our proposed conditional computing scheme including PC and ZS. We introduce the convolution loop acceleration strategy to reduce the on-/off-chip memory communication in Section III. Section IV presents the system architecture and operation of the proposed accelerator based on PC and ZS schemes. Section V describes the chip implementation and evaluation results are described in Section V. Section VI provides the conclusion.

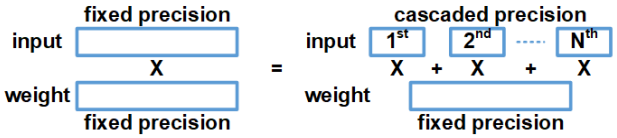


Fig. 1. PC multiplication of the input feature by weight.

II. Proposed Conditional Computing Scheme

A. Precision-Cascading (PC) Scheme

In order to reduce the spatial size of feature maps without losing critical information, pooling layers are typically employed in DCNNs between successive convolutional layers. The most common form of pooling is 2×2 max pooling, where a non-overlapping 2×2 window selects the maximum activations out of four activations and discards the other 75% of activations. Considering that these 75% of activations are results of full convolutions, these convolution results are redundant and are of waste since they are computed and burn power but never used. We aim to reduce this redundancy.

As shown in Fig. 1, we propose to split the input activations into a group of precision values and first perform approximate convolution computations with only the most significant bits (MSBs). Based on this approximate convolution, if we can find the maximum value of convolution outputs with the MSB group of four activations (for 2×2 max pooling), convolution operations of the other LSB groups on non-maximum convolution results can be skipped. As shown in Fig. 2, if the maximum value cannot be found with the MSB group, the approximate convolutions are computed in the remaining precision groups in a cascaded manner. The main advantage of this scheme is that we can reduce the convolution operations by up to

$$\text{Reduction Ratio} = 1 - \frac{p \times p \times \frac{1}{N} + \frac{N-1}{N}}{p \times p} \tag{1}$$

where $p \times p$ is the max-pooling window size, and N is the number of precision groups. The denominator represents the total number of convolution operations. $(p \times p/N)$ in the numerator represents the number of convolution operations in the first group. $(N- 1/N)$ in the numerator represents the number of the convolution operations in the remaining groups, in the case when the maximum values are found in the first group. For example, we can reduce the number of convolution operations by 50% when $p = 2$ and $N = 3$ and by 67% when $p = 3$ and $N = 4$.

However, the latency overhead should be considered, which can occur due to iterations for finding the maximum pooling output. We analyzed the latency overhead and the accuracy degradation of the proposed PC scheme for ImageNet classification with VGG-16 CNN. For different convolution layers of VGG-16, Table I shows the statistics of the percentage of the max-pooling results found in each group for two different precision group schemes. We can reduce more convolution operations when applying larger number of precision groups,

TABLE I

STATISTICS ABOUT THE PERCENTAGE OF THE MAX-POOLING RESULTS FOUND IN EACH PRECISION GROUP

	4 groups: 3b/3b/3b/3b				3 groups: 4b/4b/4b		
	1st group	2nd group	3rd group	4th group	1st group	2nd group	3rd group
Conv 1_2	22.51	71.04	3.46	0.02	85.47	11.82	0.08
Conv 2_2	92.71	5.47	0.08	0.01	97.24	1.51	0.02
Conv 3_3	99.36	0.14	0.06	0.01	99.73	0.02	0.01
Conv 4_3	99.83	0.09	0.00	0.01	99.88	0.10	0.00
Conv 5_3	7.00	92.75	0.25	0.00	99.75	0.25	0.00

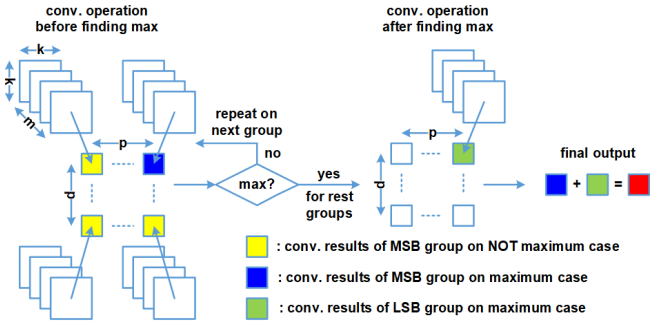


Fig. 2. Conceptual operation of the PC scheme.

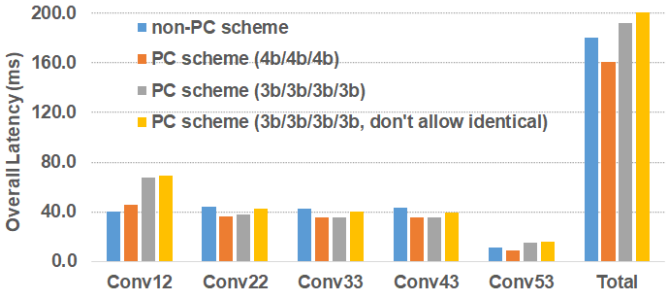


Fig. 3. Overall latency of the PC scheme compared to the non-PC scheme.

TABLE II

ANALYSIS RESULTS OF ACCURACY FOR THE IMAGENET DATA SET

	Baseline (12b)	Precision-cascading	
		3b/3b/3b/3b	4b/4b/4b
Top-1 error (%)	32.7	34.0	33.5
Top-5 error (%)	12.1	13.5	12.5

but the latency overhead increases. Considering a baseline CNN with 12-b precision, Fig. 3 shows the overall latency comparison of non-PC scheme and PC scheme with two group cases (3b/3b/3b/3b and 4b/4b/4b), for VGG-16 convolution layers that precede max-pooling layers.

On the other hand, the proposed PC scheme can slightly degrade the classification accuracy, because to reduce the

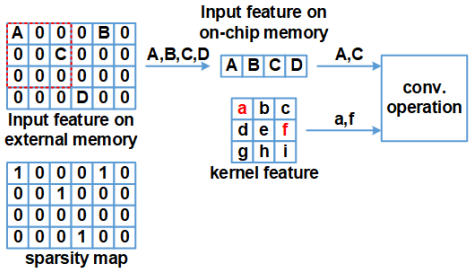


Fig. 4. Conceptual operation of the full zero-skipping scheme.

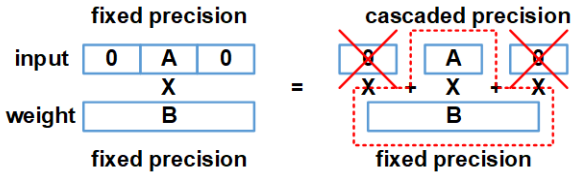


Fig. 5. Illustration of integrating PC and ZS.

number of iterations, we decide the maximum value even if two or three precision-group-wise convolution output values are identical within 2 × 2 pooling window. As shown in Fig. 3, the overall latency increases by 1.08× when we do not allow two or three values to be identical, but the accuracy degradation will not occur in this case. Table II shows the analysis results of the accuracy with ImageNet data set for two PC group schemes. The 4b/4b/4b scheme shows both better latency and accuracy, where the total number of convolution operations can be reduced by ∼2× with only <0.8% degradation in the final ImageNet classification accuracy. It should be noted that the PC scheme can only be applied on the convolution layer right before a max-pooling layer.

B. Full Zero Skipping (ZS) Scheme

Rectified linear unit (ReLU) is a widely used activation function for DCNNs. For a negative input, ReLU returns an output of 0, resulting in many zero activations. Several prior works on DCNN accelerator proposed to skip computation for zero data [8], [10], [13], but they still wasted clock cycles for 0 data. Only a few prior works [21], [22] employed the ZS scheme without redundant MACs and any wasted clock cycles. We employ a full ZS scheme similar to [22], where only non-zero input features are stored to external memory, stored to on-chip memory using a sparsity map, and then computed for convolution operations, as shown in Fig. 4. However, for the ZS-only scheme [22], limitations exist for executing parallel computation on the spatial domain such as 3 × 3 kernel window and across the input feature maps. For example, even if there are eight zero values of nine input feature data in a 3 × 3 kernel window, the eight zero values cannot be fully skipped due to one non-zero value when executing parallel computation on the nine input feature data.

C. Combination of PC Scheme and ZS Scheme

To address the limitation of PC-only and ZS-only schemes, we propose to synergistically integrate PC and ZS, as shown

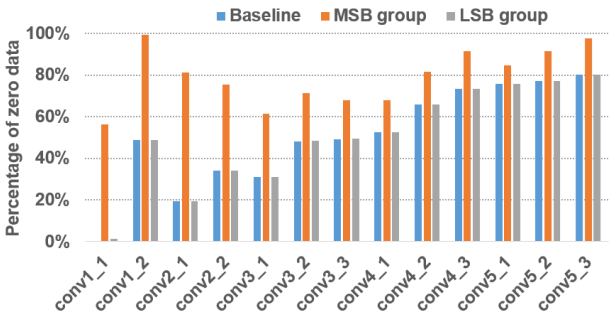


Fig. 6. Analysis results of zero percentage at VGG-16 convolution layers.

```
for (no = 0; no < Nof; no ++)                                   ---> Loop-4
  for (y = 0; y < Yo; y =+S)
    for (x = 0; x < Xo; x =+S)                                  ---> Loop-3
      for (ni = 0; ni < Nif; ni ++)                             ---> Loop-2
        for (ky = 0; ky < K; ky ++)
          for (kx = 0; kx < K; kx ++)                           ---> Loop-1
            pL(no; x, y) += pL-1(ni; x + kx, y + ky) × wtL-1(ni, no; kx, ky);
  pL (no; x, y) = pL(no; x, y) + bias(no);
```

Fig. 7. Four levels of convolution loops, where L denotes the index of convolution layer and S denotes the sliding stride.

in Fig. 5. Compared to the conventional computing scheme, simply dividing the spatial computation into temporal smaller low-precision groups does not provide benefits. However, we exploit the fact that sparsity increases when we divide the activation into a number of low-precision groups. The analysis results of VGG-16 CNN in Fig. 6 show that the MSB group has higher sparsity compared to the baseline non-PC scheme. Therefore, we can largely reduce the computation time of convolution operations on the MSB group of PC scheme, as we jointly employ the ZS scheme. We only use 2 bits of zero sparsity map data for the MSB and LSB groups that consist of the 1st group and 2nd/3rd groups, respectively, which reduces the sparsity map data size by 1.5× with minor degradation of sparsity, because the zero percentage when the value of both the 2nd group and 3rd group is 0 is slightly smaller than the zero percentage of the 2nd group and 3rd group. In addition, we propose a custom architecture to improve the aforementioned limitation of ZS-only scheme, which is described further in Section IV.

III. Convolution Loop Acceleration Strategy

Convolution is the main operation in DCNN algorithms, which involves 3-D multiply and accumulate (MAC) operations of input feature maps and kernel weights. Convolution comprised four levels of loops as shown in the pseudo codes in Fig. 7. To efficiently map and perform the convolution loops, three loop optimization techniques [23]–[25] of loop unrolling, loop tiling, and loop interchange are employed to customize the computation and communication patterns of the accelerator with three levels of memory hierarchy.

Loop unrolling determines the parallelism scheme of certain convolution loops and, thus, the required size of registers and PEs. Loop tiling determines the required capacity of on-chip buffers. It divides the loops into multiple blocks, and the data of the executing block are read from external memory and stored in on-chip buffers. Loop interchange determines the computation order of the four loops and, thus, affects the dataflow between adjacent levels of memory hierarchy. There are two kinds of loop interchange, namely, intra-tiling and inter-tiling loop orders. Intra-tiling loop order determines the pattern of data movements from on-chip buffer to register files or PEs.

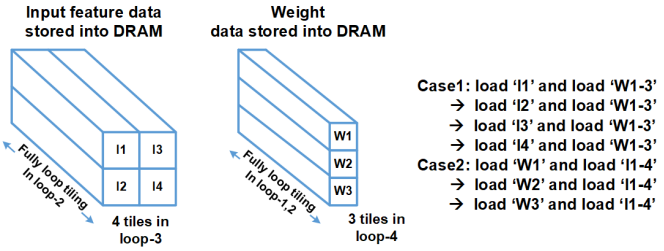


Fig. 8. Illustration of the inter-tiling loop order.

TABLE III
Analysis Results of DRAM Access on Different Cases of Inter-Tiling Strategy

	Case-1	Case-2	Case-3	Case-4	Case-5
DRAM access (MB)	73.49	117.1	67.59	57.64	55.31

To effectively reduce the latency of the proposed PC and ZS scheme, we optimized to unroll loop-1 and loop-3, by the factor of the kernel window size (e.g., $9 = 3 \times 3$), and unrolled loop-4 by the factor of up to 18. In order to minimize the partial sum storage, we fully tiled loop-1 and loop-2 and decided the intra-tiling order to be loop-1 → loop-2 → loop-4 → loop-3.

Fig. 8 illustrates different inter-tiling loop orders. Since inter-tiling loop order determines the data movement from external memory to on-chip buffer, we analyzed five cases of inter-tiling loop order in VGG-16 convolution layers.

1) *Case-1:* All tiles in loop-4 are computed first and the tiles in loop-3 are computed at the end.
2) *Case-2:* All tiles in loop-3 are computed first and the tiles in loop-4 are computed at the end.
3) *Case-3* and *Case-4*: Applying PC with ZS technique on Case-1 and Case-2, respectively.
4) *Case-5:* Applying Case-3 and Case-4 on each layer adaptively.

In Case-1, the number of DRAM accesses per pixel of input features can be 1 since input features are loaded only once from off-chip memory, but the number of DRAM accesses per pixel of weights can be the number of tiles for input features. On the other hand, in Case-2, the number of DRAM accesses per pixel of weights can be 1, but the number of DRAM accesses per pixel of input features can be the number of tiles for weights. Table III shows that the total DRAM accesses can be saved by 2.12× for Case-5, which we selected as the inter-tiling loop order strategy for our chip design.

IV. Energy-Efficient Architecture Based on PC and ZS Schemes

For the integrated PC and ZS scheme, we implemented an efficient computation and data flow, as illustrated in Fig. 9.

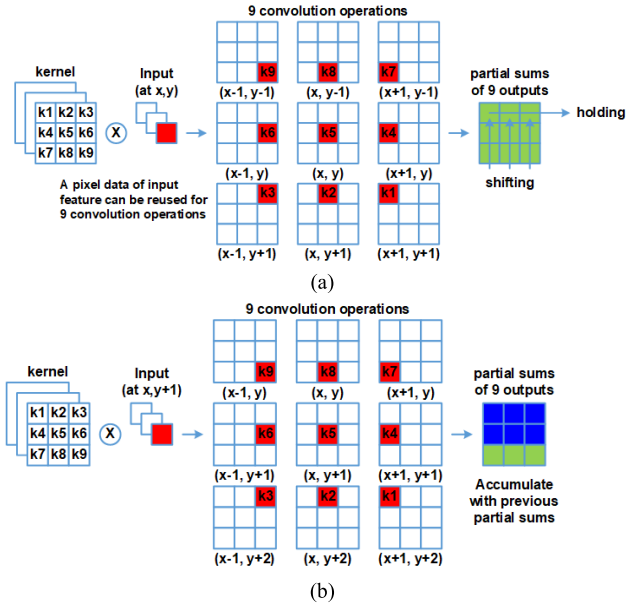


Fig. 9. Proposed computation and data flow for integrated PC/ZS scheme. (a) Example of computation at (x, y). (b) Computation of the next input pixel $(x, y+1)$ is accumulated with previous partial sums (x, y) that are shifted.

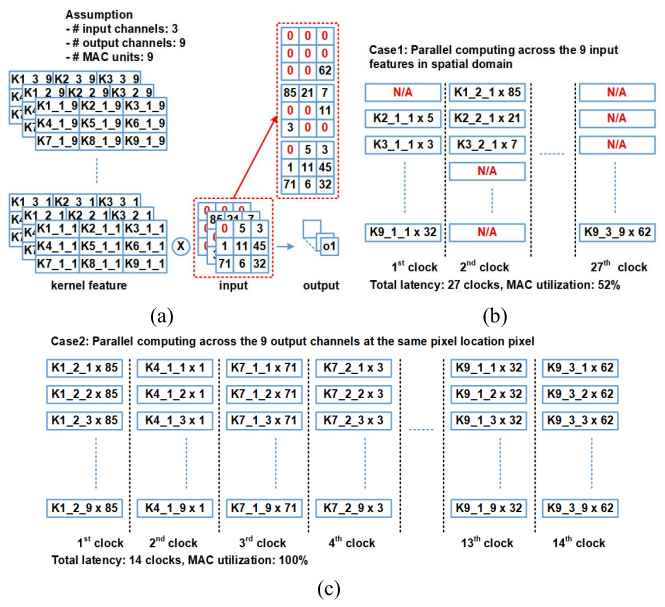


Fig. 10. Illustration of how different parallel computing methods for ZS scheme can result in different latency, using (a) example of MAC operation with three input channels including zero/non-zero data, nine output channels, and nine MAC units. This example demonstrates timing diagrams for two different parallel computing across. (b) Nine pixels of input data and (c) nine output channels.

Input feature data are loaded across the channel and convolution operations are computed with kernel features. We can load only non-zero values of input channel feature at each pixel from external DRAM/on-chip SRAM and transfer them to the PE array. These non-zero input feature data can be re-used to generate nine output features in parallel.

In addition, we hold and shift partial sums in order to maximize the input feature re-use. As shown in Fig. 9, when input features at (x, y) are loaded to the PE array, each PE computes MAC operation for different output features at different pixel locations. Then, when the next input features at $(x, y + 1)$ are loaded to the PE array, each PE computes MAC operation with the previous partial sums that are shifted. Some partial sums are held in on-chip buffers until they can be accumulated with MAC operation in PEs.

Fig. 10 shows how different parallel computing methods can affect the total latency for the ZS scheme, using a simple example of three input channels, nine output channels, and nine MAC units. For the case of parallel computing across nine pixels of input data [see Fig. 10(b)], the MAC operations can be skipped for zero input data. However, this case still wastes clock cycles for zero data, leading to total latency of 27 clocks and a low MAC utilization of 52%. On the other hand, when we compute one pixel of input data across the nine output channels [see Fig. 10(c)], we not only can skip the MAC operations but also can reduce the total latency to 14 clocks compared to Fig. 10(b). Because only non-zero input data are loaded into MAC units and computed, the MAC utilization can be 100%. While non-zero input features are loaded from on-chip memory for activations, we can solely load the kernel features that pertain to non-zero input features from on-chip buffer for weights. Therefore, we not only can fully skip zero values of input features but also can skip kernel features corresponding to the zero input features.

Fig. 11 shows the top-level block diagram and architecture including the on-chip DRAM controller, on-chip SRAMs, and PE arrays. Non-zero values of all input features and the sparsity map data are stored in the external synchronous DRAM (SDRAM) chip [26]. All weights are also stored into SDRAM. Then, input features and weights are loaded from the SDRAMs within the size of tile through SDRAM controller to be stored in the on-chip memory.

Fig. 12 shows the block diagram of ZS control module in the custom SDRAM controller that we designed. To process store operations, sparsity map (2 bits) and non-zero value of output feature (12 bits) from ReLU/pooling module are transferred into ZS control module. Since we use SDRAM with 16-bit width, there are buffers to hold 16 bits of each group of non-zero output feature data (4 bits) and sparsity map (1 bit). Then, 16 bits of output feature and sparsity map can be transferred to external SDRAM. The write addresses for output feature and sparsity map data are generated by the equations in the write address generator block based on the total activation size and the current pixel location. This way, we decode the coordinate information of the input feature data loaded from the SDRAM. To process load operations, the sparsity map data (16 bits) are loaded from the SDRAM and 64 bits of the sparsity map data are written to on-chip SRAM. The input feature data can be loaded from the SDRAM when the counter number of "1" in the sparsity map data is over 4. Then, 16 bits of input feature data are stored into on-chip SRAM. Read addresses for both sparsity map and input feature data increase by 1 for every read cycle. Since the clock frequencies for our proposed system and the external SDRAM are different, we use an FIFO that synchronizes the data with two different clocks.

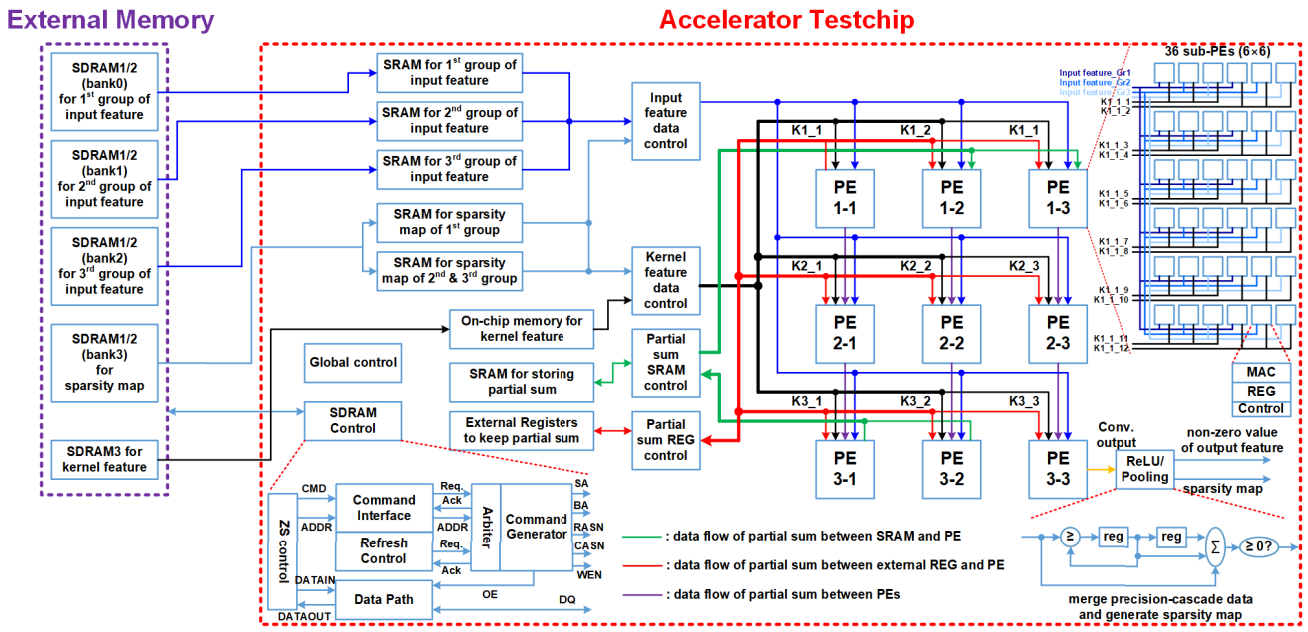


Fig. 11. Top-level block diagram and data flow of the proposed DCNN accelerator.

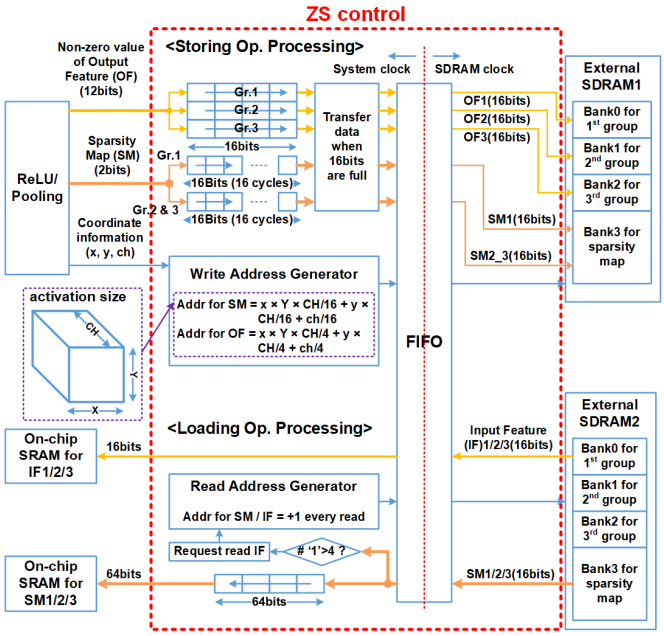


Fig. 12. Block diagram of ZS control in the custom SDRAM controller.

After finishing the storage, PE array starts to compute MAC operations with input features and weights that are loaded from SRAM by input and kernel feature data controller. Our proposed accelerator (see Fig. 11) employs nine main PEs to efficiently compute 3 × 3 convolution operation, and each main PE includes 36 sub-PEs to compute up to 18 output channels in parallel. A sub-PE consists of MAC unit, register, and control unit. The register holds partial sums and the control unit shifts partial sums to adjacent PE, external register, or SRAM. Communications of partial sums among PEs, registers, and SRAM are governed by the partial sum controller.

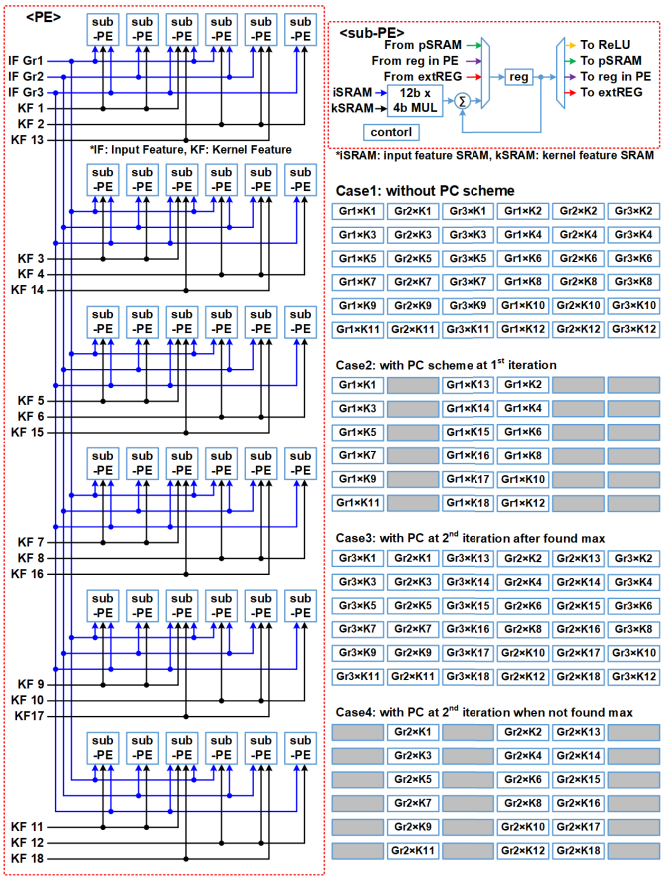


Fig. 13. Detailed block diagram of a PE module.

To achieve 100% MAC utilization for convolution layer with a 3 × 3 kernel window, PE array consists of 9 (3 × 3) PEs. Fig. 13 shows the detailed block diagram of a PE module. Each PE has 36 (6 × 6) sub-PEs to support

three precision-cascaded groups and compute up to 18 output channels in parallel. Group 1/2/3 of input feature (4 bits) from on-chip SRAM is transferred to each sub-PE. Weights (12 bits each) of up to 18 output channels data from SRAM are also transferred into the sub-PE. The 36 sub-PEs compute 12 output feature data in parallel when not applying the PC scheme. With PC sub-PEs compute 18 output features of group1 in parallel, while other 18 sub-PEs are not working to reduce the power consumption. After finding max-pooling values, 36 sub-PEs compute 18 output features of the rest groups in parallel. Although the utilization of sub-PEs can be 50% when maximum values are not found, the total utilization is over 90% since most computation in the group1 of input feature are skipped by ZS scheme and most max-pooling values are revealed at the first iteration as we analyzed in Table I and Fig. 6. Table IV shows the analysis results of the MAC utilization at VGG-16 convolution layers.

After some initial latency, the ninth PE (PE3-3 in Fig. 11) generates the final output features every cycle. The final output features go through ReLU/pooling module and subsequently stored into external memory by the SDRAM controller. All operations are fully pipelined. Fig. 14 shows the timing diagram for processing a convolution with PC and ZS schemes. In the first cycle, input feature $i(x-1, y-1)$ is transferred to all PEs. Each PE produces partial sums with input feature and the corresponding kernel feature. Then, the partial sums go through other PEs, partial sum SRAM, or external register. Since $p(x, y)$ from the first cycle can be added to the partial sums in PE2_2 in the second cycle, the partial sum data are transferred into PE2_2. In a similar manner, PE3_3 produces the final output feature data at (x, y) at the fourth cycle and the output feature data are transferred to ReLU/pooling module. The partial sums that are not consumed right away are stored in the partial sum SRAM. Since the partial sums of PE3_1 and PE3_2 at the fourth cycle can be used after several cycles, the partial sums are transferred to an external register to be held for a short time.

It should be noted that our proposed scheme is best suited for DCNNs with many max-pooling layer such as VGG-16 [20], FlowNet [28], and SegNet [31]. The proposed accelerator can be applied to any convolution layers with 3×3, 5×5, and 7×7 kernel windows, but the proposed scheme cannot support other shapes of kernel window such as 1×1. In addition, we should note that our proposed accelerator has limitations to the networks that only have a few max-pooling layers such as ResNet [32] or MobileNet [33].

V. Measurement Results

The proposed DCNN accelerator was implemented in a 40-nm CMOS technology, and the chip micrograph is shown in Fig. 15(a). The total area is 3×3 mm^2, including 324 MAC units, 339.5 kB of on-chip SRAMs, and a custom SDRAM controller. The chip specifications are summarized in Fig. 15(b).

Fig. 16 shows the prototype chip testing board and system, which includes the custom PCB mounting the 40-nm prototype chip, 256-Mb SDRAMs, and PCIe connectors to communicate with National Instrument LabView testing system. Our prototype chip performs DCNN inference, where it takes an input image data and outputs the final output feature data of convolution layer. All the processing for loading/storing data from/to external SDRAMs are done in the prototype chip. LabView testing system can configure the chip and read output feature data through the PCIe interface to evaluate the accuracy.

TABLE IV
Analysis Results of MAC Utilization at VGG-16 Convolution Layers

Conv	1_1	1_2	2_1	2_2	3_1	3_2	3_3	4_1	4_2	4_3	5_1	5_2	5_3
MAC util. (%)	100	95	100	88.8	100	100	87.7	100	100	96	100	100	98.9

We implemented a custom SDRAM controller on-chip to demonstrate energy-efficient data movement in real time between external SDRAM and our proposed accelerator. Aided by our full ZS scheme that only stores non-zero activations, we achieved 5.8×, 1.57×, and 1.44× reduction of DRAM access in VGG-16 convolution layers compared to [13], [15], and [16], respectively, as shown in Fig. 17. For all convolution layers of VGG-16, the total DRAM accesses for our accelerator are 31.3 and 24 MB for activation and weight, respectively, or 0.0018 access/MAC in average. Note that we have measured 13 convolution layers of VGG-16 excluding the fully connected layers.

The measured chip performance (frames/s) and total/leakage power consumption with dynamic voltage scaling are shown in Fig. 18. The proposed accelerator chip was fully functional down to 0.6 V where the chip demonstrated 1.9 frames/s with 28.2-mW power for VGG-16 CNN. Fig. 19 shows the area and measured power breakdown of the prototype chip. Twenty-seven percentage of the total chip area is occupied by on-chip SRAM arrays, which store the kernel weights, input/output feature maps, and partial sums. On the other hand, 10% of the total chip power was consumed by the SRAM arrays since the proposed PC with fully ZS scheme significantly reduces the power consumption of memory access. The partial sum controller consumes ∼30% of the total area and power, because it includes many registers to keep the partial sums.

Table V provides a detailed comparison with recent works on DCNN inference accelerator. Among the prior works, Lee *et al*. [15] achieved the highest throughput of 18.3 frames/s with the energy efficiency of 4.71 TOPS/W for VGG-16, but the accelerator in [15] did not include any evaluation of external memory and off-chip communication energy. We achieved a peak energy efficiency of 24.97 TOPS/W in convolution layers 5–3 and 3.49 TOPS/W in average for VGG-16 convolution layers, including off-chip memory access. Note that we achieved 6.17 TOPS/W in average for VGG-16 convolution layers that are followed directly by max-pooling layers, where we jointly applied the PC and ZS schemes. Aided by the benefits of the proposed PC and ZS scheme, we achieved 8.25 frames/s throughput

TABLE V
COMPARISON WITH PRIOR ASIC ACCELERATORS FOR DCNN

	JSSC'18 [14]	VLSI'18 [30]	JSSC'19 [9]	JSSC'19 [15]	TVLSI'20 [16]	**This work**
Technology	65nm	65nm	40nm	65nm	65nm	40nm
Core area (mm^2)	3.2 × 3.3	2.6 × 3	14.3 × 8.5	4 × 4	4 × 4	3 × 3
Supply voltage (V)	1.2	1.0	1.1	0.63-1.1	0.4/0.7	0.6-0.9
Operating frequency (MHz)	250	200	300	200	60/120	180-400
Precision (bit)	16	8	4	16	10	12
On-chip SRAM (KB)	139.6	170	1632	256	848	339.5
Power (mW)	88.6-97.3	248	2083	6.4-297	52	40.8-182.7
DRAM access (MB)	1.9 @ AlexNet	N/A	N/A	87.02 @ VGG-16	1.86 @ AlexNet 45 @ VGG-16	31.3 @ VGG-16
Throughput (fps)	17.7 @ AlexNet	N/A	N/A	0.97-18.3 @ VGG-16	26.3 @ AlexNet 2.2 @ VGG-16	3.71-8.25 @ VGG-16
Throughput (GOPS)	23.35 @ AlexNet	257.9 @ AlexNet	1825 @ AlexNet	30.1-567.3 @ VGG-16	43.16 @ AlexNet 59.8 @ VGG-16	142.39-316.1 [1] / 251.7-559.1 [2] / 1018.8-2263.7 [3] @ VGG-16
GOPS/mm^2	2.21 @ AlexNet	33.1 @ AlexNet	15.01 @ AlexNet	1.88-35.45 @ VGG-16	26.3 @ AlexNet 2.2 @ VGG-16	15.82-35.12 [1] / 27.97-62.12 [2] / 113.2-251.5 [3] @ VGG-16
TOPS/W	0.25 @ AlexNet	1.04 @ AlexNet	0.88 @ AlexNet	4.71-1.91 @ VGG-16	0.83 @ AlexNet 1.15 @ VGG-16	3.49-1.73 [1] / 6.17-3.06 [2] / 24.97-12.39 [3] @ VGG-16

[1] Average, [2] Average for convolution layers followed by max-pooling, [3] Peak
* VGG-16 includes only convolution layers
** Prior works on [14, 16] included power consumption and energy of off-chip memory communication
*** This work included power consumption and energy of the custom SDRAM controller as well as off-chip memory communication

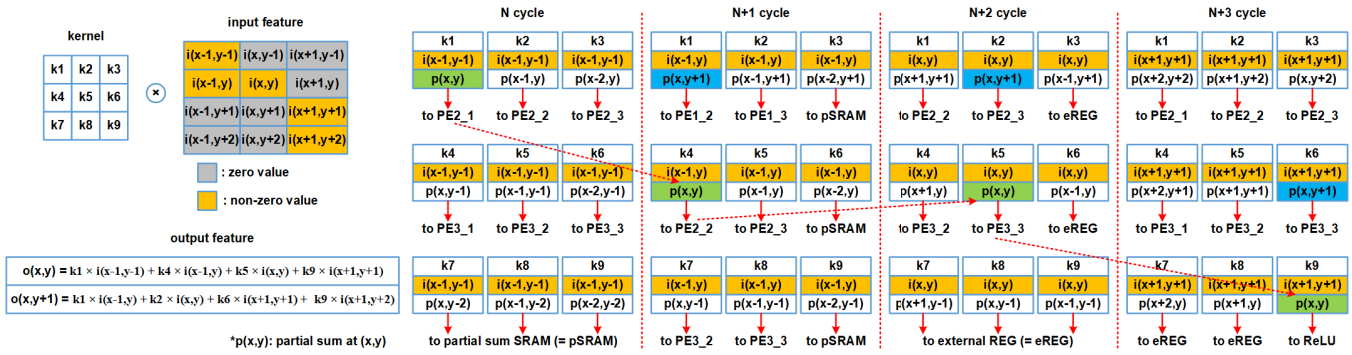
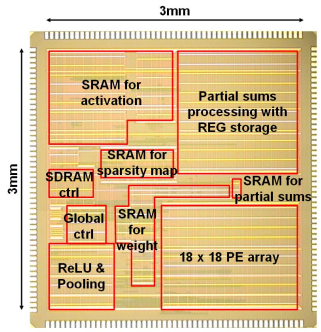


Fig. 14. Example of timing diagram for processing a convolution with PC and ZS schemes.



Technology	**40nm CMOS**
Chip size	**9 mm^2**
# MAC units	**324**
On-chip SRAM	**339.5 KB**
Frequency	**320 MHz @ 0.9 V**
Power	**151.2 mW @ 0.9 V**

(a) (b)

Fig. 15. (a) Prototype chip micrograph. (b) Chip specifications.

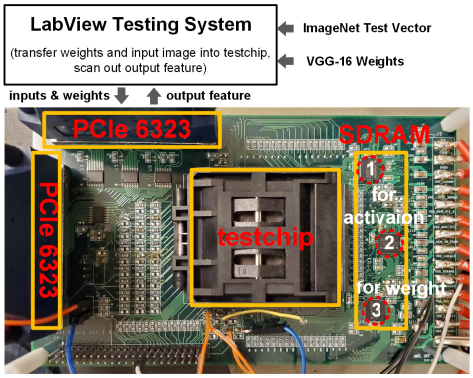


Fig. 16. Chip testing board and system.

at 1.2 V, which is 3.75× higher compared to [16] which evaluated off-chip memory access.

We compared the energy efficiency for the baseline scheme and the proposed conditional computing scheme with PC and ZS, for VGG-16 CNN with ImageNet data set. As shown in Fig. 20, when only ZS scheme is applied in the convolution layers that are not followed by max pooling, we achieve 2.1× higher energy efficiency in average. If only PC scheme is applied in the convolution layers that are followed by max pooling, the energy efficiency improvement is limited

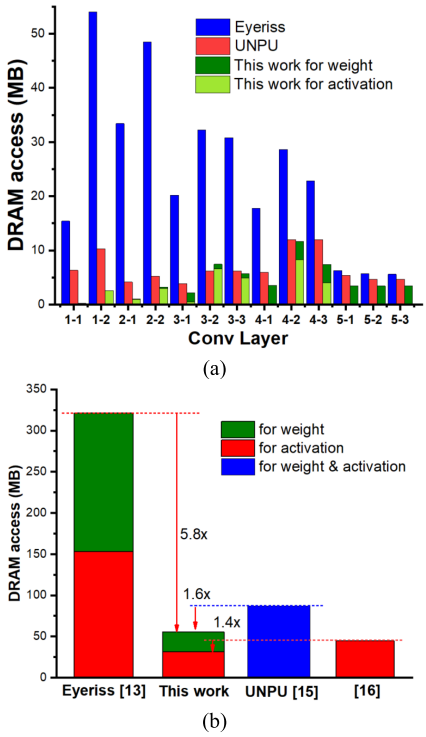


Fig. 17. DRAM access comparison to (a) [13] and [15] at each convolution layer and (b) [13] and [15], [16] in the total convolution layers for VGG-16.

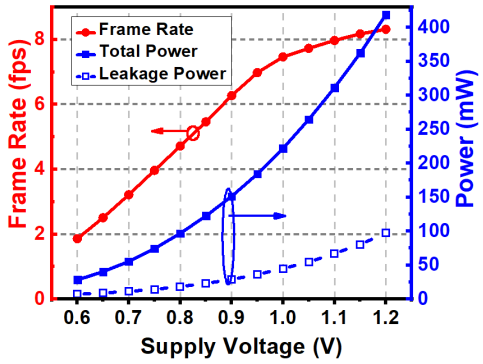


Fig. 18. Measured frame rate and total/leakage power with voltage scaling.

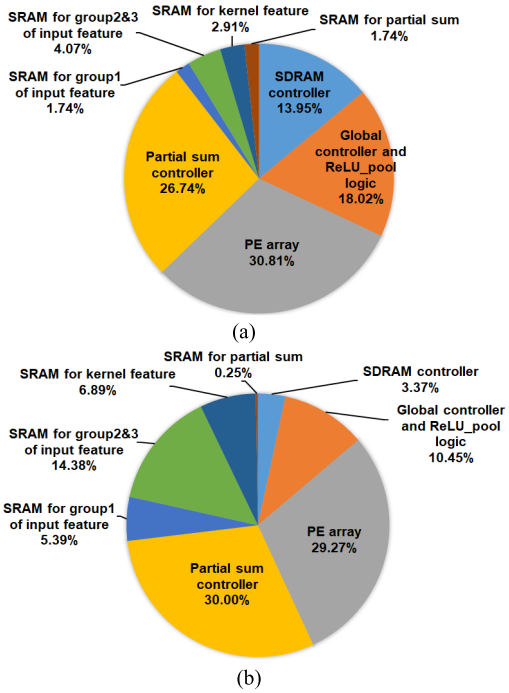


Fig. 19. (a) Power and (b) area breakdown of the DCNN accelerator chip.

TABLE VI
PERFORMANCE BREAKDOWN OF THE FIVE CONVOLUTION LAYERS IN FLOWNET

Layer	Latency (ms)	GOPS	TOPS/W	MAC utilization	Zero sparsity	DRAM access (MB/s)
Conv3_1	12.98	279.2	1.54	100%	76.4%	95.0
Conv4	2.1	3448	28.51	99.1%	95.4%	910.8
Conv4_1	8.41	431	2.38	100%	85.1%	420.6
Conv5	1.48	2443	20.19	98.6%	95.3%	2352.9
Conv5_1	2.12	427.5	2.36	100%	85.0%	1612.0

to 1.7× in average, due to the latency overhead can occur due to iterations for finding the max-pooling output (see Fig. 3). When we synergistically integrate PC and ZS schemes, we achieve significantly higher improvement in energy efficiency (7.7× in average) for convolution layers followed by max pooling.

FlowNet [28], [29] is well known for optical flow estimation and has six out of nine convolution layers followed by max pooling. We have benchmarked the proposed accelerator with FlowNet convolution layers which include max-pooling layers for Flying Chair data set [28]. Table VI shows the performance breakdown of the five convolution layers in FlowNet benchmark. As shown in Fig. 21, we achieve not only 70.7× higher energy efficiency in average for convolution layers followed by max pooling but also significantly higher improvement in energy efficiency (11.3× in average) when only ZS scheme is applied in all five convolution layers because the five convolution layers of FlowNet for Flying Chair data set have more zero sparsity compared to VGG-16 network for ImageNet data set. We achieve 13× higher energy efficiency in average for five convolution layers in FlowNet compared to the baseline non-PC/ZS schemes.

Note that ImageNet data set has a relatively small input image size of 224 × 224, which limits the number of max-pooling layers that can be employed throughout the DCNN, since each max-pooling layer reduces both the width and height of the feature map by 2×. This also limits the overall network level energy improvement (4.4×) of our proposed scheme. However, for real-world applications such as autonomous driving and medical imaging that have high-resolution input images (e.g., full HD or higher), CNNs could have many convolution layers followed by max-pooling layers. In [27], a new DCNN algorithm was presented for tuberculosis screening from high-resolution medical images, where four out of six convolution layers were followed by

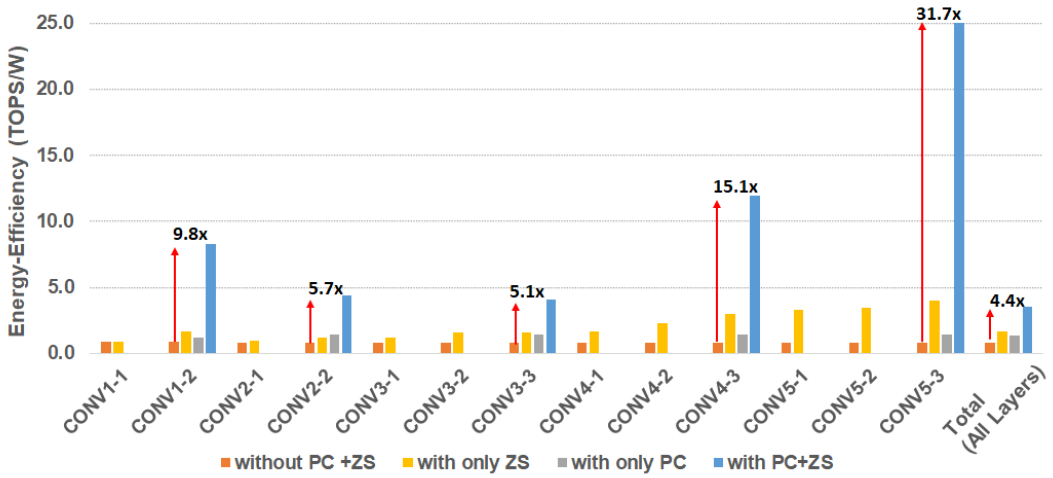


Fig. 20. Energy efficiency comparison between schemes without PC + ZS, with only ZS, with only PC, and with PC + ZS for VGG-16 with ImageNet data set.

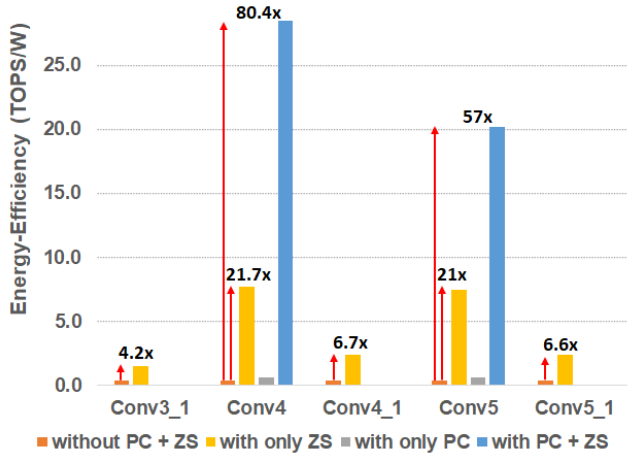


Fig. 21. Energy efficiency comparison between schemes without PC + ZS, with only ZS, with only PC, and with PC + ZS for FlowNet [28] with Flying Chair data set.

max-pooling layers. SegNet [31] is a popular DCNN model used for semantic pixel-wise segmentation. The encoder network in SegNet is topologically identical to the 13 convolutional layers in the VGG-16 that has 5 out of 13 convolution layers followed by max pooling.

For such DCNNs, the proposed PC and ZS schemes are anticipated to enable much higher improvements in system-level energy efficiency.

VI. Conclusion

In this article, we presented an energy-efficient DCNN inference accelerator in 40-nm CMOS. We proposed PC scheme to reduce the redundant convolutional operations when max-pooling operations are combined. In addition, by integrating the PC with ZS, we achieved significant reduction of energy and external memory accesses. For VGG-16 CNN, our accelerator achieved peak/average energy efficiency of 24.97/12.39 TOPS/W while consuming 40.8 mW at 0.6 V and low external memory access of 0.0018 access/MAC in average. In addition, we achieved the peak/average energy efficiency of 28.51/4.64 TOPS/W at 0.9-V supply with FlowNet for Flying Chair data set.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[2] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The microsoft 2017 conversational speech recognition system," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 5934–5938.

[3] A. Y. Hannun *et al.*, "Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network," *Nature Med.*, vol. 25, no. 1, pp. 65–69, Jan. 2019.

[4] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "DeepEye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proc. 15th Annu. Int. Conf. Mobile Syst., Appl., Services*, Jun. 2017, pp. 68–81.

[5] N. D. Lane *et al.*, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2016, pp. 1–12.

[6] S. Alyamkin *et al.*, "Low-power computer vision: Status, challenges, and opportunities," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 411–421, Jun. 2019.

[7] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[8] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Envision: A 0.26-to-10 TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 246–247.

[9] K. Ueyoshi *et al.*, "QUEST: Multi-purpose log-quantized DNN inference engine stacked on 96-MB 3-D SRAM using inductive coupling technology in 40-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 186–196, Jan. 2019.

[10] J. Song *et al.*, "7.1 An 11.5TOPS/W 1024-MAC butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile SoC," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 130–132.

[11] J. Yue *et al.*, "7.5 A 65nm 0.39-to-140.3TOPS/W 1-to-12b unified neural network processor using block-circulant-enabled transpose-domain acceleration with 8.1× higher TOPS/mm^2 and 6T HBST-TRAM-based 2D data-reuse architecture," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 138–140.

[12] C. Chen, X. Liu, H. Peng, H. Ding, and C.-J. Richard Shi, "IFPNA: A flexible and efficient deep learning processor in 28-nm CMOS using a domain-specific instruction set and reconfigurable fabric," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 346–357, Jun. 2019.

[13] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.
[14] J. Jo, S. Cha, D. Rho, and I.-C. Park, "DSIP: A scalable inference accelerator for convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 53, no. 2, pp. 605–618, Feb. 2018.
[15] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, Jan. 2019.
[16] J. Sim, S. Lee, and L.-S. Kim, "An energy-efficient deep convolutional neural network inference processor with enhanced output stationary dataflow in 65-nm CMOS," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 87–100, Jan. 2020.
[17] M. Kim and J.-S. Seo, "Deep convolutional neural network accelerator featuring conditional computing and low external memory access," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Mar. 2020, pp. 1–4.
[18] H.-S. Wu, Z. Zhang, and M. Papaefthymiou, "A 0.23 mW heterogeneous deep-learning processor supporting dynamic execution of conditional neural networks," in *Proc. IEEE 44th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sep. 2018, pp. 162–165.
[19] H. Kaul *et al.*, "14.4 A 21.5M-query-vectors/s 3.37nJ/vector reconfigurable k-nearest-neighbor accelerator with adaptive precision in 14 nm tri-gate CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Jan. 2016, pp. 260–261.
[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations (ICLR)*, May 2015, pp. 1–14.
[21] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 243–254, Jun. 2016.
[22] A. Aimar *et al.*, "NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 3, pp. 644–656, Mar. 2019.
[23] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, Feb. 2015, pp. 161–170.
[24] D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler transformations for high-performance computing," *ACM Comput. Surveys*, vol. 26, no. 4, pp. 345–420, Dec. 1994.
[25] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, Feb. 2017, pp. 45–54.
[26] Integrated Silicon Solution Inc. *Mobile SDRAM Chip, 256Mbit 16Mx16*. Accessed: Oct. 12, 2020. [Online]. Available: http://www.issi.com/WW/pdf/42-45SM-RM-VM16160K.pdf
[27] S. Hwang, H.-E. Kim, J. Jeong, and H.-J. Kim, "A novel approach for tuberculosis screening based on deep convolutional neural networks," *Proc. SPIE, Med. Imag., Comput.-Aided Diagnosis*, vol. 9785, no. 97852W, pp. 1–9, Mar. 2016.
[28] A. Dosovitskiy *et al.*, "FlowNet: Learning optical flow with convolutional networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 2758–2766.
[29] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "FlowNet 2.0: Evolution of optical flow estimation with deep networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2462–2470.
[30] Z. Yuan *et al.*, "Sticker: A 0.41-62.1 TOPS/W 8Bit neural network processor with multi-sparsity compatible convolution arrays and online tuning acceleration for fully connected layers," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 33–34.
[31] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017.
[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[34] A. S. Rakin, Z. He, and D. Fan, "TBT: Targeted neural network attack with bit trojan," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 13198–13207.
[35] T.-J. Yang and V. Sze, "Design considerations for efficient deep neural networks on Processing-in-Memory accelerators," in *IEDM Tech. Dig.*, Dec. 2019, pp. 1–22.

Minkyu Kim (Member, IEEE) received the B.E. degree in electronics and electrical engineering from Pusan National University, Busan, South Korea, in 2005, the M.S. degree in electrical engineering from the Pohang University of Science and Technology, Pohang, South Korea, in 2007, and the Ph.D. degree in electrical engineering from Arizona State University, Tempe, AZ, USA, in 2019.

From 2007 to 2013, he was with LG Display Company Ltd., Paju-si, South Korea, where he worked on the development of timing controller chip and image processing for high-quality display. He was a Summer Intern with Qualcomm Inc., San Diego, CA, USA, in 2017 and 2018. In 2019, he joined Qualcomm, San Diego, CA, USA, where he worked on neural network processor. His research interests include efficient hardware design and application of machine learning and neuromorphic algorithms.

Dr. Kim was a recipient of the LG Display Scholarship from 2005 to 2007 and the Qualcomm Roberto Padovani Scholarship Award in 2017.

Jae-sun Seo (Senior Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2001 and the M.S. and Ph.D. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2006 and 2010, respectively.

From 2010 to 2013, he was with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA, where he worked on cognitive computing chips under the DARPA SyNAPSE Project and energy-efficient integrated circuits for high-performance processors. In 2014, he joined the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA, where he is currently an Associate Professor. In 2015, he joined the Intel Circuits Research Lab, Hillsboro, OR, USA, as a Visiting Faculty. His current research interests include efficient hardware design of machine learning and neuromorphic algorithms and integrated power management.

Dr. Seo was a recipient of the Samsung Scholarship during 2004–2009, the IBM Outstanding Technical Achievement Award in 2012, and the NSF CAREER Award in 2017.