FPGA-Based Accelerator for Rank-Enhanced and Highly-Pruned Block-Circulant Neural Networks

Haena Song[1], Jongho Yoon[1], Dohun Kim[1], Eunji Kwon[1], Tae-Hyun Oh[1,2,†], and Seokhyeong Kang[1,2*]
[1]Department of EE and [2]Graduate School of AI, POSTECH, Pohang, South Korea
*shkang@postech.ac.kr

***Abstract*—Numerous network compression methods have been proposed to deploy deep neural networks in a resource-constrained embedded system. Among them, block-circulant matrix (BCM) compression is one of the promising hardware-friendly methods for both acceleration and compression. However, it has several limitations; *(i)* limited representation due to the structural characteristic of circulant matrix, *(ii)* limitation of the compression parameter, *(iii)* need to specialize the dataflow for BCM-compressed network accelerators. In this paper, rank-enhanced and highly-pruned block-circulant matrices compression (RP-BCM) framework is proposed to overcome these limitations. RP-BCM comprises two stages: Hadamard-BCM and BCM-wise pruning. Moreover, a dedicated skip scheme is introduced to processing element design for exploiting high-parallelism with BCM-wise sparsity. Furthermore, we propose specialized dataflow for a BCM-compressed network on a resource-constrained FPGA. As a result, the proposed method achieves parameter reduction and FLOPs reduction for ResNet-50 in ImageNet by 92.4% and 77.3%, respectively. Moreover, the proposed hardware design achieves 3.1× improvement in energy efficiency on the *Xilinx PYNQ-Z2* FPGA board for ResNet-18 on ImageNet compared to the GPU.**

***Index Terms*—Network Compression, Structured Pruning, CNN Accelerator, FPGA, Convolution Neural Networks**

I. Introduction

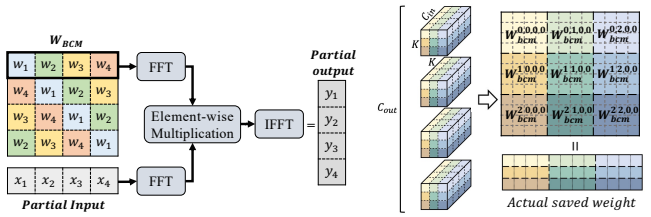
Deep neural networks (DNNs) have exhibited great performance improvement as network size increases, stacking layers deeper to extract complex features [1]. However, the enormous parameter size and computational overhead make DNNs difficult to deploy on edge devices that must operate with limited resources and low power consumption. Therefore, numerous methods have been proposed to compress and accelerate DNNs [2]. Unstructured pruning was introduced as one of the earliest methods [3]. Unstructured pruning measures the importance of weights in an individual element level, and then removes less important weights. Despite the advantage of high compression, it is difficult to accelerate on hardware, primarily because the network has an irregular sparsity, which renders it unable to maintain high-parallelism on hardware.

To resolve the problem of irregular computation patterns, network compression imposing regular structure has been introduced. As one of them, block-circulant matrix (BCM) compression divides the weight tensor into sub-blocks, where each block is represented in the form of a circulant matrix [4]. A circulant matrix has the same elements in all row vectors; but the elements in each row vector are rotated (Fig. 1a). Consequently, the memory size complexity is reduced from $O(n^2)$ to $O(n)$, since the full weight tensor can be represented with only one row vector per BCM. BCM compression can maintain high-parallelism of the computation due to the regular computation pattern of sub-circulant matrices. Moreover, the computational complexity can be reduced from $O(n^2)$ to $O(n \log n)$, since the matrix multiplication of the circulant matrix can be replaced by "FFT−Elementwise MAC (eMAC)−IFFT."

Despite these strengths, BCM compression has some critical limitations that degrade performance. First, when training the weights of network to form BCMs, most weights have limited representation, because each row of a BCM must contain the same elements of the first row. Due to this constraint, the representation of the network may be limited, resulting in a significant accuracy degradation. Second, the trade-off between compression ratio and accuracy drop cannot be flexibly determined, since the BCM size should be 2^n for the FFT calculation. Second, the trade-off between compression ratio and accuracy drop cannot be flexibly determined, since the BCM size should be 2^n for the FFT calculation. Furthermore, when the BCM size is larger, there is a significant accuracy drop because the BCM compression compresses the network without considering the importance of the weights. For this reason, it is necessary to overcome the limitations while maintaining the advantages of BCM compression.

This work was supported by LIG Nex1 Co., Ltd. and Institute of Information communications Technology Planning Evaluation (IITP) grant funded by the Korea government (MSIT).(No.2022-0-01172, DRAM PIM Design Base Technology Development).

[†]T.-H. Oh is adjunct with Yonsei University, Seoul, South Korea.



(a) (b)

Fig. 1: (a) Computation sequence of circulant matrix. (b) Illustration of BCM-compressed convolution layer. The elements of the same color form each BCM.

In this paper, we propose a *rank-enhanced and highly-pruned block-circulant matrices compression* (RP-BCM) framework. Our proposed framework compresses the network in two stages. In the first stage, we apply the Hadamard product of BCM to mitigate the accuracy degradation in the rank-perspective. In the second stage, we obtain the importance of the weights in the BCM units and further compress with BCM-wise pruning. Moreover, a dedicated hardware design for RP-BCM is proposed along with specialized dataflow for a BCM-compressed network on a resource-constrained FPGA. The main contributions of this paper are as follows:

- We introduce *Hadamard*-BCM (*hada*BCM) to overcome the lack of representation of traditional BCM compression in rank-perspective. Our method can be utilized without any overhead on the inference accelerator.
- We propose the BCM-wise pruning to yield more effective compression ways. Moreover, we show that BCM-wise pruning achieves significant compression ratio compared to other compression methods. In particular, to the best of our knowledge, this is the first time we have achieved a parameter reduction of more than 90% within similar accuracy drop as we have reported for ResNet-50 on ImageNet.
- We propose the dedicated dataflow for RP-BCM on FPGAs. We also propose the processing element design to exploit BCM-wise sparsity with high-parallelism.

The rest of this paper is organized as follows: Section II summarizes the background for BCM compression and our research motivations. Section III and IV explain the RP-BCM framework and hardware implementation, respectively. Section V reports the experimental result. Finally, the conclusion is presented in Section VI.

978-3-9819263-7-8/DATE23/© 2023 EDAA

II. Background and Motivation

A. Block-Circulant Matrices (BCM) Compression

Matrix-multiplication of the circulant matrix and "FFT-*e*MAC-IFFT" of first row vector of the circulant matrix have the same result (Fig. 1a). It can reduce the computational complexity from $O(n^2)$ to $O(n \log n)$. Also, the memory storage complexity is reduced from $O(n^2)$ to $O(n)$, since only one row vector of the circulant matrix is needed. After BCM compression, the network can be accelerated more efficiently than other compression methods, primarily because it can maintain high-parallelism on the hardware with the regular structure, BCM. Because of these advantages, BCM compression is used for various networks [4]–[8].

We briefly introduce the BCM-compressed convolution layer which is the focus of this paper. Let $\mathrm{W} \in \mathbb{R}^{K \times K \times C_{in} \times C_{out}}$ denote the weight tensor of a convolution layer, where K, C_{in}, and C_{out} represent the weight kernel size, input channel size, and output channel size, respectively. In BCM compression, W forms multiple BCMs toward the input and output channel directions. In other words, W is partitioned into the set of BCMs, $\{\mathrm{W}_{bcm}^{0,0,0,0}, \mathrm{W}_{bcm}^{0,0,0,1}, \ldots, \mathrm{W}_{bcm}^{k_h,k_w,\frac{c_{in}}{BS},\frac{c_{out}}{BS}}, \ldots, \mathrm{W}_{bcm}^{K\text{-}1,K\text{-}1,\frac{C_{in}-1}{BS},\frac{C_{out}-1}{BS}}\} \in \mathbb{R}^{BS \times BS}$, where BS indicates the size of the BCM. Fig. 1b illustrates the example of BCM-compressed convolution weights with K=3, C_{in}=C_{out}=4, and BS=4. The 'FFT$-e$MAC' between each BCM and the partial input vector is executed. Then, the fully accumulated complex-valued output is recovered to the real-valued output via the IFFT. Note that for better understanding, we notate the process of BCM compression with the entire BCM. However, in practice, only the first row vector per W_{bcm} is used in the training and inference phases. This is because the substituted computation (Fig. 1a) is applied in both backward and forward propagation [4].

B. Motivation

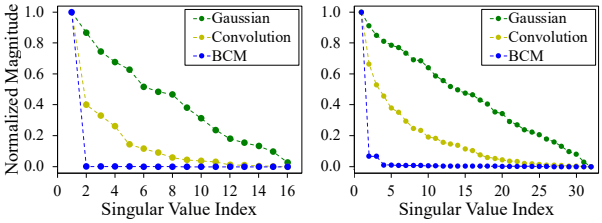


Fig. 2: Visualization of the singular values decaying of convolution layer in VGG-16 network trained on Cifar-10. The left and right graphs show singular values of 16×16 matrix and 32×32 matrix, respectively.

1) Limited Representation from Rank-Perspective: In neural networks, the matrix rank contains the information of the features [9]–[11]. Thus, the matrix rank measures the strict upper bound on the degree of expressiveness of a weight matrix [10]. However, due to the structural constraint of the BCM, numerous BCM-compressed weights eventually may have limited capability of representation. If the effective rank tends to be restricted by this constraint, each layer of the network lacks feature representation power, resulting in performance degradation. Fig. 2 visualizes this problem by showing singular value decaying graphs of different weight types. Rank-condition of a matrix can be checked by observing the decay of the singular values [12]. The singular values of a matrix close to full rank have a linear decay (e.g., Gaussian random matrix [13]). In contrast, a matrix with poor rank-condition tends to have an exponential decay. The singular values of the BCM exhibit extreme exponential decay compared to those of the Gaussian and the original convolution. This means that the rank-condition of each BCM seems to be unfavorable for sufficient feature extraction. In such a scenario, we define that the BCM is in a poor rank-condition, particularly when it has more than 50% singular values whose magnitude is less than 5% of the largest value, which can be regarded as a simple special case of the effective rank measure [14]. In VGG-16 on Cifar-10, more than 70% of BCMs are observed in poor rank-conditions across entire layers in all BCM sizes of 8, 16, and 32. This ratio is notable compared to the original convolution, where only 2% of the matrix units are in poor rank-conditions. This observation indicates that the limited representation of traditional BCM compression needs to be alleviate.

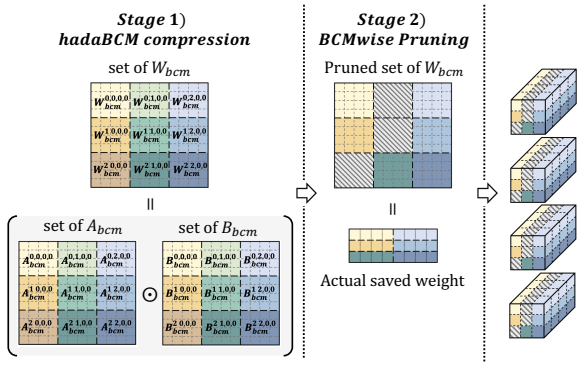
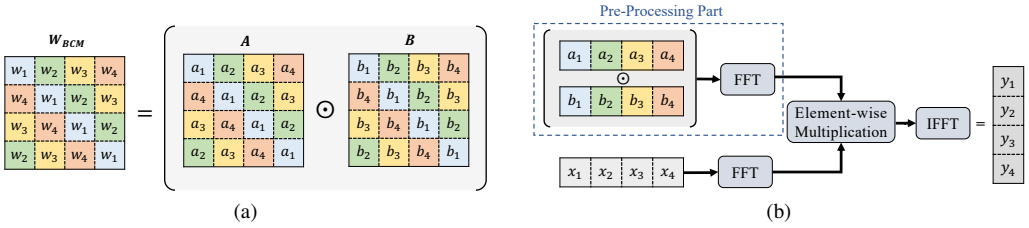


Fig. 3: Proposed two-step BCM compression framework, RP-BCM.

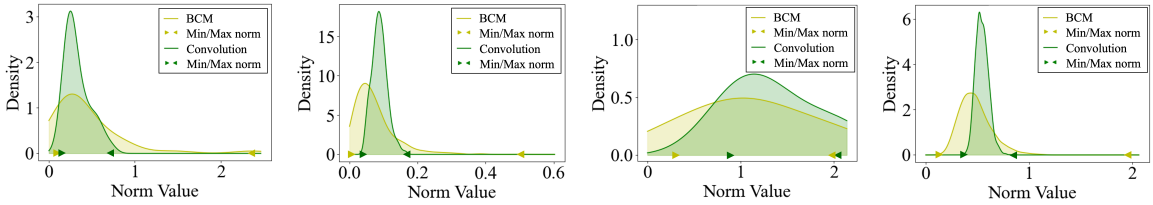
2) Limitation of Compression Parameter: BCM compression uses only one parameter, block size (BS), to determine the compression ratio. Therefore, it has several limitations. First, increasing BS directly affects the hardware cost. In general, BS should be 2^n for the FFT calculation. Thus, a larger compression ratio requires additional computation overhead. Second, the compression ratio cannot be determined at a fine-grained level. Simply adopting a large 2^n for BS can result in a significant accuracy drop, since BCM compression does not take into account the importance of weights. Therefore, there are few practical options for the compression parameters to achieve a good trade-off between accuracy and compression ratio. These limitations have been shown in the previous work. REQ-YOLO [6] applied BCM compression to a CNN-based YOLO network using only four values of BS (4, 8, 16, and 32). FTRANS [8], which applied BCM compression to a transformer-based network, used only three values of BS (4, 8, and 16). If we can assign flexible compression ratios according to importance of the weights in BCM compression, it would be beneficial to allow wider compression ratio ranges.

3) Need for Specialized Dataflow for BCM-compressed Networks on FPGAs: Depending on the buffer size, the dataflow of the CNN accelerator can be divided into four categories [15]; *i)* both inputs and weights are fully buffered, *ii)* only weights are fully buffered, *iii)* only inputs are fully buffered, and *iv)* neither inputs nor weights are fully buffered. REQ-YOLO [6] proposed a BCM-compressed network accelerator on FPGA, adopted a dataflow of *ii)*, and treated "FFT-*e*MAC-IFFT" as a computational delay in the dataflow. However, resource-constrained FPGAs cannot buffer all weight data. Therefore, most CNN accelerators for edge devices adopt the dataflow of *iv)*. In this dataflow, where off-chip access is performed tile-by-tile, it is crucial to reuse the data with a limited buffer. Also, since the input data is directly related to the amount of FFT computations in the BCM-compressed network, the resue of input data should be maximized to reduce unnecessarily repeated FFT computations. Furthermore, since each computation has a different data dependency, it is inefficient to treat "FFT-*e*MAC-IFFT" as a single computional delay in the dataflow, where all data is loaded tile-by-tile. Therefore, it is inefficient to simply adopt the existing CNN dataflow, and we need a special dataflow for BCM-compressed networks when targeting resource-constrained FPGAs.



(a) (b)

Fig. 4: (a) Each BCM is reparameterized to the Hadamard product of two BCMs. (b) Implementation of *hada*BCM. The Hadamard product and FFT can be pre-computed before the inference.



(a) ResNet-18−first layer (b) ResNet-18−last layer (c) ResNet-50−first layer (d) ResNet-50−last layer

Fig. 5: Norm distribution of pruning unit from first/last layer of ResNet-18 on Cifar-10 and ResNet-50 on ImageNet. The curves and triangle marks indicate the kernel distribution estimate (KDE) [16] of the norm distribution and min/max norm values.

III. RP-BCM FRAMEWORK

A. Hadamard-BCM Paramaterization

In Section II-B, we note that many BCMs have a limited representation in rank-perspective. For a better rank-condition of these BCMs, we propose Hadamard-BCM (*hada*BCM) in the training phase, a new parameterization for BCMs that incorporates the Hadamard product (Stage 1 in Fig. 3).

The *hada*BCM is illustrated in Fig. 4. The *hada*BCM replaces each W_{bcm} by the Hadamard product of the two same sized circulant matrices as $W_{bcm} = A_{bcm} \odot B_{bcm}$. Note that, for the circulant matrices A_{bcm} and B_{bcm}, the result of $A_{bcm} \odot B_{bcm}$ is also a circulant matrix. When $\text{rank}(A_{bcm}) = r_a$ and $\text{rank}(B_{bcm}) = r_b$, $\text{rank}(W_{bcm}) = \text{rank}(A_{bcm} \odot B_{bcm}) \leq r_a r_b$, and the upper bound is maximized when $r_a = r_b$ [10]. The weight update rule of *hada*BCM has an inherent regularization effect that makes $r_a = r_b$. Therefore, this condition can be satisfied without requiring any special training techniques other than applying the Hadamard product. The gradient of the loss function $\mathcal{L}$ with respect to A_{bcm} and B_{bcm} can be expressed as follows:

$$\frac{\partial \mathcal{L}}{\partial A_{bcm}} = \frac{\partial \mathcal{L}}{\partial W_{bcm}} \odot B_{bcm}, \quad \frac{\partial \mathcal{L}}{\partial B_{bcm}} = \frac{\partial \mathcal{L}}{\partial W_{bcm}} \odot A_{bcm}. \tag{1}$$

The gradients for A_{bcm} and B_{bcm} are regulated by the value of each other, creating an opposite feedback loop between A_{bcm} and B_{bcm}. Accordingly, this effect stabilizes the value of A_{bcm} and B_{bcm} to $r_a = r_b$ by repeating the training epoch. Using this property, the reparameterized W_{bcm} can achieve better representation by improving the rank-condition compared to the original BCM, leading to an improvement in the accuracy performance. We demonstrate this performance improvement experimentally later in Section V.

The Hadamard product can be pre-processed in the frequency domain before the inference phase (Fig. 4b). This means that *hada*BCM does not require any overhead in computation and storage on the inference accelerator for BCM-compressed networks. Accordingly, *hada*BCM can enhance accuracy while maintaining the benefits of previous BCM compression.

B. BCM-wise Pruning

BCM-wise pruning eliminates weights in the BCM unit with low importance (Stage 2 in Fig. 3). We use the ℓ_2-norm as a criterion for the importance of BCM. Norm-based pruning is one of the filter pruning methods [17]–[19]. There are two requirements [20]; *i)* the deviation of norm should be large, *ii)* the smallest norm should be small. However, since it is difficult to satisfy these points with a conventional CNN, the norm-based criterion has not been effectively used for CNNs.

We argue that the norm-based criterion is suitable for BCM-wise pruning, because the BCM-compressed network satisfies these requirements due to the structural characteristic of the circulant matrix. For the pruning unit $\mathcal{U} \in \mathbb{R}^{BS \times BS}$, the conventional CNN's $\mathcal{U}_{cnn}$ has BS^2 individual values. Meanwhile, $\mathcal{U}_{bcm}$ of the BCM-compressed network has BS values. According to the law of large number, the larger size of samples gets, the smaller the standard deviation of the sampling distribution becomes. Due to the smaller number of values contributing to the norm than $\mathcal{U}_{cnn}$, $\mathcal{U}_{bcm}$ statistically obtains a wider norm distribution than $\mathcal{U}_{cnn}$. Fig. 5 shows that $\mathcal{U}_{bcm}$ has a larger deviation than $\mathcal{U}_{cnn}$ in the real observation of the norm distribution in the trained network. It is also found that the minimum value of $\mathcal{U}_{bcm}$ is closer to zero in most of the layers. These satisfactions show that the norm-based criterion is suitable for BCM-wise pruning.

Algorithm 1: Pseudo-code for BCM-wise Pruning

input : Pre-trained $\mathbf{A}_{bcm} = \{\mathrm{A}^1_{bcm}, \mathrm{A}^2_{bcm}, ..., \mathrm{A}^{\mathrm{num}_{total}}_{bcm}\}$, and $\mathbf{B}_{bcm} = \{\mathrm{B}^1_{bcm}, \mathrm{B}^2_{bcm}, ..., \mathrm{B}^{\mathrm{num}_{total}}_{bcm}\}$, Initial pruning ratio α_{init}, Step pruning ratio α_{step}, Target Accuracy β
output: Pruned set of BCMs $\hat{\mathbf{A}}_{bcm}$ and $\hat{\mathbf{B}}_{bcm}$
Initialize α to α_{init}
Initialize $\hat{\mathbf{A}}_{bcm}$, $\hat{\mathbf{B}}_{bcm}$ to $\mathbf{A}_{bcm}$, $\mathbf{B}_{bcm}$
for *i=1 to* num_{total} **do**
  norm_list.append(ℓ_2-norm of $\mathrm{A}^i_{bcm} \odot \mathrm{B}^i_{bcm}$)
end
$\mathrm{norm\_list}_{sort} \leftarrow$ *sort*(norm_list)
while $Acc_{best} \geq \beta$ **do**
  $\mathrm{num}_{prune} \leftarrow \alpha * \mathrm{num}_{total}$
  $\mathrm{V}_{threshold} \leftarrow \mathrm{norm\_list}_{sort}[\mathrm{num}_{prune}]$
  for *i=1 to* num_{total} **do**
    if $\mathrm{norm\_list}[i] \leq \mathrm{V}_{threshold}$ **then**
      Eliminate $\hat{\mathrm{A}}^i_{bcm}$ and $\hat{\mathrm{B}}^i_{bcm}$
    end
  end
  Fine-tunining with $\hat{\mathbf{A}}_{bcm}$, $\hat{\mathbf{B}}_{bcm}$
  $\alpha \leftarrow \alpha + \alpha_{step}$
end

Algorithm 1 indicates the overall process of BCM-wise pruning. After training with *hada*BCM, we obtain the sets of BCMs, $\mathbf{A}_{bcm}$ and $\mathbf{B}_{bcm}$. We obtain ℓ_2-norm of $W^i_{bcm} = A^i_{bcm} \odot B^i_{bcm}$, where $i = 1, \ldots, \mathrm{num}_{total}$ (*Line 5*). The number of BCMs to be removed, num_{prune}, is determined according to α, the adaptive pruning ratio parameter. Accordingly, A^i_{bcm} and B^i_{bcm} that have lower norm than the $V_{threshold}$ are eliminated, and the network is fine-tuned. First,

α is initialized to pre-determined α_{init}. As long as the fine-tuned accuracy meets the target accuracy β, α is updated to a larger ratio by incrementally adding with small steps, α_{step}. Fine-tuning is done iteratively using the pruned network with updated α to obtain an optimal pruning ratio until the fine-tuned accuracy reaches β. By adjusting α and β, we can determine the degree of trade-off between the accuracy and the compression ratio at a more fine-grained level.

IV. Hardware Implementation

A. Overall Architecture

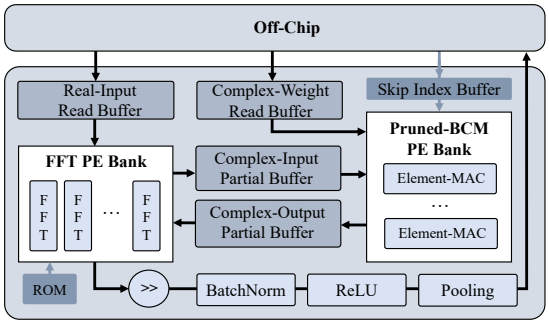


Fig. 6: Overall architecture of RP-BCM on FPGA.

The hardware architecture consists of buffers, FFT processing elements (PE) bank, Pruned-BCM PE bank, and non-linear modules (Fig. 6). The complex weights are loaded directly after pre-processing the weight data with the Hadamard product and FFT. The FFT PE performs the conversion between real data and complex data. Essential data for the FFT, such as the twiddle factor, are pre-stored in the ROM. The Pruned-BCM PE bank performs the *e*MAC by exploiting BCM-wise sparsity and produces a complex partial output.

B. PE Design for BCM-wise Pruning

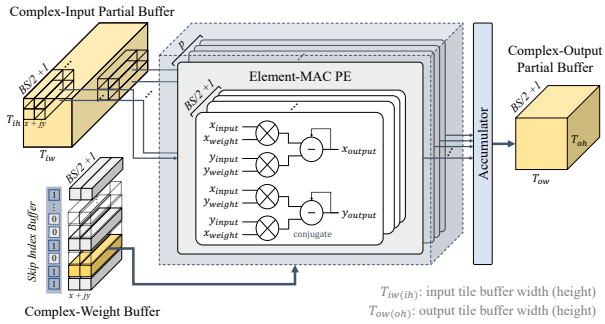


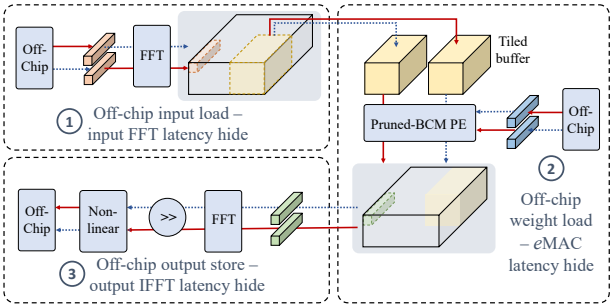
Fig. 7: Pruned-BCM PE design exploiting BCM-wise pruning. *p* indicates the parallelism factor.

Since the FFT PE is designed using the well-known Cooley-Tukey FFT algorithm [21], a detailed description is omitted. The IFFT is computed by reusing the FFT module with *BS* size-divider and conjugate. Since the FFT size is always 2^n, we implement a *BS* size-divider as a $\log_2 BS$ shift operator to reduce the expensive hardware cost of the divider. We integrate the conjugate operation into the MAC of the Pruned-BCM PE (Fig. 7).

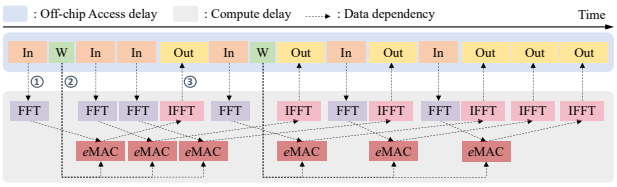
A Pruned-BCM PE includes multiple element-MAC (*e*MAC) PEs (Fig. 7). One *e*MAC PE executes the complex *e*MAC for the *BS*-size partial weights and *BS*-size partial inputs. Note that *BS*-size computation consists of only $\frac{BS}{2}$+1 MAC operations, since the FFT result of the real value is conjugate-symmetric [6]. *p* indicates the number of pruned-BCM PE in one bank. It is the parallelism factor determined according to the resource capability. *p* PEs execute *e*MAC with each different partial inputs with reusing the same *BS*-weight in parallel. Before the computation, the PE controller checks the skip index bit, which indicates whether the corresponding BCM is pruned or not. When the skip index is zero, the PE controller skips the execution of the PE banks for the corresponding pruned weight. Then, the PE controller immediately executes the PE banks for the next non-pruned BCM-weight.

Note that the dataflow can maintain high-parallelism even with sparsity, since skip processing is performed over multiplie PEs share the same partial weights. The skip index buffer is a negligible overhead, only one bit per BCM. For example, for the $K\times K\times C_{in}\times C_{out}$ size convolution layer, the skip index buffer size is only $K\times K\times \frac{C_{in}}{BS}\times \frac{C_{out}}{BS}\times 1$ *bits*. Overall, the computation process is maintained with only the costs of checking the skip index that consumes very little time compared to main computation of PE.

C. Fine-grained Dataflow for RP-BCM



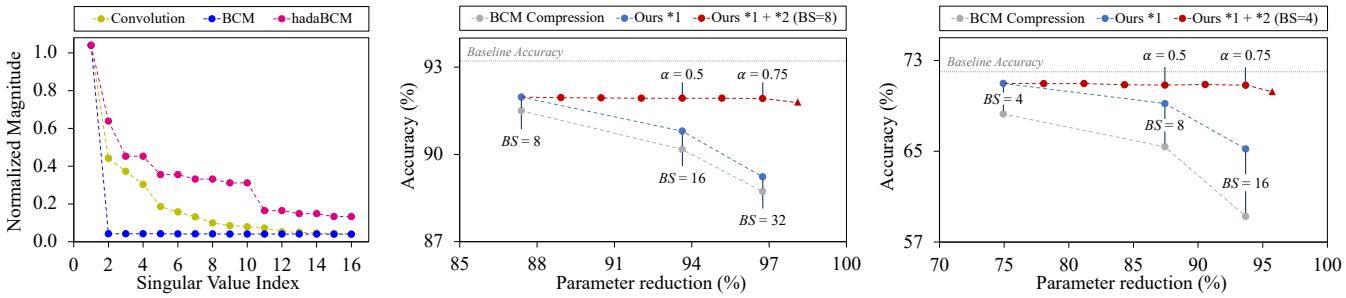
(a)



(b)

Fig. 8: (a) Proposed dataflow with separated double buffering. Red and blue arrows indicate alternative execution with each double buffering. (b) Example of dataflow in tile-by-tile. One weight tile is reused for multiple input tiles to produce multiple output tiles. For comparison with CNN's conventional dataflow, this access manner is the same as in Ma *et al.* [15].

As mentioned in Section II, most accelerators for edge devices use a dataflow of where neither the inputs nor the weights are fully buffered due to insufficient resources. We propose a fine-grained dataflow with BCM-compressed network for the tile-by-tile process (Fig. 8a). There are three tile-by-tile off-chip accesses: input read, weight read, and output store. The BCM-compressed network consists of three types of computations: FFT ($\mathcal{C}_{fft}$), *e*MAC ($\mathcal{C}_{emac}$), and IFFT ($\mathcal{C}_{ifft}$). We separate these computations to each computational delay and apply double buffering for each off-chip access. Each $\mathcal{C}$ has a different data dependency that requires off-chip access. $\mathcal{C}_{fft}$, $\mathcal{C}_{emac}$, and $\mathcal{C}_{ifft}$ need off-chip access for the real input, complex weight, and real output, respectively. Each double buffering can hide the corresponding off-chip access latency with the respective $\mathcal{C}$ latency. Double buffering for $\mathcal{C}_{fft}$ and $\mathcal{C}_{ifft}$ requires only a small overhead of two *BS*-size buffer. This is because FFT and IFFT are executed per *BS*-size. For $\mathcal{C}_{emac}$, the overhead depends on the parallelism of the PE banks. Depending on the resource capability of the targeted FPGA, the size of the complex partial input/output buffer and the parallelism of the PE bank are determined. This has a direct impact on the computation delay of $\mathcal{C}_{emac}$. Moreover, each PE independently executes the corresponding $\mathcal{C}$ in parallel, hiding the latency between each $\mathcal{C}$, not just between off-chip access and $\mathcal{C}$ (Fig. 8b).



(a) Singular values decaying (b) VGG-16 on Cifar-10 (c) VGG-19 on Cifar-100

Fig. 9: (b) and (c) indicate the accuracy and parameter reduction of proposed RP-BCM. Triangle marks indicate the break-down point of Algorithm 1 within target accuracy β (92.0% and 71.0%, respectively). Ours *1 and *2 indicate *hada*BCM and BCM-wise pruning, respectively.

TABLE I: Comparison with other compression on ImageNet.

Method	Top-1		Top-5		FLOPs ↓ (%)	Params. ↓ (%)
	Acc. (%)	△ (%)	Acc. (%)	△ (%)		
Baseline	76.15	-	92.87	-	-	-
BPPS [22]	70.58	-5.57	90.00	-2.87	75.80	68.55
GAL [23]	71.80	-4.35	90.82	2.05	55.01	24.27
HRank [9]	71.98	-4.17	91.01	-1.86	62.10	46.00
ThiNet [24]	72.04	-4.11	90.67	2.20	36.79	33.72
Ours (*BS*=8, α=0.5)	**71.99**	**-4.16**	**90.25**	**-2.62**	**77.33**	**92.40**
TRP [11]	72.69	-3.46	91.41	-1.46	56.50	N/A
BPPS [22]	73.06	-3.09	91.30	-1.57	67.97	57.49
CHIP [25]	73.30	-2.85	91.48	-1.39	76.70	68.60
FPGM [26]	74.83	-1.32	92.32	-0.55	53.50	N/A
Ours (*BS*=4, α=0.7)	**73.12**	**-3.02**	**91.42**	**-1.45**	**68.88**	**88.79**

TABLE II: Resource estimation with the proposed skip scheme.

	BRAM	DSP	FF	LUT
w/o Skip Scheme	109 (78%)	116 (53%)	30426 (29%)	17879 (34%)
w/ Skip Scheme	112.5 (80%)	117 (53%)	32105 (30%)	18203 (34%)

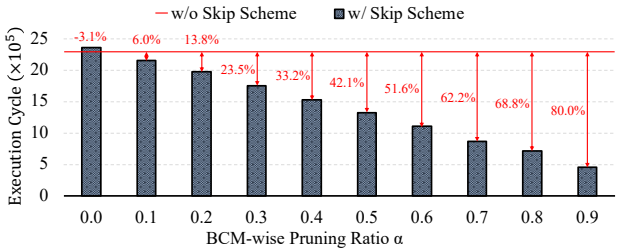


Fig. 10: Estimation of the execution cycle according to the pruning ratio.

V. Experimental Results

A. Experimental Settings

To evaluate the accuracy improvement and compression, we implemented common CNN-based networks with our proposed method. To show that the proposed method works well for various network architectures and datasets, we compressed VGG-16, VGG-19, and ResNet-50 on Cifar-10, Cifar-100, and ImageNet, respectively. For training, we used a SGD optimizer and a cosine annealing scheduler. To evaluate the performance of the proposed hardware design, we designed the hardware using *Xilinx Vivado HLS*, a high-level (C/C++) based design tool. Since we targets low-power embedded systems, we selected *Xilinx PYNQ-Z2* (XC7Z020) as the target FPGA board, which has low resources such as 630Kb BRAM, 220 DSPs.

B. Experimental Results of Proposed RP-BCM

To show the accuracy and compression performance of RP-BCM, which comprises *hada*BCM and BCM-wise pruning, we first show and analyze the performance improvement of each stage. Then, we also show the comparison with state-of-the-art compression methods.

1) Accuracy Improvement of the hadaBCM: Fig. 9a illustrates the singular values decaying of the same BCM as in the left side of Fig. 2. Compared to the BCM, which has an extremely exponential decay, the singular values of the *hada*BCM decay more linearly. A more linear decay of the singular values indicates an improved rank-condition of the matrix [12]. These improved BCMs were observed across the entire network. For the traditional BCM-compressed VGG-16 on Cifar-10, 72.2% of the BCMs have a poor rank-condition. In contrast, only 2.1% of the BCMs have poor rank-condition in the *hada*BCM-compressed network. These enhanced rank-condition of BCMs can have a better representation, leading to an accuracy improvement, as shown in Figs. 9b and 9c (Ours *1).

2) Accuracy and Compression Results of the BCM-wise Pruning: After applying *hada*BCM, we pruned the weights in BCM-wise manner, and fine-tuned VGG-16 on Cifar-10 and VGG-19 on Cifar-100 with target accuracy β=92.0%, and β=71.0%, respectively. For comparison, we indicated the α in the same total parameter reduction with traditional BCM compression in Figs. 9b and 9c. For example, our method with *BS*=8 and α=0.5 had the same parameter reduction as traditional BCM compression with *BS*=16 in VGG-16. In VGG-16 on Cifar-10, when *BS*=8 and α=0.75, it showed a 3.25% accuracy improvement compared to traditional BCM compression with the same parameter reduction. For more complex datasets (VGG-19 on Cifar-100), our result using *BS*=4 and α=0.75 showed an 11.57% accuracy improvement compared to traditional BCM compression using only *BS*=16. The impact of BCM-wise pruning can be divided into two main points. First, it provides more diverse compression ratios, in contrast to traditional BCM compression, which only has the compression parameter *BS* that increases exponentially to 2^n. The proposed method can ahieve more diverse compression ratios by setting an additional adaptive parameter, α. Also, since our method compresses the network with considering the importance of the weights, our method achieves better accuracy performance with a huge parameter reduction.

3) Accuracy and Compression of Entire Framework: Aforementioned VGG-16 and VGG-19 networks have only 1.2% and 1.3% accuracy losses, respectively. With these small accuracy losses, our method achieved 96.75% and 93.68% parameter reduction in each network. Note that, greater compression than we reported can be achieved by adjusting α and β in Algorithm 1. To evaluate on more complex datasets, we applied RP-BCM to ResNet-50 on ImageNet. Table I shows the accuracy, FLOPs reduction and parameter reduction in comparison. We conducted the experiment by setting the target Top-1 accuracy β in two categories: 92% and 93% with *BS*=8 and 4, respectively. Our method achieved 77.3% FLOPs reduction and 92.40% parameter reduction with an accuracy of 71.99%. With the 73.12% accuracy, the result showed 68.88% and 88.80% reduction in FLOPs and parameter, respectively. This means that the degree of compression can be determined in fine-grained levels by adjusting the target accuracy. One noticeable thing is that the parameter reduction of our result had quite a high ratio compared to other methods. These results support that our method facilitates deployment of neural

TABLE III: Comparison of CNN implementation with GPU, the previous works and our implementation on PYNQ-Z2 (XC7Z020).

Implementation	ResNet-18 (GTX 1080Ti)	VGG [27]	ResNet-18 [28]	ResNet-18 [28]	**ResNet-18 (Ours)**
Method	-	Quantization (W8A8)	Mixed-Precision Quantization (W4A5 + First & Last W8A5)	Mixed-Precision Quantization (95% W4A5 + 5% W8A5)	RP-BCM (*hada*BCM + Pruning)
Frequency (MHz)	-	214	100	100	100
kLUT	-	29.9 (56%)	39.1 (74%)	41.3 (78%)	18.2 (34%)
DSP	-	190 (86%)	214 (97%)	208 (95%)	117 (53%)
BRAM	-	85.5 (61%)	126.5 (90%)	123 (88%)	112.5 (80%)
Power (W)	148.54	3.5	3.0	3.5	1.83
Frame Rate (FPS)	325.73	2.72	12.9	27.8	12.5
FPS/kLUT	-	0.09	0.33	0.67	0.69
FPS/DSP	-	0.014	0.06	0.13	0.11
FPS/W	2.19	0.11	4.3	7.94	6.83

networks on embedded systems. This is because the parameter size directly affects the DRAM access that is the largest latency bottleneck.

C. Performance of Accelerator Design

1) Resource Estimation and Latency with BCM-wise Sparsity: Table II shows the resource utilization with and without applying the proposed scheme. Both designs have the same parallelism of PEs and the same dataflow. This result shows that the resource overhead of our design is quite low. Fig. 10 represents the execution cycle estimation according to α. We performed the simulation of the one layer of ResNet-18, which has the feature map size = $128\times28\times28$, and weight kernel size = 3×3. With the BCM-wise sparsity, the proposed PE showed the linear tendency of cycle reduction according to the α. This means that the proposed design can maintain high parallelism with sparsity. To estimate the overhead time of the skip operation, we compared the execution cycle of the proposed PE and the conventional PE with α=0. For the non-pruned layer, the execution cycle was increased by 3.1%, compared to conventional PE. This is quite a small overhead compared to the main computations. Consequently, our proposed design can maintain high parallelism exploiting BCM-wise sparsity with negligible extra overhead in both resources and execution time.

2) Efficiency for Embedded Systems: Resource efficiency and energy efficiency are important for low-power embedded systems. To evaluate the efficiency of our proposed design, we compared the resource efficiency (FPS/kLUT and FPS/DSP) and energy efficiency (FPS/W) with GPU (*Nvidia* GTX 1080Ti) and other CNN accelerators using the same target board, *XC7Z020* (Table III). We implemented the ResNet-18 on ImageNet with our proposed RP-BCM. The compression parameters were *BS*=8 and α=0.5, which were used in Table I. Our design achieved 0.69 FPS/kLUT, 0.11 FPS/DSP, and 6.83 FPS/W. Compared to the GPU, our design showed 3.1$\times$ higher FPS/W, indicating higher energy efficiency. Although our implementation had low resource utilization and any quantization was not applied, our design with just 16-bit fixed-point computation showed similar resource and energy efficiency as the previous implementation with up to 4-bit quantization. Note that, since the dedicated quantization methods for BCM-compressed network are available [6], [29], such quantization methods may lead to further improvement in the energy efficiency of our framework.

VI. Conclusion

In this paper, we propose a rank-enhanced and highly-pruned BCM compression (RP-BCM) framework which overcomes the limitations of existing BCM compression methods. We propose to use *hada*BCM to enhance the rank condition of BCM improving the representation of the compressed network. We also suggest BCM-wise pruning, which introduces high flexibility to choose the trade-off between accuracy and compression. We introduce dedicated PE design to exploit structured sparsity from the BCM-wise pruning, and fine-grained dataflow for BCM-compressed networks. As a result, our method achieved outstandingly high compression performance with negligible accuracy degradation. Especially, our proposed method achieved the largest parameter reduction compared to other compression methods with the same target accuracy. The experiments of hardware implementation demonstrated the suitability of our method in deploying DNNs on low-power embedded systems.

References

[1] K. Simonyan, and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *Proc. ICLR*, 2015.

[2] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey", *Proceedings of the IEEE 108(4)*, 2020, pp. 485-532.

[3] S. Han, H. Mao, and W. J. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding", *Proc. ICLR*, 2016.

[4] C. Ding, et al., "CirCNN: accelerating and compressing deep neural networks using block-circulant weight matrices", *Proc. MICRO*, 2017, pp. 395-408.

[5] Y. Cheng, and S. F. Chang et al., "An exploration of parameter redundancy in deep networks with circulant projections", *Proc. ICCV*, 2015, pp. 2857-2865.

[6] C. Ding, and Y. Liang et al., "REQ-YOLO: A resource-aware, efficient quantization framework for object detection on FPGAs", *Proc. FPGA*, 2019, pp. 33-42.

[7] S. Wang, Z. Li, C. Ding, and Y. Liang et al., "C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs", *Proc. FPGA*, 2018, pp. 11-20.

[8] B. Li, S. Pandey, H. Fang, and C. Ding et al., "Ftrans: energy-efficient acceleration of transformers using fpga", *Proc. ISLPED*, 2020, pp. 175-180.

[9] M. Lin, R. Ji, Y. Wang, and L. Shao et al., "Hrank: Filter pruning using high-rank feature map", *Proc. CVPR*, 2020, pp. 1529-1538.

[10] N. Hyeon-Woo, M. Ye-Bin, and T.-H. Oh, "FedPara: low-rank hadamard product for communication-efficient federated learning", *Proc. ICLR*, 2022.

[11] Y. Xu, and H. Xiong et al., "TRP: Trained rank pruning for efficient deep neural networks", *Proc. IJCAI*, 2020, pp. 977-983.

[12] T.-H. Oh, Y. Matsushita, Y. W. Tai, and I. S. Kweon, "Fast randomized singular value thresholding for low-rank optimization", *IEEE Transactions on PAMI*, 2017, pp. 376-391.

[13] N. Halko, P. H. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions", *SIAM review*, 2011, pp. 217-288.

[14] O. Roy, and M. Vetterli, "The effective rank: A measure of effective dimensionality", *Proc. EUSIPCO*, 2007, pp. 606-610.

[15] Y. Ma, Y. Cao, S. Vrudhula, and J. S. Seo, "Performance modeling for CNN inference accelerators on FPGA", *IEEE Transactions on CAD*, 2019, pp. 843-856.

[16] B. W. Silverman, "Density estimation for statistics and data analysis", *Routledge*, 2018.

[17] H. Li, A. Kadav, I. Duranovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets", *Proc. ICLR*, 2017.

[18] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks", *Proc. IJCAI*, 2018.

[19] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers", *Proc. ICLR*, 2018.

[20] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning geometric median for deep convolutional neural networks acceleration", *Proc. CVPR*, 2019, pp. 4340-4349.

[21] J. W. Cooley, and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series", *Mathematics of computation*, 1990, pp. 297-301.

[22] D. Li, S. Chen, X. Liu, Y. Sun, and L. Zhang, "Towards optimal filter pruning with balanced performance and pruning speed", *Proc. ACCV*, 2020.

[23] S. Lin, R. Ji, C. Yan, and D. Doermann et al., "Towards optimal structured cnn pruning via generative adversarial learning", *Proc. CVPR*, 2019, pp. 2790-2799.

[24] J. H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression", *Proc. ICCV*, 2017, pp. 5058-5066.

[25] Y. Sui, and B. Yuan et al., "CHIP: CHannel independence-based pruning for compact neural networks", *Proc. NeurIPS*, 2021, pp. 24604-24616.

[26] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration", *Proc. CVPR*, 2019, pp. 4340-4349.

[27] K. Guo, and H. Yang et al., "Angel-eye:A complete design flow for mapping CNN onto embedded FPGA", *IEEE Transactions on CAD 37(1)*, 2017, pp. 35-47.

[28] M. Sun, and Z. Fang et al., "FILM-QNN: Efficient FPGA accelerator of deep neural networks with intra-layer, mixed-precision quantization", *Proc. FPGA*, 2022, pp. 134-145.

[29] Y. He, J. Yue, Y. Liu, and H. Yang, "Block-circulant neural network accelerator featuring fine-grained frequency-domain quantization and reconfigurable FFT modules", *Proc. ASP-DAC*, 2021, pp. 813-818.