electronics

MDPI

Article

Accelerating Deep Neural Networks by Combining Block-Circulant Matrices and Low-Precision Weights

Zidi Qin [1], Di Zhu [1], Xingwei Zhu [1], Xuan Chen [1], Yinghuan Shi [2], Yang Gao [2], Zhonghai Lu [3], Qinghong Shen [1], Li Li [1] and Hongbing Pan [1,*]

1 School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China; qinzidi@smail.nju.edu.cn (Z.Q.); zhudi@smail.nju.edu.cn (D.Z.); flzs@smail.nju.edu.cn (X.Z.); cx0705@smail.nju.edu.cn (X.C.); qhshen@nju.edu.cn (Q.S.); lili@nju.edu.cn (L.L.)

2 State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China; syh@nju.edu.cn (Y.S.); gaoy@nju.edu.cn (Y.G.)

3 School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, 114 28 Stockholm, Sweden; zhonghai@kth.se

* Correspondence: phb@nju.edu.cn

Received: 23 November 2018; Accepted: 5 January 2019; Published: 10 January 2019

Abstract: As a key ingredient of deep neural networks (DNNs), fully-connected (FC) layers are widely used in various artificial intelligence applications. However, there are many parameters in FC layers, so the efficient process of FC layers is restricted by memory bandwidth. In this paper, we propose a compression approach combining block-circulant matrix-based weight representation and power-of-two quantization. Applying block-circulant matrices in FC layers can reduce the storage complexity from $\mathcal{O}(k^2)$ to $\mathcal{O}(k)$. By quantizing the weights into integer powers of two, the multiplications in the reference can be replaced by shift and add operations. The memory usages of models for MNIST, CIFAR-10 and ImageNet can be compressed by 171×, 2731× and 128× with minimal accuracy loss, respectively. A configurable parallel hardware architecture is then proposed for processing the compressed FC layers efficiently. Without multipliers, a block matrix-vector multiplication module (B-MV) is used as the computing kernel. The architecture is flexible to support FC layers of various compression ratios with small footprint. Simultaneously, the memory access can be significantly reduced by using the configurable architecture. Measurement results show that the accelerator has a processing power of 409.6 GOPS, and achieves 5.3 TOPS/W energy efficiency at 800 MHz.

Keywords: hardware acceleration; deep neural networks (DNNs); fully-connected layers; network compression; VLSI

1. Introduction

Deep neural networks (DNNs) have been widely applied to various artificial intelligence (AI) applications [1–4] and achieve great performance in many tasks such as image recognition [5–7], speech recognition [8] and object detection [9]. To complete the tasks with higher accuracy, larger and more complex DNN models emerge and become increasingly popular. Large scale DNNs, however, require massive parameters and high computation complexity. For instance, there are 138 million weights and 15.5 billion multiply-accumulate (MAC) operations in VGG-16 for image recognition [10]. As a result, the implementation of these DNN models is challenging for portable devices with restricted computational capability and memory resources.

Fully-connected (FC) layers are applied in various deep learning systems. Neurons in an FC layer have connections to all activations in the previous layer, making FC layers the most memory

intensive in large DNN networks. For example, 96% and 90% of weights storage are taken by FC layers in AlexNet and VGG-16, respectively. In models that only consist of FC layers, there can be billions of parameters [11]. The efficient process of FC layers is mainly limited by bandwidth in large networks [12]. Thus, it is essential to reduce the memory access of FC layers, especially when they are deployed in embedded devices containing limited hardware resources.

Several network compression techniques have been proposed to reduce the redundant parameters in DNNs. Targeting the compressed networks, hardware accelerators have been proposed to process DNNs with higher performance and energy efficiency. Focusing on pruning-based technique [13], a corresponding hardware accelerator has been developed in [12]. By exploiting the sparsity of both weights and activations, the weights are stored in a compressed format to save memory usage and unnecessary computations are removed. The pruning-based technique, however, leads to irregular network structure and unbalanced computation [12]. In [14–17], quantization for weights with low precision has been explored. Although bringing some accuracy loss, quantization remains an efficient way to reduce the weight memory size and improve the energy efficiency of DNN hardware. In [18–20], DNN accelerators based on binary weights have been proposed.

Using structured matrices to explore the redundancy of DNNs is another efficient technique. Circulant matrix-based methods can provide good compression ratio for weights, and Fast Fourier Transformation (FFT) can be employed to reduce the computational complexity [21–24]. In [24], block-circulant matrices are employed in FC layers and the circulant matrix-vector multiplications are accelerated by FFT-based methods in both the training and inference. Compared with pruning-based approach in [13], this method avoids irregular sparse network structure and inefficient implementation of activations and indices. Liao *et al.* [25] proposed block-circulant matrix-based DNN training and inference schemes, as well as an optimized FFT-based hardware architecture. Compared with prior work, this approach achieves significantly improvement in energy efficiency. However, using FFT and IFFT transformation limits the quantization of weights. To achieve good balance between the hardware performance and precision loss in FFT-based method, at least 12 bits are required to represent a weight. In addition, additional computation components are needed to perform FFT and IFFT.

In this paper, an efficient compression technique for fully-connected layers is proposed. We employ block-circulant matrix-based method to compress the network weight matrices and further quantize each entry in weight matrices to an integer power of two. For block-circulant matrix-based FC layers with low precision weights, a configurable hardware accelerator, called BPCA, is proposed. The main contributions of the paper are summarized as follows.

(1) An efficient compression technique combining block-circulant matrices and power-of-two quantization for fully-connected layers is proposed. This approach can significantly save the networks' storage usage. Specifically, we directly train the FC layers in DNN models with block-circulant matrices and the storage requirement can be reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. Based on the above models, we further quantize each entry in weight matrices to an integer power of two. The computational complexity can be significantly reduced because the multiplications in the forward pass can be replaced by shift operations.

(2) An efficient, configurable and scalable hardware architecture is proposed for the compressed networks. Instead of FFT-based acceleration methods, we design this architecture to take advantage of the circular feature of weight matrices and power-of-two quantization. Notably, multipliers are replaced by customized processing elements (PEs) using shift and add operations. Due to the adjustable sizes of block-circulant matrices, we design a configurable hardware architecture. A reorganization scheme is used to realize the configurability of layers and reduce the parameters' memory access.

(3) Experiments on several datasets (MNIST, CIFAR10, and ImageNet) are presented to prove the general applicability of the proposed approach. According to our experiments, 4 bits are enough to represent the weights, thus quantization provides eight times additional compression ratio.

The proposed hardware architecture was also implemented and evaluated. The implementation results demonstrate that the proposed design has high energy efficiency and area efficiency.

2. Background

2.1. Computation of FC Layers

Matrix-vector multiplication (MV) is the key computation in FC layers in the DNN inference. The computation in an FC layer is performed as follows

$$y = f(Wx + v) \tag{1}$$

where x is the input activation vector; y is the output activation vector; $W \in \mathbb{R}^{a \times b}$ is the weight matrix of this layer in which b input nodes connect with a output nodes; $v \in \mathbb{R}^a$ is the bias vector; and $f(\cdot)$ is the nonlinear activation function and the Rectified Linear Unit (ReLU) is widely adopted in various DNN models.

The space complexity and computational complexity of an FC layer are $O(ab)$. In practice, the values of a and b in an FC layer can be large and there is no data reuse, so the computation of FC layers is memory intensive. For many neural network architectures, the memory access is the bottleneck to process FC layers efficiently.

2.2. Block-Circulant Matrix

A circulant matrix $W^c \in \mathbb{R}^{k \times k}$ can be defined by a primitive vector $w_c = (w_c^1, w_c^2, \ldots, w_c^k)$, which is the first row of the circulant matrix:

$$W^c = \begin{bmatrix} {w_c}^1 & {w_c}^2 & {w_c}^3 & \cdots & {w_c}^k \\ {w_c}^k & {w_c}^1 & {w_c}^2 & \cdots & {w_c}^{k-1} \\ {w_c}^{k-1} & {w_c}^k & {w_c}^1 & \cdots & {w_c}^{k-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ {w_c}^2 & {w_c}^3 & {w_c}^4 & \cdots & {w_c}^1 \end{bmatrix}. \tag{2}$$

If a matrix is composed of a set of circulant sub-matrices (blocks), it is defined as a block-circulant matrix [24]. A block-circulant matrix W^b can be expressed by

$$W^b = \begin{bmatrix} {W^c}_{1,1} & {W^c}_{1,2} & {W^c}_{1,3} & \cdots & {W^c}_{1,j} \\ {W^c}_{2,1} & {W^c}_{2,2} & {W^c}_{2,3} & \cdots & {W^c}_{2,j} \\ {W^c}_{3,1} & {W^c}_{3,2} & {W^c}_{3,3} & \cdots & {W^c}_{3,j} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ {W^c}_{i,1} & {W^c}_{i,2} & {W^c}_{i,3} & \cdots & {W^c}_{i,j} \end{bmatrix}, \tag{3}$$

where ${W^c}_{i,j} \in \mathbb{R}^{k \times k}$ is a circulant sub-matrix and can be represented with a primitive vector of length k. Thus, if a normal matrix is transformed into a block-circulant matrix, the numbers of parameters to be stored can be reduced by k times.

3. Proposed Compression Method

The approach to present weight matrices of DNNs using block-circulant matrices is proposed in [24], where an FFT-based acceleration method is applied to accelerate the inference and training. Instead of the FFT-based method, we propose to employ the power-of-two quantization in the block-circulant matrix-based DNNs to further reduce the storage requirement and computational complexity. This section describes the proposed compression method combining block-circulant matrix-based weight representation and power-of-two quantization.

3.1. Block-Circulant Matrix-Based FC Layers

When a block-circulant matrix is applied in an FC Layer, the original weight matrix $W \in \mathbb{R}^{a \times b}$ can be represented by 2D blocks of square sub-matrices, where each sub-matrix is a circulant matrix. Assume that the weight matrix W is partitioned into $p \times q$ sub-matrices and the block size (size of each sub-matrix) is denoted by k. Here, $p = \frac{a}{k}$ and $q = \frac{b}{k}$. Then, $W = [W_{ij}]$, $i \in \{1 \dots p\}$, $j \in \{1 \dots q\}$, and assume that the primitive vector of W_{ij} is $w_{ij} = (w_{ij}^1, w_{ij}^2, w_{ij}^3, \dots, w_{ij}^k)$. Simultaneously, the input vector is divided into q sub-vectors and then $x = [x_1, x_2, x_3, \dots, x_j]^T, j \in \{1 \dots q\}$, where $x_j = [x_j^1, x_j^2, \dots, x_j^k]^T$. Then, the $M \times V$ in Equation (1) is given by (with bias and ReLU omitted):

$$y = Wx = \begin{bmatrix} \sum_{j=1}^{q} W_{1j} x_j \\ \sum_{j=1}^{q} W_{2j} x_j \\ \dots \\ \sum_{j=1}^{q} W_{pj} x_j \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_p \end{bmatrix}, \tag{4}$$

where $y_i \in \mathbb{R}^k$ and $y_i = [y_i^1, y_i^2, \dots, y_i^k]^T$.

Figure 1 illustrates the representation approach. A weight matrix is partitioned into $p \times q$ circulant sub-matrices and the block size is 4. Then, the result of Wx can be computed by the multiplications of sub-matrices and corresponding sub-vectors. As described in Section 2.2, only the primitive vector of each sub-matrix need to be stored, so the storage complexity of an FC layer will be reduced from $\mathcal{O}(ab)(\mathcal{O}(pqk^2))$ to $\mathcal{O}(pqk)$.

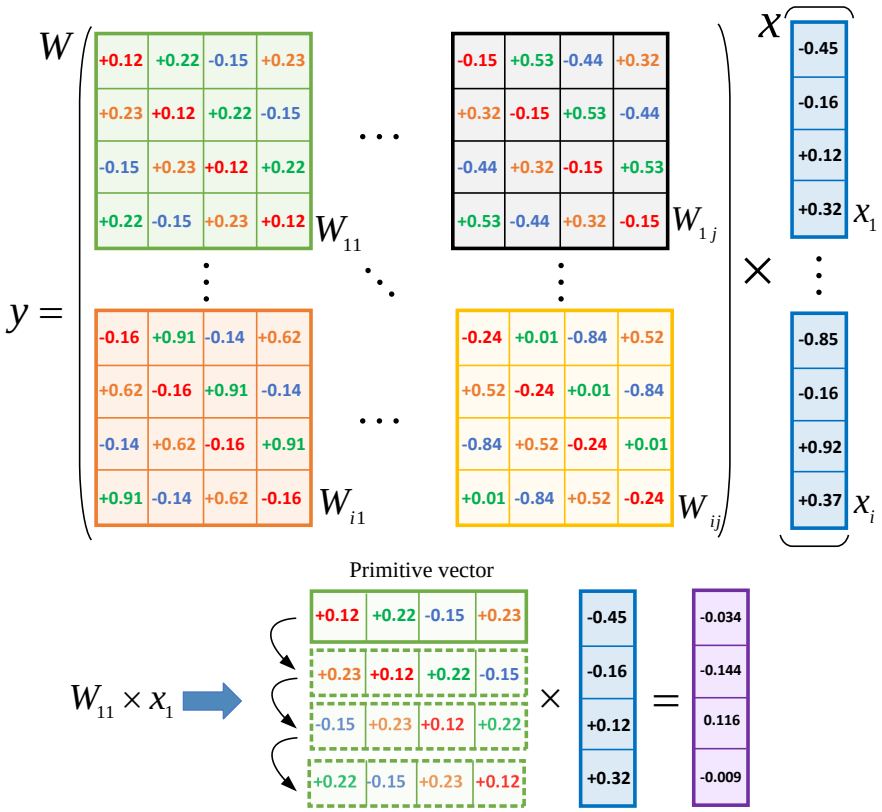

Figure 1. Block-circulant matrices representation for FC Layers.

Because of the flexible size of block-circulant matrix, the compression ratio is adjustable and determined by the block size. Larger block size brings better compression ratio, but there will be more accuracy loss. On the contrary, smaller block size provides higher prediction accuracy, but the

compression ratio may degrade. Notably, block-circulant matrix-based DNNs have been proven to asymptotically approach the accuracy of original networks using theoretical analysis [26].

For the training of block-circulant matrix-based DNNs, the corresponding training algorithm based on backward propagation is proposed in [24]. In this work, we do not use FFT in the inference and training, so we remove FFT from the training algorithm in [24]. The compressed DNNs are directly trained with block-circulant matrix-based weights, where no retrain process is required.

3.2. Power-of-Two Quantization

Suppose that a DNN model, which uses block-circulant matrices in FC layers, is pre-trained with full-precision weights. We propose to represent each entry in the weight matrix with an integer power of two. Thus, the weight matrix W can be approximated by a low-precision weight matrix $\widehat{W}$, and each entry $\hat{w}$ of $\widehat{W}$ is defined by

$$\hat{w} = \begin{cases} 0 & \text{if } w = 0 \\ sign(w) \cdot 2^n & \text{otherwise} \end{cases} \tag{5}$$

where n is an integer and sign function is used to decide the sign of $\hat{w}$. Here, n is calculated by

$$n = round(log_2 \mid w \mid), \tag{6}$$

where $n \in [n_1, n_2]$, n_1 and n_2 are two integers, and $n_1 < n_2$. Thus, all the entries of $\widehat{W}$ can be selected from a codebook set

$$S_n = \{0, \pm 2^{n_1}, \ldots, \pm 2^{n_2}\}. \tag{7}$$

If we use b-bit indices to represent the entries of S_n, there are total $M = 2^b - 1$ combinations of indices. To determine the value of n_2, we can find the maximal entry of $\widehat{W}$ and then use Equation (6) to calculate n_2. The minimum value of n_1 can be determined by $n_1 = n_2 - 2^{b-1} + 2$. For example, we use 4 bit indices and the values of original weights are limited in $[-1, 1]$. Then, n_2 is 0 and the minimal value of n_1 is -6. Thus, the codebook set $S_n = \{0, \pm 2^{-6}, \pm 2^{-5}, \pm 2^{-4}, \pm 2^{-3}, \pm 2^{-2}, \pm 2^{-1}, \pm 1\}$. Then, we use the 4-bit indices to encode each entry of S_n.

Since all entries are quantized into integer powers of two, the multiplications in the $M \times V$ can be replaced by shift operations. The computational complexity in the inference can be significantly reduced and no multipliers are required.

3.3. Training and Quantization Strategy

A two-stage training process combining block-circulant matrix-based weight representation and retrain-based quantization strategy is proposed. Algorithm 1 describes the training process. In the first stage, block-circulant matrices are employed in FC layers of a DNN model. Specifically, primitive vectors of the sub-matrices in FC layers are randomly initialized with normal distribution. Then, other parameters in the block-circulant matrices are generated by using the primitive vectors. Next, we train the network with the block-circulant matrix-based training algorithm. In the second stage, based on the pre-trained model, we further quantize the weights with power-of-two scheme, and then retrain the network with quantized parameters. At the same time, the network weight matrices are still represented with block-circulant matrices.

Algorithm 1 Training strategy with block-circulant matrices and weight quantization. PoT() is power-of-two quantization and C is the loss function.

Input: A minibatch of inputs I , targets a^t, previous weights W^t and previous learning rate η^t.

Output: Updated weights W^{t+1} and updated learning rate η^{t+1}

1: Quantize the original weights:

2: W^q = **PoT** (W^t)

3: o_t = **PoTForward**(I, W^q) //Forward propagation with the quantized parameters in block-circulant weight matrices

4: $\frac{\partial C}{\partial W^q}$ = **PoTBackward**$(\frac{\partial C}{\partial o_t}, W^q, a^t)$ //Backward propagation with the quantized weights

5: W^{t+1} = **Update**$(W^q, \frac{\partial C}{\partial W^q})$ // Update the original weights with respect to W^q

6: η^{t+1} = **Update**$(\eta^t, \frac{\partial \eta^t}{\partial t})$ // Update the learning rate

4. Efficient Hardware Architecture

As shown in Figure 2, the weight matrix of an arbitrary FC layer can be represented by $P_{row} \times P_{column}$ circulant matrices, where P_{row} and P_{column} denote the row number and column number of blocks in the weight matrix, respectively. According to Equation (4), the MV in the computation of an FC layer can be partitioned into multiplications of circulant sub-matrices and sub-vectors. Each $W_{ij}x_j$ can be computed independently and then the results of $W_{ij}x_j$ in the same row can be accumulated to get the corresponding result vector y_i. All the multiplications can be replaced by shift operations by using the quantization scheme proposed in Section 3.2. To take full advantage of the quantization scheme, we do not use the FFT-based acceleration method in [24] because: (1) FFT-based method limits the quantization of weights. In [24], the accuracy is low, if the weights are quantized to 4 bits. (2) If the weights are transformed by FFT, we cannot use shift-operations to remove multiplications.

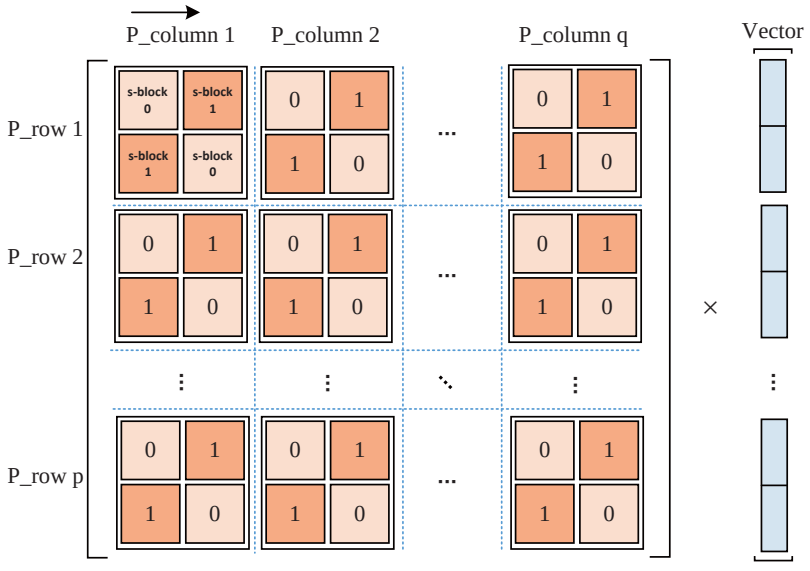

Figure 2. Computing scheme for an FC layer.

As shown in Figure 2, the weight matrix of an arbitrary FC layer can be represented by $P_{row} \times P_{column}$ circulant matrices, where P_{row} and P_{column} denote the row number and column number of blocks in the weight matrix, respectively. According to Equation (4), the MV in the computation of an FC layer can be partitioned into multiplications of circulant sub-matrices and sub-vectors. Each $W_{ij}x_j$ can be computed independently and then the results of $W_{ij}x_j$ in the same row can be accumulated

to get the corresponding result vector y_i. All the multiplications can be replaced by shift operations by using the quantization scheme proposed in Section 3.2. To take full advantage of the quantization scheme, we do not use the FFT-based acceleration method in [24] because: (1) FFT-based method limits the quantization of weights. In [24], the accuracy is low, if the weights are quantized to 4 bits. (2) If the weights are transformed by FFT, we cannot use shift-operations to remove multiplications.

4.1. Reorganization of Block-Circulant Matrix

We can directly compute the sub-matrix and sub-vector multiplications when the block size is small. However, when the block size is large, e.g. 256×256, directly computing the $W_{ij}x_j$ is inefficient. In addition, because of the variable block size, a configurable hardware architecture is required. Thus, we propose to further divide every sub-matrix into smaller sub-blocks, each of which is still a circulant matrix. Similar chessboard division method is proposed in [27] for each sub-matrix. Figure 2 shows an example of the division, where each sub-matrix is partitioned into four sub-blocks. Specifically, we use a reorganization scheme [27] in each sub-matrix to further transform it into a block pseudocirculant matrix [28,29]. The detailed transformation scheme is illustrated as follows.

We assume that a sub-matrix W_{ij} is partitioned into α^2 sub-blocks, each of which is γ by γ ($\gamma = \frac{k}{\alpha}$) short circulant matrix denoted by P_{ij}^{β}:

$$P_{ij}^{\beta} = \begin{bmatrix} w_{ij}^{\beta} & w_{ij}^{\beta+\alpha} & w_{ij}^{\beta+2\alpha} & \cdots & w_{ij}^{\beta+(\gamma-1)\alpha} \\ w_{ij}^{\beta+(\gamma-1)\alpha} & w_{ij}^{\beta} & w_{ij}^{\beta+\alpha} & \cdots & w_{ij}^{\beta+(\gamma-2)\alpha} \\ w_{ij}^{\beta+(\gamma-2)\alpha} & w_{ij}^{\beta+(\gamma-1)\alpha} & w_{ij}^{\beta} & \cdots & w_{ij}^{\beta+(\gamma-3)\alpha} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{ij}^{\beta+\alpha} & w_{ij}^{\beta+2\alpha} & w_{ij}^{\beta+3\alpha} & \cdots & w_{ij}^{\beta} \end{bmatrix}, \tag{8}$$

where $\beta = 1, 2, 3, \ldots, \alpha$. Correspondingly, vector x_j is divided into α small vectors denoted by $X_j^1, X_j^2, X_j^3, \ldots, X_j^{\alpha}$, where $X_j^{\beta} = (x_j^{\beta}, x_j^{\beta+\alpha}, x_j^{\beta+2\alpha}, \ldots, x_j^{\beta+(\gamma-1)\alpha})$, $\beta = 1, 2, \ldots, \alpha$.

The circulant matrix vector multiplication (C-MV) $y_i = W_{ij}x_j$ can be represented as follows

$$\begin{bmatrix} Y_i^1 \\ Y_i^2 \\ Y_i^3 \\ \vdots \\ Y_i^{\alpha} \end{bmatrix} = \begin{bmatrix} P_{ij}^1 & P_{ij}^2 & P_{ij}^3 & \cdots & P_{ij}^{\alpha} \\ S_{\gamma}P_{ij}^{\alpha} & P_{ij}^1 & P_{ij}^2 & \cdots & P_{ij}^{\alpha-1} \\ S_{\gamma}P_{ij}^{\alpha-1} & S_{\gamma}P_{ij}^{\alpha} & P_{ij}^1 & \cdots & P_{ij}^{\alpha-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ S_{\gamma}P_{ij}^2 & S_{\gamma}P_{ij}^3 & S_{\gamma}P_{ij}^4 & \cdots & P_{ij}^1 \end{bmatrix} \begin{bmatrix} X_j^1 \\ X_j^2 \\ X_j^3 \\ \vdots \\ X_j^{\alpha} \end{bmatrix}, \tag{9}$$

where S_{γ} is the cyclic shift operator matrix

$$S_{\gamma} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}. \tag{10}$$

Then, we can compute $y_i = (Y_i^1, Y_i^2, Y_i^3, \ldots, Y_i^{\alpha})^T$ in parallel and $Y_i^{\beta} = (y_i^{\beta}, y_i^{\beta+\alpha}, y_i^{\beta+2\alpha}, \ldots, y_i^{\beta+(\gamma-1)\alpha})$, $\beta = 1, 2, \ldots, \alpha$. For example, Y_i^1 is given by

$$Y_i^1 = P_{ij}^1 X_j^1 + P_{ij}^2 X_j^2 + P_{ij}^3 X_j^3 + \ldots + P_{ij}^{\alpha} X_j^{\alpha}. \tag{11}$$

Each $P_{ij}^{\beta} X_j^{\beta}$ in the polynomial in Equation (11) is a C-MV and can be computed separately.

4.2. Architecture of Block-MV

Using the reorganization scheme in Section 4.1, we can transform a sub-matrix into smaller sub-blocks. By observing the block size, we set the 16×16 as the basic size of a sub-block, because all block sizes in our experiment were the multiples of 16. If the block size is 16×16, we can directly compute the C-MV. If the block size is larger than 16, the matrix can be naturally reorganized into several 16×16 sub-blocks. As shown in Figure 3, the Block-MV (B-MV) module is the key computing module for processing the short C-MV, $P_{ij}^{\beta} X_j^{\beta}$. Suppose the block size $k = 64$ and $\alpha = \frac{k}{16} = 4$. Then, $W_{ij} x_j$ can be represented as follows

$$\begin{bmatrix} Y_i^1 \\ Y_i^2 \\ Y_i^3 \\ Y_i^4 \end{bmatrix} = \begin{bmatrix} P_{ij}^1 & P_{ij}^2 & P_{ij}^3 & P_{ij}^4 \\ S_\gamma P_{ij}^4 & P_{ij}^1 & P_{ij}^2 & P_{ij}^3 \\ S_\gamma P_{ij}^3 & S_\gamma P_{ij}^4 & P_{ij}^1 & P_{ij}^2 \\ S_\gamma P_{ij}^2 & S_\gamma P_{ij}^3 & S_\gamma P_{ij}^4 & P_{ij}^1 \end{bmatrix} \begin{bmatrix} X_j^1 \\ X_j^2 \\ X_j^3 \\ X_j^4 \end{bmatrix}. \tag{12}$$

When computing Y_i^1, $X_j^1 = (x_j^1, x_j^5, x_j^9, \ldots, x_j^{61})$ and the primitive vector of P_{ij}^1, $p_{ij}^1 = (w_{ij}^1, w_{ij}^5, w_{ij}^9, \ldots, w_{ij}^{61})$ will first be fetched. Then, the primitive vector will be shifted to form the other rows in P_{ij}^1. R_0 and C_0 are parameters to denote the row and column of each sub-block. If $R_0 > C_0$, p_{ij}^1 needs an additional left shift to implement the function of S_γ. Then, the activation vector and corresponding weight vectors will be sent to 16 row processing elements (R-PEs) to compute dot products in parallel.

When P_{ij}^1 is computed, X_j^2 and p_{ij}^2 will be prepared by a read controller (R-ctrl). After the calculation is finished, $P_{ij}^2 X_j^2$ will be computed in the B-MV and the result will be accumulated in R-PEs. Next, $P_{ij}^3 X_j^3$ and $P_{ij}^4 X_j^4$ will be computed successively. When computation of C-MVs in the first row is finished, the result will be sent to an accumulator module. It is the same for computing Y_i^2, Y_i^3, and Y_i^4 in the B-MV. Finally, when the B-MV finishes all the C-MVs of one sub-matrix, it will start the processing for the following sub-matrix. With the parallel processing of R-PEs, a sub-block can be processed in 1 clock, so α^2 clocks are required to compute a sub-matrix.

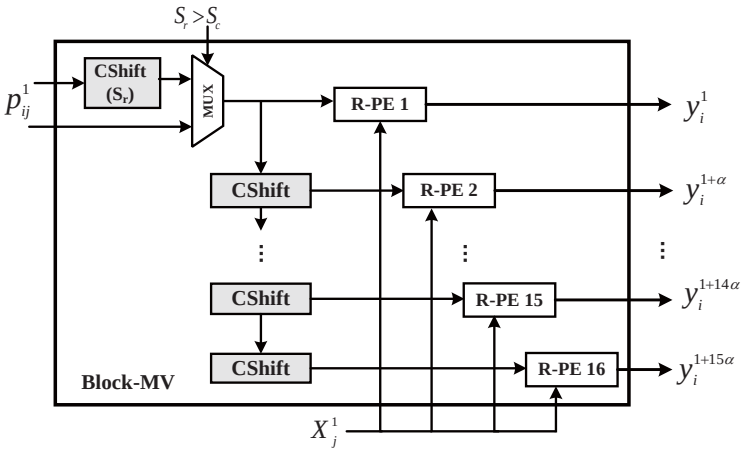

Figure 3. Architecture of Block-MV.

4.3. Row Processing Element

As shown in Figure 4, R-PEs are basic processing elements in the B-MV and each R-PE can compute m MACs in parallel. The mathematical representation of an R-PE's output is

$$P_{sum} = \begin{cases} \sum_{i=1}^{m} x[i]w[i] & \text{if reset} = 0 \\ P_{sum} + \sum_{i=1}^{k} x[i]w[i] & \text{otherwise} \end{cases} \tag{13}$$

where $w[i]$ is a weight parameter and $x[i]$ is an element of an activation vector. Since the multiplications are replaced by shift operations, the element-wise multiplication of $w[i]$ and $x[i]$ is implemented with a basic shift unit (BSU) instead of a multiplier. The BSU performs shift operations on $x[i]$ according to the weight indices. As values of the weights are limited in $[-1, 1]$, only the left-shift operations are required. In a weight index, the highest bit denoted by w_{sign} is the sign bit, and the other bits in the index are the shift bits, w_{shift}. The value of w_{shift} determines the shift bits performed on $x[i]$. Table 1 shows the shift operations of a BSU with 4-bit indices.

After $\sum_{i=1}^{m} x[i]w[i]$ is computed by the adder trees, the partial sum is stored in a register and accumulated every clock cycle. The reset signal in Figure 4 works when current partial sum is sent to the accumulator module for further processing.

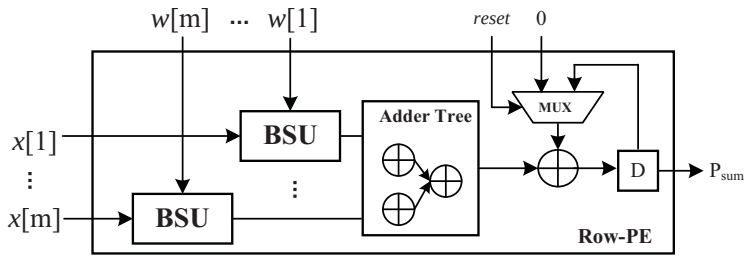

Figure 4. Architecture of row processing element.

Table 1. Weight indices and shift operations in a BSU.

w_{sign}	w_{shift}	Weight Value	Shift Bits
0	000	0	-
0	001	1/2	1
0	010	1/4	2
0	011	1/8	3
0	100	1/16	4
0	101	1/32	5
0	110	1/64	6
0	111	1	-
1	000	-	-
1	001	−1/2	1
1	010	−1/4	2
1	011	−1/8	3
1	100	−1/16	4
1	101	−1/32	5
1	110	−1/64	6
1	111	−1	-

4.4. Configurable Hardware Accelerator Architecture

The overall architecture of BPCA is shown in Figure 5. To fit for the adjustable block size and various network sizes of FC layers, BPCA is designed to provide flexibility for different needs. In general, designing a configurable accelerator involves the considerations of several factors: P_{size}, the block size of one sub-matrix; P_{row}, the row number of sub-matrices in a weight matrix; P_{column}, the column number of sub-matrices in a weight matrix; and ACC_{num}, the number of accumulators

in the Accumulator module. According to the configuration parameters, the configurable controller (Config-ctrl) controls the computing of other components in BPCA. Weights, bias, and input activations are stored in three different SRAMs, which are W-RAM, B-RAM and A-RAM in Figure 5, respectively. The weights and input vectors are sent to the B-MV by the read-controller (Read-ctrl) according to different P_{size}. The output vectors of all the sub-matrices in one row will be accumulated in the Accumulator module. In the following, we describe the configurable scheme in detail.

For different FC networks, the BPCA requires configurable parameters including P_{size}, P_{row} and P_{column}. As described in Section 4.2, the Block-MV can process a fixed-sized sub-block for one time. According to P_{size}, the Read-ctrl feeds the 16-by-16 sub-blocks and corresponding activations to the Block-MV for processing. When all sub-blocks in one sub-matrix are processed, the Config-ctrl will move on to the calculation of the next sub-matrix in the same sub-matrix row. As shown in Figure 2, BPCA performs a row-wise computing scheme. Thus, according to P_{column}, the Config-ctrl needs to judge whether current sub-matrix is the last one in current sub-matrix row. If the sub-matrix is the last one, the P_{sum} in the Accumulator module will be sent to the Bias-ReLU module for further processing, and then the final result vector of one sub-matrix row will be output. Because the size of a sub-matrix is adjustable, ACC_{num} is determined by the biggest number of P_{size} that BPCA supports. Since the size of output vector of each sub-matrix row is changeable, the Config-ctrl selects P_{size} accumulators in the Accumulator module in every computing procedure.

As the main computing element, B-MV determines the processing time of an FC layer and the system throughput. In addition, the proposed hardware architecture is scalable, and the system throughput can be improved by adding more B-MVs.

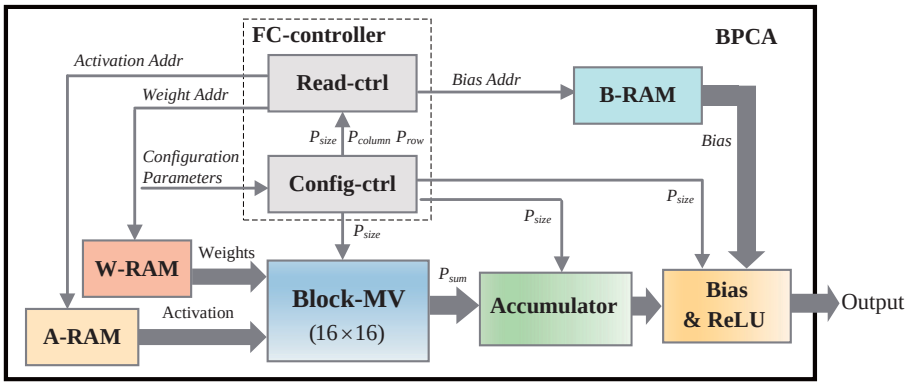

Figure 5. Hardware accelerator architecture for FC layers.

5. Experiments of The Proposed Compression Method

We verified the effectiveness of the proposed method on three standard datasets: MNIST, CIFAR-10, and ImageNet. We compared the accuracy and storage cost of the compressed neural network models and the original neural network models. Since the effectiveness of applying block-circulant matrices in FC layers has been proven in [24,25], we pay more attention to the effectiveness of power-of-two quantization on the block-circulant matrix-based FC layers. To test quantization precision's influence on prediction accuracy, we selected different n_1 for S_n and quantized the weights with different quantization precision. Notably, the aim of our experiment was not to seek the state-of-the-art results on these datasets, thus the accuracy of the original neural networks was only used as a baseline for a fair comparison with the compressed neural networks.

5.1. Experiments on MNIST

MNIST is a digit dataset that contains 28 × 28 grey-scale images of ten handwritten digits (0 to 9). In MNIST, there are 60,000 images for training and 10,000 images for testing. The baseline model that we used is a three-layer DNN denoted by Model-A. The matrix sizes and parameters of the FC layers

in Model-A are shown in Table 2. We tested the proposed compression methods on Model-A and the training results are given in Table 3. We first applied block-circulant matrices ($k = 16$) in FC1 and FC2 layers. Then, we quantized the weights under different quantization precision. When $n_1 = -2$, the quantization precision is $1/4$ and weights can be represented with 3 bits. Thus, quantization provides 10.7 times compression ratio (compared with 32-bit floating point). Other quantization precision requires 4-bit indices and the storage cost can be saved by eight times. We can see that quantization precision has little effect on the prediction accuracy. As a result, block size of 16 and 3-bit quantization can compress the FC layers by 170 times with 1% accuracy loss.

Table 2. FC Layers in the models for MNIST, CIFAR, and ImageNet.

Model-A			Model-B			Model-C		
Layer	**Matrix Size**	**Parameters**	**Layer**	**Matrix Size**	**Parameters**	**Layer**	**Matrix Size**	**Parameters**
FC1	784×2048	6.125 MB	FC4	8192×1024	32 MB	FC7	9216×4096	144 MB
FC2	2048×1024	8 MB	FC5	1024×1024	128 KB	FC8	4096×4096	64 MB
FC3	1024×10	40 KB	FC6	1024×10	40 KB	FC9	4096×1000	15.625 MB

Table 3. Experiment results of Model-A for MNIST.

Block Size (k)	Quantization Precision	Weight Bits	Compression Ratio (%)	Accuracy (%)	Parameters
-	-	32	baseline	98.47	14.125 MB
16	-	32	6.25 (16×)	97.46	904 KB
16	1/64	4	0.78 (128×)	97.58	113 KB
16	1/16	4	0.78 (128×)	97.35	113 KB
16	1/4	3	0.59 (171×)	97.06	84.6 KB
64	-	32	1.56 (64×)	94.03	226 KB
64	1/64	4	0.20 (512×)	93.96	28.3 KB
64	1/16	4	0.20 (512×)	92.75	28.3 KB
64	1/4	3	0.15 (683×)	87.83	21.2 KB

5.2. Experiments on CIFAR

CIFAR-10 dataset contains 60,000 natural 32×32 RGB images covering 10-class objects. There are 50,000 images for training and 10,000 images for testing. In our test, the network consisted of six convolutional layers and three fully-connected layers [18]. ReLU was used as the activation function. The proposed compression method was used in FC4 and FC5 layers. We also selected different n_1 to test the accuracies under different quantization precision, which is shown in Table 4. There is negligible accuracy loss when weights are quantized into 3 or 4 bits. Thus, the FC layers can be compressed by 2731× with 0.8% accuracy loss.

Table 4. Experiment results of Model-B for CIFAR-10.

Block Size (k)	Quantization Precision	Weight Bits	Compression Ratio (%)	Accuracy (%)	Parameters
-	-	32	baseline	92.00	32.125 MB
16	-	32	0.16 (16×)	92.60	2 MB
16	1/64	4	0.78 (128×)	91.72	257 KB
16	1/16	4	0.78 (128×)	91.80	257 KB
16	1/4	3	0.59 (171×)	91.58	192.4 KB
256	-	32	0.39 (256×)	92.60	125.6 KB
256	1/64	4	0.16 (2048×)	91.37	16.1 KB
256	1/16	4	0.16 (2048×)	91.66	16.1 KB
256	1/4	3	0.15 (2731×)	91.20	12 KB

5.3. Experiments on ImageNet

We further evaluated the effectiveness of the proposed compression methods on the ImageNet ILSVRC-2012 dataset, which contains images of 1000 categories of objects. There are roughly 1000 images in each of the 1000 considered categories. AlexNet [5] was used as the baseline model denoted by Model-C, which consisted of five convolutional layers, two FC layers and one final softmax layer. The Top-1 accuracy and Top-5 accuracy of Model-C are 56.257% and 79.018%, respectively. The FC layers of Model-C are shown in Table 2. We applied block-circulant weight matrix in FC7, FC8 and FC9 layers with 16×16 sub-matrices. Then, the weights were quantized with 4 bits. The length of output vector in FC9 was not a multiple of 16, thus we added eight rows at the end of the weight matrix to make it a 4096×1008 matrix. Then, the network could be trained as normal, but the eight additional outputs were dropped to get the results of 1000 categories that we wanted. As shown in Table 5, the network can be compressed by 128 times with 1.8% accuracy loss. When power-of-two quantization is used in large scale networks, there is more accuracy loss compared with small networks.

Table 5. Experiment results of Model-C for ImageNet.

Block Size (k)	Quantization Precision	Weight Bits	Compression Ratio (%)	Accuracy (%) (Top-1)	Accuracy (%) (Top-5)	Parameters
-	-	32	baseline	56.257	79.018	223.625 MB
16	-	32	1.6 (16×)	52.514	77.866	13.977 MB
16	1/64	4	0.78 (128×)	49.835	76.026	1.747 MB
16	1/16	4	0.78 (128×)	49.346	75.449	1.747 MB

5.4. Result Analysis

The power-of-two quantization scheme is effective when it is combined with block-circulant matrix-based weight representation on the three models. In general, the accuracy of a network is dominated by the block size of weight matrices. To achieve a trade-off between accuracy and compression ratio, the block size of each layer should be adjusted carefully. Quantization causes minimal accuracy loss on MNIST, CIFAR-10 and ImageNet, when the weights are quantized into 4 bits with precision 1/64.

6. Hardware Implementation Results

We evaluated the performance of the hardware architecture proposed in Section 4. The RTL of the design was implemented in Verilog and synthesized by using the Synopsys Design Compiler (DC) under the TSMC 40 nm CMOS technology. We measured the area, power, and critical path delay of the accelerator. Benchmarks were obtained from Models A–C.

6.1. Evaluation Results and Analysis

The synthesis results of the top architecture are presented in Table 6. A frequency of 800 MHz has been achieved. In our design, the basic size of a sub-block can be processed is 16×16. One B-MV containing 16 R-PEs is used for processing sub-blocks. The synthesis results of the R-PE are presented in Table 6.

Table 6. Synthesis results of the proposed hardware architecture (logic part) and row processing element.

Features	Top-Level Architecture	Row Processing Element
Technology	TSMC 40 nm	TSMC 40 nm
Clock	800 MHz	800 MHz
Core Voltage	1 V	1 V
Area	0.16 mm^2	6944 μm^2
# of Combinational Cells	5440	3540
# of Sequential Cells	706	258
Power	52.96 mW	2.1487 mW

The on-chip SRAM consisted of W-RAM, A-RAM, and B-RAM. Notably, since the accelerator needed to process different scale networks, the SRAM sizes were selected to suitable for large scale FC layers. The on-chip SRAM was modeled using Cacti [30] under 45 nm process. In our design, 16 bits and 4 bits were used to represent activations and weights, respectively. The A-RAM was 9216×16 bit $=$ 18 KB for storing activations, where the maximum input length of activations was 9216. The width of W-RAM was 64 bits and the size of W-RAM aws 1.747 MB. Thus, all compressed weights in FC layers of AlexNet could be stored in W-RAM. In addition, we used 16 bits to represent biases and B-RAM was 4096×16 bit $=$ 8 KB, where the maximum number of biases was 4096.

The power/area breakdown of the accelerator is presented in Table 7. BPCA occupies 12.72 mm^2, where the memory dominates the chip area with 98.87% of the total area. The logic part of BPCA only occupies 0.16 mm^2. The total power consumption is 77.09 mW and the memory consumes 24.13 mW.

Table 7. Synthesis results of the breakdown by module (LINE 2-3) of the accelerator. The critical path is 1 ns.

Module	Area (mm^2)	(%)	Power (mW)	(%)
Total	12.88	-	77.09	-
B-MV	0.1103	0.86%	30.27	39.27%
FC-controller	0.0378	0.29%	18.37	23.83%
Accumulator	0.0086	0.067%	3.09	4.0%
Relu,Bias	0.0033	0.026%	1.23	1.6%
On-Chip SRAM (45 nm)	12.72	98.76%	24.13	31.30%

6.1.1. Performance

To measure the performance of BPCA, we defined throughput (T_n) as the input numbers per seconds. T_n is dominated by the B-MV and is given by

$$\begin{aligned} T_n &= \frac{a \times b}{P_{size}} / ((\frac{P_{size}}{\gamma})^2 \times P_{row} \times P_{colomn} \times \frac{1}{f_{clk}}) \\ &= \frac{\gamma^2 f_{clk}}{P_{size}}. \end{aligned} \tag{14}$$

In our design, γ is 16 and f_{clk} = 800 MHz, thus $T_n = \frac{256}{P_{size}}$ numbers/s. The bigger the P_{size} is, the smaller the input throughput is, which means less memory access is required. As for the computing performance, the giga operations per second (GOPS) is 409.6 GOPS corresponding to an uncompressed layer.

The computation time to process one $a \times b$ weight matrix in the steady stage is

$$t_l = \frac{a \times b}{P_{size}} / T_n = \frac{ab}{\gamma^2 f_{clk}}. \quad (15)$$

Nine pipeline stages are introduced in the design, so the actual runtime of one layer requires an additional nine clock cycles.

We selected benchmarks from Model-A, Model-B, and Model-C to evaluate performance of the accelerator. Layers A–E had different matrix sizes and block sizes. The configuration parameters and computation time are shown in Table 8. It takes 184.32 µs to process layer C, which is the largest FC layer in AlexNet. To process middle scale layers, such as Layers A and B, only a few microseconds are consumed.

Table 8. Benchmarks from three different DNN models.

Layer Number	Matrix Size (a,b)	P_{size}	P_{column}	P_{row}	Computation Time (µs)
A	768,2048	64	12	32	7.68
B	1024,1024	256	4	4	5.12
C	9216,4096	16	576	256	184.32
D	4096,4096	16	256	256	81.92
E	4096,1000	16	256	63	20.16

6.1.2. Flexibility and Scalability

BPCA can fit for different scale FC layers with adjustable compression ratios. In this experiment, the P_{size} could be configured to 16, 32, 64, 128, and 256. The basic sub-block size, which is processed by a B-MV, can be changed according to different requirements.

To fit for larger weight matrices, BPCA can be scaled up by adding more B-PEs. As shown in Figure 6, a large weight matrix can be divided into two parts and the corresponding parameters are stored in different weight RAMs. The two-part weight matrices can be processed by two B-MVs in parallel. To further improve the system throughput, more B-MVs can be used in the system.

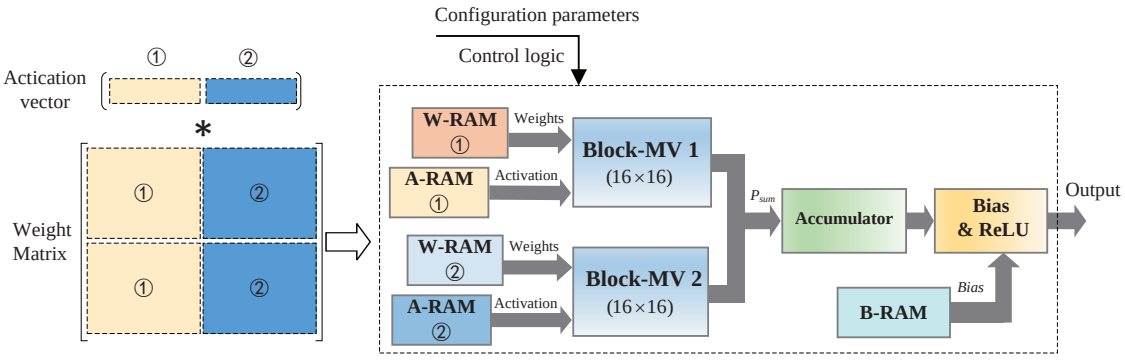

Figure 6. The scalability of the accelerator.

6.2. Comparisons with Related Works

Based on FPGA or ASIC platforms, numerous works on DNN accelerators have been proposed. We compared BPCA with representative state-of-the-art ASIC development and synthesis results. Table 9 shows the comparisons with accelerators focusing on compressed networks or uncompressed networks.

Table 9. Comparisons with state-of-the-Art DNN accelerators.

	EIE (64PE) ISCA 2016 [12]	**DNPU ISSCC 2017 [31]**	**This Work**
Target DNN	FCL	CNN/FCL/RNN	FCL
Technology	45 nm	65 nm	40 nm
Clock (MHz)	800	200	800
On-chip SRAM (MB)	10.125	0.28	1.772
Voltage (V)	-	1.1	1
Area (mm^2)	40.8	16	12.72
Power (mW)	590	21(FCN/RNN)	77.09
Peak Performance (GOPS)	102@4b	25@4b (FCN/RNN)	409.6@4b
Quantization (bits)	4	4-7 (FCN/RNN)	4
Energy efficiency (TOPS/W)	0.172@4b	1.1 (FCN/RNN)	5.3@4b
Area efficiency (GOPS/mm^2)	2.5	-	31.8

EIE [12] is a representative accelerator for processing sparse DNNs. EIE is mainly composed of customized PEs focusing on sparse matrix-vector multiplications. The computations of a network are partitioned for different PEs to perform and every PE stores a partition of the network in local SRAM. The FC layers of AlexNet can fit into 64 PEs in EIE. Using the same benchmark, AlexNet, we compared our work with EIE composed of 64 PEs.

In one PE of EIE, there are on-chip memory in sparse matrix read unit, activation read/write and pointer read unit for decoding the compressed weights. The memory takes 93.22% of the PE's area. The arithmetic unit in a PE performs multiply-accumulate operations by using one multiplier and one adder. Running at 800 MHz, the logic part of one PE takes an area of 43,258 μm^2, resulting in area efficiency of 37 GOPS/mm^2. The basic processing element in BPCA is R-PE. One R-PE can perform 16 multiply-accumulate operations in parallel using 16 adders and no multipliers. Because there are no additional decoding process, the area of an R-PE is dominated by computing units. As shown in Table 7, at 800 MHz, an R-PE takes 6944 μm^2, resulting in area efficiency of 3.7 TOPS/mm^2. Without multipliers and decoding units, R-PE achieves significantly higher area efficiency than the PE of EIE.

In EIE, the sparse weight matrices, which are encoded in compressed sparse column (CSC) format, can all be stored in on-chip SRAM. With the encoding format, additional 4-bit indices and pointer vectors are required for the decoding of compressed weight matrices. The weight SRAMs of 64 PEs is 8 MB and can hold 4-bit weights and 4-bit indices of AlexNet's FC layers with at least 8× weight sparsity. Each PE has a pointer SRAM and the total SRAM capacity for pointers is 2 MB. In addition, the activation SRAM in each PE is 2 KB and 64 PEs use 128 KB SRAM for activations. The on-chip SRAM dominates the area in EIE and takes 93.22% of the total area of EIE. At 800 MHz, EIE (64 PEs) can yield a performance of 102 GOPS and the area efficiency can achieve 2.5 GOPS/mm^2 (including SRAM). In BPCA, the accelerator also stores the FC layers of AlexNet in SRAM. Using 16 × 16 block-circulant matrix, the weight matrix can be compressed by 16×. By using power-of-two quantization, the weights can be represented with 4 bits. Storing the block-circulant matrix-based layers does not require indices or pointers such as storing the sparse matrices in EIE. Without using additional indices, 2× storage usage can be reduced. In addition, 2 MB storage can be saved by using no pointers. In the accelerator, the weight SRAM is 1.772 MB and the activation SRAM is 18 KB. The on-chip SRAM takes 98.76% of BPCA's area. For the same benchmark, AlexNet, BPCA can achieves an area efficiency of 31.8 GOPS/mm^2, which is 13× of EIE.

Our work also outperforms EIE in energy efficiency. In one PE of EIE, the memory consumes 59.15% of the total power. Sixty-four PEs use 10.125-MB SRAM, which consumes 348 mW. Compared with EIE, we use less SRAM and save 14× power of memory. The logic part of one EIE's PE consumes 3.74 mW, but the power of R-PE is only 57.5% of EIE's PE. As shown in Table 9, the energy efficiency of BPCA can achieve 5.3 TOPS/W, which is 31× of EIE.

DNPU [31] is configurable heterogeneous accelerators supporting CNNs, FCLs, and RNNs. We mainly compare BPCA with the part of DNPU on FC layers. The weights in DNPU can be quantized into 4–7 bits using dynamic fixed point. However, without other compression method, the parameters of DNPU are still required to be fetched from external memory (DRAM). By using the proposed compression method, our work can save much more DRAM access than DNPU. Besides, one compressed FC layer can be stored in on-chip SRAM, which consumes less energy than storing parameters in off-chip memory. In DNPU, quantization table-based matrix multiplication is used and 99% of the 16-bit fixed-point multiplications can be avoided. At 200 MHz, the energy efficiency of DNPU can achieve 1.1 TOPS/W. BPCA also does not use any multipliers and can achieve 5 TOPS/W at 800 MHz. Compared with DNPU, our work achieves competitive performance and energy efficiency with less memory access.

7. Conclusions

In this paper, we present an effective compression method and an efficient configurable hardware architecture for fully-connected layers. Block-circulant matrices and power-of-two quantization are employed in this approach to compress the DNNs. Block-circulant matrices provide an adjustable compression ratio for weight matrices by changing the block size. Using power-of-two quantization, we can represent the weights with low precision and reduce computational complexity by replacing the multiplications with shift operations. The approach was applied to three classic datasets and provided remarkable compression ratios. For MINIST, the FC layers can be compressed by 171× with 1% accuracy loss. The results on CIFAR-10 show that the FC layers can be compressed by 2731× with negligible accuracy loss. Our experiment on ImageNet compresses the weight storage of AlexNet by 128× with 1.8% accuracy loss. The proposed compression method outperforms pruning-based method in compression ratio and avoids irregular computations of pruned networks.

To process the compressed neural network efficiently, we develop a configurable, area-efficient and energy-efficient hardware architecture, BPCA. BPCA can be configurable to process FC layers with different compression ratios. Without using multipliers, a lot of chip area can be saved in R-PEs and system throughput can be significantly improved. The proposed design is implemented under the TSMC 40 nm CMOS technology. Using 1.772-MB SRAM, the accelerator can do 409.6 GOPS in an area of 12.72 mm^2 and dissipates 77.09 mW. Our work outperformed pruning-based accelerator, EIE [12], in energy efficiency by 31× and area efficiency by 13×.

8. Discussion

In this paper, the proposed compression scheme targets FC layers. In the experiments for Model B and Model C, we only applied the proposed compression method in FC layers and the accuracy of Model C has already shown some reduction on Imagenet dataset. Since FC layer has a lot of redundancy, the quantization may cause minimal accuracy loss. As for convolutional layers, which are sensitive to quantization, the proposed method may cause much accuracy loss. Thus, the quantization scheme should be examined carefully in block-circulant matrix-based CNNs. In our future work, we will evaluate the proposed method on convolutional layers and explore better quantization schemes on block-circulant matrix-based CNNs.

Author Contributions: Conceptualization, H.P. and Z.Q.; methodology, Z.Q. and D.Z.; software, D.Z.; validation, X.Z. and X.C.; formal analysis, Z.Q.; investigation, X.Z.; resources, H.P. and L.L.; data curation, X.Z. and X.C.; writing—original draft preparation, Z.Q.; writing—review and editing, Z.L. and Y.S.; visualization, Z.Q.; supervision, H.P. and Y.G.; project administration, Q.S.; and funding acquisition, H.P., Y.S. and Y.G.

Funding: This research was funded in part by the National Nature Science Foundation of China under Grant No. 61376075 and 41412020201, in part by the key Research and Development Program of Jiangsu Province under Grant No. BE2015153, and in part by the Priority Academic Program Development of Jiangsu Higher Education Institutions(PAPD).” “The APC was funded by the National Nature Science Foundation of China under Grant No. 61376075”.

Acknowledgments: This work was supported in part by the National Nature Science Foundation of China under Grant No. 61376075 and 41412020201, in part by the key Research and Development Program of Jiangsu Province under Grant No. BE2015153, and in part by the Priority Academic Program Development of Jiangsu Higher Education Institutions(PAPD).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Tang, Z.L.; Li, S.M.; Yu, L.J. Implementation of Deep Learning-Based Automatic Modulation Classifier on FPGA SDR Platform. *Electronics* **2018**, *7*, 122. [CrossRef]
2. Liu, X.; Yang, T.; Li, J. Real-Time Ground Vehicle Detection in Aerial Infrared Imagery Based on Convolutional Neural Network. *Electronics* **2018**, *7*, 78. [CrossRef]
3. Wang, X.; Hua, X.; Xiao, F.; Li, Y.; Hu, X.; Sun, P. Multi-Object Detection in Traffic Scenes Based on Improved SSD. *Electronics* **2018**, *7*, 302. [CrossRef]
4. Nguyen, T.V.; Mirza, B. Dual-layer kernel extreme learning machine for action recognition. *Neurocomputing* **2017**, *260*, 123–130. [CrossRef]
5. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Proceedings of the International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–8 December 2012; pp. 1097–1105.
6. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. In Proceedings of the International Conference on Learning Representations, Banff, AB, Canada, 14–16 April 2014; pp. 1–14.
7. Mirza, B.; Kok, S.; Dong, F. Multi-layer online sequential extreme learning machine for image classification. In *Proceedings of ELM-2015 Volume 1*; Springer: Berlin, Germany, 2016; pp. 39–49.
8. Mikolov, T.; Karafiát, M.; Burget, L.; Černockỳ, J.; Khudanpur, S. Recurrent neural network based language model. In Proceedings of the International Symposium on Computer Architecture, Saint-Malo, France, 19–23 June 2010; pp. 1045–1048.
9. Dominguez-Sanchez, A.; Cazorla, M.; Orts-Escolano, S. A New Dataset and Performance Evaluation of a Region-Based CNN for Urban Object Detection. *Electronics* **2018**, *7*, 301. [CrossRef]
10. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* **2017**, *105*, 2295–2329. [CrossRef]
11. Dean, J.; Corrado, G.S.; Monga, R.; Chen, K.; Devin, M.; Le, Q.V.; Mao, M.Z.; Ranzato, M.; Senior, A.; Tucker, P.; et al. Large Scale Distributed Deep Networks. In Proceedings of the International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–8 December 2012; pp. 1223–1231.
12. Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M.A.; Dally, W.J. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In Proceedings of the International Symposium on Computer Architecture, Seoul, Korea, 18–22 June 2016; pp. 243–254. [CrossRef]
13. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In Proceedings of the International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016.
14. Courbariaux, M.; Bengio, Y. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or −1. *arXiv* **2016**, arXiv:1602.02830.
15. Wu, J.; Leng, C.; Wang, Y.; Hu, Q.; Cheng, J. Quantized convolutional neural networks for mobile devices. In Proceedings of the International Conference on Computer Vision and Pattern Recogintion, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 4820–4828.
16. Choi, Y.; El-Khamy, M.; Lee, J. Towards the limit of network quantization. In Proceedings of the International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016.
17. Courbariaux, M.; Bengio, Y.; David, J.P. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In Proceedings of the International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 3123–3131.

18. Zhao, R.; Song, W.; Zhang, W.; Xing, T.; Lin, J.H.; Srivastava, M.; Gupta, R.; Zhang, Z. Accelerating binarized convolutional neural networks with software-programmable fpgas. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; ACM: New York, NY, USA, 2017; pp. 15–24.
19. Ando, K.; Ueyoshi, K.; Orimo, K.; Yonekawa, H.; Sato, S.; Nakahara, H.; Takamaeda-Yamazaki, S.; Ikebe, M.; Asai, T.; Kuroda, T.; et al. BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W. *IEEE J. Solid-State Circuits* **2018**, *53*, 983–994. [CrossRef]
20. Wang, Y.; Lin, J.; Wang, Z. An energy-efficient architecture for binary weight convolutional neural networks. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2018**, *26*, 280–293. [CrossRef]
21. Cheng, Y.; Felix, X.Y.; Feris, R.S.; Kumar, S.; Choudhary, A.; Chang, S.F. Fast neural networks with circulant projections. *arXiv* **2015**, arXiv:1502.03436.
22. Cheng, Y.; Yu, F.X.; Feris, R.S.; Kumar, S.; Choudhary, A.; Chang, S.F. An exploration of parameter redundancy in deep networks with circulant projections. In Proceedings of the International Conference on Computer Vision and Pattern Recogintion, Boston, MA, USA, 7–12 June 2015; pp. 2857–2865.
23. Sindhwani, V.; Sainath, T.; Kumar, S. Structured transforms for small-footprint deep learning. In Proceedings of the International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 3088–3096.
24. Ding, C.; Liao, S.; Wang, Y.; Li, Z.; Liu, N.; Zhuo, Y.; Wang, C.; Qian, X.; Bai, Y.; Yuan, G.; et al. Circnn: Accelerating and compressing deep neural networks using block-circulant weight matrices. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, MA, USA, 14–18 October 2017; ACM: New York, NY, USA, 2017; pp. 395–408.
25. Liao, S.; Li, Z.; Lin, X.; Qiu, Q.; Wang, Y.; Yuan, B. Energy-efficient, high-performance, highly-compressed deep neural network design using block-circulant matrices. In Proceedings of the 36th International Conference on Computer-Aided Design, Irvine, CA, USA, 13–16 November 2017; IEEE Press: Piscataway, NJ, USA, 2017; pp. 458–465.
26. Zhao, L.; Liao, S.; Wang, Y.; Tang, J.; Yuan, B. Theoretical Properties for Neural Networks with Weight Matrices of Low Displacement Rank. *CoRR* **2017**. Available online: http://xxx.lanl.gov/abs/1703.00144 (accessed on).
27. Wang, Z.; Lin, J.; Wang, Z. Accelerating Recurrent Neural Networks: A Memory-Efficient Approach. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2017**, *25*, 2763–2775. [CrossRef]
28. Teixeira, M.; Rodriguez, Y.I. Parallel Cyclic Convolution Based on Recursive Formulations of Block Pseudocirculant Matrices. *IEEE Trans. Signal Process.* **2008**, *56*, 2755–2770. [CrossRef]
29. Teixeira, M.; Rodriguez, D. A class of fast cyclic convolution algorithms based on block pseudocirculants. *IEEE Signal Process. Lett.* **1995**, *2*, 92–94. [CrossRef]
30. Muralimanohar, N.; Balasubramonian, R.; Jouppi, N. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), Chicago, IL, USA, 1–5 December 2007; pp. 3–14. [CrossRef]
31. Shin, D.; Lee, J.; Lee, J.; Yoo, H. 14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks. In Proceedings of the 2017 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 5–9 February 2017; pp. 240–241. [CrossRef]