Structured Weight Matrices-Based Hardware Accelerators in Deep Neural Networks: FPGAs and ASICs

Caiwen Ding*[1], Ao Ren*[1], Geng Yuan[1], Xiaolong Ma[1], Jiayu Li[1], Ning Liu[1], Bo Yuan[2], Yanzhi Wang[1]

[1]Syracuse University, [2]City University of New York, City College.
{cading,aren,geyuan,xma27,jli221,nliu03,ywang393}@syr.edu,byuan@ccny.cuny.edu

ABSTRACT

Both industry and academia have extensively investigated hardware accelerations. In this work, to address the increasing demands in computational capability and memory requirement, we propose structured weight matrices (SWM)-based compression techniques for both *field programmable gate array* (FPGA) and *application-specific integrated circuit* (ASIC) implementations. In algorithm part, SWM-based framework adopts block-circulant matrices to achieve a fine-grained tradeoff between accuracy and compression ratio. The SWM-based technique can reduce computational complexity from $O(n^2)$ to $O(n \log n)$ and storage complexity from $O(n^2)$ to $O(n)$ for each layer and both training and inference phases. For FPGA implementations on deep convolutional neural networks (DCNNs), we achieve at least 152X and 72X improvement in performance and energy efficiency, respectively using the SWM-based framework, compared with the baseline of IBM TrueNorth processor under same accuracy constraints using the data set of MNIST, SVHN, and CIFAR-10. For FPGA implementations on long short term memory (LSTM) networks, the proposed SWM-based LSTM can achieve up to 21X enhancement in performance and 33.5X gains in energy efficiency compared with the baseline accelerator. For ASIC implementations, the SWM-based ASIC design exhibits impressive advantages in terms of power, throughput, and energy efficiency. Experimental results indicate that this method is greatly suitable for applying DNNs onto both FPGAs and mobile/IoT devices.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**;

KEYWORDS

Deep learning, FPGA, ASIC, Accelerator, Structured weight matrices

ACM Reference Format:
Caiwen Ding[1][1], Ao Ren[1][1], Geng Yuan[1], Xiaolong Ma[1], Jiayu Li[1], Ning Liu[1], Bo Yuan[2], Yanzhi Wang[1]. 2018. Structured Weight Matrices-Based Hardware Accelerators in Deep Neural Networks: FPGAs and ASICs. In *GLSVLSI '18: 2018 Great Lakes Symposium on VLSI, May 23–25, 2018, Chicago, IL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3194554.3194625

*Caiwen Ding and Ao Ren contributed equally to this work.

GLSVLSI '18, May 23–25, 2018, Chicago, IL, USA

ACM ISBN 978-1-4503-5724-1/18/05…$15.00
https://doi.org/10.1145/3194554.3194625

1 INTRODUCTION

Deep learning has increasingly drawn attentions in many research fields, such as speech recognition [13], computer vision [12, 18], self-driving cars [14, 36], and unmanned aircraft systems [29]. Large-scale deep neural networks (DNNs) typically consist of multiple layers, and at least millions of weight parameters for the entire model [18]. One major advantage of the larger-scale DNNs is that they extract more complex high-level features from the inputs (e.g., images/videos, speeches), and as a result, achieving a significant improvement in model accuracy [36].

On the other hand, as the size of DNNs grows continuously, there exist tremendous demands in increasing computational capability and memory requirement. Therefore, improving the performance and energy efficiency while maintaining the accuracy of DNNs becomes extremely critical. Two trends have characterized the research advance in order to achieve higher performance and energy efficiency. The first trend is *hardware acceleration*. FPGA-based accelerators have the advantage of friendly programmability and high-degree parallelism. Stochastic Computing (SC), in which all the inputs and weight values are represented as streams of random bits, has been investigated and successfully applied to hardware acceleration of DNNs [22–26, 28, 33, 34, 41]. Data-path optimization technique [8] have also been studied to map a limited number of Processing elements (PEs) on FPGA and reuse the mapped PEs by iterating data through them. On the other hand, ASIC-based implementations have been explored to further accelerate DNNs. A substantial number of high-tech companies have declared their ASIC chip designs in DNNs such as Google [15] and IBM TrueNorth [6]. In the field of academia, Eyeriss [2], EIE [10], and DaDianNao [1] mainly focus on the convolutional layers, the fully-connected layers, and the memory design/organization at the architectural level, respectively.

The second trend is *model compression* motivated by energy efficiency limitation of large DNN models. Weight pruning [11] and lower rank approximation [37] have aimed to the reduce the number of operations involved in DNNs. They achieve a parameter reduction to some extent with inconsequential accuracy degradation. However, they have brought the new challenges into DNNs such as irregular network structure caused by sparsity regularization [40], and increased training complexity caused by the additional pruning process [11] or low rank approximation step [37].

In this work, to address the limitations of existing works in model size compression and acceleration and to achieve ultra-high energy efficiency and performance for FPGA and ASIC-based hardware implementations, we propose the structured weight matrices (SWM)-based compression technique on both FPGA and ASIC implementations. The SWM-based framework adopts the general block-circulant matrices to achieve a fine-grained tradeoff between accuracy and compression ratio. For FPGA implementations on DCNNs, we achieve at least 152X and 72X improvement in performance and energy efficiency, respectively using SWM-based

framework, compared with the baseline of IBM TrueNorth processor under same accuracy constraints using the data set of MNIST, SVHN, and CIFAR-10. For FPGA implementations on LSTM networks, the proposed SWM-based method can achieve up to 21X enhancement in performance and 33.5X gains in energy efficiency compared with ESE, respectively. For ASIC implementations, the proposed SWM-based design exhibits impressive advantages in terms of power, throughput, and energy efficiency. It indicates that this method is greatly suitable for applying DNNs onto both FPGAs and mobile/IoT devices.

2 BACKGROUND OF DNNS

2.1 Deep Convolutional Neural Networks

DNN systems consist of many different architectures such as DCNNs, recurrent neural network (RNNs), and deep belief networks (DBNs). Although different network structures target at specific applications, they have the similarity in construction principle, i.e., multiple layers connected in series for feature extraction [16, 21]. DNNs are commonly made up of three-layer types: Fully-connected (FC) and convolutional layers (CONV), and pooling layers (POOL).

FC layer is the most storage-intensive layer in DNNs [10, 32] since its neurons are fully connected with neurons in previous layer. The computation of an FC layer consists of matrix-vector arithmetics followed by the activation function, described as: $\mathbf{y} = \psi(\mathbf{W}\mathbf{x} + \theta)$, where $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the weight matrix of the synapses between this FC layer (with m neurons) and its previous layer (with n neurons); $\theta \in \mathbb{R}^m$ is the bias vector; and $\psi(\cdot)$ is the activation function. The calculation of $\mathbf{W}\mathbf{x}$ dominates computational complexity because the rest has lower complexity of O(n).

CONV layer performs a multi-dimensional convolution to extract features from its inputs that will be fed into subsequent layers for extracting higher-level features. A CONV layer is associated with a set of learnable filters (or kernels) [19]. A filter-sized moving window is applied to the input feature maps, calculating the convolution of the filter and input feature maps in the moving window. In practical DNN models, the CONV layers are often associated with multiple input and multiple output feature maps. As a result, the CONV layer can be expressed in *tensor computations*: $\mathcal{Y}(x, y, p) = \sum_{i=1}^{r} \sum_{j=1}^{r} \sum_{c=1}^{C} \mathcal{F}(i, j, c, p)\mathcal{X}(x + i - 1, y + j - 1, c)$, where $\mathcal{X} \in \mathbb{R}^{W \times H \times C}$, $\mathcal{Y} \in \mathbb{R}^{(W-r+1) \times (H-r+1) \times P}$, $\mathcal{F} \in \mathbb{R}^{r \times r \times C \times P}$ represent the input, output, and weight "tensors" of the CONV layer, respectively. Here, W and H are the spatial dimensions of the input maps, C is the number of input maps, r is the size of the convolutional kernel, and P is the number of output maps.

POOL layer performs a subsampling operation on the extracted features to reduce the data dimensions and mitigate overfitting issues. Max pooling is the dominant type of pooling strategy in state-of-the-art DCNNs due to its higher overall accuracy and convergence speed [1, 2].

The majority of computations occur in CONV and FC layers, while the POOL layer has a lower computational complexity of O(n). The storage requirement of DNNs is due to the weight matrices $\mathbf{W}$'s in the FC layers and the convolutional kernels $\mathbf{F}$'s in CONV layers. As a result, the FC and CONV layers become the major research focuses for energy-efficient implementation of DNNs.

2.2 Recurrent Neural Networks

RNNs have been investigated and have many applications in natural language processing, speech recognition, and machine translation [35]. As one popular type of RNNs, long short term memory (LSTM) has been broadly studied as shown in Fig. 1 [35]. An LSTM-based RNN accepts an input sequence $\mathbb{X} = (\mathbf{x}_1; \mathbf{x}_2; \mathbf{x}_3; \ldots; \mathbf{x}_T)$ (each of $\mathbf{x}_t$ is a vector corresponding to time t) with the output sequence from last step $\mathbb{Y}^{T-1} = (\mathbf{y}_0; \mathbf{y}_1; \mathbf{y}_2; \ldots; \mathbf{y}_{T-1})$ (each of $\mathbf{y}_t$ is a vector). It computes an output sequence $\mathbb{Y} = (\mathbf{y}_1; \mathbf{y}_2; \mathbf{y}_3; \ldots; \mathbf{y}_T)$ by using the following equations iteratively from $t = 1$ to T:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ir}\mathbf{y}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i), \tag{1a}$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fr}\mathbf{y}_{t-1} + \mathbf{W}_{fc}\mathbf{c}_{t-1} + \mathbf{b}_f), \tag{1b}$$
$$\mathbf{g}_t = \sigma(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{cr}\mathbf{y}_{t-1} + \mathbf{b}_c), \tag{1c}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{g}_t \odot \mathbf{i}_t, \tag{1d}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{or}\mathbf{y}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_t + \mathbf{b}_o), \tag{1e}$$
$$\mathbf{m}_t = \mathbf{o}_t \odot \mathbf{h}(\mathbf{c}_t), \tag{1f}$$
$$\mathbf{y}_t = \mathbf{W}_{ym}\mathbf{m}_t, \tag{1g}$$

where symbols $\mathbf{i}$, $\mathbf{f}$, $\mathbf{o}$, $\mathbf{c}$, $\mathbf{m}$, and $\mathbf{y}$ represent the input gate, forget gate, output gate, cell state, cell output, and projected output, respectively. The $\odot$ operation represents element-wise multiplication, and the + operation is matrix addition. The $\mathbf{W}$ terms represent weight matrices (for instance, $\mathbf{W}_{ix}$ is the weight matrix from the input vector $\mathbf{x}_t$ to the input gate), and the $\mathbf{b}$ terms are the bias vectors. Additionally, weight matrices $\mathbf{W}_{ic}$, $\mathbf{W}_{fc}$, and $\mathbf{W}_{oc}$ are diagonal matrices for peephole connections, which can be considered as vectors during matrix-vector multiplication. Therefore, $\mathbf{W}_{ic}\mathbf{c}_{t-1}$ can be calculated using $\odot$ operation. σ is the logistic activation function and h is a self-defined activation function. In this model we use hyperpolic tangent (tanh) activation function as h.

3 STRUCTURED WEIGHT MATRIX

This section discusses the inference and training algorithms of SWM-based DNNs (e.g., [5, 39]). The advantage is two-fold: *1)* it is possible to derive a fine-grained tradeoff between accuracy and compression/acceleration by changing the block size; and *2)* the method applies to both FC and CONV layers. The theoretical foundation is also derived from [42], which shows that the "effectiveness" of SWM-based DNNs is the same compared with DNNs without compression. Experimental results in [5, 39] have demonstrated a good ratio of model compression (i.e., from 41× to 256×) with small (less than 2%) overall accuracy degradation. In the following, we discuss the inference and training algorithms for FC layer, details of the CONV layer algorithms are provided in [5].

The key idea of SWM-based FC layers is to partition the original weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ into blocks of square sub-matrices, and each sub-matrix is a circulant matrix. The illustrations are shown in Fig. 2. Let k denote the *block size* (size of each sub-matrix) and assume there are $p \times q$ blocks after partitioning $\mathbf{W}$, where $p = m \div k$ and $q = n \div k$. Then $\mathbf{W} = [\mathbf{W}_{ij}]$, $i \in \{1 \ldots p\}$, $j \in \{1 \ldots q\}$. The input $\mathbf{x}$ is also partitioned as $\mathbf{x} = [\mathbf{x}_1^T, \mathbf{x}_2^T, \ldots, \mathbf{x}_q^T]^T$. Then, the *forward propagation* of FC layer in the inference is given by (with

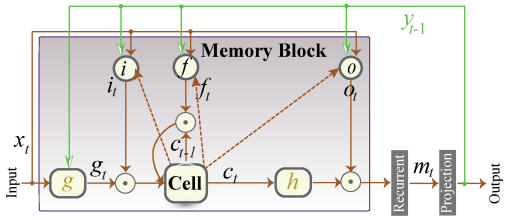

Figure 1: An example of LSTM based RNN architecture.

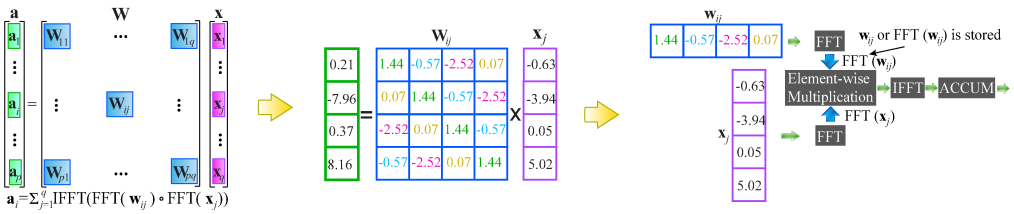

Figure 2: FFT-Based Calculation in SWM-based FC Layer.

bias and ReLU omitted for simplicity):

$$\mathbf{a} = \mathbf{W}\mathbf{x} = \begin{bmatrix} \sum_{j=1}^{q} \mathbf{W}_{1j}\mathbf{x}_j \\ \sum_{j=1}^{q} \mathbf{W}_{2j}\mathbf{x}_j \\ \cdots \\ \sum_{j=1}^{q} \mathbf{W}_{pj}\mathbf{x}_j \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \cdots \\ \mathbf{a}_p \end{bmatrix}, \tag{2}$$

where $\mathbf{a}_i \in \mathbb{R}^k$ is a column vector. Assume each circulant matrix $\mathbf{W}_{ij}$ is defined by a vector $\mathbf{w}_{ij}$, i.e., $\mathbf{w}_{ij}$ is the first row vector of $\mathbf{W}_{ij}$. According to the *circulant convolution theorem* [31], the calculation of $\mathbf{W}_{ij}\mathbf{x}_j$ can be performed as $\mathrm{IFFT}\big(\mathrm{FFT}(\mathbf{w}_{ij}) \circ \mathrm{FFT}(\mathbf{x}_j)\big)$, where $\circ$ denotes element-wise multiplications. The operation procedure is shown on the right of Fig. 2. For the inference phase, the computational complexity of this FC layer is $O(pqk \log k)$, which is equivalent to $O(n \log n)$ for small p, q values. Similarly, the storage complexity is $O(pqk)$ because only $\mathbf{w}_{ij}$ or $\mathrm{FFT}(\mathbf{w}_{ij})$ for each sub-matrix needs to be stored, which is equivalent to $O(n)$ for small p, q values. Therefore, the simultaneous acceleration and model compression are achieved.

4 MODEL COMPRESSION AND ACCURACY

To reduce the computation complexity and storage complexity, many researchers have investigated to reduce the number of weight parameters or the number of bits for weight representation. However, the compression techniques will cause the model accuracy degradation. In this section, we will discuss the trade-off between model compression and model accuracy loss of the SWM-based technique.

4.1 Quantization and Weight Reduction

Data quantization on weights and neurons is a commonly used method for model compression. We attempt to use low-bit fixed-point data to represent the neurons and weights instead of using floating point data. We design a bit-wise simulator using C++ to verify the total number of bits for both integer and fractional part. Structure weight matrix, as a low-rank representation, uses one or several block circulant matrices to replace the original weight matrix as discussed in Section. 3. Shown in Fig. 2, by partitioning the original weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ into $p \times q$ blocks of square sub-matrices, the total number of weights are reduced from $m \times n$ to $\frac{m}{k} \times \frac{n}{k} \times k = (m \times n)/k$, where each block is a $k \times k$ matrix. We further investigate the SWM-based DNN models including DCNNs and LSTMs regarding the compression ratio (block size) and model accuracy.

4.2 Accuracy Evaluation

4.2.1 Accuracy Evaluation on DCNNs. The weight storage (model size) reduction, and the test accuracy on various image recognition datasets and DCNN models: MNIST (LeNet-5), CIFAR-10, SVHN, STL-10, and ImageNet (using AlexNet structure) [3, 4, 17, 18, 30]) are discussed in [5]. 16-bit data quantization is adopted and the baselines are the original DCNN models with unstructured weight matrices and 32-bit floating point representations. The SWM-based compression technique enables 400×-4000+× reduction in model size in the corresponding FC layers. On the other hand, the accuracy is close to original DCNN models and the accuracy degradation is negligible. Moreover, another advantage of the SWM-based technique is that the storage process of weight parameter after compression is regular, while reference works [11] bring in irregularity in storing the weight parameter. The introduced irregularity requires extra index per weight parameter and therefore affects the available parallelism degree.

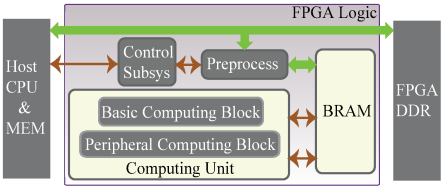

Figure 3: Overall system architecture of the proposed SWM-based FPGA compression framework.

4.2.2 Accuracy Evaluation on LSTM. We evaluate the structure matrices based compression technique using TIMIT benchmark, the most commonly used dataset for automatic speech recognition (ASR) application. The LSTM network is built by stacking multiple LSTM layers. The Google LSTM model [35] with unstructured weight matrix is selected as the baseline model. We preprocess the TIMIT audio data using FFT-based filterbank as discussed in [27, 38]. The input speech data have the same number of features and same architecture as ESE [9]. Phone Error Rate (PER) is adopted to evaluate the model prediction accuracy.

The block-circulant matrix based LSTM model enables a comprehensive tuning of model compression ratio by varying the block size k. The PER is close to baseline LSTM when the block size is 2 using SWM-based compression technique. For the SWM-based LSTM models with a block size of 8 and 16, 7.6X and 14.6X model size reduction can be achieved compared with baseline LSTM, respectively. On the other hand, the computational complexity is reduced by 2.6X and 3.7X while the PERs are only 0.32% and 1.23% higher than the baseline.

5 SWM-BASED HARDWARE DESIGN

5.1 FPGA

5.1.1 Overall Architecture. The overall SWM-based architecture is shown in Fig. 3. The Host CPU is responsible for issuing workload or instructions to the FPGA logic block and monitoring the working stats. The FPGA logic part includes computing unit (containing the basic computing block and the peripheral computing block), the control subsystem, BRAM block, and the preprocess block for certain designs when the data loaded from external memory requires preprocess. The memory hierarchy of the architecture primarily consists of three blocks: Host MEM, FPGA DDR, and on-chip block

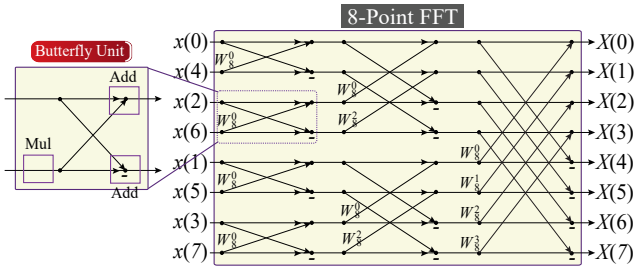

Figure 4: An example of 8-point basic computing block for FFT using butterfly units.

memory (BRAM). The control subsystem coordinates the actual FFT/IFFT operations in the basic computing block and peripheral computing block. The control subsystem also determines the input size of FFT/IFFT operations. The twiddle factors in FFT/IFFT operations are stored in BRAM (i.e., the W_n^i values including both real and imaginary parts); the weights, e.g., the FFT results FFT($\mathbf{w}_{ij}$) are also stored in BRAM.

5.1.2 Computing Unit Designs. In the computing unit, the peripheral computing block mainly focuses on component-wise multiplication, activation (ReLU, Tanh, and Sigmoid), pooling etc., which need lower computational cost and hardware footprint. The basic computing unit consists of an FFT operation with a parallelization degree of N and depth of $\log N$. Fig. 4 shows an example of 8-point FFT operation in the basic computing block using butterfly units. The IFFT operation can also be implemented using the N inputs basic computing unit in addition to a division operation (i.e., $\div N$) and two conjugations.

5.2 ASIC

In order to apply DNNs onto mobile/IoT devices, the DNN applications should be implemented in ASICs, due to the benefit of small hardware volume. The great reduction in both parameter size and computational time complexity makes our SWM-based method suitable for ASIC implementations. Figure 5 shows the architecture of our end-to-end ASIC implementation of the SWM-based DNNs. The architecture consists of four main blocks: input/output interface, storage system, processing system, and global controller.

The input/output interface is in charge of communicating with the external environment of the chip and the on-chip storage system. The input interface is composed of an input IO buffer and an input distributor. Similarly, the output interface is composed of an output IO buffer and an output distributor. In the view of data flow, the input IO buffer first receives and buffers data, including input images, weights, and biases from the external environment. For the reason that the number of IO pads are usually limited to a small number, whereas the bandwidth of the processing system is rather large for achieving high parallelism of computation. This mismatch in bandwidth requires an input distributor to temporally

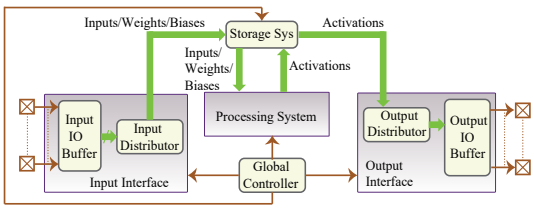

Figure 5: The architecture of the SWM-based chip.

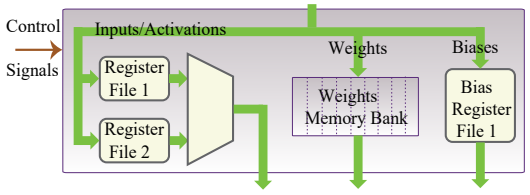

Figure 6: The architecture of the storage system.

hold the external data until the size of the data reaches the bandwidth requirement of the storage system. Besides, there are three storage modules inside the storage system for respectively storing inputs/intermediate activations, weights, and biases, the global controller will decide where the buffered data should flow. With the similar idea, the output distributor will receive final activations from the storage system and be controlled to distribute a portion of activations into the output IO buffer, which will further send them back to the external system.

As depicted in Figure 6, the storage system composes three subsystems, including a memory bank for storing weights, a register file for storing biases, and a ping-pong buffer (i.e., two alternating register files) for storing image inputs and intermediate activations.

The processing system achieves following equation for each layer: $\mathbf{y}_j = h(\Sigma\text{IFFT}(\text{FFT}(\mathbf{w}_{ij}) \circ \text{FFT}(\mathbf{x}_j)) + \mathbf{b}_j)$, where $\mathbf{w}_{ij}$ is the vector of weights at the ith row and jth column of the weight matrix, $\mathbf{x}_j$ and $\mathbf{b}_j$ are respectively the jth vector of inputs/activations and biases, and $h(\cdot)$ is an activation function. According to above equation, the processing system should contain the modules that are illustrated in Fig. 7. As the first step in the core computation, the image inputs are loaded from the storage system to the FFT module. Since the weights are repeatedly used without changes, what the weight memory bank stores are the weights in frequency domain. Thus the inputs of the multiply module are FFT($\mathbf{x}_j$) and FFT($\mathbf{w}_{ij}$). Next, the IFFT module performs the inverse FFT operation over the element-wise production vector, converting the vector from frequency domain to time domain. Then the summation is performed by the Accumulator module that generates the dot-product of inputs and weights. Finally, the Biase module adds up the biases to the dot-products, and the Activation module produces a vector of activations.

Another crucial module in the architecture is the global controller, which takes the responsibility to generate control signals to guarantee the whole system to function correctly.

6 EVALUTATION

6.1 FPGA

We implement the proposed framework on small to medium scale DNNs using the benchmarks of MNIST, SVHN, and CIFAR-10 on the low-power FPGA Intel (Altera) CyClone V 5CEA9. And we implement the proposed method LSTM on the platforms of Xilinx KU060 and Alpha Data's ADM-7V3. The ADM-7V3 board contains a Xilinx Virtex-7 (690t) FPGA and a 16GB DDR3 memory and the

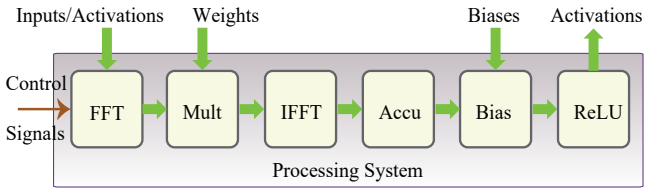

Figure 7: The architecture of the processing system.

Table 1: Comparison results on accuracy, performance, and energy efficiency of the proposed SWM-based FPGA designs and baselines.

DNN Name	Dataset	Platform	Data Quantization	Accuracy	Performance (kFPS)	Energy efficiency (kFPS/W)
Proposed MNIST 1	MNIST	CyClone V	12 bits	92.9%	8.6×10^4	1.57×10^5
Proposed MNIST 2	MNIST	CyClone V	12 bits	95.6%	2.9×10^4	5.2×10^4
Proposed MNIST 3	MNIST	CyClone V	12 bits	99.0%	363	659.5
Proposed SVHN	SVHN	CyClone V	12 bits	96.2%	384.9	699.7
Proposed CIFAR-10 1	CIFAR-10	CyClone V	12 bits	80.3%	1383	2514
Proposed CIFAR-10 2	CIFAR-10	CyClone V	12 bits	94.75%	13.95	25.4
TrueNorth ([6])	MNIST	TrueNorth	2 bits	99%+	1.0	9.26
TrueNorth ([6])	MNIST	TrueNorth	2 bits	95%	1.0	250
TrueNorth ([7])	SVHN	TrueNorth	2 bits	96.7%	2.53	9.85
TrueNorth ([7])	CIFAR-10	TrueNorth	2 bits	83.4%	1.25	6.11
LSTM Name	**Dataset**	**Platform**	**Data Quantization**	**PER Degradation**	**Performance**	**Energy efficiency**
Proposed LSTM1	TIMIT	ADM-7V3	16 bits	1.23%	330.275	14.359
Proposed LSTM1	TIMIT	KU060	16 bits	1.23%	371.095	-
Proposed LSTM2	TIMIT	ADM-7V3	16 bits	0.32%	179.687	8.168
Proposed LSTM2	TIMIT	KU060	16 bits	0.32%	195.312	-
ESE [9]	TIMIT	KU060	12 bits	0.30%	17.544	0.428

Xilinx KU060 platform contains a Xilinx XCKU060 FPGA and two 4GB DDR3 memory. We connect the ADM-7V3 to the host through PCI-e 3.0 X8 interface and the host machine used in the experiment is a sever configured with an Intel Core i7-4790 CPU. The proposed FPGA implementations of LSTMs are operating at 200MHz on both platforms.

We compare the accuracy, performance (kFPS), and energy efficiency (kFPS/W) of the proposed SWM-based FPGA implementation with the state-of-the-art IBM TrueNorth neurosynaptic processor ([6]) for DCNNs, and the state-of-the-art ESE accelerator on the platform of Xilinx KU060 [9] for LSTMs. We first demonstrate the results of three MNIST datasets targeting at different accuracies, one SVHN dataset, and two CIFAR-10 datasets targeting at different accuracies. The first two DNNs of MNIST datasets are multi-layer perceptron (MLP) models which can achieve the accuracy of 92.9% and 95.6%, respectively. The third DNN of MNIST dataset has a CNN structure similar to LeNet-5 [20], which achieves 99.0% accuracy. The first DNN of CIFAR-10 has a simple structure while the second DNN of CIFAR-10 adopts a wide ResNet model [12] which can achieve 94.75% accuracy. The baseline system (IBM TrueNorth) has two different DNNs of MNIST datasets at two accuracy levels. Experimental results show that under the similar accuracy constraint, the gains of the SWM-based framework in performance and energy efficiency are at least 152X and 72X, respectively. For the LSTM implementation, we propose two structures: (i) the proposed LSTM1 adopts a block size of 16 (FFT16), which the relative PER degradation of the model is 1.23%; (ii) the proposed LSTM2 uses a block size of 8 (FFT8), which the relative PER degradation of the model is 0.32%. On the platform of KU060, we achieve 21X and 11X performance speedup for the proposed LSTM1 and LSTM2 based compression techniques compared with ESE. On the platform of AMD-7v3, compared with ESE, we achieve 18.8X and 10.2X and performance enhancement and 33.5X and 19.1X energy efficiency gains using the proposed LSTM1 and LSTM2, respectively. Since the power consumption of SWM-based LSTM is only half of the ESE, the energy efficiency gain is higher than performance. Please note that the manufacturing process of XCKU060 FPGA is 20nm while the process of Virtex-7 is 28nm, which means the actual energy efficiency gain should be more than the report here.

6.2 ASIC

In this work, we implement an ASIC design of the SWM-based neural network for the image recognition task, and it is tested with the MNIST dataset. The implemented neural network has the original structure of $512 \times 512 - 512 \times 512 - 512 \times 64 - 64 \times 10$, and this network is transferred into an SWM-based structure. The FFT module implemented in this work is a 64-point FFT, that is, it takes a vector of 64 real value numbers as inputs and generates their frequency domain representations. Consequently, the weight matrices has the structure of $8 \times 8 \times 64 - 8 \times 8 \times 64 - 1 \times 8 \times 64 - 64 \times 10$, where $(m \times n \times s)$ represents the weight matrix has m rows and n columns, and each element is a vector containing s weights (s is 64 in this case). Our weight matrix transformation is not applied to the output layer, so the weights in this layer still keep the original structure of 64×10.

Table 2: Hardware Performance of SWM-Based Neural Network Implemented in ASIC

Metrics	Performance
Clock Frequency (MHz)	200
Area (mm^2)	1.3
Power (W)	0.14
Throughput ($Images/s$)	1.14×10^6
Energy Efficiency ($Images/J$)	8.08×10^6

Our ASIC design is implemented with SMIC 40nm technology (including memories) and synthesized with Synopsys Design Compiler 2016. Table 2 shows the hardware performance of our design. It can be observed from the table, the SWM-based neural network exhibits impressive advantages in terms of power ($0.14W$), throughput ($1.14 \times 10^6 Images/s$), and energy efficiency ($8.08 \times 10^6 Images/J$), suggesting that this method is greatly suitable for applying DNNs onto mobile/IoT devices.

7 CONCLUSION

In this work, we propose and evaluate the SWM-based compression technique on both FPGA and ASIC implementations. The SWM-based framework adopts the general block-circulant matrices to achieve a fine-grained tradeoff of accuracy and compression ratio and it works for both FC and CONV layers and contains a mathematically rigorous proof. For FPGA implementations, we achieve at least 152X and 72X improvement in performance and energy efficiency, respectively using SWM-based framework, compared with the baseline of IBM TrueNorth processor under same accuracy constraints using the data set of MNIST, SVHN, and CIFAR-10. For the LSTM network, the proposed SWM-based LSTM can achieve up

to 21X enhancement in performance and 33.5X gains in energy efficiency compared with ESE, respectively. For ASIC implementations, the proposed SWM-based design exhibits impressive advantages in terms of power, throughput, and energy efficiency. Experimental results indicate that this method is greatly suitable for applying DNNs onto both FPGAs and mobile/IoT devices.

8 ACKNOWLEDGEMENT

This work is funded by the National Science Foundation Awards CNS-1650469, CCF-1733701, CNS-1704662, CCF-1657333, CNS-1739748, and CCF-1733834.

REFERENCES

[1] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 609–622.

[2] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[3] Adam Coates, Honglak Lee, and Andrew Y Ng. 2010. An analysis of single-layer networks in unsupervised feature learning. *Ann Arbor* 1001, 48109 (2010), 2.

[4] Li Deng. 2012. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[5] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, Xiaolong Ma, et al. 2017. CirCNN: accelerating and compressing deep neural networks using block-circulant weight matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 395–408.

[6] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. 2015. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*. 1117–1125.

[7] Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. 2016. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences (PNAS)* (2016), 201604850.

[8] Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Berin Martini, and Eugenio Culurciello. 2014. A 240 g-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 682–687.

[9] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. 2017. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA.. In *FPGA*. 75–84.

[10] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 243–254.

[11] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[13] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[14] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. 2015. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716* (2015).

[15] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. *arXiv preprint arXiv:1704.04760 (published in ACM ISCA 2017)* (2017).

[16] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 1725–1732.

[17] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[20] Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, JS Denker, Harris Drucker, I Guyon, UA Muller, Eduard Sackinger, et al. 1995. Comparison of learning algorithms for handwritten digit recognition. In *ICANN*, Vol. 60. Perth, Australia, 53–60.

[21] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. 2009. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*. ACM, 609–616.

[22] Ji Li, Ao Ren, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, and Yanzhi Wang. 2017. Towards acceleration of deep convolutional neural networks using stochastic computing. In *The 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE*.

[23] Ji Li, Zihao Yuan, Zhe Li, Caiwen Ding, Ao Ren, Qinru Qiu, Jeffrey Draper, and Yanzhi Wang. 2017. Hardware-driven nonlinear activation for stochastic computing based deep convolutional neural networks. In *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 1230–1236.

[24] Ji Li, Zihao Yuan, Zhe Li, Ao Ren, Caiwen Ding, Jeffrey Draper, Shahin Nazarian, Qinru Qiu, Bo Yuan, and Yanzhi Wang. 2017. Normalization and dropout for stochastic computing-based deep convolutional neural networks. *Integration, the VLSI Journal* (2017).

[25] Zhe Li, Ao Ren, Ji Li, Qinru Qiu, Yanzhi Wang, and Bo Yuan. 2016. Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks. In *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 678–681.

[26] Zhe Li, Ao Ren, Ji Li, Qinru Qiu, Bo Yuan, Jeffrey Draper, and Yanzhi Wang. 2017. Structural design optimization for deep convolutional neural networks using stochastic computing. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 250–253.

[27] Zhe Li, Shuo Wang, Caiwen Ding, Qinru Qiu, Yanzhi Wang, and Yun Liang. 2018. Efficient Recurrent Neural Networks using Structured Matrices in FPGAs. *arXiv preprint arXiv:1803.07661* (2018).

[28] Sheng Lin, Ning Liu, Mahdi Nazemi, Hongjia Li, Caiwen Ding, Yanzhi Wang, and Massoud Pedram. 2017. FFT-Based Deep Learning Deployment in Embedded Systems. *arXiv preprint arXiv:1712.04910* (2017).

[29] Konstantinos Makantasis, Konstantinos Karantzalos, Anastasios Doulamis, and Nikolaos Doulamis. 2015. Deep supervised learning for hyperspectral data classification through convolutional neural networks. In *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*. IEEE, 4959–4962.

[30] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, Vol. 2011. 5.

[31] Victor Pan. 2012. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media.

[32] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. 2016. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 26–35.

[33] Ao Ren, Zhe Li, Caiwen Ding, Qinru Qiu, Yanzhi Wang, Ji Li, Xuehai Qian, and Bo Yuan. 2017. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 405–418.

[34] A. Ren, Z. Li, Y. Wang, Q. Qiu, and B. Yuan. 2016. Designing Reconfigurable Large-Scale Deep Learning Systems Using Stochastic Computing. *Proc. of ICRC* (2016).

[35] Haşim Sak, Andrew Senior, and Françoise Beaufays. 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*.

[36] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[37] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. 2015. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067* (2015).

[38] Shuo Wang, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, Yanzhi Wang, and Yun Liang. 2018. C-LSTM: Enabling Efficient LSTM using Structured Compression Techniques on FPGAs. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM.

[39] Yanzhi Wang, Caiwen Ding, Geng Yuan, Siyu Liao, Zhe Li, Xiaolong Ma, Bo Yuan, Xuehai Qian, Jian Tang, Qinru Qiu, and Xue Lin. 2018. Towards ultra-high performance and energy efficiency of deep learning systems: an algorithm-hardware co-optimization framework. In *AAAI Conference on Artificial Intelligence, (AAAI-18)*. AAAI.

[40] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. 2017. Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 548–560.

[41] Zihao Yuan, Ji Li, Zhe Li, Caiwen Ding, Ao Ren, Bo Yuan, Qinru Qiu, Jeffrey Draper, and Yanzhi Wang. 2017. Softmax Regression Design for Stochastic Computing Based Deep Convolutional Neural Networks. In *Proceedings of the Great Lakes Symposium on VLSI*. ACM, 467–470.

[42] Liang Zhao, Siyu Liao, Yanzhi Wang, Zhe Li, Jian Tang, and Bo Yuan. 2017. Theoretical Properties for Neural Networks with Weight Matrices of Low Displacement Rank. In *International Conference on Machine Learning*. 4082–4090.