A Hardware-Friendly Approach Towards Sparse Neural Networks Based on LFSR-Generated Pseudo-Random Sequences

Foroozan Karimzadeh, *Graduate Student Member, IEEE*, Ningyuan Cao, *Student Member, IEEE*, Brian Crafton, *Student Member, IEEE*, Justin Romberg, *Fellow, IEEE*, and Arijit Raychowdhury, *Senior Member, IEEE*

***Abstract*— The increase in the number of edge devices has led to the emergence of edge computing where the computations are performed on the device. In recent years, deep neural networks (DNNs) have become the state-of-the-art method in a broad range of applications, from image recognition, to cognitive tasks to control. However, neural network models are typically large and computationally expensive and therefore not deployable on power and memory constrained edge devices. Sparsification techniques have been proposed to reduce the memory foot-print of neural network models. However, they typically lead to substantial hardware and memory overhead. In this article, we propose a hardware-aware pruning method using linear feedback shift register (LFSRs) to generate the locations of non-zero weights in real-time during inference. We call this LFSR-generated pseudo-random sequence based sparsity (LGPS) technique. We explore two different architectures for our hardware-friendly LGPS technique, based on (1) row/column indexing with LFSRs and (2) column-wise indexing with nested LFSRs, respectively. Using the proposed method, we present a total saving of energy and area up to 37.47% and 49.93% respectively and speed up of 1.53× w.r.t the baseline pruning method, for the VGG-16 network on down-sampled ImageNet.**

***Index Terms*— Sparsity, sparse neural network, LFSR, DNN accelerator, linear feedback shift register.**

I. Introduction

EVER-INCREASING number of edge devices such as mobile, wearable and Internet of Things (IoT) devices require computations to be performed close to the source of the data on the edge [1], [2]. This is critical due to the fact that sending/receiving data to/from a centralized server increases both the latency as well as the cost of communication. Edge computing can enable real-time data analysis locally which mitigates the latency and increases privacy of the data. It is also important where we need accurate and fast computation to generate results under strict latency constraints. Over the past several years, a trend toward embedded computing of artificial neural networks (ANN) in edge devices have emerged [1]. ANN models such as deep neural networks (DNN) which are typically very large, have gained remarkable performance in data analysis. However, it is hard to deploy these models on edge devices since they are resource constrained [3].

ANNs have achieved state-of-the-art performance in various machine learning applications, such as computer vision [4]–[6], natural language processing [7] and health care [8], [9] etc. Not only ANNs' model have shown a remarkable progress in improving accuracy over time, but also grown gradually to larger and more complex models. For instance, LeNet-5 [10], a classical convolutional neural network (CNN), developed in 1998 with less that 500K parameters for handwritten digits classification, while VGG-16 [11], the winner of ImageNet competition in 2014, has more than 90M parameters.

Although the aforementioned networks are powerful, due to their large size, it is hard to accommodate large networks on an on-chip memory and an external DRAM memory is often required. In addition, large and over-parameterized ANNs are computationally expensive and consume considerable amount of memory and energy. For example, in 45nm CMOS process, accessing an external DRAM consumes 3 order of magnitude more energy that accessing the on-chip SRAM [12].

A growing body of research has been devoted to improve the efficiency of DNNs for inference and edge devices [12]–[16]. It has been shown that DNNs are mostly over-parameterized and hence parts of the network are redundant [17]. Model compression via pruning and sparsity can reduce the size of DNNs while preserving the accuracy [12], [14], [18]–[20] and therefore it can enable DNNs to fit to an on-chip SRAM. However, sparse networks add a level of irregularity to the network and the resultant sparse matrix of weight/activation lacks structure [21], [22]. Therefore, platforms such as GPUs and ASICs cannot efficiently take advantage of the sparse representation of those networks [14].

In this article, we develop LGPS, a hardware-aware pruning method to accelerate the DNNs by shrinking the size

Manuscript received May 14, 2020; revised August 24, 2020 and October 17, 2020; accepted November 4, 2020. Date of publication November 19, 2020; date of current version January 12, 2021. This work was supported by the Semiconductor Research Corporation under Grant Joint University Microelectronics Program (JUMP) Center for Brain-inspired Computing Enabling Autonomous Intelligence (C-BRIC) 2777.004, Grant 2777.005, and Grant 2777.006. This article was recommended by Associate Editor C. H. Chang. *(Corresponding author: Foroozan Karimzadeh.)*

The authors are with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: fkarimzadeh6@gatech.edu; arijit.raychowdhury@ece.gatech.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2020.3037028.

Digital Object Identifier 10.1109/TCSI.2020.3037028

of DNNs and reducing the required memory footprint to make them deployable on mobile applications and edge devices. The goal is to reduce the size of DNNs while preserving the test accuracy. We use an on-die linear feedback shift register (LFSR) using a known input seed, to generate a pseudo random sequence (PRS) that acts as the indices of non-zero weights in the sparse network. In the training step, The weights specified by the generated PRS are regularized in an iterative process to force them to zero. In the next step, the zero weights are pruned. Finally, the sparse network is retrained so that the model can compensate for the pruning process. During inference, the same LFSR are utilized to generate the same PRS in real-time as the addresses of the sparse matrix to calculate multiplication/accumulation between the sparse weight matrix and input/activation vector. The advantage of the proposed method is that we no longer need to save the remaining weights' addresses in memory and can reduce a considerable amount of storage/memory foot-print and energy consumption during inference.

The rest of the paper is as follows: a comprehensive literature review is provided in Section II. Our method is explained in Section III and experimental results are provided in Section IV. Finally, Section V is dedicated to the conclusion.

II. Related Works and Motivation

In this section, we explore the prior works that have been done on pruning and sparisification of ANNs. Then, the challenges associated with existing models and the motivation behind the proposed method are presented.

A. State-of-the-Art ANNs

Fully connected (FC) MLP (multi-layer perceptron) and CNNs are among the State-of-the-art ANNs. These networks usually have a lot of parameters that should be trained. The number of parameters of four state-of-the-art networks including LeNet-300-100, LeNet-5 [10], AlexNet [4] and VGG16 [11] are illustrated in figure 1. LeNet-300-100 is a FC network while the rest of them are CNNs. Figure 1 shows that a considerable number of parameters are related to the FC layers. For example, the number of FC connections in VGG-16 are 10× more than that of convolutional connections.

B. Sparse Neural Network

There are several prior research that have demonstrated there are significant redundancies in most of DNN models [12], [17] which result in using extra power and storage resources. As such, various model compression techniques such as Deep Belief Network (DBN) [23], regularization [24], Auto Encoder [25] have been developed to prune the redundant parameters and make DNNs more efficient without losing accuracy. Reference [26] proposed a method to compress the network by using 8-bit fixed point integer for weights value instead of 32-bit floating point representations. Reference [13] presented a method to prune the network followed by weight sharing to reduce the number of bits required to represent the weight and the activation values comparable to the original network.

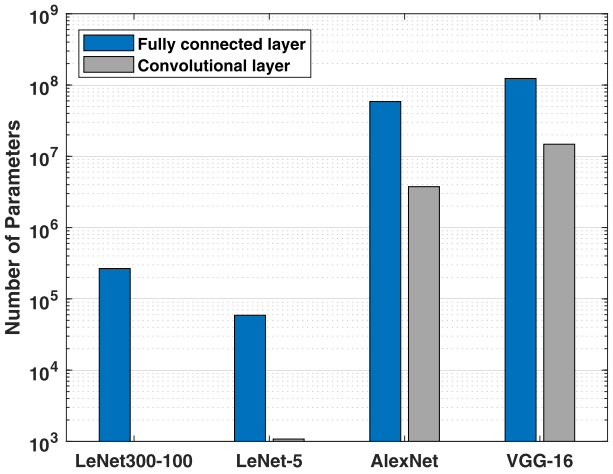


Fig. 1. Number of parameters in FC vs convolutional layers for baseline networks consists of LeNet 300-100, LeNet-5, AlexNet and VGG-16. The vertical axis is in logarithmic scale.

In general, we can divide pruning methods into two categories: (1) structured pruning, (2) unstructured pruning. Unstructured pruning is based on a criteria (e.g. magnitude, threshold, etc.) that element-wise prune the weight connections. Threshold-base pruning was applied to DNNs in [12], resulting 9× and 13× model size compression on AlexNet and VGG-16, respectively. In another method called Deepcompression [13], threshold-based method followed by weight sharing and Huffman coding are applied to compress DNN networks which achieves 49× memory saving on VGG network. Moreover, 20× model size reduction is achieved by applying magnitude-based pruning to Long Short Term Memory (LSTM) hardware [27]. In addition, [28] proposed a dynamic pruning approach. In this method, instead of permanently pruning a connection, some pruned connections are allowed to regain their importance and regrow during an iterative training process. Although unstructured and element-wise pruning results in higher parameter reduction, these methods incur considerable index memory and irregular memory access, hurting both performance and power. Several DNN accelerator have been developed to cope with the irregularity caused by unstructured pruning [3], [14]. However, as mentioned before, they need to store the addresses related to the location of random weights that have been kept after training.

On the other hand, the idea of structured pruning is to avoid irregular model compression in the obtained weight matrices after pruning [29]–[31]. Several research have been done to structurally prune DNNs by pruning the structural component such as the entire or parts of layers and filter channels based on calculating their importance with respect to the test accuracy [32], rank of the filters [33], [34], or ranked them based on the contribution of each filter to the next layers' activation [35] or through regularization [36], [37] and [24]. In [38], a block-wise sparsity technique called coarse-grain sparsification (CGS) to prune multilayer perceptrons (MLPs) is presented. However, the results demonstrate limited weight compression of 4×. In [39], energy-efficient LSTM recurrent neural network (RNN) accelerator is proposed by using an hardware-centric network compression technique called

hierarchical coarse-grain sparsity (HCGS). To compress the network, each weight layer passes through several level of CGS using various block size. By using HCGS-based block-wise recursive weight compression, the author demonstrated that LSTM networks can be compressed 16× while obtaining minimal accuracy loss. Using CGS for sparsification of a networks leads to decrease memory footprint and energy required for running inference. However, the performance of this method is highly dependent to the size of the block. Also, the value of the dropped blocks keep to be zero during the training which might negatively affect the large networks where the remaining weights could not fine-tune the effect of pruning. In addition, in the inference, the address of the selected blocks should be saved. In another work, [40] pruned the network by choosing a global percentage k and removing all the $k\%$ synapses that have the smallest weight magnitudes across all the network's layers. Reference [41] performed pruning by defining different pruning criteria using correlation-base pruning rather than magnitude-base pruning. In [42], FC layers are pruned by solving a least squares problem in which the difference between activation of the pruned and original networks is minimized.

On the hardware side, several accelerators and hardware architecture have been designed to exploit sparsity of pruned neural networks. The Cambricon-X architecture [14] takes advantage of the sparse network by proposing an indexing method to skip the zero weights. SCNN [15] is another architecture that proposes a new dataflow by encoding and maintaining non-zero weights and activations and efficiently delivering them to a multiplier array. EIE architecture [3] proposed a DNN accelerator and an indexing framework to accelerate the mathematical calculation of a sparse network for the compression method called deep compression [13] where pruning, quantization and Huffman encoding are employed to compress the network. The Eyeriss architecture [21] is designed to run compact DNNs by introducing a hierarchical mesh which is a flexible on-chip network that can be adapted to different data types and improves the utilization of the memory resources.

1) Baseline Pruning Method: In this section a baseline method is explained and the comparative results are shown in Section IV. The state-of-the-art baseline pruning method is introduced by Han *et al.* [12] in 2015. The method consists of three steps including training the network, pruning redundant weights and retraining the remaining connections. In the first step, the ANN is trained using the standard back propagation method with random initial inputs. In the next step, the weights (i.e synapses) which are less than a predefined threshold value (equal to the multiplication of a quality parameter by the standard deviation, std, of a layer's weights) are pruned. This step is repeated iteratively for several epochs until no weights can be further removed. Finally, after several iterations of retraining the remaining weights are performed to fine tune the new network. The main motivation of this method is that the small weights have less or no contribution to the overall performance of a neural network.

2) CGS and SIMD Aware Pruning: In addition to the baseline method, we compared LGPS with two other methods recently proposed. In the first one, a block-wise pruning method called Coarse Grain Sparsity (CGS) was introduced in [38]. In this method, a sparse network is achieved by dropping large blocks of weights during training instead of element-wise pruning. Those blocks are selected randomly and the corresponding connections set to zero in initialization step and remain zero until the end of training. The idea behind this method is to reduce the number of addresses (or indices) of the sparse connections that should be stored in a on-chip memory by storing the block addresses instead of element-wise indices. In the second method, called SIMD-aware pruning [43], the elements of weight matrix are divided into groups. In the pruning step, the Root-Mean-Square (RMS) of each group is calculated and groups with RMS value bellow a threshold are prunned. Finally, the pruned network is retrained for several epochs.

C. Motivation for LGPS

It has been shown that ANNs can be trained using less number of neurons and synapses [17] by sparsifying the network and removing the redundant weights. The large neural networks consume considerable amount of memory resources and power which makes it hard to deploy on battery and memory constrained edge devices. Therefore, model compression through pruning and sparsity is one solution to shrink the network size while preserving the classification accuracy. Although the baseline pruning method [12] has good performance from an algorithm perspective, from hardware prospective, it requires as high as 2× memory foot-print (for storing values and address indices in memory) compared to the model size. In the CGS method, the number of indices is reduced by limiting the sparification to the block size, but it presents an inherent trade-off between the degree of pruning and the number of block indices that need to be stored.

On the other hand, sparse networks add a level of irregularity to the network which makes it even harder to run them on hardware platforms such as GPUs or CPUs. This irregularity comes from the fact that the network's weights are pruned randomly. The existing hardware platforms such as GPU and CPU do not support efficient sparse networks with irregular weight orders. State of the art DNN accelerators also cannot take full advantage of the lower memory footprint of sparse networks [14].

Therefore, these observations motivate us to develop a hardware-aware pruning method and accelerator to remove the redundant connections and take advantage of a sparsity network and also reduce the memory footprint of storing the non-zero indices. We expect our approach to be an enabler for deploying state-of-the-art neural networks on edge devices and mobile platforms.

III. Proposed LFSR-Generated PRS Based Sparsity (LGPS) Method

In this article, we present a hardware-friendly pruning method, namely LGPS. During the *training step* in consists of the following steps:

1) In the first step of the training algorithm, a pseudo-random sequence (PRS) is generated that acts as the

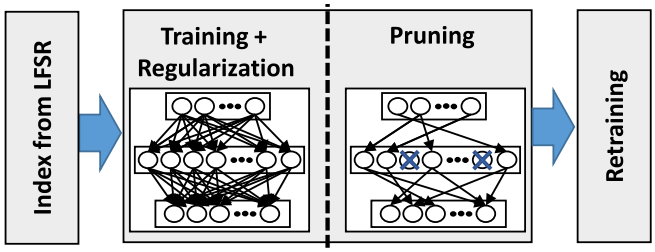


Fig. 2. LGPS method consists of four main steps: generating indices using LFSR topology, training the network with regularization of the specified weights using LFSR, pruning redundant weights and retraining the remaining connections.

indices to sparsify the synapses. The PRS can be generated on-die using linear feedback shift register (LFSR). We show that the expressibility of the network and the accuracy of the model are not compromised in the process.

2) Next the network is trained with regularization of the specific synapses whose addresses (or indices) are not covered by the PRS. Only the indices that are produced by the PRS are not regularized. All the regularized parameters are forced to zero.
3) Prune the regularized connections and retrain the remaining connections (whose indices are covered by the PRS). (figure 2).

Generating the locations of the zero weights in the connectivity matrix by using a PRS provides good performance, and also making it easier to generate the indices on the fly, without the need to be stored in a separate memory sub-bank. During deployment, the non-zero weights of the pruned network, the seed of the PRS and the structure of the LFSR are shared with the edge device. During inference, the following steps are executed:

1) The structure of the LFSR and the seed of the PRS provides the same bit-wise stream of the same PRS that was used during training.
2) The non-zero weights are read sequentially and the corresponding address is matched with the PRS value. This allows the correct activation to be multiplied with the correct weight.
3) All the non-zero weights are covered and the final result of the classification problem is generated.

A. Linear Feedback Shift Register (LFSR) Based PRS Generation

LFSR [44] is a common topology to generate pseudo random bit sequences. The block diagram of a general LFSR is shown in figure 3. It consists of a cascade of n flip-flops followed by linear feedback using a couple of exclusive-or (XOR) gates (c_i). The value used to initialize the LFSR is called input seed (s_i). The mathematical formula to calculate the LFSR output sequence is shown in Eq. 1.

$$s_j = \sum_{i=0}^{n} c_i s_{j-i}, \quad j \geq n \tag{1}$$

where s_j is the output sequence.

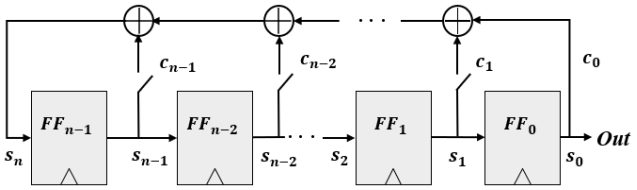


Fig. 3. A general LFSR topology with n flip-flops.

The main advantages of using the LFSR topology to generate PRS are:

1) The hardware implementation is simple and compact.
2) The PRS is generated in real-time within a clock cycle and does not require any memory foot-print.
3) The generated PRS has useful statistical properties that preserves the rank of the generated connectivity matrix [45] (to be elaborated further in section IV).
4) The maximum PRS length without repetition (equal to $2^n - 1$) can be achieved as long as the characteristic polynomials is primitive [45].

In this article, we propose two different indexing schemes to prune the networks using LFSR. In the first method (called LGPS-I), we use two separate LFSRs with different input seeds to generate indices for rows and columns separately. The row indices encode the addresses of input vector elements while the column indices indicate the address of the output vector. In the second indexing approach (called LCPS-II), we use one LFSR to generate random indices for each column of weight matrix. To generate a different PRS within each column we utilize another LFSR to generate different random input seed for each column and ensure that we preserve the rank of the sparse matrix. Finally, for both approach, to keep the generated indices within the range of the row/column length, we multiply the generated index to the size of the row/column and keep the most significant bit (MSB) as the desired index. The process of generating the addresses using LFSR is summarized in figure 4a.

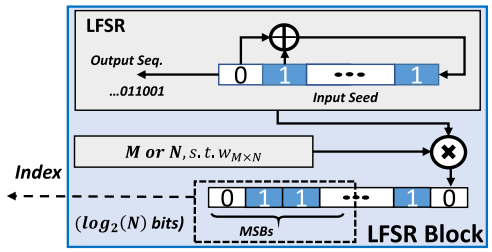
B. Training LGPS With a PRS Based Regularization

After generating the PRS using the LFSR, we use them as the indices for the connection matrix that needs to be kept and the remaining connections will be regularized and pruned. A fully connected (FC) layer of a deep neural network (DNN) with input (x), weight matrix (W) and vector of bias (b) performs the following function. The following formulation is based on our first indexing approach i.e. row/column indexing. The second method for PRS generation can also be similarly formulated and omitted here for brevity.

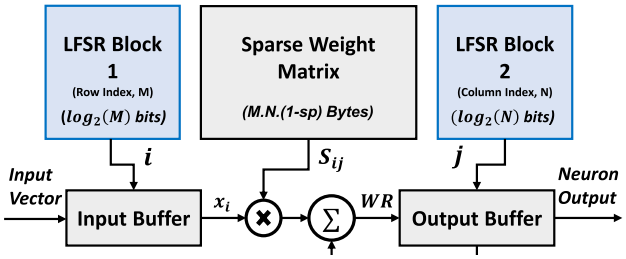
$$Z = W^T x + b \tag{2}$$

$$a = \sigma(Z) \tag{3}$$

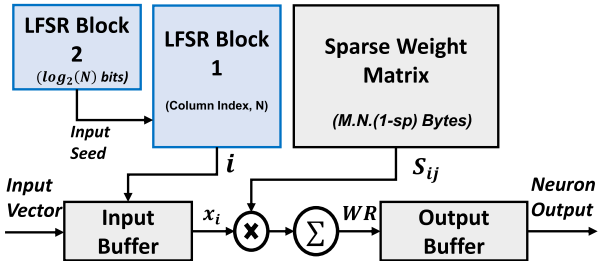
where T is a transpose function, σ is a non-linear activation function which is typically chosen to be a Rectified Linear Unit (ReLU) [46]. To simplify the above equations, we can merge vector of b with W. This can be done by appending b as an additional column to the end of matrix W. Then the



(a) LFSR Block.



(b) The proposed hardware using row/column LFSR indexing.



(c) The proposed hardware using column-wise LFSR indexing.

Fig. 4. High-level illustrations of our proposed DNN accelerator architecture for inference. The architectures of (a) LFSR Block, (b) LGPS-I and (c) LGPS-II.

above equations can be rewritten and calculated element-wise as follow:

$$a = ReLU\left(\sum_{d=0}^{n-1} W_{de}x_e\right) \tag{4}$$

where d and e are the indices of the original weight matrix corresponding to the rows and columns, respectively.

In the next step, the specific connections (weights) selected based on LFSR indexing are trained without regularization and the remaining connections are regularized to be zero during the training step. We have investigated the use of both L1 and L2 regularization methods to penalize the target connections (figure 8). L1 regularization results in more weights to be near zero which gives better performance in terms of accuracy after pruning without performing another retraining step [12]. On the other hand, L2 regularization gives the best retraining results. Regularization also prevents over-fitting as the PRS selects a random subset of target synapses.

In the regularization methods, a regularizer component is added to the cost function (J). Here, we show the formula for L2-regularization in Eq.5. In addition, weights will be updated during back propagation process as shown in Eq. 6.

$$J(W^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{d=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} ||W^{[l]}_{!i\ !j}||_F^2 \tag{5}$$

where !i and !j are the weights related to row and column indices that should be regularized to be zero. In other words, these indices are the one that are not covered by PRS from LFSR.

$$W^{[l+1]} = \begin{cases} W^{[l]}[1 - \frac{\alpha\lambda}{m}], & \text{if } d, e \neq i, j \\ W^{[l]} - \alpha d W^{[l]}, & \text{if } d, e = i, j \end{cases} \tag{6}$$

where i, j, L and α are correspond to the row and column indices generated from LFSR, the layer's number in the network, and the learning rate, respectively. λ is the regularization parameter and can be tuned where larger λ penalize the weights values more and make them closer to zero.

C. Pruning and Retraining

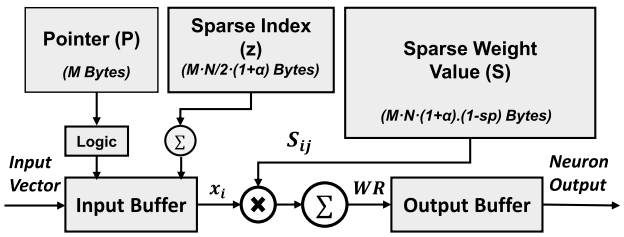
Regularization makes selected weight values to be zero or very close to zero. However, in order to design a DNN accelerator for LGPS, we need to make sure that the selected weights are exactly equal to zero. Therefore, we add a pruning step to guarantee that all the selected weights are zeroed-out. The computation of activation function with the LFSR based pruning method shown in Eq. 4 becomes

$$a = ReLU\left(\sum_{j}^{n-1} S_{ij}x_j\right) \tag{7}$$

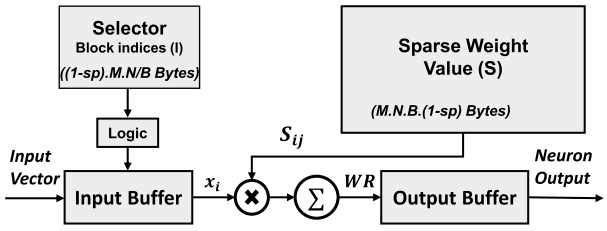
where S is correspond to the sparse weight matrix. Finally, the pruned network is retrained iterativly for several epochs to compensate for the pruned connections and fine-tune the remaining ones.

D. DNN Compression and Hardware Architectures

We design an efficient DNN hardware accelerator to perform inference on the proposed sparse networks. After training the neural network using LGPS, the compressed model is ready to be deployed. As mentioned before, a baseline pruning technique to sparsify networks add irregularity to the structure which makes it hard for the state-of-the-art DNN accelerators to fully take the advantage of the reduced memory footprint [14]. The main advantage of using LGPS method is that we generate the indices of the unpruned connections in real-time during inference and as a result only the non-zero weights need to be stored. We implement the hardware design of LGPS, baseline, the CGS and SIMD methods to compare and contrast the advantages and limitations of them. The architecture of the LFSR based indexing LGPS-I and LGPS-II architecture are illustrated in figure 4. In addition, the architecture of the baseline design and CGS methods are shown in figure 5 for completeness. The block diagrams illustrate the difference in hardware resources/operations in the proposed methodologies, baseline and CGS methods to



(a) Baseline method.



(b) CGS method.

Fig. 5. High-level illustrations of DNN accelerator for inference. The architectures of (a) baseline and (b) CGS method.

infer from a sparse FC network, with N input neuron, M output neuron and sp as the level of sparsity (i.e number of zeros). The detailed explanation of these methods are as follow:

The hardware design of the two proposed methods is demonstrated in figure 4b and 4c, respectively. We describe the architecture in the next two subsections.

1) LGPS With Row/Column Indexing Using LFSR (LGPS-I): In this scheme (figure 4b), we call LGPS-I, LFSR block1 is used to generate the PRS as an index for each of the input neurons. The way the PRS (as an index) is generating inside LFSR Block is shown in figure 4a. Since the range of generated PRS is between 1 and $2^N - 1$, to keep the range in the number of input neurons, the generated PRS is multiplied to the length of input neurons (m). Then, the most significant bits (MSBs) of the calculated value in binary form is selected. This scales the generated number within range $m \leq 2^N - 1$. If we do not perform this scaling, a PRS number greater than m can be generated and results in unused cycles, that reduces the throughput. Next, the generated index is used to select the right input to be multiplied to the corresponding weight value in the sparse weight matrix (S). The result of multiplication/accumulation is stored in the output buffer where the address comes from the second LFSR block with different input seed. Again the output of the LFSR block 2 (j) is calculated based on the process showed in 4a. This time the PRS generated by LFSR is multiplied to the column length (N) in order to keep the range of the indices (j) between 1 to N. In order to calculate the output neuron using this architecture, the output of multiplication/accumulation should write and store in the output buffer until the result for one input column is calculated which is referred to the output neuron. This leads to the extra reads and writes to/from memory. The exact number of memory reads from the input and the output buffer depend on the number of multiply and accumulate units and also the model size.

2) LGPS With Column-Wise Indexing Using Nested-LFSRs: To solve the problem of extra reads and writes in the aforementioned architecture, we introduce a slightly modified architecture based on column-wise indexing to generate a sparse network using nested-LFSRs. We call this LGPS-II. In this second methodology, we introduce a column-wise indexing approach using one LFSR to generate indices in the size of input column. Block-diagram of this approach is illustrated in figure 4c. Column indices are generated using the LFSR block 1 introduced in Figure 4a. The input seed is generated randomly using LFSR block 2. The reason behind using LFSR block 2 is to generate different input seeds for each column which cause different PRSs for each column. This will help to preserve the rank of the sparse matrix during training. Finally, the multiplication-accumulation of input and sparse weight matrix is performed column-wise and the value of each output neuron is calculated without the need to store them in the output buffer each time. The advantage of this approach is that it reduces the number of read/write from/to the output memory. In addition, this approach aids parallel processing as the indices for each column are generated in parallel and are uncorrelated to the others. We can take advantage of this by distributing over multiple sub-arrays during inference.

3) DNN Accelerator for Benchmarks: To accelerate the multiplication-accumulation of baseline algorithm, the weight matrix is typically compressed to three vectors that should be saved in the memory (Figure 5a). The first, a vector (S) consists of non-zero values of each column of weight matrix W. The second, vector (z), that has equal length to vector S, which includes the addresses of each entry in S. Third, a pointer vector (p) which keeps track of each column and points to the beginning of each column's vector, is used. In addition, the bit-width of each entry of S and z is designed to be four-bit or eight-bits, which leads to additional memory usage, since the index is represented in with a half-byte or one-byte resolution. We have denoted this ratio parameter as α. For example, when using a four bit-width representation, if more than $2^4 - 1$ zeros appear before a non-zero entry, a zero is added to the vectors S and z. This will result in a larger memory foot-print to store the sparse weight matrix.

The architecture of CGS method is demonstrated in figure 5b where the sparse weight matrix (S) and the addresses of the remaining blocks (I) should be stored in memory. The size of the block size (B) is defined during training. As an example the block size B can be 64×64. A selector and corresponding logic is required to choose the correct input for multiplication to the sparse weights. The final benchmark, SIMD method, also stores the sparse weight values, a vector containing the number of column for the first element of each group and a vector containing the row index of the first non-zero element in each row.

IV. Experimental Results

In this section, we present the experimental results of the algorithm and hardware implementation of LGPS-I and LGPS-II compared to the baseline, CGS and SIMD methods.

A. Benchmark Methods

We used three state-of-the-art pruning methods to compare the algorithmic results. In the first method [12] (baseline), the connections that have the values less than a predefined threshold are pruned. The threshold is determined by the standard deviation of the weights multiplied by a set of hyper-parameters. We choose the hyper-parameters by trial and error and this leads to fine-tuning of the level of sparsity. In the second methodology (CGS method) [38], a pruning method is proposed to randomly drop the weights in a coarse-grain block-wise basis. Note that all blocks along a row or a column should not be pruned in order to keep all the neurons during training. We choose learning rate of 0.01 for training. In the third method (SIMD) [43], the weight matrix is divided to groups of size two, and the groups that have RMS less than a threshold are pruned away.

B. Simulation Results for the Proposed Pruning Algorithm

To demonstrate the effectiveness of our proposed methodologies, we evaluate the results on both FC networks and CNNs using different datasets consists of MNIST, Cifar10 and down-sampled ImageNet and ImageNet (original size). For FC networks, we have used LeNet-300-100, and FC networks with different hidden layer sizes, we also utilized LeNet-5 and VGG-16 as CNNs. It should be noted that in order to use downsampled ImageNet, we modified the VGG16. Finally, we evaluate the pruning method on MobileNet, as an example of a network that is designed to run on mobile devices, using original ImageNet. The details are explained in the following part, IV-B2. Training step for each network is implemented on Tensorflow platform and carried out on Nvidia GTX 1080 Ti GPUs. It should be noted that we mainly focused on pruning FC layers' weights as they consume the maximum amount of memory resources. Moreover, there are fewer opportunities of effective hardware mapping of FC layer computations than convolution layers [47]. As shown in figure 1, large DNNs are over-parameterized; this is mainly because of the large number of connections in fully connected layers of these networks that do not contribute to an output activation. As an example, about 89% of VGG-16 parameters are in the FC layers. The test accuracy, number of parameters and the rate of compression of different networks including fully connected networks with different hidden layers on MNIST dataset, LeNet-5 on MNIST, VGG-16 on downsampled ImageNet and MobileNet on original ImageNet dataset are reported in Table I. These results show there are redundancy in number of parameters in neural networks and the weights can be heavily pruned while preserving the accuracy. LeNet 300-100, LeNet-5 and modified VGG-16 can be compressed, while preserving the accuracy, compared to the unpruned network by $11\times$, $12\times$ and $7\times$, respectively. We also evaluated our method on MobileNet which is designed for mobile applications and has significantly reduced number of parameters. We have achieved $1.34\times$ compression rate on this network.

1) Results on MNIST Dataset: In this section we have demonstrated the pruning results of various ANNs on MNIST. Figure 6 illustrates the accuracy ($mean \pm std$) of our pruning algorithms, LGPS-I and LGPS-II on MNIST dataset for 10 trials. We evaluated them on two different networks including LeNet-300-100 and LeNet-5. The first network, LeNet-300-100, is a fully connected network with two hidden layers of length 300 and 100 neurons each. The second one, LeNet-5, is a convolutional neural network with two convolutional layers followed by two fully connected layers of sizes 120 and 84 neurons. The results show that our pruning method can prune the networks more that 90% while preserving the accuracy. In addition, LGPS-I and LGPS-II perform equally well for both networks and different sparsity levels. As such, for the software evaluations of our method, we just present the results for LGPS-I. LGPS-II shows similar algorithmic results. This is because the LFSR generated random numbers maintain the rank of the connectivity matrix, which in turn preserves the expressibility of the matrix [48]. To evaluate this property, we compare the ranks of the unpruned networks, LGPS, baseline (threshold pruning) and CGS method using block size

TABLE I

NUMBER OF PARAMETERS, PRUNING (USING LGPS METHOD) AND REFERENCE ACCURACY AND RATE OF COMPRESSION FOR DIFFERENT NETWORKS

Network	Error	Parameters	Compression Rate
LeNet-300-100 Unpruned	4.1%	267K	
LeNet-300-100 Pruned	4.3%	24K	11×
FC-512-512 Unpruned	2.7%	669K	
FC-512-512 Pruned	2.8%	67K	10×
FC-256-256 Unpruned	2.9%	668K	
FC-256-256 Pruned	3.1%	66K	10×
LeNet-5 Unpruned	1.5%	431K	
LeNet-5 Pruned	1.6%	35K	12×
Modified VGG-16 Unpruned	48.6%	23M	
Modified VGG-16 Pruned	50.1%	3.3M	7×
MobileNet Unpruned	40.5%	4.24M	
MobileNet Pruned	43.1%	3.17M	1.34×

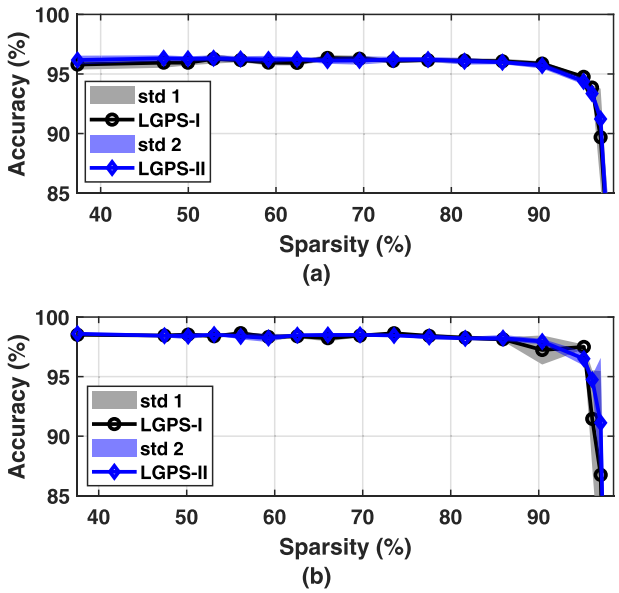


Fig. 6. The accuracy ($mean \pm std$) vs sparsity percentage of LGPS-I and LGPS-II on MNIST dataset for 10 trials and two different networks: (a) LeNet-300-100 and (b) LeNet-5.

TABLE II

RANK OF FULLY CONNECTED LAYERS OF A FULLY CONNECTED (FC) NETWORK WITH TWO HIDDEN LAYERS OF SIZE 512 EACH AND LENET-5 ON MNIST IN THREE DIFFERENT SPARSITY RATES OF UNPRUNED NETWORK, LGPS-II, BASELINE AND CGS METHOD (WITH BLOCK SIZE: 64 × 64). (LGPS-I GAIN SIMILAR RESULTS AS LGPA-II)

Network		FC Network			LeNet-5		
Sparsity \ Hidden Layer		H1	H2	H3	H1	H2	H3
Unpruned Network		512	512	10	120	84	10
LGPS-II	40(%)	**512**	**512**	**10**	**120**	**84**	**10**
	75(%)	**512**	**512**	**10**	**120**	**83**	**10**
	90(%)	**512**	**512**	**10**	**119**	**82**	**10**
Baseline	40(%)	256	238	10	67	46	10
	75(%)	210	191	10	44	27	10
	90(%)	216	179	10	39	20	10
CGS	40(%)	512	512	10	120	84	10
	75(%)	464	448	10	120	76	10
	90(%)	272	192	10	74	76	10

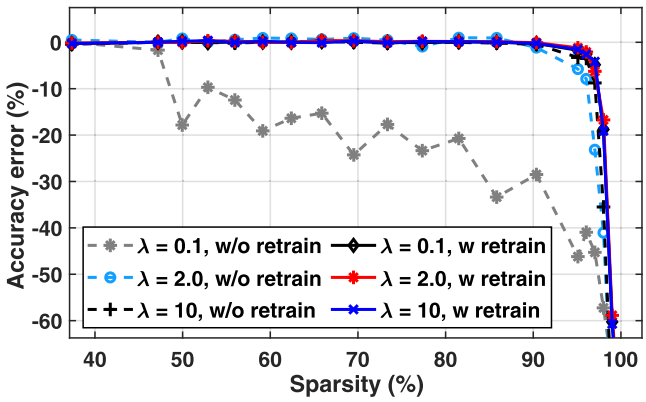


Fig. 7. The comparison between different regularization parameters (λ) for LGPS-I with/without retraining.

of 64 × 64. The results for 40%, 75% and 95% sparsity rate demonstrated in Table II. Based on our observation, rank of the weight matrix does not get affected by pruning the networks using our hardware-aware methodology. In fact, the proposed method preserve the rank of weight matrix closer to the dense matrix (before pruning). Therefore, we conclude that the expressibility of the weight matrices as well as the overall accuracy of the ANNs remain unchanged. However, ranks of matrices in baseline and CGS method have drastically reduced. The reason behind this reduction is that in the baseline, the lower weights' value that have been pruned are mainly belong to the border of the image as the handwritten letters in MNIST dataset where positioned in the middle of the image. In addition, in CGS method, dropping the weights in a block-by-block basis reduces the rank reduction, in particular for higher levels of sparsity.

Moreover, the proposed method is evaluated based on different regularization parameters (λ) to find the best λ to prune the network while preventing over-fitting. We choose different λ values equal to.1, 2 and 10 to evaluate the effects of L2 regularization on network pruning with and without retraining. Based on the results (figure 7), we choose a medium value of $\lambda = 2$ for an optimal trade-off between the

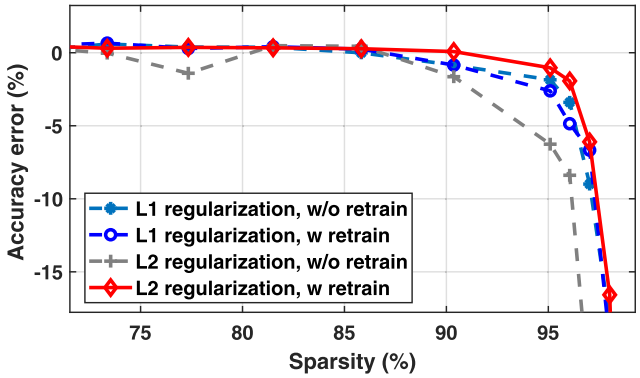
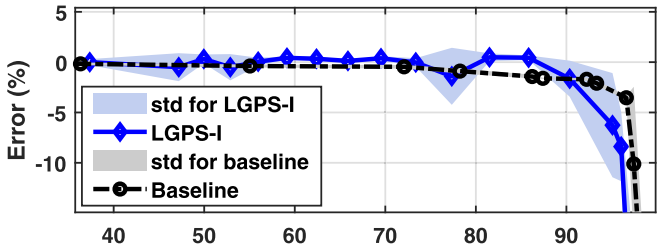


Fig. 8. The comparison between L1 and L2 regularization methods with/without retraining for LGPS-I.

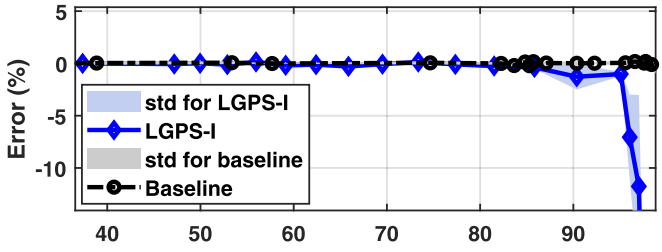
rate of pruning and preventing over-fitting. Lower values of λ cause the network to over-fit while a faster pruning rate that uses a higher value of λ reduces test accuracy. It should be noted that LGPS-II is shown the similar results. In addition, we investigate the effects of L1 and L2 regularization on pruning (figure 8). We chose L2 regularization to prune the networks because it shows better performance with retraining step.

2) Results on Cifar10 and ImageNet Datasets: Cifar 10 is a set of images with 10 different classes. The results of LeNet-5 on Cifar10 for our method in comparison with the baseline is demonstrated in figure 9c. The results demonstrates that LGPS-I can achieve the same result as baseline even in more complex datasets. Also, the baseline method has higher standard deviation compared to LGPS-I in 5 trials. LGPS-II also achieved no accuracy loss during pruning up to 90% sparsity rate. The last dataset that we evaluate our method is the ImageNet dataset [49]. This dataset contains 1000 different classes. The only pre-processing that we have done on this dataset is a single crop with no rotation and we have not performed any other pre-processing or augmentation. The largest batch size, 32 images/batch, used for implementation. We tested MobileNet [50] on ImageNet dataset and the result of accuracy in different sparsity rates of the FC layer illustrates in figure 9e. Finally, we tested our method on VGG-16 network which is a large and complex network. ImageNet dataset is used but initially down-sampled it to 64 × 64 [51]. The reason that we used down-sampled ImageNet is that the network is converged faster since the images are smaller. Second, to show that the pruning method is also work on low resolution images which is sometimes the case in the captured images from edge devices. It should be noted that, in order to fit to the spatial size (i.e. 64 × 64) of down-sampled ImageNet, we have modified VGG-16 by just changing the FC layer size to 2048 and eliminating the last pooling layer. This is due to the fact that the feature size should maintain enough spatial coverage before each pooling layer. The results are illustrated in figure 9d which demonstrate that the proposed pruning method based on LFSR indexing can prune the network while preserving the accuracy as the level of sparsity changes.

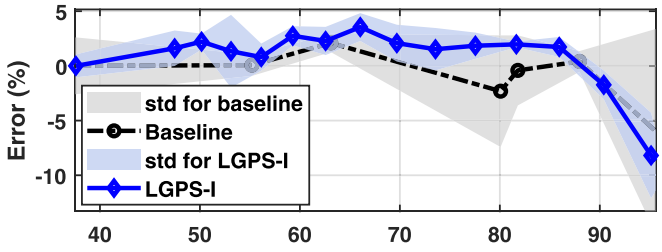
In addition, we compare the accuracy of LGPS-I with CGS method on different networks including MNIST on



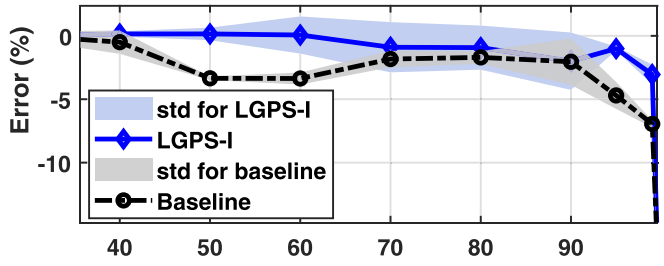
(a) LeNet-300-100 on MNIST



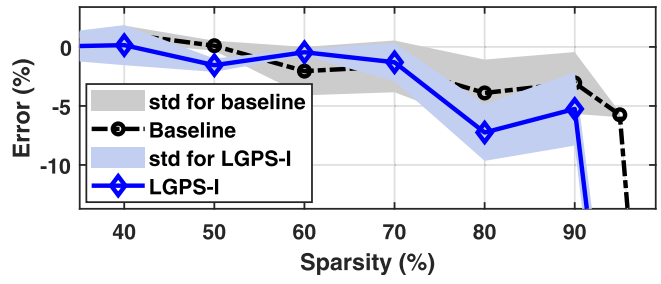
(b) LeNet-5 on MNIST



(c) LeNet-5 on Cifar10



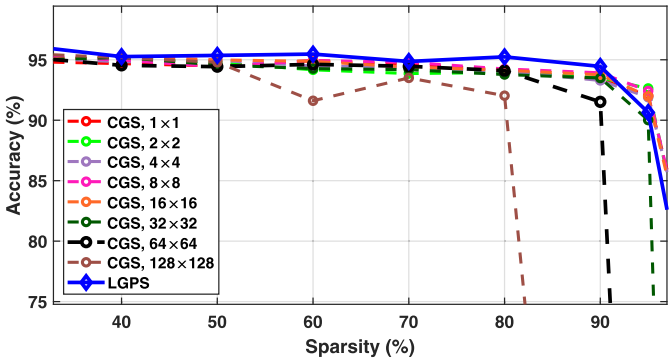
(d) Downsampled ImageNet on VGG-16



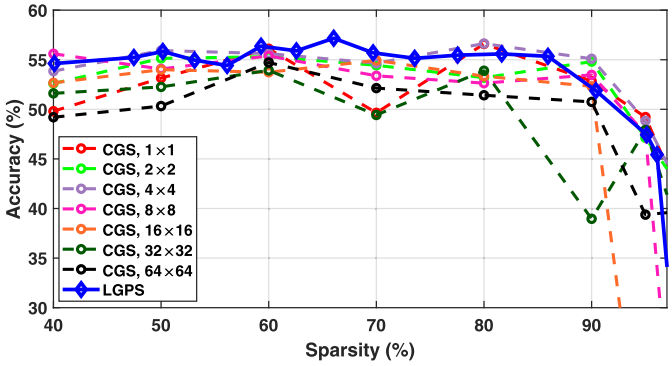
(e) ImageNet on MobileNet

Fig. 9. Accuracy error (%) of LGPS-I in comparison with the baseline method on different sparsity rates. LGPS-II shows similar algorithmic results as well.

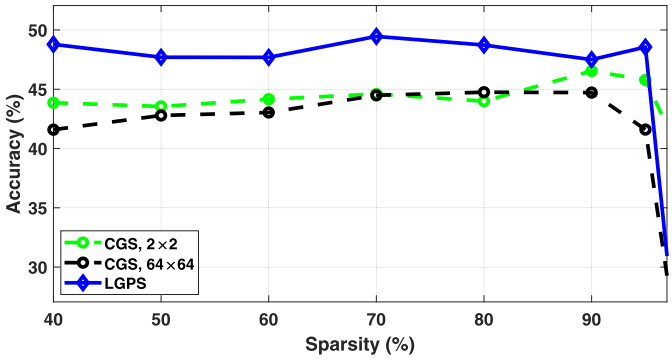
FC network with two hidden layers of size 512 (figure 10a), Cifar10 on LeNet5 (figure 10b) and VGG16 on downsampled Imagenet (figure 10c). The results of CGS method different block sizes from 1×1, 2×2, etc, on MNIST and Cifar10 datasets show that as the size of blocks increases,



(a) FC network on MNIST



(b) LeNet-5 on Cifar10



(c) VGG16 on downsampled ImageNet

Fig. 10. Accuracy error (%) of LGPS-I in comparison with the CGS method on different sparsity rates and block sizes (CGS, block size). LGPS-II shows similar algorithmic results as well.

the accuracy drops. The results demonstrate that LGPS can preserve the accuracy better in different sparsity rates. In addition, the results of LGPS-I compared to SIMD method is demonstrated in figure 11. LGPS-I can preserve the accuracy in different sparsity rates in comparison to SIMD method. LGPS-II shows similar algorithmic results as well, for example, it achieved 98.2% accuracy in 90% sparsity rate when using LeNet-5 network on MNIST.

C. Comparison of Hardware Implementation in 65nm CMOS

In this section, the results of hardware implementation for the baseline and LGPS-I and LGPS-II are demonstrated to estimate key hardware metrics. The baseline and LGPS-I, LGPS-II and CGS architectures have been synthesized

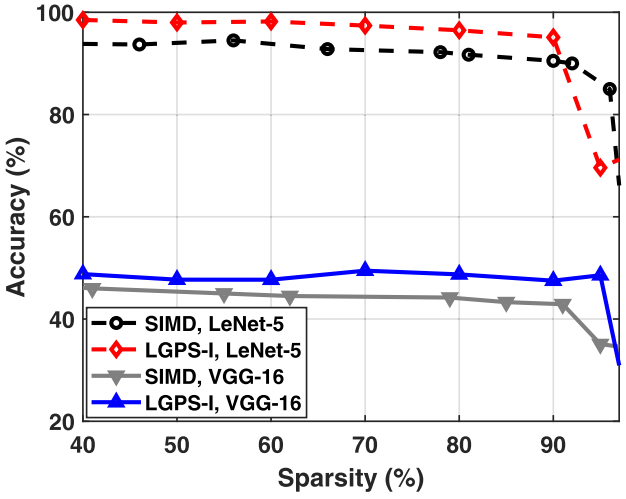


Fig. 11. Accuracy of LGPS-I and SIMD for LeNet-5 on MNIST and modified VGG-16 on downsampled ImageNet in different sparsity rates.

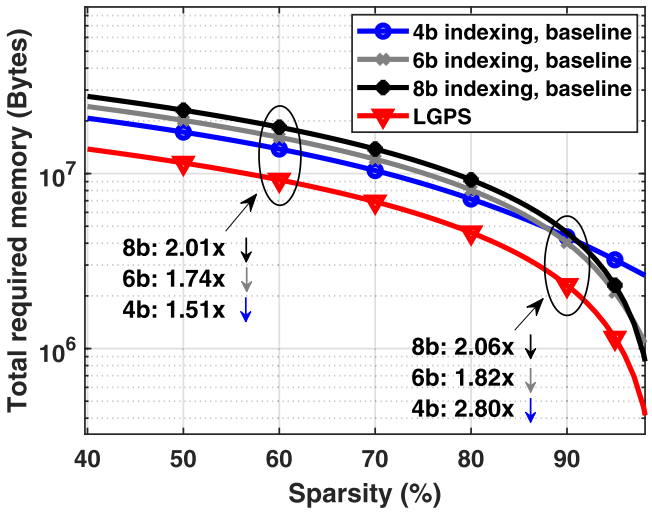


Fig. 12. Total memory required for baseline method (contains indices, pointer and overhead) and LGPS-I for 4, 6 and 8 index bit precision with different sparsity levels. Vertical axis is in Logarithmic scale. LGPS-II shows similar algorithmic results as well.

TABLE III
HARDWARE PARAMETERS

Technology Node	TSMC 65nm
Supply Voltage	1V
Datapath Bitwidth	8b
Index Bitwidth	4b, 8b
Clock Frequency	1GHZ
Memory Bank Size	256B, 512B, 1KB, 4KB

using 65nm CMOS technology. The hardware accelerators are synthesized with Synopsis Design Compiler. The measurements are combinations of synthesized module results and calculations. To make fair comparisons, only single MAC unit computation and full-on-chip SRAM architecture are implemented. Table III represents the implementation parameters.

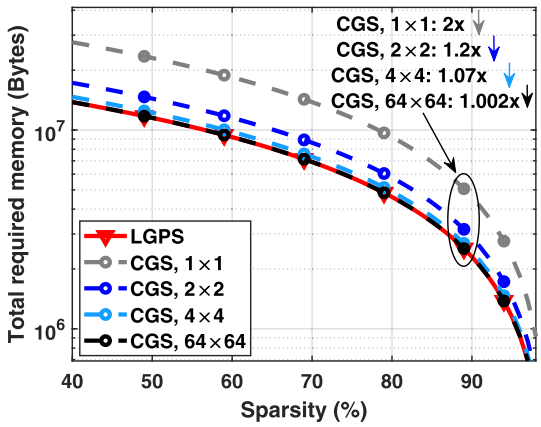
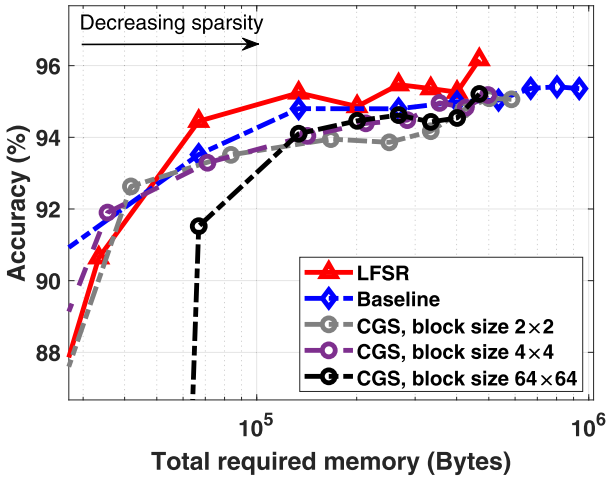
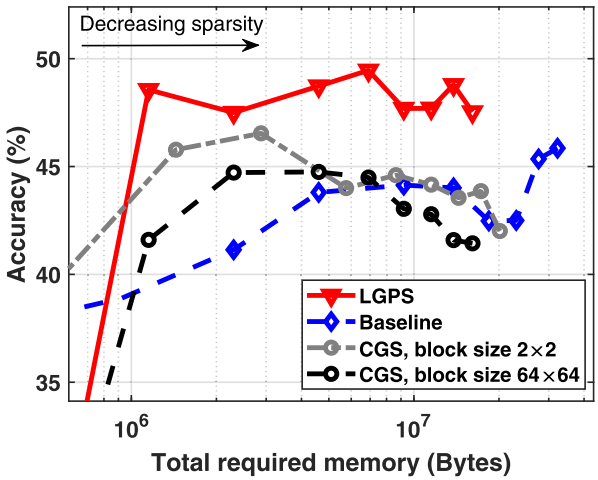


Fig. 13. Total memory required for CGS method for different block sizes (contains indices, pointer and overhead) and LGPS-I with different sparsity levels. Vertical axis is in Logarithmic scale. LGPS-II shows similar algorithmic results as well.



(a) FC network on MNIST



(b) VGG-16 on downsampled ImageNet

Fig. 14. Accuracy vs total memory required for baseline method (contains indices, pointer and overhead), LGPS-I and CGS method for 8-bit index precision with different sparsity levels on (a) fully connected network with two hidden layers of size 512 each and MNIST dataset.

Memory bank sizes are different sizes of synthesized SRAM. The on-chip SRAM can be much larger than these sizes by

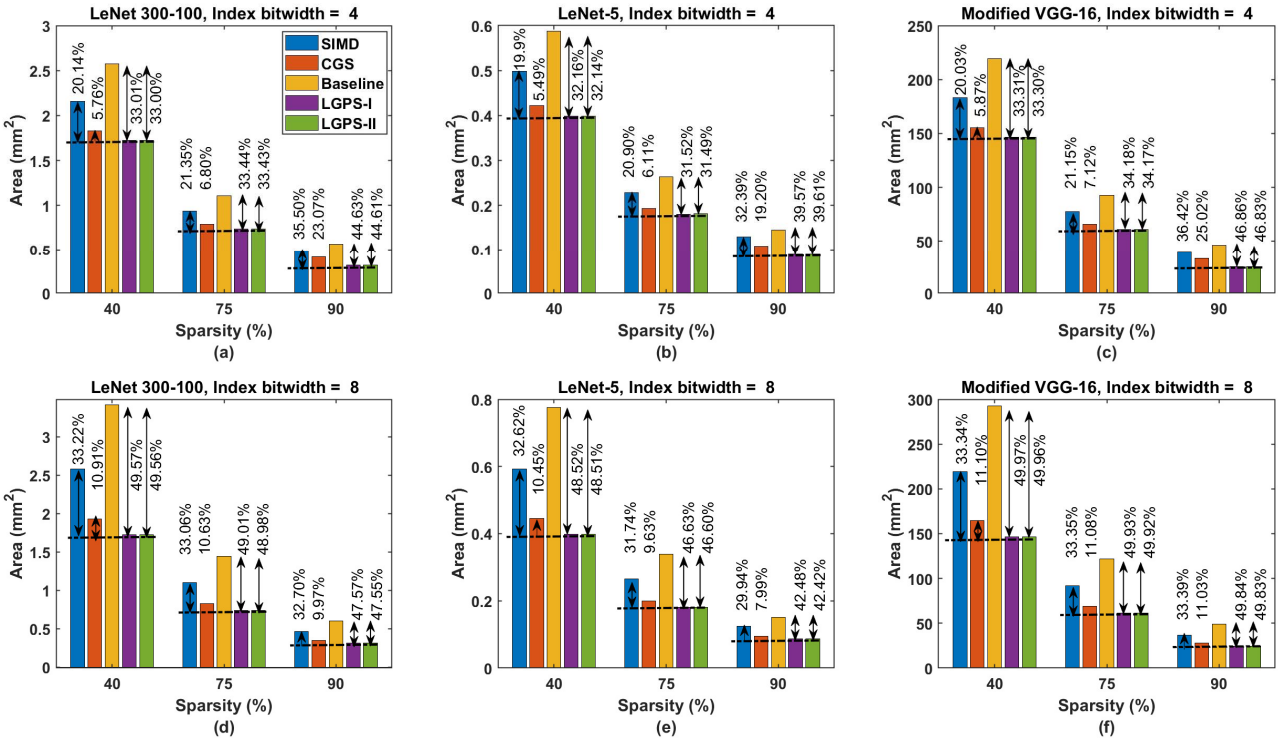


Fig. 15. The measured area (mm^2) of the overall systems consists of accumulator, multiplier, and input/output buffers for baseline method, LGPS-I and LGPS-II, CGS (block size 4×4) and SIMD methods. Measurements performed for three different networks (LeNet-300-100, LeNet-5 and modified VGG-16) at various sparsity and 4-8 bit-width indexing precision.

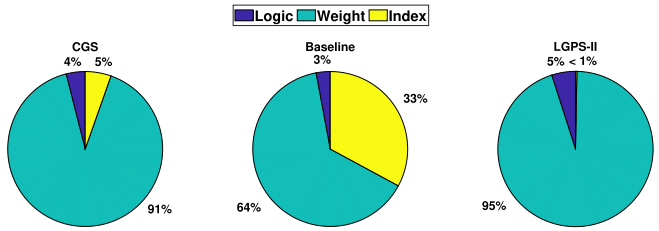


Fig. 16. An example of the breakdown of area among memory, index vs. logic, for LGPS-II, and baseline and CGS method on Lenet300-100 network with 90% sparsity.

having more of them. The pre-layout analysis shows that the required memory-footprint of LGPS-I can be reduced by 1.51× to 2.80× compared to the baseline method. The results for the baseline method with 4, 6 and 8 bit-width representation of the index and the proposed LFSR-based indexing are illustrated in figure 12. It should be noted that LGPS-II shows similar results in terms of required memory as well. In addition, the comparison between the required memory of LGPS-I and CGS method with four different block sizes are illustrated in 13. The index bit-with are chosen to be 8 bit. The results show that LGPS-I requires 2×, 1.2×, 1.07× and 1.002× less memory compared to the CGS with 1×1, 2×2, 4×4 and 64×64 block sizes, respectively. Although the required memory of LGPS-I and CGS with block size 64×64 are very close, CGS could not preserve the accuracy in higher block sizes (Figure 14). The comparison between the accuracy versus the total required memory of LGPS-I, baseline and CGS is shown in figure 14. The memory calculation is done on (a) fully connected network with two hidden layers of 512 neurons each on MNIST and (b) the modified VGG-16 network on downsampled ImageNet with 23M parameters.

TABLE IV
MEASURED LATENCY (ms) OF THE OVERALL SYSTEM FOR OUR LGPS-I, LGPS-II, BASELINE AND CGS

	Sparsity \ Network	LeNet-300-100	LeNet-5	modified VGG-16
LGPS-I	40%	0.95	0.20	82.8
	75%	0.39	0.08	34.5
	90%	0.15	0.03	13.8
LGPS-II	40%	**0.63**	**0.14**	**55.2**
	75%	**0.26**	**0.05**	**23.0**
	90%	**0.10**	**0.02**	**9.21**
Baseline Method	40%	0.81	0.18	69.1
	75%	0.34	0.07	29.1
	90%	0.17	0.04	14.1
CGS	40%	0.65	0.15	56.1
	75%	0.28	0.06	23.6
	90%	0.56	0.03	11.1

In addition to memory measurement, the overall system (memory, multiplier, accumulator and input/output buffers) parameters consists of area and power of the two proposed architecture in comparison to baseline, CGS and SIMD methods are also measured. For CGS method, we chose block size of 4×4 as it shows close accuracy percentage compared to LGPS. Figure 15 illustrated the area measurements for three different networks including LeNet-300-100, LeNet-5 and

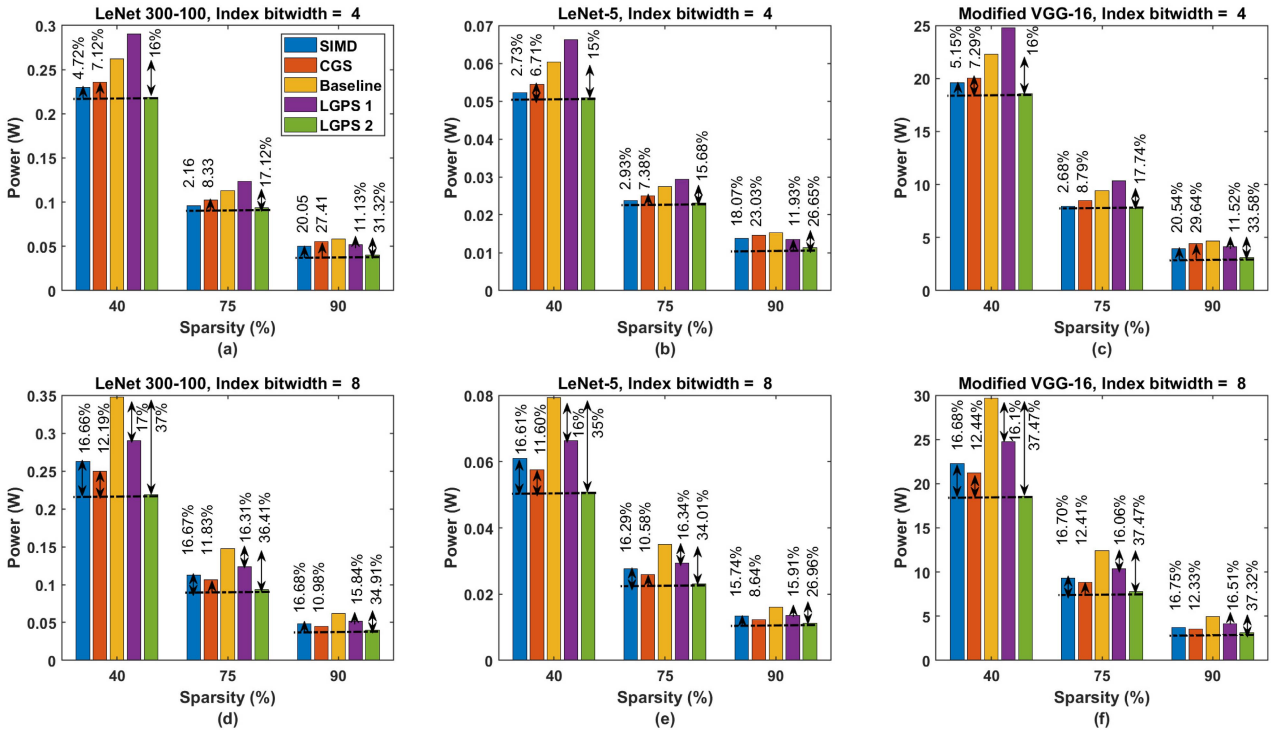


Fig. 17. The measured power (W) of overall systems consists of accumulator, multiplier, and input/output buffers for baseline method, LGPS-I and LGPS-II, CGS (block size 4×4) and SIMD methods methods.Measurements performed for three different networks (LeNet-300-100, LeNet-5 and modified VGG-16) at various sparsity and 4-8 bit-width indexing precision.

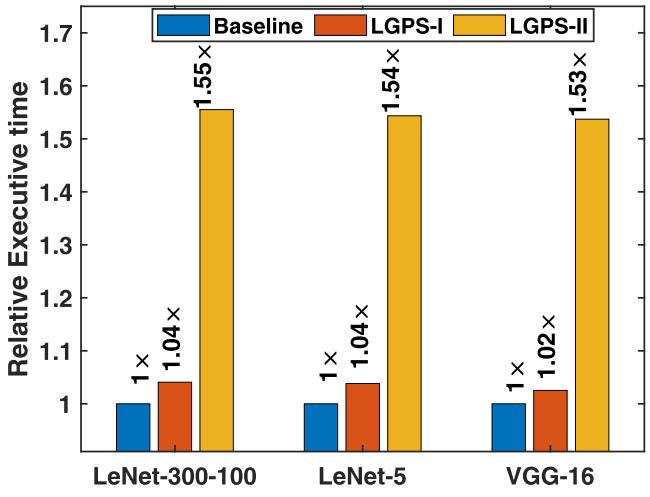


Fig. 18. The relative execution time of overall systems during inference of LGPS-I and LGPS-II with respect to baseline method on LeNet-300-100, LeNet-5 and modified VGG-16 at sparsity rate of 90% and 4 bit-width indexing precision.

TABLE V
MEASURED PERFORMANCE (NUMBER OF FRAMES PROCESSED PER *ms*) OF THE OVERALL SYSTEM FOR LGPS-I, LGPS-II, BASELINE AND CGS

	Sparsity \ Network	LeNet-300-100	LeNet-5	modified VGG-16
LGPS-I	40%	1.05	5.0	0.012
	75%	2.56	12.5	0.028
	90%	6.66	33.3	0.072
LGPS-II	40%	1.58	7.14	0.018
	75%	3.84	20	0.04
	90%	10	50	0.11
Baseline Method	40%	1.23	5.55	0.01
	75%	2.94	14.28	0.03
	90%	5.8	25	0.07
CGS	40%	1.53	6.66	0.017
	75%	3.57	16.6	0.04
	90%	1.78	33.3	0.09

modified VGG-16 at 4 and 8 bit-width index precisions. The results demonstrate that LFSR based indexing has considerable advantages over baseline method in terms of area saving. Our proposed method can save up to 50% area even in large and complex network like modified VGG-16. Figure 16 illustrates an example of the breakdown of area among memory, index vs. logic, for LGPS-II, and baseline and CGS method on Lenet300-100 network with 90% sparsity.

The power measurements of the three networks are also demonstrated in Figure 17 and a maximum of 37.03% power savings across various sparsity rates and indexing bit-widths are reported. Although significant power savings are reported for both of our proposed methods, it should also be noted that in some of the cases (Figure 17, a,b and c) of the LGPS-I, the measured power in slightly higher than the baseline. This is because row/column LFSR indexing introduces additional output buffer access (2 cycle read and 1 cycle write) which increases the power usage. This additional numbers of read and write are calculated and included in our design and results. We address this problem by introducing LGPS-II. In this case, we reduce the number of read and write to/from memory by performing the multiplication/accumulation for each column and then saving the final result in the memory, which reduces

the power consumption significantly. We also evaluate the execution time of LGPS-I and LGPS-II with respect to the baseline method during inference. The relative execution time of overall systems on LeNet-300-100, LeNet-5 and modified VGG-16 at sparsity rate of 90% and 4 bit-width indexing precision is illustrated in figure 18. The results shows that our methods improve the execution time. LGPS-II is 1.53× faster that the baseline pruning method. In addition, we evaluate the latency (in ms) and performance (number of frames processed per second) of the two proposed architecture, the baseline and CGS. The results are shown in Table IV and V.

V. Conclusion

In this article, we propose a DNN accelerator for sparse network generated by LFSR-based indexing. We investigate the performance of two different LFSR-based indexing methods including row/column wise indexing (LGPS-I) and column-wise nested-LFSR based indexing (LGPS-II). The advantage of our proposed LFSR-based pruning is that we solve the problem of irregular sparse network and we no longer need to store the address of the unpruned weights. The propose method enables us to deploy large DNNs on inference and edge devices due to the significant reduction both in memory foot-print and access energy. We have shown that LGPS can preserve the accuracy while pruning DNNs, and can achieve a maximum of 37.35% power savings and 49.84% area savings across varying sparsity rates.

Smaller machine learning models are easy to deploy and update [13]. In this article, we show that the proposed methods can prune a large class of neural network models and drastically reduce their size compared to the sate of the art. This facilitates not only deployment of the models as well as the memory required to store the model on the edge devices. For example, Figure 14 shows the advantage of LGPS as we can now compress LeNet-300-100 and modified VGG-16 to less than 1MB and 100MB storage memory, respectively. We expect such techniques to be practical solutions towards deploying ML models on the edge.

References

[1] M. Verhelst and B. Moons, "Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to IoT and edge devices," *IEEE Solid State Circuits Mag.*, vol. 9, no. 4, pp. 55–65, Fall 2017.

[2] Y. Huang, X. Ma, X. Fan, J. Liu, and W. Gong, "When deep learning meets edge computing," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–2.

[3] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[5] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Comput. Intell. Neurosci.*, vol. 2018, pp. 1–13, Feb. 2018.

[6] S. Ma, X. Zhang, C. Jia, Z. Zhao, S. Wang, and S. Wang, "Image and video compression with neural networks: A review," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 6, pp. 1683–1698, Jun. 2020.

[7] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing [Review Article]," *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, Aug. 2018.

[8] R. Boostani, F. Karimzadeh, and M. Nami, "A comparative review on sleep stage classification methods in patients and healthy individuals," *Comput. Methods Programs Biomed.*, vol. 140, pp. 77–91, Mar. 2017.

[9] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: Review, opportunities and challenges," *Briefings Bioinf.*, vol. 19, no. 6, pp. 1236–1246, Nov. 2018.

[10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[12] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[13] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: http://arxiv.org/abs/1510.00149

[14] S. Zhang *et al.*, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*. Taoyuan, Taiwan: IEEE Press, Oct. 2016, p. 20.

[15] A. Parashar *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 27–40.

[16] Z. Chen, Z. Chen, J. Lin, S. Liu, and W. Li, "Deep neural network acceleration based on low-rank approximated channel pruning," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 4, pp. 1232–1244, Apr. 2020.

[17] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, "Predicting parameters in deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2148–2156.

[18] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 784–800.

[19] X. Ding *et al.*, "Global sparse momentum SGD for pruning very deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 6382–6394.

[20] Z. Chen *et al.*, "Exploiting weight-level sparsity in channel pruning with low-rank approximation," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[21] Y.-H. Chen, T.-J. Yang, J. S. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[22] J. Li *et al.*, "SqueezeFlow: A sparse CNN accelerator exploiting concise convolution rules," *IEEE Trans. Comput.*, vol. 68, no. 11, pp. 1663–1677, Nov. 2019.

[23] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area v2," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 2008, pp. 873–880.

[24] F. Karimzadeh, N. Cao, B. Crafton, J. Romberg, and A. Raychowdhury, "Hardware-aware pruning of DNNs using LFSR-generated pseudo-random indices," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[25] M. Ranzato, C. Poultney, S. Chopra, and Y. L. Cun, "Efficient learning of sparse representations with an energy-based model," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, pp. 1137–1144.

[26] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Deep Learn. Unsupervised Feature Learn. Workshop, NIPS*, 2011.

[27] S. Han *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2017, pp. 75–84.

[28] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1379–1387.

[29] H. Mao *et al.*, "Exploring the regularity of sparse structure in convolutional neural networks," 2017, *arXiv:1705.08922*. [Online]. Available: http://arxiv.org/abs/1705.08922

[30] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 11264–11272.

[31] J. Kepner and R. Robinett, "RadiX-net: Structured sparse matrices for deep neural networks," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2019, pp. 268–274.

[32] S. Chen and Q. Zhao, "Shallowing deep networks: Layer-wise pruning based on feature representations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 12, pp. 3048–3056, Dec. 2019.

[33] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1389–1397.
[34] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 5058–5066.
[35] Z. Zhuang *et al.*, "Discrimination-aware channel pruning for deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 875–886.
[36] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.
[37] R. Yu *et al.*, "NISP: Pruning networks using neuron importance score propagation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9194–9203.
[38] D. Kadetotad, S. Arunachalam, C. Chakrabarti, and J.-S. Seo, "Efficient memory compression in deep neural networks using coarse-grain sparsification for speech applications," in *Proc. 35th Int. Conf. Comput. Aided Design*, Nov. 2016, pp. 1–8.
[39] D. Kadetotad, V. Berisha, C. Chakrabarti, and J.-S. Seo, "A 8.93-TOPS/W LSTM recurrent neural network accelerator featuring hierarchical coarse-grain sparsity with all parameters stored on-chip," in *Proc. IEEE 45th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sep. 2019, pp. 119–122.
[40] A. Marchisio, M. A. Hanif, M. Martina, and M. Shafique, "PruNet: Class-blind pruning method for deep neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–8.
[41] Y. Sun, X. Wang, and X. Tang, "Sparsifying neural network connections for face recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4856–4864.
[42] L. Mauch and B. Yang, "A novel layerwise pruning method for model reduction of fully connected deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 2382–2386.
[43] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN pruning to the underlying hardware parallelism," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 548–560, 2017.
[44] R. Mita, G. Palumbo, S. Pennisi, and M. Poli, "A novel pseudo random bit generator for cryptography applications," in *Proc. 9th Int. Conf. Electron., Circuits Syst.*, vol. 2, 2002, pp. 489–492.
[45] T. W. Cusick and P. Stanica, *Cryptographic Boolean Functions and Applications*. New York, NY, USA: Academic, 2017.
[46] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP Laboratories*, vol. 27, p. 28, Apr. 2009.
[47] M. Verhelst and B. Murmann, "Machine learning at the edge," in *Proc. NANO-CHIPS*. Cham, Switzerland: Springer, 2020, pp. 293–322.
[48] J. Gu, Z. Zhao, C. Feng, M. Liu, R. T. Chen, and D. Z. Pan, "Towards area-efficient optical neural networks: An FFT-based architecture," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 476–481.
[49] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[50] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861
[51] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," 2016, *arXiv:1601.06759*. [Online]. Available: http://arxiv.org/abs/1601.06759

Foroozan Karimzadeh (Graduate Student Member, IEEE) received the B.S. degree in electrical and computer engineering and the M.Sc. degree in biomedical engineering from Shiraz University, Iran, in 2013 and 2016, respectively. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, under the supervision of Dr. Raychowdhury. Her main research interest includes developing novel algorithms and hardware for energy efficient machine learning techniques.

Ningyuan Cao (Student Member, IEEE) received the B.S. degree in electrical engineering from Shanghai Jiaotong University, Shanghai, China, in 2013, and the M.S. degree in electrical engineering from Columbia University, New York City, USA. He is currently pursuing the Ph.D. degree with the School of Electrical and Computer Engineering, Georgia Institute of Technology.

From June 2014 to May 2015, he was a Staff Research Associate with the Columbia Integrated Circuit Laboratory (CISL), Columbia University. His research interests include low-power machine learning ASIC design, wireless sensor SOC design, and so on. These interests are directed towards enhancing computation, communication, and control performance on resource-constrained edge devices using advanced circuit and architecture technology. His on-going Ph.D. research has so far resulted in numerous top-tier IEEE journals/conference publications, including ISSCC, JSSC, TIE, ISCAS, TCAS-I, IMS, *Sensors*, and so on. He will continue his exploration in the fields to enable next-generation edge intelligence, including brain-inspired edge computation, hardware security, and RF ML systems.

Brian Crafton (Student Member, IEEE) received the B.S. degree in computer engineering from Northeastern University, Boston, MA, USA, in 2017. He is currently pursuing the Ph.D. degree with the Georgia Institute of Technology under the supervision of Dr. Raychowdhury. He held various internship positions with Advanced Micro Devices, Boxborough, MA, USA, and Intel, San Francisco, CA, USA. His research interest includes in-memory and near-memory computing for energy efficient machine learning.

Justin Romberg (Fellow, IEEE) received the B.S.E.E., M.S., and Ph.D. degrees from Rice University, Houston, TX, in 1997, 1999, and 2004, respectively. From Fall 2003 to Fall 2006, he was a Post-Doctoral Scholar in applied and computational mathematics with the California Institute of Technology. He spent the Summer of 2000 as a Researcher at Xerox PARC, the Fall of 2003 as a Visitor at the Laboratoire Jacques-Louis Lions, Paris, and the Fall of 2004 as a fellow at the UCLA's Institute for Pure and Applied Mathematics. In the Fall of 2006, he joined the Georgia Tech ECE Faculty. He is currently a Professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology. In 2008, he received the ONR Young Investigator Award, and the PECASE Award and the Packard Fellowship, in 2009. In 2010, he was named as a Rice University Outstanding Young Engineering Alumnus. From 2006 to 2007, he was a Consultant for the TV show "Numb3rs." He was an Associate Editor of the IEEE TRANSACTIONS ON INFORMATION THEORY from 2008 to 2011 and the *SIAM Journal on Imaging Science* from 2013 to 2018. He has been an Associate Editor of the *SIAM Journal on Mathematics of Data Science* since 2018.

Arijit Raychowdhury (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2007.

His industry experience includes five years as a Staff Scientist with the Circuits Research Laboratory, Intel Corporation, Portland, OR, USA, and a year as an Analog Circuit Researcher with Texas Instruments Inc., Bengaluru, India. He joined the Georgia Institute of Technology, Atlanta, GA, USA, in 2013, where he is currently an Associate Professor with the School of Electrical and Computer Engineering and also holds an ON Semiconductor Junior Professorship. He holds more than 25 U.S. and international patents and has published over 100 papers in journals and refereed conferences. His research interests include low-power digital- and mixed-signal circuit design, device–circuit interactions, and novel computing models and hardware realizations.

Dr. Raychowdhury was a recipient of the Dimitris N. Chorafas Award for Outstanding Doctoral Research in 2007, the Intel Labs Technical Contribution Award in 2011, the Best Thesis Award from the College of Engineering, Purdue University, in 2007, the Intel Early Faculty Award in 2015, the NSF CISE Research Initiation Initiative Award (CRII) in 2015, and multiple best paper awards and fellowships.