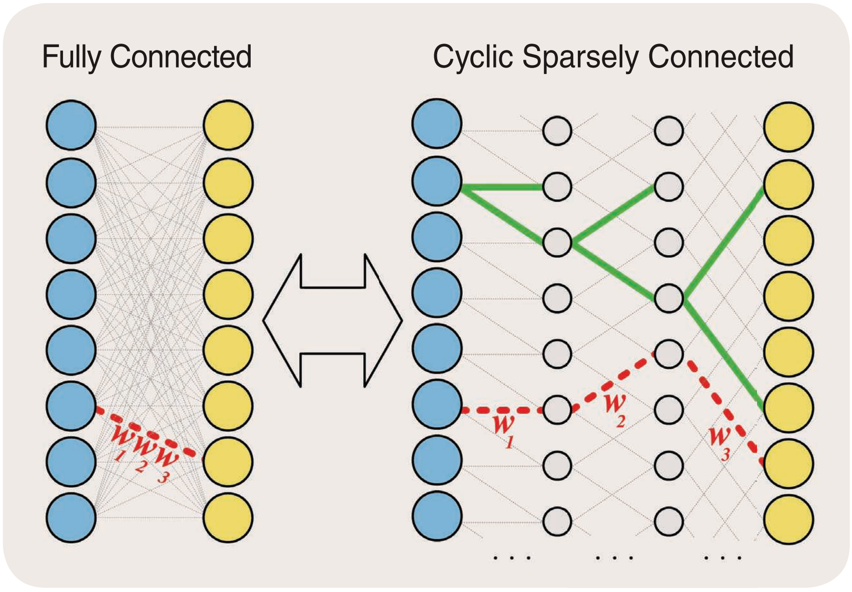
Morteza Hosseini, Nitheesh Manjunath, Uttej Kallakuri, Hamid Mahmoodi, Houman Homayoun, and Tinoosh Mohsenin

Cyclic Sparsely Connected Architectures

From foundations to applications



Deep neural networks (DNNs) are compute-intensive nonlinear mathematical functions that employ matrix/tensor operators at their core to identify temporal and/or spatial correlations within input data. Common techniques, such as pruning, quantization, and compact model design, have been proposed by researchers and extensively utilized by developers to reduce the computation and bulky size of DNNs.

Under the category of compact model design, cyclic sparsely connected (CSC) architectures [1] are structurally sparsified graphs that can be used to effectively compress DNNs as an alternative to pruning methods with the advantage of imposing less of a memory footprint. CSC architectures have a memory/computation complexity of $\mathcal{O}(N \log N)$ that can be used as an overlay for a fully connected (FC) of $\mathcal{O}(N^2)$, where N is the number of input–output (I/O) nodes given an equally sized FC layer.

Digital Object Identifier 10.1109/MSSC.2021.3111431
Date of current version: 17 November 2021

Throughout this article, we focus on the complexity of FC layers and how CSC architectures can be used as a compact overlay for them in DNNs. In the end, we briefly remark on how CSC architectures can be used for compact convolution layers, too.

Motivation and Background

Pruning Versus Structured Sparsity

Despite their effectiveness, pruning methods result in the irregularity of the pattern of nonzero weights in the pruned model of a neural network, necessitating an additional indexing. Thus, their compressed model has more parameters than the sheer number of nonzero weights, and their implementation is deteriorated by the model decompression.

As an example, our experiments on the NVIDIA TX2 GPU at a clock rate of 1.3 GHz indicates that a matrix–vector multiplication using a matrix of size 16,384 by 16,384 can be implemented by the NVIDIA CuBLAS library, which is a GPU-accelerated implementation library of the basic linear algebra subroutines (BLAS), with a performance of approximately 11.4 giga operations per second (GOPS), whereas the implementation of the same matrix with 95% zero values compressed with the compressed sparse row (CSR) storage format can be executed using the CuSPARSE library, which is a GPU-accelerated implementation library for sparse matrices, with a performance of approximately 3.9 GOPS that merely operates over the 5% nonzero values.

Despite the fact that CuSPARSE can implement the latter problem 6.8 times faster than the CuBLAS, its advantage is gained by the existence of 20 times fewer operations, yet its degraded performance is caused by the decompressing algorithm applied to the matrix compressed by the CSR format. In structured sparsity, on the other hand, indexing nonzero values is eliminated, and the implementation can be handled as high performance as dense matrices.

In structured sparsity, on the other hand, indexing nonzero values is eliminated, and the implementation can be handled as high performance as dense matrices.

Low-Bit-Width Neural Networks

When quantization and pruning methods are used in tandem, unlike the DNN weights and activations, the extra indexing memory imposed by the pruning method is not quantizable, and, therefore, the extra indexing memory might be intolerable in resource-bound hardware that employs low bit width, such as binary-/ternary-weight neural networks. Pruning such neural networks can result in a larger model size as compared to their uncompressed models.

For instance, if a ternary-weight 16,384-by-16,384 matrix with 90% zero values is compressed with a coordinate format, it requires 97 MB of storage (dedicating 1 and 28 b to the nonzero value and its index, respectively), whereas its uncompressed model requires 67 MB (dedicating 2 b per matrix weight). The second motivation of devising a CSC architecture is to develop and employ structurally compact models that require no indexing for low-precision neural networks.

Problem Statement and Formulation

Throughout this work, we use the italic lowercase letter x to represent generator polynomials (e.g., $p(x)=1+x+x^2$), italic capital letters (e.g., N) for integer values, and bold uppercase letters for matrices (e.g., $\mathbf{W}$) and vectors (e.g., $\mathbf{X}$). We use italic lowercase letters in brackets for the elements of a matrix or a vector (e.g., $\mathbf{W}[i, j]$).

Inspired by the butterfly diagrams of the radix-2 fast Fourier transform (FFT), we seek SC layers composed of a unidirectional sequence of layers: an Input layer, $L-1$ layers in between referred to as *support layers*, and an Output layer. We denote the first (Input) and last (Output) layers by capitalizing their first letters. All consecutive layers in the graph are connected to each other via edges (synapses). We call this graph *homogeneous* if the size of every layer is N nodes and its intermediate connectivity is such that the fan-out of every node in every layer excluding the Output layer as well as the fan-in of every layer excluding the Input layer is exactly F. Therefore, the total number of edges, E, in this graph is equal to

$$E = NFL. \tag{1}$$

For every layer in the graph, we define an adjacency matrix, $\mathbf{A}$, whose length and width are equal to the input and output sizes of the layer, respectively (N in this statement) and whose elements $\mathbf{A}[i, j]$ indicate the number of edges that connect the input node i to the output node j. Therefore, NF out of N^2 elements of the adjacency matrix of every support layer in our problem statement are ones, and the rest are zeros, indicating a sparsity of F/N for the support layer. We then define a merit of connectivity, C, and constrain the graph to provide C and only C paths between every pair of nodes chosen arbitrarily from the graph Input and Output layers. For this homogeneous graph, one can show

$$F^L = NC. \tag{2}$$

Proof

Starting from the first layer, every neuron from the input layer has F possibilities in every forward hop through L layers to reach to the output layer nodes. Thus, NF^L paths exist in total. Also, between every Input/Output pair exist C paths; therefore, in total there are CN^2 paths. Thus, $F^L = NC$.

The objective of homogeneity is to provide a basis where every arbitrary node in the graph is equally exploited, every arbitrary pair of nodes from the Input and Output layers are equally connected, the flux through the support layers is fairly equal, and the rank of the transforming matrix has the potential to remain full. This foundation is proposed as an overlay and replacement for FC layers and defined such that an FC layer of size N by N is concluded as a special case in which $F=N$, $L=1$, and $C=1$. Combining (1) and (2), a logarithmic relationship between the number of edges and size of the layers is inferred as

$$E = NF\log_F(NC), \tag{3}$$

where E is the number of edges that corresponds to the number of nonzero elements of the L layers and governs the number of multiply–accumulate (MAC) operations in the graph. As an example, given $F=2$ and $C=1$, (2) and (3) infer $L=\log_2 N$ and $E=2N\log_2 N$, which is the case in radix-2 butterfly diagrams. Figure 1(a)–(e) depicts an FC layer, the problem statement, and a heterogeneous and two homogeneous SC graphs, respectively.

I/O Adjustment

To replace an FC of Input size N_I and Output size N_O with an SC that has a different value for N, we tile and truncate only the Input and Output layers of the SC to match them to those of the FC layer. To adjust the Input layer to an input vector of size N_I, we remove rows from the bottom of the adjacency matrix if $N_I < N$ (truncation), or

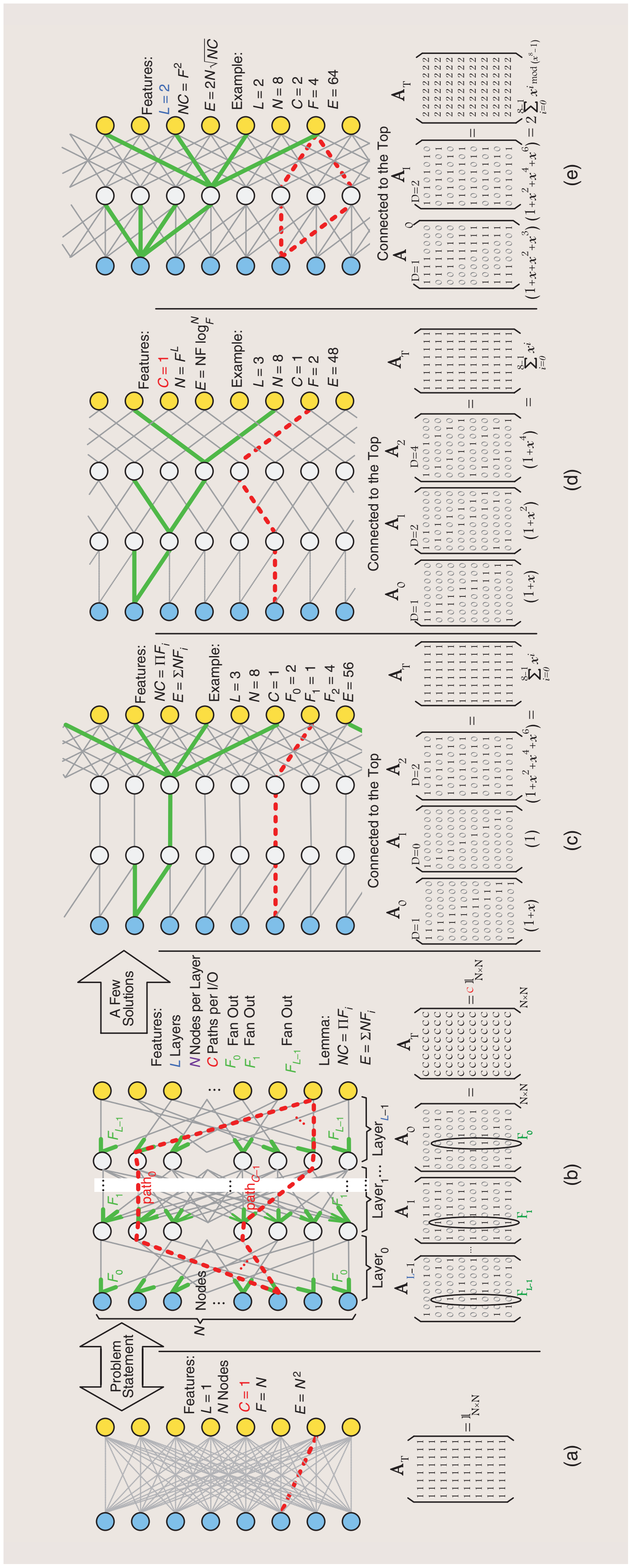


FIGURE 1: Exploring an architecture that approximates a weighted dense layer into the factorization of weighted sparse layers. (a) An FC with size *N*. (b) The problem statement highlights a directed acyclic graph (DAG) with *L* layers, *N* nodes per layer, equally fanned-out layers, and equally connected nodes with *C* paths between its arbitrary I/O nodes. A few solutions of this can be a (c) heterogeneous CSC graph that satisfies the problem statement, (d) homogeneous CSCI graph where C = 1, and (e) homogeneous CSCII graph where the number of layers (*L*) is two. For all of the DAGs, bi-adjacency matrices at the bottom of the figure highlight the connectivity of the nodes, the product of which reveals the number of paths for all I/O nodes. Generator polynomials define the formation of each graph. CSCI: a CSC with the connectivity equal to one; CSCII: a CSC with the layers equal to two.

we recursively copy from the first consecutive rows of the adjacency matrix and add them to its bottom if $N_I > N$ (tiling) until its number of rows is equal to N_I.

For the Output layer to be adjusted to an output of size N_O, we manipulate the columns of the last adjacency matrix similarly. By doing so, the connectivity between the adjusted Input and Output layers still remains C, but the fan-in of the layer following the adjusted Input and the fan-out of the layer preceding the adjusted Output layers deviate from F. For an SC with parameters N, F, L, and C as well as adjusted Input and Output sizes of, respectively, N_I and N_O, the number of edges is

$$E_{\text{adj}} = NF(L-2) + N_I F + N_O F. \qquad (4)$$

Compression Rate

Since the adjusted SC graph with E_{adj} synapses is going to substitute an FC layer with N_I Input nodes, N_O Output nodes, and $N_I N_O$ synapses, the compression rate, denoted by γ, is equal to

$$\gamma = \frac{N_I N_O}{E_{\text{adj}}}. \qquad (5)$$

Clearly, γ should be smaller than one to indicate compression. It can also be considered as the average number of synapses from the SC that contribute to forming every synapse from the equivalent FC. If the SC is homogeneous, each of its synapses equally contributes to the formation of every synapse from the equivalent FC layer. For an FC, $\gamma = 1$, indicating no compression and that all connections are independent. For a homogeneous SC, $\gamma = N/(F \log_F(NC))$, which is the maximum given $C = 1$ and $F = e$, where e is Napier's constant (≈ 2.718). For an SC with adjusted I/O, the largest value for γ is given by the smallest E_{adj}, which is derived for $L = 2$ and $F = 1$, that results in $\gamma_{\text{most}} = (N_I^{-1} + N_O^{-1})^{-1}$ and is equivalent to a rank-one FC layer.

Solution to the Problem Statement

There is no unique solution for the graphs that meet the problem statement.

We show that circulant matrices generated from generator polynomials, as in cyclic error-correcting codes, give a set of solutions for the bi-adjacency matrices of the layers that satisfy all of the requisites in our problem statement. From here on, we call these graphs *CSC architectures*.

The second motivation of devising a CSC architecture is to develop and employ structurally compact models that require no indexing for low-precision neural networks.

Suppose the bi-adjacency matrix $\mathbf{A}_l$ $(0 \le l \le L-1)$ of every factorized layer l in our CSC graph has a generator polynomial $p_l(x)$ that has F terms, which generates a cyclic bi-adjacency matrix of block length N. It can be shown that the product of the L bi-adjacency matrices attributed to the L layers is a matrix $\mathbf{A}_T$, where $\mathbf{A}_T(i,j)$ represents the number of paths between the Input node i and Output node j of the CSC graph.

In the problem statement, the connectivity should be equal to C for all arbitrary pairs of Input and Output nodes, and, thus, $\mathbf{A}_T = C\mathbf{J}$, where $\mathbf{J}$ is an N-by-N all-one matrix. The all-C matrix $\mathbf{A}_T$ can be attributed to another generator polynomial $p_T(x) = C\Sigma_{i=0}^{N-1} x^i$, which constructs a cyclic matrix as follows: the first row of the matrix corresponds to the coefficients of the polynomial—represented as big-endian in this article—and, then, every next row is a cyclic right shift of its previous row. The cyclic bi-adjacency matrices in this article are not ideals, as in ideal cyclic codes, and might have nondistinctive rows. We provide two different factorization sets of $p_T(x)$, connectivity equal to one (CSCI) and layers equal to two (CSCII).

CSCI

If $C=1$, and F, L, and N are such that $N = F^L$, then $p_T(x) = \Sigma_{i=0}^{N-1} x^i$ can be factorized as follows:

$$\begin{cases} \sum_{i=0}^{N-1} x^i = \prod_{l=0}^{L-1} p_l(x) \\ p_l(x) = \sum_{i=0}^{F-1} x^{D_l.i}, \quad D_l = F^l. \end{cases} \tag{6}$$

By assigning $p_l(x)$ as the generator polynomial of the factorized layer l, the CSC layers of the graph are completely defined. D_l is the dilation parameter that indicates the distance between elements of value one in the first row of the bi-adjacency matrix of layer l. It also represents the dilation between the fanned-out edges in its corresponding layer. For small values of F near Napier's constant (e.g., two or three) and, consequently, more layers (e.g., $\log_2 N$ or $\log_3 N$), the least number of synapses results but at the expense of elongating the graph, which, by itself, will increase the memory communication during hardware implementation.

Figure 1(d) shows a CSCI graph with three layers and generator polynomial $p_l(x) = \Sigma_{j=0}^{2-1} x^{2^l j}$ for the layer l. For $F=2$ and $C=1$, the numbers of edges and MAC operations are equal to $2N\log_2 N$, which makes the computation complexity $\mathcal{O}(N\log_2 N)$. We evaluate LeNet-300-100, which is an early neural network proposed by Yann LeCun, on MNIST, which is a data set of handwritten digits, by replacing FC layers with CSCI.

CSCII

If $L=2$, and F, C, and N are integers to satisfy $NC = F^2$, then $p_T(x) = C\Sigma_{i=0}^{N-1} x^i$ can be factorized as follows:

$$\begin{cases} C\sum_{i=0}^{N-1} x^i = p_0(x).p_1(x) \bmod (x^N - 1) \\ p_0(x) = \sum_{i=0}^{F-1} x^{D_0.i}, \quad D_0 = 1 \\ p_1(x) = \sum_{i=0}^{F-1} x^{D_1.i}, \quad D_1 = \dfrac{F}{C}. \end{cases} \tag{7}$$

By assigning $p_0(x)$ and $p_1(x)$ to the first and second layers of the graph, respectively, the CSCII graph is defined. In CSCII graphs, $E = 2N\sqrt{NC}$, which declares compression given $C < N/4$ and a computation of $\mathcal{O}(N\sqrt{N})$.

Figure 1(e) shows a CSCII graph with $C=2$ and generator polynomial $p_l(x) = \Sigma_{j=0}^{4-1} x^{2^l j}$ for layer l. In the "CSC Architectures for FC Layers" section, we evaluate replacing FC layers of AlexNet, which is a popular neural network proposed by Alex Krizhevsky, with CSCII architectures.

CSC Architectures for FC Layers

In this section, we denote the computation of weighted matrices trained using a CSC architecture. We then explain a bottom-up algorithm to seek appropriate CSC architectures. For simplicity, we ignore the term bias in all equations in this section and assume the numbers of nodes in the I/O of the FC layers are all equal to N. As a result, vectors/matrices in this section are $\mathbf{Y} \in \mathbf{IR}^N$, $\mathbf{W} \in \mathbf{IR}^{N\times N}$, $\mathbf{X} \in \mathbf{IR}^N$. If layers are not equally sized, we adjust their I/O according to the "I/O Adjustment" section.

A standard FC layer is representable with a dense weight matrix $\mathbf{W}$ whose operation on an input vector $\mathbf{X}$ is equivalent to a matrix–vector multiplication that generates a vector $\mathbf{Y}$, subject to

$$\mathbf{Y}[i] = (\mathbf{WX})[i] = \sum_{j=0}^{N-1} \mathbf{W}[i,j]\mathbf{X}[j], \tag{8}$$

which is a computation of $\mathcal{O}(N^2)$. If $\mathbf{W}$ is factorizable into L matrices, or, correspondingly, if a cascade of L CSC layers with linear activations as well as parameters N and F are weighted to equate a factorizable FC layer, then the equivalent $\mathbf{W}_T = \Pi_{l=0}^{L-1}\mathbf{W}_l$. Because of the associative property of matrix–matrix multiplication, the computation of $\mathbf{W}_T\mathbf{X}$ can start from the rightmost matrix–vector multiplication and propagate to the leftmost matrix. As such,

Using a total of four MAC units to perform 32 MAC operations in eight clock cycles indicates that the design meets the maximum achievable performance with the available resources.

the operation between an incoming input vector $\mathbf{X}_l$ with the lth layer that has a weight matrix $\mathbf{W}_l$ with hyperparameters F and D_l results in an output vector $\mathbf{Y}_l$ with elements calculated as

$$\mathbf{Y}_l[i] = \sum_{j=0}^{F-1} \mathbf{W}_l[i,(i+jD_l) \bmod N] \times \mathbf{X}_l[(i+jD_l) \bmod N], \quad (9)$$

which inherently skips the zero values in $\mathbf{W}_l$ by taking only the nonzero values that are dilated D_l elements apart. Thus, the computation is of $\mathcal{O}(NF)$ for one single $\mathbf{W}_l\mathbf{X}$ and of $\mathcal{O}(NF\log_F(NC))$ for $\mathbf{W}_T\mathbf{X} = (\Pi_{l=0}^{L-1}\mathbf{W}_l)\mathbf{X}$, which corresponds to a cascade of L homogeneous weighted CSC layers with connectivity C and fan-out F operating on input vector $\mathbf{X}$.

For every CSC weight matrix $\mathbf{W}_l$, we define a compressed format $\hat{\mathbf{W}}_l \in \mathrm{IR}^{N\times F}$, which is an N-by-F matrix that contains only NF nonzero entries of $\mathbf{W}_l$, with the following rearrangement:

$$\hat{\mathbf{W}}_l[i,j] = \mathbf{W}_l[i,(i+jD_l) \bmod N]. \quad (10)$$

As a consequence of this, (9) can be altered as

$$\mathbf{Y}_l[i] = \sum_{j=0}^{F-1} \hat{\mathbf{W}}_l[i,j]\mathbf{X}_l[(i+jD_l) \bmod N], \quad (11)$$

to which we refer as a *cyclic dilated matrix–vector multiplication* for a CSC layer. For $F=N$ and $D=1$ and given a new rearrangement of a dense $\mathbf{W}_l$ into a compressed format $\hat{\mathbf{W}}_l$, (11) corresponds to (8), revealing a novel approach to computation in standard matrix–vector multiplications using cyclic dilated matrix–vector multiplication.

Bottom-Up Training

The DNN model with FC layers is trained first, and a reference accuracy λ_{FC} is obtained. Then, the FC is replaced with an adjusted CSC architecture, starting from the most compressed CSC architecture and directed toward compression reduction. With no strict definition, we consider that the most compressed CSC architecture is given by small values for F (e.g., two, three, and four) if adopting a CSCI architecture and by $C=1$ if adopting a CSCII architecture. The experiments begin by targeting the bulky layers of a DNN. In each experiment, if the accuracy λ_{CSC} is ϵ less than λ_{FC}, the procedure is terminated, and the CSC model is accepted. If no CSC layer replacement satisfies the accuracy loss, the original layer is restored. The criterion ϵ is chosen to be 2% in all experiments of this article.

After training the CSC layers embedded in the DNN, every weight matrix $\mathbf{W}_i$ from layer i has nonzero weights located at corresponding nonzero elements of its adjacency matrix $\mathbf{A}_i$. The weighted CSC layers, which have substituted one FC layer, transform the Input to the Output linearly, given linear activation for support layers, and can be composed into an equivalent FC layer with $\mathbf{W}_T = \Pi_{i=0}^{L-1}\mathbf{W}_i$. It is axiomatic that an arbitrary FC layer with a weight matrix $\mathbf{W}_T$ cannot be losslessly decomposed into CSC layers that have fewer synapses in total. However, since training does not usually produce a concrete solution for $\mathbf{W}_T$, we use the backpropagation algorithm to learn good weights for the CSC layers.

Training Experiments

As depicted in Figure 2, we replace the FC layers of LeNet-300-100 and AlexNet with CSCI and CSCII layers and fine-tune them in 50 epochs. For AlexNet, the pretrained weights obtained from the baseline model are frozen for the convolution layers, and only the dense layers are retrained. Tables 1 and 2 show the compression and accuracy of the selected CSC configurations for the two DNNs in this work and compare them with related nonstructural pruning methods from the literature. It is noteworthy that, in all nonstructural pruning methods, there exists another implicit memory space for storing nonzero weight indexes. Thus, if every nonzero weight requires an index with the same number of bits, then the actual compression in nonstructural pruning methods is approximately half the reported rate.

CSC for Convolution Layers

CSC architectures can be used for convolution layers as well. Similar to the scalar values that represent weighted synapses and I/O nodes of an FC layer, a traditional 2D convolution layer can be viewed as an FC graph, as in Figure 1(a), in which every synapse and I/O node represent a 2D kernel and channel, respectively. Thus, every output node is a 2D channel that results from summing over all of its connecting kernels operating on their connected input channels.

With that perspective, each of the CSC graphs in Figure 1(c)–(e) can accommodate a combination of 3×3 and 1×1 (or 3×1, 1×3, and 1×1) kernels to approximate a traditional 2D convolution with 3×3 kernels. In [5], CSC architectures are used to compress convolution layers of an AlexNet, resulting in a model compressed by seven and three times and in size and computation, respectively.

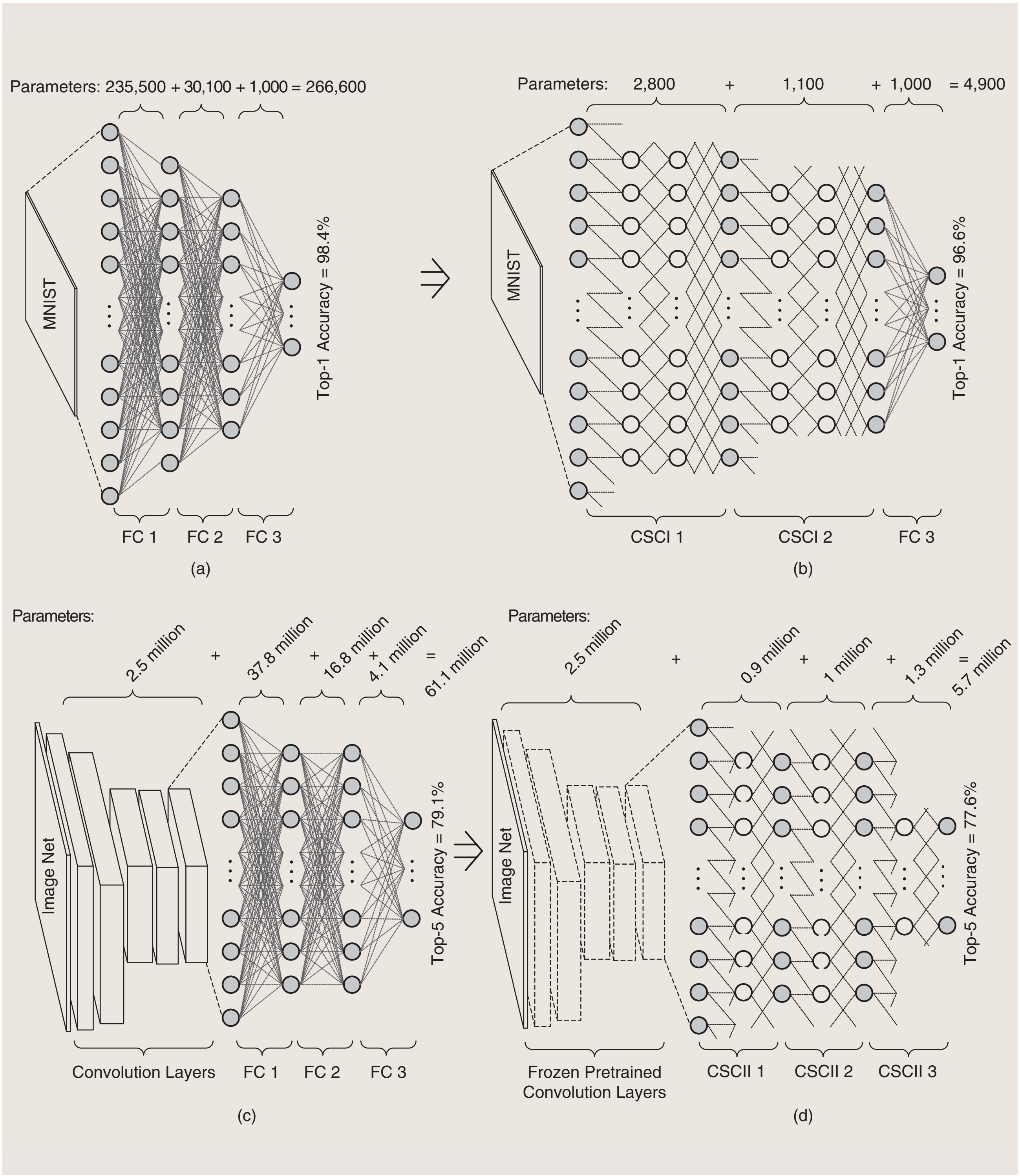
Hardware

Accelerator Architecture

Similar to GPUs that are engineered to accelerate matrix operations, we design a hardware, referred to as the *CSC accelerator processor* (*CSC-AP*), to efficiently implement (11), which performs a cyclic dilated matrix–vector multiplication, i.e.,

$\mathbf{Y}_l[i] = \Sigma_{j=0}^{F-1} \hat{\mathbf{W}}_l[i, j] \mathbf{X}_l[(i + jD_l) \bmod N]$. Our design, briefly depicted in Figure 3 and described in the Verilog hardware description language, comprises three main blocks:

- an array of processing engines (PEs) that perform MAC operations and accommodate a compressed weight matrix $\hat{\mathbf{W}}$ [(10)] partitioned in weight memory units
- a partitioned input memory that stores intermediate input feature map (IFMAP) data $\mathbf{X}_l$
- a CSC-based router that implements the modular arithmetic



Parameters: 235,500 + 30,100 + 1,000 = 266,600
MNIST
Top-1 Accuracy = 98.4%
FC 1
FC 2
FC 3
(a)
Parameters: 2,800 + 1,100 + 1,000 = 4,900
MNIST
Top-1 Accuracy = 96.6%
CSCI 1
CSCI 2
FC 3
(b)
Parameters:
2.5 million + 37.8 million + 16.8 million + 4.1 million = 61.1 million
Image Net
Top-5 Accuracy = 79.1%
Convolution Layers
FC 1
FC 2
FC 3
(c)
Parameters:
2.5 million + 0.9 million + 1 million + 1.3 million = 5.7 million
Image Net
Top-5 Accuracy = 77.6%
Frozen Pretrained Convolution Layers
CSCII 1
CSCII 2
CSCII 3
(d)

FIGURE 2: Configurations of the (a) LeNet-300-100 with FC layers, (b) LeNet-300-100 with CSCI layers, (c) AlexNet with FC layers, and (d) AlexNet with CSCII layers. For AlexNet, we use pretrained weights for its convolution layers and have them frozen when training on CSC configurations.

CSC architectures can effectively compress the traditional FC layers of neural networks on par with pruning methods.

of the cyclic dilated matrix–vector multiplication and links the array of PEs to the array of subbanks in the IFMAP memory with a high internal memory bandwidth.

Each PE also incorporates another static random-access memory that stores output feature map (OFMAP) $\mathbf{Y}_l$ in partitions. The compressed format $\hat{\mathbf{W}}_l$ given a CSC layer and its IFMAP $\mathbf{X}$, are partitioned and evenly distributed between the number of PEs weight and number of PEs input fmap memory subbanks, respectively; i.e., given a $\hat{\mathbf{W}}_l \in \mathbf{IR}^{N \times F}$ the NF parameters of $\hat{\mathbf{W}}_l$ and N parameters of the fmap are equally partitioned and stored in the number of PEs weight and number of PEs input fmap subbanks, respectively. Thus, each of the PEs is designated to compute 1/(number of PEs) of the total computation attributed to the layer and should have direct access to one and only one partitioned weight memory at all times as well as be provided with access to each and every number of PEs memory subbank at different computation cycles.

High-Bandwidth Router Adopting Cyclic Architectures

To effectively tile the computation for the layer and link the individual input fmap memory subbanks with the PEs, we design a router that adopts a cyclic architecture that essentially implements the modulo operator existing in (11). The cyclic router is composed of switches that are controlled by a state machine driven with the CSC layer's hyperparameters to provide a one-to-one cyclic link between IFMAP subbanks and MAC units from the PEs.

Figure 4 reflects the idea of tiled computation by illustrating the operation between an eight-by-eight CSC matrix ($N=8$, $F=4$, $D=2$) and a vector of size eight, using distributed IFMAP, OFMAP, and weight memory units that are interlinked using a fast router. The hardware is configured with four two-word input subbanks, four PEs (with 1 MAC unit), four eight-word weights, and four two-word output subbanks. The total computation for this example is equal to 32 MAC operations, which is carried out in eight clock cycles using a router with switches that takes advantage of the cyclic structure and instantly links the computing resources to the required data. Using a total of four MAC units to perform 32 MAC operations in eight clock cycles indicates that the design meets the maximum achievable performance with the available resources.

Configuration and Fabrication

The CSC-AP hardware was configured to have a data width of 8 b with 16 PEs each incorporating 1 MAC unit, 1 kB of weight memory and 512 B of Input and 512 B of Output memory, as illustrated in Figure 3. The configured hardware was fabricated using Taiwan Semiconductor Manufacturing Company CMOS technology in 65 nm on a die area of 7 mm^2. In total, the hardware has 16 kB of availability for weight memory, 8 kB for the IFMAP, and 8 kB for the OFMAP.

For the router, we adopted a CSC structure with the parameters $L=2$ stages, $N=16$ switches per stage, and $F=4$ degrees of switches, which provides an internal bandwidth of 1.6 GB/s at a clock rate of 100 MHz. Figure 5(a) shows the post

TABLE 1. COMPRESSING LENET-300-100 BY MEANS OF REPLACING FC WITH CSCI LAYERS COMPARED TO RELATED PRUNING METHODS.

LAYER	PARAMETERS (BASELINE)	PRUNING [2]	PRUNING [3]	CSCI (COMPRESSION γ)
Index memory required?		Yes	Yes	No
FC1	236,000	8%	1.8%	1.5% (84 times)
FC2	30,000	9%	1.8%	4.4% (27 times)
FC3	1,000	26%	5.5%	100% (one time)
Total	267,000	8% (12 times)	1.8% (56 times)	2.2% (46 times)
Top-1 accuracy	98.4%	98.4%	98%	97.2%

TABLE 2. COMPRESSING ALEXNET BY MEANS OF REPLACING FC WITH CSCII LAYERS COMPARED TO RELATED PRUNING METHODS.

LAYER	PARAMETERS (BASELINE)	PRUNING [2]	PRUNING [4]	CSCII (COMPRESSION γ)
Index memory required?		Yes	Yes	No
All convolution layers	2.5 million	36%	25%	100% (N/A)
FC1	38 million	9%	7%	2% (42 times)
FC2	17 million	9%	7%	6% (17 times)
FC3	4 million	25%	18%	33% (three times)
Total	61 million	11% (nine times)	8% (12 times)	10% (10 times)
Top-5 accuracy	79.1%	80.2%	80.4%	77.6%

N/A: not applicable.

place-and-route physical design of the actual fabricated die that incorporates the CSC-AP, and Figure 5(b) shows the die (packaged) along with a U.S. quarter coin for a scaled comparison.

Evaluation

Figure 5(c) shows the evaluation setup to measure the characteristics of the chip given various voltage and frequency ranges. The setup includes a dual-tracking supply voltage to power up the chip with core (0.62–1 V) and I/O voltages (1.8 V), a VC707 evaluation board that sends test signals through a field-programmable gate array mezzanine card cable with a high pin count interfaced with the chip and monitors the response sent back from the chip. An oscilloscope and a multimeter are also used for the verification and precise measurements of signals. The peak performance and efficiency in a well-engineered accelerator with the number of MAC units at clock rate *freq* are calculated as

$$\text{Performance}_{\text{true}} = 2 \times \#\,\text{MAC} \times freq$$
$$\text{Efficiency}_{\text{true}} = \frac{\text{Performance}_{\text{true}}}{\text{Power}}. \tag{12}$$

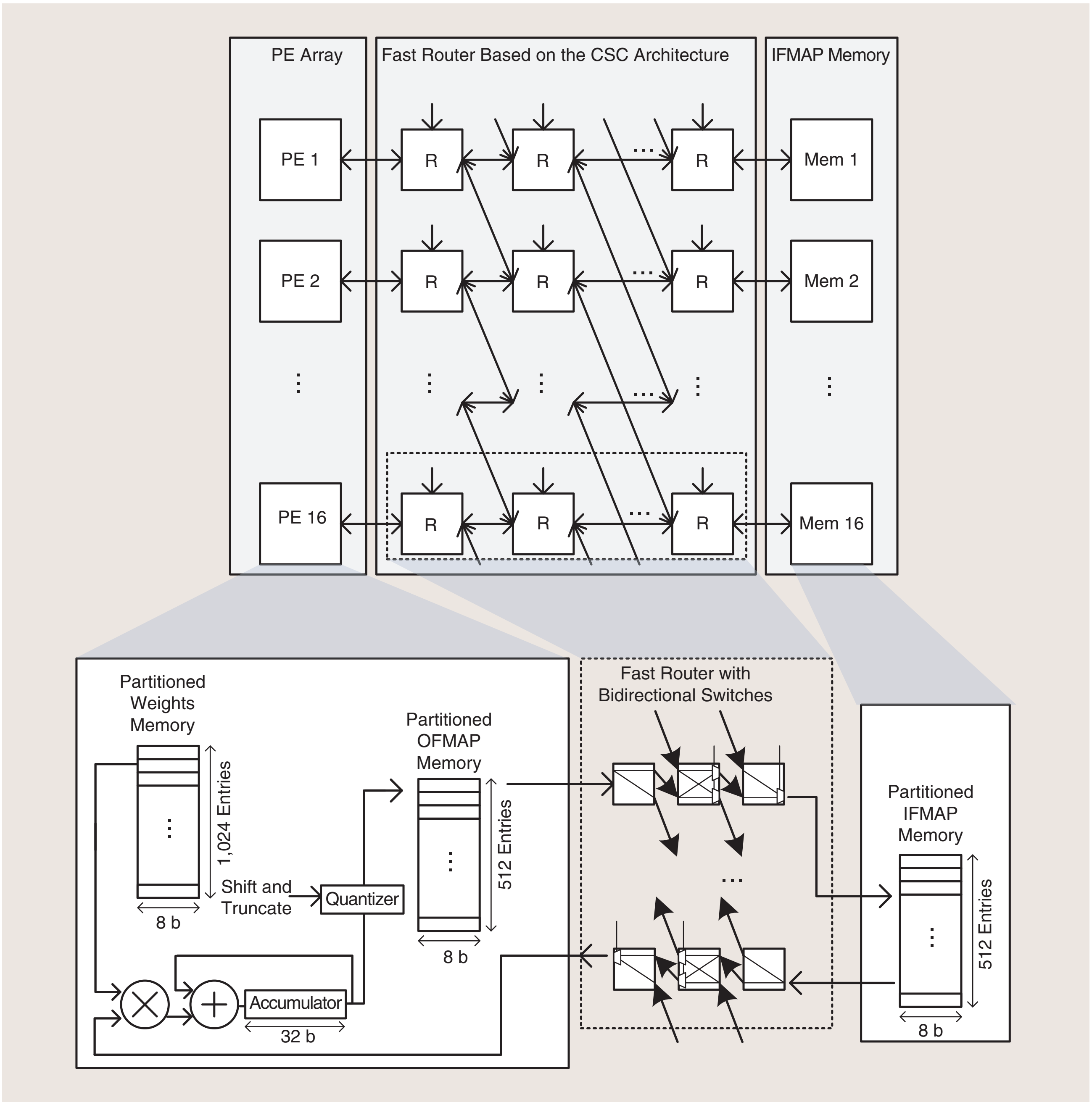


FIGURE 3: An accelerator hardware design to implement cyclic dilated matrix–vector multiplication configurable with the number of PEs and dataflow bit width, highlighting the array of PEs and IFMAP memory units interlinked with a high-bandwidth router. IFMAP: input feature map; Mem: memory; OFMAP: output feature map; PE: processing engine; R: router.

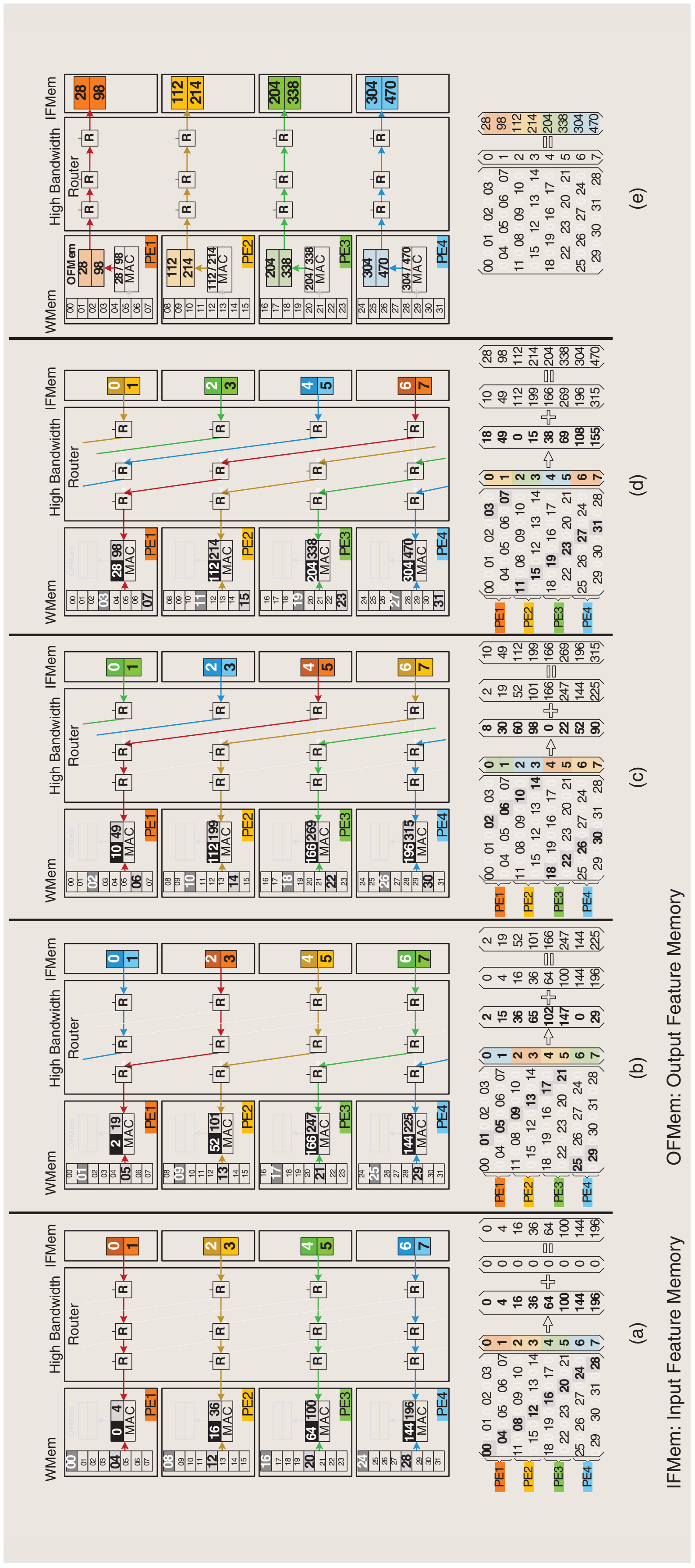


FIGURE 4: The high-bandwidth router that adopts a CSC architecture to implement a CSC DNN layer, illustrated with a simple example: multiplication between a cyclic weight matrix with $F = 4$, $D = 2$, $N = 8$, and a vector with size eight, using four processing engines and performed in eight cycles. (a) Operate the first diag (cycles 1 and 5). (b) Operate the second diag (cycles 2 and 6). (c) Operate the third diag (cycles 3 and 7). (d) Operate the fourth diag (cycles 4 and 8). (e) Write back (cycles 9 and 10). Diag: diagonal; WMem: weight memory.

The $\text{Performance}_{\text{true}}$ indicates the number of multiplication or addition operations an accelerator is able to carry out in a second, and we measure it in terms of GOPS. The $\text{Efficiency}_{\text{true}}$ indicates the number of multiplication or addition operations an accelerator is able to perform per unit of energy, i.e., joules, and we measure it in terms of giga operations per joule (GOPJ) or tera operations per joule (TOPJ). Note that these metrics are attributed to an accelerator that performs only the workload it is given to regardless of the skipped computation, hence denoting the metrics with the subscription *true*.

On the other hand, for an accelerator that implements a model whose computation is reduced by γ times (i.e., $(\gamma - 1)/\gamma$, the computation of which is reduced using techniques such as zero-skipping), an equivalent performance and efficiency can be accounted for the accelerator that would take the zero computations into account using the following relations:

$$\text{Performance}_{\text{eq}} = \gamma \times \text{Performance}_{\text{true}}$$
$$\text{Efficiency}_{\text{eq}} = \gamma \times \text{Efficiency}_{\text{true}}. \quad (13)$$

For example, the matrix–vector multiplication, as illustrated in the Figure 4, requires 64 multiplications, 32 of which are zeros that can be skipped. The matrix can be packed in a compressed form according to (10) with two times fewer parameters that include only nonzero values, and the matrix–vector multiplication can be converted to a dilated matrix–vector multiplication that includes two times fewer operations. When a hardware implements such a matrix–vector multiplication, it can perform only nonzero multiplications in eight cycles. Thus, its true performance can be computed as 32/8 multiplications per cycle. However, when the skipped zero multiplications are also accounted for, the equivalent performance is 64/8 multiplications per cycle, which is two (compression rate) times the true performance.

Figure 6(a) shows the power consumption of the CSC-AP versus clock frequency for core voltages of 0.62 and 1 V. Within a range of the clock frequency of 100 MHz, the core voltage 0.62 V is the minimum operational voltage that delivers the minimum power dissipation and highest efficiency. Thus, we adopt this voltage for sub-100-MHz frequency ranges for low-power, high-efficiency applications. Figure 6(b) shows the performance and efficiency of the CSC-AP that corresponds to (12) given the power consumption at a core voltage of 0.62 V.

With 16 MAC units operating at 100 MHz, the peak performance of the accelerator reaches 3.2 GOPS. The power dissipation and, consequently, true efficiency of the chip at a core voltage of 0.62 V are approximately 74.3 mW and 43 (=3.2/0.074) GOPJ. The true performance and power consumption of the hardware are agnostic toward the compression rate of the CSC layer it implements. In other words, the chip implements a CSC layer with the same actual performance and power consumption regardless of what N, F, or D are in (11). As a result, given a CSC layer with compression $1<\gamma<84$ (as in the compressed LeNet-300-100), the equivalent performance and efficiency of the chip range between 3.2 GOPS and 43 GOPJ for an FC layer where $\gamma=1$ and 268.8 GOPS and 3.6 TOPJ for a highly compressed CSC layer where $\gamma=84$, respectively.

CSC architectures, similar to butterfly networks, can be categorized as one of the high-bandwidth communication networks for parallel system interconnections.

Related Work and Comparison

Deng et al. [6] proposed PermDNN, which is an approach to customizing sparse weight matrices with a diagonalization scheme, for which they designed a 32-PE (256-multiplier) accelerator with 16-b precision dataflow synthesized in 28 nm and operating at 1.2 GHz. At first glance, the CSC layers look similar to the permuted diagonal matrices in the PermDNN. However, in PermDNN, a weight matrix is chunked into many smaller diagonalized matrices, and the connectivity of consecutive layers as a tweakable parameter is not defined, whereas a CSC layer uses one continuous diagonalized (cyclic) weight matrix per layer, the cascade of a few of which guarantees the full connectivity between alternate layers.

CIRCNN [7] is another similar work to the CSC layers in which a weight matrix in a DNN is partitioned into K-by-K submatrices, each of which, due to their circulant structure, can be defined with only K scalars, thus compressing the model by K times. The convolution operation of the block-circulant matrices can be efficiently performed using FFT, elementwise matrix multiplication, and inverse FFT operations. The authors of CIRCNN also proposed an accelerator whose synthesis results in 45 nm and 200 MHz and shows an equivalent efficiency of nearly 10 TOPJ. Despite its significant complexity reduction

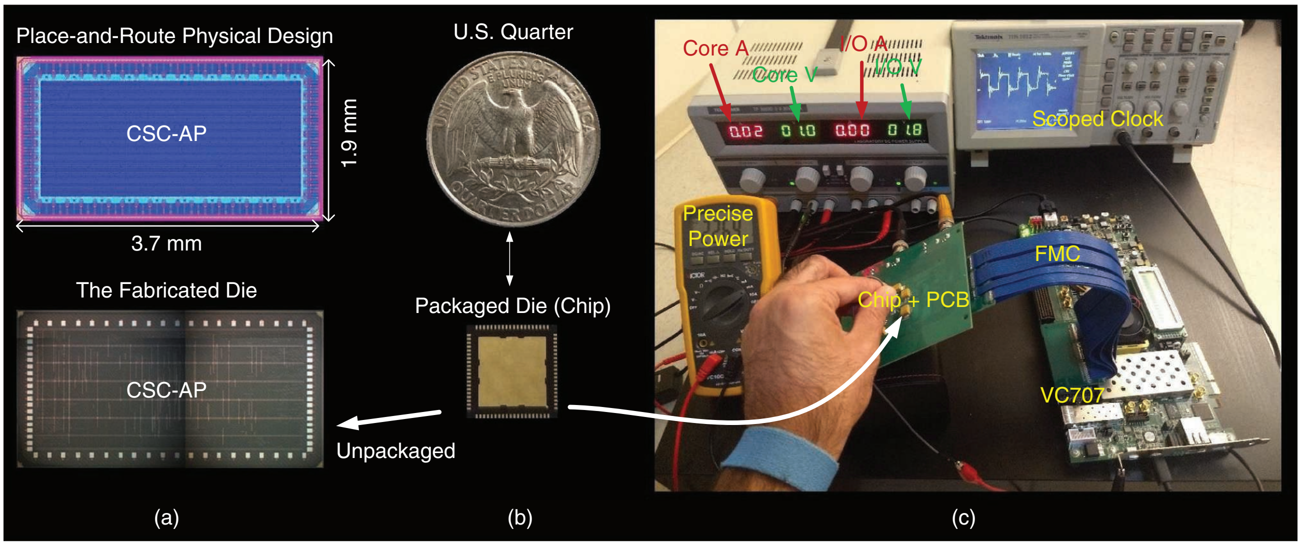


FIGURE 5: (a) The post place-and-route physical design and actual fabricated die of the CSC-AP. (b) The packaged die compared to the size of a U.S. quarter coin. (d) The test, verification, and measurement setup, highlighting a dual-tracking voltage supply to power up the chip mounted on a printed circuit board (PCB). A VC707 platform is used to drive the chip with test benches and signals using a field-programmable gate array mezzanine card (FMC) high-pin-count cable; a multimeter and an oscilloscope are employed for signal monitoring and measurement.

and efficiency, however, CIRCNN has two drawbacks: 1) the computation needs to be carried out in the realm of complex numbers and 2) the method cannot be used in tandem with extreme quantization, as the precision of the quantized weight matrices do not propagate in their transformation into the Fourier domain.

We compare the performance, power, and efficiency of a CSC-AP with a Pascal-family GPU from the NVIDIA Jetson TX2 system on chip, which is a commercial off-the-shelf (COTS) device optimized for deep learning applications, and with three related application-specific ICs (ASICs) that accelerate processing neural networks in the same fabrication technology, i.e., 65 nm. We note that comparing the performance and efficiency of hardware accelerators with different data-width formats

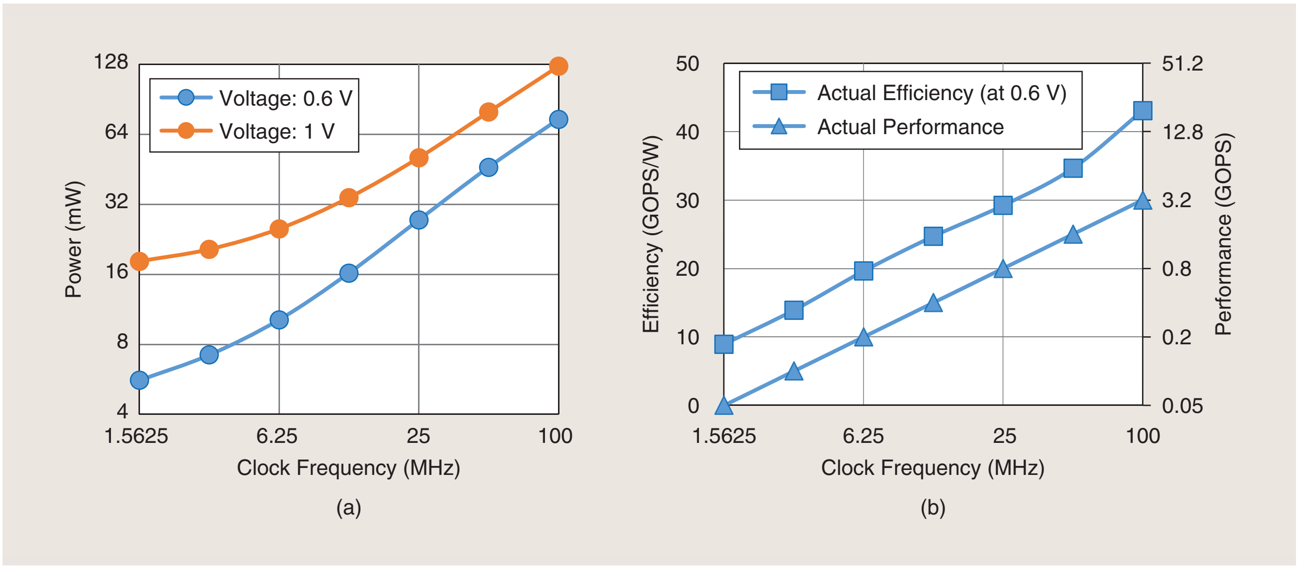


FIGURE 6: The (a) power as well as (b) performance and efficiency measurements of the CSC-AP, given a clock frequency ranging from 1 to 100 MHz.

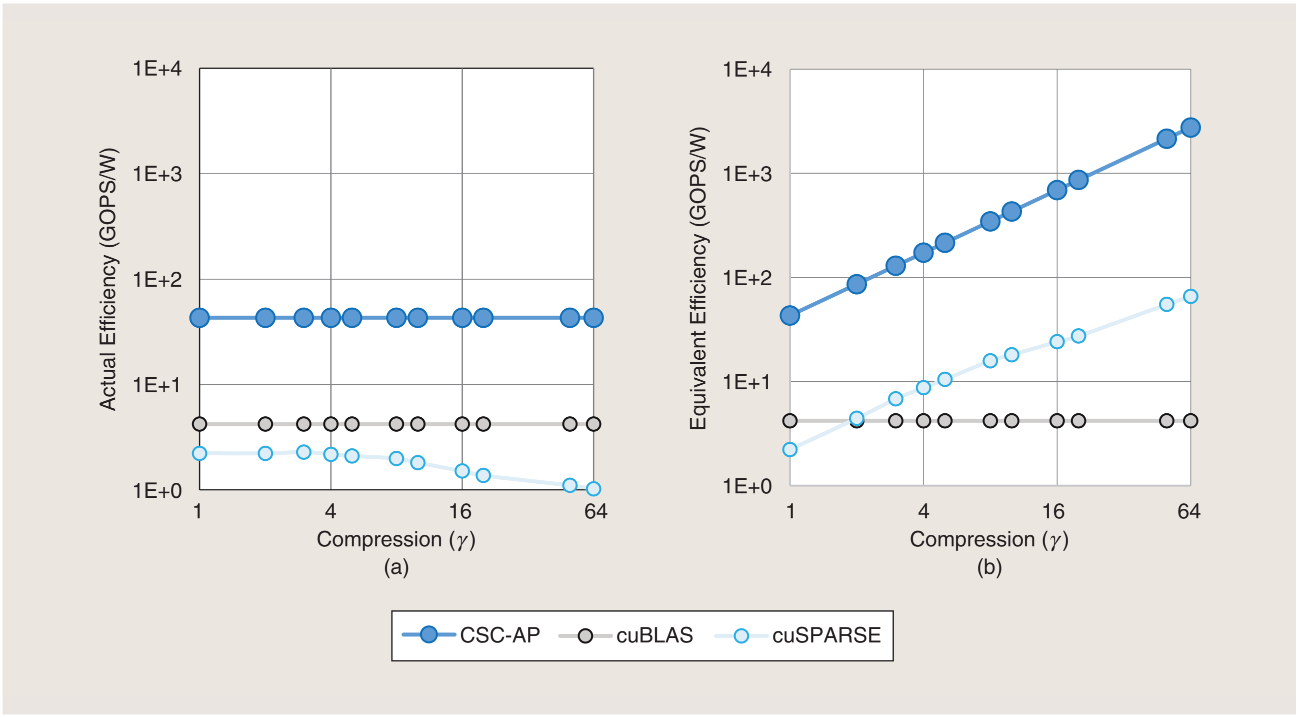


FIGURE 7: A comparison of the (a) actual and (b) equivalent efficiencies of the cuBLAS and cuSPARSE libraries on the NVIDIA TX2 GPU with those of the CSC-AP for a CSC matrix–vector multiplication with N = 1,024, given various compression rates.

(e.g., floating point, 16-b, 8-b, and binarized) may not be completely fair; however, it is the standard practice.

COTS Devices

The NVIDIA Pascal-family GPU is loaded with 8 GB of memory and 256 cores that can operate at a clock frequency of 1.3 GHz and deliver up to 1.33 teraflops for high-performance computing workloads, like matrix–matrix multiplication using developed libraries, such as cuBLAS. Such performance is only very high for large workloads, and it degrades when the workload is small. CuSPARSE is another useful library that performs sparse matrix operations, such as a pruned matrix–vector multiplication, and it outperforms cuBLAS if the matrix is sufficiently sparsified.

We conduct a set of experiments of matrix–vector multiplication over the GPU using both the cuBLAS and cuSPARSE libraries as well as over the CSC-AP for a CSC matrix of size 1,024 by 1,024 with various compression rates. We measure the performance and efficiency of these implementations only after the data are loaded to the on-chip memory of the two devices. Figure 7(a) shows the true efficiency of the CSC-AP at 100 MHz (0.62 V) and compares it with that of the GPU at 1.3 GHz using either of the two NVIDIA Compute Unified Device Architecture libraries over the given workload with various compression rates. Figure 7(b) plots the equivalent efficiency of the experiments by taking the compression rate into account according to (13).

According to this experiment, the performance and efficiency of cuBLAS is constant for variously pruned matrices, and, since it is agnostic to the sparsity of its workload, its equivalent performance and efficiency are the same as its actual ones. On the other hand, the equivalent performance and efficiency of both the CSC-AP and cuSPARSE libraries increase proportionally to the compression rate of the CSC matrix. Consequently, for a matrix–vector multiplication with $N = 1{,}024$, the efficiency of the CSC-AP ranges between 43 and 2,756 GOPJ compared to that of the cuSPARSE, ranging between 2 and 66 floating point operations per joule (FLOPJ) and efficiency of cuBLAS, which is constant 4.2 FLOPJ for compression rates ranging between one and 64 times.

TABLE 3. A COMPARISON WITH RELATED WORK.

	EYERISS [8]	UNPU [9]	STICKER [10]	CSC-AP
Layer type	CONV	CONV + FC	CONV + FC	FC
Sparsity mode	Dense	Dense	Sparse	CSC
Tech (nm)	65	65	65	65
Die area (mm^2)	12.25	16	7.8	7
Data width (b)	16	1–16	8	8
Multiplier count	168	13,824	256	16
On-chi buffer (kB)	108	256	170	32
Supply voltage (V)	0.82–1.17	0.63–1.1	0.67–1	0.62–1
Power (mW)	235–332	3.2–297	20.5–284.4	5.6–74.3
Frequency (MHz)	100–250	5–200	20–200	1–100
Peak efficiency (TOPJ)	0.31	3.08	0.41–62.1	0.04–3.61

CONV: convolutional.

ASICs

Table 3 summarizes three accelerators fabricated in 65-nm technology that process neural network layers and compares them to the CSC-AP. The main purpose of this table is to compare the related work and review the potentials that the 65-nm technology brings about, given different approaches and architectures.

Eyeriss [8] is an accelerator for deep convolutional neural networks optimized for energy efficiency and 16-b models. It encompasses a wide range of neural network layers; however, it delivers the same amount of efficiency given pruned DNN layers, whereas the efficiency of the CSC-AP can approach 3.6 TOPJ for highly compressed CSC layers.

A unified neural processing unit (UNPU) [9] is a unified DNN accelerator that supports variable weight precision ranging from 1 to 16 b for the efficient implementation of DNNs within a mobile environment. While the CSC-AP consumes two times less silicon area, it can consume as little as 5.6 mW at low-frequency settings, which is comparable to the UNPU's low power consumption of 3.2 mW at the same frequency range.

Lastly, STICKER [10] is an 8-b accelerator that explores a neural network's sparsity both in weights and the fmap data and can deliver up to 62.1 TOPJ, a high amount of equivalent efficiency, which is accounted for by an fmap and neural network layer both with 10% sparsity. Compared to STICKER, the CSC-AP has 16 times fewer multipliers and five times less on-chip memory, thus delivering approximately 10–20 times less efficiency yet consuming approximately four times less power.

Conclusions

CSC architectures are structurally sparse graphs that follow a circular arrangement in their design and have a density of $\mathcal{O}(N\log N)$; they can be used to reduce the memory/computation complexity of FC layers of $\mathcal{O}(N^2)$ where N is the number of nodes in a given layer. CSC architectures can effectively compress the traditional FC layers of neural networks on par with pruning methods, and then can be implemented using a hardware-friendly style consuming minimal hardware resources and power as well as providing high energy efficiency.

The CSC-AP is a chip with a die size of 7 mm^2 fabricated in 65 nm that can process neural networks compressed with CSC architectures and can deliver up to 3.6 TOPJ which is on par with related accelerators fabricated in the same technology. Moreover, CSC architectures, similar to butterfly networks, can be categorized as one of the high-bandwidth communication networks for parallel system interconnections. Such interconnection is used in the design of a high-bandwidth router embedded into the CSC-AP.

Acknowledgments

We would like to thank Tejas Kadale and Kevin Gubbi from San Francisco State University for their help and valuable input on this project.

References

[1] M. Hosseini, M. Horton, H. Paneliya, U. Kallakuri, H. Homayoun, and T. Mohsenin, "On the complexity reduction of dense layers from O (*n2*) to O (*n*log*n*) with cyclic sparsely connected layers," in *Proc. 56th ACM/IEEE Des. Autom. Conf. (DAC)*, 2019, pp. 1–6.

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

[3] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1387–1395.

[4] H. Mao et al., "Exploring the regularity of sparse structure in convolutional neural networks," 2017, arXiv:1705.08922.

[5] M. Hosseini, N. K. Manjunath, U. Kallakuri, V. Chandrareddy, and T. Mohsenin, "Cyclic sparsely connected architectures for compact deep convolutional neural networks," in *Proc. IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, 2021, pp. 1–8.

[6] C. Deng, S. Liao, Y. Xie, K. K. Parhi, X. Qian, and B. Yuan, "PermDNN: Efficient compressed DNN architecture with permuted diagonal matrices," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarch. (MICRO)*, 2018, pp. 189–202. doi: 10.1109/MICRO.2018.00024.

[7] C. Ding et al., "CirCNN: Accelerating and compressing deep neural networks using block-circulant weight matrices," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarch.*, 2017, pp. 395–408.

[8] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2016. doi: 10.1109/JSSC.2016.2616357.

[9] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2018, pp. 218–220.

[10] Z. Yuan et al., "Sticker: A 0.41–62.1 TOPS/W 8bit neural network processor with multi-sparsity compatible convolution arrays and online tuning acceleration for fully connected layers," in *Proc. IEEE Symp. VLSI Circuits*, 2018, pp. 33–34. doi: 10.1109/VLSIC.2018.8502404.

[11] A. Shiri et al., "Energy-efficient hardware for language guided reinforcement learning," in *Proc. Great Lakes Symp. VLSI*, 2020, pp. 131–136.

[12] H. Ren et al., "End-to-end scalable and low power multi-modal CNN for respiratory-related symptoms detection," in *Proc. IEEE 33rd Int. System-on-Chip Conf. (SOCC) (SOCC 2020)*, Sept. 2020, pp. 102–107. doi: 10.1109/SOCC49529.2020.9524755.

[13] B. Prakash, N. Waytowich, A. Ganeshan, T. Oates, and T. Mohsenin, "Guiding safe reinforcement learning policies using structured language constraints," in *Proc. SafeAI Workshop 34th AAAI Conf. Artif. Intell.*, 2020. [Online]. Available: http://eehpc.csee.umbc.edu/publications/pdf/2020/AAAI_RL_Workshop.pdf

[14] A. N. Mazumder et al., "Automatic detection of respiratory symptoms using a low power multi-input CNN processor," *IEEE Des. Test.*, early access, May 11, 2021. doi: 10.1109/MDAT.2021.3079318.

About the Authors

Morteza Hosseini (hs10@umbc.edu) received his M.Sc. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2012. He is working toward his Ph.D. degree in computer engineering at the University of Maryland, Baltimore County, Baltimore, Maryland, 21250, USA. His research interests include optimizing deep neural networks for energy-efficient deployment to hardware.

Nitheesh Manjunath (n67@umbc.edu) received his M.Sc. degree in computer engineering from the University of Maryland, Baltimore County, Baltimore, Maryland, 21250, USA, in 2021. He is a field-programmable gate array (FPGA) design engineer at Broadcast Sports International, LLC. His research interests include energy-efficient machine learning as well as edge artificial intelligence accelerator design and implementation on FPGAs and application-specific ICs.

Uttej Kallakuri (ukalla1@umbc.edu) is working toward his Ph.D. degree in computer engineering from the University of Maryland, Baltimore County, Baltimore, Maryland, 21250, USA, where he earned his M.Sc. degree in 2019. His research interests include the design and implementation of energy-efficient high-performance machine learning algorithms on field-programmable gate arrays and application-specific ICs.

Hamid Mahmoodi (mahmoodi@sfsu.edu) received his Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, Indiana, in 2005. He is a professor of electrical and computer engineering in the School of Engineering at San Francisco State University, San Francisco, California, 94132, USA. His research interests include low-power, reliable, and high-performance circuit design in nanoelectronic technologies. He has published more than 100 technical papers in journals and conferences and holds five U.S. patents.

Houman Homayoun (hhomayoun@ucdavis.edu) is an associate professor in the Department of Electrical and Computer Engineering at University of California, Davis, Davis, California, 95616, USA. He is also the director of the National Science Foundation Center for Hardware and Embedded Systems Security and Trust. His research interests include hardware security and trust, applied machine learning and artificial intelligence, and data-intensive and heterogeneous computing. He has published more than 200 technical papers at prestigious conferences and in journals and directed more than US$8 million in research funding from the National Science Foundation, DARPA, Air Force Research Laboratory, the National Institute of Standards and Technology, U.S. Congress, and various industrial sponsors.

Tinoosh Mohsenin (tinoosh@umbc.edu) is an associate professor with the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, Baltimore, Maryland, 21250, USA, where she is also the director of the Energy-Efficient High-Performance Computing Lab. She received her Ph.D. from the University of California, Davis, in 2010 and her M.Sc. degree from Rice University in 2004, both in electrical and computer engineering. She has authored or coauthored more than 130 peer-reviewed journal and conference publications. Her research interests include designing energy-efficient embedded processors for machine learning and signal processing, knowledge extraction techniques for autonomous systems, wearable smart health monitoring, and embedded big data computing. SSC