SONA: An Accelerator for Transform-Domain Neural Networks with Sparse-Orthogonal Weights

Pierre Abillama, Zichen Fan, Yu Chen, Hyochan An, Qirui Zhang,
Seungkyu Choi, David Blaauw, Dennis Sylvester, Hun-Seok Kim
Department of Electrical and Computer Engineering, University of Michigan, MI, USA
Email: {pabillam, zcfan, unchenyu, hyochan, qiruizh, seungkc, blaauw, dmcs, hunseok}@umich.edu

***Abstract*—Recent advances in model pruning have enabled sparsity-aware deep neural network accelerators that improve the energy-efficiency and performance of inference tasks. We introduce SONA, a novel transform-domain neural network accelerator in which convolution operations are replaced by element-wise multiplications with sparse-orthogonal weights. SONA employs an output stationary dataflow coupled with an energy-efficient memory organization to reduce the overhead of sparse-orthogonal transform-domain kernels that are concurrently processed without any conflicts. Weights in SONA are non-uniformly quantized with bit-sparse canonical-signed-digit representations to reduce multiplications to simple additions. Moreover, for sparse fully-connected layers (FCLs), SONA introduces column-based-block structured pruning, which is integrated into the same architecture that maintains full multiply-and-accumulate (MAC) array utilization. Compared to prior dense and sparse neural networks accelerators, SONA can reduce inference energy by $5.1\times$ and $2.4\times$ and increase performance by $5.2\times$ and $2.1\times$, respectively, for convolution layers. For sparse FCLs, SONA can reduce inference energy by $2.4\times$ and increase performance by $2\times$ compared to prior work.**

I. Introduction

Convolutional neural networks (CNNs) have become a fundamental technique in machine learning tasks, but their extension to low-cost and energy-constrained applications is limited by CNN model sizes and complexity. Recently, pruning methods aimed at reducing the number of non-zero parameters have been proposed to decrease model size and lower complexity [1], [2]. In turn, sparsity-aware accelerators that directly leverage sparsity have been proposed [3]–[7] to improve the energy-efficiency of inference tasks.

Another way to reduce complexity is via transform-domain computations that replace convolution in CNNs with element-wise multiplications. However, it is difficult to combine both sparsity and transform-domain computations since transform-domain models often do not allow aggressive weight pruning as shown in [8], [9]. Moreover, the sparsity that these works exploited was unstructured (random). Applying sparsity-aware architectures to these transform-domain models with less aggressive pruning can yield limited gains due to the unstructured nature of sparsity which imposes significant overheads on accelerator flexibility or results in reduced hardware utilization.

To overcome these challenges, a heterogeneous transform-domain neural network (HTNN) [10] was proposed as a framework to learn structured sparse-orthogonal weights where convolutions are replaced by element-wise multiplications. In an HTNN, *two or more kernels in different transform domains share a multiplier without conflict* as the non-zero weight positions are strictly orthogonal to each other. The authors in [10] demonstrate various CNN workloads that can be trained, pruned, and quantized in heterogeneous transform domains while maintaining inference accuracy. However, the expectation that HTNNs [10] can reduce computational complexity compared to equivalent sparse CNNs has not been demonstrated in hardware. Efficiently mapping HTNN models to a hardware architecture in a way that maximizes observable gain remains a significant challenge.

In this paper, we propose SONA, a novel energy-efficient accelerator architecture for HTNNs. We propose an HTNN-specific output stationary dataflow coupled with an energy-efficient transform memory organization to reduce the overhead of overlapped transform-domain convolution. The proposed architecture demonstrates reconfigurable datapaths to compute the permuted variants of the Walsh-Hadamard transform (WHT) for concurrently executed transform-domain kernels. Moreover, we extend the sparse-orthogonal weight concept of HTNN to fully connected layers (FCLs) by proposing a column-based-block (CBB) structured sparsity pattern. Structured sparsity in FCLs allows SONA to share a *unified datapath between sparse convolution and sparse FCLs without compromising MAC array utilization*. Furthermore, HTNN employs non-uniformly quantized bit-sparse canonical-signed-digit (BS-CSD) weights. At the circuit level, SONA proposes a novel BS-CSD-MAC unit (CMU) to replace multiplications with bit-shifts and additions.

We evaluate SONA on CIFAR-10/100 and ImageNet and compare performance and energy-efficiency against representative dense and sparse DNN accelerator designs. Compared to prior dense and sparse CNN accelerators, SONA can reduce inference energy by $5.1\times$ and $2.4\times$, and increase performance by $5.2\times$ and $2.1\times$, respectively, for convolution layers. For sparse FCLs, SONA can reduce inference energy by $2.4\times$ and increase performance by $2.1\times$ compared to prior work. The main contributions of this paper are:

- SONA, the first demonstration of an energy-efficient hardware accelerator architecture for HTNNs with an HTNN-specific output stationary dataflow coupled with an energy-efficient memory organization.
- A CBB structured sparsity pattern for FCLs which enables SONA to share the datapath between sparse

DOI 10.1109/ASAP57973.2023.00015

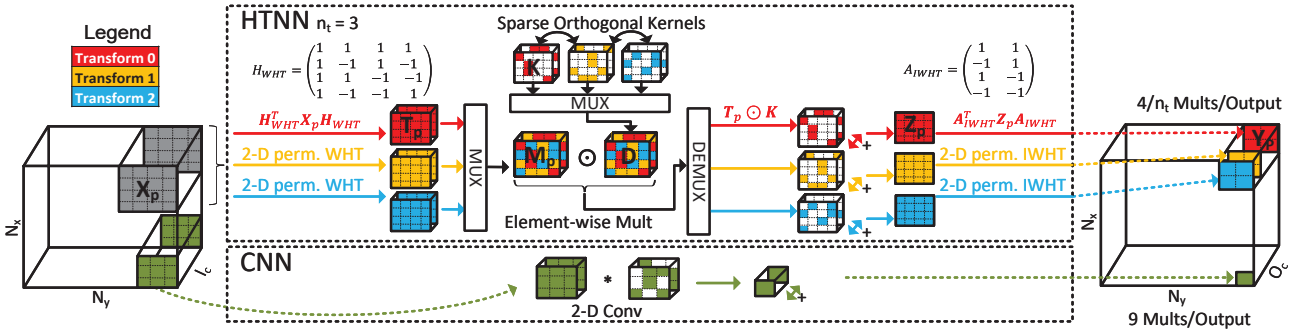

Fig. 1. Comparison between HTNN WHT domain convolution layer and conventional CNN convolutional layer.

transform-domain convolution and sparse FCLs without compromising MAC array utilization.

- A BS-CSD-MAC unit to replace multiplications with bit-shifts/additions and concurrently process output channels.
- Evaluation of SONA against prior dense and sparse DNN accelerators using a simulator as well as RTL implementation on CIFAR-10/100 and ImageNet.

II. Background

A. WHT-Domain Convolution

A CNN consists of a set of cascaded layers. We refer to elements of a given layer's input feature map as activations. The input feature map is of size $N_x \times N_y \times I_C$, where N_x and N_y are the spatial dimensions and I_C is the number of input channels. We refer to the set of learnable parameters in a given layer as weight kernels. The cardinality of this set is O_C, the layer's number of output channels. Each weight kernel is a $K_x \times K_y \times I_C$ tensor where K_x and K_y denote the spatial dimensions. WHT [11] is a generalized class of Fourier transforms given by a symmetric transform matrix $\mathbf{H}$ that contains only binary values ± 1. Although WHT-domain convolution is generalizable to kernels of any size, we focus on the case proposed in [10] where stride-1 3×3 convolutions are replaced by stride-2 4×4 WHT-domain kernels given that it is a very popular configuration in CNNs. Note that the 1×1 point-wise convolution commonly used in MobileNetV1/2 [12] is identical in both HTNNs and CNNs.

As shown in Figure 1 (red), WHT-domain convolution is based on 4×4 activation patches $\mathbf{X_p^c}$ for each input channel c extracted from the input with a stride of 2×2. To calculate the 2×2 output channel $\mathbf{Y_p}$ of a layer for patch position $\mathbf{p}$, a 2D 4-point WHT is first applied to each channel in $\mathbf{X_p}$ to yield 3D output $\mathbf{T_p}$. The 2D 4-point WHT is obtained by applying the WHT $\mathbf{H_{WHT}}$ matrix in (1) to the rows and then columns of each channel $\mathbf{X_p^c}$ to yield $\mathbf{T_p^c} = \mathbf{H_{WHT}}^{\mathrm{T}}\mathbf{X_p^c}\mathbf{H_{WHT}}$. Similarly, $\mathbf{H_{WHT}}$ is applied to the corresponding weight kernel. However, a kernel can be *trained offline in the WHT-domain* [10], so its transform-domain representation $\mathbf{K}$ is directly used during inference without explicit transform computations. Subsequently, each channel in the transform-domain kernel $\mathbf{K}$ and the WHT representation of $\mathbf{X_p}$ are multiplied *element-wise* $\mathbf{T_p} \odot \mathbf{K}$, and the $4\times4\times I_C$ product is reduced to a 4×4 patch $\mathbf{Z_p}$ by accumulating the input channels as shown in Figure 1. After applying the inverse WHT (i.e., $\mathbf{H_{WHT}}^{-1}$) to $\mathbf{Z_p}$, $\mathbf{Y_p}$ is obtained by taking the central 2×2 block of the inverse transform result. Thus the process can be simplified to applying a *binary-valued* 4×2 matrix $\mathbf{A_{IWHT}}$ in (1) (i.e., $\mathbf{A_{IWHT}}^{\mathrm{T}}\mathbf{Z_p}\mathbf{A_{IWHT}}$), which corresponds to the middle two columns of $\mathbf{H_{WHT}}^{-1}$. The operation is summarized in (2).

$$\mathbf{H_{WHT}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}, \ \mathbf{A_{IWHT}} = \begin{pmatrix} 1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & -1 \end{pmatrix} \quad (1)$$

$$\mathbf{Y_p} = \mathbf{A_{IWHT}}^{\mathrm{T}} \left[\sum_{c=1}^{I_C} \left[\mathbf{H_{WHT}}^{\mathrm{T}} \mathbf{X_p^c} \mathbf{H_{WHT}} \right] \odot \mathbf{K^c} \right] \mathbf{A_{IWHT}} \quad (2)$$

B. Sparse-Orthogonal Kernels in HTNNs

HTNN [10] imposes a sparse-orthogonality property to transform-domain convolutions and deliberately searches multiple heterogeneous transforms such that the position of important weights in one transform domain is less likely to overlap with the ones in other transform domains. In HTNN, *kernels belonging to different transform domains are pruned such that their non-zero weights do not overlap (i.e., sparse-orthogonal)*. The number of heterogeneous transform variants in an HTNN is typically $n_t = 2$ or 3, and each variant is defined by multiplying a unique permutation matrix $\mathbf{P}$ to the left of $\mathbf{H_{WHT}}$. Kernels that are sparse-orthogonal to each other can be gathered to form a *dense overlapped kernel* $\mathbf{D}$ *to process* n_t *output channels concurrently* as shown in Figure 1 for $n_t = 3$. To concurrently compute the n_t output channels for a given patch position $\mathbf{p}$, activation patches $\mathbf{X_p}$ are transformed in n_t different variants and overlapped using multiplexers to match the overlapping pattern of $\mathbf{D}$. The overlapped result $\mathbf{M_p}$ is multiplied *element-wise* with $\mathbf{D}$. The intermediate product's overlapping is then undone with de-multiplexers to accumulate the input channels for each of the n_t output channels. Finally, the n_t accumulated patches are inverse transformed.

C. Bit-Sparse Canonical-Signed-Digit (CSD) Computation

HTNN [10] extends structured sparsity to the number representation system by training weights in a bit-sparse CSD format. CSD [13] is a numbering system that uses ternary digits $\{-1, 1, 0\}$ to represent an N-bit number with reduced non-zero digits. For example, 23 requires 4 non-zero bits (010111_2) but only requires 3 non-zero digits in CSD $(1, 0, -1, 0, 0, -1)$ since $23 = 2^5 - 2^3 - 2^0$ holds. HTNNs impose an additional bit-sparsity constraint to limit the number of non-zero digits in the CSD representation of weights to *at most* 2. This technique allows *replacing multiplication with a single addition/subtraction* without impacting the inference accuracy [10]. In SONA, we implement 8-bit input MAC units for BS-CSD weights with at most 2 non-zero digits, whereas activations are quantized in 8-bit two's complement.

D. Fully-Connected Layer (FCL) with Structured Sparsity

FCLs constitute a large portion of the weights that can be pruned to sparse matrices. Accelerators optimized for sparse FCLs such as [4] have shown that the location of non-zero weights significantly impacts compute and memory resource utilization. Despite employing compressed formats for data storage, unstructured sparsity in FCLs leads to extraneous zero padding and load imbalance where processing elements (PEs) are assigned a varying number of non-zero weights to multiply the same activation value. Although the original HTNN [10] addresses sparse convolution overhead, it does not provide a framework for learning structured sparsity for FCLs. To overcome this, SONA introduces a new column-based-block (CBB) structured sparsity pattern extending HTNNs to sparse FCLs under a unified architecture. We train HTNNs with CBB sparse weights and employ an index-based weight-encoding representation to minimize the overhead of zero padding while maximizing the potential memory and MAC utilization and sharing the same hardware with transform-domain convolutions.

III. SONA Architecture

The proposed SONA architecture is motivated by the drawbacks of existing unstructured sparse DNN accelerator architectures such as [5] and [4] as well as by the promise of HTNNs. Although the authors in [10] claim that HTNNs can achieve $4.9 - 6.8\times$ complexity reduction compared to sparse CNN models without compromising the inference accuracy, this result has not been demonstrated in hardware. While the software implementation of HTNNs performs fewer operations, the expected gain is much smaller as it incurs overhead in handling WHT and sparse-orthogonality. The theoretical analysis for HTNN's gain in [10] does not address hardware challenges for devising an accelerator architecture that can efficiently maximize the observable gain.

A. Overall Architecture

Figure 2 shows the overall SONA architecture that unifies WHT-domain sparse-orthogonal convolution and CBB-sparse FCLs. All of the weights and activations are initially stored in off-chip DRAM before being loaded into weight and activation memories, respectively. Weights are converted into their BS-CSD representation (Sec. III-F) before being loaded into weight memory. SONA makes use of a $4 \times 4 \times n_t$ BS-CSD MAC unit (CMU) array and a $N_P \times n_t$ transform datapath array to process N_P patches in parallel using n_t WHT variants.

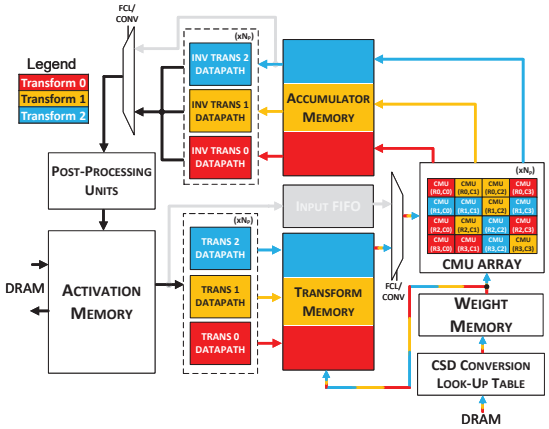

Fig. 2. Overall architecture with highlighted WHT convolution layer path.

The execution of a WHT-domain convolution layer is illustrated in Figure 2. Paths that are not active during transform convolution are greyed out. Different colors in Figure 2 indicate $n_t = 3$ WHT variants. The nested-loop control to sweep the patch position, output channels, and input channels follows an output stationary (OS) ordering (Sec. III-C). Once weights and activations are loaded into their respective memories, an input channel tile of size I for N_P 4×4 patches are sent to the transform array. The resulting $N_P \times n_t$ transformed $4 \times 4 \times I$ patches are stored in transform memory consisting of $4 \times 4 \times n_t$ banks of depth I and data width $N_P \times 8$ bits. Transform memory organization details are discussed in Sec. III-D. Subsequently, SONA performs element-wise BS-CSD MAC operations in the CMU array. All N_P activation $4 \times 4 \times I$ patches multiply the same $4 \times 4 \times I$ kernel that merges n_t sparse-orthogonal weights, which are color-coded in Figure 2. Accumulation across the I channels is done locally at the CMU level. Upon completing the accumulation for a tile, the following group of orthogonal output channels is processed to maximize the reuse of the transformed activations. If the layer's number of input channels is larger than I, the intermediate result from the previous set of output channels is stored in accumulator memory. The subsequently processed input channel tiles are added to the stored result in accumulator memory until all tiles are processed. Then, the N_P accumulated $4 \times 4 \times O_C$ patches are sent to the inverse transform array and post-processing units (e.g., ReLU) while the next set of N_P patches are transformed. The final $2 \times 2 \times O_C$ patches are written to the activation memory.

B. Reconfigurable Transform Datapaths

HTNNs require multiple (n_t) WHT variants that can vary depending on the network model. The transform datapaths need to handle different permuted variants of the 2D WHT,

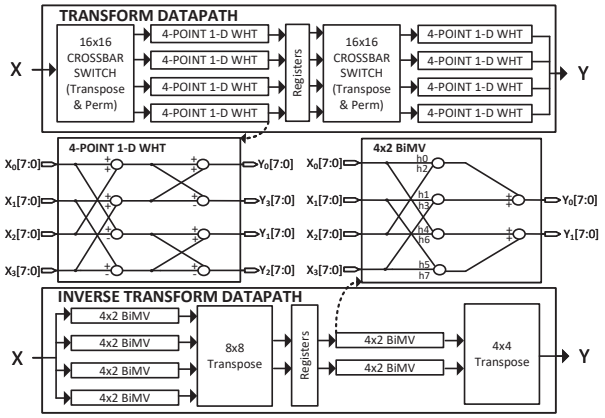

Fig. 3. Permuted reconfigurable transform datapaths

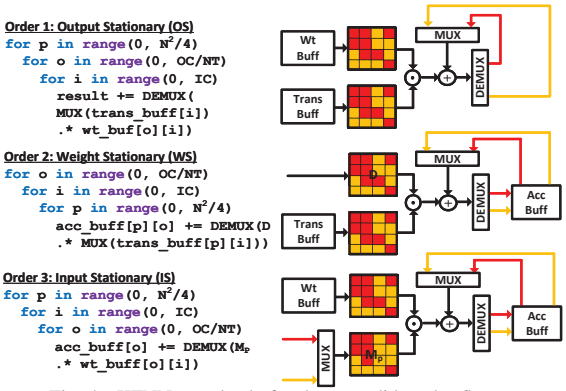

Fig. 4. HTNN pseudcode for three candidate dataflows.

which are defined as $\mathbf{H_P} = \mathbf{PH_{WHT}}$ where $\mathbf{P}$ is the corresponding permutation matrix. A 4×4 2D non-permuted fast WHT requires $8 \times 8 = 64$ adders/subtractors. The transform operation can be reordered and split into two identical operations as in $\mathbf{Y} = \mathbf{H_P}^T\mathbf{XH_P} = (((\mathbf{X}^T\mathbf{P})\mathbf{H_{WHT}})^T\mathbf{P})\mathbf{H_{WHT}}$. First, the 4×4 input patch $\mathbf{X}$ is transposed and permuted. A transform is then applied to each row of the intermediate result. The operation is repeated a second time to produce the final transformed patch $\mathbf{Y}$ as shown in Figure 3 (top). The transpose and permutation operations are combined into a 16×16 reconfigurable crossbar switch. To implement the permuted variants of the inverse WHT as defined in $\mathbf{Y} = \mathbf{A_P}^T\mathbf{XA_P}$, note that $\mathbf{H_{WHT}}$ is an orthogonal matrix and that we can reuse the WHT transform datapath in Figure 3 (top) to compute the inverse WHT. However, this would be wasteful as we only need the central 2×2 block. The permuted inverse WHT variants ($\mathbf{A_P}$) correspond to the binary-valued (± 1) middle two columns of the $\mathbf{H_P}^{-1}$ matrices. Therefore, the inverse transform is implemented using reconfigurable 4×2 binary matrix-vector multiplication (BiMV) blocks and transpose interconnect networks as shown in Figure 3 (bottom). Each datapath requires $6 \times 6 = 36$ adders/subtractors but no multiplications. SONA employs $n_t \times N_P$ datapaths for each of the (inverse) transform to process N_P individual 4×4 patches using up to n_t variants in parallel. These datapaths are pipelined to not limit the performance of SONA.

C. WHT-Domain Sparse-Orthogonal Convolution Dataflow

The dataflow dictates memory access patterns and plays a significant role in maximizing the energy-efficiency. Although CNN accelerator dataflows have been extensively studied [14], [15], they are not directly transferable to HTNN as its internal dataflow is dissimilar to that of ordinary convolution. Therefore, we studied the choice of heterogeneous WHT-domain convolution dataflow prior to conceiving SONA.

For n_t transform-domain variants, an HTNN layer loops over three parameters: patch position $\mathbf{p}$, orthogonal output channels $\frac{O_C}{n_t}$, and input channels I_C. For a candidate architecture, the memory sizes and access frequencies depend on the input and output channels processing order. Figure 4 outlines three dataflows corresponding to different loop reorderings. The first is *output stationary* (OS), where the input is processed by applying n_t WHTs to each input channel patch at a given position and accumulations are done locally at the PE level. Transformed patches are reused by multiple kernels and are stored in a buffer. Kernels are also reused across different patch positions and are stored in a buffer. The second is *weight stationary* (WS), where the weight is fixed at the PE level. The third is *input stationary* (IS) where the input is processed by applying n_t WHTs to a patch position at an input channel, so there is no buffer for transformed activations. IS and WS need a buffer for output product patches as accumulation cannot be done at the PE level, but the buffer in IS is smaller than that in WS.

To identify an energy-efficient dataflow, we perform a case study on WHT-domain convolution layers of ResNet-20 and ConvPool-CNN-C (C-CNN-C). We quantify the buffer (SRAM) sizes and access frequencies in terms of generic layer parameters. We then outline a memory architecture where each layer has on-chip SRAM macros that are sized to fit its layer parameters without needing to tile within the layer, therefore off-chip memory accesses are identical for all dataflows and are excluded. We use TSMC 28/22nm memory compilers to obtain SRAM access energies and exclude PE-internal register access as its contribution is negligible relative to SRAM. Figure 5 indicates that OS is the most energy-efficient dataflow for WHT-domain convolution. WS and IS suffer from a large amount of read-modify-writes, whereas OS accumulations can be done at the PE level. In OS, the bandwidth of the transform buffer is less than that of the accumulator buffer because only $1/n_t$ of the transformed patches are read in a cycle to concurrently process n_t output channels. Although the transform buffer requires more write bandwidth than the accumulator buffer, the former is written to less frequently as transformed patches are re-used for multiple kernels. To take advantage of these observations, the HTNN-specific OS dataflow requires an efficient memory organization (Sec. III-D) for transform buffer access.

D. Memory Organization

We overlap $n_t \leq 3$ orthogonal weight kernels prior to storing them in weight memory and associate a 2-bit mask

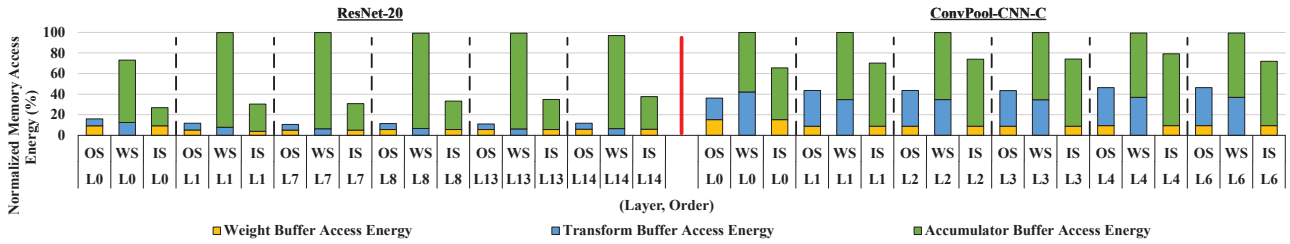

Fig. 5. Memory access energy comparison for C-CNN-C and ResNet-20 against HTNN dataflows: output/weight/input stationary (OS/WS/IS)

with each weight to indicate its corresponding WHT variant. The input activation patches are transformed in all n_t variant domains and re-used across the output channel dimension, but only $1/n_t$ are used during element-wise multiplications. It must be noted that the *overlapping pattern is different from channel to channel* within a single layer, which makes reusing transformed patch non-trivial. Therefore, it is critical to devise an energy-efficient memory organization that *limits the access to only the required transformed patches in each cycle*. To process N_P patches in parallel, the transform memory should hold $I \times N_P$ 8-bit transformed activation patches where I is the tile size for I_C. The read and write bandwidths of this memory are $4 \times 4 \times N_P$ and $4 \times 4 \times N_P \times n_t$, respectively. One approach, single patch single row (SPSR), consists of having $N_P \times n_t$ banks of depth I and word width $4 \times 4 \times 8$ bits. Another approach, single activation single row (SASR), consists of having $N_P \times n_t \times 4 \times 4$ banks of depth I and word width 8 bits. SASR provides more granularity than SPSR in controlling which activations are read at once. With SPSR, $n_t \times N_P$ patches are read when only N_P are needed while SASR helps limit the number of unnecessary memory accesses. However, SASR incurs a larger peripheral memory circuity overhead from employing many more smaller banks and has the potential to be less area and energy efficient.

As a compromise, we propose the multiple activation single row (MASR) scheme illustrated in Figure 6. In MASR, we have $n_t \times 4 \times 4$ banks of depth I and word width $N_P \times 8$ bits. The overlap pattern in MASR only depends on the kernel processed and all N_P patch positions are multiplied by the same kernel in a cycle. During transform memory read, the weight mask can be used to disable $\frac{n_t-1}{n_t}$ of the banks and load only the N_P overlapped transformed patches that are needed. Figure 6 illustrates MASR for $I = 32$, $N_P = 4$, $n_t = 3$. The top left kernel weight is associated with variant 0, and all the activations of each transformed patch using variant 0 are in bank 0. For element-wise multiplication, we enable that bank and disable the other two corresponding to variants 1 and 2. Our experimental results using Arm register file compilers in TSMC 28/22nm technology indicate that for $I = 32$, MASR has $\{1.2\times, 1.7\times\}$ and $\{1.6\times, 2.4\times\}$ less access energy than SPSR and SASR, respectively, for $N_P = \{2, 4\}$ at the cost of being $\{1.8\times, 1.3\times\}$ less area efficient than SPSR. Note that SASR is overall the most flexible but it is the least area-efficient. It is also not scalable as the increased number of small memory banks incurs peripheral memory circuity energy overhead. Thus to exploit patch parallelism, MASR becomes necessary to maximize the energy-efficiency of the design.

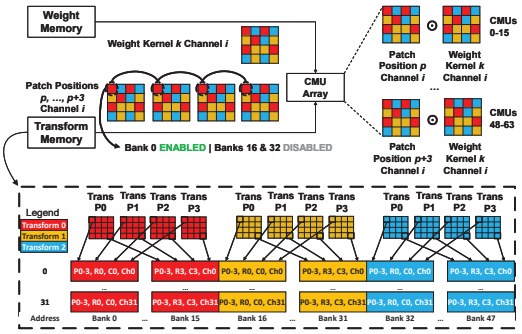

Fig. 6. MASR transform memory organization ($I = 32$, $N_P = 4$, $n_t = 3$).

E. Column-Based-Block Sparse Fully-Connected Layer

To fully exploit both weight and activation sparsity, we explore an outer-product sparse FCL implementation based on an IS dataflow. Consider employing index-based compression to store the weight matrix $\mathbf{W}$ where the location of non-zeros are random (unstructured) as in Figure 7. We first reshape the original column into a C/B-row matrix, where C is the number of weights in a compressed word and B is the weight block size. Then to compress $\mathbf{W}$ along its columns, the weights are packed densely in the direction shown in Figure 7. Each compressed word contains C/B blocks of weights. Each weight is associated with its originating column number in the C/B-row matrix. Each weight's position in the compressed word corresponds to its position in the chunk. Although $\mathbf{W}$ can be aggressively pruned, unstructured weight sparsity could lead to collisions after reshaping the column and non-dense (underutilized) weight blocks. For example, the weight block collisions in Figure 7 leads to having an additional weight word just for one non-zero block. In general, these overheads translate to both inefficient memory and MAC utilization.

To combat this inefficiency, we propose a novel CBB structured pruning method for sparse FCLs to minimize zero padding overhead while sharing the same hardware with WHT-domain convolutions. During FCL training, we impose the following sparsity constraint. Given a target density d, we prune $\mathbf{W}$ such that the number of block collisions in each row of the reshaped column is the same. As a result, we minimize zero padding and maximize potential memory and MAC utilization. To verify whether CBB pruning can achieve high sparsity while maintaining inference accuracy, we test this approach's feasibility on the FCLs of VGG-Nagadomi

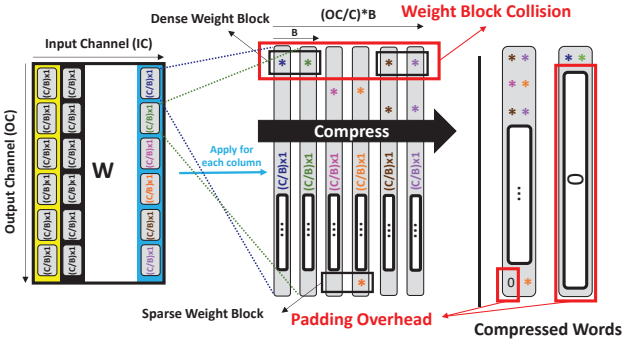

Fig. 7. Index-based encoding overhead for unstructured sparse FCL weights.

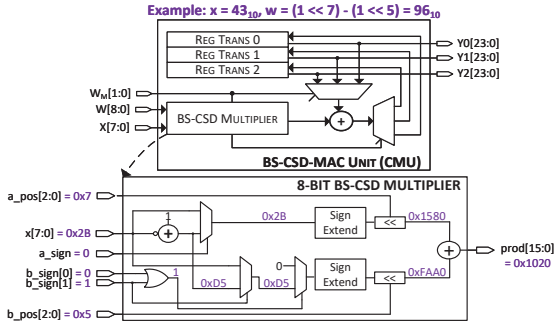

Fig. 8. BS-CSD-MAC unit and 8-bit BS-CSD multiplier.

(VGG-N) HTNN [10]. We train, prune, and quantize FCL weights using 8-bit BS-CSD in PyTorch with $C = 64$, $B = 4$, and $d = 6.25\%$ for the CIFAR-10 dataset. Experiments show that the top-1 accuracy post-training, pruning, and BS-CSD quantization are 93.01%, 92.84%, and 92.80%, respectively. CBB pruning can operate at different layer-dependent densities (in the range of 6.25% – 50%) that do not degrade the accuracy (in the range of 92.80% – 92.98%) by controlling the number of block collisions in each row of the reshaped columns of $\mathbf{W}$.

FCL execution in SONA disables all (inverse) transform datapaths and memories and employs an IS dataflow where only non-zero activations in the input FIFO of Figure 2 are detected and broadcast to the CMU array. We also implement outer-product multiplication for full CMU array utilization using only $1/n_t$ of the accumulator memory banks. C and B are selected based on N_P and the 4×4 patch-based dataflow such that $C = 4 \times 4 \times N_P$ and $B = N_P$ in order to *share the datapath between WHT-domain convolution and sparse FCL*. Each one of the utilized accumulator memory bank maps to one of the C/B rows in the reshaped columns of $\mathbf{W}$ and each block maps to an address in a bank. The maximum O_C supported for an FCL is bounded by $4 \times 4 \times N_P \times O_C/n_t$ where the depth of an accumulator memory bank is O_C/n_t.

F. Bit-Sparse Canonical-Sign-Digit (BS-CSD) MAC Unit

To take advantage of BS-CSD and non-uniform quantization when performing MAC operations, a hardware-friendly representation is required to represent non-zero weights. Given at most 2 non-zero CSD digits, which we will refer to as bits a and b, we encode the respective signs, a_{sign} and b_{sign}, and positions, a_{pos} and b_{pos}, of these bits to determine the operands of the final addition/subtraction. Without loss of generality, we assume that $a_{pos} > b_{pos}$ ($a_{pos} \in \{0, 1, \ldots, 7\}$ and $b_{pos} \in \{0, 1, \ldots, 5\}$) and encode the traditionally 8-bit weights w using 9 bits in the form of $w = (a_{sign} \ll a_{pos}) + (b_{sign} \ll b_{pos})$ where $\ll$ denotes the left bit-shift.

Note that the CSD representation of a number does not contain two adjacent non-zero digits. Thus, the relationship between a_{pos} and b_{pos} becomes $a_{pos} > b_{pos}+1$. Also note that having 87 quantization levels (byproduct of having $\leq$2 non-zero digits) enables memory footprint reduction in off-chip memory by storing each weight as a 7-bit code, which can be converted to a 9-bit representation using a look-up-table before storing it in weight memory. Figure 8 shows the circuit implementation for BS-CSD weight multiplication and illustrates an example where $x = 43$ and $w = (1, 0, -1, 0, 0, 0, 0, 0) = 96$. Using the proposed representation, 8-bit multiplication is replaced by two bit-shifts and one 16-bit addition: $x \cdot w = ((a_{sign} \cdot x) \ll a_{pos}) + ((b_{sign} \cdot x) \ll b_{pos})$. SONA employs $4 \times 4 \times N_P$ BS-CSD-MAC units (CMUs) shown in Figure 8 to perform dense (after merging n_t sparse-orthogonal weights) element-wise multiplications. In each cycle, a CMU performs at most one non-zero and $n_t - 1$ implicit zero MAC operations. The inputs are the 8-bit two's complement activation x, the 9-bit BS-CSD weight w and a 2-bit weight mask w_M indicating the transform variant associated with w. As discussed in Sec. III-C, accumulation is done locally in registers that map to the concurrently computed n_t output channels.

IV. Evaluation

We evaluate SONA using $N_P = 4$ and a tile size of $I = 32$. Table I summarizes the SONA accelerator configuration. To estimate energy efficiency, performance, and area, we fully implement the RTL design of SONA in Verilog and synthesize it using Synopsys Design Compiler in TSMC 22nm. The design is synthesized at 500MHz. PrimeTime PX (PTPX) is used to estimate the energy consumption of the synthesized design using the gate-level netlist and fast signal database (FSDB) file containing switching activity. To compare against prior dense and sparse accelerator designs whose RTL implementations are not publicly available, we utilize a simulator verified against our RTL implementation. For a {dense, sparse} CNN benchmark, we employ {Eyeriss [14], SCNN [5]} and {TimeLoop [16], DNNSim [17]} to simulate their performance and memory access traces, respectively. For a sparse FCL benchmark, we employ EIE [4] and built our own cycle-accurate simulator that was verified against the results reported in [4]. Arm memory compilers in TSMC 22nm are used to estimate the energy consumption of on-chip buffers/memories. We approximate the energy of DRAM accesses to 100pJ per 8 bits as in [18]. All designs are adjusted to have the same number of multipliers. We obtain the energy costs of multipliers and adders under the same

TABLE I
SONA CONFIGURATION AND AREA SUMMARY.

Component	Size	Area (mm^2)	Power (mW)
Activation Memory	420KB	1.262	2.3
Transform Memory	6KB	0.083	9.7
Accumulation Memory	18KB	0.192	4.9
Weight Memory	352KB	1.073	6.5
Input FIFO	1 KB	0.003	0.3
(Inverse) Transform	(12) 12 datapaths	(0.022) 0.039	(1.6) 2.3
CMU Array	64 CMUs	0.023	8.6
Other	N/A	0.152	10.4
Total	N/A	2.85	46.2

22nm technology node to estimate the computation energy expenditure of SONA and baseline designs. All evaluations include (inverse) transform overheads that are insignificant ($\approx 8.03\%$ and 4.17% of SONA's overall latency and energy). Finally, we compare against [19], an optimized dense CNN accelerator whose RTL implementation was made available to us, using PTPX and post-synthesis FSDB dumps to get an accurate relative energy-efficiency of SONA.

We make use of the neural network workloads provided in [2], [9], [10], ResNet-20 ($n_t = 3$) and VGG-N ($n_t = 2$) using CIFAR-10, and C-CNN-C ($n_t = 3$) using CIFAR-100, as well as their respective equivalent CNN workloads. Furthermore, we evaluate an additional benchmark using ImageNet, ResNet-18 and its equivalent HTNN ($n_t = 3$). As the original HTNN [10] targets smaller networks for edge applications where training time is manageable, the training time is exacerbated for larger networks and datasets, rendering tasks such as ImageNet ResNet-18 unfeasible. To overcome the limitations of the training code released with [10], we implement custom CUDA extensions. We also supplement it with CBB-structured sparsity learning for VGG-N FCLs (Sec. III-E) unavailable in [10]. We trained these models in PyTorch using our functionally equivalent custom CUDA layers. Overall, our optimizations improve the training speed by $2-13\times$ on an Nvidia A40 GPU compared to the code in [10]. For ResNet-18 HTNN, training is done using 4 NVIDIA A40 GPUs at a rate of ≈ 40 minutes/epoch. For reference, ResNet-18 CNN can be trained using an NVIDIA A40 GPU at the same rate. With our optimized training framework, the top-1 accuracies of {ResNet-20, VGG-N, C-CNN-C, ResNet-18} HTNNs and baseline sparse CNNs are $\{91.56\%, 92.80\%, 70.51\%, 69.8\%\}$ and $\{91.44\%, 93.08\%, 70.55\%, 69.2\%\}$, respectively. For HTNNs, weights are 8-bit BS-CSD quantized. For sparse CNNs, we apply the deep compression strategy described in [2] and weights are 8-bit uniformly quantized.

A. Simulator-Based Results

Speedup Comparison: Figure 9 shows the speedup of SONA over Eyeriss and SCNN for the convolutional layers of ResNet-20, VGG-N, C-CNN-C, ResNet-18. While SCNN leverages weight and activation sparsity, SONA only leverages weight sparsity for WHT-domain convolution. Nevertheless, SONA can achieve $\{4.4\times, 2.1\times\}$, $\{4.2\times, 1.5\times\}$, $\{6.1\times, 2.7\times\}$, $\{6.1\times, 2.1\times\}$ overall speedup compared to {Eyeriss, SCNN} across all layers of ResNet-20 (CIFAR-10), VGG-N, C-CNN-C, ResNet-18 (ImageNet), respectively. Although

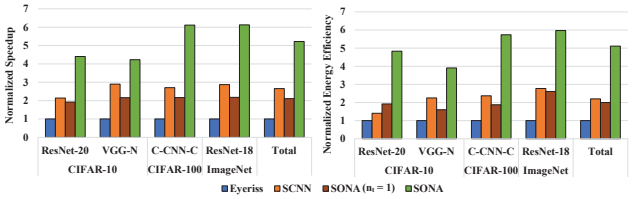

Fig. 9. Conv. layer speedup and energy-efficiency of Eyeriss, SCNN, SONA

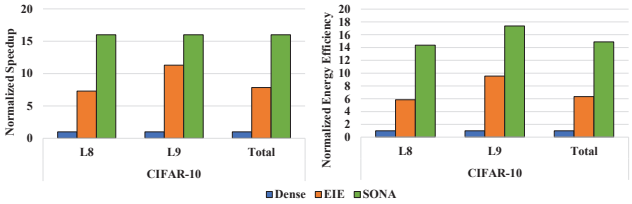

Fig. 10. VGG-N FCL speedup and energy-efficiency of dense, EIE, SONA

the relative speedup of SONA over SCNN diminishes when the activation density is unusually low ($\approx 20\%$) in later layers, SONA maintains higher speedup because it does not incur frequent execution stalls from intra-PE fragmentation and from the way SCNN's PT-IS-CP-sparse dataflow partitions the work across the PEs. SONA can concurrently compute $n_t > 1$ sparse-orthogonal channels while benefiting from the multiplicative gains of replacing convolution with element-wise multiplications in heterogeneous transform domains. For comparison, Figure 9 shows the inference speed of SONA when only one transform domain is utilized ($n_t = 1$) for convolution layers. Figure 10 shows that SONA can achieve $\approx 16\times$ and $2\times$ overall speedup on FCLs of VGG-N over EIE and a naive dense architecture that leverages neither weight nor activation sparsity, respectively. EIE suffers from load imbalance because it uses compressed sparse column weight representation despite using an activation queue. On the contrary, the proposed CBB-structured sparsity allows SONA to maintain full MAC array utilization (Sec. III-E).

Energy-Efficiency Comparison: Figure 9 also shows that SONA can achieve $\{4.8\times, 3.4\times\}$, $\{3.9\times, 1.7\times\}$, $\{5.7\times, 2.4\times\}$, $\{5.9\times, 2.2\times\}$ overall energy-efficiency gain compared to {Eyeriss, SCNN} across layers of ResNet-20 (CIFAR-10), VGG-N, and C-CNN-C, ResNet-18 (ImageNet), respectively. SCNN's outer product leads to an overhead of accumulation buffers with expensive read-modify-write accesses that hurt SCNN's energy efficiency and detract from the multiplicative gains expected from leveraging sparsity. SONA stands out because it is able to process sparse-orthogonal transform-domain convolution ($n_t > 1$) using the HTNN-specific OS dataflow. For ResNet-20, the transform memory accounts for 15% of the total energy (excluding DRAM), which, in the absence of MASR (Sec. III-D), would increase by at least $1.7\times$. The BS-CSD MAC array accounts for 24% of the total energy (excluding DRAM) which, in the absence of BS-CSD, would increase by $1.6\times$. Figure 10 shows that SONA improves the energy-efficiency of the naive architecture and EIE by $14.9\times$ and $2.4\times$, respectively, across FCLs of VGG-N. Although both approaches show improvements over the

TABLE II
POST-SYNTHESIS DESIGN COMPARISON SUMMARY.

Design synthesized (22nm, 0.8V)	**SONA**	**[19]**	
# MACs	64	512	
Clock Frequency (MHz)	500	250	62.5
Energy Efficiency(Inference/J, ResNet-20)	60247	21031	15610

TABLE III
ACCELERATOR BENCHMARK COMPARISON SUMMARY.

Accelerator A	[14]	[5]	[3]	[21]	[6]	[7]	[20]
SONA Energy Efficiency over A	5.1	2.4	1.7	2.7	1.7	1.0	0.8
SONA Speedup over A	5.2	2.1	1.3	1.5	2.1	1.2	2.5

naive architecture, SONA does not incur unnecessary memory storage and access as it capitalizes on CBB sparsity's ability to maximize memory utilization and limit zero padding.

B. Post-Synthesis Design Comparison

To better situate SONA's energy efficiency if it were to be fabricated on a chip, we compare synthesized SONA against the synthesized CNN accelerator design made available to us in [19] in TSMC 22nm. We evaluate both at their minimum slack design point (250 MHz for [19] and 500 MHz for SONA). Since the design in [19] has an 512 MAC array that is $8\times$ more than that in SONA (Table II), we also evaluate [19] at 62.5MHz so that both have the same number of MAC operations per unit time. We exclude off-chip DRAM access energy because both designs store all weights and activations in on-chip memory for ResNet-20. If DRAM was included for bigger networks, SONA would benefit from fewer accesses due to fewer sparse-orthogonal overlapped kernels. Compared to [19] synthesized at $\{250, 62.5\}$ MHz, SONA is $\{2.9\times, 3.9\times\}$ more energy-efficient for ResNet-20.

C. Comparisons with Sparsity-Aware Accelerators

We also present comparisons with recent sparsity-aware accelerators listed in Table III. As open-source simulators are not available to perform direct comparisons for those designs (except [5], [14]), we provide indirect comparisons via the gains claimed in [3], [6], [7], [20], [21] for common/similar benchmarks. For example, [21] (8-bit) reports $1.4\times$ speedup and $0.9\times$ energy-efficiency compared to [5] (8-bit), overall. We can then extrapolate that SONA is $2.7\times$ more energy efficient and achieves a $1.5\times$ speedup over [21], overall. The methodology described in this example is identical for [3], [6], [7], [20]. Table III shows that SONA exhibits higher energy efficiency and speedup in general. Sec. V discusses the inefficiencies in those designs and SONA's relative advantages.

V. RELATED WORKS

Sparse-Winograd CNNs have been studied prior to HTNNs to reduce the computational complexity of convolution [8], [9]. However, our training of ResNet18 using ImageNet shows that HTNN can outperform [8] in terms of top-1 accuracy (69.8% for HTNN vs. 66.8% in [8]). It is also shown in [10] that the complexity of HTNN is 1.93 – $5.12\times$ lower than that of sparse-Winograd models. SONA is able to support a variety of CNNs since HTNN is generalizable to kernels of any size and in particular can replace the commonly used 3×3 convolution without accuracy degradation. For depthwise separable convolutions, kernel sparse-orthogonality is foregone (i.e., $n_t = 1$) and SONA's improvement will solely stem from transform-domain element-wise multiplications. For example, if applied to MobileNetV1/2, SONA can theoretically save $\approx 48\%/56\%$ MAC operations for those layers, respectively. However, MobileNetV1/2 cannot fully take advantage of HTNN optimizations as their primary workload is 1×1 point-wise convolution (supported by SONA but identical in both HTNNs and CNNs).

To take advantage of both weight and/or activation sparsity, several sparsity-aware accelerators have been proposed in the literature. Both [5] and [4] explore unstructured sparsity and suffer from load imbalance. [3] only performs the necessary multiplications using a high-energy intersection operation which causes frequent stalls. [6] and [7] use an indexing module requiring wide and costly multiplexers. [7] reduces the multiplexer size in [6] by grouping weights but it observes a performance overhead from irregular weight block locations. Although SONA only targets weight sparsity, it mitigates load imbalance via structured sparse-orthogonal weights. Other works exploit inherent bit-level sparsity. Most recently, [21] eliminates ineffectual multiplications with intermediate zero bits, targeting varying bit-widths and exhibiting significant gains for higher bit precisions (16 bits). Meanwhile, SONA weights are 8-bit BS-CSD, therefore trading off configurability for increased opportunity of more specific optimizations in addition to using sparse orthogonality. Other approaches target structured model compression. [22] leverages block-circulant matrices and Fast Fourier Transform (FFT) based convolution. Although [22] supports convolution and FCL, it does not leverage sparsity and requires complex FFT hardware, whereas SONA employs patch-based WHT only requiring additions. [20] leverages structured sparsity using a time-unrolled systolic array to serialize MAC processing, demonstrating gains in energy at the cost of reduced speedup.

VI. CONCLUSION

In this paper we introduce SONA, a novel accelerator for transform-domain neural networks with structured sparse-orthogonal weights where convolution operations are replaced by element-wise multiplications. SONA employs an HTNN-specific OS dataflow coupled with an energy-efficient memory organization to reduce the overhead of sparse-orthogonal WHT-domain concurrent processing. SONA leverages BS-CSD weights to reduce multiplications to simpler additions. Moreover, SONA uses CBB structured pruning to efficiently support sparse FCLs while maintaining full MAC array utilization. For convolution layers and FCLs, SONA can reduce the inference energy by $5.1\times$ and $2.4\times$, respectively, compared to prior accelerator designs.

REFERENCES

[1] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[2] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2016.

[3] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. N. Vijaykumar, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 151–165. [Online]. Available: https://doi.org/10.1145/3352460.3358291

[4] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 243–254.

[5] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 27–40. [Online]. Available: https://doi.org/10.1145/3079856.3080254

[6] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[7] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 15–28.

[8] S. R. Li, J. Park, and P. T. P. Tang, "Enabling sparse winograd convolution by native pruning," *CoRR*, vol. abs/1702.08597, 2017. [Online]. Available: http://arxiv.org/abs/1702.08597

[9] X. Liu, J. Pool, S. Han, and W. J. Dally, "Efficient sparse-winograd convolutional neural networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=HJzgZ3JCW

[10] Y. Chen, B. Liu, P. Abillama, and H.-S. Kim, "Htnn: Deep learning in heterogeneous transform domains with sparse-orthogonal weights," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2021, pp. 1–6.

[11] W. K. Pratt, J. Kane, and H. C. Andrews, "Hadamard transform image coding," *Proceedings of the IEEE*, vol. 57, no. 1, pp. 58–68, 1969.

[12] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. [Online]. Available: http://arxiv.org/abs/1801.04381

[13] R. M. Hewlitt and E. Swartzlantler, "Canonical signed digit representation for fir digital filters," in *2000 IEEE Workshop on SiGNAL PROCESSING SYSTEMS. SiPS 2000. Design and Implementation (Cat. No. 00TH8528)*. IEEE, 2000, pp. 416–426.

[14] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367–379.

[15] S. Gudaparthi, S. Narayanan, R. Balasubramonian, E. Giacomin, H. Kambalasubramanyam, and P.-E. Gaillardon, "Wire-aware architecture and dataflow for cnn accelerators," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–13. [Online]. Available: https://doi.org/10.1145/3352460.3358316

[16] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.

[17] isakedo, "Dnnsim," https://github.com/isakedo/DNNsim, 2020.

[18] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina, C. Kozyrakis, and M. Horowitz, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 369–383. [Online]. Available: https://doi.org/10.1145/3373376.3378514

[19] H. An, S. Schiferl, S. Venkatesan, T. Wesley, Q. Zhang, J. Wang, K. D. Choo, S. Liu, B. Liu, Z. Li, L. Gong, H. Zhong, D. Blaauw, R. Dreslinski, H. S. Kim, and D. Sylvester, "An ultra-low-power image signal processor for hierarchical image recognition with deep neural networks," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 4, pp. 1071–1081, 2021.

[20] Z.-G. Liu, P. N. Whatmough, Y. Zhu, and M. Mattina, "S2ta: Exploiting structured sparsity for energy-efficient mobile cnn acceleration," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 573–586.

[21] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos, "Laconic deep learning inference acceleration," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 304–317.

[22] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan, X. Ma, Y. Zhang, J. Tang, Q. Qiu, X. Lin, and B. Yuan, "Circnn: Accelerating and compressing deep neural networks using block-circulant weight matrices," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 395–408. [Online]. Available: https://doi.org/10.1145/3123939.3124552