Winols: A Large-Tiling Sparse Winograd CNN Accelerator on FPGAs

KUNPENG XIE, YE LU, XINYU HE, DEZHI YI, and HUIJUAN DONG, Nankai University, Tianjin Key Laboratory of Network and Data Security Technology, and the Key Laboratory of Data and Intelligent System Security, Ministry of Education, Tianjin, China
YAO CHEN, National University of Singapore, Singapore, Singapore

Convolutional Neural Networks (CNNs) can benefit from the computational reductions provided by the Winograd minimal filtering algorithm and weight pruning. However, harnessing the potential of both methods simultaneously introduces complexity in designing pruning algorithms and accelerators. Prior studies aimed to establish regular sparsity patterns in the Winograd domain, but they were primarily suited for small tiles, with domain transformation dictating the sparsity ratio. The irregularities in data access and domain transformation pose challenges in accelerator design, especially for larger Winograd tiles. This paper introduces "Winols," an innovative algorithm-hardware co-design strategy that emphasizes the strengths of the large-tiling Winograd algorithm. Through a spatial-to-Winograd relevance degree evaluation, we extensively explore domain transformation and propose a cross-domain pruning technique that retains sparsity across both spatial and Winograd domains. To compress pruned weight matrices, we invent a relative column encoding scheme. We further design an FPGA-based accelerator for CNN models with large Winograd tiles and sparse matrix-vector operations. Evaluations indicate our pruning method achieves up to 80% weight tile sparsity in the Winograd domain without compromising accuracy. Our Winols accelerator outperforms dense accelerator by a factor of 31.7× in inference latency. When compared with prevailing sparse Winograd accelerators, Winols reduces latency by an average of 10.9×, and improves DSP and energy efficiencies by over 5.6× and 5.7×, respectively. When compared with the CPU and GPU platform, Winols accelerator with tile size 8×8 achieves 24.6× and 2.84× energy efficiency improvements, respectively.

CCS Concepts: • **Hardware → Reconfigurable logic and FPGAs**; **Emerging technologies;**

Additional Key Words and Phrases: Large-tiling, sparse CNNs, cross-domain pruning, Winograd accelerator

ACM Reference Format:
Kunpeng Xie, Ye Lu, Xinyu He, Dezhi Yi, Huijuan Dong, and Yao Chen. 2024. Winols: A Large-Tiling Sparse Winograd CNN Accelerator on FPGAs. *ACM Trans. Arch. Code Optim.* 21, 2, Article 31 (March 2024), 24 pages. https://doi.org/10.1145/3643682

This work is partially supported by the National Natural Science Foundation (62372253, 62002175), the Natural Science Foundation of Tianjin Fund(23JCYBJC00010), the CCF-Baidu Open Fund(No.CCF-Baidu202310), and the Open Project Fund of State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences (CARCHB202016).

Authors' addresses: K. Xie, Y. Lu (Corresponding author), X. He, D. Yi, and H. Dong, Nankai University, Tianjin Key Laboratory of Network and Data Security Technology, and the Key Laboratory of Data and Intelligent System Security, Ministry of Education, Tongyan Road 38, Tianjin, China, 300350; e-mails: xkp@mail.nankai.edu.cn, luye@nankai.edu.cn, xyhe@mail.nankai.edu.cn, ydz@mail.nankai.edu.cn, donghj@mail.nankai.edu.cn; Y. Chen, National University of Singapore, 15 Computing Drive, Singapore, Singapore, 117418; e-mail: yaochen@nus.edu.sg.

ACM 1544-3566/2024/03-ART31
https://doi.org/10.1145/3643682

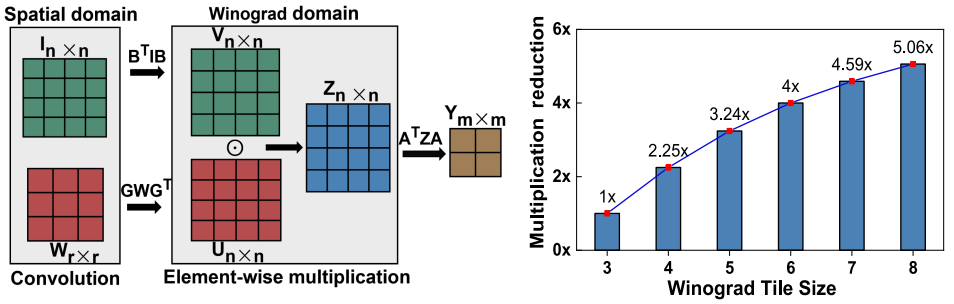
1 INTRODUCTION

Convolutional neural networks (CNNs) have been widely applied for promoting modern artificial intelligence systems [1, 16, 20, 38]. However, deploying large-scale CNNs on edge platforms is challenging due to their high computational and memory bandwidth requirements [3, 11, 18, 34, 53]. To address these issues, Winograd minimal filtering algorithm (in short, Winograd) has been applied in many popular CNN inference frameworks, such as cuDNN [8] and MKL-DNN [39] to facilitate the acceleration of CNN computations. Winograd has been proven effective in reducing costly multiplication operations through using cheaper additions, thus accelerating CNN inference [17, 22, 27]. This is even more popular for FPGAs that require either costly **Digital Signal Processing units (DSPs)** or huge amounts of **Look-Up Tables (LUTs)** to calculate multiplication [4–6, 10, 13, 52].

The Winograd algorithm comprises four distinct steps, namely input transformation, kernel transformation, element-wise matrix multiplication, and output transformation [22]. Throughout these processes, the original input feature map and weight undergo a transformation from the spatial domain to the Winograd domain. The computational demands in the Winograd domain are fewer than those in the spatial domain, resulting in reduced hardware intense multiplication operations. As shown in Figure 1(a), the input feature maps and weights are transformed into the Winograd domain and processed tile by tile. The original results in the spatial domain require $m \times m \times r \times r$ multiplications and $m \times m \times (r \times r - 1)$ addition operations. In the Winograd domain, obtaining the same results only requires $(m + r - 1) \times (m + r - 1)$ multiplications by the element-wise matrix multiplication. In Figure 1(a), the domain transformation can be processed pre- or after-computation, including input transformation $B^T IB$, kernel transformation GWG^T and output transformation $A^T ZA$. These domain transformation operations are processed with constant matrices B, G, and A, which can be further processed with just addition operations to save resource cost [22]. The number of required additions is determined by the element values within these constant domain transformation matrices. By employing varied tile sizes, Winograd exhibits different normalized reduction ratios of multiplications. As shown in Figure 1(b), it highlights that larger tile sizes enable higher reduction ratios of multiplications, which can facilitate enhanced hardware efficiency and improved performance. For example, when increasing the input tile size from 4×4 to 6×6, 1.72× performance enhancement can be obtained, as mentioned in [19].

Pruning techniques effectively eliminate redundant or insignificant weights in both the spatial domain [15, 21, 32, 51] and Winograd domain [28, 47] along with reduced multiplication operations. Researchers have leveraged the Winograd domain transformation and pruning algorithm to achieve higher multiplication reduction ratios, such as Winograd-ReLU pruning [28] and **Sub-Row-Balanced Sparsity (SRBS)** pruning [47]. However, directly pruning weights in the Winograd domain causes obvious accuracy loss. For instance, SRBS [47] analyzes the importance of Winograd weights and prunes them in the Winograd domain, which achieves 75% sparsity for ResNet18, but at the cost of a 2% reduction in model accuracy. When pruning weights in the spatial domain, the sparsity cannot be sustained after transforming into the Winograd domain. For example, the 85.6% weight sparsity in the spatial domain for AlexNet diminishes to 25.3% in the Winograd domain after domain transformation [37]. These issues become more severe with larger Winograd tile sizes, thus hindering the benefits of higher multiplication reduction.

Sparse Winograd accelerators such as **sparse Winograd convolution accelerator (SpWA)**, [30] and **sparse-Winograd matrix multiplication CNN accelerator (SWM)** [43], have been developed to exploit the inherent sparsity in CNN models. However, the varying sparsity levels of Winograd filters across different CNN layers critically impede the efficiency of both SpWA and SWM. This diversity in sparsity leads to idle cycles within computing units. For instance, the SpWA accelerator experiences a high idle proportion of 50% or more. To resolve



(a) Winograd transformation (b) Multiplication reduction

Fig. 1. Winograd transformation and normalized multiplication reduction ratio to general convolution.

this problem, a workload-balanced sparse Winograd accelerator named WSRBS [47] is proposed to eliminate idle cycles and maximize the utilization of the computing unit. WSRBS leverages the regular sparsity of the SRBS pruning method to enhance CNN model inference efficiency on FPGA. This accelerator employs a Dense-Multiplication-Sparse-Addition module for sparse matrix-vector multiplication, but its data multiplexers consume a considerable amount of LUTs. Specifically, WSRBS utilizes up to 81% of on-chip LUTs, which limits its parallelism and constrains its operating frequency to 166MHz. Additionally, increasing the Winograd tile size exacerbates the high LUT utilization problem during hardware implementation.

Existing pruning algorithms together with sparse accelerators are constrained by their support for only a small Winograd tile size of 4×4 [30, 43, 47]. This limitation severely restricts the reduction in multiplication operations. When employing a large Winograd tile size, previous pruning algorithms incur a noticeable loss in accuracy [47] as well as fail to maintain the sparsity in the Winograd domain [37]. Current sparse accelerators for a large tile size introduce significant hardware resource costs and suffer from low operating frequency [43, 47]. As a result, previous research cannot support a larger Winograd tile size. Exploring the effective utilization of larger tiles is thus valuable for enhancing data-parallel computing capabilities.

To overcome the above challenges, we propose Winols, the first sparse Winograd accelerator for CNNs on FPGAs that supports large tile sizes up to 8×8. In Winols, we first observe the weight transformation relation between the spatial domain and Winograd domain, and conclude a guideline for subsequent pruning algorithms. We then propose a novel pruning algorithm for spatial domain weights which can improve the sparsity in the Winograd domain while maintaining the accuracy. Next, we construct a parameterized workload-adaptive PE array with a sequence of optimization techniques, including the sparse weight encoding method and sparsity-aware PE architecture to accelerate the sparse Winograd processing. At last, we build a resource and latency analysis model to search expected hardware configuration parameters to facilitate Winols accelerator design. The contributions of this paper can be summarized as follows:

— **Cross-domain sparsity transformation**. We define a relevancy degree evaluation to measure the relation between spatial and Winograd domain weights. On top of it, we present a cross-domain block-balanced pruning method which can ensure up to 80% sparsity with negligible accuracy loss in the Winograd domain.

— **Relative-index encoding scheme**. We design the **relative column index (RCI)** encoding scheme to compress the sparse weight matrices. RCI simultaneously indicates the positions of nonzero elements in both the original sparse matrix and the targeted dense matrix. This prompts constraining the range of candidate values in the data selection process, thus reducing the resource consumption required for the data multiplexer.

— **Sparsity-aware accelerator architecture**. We design a novel sparsity-aware accelerator architecture to support multiple PEs aiming at parallelly processing different tasks with various sparsity. These PEs directly take our encoded matrices as inputs, and utilize the sample module and multiplication module to perform the sparse Winograd calculation.
— **State-of-the-art performance**. Compared with the state-of-the-art Winograd accelerator, when implemented on the Xilinx ZCU102 platform, experimental results show that Winols accelerator averagely reduces the latency by **10.9**×, and boosts the DSP efficiency by **5.6**× and energy efficiency by **5.7**×. When implemented on the Xilinx VCU118 platform, Winols achieves an extra **1.42**× improvement in inference speed compared with the Xilinx ZCU102 platform.

2 BACKGROUND AND MOTIVATION

2.1 Winograd Minimal Filtering Algorithm

Winograd minimal filtering algorithm reduces the number of multiplications of convolution at the cost of extra addition operations [22], as shown in Figure 1. The processing of output feature Y can be described with the following formula:

$$Y = A^T(GWG^T \odot B^T IB)A. \tag{1}$$

Here, $\odot$ denotes element-wise multiplication. The spatial domain convolutional kernel W and input tile I have sizes $r \times r$ and $n \times n$, respectively. The transformation matrices A, B, and G vary according to the Winograd tile size. Winograd denotes the computation of the $m \times m$ outputs with a filter size of $r \times r$ as $F(m \times m, r \times r)$ [22]. For a specific $F(m \times m, r \times r)$, the domain transformation matrices A, B, and G remain constant. As an example, when configuring m as 4 and r as 3, the matrices A, G, and B for $F(4 \times 4, 3 \times 3)$ are detailed as follows:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 2 & 4 & 8 \\ 1 & -2 & 4 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix}, G = \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ -\frac{1}{6} & -\frac{1}{6} & -\frac{1}{6} \\ -\frac{1}{6} & \frac{1}{6} & -\frac{1}{6} \\ \frac{1}{24} & \frac{1}{12} & \frac{1}{6} \\ \frac{1}{24} & -\frac{1}{12} & \frac{1}{6} \\ 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -4 & 4 & -2 & 2 & 4 \\ -5 & -4 & -4 & -1 & -1 & 0 \\ 0 & 1 & -1 & 2 & -2 & -5 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2}$$

During the domain transformation process, the multiplication calculations involving A and A^T or B and B^T can be replaced with addition operations [22]. In addition, the domain transformation of weights can be pre-processed since the model weights remain constant during the inference stage. We can observe that the required number of multiplications is $(m+r-1) \times (m+r-1)$, while conventional convolution requires $m \times m \times r \times r$ multiplications. When increasing the Winograd tile size, the Winograd convolution can significantly reduce the number of multiplications.

2.2 Integration of Weight Pruning with Winograd

Many pruning methods have been proposed to enhance the inference speed of CNN models [15, 23, 24, 29, 32, 35]. These methods can be classified into two types based on their pruning patterns: unstructured pruning and structured pruning. In unstructured pruning, all weights in the filters are evaluated, and those below a specific threshold are removed. This approach achieves high sparsity and model accuracy compared with other pruning methods [32]. However, it introduces serious irregularity to the weights, and results in complex control flow and inefficient data access during computation. Structured pruning methods employ relatively structured patterns, such as channel-level pruning [23], filter-level pruning [29], and pattern-based pruning [35].

These structured patterns effectively reduce irregularity but achieve lower sparsity compared with unstructured pruning methods.

The sparsity introduced by pruning can reduce the computational requirement and memory footprint [12, 36, 44, 45, 50]. This drives the researchers to take advantage of integrating the Winograd algorithm with pruning methods to further reduce the computation costs. There are three modes to integrate the pruning method with the Winograd algorithm:

— **pruning and retraining both in spatial domain**. Spatial-Winograd [48] introduces the concept of a relevance set and employs the maximum norm of the relevance set to prune and retrain CNN weights in the spatial domain. However, this approach overlooks the impact of pruning on spatial-domain weights, thus leading to high sparsity in the spatial domain but low sparsity in the Winograd domain. Moreover, the transformation from the spatial domain to the Winograd domain introduces irregular sparsity, posing challenges for the CNN accelerator design.

— **pruning and retraining both in the Winograd domain**. Native Pruning [26] necessitates an extremely small learning rate, which results in lengthy training time and convergence issues for deeper networks. SRBS [47] adopts the Pearson coefficient [2] to help model train but requires a small Winograd tile size, limiting the benefits of using larger Winograd tiles.

— **pruning in Winograd domain and retraining in spatial domain**. This approach introduces inconsistencies between the spatial and Winograd domain due to the domain transformation, ultimately leading to decreased model accuracy [28].

2.3 Winograd-based CNN Accelerator

Various accelerator architectures have been proposed to enhance the inference speed of CNN models [9, 14, 42, 49]. Based on their computing architecture, these accelerators can be classified into two types:

— Accelerators for dense neural networks. These accelerators primarily utilize matrix multiplication processing cores [49], systolic array architectures [42], either **application-specific integrated circuit (ASIC)** or reconfigurable hardware implementations [7, 14] to accelerate convolution computations. However, the limitations imposed by their available on-chip resources hinder further reduction in processing latency and computational efficiency improvements.

— Accelerators for sparse neural networks. These accelerators [25] introduce sparsity into DNN models and design customized computing engines to effectively exploit the sparsity. Nevertheless, these accelerators offer limited flexibility for sparse models. Sparse tensor cores are proposed to bypass multiplication and accumulation calculations involving zero-valued weights, but they introduce pipeline bubbles and impose significant complexity on the computing engine [41, 54].

Despite the domain-specific accelerators designed for both dense and sparse models, numerous Winograd-based CNN accelerators have been proposed to exploit the advantage of Winograd algorithm. The authors in [31] first propose a hardware architecture for Winograd-based CNN inference on FPGA. Their design incorporates a pipeline Winograd engine and demonstrates the effectiveness of the Winograd-based FPGA accelerator. WinoCNN [27] introduces a highly efficient systolic array accelerator that supports multiple convolution kernel sizes using the same computing resources, eliminating the need for reconfiguration. UniWiG [19] presents a unified architecture that leverages a shared set of processing elements for both Winograd-based convolution and GEMM computations. This unified design enables efficient utilization of FPGA

hardware resources. However, none of the above accelerators adopt pruning methods and thus cannot leverage the sparsity in both Winograd domain and spatial domain.

To enhance the inference efficiency of sparse models, specialized hardware architectures are required to handle data access and computation effectively. Approaches such as SWM [43] and LSW-CNN [40] employ the ReLU-modified algorithm to leverage sparsity in both Winograd weights and activations. SpWA [30] introduces a sparse Winograd convolution accelerator that assigns sparse matrices to different **processing elements (PEs)**. Incidentally, each PE has its unique sparsity ratio for activation and filter, which varies across different CNN layers. It implies that the fastest PE with a high sparsity ratio must wait for the slowest PE until its computation finishes. PEs' entire efficiency improvement is thus impeded in SpWA, SWM, and LSW-CNN. For example, in the SpWA accelerator, the Winograd weight sparsity differences can cause massive idle cycles in PEs, and even achieve a 50% idle proportion. WSRBS [47] proposes a dedicated accelerator tailored to different sparsity patterns but utilizes up to 81% of on-chip LUTs for sparse addition. This intensive utilization of LUTs not only limits the parallelism but also lowers the operating frequency to 166MHz.

In summary, firstly, the high sparsity ratio attained through weight pruning in the spatial domain is broken by the transformation to the Winograd domain. Secondly, current Winograd-based CNN accelerators can not effectively exploit larger Winograd tile sizes to achieve more multiplication reductions. These two reasons motivate us to propose a cross-domain pruning method to maintain the sparsity ratio without compromising model accuracy, and design an accelerator with a larger Winograd tile size to further release the computational effectiveness.

3 SPARSITY MAINTAINED CROSS-DOMAIN PRUNING

3.1 Spatial-to-Winograd Relevance Matrix

Instead of directly pruning the weights in Winograd domain and suffering from accuracy loss [47] or long training overhead with an extremely small learning rate [26], we prune the weights in the spatial domain with careful consideration of the impact on the Winograd domain and maintain the controllable sparsity transition from the spatial domain to the Winograd domain.

To enable convenient expression and subsequent processing, we refer to the matrices GWG^T and B^TIB mentioned earlier as U and V matrices. They are the Winograd domain weights and features obtained through the transformation of spatial domain weight matrix W and feature matrix I, respectively. The element in U can be represented as:

$$U_{h,v} = \sum_{0 \le p,q \le r-1} \alpha_{h,v,p,q} \cdot w_{p,q}, \ (h, v \in [0, n-1]). \tag{3}$$

Here, $w_{p,q}$ represents the element in the matrix W. α is a coefficient tensor for weight transformation from spatial domain to Winograd domain. h and v refer to the horizontal and vertical indexes of the element in the matrix U.

Relevance set. We define a relevance set S which contains the elements of matrix W that are associated with nonzero coefficients to form the elements in U. For a specific element $U_{h,v}$, its corresponding S can be denoted as:

$$S_{h,v} = \{w_{p,q} | \alpha_{h,v,p,q} \neq 0, 0 \le p, q \le r-1\}, \ (h, v \in [0, n-1]). \tag{4}$$

Here, pruning the weight set $S_{h,v}$ in the spatial domain leads to an equivalent pruning of the corresponding weight $U_{h,v}$ in the Winograd domain. The relevance set S helps maintain sparsity from the spatial domain to the Winograd domain.

We have observed that the weight set S exhibits varying degrees across different Winograd weights. So we further investigate the mapping relationship indicated by the relevance set S

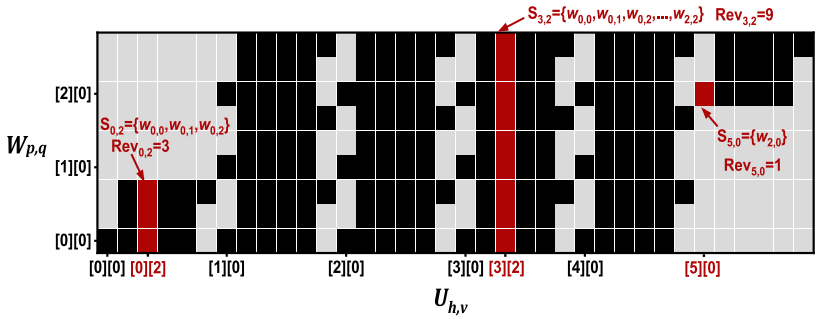


Fig. 2. Relevance matrix of U and W for F(4×4, 3×3). The black and red squares represent non-zero coefficients $\alpha_{h,v,p,q}$, while the grey square indicates a zero coefficient.

during the domain transformation, in contrast to the threshold-based pruning strategy such as the aforementioned Spatial-Winograd [48]. Spatial-Winograd only employs the maximum norm value of the relevance set S and the predefined threshold to prune the spatial domain weights. When the maximum norm value is below the threshold, the corresponding weight set S is pruned. Besides, all the Winograd weights are pruned with the same probability. Hence, in addition to the weight threshold, we also take into account the relevance involved.

Cardinal of relevance set. To account for the variability of relevancy set, we assign distinct probabilities to each Winograd weight based on the *cardinal* of set S. Here, the *cardinal* is defined as the relevance Rev. The value $Rev_{h,v}$ represents the number of elements in matrix W associated with the element $U_{h,v}$. By utilizing Rev, we can measure the relevance between the element $U_{h,v}$ in the Winograd domain and the matrix W in the spatial domain. To help better understand our design approach, we illustrate the relevance of 6×6 Winograd tile U and 3×3 convolution kernel W as an example in Figure 2. The two-dimensional matrix U is flattened and viewed as the horizontal index of the graph, with a similar process applied to matrix W. Considering the examples of $U_{0,2}$, $U_{3,2}$, and $U_{5,0}$, their calculations involve the summation of multiple spatial domain weights multiplied by their respective non-zero coefficients. The calculation of $U_{3,2}$ encompasses all the elements from W, so its relevance Rev is 9. Different Winograd domain weights exhibit varying degrees of relevance. For instance, the relevances of $U_{5,0}$ and $U_{0,2}$ are 1 and 3, respectively.

The *cardinal Rev* indicates the relevance of a weight in Winograd domain with the weights in spatial domain. To maintain desired sparsity in Winograd domain, Rev serves as a measure to reflect the number of weights that require pruning in the spatial domain. In addition, it highlights the varying importance of weights in the Winograd domain.

3.2 Relevance-Based Sparsity Re-distribution

Within a layer of the CNN model, the output and input feature maps consist of oc and ic channels, respectively. One output tile of the feature map of the ith channel, denoted as Y_i, is computed as follows:

$$Y_i = \sum_{0 \le j \le ic-1} A^T(\mathbf{U}_{i,j} \odot \mathbf{V}_j)A, \ (i \in [0, oc-1]). \tag{5}$$

Here, $\mathbf{U}$ and $\mathbf{V}$ represent tensors of the weights and input feature maps with multiple channels in Winograd domain, respectively. Specifically, $\mathbf{U}_{i,j}$ denotes the weight matrix used for element-wise multiplication with the jth input channel in order to process the ith output channel. Similarly, $\mathbf{V}_j$ represents the feature matrix of the jth input channel. The output transformation is required for all $\mathbf{U}_{i,j} \odot \mathbf{V}_j$ matrices across input channels.

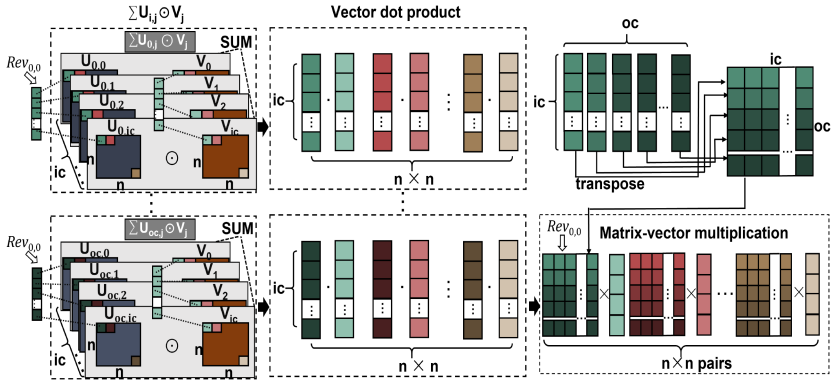


Fig. 3. Reorganization of Winograd convolution computation.

To reduce the total number of operations for intermediate matrix transformation, we can exchange the order of the summations in Equation (5). As a result, only one matrix transformation is required. The same output channel is then computed as:

$$Y_i = A^T \left(\sum_{0 \le j \le ic-1} \mathbf{U}_{i,j} \odot \mathbf{V}_j \right) A, \ (i \in [0, oc-1]). \tag{6}$$

Each tensor $\mathbf{U}_i$ has the same number of input channels with tensor $\mathbf{V}$. With Equation (6), the summations are immediately conducted after the element-wise multiplications between $\mathbf{U}_i$ and $\mathbf{V}$. Here, by flattening the tensors from the input channel dimension, the element-wise multiplications and summations can be seen as multiple vector-vector multiplications. As shown in Figure 3, the computation of $\mathbf{U}_0$ and $\mathbf{V}$ can be transformed into $n \times n$ vector-vector multiplications. By combining multiple $\mathbf{U}_i$ tensors that share the same $\mathbf{V}$ tensor, we can obtain $n \times n$ matrix-vector multiplication sets. For each matrix, the numbers of the row and column are equal to the numbers of the output channel and input channel, respectively.

It is worth noting that although the weight elements in a matrix-vector pair are transformed from different weight matrices, they have the same *Rev* value when they are from the same position of different matrices. For example, as shown in Figure 3, all the weight elements in the first matrix-vector pair have the same relevance $Rev_{0,0}$. All the matrix-vector pairs have their own specific *Rev* values. The reorganized computing pattern enables different sparsity for each matrix-vector pair, which is denoted as λ. When the desired sparsity ratio for a given CNN layer is ρ, the sparsity of the total weight matrices can be constrained as follows:

$$\rho = \frac{\sum_{0 \le i,j \le n-1} \lambda_{i,j}}{n \times n}, \ (0 \le \rho \le 1). \tag{7}$$

With our definition of the *cardinal Rev*, to prune the Winograd weight matrix with higher relevance, we need to trim more elements from the spatial domain weights matrix. To prevent from pruning more spatial domain weights, we assign a smaller sparsity-related factor for the Winograd weight matrix with higher relevance, which means the corresponding weights in the spatial domain will have a lower probability of being pruned. We then formulate a proportion of non-zero elements in the Winograd domain weight matrix as:

$$\begin{aligned} (1-\lambda_{0,0}) : (1-\lambda_{0,1}) : \ldots : (1-\lambda_{n-1,n-1}) = \\ f(Rev_{0,0}) : f(Rev_{0,1}) : \ldots : f(Rev_{n-1,n-1}). \end{aligned} \tag{8}$$

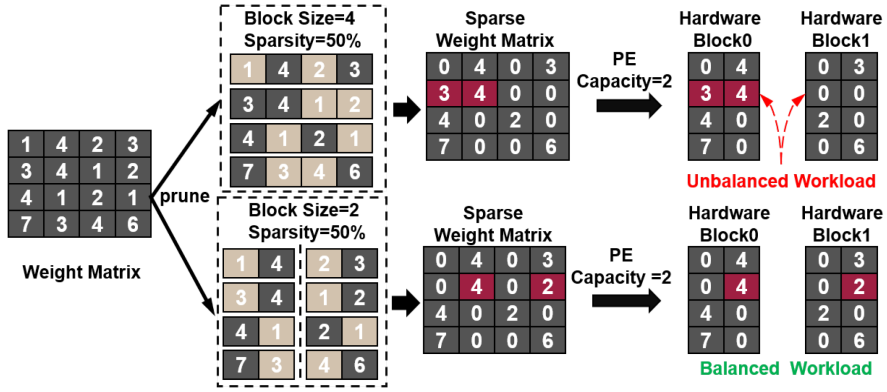


Fig. 4. Sparse patterns of different column granularity.

where the smooth function $f(\cdot)$ adopts $sqrt()$ for simplicity to avoid extreme sparsity allocation. Combining Equations (7) and (8), the sparsity of the weight matrix in a matrix-vector pair can be thus obtained as:

$$\lambda_{i,j} = 1 - \frac{n \times n \times (1-\rho) \times f(Rev_{i,j})}{\sum_{0 \le p,q \le n-1} f(Rev_{p,q})}. \tag{9}$$

3.3 Block-Balanced Pruning

Due to the limitations of on-chip memory in FPGAs, processing the entire $oc \times ic$ weight matrix as a whole is infeasible, so this matrix is processed block by block. We take an example to explain its details. As shown in Figure 4, directly pruning the weights only at the column dimension [46, 47] introduces imbalanced sparsity across different blocks. This imbalance easily leads to idle processing cycles during runtime for the processing elements. To address this issue and ensure balanced sparsity, we propose a block-balanced pruning method. By leveraging the relevance-based sparsity re-distribution strategy, each matrix-vector pair is assigned a specific sparsity value $\lambda_{i,j}$ during pruning. We then divide the weight matrix into multiple blocks based on the processing capacity of the processing elements and assign the sparsity value of $\lambda_{i,j}$ to each block. Moreover, to facilitate weight matrix computations, we maintain the sparsity for the rows in the weight matrix as $\lambda_{i,j}$. As shown in Figure 4, the sparsity of $\lambda_{i,j}$ is 50%. This block-balanced pruning method guarantees that each block has the same sparsity and further reduces the runtime complexity of the processing elements as presented in the following Section 4.

Following the allocation of specific sparsity to each block, pruning can be performed. Initially, a dense model is trained in the spatial domain by using Winograd computation for convolution layers. During the subsequent pruning phase, the weight matrix of the pre-trained dense model is divided into multiple blocks. Within each block, weight elements with lower values are selectively pruned, and 0/1 masks are utilized to identify the pruned weight elements. The sparse model is then fine-tuned with these masks to restore accuracy. To minimize substantial accuracy reduction, an iterative pruning strategy is employed, which gradually increases sparsity until the expected ratio is achieved.

4 SPARSITY-AWARE LARGE-TILE WINOGRAD ACCELERATOR ARCHITECTURE

To enhance the efficiency of our Winograd accelerator, we propose the sparsity-maintained cross-domain pruning method that obtains high sparsity for Winograd domain weights without model

```
①:for(toc=0;toc<OC;toc+=Moc) //output channel
   load_weight_and_mask()
   for(b=0;b<BATCH;b++) //batch
   for(th=0;th<OH;th+=m) //output row
       load_input()
       computing()
       store()
                    ②:for(ttoc=0;ttoc<Moc;ttoc+=Poc) //output channel
                       for(tic=0;tic<IC;tic+=Pic) //input channel
                       for(tw=0;tw<OW;tw+=m) //output column
                          #pragma HLS PIPELINE
                          ③:winograd_engine()
```

Fig. 5. Nested-loop expression of a convolution layer.

accuracy degradation, which further reduces the requirement of multiplication operations. In this section, we introduce the details of our sparsity-aware large-tiling accelerator architecture. To support large Winograd tile size for higher computation efficiency, there are two primary difficulties to overcome:

— High resource consumption of processing elements with larger tile sizes. During sparse matrix-vector multiplication, we strive to skip the computation which involves zero weights. So we propose a sparse data encoding scheme that retains only non-zero weight values and utilizes indexes to determine their respective positions. By employing these indexes, we select the appropriate input elements from the vector to calculate multiplication with the non-zero weights. However, selecting elements in parallel by utilizing multiple indexes will introduce irregular data access and consume a substantial amount of logic resources. It is important to note that the Winograd domain transformation utilizes additions to replace multiplications that have already consumed logic resources. When the Winograd tile size increases, the data values within the Winograd domain transformation matrices grow significantly, thus leading to more resource consumption for addition operations. For instance, sparse Winograd accelerator WSRBS [47], whose tile size is 4×4 consumes 81% LUT resources of the targeted platform. High resource utilization can constrain both parallelism and operating frequency.
— Irregular sparsity patterns within weight matrices. Our cross-domain pruning method assigns different sparsity to matrix-vector pairs based on their *Rev* values. However, processing these matrix-vector multiplications on the same hardware module leads to reduced efficiency due to inconsistent sparsity requiring varied computational modes. This inspires the design of a specialized processing flow that effectively disassembles the Winograd convolution into multiple matrix-vector multiplications and enables their parallel execution.

4.1 Overall Accelerator Framework

The pseudocode of the nested loop for a convolution layer is shown in Figure 5, including three prominent phases of loops. ① During the first phase, data transfer occurs between the off-chip memory and on-chip BRAMs. To ensure efficiency, we employ a double buffering mechanism to overlap computing and data transfer. ② In the second phase, the on-chip input feature maps and weights are accessed in blocks of size $Poc \times Pic$ by using a pipelined approach. Each block comprises $n \times n$ vectors and matrices. The optimal configuration of *Poc* and *Pic* is explored by our design space exploration model. ③ In the third phase, the input feature tile and compressed weight tile are processed by our Winograd engine. The Winograd engine handles the transformation of inputs to Winograd domain, element-wise matrix multiplication as well as the subsequent conversion of the output back to the spatial domain.

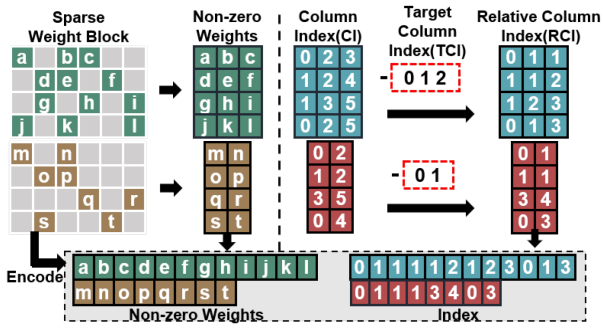


Fig. 6. RCI encoding for sparse weight matrix.

4.2 Sparse Weight Encoding

To effectively bypass the zero values within the pruned Winograd weight matrices, we propose a sparse encoding scheme based on our **relative column index (RCI)**. RCI encoding method skips the zero elements and only reserves the non-zero elements with their column indexes. So this encoding enables the compression of the weight block into two components: a non-zero weight matrix and an index matrix. Numbers of non-zero elements and indexes are determined by the sparsity ratio assigned via the sparsity re-distribution strategy.

An illustrative example of the details of our relative column index (RCI) encoding approach is shown in Figure 6. Benefiting from our block-balanced pruning, the compressed non-zero weight matrix has the same number of elements in each row. The **column index (CI)** is sufficient to locate the non-zero weights within a block. Furthermore, we construct the RCI by calculating the difference between CI and the **target column index (TCI)**, where TCI represents the offset of each CI index along the column dimension in the column index matrix. By employing the RCI encoding scheme, the compressed weight matrix and the corresponding index matrix can be obtained together as shown in Figure 6. The data width of value in the index matrix only requires a few bits to store the offset of columns. RCI can help achieve efficient compression of the weight block. When the Winograd sparsity exceeds a specific threshold such as 33.33% in Winols, RCI can reduce storage requirements obviously.

4.3 Parallelism and Pipeline Optimizations

We subsequently conduct an analysis on the potential for parallelism and pipeline optimization within the Winograd algorithm, aiming to enhance the efficiency of the accelerator, as illustrated in Figure 7. After reorganizing the computation, we can derive $n \times n$ independent matrix-vector pairs. The inherent independence enables us to parallelize the computation of these pairs, thereby achieving parallelism in both the row dimension and the column dimension of the output feature map. Besides, we introduce parallelism into the output channels and the input channels by unrolling each matrix-vector multiplication. Overall, we obtain parallelism in four dimensions, which are the output channel, input channel, output row, and output column. Furthermore, we treat the processing of each $n \times n$ matrix-vector pair as an individual subtask, and multiple such independent subtasks constitute a complete convolutional layer processing. Because these independent subtasks can be processed in a pipelined manner to maximize the reuse of hardware resources, we propose the design of a specialized Winograd engine that is capable of processing $n \times n$ matrix-vector pairs within a single clock cycle. This design enables us to establish a deep pipeline for enhanced computational throughput and efficiency.

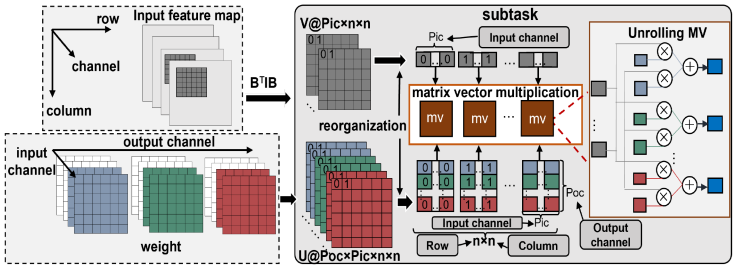
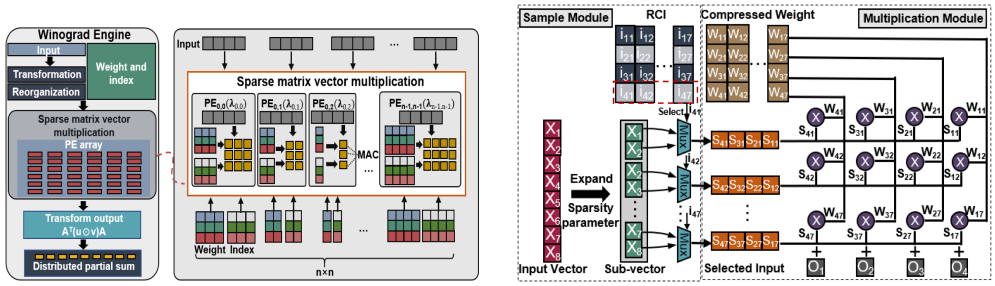


Fig. 7. Parallelism and pipeline of Winograd algorithm.



(a) Sparsity-aware Winograd PE array (b) PE micro-architecture

Fig. 8. Sparsity-aware Winograd PE array and PE micro-architecture.

4.4 Sparsity-aware PE Architecture

The core component of the Winograd engine is the sparse matrix-vector multiplication module, as shown in Figure 8(a). We construct $n \times n$ **processing elements (PEs)** to process the $n \times n$ **matrix-vector (MV)** pairs in parallel. Each PE is responsible for processing one matrix-vector pair with a specific sparsity. PE is designed to directly process the compressed matrix to save hardware resources. As previously mentioned in Section 3, different matrix-vector pairs have different computation workloads because of the different sparsity. To be adaptive to different computation workloads, each PE is configured with a certain number of **multiplier accumulators (MACs)** to process the sparse matrix-vector multiplication based on the sparsity of the inputs.

Within Winols PE, we introduce a sample module and a multiplication module, as shown in Figure 8(b). The sample module selects input elements from the appropriate offset of the input vector by utilizing our RCI index. The input vector is expanded into multiple sub-vectors, each consisting of consecutive elements from the original vector. The size (S_{sv}) and number (N_{sv}) of these sub-vectors are based on the sparsity factor and block size, as defined in Equation (10):

$$\begin{aligned} S_{sv} &= \lceil Pic \times \lambda_{i,j} \rceil + 1, \\ N_{sv} &= \lceil Pic \times (1 - \lambda_{i,j}) \rceil. \end{aligned} \tag{10}$$

By using RCI as the absolute index, each index selects a single data from the corresponding sub-vector. The sample module and RCI encoding method can help reduce the resources of each multiplexer. In the multiplication module, the selected input elements are multiplied with the weights to generate the output. It is easy to know that the PEs with lower sparsity require fewer logical resources for data extraction, and the PEs with high sparsity consume fewer resources for multiplication.

In addition to the above calculation process, the data precision configuration must also be taken into consideration. In the Winograd engine, both the input transformation and sparse matrix-vector multiplication will generate the intermediate results. When increasing the Winograd tile size, the data values within the Winograd domain transformation matrix grow significantly, thus leading to more bits are required to achieve higher precision for these intermediate results. For example, in the case of input transformation, the calculation results of F(4×4, 3×3) needs an extra five bits compared with F(2×2, 3×3). To meet these diverse precision requirements, we can evaluate the data distribution of the intermediate results and adopt different bitwidths for different tile sizes.

4.5 Resource and Performance Models

To search the desired hardware configurations, we select DSP and BRAM to build the resource usage model, because they are the most essential computation and storage resource in Winols accelerator. The DSP is mainly used by the MV multiplication that is related to the specific sparsity and some additional costs such as index decoding. The total number of DSP usage can be formulated as:

$$R_{total}^{DSP} = \sum_{0 \le i,j \le n-1} Poc \times Pic \times (1 - \lambda_{i,j}) + \mathcal{E}. \tag{11}$$

where $Poc \times Pic \times (1 - \lambda_{i,j})$ is the number of multiplications in each PE and $\mathcal{E}$ represents the additional DSP usage.

BRAM is mainly utilized to store the input, weight, and output data. When storing an s size of data in a bw width of BRAM buffer, the cost of BRAM consumption depends on the capacity and maximum bit width of the BRAM block. For instance, on Xilinx FPGA platform, the BRAM block has a capacity of $18k$ and a maximum bit width of 36. Hence, the number of BRAM required can be modeled by:

$$B(s, bw) = \left\lceil \frac{s}{\lceil bw/36 \rceil \times 18k} \right\rceil \times \left\lceil \frac{bw}{36} \right\rceil. \tag{12}$$

In each clock, $n \times n$ input vectors with the size of ic can be accessed in parallel. The number of BRAM usage for input data can be defined as:

$$R_{input}^{BRAM} = 2 \times n \times n \times B\left(\left\lceil \frac{input\_size}{n \times n \times ic} \right\rceil, ic \times input\_width\right). \tag{13}$$

input_size and *input_width* represent the on-chip input buffer size and data width, respectively. The number of BRAM usage for weights and output feature maps can be obtained in a similar process.

For a CNN layer, the output feature maps are generated tile by tile. Each tile contains m rows of Moc channels output feature maps. The approximated computing latency for a tile can be formulated as:

$$L_{tile} = \left\lceil \frac{Moc}{Poc} \right\rceil \times \left\lceil \frac{IC}{Pic} \right\rceil \times \left\lceil \frac{OW}{m} \right\rceil. \tag{14}$$

IC and OW are the channel number of the input feature maps and the column number of the output feature maps, respectively. The number of tiles to be processed for a layer is

$$N_{tile} = \left\lceil \frac{OC}{Moc} \right\rceil \times BATCH \times \left\lceil \frac{OH}{m} \right\rceil. \tag{15}$$

OH represents the row number of the output feature maps. Since data transfer and computing are overlapped, the overall latency equals the maximum of data transfer latency and computing latency.

Table 1. The Top-1 Accuracy of Various CNN Models with Different Winograd Tile Sizes and Pruning Methods

Model	Dataset	Method	Tile	Sparsity(%)	Accuracy/mAP(%)
VGG16	Cifar10	SRBS	4×4	80	92.56
		Winols Pruning	$4 \times 4/6 \times 6/8 \times 8$	80	**93.19**/92.61/92.84
VGG-NaGaDoMi	Cifar10	Winograd_ReLU	4×4	80	92.21
		Spatial-Winograd	6×6	80	93.34
		Winols Pruning	$4 \times 4/6 \times 6/8 \times 8$	80	92.76/92.57/**93.58**
ConvNet	Cifar100	Winograd_ReLU	4×4	60	68.55
		Spatial-Winograd	6×6	60	69.09
		Winols Pruning	$4 \times 4/6 \times 6/8 \times 8$	60	**69.16**/68.89/68.72
ResNet18	ImageNet	SRBS	4×4	70	66.14
		Winograd_ReLU	4×4	70	66.61
		Spatial-Winograd	6×6	70	69.54
		Winols Pruning	$4 \times 4/6 \times 6/8 \times 8$	70	69.42/69.25/**70.06**
ResNet34	ImageNet	SRBS	4×4	75	70.27
		Winols Pruning	$4 \times 4/6 \times 6/8 \times 8$	80	70.9/71.04/**71.57**
ConvNet	Cifar10	**Winols Pruning**	$4 \times 4/6 \times 6/8 \times 8$	80	90.11/**90.13**/89.94
VGG16	Cifar100	**Winols Pruning**	$4 \times 4/6 \times 6/8 \times 8$	70	**71.18**/70.83/71.09
VGG16	ImageNet	**Winols Pruning**	$4 \times 4/6 \times 6/8 \times 8$	70	72.56/72.36/**72.58**
AlexNet	ImageNet	**Winols Pruning**	$4 \times 4/6 \times 6/8 \times 8$	60	60.2/**60.23**/60.22
TinyYoloV3	VOC	**Winols Pruning**	$4 \times 4/6 \times 6/8 \times 8$	60	64.37/64.25/**64.5**

5 EVALUATION

To validate the effectiveness of Winols, we conduct a series of experiments on both pruning CNN models with Winols pruning and performing inference with our Winols FPGA-based accelerator. Throughout these experiments, we utilize various Winograd tile configurations to showcase the advantages of Winols. Moreover, we make a comprehensive comparison between Winols and state-of-the-art pruning algorithms, as well as sparse Winograd accelerators.

5.1 Experiment Setup

We validate a variety of representative CNN models for image classification and object detection. All the CNN models are trained and pruned with Pytorch v1.7.1 which integrates our pruning method. Our experiments are conducted on two distinct platforms: the Xilinx ZCU102 SoC and Virtex VCU118. The accelerators are programmed using C++ with the Xilinx Vitis **High-Level Synthesis (HLS)** (v2021.1) serving as the design and deployment toolchain. Our evaluation encompasses three primary objectives. Firstly, we assess the accuracy and obtained sparsity of the Winols pruning algorithm. Secondly, we evaluate the efficiency of Winols accelerator and compare it with other **state-of-the-art** (in short, **SOTA**) accelerators. Lastly, we provide valuable insights into how Winols accelerators outperform other peers.

5.2 Effectiveness of Winols Cross-Domain Pruning

5.2.1 Accuracy Analysis. We compare our cross-domain pruning method with the SOTA structured pruning method SRBS [47], and the unstructured pruning method Winograd-ReLU [28] and Spatial-Winograd [48] across various CNN models. The Top-1 accuracy results obtained by these methods on various CNN models and datasets are presented in Table 1. Our Winols surpasses all comparative objects.

Firstly, Winols pruning with the same tile size 4×4 consistently outperforms SRBS by 0.63%, 3.28%, and 0.63% in accuracy for VGG16 on Cifar10, Reset18 and ResNet34 on ImageNet, respectively. Winols pruning also exceeds Winograd-ReLU by 1.37%, 0.61%, and 3.45% in accuracy for VGG-NaGaDoMi on Cifar10, ConvNet on Cifar100, and ResNet18 on ImageNet, respectively. Especially, when applied to ResNet18 on ImageNet, Winols pruning achieves an impressive accuracy of up to 69.42%, while SRBS and Winograd-ReLU pruning attain accuracies of 66.14% and 66.61%, respectively. The Spatial-Winograd method demonstrates comparable accuracy results with a tile size of 6×6. However, this particular pruning algorithm brings irregular sparsity into the Winograd domain weights, thus causing difficulty in achieving hardware acceleration. Until now, there is no reported accelerator which can support the Spatial-Winograd pruning algorithm. Furthermore, we apply Winols pruning to the object detection model TinyYoloV3, and achieve an mAP of 64.37% with a sparsity of 60% which is never reported by the other pruning methods.

Secondly, other existing methods only facilitate tile sizes of either 4×4 or 6×6, but our Winols pruning can support Winograd tile size up to 8×8, thereby enabling the exploitation of enhanced computational efficiency offered by the Winograd algorithm. With the larger tile size of 8×8, our Winols cross-domain pruning can achieve comparable accuracy even compared with our tile size of 4×4 or 6×6. For instance, when applied to VGG-NaGaDoMi on Cifar10, ResNet18, and ResNet34 on ImageNet, Winols pruning with a tile size of 8×8 outperforms the accuracy achieved with tile sizes of 4×4 and 6×6. Overall, our cross-domain pruning algorithm exhibits the ability to accommodate diverse Winograd tile sizes. This feature contributes to enhancing the computational efficiency during accelerator design oriented to the Winograd algorithm.

5.3 Efficiency of Winols Accelerator System

We use different Winograd tile sizes and sparsity configurations to validate the performance of our Winols accelerators. We compare them with a Winograd CNN accelerator optimized for dense networks (called as Dense Accelerator) [31] and three SOTA sparse accelerators, namely SpWA [30], SRBS [47], and BISWSRBS [46], respectively. It should be noted that these sparse accelerators only support a Winograd tile size of 4×4. To ensure a fair comparison, we evaluate several performance metrics, including latency, DSP efficiency, and energy efficiency by performing various CNN models.

5.3.1 Resource Consumption and Frequency. The resource consumption and system configurations for different accelerators are shown in Table 2. Winols accelerator can achieve higher computation frequency to 200MHz. Winols accelerators comprise $n \times n$ PEs, each PE is responsible for a specific matrix-vector block with a particular sparsity level. With tile sizes of 4×4 and 6×6, DSPs are primarily used by the multiplications in PE modules. When using a tile size of 8×8, the Winograd domain transformation requires extra utilization of DSP resources, so the overall DSP cost is higher than the number of DSPs utilized by PEs. For sparse matrix-vector multiplication, Winols reduces the LUT resource consumption and achieves high operating frequencies because of the adoption of the RCI encoding and sampling in PE, which also reduces the irregular data access. It should be pointed out that SpWA and WSRBS employ irregular data access for sparse matrix multiplication, leading to higher LUT usage. These methods have limitations in terms of the supported number of PEs and their achievable frequency, typically restricted to 166 MHz. BISWSRBS adopts mixed precision quantization to reduce the computational complexity for bit-level operations and save DSPs but still results in a reduced operating frequency of 166 MHz.

5.3.2 Model Execution Latency. We first compare the overall latency of Winols accelerator with other accelerators on various CNN models. Winols accelerators are configured with the sparsity of 50% and 75% for Winograd weight tile, respectively. For the Dense Accelerator, SpWA, WSRBS,

Table 2. The Resource Consumption and Frequency of Different Accelerators

	Device	Freq. (MHz)	Power (W)	Tile $n \times n$	Sparsity	Poc	Pic	BRAM	DSP PE/Overall	FF	LUT
Dense Accelerator	ZC706	200	12.3	4×4	–	–	–	540	–/532	91874	89628
SpWA	ZC706	166	–	4×4	–	–	–	732	–/768	153020	155206
WSRBS	ZCU102	166	14.2	4×4	80%	–	–	736	–/525	107548	221376
BISWSRBS	ZCU102	166	11.6	4×4	75%	–	–	739	0/5	190266	133470
Winols	ZCU102	200	26.2	4×4	50%	16	16	989	2048/2098	74584	109892
Winols	ZCU102	200	26.2	4×4	75%	32	16	1041	2048/2098	95743	164069
Winols	ZCU102	200	26.3	6×6	50%	16	8	1446	2304/2346	89416	141970
Winols	ZCU102	200	26.2	6×6	75%	16	8	893	1152/1206	98947	153527
Winols	ZCU102	200	26.2	8×8	75%	8	8	1325	1024/1545	113609	165403

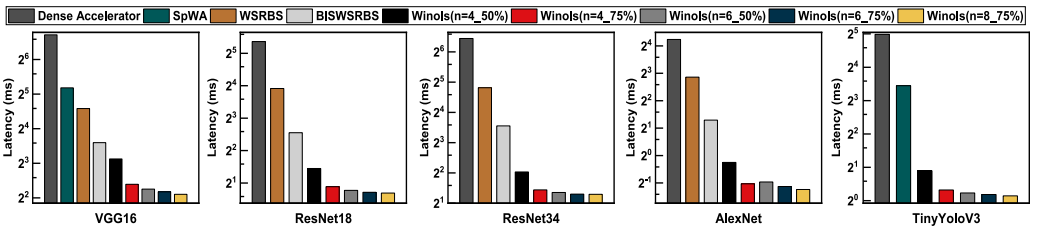
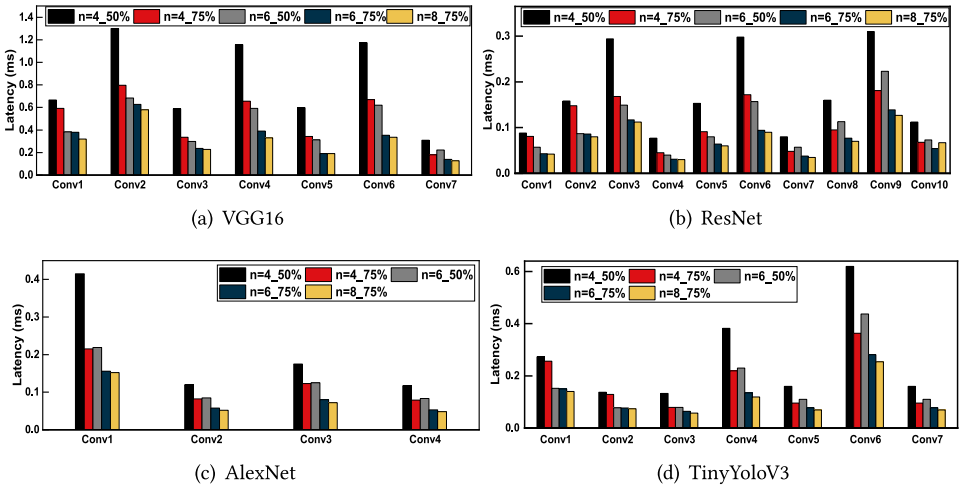


Fig. 9. Latency of total convolutional layers for different CNN models on different accelerators.

and BISWSRBS, the tile sizes are configured to 4×4 due to their limitations. VGG16, ResNet18, ResNet34, and AlexNet are deployed with sparsity of 80%, 76%, 75%, and 75%, respectively. The sparsity configuration of TinyYoloV3 is not provided in these accelerators.

The results of the total latency of convolution layers across various CNN models on the ZCU102 platform are as shown in Figure 9. With the adoption of larger tile sizes and increased sparsity, Winols exhibits a substantial reduction in inference latency. Winols accelerator with a tile size of 8×8 and sparsity of 75% provides the best performance, compared with other accelerators and our Winols accelerators with small tile sizes of 4×4 and 6×6. Specifically, the average inference speedup grows up to **31.7**×, **9.1**×, **10.9**×, and **4.3**×, when compared with the Dense Accelerator, SpWA, WSRBS, and BISWSRBS, respectively. With 6×6 tile size and 75% sparsity, Winols accelerator also surpasses these accelerators with an average inference speedup of **30.5**×, **8.8**×, **10.4**× and **4.1**×, respectively.

Even with the tile size 4×4, our Winols accelerators also show reduced latency compared with other accelerators under all evaluated models. This notable improvement benefits from our parallel and pipelined architecture, efficient utilization of DSP blocks, and sparsity-aware PEs. For the VGG16 model, Winols accelerator with a sparsity of 50% achieves speedup of up to 12× compared with the Dense Accelerator. When increasing the sparsity, Winols accelerator with 75% sparsity attains an impressive speedup of 20×. We take the sparsity configurations of 50% and 75% as an example to compare the performance of Winols on various models. For ResNet18, despite WSRBS adopting 76% sparsity, Winols with sparsity configurations of 50% and 75% still achieve impressive speedups of **5.5**× and **8.1**×, respectively. Moreover, with the same sparsity of 50% and 75%, Winols accelerators can also outperform BISWSRBS by 2.1× and 3.2×, respectively. For AlexNet, Winols outperforms all the other evaluated accelerators, and obtains 3.3× and 5.8× speedups, respectively, over the previous SOTA solution BISWSRBS. For TinyYoloV3, Winols accelerators outperform SpWA and achieve up to 5.9× and 8.7× performance improvements across different sparsity levels, respectively. Overall, our Winols accelerators exhibit substantial advancements in terms of latency reduction and outperform existing accelerators across various CNN models.



(a) VGG16 (b) ResNet

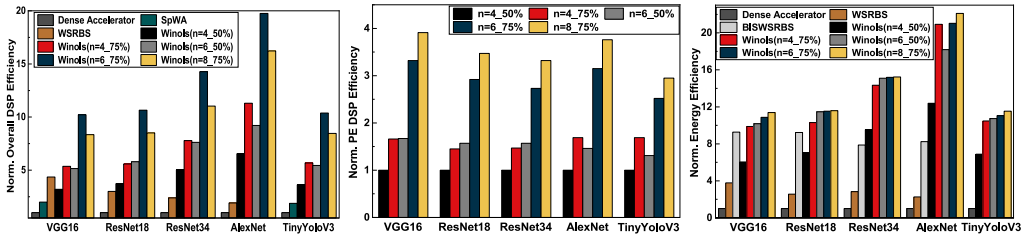
(c) AlexNet (d) TinyYoloV3

Fig. 10. Latency(ms) of total convolutional layers for different CNN models on Winols accelerators.

5.3.3 Layer-wise Latency Breakdown. To gain further insights into the performance improvements associated with larger Winograd tile sizes, we conduct a breakdown analysis. We evaluate the layer-wise inference speed of our accelerators utilizing different tile sizes, as illustrated in Figure 10.

To enhance clarity, the same layer configuration is only shown once for each CNN model. ResNet18 and ResNet34 share the same layer configuration, except for the number of layers. We take Winols accelerator with 4×4 tile size as baseline. When employing a sparsity of 50%, Winols accelerator with 6×6 tile size can achieve 1.82×,1.68×,1.53×, and 1.6× inference speedups on average across different models compared with 4×4 configuration, respectively. Similarly, when increasing the sparsity to 75%, Winols accelerator with 6×6 also obtains better performance, achieving a maximum 1.56× inference speedup compared with 4×4 configuration. Because Winols cross-domain pruning algorithm trims the weights along the channel dimension, Winols accelerators thus can increase channel parallelism when utilizing a higher sparsity. So the convolution layers which feature a large number of input and output channels can obtain better performance with higher sparsity even tile is small. For example, When adopting the sparsity 75%, Winols with 4×4 can obtain comparable performance to the 6×6 with 50% sparsity configuration. In addition, under the same tile size, higher sparsity can prompt better inference performance. For example, when employing the 4×4 tile size, Winols accelerator with sparsity of 75% can achieve 1.64×, 1.57×, 1.57×, and 1.51× inference speedups across different models compared with 50% sparsity, respectively.

When adopting a larger tiling size of 8×8, Winols accelerator yields optimal performance, with inference speedups of **1.7**× and **1.1**× compared with tile sizes of 4×4 and 6×6, respectively. It is worth mentioning that the Winols accelerator with 6×6 displays lower latency compared with 8×8 for the last layer of the ResNet model. In this layer, the input feature map is padded to a size of 9×9, which is extremely smaller than other layers such as 58×58. Winols accelerators with tile sizes of 8×8 and 6×6 both necessitate four input tiles to process the 9×9 feature map by using the round-up rule. The implementation of 6×6 in the Winols accelerator can be actually configured as higher parallelism in the channel dimension. As the above mentioned in Table 2, the channel parallelism $Poc \times Pic$ with tile size 6×6 and 75% sparsity is configured as 16×8, while the parallelism with tile size 8×8 is configured as 8×8, so Winols with tile size 6×6 which has higher parallelism in channel dimension can perform better than tile size 8×8 for this layer. For



(a) Normalized overall DSP efficiency (b) Normalized PE DSP efficiency (c) Normalized energy efficiency

Fig. 11. The normalized overall DSP efficiency (over dense Winograd accelerator), PE DSP efficiency (over Winols n=4_50% accelerator) and energy efficiency (over dense Winograd accelerator) of our accelerators and other Winograd accelerators.

other layers which have larger feature map sizes, Winols accelerator with 8×8 proves to be more suitable than 6×6 due to higher parallelism in the feature map dimension. Overall, both the larger tile size and higher sparsity contribute to improving the performance of the model.

5.3.4 Accelerator Efficiency Evaluation. Winols, leveraging its sparse-aware architecture and large Winograd tile size, achieves superior improvements in inference latency compared with SOTA Winograd accelerators. To evaluate the efficiency of Winols accelerators independent of the hardware platform, we compare their inference performances per DSP and per Watt. To show the effectiveness of the sparsity-aware PE design, we evaluate the efficiency of the DSPs occupied by the PE and the overall accelerator. The normalized overall DSP efficiency towards Dense Accelerator, PE DSP efficiency towards Winols accelerator with tile size 4×4, and energy efficiency of overall accelerator systems are shown in Figure 11.

Among various Winograd accelerators, Winols with a tile size of 6×6 and sparsity of 75% outperform other accelerators in terms of DSP efficiency, as shown in Figure 11(a). With tile size 6×6 and 75% sparsity, Winols achieves **13.1**× DSP efficiency improvement over the Dense Accelerator, **5.3**× DSP efficiency improvement over SpWA, and **5.6**× DSP efficiency improvement over WSRBS, respectively. This performance enhancement can be attributed to the dedicated architecture design and the utilization of a larger Winograd tile size. Specifically, for VGG16, Winols with tile size 6×6 achieves 2.36× DSP efficiency improvement over WSRBS with tile 4×4. Because of our sparsity-aware Winols architecture, the performance improvement is higher than the 1.78× multiplication reduction by larger tile as afore introduced in Figure 1(b). With a sparsity of 75%, Winols accelerator with a tile size of 8×8 outperforms both tile size 4×4 and other accelerators. Compared with Dense Accelerator, SpWA, and WSRBS, Winols achieves 10.5×, 4.3×, and 4.5× DSP efficiency improvement, respectively. However, Winols accelerator with tile size 8×8 exhibits lower DSP efficiency than 6×6. The reason is that the Winograd domain transformation with tile size 8×8 requires substantial demands for DSP resources. When focusing on the DSP efficiency of the PEs, Winols accelerator with tile size 8×8 achieves the best DSP efficiency, as illustrated in Figure 11(b).

The inference performance per Watt as energy efficiency is normalized with respect to the Dense Accelerator, as shown in Figure 11(c). When employing a tile size of 8×8 and a sparsity of 75%, Winols achieves an average energy efficiency improvement of **14.3**× over the Dense Accelerator. Additionally, Winols showcases a **5.7**× improvement over WSRBS and a **1.8**× improvement over BISWSRBS. With a tile size of 6×6 and sparsity of 75%, Winols achieves **13.9**×, **5.5**×, and **1.7**× energy efficiency improvement compared with Dense Accelerator, WSRBS, and BISWSRBS, respectively. Furthermore, it is worth noting that Winols with 4×4 and 50% sparsity can outperform all

Table 3. The Resource Consumption and Frequency of Winols Accelerators on VCU118 Platform

	Freq. (MHz)	Power (W)	Tile $n \times n$	Sparsity	Poc	Pic	BRAM	DSP PE/Overall	FF	LUT
Winols	200	26.5	6×6	75%	**16**	**16**	1432	2304/2361	153325	217657
Winols	200	26.5	8×8	75%	**16**	**8**	1657	2048/2749	154140	237690

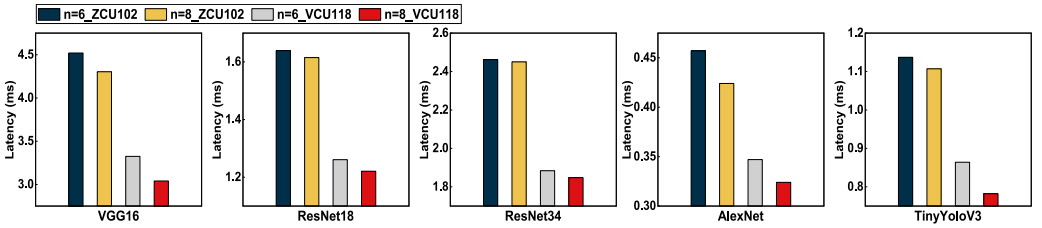


Fig. 12. Latency(ms) of total convolutional layers for different CNN models on ZCU102 and VCU118 paltforms.

the other accelerators except BISWSRBS on VGG16 and ResNet18. For these two models, although BISWSRBS employs higher sparsity of 80% and 75%, it still suffers from accuracy degradation.

5.3.5 Accelerator Scalability Evaluation. To validate the scalability of Winols accelerators, we set the $Poc \times Pic$ configuration for Winols accelerators with tile size of 6×6 as 16×16, and for the tile size 8×8 as 16×8, respectively. We implement our scaled system on the VCU118 platform since the resource of ZCU102 is no longer feasible for these settings. The resource consumption and frequency of Winols accelerators on the VCU118 platform are presented in Table 3. With larger tile sizes and $Poc \times Pic$ values, Winols accelerator processes larger matrix-vector blocks in parallel and occupies more DSP and LUTs. Specifically, with a tile size of 6×6 and 75% sparsity, we can scale the block size of Winols accelerator to 16×16 on the VCU118 platform. In contrast, due to limited LUT resources, only a 16×8 block can be supported on the ZCU102 platform.

We also compare the overall latency of different models with different tile sizes on the ZCU102 and VCU118 platforms. As shown in Figure 12, Winols with a tile size of 8×8 exhibits superior performance compared with 6×6 on both the ZCU102 and VCU118 platforms. The Winograd algorithm with tile size 8×8 offers a theoretical multiplication reduction of 1.265× when compared with tile size 6×6. In our on-board evaluation, Winols with 8×8 provides up to 1.08× and 1.1× latency reduction than 6×6 on ZCU102 and VCU118, respectively. Considering different hardware platforms, Winols can further reduce the inference latency on the VCU118 platform because of the increased parallelism. When compared with the ZCU102 platform, the Winols accelerators on the VCU118 platform achieve up to **1.36**× and **1.42**× inference speedup for the tile size of 6×6 and 8×8, respectively. Winols accelerators on the VCU118 platform have a higher utilization of DSP resources, which is close to 2× utilization on the ZCU102 platform. However, both platforms demonstrate similar DDR4 bandwidth performance. Due to limited DDR bandwidth, Winols accelerator cannot achieve a 2× inference improvement, resulting in reduced DSP efficiency on the VCU118 platform. Despite the higher utilization of hardware resources, Winols accelerators on the VCU118 platform exhibit comparable power consumption with the accelerators on the ZCU102 platform. Due to the better inference speed, Winols accelerators deployed on the VCU118 platform offer higher energy efficiency.

5.3.6 Comparison with CPU and GPU Platforms. We further compare the inference performance of Winols accelerators with CPU and GPU-based solutions. We evaluate the twenty-time

Table 4. Comparisons with CPU and GPU Platforms

Implementation	oneDNN	cuDNN	**Winols**	**Winols**
Device	Xeon 4210	RTX3090	ZCU102	VCU118
Technology	14nm	8nm	16nm	16nm
Average performance (TOPS)	1.03	28.78	6.04	7.81
Power (W)	85	277	26.2	26.5
Energy efficiency (GOPS/W)	12	104	**231**	**295**

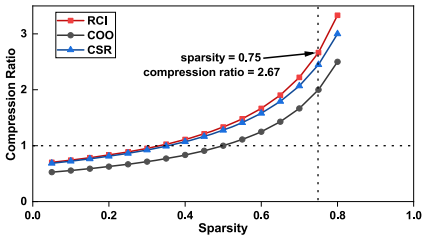


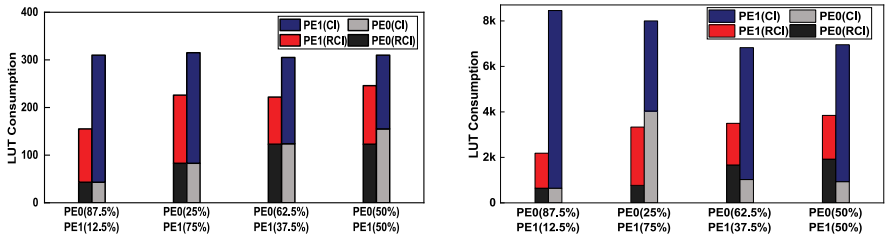
Fig. 13. Compression ratio of RCI encoding strategy.

average performance of VGG16 model with a batch size of 64 on Intel Xeon Silver 4210 CPU and NVIDIA RTX 3090 GPU platforms. The reported performance is the effective performance, which is calculated by dividing the total operations by the total processing time. The power data of Winols accelerators are collected with a bump-in-wire power meter during the on-board evaluation. For the GPU platform, the runtime power data is obtained using the NVIDIA system management interface ("nvidia-smi" version 510.54). To ensure a fair comparison, we collect performance data using the kernel level library provided by the latest oneDNN 2023.1.0 for CPU and cuDNN 8.6 for GPU, both of which include the Winograd algorithm implementation for convolution operations. As illustrated in Table 4, the RTX 3090 GPU provides the leading performance, but our implementation on the Xilinx VCU118 FPGA achieves higher energy efficiency, with a factor of **2.84**× compared with GPU implementation. Furthermore, when compared with the CPU platform, Winols accelerator on the VCU118 platform achieves an impressive energy efficiency improvement of **24.6**×.

5.3.7 Efficiency of Sparse Weight Encoding. We validate the efficacy of our sparse weight encoding method. For storage requirements, our RCI encoding method can achieve efficient compression of weights. For computation requirements, RCI can visibly reduce the LUT resource consumption.

We first compare the compression ratio of the sparse weight encoding strategy RCI with other encoding schemes, **coordinate format (COO)** and **compressed sparse row (CSR)** [33], as shown in Figure 13. To index the number of columns, the data width of value in the index matrix requires only a few bits. In our implementation, only 4 bits are enough to represent the offset of *Pic* columns. This prompts efficient compression of weights when the Winograd sparsity exceeds 33.3%. In particular, a compression ratio of 2.67× can be achieved when the sparsity is 75%. Due to the block-based pruning strategy employed, the row information in RCI can be disregarded. Therefore, RCI can achieve a higher compression ratio than COO and CSR.

We then evaluate the saving of LUT resources for vector-vector multiplication and matrix-vector multiplication with our RCI encoding, which are characterized by varying levels of computation complexity. The matrix-vector multiplication is the fundamental computational task within our PE array. Considering the sparsity re-distribution strategy, we divide the evaluation into four groups



(a) Sparse Vector-Vector Multiplication (b) Sparse Matrix-Vector Multiplication

Fig. 14. LUT consumption of different encoding methods.

with distinct sparse patterns. Each group consists of two PEs with different sparsity that sum up to 50%. The LUT consumptions of our RCI method and the direct column index (CI) method are shown in Figure 14. Our RCI encoding method combined with the corresponding PE architecture design can efficiently reduce LUT resources compared to the direct column index method for vector-vector multiplication. For large-scale computational tasks such as matrix-vector multiplication, the RCI method yields an average LUT resource savings of approximately 60%. Moreover, by employing the sparsity re-distribution strategy for different PEs, the RCI method can further reduce LUT resource consumption. In our Winograd accelerator, $n \times n$ PEs are constructed to execute matrix-vector multiplications in parallel. Winols accelerator by exploiting the RCI encoding method thus can save a considerable amount of LUT resources.

6 CONCLUSION

In this paper, we present a novel large-tiling sparse Winograd accelerator named Winols to boost the inference efficiency of CNN models on FPGAs. We propose a relevance-based cross-domain pruning method that maintains the sparsity for Winograd transformation, and design a sparsity-aware hardware architecture with a sparse encoding scheme. The cross-domain pruning method and Winols accelerators with our resource and parallelism optimizations can support larger input tile 8×8 to release the advantage of the Winograd algorithm. Extensive experiments demonstrate that our cross-domain pruning method shows competitive accuracy to SOTA methods. Our Winols accelerators with tile size 6×6 and 8×8 outperform the SOTA CNN accelerators with an average **10.4**× and **10.9**× inference speedup, **5.6**× and **4.5**× improvement in DSP efficiency, and **5.5**× and **5.7**× in energy efficiency, respectively. When compared to the CPU and GPU platforms, Winols accelerator with tile size 8×8 achieves **24.6**× and **2.84**× energy efficiency improvements, respectively.

REFERENCES

[1] Pranav Adarsh, Pratibha Rathi, and Manoj Kumar. 2020. YOLO v3-tiny: Object detection and recognition using one stage improved model. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*. 687–694.

[2] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. *Pearson Correlation Coefficient*. Springer Berlin, Berlin, 1–4. https://doi.org/10.1007/978-3-642-00296-0_5

[3] Xing Chen, Jianshan Zhang, Bing Lin, Zheyi Chen, Katinka Wolter, and Geyong Min. 2022. Energy-efficient offloading for DNN-based smart IoT systems in cloud-edge environments. *IEEE Transactions on Parallel and Distributed Systems* 33, 3 (2022), 683–697. https://doi.org/10.1109/TPDS.2021.3100298

[4] Yao Chen, Cole Hawkins, Kaiqi Zhang, Zheng Zhang, and Cong Hao. 2021. 3U-EdgeAI: Ultra-low memory training, ultra-low bitwidth quantization, and ultra-low latency acceleration. In *Proceedings of the 2021 on Great Lakes Symposium on VLSI* (Virtual Event, USA) *(GLSVLSI '21)*. Association for Computing Machinery, New York, NY, USA, 157–162.

[5] Yao Chen, Jiong He, Xiaofan Zhang, Cong Hao, and Deming Chen. 2019. Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19)*. Association for Computing Machinery, New York, NY, USA, 73–82.

[6] Yao Chen, Kai Zhang, Cheng Gong, Cong Hao, Xiaofan Zhang, Tao Li, and Deming Chen. 2019. T-DLA: An open-source deep learning accelerator for ternarized DNN models on embedded FPGA. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 13–18. https://doi.org/10.1109/ISVLSI.2019.00012

[7] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138. https://doi.org/10.1109/JSSC.2016.2616357

[8] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient primitives for deep learning. *CoRR* abs/1410.0759 (2014). arXiv:1410.0759

[9] Ziaul Choudhury, Shashwat Shrivastava, Lavanya Ramapantulu, and Suresh Purini. 2022. An FPGA overlay for CNN inference with fine-grained flexible parallelism. *ACM Trans. Archit. Code Optim.* 19, 3, Article 34 (May 2022), 26 pages.

[10] Jason Cong, Peng Wei, Cody Hao Yu, and Peipei Zhou. 2017. Bandwidth optimization through on-chip memory restructuring for HLS. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/3061639.3062208

[11] Prakhar Ganesh, Yao Chen, Yin Yang, Deming Chen, and Marianne Winslett. 2022. YOLO-ReT: Towards high accuracy real-time object detection on edge GPUs. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE Computer Society, Los Alamitos, CA, USA, 1311–1321.

[12] Ashish Gondimalla, Noah Chesnut, Mithuna Thottethodi, and T. N. Vijaykumar. 2019. SparTen: A sparse tensor accelerator for convolutional neural networks. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12–16, 2019*. ACM, 151–165.

[13] Cheng Gong, Ye Lu, Kunpeng Xie, Zongming Jin, Tao Li, and Yanzhi Wang. 2022. Elastic significant bit quantization and acceleration for deep neural networks. *IEEE Transactions on Parallel and Distributed Systems* 33, 11 (2022), 3178–3193. https://doi.org/10.1109/TPDS.2021.3129615

[14] Lei Gong, Chao Wang, Xi Li, and Xuehai Zhou. 2021. Improving HW/SW adaptability for accelerating CNNs on FPGAs through a dynamic/static co-reconfiguration approach. *IEEE Transactions on Parallel and Distributed Systems* 32, 7 (2021), 1854–1865. https://doi.org/10.1109/TPDS.2020.3046762

[15] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016*. IEEE Computer Society, 770–778.

[17] Di Huang, Xishan Zhang, Rui Zhang, Tian Zhi, Deyuan He, Jiaming Guo, Chang Liu, Qi Guo, Zidong Du, Shaoli Liu, Tianshi Chen, and Yunji Chen. 2020. DWM: A decomposable Winograd method for convolution acceleration. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7–12, 2020*. AAAI Press, 4174–4181.

[18] Suyeon Hur, Seongmin Na, Dongup Kwon, Joonsung Kim, Andrew Boutros, Eriko Nurvitadhi, and Jangwoo Kim. 2023. A fast and flexible FPGA-based accelerator for natural language processing neural networks. *ACM Trans. Archit. Code Optim.* 20, 1 (2023), 11:1–11:24. https://doi.org/10.1145/3564606

[19] S. Kala, Babita R. Jose, Jimson Mathew, and S. Nalesh. 2019. High-performance CNN accelerator on FPGA using unified Winograd-GEMM architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 12 (2019), 2816–2828. https://doi.org/10.1109/TVLSI.2019.2941250

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger (Eds.). 1106–1114.

[21] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham M. Kakade, and Ali Farhadi. 2020. Soft threshold weight reparameterization for learnable sparsity. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 5544–5555.

[22] Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[23] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient ConvNets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings*. OpenReview.net.

[24] Shiyu Li, Edward Hanson, Hai Li, and Yiran Chen. 2020. PENNI: Pruned kernel sharing for efficient CNN inference. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 5863–5873.

[25] Shiyu Li, Edward Hanson, Xuehai Qian, Hai (Helen) Li, and Yiran Chen. 2021. ESCALATE: Boosting the efficiency of sparse CNN accelerator with kernel decomposition. In *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18–22, 2021*. ACM, 992–1004.

[26] Sheng R. Li, Jongsoo Park, and Ping Tak Peter Tang. 2017. Enabling sparse Winograd convolution by native pruning. *CoRR* abs/1702.08597 (2017). arXiv:1702.08597

[27] Xinheng Liu, Yao Chen, Cong Hao, Ashutosh Dhar, and Deming Chen. 2021. WinoCNN: Kernel sharing Winograd systolic array for efficient convolutional neural network acceleration on FPGAs. In *32nd IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2021, Virtual Conference, USA, July 7–9, 2021*. IEEE, 258–265.

[28] Xingyu Liu and Yatish Turakhia. 2016. Pruning of Winograd and FFT based convolution algorithm. In *Proc. Convolutional Neural Netw. Vis. Recognit.* 1–7.

[29] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22–29, 2017*. IEEE Computer Society, 2755–2763.

[30] Liqiang Lu and Yun Liang. 2018. SpWA: An efficient sparse Winograd convolutional neural networks accelerator on FPGAs. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6.

[31] Liqiang Lu, Yun Liang, Qingcheng Xiao, and Shengen Yan. 2017. Evaluating fast algorithms for convolutional neural networks on FPGAs. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 101–108. https://doi.org/10.1109/FCCM.2017.64

[32] Xiaolong Ma, Sheng Lin, Shaokai Ye, Zhezhi He, Linfeng Zhang, Geng Yuan, Sia Huat Tan, Zhengang Li, Deliang Fan, Xuehai Qian, Xue Lin, Kaisheng Ma, and Yanzhi Wang. 2021. Non-structured DNN weight pruning–is it beneficial in any platform? *IEEE Transactions on Neural Networks and Learning Systems* (2021), 1–15.

[33] Michele Martone, Salvatore Filippone, Salvatore Tucci, Paweł Gepner, and Marcin Paprzycki. 2010. Use of hybrid recursive CSR/COO data structures in sparse matrix-vector multiplication. In *Proceedings of the International Multiconference on Computer Science and Information Technology*. 327–335. https://doi.org/10.1109/IMCSIT.2010.5680039

[34] Huiyu Mo, Leibo Liu, Wenjing Hu, Wenping Zhu, Qiang Li, Ang Li, Shouyi Yin, Jian Chen, Xiaowei Jiang, and Shaojun Wei. 2020. TFE: Energy-efficient transferred filter-based engine to compress and accelerate convolutional neural networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 751–765.

[35] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 907–922.

[36] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel S. Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24–28, 2017*. ACM, 27–40.

[37] Jongsoo Park, Sheng Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. 2016. Faster CNNs with direct sparse convolutions and guided pruning. *arXiv preprint arXiv:1608.01409* (2016).

[38] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

[39] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. 2014. *Intel Math Kernel Library*. 167–188. https://doi.org/10.1007/978-3-319-06486-4_7

[40] Haonan Wang, Wenjian Liu, Tianyi Xu, Jun Lin, and Zhongfeng Wang. 2019. A low-latency sparse-Winograd accelerator for convolutional neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019*. IEEE, 1448–1452. https://doi.org/10.1109/ICASSP.2019.8683512

[41] Yang Wang, Chen Zhang, Zhiqiang Xie, Cong Guo, Yunxin Liu, and Jingwen Leng. 2021. Dual-side sparse tensor core. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1083–1095. https://doi.org/10.1109/ISCA52012.2021.00088

[42] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference, DAC 2017, Austin, TX, USA, June 18–22, 2017*. ACM, 29:1–29:6.

[43] Di Wu, Xitian Fan, Wei Cao, and Lingli Wang. 2021. SWM: A high-performance sparse-Winograd matrix multiplication CNN accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 5 (2021), 936–949.

[44] Xinfeng Xie, Zheng Liang, Peng Gu, Abanti Basak, Lei Deng, Ling Liang, Xing Hu, and Yuan Xie. 2021. SpaceA: Sparse matrix vector multiplication on processing-in-memory accelerator. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 570–583.

[45] Weizhi Xu, Yintai Sun, Shengyu Fan, Hui Yu, and Xin Fu. 2023. Accelerating convolutional neural network by exploiting sparsity on GPUs. *ACM Trans. Archit. Code Optim.* 20, 3 (2023), 36:1–36:26. https://doi.org/10.1145/3600092

[46] Tao Yang, Zhezhi He, Tengchuan Kou, Qingzheng Li, Qi Han, Haibao Yu, Fangxin Liu, Yun Liang, and Li Jiang. 2021. BISWSRBS: A Winograd-based CNN accelerator with a fine-grained regular sparsity pattern and mixed precision quantization. *ACM Trans. Reconfigurable Technol. Syst.* 14, 4 (2021), 18:1–18:28.

[47] Tao Yang, Yunkun Liao, Jianping Shi, Yun Liang, Naifeng Jing, and Li Jiang. 2020. A Winograd-based CNN accelerator with a fine-grained regular sparsity pattern. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. 254–261.

[48] Jiecao Yu, Jongsoo Park, and Maxim Naumov. 2019. Spatial-Winograd pruning enabling sparse Winograd convolution. *CoRR* abs/1901.02132 (2019). arXiv:1901.02132

[49] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. 2019. Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 38, 11 (2019), 2072–2085.

[50] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15–19, 2016*. IEEE Computer Society, 20:1–20:12.

[51] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A systematic DNN weight pruning framework using alternating direction method of multipliers. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8–14, 2018, Proceedings, Part VIII (Lecture Notes in Computer Science, Vol. 11212)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer, 191–207.

[52] Jieru Zhao, Liang Feng, Sharad Sinha, Wei Zhang, Yun Liang, and Bingsheng He. 2020. Performance modeling and directives optimization for high-level synthesis on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 7 (2020), 1428–1441. https://doi.org/10.1109/TCAD.2019.2912916

[53] Xuda Zhou, Zidong Du, Qi Guo, Shaoli Liu, Chengsi Liu, Chao Wang, Xuehai Zhou, Ling Li, Tianshi Chen, and Yunji Chen. 2018. Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach. In *51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2018, Fukuoka, Japan, October 20–24, 2018*. IEEE Computer Society, 15–28.

[54] Chaoyang Zhu, Kejie Huang, Shuyuan Yang, Ziqi Zhu, Hejia Zhang, and Haibin Shen. 2020. An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 9 (2020), 1953–1965. https://doi.org/10.1109/TVLSI.2020.3002779

Received 13 September 2023; revised 4 December 2023; accepted 18 January 2024