A LOW-LATENCY SPARSE-WINOGRAD ACCELERATOR FOR CONVOLUTIONAL NEURAL NETWORKS

Haonan Wang, Wenjian Liu, Tianyi Xu, Jun Lin, and Zhongfeng Wang

School of Electronic Science and Engineering, Nanjing University, P.R. China

ABSTRACT

Low-latency and low-power implementations of Convolutional Neural Network (CNN) are highly desired for budget-restricted scenarios. Pruning and Winograd algorithm are two representative approaches to reduce the computation complexity of CNNs. Coupling them is very attractive, but the Winograd transformation removes data sparsity brought by pruning. In this paper, we present a low-latency sparse-Winograd CNN accelerator (LSW-CNN) for pruned Winograd CNN models. The ReLU-modified algorithm is employed to solve the zero refilling issue. Our design fully leverages the sparsity in both weights and activations, and thus eliminates all unnecessary computation and cycles. Moreover, a novel fast mask indexing algorithm for sparse data compression is developed. Accumulation buffers are scaled to reduce the latency brought by irregular serial channel merging. On VGG-16, experimental results demonstrate that the latency of LSW-CNN is reduced by 5.1 and 1.7 times, respectively, compared with state-of-the-art dense-Winograd and sparse-Winograd accelerators. Besides, the consumed hardware resource is also significantly reduced.

***Index Terms*—** Convolutional Neural Network, accelerator, sparse, Winograd, low-latency

1. INTRODUCTION

Convolutional Neural Network (CNN) has shown great performance in numerous challenging tasks, such as document recognition [1] and image classification [2]. However, when state-of-the-art CNNs achieve significant accuracy improvement [3], their sharply increased requirement for computing resource makes the implementations more expensive and slow on general-purpose processors [4, 5]. Hence, many works of CNN accelerators [6–8] have been proposed based on the Field Programmable Gate Arrays (FPGA) or Application Specific Integrated Circuit (ASIC) to achieve lower latency, less energy consumption, and higher flexibility. These platforms are suitable for the deployment of large scale CNNs on budget-restrict scenario such as edge computing.

On the other hand, many algorithms are proposed to decrease the computation complexity of CNN models. Domain transformation algorithms are employed to reduce the computation complexity of convolutions, such as Fast Fourier Transform [9], Winograd Algorithm (WinoA) [10], and fast FIR algorithm [11]. For instance, WinoA can reduce the workload of multiplication by 2.3× [12]. In addition, Han *et al.* [13] shows that most connections with insignificant weights are redundant, and thus up to 95% computations can be eliminated without accuracy loss. Then native pruned Winograd CNN [14] directly trains and prunes weights in Winograd domain, leading to 10× compression rate on Winograd parameters with negligible accuracy loss.

Several CNN accelerators were proposed based on the above algorithms. However, these designs also have drawbacks. SCNN [15] improves performance by leveraging all non-zero values in both activations and weights, but it cannot take the advantage of arithmetic transformation. Moreover, the sparse data will result in workload imbalance. In [12] and [16], architectures for Winograd CNN (Wino-CNN) on FPGA were proposed. [17] further presented a design optimized for native pruned Winograd CNN, while it did not solve the refilling issue of activations. Besides, its indexing scheme brings excessive implementation overhead, leading to higher hardware consumption than that of Wino-CNN.

In this paper, we propose a low-latency sparse-Winograd CNN accelerator (LSW-CNN), in which the ReLU-modified algorithm [18] is employed to tackle the zeros refilling issue. Our design fully exploits sparsity of both weights and activations, and thus eliminates all unnecessary computations. A fast mask indexing algorithm is developed to efficiently access sparse data, since it achieves the optimal memory footprint and indexing speed. Moreover, we use scaled accumulation buffers followed by adder trees to reduce the latency brought by irregular serial channel merging. On VGG-16 [19], the LSW-CNN achieves 5.1× and 1.7× overall speedup than state-of-the-art Wino-CNN and SpWA separately. Besides, it also achieves 2.0×, 1.4×, 1.7× and 1.6× saving on DSP, BRAM, LUT and Flip Flop, respectively, compared with SpWA.

2. WINOGRAD ALGORITHM AND SPARSITY

Naive WinoA. WinoA [20] is a computation complexity reduction approach for short convolutions in the signal processing field. Lavin *et al.* [10] first introduces it to CNNs to re-

duce the workload. The 2D WinoA is performed on $n \times n$ tiles (denoted by d), which are fetched from $H \times W$ input feature maps with overlaps of $(r-1) \times n$, and the size of convolution kernels (denoted by g) is $r \times r$. After the Winograd transformation as shown in Eq. 1, each input tile yields an $m \times m$ output tile (denoted by Y), where $n = m + r - 1$.

$$Y = A^T(GgG^T) \odot (B^T dB)A. \quad (1)$$

A, B, and G denotes post-process, tile-transform, and weight-transform matrices, respectively. It is noted that $\odot$ represents element-wise matrix multiplication (EWMM). So the WinoA can reduce the number of convolutional multiplications from m^2r^2 to n^2 at the cost of addition increasing. In this work, we set $r = 3$ and $m = 2$, which means an input activation needs to be split into 4×4 tiles with overlapping step of 2.

Naive sparse WinoA. As shown in Eq. 2, if the WinoA is naively applied to the spare neural network [13], the pruned g and ReLU-ed d with high sparsity are refilled in because of the affine transformation $G(\cdot)G^T$ and $B^T(\cdot)B$.

$$Y = A^T(G\text{Prune}(g)G^T) \odot (B^T\text{ReLU}(d)B)A. \quad (2)$$

Native Pruned WinoA. As shown in Eq. 3, Li *et al.* [14] proposed a native pruned Winograd model, in which the kernel weights are directly trained and pruned in the Winograd domain without weight transformation,

$$Y = A^T(\text{Prune}(\text{Train}(g'))) \odot (B^T\text{ReLU}(d)B)A, \quad (3)$$

where g' is the transformed matrix to be learned. This method can effectively introduce desired sparsity to weights in Winograd domain. However, the sparsity in activations brought by the inter-layer ReLU function still vanishes due to the domain transformation $B^T(\cdot)B$.

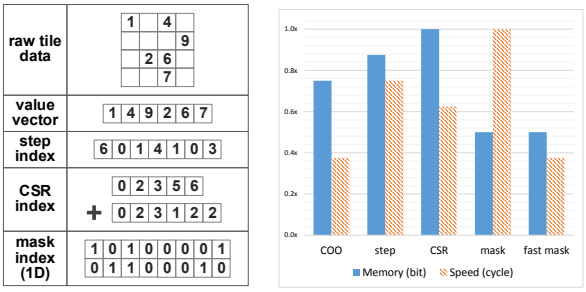
ReLU-modified WinoA. To further explore the sparsity in both sides, we employ a ReLU-modified approach as shown in Eq. 4, in which the ReLU is performed after the tile transformation in Winograd domain. It achieves significant workload reduction without accuracy loss.

$$Y = A^T(\text{Prune}(\text{Train}(g'))) \odot (\text{ReLU}(B^T dB))A. \quad (4)$$

3. SPARSE-WINOGRAD ARCHITECTURE DESIGN

According to the above discussion, the sparse-Winograd CNN models are very suitable for energy-efficient CNN implementations in budget-restricted scenarios. However, a sparse model will destroy the structured dataflow and result in performance degradation when implemented on the CNN accelerators that are specifically designed for high-throughput dense models.

To tackle this problem, we propose an architecture which is dedicatedly optimized for sparse-Winograd models, by which CNNs can be processed with low latency and high hardware efficiency. Besides, the proposed design can also implement dense models while leveraging the advantage of WinoA, at the cost of little overhead on the indexing scheme.



(a) Different Index Schemes. (b) Performance Comparsion.

Fig. 1. Illustration on a 4×4 tile with 37.5% density.

3.1. Index schemes and Fast mask indexing

In order to exploit sparsity in both activations and weights and reduce memory access, an indexing module is required to indicate the position of non-zeros. It is noted that this module will be used frequently through the dataflow. To achieve high indexing speed and low memory footprint, we propose a fast mask indexing (FMI) algorithm. As shown in Fig. 1(a), a sparse 4×4 tile using typical indexing schemes is illustrated.

Coordinate (COO). COO stores a list of data coordinates consisting of indices of row and column.

Step Index. In SCNN, to access all data index, the step indices need to be accumulated serially.

Compressed Sparse Row (CSR). SpWA exploits a variant of CSR [21], which requires more memory footprint.

Mask Index. This method marks non-zero values using 1-s. Each element only needs 1 bit. As shown in Fig. 1(b), it can save up to 50% of indexing memory overhead. On the other hand, the fixed data structure is hardware-friendly to achieve high-bandwidth memory access. Moreover, mask index can be easily scaled as the tile size increases, while the memory requirement of other schemes will grow sharply.

Fast Mask Indexing. However, mask index abandons the one-to-one correspondence with the value vector, so cycles of its length are required to sequentially traverse all bits to fetch the index and corresponding value. In order to accelerate the value access speed, we propose an FMI algorithm as shown in Algorithm 1. A mask index vector can be viewed as a 16-bit binary variable denoted by a. Note that bitwise AND operation $(a\&(a-1))$ can erase the first 1 from right, so we apply a low-priority encoder to the input of bit-reversed $\tilde{a}$. Hence, in the i-th cycle, the FMI outputs the coordinate corresponding to the i-th 1 of a from left. As shown in Fig. 1(b), not only does it require the minimal memory footprint with 74% overhead saving than that of [17], but also it achieves the fastest speed, since only cycles of the number of non-zero values are required. The architecture of FMI is illustrated in 3(a).

Algorithm 1: Fast Mask Indexing Algorithm

Input: 16-bit mask index a, value vector $v[\cdot]$;
Output: 1D coordinate b, $value$, non_zero;

do $\tilde{a}$ = bit_reverse(a);
do non_zero = fast_bit_sum(a);
for $i = 1$ to non_zero **and** $\tilde{a} \neq 0$ **do**
    $value = v[i]$;
    $b =$ low_priority_encoder($\tilde{a}$);
    $\tilde{a} = \tilde{a}$ & $(\tilde{a} - 1)$;
end for

3.2. Architecture Design

Based on the FMI scheme, we propose an LSW-CNN architecture that can implement both dense and sparse-Winograd CNN models as shown in Fig. 2. The dataflow mainly contains four processes, which are tiling, transformation, EWMM, and post-processing with stitching. Our architecture is dedicatedly optimized for ReLU-modified model to efficiently exploit the sparsity in both weights and activations.

Memory Hierarchy. Weights of all layers initially stored in DRAM are in the form of compressed structure, because they are already trained with sparsity and can be compressed at the same time. Because the energy of direct memory access from DRAM is considerable [22], the FMI compressed data structure can significantly reduce associated energy consumption. Buffers are used between modules to form the inter-module pipeline, which can balance the latency of each part of dataflow and thus improve the overall performance.

Tiling and Transformation. For the simplicity of dataflow, we first apply padding with size of 2 to input activations instead of 1, so the outside halo of output activations should be removed. Every input pixel is broadcast to 4 corresponding tiles, thus each tile buffer consists of 4 banks to have the write operations finished in one cycle. Next, each cycle transform units performs pre-transformation and ReLU on 4×4 tiles as shown in Eq. 4, and indexes them to the compressed structure in a parallelism of 4, since each trans buffer is also composed of 4 banks. This intra-channel parallelism is exploited to accelerate data transmission from tiling to transformation modules.

EWMM. In the EWMM module, the weight broadcast strategy is leveraged to avoid repeated memory access from the DRAM, so every weight tile is fetched to local buffers of PEs only once and reused to perform EWMM operations with different activation tiles. As shown in Fig. 3(b), by comparing the indices of weights and activations and finding the position in which both values are non-zero, the EWMM module eliminates all unnecessary multiplications, thus significantly reduces the computing resources. However, it should be noted the nearly random distribution of sparse values may result in workload imbalance between PEs with synchronous tile access, because the fastest PE must wait for the slowest one to access the next tile of data. Hence, controllers are allocated

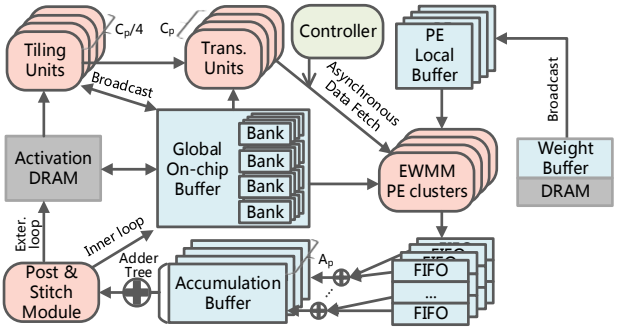
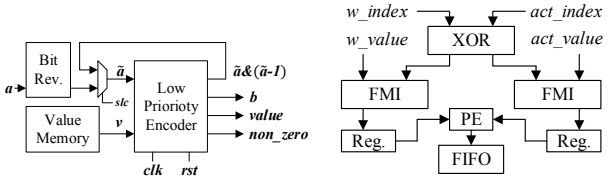


Fig. 2. Overview of the LSW-CNN architecture.



(a) Architecture of FMI module. (b) A quarter of EWMM module.

Fig. 3. Hardware implementations of two modules.

to each PE to read data asynchronously once the multiplication of two tiles is finished, while it avoids the memory access conflict that the intra-channel buffer are split to banks corresponding to PEs. As the number of multiplications serially processed by one PE increases, the imbalance problem will be rapidly alleviated.

Accumulation. In order to achieve low latency, the inter-channel parallelism strategy is employed. The computations related to C_p channels are processed in parallel until reaching the accumulation buffer, where they are merged to obtain an output channel. But it is inefficient to use a C_p-width adder tree to perform the channel accumulation, because the irregular sparse data structure may result in a lot of idle cycles and adders. So we allocate a FIFO to each PE, then the accumulation buffer access FIFOs of different channels sequentially. But when the parallelism C_p is very large, it will take a lot of cycles. Hence, we provide A_p copies of accumulation buffer, each of which is connected to C_p/A_p FIFOs, leading to significant speedup. Then an A_p-width adder tree is employed to merge the partial sums.

Post Processing. This module fetches tiles from accumulation buffer to perform the post-transformation and then stitches them into an output activation using the same storage structure of input activations. For small models like ResNet-18, the size of activations can fit in the on-chip memory. Thus the dataflow forms a inner loop and activations remain in the on-chip memory for the entire model. But for large models such as VGGNet, activations are needed to be written to and restored from the DRAM [15].

4. EXPERIMENTAL EVALUATION AND COMPARISON

In this work, we perform functional simulation of the LSW-CNN using both C and RTL code. Considering that both Wino-CNN and SpWA are based on the Xilinx ZC706, we also use the vivado synthesis tool to implement our design on the same platform for fair comparison. The implementation of our design can run over the frequency of 250MHz. In this work, weight and activation values are all quantized into 16-bit fixed point numbers. We explore the design space and make a tradeoff between memory footprint and speed by setting the parallelism C_p and A_p to 32 and 16, respectively.

4.1. Sparsity and Workload

It is shown in Fig. 4 that the comparison of four methodologies applied to the typical ResNet-18 on ImageNet with respect to their sparsity and workload of different blocks. These four methodologies are sparse neural network [13], naive WinoA [10], native pruned WinoA [14], and ReLU-modified WinoA [18], which are leveraged in SCNN [15], Wino-CNN [12], SpWA [17], and our LSW-CNN, respectively. It is noted that the ReLU-modified WinoA reaches to considerable low density in both activations and weights. So the proposed design can achieves an ultra low workload (8.0%, 17.7%, 28.3%, and 35.5% of conventional CNN, Wino-CNN, SCNN, and SpWA, respectively). Furthermore, the very few workload also results in reduced requirements for computing units, bandwidth, memory access, and storage resources, which make our architecture more hardware-efficient.

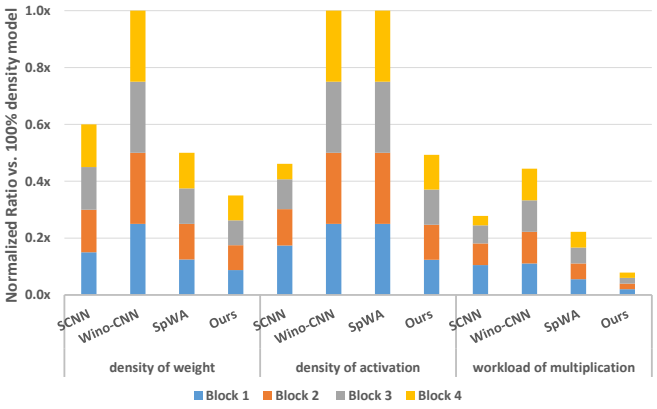


Fig. 4. Comparison of sparsity and workload on ResNet-18.

4.2. Architecture comparison and Analysis

In budget-restricted scenarios such as edge computing, strict constraints like latency, energy consumption, and hardware resource must be considered to find an optimal tradeoff during the design space exploration. On one hand, the LSW-CNN benefits from the sparse-Winograd model, both weights and activations are indexed into the compressed structure to skip unnecessary cycles and associated computing energy. In contrast, the SpWA only indexes weights, so it cannot skip the computations of activations with 0 values. On the other hand, the proposed FMI scheme brings negligible overhead for computational latency and memory. Moreover, under the same storage constrain, a highly compressed data structure allows the entire layer stays on-chip and avoid memory access to DRAM, which are very slow and high energy-consuming. Hence, the LSW-CNN takes the advantage of both sparse WinoA and dedicated hardware design, while SpWA mainly benefits from the former.

Table 1. Architecture comparison

	Wino-CNN [12, 17]	SpWA [17]	Ours
Precision	16bits fixed	16bits fixed	16bits fixed
Board	Xilinx ZC706	ZC706	ZC706
Freq. (MHz)	166	166	**250**
BRAM (Kb)	540x18	732x18	**528x18**
DSP	532	768	**380**
LUT	**90k**	155k	93k
Flip-Flop	**92k**	153k	96k

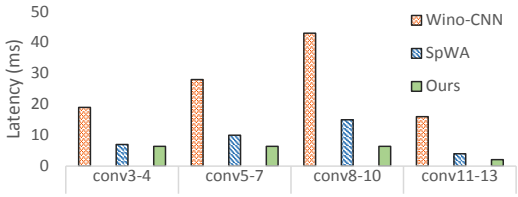


Fig. 5. Latency on VGG-16 with 20% weight density [17]

The hardware implementation results of similar architectures and our design are listed in Table. 1. Here, the data of Wino-CNN on Xilinx ZC706 comes from [17], because [12] only shows results on Xilinx ZCU102. As shown in Fig. 5, we compare the latency of these designs on ZC706 using VGG-16 model with 20% weight density [17]. It is noted that, when compared with Wino-CNN which is aimed at dense models, our design reaches 5.1× overall speedup with similar hardware resource consumption. Besides, the LSW-CNN achieves 1.7× latency improvement, and 2.0× DSP, 1.7× LUT, 1.6× flip-flop and 1.4× BRAM saving than the state-of-the-art SpWA.

5. CONCLUSION

This paper proposes a LSW-CNN architecture, which fully leverages sparsity in both activations and weights and alleviates the workload unbalance via dedicated hardware design. It achieves significant speedup and considerable power and hardware resource savings compared with other state-of-the-art architectures. Experimental results show that our design is much more suitable for budget-restricted scenarios.

6. REFERENCES

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

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of GPU-based convolutional neural networks," in *2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2010, pp. 317–324.

[5] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[6] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 367–379.

[7] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[8] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss v2: A flexible and high-performance accelerator for emerging deep neural networks," *arXiv preprint arXiv:1807.07928*, 2018.

[9] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," *arXiv preprint arXiv:1312.5851*, 2013.

[10] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4013–4021.

[11] J. Wang, J. Lin, and Z. Wang, "Efficient convolution architectures for convolutional neural network," in *International Conference on Wireless Communications & Signal Processing*, 2016, pp. 1–5.

[12] L. Lu, Y. Liang, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on fpgas," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2017, pp. 101–108.

[13] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[14] S. Li, J. Park, and P. T. P. Tang, "Enabling sparse winograd convolution by native pruning," *arXiv preprint arXiv:1702.08597*, 2017.

[15] 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 *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2. ACM, 2017, pp. 27–40.

[16] A. Xygkis, L. Papadopoulos, D. Moloney, D. Soudris, and S. Yous, "Efficient winograd-based convolution kernel implementation on edge devices," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 136.

[17] L. Lu and Y. Liang, "Spwa: an efficient sparse winograd convolutional neural networks accelerator on fpgas," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[18] X. Liu, J. Pool, S. Han, and W. J. Dally, "Efficient sparse-winograd convolutional neural networks," *arXiv preprint arXiv:1802.06367*, 2018.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] S. Winograd, *Arithmetic complexity of computations*. Siam, 1980, vol. 33.

[21] "Sparse_matrix," https://en.wikipedia.org/wiki/Sparse_matrix, 2018.

[22] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities," in *2017 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2017, pp. 1–8.