A Winograd-based CNN Accelerator with a Fine-grained Regular Sparsity Pattern

Tao Yang*, Yunkun Liao*, Jianping Shi†, Yun Liang‡, Naifeng Jing*,and Li Jiang*§
*School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University
§MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
†SenseTime Group Limited ‡School of EECS, Peking University

***Abstract*—Field-Programmable Gate Array (FPGA) is a high-performance computing platform for Convolution Neural Networks (CNNs) inference. Winograd transformation and weight pruning are widely adopted to reduce the storage and arithmetic overhead in matrix multiplication of CNN on FPGAs. Recent studies strive to prune the weights in the Winograd domain, however, resulting in irregular sparse patterns and leading to low parallelism and reduced utilization of resources.**

In this paper, we propose a regular sparse pruning pattern in the Winograd-based CNN, namely Sub-Row-Balanced Sparsity (SRBS) pattern, to overcome the above challenge. Then, we develop a 2-step hardware co-optimization approach to improve the model accuracy using the SRBS pattern. Finally, we design an FPGA accelerator that takes advantage of the SRBS pattern to eliminate low-parallelism computation and irregular memory accesses. Experimental results on VGG16 and Resnet-18 with CIFAR-10 and Imagenet show up to 4.4× and 3.06× speedup compared with the state-of-the-art dense Winograd accelerator and 52% (theoretical upper-bound is 72%) performance enhancement compared with the state-of-the-art sparse Winograd accelerator. The resulting sparsity ratio is 80% and 75% and the loss of model accuracy is negligible.

I. Introduction

Deep convolutional neural networks(CNN) have achieved remarkable momentum in various computer vision tasks [1]. Nevertheless, the computation of the convolutional layers accounts for more than 90% of modern CNNs [2]. FPGA has become a promising platform for accelerating CNN because of its high efficiency and high programmability [2]. A large amount of feature maps and filters in CNNs, leading to explosive computational complexity and memory consumption, limits the performance of these accelerators.

Various methods propose to reduce the amount of computation in the convolutional layer. A practical approach is weight pruning [3]. CNN is over-parametrized, and thereby a large number of weights is redundant and is pruned without incurring critical model accuracy loss [4][5]. Another way is to use an alternative matrix-multiplication algorithm, e.g., Winograd [6], to reduce the multiplication operation at the cost of increasing addition operations. Lu et al. [7] proposed a Winograd-based CNN accelerator by recognizing the DSP-friendly property of the Winograd algorithm. The challenge to derive a sparse Winograd-based CNNs lies in the Winograd transformation process, which cannot preserve the sparsity derived by pruning the convolution layers in the spatial domain.

Some practical strategies achieve sparsity in the Winograd domain [8], [9]. These works strive to sort all the "weights" in the Winograd domain, considering the weight value associated with an "importance" factor, and prune the less important ones. However, the resulting sparsity is irregular, which causes imbalanced workloads among the processing elements (PE) and irregular memory access patterns in computation. Thus, it is hard to obtain the expected speedup ratio in the customized accelerator in FPGA. Based on these works, Lu [10] introduced an accelerator for sparse Winograd-based CNN called SpWA [10]. SpWA groups the sparse vectors with similar sparsity ratios and attributes each group to a PE. The number of DSPs in each PE is inversely proportional to the sparsity ratio of this group. This coarse-grained grouping method relieves the 'irregular sparsity' problem to a certain degree. However, it ignores the dramatic load imbalance among those vectors within a group.

In this paper, we propose a novel software-hardware co-design approach to resolve the "irregular sparsity" problem in the Winograd-based CNN. The main contributions of this paper are summarized as follows:

- We propose a novel pruning method amendable to Winograd-CNN. We discover a pruning pattern, namely Sub-Row-Balanced sparsity (SRBS), whose granularity is optimized for high parallelism. We propose *Position-Sensitive Pruning* strategy to attribute sparsity ratios to Winograd-matrices dynamically and a two-step pruning process using SRBS pattern to maintain the accuracy of models with fast convergence.
- We design a novel FPGA-based accelerator for the derived sparse Winograd-CNN. In order to balance the workloads among PEs, we allocate DSPs to each Processing Elements (PEs) based on the position-sensitive sparsity ratios leading by our Pruning Strategy. The Processing Elements (PEs) carry out dense multiplications and sparse addition. Such design improves the DSP efficiency and allows us to make a further trade-off between the DSPs and LUTs.
- Experiments show that our design achieves 4.4× and 3.06× speedup compared with the state-of-the-art dense Winograd accelerator on VGG16 and Resnet-18 with negligible loss of model accuracy under the sparsity

978-1-7281-9902-3/20/$31.00 ©2020 IEEE
DOI 10.1109/FPL50879.2020.00050

ratio of 80% and 75%. Further comparison shows that our accelerator outperforms the state-of-the-art sparse Winograd technique. We prove that the results are close to the theoretical upper-bound of speedup.

The remainder of this paper is organized as follows: Section II introduces the background. Section III and IV show the proposed pruning algorithm and architecture design. The experiments are shown in Section V. We conclude the paper in Section VI.

II. BACKGROUND

A. Convolution Using Winograd Algorithm

The convolution operation between M channels of $H \times W$ input feature maps (denoted as $I_{M\times H\times W}$) and $N \times M$ spatial filters with the size of $r\times r$ (denoted as $G_{r\times r\times M\times N}$), generates N channels of $R\times C$ output feature maps (denoted as $O_{N\times R\times C}$). Equation 1 describes the spatial convolution operation in one layer with stride S.

$$O(k,i,j)=\sum_{t=1}^{M}\sum_{p=1}^{r}\sum_{q=1}^{r}G(p,q,t,k)\times I(t,i*S+p,j*S+q) \tag{1}$$

Winograd minimal filter algorithm computes minimal complexity convolution over small tiles. Here we denote 2D Winograd minimal filtering algorithm as $F(m\times m, r\times r)$. A, B, and G are corresponding transformation matrix determined by F. The process is denoted as Equation 2. The number of multiplications is determined by $\odot$ (Hadamard multiplication) operation. The size of (GgG^T) and (B^TdB) are $n\times n$ $(n=m+r-1)$. For $F(2\times 2, 3\times 3)$, to give 2×2 results of 2D convolution, the number of multiplications we need is 16. As for spatial convolution, the number is 36 according to Equation 1. Winograd can significantly reduce multiplication operations in convolution.

$$O=A^T[(GgG^T)\odot(B^TdB)]A \tag{2}$$

B. Weight Pruning

Weight pruning exploits the sparsity of the model to reduce the overall memory and computational cost. The pruning methods are classified as unstructured pruning, structured pruning, and hardware-aware regular pruning. Each of them corresponds to a different granularity of sparsity. The unstructured pruning method evaluates all the weights in a filter and removes those weights below a small threshold. This pruning method has a high sparsity ratio and high model accuracy, however, results in irregular sparsity in the weight matrix. Consequently, two critical issues arise. First, skipping those pruned weights incurs the imbalance data requests across different memory banks in FPGA, or concurrent data requests to the same memory bank, leading to discontinuous memory access and bank conflict, respectively. Second, the imbalance workloads cause tremendous idle cycles on DSPs, leading to low hardware efficiency.

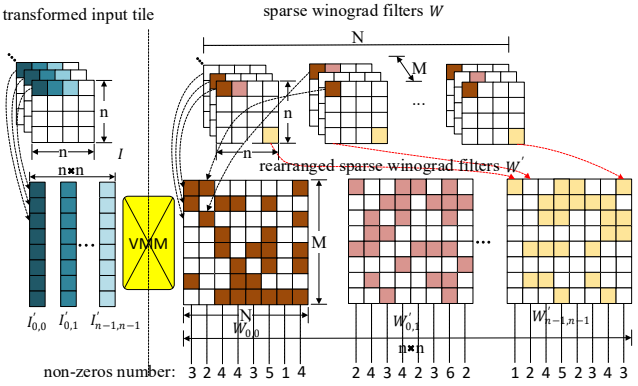

Figure 1: Rearrange filter and input feature maps to exploit VMM

Structured pruning, such as channel-level [4] and filter-level[5] pruning, exploits the sparsity in CNN structures to mitigate the above issues. The parallel computing architecture can inherently and fully utilize this regular sparsity to improve the performance. However, these pruning methods change the dimensions of the feature maps in the CNN structure, and subsequently, a significant accuracy loss if pursuing a high sparsity ratio.

A compromise solution is a hardware-aware regular pruning method. It prunes finer-grained CNN structures according to some hardware-friendly sparse patterns and looks for a good tradeoff between the model accuracy and hardware efficiency. Coarse-grained hardware-aware pruning methods, such as pruning weights in the granularity of blocks of weights [11], relieve the low hardware efficiency problem, but still suffer from the low model accuracy if we need high sparsity ratio. Intuitively, it can reduce the accuracy loss of a CNN to prune the weights in finer granularity. Cao [12] proposes a bank-balanced sparse pattern. Each weight matrix row is divided into multiple small banks and adopts fine-grained pruning to each bank independently to obtain the same sparsity among banks. This pattern can offer balanced workload for vector-matrix multiplications without noticeable loss of model accuracy with a high sparsity ratio in LSTM. Until now, no one ever shows a feasible hardware-aware regular pruning method in Winograd-based CNN without a noticeable loss of accuracy.

C. Sparse Winograd-based CNN Accelerator

To obtain sparsity in Winograd-based CNN, Li[9] and Liu[8] both replace a spatial convolutional layer with a Winograd-based convolutional layer, train the replaced model and use unstructured pruning on the weights in the Winograd domain. Based on their works, Lu[10] build a sparse Winograd-based CNN accelerator called SpWA. SpWA transforms element-wise matrix multiplication and accumulation (EWMM-accumulation) operation to Vector-Matrix Multiplication (VMM). As shown in Fig. 1, sparse Wiongrad filters are transformed weights (GgG^T), denoted

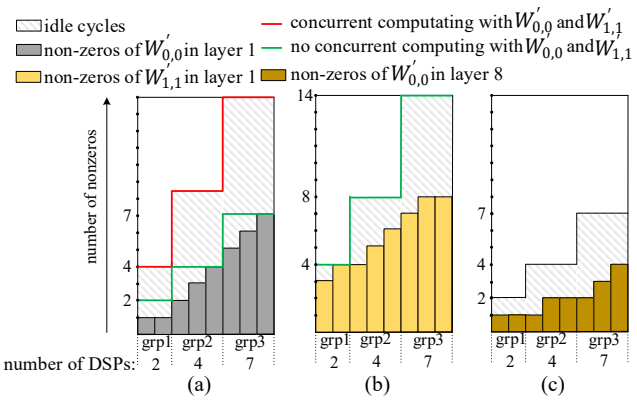

Figure 2: Several cases of the method to improve the load balance in SpWA: (a) $W'_{0,0}$ in layer 1; (b) $W'_{1,1}$ in layer 1; (c) $W'_{0,0}$ in layer 8.

as $W_{N\times M\times n\times n}$. Transformed input tiles $(B^T dB)$ are denoted as $I_{M\times n\times n}$. We denote the corresponding Winograd-domain output feature maps as $O_{N\times n\times n}$. The EWMM-accumulation operation in Winograd algorithm can be expressed as Equation 3.

$$O_{tn} = \sum_{tm=1}^{M} W_{tn,tm} \odot I_{tm} \tag{3}$$

SpWA rearranges transformed input tile to $n \times n$ separate vectors I' with the length of M and rearranges the sparse Wiongrad filters from $N \times M$ matrices to $n \times n$ matrices W'. After that, VMM is executed between $I'_{i,j}$ and corresponding $W'_{i,j}$ to calculate the Winograd-domain output feature maps.

To balance the workloads of DSPs in VMM, SpWA uses a greedy-based algorithm to allocate computing resources. Using the rearranged sparse Winograd filter W' in Fig. 1 as an example, where $M = N = 8$, the SpWA sorts the columns according to the number of nonzeros and traverse all $W'_{i,j}$ of all layers in the CNN model using a greedy-based algorithm to find a grouping method with minimum idle cycles. As shown in Fig. 2(a), the columns in each $W'_{i,j}$ are divided into three groups using this greedy-based method. The workload of each group is allocated to a PE. SpWA distributes the available DSPs to each PE according to the number of nonzeros in each group. Consequently, the execution time of each group is close to each other.

The above strategy, however, cannot provide a fine-grained load balance. As shown in Fig. 2(a), different columns within a group have diverse amounts of nonzeros so that idle cycles are inevitable. Moreover, different rearranged Winograd filters may exhibit significant diversity in sparsity ratios. Fig. 2(b) shows the idle cycles in the computation of $W'_{1,1}$ with the same number of DSPs allocated in the three groups in Fig. 2(a), we can observe more idle cycles than that in $W'_{0,0}$. More idle cycles occur if the accelerator may compute with $W'_{0,0}$ and $W'_{1,1}$ concurrently. The reason is that the number of nonzeros in the columns of $W'_{1,1}$ is bigger than the number of allocated DSPs in this group, so that the computation with $W'_{1,1}$ may need multiple cycles, which blocks the pipeline of the parallel execution of $W'_{0,0}$. We can also observe a larger difference in the sparsity ratio among different convolution layers, as shown in Fig. 2(c), leading to severer load imbalance.

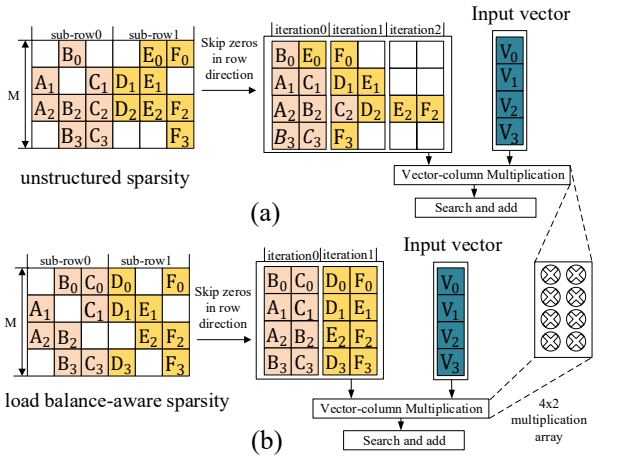

Figure 3: The difference between unstructured sparsity and Sub-Row-Balanced sparsity

We find that the variance of sparsity ratios becomes more significant among larger CNN structures, e.g., columns, matrixes and layers. Thus, the load imbalanced induced by the unstructured pruning method in Winograd domain cannot be easily mitigated by sole accelerator architecture optimization like [13], [9].

III. Proposed CNN Pruning Algorithm

In this section, we first explore the Winograd sparsity pattern with suitable pruning granularity optimized for high parallelism. Then, we propose the Position-Sensitive Pruning strategy and use the resulting position-aware sparsity to guide the allocation of DSPs. Finally, we propose the Two-Step Pruning Process to retrain the sparse Winograd-based CNN to maintain high accuracy with fast convergence.

A. Sub-Row-Balanced Sparsity in Winograd domain

Generally, it is mandatory to divide a large calculation into small calculations for parallel computing because this method lowers requirements for valuable on-chip storage resources and computing resources. For example, in Fig. 3, we make each three columns in the rearranged Winograd filter into a group and execute the VMM in one group each time to get three outputs. In order to improve the efficiency of precious multiplication resources in hardware, we merge the sparse matrix by skipping zeros in the row direction to make the multiplications between the input vector and the merged matrix denser. In order to get the correct accumulative result of each output channel, we must fetch the exact operands and add them together from the result of dense multiplications. For example, in Fig. 3(a), to get the

accumulative result of column 0, we need fetch $A_1 \times V_1$ and $A_2 \times V_2$. However, unstructured pruning causes an irregular nonzeros distribution in the weight matrix, resulting in that the operands needed by a result of an output channel may occur in an enormous scope of areas in the memory of multiplication results. In Fig. 3(a), for example, given a 4×2 multiplication array, to get the accumulative result of column 4, we need results of 3 iterations of multiplications: $E_0 \times V_0$ from iteration 0, $E_1 \times V_1$ from iteration 1 and $E_2 \times V_2$ from iteration 2. This data dependence from iteration to iteration will block the pipeline of parallel resources for addition. Furthermore, it is a significant waste of on-chip memory because we need to store the multiplication results from multiple iterations. For both the memory resources on FPGA, BRAM and registers, it's a disaster to store the multiplication results with this irregular data distribution. For BRAMs, the storage access conflict may occur because each BRAM can offer as much as two data in a cycle. However, the accumulators are more likely to fetch operands from more than two data in one BRAM at a time due to the irregular distribution of the addends. Registers, instead, can offer big fan out, but the operands of additions need to be searched from a large memory scope, resulting in an enormous consumption of logic resources.

For the previous reason, we restrict each sub-row to have the same sparsity ratio, so that the elements of the same group appear in one rectangular block after the merge operation. For example, in Fig. 3(b), we set the sparsity of each sub-row to $\frac{1}{3}$. Given a 4×2 multiplication array, the operands of additions required for every three columns are located in the results of one iteration of multiplications. Compared with the unstructured approach, there is no data dependence relations among iterations, which significantly reduce the enormous memory consumption. Moreover, we search the corresponding addends for an adder in a relatively small storage space, which reduces the huge logical consumption.

B. Position-Sensitive Pruning In Rearranged Filters

Pruning CNN in the Winograd domain is different from that in the spatial domain. Each weight in the Winograd domain is the linear transformation of several weights in the spatial domain, so the effects of applying the same pruning rules in the spatial domain and the Winograd domain are different. Previous work [14] has proved that weights in different positions of a Winograd filter contribute differently to the output feature maps, the contribution of each weight is determined by which $W^{'}_{i,j}$ the weight belongs to. Besides, as shown in Fig. 1, VMMs are executed among the n^2 rearranged Winograd sparse filters and corresponding transformed input vectors with no data dependence relations between each other. These two features both lead us to divide the VMMs in the Winograd domain into n^2 groups and treat these groups differentially in pruning.

Based on the discussion before, our pruning strategy's target is to give a unique sparsity ratio for each sub-row in a $W^{'}_{i,j}$ among different layers. Meanwhile, the strategy should maintain the accuracy of the model as high as possible.

We use the equation 4 to illustrate the overall sparsity ratio constraint, where ρ is the target sparsity in a model and $\rho_{i,j}$ is the sparsity ratio of $W^{'}_{i,j}$.

$$\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\rho_{i,j}}{n \times n} = \rho \tag{4}$$

The setting of the sparsity ratio for each rearranged filter follows the principle: the more important the $W^{'}_{i,j}$ is, the smaller sparsity ratio for $W^{'}_{i,j}$. We first consider the absolute value of individual weight. As shown in Equation 5, we average the absolute value of weights in each $W^{'}_{i,j}$ as an importance factor of $W^{'}_{i,j}$. C_{out} and C_{in} are output and input channel dimensions of the current Winograd convolutional layer.

$$\overline{|W^{'}_{i,j}|} = \frac{\sum_{tn=1}^{C_{out}}\sum_{tm=1}^{C_{in}}|W^{'}_{i,j,tm,tn}|}{C_{out} \times C_{in}} \tag{5}$$

Then, a $n \times n$ matrix P is used to describe the contribution of each $W^{'}_{i,j}$ to the output feature maps. Yu proved that $P_{i,j}$ can be computed by Equation 6 [14].

$$P_{i,j} = \sqrt{\sum_{1 \le x,y \le n, 1 \le s,t \le r} A_{i,x}A_{j,y}B_{s,j}B_{t,j}} \tag{6}$$

We adjust $\overline{|W^{'}_{i,j}|}$ by multiplying $P_{i,j}$. Then, $\overline{|W^{'}_{i,j}|}P_{i,j}$ is used to evaluate the importance of $W^{'}_{i,j}$. We approximate the pruning ratio of $W^{'}_{i,j}$ by Equation 7 (b is a scaling factor).

$$\rho_{i,j} = 1 - b \times \overline{|W^{'}_{i,j}|}P_{i,j}, b > 0 \tag{7}$$

Combine Equation 7 with Equation 4, we can get Equation 8, which approximates $\rho_{i,j}$ from both weight magnitude and the contribution of weights in Winograd domain to output feature maps.

$$\rho_{i,j} = 1 - \frac{\overline{|W^{'}_{i,j}|}P_{i,j} \times n \times n \times (1-\rho)}{\sum_{i=1}^{n}\sum_{j=1}^{n}(\overline{|W^{'}_{i,j}|}P_{i,j})} \tag{8}$$

Because $\overline{|W^{'}_{i,j}|}P_{i,j}$ is not bounded, some $\rho_{i,j}$ could be less than 0 according to Equation 8, which means corresponding $W^{'}_{i,j}$ is extremely important. But such a situation is meaningless in reality, so We force these $\rho_{i,j}$ to be 0 and compensate other less important $W^{'}_{m,n}$ by decreasing its $\rho_{m,n}$. Algorithm 1 illustrates the details. We first sort these $\rho_{i,j}$ in ascending order. Then, in each reassigning iteration, the smallest $\rho_{i,j}$ less than zero is reassigned to zero and this $\rho_{i,j}$ is added to the Adjacent $\rho_{i,j}$ to maintain the overall sparsity ratio.

For a typical Winograd-based CNN, suppose there are total c Winograd-based convolutional layers denoted as $L_1, .., L_c$, these layers share the same acceleration core in FPGA-based accelerator [2]. The difference in sparsity ratios

Algorithm 1: Pruning ratio adjusting algorithm

Input: The original pruning ratio list $L : \{\rho_{1,1}, ..., \rho_{n,n}\}$
Output: The adjusted pruning ratio list L

```
/* argsort returns the indices that sort
   an list in ascending order                */
sorted_index ← argsort (pruneRatio), i ← 1
while L[sorted_index[i]] < 0 do
    L[sorted_index[i+1]] ←
      L[sorted_index[i]] + L[sorted_index[i+1]]
    L[sorted_index[i]] ← 0
    i ← i + 1
end while
return L
```

between layers can bring imbalanced workloads. To address the problem, we propose to uniform the sparsity ratios calculated layer by layer. Within our experiments, averaging is the most simple and effective method. For simplicity, we denote the calculated sparsity ratio in layer l as $\rho_{l,i,j}$ and the unified sparsity ratio as $\rho_{i,j}$. The averaging method is shown in Equation 9.

$$\rho_{i,j} = \frac{\sum_{l=L_1}^{L_c} \rho_{l,i,j}}{L_c} \tag{9}$$

C. Two-Step Pruning Process

Given sub-row size S, we divided each row of the rearranged sparse Winograd filter $W'_{i,j}$ into $\frac{N}{S}$ sub-rows and perform fine-grained weight pruning inside each sub-row according to its sparsity ratio $\rho_{i,j}$. Like the previous approach[3], we prune the weights of which the absolute value are small. However, this boorish pruning leads to intolerable accuracy loss in our experiments. To address this problem, we propose a progressive pruning method. Our method is different with the previous iterative pruning method used in BBS [12] which slowly increases the pruning percentage from 0% to the target sparsity ratio, there are only two iterations in our pruning method. At the first step we prune $\lfloor N \times \rho_{i,j} \rfloor$ within each entire row. Second, we prune $\lfloor S \times \rho_{i,j} \rfloor$ weights in each sub-row. The Two-Step pruning method maintains model accuracy effectively while reducing the convergence time significantly in our experiments.

In the first step, We observe that the pruning space of the total row-sparse pattern is close to the unstructured pattern theoretically. The row-sparse model of the first step is friendly for sub-row pruning because the first step behaves as a regularization process for each row of the weight matrix. The regularization process guarantees enough sparsity for next step pruning.

IV. ARCHITECTURE DESIGN

In this section, we describe an FPGA accelerator based on the Sub-Row-Balanced sparsity pattern in Winograd domain named WSRBS accelerator, WSRBS accelerator is

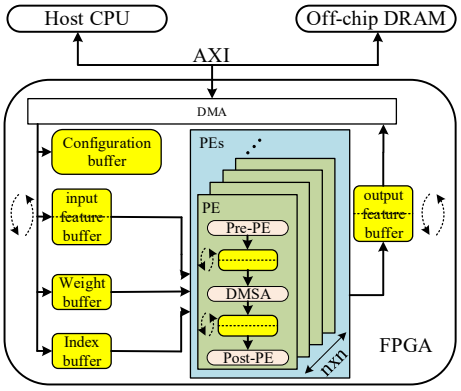

Figure 4: Overall architecture of WSRBS accelerator

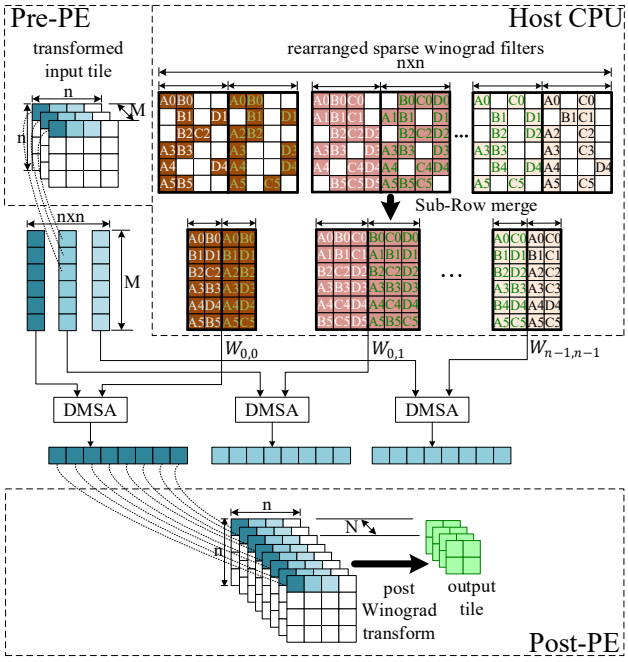

Figure 5: Dataflow of WSRBS accelerator

implemented as an accelerator on the AXI bus to serve CNN inference request from the host CPU.

A. Architecture Overview

Fig. 4 shows the overall architecture of the WSRBS accelerator. Each PE consists of the pre-processing element (denote as pre-PE), the Dense-Multiplication-Sparse-Addition (denoted as DMSA) core and the post-processing element(post-PE). We store $(n + m)$ rows of feature maps in the input feature buffer, and each row occupies a sub-line-buffer. We use n of the $(n + m)$ sub-line-buffers each time to behave as ping-pong buffers. Correspondingly, on the output side, we also use $(2m)$ sub-line-buffers as ping-pong buffers. Moreover, we allocate double buffers among pre-PE, DMSA and post-PE. Thus we can overlap the transformation operations and computations.

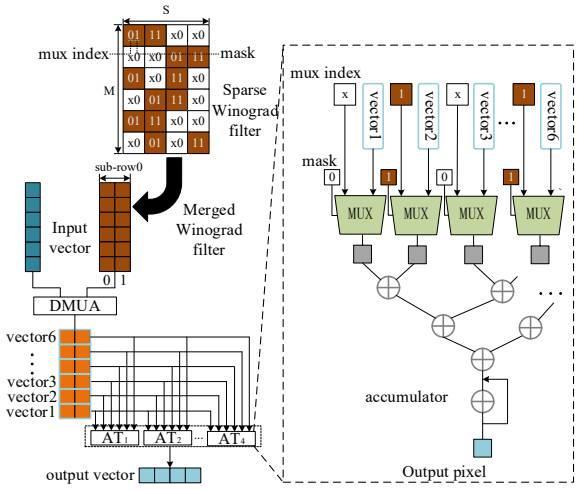

Figure 6: DMSA Architecture

B. Dataflow

Fig. 5 shows the dataflow of the WSRBS accelerator. During initialization, the host CPU does the one-time job to merge the sparse Winograd filters by skipping the zeros in each sub-row. When we start the calculation stage, Pre-PEs execute Winograd input transformation $(B^T dB)$ and rearrange the transformed input tile into several vectors. Then, the vectors and merged sparse Winograd filters with different $\rho_{i,j}$ are fed into the corresponding DMSAs to calculate the Winograd-domain output feature maps. After getting all the output vectors of DMSAs, the Post-PEs inversely rearrange the vectors and use post-Winograd transform to get the output tile and store the results in output feature buffers. Finally, the output tile is sent back to Off-chip DRAM by DMA.

C. DMSA Architecture

The architecture of DMSA is shown in Fig. 6. After the input vectors and weights are fed into the DMSA, each vector is multiplied by each column of merged Winograd filter in dense multiplication array (DMUA) using DSPs. Then the multiplication results are rearranged into M vectors, and each vector broadcasts to S addition-trees (ATs). Each AT corresponds to an item of the output vector. Besides the calculated vectors, the index matrix is also sent to ATs. As shown in Fig. 6, each item in the index matrix is composed of a mask and an index. The mask is a nonzero identifier to determine whether the corresponding addend is in the vector. The index means the column number in the merged Winograd filter matrix with which we can fetch the corresponding addend in the vector. Correspondingly, a multiplexer with enable signal is allocated on each port of the AT to select the right addend.

In Table I, We compare two widely accepted sparse index format with ours, where we denote $S \times (1 - \rho_{i,j})$ as N_d and the number of nonzeros in a $M \times N$ filter matrix as

	The number of indexes	Memory Usage
CSC	$Z + N$	$Z \times log_2 M + N \times log_2 Z$
Re-CSC	$Z + 2N$	$Z \times log_2 M + N \times log_2 Z + N \times log_2 N$
Ours	$M \times N$	$M \times N \times (log_2 N_d + 1)$

Table I: Comparison between our index format and two CSC formats

Z. We can figure that although the number of the indexes in our format is bigger than the other two formats, we only need $1 + log_2 N_d$ bits for one item in index matrix, which means compensation for memory usage. Giving a 32×16 filter matrix and setting $S = 8$ and $\rho_{i,j} = 0.75$, the memory usage of indexes in CSC and re-CSC [10] is 752 bits and 816 bits, and the memory usage of our method is 1024 bits. Our format only consumes 25% memory more than re-CSC format for indexes. As for Weights, there is no difference because all the three methods only store nonzeros.

D. DMSA Efficiency

In each DMSA, the Sub-Row-Balanced pattern restricts the dependence relations among multiple iterations of calculations in sub-row groups, which provides us with opportunities to gain parallelism within a sub-row group and among various sub-row groups. Moreover, the number of DSPs allocated in each DMSA is $G \times M \times S \times (1 - \rho_{i,j})$, G refers to the parallel number of sub-row groups. With this dynamic DSPs allocation strategy, we can make sure that the latencies of multiplications in each sub-row group are the same because the number of DSPs in each DMSA is proportional to $\rho_{i,j}$, which means a complete balance of workloads to DSPs in all DMSAs. The number of ATs allocated in each DMSA is $G \times S$, which can promise the throughputs of the multiplications and the additions are the same. Besides, our algorithm can guarantee the same sparsity ratio of each layer, with which we can keep this high efficiency through the Winograd-based CNN model.

V. EXPERIMENT EVALUATION

A. Algorithm Part

1) Experiment Setting: To evaluate our pruning method, we perform the experiments on three datasets: Imagenet(ILSVRC-2012)[15], CIFAR-10[16], and CIFAR-100[16]. We test ResNet-18[1], VGG16[17], and ConvPool-CNN-C[18] respectively on the three datasets. pytorch[19] is used to implement the pruning framework. In the experiments, we chose $F(2 \times 2, 3 \times 3)$ Winograd Convolution and set the size of a sub-row to 8. We define the average Winograd layer sparsity as the average sparsity of all the pruned Winogad layers unless otherwise specified. We use Imagenet to evaluate the effectiveness of our proposed pruning methods for its diversity and richness.

2) Resnet-18 on Imagenet: To better compare the model accuracy of our pruning method with the SOTA approach of unstructured pruning for Winograd-CNN proposed by

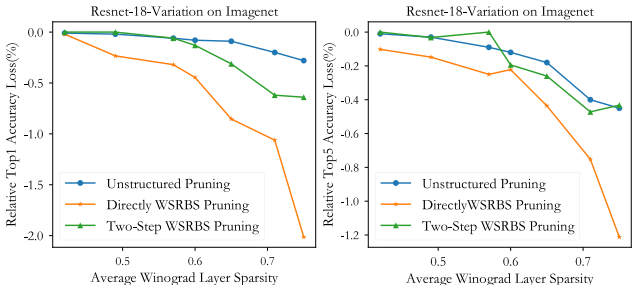

Figure 7: Top-1/Top-5 validation accuracy loss vs average layer sparsity for three pruning algorithms on ResNet-18

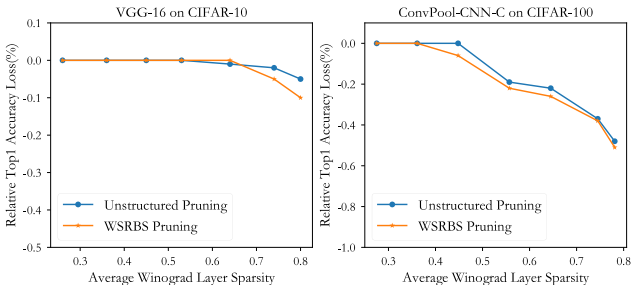

Figure 8: Top-1 validation accuracy loss vs average layer sparsity for two pruning algorithms on CIFAR-10/100

Liu[8], we adopt the Resnet-18-Variation model offered by Liu[8] and follow the same settings except for pruning algorithm and hyperparameters. The corresponding Resnet-18-Variation model can achieve validation top-1/top-5 accuracy of 66.78%/87.43% in Winograd domain.

Importance Adjustment: To test the effectiveness of our importance adjustment algorithm in section·III-B, we design a simple but insightful experiment. For simplicity, we follow the symbol system in section III-B. We separately prune each $W^{'}_{i,j}(1 \leq i,j \leq 4)$ in all the sixteen Winograd layers of Resnet-18 with sub-row size of N (N is the the number of filters in a Winograd layer) and 60% sparsity. The corresponding top-1 error of pruned $\text{Model}_{i,j}$ without time-consuming finetuning is $E_{i,j}$. Then, we use Pearson coefficient(denote as $corr$) to measure $E_{i,j}$'s relevance to weight importance $\overline{|W^{'}_{i,j}|}$ and adjusted weight importance $\overline{|W^{'}_{i,j}|}P_{i,j}$.

The experiment result shows $corr(E_{i,j}, \overline{|W^{'}_{i,j}|})$ equals to 0.737 while $corr(E_{i,j}, \overline{|W^{'}_{i,j}|}P_{i,j})$ equals to 0.803, which indicates that $\overline{|W^{'}_{i,j}|}P_{i,j}$ can better evaluate the importance of $W^{'}_{i,j}$.

Winograd-domain SRBS on Imagenet: Fig. 7 shows the accuracy as a function of average Winograd layer sparsity for three pruning algorithms in the Winograd domain, which are unstructured pruning, directly SRBS pruning and two-step SRBS pruning. We use the accuracies reported in Liu [8] as the baseline of unstructured pruning. The result shows that the sparse model with SRBS pattern in the Winograd domain can approach the accuracy($< 0.5\%$) of the sparse model with irregular pattern in the Winograd domain. On the other hand, the result also shows that our Two-Step Pruning is effective.

3) VGG16/ConvPool-CNN-C on CIFAR: We also apply our pruning method on CIFAR-10/CIFAR-100. For the CIFAR-10 dataset, the VGG16 model is tested. The Winograd baseline model can achieve validation set accuracy of 92.66%. The pruning framework is implemented with pytorch and ReLU is removed from Winograd domain. We keep the first Winograd layer with a fixed sparsity of 20% for its sensibility and prune the remaining 12 Winograd layers. For the CIFAR-100 dataset, the ConvPool-CNN-C model is tested. The Winograd baseline model can achieve validation set accuracy of 71.98%, which is higher than the reported accuracy in Liu[8](69.75%). We keep the first Winograd layer with a fixed sparsity of 20% too and prune the remaining 7 Winograd layers. The average Winograd layer sparsity is defined as the average sparsity of all the pruned Winogad layers without the first Winograd layer. The results are shown in Fig. 8, which shows that our pruning algorithm is a little more effective in small datasets.

B. Hardware Part

1) Experiment Setting: We implement the WSRBS accelerator using Xilinx Vivado HLS 2018.3. Then we use the SDSoC to generate the bitstream. We evaluate our techniques on the Xilinx ZCU102 platform. We use an 8-bits fixed data type, which is widely accepted in the study because of the guaranteed accuracy compared with the float data type. Our FPGA implementation operates at 166MHz. In the experiments, we use a typical input picture map size of 224x224, which is the same with [7] and [10]. Besides, in both the below cases, We set the size of a sub-row to 8, and $G = 1$.

2) Case study of VGG16 and Resnet-18: The total sparsity ratio of the VGG16 model is 80%, and the comparison in Fig. 8(a) shows that the accuracy achieved by our pruning method is comparable to the accuracy achieved by unstructured pruning Which is used in SpWA. With this sparsity ratio in Weights, the theoretical speedup is 5x compared with dense Winograd design [7] if we do not put the pipeline depth and bandwidth limit into consideration. It means a 72% performance enhancement upper bound for SpWA (2.9x speedup compared with dense Winograd design using VGG16). As shown in Fig. 9(a), pruning using the number of nonzeros in sub-rows in each $W^{'}_{i,j}$ with $M = 20$, we achieve 4.4x speedup compared with dense Winograd design, 1.52x speedup (52% performance enhancement) compared with SpWA with the same sparsity. We also exploit the sparse Resnet-18 on our accelerator. The sparsity ratio of the weights in Resnet-18 is 75.78%. The case is shown in Fig. 9(b). We set $M = 16$ and achieve 3.06x speedup compared with dense Winograd design [7]. The speedups in the last few layers are not significant. The

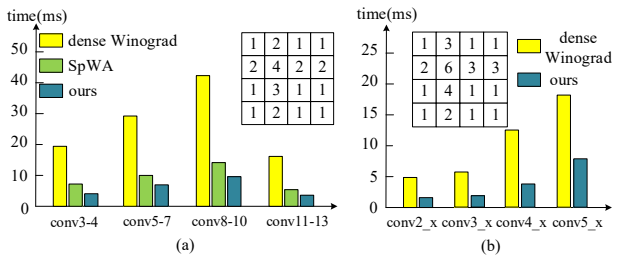

Figure 9: Experimental results: (a) number of nonzeros in sub-rows of each $W^{'}_{i,j}$ and comparison between prior works and our design on VGG16; (b) number of nonzeros in sub-rows of each $W^{'}_{i,j}$ and comparison between prior works and our design on Resnet-18.

	BRAM18K	DSP48E	FF	LUT
Dense Winograd	540	532	91874	89628
SpWA	732	768	153020	155206
ours on VGG16	**736**	**525**	**107548**	**221370**
ours on Resnet-18	**739**	**517**	**130463**	**204496**

Table II: Resources utilization

reason is that the size of feature maps of these layers is too small, which incurs short pipelines and the pipelines account for most of the total latencies in these layers.

The resources utilization of the three designs is shown in Table II. We can figure that our design occupies the least number of DSPs but achieves the best performance. The reason is that our pattern and hardware architecture can entirely solve the imbalanced workloads of DSPs. Our design uses more LUTs because the number of ATs used in each DMSA is S in order to guarantee adequate throughput. Besides, the size of the index matrix in our design is a little bigger than that in SpWA, which needs a little more BRAMs.

VI. Conclusion

Hardware efficiency is always quite important concerning the crucial requirements of latency on many occasions. In our work, we first give a hardware friendly pattern named SRBS, which can guarantee balanced workloads for the most valuable resources on FPGA(e.g., DSPs). Then we introduce a novel algorithm and a Two-Step Pruning Process to get position-aware sparsity ratios using SRBS pattern in the Winograd domain, which helps to maintain model accuracy. Finally, we exploit an accelerator to prove that our design has the superiority in performance over the previous Winograd-based architectures. Our design achieves 4.4x and 3.06x speedup on VGG16 and Resnet-18 compared with state-of-the-art dense Winograd architecture with maintained model accuracy.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China (Grant No. 61834006), National Key Reserch and Development Program of China (2018YFB1403400), Shanghai Science and Technology Committee (No.18ZR1421400). The author Tao Yang is supported by Wu Wen Jun Honorary Doctoral Scholarship, AI Institute, Shanghai Jiao Tong University. Corresponding author: Li Jiang.

References

[1] W. Liu *et al.*, "A survey of deep neural network architectures and their applications," *Neurocomputing*, 2017.

[2] K. Guo *et al.*, "[dl] a survey of fpga-based neural network inference accelerators," *TRETS*, 2019.

[3] S. Han *et al.*, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *ICLR*, 2016.

[4] H. Li *et al.*, "Pruning filters for efficient convnets," in *ICLR*, 2017.

[5] Z. Liu *et al.*, "Learning efficient convolutional networks through network slimming," *ICCV*, 2017.

[6] Lavin *et al.*, "Fast algorithms for convolutional neural networks," in *CVPR*, 2016.

[7] L. Lu *et al.*, "Evaluating fast algorithms for convolutional neural networks on fpgas," in *FCCM*, 2017.

[8] X. Liu *et al.*, "Efficient sparse-winograd convolutional neural networks," in *ICLR*, 2018.

[9] S. R. Li *et al.*, "Enabling sparse winograd convolution by native pruning," *ArXiv*, vol. abs/1702.08597, 2017.

[10] L. Lu and Y. Liang, "Spwa: An efficient sparse winograd convolutional neural networks accelerator on fpgas," in *DAC*, 2018.

[11] H. Mao *et al.*, "Exploring the granularity of sparsity in convolutional neural networks," in *CVPR*, 2017.

[12] S. Cao *et al.*, "Efficient and effective sparse lstm on fpga with bank-balanced sparsity," in *FPGA*, 2019.

[13] H. Wang *et al.*, "A low-latency sparse-winograd accelerator for convolutional neural networks," in *ICASSP*, 2019.

[14] J. Yu *et al.*, "Spatial-winograd pruning enabling sparse winograd convolution," *ArXiv*, vol. abs/1901.02132, 2018.

[15] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *IJCV*, 2014.

[16] A. Krizhevsky, "Learning multiple layers ofmul features from tiny images," *University of Toronto*, 2012.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[18] C. Szegedy *et al.*, "Going deeper with convolutions," in *CVPR*, 2015.

[19] Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019.