CRSPU: Exploit Commonality of Regular Sparsity to Support Various Convolutions on Systolic Arrays

Jianchao Yang, Mei Wen✉, Junzhong Shen, Yasong Cao, Minjin Tang, Renyu Yang, Xin Ju and Chunyuan Zhang
College of Computer, National University of Defense Technology, Changsha, China
{yangjianchao16, meiwen, shenjunzhong, caoyasong, tangminjin14, yangrenyu, jx, cyzhang}@nudt.edu.cn

***Abstract*—Dilated convolution (DCONV) and transposed convolution (TCONV) are involved in the training of GANs and CNNs and introduces numerous regular zero-spaces into the feature maps or kernels. Existing accelerators typically pre-reorganize the zero-spaces, and then perform sparse computation to accelerate them, resulting in huge hardware resource overhead and control complexity. While the systolic array has proven advantages when it comes to accelerating convolutions, counter-measures for deploying DCONV and TCONV on systolic arrays are rarely proposed. Therefore, we opt to improve the traditional im2col algorithm to make full use of the regular sparsity and avoid data reorganization, thereby facilitating the use of systolic arrays in this context. Public Dimension Compression and Similar Sparsity Merging mechanisms are also designed to implement sparse computing, eliminating unnecessary computing caused by zero-spaces. We propose a systolic array-based processing unit, named CRSPU. Experiments show that CRSPU exhibits more competitive performance than the state-of-the-art baseline accelerator GANPU. Furthermore, CRSPU's ability to avoid zero-space data reorganization represents a huge advantage for bandwidth-unfriendly accelerators.**

***Index Terms*—convolutions, im2col, systolic array**

I. Introduction

Convolutional neural networks (CNNs) and generative adversarial networks (GANs) have been widely deployed in the fields of image classification, image super-resolution, and video prediction. The kernel operation during the procedure of inference and training of CNNs and GANs will unavoidably involve convolution (CONV), dilated convolution (DCONV) and transposed convolution (TCONV), as detailed in TABLE I. Different from the downsampling CONV, DCONV and TCONV insert large numbers of zeros into the feature map and the convolving kernel (see Fig. 1), thereby realizing upsampling and increasing the receptive field size in cost-efficient ways respectively. Notably, these three types of convolution require complicated computation and large amounts of memory, resulting in significant resource overhead and power consumption. However, the large numbers of inserted zeros involved in DCONV and TCONV further aggravate this problem. Previous accelerators have realized the acceleration of DCONV and TCONV by supporting sparse computation.

The CNN accelerators SIGMA [1], along with the GAN accelerators FlexiGAN [2] and Kn2row [3], all of which support sparse computation, can realize zero-skipping computation by inserting zeros into the input feature map or kernel in advance. However, hardware that supports zero-skipping generally requires indexes, while the complexity of the data preprocessing involved is difficult to avoid introducing resource and power overheads [4]. In addition, some GAN accelerators [2], [3], [5], [6] do not fully exploit the prior knowledge of the regular sparsity(RS; sparsity introduced by regularly inserted zeros) of DCONV and TCONV. Moreover, the perceptual zero-skipping greatly increases the computational delay. DT-CNN [6] and GANPU [7] achieve imprecise computation by skipping the Multiply-Accumulate operations (MACs); as a result of this, the input or output feature map (IMP or OMP) is predicted to involve zeros, resulting in reduced inference accuracy. The cold buffer of GNA [8] is used to handle the overlap of partial sums without a zero-skipping mechanism, which not only leads to higher hardware overhead during data preprocessing, but also increases the complexity of control. TDC [9] attempts to convert TCONV into CONV, but the insertion of zeros in the weight blocks leads to unbalanced calculation load. F-DNA [10] requires more complex overall logic than TDC to eliminate the unbalanced calculation load. In addition, due to the complexity of zero-skipping logic, most GAN accelerators [3], [6], [11], [12] have no or only partial data multiplexing in their PE (processing element) arrays, with some even adopting broadcast mode [6], [11], [12], resulting in dramatically increased bandwidth requirements and low PE utilization. Most importantly, the zero pre-insertion computing mode requires users to be familiar with the underlying algorithm, which makes it difficult to build a complete accelerator ecosystem. Basically, the GAN accelerators cannot make full use of data multiplexing and improve PE utilization, for the reason that they mostly adopt direct-convolution to accelerate DCONV and TCONV.

CNN accelerators [1] have effectively proved that converting CONV into GEMM through im2col and mapping it on systolic arrays can reduce the bandwidth and memory resources required. Moreover, systolic arrays increase the on-chip resident time of the data, thereby increasing the PE utilization. Since im2col+GEMM has a high degree of coupling to the data flow, the zero-skipping computation is not suitable for direct mapping on a systolic array; in addition, simply designing three sets of hardware to support CONV, DCONV and TCONV respectively results in seriously inefficient resource utilization. Therefore, it is necessary and urgent to combine the RS characteristics of the three convolutions, the high data multiplexing of the systolic array, and the optimization of implicit im2col for collaborative design. Our main contributions can

* Supported by National Nature Science Foundation of China under NCM No. 61802420 and 62002366.
✉ Corresponding Author.

978-3-9819263-7-8/DATE23/© 2023 EDAA

TABLE I
CONVOLUTION TYPES INVOLVED IN INFERENCE AND TRAINING OF DNNS.

	Feedforward	Backpropagation	Weight Updating
CNN	CONV	TCONV	DCONV
GAN	DCONV	DCONV	CONV
	TCONV	CONV	TCONV

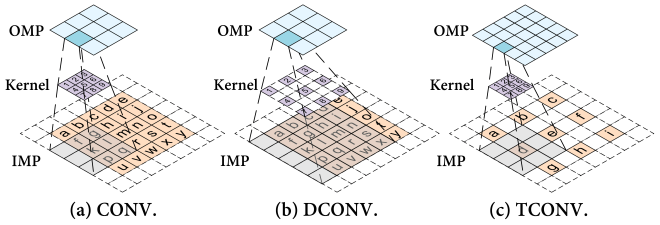


Fig. 1. Convolution operations of single-channel IMP and OMP.

be summarized as follows:

- We summarize the challenges faced by existing sparse accelerators in supporting DCONV and TCONV, and exploit the commonality of their regular sparsity during im2col to propose a systolic array-based accelerator, CRSPU.
- We deploy CRSPU with Public Dimension Compression and Similar Sparsity Merging mechanisms to skip the invalid calculations caused by regular sparsity, while utilizing the high data reusability of the systolic array.
- Compared with sparse GANPU, CRSPU achieves a 8.43× speedup in the training performance of CycleGAN, and the PE efficiency is increased by 44.23×. Additionally, it reduces the on-chip bandwidth required by DCONV and TCONV by up to 31.66% and 54.63%, respectively. CRSPU also has huge advantages in PE utilization and memory savings.

For clarity, TABLE II lists explanations for the notations used in this paper.

TABLE II
EXPLANATION OF THE NOTATIONS.

Notation	Explanation
$K_h \times K_w$	Kernel size of convolutions.
B, C, N	Batch size, the input and output channel of convolutions.
$O_h \times O_w$	Height and width of the OMP.
δ	Dilation coefficient of DCONV.

II. COMMONALITY REDUCES COMPLEXITY

Due to the high data coupling of the systolic array, convolutions need to be converted into GEMM through im2col. Fig. 2(a) shows that traditional im2col mainly supports the feedforward in CONV. To support DCONV and TCONV, it must insert zeros in the IMP, such that the sparsity reaches 75% with a stride of 2 (see TABLE IV). Furthermore, the intensive memory access involved incurs huge delays that cannot be sufficiently hidden during computation. Our previous work, BP-im2col [4], improved the im2col algorithm by avoiding zero-insertion and restoring zeros during computation by means of on-chip address mapping units. However, the units cause a 10× increase in area overhead while failing to solve the core problem, namely the low throughput and high energy consumption of accelerators incurred by invalid calculations.

Fig. 1 illustrates the regular sparsity in the IMP and kernel. Here, the sliding order of the kernel contained in the im2col

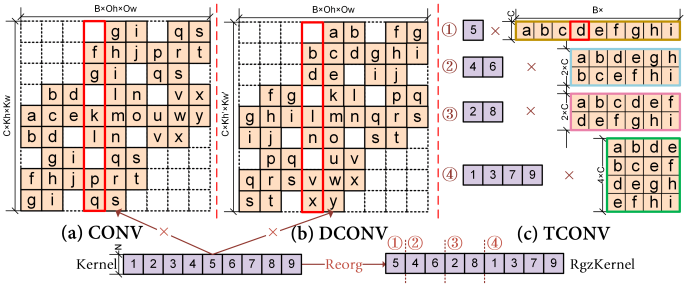


Fig. 2. Im2col with zero-skipping mechanism. All inserted zeros have been eliminated, and the white squares in the figure are all padded zeros. The red boxes corresponds to the sliding window in Fig. 1.

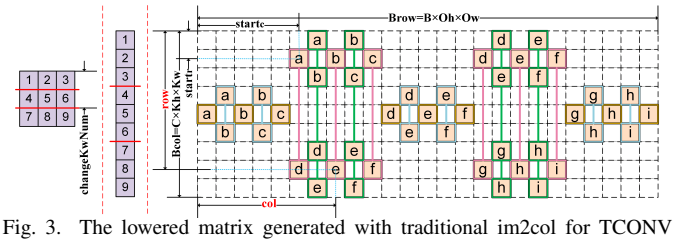


Fig. 3. The lowered matrix generated with traditional im2col for TCONV shown in Fig. 1(c). The elements marked by the four differently colored boxes are divided into four groups shown in Fig. 2(c).

process will extend this regular sparsity to the lowered matrix. This rule can be explained as follows: (1) the lowered matrix of DCONV generates large amounts of invalid calculations in the public dimension of GEMM due to the regular zero distribution of the kernel; (2) the lowered matrix of TCONV yields many similarities that can be merged in the column direction, while the merged sub-matrices also contain large numbers of invalid calculations in their public dimension. With this feature in mind, we design the following mechanisms to exploit the commonality of the lowered matrices in an attempt to reduce the complexity of the calculation.

A. Public Dimension Compression (PDC)

The lowered matrix of DCONV obtained by traditional im2col will contain numerous invalid calculations across the entire row of elements, because it will perform multiplication involving the entire column of zeros in the kernel that is stretched into N rows. The zeros of the entire column of the stretched kernel are all located in the public dimension of GEMM. Therefore, we skip the entire column of zeros in the public dimension of the stretched kernel and the corresponding row of elements in the lowered matrix, thereby ensuring we can complete the compression of the invalid calculation in the public dimension. In fact, the skipped elements are eliminated from the entire row of the lowered matrix, and the *dilation* coefficient of DCONV allows us to solve this problem at the algorithm level. Compared to accelerators that support sparse computation, the mechanism of Public Dimension Compression can be easily incrementally scaled on traditional CONV-enabled im2col, as detailed in Algorithm 1.

B. Similar Sparsity Merging (SSM)

TCONV has more complex sparsity distribution, as its IMP needs to execute the im2col and be inserted with zeros simultaneously, as shown in Fig. 3. Therefore, Public

Algorithm 1: Im2col with zero-skipping mechanism.

Input: Position of a pixel in lowered matrix, row and col.
Output: Address in the original IMP, $addr_{out}$.

$b, temp_1 = \lfloor col/(O_h \cdot O_w) \rfloor, col\%(O_h \cdot O_w)$;
$||temp_2, w_k = \lfloor row/K_w \rfloor, row\%K_w$; // Parallel with 1.
$c, h_k = \lfloor temp_2/K_h \rfloor, temp_2\%K_h$;
$||h_I, w_I = \lfloor temp_1/Ow \rfloor, temp_1\%Ow$; // Parallel with 3.
if *CONV or DCONV* **then**
  $\delta = 1$ **if** *CONV* **else** *dilation* ; // Zero-Skipping.
  $\Delta_h = h_I \cdot S + h_k \cdot \delta - P_h$ and $\Delta_w = w_I \cdot S + w_k \cdot \delta - P_w$;
  if $\Delta_h < 0$ *or* $\Delta_h >= I_h$ *or* $\Delta_w < 0$ *or* $\Delta_w >= I_w$ **then**
    $addr_{out} = NULL$; // Invalid Padded Zeros.
  else
    $channel_{offset} = b \cdot C \cdot I_h \cdot I_w + c \cdot I_h \cdot I_w$;
    $||img_{offset} = \Delta_h \cdot I_w + \Delta_w$; // Parallel with 11.
    $addr_{out} = channel_{offset} + img_{offset}$;
  end
else if *TCONV* **then**
  $\Delta_h = h_I + h_k - P_h$ and $\Delta_w = w_I + w_k - P_w$;
  if $\Delta_h < 0$ *or* $\Delta_h >= I_h$ *or* $\Delta_w < 0$ *or* $\Delta_w >= I_w$ **then**
    $addr_{out} = NULL$; // Invalid Padded Zeros.
  else
    if $\Delta_h \% S \neq 0$ *or* $\Delta_w \% S \neq 0$ **then**
      $addr_{out} = NULL$; // Invalid Inserted Zeros.
    else
      $channel_{offset} = b \cdot C \cdot I_h \cdot I_w + c \cdot I_h \cdot I_w$;
      $||img_{offset} = \frac{\Delta_h}{S} \cdot I_w + \frac{\Delta_w}{S}$; // Zero-Skipping.
      $addr_{out} = channel_{offset} + img_{offset}$;
    end
  end
end

Dimension Compression is no longer suitable for TCONV. However, noted that the sparsity distribution in the lowered matrix generated by TCONV through traditional im2col (see Fig. 3) is extended from that of the original feature map. For TCONV with a stride of S, all columns in the lowered matrix can be divided into S^2 groups, at which time the zeros in all columns of each group are distributed over the entire row. The columns marked with the same color in Fig. 3 all belong to the same subgroup. The zeros within the subgroups are distributed in the public dimension, which provides the possibility of performing Public Dimension Compression; the compressed subgroup series is shown in Fig. 2(c). Since the column order in the lowered matrix is changed, that of the kernel must also be reorganized accordingly. The reorganized kernel and lowered matrix perform the grouping matrix multiplication to calculate the OMP. The calculated OMP then needs to perform reorganization before being used as the input of the next layer. The order of reorganization is opposite to the order of merging of the lowered matrices, which we will introduce in Section III.

III. Proposed Architecture

A. System Architecture

The architecture of the proposed CRSPU (illustrated in Fig. 4) consists of a systolic array, a shared memory, and a host processor. The accelerator consists of a master control unit, a computing configuration unit, 6 MB on-chip buffers (the IMP buffer, kernel buffer, and OMP buffer are all 1 MB dual-buffers), the local router, two address generation

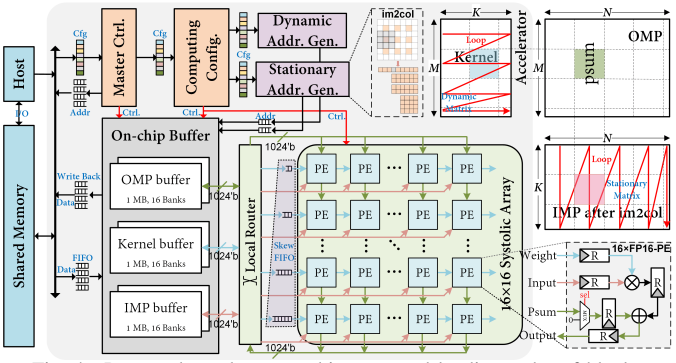


Fig. 4. Proposed accelerator architecture and loading order of blocks.

units, and a 16×16 PE array. The master control unit is responsible for activating the computing configuration unit and controlling data transmission off- and on-chip. The computing configuration unit generates the configurations required for CRSPU according to the parameters of convolutions issued by the master control unit. The PE array adopts a tight coupled input-stationary data flow, which ensures very good reusability. The two address generation units execute the implicit im2col calculation, along with the address issuance of the stationary and dynamic matrices. The addresses of the partial-sum matrix are directly generated by the OMP buffer.

The total accelerator is managed by the master control unit. Instructions can be divided into configuration and execution, both of which are obtained from the host through I/O and decoded in the master control unit, which in turn determines when the accelerator will start and execute. The configuration instruction is transmitted to the computing configuration unit, after which the mode of computing is configured. At the same time, the buffers begin to pre-fetch data, the address generation units start to generate addresses, and the PE array will load data from the buffers for calculation. Rather than skewing the address of the dynamic matrix, we design 16 FIFOs with different depths between the kernel buffer and the PE array to skew the data layout. The implementation of Public Dimension Compression and Similar Sparsity Merging is executed by the cooperation of the master control unit, the calculation configuration unit, and the address generation units. The inserted zeros will not be stored in buffers or loaded into the PE array to participate in the calculation, which enables the acceleration effect to be achieved. To ensure that DCONV and TCONV can fetch the correct non-zeros from on-chip buffers without inserting zeros into the kernel or IMP, we improve the traditional im2col algorithm, as detailed in Algorithm 1.

B. PDC and SSM Mechanisms

The address generation units focus on generating the addresses of data that participate in valid calculations. For the DCONV-oriented PDC mechanism, the data of the lowered matrix is fetched and sent to the PE array; however, it should be noted that only the data involved in valid calculations is stretched into one column of the lowered matrix. As shown in Fig. 5, the im2col process (see Algorithm 1) is controlled to skip the generation of invalid addresses via the dilation

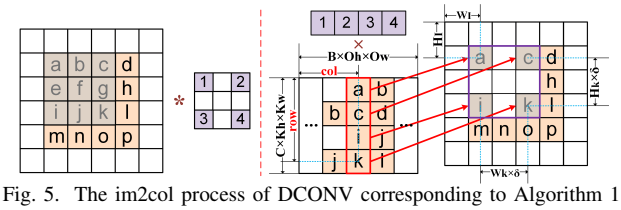


Fig. 5. The im2col process of DCONV corresponding to Algorithm 1.

coefficient of DCONV. The kernel can be directly stretched into N rows to perform GEMM with the lowered matrix. The traditional im2col only increases the dilation coefficient δ to support DCONV, which has almost no hardware overhead.

As the SSM mechanism changes the order of GEMM for TCONV, the address generation logic is more complex. The address generation of S^2 sub-matrices is detailed in Algorithm 2, while several of the important symbols used in this algorithm are defined in Fig. 3. Once a valid address is generated, the im2col algorithm is called to map it to the on-chip buffer. Note that while generating the address of the sub-matrix, the tuple (row, col) that identifies the position of non-zeros (see Algorithm 2) is the absolute position in the lowered matrix generated by the traditional im2col, which we have marked in Fig. 3 up to this point. An important advantage of this approach is that the hardware can write the partial sum calculated on the PE array for each sub-matrix back to the correct OMP buffer address based on this absolute position. This not only avoids repeating the logic of address calculation in the OMP buffer, but also greatly reduces the latency of data write-back. While ensuring that the address generation logic of TCONV is independent will generate additional hardware and control overhead, this is acceptable in light of the speedup ratio brought about by the huge sparsity.

Algorithm 2: Similar Sparsity Merging for TCONV.

Input: The size of the stationary lowered matrix (B_{row}, B_{col}).

for _ *in range*(S^2) **do**
  $rS,\ cS,\ channel_{id}$ = $\lfloor \_/S \rfloor,\ \_\%S,\ 1$;
  tmp_1 = 0 **if** $(rS == 0)$ **else** $(S - rS)$;
  tmp_2 = 0 **if** $(cS == 0)$ **else** $(S - cS)$;
  $start_r,\ start_c$ = $tmp_1 \cdot K_w + tmp_2,\ rS \cdot O_w + cS$;
  $row,\ col,\ R_{Kw}$ = $start_r,\ start_c,\ start_r\%K_w$;
  $changeKwNum,\ C_{Ow}$ = $\lfloor start_r/K_W \rfloor,\ start_c\%O_w$;
  while *not* $(row \geq B_{row})$ **do**
    while *not* $(col \geq B_{col})$ **do**
      Call Im2col(row, col); // Call Algorithm 1.
      if $C_{Ow} + S \geq O_w$ **then**
        col += $O_w - C_{Ow} + (S-1) \cdot O_w + start_c\%O_w$;
        $C_{Ow} = start_c\%O_w$;
      else
        $col,\ C_{Ow}$ = $col + S,\ C_{Ow} + S$;
      end
    end
    $col = start_c$;
    if $R_{Kw} + S \geq K_w$ **then**
      row += $K_w - R_{Kw} + (S-1) \cdot K_w + start_r\%K_w$;
      $R_{Kw} = start_r\%K_w$;
      $changeKwNum = changeKwNum + S$;
      if $changeKwNum \geq K_h$ **then**
        $changeKwNum = \lfloor start_r/K_w \rfloor$;
        $row = channel_{id} \cdot K_h \cdot K_w + start_r$;
        $channel_{id} = channel_{id} + 1$;
      end
    else
      $row,\ R_{Kw}$ = $row + S,\ R_{Kw} + S$;
    end
  end
end

IV. Implementation Results

CRSPU was synthesized with a 7 nm predictive PDK library, and it occupies a chip area of 3.78 mm^2. It operates at a 2 GHz clock frequency with 0.7 V supply voltage, consuming 5395.8 mW of power. CRSPU is equipped with 256 64-bit wide cross-multiply FP16-PEs, and the instruction throughput of each PE is 16/cycle. Note that the performance evaluations of all other accelerators below do not include the time required to pre-insert zeros into data; for its part, CRSPU does not require pre-insertion of zeros.

A. Performance Comparison

TABLE III shows the peak performance comparison of several start-of-the-art accelerators that support DCONV or TCONV based on the FPGA or ASIC platforms. Compared with the direct convolution (or Kn2row) route adopted by other accelerators, CRSPU uses im2col+GEMM (with zero-skipping mechanism) to map CONV, DCONV, and TCONV onto the systolic array. The peak throughput of CRSPU achieves 7736 GOPS without computing the values of the virtual zeros included by DCONV and TCONV, which is higher than all other compared accelerators except for NVIDIA V100 GPU. Moreover, the area efficiency of CRSPU is as high as 2046.56 GOPS/mm^2, which is much higher than NVIDIA V100 GPU and other accelerators based on the ASIC platform. Furthermore, the energy efficiency of CRSPU reaches 1.43 GOPS/W, indicating obvious advantages over all FPGA-based accelerators. In addition, CRSPU has the highest PE efficiency of 30.22 GOPS/PE among all accelerators.

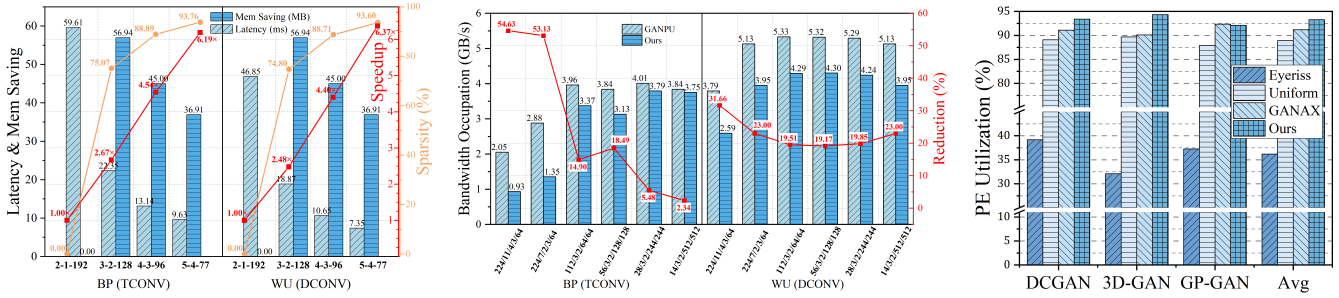
TABLE IV shows the benchmark configuration on the training sequence of CycleGAN [13] with a batch size of 4 and FP16 data type. GANPU [7] provides two versions of the performance evaluation on CycleGAN, which support dense and sparse computing respectively. The performance comparison between CRSPU and GANPU is illustrated in TABLE V. Compared with the dense and sparse versions of GANPU, the performance of CRSPU is significantly improved: more specifically, CRSPU exhibits an average performance improvement of 11.20× and 8.43× during training, along with an average improvement of 58.80× and 44.23× on the PE efficiency. The PE efficiency can also reach 5.88× and 4.42× with frequency normalized.

TABLE VI presents the performance comparison between CRSPU and GPU; the results on GPU are obtained on CUDA Cores with a batch size of 16. Due to the substantial disparity in the quantity of PEs, the V100 achieves a significant speedup compared to CRSPU. Notably, however, the number of CUDA cores in the V100 is 5120, and its instruction throughput of FP-16 data type is 256/cycle per core, while CRSPU has only 256 PEs and its instruction throughput is 16/cycle per PE. In

TABLE III
PEAK PERFORMANCE COMPARISON WITH STATE-OF-THE-ART ACCELERATORS.

	[14] TRETS'18	[9] TCSVT'18	[12] FPL'19	[10] VLSI'20	[15] J.Imaging'21	[3] HiPC'21	NVIDIA V100 GPU	[8] TCAD'18	[6] ISCAS'19	[11] SSCL'19	[5] VLSI'20	[7] JSSC'21	Ours
Route	Direct	Direct	Direct	Direct	Direct	Kn2row	Multiple	Direct	Direct	Direct	Direct	Direct	Im2col
Training	✗	✗	✗	✗	✗	✗	✓	✓	✗	✓	✗	✓	✓
Platform	FPGA	FPGA	FPGA	FPGA	FPGA	FPGA	Software	ASIC	ASIC	ASIC	ASIC	AISC	ASIC
Zero-skipping	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓
Support C/D/T	C/T	C/T	C/T	C/T	C/T	C/T	C/D/T	C/T	C/D/T	C/D/T	C/D/T	C/D/T	C/D/T
Technology (nm)	\	\	\	\	\	\	7	28	65	65	65	65	7
Area (mm^2)	\	\	\	\	\	\	826	1.387	6.8	0.703	2.56	32.44	3.78
Power (mW)	9600	5380	32000	3710	4000	9000	300000	142	196	20.3	50.1	647	5395.8
Frequency (MHz)	200	130	200	200	250	300	1410	200	200	200	200	200	2000
Voltage (V)	\	\	\	\	\	\	\	0.9	1.2	1.1	1.2	1.1	0.7
Throughput (GOPS)	107	780	1578	824	721	1125	624000	410	640	38	196	538	7736
Area Efficiency (GOPS/mm2)	\	\	\	\	\	\	755.44	295.31	94.07	53.34	76.64	16.58	2046.56
Power Efficiency (TOPS/W)	0.01	0.15	0.05	0.22	0.19	0.13	2.08	2.88	3.26	1.84	3.92	0.83	1.43
Total PEs	900	1512	1688	576	1296	2176	5120	256	60	48	128	1344	256
PE Efficiency (GOPS/PE)	0.12	0.52	0.93	1.43	0.56	0.52	3.81	1.60	10.66	0.78	1.53	0.40	30.22
Precision (W-A)	INT 16	INT 13	INT 8	INT 16-8	INT 16-10	INT 16	FP 16	INT 8	INT 8	FP 8/16 mix	INT 2-16	FP 16	FP 16

* Note: 1. *Direct* represents direct convolution; *CDT* represents convolution, dilated convolution and transposed convolution, respectively.
2. CRSPU uses 64-bit wide cross-multiply PEs, which have not been optimized, resulting in excessive power.



(a) Performance and memory savings. (b) Reduced bandwidth occupation. (c) Average PE utilization.

Fig. 6. (a) Effect of regular sparsity on performance and memory savings. The configuration in the figure represents K-S-C$_o$. The IMP size and number of input channels are 256 and 64 respectively, while the total amount of operation is all 25.77 GOP. (b) The reduced bandwidth occupation of TCONV and DCONV performed on the CRSPU accelerator. The parameter of convolutional layers represents I-K-S-C$_i$-C$_o$. (c) Average PE utilization for GAN models.

TABLE IV
LAYERS OF PERFORMANCE COMPARISON FROM CYCLEGAN [13].

Layer	Configuration (I-K-S-Ci-Co)	Dense OPs (GOP) FF	BP	WU	Regular Sparsity (%) FF	BP	WU
CONV 0	256-7-1-3-64	4.93	4.93	4.93	1.33	1.33	1.33
CONV 1	256-3-2-64-128	9.66	38.65	38.65	1.04	75.07	75.07
CONV 2	128-3-2-128-256	9.66	38.65	38.65	2.07	75.13	75.13
TCONV 3	64-3-2-256-128	38.65	9.66	38.65	75.13	2.07	75.13
TCONV 4	128-3-2-128-64	38.65	9.66	38.65	75.07	1.04	75.07
CONV 5	256-7-1-64-3	4.93	4.93	4.93	1.33	1.33	1.33

theory, the V100 should achieve a peak speedup of $\frac{5120\times256}{256\times16} = 320\times$. However, the average speedup of the V100 relative to CRSPU during feedforward and training is only 51.76× and 37.33× respectively. Accordingly, the PE efficiency of CRSPU is higher than that of V100 GPU, even though V100 may not be able to run at full capacity on our selected benchmarks.

B. Effect of Regular Sparsity on Performance

Fig. 6(a) shows the effect of regular sparsity on the performance and memory-saving capacity of CRSPU. We have chosen CNN convolutional layers with different strides to perform backpropagation (TCONV) and weight updating (DCONV) with a batch size of 4 and FP16 data type. Different strides provide different regular sparsity: as the stride increases, the sparsity of TCONV and DCONV also increases. At the same time, the suitable number of output channels ensures all layers that perform backpropagation (TCONV) and weight updating (DCONV) have the same amount of operations. Fig. 6(a) shows that with the increase of sparsity, the speedup of TCONV and DCONV on the CRSPU accelerator gradually increases; when the sparsity is about 75%, 88%, and 93%, the speedup can reach 2.48×, 4.40×, and 6.19×, respectively. The memory saving is mainly due to the inserted zeros of the loss feature map, which means it is affected by both the output channel and the stride; the proportion of memory saved by each convolutional layer is approximately equal to its regular sparsity.

C. Reduced Bandwidth Occupation

Fig. 6(b) shows the bandwidth occupation of CRSPU when performing the backpropagation (TCONV) and weight updating (DCONV) processes of some common CNN convolutional layers. The techniques adopted by CRSPU to avoid the need to transmit large numbers of zeros reduce the bandwidth requirements of on-chip buffers. Compared with GANPU, CRSPU reduces the bandwidth occupation of the on-chip stationary matrix buffer by 2.34%∼54.63% during

TABLE V
PERFORMANCE COMPARISON WITH GANPU [7] ON CYCLEGAN [13].

	Layer	FF (ms,GOPS/PE)		BP (ms,GOPS/PE)		WU (ms,GOPS/PE)	
		Latency	PE EFF	Latency	PE EFF	Latency	PE EFF
GANPU▲	CONV 0	34.69	0.11	34.74	0.11	42.19	0.09
	CONV 1	78.79	0.09	78.79	0.36	30.76	0.93
	CONV 2	78.93	0.09	78.92	0.36	49.95	0.58
	TCONV 3	78.94	0.36	79.00	0.09	41.80	0.69
	TCONV 4	78.77	0.37	78.85	0.09	53.62	0.54
	CONV 5	30.78	0.12	30.73	0.12	28.82	0.13
	Total	380.9	0.21	381.03	0.21	247.14	0.50
	FF+BP+WU	Latency: 1008.17 ms			PE EFF: 0.28 GOPS/PE		
GANPU▼	CONV 0	34.23	0.11	34.28	0.11	42.19	0.09
	CONV 1	77.97	0.09	19.67	1.46	29.39	0.98
	CONV 2	77.30	0.09	19.63	1.46	48.58	0.59
	TCONV 3	19.63	1.46	77.36	0.09	39.14	0.73
	TCONV 4	19.67	1.46	78.03	0.09	52.53	0.55
	CONV 5	30.37	0.12	30.32	0.12	28.36	0.13
	Total	259.17	0.31	259.29	0.31	240.19	0.51
	FF+BP+WU	Latency: 758.65 ms			PE EFF: 0.23 GOPS/PE		
Ours	CONV 0	2.32 14.94× 14.74×	8.28 75.27× 75.27×	10.24 3.39× 3.35×	1.88 17.09× 17.09×	1.82 23.15× 23.15×	10.56 117.33× 117.33×
	CONV 1	4.41 17.87× 17.68×	8.56 95.11× 95.11×	4.27 18.45× 4.61×	35.36 98.22× 24.22×	3.74 8.22× 7.86×	40.36 43.40× 41.18×
	CONV 2	5.43 14.54× 14.24×	6.96 77.33× 77.33×	4.28 18.43× 4.58×	35.24 97.89× 24.14×	4.03 12.39× 12.05×	37.44 64.55× 63.46×
	TCONV 3	4.28 18.43× 4.58×	35.24 97.89× 24.14×	5.46 14.48× 14.17×	6.92 76.89× 76.89×	3.57 11.71× 10.96×	42.28 61.28× 57.92×
	TCONV 4	4.27 18.44× 4.61×	35.36 95.57× 24.22×	4.44 17.76× 17.57×	8.48 94.22× 94.22×	3.85 13.94× 13.65×	39.24 72.67× 71.35×
	CONV 5	10.21 3.02× 2.98×	1.88 15.67× 15.67×	2.31 13.30× 13.12×	8.32 69.33× 69.33×	11.10 2.60× 2.56×	1.72 13.23× 13.23×
	Total	30.92 12.32× 8.38×	20.76 98.86× 66.97×	31.00 12.29× 8.36×	20.72 98.67× 66.84×	28.11 8.79× 8.54×	22.84 45.68× 44.78×
	FF+BP+WU	Latency: 90.03 ms 11.20×, 8.43×			PE EFF: 16.38 GOPS/PE 58.80×, 44.23×		

* Note: ▲ represents dense version of GANPU; ▼ represents sparse version of GANPU, with only inserted zeros (regular sparsity) are skipped.

TABLE VI
PERFORMANCE COMPARISON WITH NVIDIA V100 GPU.

	NVIDIA V100 GPU (ms)				Ours (ms)	
	FF	Speedup	BP+WU	Speedup	FF	BP+WU
CONV 0	0.22	42.06×	0.81	59.55×	9.29	48.25
CONV 1	0.56	31.49×	0.94	34.03×	17.64	32.04
CONV 2	0.46	47.61×	1.36	24.53×	21.71	33.26
TCONV 3	0.25	68.68×	1.34	26.98×	17.13	36.11
TCONV 4	0.45	37.70×	0.97	34.33×	17.08	33.15
CONV 5	0.45	90.79×	0.92	58.12×	40.82	53.63
Total	2.39	51.76×	6.33	37.33×	123.67	236.44

the backpropagation process; moreover, during the weight updating process, CRSPU reduces the bandwidth occupation of the on-chip dynamic matrix buffer by 19.17%∼31.66%. In general, the effect of reducing the bandwidth occupation of on-chip buffers becomes more obvious as the stride of the convolutional layers increases.

D. PE Utilization

Fig. 6(c) presents a comparison in terms of PE utilization between Eyeriss [16], Uniform [17], GANAX [18], and CRSPU when performing the training of DCGAN, 3D-GAN, and GP-GAN. As a traditional CNN accelerator, Eyeriss has no ability to deal with zero-skipping computation; moreover, the GAN accelerators Uniform and GANAX adopting a direct convolution route does not fully utilize the regular sparsity of TCONV and DCONV, resulting in a lower PE utilization than CRSPU. The average PE utilization of CRSPU can reach 93.27% when performing training of the above GAN models.

V. CONCLUSION

This paper proposes a co-design for accelerating various convolutions based on systolic arrays, while analyzing the shortcomings of previous sparse accelerators when it comes to accelerating DCONV and TCONV. The proposed accelerator, CRSPU, improves the traditional im2col to avoid the reorganization of zero-spaces. PDC and SSM mechanisms make full use of the regular sparsity; thus, CRSPU can skip invalid calculations during im2col. Compared with state-of-the-art accelerators, CRSPU facilitates significant improvements in the training performance of various models, and also offers the significant advantages in reducing the bandwidth of on-chip buffers and saving on-chip memory.

REFERENCES

[1] E. Qin *et al.*, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *HPCA*, 2020.

[2] A. Yazdanbakhsh *et al.*, "Flexigan: An end-to-end solution for fpga acceleration of generative adversarial networks," in *FCCM*, 2018.

[3] Y. Meng *et al.*, "How to avoid zero-spacing in fractionally-strided convolution? a hardware-algorithm co-design methodology," in *HiPC*, 2021.

[4] J. Yang *et al.*, "Bp-im2col: Implicit im2col supporting ai backpropagation on systolic arrays," in *ICCD*, 2022.

[5] Q. Chen *et al.*, "An efficient accelerator for multiple convolutions from the sparsity perspective," *IEEE Trans. VLSI Syst.*, 2020.

[6] D. Im *et al.*, "Dt-cnn: Dilated and transposed convolution neural network accelerator for real-time image segmentation on mobile devices," in *ISCAS*, 2019.

[7] S. Kang *et al.*, "Ganpu: An energy-efficient multi-dnn training processor for gans with speculative dual-sparsity exploitation," *IEEE J. Solid-State Circuits*, 2021.

[8] J. Yan *et al.*, "Gna: Reconfigurable and efficient architecture for generative network acceleration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2018.

[9] J.-W. Chang *et al.*, "An energy-efficient fpga-based deconvolutional neural networks accelerator for single image super-resolution," *IEEE Trans. Circuits Syst. Video Technol.*, 2020.

[10] W. Mao *et al.*, "F-dna: Fast convolution architecture for deconvolutional network acceleration," *IEEE Trans. VLSI Syst.*, 2020.

[11] J. Lee *et al.*, "An energy-efficient sparse deep-neural-network learning accelerator with fine-grained mixed precision of fp8–fp16," *IEEE J. Solid-State Circuits*, 2019.

[12] S. Liu *et al.*, "Towards an efficient accelerator for dnn-based remote sensing image segmentation on fpgas," in *FPL*, 2019.

[13] J.-Y. Zhu *et al.*, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *ICCV*, 2017.

[14] S. Liu *et al.*, "Optimizing cnn-based segmentation with deeply customized convolutional and deconvolutional architectures on fpga," *ACM Trans. Reconfigurable Technol. Syst.*, 2018.

[15] C. Sestito *et al.*, "Design of flexible hardware accelerators for image convolutions and transposed convolutions," *J. Imaging*, 2021.

[16] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *ISSCC*, 2016.

[17] D. Wang *et al.*, "Towards a uniform architecture for the efficient implementation of 2d and 3d deconvolutional neural networks on fpgas," in *ISCAS*, 2019.

[18] A. Yazdanbakhsh *et al.*, "Ganax: A unified mimd-simd acceleration for generative adversarial networks," in *ISCA*, 2018.