Search-free Accelerator for Sparse Convolutional Neural Networks

Bosheng Liu*+, Xiaoming Chen*+#, Yinhe Han*#, Ying Wang*, Jiajun Li*+, Haobo Xu*+, and Xiaowei Li*
*State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, China
+University of Chinese Academy of Sciences, Beijing, China
#Corresponding authors (e-mail: chenxiaoming@ict.ac.cn, yinhes@ict.ac.cn)

***Abstract*—Sparsification is an efficient solution to reduce the demand of on-chip memory space for deep convolutional neural networks (CNNs). Most of state-of-the-art CNN accelerators can deliver high throughput for sparse CNNs by searching pairs of nonzero weights and activations, and then sending them to processing elements (PEs) for multiplication-accumulation (MAC) operations. However, their PE scales are difficult to be increased for superior and efficient computing because of the significant internal interconnect and memory bandwidth consumption. To deal with this dilemma, we propose a sparsity-aware architecture, called Swan, which frees the search process for sparse CNNs under limited interconnect and bandwidth resources. The architecture comprises two parts: a MAC unit that can free the search operation for the sparsity-aware MAC calculation, and a systolic compressive dataflow that well suits the MAC architecture and greatly reuses inputs for interconnect and bandwidth saving. With the proposed architecture, only one column of the PEs needs to load/store data while all PEs can operate in full scale. Evaluation results based on a place-and-route process show that the proposed design, in a compact factor of 4096 PEs, 4.9TOP/s peak performance, and 2.97W power running at 600MHz, achieves 1.5-2.1× speedup and 6.0-9.1× higher energy efficiency than state-of-the-art CNN accelerators with the same PE scale.**

***Index Terms*—Sparse convolution neural networks, sparsity-aware CNN accelerator, internal interconnect, memory bandwidth.**

I. Introduction

With continuous breakthroughs in deep learning, convolutional neural networks (CNNs) [1]–[4] have been widely applied in numerous computer version tasks such as image classification and recognition. Recently, instead of pursuing accuracy improvements, researchers, who concern the applicability of CNNs in mobile or embedded systems, are investigating model compression based on neural network pruning techniques [5]–[7], which not only significantly reduce the memory storage but also succeed in preserving inference accuracy. For example, Han et al. [5] achieves up to 35× model size reduction for the representative Alexnet model while preserving a top-5 accuracy of 80.3%. To fully utilize sparse CNN models, it is expected to rely on sparsity-aware hardware engines, especially those based on the low-power application-specific integrated circuits (ASICs), to get high throughput and energy efficiency.

Several dedicated accelerators have been proposed for sparse CNN accelerations. For example, Cambricon-X, EIE, Cnvlutin, and SCNN are typical sparsity-aware accelerators that support sparse weights and/or activations [8]–[11], as listed in Table I. Compared with the non-sparsity-aware accelerators [12], [13], the sparsity-aware ones can efficiently boost inference performance by eliminating unnecessary data storage, movement, and computation in sparse CNNs. For example, Cambricon-X [9] achieves 7.2× speedup over Diannao [12] by exploiting the sparsity of weights, and Cnvlutin [10] exploits activation sparsity and achieves 1.37× better throughput than Dadiannao [14]. Since input activations and/or weights are compressed and accommodated in on-chip memories in an irregular fashion, prior accelerators need to search pairs of nonzero weights and activations from all multiplication-accumulation (MAC) operations. They use customized search architectures to find nonzero patterns, and then send them to processing elements (PEs) for MAC operations, which receive multifold benefits:

–Sparsity-aware acceleration engines exhibit high throughput by fully utilizing the PE resource, because the nonzero patterns are detected before the MAC operations.

–The search process for nonzero patterns can be implemented in constant cycles. For example, Cambricon-X utilizes an indexing module (IM) to retrieve activations for the compressed weights in only one cycle.

Nevertheless, the scheme of customizing search architectures makes their PE scale difficult to be increased because of the significant interconnect and memory bandwidth consumption. This is because all PEs need to be fed with pairs of nonzero weights and activations concurrently. For example, the demand of high-bandwidth on-chip storage and interconnect makes Cnvlutin seek solutions from expensive embedded dynamic random-access memory (eDRAM) and high-speed interconnect, which makes them difficult to be suited to resource-limited embedded systems. To sum it up, existing sparse CNN accelerators cannot provide superior and efficient accelerations for resource-limited systems.

In contrast, this work builds a search-free sparsity-aware accelerator called Swan. Since weights are pre-trained, they are

TABLE I
Recent representative accelerators.

Accelerator	On-chip memory type	weight/activation (W/A) sparsity	PE #
Cambricon-X	SRAM	W	256
EIE	SRAM	W+A	64
SCNN	SRAM	W+A	1024
Dadiannao	eDRAM+SRAM	–	4096
Cnvlutin	eDRAM+SRAM	A	4096
Swan	SRAM	W+gated A	4096

978-1-7281-4123-7/20/$31.00 © 2020 IEEE

compressed offline, while activations are produced on-the-fly so we do not compress them to avoid the runtime compression overhead. To achieve the best efficiency by fully utilizing the potential brought by both sparse weights and activations, we propose a novel strategy that skips zero weights and gates zero input activations. We develop a novel sparsity-aware MAC unit that frees the search process from the sparsity-aware MAC operations. The key technique to avoid search is that the offset indexes of the compressed weights are utilized as the trigger signals to pair the corresponding (uncompressed) input activations directly. We then introduce a systolic compressive dataflow that well suits the proposed MAC unit and reuses input activations and compressed weights for interconnect and bandwidth saving. Our work is a significant advance over existing sparse CNN accelerators which need the search process for pairing the nonzero weights and input activations.

Evaluations over a 65-nm place-and-route process show that the proposed design, composed of 4096 PEs, features 4.9TOP/s peak performance, 2.97W power running at 600MHz. Swan achieves 1.5-2.1× speedup and 6.0-9.1× energy efficiency compared with the state-of-the-art accelerators with the same PE scale. The details of the contributions are summarized as follow:

- We propose a search-free sparsity-aware accelerator to efficiently exploit the sparsity of CNNs under limited interconnect and bandwidth provision.
- We develop a sparsity-aware MAC unit that can free the search process from sparsity-aware MAC operations. More specifically, the offset indexes of the compressed weights function as the trigger signals to couple the corresponding activations for MAC operations.
- We present a systolic compressive dataflow to suit the proposed MAC unit and move both activations and compressed weights to adjacent PE columns for maximum reuse for interconnect and bandwidth saving. Only one column of the PEs load/store data, while all PEs perform in full scale.

II. Background

Basics of Sparse CNNs. The sparsity of CNNs denotes the proportion of zero weights/activations in all weights/activations. The weight sparsity comes from neural network pruning techniques, where unimportant weight parameters are replaced with zeros. Existing pruning approaches [5]–[7] have well reduced the model size while preserving the accuracy. On the other hand, the zero activations are dynamically generated during the layers' computation, especially after the rectified linear unit (ReLU) function.

Key Challenge for Increasing PE Scale. To better understand the challenge, Fig. 1(a) illustrates a simple conceptual architecture for sparsity-aware MAC operations, which has been utilized in Cambricon-X, EIE, and Cnvlutin. The architecture comprises the on-chip buffers for data storage, the PEs for MAC operations, and the customized search component (SC) for finding pairs of nonzero weights (w) and activations (ia), illustrated based on Fig. 1(b). SC searches pairs of nonzero weights and activations, and then sends them to the PE array for MAC operations. Therefore, they can provide sufficient nonzero input patterns for high throughput.

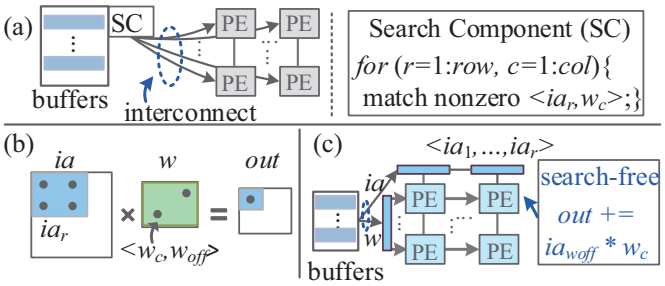


Fig. 1. Motivation example. (a) Interconnect overhead for nonzero pattern search, and then send them to PEs for computing, illustrated based on the sub-figure (b). And (c) interconnect overhead for search-free sparsity-aware MAC operations.

However, the internal interconnect and the on-chip memory bandwidth consumption become the paramount bottleneck when building superior PE scale for high-performance sparse CNN accelerations. This is because they need to send pairs of nonzero weights and activations to all PEs concurrently for computation. As a result, significant interconnect and memory bandwidth are required to ensure fast data transmission. For example, Cnvlutin has to deploy the expensive eDRAM and high-speed interconnect to satisfy the provision of 65,536-bit weights for 4096 PEs.

Our goal is to accelerate sparse CNNs with pruned weights on enlarged PE arrays with limited bandwidth and interconnect consumption. Based on our observations, the search process and MAC operations can be coupled and implemented in a constant cycle, as shown in Fig. 1(c). The corresponding activations ($ia_{w_{off}}$) for the compressed weights can be directly indexed by the offset indexes of the weights. Therefore, the corresponding process and MAC operations can be completed in the same cycle. In this case, the search process looks like to be freed from the sparsity-aware MAC operations. We further seek to save the interconnect and bandwidth based on the search-free sparsity-aware MAC architecture. We reorganize the dataflow and increase the reusability of inputs in the PE array. It is known that all activations on each output feature map share the same weights in convolutional (Conv) layers, which offers the opportunity of reusing more data in the PE array. In our design, activations and/or compressed weights are supplied to only one column of PEs, and at most one column of PEs needs to store the output activations in each cycle, while all PEs can operate in full scale.

III. Proposed Compressive Dataflow

In this section, we first illustrate the offline compression process for sparse weights, and then introduce the mapping scheme for sparse CNNs. Finally, we outline how we free the search process for sparsity-aware MAC operations.

Fig. 2 depicts the weight compression and the accommodation of compressed weights in on-chip buffers. In a Conv layer, the weight tensor features M output channels, N input channels, and a $K \times K'$ kernel size, as shown in Fig. 2(a). The nonzero weights are compressed along the input channel, where each nonzero weight is encoded in an offset manner [10]. w_{off} records the offset of a nonzero weight located in its input channel to the first channel. Fig. 2(b) depicts how

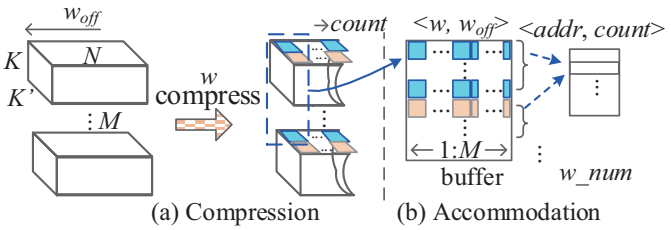


Fig. 2. Compression and accommodation for sparse weights.

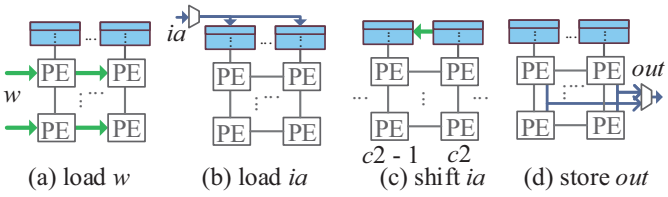
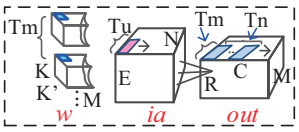


Fig. 4. Weights and activations operations on the PE array.

Inputs: R: # of output rows, M: # of output channels;
C: # of output column, N: # of input channels, *ia*:activation;
w: weights, w_{off}: weight offset, *w_num*(.): weight index;
K and K': # of kernel, Tm, Tn, and Tu: tiling parameters.

Initialization: c1=0; c2=1.




```
1  for (ro = 1:R; co = 1:Tn:C){
2    max = (co+Tn<C)? Tn:(C-co);
3    for (mo = 1:Tm:M){
4    for (ni=1:Tu:N; i=1:K){c1=0;
5    for(j=1:K'){     <addr, count>=w_num(mo, ni, i, j);
6    for(ni2=1:count){ ro2 = S*(ro-1)+i; co2 = S*(co-1)+j;
       //shift weights and offsets of PEs from left to right;
7      Shift-right (w(col(0:max-2), col(1:max-1)));
       //load Tm weights (<w,w_off>) to col 0
8      Load (w(Tm, addr, ni2), col(0));
9      if(c1<max){co3=co2+c1*S//load Tu activations to c1-th col
10       Load (ia((ni, Tu), ro2, co3), col(c1)); c1++; }//end if
11     if(j>1){  //shift ia from c2-th  to (c2-1)-th col
12       if(c2<max){Shift-left(ia(col(c2), col(c2-1))); c2++;}
13       else{co4=co2+(max-1)*S//load Tu ia to (max-1)-th col
14         Load(ia((ni, Tu), ro2, co4), col(max-1)); c2=1; }}
       //trigger one of Tu activations (ia_woff), MAC in PEs
15     out[Tm][max][co2] +=ia(ni+w_off, ro2, co2) * ...
                 w((mo,Tm), (ni,max), i, j);          }}} //end ni
     //store Tm output activations sequentially for 1:max PE col
16   Store (no, (mo, Tm), ro, (co,1:max));          }} //end ro
```

Fig. 3. Tiling-based compressive dataflow on an $m \times n$ PE array with compressed weights.

we accommodate the compressed weights in on-chip buffers. Both the nonzero weights and their offsets ($< w, w_{off} >$) are stored in a weight buffer, while their start address and maximum count ($< addr, count >$) for each group of the compressed weights are recorded in an index buffer (w_num). Since our work is only for CNN inference, the sparse weights are compressed and reorganized offline.

Fig. 3 depicts the sparsity-aware tiling schedule for an $m \times n$ PE array. To fit the $m \times n$ PE array, the M output channels and C column of the output feature maps are partitioned into groups of sizes Tm and Tn ($Tm = m$ and $Tn = n$), respectively, as shown in lines 1 and 3. That is, each output channel group has Tm channels and each group of output feature maps has Tn columns. Along the input channel direction, we partition all the N input channels into groups of size Tu, as shown in line 4. That is, each input channel group has Tu channels.

The key operations of the sparsity-aware tiling schedule include four steps: weight provision, activation provision, MAC operations, and output activation storage. The weight provision process is shown in lines 5-8. Specifically, the addresses for loaded compressed weights and their offsets are generated based on line 5. All weights and their offsets in the PE array are moved from left to their adjacent right PE column (line 7). At the same time, Tm compressed weights are loaded to the leftmost PE column (line 8). Details of the operations in lines 7-8 are summarized in Fig. 4(a).

The activation provision process includes two key operations: loading activations to one column of PEs (lines 10 and 14) and shifting the activations in one PE column to their adjacent left one (line 12). $c1$ and $c2$ are two signals respectively identify the current PE column for the load and shift-left operations. Specifically, when $c1 < max$ (max is defined in line 2, where max equals to Tn if $co + Tn < C$, otherwise it equals to $C - co$), the $c1$-th PE column is loaded with Tu input activations, and $c1$ is updated by a self-increment operation, as shown in lines 9 and 10. Details of the operations are illustrated in Fig. 4(b). $c1$ is reset to zero when computing next kernel row, as shown in line 4. When $j > 1$ (line 11), which denotes that the previously loaded activations can be reused by the shift-left operation. Specifically, when $c2 < max$, as shown in line 12, the activations in $c2$-th PE column are shifted to the adjacent PE column in left side, and $c2$ is updated by a self-increment operation. The shift-left operation for activations is illustrated in Fig. 4(c). Otherwise, it needs to add Tu input activations to the (max-1)-th PE column and reset $c2$ to 1, as shown in line 14. The MAC operations are described in line 15, which will be discussed in the following paragraph. Finally, we store the output activations after the MAC operations, as shown in line 16 and summarized in Fig. 4(d). At most one PE column needs to store output activations in a cycle because of the scheme of shifting weights.

Search-free Sparsity-aware MAC Operations. During MAC operations, the required input activations can be indexed by the offsets of the compressed weights ($ia_{w_{off}}$), as shown in line 15. w_{off} is utilized as the trigger signal to activate one of the Tu activations for the MAC operations directly. In this case, the corresponding process and the MAC operations can be coupled to free the search process for the sparsity-aware MAC operations.

For the proposed dataflow that shifts activations to adjacent PE columns for data reuse, large-than-1 strides may degrade the performance. To solve this issue, for larger-than-1 strides, the loaded adjacent input activations are also located in the same input row but with the corresponding stride size. As a result, adjacent PEs can also reuse the adjacent input activations. Consequently, the proposed design can efficiently avoid the impacts of the stride size.

The weight loading operations may temporally be stalled to wait the activations to be updated. This is because the weight loading operations can be conducted only after the corresponding input activations are loaded to the PE array. The number of stalled cycles SC is formulated as follow:

$$SC = \begin{cases} max - \sum_{1 \le j \le K'} w\_num(j); & \text{if } SC > 0 \\ 0; & \text{otherwise} \end{cases} \quad (1)$$

where max is the maximum active PE column index, which is defined in line 2. $w\_num(j)$ denotes the number of nonzero weights in the j-th kernel column, which is counted based on input channels (Fig. 2(b)). One extreme situation is the Conv-1 layer of CNNs with only 3 input channels. We enlarge the input channels virtually like [13] by utilizing the large-than-1 stride. Differently, we do not need adder-trees for accumulation. In fact, stalling is almost not occurred because $\sum_{1 \le j \le K'} w\_num(j)$ is almost always larger than max.

The proposed sparsity-aware dataflow can achieve efficient computation under limited internal interconnect and bandwidth. The bandwidth saving comes from three aspects. First, only the leftmost PE column needs to load m compressed weights, and all the other PEs reuse the weights by shifting them from left to right. Second, at most Tu input activations are loaded to one column of PEs. Third, the maximum stored output activations reach only m number for the PE array. The consumed internal interconnect and memory bandwidth only depend on the row number of the PE array, so that we can dramatically save the bandwidth to increase the PE scale, and in turn, achieve higher performance.

Though the proposed dataflow dramatically reduces the internal memory bandwidth by data reuse in Conv layers, it does impose some challenges when tackling fully-connected (FC) layers. This is because there are not any reusable weights in the FC layers. At most $1/m$ of the PEs are active for FC layers. Nevertheless, we can achieve efficient accelerations for sparse CNNs, owing to two sources. First, compared with Conv layers, the FC layers involve less computation in CNNs. For example, the FC layers occupy less 10% of the total operations in most popular CNNs such as Googlenet and Resnet [15]. Second, the sparse weights in FC layers can significantly reduce the volumes of both computation and memory access. For example, 89.9% of the weights of the FC layers in Alexnet can be pruned without accuracy loss [5]. To sum it up, the proposed design can achieve efficient sparsity-aware accelerations under limited internal bandwidth.

IV. Hardware Design

Fig. 5 outlines the proposed sparsity-aware CNN accelerator architecture. The accelerator comprises three main components: a neural processing element (NPE), a central controller (Ctl), and on-chip buffers (a weight buffer (WB) and two activation buffers (ABin and ABout)). NPE mainly performs the sparsity-aware MAC operations. WB accommodates compressed weights, their offsets, and w_num. Ctl triggers the data movements and schedules the sparsity-aware MAC operations. Ctl activates the direct memory access (DMA) component to perform data movements between the on-chip buffers (i.e., WB, ABin, and ABout) and off-chip memory.

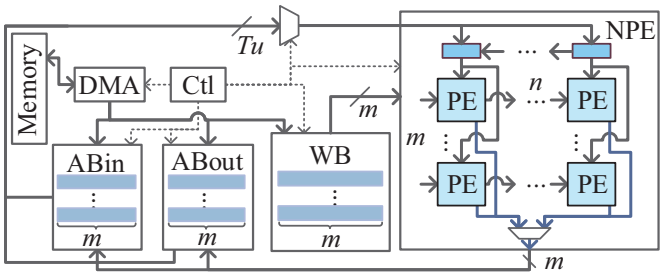


Fig. 5. Overview of the proposed architecture.

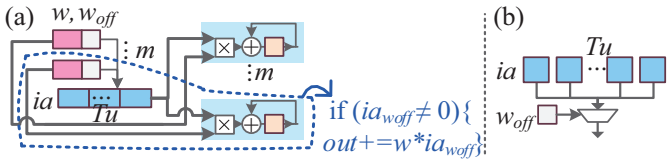


Fig. 6. Search-free MAC unit. (a) Architecture of one column of PEs. (b) Architecture for $ia_{w_{off}}$ in a PE.

ABin and ABout. ABin and ABout store the input and output activations alternately. At most m output activations need to be stored from the NPE to ABout/ABin in each cycle. During the activation load operations, ABin/ABout needs to supply Tu input activations to one column of the PEs. The operation of loading Tu input activations can be implemented by partitioning the m activations of a buffer row into m/Tu groups.

NPE. NPE comprises $m \times n$ PEs for sparsity-aware MAC operations. It receives m input weights and their offsets in the leftmost PE column and shifts them from left to right for weight reuse. Each column of PEs shares the same Tu input activations. NPE performs MAC operations based on the compressed weights and offset indexes and generates at most m output activations in each cycle.

NPE functions as a systolic array that can move compressed weights and uncompressed activations in the PE array for reuse. Comparatively, the activations and weights are reused in three aspects. First, the weights in each PE column share the input activation register vector. Second, the input activations are reused by adjacent left PEs according to the shift-left operation. Third, the compressed weights are reused by the adjacent PE columns on the right side.

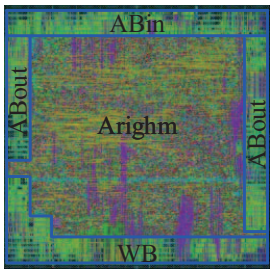
Search-free MAC Architecture. Fig. 6 depicts the search-free sparsity-aware MAC architecture based on m PEs in the same column. Tu input activations are shared by the m nonzero weights, as shown in Fig. 6(a). Each PE includes a multiplier, an accumulator, an input weight register for storing an input weight and their offset, and an output register for partial sum accumulation. The PE can be gated for energy saving if the input activation is zero. In MAC computation, the Tu input activations are indexed by each PE according to the offset indexes of the weights so that the search process is avoided, as shown in Fig. 6(b). Therefore, the proposed architecture can outperform prior accelerators, Cambricon-X, EIE, and Cnvlutin, which utilize customized search architectures

TABLE II
STATISTICAL NONZERO WEIGHTS AND ACTIVATIONS OF BOTH CONV AND FC LAYERS IN CNN MODELS.

Weights	Total	Conv	FC
Alexnet	21.67M(18.6%)	0.56M(12.6%)	21.11M(18.9%)
Googlenet	4.95M(37.1%)	4.77M(41.9%)	0.18M(9.3%)
Resnet18	4.15M(16.5%)	3.82M(17.9%)	0.33M(8.5%)
Zfnet	14.68M(12.4%)	2.17M(30.6%)	12.51M(12.6%)
Activations	**Total**	**Conv**	**FC**
Alexnet	0.87M(79.4%)	0.86M(81.0%)	$0.97e^{-2}$M(29.2%)
Googlenet	5.0M(56.0%)	5.0M(56.0%)	$0.15e^{-2}$M(77.1%)
Resnet18	2.79M(61.0%)	2.78M(61.0%)	$0.34e^{-2}$M(88.1%)
Zfnet	0.71M(58.5%)	0.69M(58.5%)	0.02M(61.9%)

TABLE III
COMPARISONS OF REPRESENTATIVE ACCELERATORS WITH SWAN.

	Cambr-icon-X	Dadian-nao	EIE	SCNN	Cnvl-utin	Swan
Technology (nm)	65	28	45	16	28	65
Clock (MHz)	1000	606	800	1000	606	600
# of PEs	256	4096	64	1024	4096	4096
Area (mm^2)	6.38	67.7	40.8	7.9	–	51.5
Peak perf. (GOP/s)	512	5580	102	–	5580	4915
Power (W)	0.799	15.97	0.59	–	14.85	2.97
Power efficiency (GOP/s/W)	**640.8**	**349.4**	**172.8**	–	**334.3**	**1654.9**



Component	Power (W)	Area (mm^2)
WB	0.090(3.0%)	6.72(13.0%)
ABin	0.094(3.2%)	5.38(10.5%)
ABout	0.087(2.9%)	5.45(10.6%)
NPE	2.625(88.4%)	14.88(28.9%)
other	0.074(2.5%)	1.10(2.1%)
Fill cell	-	17.97(34.9%)
Total	**2.97**	**51.5**

Fig. 7. The layout achieved after placed and routed on 65-nm technology.

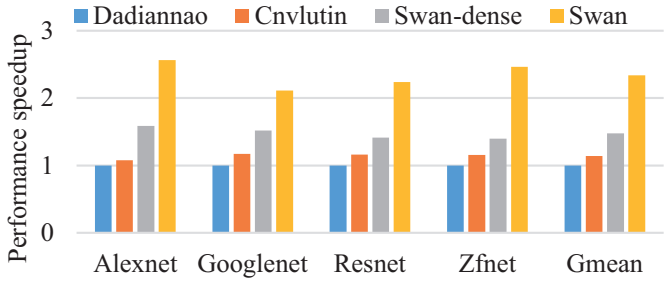


Fig. 8. Normalized speedup comparison with the Cnvlutin and Dadiannao baselines.

for sparsity-aware MAC operations.

V. EVALUATION

A. Experimental Setup

We use four representative CNNs, Alexnet, Googlenet, Resnet18, and Zfnet, with image inputs from ImageNet as our benchmarks. The weights are pruned based on the pruning technique introduced in [5]. Details of the proportion of the nonzero weights and activations are shown in Table II. The nonzero activations are statistical results based on randomly 1000 input images. We compare our accelerator with several state-of-the-art accelerators, including Cambricon-X, Dadiannao, EIE, SCNN, and Cnvlutin.

We implement the proposed accelerator with RTL in Verilog. We synthesize the proposed design by Synopsys Design Compiler under TSMC 65-nm technology, and then place and route it in Synopsys IC compiler. The on-chip memory is generated by Memory Compiler. The off-chip memory is evaluated by CACTI [16]. The total capacity of the on-chip buffers for weights and activations is 417KB. Specifically, the on-chip buffers include 161 KB WB for compressed weights, 128KB ABin for input activations, and 128KB ABout for output activations. In particular, WB comprises 128KB of nonzero weights, 32KB of the offsets of the nonzero weights, and 1KB of w_num. The proposed design has 4096 PEs in total, with $m = 256$, $n = 16$, and $Tu = 16$. Weights and activations are encoded in 16-bit fixed points, and the offset of each weight is 4-bit.

B. Evaluation Results

Profile of the Proposed Design. Table III shows the comparison of the proposed Swan accelerator with the recent representative works. Based on the 65-nm technology, Swan features 4915GOP/s peak performance, 2.97W power, and 51.5mm^2 area running at 600MHz frequency. The peak performance of the accelerators for comparison are calculated according to the PE number and the default operating frequency reported in prior literatures. It can be observed that Swan gains over 2.6× better peak power efficiency than previous representative sparsity-aware accelerators. This is because Swan can efficiently save internal interconnect and on-chip memory bandwidth resources for the superior PE scale. Fig. 7 shows the power and area breakdowns of Swan. "other" refers to the controller and the interconnect. It can be observed that NPE dominates the power with 88.4% of the total power, owing to the big PE scale.

Performance. We compare our accelerator with sparsity-aware Cnvlutin and non-sparsity-aware Dadiannao baselines, which have the same PE array size, based on all benchmarks listed in Table II. For a fair comparison, we evaluate the performance of our accelerator for dense representation (i.e., Swan-dense) as well. Fig. 8 shows the normalized performance comparison. The performance of all accelerators are normalized to that of the Dadiannao baseline. It can be observed that Swan achieves 2.1× speedup than Cnvlutin though the frequency is only 0.99×, because Swan can effectively tackle the Conv layers for sparsity-aware computation. On average, Swan-dense achieves 1.5× speedup than Dadiannao, because we can efficiently avoid the PE under-utilization impact by tiling the Conv layers based on the output channel, input channel, and output column, to well suit the PE array, while Dadiannao tiles only the input and output channels.

Energy Efficiency. Fig. 9 shows the normalized energy efficiency comparison against the baselines. Compared with Cnvlutin, Swan achieves 9.1× better energy efficiency for the sparse workloads. The better energy efficiency come from the higher performance and significantly lower power consumption for CNN benchmarks. Also, Swan-dense gains

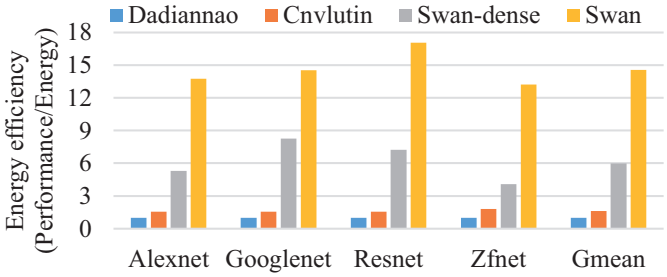


Fig. 9. Normalized energy efficiency comparison against the baselines.

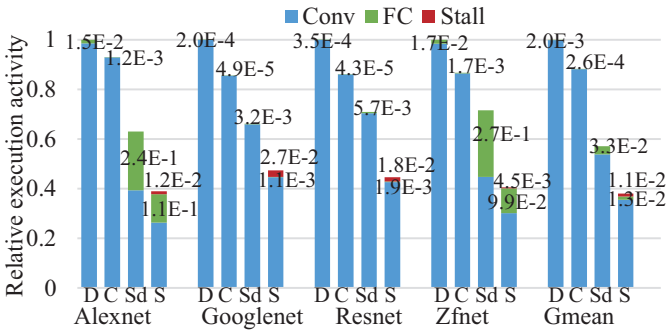


Fig. 10. Normalized performance breakdown. Swan (S), Swan-dense (Sd), and Cnvlutin (C) are normalized to the Dadiannao (D) baseline. Both FC layers and stall operation are noted with labels.

6.0× higher energy efficiency than the non-sparse workloads. Consequently, the proposed design can achieve significant energy efficiency improvement for CNN deployments.

Breakdown. Fig. 10 shows the performance breakdown compared with the Cnvlutin and Dadiannao baselines, including the impacts of Conv and FC layers and the stalled cycles in weight loading. It can be observed that (1) the performance benefits of our accelerator mainly come from the Conv layers; (2) Swan can efficiently alleviate the performance overhead for FC layers by exploiting the sparsity of inputs. For example, Swan takes only 38% of the execute time than Swan-dense in FC layers; And (3), the stalling overhead of Swan is very small. On average, the stalling latency of Swan occupies 3.0% of the total execution time (the percentages are respectively 3.1%, 5.7%, 4.1% and 1.1% for the four workloads). To sum it up, the proposed design can achieve efficient accelerations for sparse CNNs.

VI. Conclusion

In this work, we propose a superior sparsity-aware accelerator that can significantly reduce the internal interconnect and bandwidth for embedded systems. The proposed accelerator comprises (a) an elaborate MAC unit that can free the search process for sparsity-aware MAC operations and (b) a compressive dataflow that can well suit the proposed MAC unit and fully exploit data reuse with restricted interconnect and on-chip memory bandwidth. Evaluation results show that the proposed design can achieve 1.5-2.1× speedup and 6.0-9.1× higher energy efficiency than the state-of-the-art accelerators under the same PE scale.

Acknowledgment

This work was supported in part by the National Key R&D Program of China under Grant 2018YFA0701500, in part by the 0-to-1 Innovative Project of the CAS Fundamental Frontier Scientific Research Program under Grant ZDBS-LY-JSC012, in part by National Natural Science Foundation of China under Grants 61804155 and 61834006, in part by Beijing Municipal Science & Technology Commission under Grants Z171100000117019 and Z181100008918006, in part by the Youth Innovation Promotion Association CAS, in part by the Young Elite Scientists Sponsorship Program by CAST under Grant 2018QNRC001, and in part by the Beijing Academy of Artificial Intelligence under Grant BAAI2019QN0402.

References

[1] Krizhevsky, A., et al., "Imagenet classification with deep convolutional neural networks", In *Advances in neural information processing systems*, 2012, pp. 1097-1105.
[2] Szegedy, C., Liu, W., Jia, Y., et al. "Going deeper with convolutions", In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1-9.
[3] He, K., Zhang, X., Ren, S., et al., "Deep residual learning for image recognition", In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770-778.
[4] Zeiler, M. D., and Fergus, R., "Visualizing and understanding convolutional networks", In *European conference on computer vision*, 2014, pp. 818-833.
[5] Han, S., Mao, H., and Dally, W. J. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding", in *International Conference on Learning Representations*, 2016, pp. 1-14.
[6] Park, J., Li, S., Wen, W., et al., "Faster cnns with direct sparse convolutions and guided pruning", in *International Conference on Learning Representations*, 2017, pp.1-11.
[7] Guo, Y., Yao, A., and Chen, Y. "Dynamic network surgery for efficient dnns", In *Neural Information Processing Systems*, 2016, pp. 1379-1387.
[8] Han, S., Liu, X., Mao, H., et al., "EIE: efficient inference engine on compressed deep neural network", In *ACM/IEEE 43rd Annual International Symposium on Computer Architecture*, 2016, pp. 243-254.
[9] Zhang, S., Du, Z., Zhang, L., et al., "Cambricon-x: An accelerator for sparse neural networks", In *49th Annual IEEE/ACM International Symposium on Microarchitecture*, 2016, p. 20.
[10] Albericio, J., et al., "Cnvlutin: Ineffectual-neuron-free deep neural network computing", In *ACM SIGARCH Computer Architecture News*, 2016, vol. 44, pp. 1-13.
[11] Parashar, A., et al., "Scnn: An accelerator for compressed-sparse convolutional neural networks", In *ACM/IEEE 44th Annual International Symposium on Computer Architecture*, 2017, pp. 27-40.
[12] Chen, T., Du, Z., Sun, N., et al., "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning", In *ACM Sigplan Notices*, 2014, pp. 269-284.
[13] Liu, B., Chen, X., Wang, Y., et al., "Addressing the issue of processing element under-utilization in general-purpose systolic deep learning accelerators", In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 733-738.
[14] Chen, Y., Luo, T., Liu, S., et al., "Dadiannao: A machine-learning supercomputer", In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 609-622.
[15] Zhang, C., et al., "Optimizing fpga-based accelerator design for deep convolutional neural networks", in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015, pp. 161-170.
[16] Muralimanohar, N., Balasubramonian, R., and Jouppi, N. "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0", In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 3-14.