Sense: Model-Hardware Codesign for Accelerating Sparse CNNs on Systolic Arrays

Wenhao Sun, Deng Liu, Zhiwei Zou, Wendi Sun, Song Chen, *Member, IEEE*, and Yi Kang, *Member, IEEE*

***Abstract*— Sparsity is an intrinsic property of convolutional neural networks (CNNs), worth exploiting for CNN accelerators. However, the extra processing involved comes with hardware overhead, resulting in only marginal profits for most architectures. Meanwhile, systolic arrays have become increasingly competitive on CNN acceleration for its high spatiotemporal locality and low hardware overhead. However, the irregularity of sparsity induces imbalanced workloads under the rigid systolic dataflow, causing performance degradation. Thus, this article proposed a systolic-array-based architecture, called Sense, for sparse CNN acceleration by model-hardware codesign, enabling large performance gains. To balance input feature map (IFM) and weight loads across the processing element (PE) array, we applied channel clustering to gather IFMs with approximate sparsity for array computation and codesigned a load-balancing weight pruning method to keep the sparsity ratio of each kernel at a certain value with little accuracy loss, improving PE utilization and overall performance. In addition, adaptive dataflow configuration was applied to determine the computing strategy based on the storage ratio of IFMs and weights, lowering 1.17×–1.8× dynamic random access memory (DRAM) access compared with Swallow and further reducing system energy consumption. The whole design was implemented on ZynqZCU102 with 200 MHz and performs at 471, 34, 53, and 191 image/s for AlexNet, VGG-16, ResNet-50, and GoogleNet, respectively. Compared with sparse systolic-array-based accelerators, Swallow, fusion-enabled systolic architecture (FESA), and SPOTS, Sense achieves 0.97×–2.18×, 1.3×–1.67×, and 0.94×–1.82× energy efficiency (image/J) on these CNNs, respectively.**

***Index Terms*— Convolutional neural network (CNN), hardware accelerator, sparsity, systolic array, weight pruning.**

I. Introduction

NOWADAYS, neural networks have been widely applied in numerous domains, such as image recognition [1], speech recognition [2], object detection [3], and computer vision [4]. Convolutional neural networks (CNNs) [1], [3], [5], [6], [7] have excelled in image recognition and object detection. As higher precision and complication demands arise, the workloads of network computing continue to increase. Therefore, researchers have been striving for neural network hardware accelerators to catch up with software development.

Although CNNs bring intensive computations, there exist monotonous operations; thus, many hardware architects attempt to reduce runtime by improving computation parallelism. Meanwhile, weights and input feature maps (IFMs) can be extensively reused across operations. Since data movement consumes much more energy than computation [8], especially with dynamic random access memory (DRAM), many sophisticated dataflow strategies have been designed to improve the data reuse rate. For example, Eyeriss [9] applies row stationary dataflow to minimize movement consumption; tensor processing unit (TPU) [10] revives systolic array to realize high-energy efficiency and throughput with structured dataflow and low bandwidth; Thinker [11] supports bit-width adaptive computing and on-demand array partitioning and reconfiguration to maximize resource utilization. These works primarily actualize acceleration through parallelism improvement and dataflow optimization, without consideration of sparsity processing on IFMs and weights, causing redundant storage and computation.

Furthermore, sparsity is an intrinsic property of CNNs. For IFMs, approximately half of their elements are zeros because of the widely used nonlinear function rectified linear unit (ReLU); for weights, the fault tolerance of CNN allows of a large number of redundant zeros, which inspires researchers to develop pruning and quantization methods [12], [13], [14], [15], [16], [17], [18]. Therefore, various sparsity-aware CNN accelerators have been designed. EIE [19] supports both sparse weight and IFM processing, but only focused on fully-connected (FC) layers. Cambricon-X [20] and Cnvlutin [21] take advantage of sparse weights and IFMs, respectively, to eliminate operations induced by zero. SCNN [22] can support arbitrary sparse patterns but fails to fully leverage the benefits provided by sparse IFMs and weights owing to imbalanced workloads and access contentions across independent processing elements (PEs). Zhu et al. [23] propose an FPGA architecture to deal with the irregular connection in sparse convolution (CONV) layers. However, huge LUT resources are consumed to match the indexes between weights and IFMs and increase linearly with PE array scaling, which becomes the bottleneck of performance. HPIPE [36] presents a cross-layer-pipelined architecture and codesigns a network compiler, which statically partitions available FPGA resources

Manuscript received 5 August 2022; revised 2 November 2022 and 23 December 2022; accepted 21 January 2023. Date of publication 13 February 2023; date of current version 22 March 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB2204800, in part by the National Natural Science Foundation of China (NSFC) under Grant 61931008, in part by the CAS Project for Young Scientists in Basic Research under Grant YSBR-029k, and in part by the Strategic Priority Research Program of Chinese Academy of Sciences under Grant XDB44000000. *(Corresponding author: Song Chen.)*

Wenhao Sun, Deng Liu, Zhiwei Zou, and Wendi Sun are with the School of Microelectronics, University of Science and Technology of China, Hefei 30332, China (e-mail: wh1997@mail.ustc.edu.cn).

Song Chen and Yi Kang are with the School of Microelectronics, University of Science and Technology of China, Hefei 30332, China, and also with the Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, Hefei 30332, China (e-mail: songch@ustc.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2023.3241933.

Digital Object Identifier 10.1109/TVLSI.2023.3241933

and builds custom-tailored hardware for each layer of a certain CNN according to the computing complexity and weight sparsity. But it needs to store weights of the entire network on FPGA memory, which cannot be implemented for those CNNs whose weights exceed the storage of FPGA. Lu et al. [24] design an FPGA accelerator to avoid extra coordinate computation for connection reconstructing or output locating with a sparsewise dataflow. However, since this accelerator consumes large block random access memory (BRAM) resources for output feature map (OFM) storage in each PE, the performance is bounded by the number of BRAM on FPGA. Therefore, these architectures suffer from one-side sparsity exploration, access contention, or inefficient resource utilization.

Since systolic arrays [25] are widely applied to accelerate CNNs [10], [11], [26], [27] because of its high bandwidth-saving capabilities and low hardware overhead, achieving higher energy efficiency under the same peak throughput compared with other architectures. Thus, researchers attempt to process sparsity with systolic architectures for improving the overall benefits. Swallow [28] overcomes the inability to exploit the sparsity of weights and IFMs, CONV layers and FC layers of CNNs with limited resources in a systolic array, and introduces a sparse-aware dataflow to boost PE utilization, achieving high bandwidth saving and energy efficiency. However, the structured systolic dataflow essentially contradicts with the irregularity of sparsity, causing imbalanced PE loads. Considering that, fusion-enabled systolic architecture (FESA) [29] prunes the kernels to 2–7 formalized zero distribution patterns and leaves IFM unprocessed to regularize dataflow as the dense systolic tempo, achieving lower sparsity processing overhead. However, this pruning method is currently only implemented on Cifar-10 and Cifar-100 [30]. Thus, to balance workloads with higher versatility in systolic arrays, SPOTS [31] designs a groupwise pruning method to divide weights into groups and prunes some elements at the same position in each group, which achieves similar versatility with shapewise pruning method [32] and improves compatibility with systolic arrays. Accordingly, SPOTS applies image to column (Im2Col) transformation of IFMs coupled with general matrix-matrix multiplication (GEMM) to better fit its pruning scheme into systolic arrays by skipping the weight rows and IFM columns with all zeros. However, since its pruning method is too fine-grained, the sparsity of weights is bounded by accuracy. Besides, SPOTS fails to exploit the sparsity in those rows and columns with some zeros, causing inefficient acceleration.

Furthermore, memory access occupies a huge proportion of system energy consumption, making it critical to further reduce memory access through dataflow. But sparse IFMs and weights can be irregular and fragmented, which leads to lower memory access efficiency. SCNN [22] employs a novel dataflow to eliminate unnecessary data transfers, but access contentions occur when routing the products to the accumulator buffer owing to irregular sparse patterns. Lu et al. [24] propose a weight layout to enable efficient memory access without conflicts, but huge LUT consumption restrains the performance. Swallow harnesses a sparsity-aware dataflow with matrix multiplication tiling to promote data reuse within each channel, reducing DRAM access with little overhead. However, Swallow always preferentially reuses IFMs, whereas DRAM access can be variable if we choose different reuse strategies. Thus, there is still room to further lower DRAM access by choosing dataflow according to the storage ratio of IFMs and weights in each layer.

These previous sparse systolic accelerators suffers from imbalanced workloads, lack of versatility, or low sparsity of weight pruning. Besides, the dataflow is inflexible to the variable ratio of IFM and weight in each layer. Thus, this article aims to balance workloads to fit with the sparse systolic array, while maintaining the sparsity and versatility of weight pruning with reasonable overhead, and further optimizing DRAM access. A model-hardware codesign of the sparse CNN accelerator based on systolic arrays is proposed to improve system performance and energy efficiency. Our main contributions are as follows.

1) We proposed a systolic-array-based accelerator for both sparse IFM and weight processing with reasonable overhead. It supports the entire kernel and IFM block compression and eliminates zero element operations to accelerate computing.
2) We applied channel clustering to gather IFMs with approximate sparsity to compute in the PE array by sorting channel indexes according to the numbers of nonzero elements (NZEs) in IFMs and codesigned a load-balancing weight pruning method to keep the sparsity ratio of each kernel at a certain value with little accuracy loss, which can balance PE loads across the systolic array and improve performance.
3) We proposed an adaptive dataflow configuration to optimize the data reuse rate and further reduce DRAM access. The reuse strategy of our dataflow can be flexibly configured according to the storage ratio of IFMs and weights.

Comparing with the sparse systolic accelerator, Swallow [28], we balanced the workloads of sparse IFMs and weights in the systolic array and achieved $1\times$–$2.25\times$ speedup, and $0.97\times$–$2.2\times$ energy efficiency with 1.6% LUT, 7% BRAM, and 3% power overhead. Based on the adaptive dataflow configuration, we reduced $1.17\times$–$1.8\times$ DRAM access. Besides, comparing with the pruning method of FESA [29], we achieved $1.2\times$–$1.3\times$ speedup in terms of weights and implemented on more complicated datasets, such as ImageNet, with higher versatility. By exploiting the sparsity of both IFMs and weights, we achieved $1.95\times$–$2.5\times$ performance with $1.5\times$ power. Further, compared with SPOTS [31], our pruning method obtained $1\times$–$2.02\times$ sparsity of weights, achieving $1.17\times$–$2.37\times$ speedup with $1.3\times$ power.

II. Preliminary

According to the computing process of CNN, skipping zeros of IFMs and weights can effectively accelerate execution. However, for systolic arrays, there still exist problems of imbalanced workloads, that is, PEs with IFMs and weights of high sparsity rate (0%) must wait for those with low sparsity rate to finish. Thus, lots of PEs will be idle, reducing array

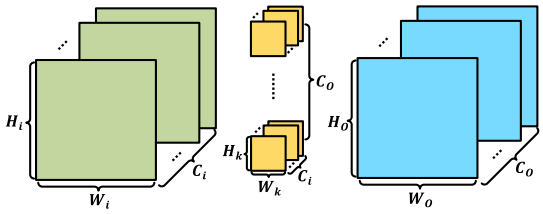

Fig. 1. CONV process.

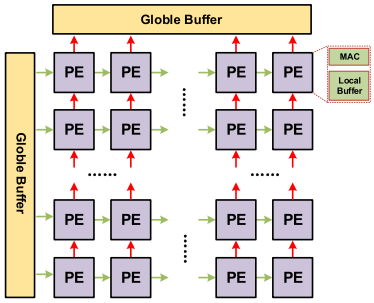

Fig. 2. Systolic array architecture.

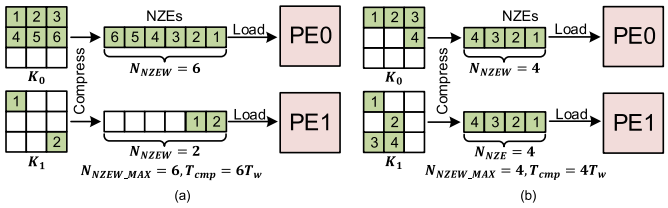

Fig. 3. (a) Imbalanced and (b) balanced loads of sparse weights in the systolic array.

utilization rate. In addition, when a certain PE contains data with really low sparsity rate, it significantly blocks the overall acceleration. Consequently, the key challenge is to balance workloads across systolic array.

A. Computing Process of CNN

The computing process of CNNs is shown in Fig. 1. A CONV layer receives C_i input channels (ICs) of IFMs with a size of $H_i \times W_i$. Each OFM of C_o output channels (OCs), with a size of $H_o \times W_o$, is generated by sliding an $H_k \times W_k$ kernel window on one IFM and then accumulating across the temporary results of C_i ICs. Finally, C_o OFMs are calculated by convolving C_o groups of kernels with shared IFMs. Set IFMs as matrix $I[C_i,H_i,W_i]$, weights as $W[C_o,C_i,H_k,W_k]$ and OFMs as $O[C_o,H_o,W_o]$, and the process can be described as follows:

$$O[o,x,y] = \sum_{i=0}^{C_i}\sum_{a=0}^{H_k}\sum_{b=0}^{H_k} I[i,x+a,y+b] \times W[o,i,a,b] \quad (0 \le o < C_o, 0 \le x < H_o, 0 \le y < W_o). \quad (1)$$

B. Systolic Array

Systolic arrays [25] are widely adopted for CNN acceleration [10], [11], [26], [27] with distinct advantages. First, a systolic architecture consists of numerous simple and uniform PEs, achieving high performance with low hardware cost. Second, systolic arrays propagate data horizontally and vertically with simplified inter-PE connection, which can highly reuse data and reduce data movement consumption. Finally, communication with the outside occurs only at the boundary PEs, providing huge bandwidth saving. An example of a systolic array is shown in Fig. 2. Each PE conducts multiplication-accumulation (MAC) operation with locality storage. And they can station one input of MAC temporally and transfer other inputs to neighbor PEs spatially. Due to the structured dataflow, the control unit only takes up a small portion of system hardware cost, achieving high resource utilization. Despite these strong points, there still exist challenges in accelerating sparse CNNs on systolic arrays.

C. Challenges of Sparse Systolic Accelerator

To accelerate CNN processing, mapping computations on a systolic array can significantly improve parallelism. Besides, the sparsity of IFMs and weights can be 10%–90% [28], which theoretically provides up to 20× performance improvement by skipping zero computations. However, the irregularity of sparsity conflicts with the structured dataflow in systolic array, which may induce imbalanced workloads, causing low PE utilization and insufficient acceleration. Since CONV layers account for the majority of the computations involved in CNNs, the challenges are analyzed from the perspective of weights and IFMs in CONV layers.

For sparsity of weights, they contain a large number of redundant zeros after pruning, but the sparsity ratio of each kernel differs, causing imbalanced workloads across the PE array. An example of computing flows toward weights with imbalanced loads after pruning is shown in Fig. 3(a). Kernel K_0 and K_1 are compressed and loaded in PE_0 and PE_1, respectively. Assuming the number of NZEs in each kernel is N_{NZEW} with the largest value being $N_{\text{NZEW\_MAX}}$, and the computing time needed for a single weight is T_W, the total computing time of PE_0 and PE_1 are $6T_W$ and $2T_W$, respectively. Based on the regular dataflow of systolic arrays, the computing time of the entire system is blocked to $6T_W$ and PE_1 will be idle for $4T_W$, indicating low performance and PE utilization rate.

For sparsity of IFMs, there also exist many unevenly distributed zero elements after activation functions like ReLU, which still induces imbalanced workloads across the PE array. Moreover, unlike weights, they are dynamically produced during execution and cannot be trained offline. Therefore, the sparsity rate of IFMs in different channels is unpredictable. As shown in Fig. 4(a), IFMs of four channels are loaded into a 1×2 PE array in two time steps. Assume the computation time of a single IFM element is T_i and the NZEs in each IFM, N_{NZEI}, are 8, 4, 8, and 3, respectively. If we load the compressed IFMs in order of I_0 and I_1, I_2 and I_3, the total computing time in Time0 and Time1 are both blocked to $8T_i$ because the largest numbers of NZEs, $N_{\text{NZEW\_MAX}}$, is 8, and PE_1 will be idle for $9T_i$ in total, owing to the imbalanced loads.

Although there are some previous works of sparse CNN acceleration in systolic arrays, such as Swallow [28], FESA [29] and SPOTS [31], they suffer from inadequate use of

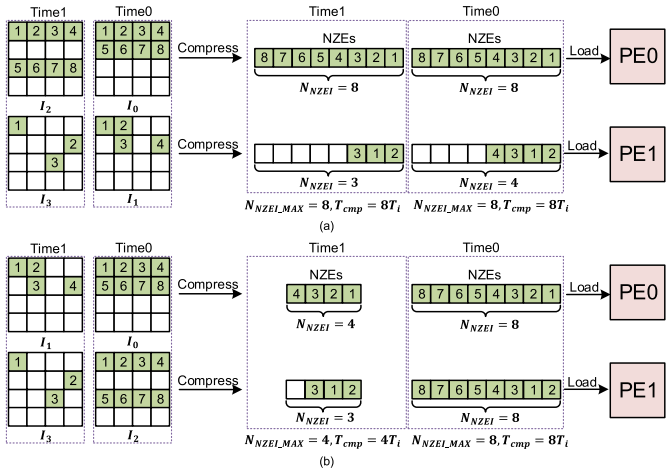

Fig. 4. (a) Imbalanced and (b) balanced loads of sparse IFMs in the systolic array.

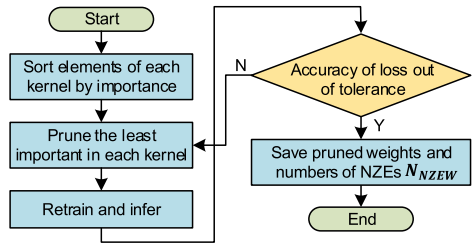

Fig. 5. Load-balancing weight pruning flow.

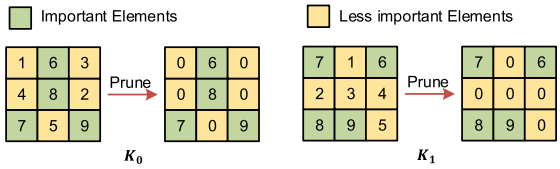

Fig. 6. Example of load-balancing weight pruning.

sparsity of IFMs and weights owing to imbalanced workloads. Thus, in this work, we aim to balance the loads of sparse weights and IFMs in systolic array architectures.

III. Methodology

To deal with imbalanced workloads in the systolic array, we applied load-balancing weight pruning for sparse weights and channel clustering for sparse IFMs, achieving significant improvement of PE utilization and performance. For load-balancing weight pruning, we ranked the elements of each kernel by importance and then pruned the less important elements to a certain sparsity ratio. For channel clustering of IFMs, we recorded the NZEs in each IFM and accordingly ranked the channel indexes. Subsequently, IFMs could be accessed in the sequence of sorted channel indexes and gathered with approximate sparsity ratios.

To fit the strategies of balancing workload in CONV layers, we directly compressed IFMs and kernels for efficient sparsity utilization and applied weight-oriented [23] computing flow to coordinate with the compression format. Since the FC layers, whose computation pattern is generally matrix-vector multiplication (GEMV), only takes up a small proportion in the entire network, we mapped its computing flow as a CONV pattern with the outer product [33] to reduce the control overhead. In addition, for FC layers, we randomly applied a pruning method [19] to maximize the sparsity of weights and channel clustering to balance the workloads of each column in weight matrices.

A. Load-Balancing Weight Pruning

CNNs contain many redundant weights, and with smaller absolute value of weight comes less importance. Thus, to balance computing loads across the systolic array, we pruned each kernel by importance to set several less important elements as zero and maintain the sparsity ratios of all kernels at a certain value. An example of the computing flows toward load-balancing weight pruning is shown in Fig. 3(b). Compared with the load-imbalanced weight pruning in Fig. 3(a), the NZEs in K_0 and K_1 are both four, eliminating idle PEs. Therefore, we obtained the total computing time of $4T_w$ with $1.5\times$ speedup and significantly improved the PE utilization rate.

The specific flow of load-balancing weight pruning is shown in Fig. 5. First, we sorted the elements of each kernel by importance and then set the least important element as zero. Next, the previously pruned weights were retrained and testified if the accuracy dropped out of boundary. If not, we continued to prune and train the remaining elements; otherwise, we saved the final pruned weights and recorded N_{NZEW}. Fig. 6 shows a pruning example of two 3×3 kernels, K_0 and K_1. We set the five least important elements as zero in each kernel. Although the distribution of zero elements is irregular, the workloads of each kernel are balanced, thereby achieving $9/4 = 2.25\times$ speedup.

B. Channel Clustering

Since IFMs are generated dynamically, they cannot be trained offline to balance IFM workload. Thus, we ranked ICs by the NZEs in IFMs on hardware to enable channels with approximate sparsity ratios to be clustered for computing in the PE array. An example of channel clustering is shown in Fig. 4(b). Compared with the flow without channel clustering in Fig. 4(a), we clustered I_0 and I_2, as well as I_1 and I_3, and the total computing time in Time0 and Time1 reduced to $8T_i$ and $4T_i$, respectively, achieving $1.33\times$ speedup and significantly eliminating idle PEs.

The specific flow of channel clustering is shown in Fig. 7. From the perspective of dataflow, each PE row shares the same IFMs and each column outputs one OFM to save bandwidth and improve parallelism, which will be further discussed in Section V. Assuming W_step0/1 are two steps of the process in which the PE array calculates OFMs and writes them into memory, col_0 and col_1 concurrently produce two channels of OFMs in each step during layer i, $O_{i,0}$, and $O_{I,1}$ of W_step0, $O_{i,2}$ and $O_{i,3}$ of W_step1. Meanwhile, we sorted $O_{i,0}$–$O_{i,3}$ by the NZEs to cluster OFMs with approximate sparsity. Assuming that the memory of each address can accommodate the outputs of two PE columns, if we directly write in the

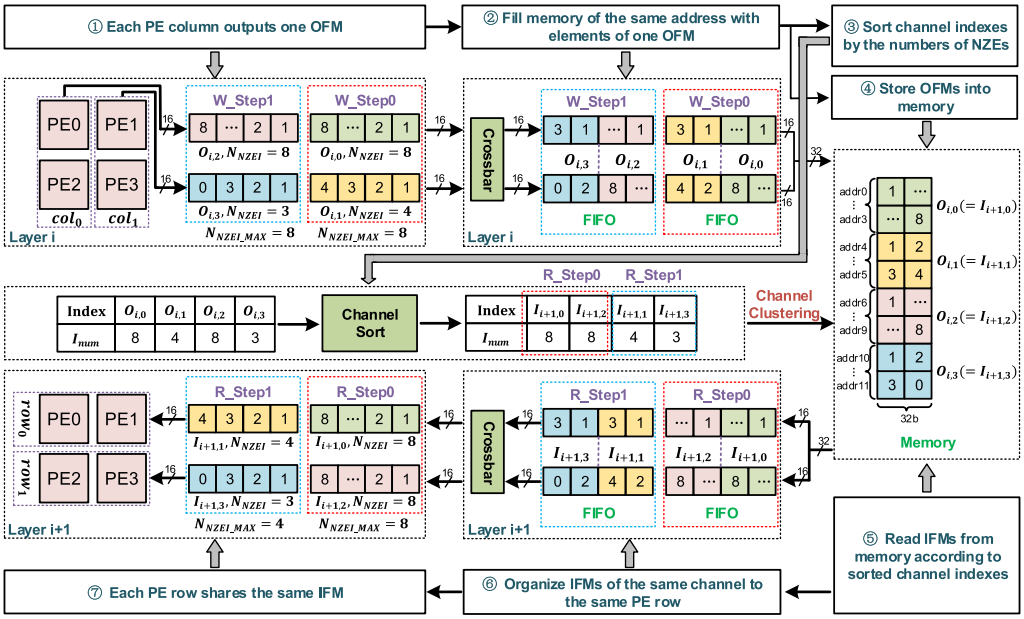

Fig. 7. Process of channel clustering. $I_{i+1,0}$, $I_{i+1,2}$, and $I_{i+1,1}$, $I_{i+1,3}$ are gathered to compute.

OFMs from the W_step stage, each input data of memory can contain elements from two different channels, making it harder to read out a complete channel. However, owing to the irregular access order of the channel clustering of OFMs and variation of the NZEs, N_{NZEI}, in each channel, it clearly consumes more time and energy to read out all required data based on this storage layout of OFMs. Thus, we filled the memory of the same address with the OFM elements of the same channel through the crossbar and first in first out (FIFO). Assuming that R_step0/1 are two steps of the process that the PE array reads IFMs from memory, the channels can be efficiently accessed according to the sorting order of channel clustering, $I_{i+1,0}$ (OFM $O_{i,0}$ in previous layer i) and $I_{i+1,2}$ for R_step0, $I_{i+1,1}$ and $I_{i+1,3}$ for R_step1. Before sending IFMs to the PE array, we needed to organize IFMs of the same channel to the same PE row to fit the computing flow. Eventually, row_0 and row_1 received IFMs with approximate sparsity ratios, which balanced the computing loads across the PE array and reduced the idle duration of each PE, achieving 1.33× PE utilization rate with little hardware cost.

C. Sparsity Processing of CONV Layers

For CONV layer, we compressed the entire kernel and IFM block to maximize the sparsity of IFMs and weights with bitmap compression format [34], as shown in Fig. 8. Each compressed data block contains the data_length, bitmap, and NZEs. The data_length, N_{NZEI} and N_{NZEW}, record the NZEs, the bitmap records the data zero flags (0 for zero, 1 for nonzero), and NZEs are represented as I_{NZ} and W_{NZ}. To fit with the compression format, we applied weight-oriented [23] computing flow, as shown in Fig. 9. It takes one weight each time to compute with the entire IFM, repeatedly, whose results will be accumulated until the final results are obtained. To obtain OFMs in sparse processing, we first traversed IFMs, then weights, and decompressed the corresponding bitmap

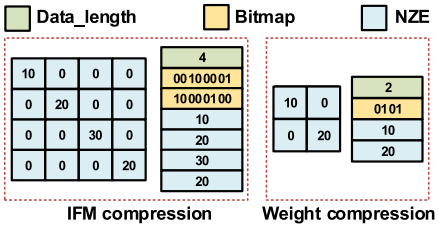

Fig. 8. Compression patterns for the CONV layer.

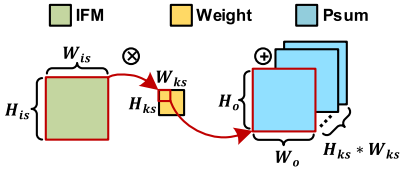

Fig. 9. Weight-oriented dataflow.

into location info, $(I_{\text{row}}, I_{\text{col}})$ and $(W_{\text{row}}, W_{\text{col}})$. Based on that, the output location $[P\text{sum}_{\text{row}}(= I_{\text{row}} - W_{\text{row}}), P\text{sum}_{\text{col}}(= I_{\text{col}} - W_{\text{col}})]$ was calculated and set as invalid when out of the boundary. Next, according to the output location and edge size of the OFM, W_o, we obtained the buffer address, $P\text{sum}_{\text{addr}}(= P\text{sum}_{\text{row}} * W_o + P\text{sum}_{\text{col}})$, of the accumulated partial sums (Psums) for the product of I_{nz} and W_{nz}.

Fig. 10 shows an example. The decompression of IFMs is shown in Fig. 10(a). Assuming $W_o = 3$, we have $I_{\text{NZ}} = \{10, 20, 30, 40\}$, whose location info are $(I_{\text{row}}, I_{\text{col}}) = \{(0, 0), (1, 1), (2, 2), (3, 3)\}$. Fig. 10(b) shows the decompression of kernels. Assuming $W_{\text{NZ}} = \{10, 20\}$, we have the location info, $(W_{\text{row}}, W_{\text{col}}) = \{(0, 0), (1, 1)\}$. The computing flow is shown in Fig. 10(c). Assuming the initial value of Psum is zero, at T_0, we fetch $I_{\text{NZ}} = 10$, $(I_{\text{row}}, I_{\text{col}}) = (0, 0)$, and $W_{\text{NZ}} = 10$, $(W_{\text{row}}, W_{\text{col}}) = (0, 0)$ and calculate $P\text{sum} = 100$, and $P\text{sum}_{\text{addr}} = 0$; thus, value 100 is accumulated in the address 0 of the Psum buffer. Similarly, at T_1, T_2, the

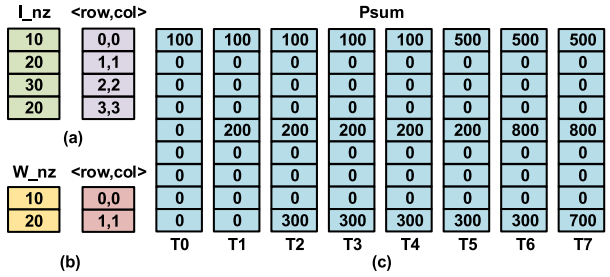

Fig. 10. Sparse computing process of the CONV layer. (a) Compressed IFMs. (b) Compressed weights. (c) Psum results in each cycle.

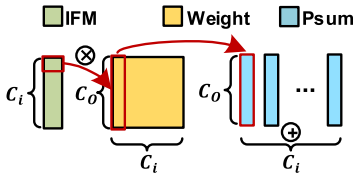

Fig. 11. Process of outer product.

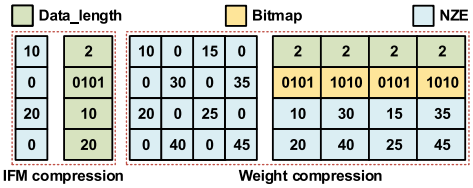

Fig. 12. Compression patterns for FC layer.

value 200 and 300 are accumulated in addresses 4 and 8, respectively. Since $Psum_{addr}$ is out of the boundary at T_3 and T_4, the operations are regarded as invalid. Finally, at T_5, T_6, and T_7, 400, 600, and 400 are accumulated with previous values in addresses 0, 4, and 8, respectively. The entire process takes eight cycles, which is 8× faster than 64 cycles of the original process.

D. Sparse Processing of FC Layers

Because FC layers mainly conduct GEMV and only take up a small proportion of CNN, we mapped the computing flow as the CONV pattern to reduce control overhead. Therefore, we applied the outer product [33] to process FC layers, as shown in Fig. 11. One element of IFM is required each time to compute with the corresponding column of weight matrices, repeatedly, and the results are accumulated until all inputs are traversed. Correspondingly as in the CONV layers, the temporary results of each IC are accumulated to obtain a complete OFM. To efficiently exploit sparsity, IFMs and weights are compressed by column, as shown in Fig. 12. Because the computation is mainly GEMV, the total runtime is determined by the size of weight matrix. Thus, we performed random pruning [19] to maximize weight sparsity. Moreover, for weight loads balancing of FC layers, channel clustering was applied like CONV layers by sorting the NZEs in each weight column, further improving performance.

The sparse process of FC layers is similar to CONV layers. First, the location info of IFMs and weights are decompressed. Then, the nonzero IFM element, I_{NZ} with the location info of I_{col} and nonzero weights of the I_{col} column of weight matrix

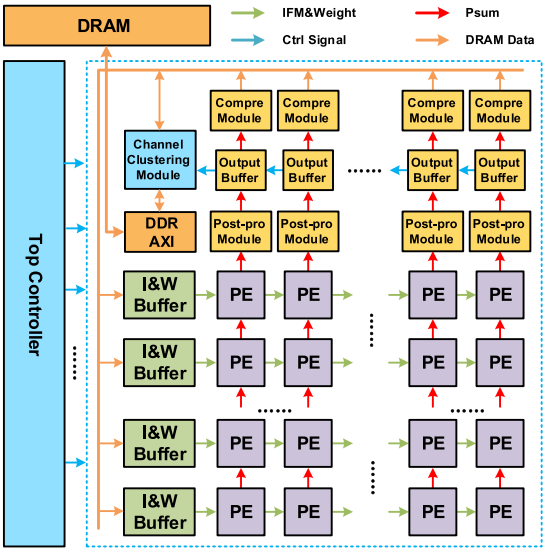

Fig. 13. Top-level architecture.

with the location info of (W_{row}, W_{col}), are fetched. Eventually, we further calculated $Psum_{addr}(= W_{row})$ of Psum Buffer for Psum accumulation.

IV. Architecture

To achieve high speedup and energy efficiency at a low hardware cost, we chose the systolic array as the mainstay. Then, efficiently fitting in the sparsity processing becomes the major issue. We equipped the compressing module (compre module) to store data in compressed format and added the channel clustering module to gather IFM computations with approximate sparsity ratio, balancing the computing workload of IFMs. Accordingly, the decompressing module was designed for IFM and weight inputs to decompress the location info. Moreover, based on our dataflow, we surrounded the array with input, weight, and output buffers to maximize the data reuse and minimize DRAM access. The design of the overall architecture aims to balance the workloads of sparse IFMs and weights in the systolic array, collaborating with load-balancing weight pruning and channel clustering.

A. Overview

As shown in Fig. 13, the top-level architecture mainly consists of input-weight buffers (I&W buffer), a PE array, postprocessing modules (postpro modules), output buffers, compre modules, a top controller, a channel clustering module, and a double data rate-advanced extensible interface (DDR-AXI). Initially, we loaded pruned weights, input images, and parameters directly into DRAMs. Next, the compressed data were fetched and decompressed by I&W buffers to obtain NZEs and location info for the PE array. After parallel MAC operations by PEs, postpro modules take in the outputs to perform ReLU and pooling, whose results were buffered in the output buffers. Before storing to DRAM, the OFMs were compressed by the compress modules to reduce DRAM access. Meanwhile, channel clustering module sorted the OC indexes according to the NZEs in OFMs to gather with approximate

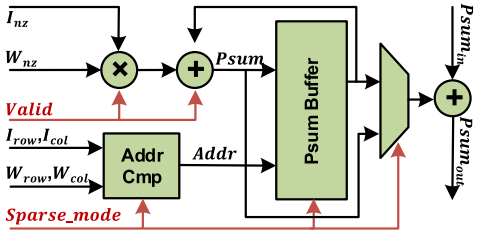

Fig. 14. PE unit.

sparsity. We executed layerwise, and the OFMs of the last layer were classified to output the final results. The top controller managed the entire computing flow and the ON/OFF-chip data were transported through the DDR-AXI interface.

B. On-Chip Buffer

To reduce energy consumption of the data movement through the ON/OFF chip, we needed to buffer IFMs, weights and OFMs for data reusing. Thus, the on-chip buffers mainly consist of I&W buffers and output buffers. The I&W buffer consists of two BRAMs for IFMs and weights, two BRAMs for location info, and a decompressing unit for bitmaps. The output buffer consists of two BRAMs for OFMs. To avoid performance from data loading, we utilized each group of two BRAMs as Ping-Pong buffers, one for DRAM access and the other for PE transporting. The inputs of each row of the PE array were supplied by one I&W buffer, and the outputs of each column were collected by one output buffer. Furthermore, to support different reuse strategies, the storage can be reallocated for IFMs and weights.

C. PE Array

The PE array, in charge of MACs, is composed of $N_{\text{PE}} \times N_{\text{PE}}$ PEs, among which PEs of the same row share IFMs of one IC, and PEs of the same column process OFMs of one OC. Each PE contains an MAC unit, an address computing unit, and a Psum buffer, as shown in Fig. 14. During execution, we multiplied the IFMs with weights and calculated the location info. The results were accumulated with temporarily stored Psums of corresponding addresses in the Psum Buffer. When all ICs of this OFM block were finished, we paused the computation, accumulated the PEs, and finally sent the results to the postpro module. To reduce power consumption, the MAC unit is gated when the IFM/weight is zero or the location info is invalid. Moreover, in the case of low sparsity, we switched the PE array to dense mode and accumulated the products of IFM and weight with the adjacent PE, which can shield the address calculation unit and Psum buffer, thereby reducing the power consumption of PEs.

D. Channel Clustering Module

To balance the load of sparse IFMs in the PE array, we ranked ICs by the NZEs in IFMs such that channels with approximate sparsity ratios were gathered in the channel clustering module, which consists of a channel index buffer, an NZEs number buffer, a ranking unit, a crossbar, and a group of FIFOs. Because only N_{PE} OFMs are produced each time, we used the "divide and rule" principle, ranking each group of OFMs separately. According to the NZEs stored in the NZE number buffer, we sorted the channel indexes with the Merge Sort algorithm in the ranking unit and the results were stored in the channel index buffer. Because the IFM access of DRAM became irregular after channel clustering, we utilized a crossbar and a group of FIFOs to rearrange the OFMs layout, allowing efficient random access of OFMs. When processing the next layer, IFMs (OFMs in previous layer) were read based on ranking indexes and gathered with approximate sparsity ratios to further balance the IFM computing load in PE array.

V. Dataflow

Because IFMs and weights cannot be stored on-chip, we partitioned IFMs into independent subtiles to decrease the on-chip storage requirement. Furthermore, for bandwidth saving and DRAM access reduction, we introduced several data reuse strategies and designed the Adaptive Dataflow Configuration to decide a more suitable dataflow of each layer based on the storage ratios of weights and IFMs. Finally, a mapping algorithm was designed to map various networks on our architecture, based on provided network parameters.

A. IFM and Weight Partition

For a limited on-chip buffer and PE array size, we needed to partition IFMs into independent subtiles in terms of feature map edge size and IC/OC. Based on CONV, there was overlap between subtiles. To reduce overlapping, we preferred a square-shaped partition. Considering the resources of Psum storage, the size of the feature map subtiles was set no larger than $N_{\text{is}} \times N_{\text{is}}$, and the numbers of IFM and OFM tiles on the row/column dimension were $T_{\text{ifm\_row}}$, $T_{\text{ifm\_col}}$ and $T_{\text{ofm\_row}}$, $T_{\text{ofm\_col}}$, respectively. For limited PE array size, we only processed N_{PE} IC and OC concurrently, with tiling numbers of T_{ic} and T_{oc}. To obtain complete OFMs, all the input subtiles convolve with the kernels in turn.

B. Reuse Strategy

To avoid same IFMs and weights being read repeatedly from DRAM and reduce Psum storage, we attempted to consume data as soon as possible and write back only the final results to DRAM. Therefore, we applied the output stationary strategy [35], taking IFMs and weights of different ICs on-chip to compute and accumulate in the same column to obtain the OFM of this OC. Furthermore, based on the intrinsic property of systolic architectures, the same PE row shares one IFM, which actualizes the input stationary strategy [35] and improves the bandwidth utilization rate. As for a single PE, we introduced the weight stationary strategy [35], temporally reusing weights when the entire IFM streams through. In this way, we maximized data reuse by fully applying input, weight, and output stationary strategies, minimizing data movement consumption. For FC layers, because there exists no weight reusing in GEMV operation, the computing time was mainly determined by weight loading. Limited by the bandwidth, we only applied one PE column to process FC layers for energy reduction of the entire system.

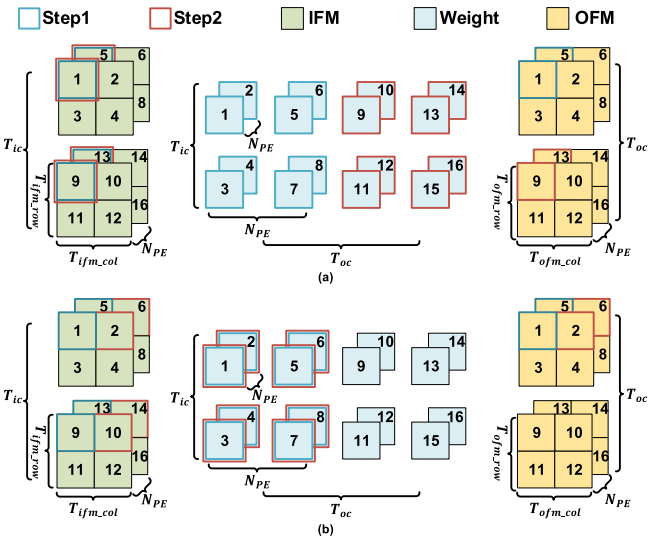

Fig. 15. Adaptive dataflow configuration. (a) RIF. (b) RWF.

C. Adaptive Dataflow Configuration

After data partition, the OFMs were tiled in edge size and channel dimension. Thus, the dataflow of OFMs faces two choices: channel first or edge first. To further reduce DRAM access, we discussed the impact of two dataflows on DRAM access, as shown in Fig. 15. For channel first, we calculated I_1, I_5, I_9, and I_{13} with K_1–K_8 to obtain O_1 and O_5 in Step0, and then we stationed IFMs and traversed weights K_9–K_{16} to obtain O_9 and O_{13}, as shown in Fig. 15(a). This dataflow reuses IFMs and repeatedly accesses weights $T_{\text{ifm\_col}} \times T_{\text{ifm\_row}}$ times; thus, it is called "Reuse-IFM-First (RIF)." Assuming that the storage of IFMs and weights are I_{mem} and W_{mem}, respectively, the total DRAM access, D_{mem}, was $W_{\text{mem}} \times T_{\text{ifm\_row}} \times T_{\text{ifm\_row}} + I_{\text{mem}}$.

On the contrary, if we calculate the subtiles O_1, O_5 first and then O_2, O_6, as shown in Fig. 15(b), we station weights and repeatedly access IFMs T_{oc} times, which is called "Reuse-Weight-First (RWF)." Thus, D_{mem} is $I_{\text{mem}} \times T_{\text{oc}} + W_{\text{mem}}$. Furthermore, when all weights can be stored on-chip, we can station IFMs without loading the weight repeatedly. Therefore, D_{mem} is $I_{\text{mem}} + W_{\text{mem}}$. As DRAM access varies based on different dataflows, to further reduce DRAM access, we applied the adaptive dataflow configuration to decide a more suitable reuse strategy ("RIF" or "RWF") of each layer based on the storage ratios of weights and IFMs. For FC layers, because IFMs can be totally stored on-chip and the weights require no reusing, D_{mem} is $I_{\text{mem}} + W_{\text{mem}}$.

Table I shows an example of the Adaptive Dataflow Configuration on different layers in ResNet-50. Assuming that N_{is} is 7×7 and N_{PE} is 32, we applied different reuse strategies on layer-3, layer-15, and layer-48. In layer-3, $T_{\text{ifm\_row}}$ and $T_{\text{ifm\_col}}$ are both 8; T_{ic} and T_{oc} are both 2. Because weights can entirely be loaded on-chip for their small quantity, we applied the RIF strategy to avoid repeated access of weights. In layer-15, $T_{\text{ifm\_row}}$, $T_{\text{ifm\_col}}$, T_{ic}, and T_{oc} are all 4. The RWF strategy reduces DRAM access by 4.49× compared with the RIP strategy. In layer-48, when $T_{\text{ifm\_row}}$ and $T_{\text{ifm\_col}}$ are both 1, T_{ic} and T_{oc} are both 16, the IFMs are reused prior, achieving 1.16× reduction of DRAM access.

TABLE I
REUSE STRATEGIES IN DIFFERENT CASES

Layer	T_{ic}, T_{oc}, T_{ifm_row}, T_{ifm_col}	I_{mem}	W_{mem}	RWF D_{mem}	RIF D_{mem}
Layer-3	2,2,8,8	196K	36K	/	232K
Layer-15	4,4,4,4	98K	144K	536K	2.35M
Layer-48	16,16,1,1	24.5K	1.99M	2.63M	2.27M

TABLE II
COMPUTING FLOW

Input: parameters of architecture configuration; IFMs; weights;
Output: OFMs

When RIF, $T_{oc\_outter} = 1, T_{oc\_inner} = Toc; otherwise,$
$T_{oc\_outter} = Toc, T_{oc\_inner} = 1$
1 $for(a = 0; a < T_{oc\_outter}; a = a + 1)$
2 $for(b = 0; b < T_{ifm\_row}; b = b + 1)$
3 $for(c = 0; c < T_{ifm\_col}; c = c + 1)$
4 $for(d = 0; f < T_{oc\_inner}; d = d + 1)$
5 $for(e = 0; e < T_{ic}; e = e + 1)$
6 $for(f = 0; f < N_{NZEW\_MAX}; f = f + 1)$
7 $for(g = 0; g < N_{NZEI\_MAX}; g = g + 1)$
8 $Psum+ = W_{NZ} \times I_{NZ}$

D. Mapping Algorithm

Considering dataflow and partition of IFM and weight, to map various networks on Sense, we designed a network mapping algorithm, obtaining architecture configuration parameters through feeding network structure parameters. With configuration parameters loaded, the controller can manage the computing flow of the entire network.

The computing flow description is shown in Table II. The first and fourth row decides the dataflow according to reuse strategy; for "RIF," we finished the OFM computing of all OCs for one output tile before switching to the next output tile; otherwise, we finished computing of all output tiles for one OC before switching to the next OC. The second and third row describe the partition of OFM, with a row number of $T_{\text{ofm\_row}}$ and column number of $T_{\text{ofm\_col}}$. The fifth row drives the accumulation of ICs to obtain the final Psums. The sixth and seventh row represent the NZEs in one tile of IFM and one kernel. $N_{\text{NZEW\_MAX}}$ is loaded as a parameter because the weights from training are fixed and $N_{\text{NZEI\_MAX}}$ is gained during data loading. The eighth row indicates that the PE is performing the MAC operation with NZEs of IFM and weight.

VI. EXPERIMENTS

A. Implementation

We implemented Sense in Verilog and conducted a simulation on Vivado 2021.1. We evaluated our design on the Xilinx ZCU102 at 200 MHz. The original input images, weights, and configuration parameters were stored in an SD card first and then transferred into a processing system (PS)-side DDR4 under the control of ARM CortexA53 processors. Finally, we conducted the CNN inference on FPGA and measured the power through the Maxim Power Tool kit. The reports of resource utilization and power breakdown are shown in Table III.

The size of the PE array was 32×32, achieving a peak throughput of 1024×0.2 GHz = 204.8 GMAC/s. The IFMs and weights were encoded in 16 bits and the Psum was

TABLE III
RESOURCE AND POWER PROPORTION

Module	DSP	LUT	BRAM	POWER
I&W Buffer	0	8 K (1.4%)	320 (17.5%)	1.08 W (10%)
PE Array	1024 (40.6%)	241 K (40%)	0	5.63 W (52%)
Post_pro Module	0	11 K (1.8%)	32 (1.8%)	0.1 W (1%)
Output Buffer	0	4 K (0.7%)	64 (3.5%)	0.1 W (1%)
Compre Module	0	13 K (2.2%)	0	0.1 W (1%)
Channel Clustering	0	6 K (1.1%)	35 (1.9%)	0.3 W (3%)
DDR-AXI	10 (0.3%)	37 K (6.1%)	51 (2.8%)	2.28 W (21%)
Controller	34 (1.3%)	28 K (4.7%)	0	1.19 W (11%)
Total	1061 (42.2%)	348 K (58%)	502 (27.5%)	11 W (100%)

truncated to 16 bits for storage reduction with little accuracy loss through training. To balance the data reuse efficiency and resource overhead, we set the IFM tiling unit as 7×7. Thus, the buffers in the PE array and compress modules were implemented with 64×16-b LUT RAM to minimize on-chip storage. As shown in the resource utilization breakdown shown in Table III, LUTs and DSPs were mainly used to construct the PE array for MAC operation, and the majority of BRAMs were consumed in the I&W buffers for data reuse. Besides, the power and resource of the channel clustering module only takes up a small proportion, indicating low cost for sparse processing in our architecture.

To verify the superiority of Sense, we set some baseline accelerators, as shown below.

1) Swallow [28] is a systolic architecture that utilizes the sparsity of both weights and IFMs in CONV and FC layers and applies a sparse-aware dataflow to optimize data reusing and lower DRAM access. By comparison, Sense achieves higher speedup through load-balancing weight pruning and channel clustering. Furthermore, we further lowered the DRAM access by applying the adaptive dataflow configuration.
2) FESA [29] proposes a software–hardware codesign to address the problem of no-load PEs for sparse networks and reduce the kernel pattern numbers of irregularly pruned networks, significantly improving PE utilization and performance. However, it has not yet been implemented on complex datasets such as ImageNet, for its relatively strict constraints of the pruning method. By comparison, we obtained a similar performance improvement based on weight sparsity, which was also verified on ImageNet.
3) SPOTS [31] is a systolic-array-based accelerator for sparse CNNs, created by building a hardware unit to perform Im2Col transformation of IFMs coupled with a GEMM unit. Moreover, a groupwise pruning method was designed to avoid the potential load imbalance caused by irregularity of sparse weights. By comparison, Sense can achieve higher sparsity of pruned weights with a more coarse-grained pruning method.

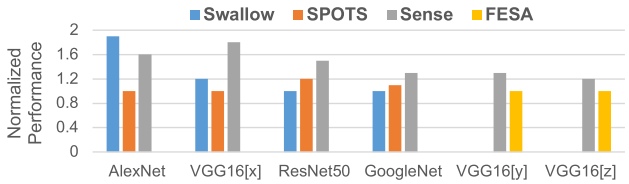

Fig. 16. Performance comparison with Swallow, FESA, and SPOTS in terms of sparse weights.

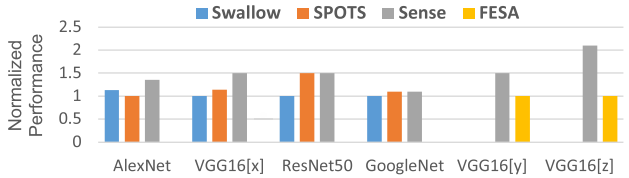

Fig. 17. Performance comparison with Swallow, FESA, and SPOTS in terms of sparse IFMs.

4) Zhu et al. [23], HPIPE [36], and Lu et al. [24] are sparsity-aware CNN accelerators based on FPGA. Unlike systolic dataflow, they process sparse computing in independent PE groups with large logic and BRAM resource overhead. By comparison, we achieved approximate performance with less resource consumption. As for benchmarks, we conducted inference on AlexNet [1], VGG-16 [5], ResNet-50 [7], GoogleNet [6], and compared our design with each baseline.

B. Performance

To show the advantages of sparse processing methods in Sense, we compared our design against Swallow, FESA, and SPOTS on performance, achieving $1\times$–$2.25\times$, $1.95\times$–$2.5\times$, and $1.17\times$–$2.37\times$ speedup, respectively, as shown in Fig. 18. The normalized performance is calculated by P_{se}/P_{ot}, where P_{se} and P_{ot} are the speedup of Sense and other accelerators against dense systolic array. Table IV displays the sparsity ratios of IFMs and weights in the CONV and FC layers. In AlexNet, VGG-16[x] (ImageNet), ResNet-50 and GoogleNet, we cut down the first 34%–56% small elements of each kernel in the CONV layers with load-balancing weight pruning, which can skip all zero elements and achieve $1.5\times$–$2.25\times$ performance improvement. In FC layers, the weight sparsity was 80%; however, only about 60%–80% zero elements can be skipped for the irregularity caused by random weight pruning, achieving $3\times$–$4\times$ performance improvement. In VGG-16[y] (Cifar-10) and VGG-16[z] (Cifar-100), we cut 78% elements of each kernel in CONV layers and 80% elements in FC layers. The columns of Top 1 and Top 5 accuracy show the inference accuracy of each CNN and its corresponding accuracy loss in the brackets. The training structures refer to Pytorch [37].

From the perspective of weight sparsity, in Swallow, the distribution of NZEs in each kernel was irregular, and the computing duration was determined by the least sparse kernel, causing unstable speedup. Although our pruning method may lead to lower sparsity due to the constraints, we managed to maintain the sparsity ratio of each kernel at a certain value, which further balanced the PE loads and maintained a stable

TABLE IV
IFM AND WEIGHT SPARSITY RATIOS AND ACCURACY LOSS OF EACH CNN

Accelerator	CNN type	W_{CONV}	W_{FC}	IFM_{CONV}	IFM_{FC}	Top 1 Accuracy	Top 5 Accuracy
Swallow [28]	Alexnet	87.4%	81.1%	19.0%	71.8%	-	81.3%(-0.3%)
	VGG-16[x]	62.8%	82.5%	39.5%	33.4%	-	88.2%(-0.1%)
	ResNet-50	46.9%	91.5%	46.2%	22.0%	-	90.6%(-0.0%)
	GoogleNet	58.1%	90.7%	44.0%	22.9%	-	91.0%(-1.0%)
FESA [29]	VGG-16[y]	82.5%	-	-	-	91.9%(-0.9%)	-
	VGG-16[z]	80.6%	-	-	-	72.4%(-0.4%)	91.9%(+0.5%)
SPOTS [31]	Alexnet	56.8%	-	34.2%	-	55.3%(-1.5%)	78.6%(-1.3%)
	VGG-16[x]	27.5%	-	49.7%	-	67.2%(-1.1%)	88.2%(-0.2%)
	ResNet-50	31.5%	-	71.1%	-	69.7%(-3.0%)	89.3%(-1.4%)
	GoogleNet	25.1%	-	41.2%	-	66.2%(-2.7%)	87.6%(-1.5%)
Zhu et al [23]	Alexnet	67.1%	-	60.4%	-	-	-
	VGG-16[x]	63.2%	-	60.5%	-	-	-
HPIPE [36]	ResNet-50	85%	-	-	-	71.9%(-4.2%)	90.8%(-2.1%)
Sense	Alexnet	55.6%	80.0%	35.8%	76.3%	55.4%(-0.8%)	77.9%(-0.4%)
	VGG-16[x]	55.6%	80.0%	49.2%	83.2%	70.7%(-0.9%)	90.0%(-0.4%)
	ResNet-50	34.4%	80.0%	46.5%	70.5%	75.4%(-0.7%)	92.6%(-0.3%)
	GoogleNet	33.5%	80.0%	34.7%	60.2%	70.2%(-0.8%)	90.3%(-2.2%)
	VGG-16[y]	77.8%	80.0%	43.6%	47.1%	91.6%(-0.5%)	-
	VGG-16[z]	77.8%	80.0%	57.8%	63.1%	91.8%(-0.3%)	99.8%(-0.0%)

[1] VGG-16[x],VGG-16[y] and VGG-16[z] are verified on dataset of ImageNet, Cifar-10 and Cifar-100 respectively.

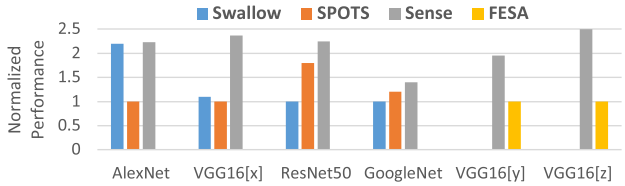

Fig. 18. Overall performance comparison with Swallow, FESA, and SPOTS.

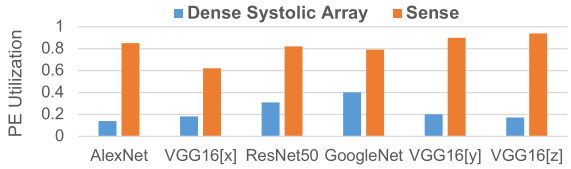

Fig. 19. PE utilization comparison with dense systolic array.

speedup, achieving higher performance. Thus, comparing with Swallow, we obtained 1.53×, 1.49×, 1.33× speedup on VGG-16, ResNet-50, and GoogleNet, respectively, as shown in Fig. 16. However, because the weight sparsity ratio of AlexNet CONV layers in Swallow was 31.8% higher than that of Sense, the speedup brought by computing reduction was slightly higher than load balancing. Thus, Sense was 1.19× slower than Swallow on AlexNet.

The pruning method in FESA balances PE loads by reducing the distribution patterns of zero elements in each kernel; however, it comes with strict constraints. Sense achieves 1.2×–1.3× speedup and loosens the constraints of FESA to implement more complex datasets like ImageNet.

For groupwise pruning in SPOTS, it is more general for sparse CNNs but sacrifices the sparsity of weights owing to the fine-grained pruning method. The sparsity of weights in Sense was 1×–2.02× higher than SPOTS, and we achieved 1.17×–1.8× performance compared with SPOTS.

With respect to IFMs, Swallow experiences imbalanced loads, FESA lacks IFM sparsity handling, and SPOTS can only exploit the sparsity of IFMs in rows with all zeros. To improve sparsity utilization of IFMs, we applied channel clustering and gathered IFMs with approximate sparsity ratios. This treatment further balanced the PE loads and improved performance Thus, we obtained 1.1 ×–1.5 ×, 1.5 ×–2.1 ×, and 1 ×–1.35 × speedup compared with Swallow, FESA, and SPOTS, respectively, as shown in Fig. 17.

On the whole, as shown in Fig. 18, considering sparsity of both weights and IFMs, we obtained 1×–2.25× and 1.17×–2.37× performance improvement on AlexNet, VGG-16, ResNet-50, and GoogleNet compared with Swallow and SPOTS, respectively. Furthermore, we achieved 1.95×–2.5× speedup on VGG-16 for Cifar-10 and Cifar-100, respectively, compared with FESA.

The PE utilization, $U_{\text{PE}}(= P_a/P_i)$, can be calculated by the ratio of the actual performance, P_a, and ideal performance, $P_i(= C_c/T_p)$, where C_c and T_p represent the computing complexity and peak throughput, respectively. Because the sparsity of IFMs and weights in Sense, Swallow, and FESA differ, the total computing complexity is different, which makes PE utilization an inaccurate measure of performance. Thus, to explore the advantages of sparse processing on PE utilization, we compared the dense systolic array and Sense based on the same data sparsity, which indicated that Sense achieves 1.98×–5.53× PE utilization than the dense systolic array, as shown in Fig. 19.

C. Energy Efficiency

Because we executed additional sparse processing compared with Swallow, FESA, and SPOTS, resource and power overhead was incurred, as shown in Fig. 20. To explore whether the benefits brought by these operations outweigh the incurred overheads, we compared the energy efficiency of Sense with Swallow, FESA, and SPOTS, achieving 0.97×–2.25×, 1.3×–1.67×, and 0.94×–1.82× improvement, respectively, as shown in Fig. 21. Because the implementation and PE array size of each accelerator are different, it is difficult to make an absolutely fair comparison. Besides, these accelerators are typical

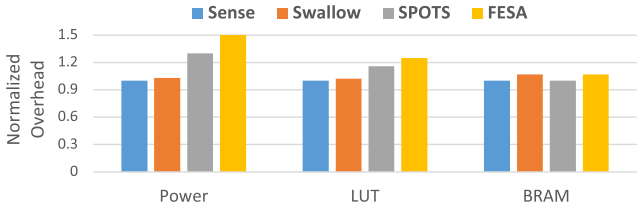

Fig. 20. Power and resource overhead compared with Swallow, FESA, and SPOTS.

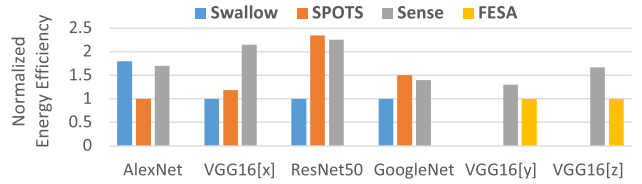

Fig. 21. Energy efficiency comparison with Swallow, FESA, and SPOTS.

systolic arrays, which are similar in overall architecture. Thus, we only considered the overhead related to sparse processing. The normalized energy efficiency was calculated by the ratio of normalized performance and normalized power. Compared with Swallow, Sense achieves its performance improvement primarily from load balancing of weights and IFMs. For weights, we applied load-balancing weight pruning on the software level only and introduced no additional hardware cost. For IFMs, we applied channel clustering, which is performed in the channel clustering hardware module, consuming $6/368 = 1.6\%$ LUT, $35/502 = 7\%$, and 3% power of the overall architecture. As for the sparse handling PE module, it consists of an MAC unit, an address computing unit, and a Psum buffer, which is similar to Swallow. The rest of Sense comprises basic systolic architecture, including no additional cost. Thus, Sense achieves 0.97×–2.18× energy efficiency improvement compared with Swallow.

For FESA, the sparse processing is only offline pruning and its dataflow is the same as the dense systolic array for the regular sparsity distribution of each kernel. Thus, compared with FESA, Sense additionally included the Psum buffer in PE, channel clustering, and compre module, which takes up 25% logic resource and 7% BRAM overhead in total. Because the PE array is always active during the execution period, Sense induced 1.5× power consumption for additional sparse processing. Thus, Sense achieves 1.3×–1.67× improvement in energy efficiency.

As for SPOTS, the Im2Col unit exploits the temporal localities of Psum when the kernels and IFMs are transported vertically and horizontally in the PE array based on the regular computing flow of GEMM, which can save the Psum buffer in each PE compared with Sense. Thus, we consumed 16% more LUT resources and 30% more power than SPOTS, obtaining 0.94×–1.82× energy efficiency.

D. DRAM Access

The power of DRAM access, which takes up a large proportion in the overall architecture, can be optimized by dataflow. Because FESA and SPOTS did not explore dataflow, we only compared the DRAM access of Sense against Swallow, achieving 1.17×–1.8× reduction, as shown in Fig. 22.

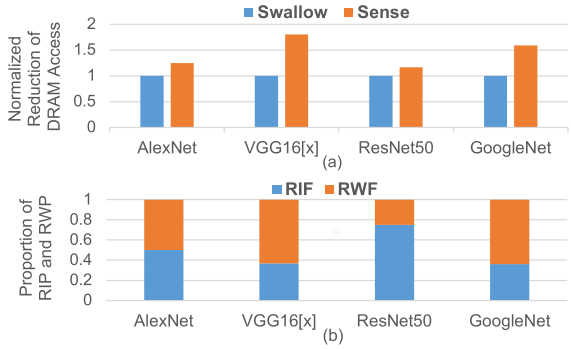

Fig. 22. DRAM access comparison with Swallow. (a) DRAM access reduction. (b) Proportion of RIF and RWF.

The dataflow of Swallow, "compute-in-row," stations one row of IFMs and access weights repeatedly, which is basically the "RIF" mode and inefficient for DRAM access when the weight storage proportion is much larger than IFMs. Compared with the single dataflow of "RIF" in Swallow, we designed a transformation between "RIF" and "RWF" dataflows based on different ratios of weights and IFM storage in each layer, further reducing DRAM access. Therefore, in VGG-16 and GoogleNet, there are nearly 60% layers of "RWF" dataflow, as shown in Fig. 22(b), indicating that these layers can be optimized through dataflow and achieve 1.59×–1.8× DRAM access reduction. However, in AlexNet and ResNet-50, "RIF" accounts for a larger proportion or there exists little difference between "RIF" and "RWF," which leads to a relatively small DRAM access reduction of 1.17×–1.25×.

E. Overall Comparison With Nonsystolic Architectures

Systolic arrays have the intrinsic advantage of processing regular dataflow with low hardware cost but is nowhere near customized nonsystolic architectures when it comes to irregular sparse processing. However, by the treatment of load-balancing weight pruning and channel clustering, Sense achieves similar performance to customized nonsystolic accelerators but consumes less resources and power. An overall comparison on Sense and previous FPGA sparse accelerators conducted by Lu et al. [24], HPIPE [36], and Zhu et al. [23] is shown in Table V. The resource utilization is analyzed in Fig. 23.

In these three architectures, the IFMs and weights are processed in each PE independently, which can naturally avoid imbalanced loads in the systolic array and achieve similar or even a slightly better performance. Lu et al. [24] only address weight sparsity and gains no performance benefits from sparse IFMs. Therefore, Sense achieves 1.05×–1.24× speedup compared with Lu et al. [24]. HPIPE [36] balances the throughput of the cross-layer-pipelined architecture according to the computing complexity and weight sparsity of each layer. The performance advantages of HPIPE over Sense mainly come from higher frequency, more DSPs, and larger weight sparsity. Thus, we implemented Sense (Sense-Plus in Table V) with 72 × 72 PE array on Stratix 10 2800 and improve the weight sparsity for a fair comparison. First, the DSPs and frequency were scaled up by 5× and 2.9×, respectively. Then,

TABLE V
OVERALL COMPARISON WITH THE PREVIOUS FPGA ACCELERATOR

Accelerator	FPGA type	Frequency (MHz)	DSP/LUT/BRAM	Power	Peak Throughput (GMAC/s)	Performance(image/s)/Energy Efficiency(image/J)			
						AlexNet	VGG-16	ResNet-50	GoogleNet
Lu *et al* [24]	ZCU102	200	1144/552K/912	23.6W	204.8	446/18.9	31/1.3	42/1.8	154/6.5
Zhu *et al* [23]	ZCU102	200	1352/390K/1460	15.4W	268.8	987/64.1	46/2.99	57/3.7	215/14.0
HPIPE [36]	S10 2800	580	5022/1735K/11K	-	5850	-/-	-/-	4550/-	-/-
Sense	ZCU102	200	1061/348K/502	11W	204.8	471/43.6	34/3.15	53/4.9	191/17.7
Sense-Plus	S10 2800	580	5225/1930K/1391	-	5939	-/-	-/-	3791/-	-/-

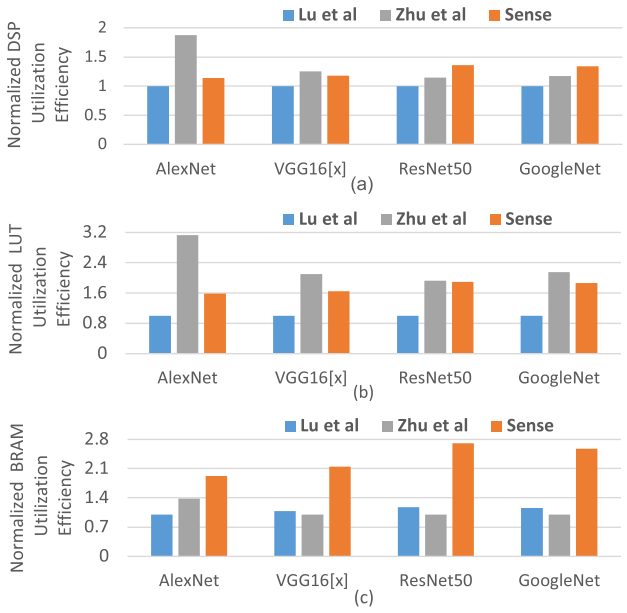

Fig. 23. Resource utilization efficiency comparison with Lu et al. [24] and Zhu et al. [23]. (a) DSP. (b) LUT. (c) BRAM.

we pruned the weight sparsity to 50.4% with 2.4% accuracy loss, achieving 3791 image/s. Although the performance of HPIPE is 1.2× higher than Sense, it is custom-tailored for ResNet-50, lacking flexibility. Zhu et al. [23] exploit both IFM and weights for acceleration. However, its PE number is 1.3× more than Sense, thereby achieving 1.3× peak throughput. Thus, Zhu et al. [23] obtained 1.06×–1.35× performance on VGG-16 and ResNet-50. For AlexNet, Zhu et al. [23] is 2.1× faster than Sense owing to 1.3× DSP, 1.3× reduction of weight NZEs, and 1.5× reduction of IFM NZEs. Because the majority of performance improvement comes from the sparsity of IFMs and weights, we attempted to improve the sparsity of weights through pruning. For 65% sparsity weights and 45% sparsity IFMs, the performance of AlexNet is 587 image/s with 2.8% accuracy loss. For 76% sparsity weights and 37% sparsity IFMs, it can be improved to 721 image/s with 2.9% accuracy loss. Thus, the performance can be improved gradually by increasing the sparsity of weights with accuracy sacrifice.

Independent data supply of each PE incurs huge overhead. In Lu et al. [24], unlike systolic dataflow, massive additional LUT and BRAM resources are consumed by channel multiplexer (CMUX) and tile look-up table (TLUT) modules because of the complex connection between the PE and output buffer for index matching. Zhu et al. [23] require large BRAM resources for OFM storage in each PE, while Sense calculates OFMs in each column. The resource utilization efficiency (performance/cell) is shown in Fig. 23. Compared with Lu et al. [24], Sense is 1.13×–1.37×, 1.66×–1.98× and 1.91×–2.45× more efficient on DSP, LUT, and BRAM utilization, respectively. Compared with Zhu et al. [23], except for AlexNet, Sense performed better, achieving 0.94×–1.17×, 0.82×–1.04× and 2.15×–2.7× resource utilization efficiency on DSP, LUT, and BRAM, respectively, indicating that we obtained approximate resource utilization on LUT and DSP and outstanding utilization on BRAM. As for AlexNet, the performance benefits of Zhu et al. [23] far outweigh our hardware saving, resulting in 1.6× and 2× better performance than Sense on DSP and LUT utilization efficiency. For HPIPE [36], although the LUT and DSP efficiency are 1.25× and 1.32× higher than Sense, it needs to store all weights on FPGA memory, which cannot be implemented for CNNs whose weights exceed the storage of FPGA. Thus, Sense achieves 6.6× higher BRAM efficiency than HPIPE.

With more resources, comes more power. Consequently, Sense consumes 2.15× and 1.43× less power than Lu et al. [24] and Zhu et al. [23], respectively, while HPIPE does not provide power information. Taking the power, resources, and performance all into account, Sense gains 2.31×–2.75×, 0.71×–1.37× energy efficiency improvement compared with Lu et al. [24] and Zhu et al. [23], respectively, as shown in Table V. Besides, when it comes to dense processing, our power consumption is 2.79× and 1.86× less than Lu et al. [24] and Zhu et al. [23], respectively.

F. Design Space Exploration

1) Sensitivity to Sparsity: Sparse processing significantly accelerates computation but comes with extra operations, inducing 30% power overhead. Based on the influence of sparsity ratio on performance and power, we determined a threshold for computing mode converting, whether dense or sparse. In this way, our architecture can handle data of any ratio of sparsity with higher energy efficiency.

Taking a sparsity ratio of 10% as the stride, we achieved different speedup and energy saving by exploiting IFMs and weights of different sparsity ratios separately, as shown in Figs. 24 and 25, respectively. Weight sparsity can be 100% exploited for acceleration based on load-balancing weight pruning, achieving 1×–10× speedup. Because IFM sparsity cannot be constrained by offline training, the efficiency of sparsity exploiting on IFMs is slightly lower than weights, achieving 1×–6× speedup. The energy saving can be calculated by the product of normalized power and speedup,

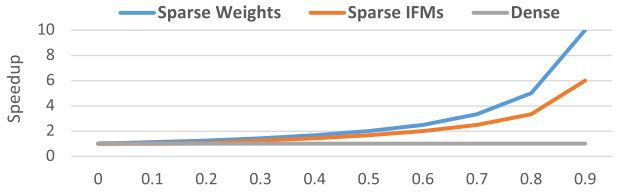

Fig. 24. Speedup by exploiting IFM and weights of different sparsity ratios separately.

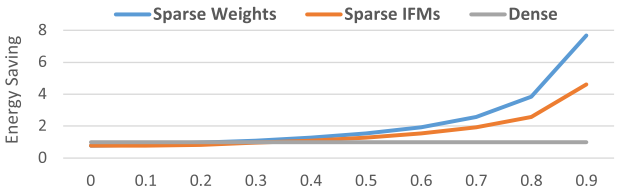

Fig. 25. Energy saving by exploiting IFMs and weights of different sparsity ratios separately.

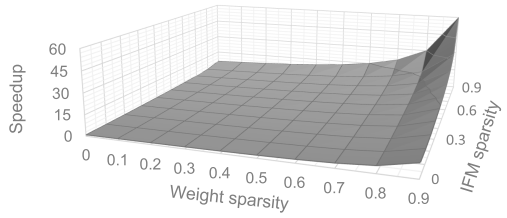

Fig. 26. Speedup by exploiting both IFM and weights of different sparsity ratios.

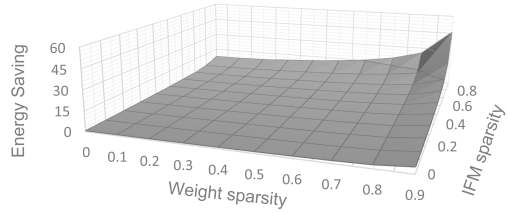

Fig. 27. Energy saving by exploiting both IFM and weights of different sparsity ratios.

indicating 0.77×–7.69× energy reduction on weights and 0.77×–4.62× on IFMs.

Figs. 26 and 27 show the speedup and energy saving by exploiting different sparsity ratios of both IFM and weights. When the sparsity ratios of IFM and weight are beyond 30% and 20%, respectively, the sparse computing mode is more energy efficient. Therefore, we combined dense and sparse processing, and the computing modes were switched based on sparsity ratios.

2) Hardware Effect of Channel Clustering on Efficiency: Because channel clustering is performed on hardware, the efficiency of performance improvement primarily depends on the size of the PE array, N_{PE}, and size of IFM subtiles, N_{is}. For the size of PE array, we processed N_{PE} IFM simultaneously; thus, the computing time is determined by the IFMs with the lowest sparsity based on systolic dataflow. The larger the N_{PE}, the more likely it is that the PE array contains an IFM with low sparsity, which can hinder performance improvement. Thus, we set three different N_{PE}, 8 × 8, 16 × 16, and 32 × 32, to explore their performance on different CNNs. As shown in Fig. 28, the performance of 8 × 8 is 1.05×–1.07× and

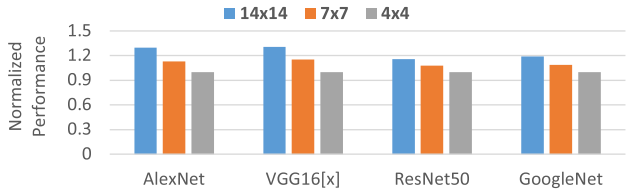

Fig. 28. Normalized Performance of channel clustering based on different sizes of PE arrays.

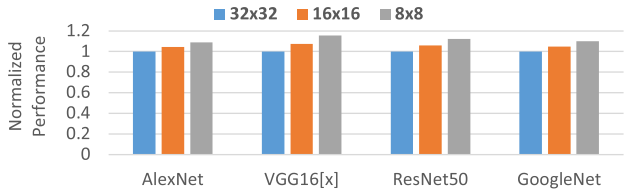

Fig. 29. Normalized Performance of channel clustering based on different sizes of IFM subtiles.

1.1×–1.15× higher than 16 × 16 and 32 × 32, respectively, indicating that PE array size has little effect on performance. The main reason is that PE arrays of small sizes only perform outstandingly on layers with small IC numbers, while the size of IC increases as the layer goes deeper in the CNNs. Thus, we chose 32 × 32 PE array for higher resource utilization of FPGA with little performance loss.

For the size of IFM subtiles, since we gathered IFMs with approximate sparsity by sorting the NZEs in each IFM, the distribution of sparsity of each subtile in the IFM can still be irregular, causing imbalanced workloads when there still exists sparsity difference in the gathered IFM subtiles. The smaller the size of the IFM subtiles, the greater the chance of imbalanced workloads. Thus, to explore the effect of N_{is} on performance, we set three different N_{is}, 14 × 14, 7 × 7 and 4 × 4, and verified on different CNNs. As shown in Fig. 29, the performance of 14 × 14 is 1.1×–1.18× and 1.3×–1.32× higher than 7 × 7 and 4 × 4 on AlexNet and VGG-16, respectively. For ResNet-50 and GoogleNet, the improvement reduces to nearly 1.1× because the IFM size in each layer is relatively small. Therefore, we chose 7 × 7 subtiles for the balance of performance and hardware overhead.

VII. Conclusion

This article proposed a sparse systolic-array-based accelerator, called Sense, for both sparse IFM and weight processing, achieving significant performance improvement with relatively small resource and power consumption. Meanwhile, we applied channel clustering to gather IFMs with approximate sparsity and codesigned a load-balancing weight pruning method to keep the sparsity ratio of each kernel at a certain value with little accuracy loss. This treatment can effectively balance the workloads of sparse IFMs and weights in the systolic array. Furthermore, adaptive dataflow configuration was applied to map various dataflows on Sense, enhancing data reuse, lowering DRAM access, and further reducing system energy consumption. Compared with the previous sparse systolic accelerators, Sense can achieve better performance and energy efficiency with reasonable overhead in addition

to maintaining versatility. Compared with nonsystolic sparse accelerators, Sense can achieve higher resource utilization efficiency.

ACKNOWLEDGMENT

The authors would like to thank Information Science Laboratory Center of USTC for the hardware and software services.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[2] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 4580–4584.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[4] Y. Jia et al., "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, New York, NY, USA, 2014, pp. 675–678, doi: 10.1145/2647868.2654889.

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

[6] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[8] R. Hameed et al., "Understanding sources of inefficiency in general-purpose chips," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 37–47, Jun. 2010, doi: 10.1145/1816038.1815968.

[9] Y.-H. Chen, T. Krishina, J.-S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Nov. 2016.

[10] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[11] S. Yin et al., "A high energy efficient reconfigurable hybrid neural network processor for deep learning applications," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 968–982, Apr. 2018.

[12] J. Park et al., "Faster CNNs with direct sparse convolutions and guided pruning," 2017, *arXiv:1608.01409*.

[13] X. Zhou et al., "Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2018, pp. 15–28.

[14] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 8609–8613.

[15] X. Dong, S. Chen, and S. J. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 4860–4874.

[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*.

[17] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2017, *arXiv:1710.01878*.

[18] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2018, *arXiv:1803.03635*.

[19] S. Han et al., "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 243–254.

[20] S. Zhang et al., "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Jun. 2016, pp. 1–12.

[21] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 1–13.

[22] A. Parashar et al., "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 27–40.

[23] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 9, pp. 1953–1965, Sep. 2020.

[24] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on FPGAs," in *Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2019, pp. 17–25.

[25] H. T. Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, Jan. 1982.

[26] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "14.2 DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 240–241.

[27] Y. Xue, Z. Qian, P. Bogdan, F. Ye, and C.-Y. Tsui, "Disease diagnosis-on-a-chip: Large scale networks-on-chip based multicore platform for protein folding analysis," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–6.

[28] B. Liu, X. Chen, Y. Han, and H. Xu, "Swallow: A versatile accelerator for sparse neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4881–4893, Dec. 2020.

[29] J. Wang et al., "High pe utilization CNN accelerator with channel fusion supporting pattern-compressed sparse neural networks," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[30] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2012, vol. 11, no. 2.

[31] M. Soltaniyeh, R. P. Martin, and S. Nagarakatte, "An accelerator for sparse convolutional neural networks leveraging systolic general matrix-matrix multiplication," *ACM Trans. Archit. Code Optim.*, vol. 19, no. 3, pp. 1–26, May 2022, doi: 10.1145/3532863.

[32] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates Inc., 2016, pp. 2082–2090.

[33] S. Pal et al., "OuterSPACE: An outer product based sparse matrix multiplication accelerator," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 724–736.

[34] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. N. Vijaykumar, "SparTen: A sparse tensor accelerator for convolutional neural networks," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, New York, NY, USA, Oct. 2019, pp. 151–165, doi: 10.1145/3352460.3358291.

[35] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 367–379.

[36] M. Hall and V. Betz, "From TensorFlow graphs to LUTs and wires: Automated sparse and physically aware CNN hardware generation," in *Proc. Int. Conf. Field-Program. Technol. (ICFPT)*, Dec. 2020, pp. 56–65.

[37] M. Paganini. *Pruning Tutorial*. Accessed: Jul. 2021. [Online]. Available: https://pytorch.org/tutorials/intermediate/pruning_tutorial.html