This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

Crane: Mitigating Accelerator Under-utilization Caused by Sparsity Irregularities in CNNs

Yijin Guan, Guangyu Sun, *Member, IEEE,* Zhihang Yuan, Xingchen Li, Ningyi Xu, Shu Chen, Jason Cong, *Fellow, IEEE,* and Yuan Xie, *Fellow, IEEE*

Abstract—Convolutional neural networks (CNNs) have achieved great success in numerous AI applications. To improve inference efficiency of CNNs, researchers have proposed various pruning techniques to reduce both computation intensity and storage overhead. These pruning techniques result in multi-level sparsity irregularities in CNNs. Together with that in activation matrices, which is induced by employment of ReLU activation function, all these sparsity irregularities cause a serious problem of computation resource under-utilization in sparse CNN accelerators. To mitigate this problem, we propose a method of load-balancing based on a workload stealing technique. We demonstrate that this method can be applied to two major inference data-flows, which cover all state-of-the-art sparse CNN accelerators. Based on this method, we present an accelerator, called Crane, which addresses all kinds of sparsity irregularities in CNNs. We perform a fair comparison between Crane and state-of-the-art prior approaches. Experimental results show that Crane improves performance by 27%∼88% and reduces energy consumption by 16%∼48%, respectively, compared to the counterparts.

Index Terms—Computer Architecture, Domain-Specific Accelerator, Deep Learning

1 Introduction

Convolutional neural networks (CNNs) have been widely used in various computer vision tasks, including face detection [1], [2], image classification [3], [4], video analysis [5], [6], etc. To enable the inference of CNNs on resource constrained mobile devices, such as UAVs, surveillance cameras, and smart-phones, researchers have proposed network pruning techniques to remove redundant connections. Such redundant connections exist in various levels of CNNs. First, researchers notice that there are a lot of weights, which are equal or close to zero-values and may be removed. Prior studies have proved that these weights inside CNNs can be pruned up to about 90% with moderate accuracy loss [7], [8]. After that, more aggressive and efficient techniques are proposed to prune at the kernel or even channel level [9], [10], [11]. Recent works [12], [13] propose to prune CNNs at run-time. These pruning techniques lead to great sparsity in CNNs. Furthermore, since a rectified linear unit (ReLU) is widely used as the non-linear activation function, the activation sparsity is about 50%∼70% in many state-of-the-art CNNs [14], [15], [16].

To leverage the sparsity in CNNs, various dedicated accelerator designs are proposed. These approaches focus on avoiding computation with the zeros in sparse weight and activation matrices [14], [17], [18], [19]. In other words, they only exploit weight sparsity and activation sparsity. The effects of kernel sparsity, channel sparsity and run-time sparsity may not be considered. More importantly, we observe that they all suffer from ***low utilization of computational resources due to irregularities of CNN sparsity***. Prior accelerators employ parallel processing elements (PEs) and allocate each PE with a fixed set of weight and activation matrices for computation. Unfortunately, the sparsity irregularities of CNNs result in serious unbalanced workloads among PEs. Thus, PEs assigned with less workloads end up earlier and have to stay idle to wait for the others. Prior works [15], [16] show that this under-utilization problem can result in 30%∼90% performance loss for different CNNs. A real case is shown in Figure 1 for processing five public pruned CNNs [3], [4], [8], [20], [21], [22] on various state-of-the-art accelerators [14], [15], [17], [18]. Gaps between theoretical performance (associated with 100% utilization of hardware PEs) and real performance improvements demonstrate the effect.

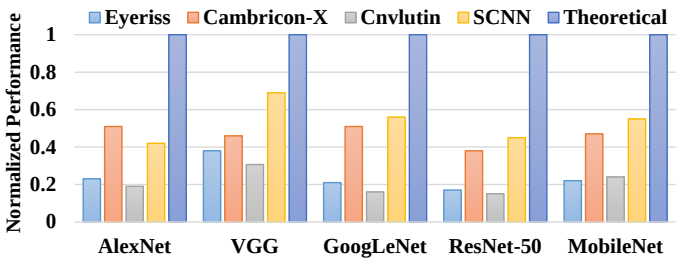


Fig. 1. Real accelerators vs. theoretical cases

Recently, researchers have presented training-assisted approaches [9], [19], [23], [24] to address the irregularity of

- *Y. Guan, G. Sun, Z. Yuan, and X. Li are with Center for Energy-efficient Computing and Applications, Peking University, Beijing 100871, China. E-mail: {guanyijin, gsun, yuanzhihang, lixingchen}@pku.edu.cn*
- *N. Xu is with Baidu Inc., Beijing 100085, China. E-mail: xuningyi@baidu.com*
- *S. Chen is with Harvest Fund, Beijing 100005, China. E-mail: chenshu@jsfund.cn*
- *J. Cong is with University of California at Los Angeles, Los Angeles, CA 900095, USA. E-mail: cong@cs.ucla.edu*
- *Y. Xie is with the SEAL Laboratory, University of California at Santa Barbara, Santa Barbara, CA 93106, USA. E-mail: yuanxie@ucsb.edu*

Manuscript received October XX, 2019; revised XXX XX, XXXX.

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

weight sparsity. The basic idea is to prune similar amount of weights or even the weights at the same positions inside a kernel. However, they have following limitations. First, these approaches apply specific constraints or rules in the CNN pruning process, which induce additional training efforts and make the hyper-parameters fine-tuning even harder. Second, prior research has demonstrated that such an approach can result in accuracy loss [23], [24] so that less number of weights may be pruned. Third, these approaches only focus on addressing the weight irregularity [19]. They ***cannot handle the sparsity irregularities caused by kernel pruning, channel pruning and run-time pruning techniques*** [9], [11], [12], [13], [25].

To overcome these limitations, we propose a hardware level load-balancing method to mitigate the under-utilization problem caused by all kinds of sparsity irregularities. Our approach improves overall utilization by dynamically "stealing" workloads among PEs. Since our approach is hardware-based, it needs no additional training efforts and brings no accuracy loss. Based on this solution, we also propose an accelerator architecture, called Crane. The key contributions of this work can be summarized as follows:

- We present a comprehensive review of sparsity in different levels of CNNs. The effects of their irregularities on two types of data-flows and corresponding state-of-the-art accelerators are analyzed in detail.
- We propose a hardware level load-balancing method to address the under-utilization problem. We demonstrate how to apply this method to two data-flows.
- We introduce Crane architecture based on the load-balancing method and present its design in detail. We also analyze the design issues of Crane architecture, and provide corresponding solutions.
- We perform a fair comparison of similar-area implementations between Crane and state-of-the-art prior approaches using a cycle-accurate simulator. We also provide design space exploration and sensitivity analysis to reveal more insights.

The rest of this paper is organized as follows. Five kinds of CNN sparsity and their effects on prior accelerators are reviewed in Section 2. In Section 3, we analyze the under-utilization in two types of data-flows and propose how to mitigate it with our load-balancing method. Section 4 presents the architecture of Crane in detail, and discusses about design issues. Evaluation methodology and results are discussed in Section 5. Section 6 presents related works, followed by a conclusion in Section 7.

2 Preliminary

In this section, we first introduce different kinds of sparsity in CNNs. Then, we introduce state-of-the-art accelerators for sparse CNNs and analyze their limitations.

2.1 Sparsity in CNNs

There exist five kinds of sparsity in CNNs: weight sparsity, channel sparsity, kernel sparsity, run-time sparsity and activation sparsity. The details of these five kinds of sparsity are introduced as follows.

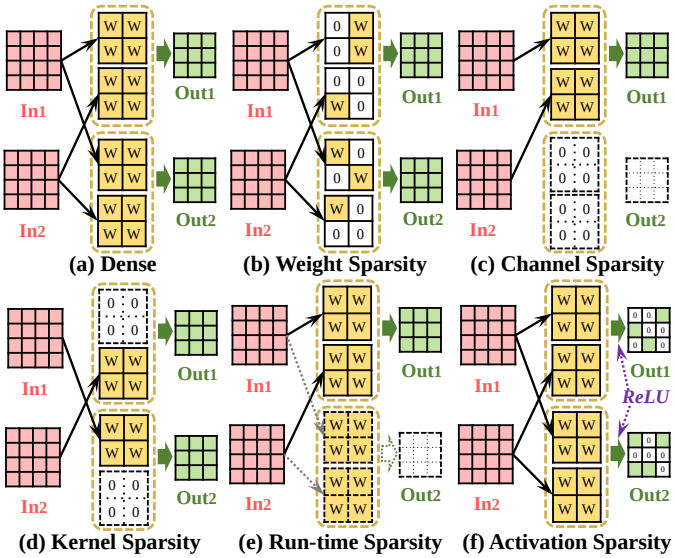


Fig. 2. Different kinds of sparsity in CNNs

Weight Sparsity. We call the fraction of zero-values in weight matrices as weight sparsity. Weight sparsity mainly comes from weight pruning techniques proposed in prior works [7], [8]. During weight pruning, if the absolute values of weights are lower than a threshold, these weights are pruned (i.e., set to zeros) from the convolutional layer. Then, the whole CNN gets re-trained to recover accuracy. Weight sparsity is demonstrated in Figure 2 (b).

Channel Sparsity. Channel sparsity is caused by channel level pruning (a.k.a. filter pruning). If a channel is pruned, all incoming and outgoing weights to/from the corresponding channel are removed. Channel pruning has been extensively researched in recent works. Channels can be pruned based on their corresponding filter norm [26], average percentage of zeros(APoZ) in the output [27], structured sparsity learning (SSL) [9], etc. A demonstration can be found in Figure 2 (c).

Kernel Sparsity. Similarly, kernel sparsity is induced by kernel level pruning. It can be considered as a fine-grain optimization of channel pruning. As shown in Figure 2 (d), only a portion of connections (weight kernels) between an input channel and an output channel are removed during pruning. Most channel pruning methods can be used in kernel pruning. Other methods (e.g. Group sparsity method [9], [10], feature reconstruction error method [11], [28], [29], and Taylor expansion method [25]) can be used in kernel pruning with minor modifications.

Run-time Sparsity. Prior pruning techniques [7], [8], [26] produce a pruned CNN with a fixed structure for all input samples. Recent works [12], [13] propose to preserve original CNN structures and perform channel pruning at run-time according to different input samples. Compared to prior approaches, run-time pruning achieves better trade-offs between performance and accuracy . We call the sparsity caused by run-time pruning as run-time sparsity, and a demonstration of run-time sparsity is shown in Figure 2 (e).

Activation Sparsity. Activation sparsity represents the zero-values in activation matrices. As shown in Figure 2 (f), it mainly comes from the activation functions applied to

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

outputs of convolutional layers. Before being forwarded to the next layer, an element-wise activation function operates on each output of current layer to generate the activation matrices. The mostly employed activation function is rectified liner unit (ReLU) in recent years [3], [4], [21]. ReLU results in high activation sparsity by converting all the negative outputs into zeros.

Obviously, all kinds of sparsity result in ***network irregularities*** in CNNs, which may downgrade inference efficiency of CNN accelerators. Since conventional accelerators assign fixed workloads to each processing element (PE), these irregularities will cause unbalanced computation tasks among PEs. Our further quantitative analysis in Section 5 shows that, the irregularity problem causes 30%∼90% performance loss for Cambricon-X [17] and SCNN [15] on different CNNs.

2.2 Limitations of Existing Accelerators

Recently, several accelerators [14], [15], [17], [18], [19] have been proposed to improve inference efficiency by addressing the sparsity in CNNs. In Table 1, we compare these approaches, in respect of various features. These features include (1) whether all kinds of sparsity are handled and employed for computation reduction (*Handle Sparsity*); (2) whether irregularities caused by different kinds of sparsity are mitigated to improve the hardware utilization (*Mitigate Irregularity*); (3) whether weight encoding or activation encoding is applied to save storage and reduce memory bandwidth requirements (*Encoding*); (4) whether the approach is a pure hardware level solution (*Pure HW*), and (5) whether the approach can keep accuracy (*Accuracy*).

Eyeriss [18] encodes activation matrices for energy savings. It suffers from weight-level, channel-level, kernel-level, and run-time sparsity and irregularities. Similarly, Cnvlutin [14] exploits activation sparsity only and suffers from all kinds of irregularities except that of activation sparsity. Cambricon-X [17] can handle weight sparsity, channel sparsity, kernel sparsity, and run-time sparsity. It stores sparse weight matrices in an encoded format to save storage and reduce memory access pressure. However, these sparsity irregularities are not solved so that the PE under-utilization is aggravated. SCNN [15] exploits every kind of sparsity, but suffers from under-utilization caused by the irregularity of activation sparsity. Since Cambricon-S [19] performs structured pruning to address weight irregularity, it is affected by channel sparsity and kernel sparsity. Besides, it is not a pure hardware level solution. It involves extra constraints in training, which may cause accuracy loss according to prior studies [23], [24]. In addition, it fails to leverage the benefits of activation encoding and run-time pruning.

To overcome the limitations of prior approaches, we decide to propose a hardware-based, high utilization approach for accelerating sparse CNNs. Our approach addresses all kinds of sparsity and stores both weight and activation in an encoded format. More importantly, ***we employ a hardware-based load balancing scheme to dynamically mitigate PE under-utilization caused by various irregularities.***

3 Idea of Load-balancing

As addressed in previous section, effects of irregularities in CNNs on an accelerator are related to its data-flow of processing inference. Data-flows of state-of-the-art accelerators can be categorized into two types: *Input Sharing* and *Weight Sharing*. In this section, we will explain under-utilization problems in these two scenarios and introduce how to mitigate them with a load-balancing method.

3.1 Input Sharing Data-flow

The *Input Sharing* data-flow is illustrated in Figure 3. As suggested by its name, input data (i.e. activation matrices) are broadcast to all PEs during inference. For each round, $K \times K \times CH_{in}$ input activation data (a.k.a an input patch) are broadcast to all PEs. Each PE generates the results for a part of output activation matrices (CH_{out}/N output channels in Figure 3). As a result, each PE is responsible for processing this input patch with $CH_{in} * CH_{out}/N$ weight kernels ($K \times K$ each) for each round. These weight kernels are cached inside a PE till all related output activation results are computed. As shown in Figure 3, each input patch is fetched in a sliding style. The arrows indicate the order of fetching input patches and generating output patches. Many sparse CNN accelerators employ the *Input Sharing* data-flow, such as Cambricon-X [17], Cambricon-S [19], Cnvlutin [14], etc.

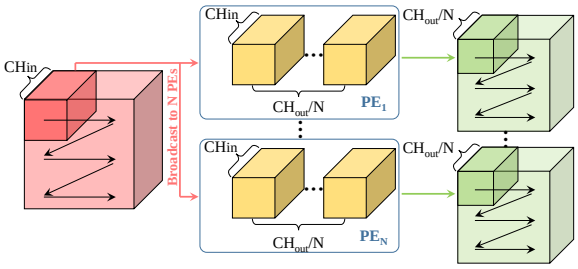


Fig. 3. *Input Sharing* data-flow

Each PE is expected to process the same amount of workloads, as they are assigned with the same number of weight kernels. Obviously, due to irregularities caused by weight sparsity, channel sparsity, kernel sparsity and run-time sparsity, the workloads can be unbalanced among PEs. Thus, PE with less workloads are under-utilized. Figure 4 (a) is an example to show this under-utilization problem. This simplified convolutional layer has two input channels and four output channels. The size of each input channel is 4×4, and the size of each weight kernel is 2×2. Since the convolution stride is set to one, the size of each output channel is 3×3. We have two parallel PEs to process this convolution. Each PE is equipped with two parallel multipliers and one adder, which enable each PE to process two multiplications and one addition at a time.

As shown in Figure 4 (a), PE_1 is responsible for processing output channel 1 and 2, and PE_2 is responsible for output channel 3 and 4. The first input patch is fetched and broadcast to two PEs. According to the activation sparsity and weight sparsity, the effectual workloads (non-zero computation) and computing process of each PE are shown in the table in Figure 4 (a). It takes one cycle and three cycles for PE_1 and PE_2 to process this input patch, respectively. Thus, PE_1 is idle for two cycles to wait for PE_2.

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

TABLE 1
Comparison of existing accelerators for sparse CNNs

Design	Handle Sparsity / Mitigate Irregularity					Encoding		Pure HW	Accuracy
	Weight	Channel	Kernel	Run-time	Activation	Weight	Activation		
Eyeriss [18]	✘ / ✘	✘ / ✘	✘ / ✘	✘ / ✘	✔ / ✔	✘	✔	✔	✔
Cnvlutin [14]	✘ / ✘	✘ / ✘	✘ / ✘	✘ / ✘	✔ / ✔	✘	✔	✔	✔
Cambricon-X [17]	✔ / ✘	✔ / ✘	✔ / ✘	✔ / ✘	✘ / ✔	✔	✘	✔	✔
SCNN [15]	✔ / ✔	✔ / ✔	✔ / ✔	✔ / ✔	✔ / ✘	✔	✔	✔	✔
Cambricon-S [19]	✔ / ✔	✘ / ✘	✘ / ✘	✘ / ✘	✔ / ✔	✔	✘	✘	✘
Ours	✔ / ✔	✔ / ✔	✔ / ✔	✔ / ✔	✔ / ✔	✔	✔	✔	✔

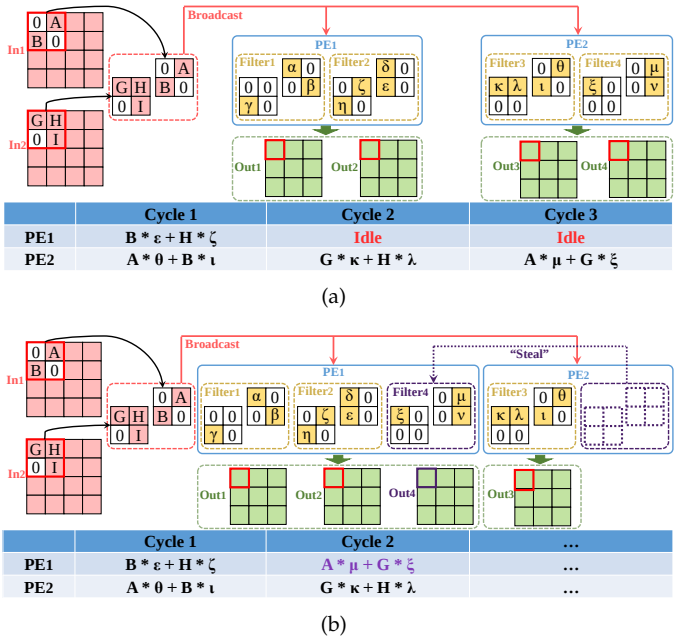


Fig. 4. Simplified convolution processing using *Input Sharing* data-flow

To address the under-utilization problem, we employ a dynamic load-balancing scheme, which is similar to the work stealing strategy [30] applied to schedule multi-thread computer programs. When a PE completes its own workloads, it can "steal" weight kernels from another PE to help.

As illustrated in Figure 4 (b), when PE_1 completes results for the two output channels assigned to it, PE_2 is still busy working on output channel 3. As a result, PE_1 can fetch the weight kernels corresponding to output channel 4. In this way, PE_1 "steals" a portion of workloads from PE_2, and generates the results for output channel 4. By applying this dynamic load-balancing scheme, it takes only two cycles to finish the processing of this input patch. As shown in Figure 4 (b), both PE_1 and PE_2 work at full utilization, and the overall performance is improved by 50%.

3.2 Weight Sharing Data-flow

Figure 5 illustrates the *Weight Sharing* data-flow. It broadcasts weights to all of the PEs during CNN inference. As shown in Figure 5, each PE generates the results for a tile of output activation data ($R_{out}/N \times C_{out}$ in each channel). Thus, the input activation matrices are partitioned for parallel processing among PEs. For each round, CH_{in} weight kernels ($K \times K$ each) are broadcast to all PEs. Each PE is responsible for generating $R_{out}/N \times C_{out}$ output data. Once the current set of weight kernels is processed, the next set corresponding to the following output channel is broadcast. The arrows show the order of weight set fetching and output generating.

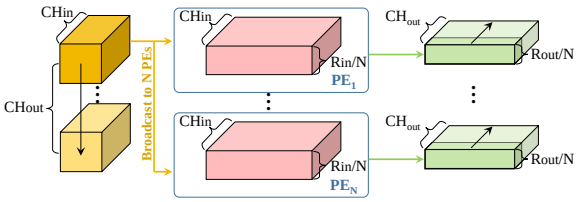


Fig. 5. *Weight Sharing* data-flow

Due to the sliding style of a convolution window, *Weight Sharing* data-flow has to duplicate some input activation data or exchange partial sums across adjacent output tiles to deal with data dependency at the edges of tiles (also called "halos" [31] [15]). Prior accelerator designs, like SCNN [15], employ the *Weight Sharing* data-flow.

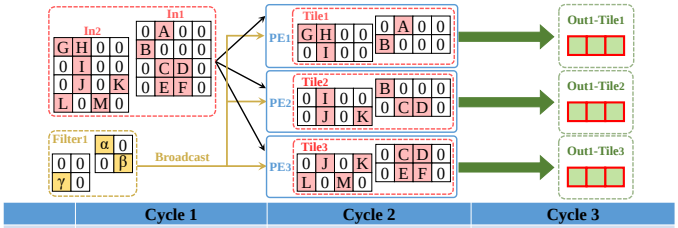
Due to the irregularity of activation sparsity, workload imbalance among PEs also happens in the *Weight Sharing* data-flow. In Figure 6 (a), we use an example to demonstrate the under-utilization problem. The configuration of this convolutional layer is the same as that in previous subsection. The PE configuration is also kept, but the number is increased to three. Each PE generates the results for a tile of output data. In this example, each output tile contains one row of output data. The "halos" of *Weight Sharing* data-flow are tackled by duplicate dependent input activation data in adjacent PEs. We assign Row_1 and Row_2 of inputs to PE_1, Row_2 and Row_3 to PE_2, and Row_3 and Row_4 to PE_3. We show the details of processing the first weight set (a.k.a. filter). The effectual workloads and computing process of each PE are also listed in Figure 6 (a). It takes one, two, and three cycles for PE_1, PE_2, and PE_3 to complete their assigned workloads, respectively.

The idea for load-balancing is similar. In Figure 6 (b), PE_1 finishes generating the output results assigned to it after $Cycle\ 1$, while PE_2 and PE_3 are still working on their own workloads. Since PE_3 has the most workloads left after $Cycle\ 1$, PE_1 can fetch the third input patch of $Tile_3$ to generate corresponding output data. Figure 6 (b) shows that the workloads of processing the third element in $Tile_3$ are "stolen" by PE_1, and workloads are balanced among PEs.

Obviously, the pivot of our load-balancing method is the control of workload stealing. Extra designs should be induced to enable weight/activation stealing and writing back results of stolen workloads. The details of the architecture design are introduced in the next section.

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers



	Cycle 1	Cycle 2	Cycle 3
PE1	A * α + I * γ	Idle	Idle
PE2	B * α + C * β	D * β + J * γ	Idle
PE3	E * β + L * γ	C * α + F * β	D * α + M * γ

(a)

	Cycle 1	Cycle 2	…
PE1	A * α + I * γ	D * α + M * γ	…
PE2	B * α + C * β	D * β + J * γ	…
PE3	E * β + L * γ	C * α + F * β	…

(b)

Fig. 6. Simplified convolution processing using *Weight Sharing* data-flow

4 Crane Architecture

In this section, we use the *Input Sharing* data-flow as an example to introduce Crane architecture. The overall architecture is presented first. Then, we introduce the design considerations and details of each module. We also provide analysis on design issues of Crane architecture.

4.1 Overall Architecture

The overall architecture of Crane is depicted in Figure 7. The whole accelerator employs a DRAM as the external memory, and the *DMA* is responsible for data movement operations between on-chip RAMs and off-chip DRAM. *Processing Unit* consists of N PEs to perform CNN inference in parallel with the *Input Sharing* data-flow. The weights for convolution are stored in *Weight RAM*, and input activation patches are broadcast into input buffers residing in each PE. Both input activation and weight matrices are stored in an encoded format. As a result, only non-zero activation data and weights can be fetched by *DMA* and stored on-chip for convolution processing. *Output RAM* stores the results generated by all PEs, and transfers these results to *Output Unit* for after-convolution processing, which includes activation function, pooling, and encoding. After that, these output results are written back to DRAM.

The key modules of Crane, which enable load-balancing, are *Workload Scheduler*, *Weight RAM Controller*, and *Output RAM Controller*. *Workload Scheduler* monitors the run-time processing status of each PE, and produces schedule information to *Weight RAM Controller* and *Output RAM Controller*. These two controllers use the schedule information to control the weight fetching and output storing, respectively. These two control modules manage the workload stealing across different PEs. *Processing Controller* is the global controller to generate detailed control signals for other modules.

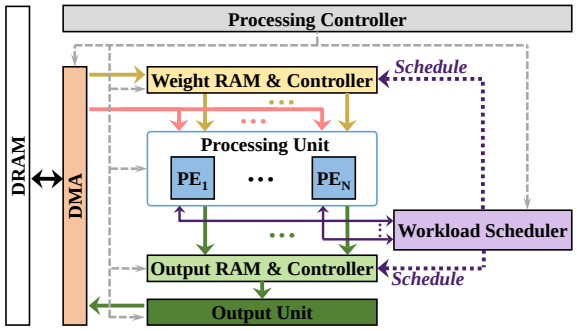


Fig. 7. Overall architecture of Crane

4.2 Processing Element (PE)

The architecture of a PE in Crane is illustrated in Figure 8. It consists of an *Input Buffer*, *Weight Regs*, an *Index Selector*, parallel multipliers and an adder tree. *Input buffer* stores the input activation patch broadcast to all PEs, including both non-zero values and indices. Non-zero weights and their indices are fetched from *Weight RAM* to *Weight Regs*. Each non-zero activation or weight is paired with an index that indicates the location of this value in the original format.

To improve performance and save energy, PEs need to skip ineffectual multiplications. *Index Selector* leverages input indices and weight indices to find the indices of effectual multiplications (both input activation and weight are non-zero values). The selected indices are used to select non-zero "activation-weight" pairs from *Input Buffer* and *Weight Regs*. All the effectual "activation-weight" pairs are fed into the multipliers to perform multiplications in parallel. The products are summed up together by an adder tree to minimize latency. The accumulation result is fed into the last-level adder or *Output RAM* depending on whether the computation for current output has finished or not. Since *Index Selector* needs to traverse indices of input data during index selection, it can easily judge if current accumulation result should be fed into *Output RAM*.

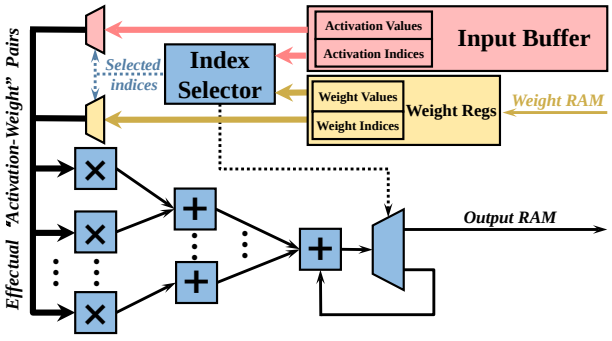


Fig. 8. Detailed architecture of a Crane PE

4.3 RAM and RAM Controller

Crane employs *Weight RAM* and *Output RAM* to store weights and output data of all PEs, respectively. Both RAMs are equipped with RAM controllers, which are in charge of managing the data accesses to each RAM. The detailed design of *Weight RAM* is illustrated in Figure 9. Crane

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

employs N parallel PEs, and each PE needs to access its own weights. As shown in Figure 9, to enable independent accesses by N parallel PEs, the whole RAM is evenly divided into N sub-buffers. In a traditional accelerator design, PE only needs to access its own sub-buffer. In order to enable dynamic workload stealing in Crane, we add an extra crossbar between PEs and *Weight RAM*. Then, a PE can access weights assigned to others and help processing. As a result, these memory accesses are not related to the distance between different PEs. With the crossbar, stealing workloads does not require remote memory accesses between different PEs, it only requires a new schedule on the crossbar between PE array and on-chip RAMs. The connections in the crossbar are controlled by the schedule generated by *Workload Scheduler*. Note that the design of *Output RAM* is similar.

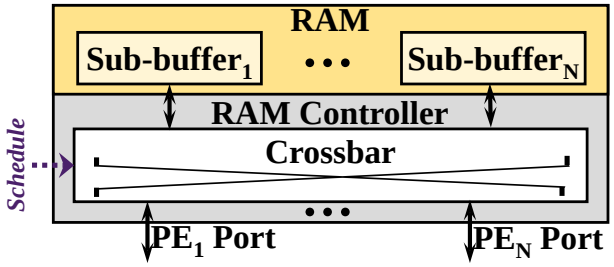


Fig. 9. Detailed architecture of *Weight RAM*

4.4 Workload Scheduler

Workload Scheduler is the key module for implementing dynamic load-balancing scheme. It works as the central control module to detect workload imbalance and perform load-balancing. Figure 10 shows that *Workload Scheduler* decouples imbalance detecting and load-balancing with two sub-modules: *Imbalance Detector* and *Schedule Generator*.

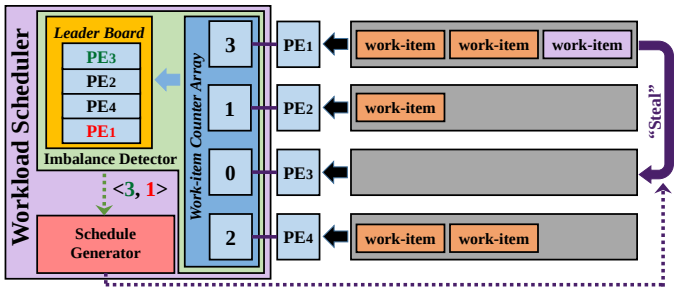


Fig. 10. Illustration of a *Workload Scheduler*

With the *Input Sharing* data-flow, each PE is originally assigned with a subset of the weights to process the convolution results for a patch of output data. If we define the processing task for one output channel as a work-item, the number of work-items assigned to each PE is identical before processing. In *Imbalance Detector*, we set one hardware counter for each PE to record the run-time number of work-items remaining to be processed. During CNN processing, *Imbalance Detector* monitors the state of every PE and updates the corresponding hardware counter. Once a PE finishes processing a work-item, it informs *Imbalance Detector* by pulling up a "valid" signal. Then, *Imbalance Detector* subtracts the corresponding counter by one.

To enable imbalance detecting, *Imbalance Detector* updates a *Leader Board* every time a PE finishes processing one work-item. This *Leader Board* is actually a list of PEs sorted in the ascending order of all the counters' values. Maintaining the *Leader Board* only requires registers for saving PE indices and corresponding logics for updates. If a work-item counter's value is larger than zero, the corresponding PE has not finished the workloads assigned to it. This PE continues processing the next work-item. However, if its counter value becomes zero, the PE has finished processing all its work-items and becomes an idle PE. Then, *Imbalance Detector* checks processing status of the PE at the bottom (i.e. slowest PE) of the *Leader Board*. If its counter value is larger than one, a workload imbalance is detected. The operations involved in *Imbalance Detector* are only updating *Work-item Counter Array* and *Leader Board*. *Imbalance Detector* simply consists of work-item counters and corresponding logics for maintaining the *Leader Board*.

Imbalance Detector will inform *Schedule Generator* of the detected imbalance. *Scheduler Generator* will schedule a work-item from the slowest PE to the idle PE for help. Thus, a workload stealing occurs. *Workload Scheduler* needs to control the crossbar properly so that weights are fetched from sub-buffer of slowest PE to idle PE for processing.

All the workload balancing operations are performed at run-time. We consider both weight sparsity and activation sparsity in Crane. Since activation sparsity is generated dynamically, the effectual workloads for each PE can only be determined at run-time. As a result, no workload balancing can be done offline. However, if we consider weight sparsity only, all the effectual workloads for each PE are fixed once the CNN model is determined. In this way, we can detect the workload imbalance offline through software-based simulation, and we can generate corresponding schedules for crossbars to balance the workloads.

When workload stealing occurs, two PEs need weights from the same sub-buffer of *Weight RAM*. Thus, port access conflicts may happen. It is solved by giving higher priority to the slowest PE. To this end, a conflict will cause one cycle of stall in PE that steals work-item. When multiple PEs have finished processing their own work-items, access conflicts may occur because all these finished PEs may try to steal work-items from the same PE. *Scheduler Generator* addresses the conflicts by insuring only one work-item movement can happen at a time. In other words, workload stealing with port access conflicts needs to be issued in serial.

4.5 Output Unit

Output Unit is in charge of after-convolution processing including activation function, pooling operations (if needed), and activation encoding. *Output Unit* performs *ReLU* function on output results in the *Output RAM*, and generates output activation data. Pooling functions mainly operate on a window of output data, and sample one element by the maximum (max-pooling) of the data inside this window. In the *Output Unit*, we implement pooling functions with dedicated logic for data accessing and comparison.

Since Crane exploits activation sparsity for performance gains and energy savings, *Output Unit* needs to encode

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

these data to eliminate all zero values. Selecting encoding method is orthogonal to the accelerator design. Crane's encoding method is shown in Figure 11, which is close to data encoding methods in prior works [15], [16], [32].

Figure 11 shows an example of encoding two output channels. The encoded data is composed of a *Value Vector* (denoted as V) and an *Index Vector* (denoted as I). A counter records the number of non-zero elements. During activation encoding, *Output Unit* traverses output data in a channel-major order. We use channel-major mapping because it avoids the frequent coordinate computing in conventional row-major mapping, while accessing a window in different input channels. The number of zeros that emerge between two non-zero values is recorded in I, i.e., $I[i]$ equals to the number of zeros between $V[i-1]$ and $V[i]$. The encoding method for weights is the same as that of activation data.

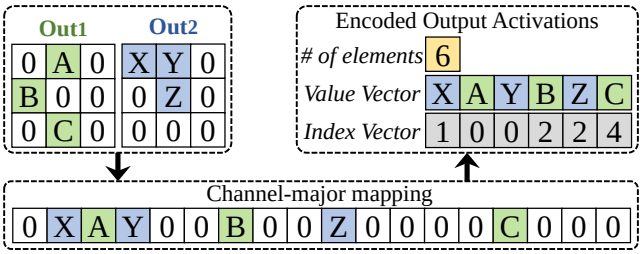


Fig. 11. Encoding method in Crane

4.6 DMA

In Crane, *DMA* is in charge of the off-chip memory access. For large convolutional layers, Crane divides input activation, weight and output activation matrices into smaller tiles. The size of each tile can fit into the capacity of corresponding on-chip RAMs. Once an input activation tile or weight tile has been fully re-used for convolution processing, *DMA* will load the next tile for remaining processing. Accordingly, once Crane finishes generating all the activation data within an output tile, *DMA* will offload the output tile to DRAM. To pipeline the off-chip memory access and convolution processing, all on-chip RAMs employ double-buffering design and operate in a ping-pong manner.

4.7 Processing Controller

Similar to prior works [15], [19], Crane employs a global controller, *Processing Controller*, to control the processing of all other modules. In *Processing Controller*, we store detailed configurations for convolution processing in a buffer. *Processing Controller* configures other modules to perform data transmissions and convolution processing accordingly.

4.8 Design Issues of Crane Architecture

In this subsection, we analyze several design issues of Crane architecture, and provide corresponding solutions.

4.8.1 PE granularity

We define PE granularity as the number of parallel multipliers residing in a Crane PE. According to the PE architecture, PE granularity also indicates the level of parallelism of a Crane PE. Thus, small PE granularity is a bad choice because it exploits low level of parallelism, and results in poor performance. However, large PE granularity may not improve the overall performance much. The reason is that there would be idle cycles of multipliers when the amount of workloads assigned or stolen cannot fully utilize all the parallel multipliers. It is also called intra-PE fragmentation in prior work [15]. As a result, we need to explore the design space carefully, and choose the optimal PE granularity while designing Crane's PEs. Detailed evaluation results of PE granularity will be presented in Section 5.2.4.

4.8.2 Load-balancing granularity

Since each PE steals one work-item at a time in Crane, the size of work-item determines the granularity of load-balancing. We define load-balancing granularity of Crane as the number of weight kernels in a work-item. More fine-grained granularity like intra-kernel granularity is not chosen to avoid complicated partial results transmissions across PEs and performance loss caused by intra-PE fragmentation. Load-balancing granularity has significant effects on the overall performance of Crane. Too small load-balancing granularity may cause performance loss due to the intra-PE fragmentation, especially for CNNs with high sparsity. However, large load-balancing granularity lacks precise and efficient control on workload stealing, which may decrease the benefits of load-balancing. Thus, we need to take performance overheads into consideration while choosing the optimal load-balancing granularity of Crane. As a result, we provide a detailed design space exploration on load-balancing granularity in Section 5.2.4.

4.8.3 Scalability

According to Section 4.3, the size of crossbar in *RAM Controller* grows exponentially when the number of PEs grows, which is quite hardware-expensive. To avoid these significant area overheads, we perform grouped load-balancing rather than global load-balancing while scaling up Crane. We divide PEs into several groups, and each PE group performs load-balancing individually with its own *Workload Scheduler*. Workload stealing can only happen across PEs within the same PE group. Compared to global load-balancing, grouped load-balancing may suffer from under-utilization and performance loss. Actually, this is a trade-off between hardware overheads and performance improvements. Detailed evaluation will be presented in Section 5.2.4.

4.8.4 Case for Weight Sharing Data-flow

Crane architecture also works with *Weight Sharing* data-flow. In contrast to *Input Sharing* data-flow, *Weight Sharing* data-flow broadcasts weights into the parallel PEs. As a result, we need to switch the role of input activation data and weights. We needs *Input RAM* to support independent accesses by parallel PEs. Each PE is equipped with a *Weight Buffer* to store the broadcast weights. The architecture designs of *Input RAM* and *Weight Buffer* are the same as the *Weight RAM* and *Input Buffer* in the *Input Sharing* data-flow, respectively. The architecture designs of other hardware modules remain the same. However, the memory size and data layout in on-chip RAMs should be re-organized, and the processing order of each PE should also be adjusted, accordingly.

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

5 Evaluation

In this section, we compare Crane with state-of-the-art accelerators for sparse CNNs, in respect of area, performance, and energy consumption. We also provide detailed design space exploration and sensitivity analysis of Crane.

5.1 Evaluation Setup

In this subsection, we introduce the setups for evaluation, including the choices of sparse CNN benchmarks and baseline designs, evaluation methodology, and detailed configurations of all the hardware accelerators.

5.1.1 Sparse CNN Benchmarks

We use five sparse CNNs as the benchmarks for the evaluation: AlexNet [20], VGG [4] GoogLeNet [21], ResNet-50 [3], and MobileNet [22]. These CNNs are all typical models for image classification on the ImageNet dataset [33]. We prune these CNNs using the approach proposed by Park et al. [8]. In our evaluation, we randomly sample 1000 images from ImageNet dataset for inference. The evaluation results are reported by the average results across these 1000 samples.

5.1.2 Baselines

We use two state-of-the-art sparse CNN accelerators as baselines: Cambricon-X [17] and SCNN [15]. As we have discussed, Cambricon-X employs *Input Sharing* data-flow, and exploits weight sparsity. SCNN represents accelerators that employ *Weight Sharing* data-flow, and it exploits both activation sparsity and weight sparsity.

5.1.3 Methodology

In order to prove the effectiveness over two baselines, we implement two versions of Crane: Crane-IS, which employs *Input Sharing* data-flow, and Crane-WS, which employs *Weight Sharing* data-flow. We implement both designs in Verilog to evaluate area and power characteristics. We also re-implement Cambricon-X and SCNN for area comparison in the same method. To perform a fair comparison on area and energy, the Verilog designs of Crane-IS and Cambricon-X are synthesized by Synopsys Design Compiler with SMIC 65nm libraries (similar to the configurations of Cambricon-X [17]). On the other hand, the Verilog designs of Crane-WS and SCNN are synthesized by Synopsys Design Compiler with a Samsumg 14nm LPP technology (quite close to the configurations of SCNN [15]). We use CACTI 6.0 [34] to estimate the energy consumption of DRAM accesses.

To evaluate the performance of Crane-IS, Cambricon-X, Crane-WS, and SCNN, we implement cycle-accurate simulators for all these designs. The simulators take the input data and weights from the benchmark CNNs as input, and process all the convolutional layers.

5.1.4 Hardware Configurations

Crane-IS and Crane-WS store both activation data and weights in an encoded format, and each non-zero value is paired with an index. In both designs, we use 16-bit fixed-point numbers for the non-zero activation data and weights. And we use 4-bit fixed-point numbers for the indices. Our indexing method causes additional overheads for storing indices. As a result, when sparsity is below 0.2 (*1-16/(16+4)=0.2*), the encoding format employed by Crane requires more memory than the original data. Accordingly, the energy consumption on data accessing increases for small pruning ratios (<0.2), compared with dense CNN accelerators.

To facilitate a fair (similar-area) comparison on area and energy, the hardware configurations of Crane-IS and Crane-WS are quite close to those of Cambricon-X and SCNN, respectively. Crane-IS employs 16 parallel PEs, and each PE is equipped with 16 multipliers and one 16-in adder tree. Thus, there are 256 parallel multipliers and 16 adder trees in total, which are the same as Cambricon-X. Cambricon-X employs an 8KB input buffer, a 32KB weight buffer, and an 8KB output buffer. The total sizes of *Input Buffer*, *Weight RAM*, and *Output RAM* in Crane-IS are 10KB, 40KB, and 8KB, respectively. Note that Crane-IS uses 25% larger buffers to store the indices. It means that two designs can store the same size of data on-chip in the worst case that there is no sparsity in weights or input activation data.

SCNN employs 64 parallel PEs, and each PE is equipped with a 16-multiplier array. As a result, SCNN performs convolution with 1024 multipliers. Similarly, Crane-WS employs 16 parallel PEs, and each PE is equipped with 64 multipliers and one 64-in adder tree. Since both SCNN and Crane-WS store activation data and weights in an encoded format, we employ the same buffer size for both designs. Specifically, both SCNN and Crane-WS have 1.25MB on-chip buffers for input/output activation data and 32KB memory for weights.

5.2 Experimental Results

In this subsection, we compare two Crane designs with counterpart baselines, in respect of area, performance, and energy consumption. Then, we present results for design space exploration and sensitivity analysis on CNN sparsity.

5.2.1 Area

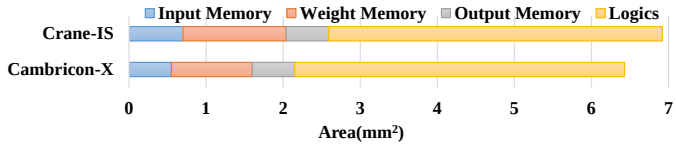
In Figure 12 (a), we show the area breakdowns and comparison between Crane-IS and Cambricon-X. The overall area of Crane-IS ($6.92mm^2$) is 7.62% larger than that of Cambricon-X ($6.43mm^2$). Crane-IS employs 27.5% larger input and weight memories than Cambricon-X to store extra indices. Logic circuits in Crane-IS includes the *Index Selector*, *Output Unit*, *DMA*, *Workload Scheduler*, etc. Since Cambricon-X does not have exactly corresponding modules, we show the area of these logic circuits together. In Cambricon-X, the logic circuits include CTFU, IM, CP, etc [17]. Logic circuits contribute the most to the area of Crane-IS (62.57%) and Cambricon-X (66.56%). In Crane-IS, the logics related to load-balancing (including both *Workload Scheduler* and crossbars) occupy only 5% of Crane-IS's total area.

The area breakdowns and comparison between Crane-WS and SCNN are shown in Figure 12 (b). Crane-WS's total area is $7.36mm^2$, which is 17.66% smaller than that of SCNN ($8.66mm^2$). Since both Crane-WS and SCNN exploit weight and activation sparsity with data encoding, the area of on-chip memories are the same. The area reduction of Crane-WS over SCNN mainly comes from the area reduction of logic circuits. According to Parashar et al. [15], SCNN employs a 16×32 crossbar and a 32-bank accumulator buffer in

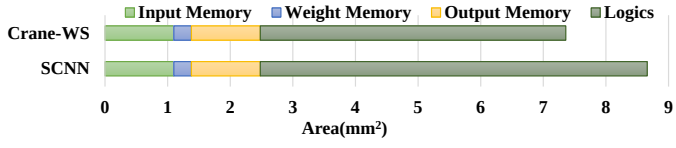
0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

each PE. As a result, there are in total 64 16×32 crossbars and 64 32-bank accumulator buffers in SCNN. These modules are extremely expensive for hardware implementation, and they contribute to about 50% of SCNN's total area. In the contrast, Crane-WS only employs two 16×16 crossbars (residing in the RAM controllers) in total, and all PEs share the crossbars and multi-bank RAMs. Though Crane-WS needs extra logic circuits (about 8%) for load-balancing, the area of logic circuits is still 26.63% smaller than that of SCNN.



(a) Crane-IS vs. Cambricon-X



(b) Crane-WS vs. SCNN

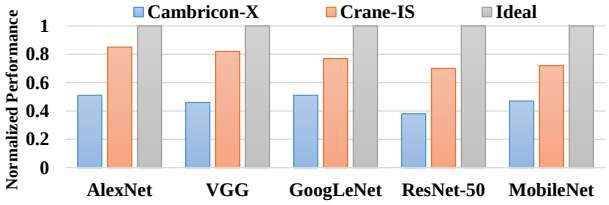
Fig. 12. Area breakdown and comparison

5.2.2 Performance

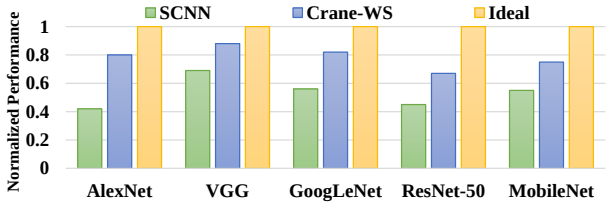
We compare the performance of Crane-IS and Cambricon-X over all the five sparse CNN benchmarks in Figure 13 (a). We also show the performance of an ideal design (denoted as Ideal), which indicates the theoretical peak performance with 100% utilization ignoring the underlying hardware implementation. Performance results of all the hardware accelerators are normalized to that of Ideal. According to Figure 13 (a), Cambricon-X achieves only 46.7% of Ideal's performance over the five sparse CNN benchmarks on average. There are two reasons for this performance gap. Firstly, Cambricon-X cannot benefit from the computation reduction provided by activation sparsity. Secondly, weight irregularity causes workload imbalance across different PEs, but Cambricon-X is not able to address this irregularity.

Crane-IS exploits both activation sparsity and weight sparsity. And it employs load-balancing to address the workload imbalance problem. As shown in Figure 13 (a), Crane-IS outperforms Cambricon-X for all the five sparse CNN benchmarks, and the average speedup is 1.67x. However, there still exists a gap between Crane-IS and Ideal. Intra-PE fragmentation prevents Crane-IS from further performance improvements. Besides, there exist extra control overheads and conflicts (as discussed in Section 4.4) during workload scheduling and stealing. Nevertheless, Crane-IS's speedup is still a big step-forward to bridge the performance gap between Cambricon-X and Ideal.

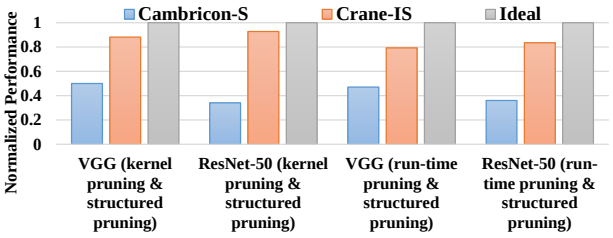
In Figure 13 (b), we make a performance comparison between SCNN and Crane-WS over five sparse CNN benchmarks. Similarly, we also add an ideal design to this comparison, and normalize all the performance results to that of Ideal. SCNN exploits both activation sparsity and weight sparsity for convolution processing, but it achieves only 53.26% of Ideal's performance on average. The reason is that SCNN suffers from severe under-utilization problem, which prevents it from fully benefiting from CNN sparsity. Compared to SCNN, Crane-WS achieves 1.27x∼1.88x speedups over five sparse CNN benchmarks.



(a) Cambricon-X vs. Crane-IS



(b) SCNN vs. Crane-WS



(c) Cambricon-S vs. Crane-IS

Fig. 13. Performance comparison

Recent works [9], [19], [23], [24] propose to address the irregularity of weight sparsity by applying structured pruning. Structured pruning balances the workloads across different PEs by pruning similar amount of weights or even the weights at the same positions in different weight kernels. Cambricon-S [19] is a customized accelerator for processing sparse CNNs pruned by structured pruning. However, structured pruning needs extra training efforts and may bring accuracy loss when the sparsity gets higher. And it still suffers from sparsity irregularity when kernel pruning or run-time pruning is applied. To clearly show this problem, we find public CNN benchmarks (VGG and ResNet-50) that employ kernel pruning and run-time pruning techniques [35]. We apply the structured pruning approach proposed by Cambricon-S to further prune these four CNN models to about 80% sparsity.

We implement a cycle-accurate simulator for Cambricon-S to evaluate its performance, and the configurations are the same as Cambricon-S. Since Cambricon-S employs *Input Sharing* data-flow, we compares its performance with Crane-IS. In Figure 13 (c), we show this comparison over the sparse CNN models pruned by kernel pruning and run-time pruning. We still show the ideal performance, and normalize results of Cambricon-S and Crane-IS to Ideal's.

It is obvious that Cambricon-S severely suffers from the irregularity of kernel sparsity and run-time sparsity even though it avoids weight irregularity with structured pruning. It only achieves 34.35%∼50.54% of Ideal's performance across all these benchmarks. By employing load-

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

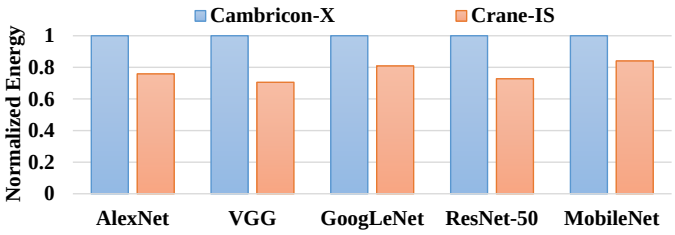
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

balancing, Crane-IS can effectively address the workload imbalance and achieve high utilization (79.29%∼92.74%) of computational resources. Figure 13 (c) shows that Crane-IS outperforms Cambricon-S by 1.69x∼2.73x when kernel pruning or run-time pruning is applied.

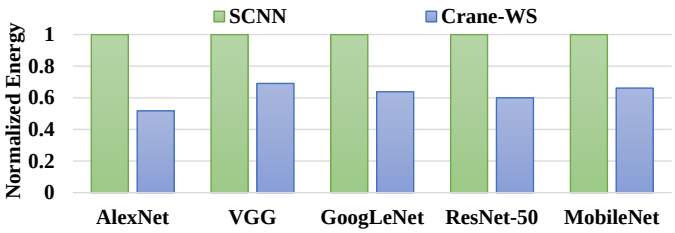
5.2.3 Energy

Figure 14 (a) shows the energy comparison of Crane-IS and Cambricon-X. The energy results are normalized to those of Cambricon-X. Compared to Cambricon-X, the energy saving of Crane-IS varies across different benchmarks from 19.12% to 27.27%, and it is 23.20% on average. The energy savings come from two sources. First, execution time of a CNN is reduced with higher PE utilization. Thus, less leakage power is induced. Second, less data are loaded to Crane-IS after encoding activation data. Note that the *Workload Scheduler* induces extra logic energy consumption, which is trivial compared to the rest.

The energy comparison results of Crane-WS and SCNN over five sparse CNN benchmarks are listed in Figure 14 (b). Similarly, the energy results are normalized to those of SCNN. According to Figure 14 (b), Crane-WS saves 30.88% to 48.24% energy over SCNN because Crane-WS improves the utilization of computational resources and avoids using tens of hardware-expensive crossbars.



(a) Cambricon-X vs. Crane-IS



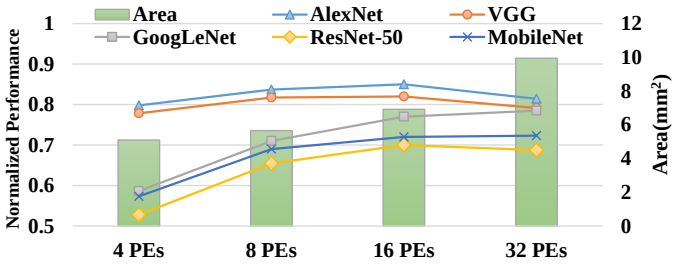
(b) SCNN vs. Crane-WS

Fig. 14. Energy comparison

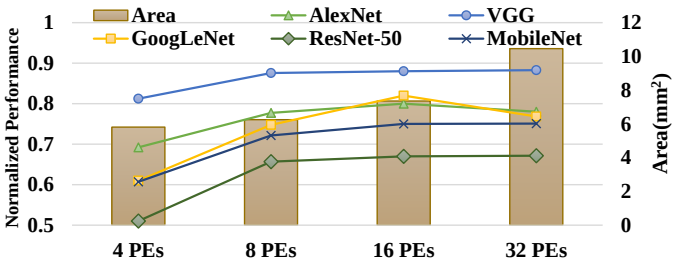
5.2.4 Design Space Exploration

PE Granularity. We perform design space exploration on PE granularity for Crane-IS and Crane-WS. We hold the total number of parallel multipliers stationary, and vary the number of PEs from 4 to 8, 16, and 32. For example, Crane-IS employs 256 parallel multipliers in total. If Crane-IS employs four PEs, PE granularity is set to 64. If Crane-IS employs eight PEs, PE granularity is set to 32. In other words, small number of PEs indicates large PE granularity.

In Figure 15, we show the performance and area results of different PE numbers for Crane-IS and Crane-WS, and the performance results are normalized to those of Ideal. For both designs, employing a small number of PEs (4) is a bad choice. One reason is that the intra-PE fragmentation problem aggravates with more parallel multipliers residing in a PE. Another reason is that sub-buffer conflicts caused by workload stealing may increase with fewer PEs. As the number of PEs gets larger (16 or 32), the performance improvements become less. The benefits of workload stealing decrease because the number of work-items assigned to each PE is reduced. It is easier to understand this with an extreme case that each PE is only responsible for generating one output activation data. In addition, Figure 15 shows that area of both designs grows rapidly with more PEs. The main reason is that the size of crossbars grows exponentially as the number of PEs grows. Considering both performance gains and area overheads, we choose to employ 16 PEs in both Crane-IS and Crane-WS. As a result, PE granularity of Crane-IS and Crane-WS is set to 16 and 64, respectively.



(a) Crane-IS



(b) Crane-WS

Fig. 15. Design space exploration on PE granularity

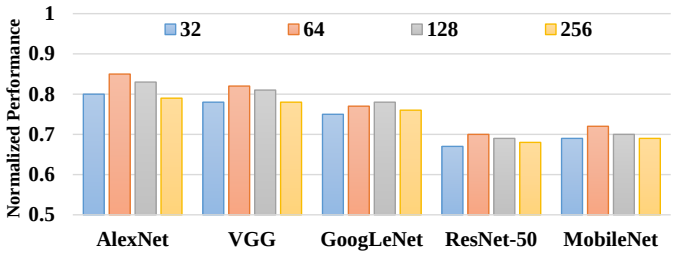
Load-balancing Granularity. In Figure 16, we perform a design space exploration on load-balancing granularity for Crane-IS and Crane-WS. The load-balancing granularity of both designs varies from 32 to 64, 128, and 256. We show the normalized performance to Ideal. For both Crane-IS and Crane-WS, the performance gains of small stealing granularity (32) are limited because of intra-PE fragmentation. Thus, load-balancing granularity smaller than 32 is not chosen. As we have analyzed in Section 4.8.2, performance of both designs is improved when load-balancing granularity gets larger. Due to the lack of precise and efficient control on workload stealing, large load-balancing granularity like 256 may cause slowdown of performance improvements or even performance loss in some benchmarks.

As shown in Figure 16, the optimal load-balancing granularity is different for Crane-IS and Crane-WS. The reason is that the number of parallel multipliers within a PE is different in Crane-IS and Crane-WS. With more multipliers within a PE (Crane-WS), a larger load-balancing granularity is demanded to address intra-PE fragmentation. According to Figure 16, the load-balancing granularity is set to 64 and

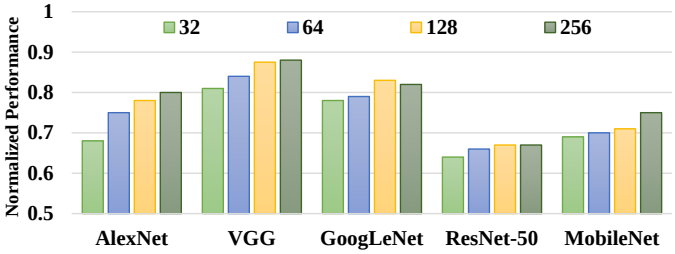
0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

256 for Crane-IS and Crane-WS, respectively.



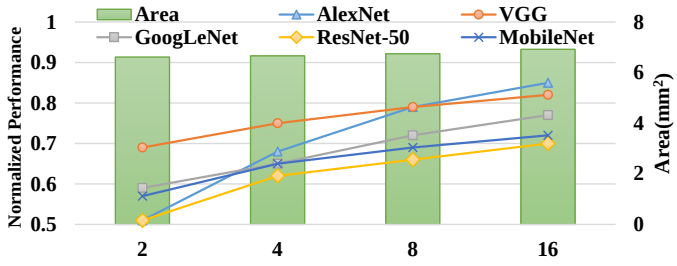
(a) Crane-IS



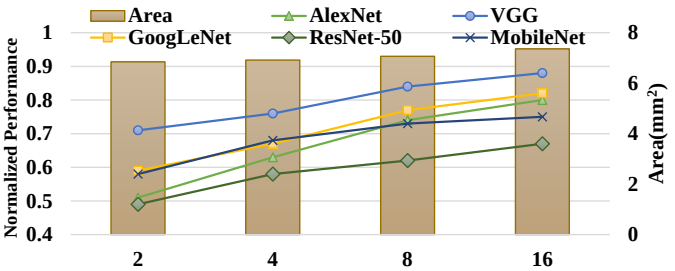
(b) Crane-WS

Fig. 16. Design space exploration on load-balancing granularity

PE group granularity. While scaling up Crane, we propose a grouped load-balancing scheme considering the trade-off between hardware overheads and performance. In Figure 17, we show how area and performance scale with different PE group granularity (i.e., the number of PEs within a PE group) for Crane-IS and Crane-WS. We hold the total number of PEs stationary to 16, and vary the number of PEs within a PE group from 2 to 4, 8, and 16.



(a) Crane-IS



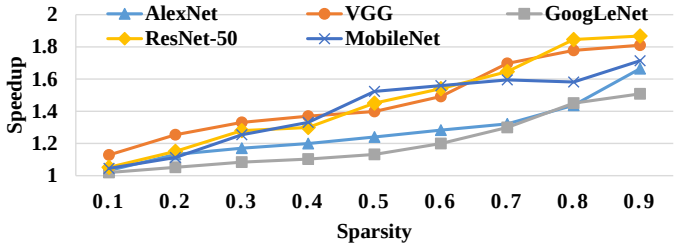
(b) Crane-WS

Fig. 17. Design space exploration on PE group granularity

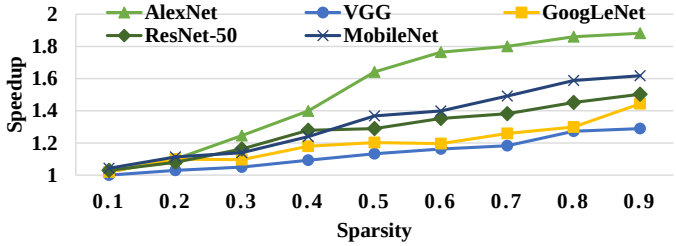
According to Figure 17, performance is improved when the PE group granularity grows for both Crane-IS and Crane-WS. Since workload stealing only occurs within a PE group, more PEs within a PE group indicates more opportunities for potential workload stealing, which is much more beneficial for utilization improvements. As shown in Figure 17, area of both Crane-IS and Crane-WS also increases as the PE group granularity gets larger. The area growths mainly come from the growing crossbars in RAM controllers. Considering both performance and area overheads, we set the PE group granularity to 16 for both Crane-IS and Crane-WS.

5.2.5 *Sparsity Sensitivity*

To analyze the performance sensitivity of Crane-IS and Crane-WS to CNN sparsity, we perform nine tests on Crane-IS, Cambricon-X, Crane-WS, and SCNN by sweeping CNN sparsity from 10% to 90% for each sparse CNN benchmark. For Crane-IS, we show its speedups over Cambricon-X in Figure 18 (a). Figure 18 (b) shows Crane-WS's speedups over SCNN. According to Figure 18, we can tell both Crane-IS and Crane-WS achieve relatively small speedups over their own baselines, when the sparsity is low (10%∼30%). However, when the sparsity goes higher, the speedups of both Crane-IS and Crane-WS grow. The reason is that the under-utilization problems get more serious when CNN sparsity gets higher, which causes significant performance loss in Cambricon-X and SCNN. Crane-IS and Crane-WS employ dynamic load-balancing to address the under-utilization problems, and the performance is not affected much.



(a) Crane-IS's speedups over Cambricon-X



(b) Crane-WS's speedups over SCNN

Fig. 18. Sensitivity analysis on CNN sparsity

6 Related Work

Prior works like Eyeriss [18], Cnvlutin [14] and Cambricon-X [17] only explore either activation sparsity or weight sparsity. Eyeriss explores activation sparsity, and gates the multiplier when input activation is zero. Cnvlutin improves performance by detecting zero-valued activations and eliminating the corresponding multiplications. Cambricon-X aims to speed up CNNs by exploring weight sparsity only, and it still wastes energy on transferring zero-valued activations. Different from these works, Crane exploits all kinds of CNN sparsity to improve performance. Crane also encodes activation and weight to reduce memory bandwidth requirements. Furthermore, Crane can handle all kinds of CNN irregularities with dynamic load-balancing.

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

Recent works such as SCNN [15] and Cambricon-S [19] exploit both activation sparsity and weight sparsity. SCNN saves weights and activation data in an encoded format, and it delivers only non-zero activation data and weights into multipliers. However, under-utilization of computational resources prevents SCNN from fully benefiting from CNN sparsity. Cambricon-S [19] employs a zero-aware PE to skip the multiplications whose results will be zero. To address the irregularity of weight sparsity, Cambricon-S proposes a structured pruning approach. However, the under-utilization caused by irregularity still exists when kernel pruning and run-time pruning techniques are applied. Besides, structured pruning needs extra training efforts, and may cause accuracy loss. Different from these works, Crane employs a pure hardware-based approach to address the under-utilization problems, and Crane's approach works for all kinds of CNN sparsity. Crane's load-balancing method can dynamically balance workloads across different PEs without any extra training efforts or accuracy loss.

EIE [16] and ESE [24] are accelerators for sparse fully-connected layers. EIE employs a deep input FIFO for each PE to address the inter-PE load imbalance, but PE has to stay idle when the input FIFO is full. ESE improves utilization with structured pruning. These two designs target the acceleration of fully-connected layers, but Crane targets to accelerate convolutional layers, which occupy over 90% of the computation time of CNN inference [36] [37].

7 Conclusion

Sparsity irregularities in CNNs can cause severe PE under-utilization in an accelerator. In this paper, we propose a hardware-based load-balancing method to mitigate this problem. We apply our method to two major data-flows for CNN inference, which cover all state-of-the-art sparse CNN accelerators. We also propose an accelerator design, Crane, to employ the load-balancing method to address all types of sparsity irregularities in sparse CNNs. Experimental results show that, compared with state-of-the-art accelerator baselines, Crane achieves 1.27x∼1.88x speedup, while saving 16%∼48% energy consumption.

Acknowledgments

This work is supported by National Natural Science Foundation of China (No.61832020, 61572045).

References

[1] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A convolutional neural network cascade for face detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5325–5334.

[2] S. S. Farfade, M. J. Saberian, and L.-J. Li, "Multi-view face detection using deep convolutional neural networks," in *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*. ACM, 2015, pp. 643–650.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

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

[5] Z. Xu, Y. Yang, and A. G. Hauptmann, "A discriminative cnn video representation for event detection," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015, pp. 1798–1807.

[6] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[8] J. Park, S. Li, W. Wen, P. T. P. Tang, H. Li, Y. Chen, and P. Dubey, "Faster cnns with direct sparse convolutions and guided pruning," *arXiv preprint arXiv:1608.01409*, 2016.

[9] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2074–2082.

[10] J. M. Alvarez and M. Salzmann, "Learning the number of neurons in deep networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2270–2278. [Online]. Available: http://papers.nips.cc/paper/6372-learning-the-number-of-neurons-in-deep-networks.pdf

[11] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *International Conference on Computer Vision (ICCV)*, vol. 2, no. 6, 2017.

[12] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Advances in Neural Information Processing Systems*, 2017, pp. 2181–2191.

[13] X. Gao, Y. Zhao, L. Dudziak, R. Mullins, and C.-z. Xu, "Dynamic channel pruning: Feature boosting and suppression," *arXiv preprint arXiv:1810.05331*, 2018.

[14] J. Albericio, P. Judd, T. H. Hetherington, T. M. Aamodt, N. D. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, 2016, pp. 1–13.

[15] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 27–40.

[16] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016, pp. 243–254.

[17] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–12.

[18] Y. Chen, J. S. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, 2016, pp. 367–379.

[19] X. Zhou, Z. Du, Q. Guo, C. Liu, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *Proceedings of the 51st IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2018.

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

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[23] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.

[24] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. B. J. Dally, "ESE: efficient speech recognition engine with sparse LSTM on FPGA," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2017, Monterey, CA, USA, February 22-24, 2017*, 2017, pp. 75–84.

[25] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

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

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.

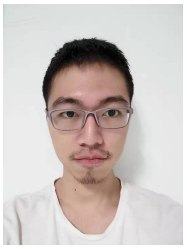
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2981080, IEEE Transactions on Computers

[27] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.
[28] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," *arXiv preprint arXiv:1707.06342*, 2017.
[29] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," *arXiv preprint arXiv:1707.01213*, 2017.
[30] M. Frigo, C. E. Leiserson, and K. H. Randall, "The implementation of the cilk-5 multithreaded language," *ACM Sigplan Notices*, vol. 33, no. 5, pp. 212–223, 1998.
[31] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 751–764, 2017.
[32] R. W. Vuduc and J. W. Demmel, *Automatic performance tuning of sparse matrix kernels*. University of California, Berkeley, 2003, vol. 1.
[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
[34] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "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*. IEEE Computer Society, 2007, pp. 3–14.
[35] "Cnn pruning," https://github.com/cnnpruning/CNN-Pruning.
[36] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *International conference on artificial neural networks*. Springer, 2014, pp. 281–290.
[37] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

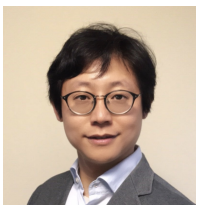
Yijin Guan received the BS degree in microelectronics from Peking University, Beijing, China, in 2014, and the PhD degree in computer science from Peking University, Beijing, China, in 2019. His current research interests include domain-specific architecture and deep learning accelerators.

Guangyu Sun received the BS and MS degrees from Tsinghua University, Beijing, China, in 2003 and 2006, respectively, and the PhD degree in computer science from Pennsylvania State University, State College, Pennsylvania, in 2011. He is currently an associate professor of CECA at Peking University, Beijing, China. His research interests include computer architecture, VLSI design, and electronic design automation (EDA). He has published more than 60 journals and refereed conference papers in these areas. He has also served as a peer reviewer and technical referee for several journals, which include the IEEE Micro, the IEEE Transactions on Very Large Scale Integration Systems (TVLSI), the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), etc. He is a member of the CCF and IEEE.

Zhihang Yuan received the BS degree from Peking University, Beijing, China, in 2017, where he is currently working toward the master degree in School of Electrical Engineering and Computer Science. His current research interests include energy-efficient algorithms and accelerators for neural networks.

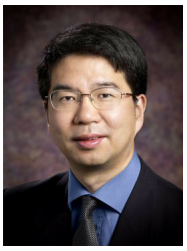


Xingchen Li received the BS degree from Peking University, Beijing, China, in 2018. He is currently working toward the PhD degree in computer science at Peking University, China. His current research interests include computer architecture, non-volatile memory, and process-in-memory technology.

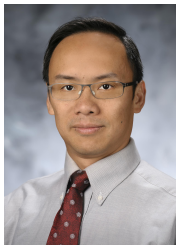


Ningyi Xu received the B.S. and Ph.D degrees from Tsinghua University in 2001 and 2006, respectively. He was a principle architect with Baidu, and a lead researcher with hardware computing group, Microsoft Research Asia, Beijing. His current research interests include domain specific architecture and heterogeneous computing for applications and services in datacenter.

Shu Chen received the BS degree in finance from Southwestern University of Finance and Economics, Chengdu, China, in 2014, and the PhD degree in finance from Peking University, Beijing, China, in 2019. She is currently an analyst of Harvest Fund, Beijing, China. Her current research interests include AI investment and deep learning.



Jason Cong received the BS degree in computer science from Peking University, in 1985, and the MS and PhD degrees in computer science from the University of Illinois at Urbana-Champaign, in 1987 and 1990, respectively. Currently, he is a Chancellor's professor with the Computer Science Department, also with joint appointment from the Electrical Engineering Department, University of California, Los Angeles, the director of Center for Domain-Specific Computing (CDSC), and the director of VLSI Architecture, Synthesis, and Technology (VAST) Laboratory. He served as the chair the UCLA Computer Science Department from 2005 to 2008, and is also a distinguished visiting professor at Peking University. His research interests include synthesis of VLSI circuits and systems, programmable systems, novel computer architectures, nano-systems, and highly scalable algorithms. He has more than 400 publications in these areas, including 10 best paper awards, two 10-Year Most Influential Paper Awards. He was elected to an IEEE fellow in 2000 and ACM fellow in 2008, and received two IEEE Technical Achievement Awards, one from the Circuits and System Society (2010) and the other from the Computer Society (2016).



Yuan Xie (M'02–SM'09–F'14) received the Ph.D. degree from the Electrical Engineering Department, Princeton University, Princeton, NJ, USA, in 2002.

From 2002 to 2003, he was with IBM, Armonk, NY, USA. From 2012 to 2013, he was with the AMD Research China Laboratory, Beijing, China. From 2003 to 2014, he was a Professor with Pennsylvania State University, State College, PA, USA. He is currently a Professor with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, USA. His current research interests include computer architecture, electronic design automation, and VLSI design.

Dr. Xie served as the TPC Chair for HPCA 2018. He is currently the Editor-in-Chief of the ACM Journal on Emerging Technologies in Computing Systems (JETC), the Senior Associate Editor (SAE) of the ACM Transactions on Design Automation for Electronics Systems (TODAES), and an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS. He is an Expert in computer architecture who has been inducted to ISCA/MICRO/HPCA Hall of Fame.

0018-9340 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Canberra. Downloaded on April 27,2020 at 13:19:39 UTC from IEEE Xplore. Restrictions apply.